aws

開発者ガイド

Amazon API Gateway

Copyright © 2024 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon API Gateway: 開発者ガイド

Copyright © 2024 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon の商標およびトレードドレスは、Amazon のものではない製品またはサービスにも関連して、お客様に混乱を招いたり Amazon の信用を傷つけたり失わせたりするいかなる形においても使用することはできません。Amazon が所有していない他のすべての商標は、それぞれの所有者の所有物であり、Amazon と提携、接続、または後援されている場合とされていない場合があります。

Table of Contents

Amazon API Gateway とは何ですか?

Amazon API Gateway は、あらゆる規模の REST、HTTP、および WebSocket API を作成、公開、維持、モニタリング、およびセキュア化するための AWS のサービスです。API 開発者は、AWS または他のウェブサービス、AWS クラウドに保存されているデータにアクセスする API を作成できます。API Gateway API デベロッパーとして、独自のクライアントアプリケーションで使用するための API を作成できます。または、API をサードパーティーのアプリ開発者に対して使用可能にできます。詳細については、「the section called “API Gateway を使用するユーザー”」を参照してください。

API Gateway は、次のような RESTful API を作成します。

- HTTP ベース。
- ステートレスなクライアント/サーバー通信を有効にします。
- GET、POST、PUT、PATCH、DELETE などの標準の HTTP メソッドを実装します。

API Gateway REST API および HTTP API の詳細については、「the section called “REST API と HTTP API 間で選択する ”」、「HTTP API の操作」、「the section called “API Gateway を使用して REST API を作成する”」、および「the section called “開発”」を参照してください。

API Gateway は、以下のような WebSocket API を作成します。

- WebSocket プロトコルを遵守します。これにより、クライアントとサーバー間のステートフルな全二重通信が可能になります。
- メッセージの内容に基づいて、受信メッセージをルーティングします。

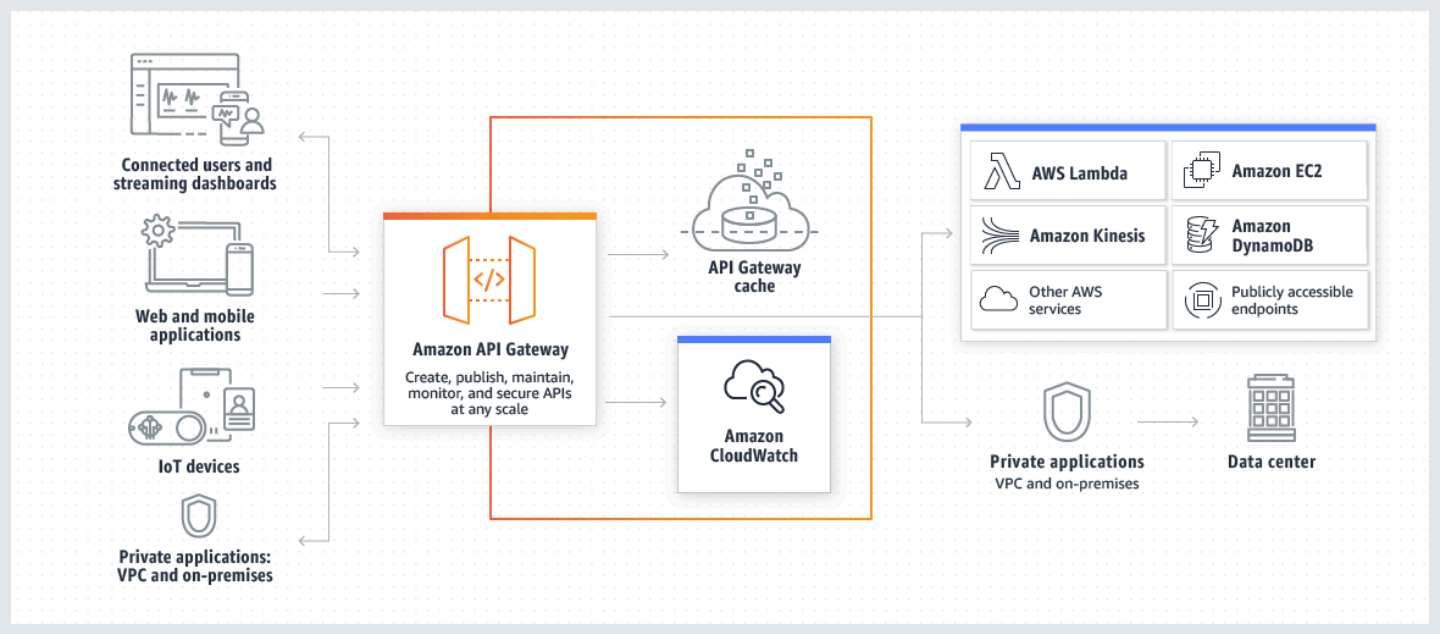
API Gateway WebSocket API の詳細については、「the section called “API Gateway を使用して WebSocket API を作成する”」および「the section called “WebSocket API について”」を参照してください。

トピック

- AWS サーバーレスインフラストラクチャの一部
- Amazon API Gateway の使用を開始する方法
- Amazon API Gateway の概念
- REST API と HTTP API 間で選択する
- REST API コンソールの開始方法

API Gateway のアーキテクチャ

API Gateway のアーキテクチャを次の図に示します。



この図は、Amazon API Gateway で構築した API が、お客様、またはお客様のデベロッパーカスタマーにAWS サーバーレスアプリケーションを構築するための統合された一貫的なデベロッパーエクスペリエンスを提供する方法を示しています。API Gateway では、最大で数十万個の同時 API コールの受け入れ処理に伴うすべてのタスクを取り扱います。これにはトラフィック管理、認証とアクセスコントロール、モニタリング、API バージョン管理が含まれます。

API Gateway は、例えば、Amazon Elastic Compute Cloud (Amazon EC2) で実行されているワークロード、AWS Lambda で実行されているコード、ウェブアプリケーション、リアルタイム通信アプリケーションなど、アプリケーションがバックエンドサービスからデータ、ビジネスロジック、機能にアクセスするための「フロントドア」として機能します。

API Gateway の特徴

Amazon API Gateway には、次のような機能があります。

- ステートフル (WebSocket) およびステートレス (HTTP と REST) API のサポート。
- AWS Identity and Access Management ポリシー、Lambda オーソライザー関数、Amazon Cognito ユーザープールなど、強力で柔軟な認証メカニズム。
- 変更を安全に進めるための Canary リリースのデプロイ。
- API の使用状況と API の変更に関する CloudTrail ログ記録とモニタリング。
- アラームの設定機能を含む、CloudWatch アクセスのログ記録と実行のログ記録。詳細については、「the section called “CloudWatch メトリクス”」および「the section called “メトリクス”」を参照してください。
- AWS CloudFormation テンプレートを使用して API の作成を有効にする機能。詳細については、Amazon API Gateway Resource Types Reference および「Amazon API Gateway V2 リソースタイプのリファレンス」を参照してください。
- カスタムドメイン名のサポート。
- 一般的なウェブの脆弱性から API を保護するための AWS WAF との統合。
- パフォーマンスのレイテンシーを理解し、対処順位を決定するための AWS X-Ray との統合。

API Gateway 機能のリリースの完全なリストについては、「ドキュメント履歴」を参照してください。

API Gateway のユースケース

トピック

- API Gateway を使用して REST API を作成する
- API Gateway を使用して HTTP API を作成する
- API Gateway を使用して WebSocket API を作成する
- API Gateway を使用するユーザー

API Gateway を使用して REST API を作成する

API Gateway REST API はリソースとメソッドで構成されます。リソースは、リソースパスを介してアプリがアクセスできる論理エンティティを表します。メソッドは、API のユーザーによって送信される REST API リクエストや、ユーザーに返るレスポンスに対応しています。

たとえば、`/incomes` はアプリユーザーの収益を表すリソースのパスです。リソースには、適切な HTTP 動詞 (GET、POST、PUT、PATCH、DELETE など) によって定義されるオペレーションを 1 つ以上含めることができます。リソースパスとオペレーションの組み合わせによって API のメソッドが識別されます。たとえば、`POST /incomes` メソッドで呼び出し元が得た収益を追加し、`GET /expenses` メソッドで、呼び出し元によって報告済みの経費をクエリすることができます。

アプリがリクエストされたデータの保存先および取り込み先をバックエンドで把握する必要はありません。API Gateway REST API で、フロントエンドは、メソッドリクエストとメソッドレスポンスによってカプセル化されます。API は統合リクエストと統合レスポンスを使用してバックエンドと連結します。

たとえば、DynamoDB をバックエンドとして使用した場合、API デベロッパーは、受信するメソッドリクエストを選択されたバックエンドに転送するように統合リクエストを設定します。この設定には、適切な DynamoDB アクションの仕様、IAM ロールとポリシー、および必要な入力データ変換などがあります。バックエンドは、統合レスポンスとして API Gateway に結果を返します。

統合レスポンスをクライアントの (一定の HTTP ステータスコードの) 適切なメソッドレスポンスにルーティングするには、統合からメソッドまで必要なレスポンスパラメータをマッピングするように統合レスポンスを設定できます。その後、必要に応じて、バックエンドの出力データ形式をフロントエンドの出力データ形式に変換します。API Gateway を使用することでペイロードのスキーマまたはモデルを定義し、本文マッピングテンプレートを容易に設定できるようにします。

API Gateway は、次のような REST API 管理機能を提供します。

- OpenAPI に対する API Gateway 拡張を使用した SDK の生成と API ドキュメントの作成のサポート。
- HTTP リクエストのスロットリング

API Gateway を使用して HTTP API を作成する

HTTP API を使用すると、REST API よりも低いレイテンシーとコストで RESTful API を作成することができます。

HTTP API を使用して、AWS Lambda 関数、またはパブリックルーティングが可能なあらゆる HTTP エンドポイントにリクエストを送信できます。

たとえば、バックエンドの Lambda 関数と統合する HTTP API を作成できます。クライアントが API を呼び出すと、API Gateway はリクエストを Lambda 関数に送信し、関数のレスポンスをクライアントに返します。

HTTP API は、OpenID Connect および OAuth 2.0 認証をサポートしています。クロスオリジンリソース共有 (CORS) と自動デプロイのサポートが組み込まれています。

詳細については、「the section called “REST API と HTTP API 間で選択する ”」を参照してください。

API Gateway を使用して WebSocket API を作成する

WebSocket API では、クライアントとサーバーの両方がいつでも相互にメッセージを送信できます。バックエンドサーバーは接続されたユーザーとデバイスに簡単にデータをプッシュでき、複雑なポーリングメカニズムを実装する必要がありません。

例えば、API Gateway WebSocket API と AWS Lambda を使用してサーバーレスアプリケーションを構築し、チャットルームで個人ユーザーまたはユーザーのグループとメッセージを送受信することができます。または、メッセージの内容に基づいて、AWS Lambda、Amazon Kinesis、または HTTP エンドポイントなどのバックエンドサービスを呼び出すこともできます。

API Gateway WebSocket API を使用して、安全なリアルタイムの通信アプリケーションを構築でき、接続または大規模なデータ交換を管理するためにサーバーをプロビジョニングまたは管理する必要がありません。ターゲットを絞ったユースケースには、次のようなリアルタイムのアプリケーションが含まれます。

- チャットアプリケーション
- 株式相場表示などのリアルタイムのダッシュボード
- リアルタイムのアラートおよび通知

API Gateway は、次のような WebSocket API 管理機能を提供します。

- 接続とメッセージのモニタリングとスロットリング
- API からバックエンドサービスを通過するメッセージの AWS X-Ray を使用したトレース
- HTTP/HTTPS エンドポイントとの簡単な統合

API Gateway を使用するユーザー

API Gateway を使用するのは、API デベロッパーとアプリデベロッパーです。

API デベロッパーは API Gateway で必要な機能を使用できるようにするため、API を作成しデプロイします。API 開発者は、API を所有する AWS アカウントのユーザーである必要があります。

アプリケーションデベロッパーは、API Gateway で API デベロッパーによって作成された WebSocket または REST API を呼び出して、AWS のサービスを呼び出すために機能するアプリケーションを構築します。

アプリ開発者は、API 開発者の顧客です。アプリケーションデベロッパーに AWS アカウントは必要ありませんが、API に IAM 許可が必要ないか、Amazon Cognito ユーザープール ID フェデレーションによってサポートされているサードパーティーのフェデレーテッド ID プロバイダー経由でのユーザー認証をサポートしていることが条件となります。そのような ID プロバイダーには、Amazon、Amazon Cognito ユーザープール、Facebook、Google などがあります。

API Gateway API の作成と管理

API デベロッパーは API 管理に API Gateway サービスコンポーネント (`apigateway`) を使用して、API を作成、設定、およびデプロイします。

API デベロッパーとして、API を作成および管理するには、「API Gateway の開始方法」の説明に従って、API Gateway コンソールを使用するか、「API リファレンス」を呼び出します。API を呼び出す方法には、複数の方法があります。これには、AWS Command Line Interface (AWS CLI) の使用や AWS SDK の使用が含まれます。また、AWS CloudFormation テンプレートまたは OpenAPI への API Gateway 拡張機能の使用 (REST API および HTTP API の場合) を使用して、API を作成することもできます。

API Gateway を使用できるリージョン、および関連する管理サービスエンドポイントのリストについては、「Amazon API Gateway エンドポイントとクォータ」を参照してください。

API Gateway API の呼び出し

アプリデベロッパーは API の実行に `execute-api` というAPI Gateway サービスコンポーネントを使用し、API Gateway で作成またはデプロイされた API を呼び出します。基礎となるプログラミングエンティティは、作成された API によって公開されます。このような API を呼び出す方法はいくつかあります。詳細については、「Amazon API Gateway での REST API の呼び出し」と「WebSocket API の呼び出し」を参照してください。

API Gateway へのアクセス

Amazon API Gateway には、以下の方法でアクセスできます。

- AWS Management Console – AWS Management Console は、API を作成して管理するためのウェブインターフェイスを提供します。「前提条件」のステップを完了したら、https://console.aws.amazon.com/apigateway で API Gateway コンソールにアクセスできます。
- AWS SDK – AWS が SDK を提供しているプログラミング言語を使用している場合は、SDK を使用して API Gateway にアクセスできます。SDK によって認証が簡素化され、開発環境との統合が容易になり、API Gateway コマンドにアクセスすることができます。詳細については、Tools for Amazon Web Services を参照してください。
- API Gateway V1 および V2 API – SDK に対応していないプログラミング言語を使用している場合、Amazon API Gateway Version 1 API Reference と Amazon API Gateway Version 2 API Reference を参照してください。
- AWS Command Line Interface – 詳細については、AWS Command Line Interface ユーザーガイドの「AWS Command Line Interface でのセットアップ」を参照してください。
- AWS Tools for Windows PowerShell – 詳細については、AWS Tools for Windows PowerShell ユーザーガイドの「AWS Tools for Windows PowerShell のセットアップ」を参照してください。

AWS サーバーレスインフラストラクチャの一部

API Gateway は AWS Lambda と連携して、AWS サーバーレスインフラストラクチャのアプリケーション向けの部分を形成します。サーバーレスの開始方法の詳細については、「サーバーレスデベロッパーガイド」を参照してください。

アプリケーションで一般的に利用可能な AWS のサービスを呼び出す場合は、Lambda を使用して必要なサービスとやり取りし、API Gateway で API メソッドを使用して Lambda 関数を公開することができます。AWS Lambda は、可用性に優れたコンピューティングインフラストラクチャでコードを実行します。また、必要に応じて、コンピューティングリソースを実行および管理します。サーバーレスアプリケーションを実現するため、API Gateway は AWS Lambda と HTTP エンドポイントによる合理化されたプロキシ統合をサポートします。

Amazon API Gateway の使用を開始する方法

Amazon API Gateway の概要については、以下を参照してください。

- 開始方法を参照してください。HTTP API を作成するための手順です。
- サーバーレスランド 、は説明のビデオです。
- Happy Little API Shorts は、簡単な講習動画シリーズです。

Amazon API Gateway の概念

API Gateway

API Gateway は以下をサポートする AWS のサービスです。

- バックエンド HTTP エンドポイント、AWS Lambda 関数、または AWS のその他サービスを公開するための RESTful アプリケーションプログラミングインターフェイス (API) の作成、デプロイ、および管理。
- AWS Lambda 関数、または AWS のその他サービスを公開するための WebSocket API の作成、デプロイ、および管理。
- フロントエンド HTTP および WebSocket エンドポイントによって公開された API メソッドの呼び出し。

API Gateway REST API

バックエンド HTTP エンドポイント、Lambda 関数、または AWS のその他サービスを使用して統合されているリソースおよびメソッドのコレクション。このコレクションは、1 つ以上のステージでデプロイできます。通常、API リソースはアプリケーションロジックに従ってリソースツリーで整理されます。各 API リソースは、API Gateway でサポートされている一意の HTTP 動詞を持つ 1 つ以上の API メソッドを公開できます。詳細については、「the section called “REST API と HTTP API 間で選択する ”」を参照してください。

API Gateway HTTP API

バックエンド HTTP エンドポイントまたは Lambda 関数と統合されるルートとメソッドのコレクション。このコレクションは、1 つ以上のステージでデプロイできます。各ルートは、API Gateway でサポートされている一意の HTTP 動詞を持つ 1 つ以上の API メソッドを公開できます。詳細については、「the section called “REST API と HTTP API 間で選択する ”」を参照してください。

API Gateway WebSocket API

バックエンド HTTP エンドポイント、Lambda 関数、または AWS のその他サービスを使用して統合されている WebSocket ルートとルートキーのコレクション。このコレクションは、1 つ以上

のステージでデプロイできます。API メソッドは、登録されたカスタムドメイン名と関連付けることができるフロントエンド WebSocket 接続を通じて呼び出されます。

API デプロイ

API Gateway API の特定時点のスナップショット。クライアントが使用できるようにするには、デプロイが 1 つ以上の API ステージに関連付けられている必要があります。

API デベロッパー

API Gateway のデプロイを所有する AWS アカウント (プログラム的なアクセスもサポートするサービスプロバイダーなど)。

API エンドポイント

特定のリージョンにデプロイされる API Gateway の API のホスト名。ホスト名の形式は *{api-id}*.execute-api.*{region}*.amazonaws.comです。次のタイプの API エンドポイントがサポートされています。

- エッジ最適化 API エンドポイント
- プライベート API エンドポイント
- リージョン API エンドポイント

API キー

API Gateway が REST または WebSocket API を使用するアプリケーションデベロッパーを識別するために使用する英数字の文字列。ユーザーに代わって API Gateway が API キーを生成することも、CSV ファイルからインポートすることもできます。API キーを Lambda オーソライザーまたは使用量プランと一緒に使用して、API へのアクセスを制御します。

「API エンドポイント」を参照してください。

API 所有者

API デベロッパーを参照してください。

API ステージ

API (例: 'dev'、'prod'、'beta'、'v2' など) のライフサイクル状態への論理的な参照。API ステージは API ID とステージ名によって識別されます。

アプリケーションデベロッパー

AWS アカウントを持っているか持っていないアプリケーション作成者であり、API 開発者によってデプロイされた API を操作します。アプリケーション開発者は、顧客です。アプリケーション開発者は通常、API キーによって識別されます。

コールバック URL

新しいクライアントに WebSocket 接続経由で接続するときは、API ゲートウェイで統合を呼び出し、クライアントのコールバック URL を保存できます。次に、このコールバック URL を使用して、バックエンドシステムからクライアントにメッセージを送信することができます。

デベロッパーポータル

お客様の顧客による API 製品 (API Gateway 使用量プラン) の登録、検出、サブスクライブ、API キーの管理、API の使用状況メトリクスの表示を許可するアプリケーション。

エッジ最適化 API エンドポイント

主に AWS リージョン全体からのクライアントアクセスを容易にするために CloudFront ディストリビューションを使用しながら、指定されたリージョンにデプロイされる API Gateway API のデフォルトのホスト名。API リクエストは最寄りの CloudFront POP (Point of Presence) にルーティングされます。一般的にはこれによって地理的に分散したクライアントへの接続時間が改善します。

「API エンドポイント」を参照してください。

統合リクエスト

API Gateway の WebSocket API ルートまたは REST API メソッドの内部インターフェイス。このインターフェイスでは、ルートリクエストまたはパラメータの本文、およびメソッドリクエストの本文を、バックエンドで必要な形式にマップします。

統合レスポンス

API Gateway の WebSocket API ルートまたは REST API メソッドの内部インターフェイス。このインターフェイスでは、バックエンドから受け取ったステータスコード、ヘッダー、ペイロードをクライアントアプリに返されるレスポンス形式にマップします。

マッピングテンプレート

Velocity Template Language (VTL) のスクリプト。リクエスト本文をフロントエンドデータ形式からバックエンドデータ形式に変換したり、レスポンス本文をバックエンドデータ形式からフロントエンドデータ形式に変換したりします。マッピングテンプレートは、統合リクエストまたは統合レスポンスで指定できます。コンテキストおよびステージ変数として、実行時に利用できるデータを参照できます。

このマッピングは、統合を通じてヘッダーまたは本文をそのままクライアントからリクエストのバックエンドに渡す、単純なアイデンティティ転換とすることができます。レスポンスについても同じです。この場合、ペイロードはバックエンドからクライアントに渡されます。

メソッドリクエスト

API Gateway での API メソッドのパブリックインターフェイスであり、アプリケーションデベロッパーが API を介してバックエンドにアクセスするために送る必要のあるパラメータと本文が定義されています。

メソッドレスポンス

REST API のパブリックインターフェイスであり、アプリケーションデベロッパーが API からのレスポンスに求めるステータスコード、ヘッダー、本文モデルが定義されています。

Mock 統合

モック統合では、API のレスポンスは統合バックエンドを必要とすることなく、API Gateway から直接生成されます。API デベロッパーは、API Gateway がモック統合リクエストにどのように応答するかを決定します。そのため、特定のステータスコードとレスポンスを関連付ける、メソッドの統合リクエストと統合レスポンスを設定します。

モデル

リクエストまたはレスポンスペイロードのデータ構造を指定するデータスキーマ。API の厳密に型指定された SDK を生成するために、モデルが必要です。また、ペイロードの検証にも使用されます。モデルは、サンプルマッピングテンプレートを生成して本稼働マッピングテンプレートの作成を開始するうえで役立ちます。このモデルは有益ですが、マッピングテンプレートを作成するうえで必須ではありません。

プライベート API

「プライベート API エンドポイント」を参照してください。

プライベート API エンドポイント

インターフェイス VPC エンドポイントを介して公開された API エンドポイント。これにより、クライアントは VPC 内部で安全にプライベート API リソースにアクセスすることができます。プライベート API はパブリックインターネットからは分離されており、アクセス権限を付与されている API Gateway の VPC エンドポイントを使用してのみアクセスできます。

プライベート統合

クライアントが、リソースをパブリックインターネットに公開することなく、プライベート REST API エンドポイントを介してお客様の VPC 内のリソースにアクセスする、API Gateway の統合タイプです。

プロキシ統合

単純化された API Gateway 統合設定。HTTP プロキシ統合または Lambda プロキシ統合としてプロキシ統合を設定できます。

HTTP プロキシ統合の場合、API Gateway はフロントエンドと HTTP バックエンド間のリクエストとレスポンスの全体を渡します。Lambda プロキシ統合の場合、API Gateway はリクエスト全体をバックエンド Lambda 関数に入力として送信します。API Gateway は、その後、Lambda 関数の出力をフロントエンドの HTTP レスポンスに変換します。

REST API でのプロキシ統合は、プロキシリソースとともに最もよく使用されます。プロキシリソースは、greedy パス変数 (`{proxy+}` など) とキャッチオールの `ANY` メソッドの組み合わせによって表されます。

クイック作成

クイック作成を使用すると、HTTP API の作成を簡素化できます。クイック作成は、Lambda または HTTP 統合、デフォルトのキャッチオールルート、および変更を自動的にデプロイするように設定されたデフォルトステージを持つ API を作成します。詳細については、「the section called “AWS CLI を使用して HTTP API を作成する”」を参照してください。

リージョン API エンドポイント

指定されたリージョンにデプロイされ、同じ AWS リージョン内の EC2 インスタンスなどのクライアントに対応する目的である API のホスト名。API リクエストは、CloudFront ディストリビューションを経由せず、リージョン固有の API Gateway API を直接ターゲットとします。リージョン内のリクエストについては、リージョンエンドポイントは、CloudFront ディストリビューションへの不要なラウンドトリップをバイパスします。

さらに、レイテンシーに基づくルーティングをリージョン別エンドポイントに適用して、同じリージョン別 API エンドポイント設定を使用して、同じカスタムドメイン名をデプロイされた各 API に設定し、さらにレイテンシーに基づく DNS レコードを Route 53 に設定してクライアントリクエストをレイテンシーが最も低いリージョンにルーティングして API を複数のリージョンにデプロイすることができます。

「API エンドポイント」を参照してください。

ルート

API Gateway の WebSocket ルートは、メッセージの内容に基づいて、受信メッセージを AWS Lambda 関数などの特定の統合に転送するために使用されます。WebSocket API を定義するときは、ルートキーおよび統合バックエンドを指定します。ルートキーはメッセージの本文中にある属性です。受信メッセージでルートキーが一致した場合、統合バックエンドが呼び出されます。

デフォルトのルートは、一致しないルートキーに対して設定したり、ルーティングを実行してリクエストを処理するバックエンドコンポーネントにそのままメッセージを渡すプロキシモデルを指定したりできます。

ルートリクエスト

API Gateway での WebSocket API メソッドのパブリックインターフェイスであり、アプリケーションデベロッパーが API を介してバックエンドにアクセスするためにリクエストで送る必要のある本文が定義されています。

ルートレスポンス

WebSocket API のパブリックインターフェイスであり、アプリケーションデベロッパーが API Gateway に求めるステータスコード、ヘッダー、本文モデルが定義されています。

使用量プラン

使用量プランにより、選択した API クライアントは、1 つ以上のデプロイされた REST API または WebSocket API にアクセスできます。使用料プランを使用して、スロットリングとクォータ制限を設定できます。これらは、個々のクライアント API キーに適用されます。

WebSocket 接続

API Gateway は、クライアントと API Gateway 自体の間の永続的な接続を維持します。API Gateway と Lambda 関数などのバックエンド統合の間には永続的な接続はありません。バックエンドサービスは、クライアントから受信したメッセージの内容に基づいて、必要に応じて呼び出されます。

REST API と HTTP API 間で選択する

REST API と HTTP API は、いずれも RESTful API 製品です。REST API は HTTP API よりも多くの機能をサポートしていますが、HTTP API は低価格で提供できるように最小限の機能で設計されています。API キー、クライアントごとのスロットリング、リクエストの検証、AWS WAF の統合、プライベート API エンドポイントなどの機能が必要な場合は、REST API を選択します。REST API に含まれる機能が必要ない場合は、HTTP API を選択します。

次のセクションは、REST API および HTTP API で使用できるコア機能をまとめたものです。

[エンドポイントタイプ]

エンドポイントタイプは、API Gateway が API 用に作成するエンドポイントを指します。詳細については、「the section called “API Gateway エンドポイントタイプ”」を参照してください。

エンドポイントタイプ	REST API	HTTP API
エッジ最適化	✓	
リージョン別	✓	✓
プライベート	✓	

セキュリティ

API Gateway は、悪意のあるアクターやトラフィックの急増など、特定の脅威から API を保護するさまざまな方法を提供します。詳細については、「the section called “保護”」および「the section called “保護”」を参照してください。

セキュリティ機能	REST API	HTTP API
相互 TLS 認証	✓	✓
バックエンド認証用の証明書	✓	
AWS WAF	✓	

認証

API Gateway は API へのアクセスを制御し管理する複数のメカニズムをサポートしています。詳細については、the section called “アクセスコントロール”およびthe section called “アクセスコントロール”を参照してください。

認証オプション	REST API	HTTP API
IAM	✓	✓
リソースポリシー	✓	
Amazon Cognito	✓	✓ [1]

認証オプション	REST API	HTTP API
AWS Lambda 関数を使用したカスタム認証	✓	✓
JSON ウェブトークン (JWT) [2]		✓

[1] Amazon Cognito は JWT オーソライザーと共に使用できます。

[2] Lambda 認証 を使用して、REST API の JWT を検証できます。

API 管理

API キーやクライアントごとのレート制限などの API 管理機能が必要な場合は、REST API を選択します。詳細については、the section called “配布”、the section called “カスタムドメイン名”、およびthe section called “カスタムドメイン名”を参照してください。

機能	REST API	HTTP API
カスタムドメイン	✓	✓
API キー	✓	
クライアントごとのレート制限	✓	
クライアントごとの使用量調整	✓	

開発

API Gateway API の開発中、API の特性をいくつか決定することになります。これらの特性は API のユースケースによって異なります。詳細については、「the section called “開発”」および「the section called “開発”」を参照してください。

機能	REST API	HTTP API
CORS の設定	✓	✓
テスト呼び出し	✓	
キャッシュ	✓	
ユーザー制御のデプロイ	✓	✓
自動デプロイ		✓
カスタムゲートウェイレスポンス	✓	
Canary リリースのデプロイ	✓	
リクエストの検証	✓	
リクエストパラメータ変換	✓	✓
リクエスト本文変換	✓	

モニタリング

API Gateway は、API リクエストをログに記録し、API をモニタリングするためのいくつかのオプションをサポートしています。詳細については、the section called “監視”および the section called “監視”を参照してください。

機能	REST API	HTTP API
Amazon CloudWatch メトリクス	✓	✓
CloudWatch Logs へのアクセスログ	✓	✓
Amazon Data Firehose へのアクセスログ	✓	

機能	REST API	HTTP API
実行ログ	✓	
AWS X-Ray トレース	✓	

統合

統合により、API Gateway API がバックエンドリソースに接続されます。詳細については、the section called “統合”および the section called “統合”を参照してください。

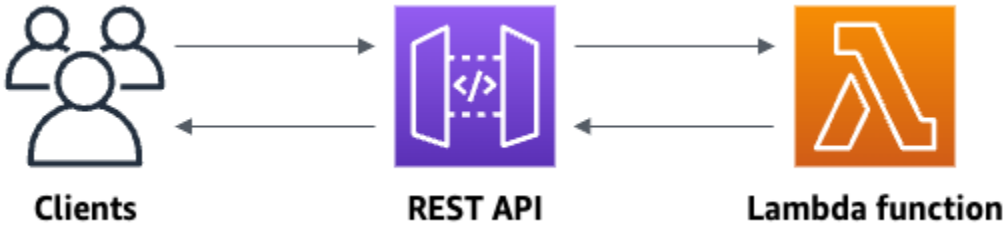
機能	REST API	HTTP API
パブリック HTTP エンドポイント	✓	✓
AWS のサービス	✓	✓
AWS Lambda 関数	✓	✓
Network Load Balancer とのプライベート統合	✓	✓
Application Load Balancer とのプライベート統合		✓
AWS Cloud Map とのプライベート統合		✓
Mock 統合	✓	

REST API コンソールの開始方法

この入門演習では、API Gateway REST API コンソールを使用してサーバーレス REST API を作成します。サーバーレス API を使用すると、サーバーのプロビジョニングや管理に時間をとられる事なく、アプリケーションに集中することができます。このエクササイズの所要時間は 20 分未満で、AWS の無料利用枠内で実行できます。

まずはじめに、Lambda コンソールを使用して Lambda 関数を作成します。次に、API Gateway REST API コンソールを使用してREST API を作成します。次に、API メソッドを作成し、Lambda プロキシ統合を使用して Lambda 関数と統合します。最後に、API をデプロイして呼び出します。

REST API を呼び出すと、API Gateway はリクエストを Lambda 関数にルーティングします。Lambda は関数を実行し、API Gateway に応答を返します。それから API Gateway はレスポンスを返します。



この演習を完了するには、AWS アカウント アカウントと、コンソールへのアクセス権がある AWS Identity and Access Management (IAM) ユーザーが必要です。詳細については、「API Gateway の開始方法の前提条件」を参照してください。

トピック

ステップ 1: Lambda 関数を作成する

API のバックエンドには Lambda 関数を使用します。Lambda は必要に応じてコードを実行し、1 日あたり数個のリクエストから 1 秒あたり数千のリクエストまで自動的にスケールします。

この演習では、Lambda コンソールで既定の Node.js 関数を使用します。

Lambda 関数を作成するには

1. Lambda コンソール (https://console.aws.amazon.com/lambda/) にサインインします。
2. [関数の作成] を選択します。

3. [基本的な情報] の [関数名] に「**my-function**」と入力します。
4. [Create function (関数の作成)] を選択します。

デフォルトの Lambda 関数コードは、次のようになります。

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

この演習の関数の応答が API Gateway が必要とする形式と一致している限り、Lambda 関数を変更できます。

デフォルトのレスポンスの本文 (`Hello from Lambda!`) を `The API Gateway REST API console is great!` に置き換えます。サンプル関数を呼び出すと、更新されたレスポンスとともにクライアントに `200` レスポンスが返されます。

ステップ 2: REST API を作成する

次に、ルートリソース (/) を使用して REST API を作成します。

REST API を作成するには

1. https://console.aws.amazon.com/apigateway で API Gateway コンソールにサインインします。
2. 次のいずれかを行います。
   - 最初の API を作成するには、[REST API] で [ビルド] を選択します。
   - 以前に API を作成した場合は、[API の作成] を選択し、REST API] の [ビルド] を選択します。
3. [API 名] に「**my-rest-api**」と入力します。
4. (オプション) [説明] に説明を入力します。
5. [API エンドポイントタイプ] を [リージョン別] に設定したままにします。
6. API の作成 を選択します。

ステップ 3: Lambda プロキシ統合を作成する

次に、ルートリソース (/) に REST API の API メソッドを作成し、プロキシ統合を使用して Lambda 関数とそのメソッドと統合します。Lambda プロキシ統合では、API Gateway はクライアントからの受信リクエストを直接Lambda関数に渡します。

Lambda プロキシ統合を作成するには

1. / リソースを選択し、[メソッドの作成] を選択します。
2. [メソッドタイプ] では、ANY を選択します。
3. [統合タイプ] で、[Lambda 関数] を選択します。
4. [Lambda プロキシ統合]を有効にします。
5. [Lambda 関数]に「**my-function**」と入力し、Lambda 関数を選択します。
6. [メソッドの作成] を選択します。

ステップ 4: API をデプロイする

次に、API デプロイを作成し、それをステージに関連付けます。

API をデプロイするには

1. [API のデプロイ] を選択します。
2. [ステージ] で [新規ステージ] を選択します。
3. [Stage name (ステージ名)] に **Prod** と入力します。
4. (オプション) [説明] に説明を入力します。
5. [デプロイ] を選択します。

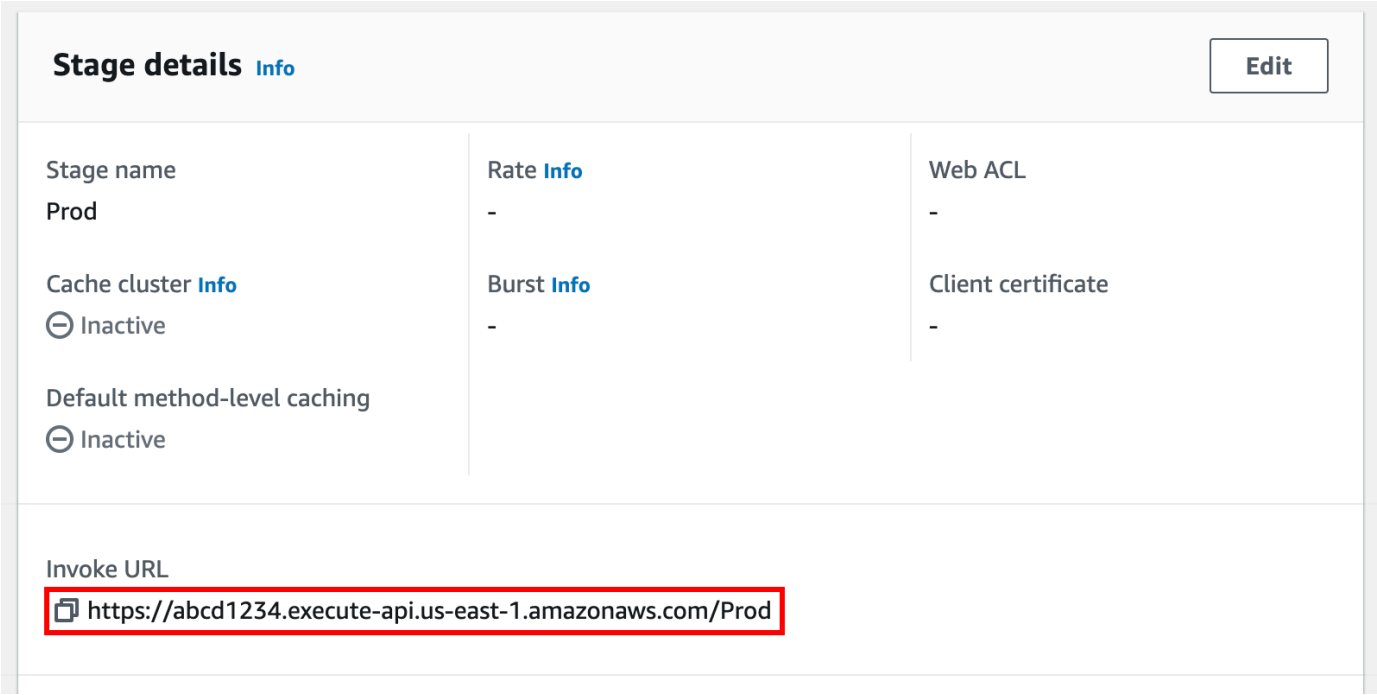
これで、クライアントは API を呼び出すことができます。API をデプロイする前に API をテストするには、オプションで ANY メソッドを選択し、[テスト] タブに移動して [テスト] を選択することもできます。

ステップ 5: API を呼び出す

API を呼び出すには

1. メインのナビゲーションペインで、[ステージ] を選択します。

2. [ステージの詳細] で、コピーアイコンを選択して API の呼び出し URL をコピーします。



3. ウェブブラウザに呼び出し URL を入力します。

   URL は次のようになります。`https://abcd123.execute-api.us-east-2.amazonaws.com/Prod`

   ブラウザが API に`GET`リクエストを送信します。

4. API の応答を確認します。ブラウザにテキスト`"The API Gateway REST API console is great!"`が表示されるはずです。

(オプション) ステップ 6: クリーンアップする

AWS アカウント にかかる不要なコストを回避するには、この演習で作成したリソースを削除します。次の手順では、REST API、Lambda 関数、および関連リソースを削除します。

REST API を削除するには

1. [リソース] ペインで、[API アクション]、[API の削除] を選択します。
2. [API の削除] ダイアログボックスに「確認」と入力し、[削除] を選択します。

Lambda 関数を削除するには

1. Lambda コンソール (https://console.aws.amazon.com/lambda/) にサインインします。
2. [関数] ページで、関数を選択します。[アクション]、[削除] の順に選択します。
3. [1 関数の削除] ダイアログボックスに「**delete**」と入力し、[削除] を選択します。

Lambda 関数のロググループを削除するには

1. Amazon CloudWatch コンソールで、[Log groups (ロググループ)] ページを開きます。
2. [ロググループ] ページで、関数のロググループ (`/aws/lambda/my-function`) を選択します。[アクション] で、[ロググループの削除] を選択します。
3. ロググループの削除ダイアログボックスで、[削除] をクリックします。

Lambda 関数の実行ロールを削除するには

1. IAM コンソールのロールページを開きます。
2. (オプション) [ロール] ページの検索ボックスに、「**my-function**」と入力します。
3. 関数のロール (例:`my-function-`*`31exxmpl`*) を選択し、[削除] を選択します。
4. [**my-function-*31exxmpl*** を削除しますか?] ダイアログボックスにロール名を入力し、[削除] を選択します。

> **Tip**
>
> AWS CloudFormation または AWS Serverless Application Model (AWS SAM) を使用して、AWS リソースの作成とクリーンアップを自動化できます。いくつかのサンプル AWS CloudFormationテンプレートについては、awsdocs GitHub リポジトリにある API Gateway のサンプルテンプレートを参照してください。

API Gateway の開始方法の前提条件

Amazon API Gateway を初めて使用する場合は、事前に以下のタスクを実行してください。

AWS アカウントへのサインアップ

AWS アカウントがない場合は、以下のステップを実行して作成します。

AWS アカウントにサインアップするには

1. https://portal.aws.amazon.com/billing/signup を開きます。
2. オンラインの手順に従います。

   サインアップ手順の一環として、通話呼び出しを受け取り、電話キーパッドで検証コードを入力するように求められます。

   AWS アカウントにサインアップすると、AWS アカウントのルートユーザーが作成されます。ルートユーザーには、アカウントのすべてのAWS のサービスとリソースへのアクセス権があります。セキュリティのベストプラクティスとして、ユーザーに管理アクセスを割り当て、ルートユーザーのみを使用してルートユーザーアクセスが必要なタスクを実行してください。

サインアップ処理が完了すると、AWS からユーザーに確認メールが送信されます。https://aws.amazon.com/ の [アカウント] をクリックして、いつでもアカウントの現在のアクティビティを表示し、アカウントを管理することができます。

管理アクセスを持つユーザーを作成する

AWS アカウント にサインアップしたら、AWS アカウントのルートユーザー をセキュリティで保護し、AWS IAM Identity Center を有効にして、管理ユーザーを作成します。これにより、日常的なタスクにルートユーザーを使用しないようにします。

AWS アカウントのルートユーザーをセキュリティで保護する

1. [ルートユーザー] を選択し、AWS アカウント のメールアドレスを入力して、アカウント所有者として AWS Management Console にサインインします。次のページでパスワードを入力します。

ルートユーザーを使用してサインインする方法については、AWS サインイン ユーザーガイドの「ルートユーザーとしてサインインする」を参照してください。

2. ルートユーザーの多要素認証 (MFA) を有効にします。

   手順については、IAM ユーザーガイドの「AWS アカウント のルートユーザーの仮想 MFA デバイスを有効にする (コンソール)」を参照してください。

管理アクセスを持つユーザーを作成する

1. IAM アイデンティティセンターを有効にします。

   手順については、「AWS IAM Identity Center ユーザーガイド」の「AWS IAM Identity Center の有効化」を参照してください。

2. IAM アイデンティティセンターで、ユーザーに管理アクセスを付与します。

   IAM アイデンティティセンターディレクトリ をアイデンティティソースとして使用するチュートリアルについては、「AWS IAM Identity Center ユーザーガイド」の「デフォルト IAM アイデンティティセンターディレクトリを使用したユーザーアクセスの設定」を参照してください。

管理アクセス権を持つユーザーとしてサインインする

- IAM アイデンティティセンターのユーザーとしてサインインするには、IAM アイデンティティセンターのユーザーの作成時に E メールアドレスに送信されたサインイン URL を使用します。

  IAM Identity Center ユーザーを使用してサインインする方法については、AWS サインイン ユーザーガイドの「AWS アクセスポータルにサインインする」を参照してください。

追加のユーザーにアクセス権を割り当てる

1. IAM アイデンティティセンターで、最小特権のアクセス許可を適用するというベストプラクティスに従ったアクセス許可セットを作成します。

   手順については、「AWS IAM Identity Center ユーザーガイド」の「権限設定を作成する」を参照してください。

2. グループにユーザーを割り当て、そのグループにシングルサインオンアクセス権を割り当てます。

手順については、「AWS IAM Identity Center ユーザーガイド」の「グループの参加」を参照してください。

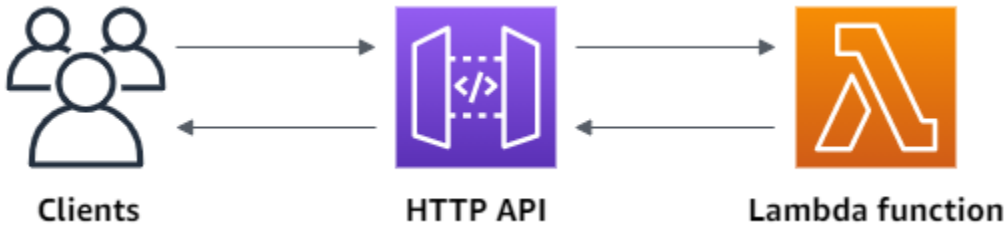
API Gateway の開始方法

この入門エクササイズでは、サーバーレス API を作成します。サーバーレス API を使用すると、サーバーのプロビジョニングや管理に時間をとられる事なく、アプリケーションに集中することができます。このエクササイズの所要時間は 20 分未満で、AWS の無料利用枠内で実行できます。

はじめに、AWS Lambda コンソールを使用して Lambda 関数を作成します。次に、API Gateway コンソールを使用して HTTP API を作成します。次に、API を呼び出します。

> **Note**
>
> この演習では、HTTP API を使用します。API Gateway は、より多くの機能を含む REST API もサポートしています。REST API を使用するチュートリアルについては、「the section called “REST API コンソールの開始方法”」を参照してください。
> HTTP API と REST API の違いの詳細については、「the section called “REST API と HTTP API 間で選択する ”」を参照してください。

HTTP API を呼び出すと、API Gateway はリクエストを Lambda 関数にルーティングします。Lambda は Lambda 関数を実行し、API Gateway に応答を返します。それから API Gateway はレスポンスを返します。



このエクササイズを完了するには、AWS アカウントと、コンソールへのアクセス権がある AWS Identity and Access Management ユーザーが必要です。詳細については、「前提条件」を参照してください。

トピック

- ステップ 1: Lambda 関数を作成する
- ステップ 2: HTTP API を作成する
- ステップ 3: API をテストする
- （オプション）ステップ 4: クリーンアップする

- 次のステップ

ステップ 1: Lambda 関数を作成する

API のバックエンドには Lambda 関数を使用します。Lambda は必要に応じてコードを実行し、1 日あたり数個のリクエストから 1 秒あたり数千のリクエストまで自動的にスケールします。

この例では、Lambda コンソールから既定の Node.js 関数を使用します。

Lambda 関数を作成するには

1. Lambda コンソール (https://console.aws.amazon.com/lambda/) にサインインします。
2. [関数の作成] を選択します。
3. [関数名] に「**my-function**」と入力します。
4. [関数の作成] を選択します。

この例の関数は、クライアントへの応答200とテキストHello from Lambda!を返します 。

関数の応答が API Gateway が必要とする形式と一致している限り、Lambda 関数を変更できます。

デフォルトの Lambda 関数コードは、次のようになります。

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

ステップ 2: HTTP API を作成する

次に、HTTP API を作成します。API Gateway は、REST API と WebSocket API もサポートしますが、このエクササイズでは HTTP API が最適です。REST API は HTTP API よりも多くの機能をサポートしますが、この演習では、これらの機能は必要ありません。HTTP API は低価格で提供できるように最小限の機能で設計されています。WebSocket API は、全二重通信のためにクライアントとの持続的接続を維持します。この例では必須ではありません。

HTTP API は、Lambda 関数の HTTP エンドポイントを提供します。API Gateway は Lambda 関数にリクエストをルーティングし、関数の応答をクライアントに返します。

HTTP API を作成するには

1. https://console.aws.amazon.com/apigateway で API Gateway コンソールにサインインします。
2. 次のいずれかを行ってください。
   - 最初の API を作成するには、[HTTP API] で [構築] を選択します。
   - 以前に API を作成した場合は、[API の作成] を選択し、[HTTP API] の [構築] を選択します。
3. [統合] で、[統合の追加] を選択します。
4. Lambda を選択します。
5. [Lambda 関数] に「**my-function**」と入力します。
6. [API 名] に「**my-http-api**」と入力します。
7. [Next (次へ)] を選択します。
8. API Gateway が作成したルートを確認し、[次へ] を選択します。
9. API Gateway によって作成されるステージを確認し 、[Next] を選択します。
10. [Create] を選択します。

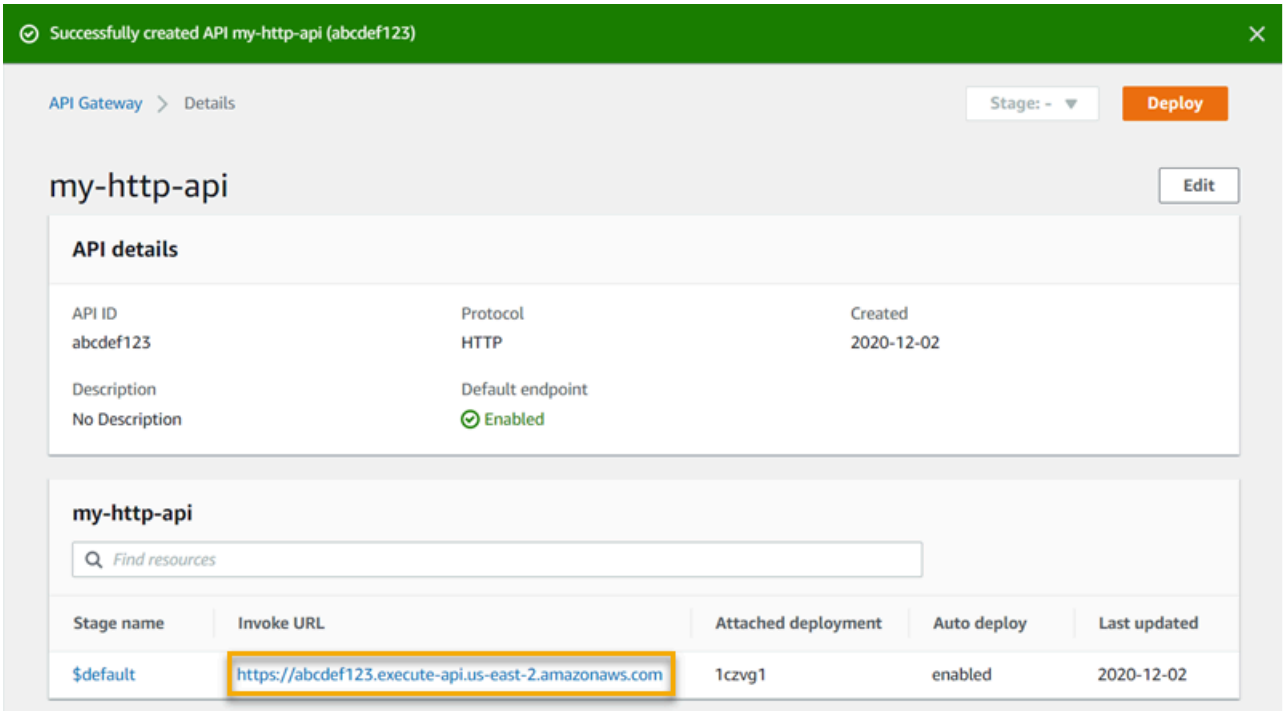
これで、クライアントからのリクエストを受信できる Lambda 統合で HTTP API を作成しました。

ステップ 3: API をテストする

次に、API をテストして作動していることを確認します。シンプルにテストをするため、ウェブブラウザを使用して API を呼び出します。

API をテストするために

1. API Gateway コンソール (https://console.aws.amazon.com/apigateway) にサインインします。
2. API を選択します。
3. API の呼び出し URL を書き留めます。



4. API の呼び出し URL をコピーし、Web ブラウザーに入力します。呼び出し URL に Lambda 関数の名前を追加して、Lambda 関数を呼び出します。デフォルトでは、API Gateway コンソールは Lambda 関数`my-function`と同じ名前のルートを作成します。

   URL は次のようになります。`https://abcdef123.execute-api.us-east-2.amazonaws.com/my-function`

   ブラウザが API に`GET`リクエストを送信します。

5. API の応答を確認します。ブラウザにテキスト`"Hello from Lambda!"`が表示されるはずです。

（オプション）ステップ 4: クリーンアップする

不要なコストを回避するには、このエクササイズで作成したリソースを削除します。次の手順では、HTTP API、Lambda 関数、および関連リソースを削除します。

HTTP API を削除するには

1. https://console.aws.amazon.com/apigateway で API Gateway コンソールにサインインします。
2. [API] ページで、API を選択します。[Actions] を選択して、[Delete] を選択します。
3. [削除] を選択します。

Lambda 関数を削除するには

1. Lambda コンソール (https://console.aws.amazon.com/lambda/) にサインインします。
2. [関数] ページで、関数を選択します。[Actions] を選択して、[Delete] を選択します。
3. [削除] を選択します。

Lambda 関数のロググループを削除するには

1. Amazon CloudWatch コンソールで、[ロググループページ]を開きます。
2. [ロググループ] ページで、関数のロググループ (`/aws/lambda/my-function`) を選択します。[Actions] (アクション) を選択してから、[Delete log group] (ロググループの削除) を選択します。
3. [削除] を選択します。

Lambda 関数の実行ロールを削除するには

1. AWS Identity and Access Management コンソールの [Roles] (ロール) ページを開きます。
2. 関数のロールを選択します (例: `my-function-31exxmpl`)。
3. [ロールの削除] を選択します。
4. [はい、削除します] を選択します。

AWS CloudFormation または AWS SAM を使用して、AWS リソースの作成とクリーンアップを自動化できます。AWS CloudFormation テンプレートの例については、「サンプル AWS CloudFormation テンプレート」を参照してください。

次のステップ

この例では、AWS Management Console を使用してシンプルな HTTP API を作成しました。HTTP API は Lambda 関数を呼び出し、クライアントに応答を返します。

API Gateway の使用を続けたまま、次のステップに進んでください。

- 以下を含む API 統合の追加のタイプを設定する
  - HTTP エンドポイント
  - VPC 内のプライベートリソース (Amazon ECS サービスなど)

  - Amazon Simple Queue Service、AWS Step Functions、Kinesis Data Streams などの AWS サービス
- API へのアクセスを制御します。
- API のログの有効化
- API のスロットリングを設定する
- カスタムドメインを作成して設定する

コミュニティから Amazon API Gateway に関するヘルプを参照するには、API Gateway Discussion Forum を参照してください。このフォーラムにアクセスするには、AWS へのサインインが必要になることがあります。

AWS から直接 API Gateway のサポートを得るには、AWS Support ページでサポートオプションを参照してください。

また、よくある質問 (FAQ) を参照したり、直接お問い合わせいただくこともできます。

Amazon API Gateway のチュートリアルとワークショップ

以下のチュートリアルとワークショップでは実践的な演習を提供し、API Gateway の学習に役立ちます。

REST API チュートリアル

- AWS Lambda 統合を選択するチュートリアル
- チュートリアル: サンプルをインポートして REST API を作成する
- HTTP 統合を選択するチュートリアル
- チュートリアル: API Gateway のプライベート統合を使用して REST API をビルドする
- チュートリアル: AWS 統合を使用して API Gateway REST API を構築する
- チュートリアル: 2 つの AWS サービス統合と 1 つの Lambda 非プロキシ統合を使用して計算ツールの REST API を作成する
- チュートリアル: API Gateway で REST API を Amazon S3 のプロキシとして作成する
- チュートリアル: API Gateway で REST API を Amazon Kinesis のプロキシとして作成する
- チュートリアル: AWS SDKまたは AWS CLI を使用してエッジ最適化 API を作成する
- チュートリアル: プライベート REST API を構築する

HTTP API チュートリアル

- チュートリアル: Lambda と DynamoDB を使用した CRUD API の構築
- チュートリアル: Amazon ECS サービスへのプライベート統合を使用した HTTP API の構築

WebSocket API のチュートリアル

- チュートリアル: WebSocket API、Lambda、DynamoDB を使用したサーバーレスチャットアプリケーションの構築

ワークショップ

- サーバーレスのウェブアプリケーションの構築
- サーバーレスアプリケーションの CI/CD
- サーバレスセキュリティワークショップ

- サーバーレス ID 管理、認証、承認
- Amazon API Gateway ワークショップ

Amazon API Gateway REST API チュートリアル

以下のチュートリアルでは API Gateway REST API の学習に役立つ実践的な演習を提供します。

トピック

- AWS Lambda 統合を選択するチュートリアル
- チュートリアル: サンプルをインポートして REST API を作成する
- HTTP 統合を選択するチュートリアル
- チュートリアル: API Gateway のプライベート統合を使用して REST API をビルドする
- チュートリアル: AWS 統合を使用して API Gateway REST API を構築する
- チュートリアル: 2 つの AWS サービス統合と 1 つの Lambda 非プロキシ統合を使用して計算ツールの REST API を作成する
- チュートリアル: API Gateway で REST API を Amazon S3 のプロキシとして作成する
- チュートリアル: API Gateway で REST API を Amazon Kinesis のプロキシとして作成する
- チュートリアル: AWS SDKまたは AWS CLI を使用してエッジ最適化 API を作成する
- チュートリアル: プライベート REST API を構築する

AWS Lambda 統合を選択するチュートリアル

Lambda 統合を使用して API を構築するには、Lambda プロキシ統合または Lambda 非プロキシ統合を使用できます。

Lambda プロキシ統合では、Lambda 関数への入力は、リクエストヘッダー、パス変数、クエリ文字列パラメータ、本文、API 設定データの任意の組み合わせとして表現されます。選択する必要があるのは、Lambda 関数のみです。API Gateway により統合リクエストおよび統合レスポンスが自動で設定されます。一度セットアップすれば、API メソッドは、既存の設定を変更することなく進化できます。これが可能なのは、バックエンドの Lambda 関数が受信リクエストデータを解析し、クライアントに応答するためです。

Lambda 非プロキシ統合では、Lambda 関数への入力が統合リクエストペイロードとして指定されていることを確認する必要があります。クライアントから提供された入力データは、リクエストパラ

メータとして、適切な統合リクエストボディにマッピングする必要があります。また、クライアントで指定したリクエストボディを Lambda 関数が認識する形式に変換する必要がある場合もあります。

Lambda プロキシまたは Lambda 非プロキシ統合のいずれかで、API を作成したアカウントとは異なるアカウントで Lambda 関数を使用できます。

トピック

- チュートリアル: Lambda プロキシ統合を使用した Hello World REST API の構築
- チュートリアル: Lambda 非プロキシ統合を使用して API Gateway REST API をビルドする
- チュートリアル:クロスアカウント Lambda プロキシ統合を使用して API Gateway REST API をビルドする

チュートリアル: Lambda プロキシ統合を使用した Hello World REST API の構築

Lambda プロキシ統合は、軽量で柔軟な API Gateway API 統合タイプであり、これを使用すると、API メソッドまたは API 全体を Lambda 関数と統合できます。Lambda 関数は、Lambda がサポートする任意の言語で記述できます。これはプロキシ統合であるため、API を再デプロイする必要がなく、いつでも Lambda 関数の実装を変更できます。

このチュートリアルでは、以下の作業を行います。

- 「Hello, World!」 Lambda 関数を API のバックエンドにします。
- 「Hello, World!」 Lambda プロキシ統合による API。

トピック

- 「Hello, World!」 Lambda 関数
- 「Hello, World!」 API
- API をデプロイしてテストする

「Hello, World!」 Lambda 関数

「Hello, World!」を作成するには Lambda コンソールで Lambda 関数を作成します。

1. Lambda コンソール (https://console.aws.amazon.com/lambda/) にサインインします。
2. AWS ナビゲーションバーで、AWS リージョンを選択します。

> **Note**
>
> Lambda 関数を作成したリージョンを書き留めます。これは、API を作成するときに必要になります。

3. ナビゲーションペインで、[関数] を選択します。
4. [関数の作成] を選択します。
5. Author from scratch を選択します。
6. [基本的な情報] で、以下の作業を行います。
   a. [関数名] に **GetStartedLambdaProxyIntegration** と入力します。
   b. [ランタイム] で、サポートされている最新の Node.js または Python ランタイムのいずれかを選択します。
   c. [Permissions] (許可) で、[Change default execution role] (デフォルトの実行ロールの変更) を展開します。[実行ロール]] ドロップダウンリストで、[AWS ポリシーテンプレートから新しいロールを作成] を選択します。
   d. [ロール名] に **GetStartedLambdaBasicExecutionRole** と入力します。
   e. [Policy templates] フィールドは空白のままにします。
   f. [関数の作成] を選択します。
7. インラインコードエディタの [Function code (関数コード)] に、以下のコードをコピーして貼り付けます。

   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. [デプロイ] を選択します。

「Hello, World!」 API

ここで、「Hello, World!」に API を作成します。API Gateway コンソールを使用して Lambda 関数を実行します。

「Hello, World!」を作成するには API

1. https://console.aws.amazon.com/apigateway で API Gateway コンソールにサインインします。
2. API Gateway を初めて使用する場合は、サービスの特徴を紹介するページが表示されます。[REST API] で、[構築] を選択します。[Create Example API (サンプル API の作成)] がポップアップ表示されたら、[OK] を選択します。

API Gateway を使用するのが初めてではない場合、[Create API] (API を作成)を選択します。[REST API] で、[構築] を選択します。

3. [API 名] に「**LambdaProxyAPI**」と入力します。
4. (オプション) [説明] に説明を入力します。
5. [API エンドポイントタイプ] を [リージョン別] に設定したままにします。
6. API の作成 を選択します。

API を作成したら、リソースを作成します。通常、API リソースはアプリケーションロジックに従ってリソースツリーに整理されます。この例では、/helloworld リソースを作成します。

リソースを作成するには

1. / リソースを選択し、[メソッドを作成] を選択します。
2. [プロキシのリソース] はオフのままにします。
3. [リソースパス] は / のままにします。
4. [リソース名] に「**helloworld**」と入力します。
5. [CORS (Cross Origin Resource Sharing)] はオフのままにします。
6. [リソースの作成] を選択します。

プロキシ統合では、任意の HTTP メソッドを表すすべての ANY メソッドをキャッチオールがらリクエスト全体がそのままバックエンド Lambda 関数に送信されます。実際の HTTP メソッドは、実行時にクライアントによって指定されます。ANY メソッドでは、単一の API メソッドのセットアップを DELETE、GET、HEAD、OPTIONS、PATCH、POST および PUT のサポートされるすべての HTTP メソッドに使用できます。

ANY メソッドを作成するには

1. /helloworld リソースを選択し、[メソッドを作成] を選択します。
2. [メソッドタイプ] で、[ANY] を選択します。
3. [統合タイプ] で、[Lambda 関数] を選択します。
4. [Lambda プロキシ統合]を有効にします。
5. [Lambda 関数] で、Lambda 関数を作成した AWS リージョンを選択し、関数名を入力します。

6. 29 秒のデフォルトのタイムアウト値を使用するには、[デフォルトタイムアウト] をオンのままにします。カスタムのタイムアウトを設定するには、[デフォルトタイムアウト] を選択してから、タイムアウト値を `50 ~ 29000` ミリ秒の間で入力します。
7. [メソッドの作成] を選択します。

API をデプロイしてテストする

API をデプロイするには

1. [API のデプロイ] を選択します。
2. [ステージ] で [新規ステージ] を選択します。
3. [Stage name (ステージ名)] に **`test`** と入力します。
4. (オプション) [説明] に説明を入力します。
5. [デプロイ] を選択します。
6. [ステージの詳細] で、コピーアイコンを選択して API の呼び出し URL をコピーします。

ブラウザと cURL を使用して Lambda プロキシ統合で API をテストする

API をテストするためにブラウザまたは cURL を使用できます。

クエリ文字列パラメータのみを使用して `GET` リクエストをテストする場合は、API の `helloworld` リソースの URL をブラウザのアドレスバーに入力できます。

API の `helloworld` リソースの URL を作成するには、リソース `helloworld` とクエリ文字列パラメータ `?greeter=John` を呼び出し URL に追加します。URL は次のようになります。

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

それ以外のメソッドの場合は、POSTMAN や cURL などの高度な REST API テストユーティリティを使用する必要があります。このチュートリアルでは cURL を使用します。以下の cURL コマンドの例は、cURL がコンピュータにインストールされていることを前提としています。

cURL を使用してデプロイした API をテストするには

1. ターミナルウィンドウを開きます。
2. 次の cURL コマンドをコピーしてターミナルウィンドウに貼り付け、呼び出し URL を前のステップでコピーした URL に置き換えて、URL の末尾に **`/helloworld`** を追加します。

> **Note**
>
> Windows でコマンドを実行している場合は、代わりに次の構文を使用してください。
>
> ```
> curl -v -X POST "https://r275xc9bmd.execute-api.us-east-1.amazonaws.com/
> test/helloworld" -H "content-type: application/json" -d "{ \"greeter\":
>  \"John\" }"
> ```

a. ?greeter=John のクエリ文字列パラメータを使用して API を呼び出すには

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

b. greeter:John のヘッダーパラメータを使用して API を呼び出すには

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

c. {"greeter":"John"} の本文を使用して API を呼び出すには

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

すべてのケースで、出力は、次のレスポンス本文を持つ 200 レスポンスです。

```
Hello, John!
```

チュートリアル: Lambda 非プロキシ統合を使用して API Gateway REST API をビルドする

このチュートリアルでは、API Gateway コンソールを使用して、クライアントが Lambda 非プロキシ統合 (カスタム統合とも呼ばれます) で Lambda 関数を呼び出すことができる API を構築しま

す。AWS Lambda および Lambda 関数の詳細については、AWS Lambda デベロッパーガイドを参照してください。

分かりやすくするために、API 設定が最小限のシンプルな Lambda 関数で、カスタムの Lambda 統合を使用して API Gateway API を構築する手順について説明します。必要に応じて、いくつかのロジックについて説明します。カスタムの Lambda 統合の詳細な例については、「チュートリアル: 2 つの AWS サービス統合と 1 つの Lambda 非プロキシ統合を使用して計算ツールの REST API を作成する」を参照してください。

API を作成する前に、次に説明されているように AWS Lambda で Lambda 関数を作成して、Lambda バックエンドをセットアップします。

トピック

- Lambda 非プロキシ統合用の Lambda 関数の作成
- Lambda 非プロキシ統合を使用して API を作成する
- API メソッドの呼び出しをテストする
- API をデプロイする
- デプロイステージで API をテストする
- クリーンアップ

Lambda 非プロキシ統合用の Lambda 関数の作成

> **Note**
>
> Lambda 関数の作成により、AWS アカウントに料金が請求される場合があります。

このステップでは、「Hello, World!」のような Lambda 関数をカスタムの Lambda 統合用に作成します。このウォークスルーでは、この関数 `GetStartedLambdaIntegration` が呼び出されます。

この `GetStartedLambdaIntegration` の Lambda 関数の実装は次のとおりです。

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}
```

カスタムの Lambda 統合の場合、API Gateway は統合リクエストボディとして入力をクライアントから Lambda 関数に渡します。Lambda 関数ハンドラの `event` オブジェクトが入力です。

Lambda 関数はシンプルです。`event`、`name`、`city`、および `time` プロパティの入力オブジェクト (day) を解析します。その後、JSON オブジェクト ({"message":greeting}) として、発信者に挨拶を返します。このメッセージは "Good [morning|afternoon|day], [*name*|you] in [*city*|World]. Happy *day*!" パターンです。これは、Lambda 関数への入力が、以下の JSON オブジェクトのいずれかであることを前提としています。

```
{
  "city": "...",
```

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

詳細については、「AWS Lambda 開発者ガイド」を参照してください。

さらに、関数は `console.log(...)` を呼び出して、その実行を Amazon CloudWatch に記録します。これは、関数のデバッグ時に呼び出しをトレースする場合に役立ちます。`GetStartedLambdaIntegration` 関数で呼び出しを記録できるようにするには、Lambda 関数の適切なポリシーで IAM ロールを設定して CloudWatch ストリームを作成し、そのストリームにログエントリを追加します。Lambda コンソールに従って、必要な IAM ロールとポリシーを作成します。

OpenAPI ファイルから API をインポートする場合など、API Gateway コンソールを使用せずに API をセットアップする場合には、必要に応じて Lambda 関数を呼び出すための API Gateway の呼び出しロールとポリシーを明示的に作成し、設定する必要があります。Lambda 呼び出しと API Gateway API の実行ロールの設定の詳細については、「IAM アクセス許可により API へのアクセスを制御する」を参照してください。

Lambda プロキシ統合用の Lambda 関数、`GetStartedLambdaProxyIntegration` と比較すると、`GetStartedLambdaIntegration` Lambda カスタム統合用の Lambda 関数は、API Gateway API 統合リクエストボディからの入力のみを受け取ります。この関数では、JSON オブジェクト、文字列、数値、ブール値、またはバイナリ BLOB の出力を返すことができます。対照的に、Lambda プロキシ統合の Lambda 関数では、リクエストデータから入力を取り込みますが、特定の JSON オブジェクトの出力を返す必要があります。Lambda カスタム 統合の `GetStartedLambdaIntegration` 関数では、入力として API リクエストパラメータを指定することができますが、クライアントリクエストをバックエンドに転送する前に、API Gateway で、必要な API リクエストパラメータが統合リクエストボディにマッピングされていることを前提としています。そのためには、API デベロッパーは、マッピングテンプレートを作成し、API 作成時に API メソッドで設定する必要があります。

`GetStartedLambdaIntegration` Lambda 関数の作成します。

`GetStartedLambdaIntegration` Lambda カスタム統合用の Lambda 関数を作成するには

1. https://console.aws.amazon.com/lambda/ で、AWS Lambda コンソールを開きます。
2. 次のいずれかを行ってください。

   - ウェルカムページが表示されたら、[Get Started Now]、[Creae function] の順に選択します。

- [Lambda > Functions (Lambda > 関数)] リストページが表示されたら、[Create function (関数の作成)] を選択します。

3. [Author from scratch] を選択します。
4. [一から作成] ペインで、次の操作を行います。

   a. [Name (名前)] に、Lambda 関数名として **GetStartedLambdaIntegration** と入力します。

   b. [ランタイム] で、サポートされている最新の Node.js または Python ランタイムのいずれかを選択します。

   c. [Permissions] (許可) で、[Change default execution role] (デフォルトの実行ロールの変更) を展開します。[実行ロール]] ドロップダウンリストで、[AWS ポリシーテンプレートから新しいロールを作成] を選択します。

   d. [Role name] に、ロール名 (**GetStartedLambdaIntegrationRole** など) を入力します。

   e. [Policy templates] で、[Simple microservice permissions] を選択します。

   f. [関数の作成] を選択します。

5. [関数の設定] ペインの [関数コード] で、以下の作業を行います。

   a. このセクションの冒頭に表示された Lambda 関数コードをコピーし、インラインコードエディターに貼り付けます。

   b. このセクションのその他のフィールドは、デフォルト設定のままにしておきます。

   c. [デプロイ] を選択します。

6. 新しく作成した関数をテストするには、[テスト] タブを選択します。

   a. イベント名()で、**HelloWorldTest** と入力します。

   b. [イベント JSON] では、デフォルトのコードを次のコードに置き換えます。

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

   c. [テスト] を選択して関数を呼び出します。[実行結果: 成功] セクションが表示されます。[詳細] を展開すると、次の出力が表示されます。

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

出力は CloudWatch Logs にも書き込まれます。

また、他にも、IAM コンソールを使用して、Lambda 関数と共に作成された IAM ロール (GetStartedLambdaIntegrationRole) を表示することができます。この IAM ロールには、2 つのインラインポリシーがアタッチされています。1 つは、Lambda を実行する上で最も基本的なアクセス許可が指定されているポリシーです。これにより、Lambda 関数が作成されたリージョン内のアカウントのすべての CloudWatch リソース CreateLogGroup に対して CloudWatch を呼び出すことができます。また、このポリシーでは、GetStartedLambdaIntegration Lambda 関数で CloudWatch ストリームやログ記録イベントを作成することもできます。

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

他のポリシードキュメントは、この例で使用されていない他の AWS のサービスの呼び出しに適用されます。この手順はスキップできます。

この IAM ロールには、信頼されたエンティティ (lambda.amazonaws.com) が関連付けられています。信頼関係は次のとおりです。

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

この信頼関係およびインラインポリシーを組み合わせることで、Lambda 関数を使用して、CloudWatch Logs のログイベントに `console.log()` 関数を呼び出すことができます。

Lambda 非プロキシ統合を使用して API を作成する

Lambda 関数 (GetStartedLambdaIntegration) を作成してテストしたら、API Gateway API からこの関数を公開できるようになります。例として、汎用的な HTTP メソッドで Lambda 関数を公開します。リクエストボディ、URL パス変数、クエリ文字列、ヘッダーを使用して、クライアントから必要な入力データを受け取ります。API の API Gateway のリクエスト検証を有効にして、必要なデータがすべて適切に定義され、指定されていることを確認します。API Gateway のマッピングテンプレートを設定して、クライアントで指定されたリクエストデータをバックエンドの Lambda 関数で定められた有効な形式に変換します。

Lambda 非プロキシ統合を使用して API を作成するには

1. https://console.aws.amazon.com/apigateway で API Gateway コンソールにサインインします。
2. API Gateway を初めて使用する場合は、サービスの特徴を紹介するページが表示されます。[REST API] で、[構築] を選択します。[Create Example API (サンプル API の作成)] がポップアップ表示されたら、[OK] を選択します。

   API Gateway を使用するのが初めてではない場合、[Create API] (API を作成)を選択します。[REST API] で、[構築] を選択します。
3. [API 名] に「**LambdaNonProxyAPI**」と入力します。
4. (オプション) [説明] に説明を入力します。

5. [API エンドポイントタイプ] を [リージョン別] に設定したままにします。
6. API の作成 を選択します。

API を作成したら、/{city} リソースを作成します。これは、クライアントから入力を受け付けるパス変数を持つリソースの例です。後で、マッピングテンプレートを使用してこのパス変数を Lambda 関数の入力にマッピングします。

リソースを作成するには

1. [リソースの作成] を選択します。
2. [プロキシのリソース] はオフのままにします。
3. [リソースパス] は / のままにします。
4. [リソース名] に「**{city}**」と入力します。
5. [CORS (Cross Origin Resource Sharing)] はオフのままにします。
6. [リソースの作成] を選択します。

/{city} リソースを作成したら、ANY メソッドを作成します。ANY HTTP 動詞は、実行時にクライアントより送信される有効な HTTP メソッドのクライアントのプレースホルダーです。この例では、ANY メソッドが、Lambda カスタム 統合と Lambda プロキシ統合に使用できることを示します。

ANY メソッドを作成するには

1. /{city} リソースを選択し、[メソッドを作成] を選択します。
2. [メソッドタイプ] で、[ANY] を選択します。
3. [統合タイプ] で、[Lambda 関数] を選択します。
4. [Lambda プロキシ統合] はオフのままにしておきます。
5. [Lambda 関数] で、Lambda 関数を作成した AWS リージョンを選択し、関数名を入力します。
6. [メソッドリクエストの設定] を選択します。

   次に、URL パス変数、クエリ文字列パラメータ、ヘッダーに対するリクエストの検証をオンにして、すべての必要なデータが定義されていることを確認します。この例では、`time` クエリ文字列パラメータと `day` ヘッダーを作成します。
7. [リクエストの検証] で、[クエリ文字列パラメータおよびヘッダーを検証] を選択します。
8. [URL クエリ文字列パラメータ] を選択してから、次の操作を行います。

a. [クエリ文字列の追加] を選択します。

b. [名前] に**time**と入力します。

c. [必須] をオンにします。

d. [キャッシュ] はオフのままにします。

9. [HTTP リクエストヘッダー] を選択し、次の操作を行います。

a. [ヘッダーの追加] を選択します。

b. [名前] に**day**と入力します。

c. [必須] をオンにします。

d. [キャッシュ] はオフのままにします。

10. [メソッドの作成] を選択します。

リクエストの検証を有効にしたら、バックエンドの Lambda 関数の要求に応じて、受信リクエストを JSON ペイロードに変換するための本文マッピングテンプレートを追加することで、ANY メソッドの統合リクエストを設定します。

統合リクエストを設定するには

1. [統合リクエスト] タブの [統合リクエストの設定] で、[編集] を選択します。
2. [リクエスト本文のパススルー] で、[テンプレートが定義されていない場合 (推奨)] を選択します。
3. [マッピングテンプレート] を選択します。
4. [マッピングテンプレートの追加] を選択します。
5. [コンテンツタイプ] に、「**application/json**」と入力します。
6. [テンプレート本文] に、次のコードを入力します。

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

7. [Save] を選択します。

API メソッドの呼び出しをテストする

デプロイ前に API の呼び出しをテストするテスト機関が API Gateway コンソールに表示されます。コンソールのテスト機能を使用して API をテストするには、以下のリクエストを送信します。

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

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

このテストリクエストで、ANY を POST、{city} を Seattle に設定し、Wednesday を day ヘッダー値、"John" を callerName 値として割り当てます。

ANY メソッドをテストするには

1. [テスト] タブを選択します。タブを表示するには、右矢印ボタンを選択する必要がある場合があります。
2. [メソッドタイプ] では、POST を選択します。
3. [パス] の [city] に、「**Seattle**」と入力します。
4. [クエリ文字列] に「**time=morning**」と入力します。
5. [ヘッダー]] に「**day:Wednesday**」と入力します。
6. [リクエスト本文] に、「**{ "callerName": "John" }**」と入力します。
7. [Test (テスト)] を選択します。

返されたレスポンスペイロードが次のようになっていることを確認します。

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

ログを表示して、API Gateway によるリクエストおよびレスポンスの処理方法を調べることもできます。

```
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
```

このログには、マッピング前の受信リクエスト、およびマッピング後の統合リクエストが示されます。テストに失敗した場合、このログは、元の出力が正しいかどうか、またはマッピングテンプレートが正しいかどうかを評価する上で役立ちます。

API をデプロイする

テスト呼び出しはシミュレーションのため、制限があります。たとえば、API で実施される任意の認証メカニズムはバイパスされます。リアルタイムで API の実行をテストするには、最初に API をデプロイする必要があります。API をデプロイするには、ステージを作成し、その時点の API のスナップショットを作成します。また、ステージ名では、API のデフォルトのホスト名の後にベースパスが定義されます。API のルートリソースはステージ名の後に追加されます。API を変更する場合は、変更が適用される前に既存または新しいステージに再デプロイする必要があります。

API をステージにデプロイするには

1. [API のデプロイ] を選択します。
2. [ステージ] で [新規ステージ] を選択します。
3. [Stage name (ステージ名)] に **`test`** と入力します。

> **Note**
>
> 入力は UTF-8 でエンコードされた (ローカライズされていない) テキストである必要があります。

4. (オプション) [説明] に説明を入力します。
5. [デプロイ] を選択します。

[ステージの詳細] で、コピーアイコンを選択して API の呼び出し URL をコピーします。API のベース URL の一般的なパターンは、`https://`*`api-id`*`.`*`region`*`.amazonaws.com/`*`stageName`* です。たとえば、`beags1mnid` リージョンで作成され、`us-west-2` ステージにデプロイされた API (`test`) の URL は、「`https://beags1mnid.execute-api.us-west-2.amazonaws.com/test`」です。

デプロイステージで API をテストする

デプロイされた API をテストするには、いくつかの方法があります。URL パス変数またはクエリ文字列パラメータのみを使用する GET リクエストの場合は、ブラウザに API のリソース URL を入

力します。それ以外のメソッドの場合は、POSTMAN や cURL などのより高度な REST API テストユーティリティを使用する必要があります。

cURL を使用して API をテストするには

1. インターネットに接続されているローカルコンピュータでターミナルウィンドウを開きます。
2. POST /Seattle?time=evening をテストするには

   以下の cURL コマンドをコピーして、ターミナルウィンドウに貼り付けます。

```
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"
}'
```

   次のペイロードで成功を示すレスポンスが返ります。

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

   このメソッドリクエストで POST から PUT に変更した場合も同じレスポンスを受け取ります。

クリーンアップ

このチュートリアルで作成した Lambda 関数が不要になった場合は、削除できます。また、付随する IAM リソースも削除できます。

> **Warning**
>
> このシリーズの他のチュートリアルを行う予定の場合は、Lambda 実行ロールまたは Lambda 呼び出しロールを削除しないでください。API が依存する Lambda 関数を削除すると、API は機能しなくなります。Lambda 関数の削除は元に戻すことができません。再度 Lambda 関数を使用するには、関数を再度作成する必要があります。

Lambda 関数が依存する IAM リソースを削除すると、Lambda 関数は機能しなくなります。また、同関数に依存する API は機能しなくなります。IAM リソースの削除は元に戻すことができません。再度 IAM リソースを使用するには、リソースを再度作成する必要があります。

Lambda 関数を削除するには

1. AWS Management Console にサインインして AWS Lambda コンソール (https://console.aws.amazon.com/lambda/) を開きます。
2. 関数の一覧から [GetStartedLambdaIntegration] を選択し、[アクション]、[関数を削除] の順に選択します。プロンプトが表示されたら、再度 [削除] を選択します。

関連付けられた IAM リソースを削除する

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. [詳細] から [ロール] を選択します。
3. ロールのリストから [GetStartedLambdaIntegrationRole] を選択し、[ロールのアクション]、[ロールの削除] の順に選択します。コンソールの手順に従ってロールを削除します。

チュートリアル:クロスアカウント Lambda プロキシ統合を使用して API Gateway REST API をビルドする

AWS Lambda 関数は、API 統合バックエンドとして、別の AWS アカウントから使用できるようになりました。各アカウントは、Amazon API Gateway を利用できるリージョンであればどのリージョンでもかまいません。これにより、複数の API 間で簡単に集中管理して、Lambda バックエンド関数を共有できるようになります。

このセクションでは、Amazon API Gateway コンソールを使用してクロスアカウント Lambda プロキシ統合を設定する方法を示します。

API Gateway のクロスアカウント Lambda 統合用の API の作成

API を作成するには

1. https://console.aws.amazon.com/apigateway で API Gateway コンソールにサインインします。

2. API Gateway を初めて使用する場合は、サービスの特徴を紹介するページが表示されます。[REST API] で、[構築] を選択します。[Create Example API (サンプル API の作成)] がポップアップ表示されたら、[OK] を選択します。

   API Gateway を使用するのが初めてではない場合、[Create API] (API を作成)を選択します。[REST API] で、[構築] を選択します。
3. [API 名] に「**CrossAccountLambdaAPI**」と入力します。
4. (オプション) [説明] に説明を入力します。
5. [API エンドポイントタイプ] を [リージョン別] に設定したままにします。
6. API の作成 を選択します。

別のアカウントで統合用の Lambda 関数を作成する

ここでは、サンプル API 作成時とは異なるアカウントで Lambda 関数を作成します。

別のアカウントで Lambda 関数を作成する

1. API Gateway API 作成時とは別のアカウントで Lambda コンソールにログインします。
2. [関数の作成] を選択します。
3. [Author from scratch] を選択します。
4. [一から作成] で、次の操作を行います。

   a. [関数名] に名前を入力します。

   b. [ランタイム] ドロップダウンリストから、サポートされている Node.js ランタイムを選択します。

   c. [Permissions (アクセス許可)] で、[実行ロールの選択または作成] を選択します。ロールを作成することも、既存のロールを選択することもできます。

   d. [関数の作成] を選択して続行します。
5. [関数コード] ペインまで下にスクロールします。
6. the section called “チュートリアル: Lambda プロキシ統合による Hello World API” から Node.js 関数の実装を入力します。
7. [デプロイ] を選択します。
8. 関数の完全な ARN をメモします (Lambda 関数ペインの右上隅)。この情報は、クロスアカウントの Lambda 統合を作成する際に必要になります。

クロスアカウントの Lambda 統合を設定する

別のアカウントで、統合用の Lambda 関数を設定したら、最初のアカウントで API Gateway コンソールを使用して API に追加します。

> **Note**
>
> クロスリージョン、クロスアカウントのオーソライザーを設定している場合、ターゲット関数に追加される `sourceArn` は、API のリージョンではなくリージョンの関数を使用する必要があります。

API を作成したら、リソースを作成します。通常、API リソースはアプリケーションロジックに従ってリソースツリーに整理されます。この例では、/helloworld リソースを作成します。

リソースを作成するには

1. / リソースを選択し、[メソッドを作成] を選択します。
2. [プロキシのリソース] はオフのままにします。
3. [リソースパス] は / のままにします。
4. [リソース名] に「**`helloworld`**」と入力します。
5. [CORS (Cross Origin Resource Sharing)] はオフのままにします。
6. [リソースの作成] を選択します。

リソースを作成したら、`GET` メソッドを作成します。別のアカウントで `GET` メソッドと Lambda 関数を統合します。

`GET` メソッドを作成するには

1. /helloworld リソースを選択し、[メソッドを作成] を選択します。
2. [メソッドタイプ] には、GET を選択します。
3. [統合タイプ] で、[Lambda 関数] を選択します。
4. [Lambda プロキシ統合]を有効にします。
5. [Lambda 関数] に、ステップ 1 の Lambda 関数の完全な ARN を入力します。

   Lambda コンソールのコンソールウィンドウの右上隅で、関数の ARN を検索できます。

6. ARN を入力すると、`aws lambda add-permission` コマンド文字列が表示されます。このポリシーは、2 番目のアカウントの Lambda 関数へのアクセスを最初のアカウントに許可します。`aws lambda add-permission` コマンド文字列を 2 番目のアカウントに設定されている AWS CLI ウィンドウにコピーして貼り付けます。
7. [メソッドの作成] を選択します。

Lambda コンソールで、関数の更新したポリシーを確認できます。

(オプション) 更新したポリシーを確認するには

1. AWS Management Console にサインインして AWS Lambda コンソール (https://console.aws.amazon.com/lambda/) を開きます。
2. Lambda 関数を選択します。
3. [Permissions] を選択します。

   `Allow` ポリシーの `Condition` 句で、`AWS:SourceArn` が API の `GET` メソッドの ARN になっています。

チュートリアル: サンプルをインポートして REST API を作成する

PetStore ウェブサイトの HTTP 統合を使用してシンプルな REST API を作成、テストするために Amazon API Gateway コンソールを使用できます。API 定義は OpenAPI 2.0 ファイルとして事前設定されています。API 定義を API Gateway にロードしたあと、API Gateway コンソールを使用して API の基本構造を確認するか、単純に API をデプロイしてテストすることができます。

PetStore サンプル API では、クライアントが`http://petstore-demo-endpoint.execute-api.com/petstore/pets`の HTTP バックエンドウェブサイトをアクセスするための以下の方法をサポートします。

> **Note**
>
> このチュートリアルでは、HTTP エンドポイントを例として使用します。独自の API を作成する場合、HTTP 統合には HTTPS エンドポイントを使用することをお勧めします。

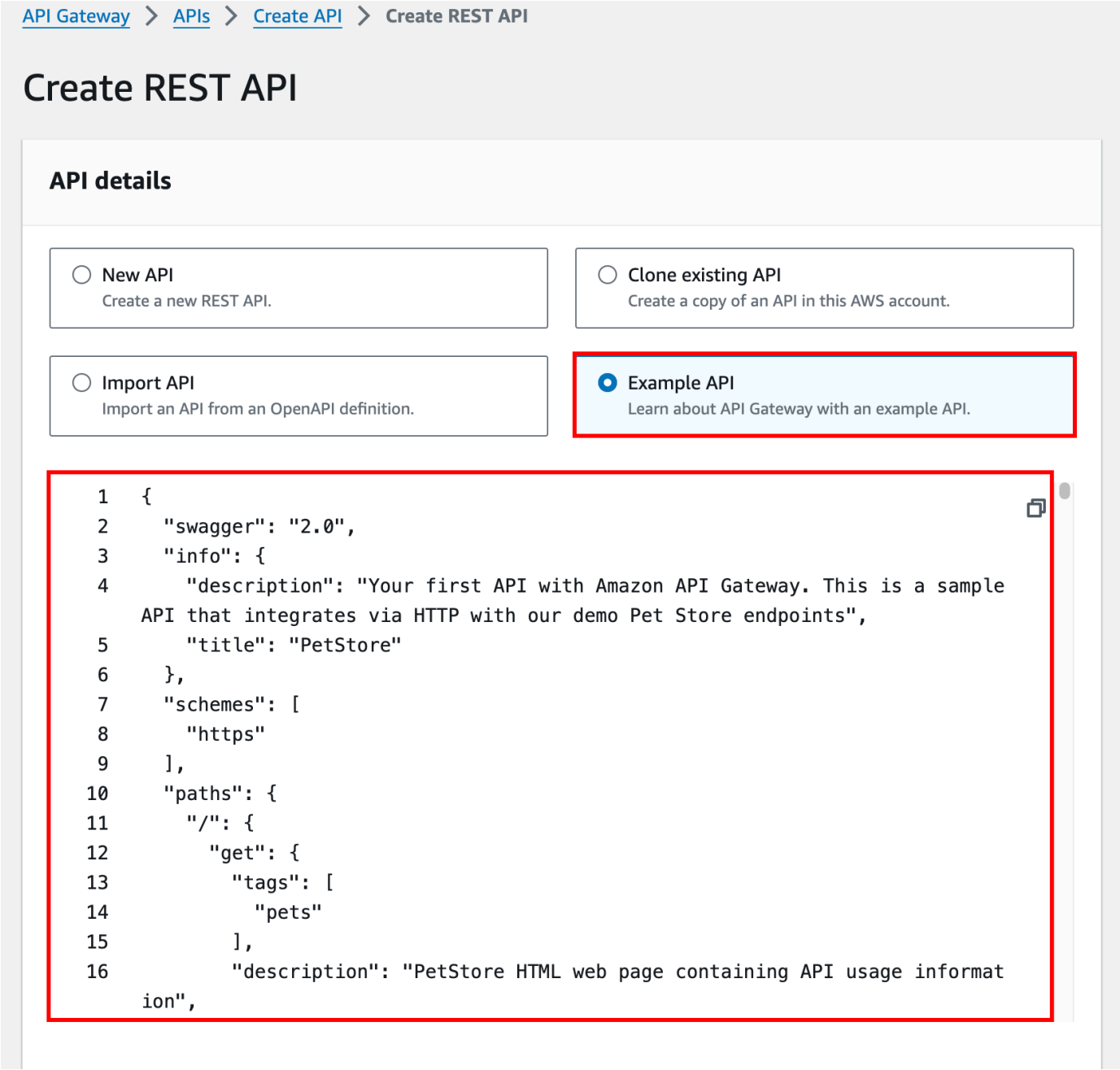
- `GET /`: どのバックエンドエンドポイントとも統合されていない API のルートリソースへの読み取りアクセス。API Gateway は PetStore ウェブサイトの概要で応答します。これは `MOCK` 統合タイプの例です。
- `GET /pets`: 同様の名前のバックエンド `/pets` リソースと統合されている API の `/pets` リソースへの読み取りアクセス。バックエンドは PetStore で利用可能なペットのページを返します。これは `HTTP` 統合タイプの例です。統合エンドポイントの URL は`http://petstore-demo-endpoint.execute-api.com/petstore/pets` です。
- `POST /pets`: バックエンド `/pets` リソースと統合されている API の `/petstore/pets` リソースへの書き込みアクセス。正しいリクエストを受信すると、バックエンドは指定されたペットを PetStore に追加し、結果を呼び出し元に返します。統合は `HTTP` でもあります。
- `GET /pets/{petId}`: 受信リクエスト URL のパス変数として指定される `petId` 値によって識別されるペットへの読み取りアクセス。このメソッドには `HTTP` 統合タイプもあります。バックエンドは PetStore で見つかった指定されたペットを返します。バックエンド HTTP エンドポイントの URL は `http://petstore-demo-endpoint.execute-api.com/petstore/pets/`*n* で、n は照会されたペットの識別子としての整数です。

API は `OPTIONS` 統合タイプの `MOCK` メソッドを通じて CORS アクセスをサポートします。API Gateway は、CORS アクセスをサポートする必要なヘッダーを返します。

次の手順では、API Gateway コンソールを使用して、サンプルから API を作成してテストする方法を説明します。

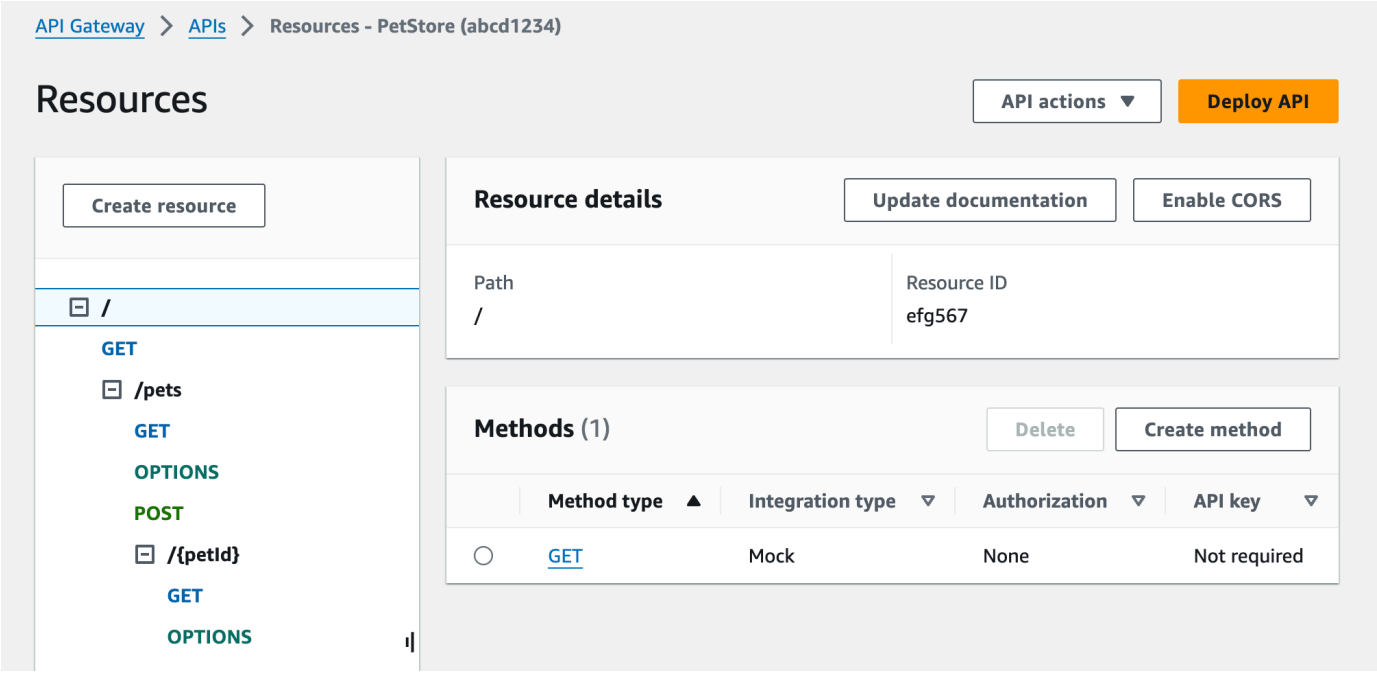
サンプル API をインポート、構築してテストするには

1. https://console.aws.amazon.com/apigateway で API Gateway コンソールにサインインします。
2. 次のいずれかを行います。
   - 最初の API を作成するには、[REST API] で [ビルド] を選択します。
   - 以前に API を作成した場合は、[API の作成] を選択し、[REST API] の [ビルド] を選択します。
3. [REST API の作成] の [サンプル API] を選択し、[API の作成] を選択して API サンプルを作成します。



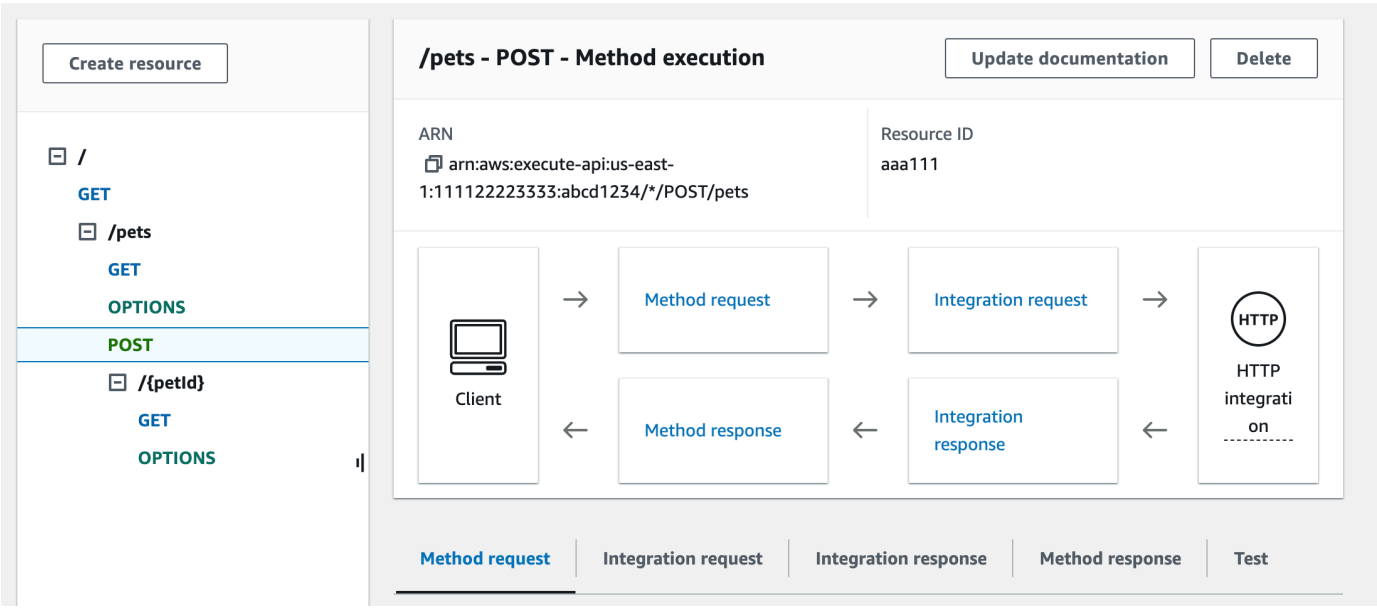
[API の作成] を選択する前に、OpenAPI 定義ファイルを下へスクロールして、この API サンプルの詳細について参照できます。

4. メインナビゲーションペインで、[リソース] を選択します。新しく作成された API は、次のように表示されます。



[リソース] ペインには、作成された API の構造が、ノードのツリーとして表示されます。各リソースで定義された API メソッドがツリーの辺になります。リソースが選択されると、そのすべてのメソッドが、右側の [メソッド] テーブルにリスト表示されます。各メソッドには、メソッドタイプ、統合タイプ、認証タイプ、API キー要件が表示されます。

5. メソッドの詳細を表示する、そのセットアップを変更する、またはメソッド呼び出しをテストするには、メソッドリストまたはリソースツリーからメソッド名を選択します。ここでは、図として POST /pets メソッドを選択します。



表示される [メソッド実行] ペインには、選択した (POST /pets) メソッドの構造と動作が論理的に表示されます。

[メソッドリクエスト] と [メソッドレスポンス] は、API のフロントエンドとのインターフェイスを表し、[統合リクエスト] と [統合レスポンス] は API とバックエンドとのインターフェイスを表します。

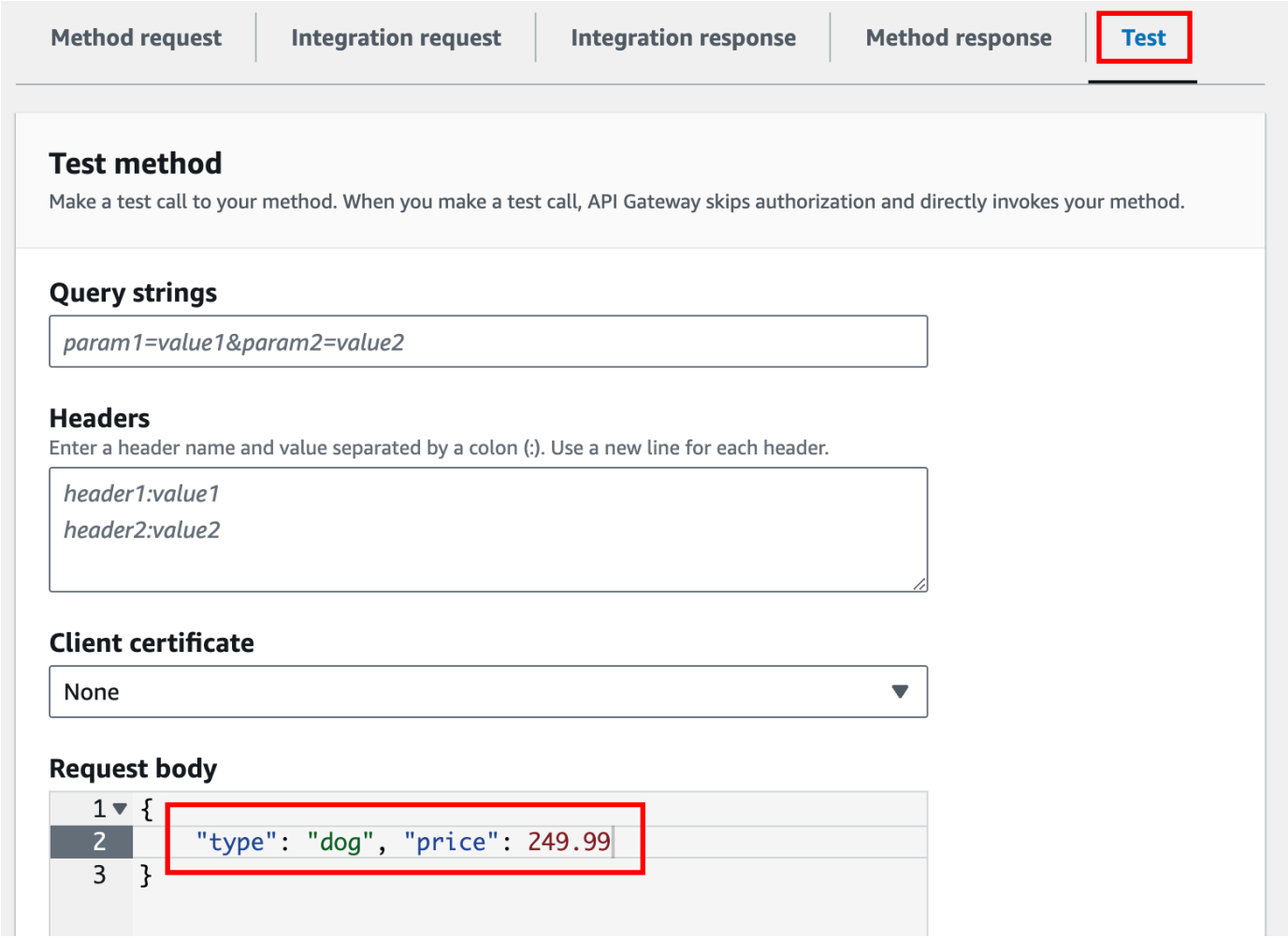
クライアントは、API を使用して [メソッドリクエスト] からバックエンド機能にアクセスします。API Gateway は、必要に応じてクライアントリクエストを [統合リクエスト] のバックエンドで許容される形式に変換してから、受信リクエストをバックエンドに転送します。変換されたリクエストは、統合リクエストと呼ばれます。同様に、バックエンドは、[統合レスポンス] で API Gateway にレスポンスを返します。API Gateway は、そのレスポンスを [Method Response (メソッドレスポンス)] にルーティングした後でクライアントに送信します。また、API Gateway は、必要に応じて、バックエンドのレスポンスデータをクライアントで予期される形式にマッピングします。

API リソースの POST メソッドで、メソッドリクエストのペイロードが統合リクエストのペイロードと同じ形式の場合、メソッドリクエストペイロードは、変更せずに統合リクエストに渡すことができます。

GET / メソッドリクエストは MOCK 統合タイプを使用し、実際のバックエンドのいずれのエンドポイントにも関連付けられません。対応する [統合レスポンス] は、静的な HTML ページを返すように設定されています。メソッドが呼び出されると、API Gateway はリクエストを受け取り、ただちに [メソッドレスポンス] で、クライアントに設定済みの統合レスポンスを返します。Mock 統合を使用して、バックエンドのエンドポイントを必要とすることなく API をテストできます。また、これを使用して、レスポンス本文マッピングテンプレートから生成されたローカルレスポンスを送信することもできます。

API デベロッパーは、メソッドリクエストとメソッドレスポンスを設定して、API のフロントエンドのやり取りの動作を制御します。統合リクエストと統合レスポンスを設定して、API のバックエンド統合の動作を制御します。これにはメソッド間のデータマッピングと、対応する統合が関連します。ここでは、エンドツーエンドのユーザー体験を提供するための API のテストについて説明します。

6. [テスト] タブを選択します。タブを表示するには、右矢印ボタンを選択する必要がある場合があります。
7. たとえば、POST /pets メソッドをテストするには、次のペイロード (**{"type": "dog","price": 249.99}**) を [リクエスト本文] に入力してから、[テスト] を選択します。



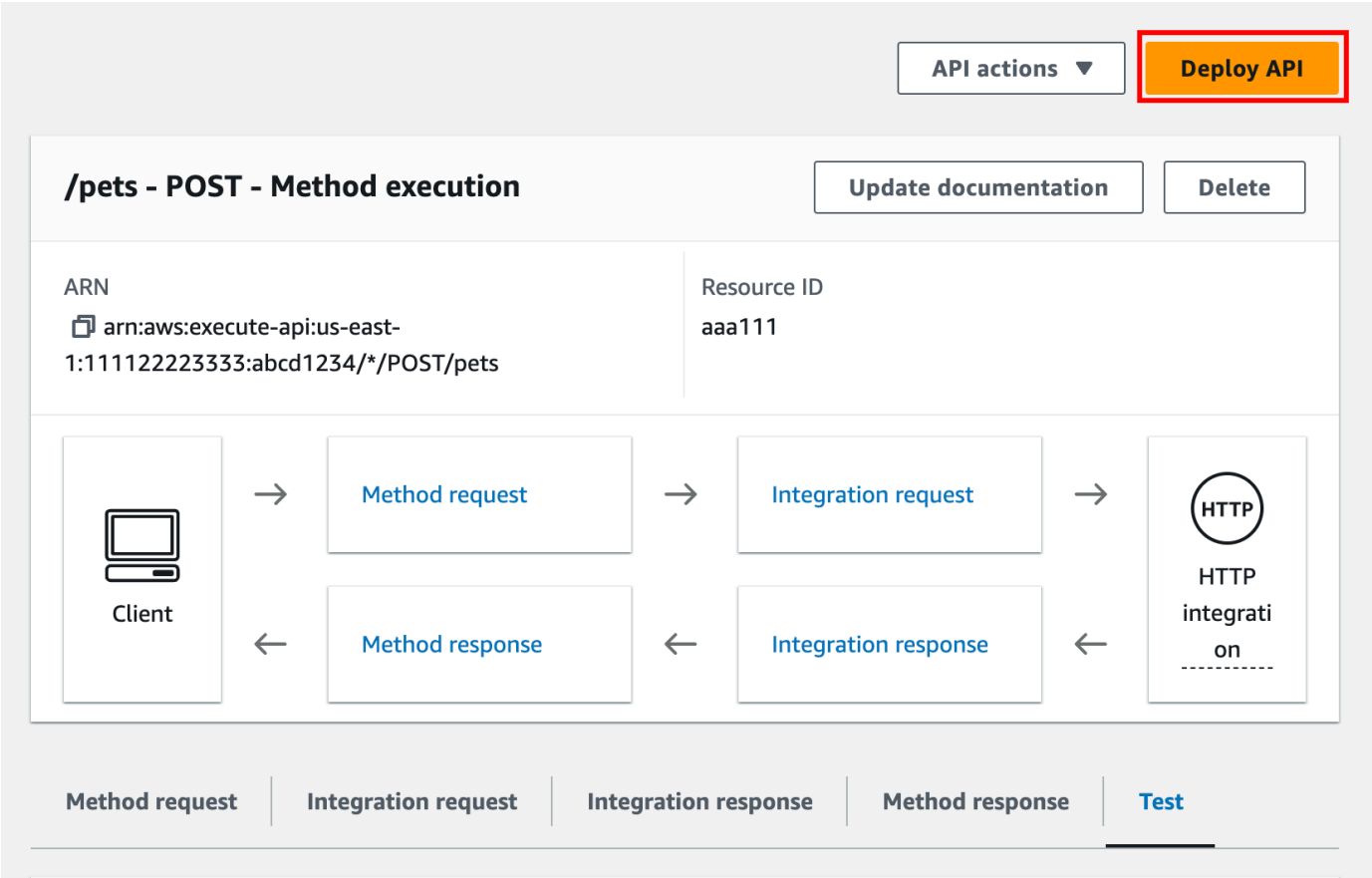
入力では、PetStore ウェブサイトでペットのリストに追加するペットの属性を指定します。

8. 結果は次のように表示されます。

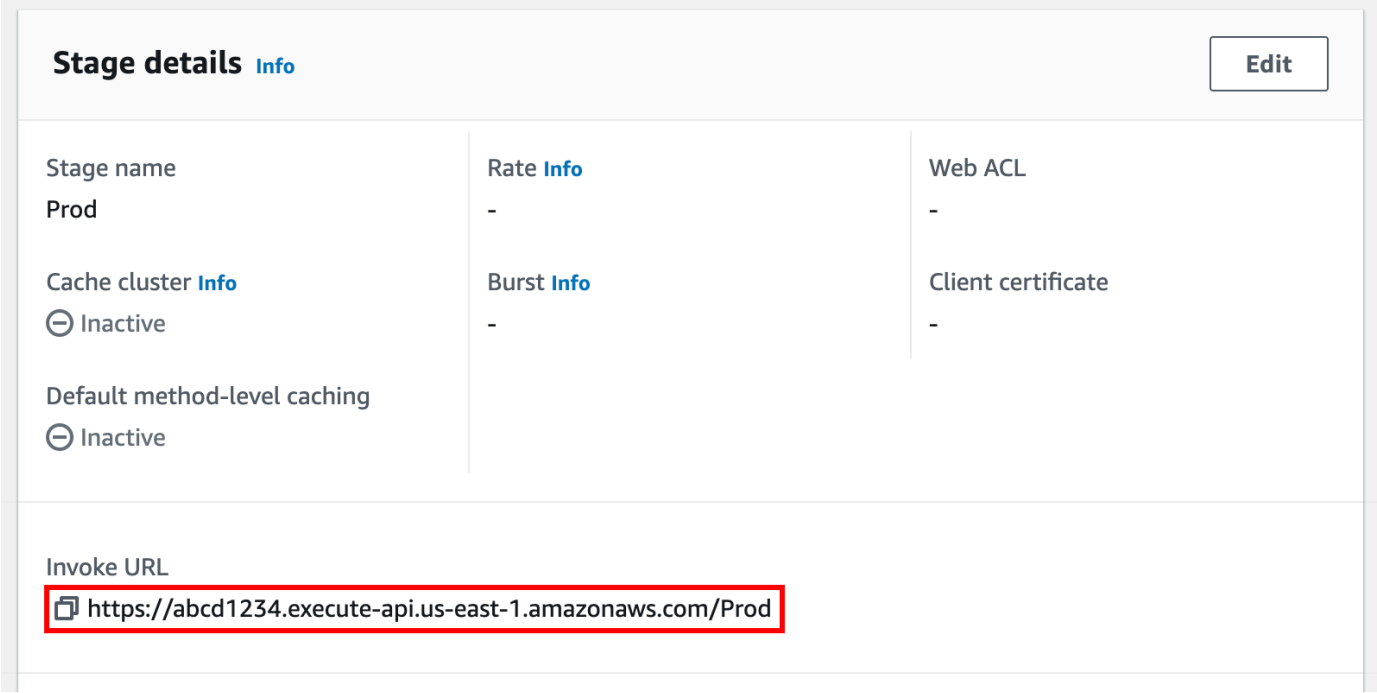
出力の [ログ] エントリは、メソッドリクエストから統合リクエストへの状態の変化と、統合レスポンスからメソッドレスポンスへの状態の変化を示します。これは、リクエストが失敗する原因となるマッピングエラーのトラブルシューティングに役立つ場合があります。この例では、マッピングは適用されません。メソッドリクエストのペイロードは、統合リクエストでバックエンドに渡されます。また、同様に、バックエンドレスポンスは、統合レスポンスからメソッドレスポンスに渡されます。

API Gateway test-invoke-request 機能以外のクライアントを使用して API をテストするには、最初に API をステージにデプロイする必要があります。

9. サンプル API をデプロイするには、[API のデプロイ] を選択します。



10. [ステージ] には、[新規ステージ] を選択し、**test** を入力します。
11. (オプション) [説明] に説明を入力します。
12. [デプロイ] を選択します。
13. 結果として表示される [ステージ] の [ステージの詳細] で、[URL を呼び出す] には API の GET / メソッドリクエストを呼び出す URL が表示されます。



14. コピーアイコンを選択して API の呼び出し URL をコピーし、Web ブラウザに API の呼び出し URL を入力します。その結果、正常なレスポンスとして、統合レスポンスのマッピングテンプレートから生成された結果が返されます。

15. [Stages] (ステージ) ナビゲーションペインで、[test] (テスト) ステージを展開し、`/pets/{petId}` で [GET] を選択してから、[Invoke URL] (呼び出し URL) の値 `https://api-id.execute-api.region.amazonaws.com/test/pets/{petId}` をコピーします。`{petId}` はパス変数を表します。

    (前のステップで取得した) [呼び出し URL] の値をブラウザのアドレスバーに貼り付け、`{petId}` を 1 などで置き換え、Enter キーを押してリクエストを送信します。200 OK レスポンスが、次の JSON ペイロードとともに返されます。

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

    このように API メソッドを呼び出すことは可能です。これは、その Authorization タイプが NONE に設定されているためです。AWS_IAM 認証が使用されている場合、署名バージョン 4 (SigV4) のプロトコルを使用してリクエストに署名します。このようなリクエストの例につい

ては、「the section called “チュートリアル: HTTP 非プロキシ統合を使用して API をビルドする”」を参照してください。

HTTP 統合を選択するチュートリアル

HTTP 統合で API をビルドするには、HTTP プロキシ統合または HTTP カスタム統合のどちらかを使用できます。

HTTP プロキシ統合では、バックエンドの要件に従って HTTP メソッドと HTTP エンドポイント URI を設定するだけで済みます。合理化された API セットアップを利用するには、可能な限り、HTTP プロキシ統合を使用することをお勧めします。

HTTP カスタム統合は、バックエンドのクライアントリクエストデータを変換するか、クライアントのためにバックエンドレスポンスデータを変換する必要がある場合に使用できます。

トピック

- チュートリアル: HTTP プロキシ統合を使用して REST API をビルドする
- チュートリアル: HTTP 非プロキシ統合を使用して REST API をビルドする

チュートリアル: HTTP プロキシ統合を使用して REST API をビルドする

HTTP プロキシ統合は、API を構築するシンプルかつパワフルで、汎用性のあるメカニズムです。これにより、単一の API メソッドのセットアップを合理化することで、ウェブアプリケーションから、統合された HTTP エンドポイントの複数のリソースや機能 (例: ウェブサイト全体) にアクセスすることができます。HTTP プロキシ統合で、API Gateway はクライアントが送信したメソッドリクエストをバックエンドに渡します。渡されるリクエストデータには、リクエストヘッダー、クエリ文字列パラメータ、URL パス変数、ペイロードなどが含まれます。バックエンド HTTP エンドポイントまたはウェブサーバーでは、受信リクエストデータを解析して、返すレスポンスを決定します。HTTP プロキシ統合では、API メソッドの設定後に API Gateway からの介入なしで、クライアントとバックエンドが直接やり取りできます (「the section called “重要な注意点”」に示されているサポートされない文字など、既知の問題が発生した場合を除く)。

また、網羅的なプロキシリソース (`{proxy+}`) と、多様な状況に対応できる HTTP メソッドの `ANY` 動詞を使用すれば、HTTP プロキシ統合を使用して、単一の API メソッドの API を作成することができます。このメソッドでは、ウェブサイトのパブリックにアクセス可能な HTTP リソースとオペレーションのセット全体を公開します。バックエンドのウェブサーバーでパブリックアクセス用のリソースが他にも開かれると、クライアントは、同じ API をセットアップしてこれらの新しいリソー

スを使用できます。このようにするために、ウェブサイトデベロッパーは、適用可能な新しいリソースやオペレーションについて、クライアントデベロッパーに明確に伝える必要があります。

以下のチュートリアルでは、HTTP プロキシ統合の概要について説明します。このチュートリアルでは、API Gateway コンソールを使用して API を作成し、汎用的なプロキシリソース `{proxy+}` から PetStore ウェブサイトと統合し、`ANY` の HTTP メソッドのプレースホルダーを作成します。

トピック

- API Gateway コンソールを使用して HTTP プロキシ統合で API を作成する
- HTTP プロキシ統合を使用して API をテストする

API Gateway コンソールを使用して HTTP プロキシ統合で API を作成する

次の手順では、API Gateway コンソールを使用して、HTTP バックエンド用のプロキシリソースで API を作成してテストする方法を説明します。HTTP バックエンドは、`PetStore` の `http://petstore-demo-endpoint.execute-api.com/petstore/pets` ウェブサイト (チュートリアル: HTTP 非プロキシ統合を使用して REST API をビルドする) です。ここでは、スクリーンショットを視覚的な補助として使用し、API Gateway UI 要素を示します。はじめて API Gateway コンソールを使用して API を作成する場合は、まず該当セクションに従って行います。

API を作成するには

1. https://console.aws.amazon.com/apigateway で API Gateway コンソールにサインインします。
2. API Gateway を初めて使用する場合は、サービスの特徴を紹介するページが表示されます。[REST API] で、[構築] を選択します。[Create Example API (サンプル API の作成)] がポップアップ表示されたら、[OK] を選択します。

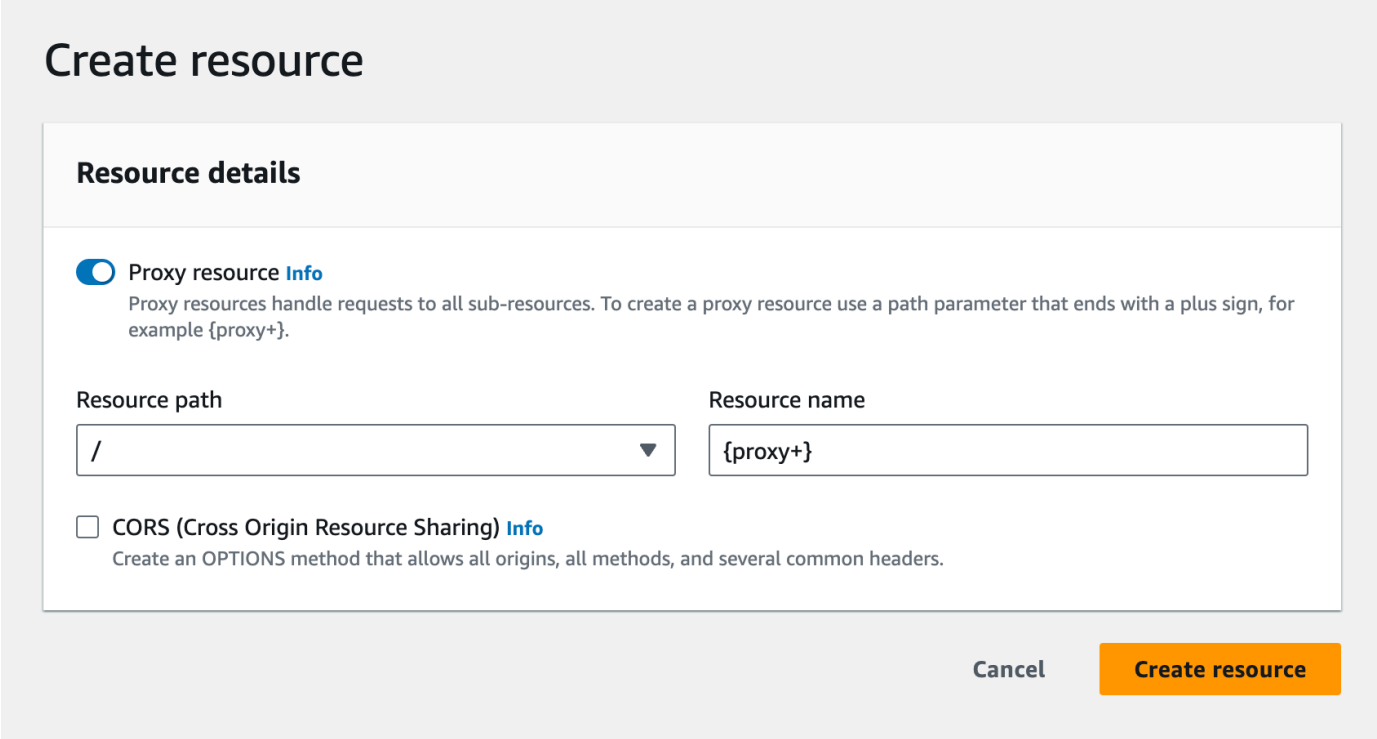
   API Gateway を使用するのが初めてではない場合、[Create API] (API を作成)を選択します。[REST API] で、[構築] を選択します。
3. [API 名] に「**`HTTPProxyAPI`**」と入力します。
4. (オプション) [説明] に説明を入力します。
5. [API エンドポイントタイプ] を [リージョン別] に設定したままにします。
6. API の作成 を選択します。

このステップでは、`{proxy+}` のプロキシリソースパスを作成します。これは、`http://petstore-demo-endpoint.execute-api.com/` のいずれかのバックエンドエンドポイントのプ

レースホルダーです。たとえば、petstore、petstore/pets、petstore/pets/{petId} のようになります。API Gateway は、{proxy+} リソースの作成時に ANY メソッドを作成し、ランタイムに、サポートされている HTTP 動詞のいずれかのプレースホルダーとして機能します。

/{proxy+} リソースを作成するには

1. API を選択します。
2. メインナビゲーションペインで、[リソース] を選択します。
3. [リソースの作成] を選択します。
4. [プロキシのリソース] を有効にします。
5. [リソースパス] は / のままにします。
6. [リソース名] に「**{proxy+}**」と入力します。
7. [CORS (Cross Origin Resource Sharing)] はオフのままにします。
8. [リソースの作成] を選択します。



このステップでは、プロキシ統合を使用して、ANY メソッドをバックエンド HTTP エンドポイントと統合します。プロキシ統合の場合、API Gateway はクライアントから送信されたメソッドリクエストを API Gateway の介入なしでバックエンドに渡します。

ANY メソッドを作成するには

1. /{proxy+} リソースを選択します。
2. ANY メソッドを選択します。
3. 警告シンボルの下にある [統合を編集] を選択します。統合のないメソッドを含む API はデプロイできません。
4. [統合タイプ] で、[HTTP] を選択します。
5. [HTTP プロキシ統合] を有効にします。
6. [HTTP メソッド] で、[ANY] を選択します。
7. [エンドポイント URL] に「**http://petstore-demo-endpoint.execute-api.com/{proxy}**」と入力します。
8. [Save] を選択します。

HTTP プロキシ統合を使用して API をテストする

特定のクライアントリクエストが成功するかは次に応じて異なります。

- バックエンドで、対応するバックエンドポイントエンドポイントが利用可能になった場合、または利用可能である場合は、必要なアクセス許可が付与されます。
- クライアントから適切に入力が行われる場合。

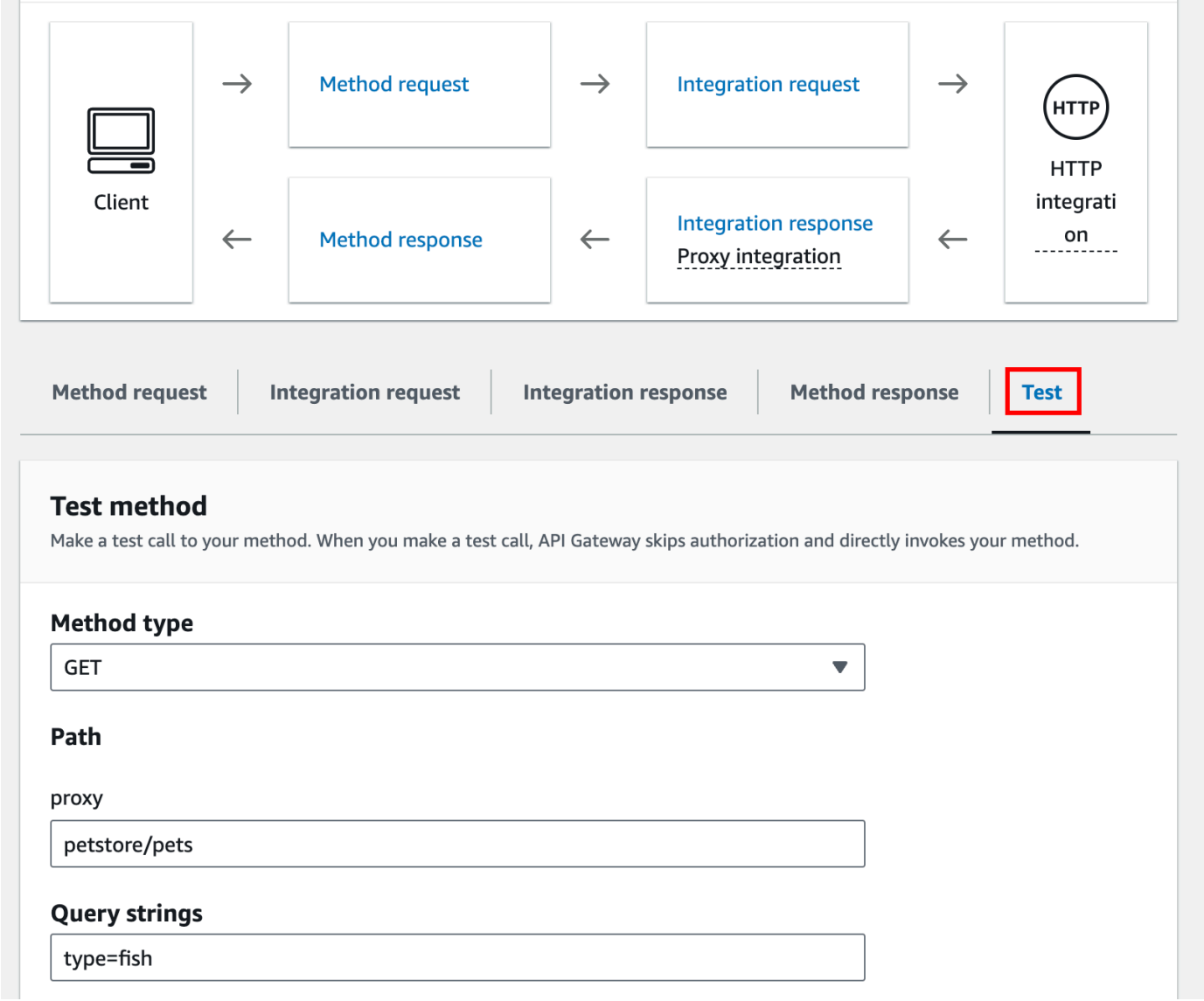
たとえば、ここで使用した PetStore API には、`/petstore` リソースは表示されません。そのため、取得するレスポンス (`404 Resource Not Found`) には、エラーメッセージ (`Cannot GET /petstore`) が含まれます。

さらに、クライアントは、結果を正しく処理するために、バックエンドの出力形式を処理できるようにする必要があります。クライアントとバックエンドの間のやり取りを容易にするために API Gateway が仲介することはありません。

プロキシリソースを通じた HTTP プロキシ統合を使用して PetStore ウェブサイトと統合された API をテストするには

1. [テスト] タブを選択します。タブを表示するには、右矢印ボタンを選択する必要がある場合があります。
2. [メソッドタイプ] では、`GET` を選択します。
3. [パス] の [プロキシ] に、「**petstore/pets**」と入力します。

4. [クエリ文字列] に「**type=fish**」と入力します。
5. [テスト] を選択します。



バックエンドのウェブサイトは、`GET /petstore/pets?type=fish` リクエストをサポートするため、次のような成功のレスポンスを返します。

```
[
  {
    "id": 1,
    "type": "fish",
    "price": 249.99
  },
  {
```

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

`GET /petstore` を呼び出そうとすると、`404` レスポンスとエラーメッセージ `Cannot GET /petstore` が返されます。これは、指定したオペレーションがバックエンドでサポートされていないためです。`GET /petstore/pets/1` を呼び出すと、リクエストは PetStore ウェブサイトでサポートされているため、`200 OK` レスポンスと次のペイロードが返されます。

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

ブラウザを使用して API をテストすることもできます。API をデプロイし、それをステージに関連付けて API の呼び出し URL を作成します。

API をデプロイするには

1. [API のデプロイ] を選択します。
2. [ステージ] で [新規ステージ] を選択します。
3. [Stage name (ステージ名)] に **`test`** と入力します。
4. (オプション) [説明] に説明を入力します。
5. [デプロイ] を選択します。

これで、クライアントは API を呼び出すことができます。

API を呼び出すには

1. https://console.aws.amazon.com/apigateway で API Gateway コンソールにサインインします。

2. API を選択します。
3. メインナビゲーションペインで、[ステージ] を選択します。
4. [ステージの詳細] で、コピーアイコンを選択して API の呼び出し URL をコピーします。

   API の呼び出し URL をウェブブラウザに入力します。

   URL は次のようになります。`https://abcdef123.execute-api.us-east-2.amazonaws.com/test/petstore/pets?type=fish`

   ブラウザが API に`GET`リクエストを送信します。
5. この結果は、API Gateway コンソールで [テスト] を使用したときに返される結果と同じであることが必要です。

チュートリアル: HTTP 非プロキシ統合を使用して REST API をビルドする

このチュートリアルでは、Amazon API Gateway コンソールを使用して、API をゼロから作成します。コンソールを API デザインスタジオとして使用して API 機能を絞り込み、その動作を確認して API を作成し、API をステージにデプロイします。

トピック

- HTTP カスタム統合を使用して API を作成する
- (オプション) リクエストパラメータをマッピングする

HTTP カスタム統合を使用して API を作成する

このセクションでは、リソースの作成、リソースでのメソッドの公開、目的の API 動作を達成するためのメソッドの設定、および API のテストとデプロイのステップを説明します。

このステップでは、空の API を作成します。次の手順では、非プロキシ HTTP 統合を使用して API を `http://petstore-demo-endpoint.execute-api.com/petstore/pets` エンドポイントに接続するためのリソースとメソッドを作成します。

API を作成するには

1. https://console.aws.amazon.com/apigateway で API Gateway コンソールにサインインします。
2. API Gateway を初めて使用する場合は、サービスの特徴を紹介するページが表示されます。[REST API] で、[構築] を選択します。[Create Example API (サンプル API の作成)] がポップアップ表示されたら、[OK] を選択します。

API Gateway を使用するのが初めてではない場合、[Create API] (API を作成)を選択します。[REST API] で、[構築] を選択します。

3. [API 名] に「**HTTPNonProxyAPI**」と入力します。
4. (オプション) [説明] に説明を入力します。
5. [API エンドポイントタイプ] を [リージョン別] に設定したままにします。
6. API の作成 を選択します。

[リソース] ツリーには、メソッドのないルートリソース (/) が表示されます。この演習では、PetStore ウェブサイト (http://petstore-demo-endpoint.execute-api.com/petstore/pets.) の HTTP カスタム統合を使用して API を作成します。わかりやすくするため、ルートの子として `/pets` リソースを作成し、クライアントが PetStore ウェブサイトから利用できる Pets 項目のリストを取得するために、このリソースで GET メソッドを公開します。

/pets リソースを作成するには

1. / リソースを選択し、[リソースを作成] を選択します。
2. [プロキシのリソース] はオフのままにします。
3. [リソースパス] は / のままにします。
4. [リソース名] に「**pets**」と入力します。
5. [CORS (Cross Origin Resource Sharing)] はオフのままにします。
6. [リソースの作成] を選択します。

このステップでは、/pets リソースで `GET` メソッドを作成します。`GET` メソッドは `http://petstore-demo-endpoint.execute-api.com/petstore/pets` ウェブサイトと統合されています。API メソッドの他のオプションには、以下があります。

- [POST]: 主に子リソースの作成に使用されます。
- PUT。既存のリソースを更新するために主に使用します (推奨はされませんが、子リソースの作成にも使用できます)。
- DELETE: リソースの削除に使用されます。
- [PATCH]; リソースの更新に使用されます。
- [HEAD]: テストシナリオで主に使用します。GET と同じですが、リソースの表現を返しません。

- [OPTIONS]: 対象サービスに使用できる通信オプションに関する情報を取得するために呼び出し元が使用できます。

統合リクエストの [HTTP メソッド] で、バックエンドによってサポートされているメソッドを選択する必要があります。HTTP または `Mock integration` の場合、メソッドリクエストと統合リクエストが同じ HTTP 動詞を使用すると意味があります。他の統合タイプの場合、メソッドリクエストでは、おそらく統合リクエストとは異なる HTTP 動詞を使用します。たとえば、Lambda 関数を呼び出すには、統合リクエストでは `POST` を使用して関数を呼び出す必要がありますが、Lambda; 関数のロジックに応じて、メソッドリクエストでは任意の HTTP 動詞を使用できます。

/pets リソースで **`GET`** メソッドを作成するには

1. /pets リソースを選択します。
2. [メソッドの作成] を選択します。
3. [メソッドタイプ] には、GET を選択します。
4. [統合タイプ] で、[HTTP 統合] を選択します。
5. [HTTP プロキシ統合] はオフのままにします。
6. [HTTP メソッド] で、[GET] を選択します。
7. [エンドポイント URL] に「**`http://petstore-demo-endpoint.execute-api.com/petstore/pets`**」と入力します。

   PetStore ウェブサイトでは、特定のページでペットの種類 (「Dog」や「Cat」など) ごとに、`Pet` 項目のリストを取得できます。
8. [コンテンツの処理] で、[パススルー] を選択します。
9. [URL クエリ文字列パラメータ] を選択します。

   PetStore ウェブサイトでは、`type` および `page` クエリ文字列パラメータを使用して入力を受け取ります。メソッドリクエストにクエリ文字列パラメータを追加して、統合リクエストの対応するクエリ文字列パラメータにマッピングします。
10. クエリ文字列パラメータを追加するには、次の操作を行います。

    a. [クエリ文字列の追加] を選択します。

    b. [名前] に「**`type`**」と入力します。

    c. [必須] と [キャッシュ] はオフのままにしておきます。

前の手順を繰り返し、**page** という名前で追加のクエリ文字列を作成します。

11. [メソッドの作成] を選択します。

これで、クライアントはリクエストを送信するときに、ペットのタイプとページ番号をクエリ文字列パラメータとして指定できます。これらの入力パラメータは、バックエンドの PetStore ウェブサイトに入力値を転送するために、統合クエリ文字列パラメータにマッピングする必要があります。

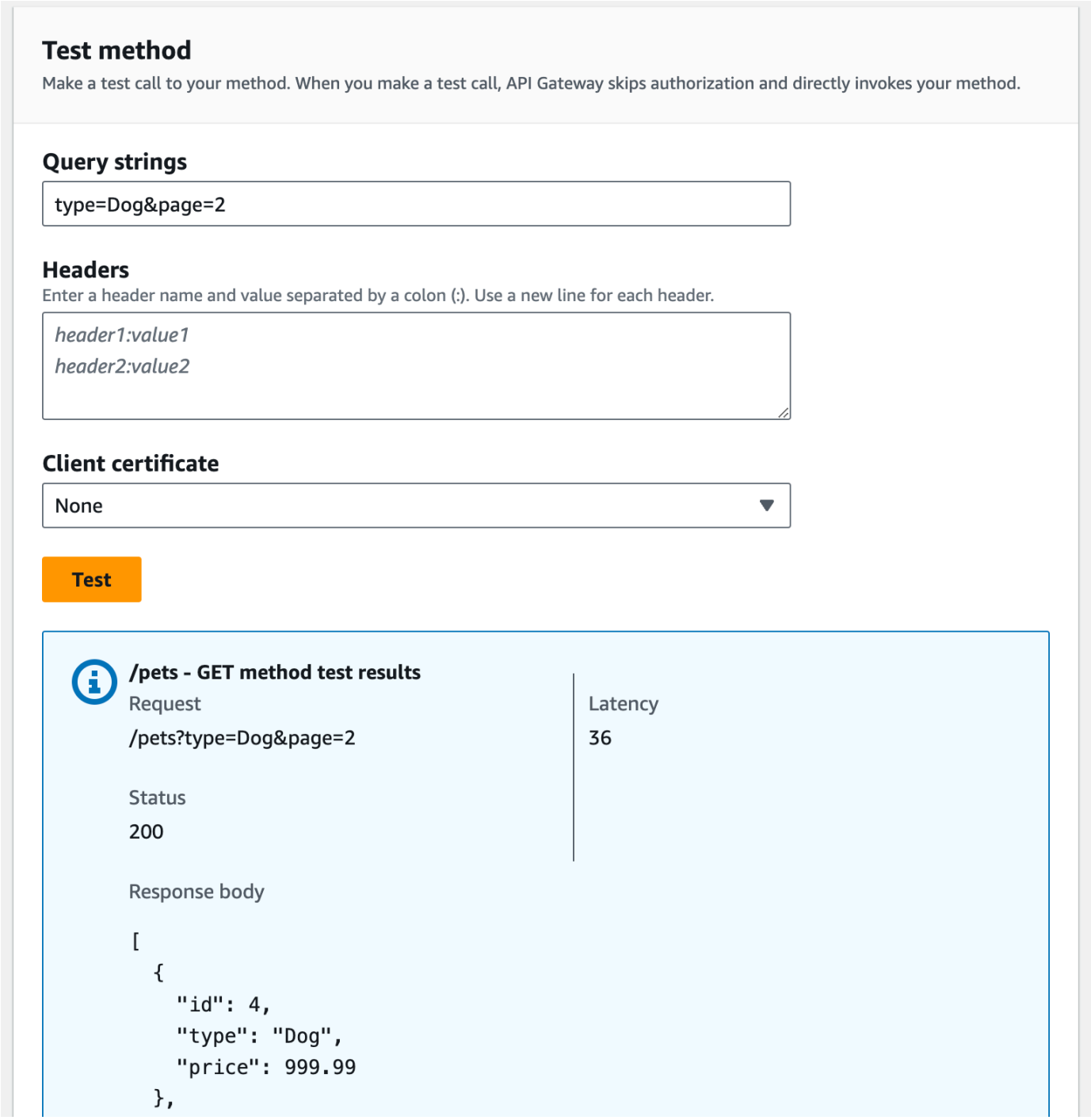
入力パラメータを統合リクエストにマッピングするには

1. [統合リクエスト] タブの [統合リクエストの設定] で、[編集] を選択します。
2. [URL クエリ文字列パラメータ] を選択し、次の操作を行います。
   a. [クエリ文字列パラメータを追加] を選択します。
   b. [名前] に**type**と入力します。
   c. [マッピング元] として「**method.request.querystring.type**」と入力します。
   d. [キャッシュ] はオフのままにします。
   e. [クエリ文字列パラメータを追加] を選択します。
   f. [名前] に**page**と入力します。
   g. [マッピング元] として「**method.request.querystring.page**」と入力します。
   h. [キャッシュ] はオフのままにします。
3. [Save] を選択します。

API をテストするには

1. [テスト] タブを選択します。タブを表示するには、右矢印ボタンを選択する必要がある場合があります。
2. [クエリ文字列] に「**type=Dog&page=2**」と入力します
3. [テスト] を選択します。

   結果は次の例のようになります。



これでテストが成功したので、API をデプロイし、公開することができます。

4. [API のデプロイ] を選択します。
5. [ステージ] で [新規ステージ] を選択します。
6. [Stage name (ステージ名)] に **`Prod`** と入力します。
7. (オプション) [説明] に説明を入力します。

8. [デプロイ] を選択します。
9. (オプション) [ステージの詳細] の [呼び出し URL] で、コピーアイコンを選択して API の呼び出し URL をコピーします。Postman や cURL のようなツールでこれを使用して API をテストできます。

SDK を使用してクライアントを作成する場合は、SDK で公開されたメソッドを呼び出して、リクエストに署名できます。実装の詳細については、選択する AWS SDK を参照してください。

> **Note**
>
> API が変更された場合、再度リクエスト URL を呼び出す前に、API を再デプロイして、新機能または更新された機能を使用できるようにする必要があります。

(オプション) リクエストパラメータをマッピングする

API Gateway API のリクエストパラメータをマッピングする

このチュートリアルでは、API のメソッドリクエスト URL の `{petId}` のパスパラメータを作成し、項目 ID を指定します。次に、これを統合リクエスト URL で `{id}` パスパラメータにマッピングし、リクエストを HTTP エンドポイントに送信します。

> **Note**
>
> 大文字の代わりに小文字を使用するなど、大文字と小文字を間違えて入力すると、チュートリアルの後半でエラーが発生します。

ステップ 1: リソースを作成する

このステップでは、パスパラメータ {petId} を使用してリソースを作成します。

{petId} リソースを作成するには

1. /pets リソースを選択し、[リソースを作成] を選択します。
2. [プロキシのリソース] はオフのままにします。
3. [リソースパス] として、[/pets/]を選択します。
4. [リソース名] に「**`{petId}`**」と入力します。

petId の周りに波括弧 ({ }) を使用し、/pets/{petId} と表示されるようにします。

5. [CORS (Cross Origin Resource Sharing)] はオフのままにします。
6. [リソースの作成] を選択します。

ステップ 2: メソッドを作成してテストする

このステップでは、{petId} パスパラメータを使用して GET メソッドを作成します。

GET メソッドをセットアップするには

1. /{petId} リソースを選択し、[メソッドを作成] を選択します。
2. [メソッドタイプ] には、GET を選択します。
3. [統合タイプ] で、[HTTP 統合] を選択します。
4. [HTTP プロキシ統合] はオフのままにします。
5. [HTTP メソッド] で、[GET] を選択します。
6. [エンドポイント URL] に「**http://petstore-demo-endpoint.execute-api.com/petstore/pets/{id}**」と入力します。
7. [コンテンツの処理] で、[パススルー] を選択します。
8. [デフォルトタイムアウト] はオンのままにします。
9. [メソッドの作成] を選択します。

次に、{petId} パスパラメータを HTTP エンドポイントの {id} パスパラメータにマッピングします。

{petId} パスパラメータをマッピングするには

1. [統合リクエスト] タブの [統合リクエストの設定] で、[編集] を選択します。
2. [URL パスパラメータ] を選択します。
3. API Gateway は、統合リクエストのパスパラメータを petId という名前で作成します。これはバックエンドでは機能しません。HTTP エンドポイントはパスパラメータとして {id} を使用します。petId の名前を **id** に変更します。

   これは petId のメソッドリクエストのパスパラメータを、id の統合リクエストのパスパラメータにマッピングします。
4. [Save] を選択します。

次にメソッドをテストします。

メソッドをテストするには

1. [テスト] タブを選択します。タブを表示するには、右矢印ボタンを選択する必要がある場合があります。
2. petId の [パス] に、「**4**」と入力します。
3. [Test (テスト)] を選択します。

   成功すると、[レスポンス本文] に次の内容が表示されます。

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

ステップ 3: API をデプロイする

このステップでは、API をデプロイして、以降 API Gateway コンソール外で呼び出せるようにします。

API をデプロイする

1. [API のデプロイ] を選択します。
2. [ステージ] で [実稼働] を選択します。
3. (オプション) [説明] に説明を入力します。
4. [デプロイ] を選択します。

ステップ 4: API をテストする

このステップでは、API Gateway コンソール外に移動し、API を使用して HTTP エンドポイントにアクセスします。

1. メインナビゲーションペインで、[ステージ] を選択します。
2. [ステージの詳細] で、コピーアイコンを選択して API の呼び出し URL をコピーします。

   次のように表示されます。

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

3. リクエストを送信する前に、この URL をブラウザの新しいタブのアドレスボックスに入力し、/pets/4 を URL に付加します。
4. ブラウザは以下を返します。

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

次のステップ

リクエストの検証の有効化、データの変換、カスタムゲートウェイレスポンスの作成により、API をさらにカスタマイズできます。

API をカスタマイズするその他の方法については、以下のチュートリアルを参照してください。

- リクエスト検証の詳細については、「API Gateway で基本的なリクエストの検証を設定する」を参照してください。
- リクエストとレスポンスのペイロードを変換する方法の詳細については、「API Gateway でのデータ変換の設定」を参照してください。
- カスタムゲートウェイレスポンスの作成方法については、「API Gateway コンソールを使用して REST API のゲートウェイレスポンスをセットアップする」を参照してください。

チュートリアル: API Gateway のプライベート統合を使用して REST API をビルドする

プライベート統合による API Gateway API を作成して、お客様が Amazon Virtual Private Cloud (Amazon VPC) 内の HTTP/HTTPS リソースにアクセスできるようにすることができます。このような VPC リソースは、VPC 内の Network Load Balancer の背後にある EC2 インスタンス上の HTTP/HTTPS エンドポイントです。Network Load Balancer は、VPC リソースをカプセル化し、受信リクエストをターゲットリソースにルーティングします。

クライアントが API を呼び出すと、API Gateway は事前構成された VPC リンクを介して Network Load Balancer に接続します。VPC リンクは、VpcLink の API Gateway リソースによってカプセル

化されています。VPC リソースへの API メソッドリクエストの転送を担当し、バックエンドレスポンスを呼び出し元に返します。API デベロッパーにとって、VpcLink は機能的に統合エンドポイントと同等です。

プライベートインテグレーションで API を作成するには、新規の VpcLink を作成するか、目的の VPC リソースをターゲットとする Network Load Balancer に接続された既存のものを選択する必要があります。VpcLink を作成および管理するための 適切なアクセス権限が必要です。次に、API メソッドを設定して VpcLink と統合します。そのためには、HTTP または HTTP_PROXY のいずれかを統合タイプとして設定し、VPC_LINK を統合の接続タイプとして設定し、統合の VpcLink で connectionId ID を設定します。

> **Note**
>
> Network Load Balancer と API は、同じ AWS アカウントによって所有されている必要があります。

VPC リソースにアクセスするための API をすばやく作成するために、API Gateway コンソールを使用したプライベートインテグレーションを使用して API を構築するための重要なステップについて説明します。API を作成する前に、次の操作を行います。

1. VPC リソースを作成し、同じリージョンのアカウントで Network Load Balancer を作成または選択し、リソースをホストする EC2 インスタンスを Network Load Balancer のターゲットとして追加します。詳細については、「API Gateway のプライベート統合の Network Load Balancer を設定する」を参照してください。
2. プライベート統合のための VPC リンクを作成するアクセス許可を付与します。詳細については、「VPC リンクを作成するためのアクセス許可の付与」を参照してください。

ターゲットグループに VPC リソースが設定された VPC リソースと Network Load Balancer を作成したら、以下の手順に従って API を作成し、プライベート統合の VpcLink を使用して VPC リソースと統合します。

プライベート統合で API を作成するには

1. https://console.aws.amazon.com/apigateway で API Gateway コンソールにサインインします。
2. API Gateway を初めて使用する場合は、サービスの特徴を紹介するページが表示されます。[REST API] で、[構築] を選択します。[Create Example API (サンプル API の作成)] がポップアップ表示されたら、[OK] を選択します。

API Gateway を使用するのが初めてではない場合、[Create API] (API を作成)を選択します。[REST API] で、[構築] を選択します。

3. エッジ最適化 REST API またはリージョン REST API を作成します。
4. API を選択します。
5. [メソッドを作成] を選択して、次の操作を行います。

   a. [メソッドタイプ] では、GET を選択します。
   b. [統合タイプ] で、[VPC リンク] を選択します。
   c. [VPC プロキシ統合] を有効にします。
   d. [HTTP メソッド] で、[GET] を選択します。
   e. [VPC リンク] で、[ステージ変数の使用] を選択し、下のテキストボックスに「**${stageVariables.vpcLinkId}**」と入力します。

      ステージに API をデプロイした後で、vpcLinkId ステージ変数を定義し、その値を VpcLink の ID に設定します。

   f. [エンドポイント URL] に URL (例: http://myApi.example.com) を入力します。

      ここでは、ホスト名 (例: myApi.example.com) が統合リクエストの Host ヘッダーを設定するために使用されます。

   g. [メソッドの作成] を選択します。

      プロキシ統合で、API はデプロイできる状態になります。それ以外の場合は、適切なメソッドレスポンスと統合レスポンスのセットアップに進む必要があります。

6. [API をデプロイ] を選択し、次の操作を行います。

   a. [ステージ] で [新規ステージ] を選択します。
   b. [ステージ名] に、ステージ名を入力します。
   c. (オプション) [説明] に説明を入力します。
   d. [デプロイ] を選択します。

7. [ステージの詳細] セクションで、生成された呼び出し URL を書き留めておきます。API を呼び出す必要があります。しかし、それを行う前に、vpcLinkId ステージ変数を設定する必要があります。
8. [ステージ] ペインで、[ステージ変数] タブを選択し、次の操作を行います。

   a. [変数を管理] を選択し、[ステージ変数を追加] を選択します。

b. [名前] に**vpcLinkId**と入力します。

c. [値] に VPC_LINK の ID (例: *gix6s7*) を入力します。

d. [Save] を選択します。

ステージ変数を使用すると、ステージ変数の値を変更することで、API の別の VPC リンクに簡単に切り替えることができます。

チュートリアル: AWS 統合を使用して API Gateway REST API を構築する

チュートリアル: Lambda プロキシ統合を使用した Hello World REST API の構築 と AWS Lambda 統合を選択するチュートリアル のトピックではいずれも、API Gateway API を作成して、統合された Lambda 関数を公開する方法について説明します。さらに、Amazon SNS、Amazon S3、Amazon Kinesis、AWS Lambda などの AWS のその他サービスを公開する API Gateway API を作成することもできます。このステップは、AWS 統合で行うことができます。Lambda 統合または Lambda プロキシ統合は、特殊なケースです。ここで、Lambda 関数の呼び出しは、API Gateway API から公開されます。

すべての AWS サービスで、これらの機能を公開するための専用 API がサポートされています。ただし、アプリケーションプロトコルやプログラミングインターフェイスは、サービスによって異なる場合があります。AWS を統合した API Gateway API には、クライアントがさまざまな AWS のサービスにアクセスできるように、一貫したアプリケーションプロトコルが使用されているという利点があります。

このチュートリアルでは、Amazon SNS を公開するための API を作成します。API を他の AWS サービスと統合するその他の例については、「Amazon API Gateway のチュートリアルとワークショップ」を参照してください。

Lambda プロキシ統合とは異なり、AWS の他のサービスに対応するプロキシ統合はありません。そのため、API メソッドは単一の AWS アクションと統合されます。柔軟性を高めるために、プロキシ統合と同様、Lambda プロキシ統合をセットアップできます。その後、Lambda 関数が AWS のその他のアクションのリクエストを解析して処理します。

エンドポイントがタイムアウトになると、API Gateway は再試行しません。API 発信者は、エンドポイントのタイムアウトを処理するよう再試行ロジックを実行する必要があります。

この演習は、「AWS Lambda 統合を選択するチュートリアル」の手順と概念に基づいています。この演習を完了していない場合は、先に完了することを推奨します。

トピック

前提条件

このウォークスルーを開始する前に、次を行う必要があります。

1. 「API Gateway の開始方法の前提条件」の各ステップを実行します。
2. `MyDemoAPI` という名前の新しい API を作成します。詳細については、「チュートリアル: HTTP 非プロキシ統合を使用して REST API をビルドする」を参照してください。
3. `test` という名前のステージに少なくとも 1 回 API をデプロイします。詳細については、『』の「API のデプロイAWS Lambda 統合を選択するチュートリアル」を参照してください。
4. 「」の残りの設定ステップを完了しますAWS Lambda 統合を選択するチュートリアル
5. Amazon Simple Notification Service (Amazon SNS) で少なくとも 1 つのトピックを作成します。デプロイした API を使用して、AWS アカウントに関連付けられている Amazon SNS のトピックのリストを取得します。Amazon SNS でトピックを作成する方法については、「トピックの作成」を参照してください (ステップ 5 で説明しているトピック ARN をコピーする必要はありません)。

ステップ 1: AWS のサービスプロキシの実行ロールを作成する

Amazon SNS のアクションを呼び出すことを API に許可するには、IAM ロールに適切な IAM ポリシーをアタッチする必要があります。

AWS のサービスプロキシの実行ロールを作成するには

1. AWS Management Console にサインインして、IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。

2. [ロール] を選択します。
3. [ロールの作成] を選択します。
4. [信頼されたエンティティの種類を選択] で [AWS のサービス] を選択し、[API Gateway]、[API Gateway が CloudWatch Logs にログをプッシュすることを許可] の順に選択します。
5. [次へ] を選択し、さらに [次へ] を選択します。
6. [ロール名] に「**APIGatewaySNSProxyPolicy**」と入力し、[ロールの作成] を選択します。
7. [ロール] リストで、作成したロールを選択します。ロールを検索するには、必要に応じてスクロールするか、検索バーを使用します。
8. 選択したロールの [アクセス許可を追加] タブを選択します。
9. ドロップダウンリストから [ポリシーをアタッチ] を選択します。
10. 検索バーに「**AmazonSNSReadOnlyAccess**」と入力し、[アクセス許可を追加] を選択します。

> **Note**
>
> このチュートリアルでは、わかりやすくするために管理ポリシーを使用しますが、独自の IAM ポリシーを作成して、必要な最小限のアクセス許可を付与するのがベストプラクティスです。

11. 新しく作成したロール ARN をメモしておきます。これは後で使用します。

ステップ 2: リソースを作成する

このステップでは、AWS サービスプロキシと AWS サービスのやり取りを可能にするリソースを作成します。

リソースを作成するには

1. https://console.aws.amazon.com/apigateway で API Gateway コンソールにサインインします。
2. API を選択します。
3. ルートリソース / (1 つのスラッシュ (/) で表されます) を選択し、[リソースを作成] を選択します。
4. [プロキシのリソース] はオフのままにします。
5. [リソースパス] は / のままにします。
6. [リソース名] に「**mydemoawsproxy**」と入力します。
7. [CORS (Cross Origin Resource Sharing)] はオフのままにします。

8. [リソースの作成] を選択します。

ステップ 3: GET メソッドを作成する

このステップでは、AWS サービスプロキシと AWS サービスのやり取りを可能にする GET メソッドを作成します。

GET メソッドを作成するには

1. /mydemoawsproxy リソースを選択し、[メソッドを作成] を選択します。
2. メソッドタイプとして、[GET] を選択します。
3. [統合タイプ] で、[AWS のサービス] を選択します。
4. [AWS リージョン] で、作成した Amazon SNS トピックを作成した AWS リージョンを選択します。
5. [AWS のサービス] で、[Amazon SNS] を選択します。
6. [AWS サブドメイン] は空白のままにします。
7. [HTTP メソッド] で、[GET] を選択します。
8. [アクションタイプ] で、[アクション名を使用] を選択します。
9. [アクション名] に「**ListTopics**」と入力します。
10. [実行ロール] に、**APIGatewaySNSProxyPolicy** のロール ARN を入力します。
11. [メソッドの作成] を選択します。

ステップ 4: メソッドの設定を指定してメソッドをテストする

これで、GET メソッドをテストし、メソッドが Amazon SNS トピックを一覧表示するように適切に設定されていることを確認できます。

GET メソッドをテストするには

1. [テスト] タブを選択します。タブを表示するには、右矢印ボタンを選択する必要がある場合があります。
2. [テスト] を選択します。

   結果には、次のようなレスポンスが表示されます。

```
{
```

```
  "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"
    }
  }
}
```

ステップ 5: API をデプロイする

このステップでは、API をデプロイして、API Gateway コンソール外で呼び出せるようにします。

API をデプロイする

1. [API のデプロイ] を選択します。
2. [ステージ] で [新規ステージ] を選択します。
3. [Stage name (ステージ名)] に **`test`** と入力します。
4. (オプション) [説明] に説明を入力します。
5. [デプロイ] を選択します。

ステップ 6: API をテストする

このステップでは、API Gateway コンソール外で AWS サービスプロキシを使用して、Amazon SNS サービスとやり取りします。

1. メインナビゲーションペインで、[ステージ] を選択します。

2. [ステージの詳細] で、コピーアイコンを選択して API の呼び出し URL をコピーします。

   プリンシパルは以下のようになります。

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

3. ブラウザの新しいタブのアドレスボックスに URL を入力します。
4. /mydemoawsproxy を追加し、URL が次のように表示されるようにします。

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

   URL を参照します。以下のような情報が表示されます。

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

ステップ 7: クリーンアップ

AWS サービスプロキシの動作に必要な IAM リソースを削除できます。

> **Warning**
>
> AWS サービスプロキシが依存している IAM リソースを削除した場合、そのリソースに依存している AWS サービスプロキシと API はいずれも機能しなくなります。IAM リソースの削除は元に戻すことができません。その IAM リソースを再び使用する場合は、作成し直す必要があります。

関連付けられた IAM リソースを削除する

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. [詳細] エリアで、[ロール] を選択します。
3. [APIGatewayAWSProxyExecRole] を選択し、[ロールアクション]、[ロールの削除] の順に選択します。プロンプトが表示されたら、[Yes, Delete (はい、削除します)] を選択します。
4. [詳細] エリアで、[ポリシー] を選択します。

5. [APIGatewayAWSProxyExecPolicy] を選択し、[Policy Actions (ポリシーアクション)]、[Delete (削除)] の順に選択します。プロンプトが表示されたら、[削除] を選択します。

このチュートリアルはこれで終了です。AWS サービスプロキシとしての API 作成の詳細な説明については、「チュートリアル: API Gateway で REST API を Amazon S3 のプロキシとして作成する」、「チュートリアル: 2 つの AWS サービス統合と 1 つの Lambda 非プロキシ統合を使用して計算ツールの REST API を作成する」、または「チュートリアル: API Gateway で REST API を Amazon Kinesis のプロキシとして作成する」を参照してください。

チュートリアル: 2 つの AWS サービス統合と 1 つの Lambda 非プロキシ統合を使用して計算ツールの REST API を作成する

非プロキシ統合の開始方法のチュートリアルでは、Lambda Function 統合のみを使用します。Lambda Function 統合は、Lambda 関数を呼び出すために必要なリソースベースのアクセス許可を自動的に追加するなど、統合設定の大部分を実行する AWS Service 統合タイプの特殊なケースです。ここでは、3 つの統合のうち 2 つが AWS Service 統合を使用しています。この統合タイプでは、より細かく制御できますが、適切なアクセス許可を含む IAM ロールの作成や指定などのタスクを手動で実行する必要があります。

このチュートリアルでは、基本的な算術演算を実装し、JSON 形式の入出力を受け入れて返す Calc Lambda 関数を作成します。次に、REST API を作成し、それを Lambda 関数と以下の方法で統合します。

1. GET リソースで /calc メソッドを公開して Lambda 関数を呼び出し、入力をクエリ文字列パラメータとして渡します (AWS Service 統合)。
2. POST リソースで /calc メソッドを公開して Lambda 関数を呼び出すことにより、メソッドリクエストのペイロードに入力を提供します (AWS Service 統合)。
3. ネストされた GET リソースで /calc/{operand1}/{operand2}/{operator} を公開して Lambda 関数を呼び出し、パスパラメータとして入力を提供します (Lambda Function 統合)。

このチュートリアルを試してみるだけでなく、Calc API 用の OpenAPI 定義ファイルを検討することをお勧めします。これは、the section called “OpenAPI” の API Gateway へのインポートの指示に従ってインポートできます。

トピック

- 引き受け可能な IAM ロールを作成する
- Calc Lambda 関数の作成
- Calc Lambda 関数をテストする
- Calc API を作成する
- 統合 1: Lambda 関数を呼び出すためのクエリパラメータを持つ GET メソッドを作成する
- 統合 2: Lambda 関数を呼び出すための JSON ペイロードを持つ POST メソッドを作成する
- 統合 3: Lambda 関数を呼び出すためのパスパラメータを持つ GET メソッドを作成する
- Lambda 関数と統合されたサンプル API の OpenAPI 定義

引き受け可能な IAM ロールを作成する

API で `Calc` Lambda 関数を呼び出すには、API Gateway が引き受け可能な IAM ロールが必要です。これは、次の信頼関係を持つ IAM ロールです。

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

作成するロールには Lambda InvokeFunction アクセス許可が必要です。それ以外の場合、API 発信者は `500 Internal Server Error` レスポンスを受け取ります。このアクセス許可をロールに付与するには、次の IAM ポリシーをアタッチします。

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

```
            "Resource": "*"
        }
    ]
}
```

これをすべて実行する方法は次のとおりです。

API Gateway の引き受け可能な IAM ロールを作成する

1. IAM コンソールにログインします。
2. [Roles (ロール)] を選択します。
3. [Create Role (ロールの作成)] を選択します。
4. [Select type of trusted entity] (信頼されたエンティティの種類を選択) の下で、[AWS Service] (AWS サービス) を選択します。
5. [このロールを使用するサービスを選択] の下で、[Lambda] を選択します。
6. [Next: Permissions (次へ: アクセス許可)] を選択します。
7. [ポリシーの作成] を選択します。

   新しい [ポリシーの作成] コンソールウィンドウが開きます。このウィンドウで、以下の操作を行います。

   a. [JSON] タブで、既存のポリシーを以下のポリシーに置き換えます。

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

   b. [ポリシーの確認] を選択します。

   c. [ポリシーの確認] で、以下の作業を行います。

      i. [名前] には **lambda_execute** などの名前を入力します。

ii. [ポリシーの作成] を選択します。

8. 元の [ロールの作成] ウィンドウで、以下の操作を行います。

   a. [アクセス権限ポリシーをアタッチする] で、ドロップダウンリストから **lambda_execute** ポリシーを選択します。

      ポリシーが一覧に表示されていない場合は、一覧の上部にある [refresh (更新)] ボタンをクリックしてください。(ブラウザページを更新しないでください)。

   b. [Next:Tags (次へ: タグ)] を選択します。

   c. 次: レビュー を選択します。

   d. [ロール名] には、**lambda_invoke_function_assume_apigw_role** などの名前を入力します。

   e. [ロールの作成] を選択します。

9. ロールのリストから **lambda_invoke_function_assume_apigw_role** 関数を選択します。

10. [信頼関係] タブを選択します。

11. [Edit trust relationship (信頼関係の編集)] を選択します。

12. 既存のポリシーを以下に置き換えます。

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

13. [Update Trust Policy] を選択します。

14. 作成したロールのロール ARN を書き留めます。後で必要になります。

Calc Lambda 関数の作成

次に、 Lambda コンソールを使用して Lambda 関数を作成します。

1. Lambda コンソールで、[Create function (関数の作成)] をクリックします。
2. [一から作成] を選択します。
3. [名前] に「**Calc**」と入力します。
4. [ランタイム] で、サポートされている最新の Node.js または Python ランタイムのいずれかを選択します。
5. [Create function (関数の作成)] を選択します。
6. 任意のランタイムで以下の Lambda 関数をコピーし、Lambda コンソールのコードエディターに貼り付けます。

   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. 実行ロールで、[Choose an existing role (既存のロールを選択)] を選択します。
8. 前に作成した **lambda_invoke_function_assume_apigw_role** ロールのロール ARN を入力します。
9. [デプロイ] を選択します。

この関数には、a 入力パラメータからの 2 つのオペランド (b と op) および演算子 (event) が必要です。入力は以下の形式の JSON オブジェクトです。

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

この関数は計算結果 (c) と入力を返します。入力が無効な場合、関数は結果として Null 値または「Invalid op」文字列のいずれかを返します。出力は以下の JSON 形式になります。

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

```
  "c": "Number" | "String"
}
```

この関数は、次のステップで API と統合する前に、Lambda コンソールでテストする必要があります。

`Calc` Lambda 関数をテストする

これが、Lambda コンソールで `Calc` 関数をテストする方法です。

1. [テスト] タブを選択します。
2. テストイベントの名前として **`calc2plus5`** と入力します。
3. テストイベントの定義を次のように置き換えます。

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

4. [保存] を選択します。
5. [Test (テスト)] を選択します。
6. [実行結果: 成功] を展開します。次のように表示されます。

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

`Calc` API を作成する

次の手順は、作成したばかりの `Calc` Lambda 関数用の API を作成する方法を示しています。次のセクションでは、リソースとメソッドをそれに追加します。

API を作成するには

1. https://console.aws.amazon.com/apigateway で API Gateway コンソールにサインインします。

2. API Gateway を初めて使用する場合は、サービスの特徴を紹介するページが表示されます。[REST API] で、[構築] を選択します。[Create Example API (サンプル API の作成)] がポップアップ表示されたら、[OK] を選択します。

   API Gateway を使用するのが初めてではない場合、[Create API] (API を作成)を選択します。[REST API] で、[構築] を選択します。
3. [API 名] に「**LambdaCalc**」と入力します。
4. (オプション) [説明] に説明を入力します。
5. [API エンドポイントタイプ] を [リージョン別] に設定したままにします。
6. [Create API] を選択します。

統合 1: Lambda 関数を呼び出すためのクエリパラメータを持つ **GET** メソッドを作成する

クエリ文字列パラメータを Lambda 関数に渡す GET メソッドを作成することで、ブラウザから API を呼び出すことができます。この方法は、特にオープンアクセスを許可する API には便利です。

API を作成したら、リソースを作成します。通常、API リソースはアプリケーションロジックに従ってリソースツリーに整理されます。このステップでは、/calc リソースを作成します。

/calc リソースを作成するには

1. [リソースの作成] を選択します。
2. [プロキシのリソース] はオフのままにします。
3. [リソースパス] は / のままにします。
4. [リソース名] に「**calc**」と入力します。
5. [CORS (Cross Origin Resource Sharing)] はオフのままにします。
6. [リソースの作成] を選択します。

クエリ文字列パラメータを Lambda 関数に渡す GET メソッドを作成することで、ブラウザから API を呼び出すことができます。この方法は、特にオープンアクセスを許可する API には便利です。

このメソッドの場合、Lambda は、すべての Lambda 関数の呼び出しに POST リクエストを使用することを要求します。この例では、フロントエンドのメソッドリクエストの HTTP メソッドは、バックエンドの統合リクエストとは異なる場合があることを示しています。

GET メソッドを作成するには

1. /calc リソースを選択し、[メソッドを作成] を選択します。
2. [メソッドタイプ] には、GET を選択します。
3. [統合タイプ] で、[AWS のサービス] を選択します。
4. [AWS リージョン] で、Lambda 関数を作成した AWS リージョンを選択します。
5. [AWS のサービス] で、[Lambda] を選択します。
6. [AWS サブドメイン] は空白のままにします。
7. [HTTP メソッド] で、[POST] を選択します。
8. [アクションタイプ] で、[パスオーバーライドを使用] を選択します。このオプションでは、実行する Invoke アクションの ARN を指定して `Calc` 関数を呼び出すことができます。
9. [パスオーバーライド] に「**2015-03-31/functions/arn:aws:lambda:*us-east-2*:*account-id*:function:Calc/invocations**」と入力します。[***account-id***] に AWS アカウント ID を入力します。[***us-east-2***] に、Lambda 関数を作成した AWS リージョンを入力します。
10. [実行ロール] に、**lambda_invoke_function_assume_apigw_role** のロール ARN を入力します。
11. [認証情報キャッシュ] と [デフォルトタイムアウト] の設定は変更しません。
12. [メソッドリクエストの設定] を選択します。
13. [リクエストの検証] で、[クエリ文字列パラメータおよびヘッダーを検証] を選択します。

    この設定により、クライアントが必要なパラメータを指定しなかった場合に、エラーメッセージが返されます。
14. [URL クエリ文字列パラメータ] を選択します。

    次に、[/calc] リソースで GET メソッドのクエリ文字列パラメータを設定し、バックエンドの Lambda 関数に代わって入力を受け取れるようにします。

    クエリ文字列パラメータを作成するには、次の操作を行います。

    a. [クエリ文字列の追加] を選択します。
    b. [名前] に**operand1**と入力します。
    c. [必須] をオンにします。
    d. [キャッシュ] はオフのままにします。

同じ手順を繰り返して、**operand2** という名前のクエリ文字列と **operator** という名前のクエリ文字列を作成します。

15. [メソッドの作成] を選択します。

次に、Calc 関数の要求に応じて、クライアントが指定したクエリ文字列を統合リクエストのペイロードに変換するためのマッピングテンプレートを作成します。このテンプレートでは、[メソッドリクエスト] で宣言した 3 つのクエリ文字列パラメータを、JSON オブジェクトの指定したプロパティ値にマッピングし、バックエンドの Lambda 関数への入力として使用します。変換された JSON オブジェクトは統合リクエストのペイロードとして含まれます。

入力パラメータを統合リクエストにマッピングするには

1. [統合リクエスト] タブの [統合リクエストの設定] で、[編集] を選択します。
2. [リクエスト本文のパススルー] で、[テンプレートが定義されていない場合 (推奨)] を選択します。
3. [マッピングテンプレート] を選択します。
4. [マッピングテンプレートの追加] を選択します。
5. [コンテンツタイプ] に、「**application/json**」と入力します。
6. [テンプレート本文] に、次のコードを入力します。

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

7. [Save] を選択します。

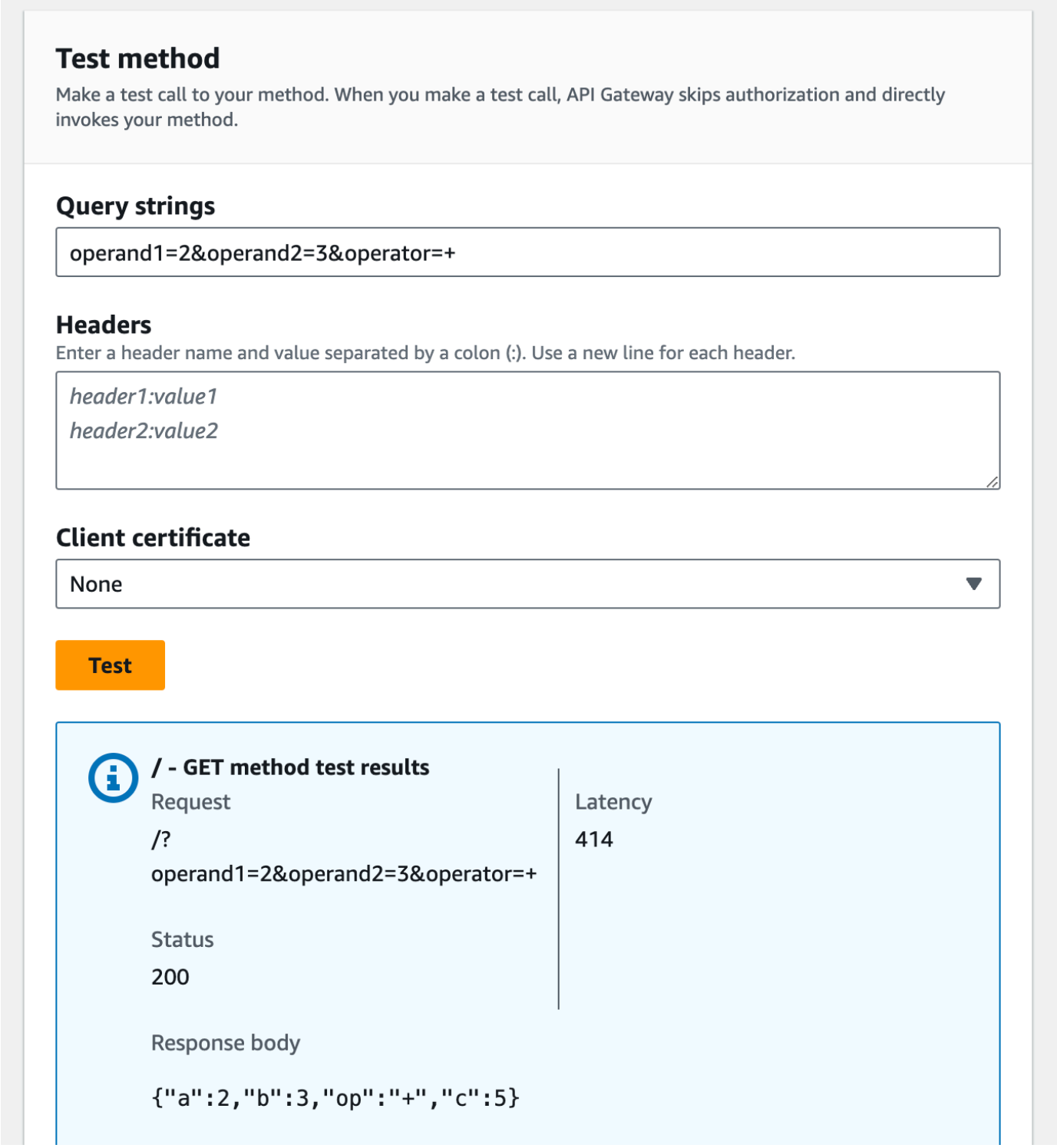
これで、GET メソッドをテストして、このメソッドが Lambda 関数を呼び出すように適切に設定されていることを確認できます。

GET メソッドをテストするには

1. [テスト] タブを選択します。タブを表示するには、右矢印ボタンを選択する必要がある場合があります。
2. [クエリ文字列] に「**operand1=2&operand2=3&operator=+**」と入力します

3. [Test (テスト)] を選択します。

結果は以下のようになります。



統合 2: Lambda 関数を呼び出すための JSON ペイロードを持つ **POST** メソッドを作成する

Lambda 関数を呼び出すための JSON ペイロードを含む POST メソッドを作成することによって、クライアントがリクエストボディのバックエンド関数に必要な入力を提供するようにします。クライアントが正しい入力データをアップロードしたことを確認するために、ペイロードに対してリクエストの検証を有効にします。

JSON ペイロードを持つ **POST** メソッドを作成するには

1. /calc リソースを選択し、[メソッドを作成] を選択します。
2. [メソッドタイプ] では、POST を選択します。
3. [統合タイプ] で、[AWS のサービス] を選択します。
4. [AWS リージョン] で、Lambda 関数を作成した AWS リージョンを選択します。
5. [AWS のサービス] で、[Lambda] を選択します。
6. [AWS サブドメイン] は空白のままにします。
7. [HTTP メソッド] で、[POST] を選択します。
8. [アクションタイプ] で、[パスオーバーライドを使用] を選択します。このオプションでは、実行する Invoke アクションの ARN を指定して `Calc` 関数を呼び出すことができます。
9. [パスオーバーライド] に「**`2015-03-31/functions/arn:aws:lambda:`*`us-east-2`*`:`*`account-id`*`:function:Calc/invocations`**」と入力します。[***`account-id`***] に AWS アカウント ID を入力します。[***`us-east-2`***] に、Lambda 関数を作成した AWS リージョンを入力します。
10. [実行ロール] に、**`lambda_invoke_function_assume_apigw_role`** のロール ARN を入力します。
11. [認証情報キャッシュ] と [デフォルトタイムアウト] の設定は変更しません。
12. [メソッドの作成] を選択します。

次に、入力データ構造を記述して受信リクエスト本文を検証するための [入力] モデルを作成します。

入力モデルを作成するには

1. ナビゲーションペインで、[モデル] を選択します。
2. [モデルの作成] を選択します。

3. [名前] に**input**と入力します。
4. [コンテンツタイプ] に、「**application/json**」と入力します。

   一致するコンテンツタイプが見つからない場合、リクエストの検証は実行されません。コンテンツタイプに関係なく同じモデルを使用するには、「**$default**」と入力します。
5. [モデルのスキーマ] に次のモデルを入力します。

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

6. [モデルの作成] を選択します。

次に、[出力] モデルを作成します。このモデルでは、バックエンドの計算結果のデータ構造を記述します。このモデルを使用して統合レスポンスデータを別のモデルにマッピングできます。このチュートリアルではパススルー動作を利用するため、このモデルは使用しません。

出力モデルを作成するには

1. [モデルの作成] を選択します。
2. [名前] に**output**と入力します。
3. [コンテンツタイプ] に、「**application/json**」と入力します。

   一致するコンテンツタイプが見つからない場合、リクエストの検証は実行されません。コンテンツタイプに関係なく同じモデルを使用するには、「**$default**」と入力します。
4. [モデルのスキーマ] に次のモデルを入力します。

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

```
}
```

5. [モデルの作成] を選択します。

次に、[結果] モデルを作成します。このモデルでは、返されたレスポンスデータのデータ構造を記述します。これは、API で定義されている [入力] スキーマと [出力] スキーマの両方を参照します。

結果モデルを作成するには

1. [モデルの作成] を選択します。
2. [名前] に**result**と入力します。
3. [コンテンツタイプ] に、「**application/json**」と入力します。

   一致するコンテンツタイプが見つからない場合、リクエストの検証は実行されません。コンテンツタイプに関係なく同じモデルを使用するには、「**$default**」と入力します。
4. [モデルスキーマ] に、*restapi-id* を使用して、次のモデルを入力します。*restapi-id* はコンソールの上部に、このフロー (API Gateway > APIs > LambdaCalc (*abc123*).) で括弧内に表示されます。

```
{
    "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. [モデルの作成] を選択します。

次に、受信リクエスト本文に対するリクエストの検証を有効にする POST メソッドのメソッドリクエストを設定します。

POST メソッドのリクエスト検証を有効にするには

1. メインナビゲーションペインで [リソース] を選択し、リソースツリーの POST メソッドを選択します。
2. [メソッドリクエスト] タブの [メソッドリクエスト設定] で、[編集] を選択します。
3. [リクエストの検証] で、[本文を検証] を選択します。
4. [リクエスト本文] を選択し、[モデルを追加] を選択します。
5. [コンテンツタイプ] に、「**application/json**」と入力します。

   一致するコンテンツタイプが見つからない場合、リクエストの検証は実行されません。コンテンツタイプに関係なく同じモデルを使用するには、「**$default**」と入力します。
6. [モデル] で [入力] を選択します。
7. [Save] を選択します。

これで、POST メソッドをテストして、このメソッドが Lambda 関数を呼び出すように適切に設定されていることを確認できます。

POST メソッドをテストするには

1. [テスト] タブを選択します。タブを表示するには、右矢印ボタンを選択する必要がある場合があります。
2. [リクエスト本文] に次の JSON ペイロードを入力します。

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

3. [テスト] を選択します。

   以下の出力が表示されます。

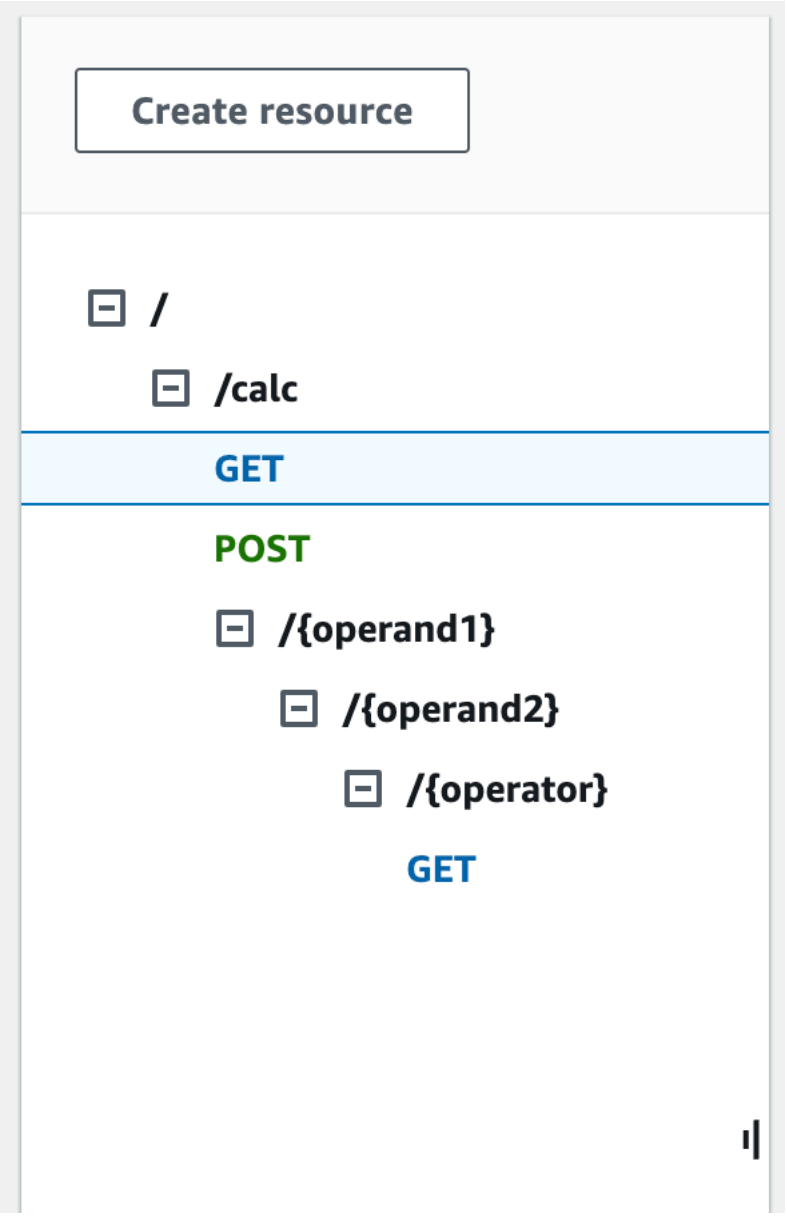
```
{
  "a": 1,
  "b": 2,
```

```
  "op": "+",
  "c": 3
}
```

統合 3: Lambda 関数を呼び出すためのパスパラメータを持つ **GET** メソッドを作成する

次に、一連のパスパラメータで指定されたリソースで GET メソッドを作成し、バックエンドの Lambda 関数を呼び出します。パスパラメータの値は、Lambda 関数への入力データを指定します。受信パスパラメータ値を必要な統合リクエストペイロードにマッピングするためのマッピングテンプレートを使用します。

生成された API リソースの構造は次のようになります。



/{operand1}/{operand2}/{operator} リソースを作成するには

1. [リソースの作成] を選択します。
2. [リソースパス] で、`/calc` を選択します。
3. [リソース名] に「**`{operand1}`**」と入力します。
4. [CORS (Cross Origin Resource Sharing)] はオフのままにします。
5. [リソースの作成] を選択します。
6. [リソースパス] で、`/calc/{operand1}/` を選択します。
7. [リソース名] に「**`{operand2}`**」と入力します。

8. [CORS (Cross Origin Resource Sharing)] はオフのままにします。
9. [リソースの作成] を選択します。
10. [リソースパス] で、`/calc/{operand1}/{operand2}/` を選択します。
11. [リソース名] に「**`{operator}`**」と入力します。
12. [CORS (Cross Origin Resource Sharing)] はオフのままにします。
13. [リソースの作成] を選択します。

今回は、API Gateway コンソールに組み込まれている Lambda 統合を使用してメソッド統合を設定します。

メソッド統合を設定するには

1. /{operand1}/{operand2}/{operator} を選択し、[メソッドを作成] を選択します。
2. [メソッドタイプ] には、GET を選択します。
3. [統合タイプ] で、[Lambda 関数] を選択します。
4. [Lambda プロキシ統合] はオフのままにしておきます。
5. [Lambda 関数] で、Lambda 関数を作成した AWS リージョンを選択し、「**`Calc`**」と入力します。
6. [デフォルトタイムアウト] はオンのままにします。
7. [メソッドの作成] を選択します。

次に、マッピングテンプレートを作成し、/calc/{operand1}/{operand2}/{operator} リソースの作成時に宣言した 3 つの URL パスパラメータを、JSON オブジェクトで指定したプロパティ値にマッピングします。URL パスは URL エンコードされる必要があるため、除算演算子を %2F ではなく / として指定してください。このテンプレート では、%2F を `'/'` に変換してから Lambda 関数に渡します。

マッピングテンプレートを作成するには

1. [統合リクエスト] タブの [統合リクエストの設定] で、[編集] を選択します。
2. [リクエスト本文のパススルー] で、[テンプレートが定義されていない場合 (推奨)] を選択します。
3. [マッピングテンプレート] を選択します。
4. [コンテンツタイプ] に、「**`application/json`**」と入力します。

5. [テンプレート本文] に、次のコードを入力します。

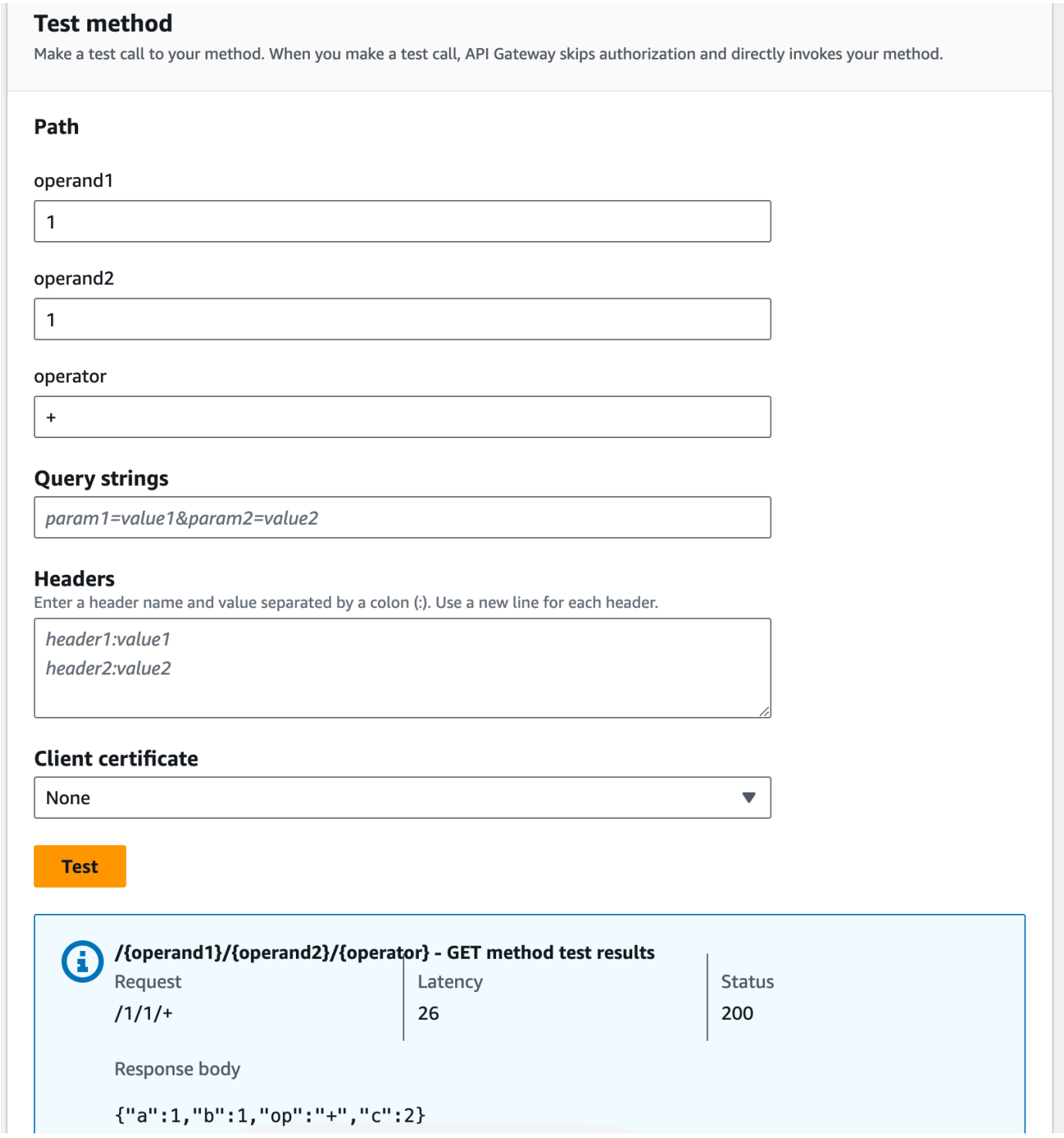
```
{
   "a": "$input.params('operand1')",
   "b": "$input.params('operand2')",
   "op":
 #if($input.params('operator')=='%2F')"/"#{else}"$input.params('operator')"#end
}
```

6. [Save] を選択します。

これで、GET メソッドをテストし、このメソッドが Lambda 関数を呼び出して、マッピングなしで統合レスポンスを介して元の出力を渡すように適切に設定されていることを確認できます。

GET メソッドをテストするには

1. [テスト] タブを選択します。タブを表示するには、右矢印ボタンを選択する必要がある場合があります。
2. [パス] で、次の操作を行います。
   a. [operand1] に「**1**」と入力します。
   b. [operand2] に「**1**」と入力します。
   c. [operator] に「**+**」と入力します。
3. [Test (テスト)] を選択します。
4. 結果は以下のようになります。



次に、`result` スキーマに従って、メソッドレスポンスペイロードのデータ構造をモデル化します。

デフォルトでは、メソッドレスポンス本文に空のモデルが割り当てられます。これにより、統合レスポンス本文がマッピングなしで渡されます。ただし、Java や Objective-C などの厳密に型指定された言語のいずれかの SDK を生成すると、SDK ユーザーは空のオブジェクトを結果として受け取りま

す。REST クライアントと SDK クライアントの両方が必要な結果を受け取るためには、事前定義済みのスキーマを使用してレスポンスデータをモデル化する必要があります。ここでは、メソッドレスポンス本文のモデルを定義し、マッピングテンプレートを構築して統合レスポンス本文をメソッドレスポンス本文に変換します。

メソッドレスポンスを作成するには

1. [メソッドレスポンス] タブの [レスポンス 200] で、[編集] を選択します。
2. [リクエスト本文] で、[モデルを追加] を選択します。
3. [コンテンツタイプ] に、「**application/json**」と入力します。
4. [モデル] で、[結果] を選択します。
5. [Save] を選択します。

メソッドレスポンス本文のモデルを設定すると、レスポンスデータは該当する SDK の `result` オブジェクトにキャストされます。それに応じて統合レスポンスデータが確実にマッピングされるようにするには、マッピングテンプレートが必要です。

マッピングテンプレートを作成するには

1. [統合レスポンス] タブの [デフォルト - レスポンス] で、[編集] を選択します。
2. [マッピングテンプレート] を選択します。
3. [コンテンツタイプ] に、「**application/json**」と入力します。
4. [テンプレート本文] に、次のコードを入力します。

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

5. [Save] を選択します。

マッピングテンプレートをテストするには

1. [テスト] タブを選択します。タブを表示するには、右矢印ボタンを選択する必要がある場合があります。
2. [パス] で、次の操作を行います。
   a. [operand1] に「**1**」と入力します。
   b. [operand2] に「**2**」と入力します。
   c. [operator] に「**+**」と入力します。
3. [テスト] を選択します。
4. 結果は次のようになります。

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

この時点では、API Gateway コンソールの [テスト] 機能を使用してのみ API を呼び出すことができます。クライアントが利用できるようにするには、API をデプロイする必要があります。リソースやメソッドを追加、変更、削除したり、データマッピングを更新したり、ステージ設定を更新したりするときには、必ず API を再デプロイしてください。そうしないと、新しい機能やアップデートは API のクライアントに利用可能になりません。デプロイ手順は次のとおりです。

API をデプロイする

1. [API のデプロイ] を選択します。
2. [ステージ] で [新規ステージ] を選択します。
3. [Stage name (ステージ名)] に **`Prod`** と入力します。
4. (オプション) [説明] に説明を入力します。
5. [デプロイ] を選択します。

6. (オプション) [ステージの詳細] の [呼び出し URL] で、コピーアイコンを選択して API の呼び出し URL をコピーします。Postman や cURL のようなツールでこれを使用して API をテストできます。

> **Note**
>
> リソースやメソッドの追加、変更、削除、データマッピングの更新、ステージ設定の更新を行う場合は、必ず API を再デプロイします。そうしない場合、新しい機能やアップデートは API のクライアントには利用できません。

Lambda 関数と統合されたサンプル API の 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
    }
  }
}
```

チュートリアル: API Gateway で REST API を Amazon S3 のプロキシとして作成する

このセクションでは、API Gateway で REST API を Amazon S3 のプロキシとして使用する例として、Amazon S3 の以下のオペレーションを公開するために REST API を作成して設定する方法について説明します。

- API のルートリソースに対するメソッドとして、呼び出し元のすべての Amazon S3 バケットを一覧表示する GET を公開する。
- Folder リソースに対するメソッドとして、Amazon S3 バケット内のすべてのオブジェクトを一覧表示する GET を公開する。
- Folder/Item リソースに対するメソッドとして、Amazon S3 バケットからオブジェクトをダウンロードする GET を公開する。

Amazon S3 のプロキシとして、「Amazon S3 のプロキシとしてのサンプル API の OpenAPI 定義」に示すように、サンプル API をインポートすることもできます。このサンプルには、より多くの公開メソッドが含まれています。OpenAPI 定義を使用して API をインポートする方法については、「OpenAPI を使用した REST API の設定」を参照してください。

> **Note**
>
> API Gateway の API を Amazon S3 と統合するには、API Gateway と Amazon S3 の両方のサービスが利用できるリージョンを選択する必要があります。利用できるリージョンについては、「Amazon API Gateway エンドポイントとクォータ」を参照してください。

トピック

- API で Amazon S3 のアクションを呼び出すための IAM アクセス許可を設定する
- Amazon S3 のリソースを表す API のリソースを作成する
- 呼び出し元の Amazon S3 バケットを一覧表示する API のメソッドを公開する
- Amazon S3 バケットにアクセスする API のメソッドを公開する
- バケット内の Amazon S3 オブジェクトにアクセスする API のメソッドを公開する
- Amazon S3 のプロキシとしてのサンプル API の OpenAPI 定義
- REST API クライアントを使用して API を呼び出す

API で Amazon S3 のアクションを呼び出すための IAM アクセス許可を設定する

Amazon S3 のアクションを呼び出すことを API に許可するには、IAM ロールに適切な IAM ポリシーをアタッチする必要があります。

AWS のサービスプロキシの実行ロールを作成するには

1. AWS Management Console にサインインして、IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. [ロール] を選択します。
3. [ロールの作成] を選択します。
4. [信頼されたエンティティの種類を選択] で [AWS のサービス] を選択し、[API Gateway]、[API Gateway が CloudWatch Logs にログをプッシュすることを許可] の順に選択します。
5. [次へ] を選択し、さらに [次へ] を選択します。
6. [ロール名] に「**APIGatewayS3ProxyPolicy**」と入力し、[ロールの作成] を選択します。
7. [ロール] リストで、作成したロールを選択します。ロールを検索するには、必要に応じてスクロールするか、検索バーを使用します。
8. 選択したロールの [アクセス許可を追加] タブを選択します。
9. ドロップダウンリストから [ポリシーをアタッチ] を選択します。
10. 検索バーに「**AmazonS3FullAccess**」と入力し、[アクセス許可を追加] を選択します。

> **Note**
>
> このチュートリアルでは、わかりやすくするために管理ポリシーを使用しますが、独自の IAM ポリシーを作成して、必要な最小限のアクセス許可を付与するのがベストプラクティスです。

11. 新しく作成したロール ARN をメモしておきます。これは後で使用します。

Amazon S3 のリソースを表す API のリソースを作成する

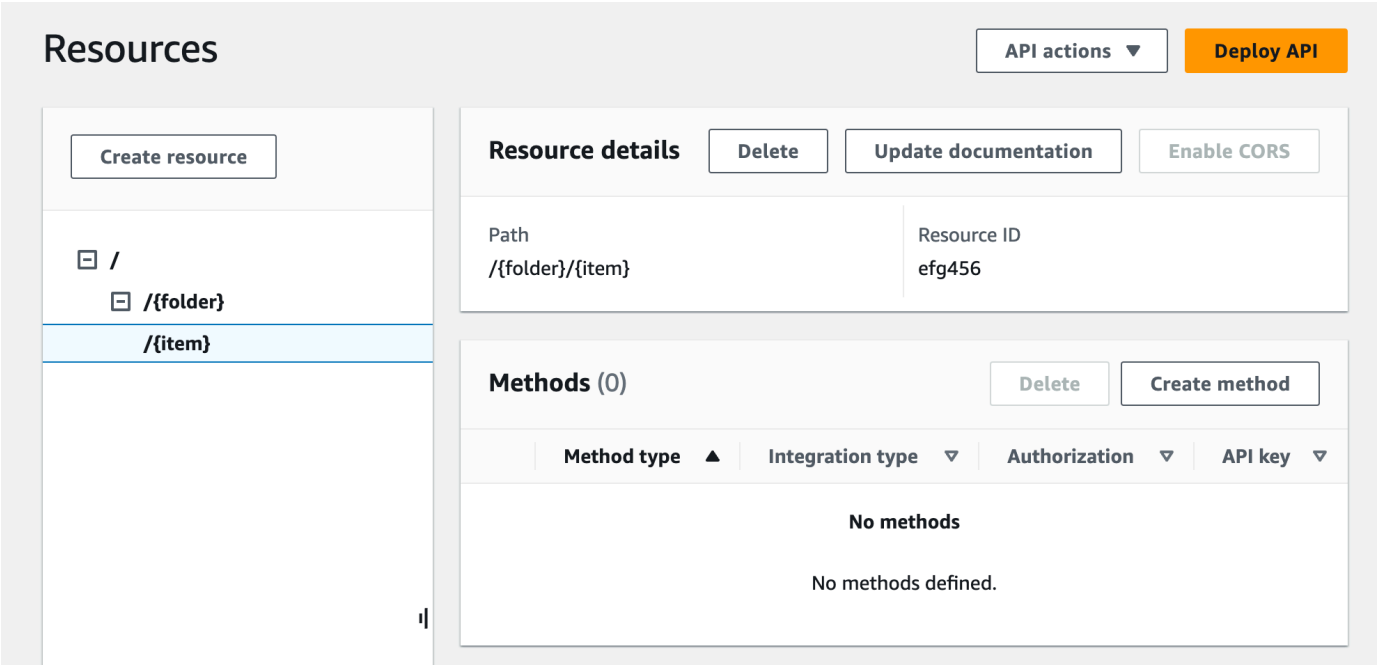
API のルート (/) リソースを、認証された呼び出し元の Amazon S3 バケットのコンテナとして使用します。また、特定の Amazon S3 バケットと特定の Amazon S3 オブジェクトを表す `Folder` リソースと `Item` リソースもそれぞれ作成します。フォルダ名とオブジェクトキーは、発信者により、リクエスト URL の一部としてパスパラメータの形式で指定されます。

> **Note**
>
> オブジェクトキーに / やその他の特殊文字が含まれているオブジェクトにアクセスする場合は、文字を URL エンコードする必要があります。たとえば、`test/test.txt` は `test%2Ftest.txt` にエンコードする必要があります。

Amazon S3 サービスの機能を公開する API のリソースを作成するには

1. Amazon S3 バケットを作成したのと同じ AWS リージョンで、MyS3 という名前の API を作成します。この API のルートリソース (/) は Amazon S3 サービスを表します。このステップでは、/{folder} と /{item} という 2 つの追加のリソースを作成します。
2. API のルートリソースを選択し、[リソースを作成] を選択します。
3. [プロキシのリソース] はオフのままにします。
4. [リソースパス] で、[/] を選択します。
5. [リソース名] に「**`{folder}`**」と入力します。
6. [CORS (Cross Origin Resource Sharing)] はオフのままにします。
7. [リソースの作成] を選択します。
8. /{folder} リソースを選択し、[リソースを作成] を選択します。
9. 前のステップを使用して、/{folder} の子リソースを **`{item}`** という名前で作成します。

最終的な API は次のようになります。



呼び出し元の Amazon S3 バケットを一覧表示する API のメソッドを公開する

呼び出し元の Amazon S3 バケットの一覧を取得するには、Amazon S3 に対して GET サービスアクションを呼び出す必要があります。API のルートリソース (/) で、GET メソッドを作成します。GET メソッドを設定して Amazon S3 と統合するには、以下の手順に従います。

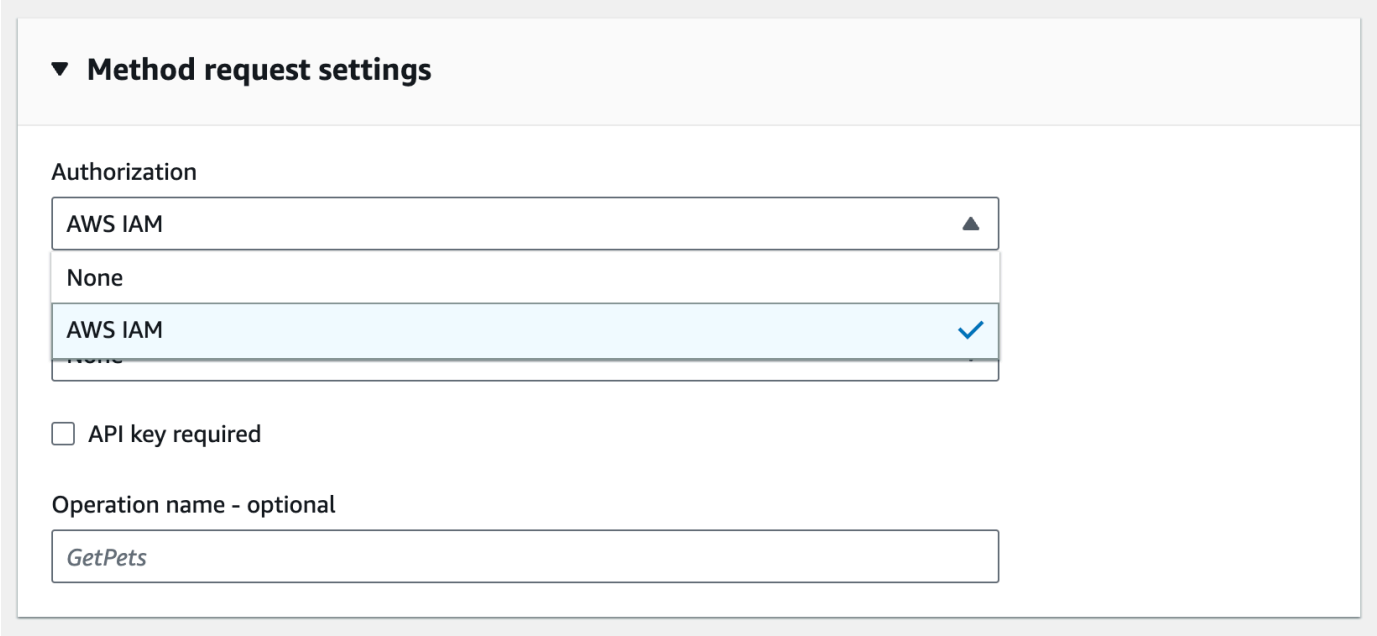
API の **GET /** メソッドを作成し、初期化するには

1. / リソースを選択し、[メソッドを作成] を選択します。
2. メソッドタイプとして、[GET] を選択します。
3. [統合タイプ] で、[AWS のサービス] を選択します。
4. [AWS リージョン] で、Amazon S3 バケットを作成した AWS リージョンを選択します。
5. [AWS のサービス] で、[Amazon Simple Storage Service] を選択します。
6. [AWS サブドメイン] は空白のままにします。
7. [HTTP メソッド] で、[GET] を選択します。
8. [アクションタイプ] で、[パスオーバーライドを使用] を選択します。

パスの上書きを使用すると、API Gateway はクライアントのリクエストを対応する Amazon S3 の REST API のパス形式のリクエストとして Amazon S3 に転送します。このリクエストでは、Amazon S3 のリソースは `s3-host-name/bucket/key` 形式のリソースパスで表されます。API Gateway は `s3-host-name` を設定し、クライアントが指定した `bucket` と `key` をクライアントから Amazon S3 に渡します。

9. [パスオーバーライド]] に「/」と入力します。
10. [実行ロール] に、**APIGatewayS3ProxyPolicy** のロール ARN を入力します。
11. [メソッドリクエストの設定] を選択します。

    メソッドリクエストの設定を使用して、この API メソッドを誰が呼び出せるかを制御します。
12. [認可] で、ドロップダウンメニューから [AWS_IAM] を選択します。



13. [メソッドの作成] を選択します。

この設定には、フロントエンドの `GET https://your-api-host/stage/` リクエストとバックエンドの `GET https://your-s3-host/` が含まれます。

API から呼び出し元に正常なレスポンスと例外を適切に返すようにするには、[メソッドレスポンス] で 200、400、500 のレスポンスを宣言します。200 レスポンスのデフォルトのマッピングを使用すると、ここで宣言されていないステータスコードのバックエンドレスポンスが 200 レスポンスとして呼び出し元に返されます。

GET / メソッドのレスポンスタイプを宣言するには

1. [メソッドレスポンス] タブの [レスポンス 200] で、[編集] を選択します。
2. [ヘッダーを追加] を選択し、次の操作を行います。

   a. [ヘッダー名] に「**Content-Type**」と入力します。
   b. [ヘッダーの追加] を選択します。

   次の手順を繰り返して、**Timestamp** ヘッダーと **Content-Length** ヘッダーを作成します。
3. [Save] を選択します。
4. [メソッドレスポンス] タブの [メソッドレスポンス] で、[レスポンスを作成] を選択します。
5. [HTTP ステータスコード] に「400」と入力します。

   このレスポンスにはヘッダーを設定しません。
6. [Save] を選択します。
7. 次の手順を繰り返して 500 レスポンスを作成します。

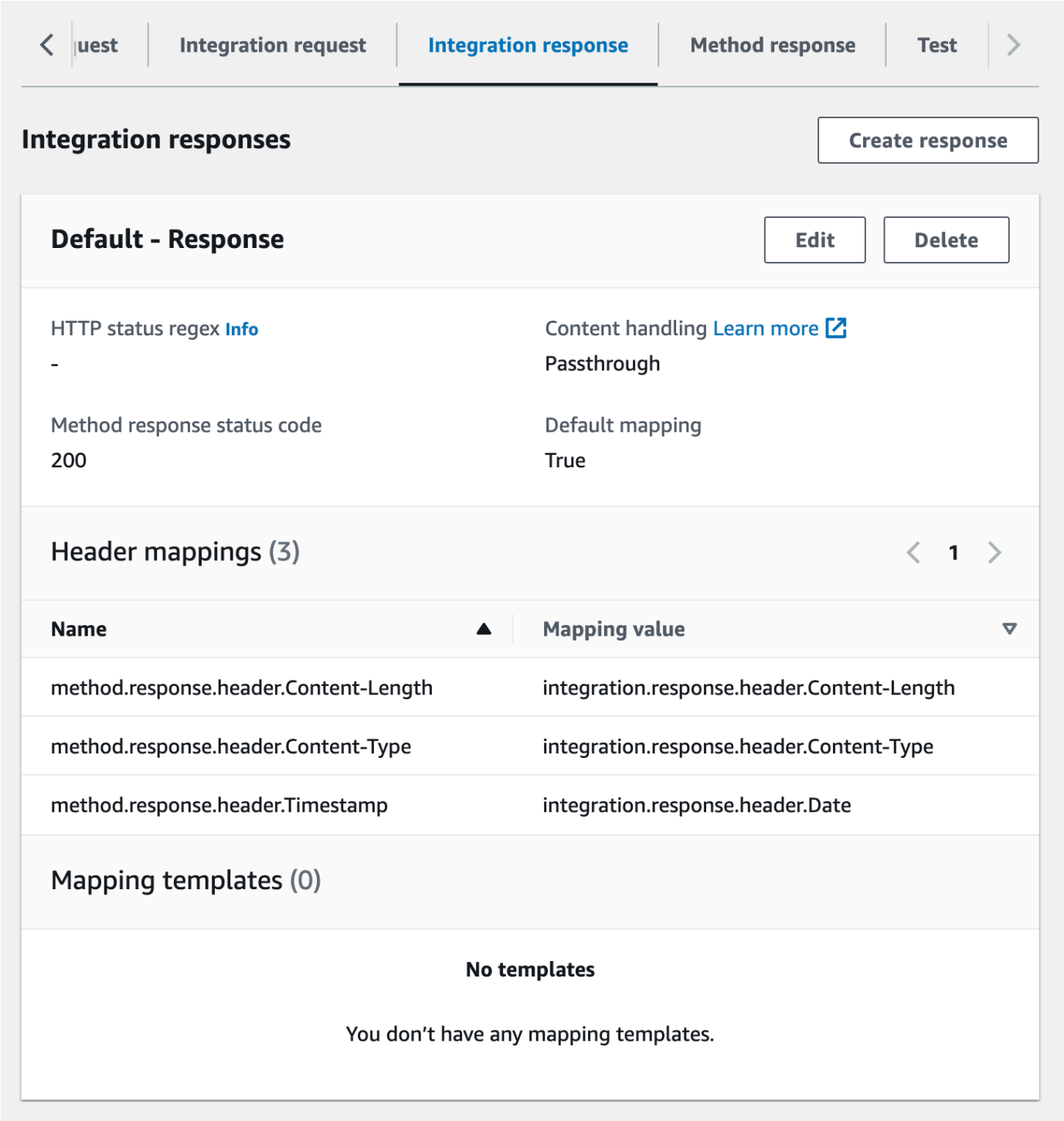
   このレスポンスにはヘッダーを設定しません。

Amazon S3 からの正常な統合レスポンスはバケットのリストを XML ペイロードとして返し、API Gateway からのデフォルトのメソッドレスポンスは JSON ペイロードを返すため、バックエンドの Content-Type ヘッダーパラメータ値をフロントエンドの対応する値にマッピングする必要があります。そうしないと、クライアントは、レスポンスの本文が実際には XML 文字列である場合に `application/json` のコンテンツタイプを受け取ってしまいます。以下に、設定の手順を示します。また、Date や Content-Length などの他のヘッダーパラメータをクライアントに表示することもできます。

GET / メソッドのレスポンスヘッダーのマッピングを設定する

1. [統合レスポンス] タブの [デフォルト - レスポンス] で、[編集] を選択します。
2. Content-Length ヘッダーに、マッピング値として「**integration.response.header.Content-Length**」と入力します。
3. Content-Type ヘッダーに、マッピング値として「**integration.response.header.Content-Type**」と入力します。
4. Timestamp ヘッダーに、マッピング値として「**integration.response.header.Date**」と入力します。

5. [Save] を選択します。結果は次のようになります。



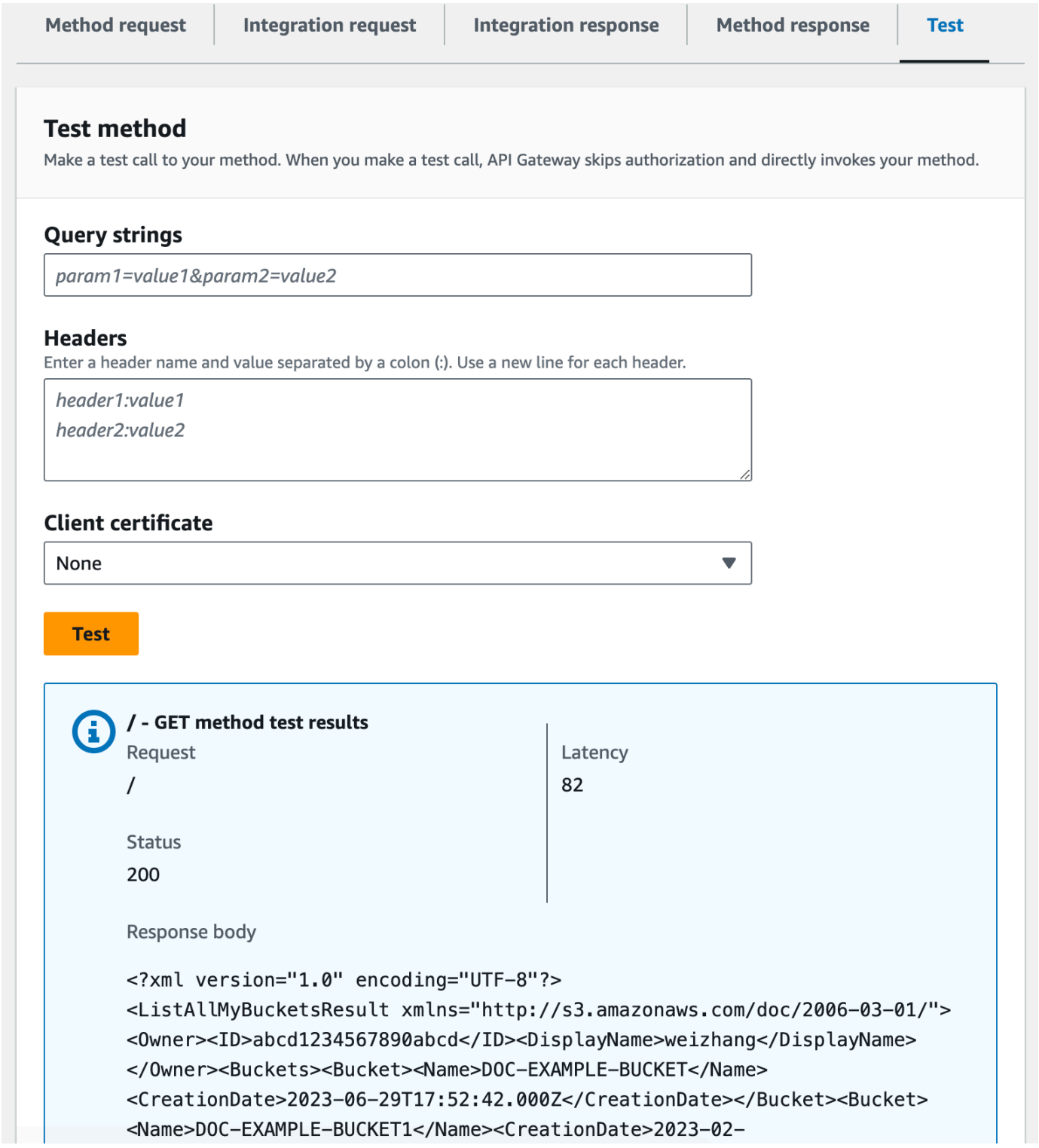
6. [統合レスポンス] タブの [統合レスポンス] で、[レスポンスを作成] を選択します。
7. [HTTP status regex (HTTP ステータスの正規表現)]に「**4\d{2}**」と入力します。これにより、すべての 4xx HTTP レスポンスのステータスコードがメソッドレスポンスにマッピングされます。

8. [メソッドレスポンスのステータスコード] で [**400**] を選択します。
9. [Create] (作成) を選択します。
10. 次の手順を繰り返して、500 メソッドレスポンスの統合レスポンスを作成します。[HTTP status regex (HTTP ステータスの正規表現)]に「**5\d{2}**」と入力します。

作業を適切に進めるために、ここまで設定した API をテストできます。

GET / メソッドをテストするには

1. [テスト] タブを選択します。タブを表示するには、右矢印ボタンを選択する必要がある場合があります。
2. [テスト] を選択します。結果は次の図のようになります。


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-

Amazon S3 バケットにアクセスする API のメソッドを公開する

Amazon S3 バケットを使用するには、/{folder} リソースの GET メソッドを公開して、バケット内のオブジェクトを一覧表示します。手順は、呼び出し元の Amazon S3 バケットを一覧表示する API のメソッドを公開する で説明されている手順と同様です。その他のメソッドについては、サンプル API をこちら (Amazon S3 のプロキシとしてのサンプル API の OpenAPI 定義) でインポートできます。

フォルダリソースで GET メソッドを公開するには

1. /{folder} リソースを選択し、[メソッドを作成] を選択します。
2. メソッドタイプとして、[GET] を選択します。
3. [統合タイプ] で、[AWS のサービス] を選択します。
4. [AWS リージョン] で、Amazon S3 バケットを作成した AWS リージョンを選択します。
5. [AWS のサービス] で、[Amazon Simple Storage Service] を選択します。
6. [AWS サブドメイン] は空白のままにします。
7. [HTTP メソッド] で、[GET] を選択します。
8. [アクションタイプ] で、[パスオーバーライドを使用] を選択します。
9. [パスオーバーライド] に「**{bucket}**」と入力します。
10. [実行ロール] に、**APIGatewayS3ProxyPolicy** のロール ARN を入力します。
11. [メソッドの作成] を選択します。

Amazon S3 エンドポイントの URL で {folder} パスパラメータを設定します。メソッドリクエストの {folder} パスパラメータを統合リクエストの {bucket} パスパラメータにマッピングする必要があります。

{folder} を **{bucket}** にマッピングするには

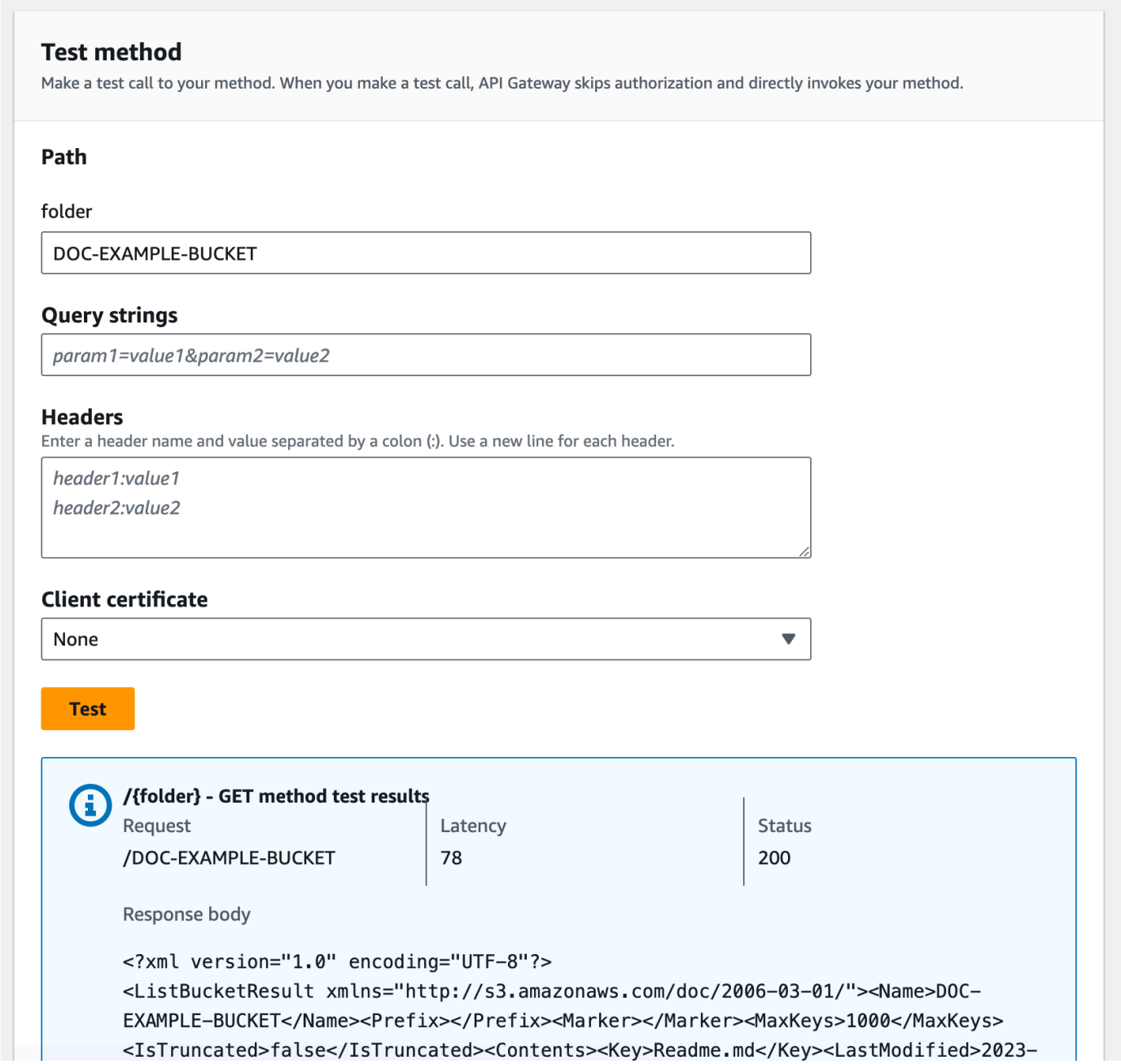
1. [統合リクエスト] タブの [統合リクエストの設定] で [編集] を選択します。
2. [URL パスパラメータ]、[パスパラメータを追加] を選択します。
3. [名前] に**bucket**と入力します。
4. [マッピング元] として「**method.request.path.folder**」と入力します。
5. [Save] を選択します。

次に API をテストします。

/{folder} GET メソッドをテストするには

1. [テスト] タブを選択します。タブを表示するには、右矢印ボタンを選択する必要がある場合があります。
2. [パス] の [folder] に、バケットの名前を入力します。
3. [テスト] を選択します。

   テスト結果には、バケット内のオブジェクトのリストが含まれます。



バケット内の Amazon S3 オブジェクトにアクセスする API のメソッドを公開する

Amazon S3 では、GET、DELETE、HEAD、OPTIONS、POST、PUT の各アクションで特定のバケット内のオブジェクトにアクセスし、管理することができます。このチュートリアルでは、`{folder}/{item}` リソースでバケットから画像を取得する `GET` メソッドを公開します。`{folder}/{item}` リソースのその他の用途については、サンプル API (Amazon S3 のプロキシとしてのサンプル API の OpenAPI 定義) を参照してください。

項目リソースで GET メソッドを公開するには

1. /{item} リソースを選択し、[メソッドを作成] を選択します。
2. メソッドタイプとして、[GET] を選択します。
3. [統合タイプ] で、[AWS のサービス] を選択します。
4. [AWS リージョン] で、Amazon S3 バケットを作成した AWS リージョンを選択します。
5. [AWS のサービス] で、[Amazon Simple Storage Service] を選択します。
6. [AWS サブドメイン] は空白のままにします。
7. [HTTP メソッド] で、[GET] を選択します。
8. [アクションタイプ] で、[パスオーバーライドを使用] を選択します。
9. [パスオーバーライド] に、「{bucket}/{object}」と入力します。
10. [実行ロール] に、**APIGatewayS3ProxyPolicy** のロール ARN を入力します。
11. [メソッドの作成] を選択します。

Amazon S3 エンドポイント URL で `{folder}` パスパラメータと `{item}` パスパラメータを設定します。メソッドリクエストの パスパラメータを統合リクエストの パスパラメータにマッピングする必要があります。

このステップでは、次の作業を行います。

- メソッドリクエストの `{folder}` パスパラメータを統合リクエストの `{bucket}` パスパラメータにマッピングします。
- メソッドリクエストの `{item}` パスパラメータを統合リクエストの `{object}` パスパラメータにマッピングします。

{folder} を {bucket} に、{item} を {object} にマッピングするには

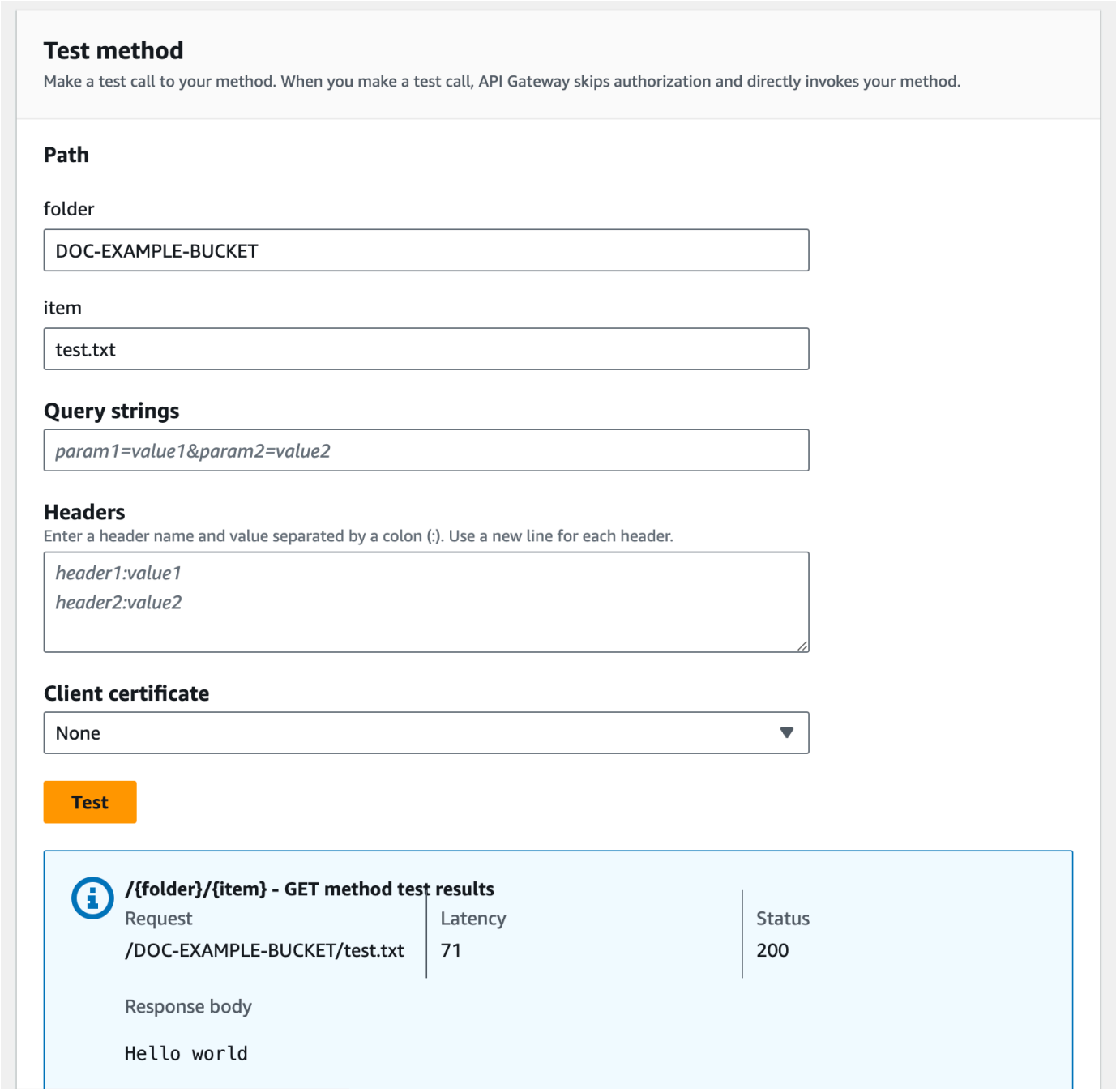
1. [統合リクエスト] タブの [統合リクエストの設定] で、[編集] を選択します。

2. [URL パスパラメータ] を選択します。
3. [パスパラメータを追加] を選択します。
4. [名前] に**bucket**と入力します。
5. [マッピング元] として「**method.request.path.folder**」と入力します。
6. [パスパラメータを追加] を選択します。
7. [名前] に**object**と入力します。
8. [マッピング元] として「**method.request.path.item**」と入力します。
9. [Save] を選択します。

/{folder}/{object} GET メソッドをテストするには

1. [テスト] タブを選択します。タブを表示するには、右矢印ボタンを選択する必要がある場合があります。
2. [パス] の [folder] に、バケットの名前を入力します。
3. [パス] の [item] に、項目の名前を入力します。
4. [テスト] を選択します。

   レスポンス本文に項目の内容が含まれます。



リクエストは、指定した Amazon S3 バケット (DOC-EXAMPLE-BUCKET) の指定したファイル (test.txt) のコンテンツとして、プレーンテキストの (「Hello world」) を正しく返します。

API Gateway で UTF-8 でエンコードされた JSON 以外のコンテンツとみなされるバイナリファイルをダウンロードまたはアップロードするには、API に追加の設定が必要です。次のように説明されています。

S3 からバイナリファイルをダウンロードまたは S3 にアップロードするには

1. 影響を受けるファイルのメディアの種類を API の binaryMediaTypes に登録します。これは、コンソールで登録できます。

   a. API の [API 設定] を選択します。
   b. [バイナリメディアタイプ] で、[メディアタイプを管理] を選択します。
   c. [バイナリメディアタイプを追加] を選択し、必要なメディアタイプ (例: `image/png`) を入力します。
   d. [Save changes] (変更の保存) を選択して設定を保存します。
2. `Content-Type` ヘッダー (アップロードの場合) と `Accept` ヘッダー (ダウンロードの場合) をメソッドリクエストに追加し、必要なバイナリメディアタイプを指定するようにクライアントに要求して、統合リクエストにマッピングします。
3. 登録リクエスト (アップロードの場合) および 統合レスポンス (ダウンロードの場合) で、[コンテンツの処理] を [`Passthrough`] に設定します。影響のあるコンテンツタイプで定義されているマッピングテンプレートがないことを確認します。詳細については、「統合パススルーの動作」と「VTL マッピングテンプレートを選択する」を参照してください。

ペイロードサイズの上限は 10 MB です。「REST API の設定および実行に関する API Gateway クォータ」を参照してください。

Amazon S3 のファイルのメタデータに正しいコンテンツタイプが追加されていることを確認してください。ストリーミング可能なメディアコンテンツの場合も、`Content-Disposition:inline` をメタデータに追加する必要がある場合があります。

API Gateway でのバイナリのサポートの詳細については、「API Gateway でのコンテンツタイプの変換」を参照してください。

Amazon S3 のプロキシとしてのサンプル API の OpenAPI 定義

以下の OpenAPI 定義では、Amazon S3 のプロキシとして動作する API について説明しています。この API には、チュートリアルで作成した API よりも多くの Amazon S3 オペレーションが含まれています。OpenAPI 定義では以下のメソッドが公開されています。

- API のルートリソースに対するメソッドとして、呼び出し元のすべての Amazon S3 バケットを一覧表示する GET を公開する。
- Folder リソースに対するメソッドとして、Amazon S3 バケット内のすべてのオブジェクトを一覧表示する GET を公開する。

- Folder リソースに対するメソッドとして、Amazon S3 にバケットを追加する PUT を公開する。
- Folder リソースに対するメソッドとして、Amazon S3 からバケットを削除する DELETE を公開する。
- Folder/Item リソースに対するメソッドとして、Amazon S3 バケットからオブジェクトをダウンロードする GET を公開する。
- Folder/Item リソースに対するメソッドとして、Amazon S3 バケットにオブジェクトをアップロードする PUT を公開する。
- Folder/Item リソースに対するメソッドとして、Amazon S3 バケット内のオブジェクトのメタデータを取得する HEAD を公開する。
- Folder/Item リソースに対するメソッドとして、Amazon S3 バケットからオブジェクトを削除する DELETE を公開する。

OpenAPI 定義を使用して API をインポートする方法については、「OpenAPI を使用した REST API の設定」を参照してください。

同様の API を作成する方法の手順については、「チュートリアル: API Gateway で REST API を Amazon S3 のプロキシとして作成する」を参照してください。

この API を、AWS IAM 認証をサポートする Postman を使用して呼び出す方法については、「REST API クライアントを使用して API を呼び出す」を参照してください。

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"
      }
    }
  }
}
```

REST API クライアントを使用して API を呼び出す

エンドツーエンドのチュートリアルを提供するため、ここで AWS IAM 認証をサポートする Postman を使用して API を呼び出す方法を説明します。

Postman を使用して Amazon S3 のプロキシの API を呼び出すには

1. API をデプロイまたは再デプロイします。[ステージエディタ] の最上部にある [呼び出し URL] の横に表示された API のベース URL を書き留めます。
2. Postman を起動します。

3. [Authorization] (認可) を選択し、次に `AWS Signature` を選択します。[AccessKey] と [SecretKey] のそれぞれの入力フィールドに、IAM ユーザーのアクセスキー ID とシークレットアクセスキーを入力します。[AWS Region] (AWS リージョン) テキストボックスに API のデプロイ先の AWS リージョンを入力します。[サービス名] 入力フィールドに `execute-api` と入力します。

   キーのペアは、IAM マネジメントコンソールの IAM ユーザーアカウントの [Security Credentials (認証情報)] タブで作成できます。

4. 次の手順に従って、`apig-demo-5` リージョンの Amazon S3 アカウントに *`{region}`* という名前のバケットを追加します。

   > **Note**
   > バケット名がグローバルに一意であることを確認します。

   a. ドロップダウンリストから [PUT] を選択し、メソッド URL (`https://`*`api-id`*`.execute-api.`*`aws-region`*`.amazonaws.com/`*`stage`*`/`*`folder-name`*) を入力します。

   b. `Content-Type` ヘッダーの値を `application/xml` に設定します。コンテンツタイプを設定する前に、既存のヘッダーを削除しなければならない場合があります。

   c. [本文] メニュー項目を選択し、リクエストの本文として次の XML フラグメントを入力します。

   ```
   <CreateBucketConfiguration>
     <LocationConstraint>{region}</LocationConstraint>
   </CreateBucketConfiguration>
   ```

   d. [送信] を選択してリクエストを送信します。成功したときは、空のペイロードを持つ `200 OK` のレスポンスを受け取ります。

5. バケットにテキストファイルを追加するには、上記の手順に従います。**`apig-demo-5`** に対して `{folder}` というバケット名を指定し、URL の中の **`Readme.txt`** に `{item}` というファイル名を指定して、ファイルの内容として **`Hello, World!`** というテキスト文字列を入力した場合 (したがってそれをリクエストペイロードにした場合)、リクエストは次のようになります。

   ```
   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!
```

すべてが正常に機能したときは、空のペイロードを持つ 200 OK レスポンスを受け取ります。

6. 先ほど Readme.txt バケットに追加した apig-demo-5 ファイルのコンテンツを取得するには、次のような GET リクエストを実行します。

```
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
```

成功したときは、200 OK というテキスト文字列のペイロードを持つ Hello, World! レスポンスを受け取ります。

7. apig-demo-5 バケット内の項目をリストするには、次のリクエストを送信します。

```
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
```

成功したときは、このリクエストを送信する前にバケットにファイルを追加した場合を除き、指定のバケット内に項目が 1 つだけ表示されている XML ペイロードを持った 200 OK レスポンスを受け取ります。

```
<?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**
>
> 画像をアップロードまたはダウンロードするには、処理したコンテンツを [バイナリに変換] に設定する必要があります。

チュートリアル: API Gateway で REST API を Amazon Kinesis のプロキシとして作成する

ここでは、AWS との統合のタイプで REST API を作成および設定して、Kinesis にアクセスする方法について説明します。

> **Note**
>
> API Gateway の API を Kinesis と統合するには、API Gateway と Kinesis の両方のサービスが利用できるリージョンを選択する必要があります。利用できるリージョンについては、「サービスエンドポイントとクォータ」を参照してください。

この図では、サンプル API を作成して、クライアントが次の操作を行うことができるようにします。

1. ユーザーが Kinesis にあるストリームを一覧表示する
2. 指定されたストリームを作成、説明、または削除する
3. 指定されたストリームからデータレコードを読み取る、または書き込む

前述のタスクを完了するため、API はさまざまなリソースでメソッドを公開し、それぞれ次のものを呼び出します。

1. Kinesis の `ListStreams` アクション
2. `CreateStream`、`DescribeStream`、または `DeleteStream` アクション
3. Kinesis の `GetRecords` または `PutRecords` (`PutRecord` を含む) アクション

具体的には、次のように API を作成します。

- API の `/streams` リソースに対する HTTP GET メソッドを公開し、そのメソッドを Kinesis の ListStreams アクションと統合して、呼び出し元のアカウントでストリームを一覧表示します。
- API の `/streams/{stream-name}` リソースに対する HTTP POST メソッドを公開し、そのメソッドを Kinesis の CreateStream アクションと統合して、呼び出し元のアカウントで指定したストリームを作成します。
- API の `/streams/{stream-name}` リソースに対する HTTP GET メソッドを公開し、そのメソッドを Kinesis の DescribeStream アクションと統合して、呼び出し元のアカウントで指定したストリームを表示します。
- API の `/streams/{stream-name}` リソースに対する HTTP DELETE メソッドを公開し、そのメソッドを Kinesis の DeleteStream アクションと統合して、呼び出し元のアカウントでストリームを削除します。
- API の `/streams/{stream-name}/record` リソースに対する HTTP PUT メソッドを公開し、そのメソッドを Kinesis の PutRecord アクションと統合します。これにより、クライアントは名前付きストリームに 1 つのデータレコードを追加できます。
- API の `/streams/{stream-name}/records` リソースに対する HTTP PUT メソッドを公開し、そのメソッドを Kinesis の PutRecords アクションと統合します。これにより、クライアントは名前付きストリームにデータレコードのリストを追加できます。
- API の `/streams/{stream-name}/records` リソースに対する HTTP GET メソッドを公開し、そのメソッドを Kinesis の GetRecords アクションと統合します。これにより、クライアントは名

前付きストリームで指定されたシャードイテレーターとともにデータレコードを一覧表示できます。シャードイテレーターは、データレコードの逐次読み取りを開始する、シャードの位置を指定します。

- API の `/streams/{stream-name}/sharditerator` リソースに対する HTTP GET メソッドを公開し、そのメソッドを Kinesis の GetShardIterator アクションと統合します。このヘルパーメソッドは、Kinesis の `ListStreams` アクションに提供する必要があります。

ここに示す手順は、Kinesis の他のアクションにも適用できます。Kinesis のアクションの一覧については、Amazon Kinesis API Reference を参照してください。

API Gateway コンソールを使用してサンプルの API を作成する代わりに、API Gateway のインポート API を使用してサンプルの API を API Gateway にインポートできます。API のインポート機能の使用方法の詳細については、「OpenAPI を使用した REST API の設定」を参照してください。

API が Kinesis にアクセスするための IAM ロールと IAM ポリシーを作成する

Kinesis のアクションを呼び出すことを API に許可するには、IAM ロールに適切な IAM ポリシーをアタッチする必要があります。

AWS のサービスプロキシの実行ロールを作成するには

1. AWS Management Console にサインインして、IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. [ロール] を選択します。
3. [ロールの作成] を選択します。
4. [信頼されたエンティティの種類を選択] で [AWS のサービス] を選択し、[API Gateway]、[API Gateway が CloudWatch Logs にログをプッシュすることを許可] の順に選択します。
5. [次へ] を選択し、さらに [次へ] を選択します。
6. [ロール名] に「**APIGatewayKinesisProxyPolicy**」と入力し、[ロールの作成] を選択します。
7. [ロール] リストで、作成したロールを選択します。ロールを検索するには、必要に応じてスクロールするか、検索バーを使用します。
8. 選択したロールの [アクセス許可を追加] タブを選択します。
9. ドロップダウンリストから [ポリシーをアタッチ] を選択します。
10. 検索バーに「**AmazonKinesisFullAccess**」と入力し、[アクセス許可を追加] を選択します。

> **Note**
>
> このチュートリアルでは、わかりやすくするために管理ポリシーを使用しますが、独自の IAM ポリシーを作成して、必要な最小限のアクセス許可を付与するのがベストプラクティスです。

11. 新しく作成したロール ARN をメモしておきます。これは後で使用します。

API を Kinesis のプロキシとして作成する

以下の手順に従って、API Gateway コンソールで API を作成します。

API を Kinesis の AWS サービスプロキシとして作成するには

1. https://console.aws.amazon.com/apigateway で API Gateway コンソールにサインインします。
2. API Gateway を初めて使用する場合は、サービスの特徴を紹介するページが表示されます。[REST API] で、[構築] を選択します。[Create Example API (サンプル API の作成)] がポップアップ表示されたら、[OK] を選択します。

   API Gateway を使用するのが初めてではない場合、[Create API] (API を作成)を選択します。[REST API] で、[構築] を選択します。
3. [新しい API] を選択します。
4. [API name (API 名)] に「**`KinesisProxy`**」と入力します。他のすべてのフィールドでは、デフォルト値をそのまま使用します。
5. (オプション) [説明] に説明を入力します。
6. [Create API] を選択します。

API が作成されると、API Gateway コンソールに API のルート (/) リソースのみを含む [Resources (リソース)] ページが表示されます。

Kinesis のストリームを一覧表示する

Kinesis では、次の REST API コールによる `ListStreams` アクションがサポートされています。

```
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>

{
   ...
}
```

上記の REST API リクエストでは、このアクションは、Action クエリパラメータで指定されます。または、代わりに、X-Amz-Target ヘッダーでこのアクションを指定することもできます。

```
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
{
   ...
}
```

このチュートリアルでは、クエリパラメータを使用してアクションを指定します。

API で Kinesis のアクションを公開するには、API のルートに /streams リソースを追加します。次に、そのリソースに対する GET メソッドを設定し、そのメソッドを Kinesis の ListStreams アクションと統合します。

以下の手順では、API Gateway コンソールを使用して Kinesis のストリームを一覧表示する方法について説明します。

API Gateway コンソールを使用して Kinesis のストリームを一覧表示するには

1. / リソースを選択し、[リソースを作成] を選択します。
2. [リソース名] に「**streams**」と入力します。
3. [CORS (Cross Origin Resource Sharing)] はオフのままにします。
4. [リソースの作成] を選択します。
5. /streams リソースを選択し、[メソッドを作成] を選択して、次の操作を行います。

a. [メソッドタイプ] には、GET を選択します。

> **Note**
>
> クライアントによって呼び出されるメソッドの HTTP 動詞は、バックエンド統合が必要な場合の HTTP 動詞とは異なる場合があります。ストリームの一覧表示は、直感的に、読み取りオペレーションであるため、ここでは GET を選択します。

b. [統合タイプ] で、[AWS のサービス] を選択します。

c. [AWS リージョン] で、Kinesis ストリームを作成した AWS リージョンを選択します。

d. [AWS のサービス] で、[Kinesis] を選択します。

e. [AWS サブドメイン] は空白のままにします。

f. [HTTP メソッド] で、[POST] を選択します。

> **Note**
>
> ここで POST を選択したのは、Kinesis では POST を使用して `ListStreams` アクションを実行する必要があるためです。

g. [アクションタイプ] で、[アクション名を使用] を選択します。

h. [アクション名] に「**`ListStreams`**」と入力します。

i. [実行ロール] に、実行ロールの ARN を入力します。

j. [コンテンツの処理] の [パススルー] はデフォルトのままにします。

k. [メソッドの作成] を選択します。

6. [統合リクエスト] タブの [統合リクエストの設定] で、[編集] を選択します。

7. [リクエスト本文のパススルー] で、[テンプレートが定義されていない場合 (推奨)] を選択します。

8. [URL リクエストヘッダーのパラメータ] を選択し、次の操作を行います。

a. [リクエストヘッダーのパラメータを追加] を選択します。

b. [名前] に**`Content-Type`**と入力します。

c. [マッピング元] として「**`'application/x-amz-json-1.1'`**」と入力します。

リクエストパラメータマッピングを使用して Content-Type ヘッダーを静的な値の 'application/x-amz-json-1.1' に設定することで、入力が特定のバージョンの JSON であることを Kinesis に知らせます。

9. [マッピングテンプレート]、[マッピングテンプレートの追加] の順に選択し、次の操作を行います。

   a. [コンテンツタイプ] に「**application/json**」と入力します。

   b. [テンプレート本文] に「**{}**」と入力します。

   c. [Save] を選択します。

   ListStreams のリクエストのペイロードは、次の形式の JSON です。

```
{
    "ExclusiveStartStreamName": "string",
    "Limit": number
}
```

   ただし、プロパティはオプションです。デフォルト値を使用するため、ここでは空の JSON ペイロードを選択しました。

10. Kinesis で ListStreams アクションを呼び出す GET メソッドを /streams リソースでテストします。

    [テスト] タブを選択します。タブを表示するには、右矢印ボタンを選択する必要がある場合があります。

    [テスト] を選択してメソッドをテストします。

    既に Kinesis で「myStream」と「yourStream」という 2 つのストリームが作成されている場合、テストが成功すると次のペイロードが含まれる 200 OK レスポンスが返されます。

```
{
     "HasMoreStreams": false,
     "StreamNames": [
         "myStream",
```

```
            "yourStream"
        ]
}
```

Kinesis でストリームを作成、表示、削除する

Kinesis でストリームを作成、表示、削除するには、それぞれ次の Kinesis の REST API へのリクエストを作成する必要があります。

```
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"
}
```

API を作成して、必要な入力をメソッドリクエストの JSON ペイロードとして受け取り、ペイロードを統合リクエストに渡すことができます。ただし、メソッドリクエストと統合リクエスト、およびメソッドレスポンスと統合レスポンスの間のデータマッピングの例を詳細に示すため、API は少し異なる方法で作成します。

これから名前を付ける GET リソースの HTTP メソッド (POST、Delete、Stream) を公開します。{stream-name} パス変数をこのストリームリソースのプレースホルダーとして使用して、これらの API のメソッドを Kinesis の DescribeStream、CreateStream、DeleteStream アクションとそれぞれ統合します。クライアントは、他の入力データをメソッドリクエストのヘッダー、クエリパラメータ、またはペイロードとして渡す必要があります。必要な統合リクエストペイロードにデータを変換するためのマッピングテンプレートが用意されています。

{stream-name} リソースを作成するには

1. [/streams] リソースを選択し、[リソースを作成] を選択します。
2. [プロキシのリソース] はオフのままにします。
3. [リソースパス] で、[/streams] を選択します。
4. [リソース名] に「**{stream-name}**」と入力します。
5. [CORS (Cross Origin Resource Sharing)] はオフのままにします。
6. [リソースの作成] を選択します。

ストリームリソースで GET メソッドを設定し、テストするには

1. [/{stream-name}] リソースを選択し、[メソッドを作成] を選択します。
2. [メソッドタイプ] には、GET を選択します。
3. [統合タイプ] で、[AWS のサービス] を選択します。
4. [AWS リージョン] で、Kinesis ストリームを作成した AWS リージョンを選択します。
5. [AWS のサービス] で、[Kinesis] を選択します。
6. [AWS サブドメイン] は空白のままにします。
7. [HTTP メソッド] で、[POST] を選択します。

8. [アクションタイプ] で、[アクション名を使用] を選択します。
9. [アクション名] に「**DescribeStream**」と入力します。
10. [実行ロール] に、実行ロールの ARN を入力します。
11. [コンテンツの処理] の [パススルー] はデフォルトのままにします。
12. [メソッドの作成] を選択します。
13. [統合リクエスト] セクションで、以下の URL リクエストヘッダーのパラメータを追加します。

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

このタスクでは、同じ手順を使用して GET /streams メソッドのリクエストパラメータマッピングを設定します。

14. 次の本文マッピングテンプレートを追加して、GET /streams/{stream-name} メソッドリクエストから POST /?Action=DescribeStream 統合リクエストにデータをマッピングします。

```
{
    "StreamName": "$input.params('stream-name')"
}
```

このマッピングテンプレートでは、Kinesis の DescribeStream アクションに必要な統合リクエストのペイロードをメソッドリクエストの stream-name パスパラメータ値から生成します。

15. Kinesis で DescribeStream アクションを呼び出す GET /stream/{stream-name} メソッドをテストするには、[テスト] タブを選択します。
16. [パス] の [stream-name] に、既存の Kinesis ストリームの名前を入力します。
17. [テスト] を選択します。テストに成功すると、200 OK レスポンスが、次のようなペイロードとともに返されます。

```
{
  "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"
  }
}
```

API をデプロイした後は、この API メソッドに対して REST リクエストを行うことができます。

```
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
```

ストリームリソースで POST メソッドを設定し、テストするには

1. [/{stream-name}] リソースを選択し、[メソッドを作成] を選択します。
2. [メソッドタイプ] では、POST を選択します。
3. [統合タイプ] で、[AWS のサービス] を選択します。
4. [AWS リージョン] で、Kinesis ストリームを作成した AWS リージョンを選択します。
5. [AWS のサービス] で、[Kinesis] を選択します。
6. [AWS サブドメイン] は空白のままにします。
7. [HTTP メソッド] で、[POST] を選択します。
8. [アクションタイプ] で、[アクション名を使用] を選択します。
9. [アクション名] に「**CreateStream**」と入力します。
10. [実行ロール] に、実行ロールの ARN を入力します。
11. [コンテンツの処理] の [パススルー] はデフォルトのままにします。
12. [メソッドの作成] を選択します。
13. [統合リクエスト] セクションで、以下の URL リクエストヘッダーのパラメータを追加します。

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

    このタスクでは、同じ手順を使用して GET /streams メソッドのリクエストパラメータマッピングを設定します。
14. 次の本文マッピングテンプレートを追加して、POST /streams/{stream-name} メソッドリクエストから POST /?Action=CreateStream 統合リクエストにデータをマッピングします。

    ```
    {
        "ShardCount": #if($input.path('$.ShardCount') == '') 5 #else
     $input.path('$.ShardCount') #end,
        "StreamName": "$input.params('stream-name')"
    }
    ```

    前述のマッピングテンプレートで、クライアントがメソッドリクエストペイロードで値を指定しなかった場合は、ShardCount を固定値 5 に設定します。
15. Kinesis で CreateStream アクションを呼び出す POST /stream/{stream-name} メソッドをテストするには、[テスト]] タブを選択します。

16. [パス] の [stream-name] に、新しい Kinesis ストリームの名前を入力します。
17. [テスト] を選択します。テストが完了すると、データなしで 200 OK レスポンスが返されます。

    API をデプロイしたら、ストリームリソースの POST メソッドに対する REST API リクエストを行って、Kinesis で `CreateStream` アクションを呼び出すこともできます。

```
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
}
```

ストリームリソースで DELETE メソッドを設定し、テストする

1. [/{stream-name}] リソースを選択し、[メソッドを作成] を選択します。
2. [メソッドタイプ] で、[DELETE] を選択します。
3. [統合タイプ] で、[AWS のサービス] を選択します。
4. [AWS リージョン] で、Kinesis ストリームを作成した AWS リージョンを選択します。
5. [AWS のサービス] で、[Kinesis] を選択します。
6. [AWS サブドメイン] は空白のままにします。
7. [HTTP メソッド] で、[POST] を選択します。
8. [アクションタイプ] で、[アクション名を使用] を選択します。
9. [アクション名] に「**`DeleteStream`**」と入力します。
10. [実行ロール] に、実行ロールの ARN を入力します。
11. [コンテンツの処理] の [パススルー] はデフォルトのままにします。
12. [メソッドの作成] を選択します。
13. [統合リクエスト] セクションで、以下の URL リクエストヘッダーのパラメータを追加します。

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

このタスクでは、同じ手順を使用して GET /streams メソッドのリクエストパラメータマッピングを設定します。

14. 次の本文マッピングテンプレートを追加して、DELETE /streams/{stream-name} メソッドリクエストから POST /?Action=DeleteStream の該当する統合リクエストにデータをマッピングします。

```
{
    "StreamName": "$input.params('stream-name')"
}
```

このマッピングテンプレートでは、DELETE /streams/{stream-name} のクライアントが指定した URL パス名値から stream-name アクションに必要な入力が生成されます。

15. Kinesis で DeleteStream アクションを呼び出す DELETE /stream/{stream-name} メソッドをテストするには、[テスト]] タブを選択します。
16. [パス] の [stream-name] に、既存の Kinesis ストリームの名前を入力します。
17. [テスト] を選択します。テストが完了すると、データなしで 200 OK レスポンスが返されます。

API をデプロイしたら、ストリームリソースの DELETE メソッドに対する以下の REST API リクエストを行って、Kinesis で DeleteStream アクションを呼び出すこともできます。

```
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

{}
```

Kinesis のストリームに対してレコードの取得や追加を行う

Kinesis でストリームを作成すると、ストリームにデータレコードを追加したり、ストリームからデータを読み取ったりすることができます。データレコードを追加するには、Kinesis で PutRecords アクションまたは PutRecord アクションを呼び出す必要があります。前者は複数のレコードを追加し、後者は 1 つのレコードをストリームに追加します。

```
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"
}
```

または

```
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"
}
```

ここで、StreamName はレコードを追加するターゲットストリームを識別します。StreamName、Data、および PartitionKey は必須の入力データです。この例では、すべてのオプション入力データにデフォルト値を使用し、メソッドリクエストへの入力ではそれらの値を明示的に指定しません。

Kinesis のデータを読み取るには、GetRecords アクションを呼び出します。

```
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
}
```

レコードを取得するソースのストリームは、必須の ShardIterator の値で指定します。シャードイテレーターは、次の Kinesis のアクションで取得します。

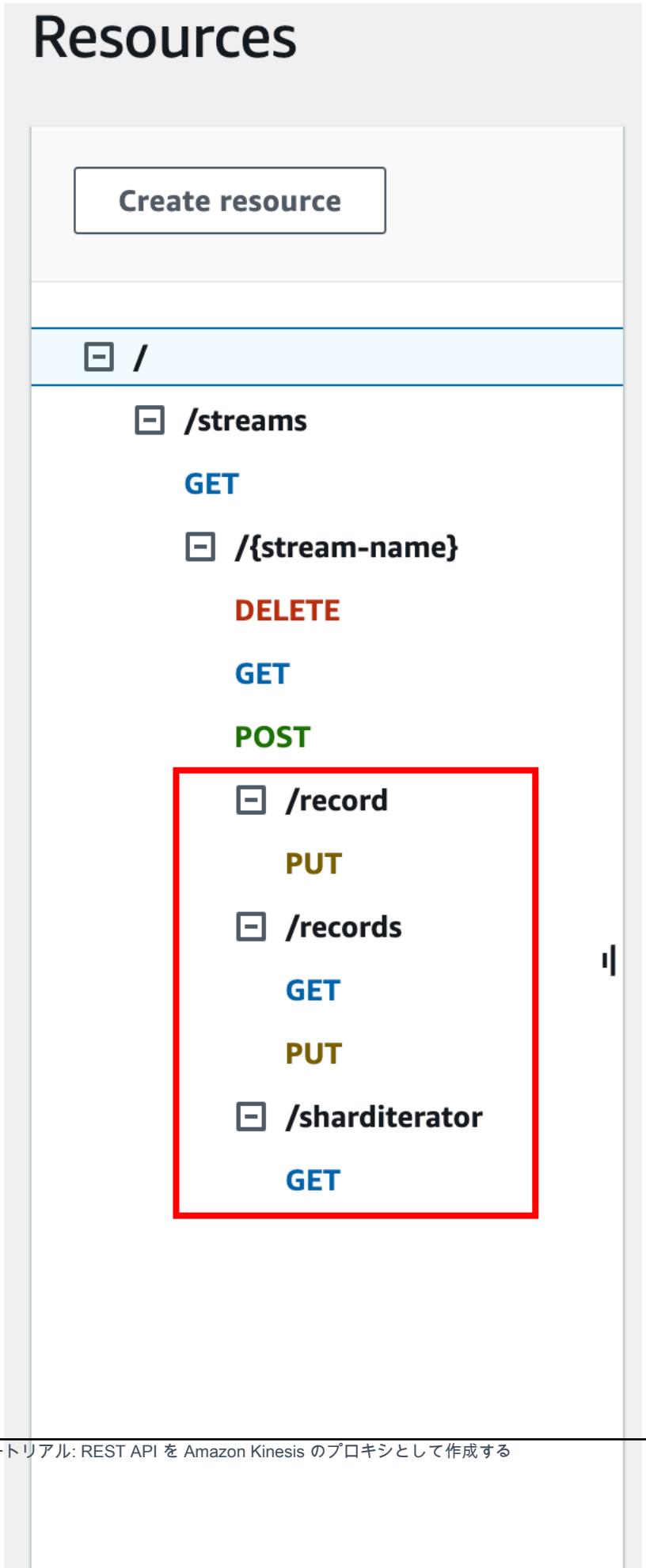
```
POST /?Action=GetShardIterator HTTP/1.1
Host: kinesis.region.domain
Authorization: AWS4-HMAC-SHA256 Credential=..., ...
...
Content-Type: application/x-amz-json-1.1
Content-Length: PayloadSizeBytes

{
    "ShardId": "string",
    "ShardIteratorType": "string",
    "StartingSequenceNumber": "string",
    "StreamName": "string"
}
```

GetRecords および PutRecords アクションでは、それぞれ GET および PUT メソッドを、名前付きストリームリソース (/records) に追加される /{stream-name} リソースで公開します。同様に、PutRecord アクションは PUT リソースで /record メソッドとして公開します。

GetRecords アクションは、ShardIterator ヘルパーアクションを呼び出して取得される GetShardIterator 値を入力として受け取るため、GET リソース (ShardIterator) で /sharditerator ヘルパーメソッドを公開します。

/record、/records、および /sharditerator リソースを作成するには

1. [/{stream-name}] リソースを選択し、[リソースを作成] を選択します。
2. [プロキシのリソース] はオフのままにします。
3. [リソースパス] で、[/{stream-name}] を選択します。
4. [リソース名] に「**record**」と入力します。
5. [CORS (Cross Origin Resource Sharing)] はオフのままにします。
6. [リソースの作成] を選択します。
7. 前の手順を繰り返して /records リソースと /sharditerator リソースを作成します。最終的な API は次のようになります。


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

次の 4 つの手順では、各メソッドの設定方法、メソッドリクエストから統合リクエストにデータをマッピングする方法、およびメソッドをテストする方法について説明します。

Kinesis で **PutRecord** を呼び出すように **PUT /streams/{stream-name}/record** メソッドを設定してテストするには

1. [/record] を選択し、[メソッドを作成] を選択します。
2. [メソッドタイプ] で、[PUT] を選択します。
3. [統合タイプ] で、[AWS のサービス] を選択します。
4. [AWS リージョン] で、Kinesis ストリームを作成した AWS リージョンを選択します。
5. [AWS のサービス] で、[Kinesis] を選択します。
6. [AWS サブドメイン] は空白のままにします。
7. [HTTP メソッド] で、[POST] を選択します。
8. [アクションタイプ] で、[アクション名を使用] を選択します。
9. [アクション名] に「**PutRecord**」と入力します。
10. [実行ロール] に、実行ロールの ARN を入力します。
11. [コンテンツの処理] の [パススルー] はデフォルトのままにします。
12. [メソッドの作成] を選択します。
13. [統合リクエスト] セクションで、以下の URL リクエストヘッダーのパラメータを追加します。

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

    このタスクでは、同じ手順を使用して `GET /streams` メソッドのリクエストパラメータマッピングを設定します。

14. 次の本文マッピングテンプレートを追加して、`PUT /streams/{stream-name}/record` メソッドリクエストから `POST /?Action=PutRecord` の該当する統合リクエストにデータをマッピングします。

    ```
    {
        "StreamName": "$input.params('stream-name')",
        "Data": "$util.base64Encode($input.json('$.Data'))",
        "PartitionKey": "$input.path('$.PartitionKey')"
    }
    ```

このマッピングテンプレートでは、メソッドリクエストのペイロードが次の形式であることを想定しています。

```
{
   "Data": "some data",
   "PartitionKey": "some key"
}
```

このデータは、次の JSON スキーマでモデル化することができます。

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "title": "PutRecord proxy single-record payload",
  "type": "object",
  "properties": {
      "Data": { "type": "string" },
      "PartitionKey": { "type": "string" }
  }
}
```

モデルを作成してこのスキーマを含め、このモデルを使用してマッピングテンプレートを生成を容易にすることができます。ただし、モデルを使用せずにマッピングテンプレートを生成することができます。

15. PUT /streams/{stream-name}/record メソッドをテストするには、stream-name パス変数を既存のストリームの名前に設定し、必要な形式のペイロードを指定して、メソッドリクエストを送信します。成功した場合の結果は 200 OK レスポンスと、次の形式のペイロードとなります。

```
{
  "SequenceNumber": "49559409944537880850133345460169886593573102115167928386",
  "ShardId": "shardId-000000000004"
}
```

Kinesis で **PUT /streams/{stream-name}/records** を呼び出すように **PutRecords** メソッドを設定してテストするには

1. [/records] リソースを選択し、[メソッドを作成] を選択します。
2. [メソッドタイプ] で、[PUT] を選択します。
3. [統合タイプ] で、[AWS のサービス] を選択します。
4. [AWS リージョン] で、Kinesis ストリームを作成した AWS リージョンを選択します。
5. [AWS のサービス] で、[Kinesis] を選択します。
6. [AWS サブドメイン] は空白のままにします。
7. [HTTP メソッド] で、[POST] を選択します。
8. [アクションタイプ] で、[アクション名を使用] を選択します。
9. [アクション名] に「**PutRecords**」と入力します。
10. [実行ロール] に、実行ロールの ARN を入力します。
11. [コンテンツの処理] の [パススルー] はデフォルトのままにします。
12. [メソッドの作成] を選択します。
13. [統合リクエスト] セクションで、以下の URL リクエストヘッダーのパラメータを追加します。

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

    このタスクでは、同じ手順を使用して GET /streams メソッドのリクエストパラメータマッピングを設定します。
14. 次の本文マッピングテンプレートを追加して、PUT /streams/{stream-name}/records メソッドリクエストから、POST /?Action=PutRecords の対応する統合リクエストにデータをマッピングします。

    ```
    {
        "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
        ]
    }
    ```

このマッピングテンプレートでは、メソッドリクエストのペイロードが次の JSON スキーマでモデル化できることを想定しています。

```
{
  "$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" }
        }
      }
    }
  }
}
```

モデルを作成してこのスキーマを含め、このモデルを使用してマッピングテンプレートを生成を容易にすることができます。ただし、モデルを使用せずにマッピングテンプレートを生成することができます。

このチュートリアルでは、API デベロッパーがバックエンドデータ形式をクライアントに開示するか、クライアントから見えないようにするかを選択できることを示すため、わずかに異なる 2 つのペイロード形式を使用しました。1 つは PUT /streams/{stream-name}/records メソッド (上記) 用の形式、もう 1 つは PUT /streams/{stream-name}/record メソッド (前の手順) 用の形式です。実稼働環境では、両方の形式の一貫性を保ってください。

15. PUT /streams/{stream-name}/records メソッドをテストするには、stream-name パス変数を既存のストリームに設定し、前に示したように次のペイロードを指定して、メソッドリクエストを送信します。

```
{
    "records": [
        {
            "data": "some data",
```

```
            "partition-key": "some key"
        },
        {
            "data": "some other data",
            "partition-key": "some key"
        }
    ]
}
```

成功した場合の結果は 200 OK レスポンスと、以下の出力例のようになります。

```
{
  "FailedRecordCount": 0,
  "Records": [
    {
      "SequenceNumber": "49559409944537880850133345460167468741933742152373764162",
      "ShardId": "shardId-000000000004"
    },
    {
      "SequenceNumber": "49559409944537880850133345460168677667753356781548470338",
      "ShardId": "shardId-000000000004"
    }
  ]
}
```

Kinesis で **GET /streams/{stream-name}/sharditerator** を呼び出すように **GetShardIterator** メソッドを設定してテストするには

GET /streams/{stream-name}/sharditerator メソッドは、GET /streams/{stream-name}/records メソッドを呼び出す前に必須のシャードイテレーターを取得するためのヘルパーメソッドです。

1. [/sharditerator] リソースを選択し、[メソッドを作成] を選択します。
2. [メソッドタイプ] には、GET を選択します。
3. [統合タイプ] で、[AWS のサービス] を選択します。
4. [AWS リージョン] で、Kinesis ストリームを作成した AWS リージョンを選択します。
5. [AWS のサービス] で、[Kinesis] を選択します。
6. [AWS サブドメイン] は空白のままにします。

7. [HTTP メソッド] で、[POST] を選択します。
8. [アクションタイプ] で、[アクション名を使用] を選択します。
9. [アクション名] に「**GetShardIterator**」と入力します。
10. [実行ロール] に、実行ロールの ARN を入力します。
11. [コンテンツの処理] の [パススルー] はデフォルトのままにします。
12. [URL クエリ文字列パラメータ] を選択します。

    GetShardIterator アクションでは、ShardId 値を入力する必要があります。クライアントが指定した ShardId 値を渡すには、次のステップに示すように、shard-id クエリパラメータをメソッドリクエストに追加します。
13. [クエリ文字列の追加] を選択します。
14. [名前] に**shard-id**と入力します。
15. [必須] と [キャッシュ] はオフのままにしておきます。
16. [メソッドの作成] を選択します。
17. [統合リクエスト] セクションで、次のマッピングテンプレートを追加して、メソッドリクエストの shard-id パラメータと stream-name パラメータの GetShardIterator アクションに必要な入力 (ShardId および StreamName) を生成します。また、マッピングテンプレートでも、ShardIteratorType は、デフォルトとして TRIM_HORIZON に設定されます。

    ```
    {
        "ShardId": "$input.params('shard-id')",
        "ShardIteratorType": "TRIM_HORIZON",
        "StreamName": "$input.params('stream-name')"
    }
    ```
18. API Gateway コンソールの [Test (テスト)] オプションを使用して、既存のストリームの名前を stream-name の [Path (パス)] 変数値に入力し、shard-id の [Query string (クエリ文字列)] を既存の ShardId 値 (例: shard-000000000004) に設定して、[Test (テスト)] を選択します。

    成功のレスポンスペイロードは以下の出力例のようになります。

    ```
    {
      "ShardIterator": "AAAAAAAAAAFYVN3VlFy..."
    }
    ```

    ShardIterator の値をメモしておきます。これは、ストリームからレコードを取得するために必要です。

Kinesis で **GET /streams/{stream-name}/records** アクションを呼び出すように **GetRecords** メソッドを設定してテストするには

1. [/records] リソースを選択し、[メソッドを作成] を選択します。
2. [メソッドタイプ] には、GET を選択します。
3. [統合タイプ] で、[AWS のサービス] を選択します。
4. [AWS リージョン] で、Kinesis ストリームを作成した AWS リージョンを選択します。
5. [AWS のサービス] で、[Kinesis] を選択します。
6. [AWS サブドメイン] は空白のままにします。
7. [HTTP メソッド] で、[POST] を選択します。
8. [アクションタイプ] で、[アクション名を使用] を選択します。
9. [アクション名] に「**GetRecords**」と入力します。
10. [実行ロール] に、実行ロールの ARN を入力します。
11. [コンテンツの処理] の [パススルー] はデフォルトのままにします。
12. [HTTP リクエストヘッダー] を選択します。

    GetRecords オペレーションでは、ShardIterator の値の入力が必要です。クライアントが指定した ShardIterator 値を渡すには、Shard-Iterator ヘッダーパラメータをメソッドリクエストに追加します。
13. [ヘッダーの追加] を選択します。
14. [名前] に**Shard-Iterator**と入力します。
15. [必須] と [キャッシュ] はオフのままにしておきます。
16. [メソッドの作成] を選択します。
17. [統合リクエスト] セクションで、次の本文マッピングテンプレートを追加して、Shard-Iterator ヘッダーパラメータ値を Kinesis の GetRecords アクションの JSON ペイロードの ShardIterator プロパティ値にマッピングします。

    ```
    {
        "ShardIterator": "$input.params('Shard-Iterator')"
    }
    ```
18. API Gateway コンソールの [テスト] オプションを使用して、既存のストリームの名前を stream-name の [パス] 変数値として入力し、Shard-Iterator の [ヘッダー] を GET /streams/{stream-name}/sharditerator メソッド (上記) のテスト実行で取得した ShardIterator 値に設定して、[テスト] を選択します。

成功のレスポンスペイロードは以下の出力例のようになります。

```
{
  "MillisBehindLatest": 0,
  "NextShardIterator": "AAAAAAAAAAF...",
  "Records": [ ... ]
}
```

Kinesis のプロキシとしてのサンプル API の OpenAPI 定義

以下の OpenAPI 定義は、このチュートリアルで使用されている Kinesis のプロキシとしてのサンプル API のものです。

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"
              }
            }
          }
        }
      }
    }
  }
}
```

チュートリアル: AWS SDKまたは AWS CLI を使用してエッジ最適化 API を作成する

次のチュートリアルは、`GET /pets` メソッドと `GET /pets/{petId}` メソッドをサポートする PetStore API を作成する方法を示しています。各メソッドは HTTP エンドポイントと統合されています。このチュートリアルは、AWS SDK for JavaScript、SDK for Python (Boto3)、または AWS CLI を使用して実行できます。API をセットアップするには、以下の関数またはコマンドを使用します。

JavaScript v3

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

Python

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

AWS CLI

- create-rest-api
- create-resource
- put-method
- put-method-response

- put-integration
- put-integration-response
- create-deployment

AWS SDK for JavaScript v3 の詳細については、「AWS SDK for JavaScript とは」を参照してください。SDK for Python (Boto3) の詳細については、「AWS SDK for Python (Boto3)」を参照してください。AWS CLI の詳細については、「AWS CLI とは」を参照してください。

エッジ最適化 PetStore API をセットアップする

このチュートリアルのコマンド例では、API ID やリソース ID などの値 ID にプレースホルダー値を使用します。チュートリアルを完了したら、これらの値を独自の値に置き換えてください。

AWS SDK を使用してエッジ最適化 PetStore API をセットアップするには

1. 次の例を使用して `RestApi` エンティティを作成します。

   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)
   }
   })();
   ```

   呼び出しが成功すると、API ID と API のルートリソース ID が次のような出力で返されます。

```
{
  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)
```

呼び出しが成功すると、API ID と API のルートリソース ID が次のような出力で返されます。

```
{'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
```

このコマンドの出力は次のとおりです。

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

作成した API の API ID は abcd1234、ルートリソース ID は efg567 です。これらの値は API のセットアップで使用します。

2. 次に、ルート下に子リソースを追加し、RootResourceId を parentId プロパティ値として指定します。次の例を使用して、API の /pets リソースを作成します。

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)
}
})();
```

呼び出しが成功すると、リソースに関する情報が次のような出力で返されます。

```
{
    "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)
```

呼び出しが成功すると、リソースに関する情報が次のような出力で返されます。

```
{'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
```

このコマンドの出力は次のとおりです。

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

作成した /pets リソースのリソース ID は aaa111 です。この値は API のセットアップで使用します。

3. 次に、/pets リソースの下に子リソースを追加します。このリソース /{petId} には、{petId} のパスパラメータがあります。パスパートをパスパラメータにするには、中括弧のペア { } で囲みます。次の例を使用して、API の /pets/{petId} リソースを作成します。

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)
}
})();
```

呼び出しが成功すると、リソースに関する情報が次のような出力で返されます。

```
{
    "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)
```

呼び出しが成功すると、リソースに関する情報が次のような出力で返されます。

```
{'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}'
```

成功すると、このコマンドは以下のようなレスポンスを返します。

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

作成した /pets/{petId} リソースのリソース ID は bbb222 です。この値は API のセットアップで使用します。

4. 次の 2 つのステップでは、HTTP メソッドをリソースに追加します。このチュートリアルでは、authorization-type を NONE に設定して、オープンアクセスを持つようにメソッドを設定します。認証されたユーザーにのみ、メソッドの呼び出しを許可するには、IAM ロールおよびポリシー、Lambda オーソライザー (以前のカスタムオーソライザー)、または Amazon Cognito ユーザープールを使用します。詳細については、「the section called “アクセスコントロール”」を参照してください。

   次の例では、GET HTTP メソッドを /pets リソースに追加します。

   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)
}
})();
```

呼び出しに成功すると、次の出力が返されます。

```
{
    "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)
```

呼び出しに成功すると、次の出力が返されます。

```
{'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
```

正常に実行された場合、このコマンドの出力は以下のようになります。

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

5. 次の例では、GET HTTP メソッドを /pets/{petId} リソースに追加し、クライアントが指定した petId 値をバックエンドに渡すように requestParameters プロパティを設定します。

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)
```

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

呼び出しに成功すると、次の出力が返されます。

```
{
    "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)
```

呼び出しに成功すると、次の出力が返されます。

```
{'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
```

正常に実行された場合、このコマンドの出力は以下のようになります。

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

6. 次の例を使用して、GET /pets メソッドの 200 OK メソッドレスポンスを追加します。

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)
}
})();
```

呼び出しに成功すると、次の出力が返されます。

```
{
    "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)
```

呼び出しに成功すると、次の出力が返されます。

```
{'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
```

このコマンドの出力は次のとおりです。

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

7. 次の例を使用して、GET /pets/{petId} メソッドの 200 OK メソッドレスポンスを追加します。

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)
}
})();
```

呼び出しに成功すると、次の出力が返されます。

```
{
    "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)
```

呼び出しに成功すると、次の出力が返されます。

```
{'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
```

このコマンドの出力は次のとおりです。

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

8. 次の例では、GET /pets メソッドと HTTP エンドポイントの統合を設定します。HTTP エンドポイントは http://petstore-demo-endpoint.execute-api.com/petstore/pets です。

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)
}
})();
```

呼び出しに成功すると、次の出力が返されます。

```
{
    "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)
```

呼び出しに成功すると、次の出力が返されます。

```
{'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
```

このコマンドの出力は次のとおりです。

```
{
    "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. 次の例では、GET /pets/{petId} メソッドと HTTP エンドポイントの統合を設定します。HTTP エンドポイントは http://petstore-demo-endpoint.execute-api.com/petstore/pets/{id} です。このステップでは、パスパラメータ petId を id の統合エンドポイントパスパラメータにマッピングします。

   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)
   }
   })();
   ```

   呼び出しに成功すると、次の出力が返されます。

   ```
   {
   ```

```
    "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)
```

呼び出しに成功すると、次の出力が返されます。

```
{'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
```

このコマンドの出力は次のとおりです。

```
{
    "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. 次の例では、GET /pets 統合の統合レスポンスを追加します。

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)
}
})();
```

呼び出しに成功すると、次の出力が返されます。

```
{
    "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)
```

呼び出しに成功すると、次の出力が返されます。

```
{'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
```

このコマンドの出力は次のとおりです。

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

11. 次の例では、GET /pets/{petId} 統合の統合レスポンスを追加します。

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)
}
})();
```

呼び出しに成功すると、次の出力が返されます。

```
{
    "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)
```

呼び出しに成功すると、次の出力が返されます。

```
{'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
```

このコマンドの出力は次のとおりです。

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

統合レスポンスを作成すると、API は PetStore ウェブサイトで利用可能なペットをクエリし、指定した識別子のペットを個別に表示できます。顧客が API を呼び出せるようにするには、事前に API をデプロイする必要があります。API は、デプロイする前にテストすることをお勧めします。

12. 次の例では、GET /pets メソッドをテストします。

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. 次の例では、petId として 3 を使用して GET /pets/{petId} メソッドをテストします。

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'
```

API のテストに成功したら、ステージにデプロイできます。

14. 次の例では、 API を `test` という名前のステージにデプロイします。API をステージにデプロイすると、API の呼び出し元は API を呼び出すことができます。

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'
```

このコマンドの出力は次のとおりです。

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

これで、API は顧客から呼び出せるようになりました。この API をテストするには、`https://abcd1234.execute-api.us-west-2.amazonaws.com/test/pets` URL をブラウザに入力し、`abcd1234` を API の識別子に置き換えます。

AWS SDK または AWS CLI を使用して API を作成または更新する方法の他の例については、「AWS SDK を使用する API Gateway のアクション」を参照してください。

API のセットアップを自動化する

API をステップバイステップで作成する代わりに、OpenAPI、AWS CloudFormation、または Terraform を使用して API を作成することで、AWS リソースの作成とクリーンアップを自動化できます。

OpenAPI 3.0 定義

OpenAPI 定義を API Gateway にインポートできます。詳細については、「the section called "OpenAPI"」を参照してください。

```
{
  "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 テンプレート

AWS CloudFormation テンプレートをデプロイするには、「AWS CloudFormation コンソールでのスタックの作成」を参照してください。

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

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

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

Terraform の設定

Terraform の詳細については、「Terraform」を参照してください。

```
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"
}
```

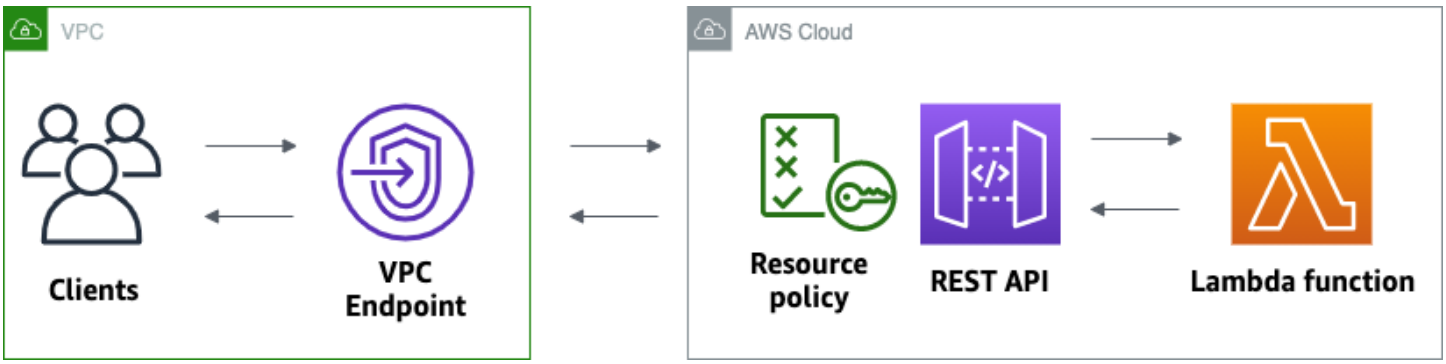
```
resource "aws_api_gateway_integration" "petsIntegration" {
  rest_api_id = aws_api_gateway_rest_api.Api.id
  resource_id = aws_api_gateway_resource.petsResource.id
  http_method = aws_api_gateway_method.petsMethodGet.http_method
  type        = "HTTP"

  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
}
```

チュートリアル: プライベート REST API を構築する

このチュートリアルでは、プライベート REST API を作成します。クライアントは Amazon VPC 内からのみ API にアクセスできます。API は、一般的なセキュリティ要件であるパブリックインターネットから分離されています。

このチュートリアルの完了には約 30 分かかります。まず、AWS CloudFormation テンプレートを使用して Amazon VPC、VPC エンドポイント、AWS Lambda 関数を作成し、API のテストに使用する Amazon EC2 インスタンスを起動します。次に、AWS Management Console を使用してプライベート API を作成し、VPC エンドポイントからのアクセスのみを許可するリソースポリシーをアタッチします。最後に、API をテストします。



このチュートリアルを完了するには、AWS アカウントと、コンソールへのアクセス権がある AWS Identity and Access Management ユーザーが必要です。詳細については、「前提条件」を参照してください。

このチュートリアルでは、 を使用しますAWS Management Console この API とすべての関連リソースを作成する AWS CloudFormation テンプレートについては、template.yaml を参照してください。

トピック

- ステップ 1: 依存関係を作成する
- ステップ 2: プライベート API を作成する
- ステップ 3: メソッドと統合を作成する
- ステップ 4: リソースポリシーをアタッチする
- ステップ 5: API をデプロイする
- ステップ 6: API がパブリックにアクセスできないことを確認する
- ステップ 7: VPC のインスタンスに接続し、API を呼び出す
- ステップ 8: クリーンアップする
- 次のステップ: AWS CloudFormation を使用して自動化する

ステップ 1: 依存関係を作成する

この AWS CloudFormation テンプレートをダウンロードして解凍します。テンプレートを使用して、プライベート API のすべての依存関係を作成します。これには、Amazon VPC、VPC エンドポイント、API のバックエンドとして機能する Lambda 関数が含まれます。プライベート API は後で作成します。

AWS CloudFormation スタックを作成するには

1. https://console.aws.amazon.com/cloudformation で AWS CloudFormation コンソール を開きます。
2. [スタックの作成] を選択し、[With new resources (standard) 新しいリソースを使用 (標準)] を選択します。
3. [Specify template (テンプレートの指定)] で、[Upload a template file (テンプレートファイルのアップロード)] を選択します。
4. ダウンロードしたテンプレートを選択します。
5. [Next (次へ)] を選択します。
6. [Stack name] (スタックの名前) で、**`private-api-tutorial`** と入力し、[Next] (次へ) を選択します。
7. [Configure stack options] (スタックオプションの設定) で、[Next] (次へ) を選択します。
8. [Capabilities] (機能) で、AWS CloudFormation がアカウントに IAM リソースを作成できることを承認します。
9. 送信 を選択します。

AWS CloudFormation は API の依存関係をプロビジョニングします。これには数分かかる場合があります。AWS CloudFormation スタックのステータスが CREATE_COMPLETE の場合は、[Outputs] (出力) を選択します。VPC エンドポイント ID を書き留めます。このチュートリアルの後の手順で必要になります。

ステップ 2: プライベート API を作成する

VPC 内のクライアントのみがアクセスできるようにプライベート API を作成します。

プライベート API を作成するには

1. https://console.aws.amazon.com/apigateway で API Gateway コンソールにサインインします。
2. [Create API] (API を作成) を選択し、[REST API] で [Build] (構築) を選択します。
3. [API 名] に「**`private-api-tutorial`**」と入力します。
4. [API エンドポイントタイプ] で、[プライベート] を選択します。
5. [VPC エンドポイント ID] に、AWS CloudFormation スタックの [出力] からの VPC エンドポイント ID を入力します。
6. API の作成 を選択します。

ステップ 3: メソッドと統合を作成する

API への GET リクエストを処理する GET メソッドと Lambda 統合を作成します。クライアントが API を呼び出すと、API Gateway はステップ 1 で作成した Lambda 関数にリクエストを送信し、クライアントに応答を返します。

メソッドと統合を作成するには

1. https://console.aws.amazon.com/apigateway で API Gateway コンソールにサインインします。
2. API を選択します。
3. / リソースを選択し、[メソッドを作成] を選択します。
4. [メソッドタイプ] で、[GET] を選択します。
5. [統合タイプ] で、[Lambda 関数] を選択します。
6. [Lambda プロキシ統合]を有効にします。Lambda プロキシ統合では、API Gateway は定義された構造を使用して Lambda にイベントを送信し、Lambda 関数からの応答を HTTP 応答に変換します。
7. [Lambda function] (Lambda 関数) には、ステップ 1 で AWS CloudFormation テンプレートを使用して作成した関数を選択します。関数の名前は **`private-api-tutorial`** で始まります。
8. [メソッドの作成] を選択します。

ステップ 4: リソースポリシーをアタッチする

クライアントが VPC エンドポイントを介してのみ API を呼び出すことを許可するリソースポリシーを API にアタッチします。API へのアクセスをさらに制限するには、VPC エンドポイントの VPC エンドポイントポリシーを設定することもできますが、このチュートリアルでは不要です。

リソースポリシーをアタッチするには

1. https://console.aws.amazon.com/apigateway で API Gateway コンソールにサインインします。
2. API を選択します。
3. [リソースポリシー]、[ポリシーを作成] の順に選択します。
4. 以下のポリシーを入力します。*`vpceID`* を、AWS CloudFormation スタックの [Outputs] (出力) からの VPC エンドポイント ID に置き換えます。

```
{
```

```
    "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. [Save changes] (変更の保存) をクリックします。

ステップ 5: API をデプロイする

次に、API をデプロイして、Amazon VPC のクライアントが使用できるようにします。

API をデプロイするには

1. https://console.aws.amazon.com/apigateway で API Gateway コンソールにサインインします。
2. API を選択します。
3. [API のデプロイ] を選択します。
4. [ステージ] で [新規ステージ] を選択します。
5. [Stage name (ステージ名)] に **`test`** と入力します。
6. (オプション) [説明] に説明を入力します。
7. [デプロイ] を選択します。

これで、API をテストする準備ができました。

ステップ 6: API がパブリックにアクセスできないことを確認する

curl を使用して、Amazon VPC の外部から API を呼び出すことができないことを確認します。

API をテストするために

1. API Gateway コンソール (https://console.aws.amazon.com/apigateway) にサインインします。
2. API を選択します。
3. メインナビゲーションペインで、[ステージ]、[テスト] ステージの順に選択します。
4. [ステージの詳細] で、コピーアイコンを選択して API の呼び出し URL をコピーします。URL は `https://abcdef123.execute-api.us-west-2.amazonaws.com/test` のようになります。ステップ 1 で作成した VPC エンドポイントではプライベート DNS が有効になっているため、提供された URL を使用して API を呼び出すことができます。
5. curl を使用して、VPC の外部からの API の呼び出しを試みる

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

   Curl は、API のエンドポイントを解決できないことを示します。別の応答が返された場合は、ステップ 2 に戻り、API のエンドポイントタイプに [Private] (プライベート) を選択します。

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

次に、VPC 内の Amazon EC2 インスタンスに接続して API を呼び出します。

ステップ 7: VPC のインスタンスに接続し、API を呼び出す

次に、Amazon VPC 内から API をテストします。プライベート API にアクセスするには、VPC 内の Amazon EC2 インスタンスに接続し、curl を使用して API を呼び出します。ブラウザでインスタンスに接続するには、Systems Manager の Session Manager を使用します。

API をテストするには

1. Amazon EC2 コンソール (https://console.aws.amazon.com/ec2/) を開きます。
2. [Instances] を選択します。
3. ステップ 1 で AWS CloudFormation テンプレートを使用して作成した private-api-tutorial という名前のインスタンスを選択します。

4. [Connect] (接続) を選択し、[Session Manager] を選択します。
5. [Connect] (接続) を選択して、インスタンスへのブラウザベースのセッションを起動します。
6. Session Manager セッションで、curl を使用して API を呼び出します。Amazon VPC でインスタンスを使用しているため、API を呼び出すことができます。

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

応答 `Hello from Lambda!` が得られたことを確認します。

Amazon VPC 内からのみアクセスできる API を正常に作成し、正常に動作することを確認しました。

ステップ 8: クリーンアップする

不要なコストを回避するには、このチュートリアルで作成したリソースを削除します。次の手順は、REST API と AWS CloudFormation スタックを削除します。

REST API を削除するには

1. https://console.aws.amazon.com/apigateway で API Gateway コンソールにサインインします。
2. [API] ページで API を選択します。[API アクション]、[API を削除] の順に選択し、選択を確定します。

AWS CloudFormation スタックを削除するには

1. AWS CloudFormation コンソール (https://console.aws.amazon.com/cloudformation) を開きます。

2. AWS CloudFormation スタックを選択します。
3. [Delete] (削除) を選択し、選択を確定します。

次のステップ: AWS CloudFormation を使用して自動化する

このチュートリアルで使用するすべての AWS リソースの作成とクリーンアップを自動化できます。AWS CloudFormation テンプレートの完全な例については、template.yaml を参照してください。

Amazon API Gateway HTTP API チュートリアル

以下のチュートリアルでは API Gateway HTTP API の学習に役立つ実践的な演習を提供します。

トピック

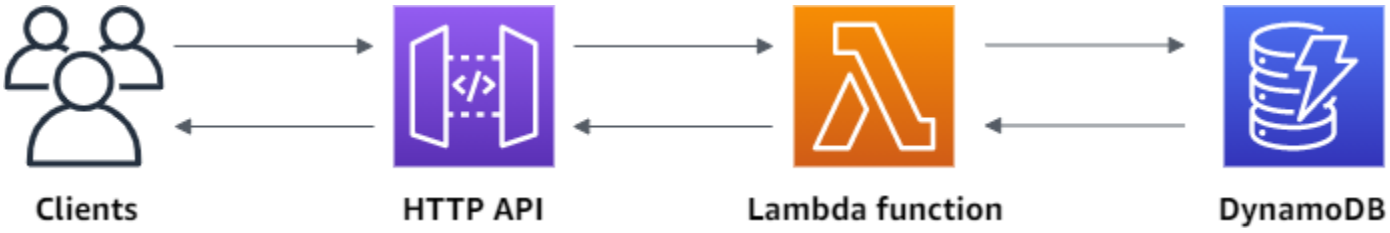
- チュートリアル: Lambda と DynamoDB を使用した CRUD API の構築
- チュートリアル: Amazon ECS サービスへのプライベート統合を使用した HTTP API の構築

チュートリアル: Lambda と DynamoDB を使用した CRUD API の構築

このチュートリアルでは、DynamoDB テーブルから項目を作成、読み取り、更新、削除するサーバーレス API を作成します。DynamoDB は、高速で予測可能なパフォーマンスとシームレスな拡張性を特長とするフルマネージド NoSQL データベースサービスです。このチュートリアルの所要時間は約 30 分で、AWS の無料利用枠内で実行できます。

まず、DynamoDB コンソールを使用して DynamoDB テーブルを作成します。次に、AWS Lambda コンソールを使用して Lambda 関数を作成します。次に、API Gateway コンソールを使用して HTTP API を作成します。最後に、API をテストします。

HTTP API を呼び出すと、API Gateway はリクエストを Lambda 関数にルーティングします。Lambda 関数は DynamoDB と対話し、API ゲートウェイにレスポンスを返します。それから API Gateway はレスポンスを返します。



このエクササイズを完了するには、AWS アカウントと、コンソールへのアクセス権がある AWS Identity and Access Management ユーザーが必要です。詳細については、「前提条件」を参照してください。

このチュートリアルでは、 を使用しますAWS Management Console この API とすべての関連リソースを作成する AWS SAM テンプレートについては、`template.yaml` を参照してください。

トピック

ステップ 1: DynamoDB テーブルを作成する

DynamoDB テーブルを使用して API のデータを保存します。

各項目には一意の ID があり、これをテーブルのパーティションキーとして使用します。

DynamoDB テーブルを作成するには

1. https://console.aws.amazon.com/dynamodb/ で DynamoDB コンソールを開きます。
2. [Create table] を選択します。
3. [テーブル名] に「**`http-crud-tutorial-items`**」と入力します。
4. [パーティションキー] に「**`id`**」と入力します。
5. [Create table (テーブルの作成)] を選択します。

ステップ 2: Lambda 関数を作成する

API のバックエンドに Lambda 関数を作成します。この Lambda 関数は、DynamoDB から項目を作成、読み取り、更新、および削除します。この関数は、API ゲートウェイのイベントを使用して

DynamoDB との対話方法を決定します。わかりやすくするために、このチュートリアルでは 1 つの Lambda 関数を使用しますが、ルートごとに個別の関数を作成するのがベストプラクティスです。

Lambda 関数を作成するには

1. Lambda コンソール (https://console.aws.amazon.com/lambda/) にサインインします。
2. [関数の作成] を選択します。
3. [関数名] に「**http-crud-tutorial-function**」と入力します。
4. [ランタイム] で、サポートされている最新の Node.js または Python ランタイムのいずれかを選択します。
5. [アクセス許可] で [デフォルトの実行ロールの変更] を選択します。
6. [Create a new role from AWS policy templates] (AWS ポリシーテンプレートから新しいロールを作成) を選択します。
7. [ロール名] に「**http-crud-tutorial-role**」と入力します。
8. [ポリシーテンプレート] では、[**Simple microservice permissions**] を選択します。このポリシーは、Lambda 関数に DynamoDB と対話するためのアクセス許可を付与します。

   > **Note**
   >
   > このチュートリアルでは、わかりやすくするために管理ポリシーを使用しますが、独自の IAM ポリシーを作成して、必要な最小限のアクセス許可を付与するのがベストプラクティスです。

9. [関数の作成] を選択します。
10. コンソールのコードエディタで Lambda 関数を開き、その内容を次のコードに置き換えます。[デプロイ] を選択して、関数を更新します。

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
```

ステップ 3: HTTP API を作成する

HTTP API は、Lambda 関数の HTTP エンドポイントを提供します。このステップでは、空の API を作成します。次のステップでは、API と Lambda 関数を接続するようにルートと統合を設定します。

HTTP API を作成するには

1. https://console.aws.amazon.com/apigateway で API Gateway コンソールにサインインします。
2. [API を作成] を選択し、[HTTP API] で [構築] を選択します。
3. [API 名] に「**http-crud-tutorial-api**」と入力します。
4. [Next (次へ)] を選択します。
5. [ルートの設定] で、[次へ] を選択してルートの作成をスキップします。ルートは後で作成します。
6. API Gateway によって作成されるステージを確認し、[次へ] をクリックします。
7. [Create] を選択します。

ステップ 4: ルートを作成する

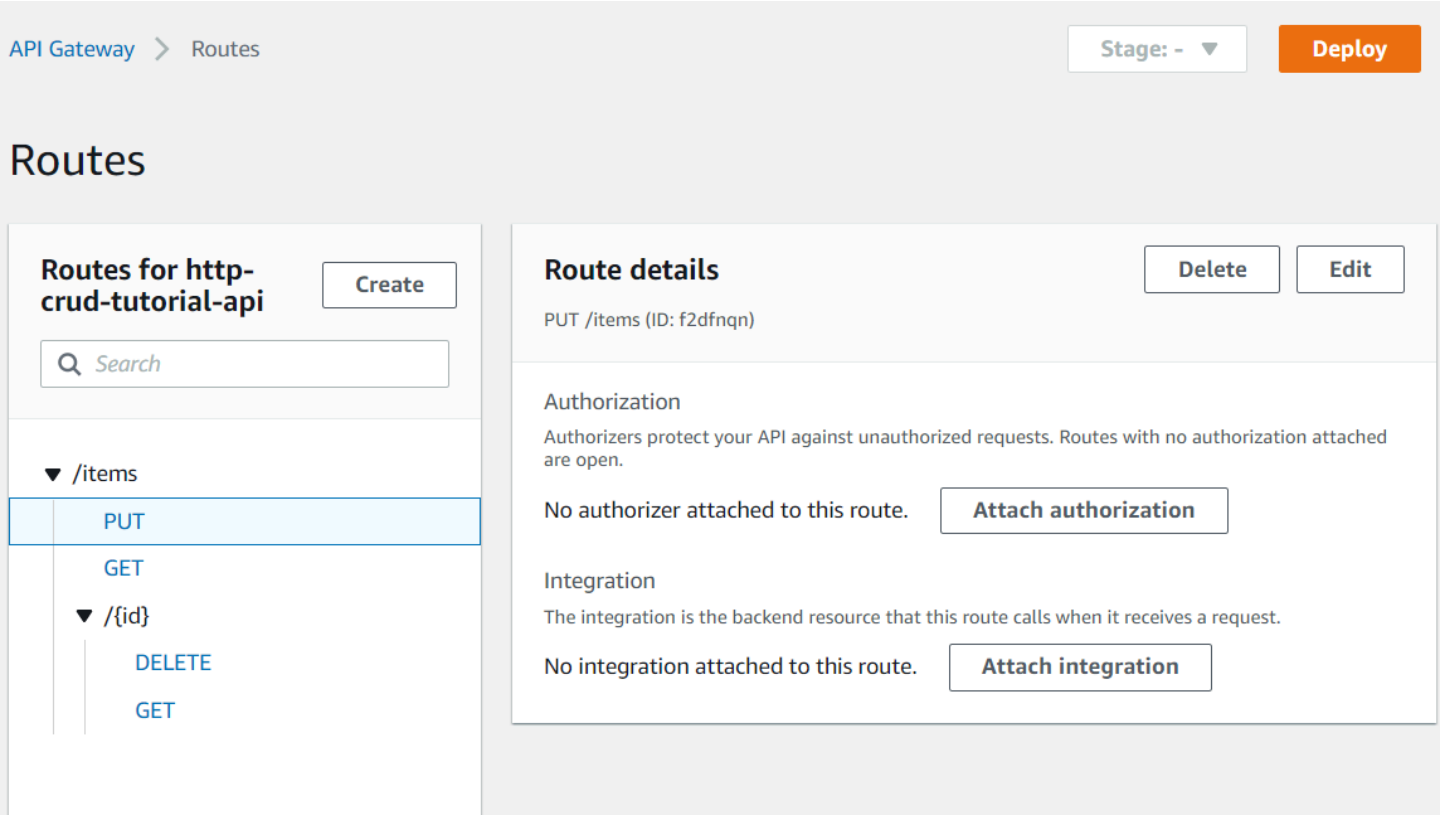
ルートは、着信 API リクエストをバックエンドリソースに送信する方法です。ルートは、HTTP メソッドとリソースパスという 2 つの部分で構成されます (例: `GET /items`)。この例の API では、次の 4 つのルートを作成します。

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

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

ルートを作成するには

1. https://console.aws.amazon.com/apigateway で API Gateway コンソールにサインインします。
2. API を選択します。
3. [ルート] をクリックします。
4. [Create] を選択します。
5. [Method] (メソッド) で、[**GET**] を選択します。
6. パスには、「**`/items/{id}`**」と入力します。パスの末尾にある `{id}` は、クライアントがリクエストを行うときに API Gateway がリクエストパスから取得するパスパラメータです。
7. [Create] を選択します。
8. `GET /items`、`DELETE /items/{id}`、および `PUT /items` について、ステップ 4～7 を繰り返します 。



ステップ 5: 統合を作成する

ルートをバックエンドリソースに接続するための統合を作成します。この API の例では、すべてのルートに使用する 1 つの Lambda 統合を作成します。

統合を作成するには

1. https://console.aws.amazon.com/apigateway で API Gateway コンソールにサインインします。
2. API を選択します。
3. [統合] を選択します。
4. [統合を管理] を選択し、[作成] をクリックします。
5. [この統合をルートにアタッチする] はスキップします。これは、後の手順で完了します。
6. [統合タイプ] で、[Lambda 関数] を選択します。
7. [Lambda 関数] に「**`http-crud-tutorial-function`**」と入力します。
8. [Create] を選択します。

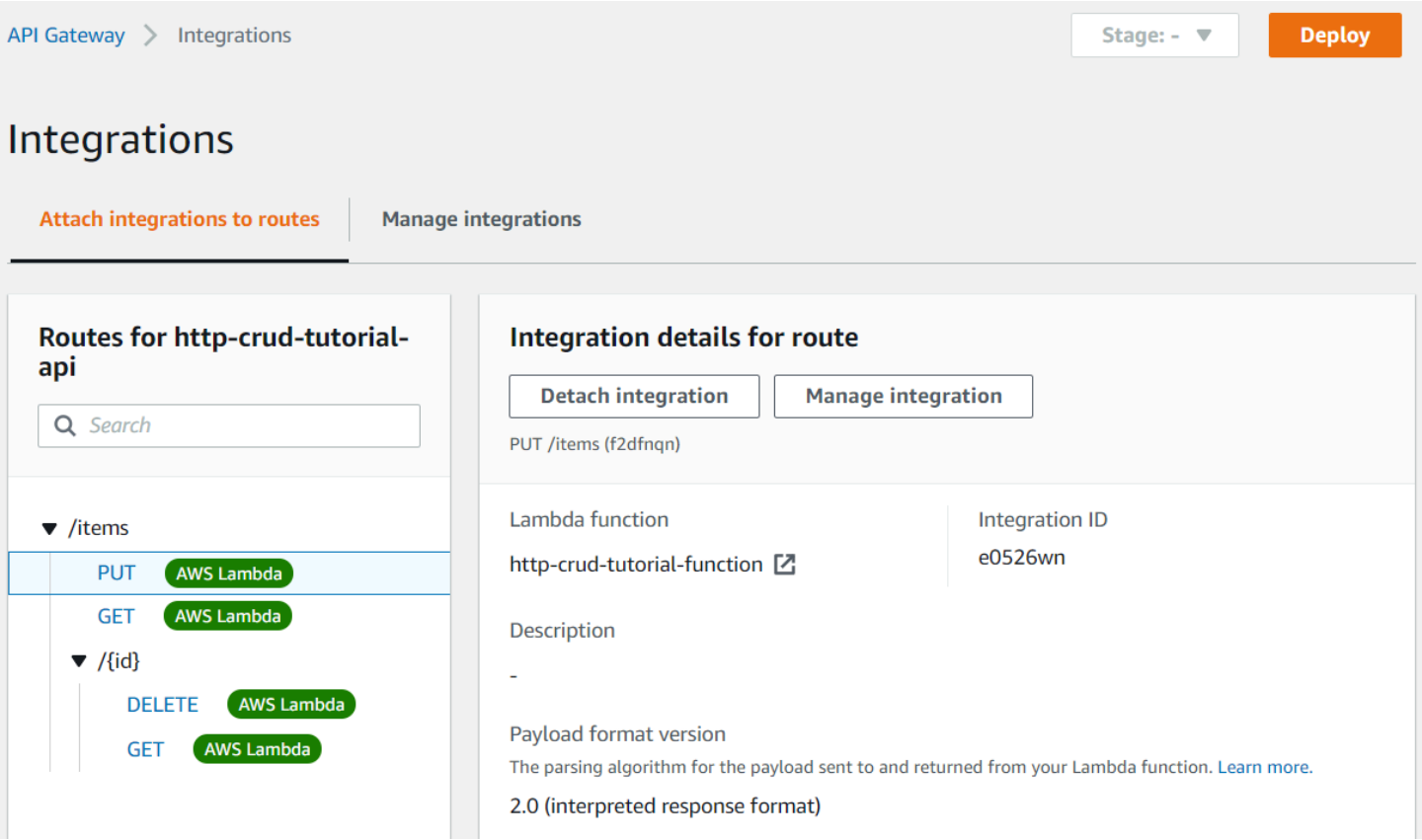
ステップ 6: 統合をルートにアタッチする

この API の例では、すべてのルートで同じ Lambda 統合を使用します。統合を API のすべてのルートにアタッチすると、クライアントがいずれかのルートを呼び出すと Lambda 関数が呼び出されます。

統合をルートにアタッチするには

1. https://console.aws.amazon.com/apigateway で API Gateway コンソールにサインインします。
2. API を選択します。
3. [統合] を選択します。
4. ルートを選択します。
5. [既存の統合を選択する] で、[**`http-crud-tutorial-function`**] を選択します。
6. [統合をアタッチする] を選択します。
7. すべてのルートについて、ステップ 4 ~ 6 を繰り返します。

すべてのルートが、AWS Lambda 統合がアタッチされていることを示します。



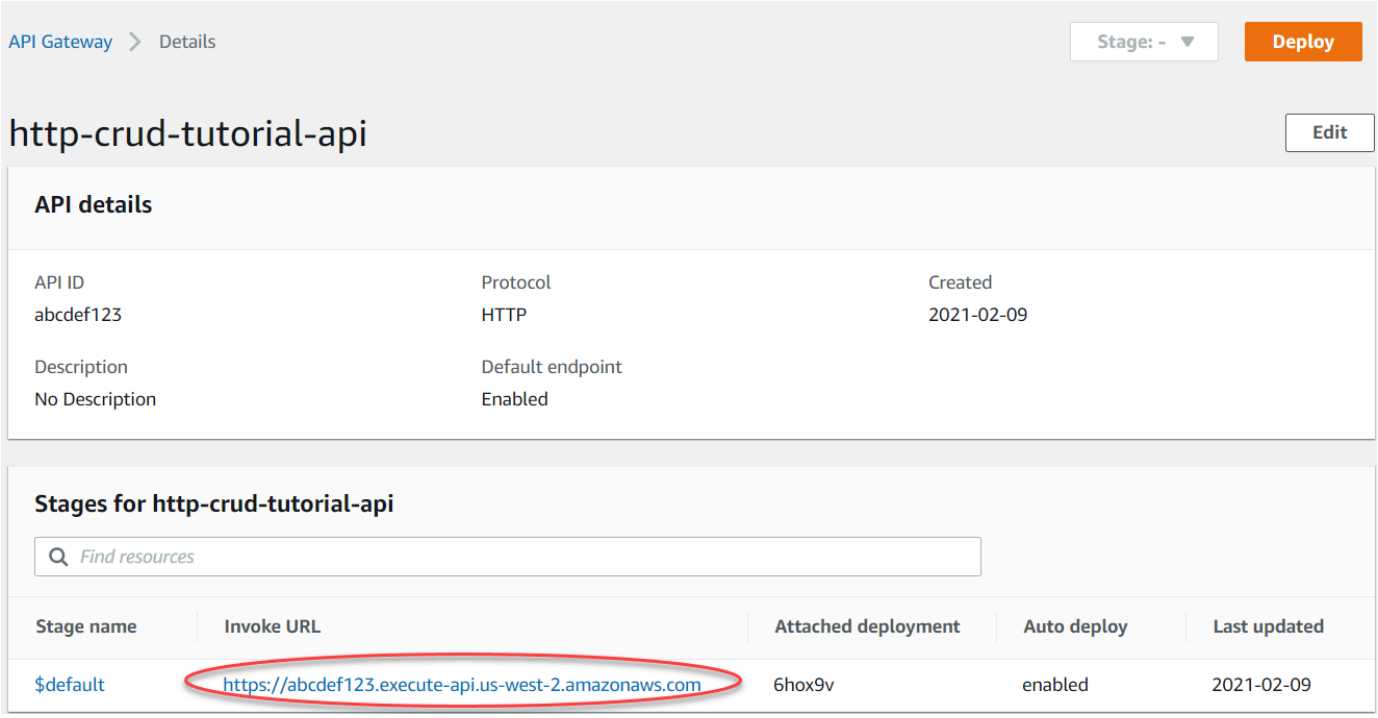
ルートと統合を持つ HTTP API ができたので、API をテストできます。

ステップ 7: API をテストする

API が動作していることを確認するには、curl を使用します。

API を呼び出すための URL を取得するには

1. https://console.aws.amazon.com/apigateway で API Gateway コンソールにサインインします。
2. API を選択します。
3. API の呼び出し URL を書き留めます。URL は、[詳細] ページの [URL を呼び出す] の下に表示されます。



4. API の呼び出し URL をコピーします。

   完全な URL は `https://abcdef123.execute-api.us-west-2.amazonaws.com` のように見えます。

項目の作成または更新

- 次のコマンドを使用して、項目を作成または更新します。コマンドには、項目の ID、料金、名前を含むリクエスト本文が含まれます。

```
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
```

すべての項目を取得するには

- すべての項目を一覧表示するには、次のコマンドを使用します。

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

1 つの項目を取得するには

- ID で 1 つの項目を取得するには、次のコマンドを使用します。

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

項目を削除するには

1. 項目を削除するには、次のコマンドを使用します。

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

2. 項目が削除されたことを確認するには、すべての項目を取得します。

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

ステップ 8: クリーンアップする

不要なコストを回避するには、このエクササイズで作成したリソースを削除します。次の手順では、HTTP API、Lambda 関数、および関連リソースを削除します。

DynamoDB テーブルを削除するには

1. DynamoDB コンソール (https://console.aws.amazon.com/dynamodb/) を開きます。
2. テーブルを選択します。
3. [テーブルの削除] を選択します。
4. 選択を確認して、[削除] をクリックします。

HTTP API を削除するには

1. https://console.aws.amazon.com/apigateway で API Gateway コンソールにサインインします。
2. [API] ページで、API を選択します。[Actions] を選択して、[Delete] を選択します。
3. [削除] を選択します。

Lambda 関数を削除するには

1. Lambda コンソール (https://console.aws.amazon.com/lambda/) にサインインします。
2. [関数] ページで、関数を選択します。[Actions] を選択して、[Delete] を選択します。
3. [削除] を選択します。

Lambda 関数のロググループを削除するには

1. Amazon CloudWatch コンソールで、[ロググループページ]を開きます。
2. [ロググループ] ページで、関数のロググループ (`/aws/lambda/http-crud-tutorial-function`) を選択します。[Actions] (アクション) を選択してから、[Delete log group] (ロググループの削除) を選択します。
3. [削除] を選択します。

Lambda 関数の実行ロールを削除するには

1. AWS Identity and Access Management コンソールの [Roles] (ロール) ページを開きます。
2. 関数のロールを選択します (例: `http-crud-tutorial-role`)。
3. [ロールの削除] を選択します。
4. [はい、削除します] を選択します。

次のステップ: AWS SAM または AWS CloudFormation を使用して自動化する

AWS CloudFormation または AWS SAM を使用して、AWS リソースの作成とクリーンアップを自動化できます。このチュートリアルのサンプル AWS SAM テンプレートについては、`template.yaml` を参照してください。

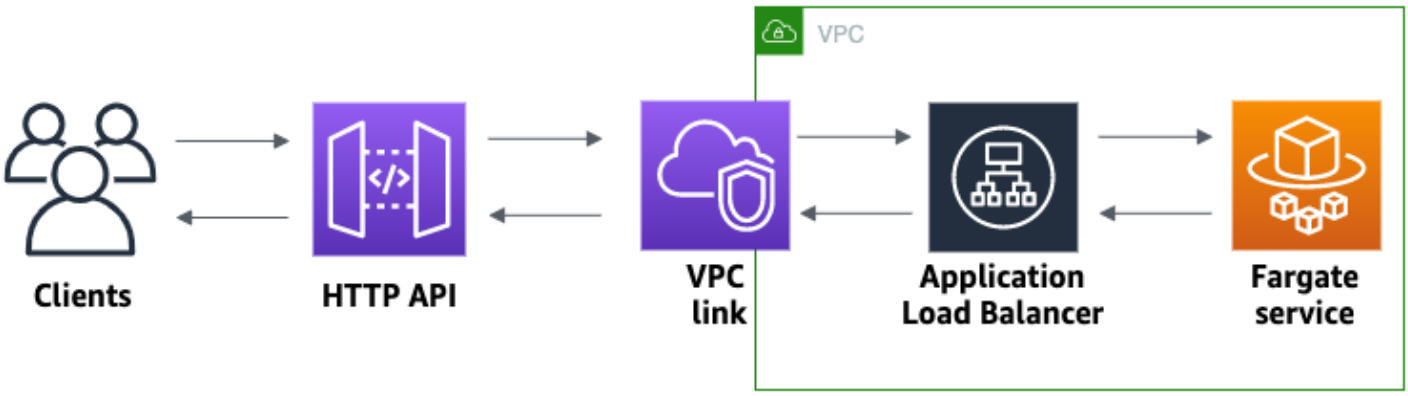
AWS CloudFormation テンプレートの例については、「サンプル AWS CloudFormation テンプレート」を参照してください。

チュートリアル: Amazon ECS サービスへのプライベート統合を使用した HTTP API の構築

このチュートリアルでは、Amazon VPC で実行される Amazon ECS サービスに接続するサーバーレス API を作成します。Amazon VPC 外のクライアントは、API を使用して Amazon ECS サービスにアクセスできます。

チュートリアルの所要時間は約 1 時間です。まず、AWS CloudFormation テンプレートを使用して Amazon VPC と Amazon ECS サービスを作成します。次に、API Gateway コンソールを使用して VPC リンクを作成します。VPC リンクは、API Gateway が Amazon VPC で実行されている Amazon ECS サービスにアクセスすることを許可します。次に、VPC リンクを使用して Amazon ECS サービスに接続する HTTP API を作成します。最後に、API をテストします。

HTTP API を呼び出すと、API Gateway は VPC リンクを介して Amazon ECS サービスにリクエストをルーティングし、サービスからの応答を返します。



このチュートリアルを完了するには、AWS アカウントと、コンソールへのアクセス権がある AWS Identity and Access Management ユーザーが必要です。詳細については、「前提条件」を参照してください。

このチュートリアルでは、 を使用しますAWS Management Console この API とすべての関連リソースを作成する AWS CloudFormation テンプレートについては、template.yaml を参照してください。

トピック

- ステップ 1: Amazon ECS サービスを作成する
- ステップ 2: VPC リンクを作成する
- ステップ 3: HTTP API を作成する
- ステップ 4: ルートを作成する
- ステップ 5: 統合を作成する
- ステップ 6: API をテストする
- ステップ 7: クリーンアップ
- 次のステップ: AWS CloudFormation を使用して自動化する

ステップ 1: Amazon ECS サービスを作成する

Amazon ECS は、クラスターで Docker コンテナを簡単に実行、停止、管理できるようにするコンテナ管理サービスです。このチュートリアルでは、Amazon ECS によって管理されるサーバーレスインフラストラクチャでクラスターを実行します。

この AWS CloudFormation テンプレートをダウンロードして解凍すると、Amazon VPC を含むサービスのすべての依存関係が作成されます。テンプレートを使用して、Application Load Balancer を使用する Amazon ECS サービスを作成します。

AWS CloudFormation スタックを作成するには

1. https://console.aws.amazon.com/cloudformation で AWS CloudFormation コンソール を開きます。
2. [スタックの作成] を選択し、[With new resources (standard) 新しいリソースを使用 (標準)] を選択します。
3. [Specify template (テンプレートの指定)] で、[Upload a template file (テンプレートファイルのアップロード)] を選択します。
4. ダウンロードしたテンプレートを選択します。
5. [Next (次へ)] を選択します。
6. [Stack name] (スタックの名前) で、**`http-api-private-integrations-tutorial`** と入力し、[Next] (次へ) を選択します。
7. [Configure stack options] (スタックオプションの設定) で、[Next] (次へ) を選択します。
8. [Capabilities] (機能) で、AWS CloudFormation がアカウントに IAM リソースを作成できることを承認します。
9. 送信 を選択します。

AWS CloudFormation は ECS サービスをプロビジョニングします。これには数分かかる場合があります。AWS CloudFormation スタックのステータスが CREATE_COMPLETE の場合は、次のステップに進む準備ができています。

ステップ 2: VPC リンクを作成する

VPC リンクは、API Gateway が Amazon VPC 内のプライベートリソースにアクセスすることを許可します。VPC リンクを使用して、クライアントが HTTP API を介して Amazon ECS サービスにアクセスできるようにします。

VPC リンクを作成するには

1. https://console.aws.amazon.com/apigateway で API Gateway コンソールにサインインします。
2. メインナビゲーションペインで、[VPC リンク]、[作成] の順に選択します。

   メインナビゲーションペインを開くには、必要に応じて、メニューアイコンを選択します。
3. [Choose a VPC link version] (VPC リンクバージョンを選択) で、[VPC link for HTTP APIs] (HTTP API の VPC リンク) を選択します。
4. [名前] に「**`private-integrations-tutorial`**」と入力します。
5. [VPC] で、ステップ 1 で作成した VPC を選択します。名前は、PrivateIntegrationsStack で始まる必要があります。
6. [Subnets] (サブネット) で、VPC 内の 2 つのプライベートサブネットを選択します。名前の末尾は `PrivateSubnet` です。
7. [Create] を選択します。

VPC リンクを作成すると、API Gateway は、VPC にアクセスするために Elastic Network Interface をプロビジョニングします。プロセスには数分かかることがあります。その間、API を作成できます。

ステップ 3: HTTP API を作成する

HTTP API は、Amazon ECS サービスの HTTP エンドポイントを提供します。このステップでは、空の API を作成します。ステップ 4 と 5 では、API と Amazon ECS サービスを接続するためのルートと統合を設定します。

HTTP API を作成するには

1. https://console.aws.amazon.com/apigateway で API Gateway コンソールにサインインします。
2. [API を作成] を選択し、[HTTP API] で [構築] を選択します。
3. [API 名] に「**`http-private-integrations-tutorial`**」と入力します。
4. [Next (次へ)] を選択します。
5. [ルートの設定] で、[次へ] を選択してルートの作成をスキップします。ルートは後で作成します。

6. API Gateway が作成するステージを確認します。API Gateway は、自動デプロイを有効にした `$default` ステージを作成します。これは、このチュートリアルでは最適な選択肢です。[Next (次へ)] を選択します。
7. [Create] を選択します。

ステップ 4: ルートを作成する

ルートは、着信 API リクエストをバックエンドリソースに送信する方法です。ルートは、HTTP メソッドとリソースパスという 2 つの部分で構成されます (例: `GET /items`)。この例の API では、1 つのルートを作成します。

ルートを作成するには

1. https://console.aws.amazon.com/apigateway で API Gateway コンソールにサインインします。
2. API を選択します。
3. [ルート] をクリックします。
4. [Create] を選択します。
5. [Method] (メソッド) で、[**ANY**] を選択します。
6. パスには、「**`/{proxy+}`**」と入力します。パスの最後の `{proxy+}` は、最大一致のパス変数です。API Gateway は、API へのすべてのリクエストをこのルートに送信します。
7. [Create] を選択します。

ステップ 5: 統合を作成する

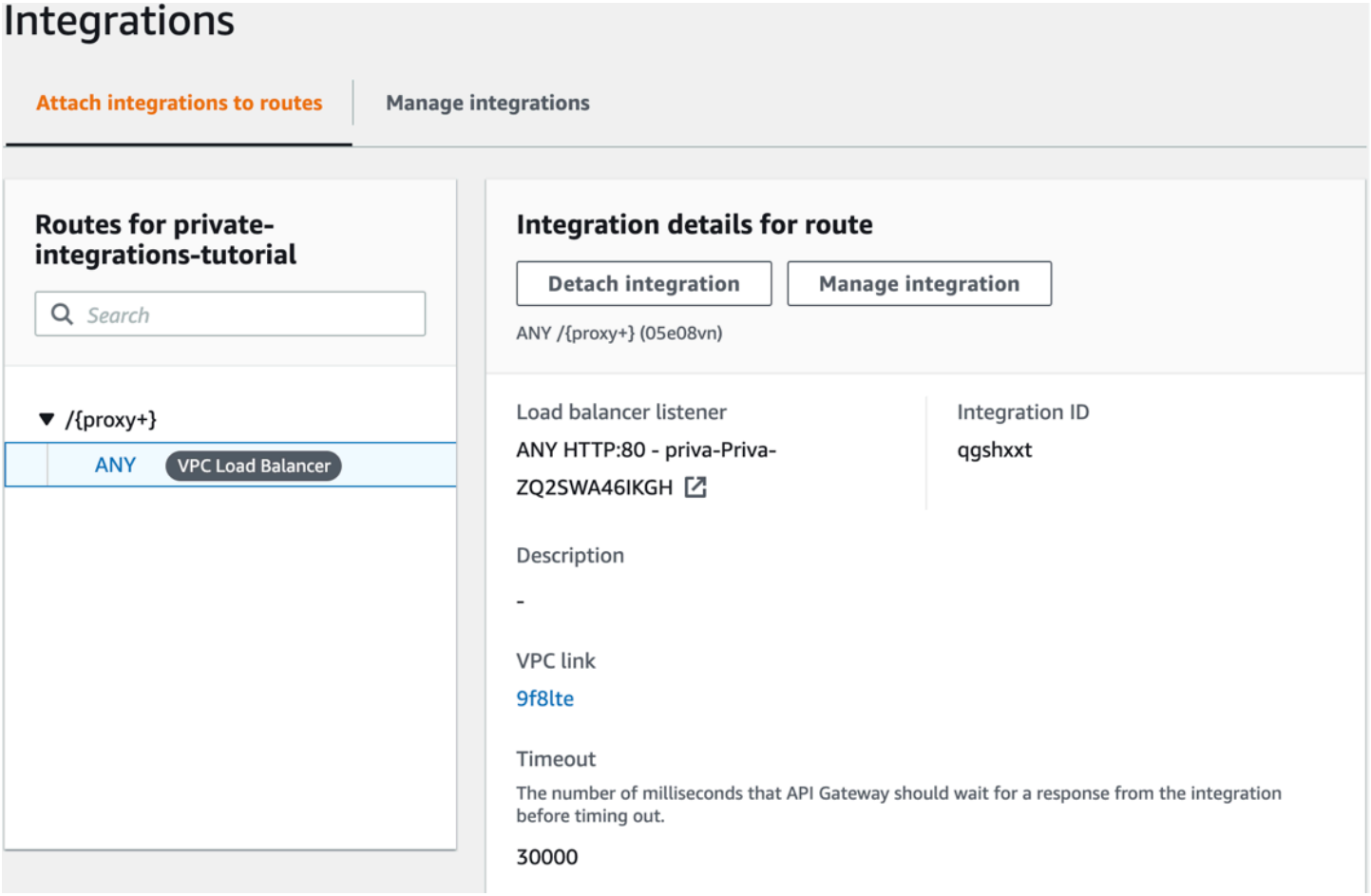
ルートをバックエンドリソースに接続するための統合を作成します。

統合を作成するには

1. https://console.aws.amazon.com/apigateway で API Gateway コンソールにサインインします。
2. API を選択します。
3. [統合] を選択します。
4. [統合を管理] を選択し、[作成] をクリックします。
5. [Attach this integration to a route] (この統合をルートにアタッチする) で、前に作成した ANY / {proxy+} ルートを選択します。
6. [Integration type] (統合タイプ) で、[Private resource] (プライベートリソース) を選択します。

7. [Integration details] (統合の詳細) で、[Select manually] (手動での選択) を選択します。
8. [Target service] (ターゲットサービス) で、[ALB/NLB] を選択します。
9. [Load Balancer] (ロードバランサー) には、ステップ 1 で AWS CloudFormation テンプレートを使用して作成したロードバランサーを選択します。名前は http-Priva で始まる必要があります。
10. [Listener] (リスナー) で、**`HTTP 80`** を選択します。
11. [VPC link] (VPC リンク) で、ステップ 2 で作成した VPC リンクを選択します。名前は `private-integrations-tutorial` である必要があります。
12. [Create] を選択します。

ルートと統合が正しく設定されていることを確認するには、[Attach integrations to routes] (統合をルートにアタッチする) を選択します。コンソールには、VPC ロードバランサーへの統合を含む `ANY /{proxy+}` ルートがあることが示されます。



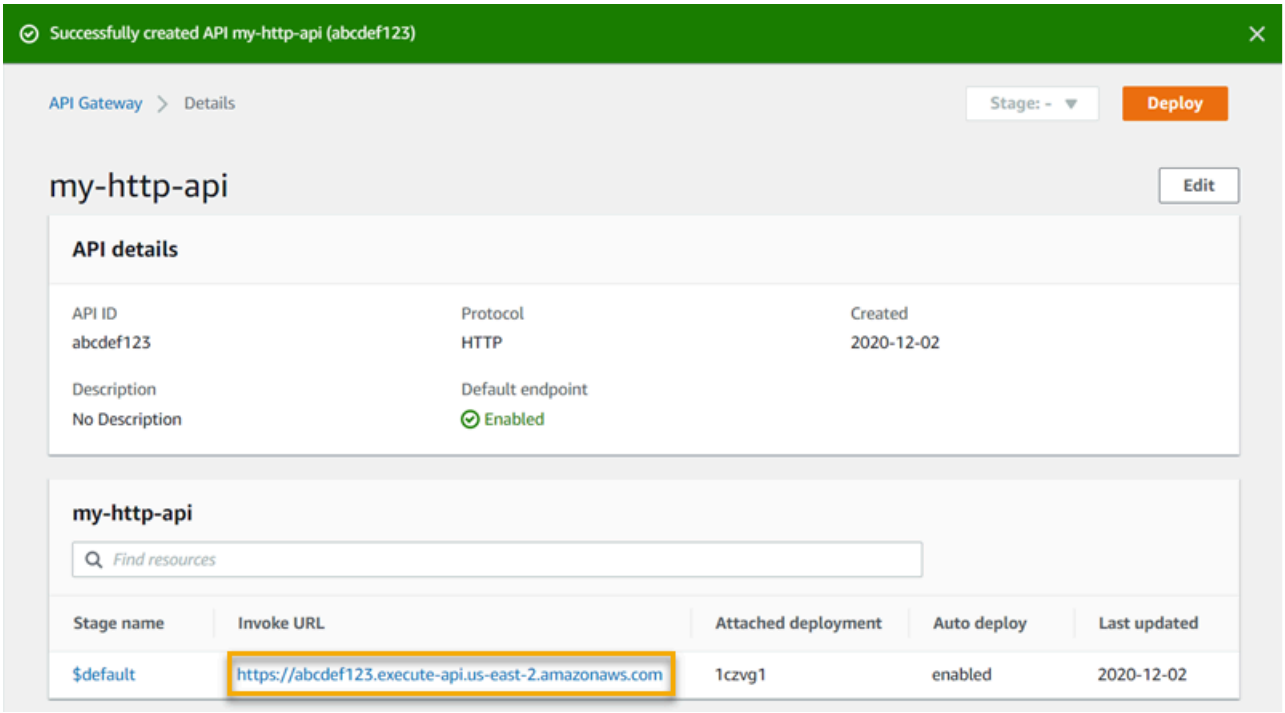
これで、API をテストする準備ができました。

ステップ 6: API をテストする

次に、API をテストして作動していることを確認します。シンプルにテストをするため、ウェブブラウザを使用して API を呼び出します。

API をテストするために

1. API Gateway コンソール (https://console.aws.amazon.com/apigateway) にサインインします。
2. API を選択します。
3. API の呼び出し URL を書き留めます。



4. ウェブブラウザで、API の呼び出し URL にアクセスします。

   URL は次のようになります。`https://abcdef123.execute-api.us-east-2.amazonaws.com`

   ブラウザが API に`GET`リクエストを送信します。

5. API の応答が、アプリケーションが Amazon ECS で実行されていることを示すウェルカムメッセージであることを確認します。

   Amazon VPC で実行される Amazon ECS サービスを正常に作成し、VPC リンクを持つ API Gateway HTTP API を使用して Amazon ECS サービスにアクセスすると、ウェルカムメッセージが表示されます。

ステップ 7: クリーンアップ

不要なコストを回避するには、このチュートリアルで作成したリソースを削除します。次の手順では、VPC リンク、AWS CloudFormation スタック、および HTTP API を削除します。

HTTP API を削除するには

1. https://console.aws.amazon.com/apigateway で API Gateway コンソールにサインインします。
2. [API] ページで API を選択します。[Actions] (アクション)、[Delete] (削除) の順に選択し、選択を確定します。

VPC リンクを削除するには

1. https://console.aws.amazon.com/apigateway で API Gateway コンソールにサインインします。
2. [VPC link] (VPC リンク) を選択します。
3. VPC リンクを選択し、[Delete] (削除) を選択して、選択を確定します。

AWS CloudFormation スタックを削除するには

1. AWS CloudFormation コンソール (https://console.aws.amazon.com/cloudformation) を開きます。
2. AWS CloudFormation スタックを選択します。
3. [Delete] (削除) を選択し、選択を確定します。

次のステップ: AWS CloudFormation を使用して自動化する

このチュートリアルで使用するすべての AWS リソースの作成とクリーンアップを自動化できます。AWS CloudFormation テンプレートの完全な例については、template.yaml を参照してください。

Amazon API Gateway WebSocket API チュートリアル

以下のチュートリアルでは API Gateway WebSocket API の学習に役立つ実践的な演習を提供します。

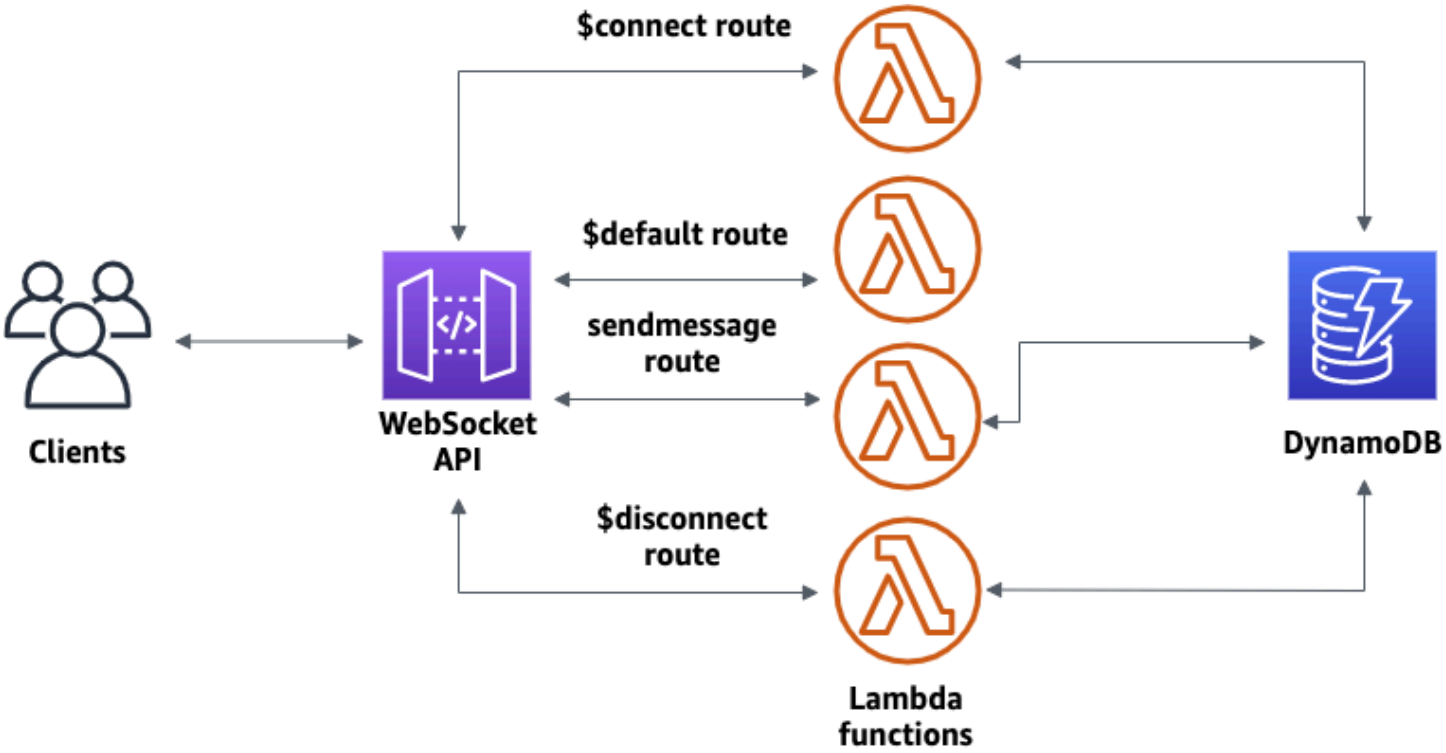
トピック

- チュートリアル: WebSocket API、Lambda、DynamoDB を使用したサーバーレスチャットアプリケーションの構築
- チュートリアル: 3 つの統合タイプを使用してサーバーレスアプリケーションを構築する

チュートリアル: WebSocket API、Lambda、DynamoDB を使用したサーバーレスチャットアプリケーションの構築

このチュートリアルでは、WebSocket API を使用してサーバーレスチャットアプリケーションを作成します。WebSocket API を使用すると、クライアント間の双方向通信をサポートできます。クライアントは、更新をポーリングしなくてもメッセージを受信できます。

このチュートリアルの完了には約 30 分かかります。最初に、AWS CloudFormation テンプレートを使用して API リクエストを処理する Lambda 関数と、クライアント ID を保存する DynamoDB テーブルを作成します。次に、API Gateway コンソールを使用して、Lambda 関数と統合する WebSocket API を作成します。最後に、API をテストして、メッセージが送受信されることを確認します。



このチュートリアルを完了するには、AWS アカウントと、コンソールへのアクセス権がある AWS Identity and Access Management ユーザーが必要です。詳細については、「前提条件」を参照してください。

また、API に接続するには `wscat` も必要です。詳細については、「the section called “`wscat` を使用した WebSocket API への接続とメッセージの送信”」を参照してください。

トピック

ステップ 1: Lambda 関数と DynamoDB テーブルを作成する

AWS CloudFormation のアプリケーション作成テンプレートをダウンロードして解凍します。このテンプレートを使用して、アプリケーションのクライアント ID を保存する Amazon DynamoDB テーブルを作成します。接続されている各クライアントには、テーブルのパーティションキーとして使用する一意の ID があります。このテンプレートは、DynamoDB のクライアント接続を更新し、接続されたクライアントへのメッセージの送信を処理する Lambda 関数も作成します。

AWS CloudFormation スタックを作成するには

1. https://console.aws.amazon.com/cloudformation で AWS CloudFormation コンソール を開きます。
2. [スタックの作成] を選択し、[With new resources (standard) 新しいリソースを使用 (標準)] を選択します。
3. [Specify template (テンプレートの指定)] で、[Upload a template file (テンプレートファイルのアップロード)] を選択します。
4. ダウンロードしたテンプレートを選択します。
5. [Next (次へ)] を選択します。
6. [Stack name] (スタックの名前) で、**`websocket-api-chat-app-tutorial`** と入力し、[Next] (次へ) を選択します。
7. [Configure stack options] (スタックオプションの設定) で、[Next] (次へ) を選択します。
8. [Capabilities] (機能) で、AWS CloudFormation がアカウントに IAM リソースを作成できることを承認します。
9. 送信 を選択します。

AWS CloudFormation は、テンプレートで指定されたリソースをプロビジョニングします。リソースのプロビジョニングには数分かかることがあります。AWS CloudFormation スタックのステータスが CREATE_COMPLETE の場合は、次のステップに進む準備ができています。

ステップ 2: WebSocket API を作成する

WebSocket API を作成して、クライアント接続を処理し、ステップ 1 で作成した Lambda 関数にリクエストをルーティングします。

WebSocket API を作成するには

1. https://console.aws.amazon.com/apigateway で API Gateway コンソールにサインインします。
2. [API の作成] を選択します。次に、[WebSocket API] で [Build] (ビルド) を選択します
3. [API 名] に「**`websocket-chat-app-tutorial`**」と入力します。
4. [Route selection expression] (ルート選択式) に「**`request.body.action`**」と入力します。ルート選択式は、クライアントがメッセージを送信したときに API Gateway が呼び出すルートを決定します。
5. [Next] を選択します。
6. [Predefined routes] (定義済みのルート) で、[Add $connect] ($connect の追加)、[Add $disconnect] ($disconnect の追加)、および [Add $default] ($default の追加) を選択します。[$connect] および [$disconnect] ルートは、クライアントが API に接続または切断したときに API Gateway が自動的に呼び出す特別なルートです。API Gateway は、他のルートがリクエストに一致するルートがない場合に、`$default` ルートを呼び出します。
7. [Custom routes] (カスタムルート) で、[Add custom route] (カスタムルートの追加) を選択します。[Route key] (ルートキー) に「**`sendmessage`**」と入力します。このカスタムルートは、接続されたクライアントに送信されるメッセージを処理します。
8. [Next] を選択します。
9. [Attach integrations] (統合のアタッチ) で、各ルートと [Integration type] (統合タイプ) ごとに、Lambda を選択します。

   [Lambda] には、ステップ 1 で AWS CloudFormation を使用して作成した、対応する Lambda 関数を選択します。各関数の名前はルートと一致します。例えば、[$connect] ルートの場合は、**`websocket-chat-app-tutorial-ConnectHandler`** という名前の関数を選択します。
10. API Gateway が作成するステージを確認します。デフォルトでは、API Gateway はステージ名 `production` を作成し、API をそのステージに自動的にデプロイします。[Next] を選択します。

11. [Create and deploy] (作成してデプロイ) を選択します。

ステップ 3: API をテストする

次に、API をテストして正しく動作していることを確認します。API に接続するには、`wscat` コマンドを使用します。

API の呼び出し URL を取得するには

1. https://console.aws.amazon.com/apigateway で API Gateway コンソールにサインインします。
2. API を選択します。
3. [Stages] (ステージ) を選択し、[production] (本稼働) を選択します。
4. API の [WebSocket URL] を書き留めます。URL は `wss://abcdef123.execute-api.us-east-2.amazonaws.com/production` のようになります。

API に接続するには

1. API に接続するには、以下のコマンドを使用します。API に接続すると、API Gateway は `$connect` ルートを呼び出します。このルートが呼び出されると、DynamoDB に接続 ID を保存する Lambda 関数が呼び出されます。

```
wscat -c wss://abcdef123.execute-api.us-west-2.amazonaws.com/production
```

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

2. 新しいターミナルを開き、次のパラメータを指定して wscat コマンドを再度実行します。

```
wscat -c wss://abcdef123.execute-api.us-west-2.amazonaws.com/production
```

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

これにより、メッセージを交換できる 2 つの接続されたクライアントが提供されます。

単一のメッセージを送信するには

- API Gateway は、API のルート選択式に基づいて、呼び出すルートを決定します。API のルート選択式は `$request.body.action` です。その結果、API Gateway は次のメッセージを送信したときに `sendmessage` ルートを呼び出します。

  ```
  {"action": "sendmessage", "message": "hello, everyone!"}
  ```

  呼び出されたルートに関連付けられた Lambda 関数は、DynamoDB からクライアント ID を収集します。次に、Lambda 関数は API Gateway Management API を呼び出し、それらのクライアントにメッセージを送信します。接続されているすべてのクライアントが、次のメッセージを受け取ります。

  ```
  < hello, everyone!
  ```

API の $default ルートを呼び出すには

- API Gateway は、定義されたルートと一致しないメッセージをクライアントが送信すると、API のデフォルトルートを呼び出します。`$default` ルートに関連付けられた Lambda 関数は、API Gateway Management API を使用して、接続に関するクライアント情報を送信します。

  ```
  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
  ```

API から切断するには

- API から切断するには、**CTRL+C** を押します。クライアントが API から切断されると、API Gateway は API の `$disconnect` ルートを呼び出します。API の `$disconnect` ルートの Lambda 統合は、DynamoDB から接続 ID を削除します。

ステップ 4: クリーンアップする

不要なコストを回避するには、このチュートリアルで作成したリソースを削除します。次のステップでは、AWS CloudFormation スタックと WebSocket API を削除します。

WebSocket API を削除するには

1. https://console.aws.amazon.com/apigateway で API Gateway コンソールにサインインします。
2. [API] ページで `websocket-chat-app-tutorial` API を選択します。[アクション]、[削除] の順に選択し、選択を確定します。

AWS CloudFormation スタックを削除するには

1. AWS CloudFormation コンソール (https://console.aws.amazon.com/cloudformation) を開きます。
2. AWS CloudFormation スタックを選択します。
3. [Delete] (削除) を選択し、選択を確定します。

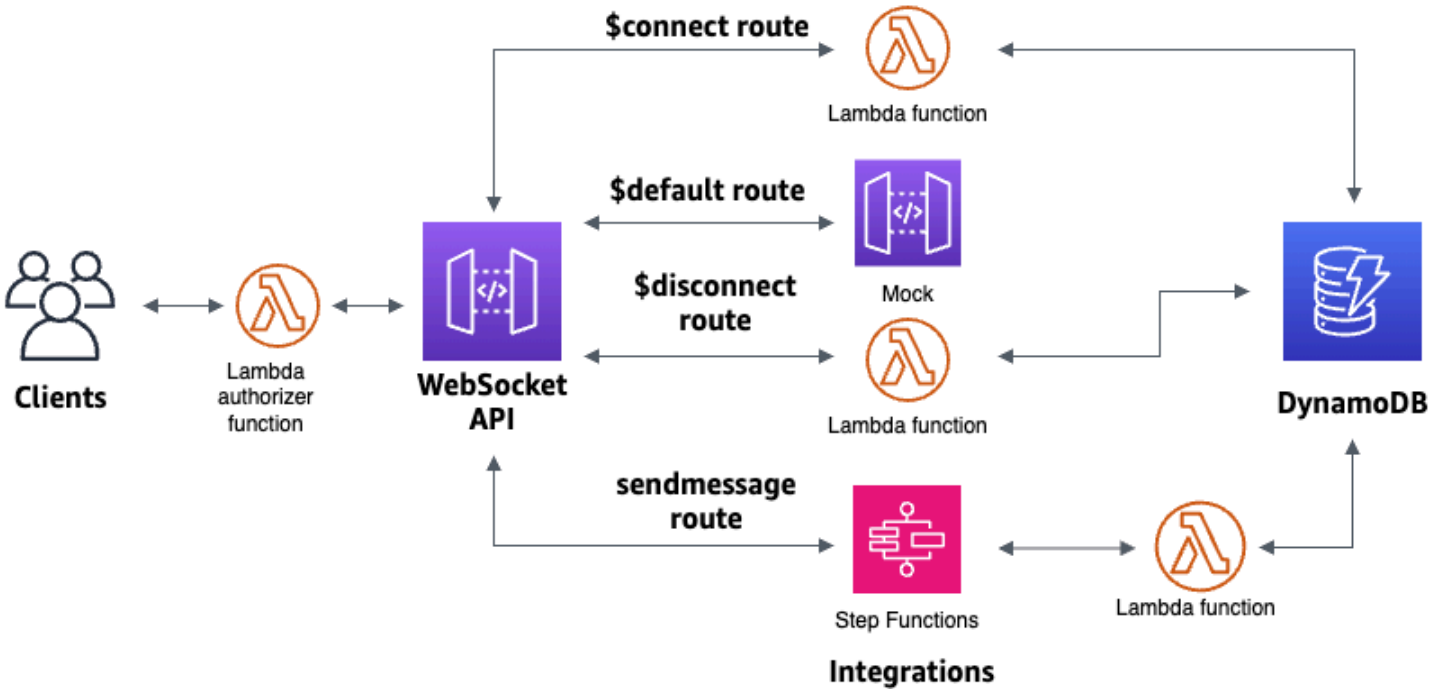
次のステップ: AWS CloudFormation を使用して自動化する

このチュートリアルで使用するすべての AWS リソースの作成とクリーンアップを自動化できます。この API とすべての関連リソースを作成する AWS CloudFormation テンプレートについては、chat-app.yaml を参照してください。

チュートリアル: 3 つの統合タイプを使用してサーバーレスアプリケーションを構築する

このチュートリアルでは、WebSocket API を使用してサーバーレスブロードキャストアプリケーションを作成します。クライアントは、更新をポーリングしなくてもメッセージを受信できます。

このチュートリアルでは、接続しているクライアントにメッセージをブロードキャストする方法を示します。また、Lambda オーソライザー、Mock 統合、Step Functions への非プロキシ統合の例も含まれています。



AWS CloudFormation テンプレートを使用してリソースを作成したら、API Gateway コンソールを使用して、AWS リソースと統合する WebSocket API を作成します。Lambda オーソライザーを API にアタッチし、AWS のサービスと Step Functions の統合を作成して、ステートマシンの実行を開始します。Step Functions ステートマシンは Lambda 関数を呼び出し、接続しているすべてのクライアントにメッセージを送信します。

API を構築したら、API への接続をテストし、メッセージが送受信されることを確認します。このチュートリアルの完了には約 45 分かかります。

トピック

- 前提条件
- ステップ 1: リソースを作成する
- ステップ 2: WebSocket API を作成する
- ステップ 3: Lambda オーソライザーを作成する
- ステップ 4: Mock 双方向統合を作成する
- ステップ 5: Step Functions で非プロキシ統合を作成する
- ステップ 6: API をテストする
- ステップ 7: クリーンアップする
- 次のステップ

前提条件

次の前提条件を満たしている必要があります。

- コンソールにアクセスできる AWS アカウントと AWS Identity and Access Management ユーザー。詳細については、「前提条件」を参照してください。
- API に接続するための `wscat`。詳細については、「the section called “wscat を使用した WebSocket API への接続とメッセージの送信”」を参照してください。

このチュートリアルを開始する前に、WebSocket チャットアプリケーションのチュートリアルを完了することをお勧めします。WebSocket チャットアプリケーションのチュートリアルを完了するには、「the section called “WebSocket チャットアプリケーション”」を参照してください。

ステップ 1: リソースを作成する

AWS CloudFormation のアプリケーション作成テンプレートをダウンロードして解凍します。このテンプレートを使用して以下を作成します。

- API リクエストを処理し、API へのアクセスを許可する Lambda 関数。
- Lambda オーソライザーから返されるクライアント ID とプリンシパルユーザー ID を保存する DynamoDB テーブル。
- 接続しているクライアントにメッセージを送信する Step Functions ステートマシン。

AWS CloudFormation スタックを作成するには

1. https://console.aws.amazon.com/cloudformation で AWS CloudFormation コンソール を開きます。
2. [スタックの作成] を選択し、[With new resources (standard) 新しいリソースを使用 (標準)] を選択します。
3. [Specify template (テンプレートの指定)] で、[Upload a template file (テンプレートファイルのアップロード)] を選択します。
4. ダウンロードしたテンプレートを選択します。
5. [Next (次へ)] を選択します。
6. [Stack name] (スタックの名前) で、**`websocket-step-functions-tutorial`** と入力し、[Next] (次へ) を選択します。

7. [Configure stack options] (スタックオプションの設定) で、[Next] (次へ) を選択します。
8. [Capabilities] (機能) で、AWS CloudFormation がアカウントに IAM リソースを作成できることを承認します。
9. 送信 を選択します。

AWS CloudFormation は、テンプレートで指定されたリソースをプロビジョニングします。リソースのプロビジョニングには数分かかることがあります。[出力] タブを選択して、作成したリソースとその ARN を表示します。AWS CloudFormation スタックのステータスが CREATE_COMPLETE の場合は、次のステップに進む準備ができています。

ステップ 2: WebSocket API を作成する

WebSocket API を作成して、クライアント接続を処理し、ステップ 1 で作成したリソースにリクエストをルーティングします。

WebSocket API を作成するには

1. https://console.aws.amazon.com/apigateway で API Gateway コンソールにサインインします。
2. [API の作成] を選択します。次に、[WebSocket API] で [Build] (ビルド) を選択します
3. [API 名] に「**`websocket-step-functions-tutorial`**」と入力します。
4. [Route selection expression] (ルート選択式) に「**`request.body.action`**」と入力します。

   ルート選択式は、クライアントがメッセージを送信したときに API Gateway が呼び出すルートを決定します。
5. [Next] を選択します。
6. [事前定義されたルート] で、[$connect を追加]、[$disconnect を追加]、[$default を追加] を選択します。

   $connect ルートと $disconnect ルートは、クライアントが API との接続または切断を行ったときに、API Gateway が自動的に呼び出す特別なルートです。API Gateway は、リクエストと一致するルートがないと、$default ルートを呼び出します。API を作成したら、Step Functions に接続するためのカスタムルートを作成します。
7. [Next] を選択します。
8. [$connect の統合] で、次の操作を行います。

   a. [統合タイプ] で、[Lambda] を選択します。

b. [Lambda 関数] で、ステップ 1 で AWS CloudFormation によって作成した該当する $connect Lambda 関数を選択します。Lambda 関数名は **websocket-step** で始まる必要があります。

9. [$disconnect の統合] で、次の操作を行います。

a. [統合タイプ] で、[Lambda] を選択します。

b. [Lambda 関数] で、ステップ 1 で AWS CloudFormation によって作成した該当する $disconnect Lambda 関数を選択します。Lambda 関数名は **websocket-step** で始まる必要があります。

10. [$default の統合] で、[Mock] を選択します。

Mock 統合の場合、API Gateway は統合バックエンドなしでルートレスポンスを管理します。

11. [Next] を選択します。

12. API Gateway が作成するステージを確認します。デフォルトでは、API Gateway は production という名前のステージを作成し、このステージに API を自動的にデプロイします。[Next] を選択します。

13. [Create and deploy] (作成してデプロイ) を選択します。

ステップ 3: Lambda オーソライザーを作成する

WebSocket API へのアクセスを制御するには、Lambda オーソライザーを作成します。Lambda オーソライザー関数は、AWS CloudFormation テンプレートで自動的に作成されています。この Lambda 関数は、Lambda コンソールで確認できます。名前は **websocket-step-functions-tutorial-AuthorizerHandler** で始まります。この Lambda 関数は、`Authorization` ヘッダーが `Allow` である場合を除いて、WebSocket API へのすべての呼び出しを拒否します。また、Lambda 関数は `$context.authorizer.principalId` 変数を API に渡します。この変数は、後で DynamoDB テーブルで API の呼び出し元を識別するために使用します。

このステップでは、Lambda オーソライザーを使用するように $connect ルートを設定します。

Lambda オーソライザーを作成するには

1. https://console.aws.amazon.com/apigateway で API Gateway コンソールにサインインします。
2. メインナビゲーションペインで、[オーソライザー] を選択します。
3. [オーソライザーを作成] を選択します。
4. [オーソライザー名] に、「**LambdaAuthorizer**」と入力します。

5. [オーソライザー ARN] に、AWS CloudFormation テンプレートで作成したオーソライザーの名前を入力します。名前は **`websocket-step-functions-tutorial-AuthorizerHandler`** で始まります。

   > **Note**
   >
   > このサンプルオーソライザーは、本稼働用 API で使用しないことをお勧めします。

6. [ID ソースタイプ] で、[ヘッダー] を選択します。[Key] (キー) に「**`Authorization`**」と入力します。
7. [オーソライザーの作成] を選択します。

オーソライザーを作成したら、API の $connect ルートにアタッチします。

オーソライザーを $connect ルートにアタッチするには

1. メインナビゲーションペインで、[ルート] を選択します。
2. [$connect] ルートを選択します。
3. [ルートリクエストの設定] セクションで、[編集] を選択します。
4. [認可] で、ドロップダウンメニューを選択し、リクエストオーソライザーを選択します。
5. [変更を保存] を選択します。

ステップ 4: Mock 双方向統合を作成する

次に、$default ルートの双方向 Mock 統合を作成します。Mock 統合では、バックエンドを使用することなく、クライアントにレスポンスを送信できます。$default ルートの統合を作成すると、API の操作方法をクライアントに示すことができます。

クライアントに sendmessage ルートを使用することを通知するように $default ルートを設定します。

Mock 統合を作成するには

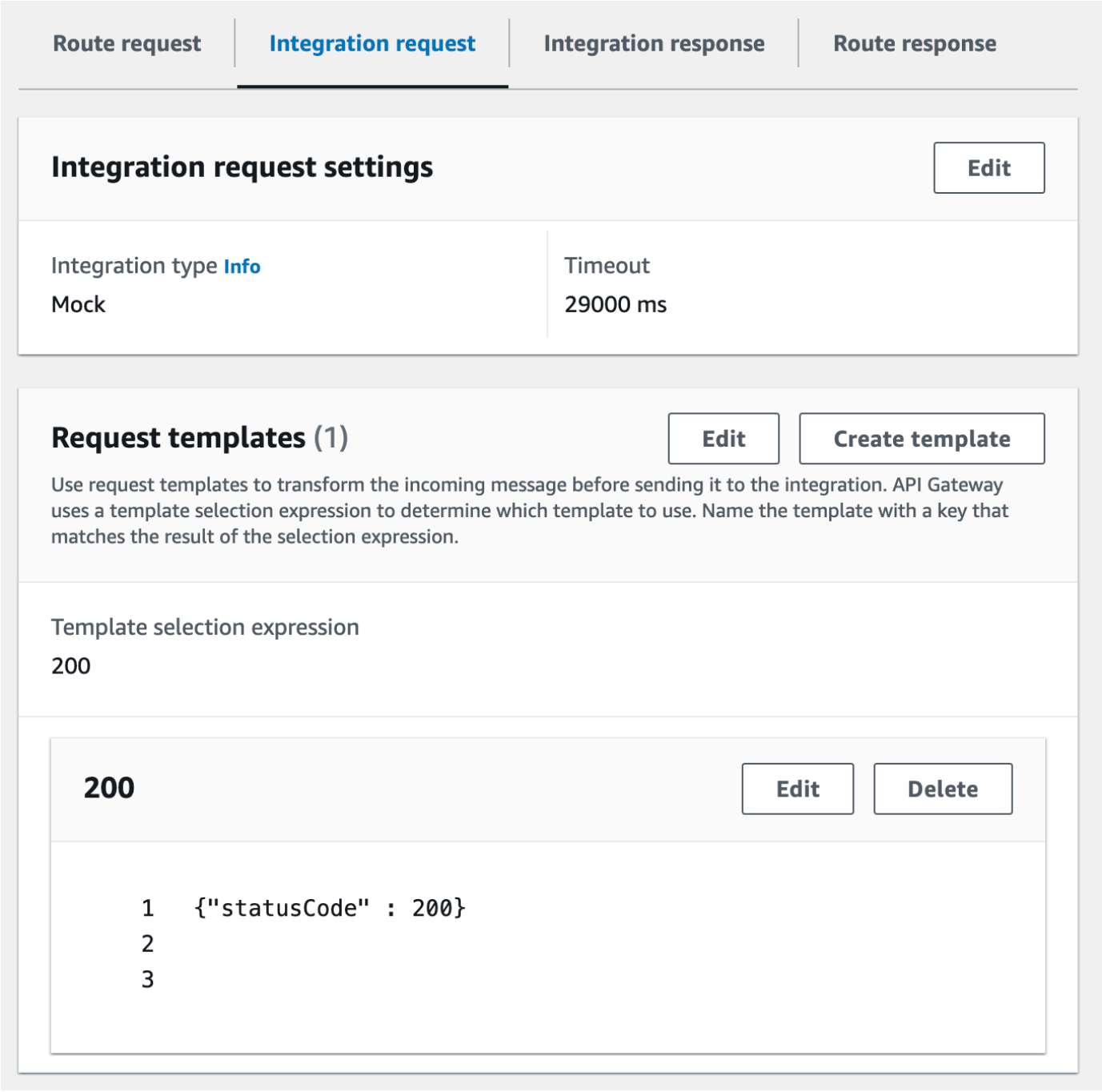
1. https://console.aws.amazon.com/apigateway で API Gateway コンソールにサインインします。
2. $default ルートを選択し、[統合リクエスト] タブを選択します。
3. [リクエストテンプレート] で、[編集] を選択します。

4. [テンプレート選択式] に「**200**」と入力して、[編集] を選択します。
5. [統合リクエスト] タブの [リクエストテンプレート] で、[テンプレートを作成] を選択します。
6. [テンプレートキー] に「**200**」と入力します。
7. [テンプレートを生成] で、次のマッピングテンプレートを入力します。

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

[テンプレートを作成] をクリックします。

結果は次のようになります。



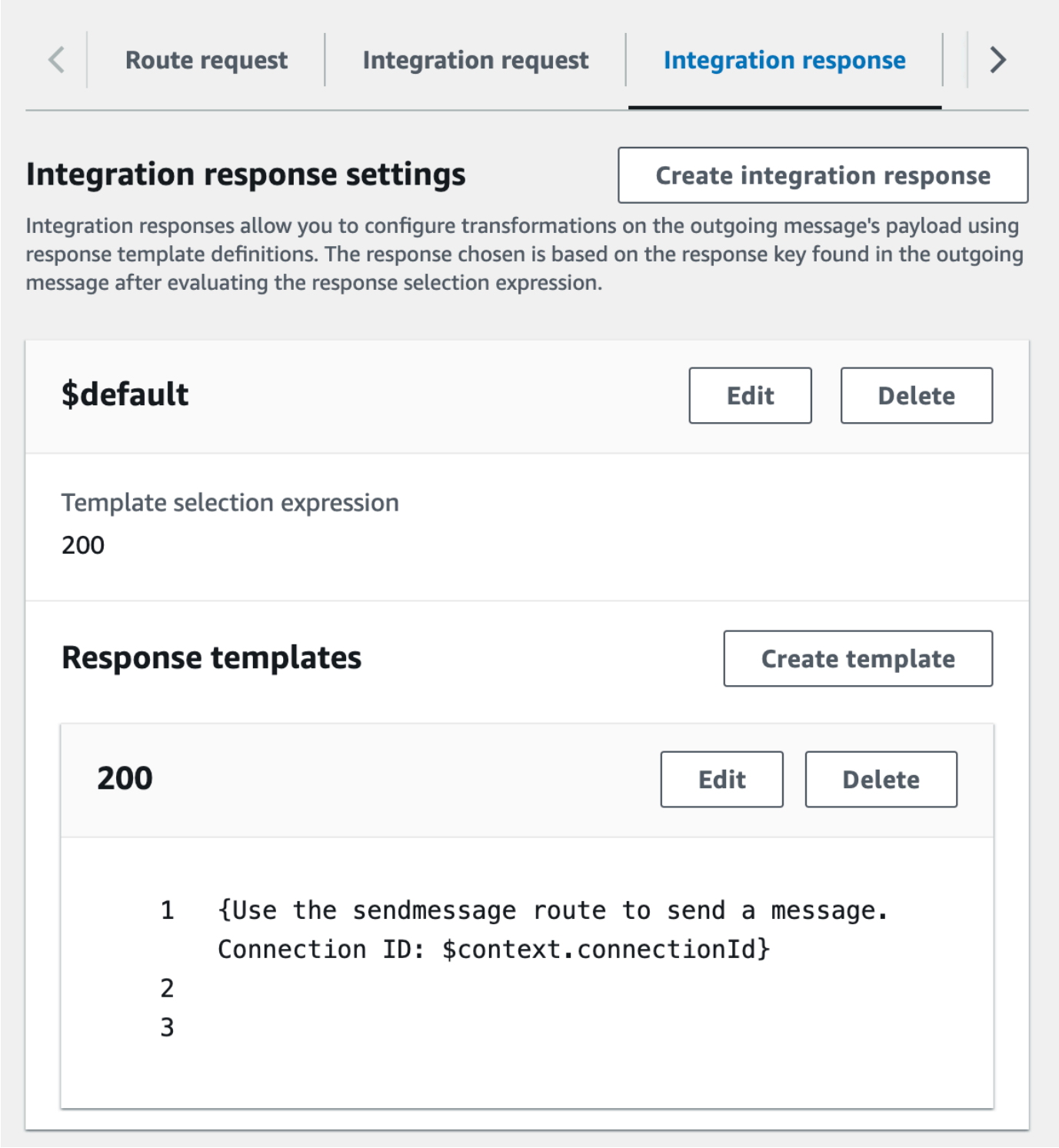
8. [$default ルート] ペインで、[双方向通信を有効にする] を選択します。
9. [統合レスポンス] タブ、[統合レスポンスを作成] の順に選択します。
10. [レスポンスキー] に「**`$default`**」と入力します 。
11. [テンプレート選択式] に「**`200`**」と入力します。
12. [レスポンスの作成] を選択します。
13. [レスポンステンプレート] で、[テンプレートを作成] を選択します。

14. [テンプレートキー] に「**200**」と入力します。
15. [レスポンステンプレート] に、次のマッピングテンプレートを入力します。

```
{"Use the sendmessage route to send a message. Connection ID:
 $context.connectionId"}
```

16. [テンプレートを作成] をクリックします。

    結果は次のようになります。



ステップ 5: Step Functions で非プロキシ統合を作成する

次に、sendmessage ルートを作成します。クライアントは、sendmessage ルートを呼び出して、接続しているすべてのクライアントにメッセージをブロードキャストできます。sendmessage ルー

トには、AWS のサービスと AWS Step Functions の非プロキシ統合が含まれています。この統合は、AWS CloudFormation テンプレートで自動的に作成した Step Functions ステートマシンに対して StartExecution コマンドを呼び出します。

非プロキシ統合を作成するには

1. https://console.aws.amazon.com/apigateway で API Gateway コンソールにサインインします。
2. [ルートの作成] を選択します。
3. [Route key] (ルートキー) に「**sendmessage**」と入力します。
4. [統合タイプ] で、[AWS のサービス] を選択します。
5. [AWS リージョン] に、AWS CloudFormation テンプレートをデプロイしたリージョンを入力します。
6. [AWS のサービス] で、[Step Functions] を選択します。
7. [HTTP メソッド] で、[POST] を選択します。
8. [アクション名] に「**StartExecution**」と入力します。
9. [実行ロール] に、AWS CloudFormation テンプレートで作成した実行ロールを入力します。名前は WebSocketTutorialApiRole にする必要があります。
10. [ルートの作成] を選択します。

次に、Step Functions ステートマシンにリクエストパラメータを送信するためのマッピングテンプレートを作成します。

マッピングテンプレートを作成するには

1. sendmessage ルートを選択し、[統合リクエスト] タブを選択します。
2. [リクエストテンプレート] セクションで、[編集] を選択します。
3. [テンプレート選択式] に「**\$default**」と入力します。
4. [編集] を選択します。
5. [リクエストテンプレート] セクションで、[テンプレートを作成] を選択します。
6. [テンプレートキー] に「**\$default**」と入力します。
7. [テンプレートを生成] で、次のマッピングテンプレートを入力します。

```
#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"
}
```

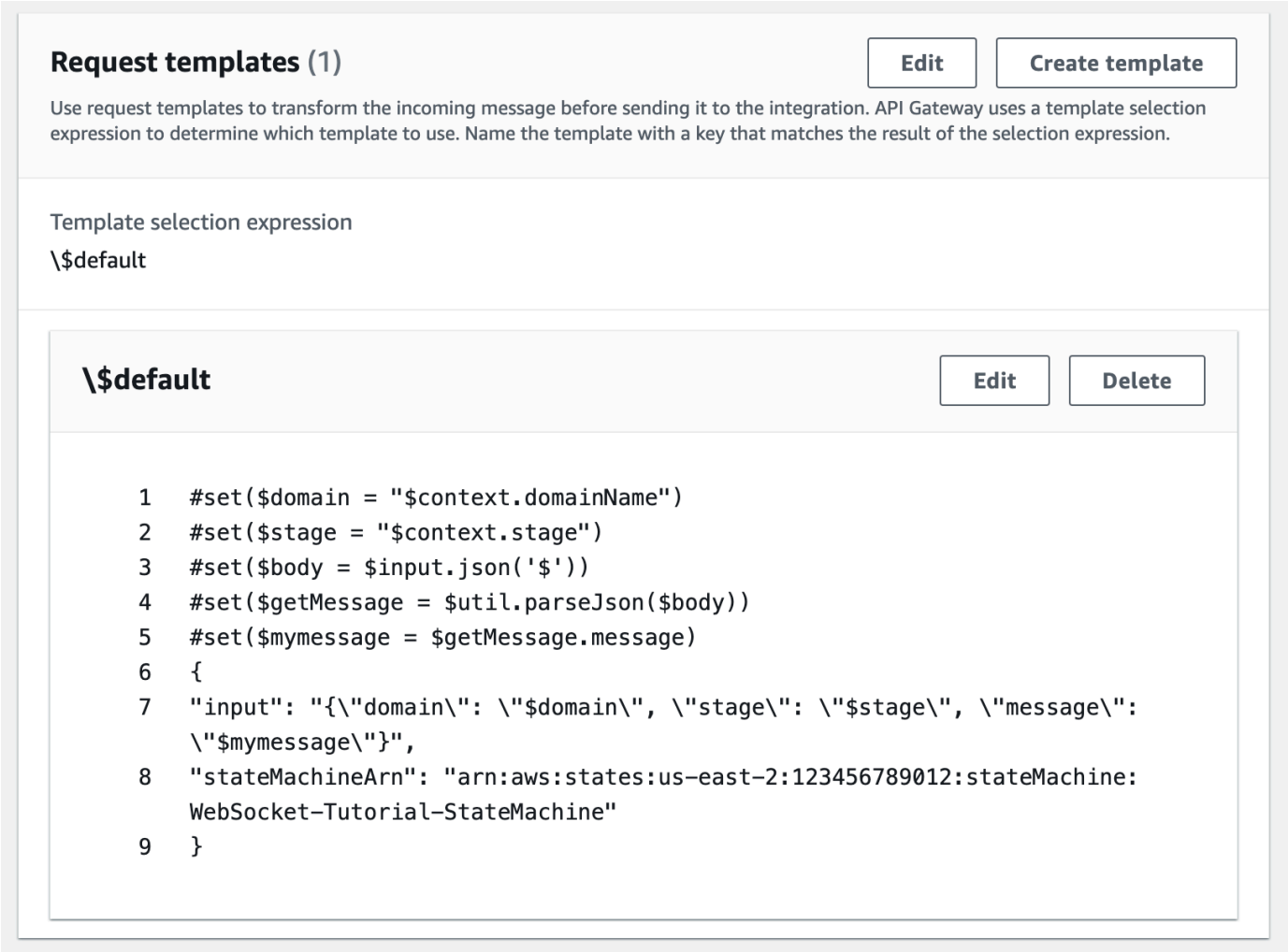
stateMachineArn を、AWS CloudFormation で作成したステートマシンの ARN に置き換えます。

マッピングテンプレートは、次の操作を行います。

- コンテキスト変数 domainName を使用して変数 $domain を作成します。
- コンテキスト変数 stage を使用して変数 $stage を作成します。

  コールバック URL を作成するには、$domain 変数と $stage 変数が必要です。
- 着信する sendmessage JSON メッセージを取り込み、message プロパティを抽出します。
- ステートマシンの入力を作成します。入力は、WebSocket API のドメインとステージ、および sendmessage ルートからのメッセージです。

8. [テンプレートを作成] をクリックします。



非プロキシ統合を $connect ルートまたは $disconnect ルートで作成し、Lambda 関数を呼び出すことなく、DynamoDB テーブルの接続 ID を直接追加または削除できます。

ステップ 6: API をテストする

次に、API をデプロイしてテストし、正しく動作することを確認します。`wscat` コマンドを使用して API に接続し、スラッシュコマンドで ping フレームを送信して WebSocket API への接続をチェックします。

API をデプロイするには

1. https://console.aws.amazon.com/apigateway で API Gateway コンソールにサインインします。
2. メインナビゲーションペインで、[ルート] を選択します。
3. [API のデプロイ] を選択します。

4. [ステージ] で、[production] を選択します。
5. (オプション)[デプロイの説明] に説明を入力します。
6. [デプロイ] を選択します。

API をデプロイしたら、これを呼び出すことができます。API を呼び出すには、呼び出し URL を使用します。

API の呼び出し URL を取得するには

1. API を選択します。
2. [Stages] (ステージ) を選択し、[production] (本稼働) を選択します。
3. API の [WebSocket URL] を書き留めます。URL は `wss://abcdef123.execute-api.us-east-2.amazonaws.com/production` のようになります。

呼び出し URL を取得したので、WebSocket API への接続をテストできます。

API への接続をテストするには

1. API に接続するには、以下のコマンドを使用します。まず、/ping パスを呼び出して接続をテストします。

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

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

2. 次のコマンドを入力してコントロールフレームに ping を送信します。コントロールフレームは、クライアント側からのキープアライブ目的に使用できます。

```
/ping
```

結果は次のようになります。

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

接続のテストが完了したので、API が正しく動作することをテストできます。このステップでは、新しいターミナルウィンドウを開いて、接続しているすべてのクライアントに WebSocket API からメッセージを送信できるようにします。

API をテストするには

1. 新しいターミナルを開き、次のパラメータを指定して wscat コマンドを再度実行します。

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

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

2. API Gateway は、API のルートリクエスト選択式に基づいて、どのルートを呼び出すかを決定します。API のルート選択式は $request.body.action です。その結果、API Gateway は次のメッセージを送信したときに sendmessage ルートを呼び出します。

```
{"action": "sendmessage", "message": "hello, from Step Functions!"}
```

ルートに関連する Step Functions ステートマシンは、メッセージとコールバック URL を使用して Lambda 関数を呼び出します。Lambda 関数は API Gateway 管理 API を呼び出し、接続しているすべてのクライアントにメッセージを送信します。すべてのクライアントは、次のメッセージを受け取ります。

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

WebSocket API のテストが完了したので、API との接続を切断できます。

API から切断するには

- API から切断するには、CTRL+C を押します。

  クライアントが API から切断すると、API Gateway は API の $disconnect ルートを呼び出します。API の $disconnect ルートの Lambda 統合は、DynamoDB から接続 ID を削除します。

ステップ 7: クリーンアップする

不要なコストを回避するには、このチュートリアルで作成したリソースを削除します。次のステップでは、AWS CloudFormation スタックと WebSocket API を削除します。

WebSocket API を削除するには

1. https://console.aws.amazon.com/apigateway で API Gateway コンソールにサインインします。
2. [API] ページで、[websocket-api] を選択します。
3. [アクション]、[削除] の順に選択し、選択を確定します。

AWS CloudFormation スタックを削除するには

1. AWS CloudFormation コンソール (https://console.aws.amazon.com/cloudformation) を開きます。
2. AWS CloudFormation スタックを選択します。
3. [Delete] (削除) を選択し、選択を確定します。

次のステップ

このチュートリアルに関連するすべての AWS リソースの作成とクリーンアップは自動化できます。このチュートリアルでこれらのアクションを自動化する AWS CloudFormation テンプレートの例については、「ws-sfn.zip」を参照してください。

REST API の操作

API Gateway の REST API は、バックエンドの HTTP エンドポイント、Lambda 関数、その他の AWS のサービスを使用して統合されているリソースおよびメソッドのコレクションです。API Gateway 機能を使用すると、作成から本番稼働 API のモニタリングまで、API ライフサイクルのあらゆる側面を支援できます。

API Gateway REST APIは、クライアントがサービスにリクエストを送信し、サービスが同期的に応答するリクエスト/レスポンスモデルを使用します。この種のモデルは、同期通信に依存する多くの異なる種類のアプリケーションに適しています。

トピック

API Gateway での REST API の開発

Amazon API Gateway では、REST API を、API Gateway リソースと呼ばれるプログラム可能なエンティティのコレクションとして構築します。たとえば、RestApi リソースを使用してリソースエンティティのコレクションを含むことができる API を表します。

各 Resource エンティティは、メソッドリソースを 1 つ以上持つことができます。Method は、クライアントから送信された受信リクエストであり、リクエストのパラメータと本文で示されます。これは、公開された Resource にクライアントがアクセスするためのアプリケーションプログラミングインターフェイスを定義します。Method をバックエンドエンドポイント (統合エンドポイントとも呼ばれます) と統合するには、統合リソースを作成します。これにより、受信リクエストが指定先の統合エンドポイント URI に転送されます。必要に応じて、バックエンド要件を満たすようにリクエストのパラメータや本文を変換できます。

レスポンスでは、MethodResponse を作成してクライアントが受信するリクエストを表し、IntegrationResponse リソースを作成してバックエンドから返されるリクエストレスポンスを表すことができます。統合レスポンスを設定し、データがクライアントに返される前にバックエンドレ

スポンスデータを変換するか、バックエンドレスポンスをそのままクライアントにパスすることができます。

ユーザーが API を理解しやすいように、API 作成の一環として、または API 作成後に API のドキュメントを提供することもできます。これを行うには、サポートされている API エンティティに DocumentationPart リソースを追加します。

クライアントが API を呼び出す方法を制御するには、 IAM アクセス許可、Lambda 認証、または Amazon Cognito ユーザープールを使用します。API の使用を計測するには、API リクエストを調整するように使用量プランを設定します。これらは、API の作成時または更新時に有効にすることができます。

API の作成方法の概要については、「the section called “チュートリアル: Lambda プロキシ統合による Hello World API”」を参照してください。REST API の開発時に使用できる API Gateway 機能の詳細については、以下のトピックを参照してください。これらのトピックには、概念情報に加えて、API Gateway コンソール、API Gateway REST API、AWS CLI、またはいずれかの AWS SDK を使用して実行できる手順が含まれています。

トピック

- API Gateway API エンドポイントタイプ
- API Gateway の REST API のメソッド
- API Gateway での REST API へのアクセスの制御と管理
- REST API 統合の設定
- API Gateway でリクエストの検証を使用する
- REST API のデータ変換の設定
- API Gateway でのゲートウェイレスポンス
- REST API リソースの CORS を有効にする
- REST API のバイナリメディアタイプの使用
- Amazon API Gateway での REST API の呼び出し
- OpenAPI を使用した REST API の設定

API Gateway API エンドポイントタイプ

API エンドポイントタイプは、API のホスト名を参照します。API エンドポイントタイプは、API トラフィックの大部分の発信元となっている場所に応じて、エッジ最適化、リージョン別、またはプライベートとすることができます。

エッジ最適化 API エンドポイント

エッジ最適化 API エンドポイントは通常、リクエストを最寄りの CloudFront POP (Point of Presence) にルーティングします。これは、クライアントが地理的に分散されている場合に役立ちます。これは、API Gateway REST API のデフォルトのエンドポイントタイプです。

エッジ最適化された API では、HTTP ヘッダーの名前の最初の文字は大文字になります (例: `Cookie`)。

CloudFront は、リクエストをオリジンに転送する前に、Cookie 名の自然な順序で HTTP Cookie を並べ替えます。CloudFront が Cookie を処理する方法の詳細については、「Cookie に基づいたコンテンツのキャッシュ」を参照してください。

エッジ最適化された API に使用するカスタムドメイン名はすべてのリージョンに適用されます。

リージョン API エンドポイント

リージョン API エンドポイントは、同じリージョン内のクライアントを対象としています。EC2 インスタンスで実行されているクライアントが同じリージョン内の API を呼び出すか、API が要求の高い少数のクライアントへのサービスを目的としている場合、リージョン API は接続のオーバーヘッドを減らします。

リージョン API の場合、ユーザーが使用するカスタムドメイン名は API がデプロイされているリージョンに固有です。複数のリージョンでリージョン別 API をデプロイする場合、すべてのリージョンで同じカスタムドメイン名を使用できます。カスタムドメインを Amazon Route 53 と組み合わせて使用すると、レイテンシーベースのルーティングなどのタスクを実行できます。詳細については、「the section called “リージョン別カスタムドメイン名の設定”」および「the section called “エッジ最適化カスタムドメイン名の作成”」を参照してください。

リージョン別 API エンドポイントは、すべてのヘッダー名をそのまま渡します。

> **Note**
>
> API クライアントが地理的に分散している場合にも、リージョン API エンドポイントを独自の Amazon CloudFront ディストリビューションと一緒に使用するのが合理的です。この場合、API Gateway は、サービスが制御する CloudFront ディストリビューションに API を関連付けないようにします。このユースケースの詳細については、「独自の CloudFront ディストリビューションで API Gateway をセットアップする方法を教えてください」を参照してください。

プライベート API エンドポイント

プライベート API エンドポイントは、Amazon Virtual Private Cloud (VPC) からしかアクセスできない API エンドポイントです。インターフェイス VPC エンドポイントは、VPC 内に作成するエンドポイントネットワークインターフェイス (ENI) です。詳細については、「the section called “プライベート REST API”」を参照してください。

Private API エンドポイントは、すべてのヘッダー名をそのまま渡します。

API Gateway で API エンドポイントタイプを変更する (パブリックまたはプライベート)

API エンドポイントタイプを変更するには、API の設定を更新する必要があります。API Gateway コンソール、AWS CLI、または API Gateway 用の AWS SDK を使用して、既存の API タイプを変更できます。現在の変更が完了するまでそのエンドポイントタイプを再度変更することはできませんが、API は使用可能になります。

次のエンドポイントタイプの変更がサポートされています。

- エッジ最適化からリージョンまたはプライベートへ
- リージョンからエッジ最適化またはプライベートへ
- プライベートからリージョンへ

プライベート API をエッジ最適化 API に変更することはできません。

パブリック API をエッジ最適化からリージョン、またはその逆に変更する場合、エッジ最適化 API とリージョン API とでは動作が異なることがあります。たとえば、エッジ最適化 API は `Content-MD5` ヘッダーを削除します。バックエンドに渡される MD5 ハッシュ値はすべて、リクエスト文字列パラメータまたは body プロパティで表現できます。ただし、リージョン API は (ヘッダー名を別の名前に再マップする場合がありますが) このヘッダーを渡します。この違いを理解することは、エッジ最適化 API をリージョン API に更新する方法、またはリージョン API をエッジ最適化 API に更新する方法を決めるのに役立ちます。

トピック

- API Gateway コンソールを使用して API エンドポイントの種類を変更する
- AWS CLI を使用して API エンドポイントタイプを変更する

API Gateway コンソールを使用して API エンドポイントの種類を変更する

API の API エンドポイントタイプを変更するには、次の一連の手順のいずれかを実行します。

エンドポイントをリージョン別またはエッジ最適化 (またはその逆) からパブリックに変換するには

1. https://console.aws.amazon.com/apigateway で API Gateway コンソールにサインインします。
2. REST API を選択します。
3. [API 設定] を選択します。
4. [API の詳細] セクションで [編集] を選択します。
5. API エンドポイントタイプには、[エッジ最適化] または [リージョン別] を選択します。
6. [Save changes] (変更の保存) をクリックします。
7. API を再デプロイします。これにより変更が有効になります。

プライベートエンドポイントをリージョンエンドポイントに変換するには

1. https://console.aws.amazon.com/apigateway で API Gateway コンソールにサインインします。
2. REST API を選択します。
3. VPC の外部からの API コールも VPC 内部と同様に成功するように、API のリソースポリシーを編集して、VPC または VPC エンドポイントの指定を削除します。
4. [API 設定] を選択します。
5. [API の詳細] セクションで [編集] を選択します。
6. [API エンドポイントタイプ] で、[リージョン別] を選択します。
7. [Save changes] (変更の保存) をクリックします。
8. リソースポリシーを API から削除します。
9. API を再デプロイします。これにより変更が有効になります。

リージョンエンドポイントをプライベートエンドポイントに変換するには

1. https://console.aws.amazon.com/apigateway で API Gateway コンソールにサインインします。
2. REST API を選択します。
3. VPC または VPC エンドポイントへのアクセスを許可するリソースポリシーを作成します。詳細については、「???」を参照してください。

4. [API 設定] を選択します。
5. [API の詳細] セクションで [編集] を選択します。
6. [API エンドポイントタイプ] で、[プライベート] を選択します。
7. (オプション) [VPC エンドポイント ID] で、プライベート API と関連付ける VPC エンドポイント の ID を選択します。
8. [Save changes] (変更の保存) をクリックします。
9. API を再デプロイします。これにより変更が有効になります。

AWS CLI を使用して API エンドポイントタイプを変更する

AWS CLI を使用して *{api-id}* という API の ID を持つエッジ最適化の API を更新するには、以下のように update-rest-api の を呼び出します。

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

成功のレスポンスには、`200 OK` ステータスコードと以下のようなペイロードが含まれます。

```
{

    "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"
}
```

逆に、以下のようにリージョン API をエッジ最適化 API に更新します。

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

put-rest-api は API 定義の更新用なので、API エンドポイントタイプの更新には適用されません。

API Gateway の REST API のメソッド

API Gateway では、API メソッドにメソッドリクエストとメソッドレスポンスが含まれます。API メソッドをセットアップし、バックエンドのサービスへのアクセスをリクエストするためにクライアントが実行しなければならない操作を定義し、その操作によってクライアントが受け取るレスポンスを定義します。入力では、クライアント用にメソッドリクエストパラメータか該当するペイロードを選択し、実行時に必須データやオプションデータを提供できます。出力では、メソッドレスポンスステータスコード、ヘッダー、該当する本文をターゲットとして定義し、クライアントに返される前のバックエンドレスポンスデータをマッピングできます。API の動作および入出力形式に関するデベロッパーの理解を促進するために、API をドキュメント化し、無効なリクエストに関する適切なエラーメッセージを提供することができます。

API メソッドリクエストは、HTTP リクエストです。メソッドリクエストをセットアップするには、HTTP メソッド (または動詞)、API リソースへのパス、ヘッダー、該当するクエリ文字列パラメータを設定します。HTTP メソッドが POST、PUT、または PATCH の場合、ペイロードも設定できます。たとえば、PetStore サンプル API を使用してペットを取得するには、`GET /pets/{petId}` の API メソッドリクエストを定義します。`{petId}` は、実行時に数字をとることができるパスパラメータです。

```
GET /pets/1
Host: apigateway.us-east-1.amazonaws.com
...
```

クライアントが誤ったパスを指定すると (`/pet/1` ではなく `/pets/one` や `/pets/1` など)、例外がスローされます。

API メソッドレスポンスは、指定のステータスコードの HTTP レスポンスです。非プロキシ統合では、メソッドレスポンスをセットアップしてマッピングの必須ターゲットまたはオプションのペアを指定する必要があります。これらは、統合レスポンスのヘッダーや本文を関連するメソッドレスポンスのヘッダーや本文に変換します。このマッピングは、そのまま統合を介してヘッダーまたは本文を渡す単純な ID 変換です。たとえば、次の `200` メソッドレスポンスに、成功した統合レスポンスがそのまま通過する例を示します。

```
200 OK
Content-Type: application/json
...

{
    "id": "1",
```

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

原理上は、バックエンドからの特定のレスポンスに対応するメソッドレスポンスを定義することができます。通常、これにはあらゆる 2XX、4XX、および 5XX レスポンスが含まれます。ただし、バックエンドが返すすべてのレスポンスを前もって把握できない可能性があるため、この手順は実用的でないことがあります。実際には、1 つのメソッドレスポンスをデフォルトとして指定し、バックエンドからの不明なレスポンスやマッピングされていないレスポンスを処理することができます。デフォルトとして 500 レスポンスを指定することをお勧めします。いずれの場合も、非プロキシ統合に対してメソッドレスポンスを 1 つ以上セットアップする必要があります。そうしなければ、バックエンドでリクエストが成功した場合であっても API Gateway が クライアントに 500 のエラーレスポンスを返します。

Java SDK など厳密に型指定された SDK を API 用にサポートするには、メソッドリクエストの入力用とメソッドレスポンスの出力用のデータモデルを定義する必要があります。

前提条件

API メソッドをセットアップする前に、以下について検証します。

- メソッドが API Gateway で使用可能であることが必要です。「チュートリアル: HTTP 非プロキシ統合を使用して REST API をビルドする」の手順に従います
- メソッドと Lambda 関数を通信させるには、IAM で Lambda 呼び出しロールと Lambda 実行ロールを作成済みである必要があります。メソッドが AWS Lambda で通信するための Lambda 関数も作成しておく必要があります。ロールと関数を作成するには、AWS Lambda 統合を選択するチュートリアル の「Lambda 非プロキシ統合用の Lambda 関数の作成」で説明する指示に従ってください。
- メソッドを HTTP または HTTP プロキシ統合と通信させるには、メソッドが通信する HTTP エンドポイント URL をすでに作成してアクセスを確立している必要があります。
- HTTP および HTTP プロキシエンドポイントの証明書が API Gateway によりサポートされていることを確認します。詳細については、「HTTP および HTTP プロキシ統合のために API Gateway によりサポートされる証明機関」を参照してください。

> **Note**
>
> REST API コンソールを使用してメソッドを作成する場合、統合リクエストとメソッドリクエストの両方を設定します。詳細については、「the section called “ コンソールを使用して統合リクエストを設定する”」を参照してください。

トピック

- API Gateway でメソッドリクエストをセットアップする
- API Gateway のメソッドレスポンスをセットアップする
- API Gateway コンソールを使用してメソッドをセットアップする

API Gateway でメソッドリクエストをセットアップする

メソッドリクエストのセットアップするには、RestApi リソースを作成した後に次のタスクを実行する必要があります。

1. 新しい API を作成するか、既存の API リソース エンティティを選択する。
2. 新しいまたは選択された API `Resource` の固有の HTTP 動詞になる API メソッドリソースを作成する。このタスクは、さらに次のサブタスクに分けられます。
   - HTTP メソッドをメソッドリクエストに追加する
   - リクエストパラメータを設定する
   - リクエストボディのモデルを定義する
   - 認証スキームを実行する
   - リクエストの検証を有効化する

これらのタスクは次のメソッドを使用して実行できます。

- API Gateway コンソール
- AWS CLI コマンド (create-resource と put-method)
- AWS SDK 関数 (Node.js の場合は createResource と putMethod など)
- API Gateway REST API (resource:create と method:put)

トピック

- API リソースをセットアップする
- HTTP メソッドをセットアップする
- メソッドリクエストパラメータをセットアップする
- メソッドリクエストモデルをセットアップする
- メソッドリクエスト認証をセットアップする
- メソッドリクエスト検証をセットアップする

API リソースをセットアップする

API Gateway API で、API リソースエンティティとしてアドレス可能なリソースを階層上部のルートリソース (/) とともに公開します。ルートリソースは API のベース URL に関連し、API エンドポイントとステージ名を構成します。API Gateway コンソールで、この基本 URI は Invoke URI と呼ばれ、API のデプロイ後に API のステージエディタに表示されます。

API エンドポイントは、デフォルトのホスト名やカスタムドメイン名にすることができます。デフォルトのホスト名は次の形式になります。

```
{api-id}.execute-api.{region}.amazonaws.com
```

この形式で、*{api-id}* は、API Gateway によって生成された API 識別子を示します。*{region}* 変数は、API 作成時に選択した AWS リージョン (us-east-1 など) を表します。カスタムドメイン名は、有効のインターネットドメインの下に存在するユーザーが使いやすい任意の名前です。たとえば、example.com のインターネットドメインを登録している場合は、あらゆる *.example.com がカスタムドメイン名として有効です。詳細については、「カスタムドメイン名を作成する」を参照してください。

PetStore サンプル API では、ルートリソース (/) がペットストアを公開します。/pets リソースは、ペットストアで使用可能なペットのコレクションを表します。/pets/{petId} は、指定の識別子 (petId) の個別のペットを公開します。{petId} のパスパラメータは、リクエストパラメータの一部です。

API リソースをセットアップするには、親として既存のリソースを選択した後、親リソースの下の子リソースを選択します。最初に親となるルートリソースから開始します。この親に子リソースを追加し、その子リソースに新しい親として別のリソースをする手順を親識別子まで繰り返します。その後、名前の付いたリソースを親に追加します。

AWS CLI により、get-resources コマンドを呼び出し、API のどのリソースが使用可能かを調べることができます。

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

結果として、API の現在使用可能なリソースのリストが返されます。PetStore サンプル API では、リストは次のように表示されます。

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

各アイテムは、ルートリソースを除くリソース (id) の識別子、直接の親 (parentId)、リソース名 (pathPart) をリストします。ルートリソースは他と異なり、親を持ちません。親としてリソースを選択した後、次のコマンドを呼び出して子リソースを追加します。

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

たとえば、PetStore ウェブサイトの商品にペットフードを追加するには、food を path-part に、food を parent-id に設定し、svzr2028x8 リソースをルート (/) に追加します。結果は次のようになります。

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

プロキシリソースを使用して API セットアップを効率化する

ビジネスの成長に応じて、PetStore のオーナーはフードや玩具などのペット関連のアイテムを商品に追加する場合があります。これをサポートするため、ルートリソースの下に /food や /toys などのリソースを追加できます。各販売カテゴリの下で、/food/{type}/{item} や /toys/{type}/{item} などのリソースをさらに追加する必要もあります。この手順には手間がかかる場合があります。ミドルレイヤー {subtype} をリソースパスに追加してパス階層を /food/{type}/{subtype}/{item} や /toys/{type}/{subtype}/{item} などに変更すると、この変更によって既存の API のセットアップは無効になります。これを回避するため、API Gateway プロキシリソースを使用して 1 度にすべての API リソースのセットを公開できます。

API Gateway はプロキシリソースを、リクエストが送信された際に指定されるリクエストのプレースホルダーとして定義しています。プロキシリソースは、greedy パスパラメータと呼ばれることも多い {proxy+} の特別なパスパラメータで示されます。+ マークは、付加されている子リソースを示します。/parent/{proxy+} プレースホルダ–は、/parent/* のパスパターンに一致するすべてのリソースを表します。greedy パスパラメータの名前である proxy は、通常のパスパラメータ名を扱うのと同じ方法で、別の文字列で置き換えることができます。

AWS CLI を使用して次のコマンドを呼び出し、ルート (/{proxy+}) の下に次のプロキシリソースをセットアップできます。

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

結果は次の例のようになります。

```
{
```

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

PetStore API の例では、/{proxy+} を使用して /pets と /pets/{petId} の両方を表すことができます。このプロキシリソースは、/food/{type}/{item} や /toys/{type}/{item} または /food/{type}/{subtype}/{item} や /toys/{type}/{subtype}/{item} など、他のリソース (既存のものや追加されるもの) も参照できます。バックエンド開発者はリソース階層を決定し、クライアント開発者はそれを理解する必要があります。API Gateway は単純に、クライアントがバックエンドに送信したものをすべてパスします。

API のプロキシリソースは、複数の場合があります。たとえば、API 内で次のプロキシリソースが許可されます。

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

プロキシリソースに非プロキシの兄弟がない場合、兄弟リソースはプロキシリソースの表現から除外されます。前の例では、/{proxy+} はルートリソースの下の /parent[/*] リソース以外のあらゆるリソースを表します。言い換えると、特定のリソースに対するメソッドリクエストは、リソース階層の同じレベルの一般的なリソースに対するメソッドリクエストより優先されます。

プロキシリソースは、子リソースを持つことができません。{proxy+} の後の API リソースは冗長かつあいまいです。API では次のプロキシリソースは許可されません。

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

HTTP メソッドをセットアップする

API メソッドリクエストは、API Gateway メソッドリソースに封入されます。メソッドリクエストをセットアップするには、最初に Method リソースをインスタンス化し、HTTP メソッドと認証タイプを 1 つ以上メソッドに設定します。

API Gateway はプロキシリソースに厳密に関連付けられており、ANY の HTTP メソッドをサポートします。この ANY メソッドは、実行時に指定されるすべての HTTP メソッドを表します。これによ

り、単一の API メソッドのセットアップを DELETE、GET、HEAD、OPTIONS、PATCH、POST および PUT のサポートされるすべての HTTP メソッドに使用できます。

同様に非プロキシリソースに ANY メソッドをセットアップできます。ANY メソッドをプロキシリソースと組み合わせることで、API のすべてのリソースに対し、サポートされる HTTP メソッドのための単一の API メソッドのセットアップを使用できます。さらに、バックエンドは既存の API セットアップを無効化することなく進化できます。

API メソッドをセットアップする前に、メソッドを呼び出すことができるユーザーについて考慮します。プランに従って認証タイプを設定します。オープンアクセスの場合は、NONE に設定します。IAM アクセス許可を使用するには、認証タイプを AWS_IAM に設定します。Lambda オーソライザー関数を使用するには、このプロパティを CUSTOM に設定します。Amazon Cognito ユーザープールを使用するには、認証タイプを COGNITO_USER_POOLS に設定します。

次の AWS CLI コマンドは、指定されたリソース (6sxz2j) のアクセスの制御に IAM 許可を使用して、このリソースに対する ANY 動詞のメソッドリクエストを作成する方法を示しています。

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

別の認証タイプで API メソッドリクエストを作成するには、「the section called “メソッドリクエスト認証をセットアップする”」を参照してください。

メソッドリクエストパラメータをセットアップする

リクエストパラメータは、クライアントがメソッドリクエストの完了に必要な入力データや実行コンテクストを提供する方法の 1 つです。メソッドパラメータは、パスパラメータ、ヘッダー、クエリ文字列パラメータにできます。メソッドリクエストのセットアップの一環として、必要なリクエストパラメータを宣言し、クライアントが使用できるようにする必要があります。非プロキシ統合では、これらのリクエストパラメータをバックエンド要件に適合する形式に変換できます。

たとえば、GET /pets/{petId} メソッドリクエストでは {petId} パス変数は必須リクエストパラメータです。このパスパラメータは、AWS CLI の put-method コマンドを呼び出す際に宣言できます。次の図に説明を示します。

```
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
```

必須ではないパラメータは、false で request-parameters に設定できます。たとえば、GET /pets メソッドが type のオプションクエリ文字列パラメータと breed のオプションヘッダーパラメータを使用する場合、/pets リソースの id が 6sxz2j であることを前提として、次の CLI コマンドを使用してこれらを宣言できます。

```
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
```

この省略形式の代わりに、JSON 文字列を使用して request-parameters 値を設定できます。

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

このようにセットアップすると、クライアントはペットをタイプ別にクエリできます。

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

さらに、クライアントは次のようにプードル種の犬をクエリできます。

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

メソッドリクエストパラメータを統合リクエストパラメータにマッピングする方法の詳細については、「the section called “統合”」を参照してください。

メソッドリクエストモデルをセットアップする

ペイロードに入力データを取ることができる API メソッドでは、モデルを使用できます。モデルは JSON スキーマのドラフト 4 で表され、リクエストボディのデータ構造を説明します。モデルにより、クライアントは入力としてメソッドリクエストペイロードを作成する方法を決定できます。さらに重要なことに、API Gateway はモデルを使用し、API Gateway コンソールで統合をセットアッ

プするためにリクエストを検証し、SDK を作成し、マッピングテンプレートを初期化します。モデルの作成方法については、「データモデルを理解する」を参照してください。

メソッドペイロードはコンテンツタイプに応じて異なる形式になる場合があります。モデルは適用されたペイロードのメディアタイプに対してインデックス作成されます。API Gateway は、Content-Type リクエストヘッダーを使用してコンテンツタイプを決定します。メソッドリクエストモデルをセットアップするには、AWS CLI put-method コマンドを呼び出す際に "*<media-type>*":"*<model-name>*" 形式のキー値のペアを requestModels マップに追加します。

コンテンツタイプに関係なく同じモデルを使用するには、キーとして $default を指定します。

たとえば、PetStore サンプル API の POST /pets メソッドリクエストの JSON ペイロードにモデルを設定するため、次の AWS CLI コマンドを呼び出すことができます。

```
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"}'
```

ここで petModel は、ペットを説明する name リソースの Model プロパティ値です。実際のスキーマ定義は、schema リソースの Model プロパティの JSON 文字列値として表されます。

Java または厳密に型指定された API のその他の SDK では、入力データはスキーマ定義から取得された petModel クラスとしてキャストされます。リクエストモデルを使用すると、作成された SDK 内の入力データは、デフォルトの Empty モデルから取得された Empty クラスにキャストされます。この場合、クライアントは正しいデータクラスをインスタンス化して必要な入力を提供することができません。

メソッドリクエスト認証をセットアップする

API メソッドを呼び出すことができるユーザーを制御するため、メソッドの認証タイプを設定できます。このタイプを使用し、IAM ロールとポリシー (AWS_IAM)、Amazon Cognito ユーザープール (COGNITO_USER_POOLS)、Lambda オーソライザー (CUSTOM) など、サポートされているオーソライザーのいずれかを有効にできます。

API メソッドへのアクセスを認証する IAM アクセス許可を使用するには、authorization-type への **AWS_IAM** 入力プロパティを設定します。このオプションを設定すると、API Gateway は、発信

者の証明情報に基づいて、リクエストにある発信者の署名を検証します。検証されたユーザーにメソッドを呼び出す権限がある場合、リクエストは承諾されます。それ以外の場合、リクエストは拒否され、発信者は未承認エラーレスポンスを受信します。発信者に API メソッドを呼び出す権限がない限り、メソッドの呼び出しは成功しません。以下の IAM ポリシーでは、同じ AWS アカウント 内で作成されたすべての API メソッドを呼び出す権限を発信者に付与します。

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

詳細については、「the section called “IAM アクセス許可を使用する”」を参照してください。

現在、このポリシーは API 所有者の AWS アカウント 内のユーザー、グループ、ロールにのみ付与できます。別の AWS アカウント のユーザーは、`execute-api:Invoke` アクションを呼び出すために必要なアクセス許可がある API 所有者の AWS アカウント 内のロールを引き受けることが許可されている場合のみ、API メソッドを呼び出すことができます。クロスアカウントアクセス許可の詳細については、「IAM ロールの使用」を参照してください。

AWS CLI や AWS SDK を使用するか、Signature Version 4 (SigV4) 署名を実装する Postman などの REST API クライアントを使用できます。

Lambda オーソライザーを使用して API メソッドへのアクセスを承認するには、`authorization-type` 入力プロパティを `CUSTOM` に設定し、`authorizer-id` 入力プロパティを既存の Lambda オーソライザーの `id` プロパティ値に設定します。参照された Lambda オーソライザーは、`TOKEN` または `REQUEST` タイプになります。Lambda オーソライザーの作成については、「the section called “Lambda 認証を使用する”」を参照してください。

Amazon Cognito ユーザープールを使用して API メソッドへのアクセスを承認するには、`authorization-type` 入力プロパティを `COGNITO_USER_POOLS` に設定し、`authorizer-id` 入力プロパティを作成済みの `COGNITO_USER_POOLS` オーソライザーの `id` プロパティ値に設定します。Amazon Cognito ユーザープールオーソライザーの詳細な作成方法については、「the

section called “REST API のオーソライザーとして Amazon Cognito ユーザープールを使用する”」を参照してください。

メソッドリクエスト検証をセットアップする

API メソッドリクエストをセットアップする際は、リクエスト検証を有効化できます。最初にリクエストバリデーターを作成する必要があります。

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

この CLI コマンドは、本文のみのリクエストバリデーターを作成します。次に出力例を示します。

```
{
    "validateRequestParameters": false,
    "validateRequestBody": true,
    "id": "jgpyy6",
    "name": "bodyOnlyValidator"
}
```

このリクエストバリデーターでは、メソッドリクエストのセットアップの一環として、リクエスト検証を有効化できます。

```
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
```

リクエスト検証に含まれたリクエストパラメータは、必須として宣言される必要があります。ページのクエリ文字列パラメータがリクエスト検証に使用されている場合、前の例の `request-parameters` マップは `'{"method.request.querystring.type": false, "method.request.querystring.page":true}'` として指定される必要があります。

API Gateway のメソッドレスポンスをセットアップする

API メソッドレスポンスは、クライアントが受信する API メソッドリクエストの出力をカプセル化します。出力データには、HTTP ステータスコード、一部のヘッダー、さらに場合によっては本文が含まれます。

非プロキシ統合を使用すると、指定されたレスポンスパラメータと本文は、関連する統合レスポンスデータからマッピングできます。また、マッピングに従って特定の静的な値を割り当てることができます。これらのマッピングは統合レスポンスで指定されています。マッピングは、そのままで統合レスポンスを通過する同一の変換になることがあります。

プロキシ統合により、API Gateway はバックエンドレスポンスを自動的にメソッドレスポンスにパスします。API メソッドレスポンスをセットアップする必要はありません。ただし、Lambda プロキシ統合の場合、Lambda 関数は API Gateway 用にこの出力形式で結果を返し、統合レスポンスをメソッドレスポンスにマッピングする必要があります。

プログラム上、メソッドレスポンスのセットアップは、API Gateway の MethodResponse リソースの作成および statusCode、responseParameters、responseModels のプロパティの設定を意味します。

API メソッドにステータスコードを設定する場合、予期せぬステータスコードのあらゆる統合レスポンスを処理するためのデフォルト設定を 1 つ選択する必要があります。これはキャスティングやマッピングされないサーバー側エラーのレスポンスになるため、デフォルトとして `500` を設定することが合理的です。ここでは説明のために、API Gateway コンソールはデフォルトとして `200` レスポンスを設定しています。しかし、これは `500` レスポンスにリセットできます。

メソッドレスポンスを設定するには、先にメソッドリクエストを作成している必要があります。

メソッドレスポンスステータスコードをセットアップする

メソッドレスポンスのステータスコードは、レスポンスのタイプを定義します。たとえば、200、400、500 のレスポンスは、それぞれ正常なクライアント側エラーレスポンスを示します。

メソッドレスポンスステータスコードをセットアップするには、`statusCode` プロパティを HTTP ステータスコードに設定します。次の AWS CLI コマンドは、`200` のメソッドレスポンスを作成します。

```
aws apigateway put-method-response \
       --region us-west-2  \
```

```
        --rest-api-id vaz7da96z6 \
        --resource-id 6sxz2j \
        --http-method GET \
        --status-code 200
```

メソッドレスポンスパラメータをセットアップする

メソッドレスポンスパラメータは、関連するメソッドリクエストへのレスポンスとしてクライアントが受信するヘッダーを定義します。レスポンスパラメータは、API Gateway が API メソッドの統合レスポンスに示されたマッピングに応じて統合レスポンスパラメータをマッピングするターゲットも指定します。

メソッドレスポンスパラメータをセットアップするには、responseParameters 形式の MethodResponse キー値のペアの "{parameter-name}":"{boolean}" マップに追加します。次の CLI コマンドは、my-header ヘッダーの設定例を示しています。

```
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
```

メソッドレスポンスモデルをセットアップする

メソッドレスポンスモデルは、メソッドレスポンス本文の形式を定義します。レスポンスモデルを設定する前に、API Gateway でモデルを作成する必要があります。そのために、create-model コマンドを呼び出すことができます。次の例に、PetStorePet メソッドリクエストへのレスポンスの本文を説明する GET /pets/{petId} モデルの作成方法を示します。

```
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" } \
                } \
            }'
```

結果は、API Gateway `Model` リソースとして作成されます。

メソッドレスポンスモデルをセットアップしてペイロード形式を定義するには、"application/json":"PetStorePet" キー値のペアを `MethodResponse` リソースの `requestModels` マップに追加します。次の `put-method-response` の AWS CLI コマンドにその方法を示します。

```
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"}'
```

メソッドレスポンスモデルのセットアップは、API のために厳密に型指定された SDK を作成する際に必要です。これにより、出力が Java や Objective-C の適切なクラスに確実にキャストされるようになります。それ以外の場合は、モデルの設定はオプションです。

API Gateway コンソールを使用してメソッドをセットアップする

REST API コンソールを使用してメソッドを作成する場合、統合リクエストとメソッドリクエストの両方を設定します。デフォルトでは、API Gateway はメソッドの `200` メソッドレスポンスを作成します。

以下の手順では、メソッドリクエスト設定を編集する方法と、メソッドに追加のメソッドレスポンスを作成する方法を示します。

トピック

- API Gateway コンソールで API Gateway メソッドリクエストを編集する
- API Gateway コンソールで API Gateway メソッドレスポンスをセットアップする

API Gateway コンソールで API Gateway メソッドリクエストを編集する

以下の手順では、メソッドリクエストを作成済みであることを前提としています。メソッドの作成方法の詳細については、「the section called “ コンソールを使用して統合リクエストを設定する”」を参照してください。

1. [リソース] ペインで、メソッドを選択し、[メソッドリクエスト] タブを選択します。
2. [メソッドリクエストの設定] セクションで、[編集] を選択します。
3. [承認] で、使用可能なオーソライザーを選択します。

   a. すべてのユーザーでメソッドへのオープンアクセスを有効化するには、[なし] を選択します。デフォルト設定が変更されていない場合、このステップはスキップできます。

   b. IAM アクセス許可を使用してメソッドへのクライアントアクセスを制御するには、[AWS_IAM] を選択します。これを選択した場合、適切な IAM ポリシーがアタッチされた IAM ロールのユーザーのみがこのメソッドを呼び出すことができます。

      IAM ロールを作成するには、以下のような形式のアクセスポリシーを指定します。

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

      このアクセスポリシーにおいて、*resource-statement* はメソッドの ARN です。メソッドの ARN は、[リソース] ページでメソッドを選択することで確認できます。これらの IAM アクセス許可の詳細については、「IAM アクセス許可により API へのアクセスを制御する」を参照してください。

      IAM ロールを作成するには、チュートリアル「???」の手順を応用できます。

c. Lambda オーソライザーを使用するには、トークンまたはリクエストオーソライザーを選択します。この選択肢をドロップダウンメニューに表示するには、Lambda オーソライザーを作成します。Lambda オーソライザーの詳細な作成方法については、「API Gateway Lambda オーソライザーを使用する」を参照してください。

d. Amazon Cognito ユーザープールを使用するには、[Cognito ユーザープールオーソライザー] で使用可能なユーザープールを選択します。この選択をドロップダウンメニューに表示するには、Amazon Cognito でユーザープールを作成し、API Gateway で Amazon Cognito ユーザープールオーソライザーを作成します。Amazon Cognito ユーザープール認証の作成方法については、「Amazon Cognito ユーザープールをオーソライザーとして使用して REST API へのアクセスを制御する」を参照してください。

4. リクエスト検証を指定するには、[リクエストの検証] ドロップダウンメニューから値を選択します。リクエスト検証を無効にするには、[なし] を選択します。各オプションの詳細については、「API Gateway でリクエストの検証を使用する」を参照してください。

5. [API キーの必要性] を選択すると、API キーを要求できるようになります。有効にすると、API キーは使用量プランで使用され、クライアントトラフィックを絞り込みます。

6. (オプション) API Gateway で生成されたこの API の Java SDK にオペレーション名を割り当てるには、[オペレーション名] に名前を入力します。たとえば、`GET /pets/{petId}` のメソッドリクエストでは、対応する Java SDK のオペレーション名は、デフォルトで `GetPetsPetId` です。この名前はメソッドの HTTP 動詞 (`GET`) とリソースパスの変数名 (`Pets` と `PetId`) から構成されています。オペレーション名を `getPetById` に設定した場合、SDK オペレーション名は `GetPetById` になります。

7. クエリ文字列パラメータをメソッドに追加するには、以下の操作を実行します。

   a. [URL クエリ文字列パラメータ] を選択してから、[クエリ文字列の追加] を選択します。

   b. [名前] に、クエリ文字列パラメータの名前を入力します。

   c. 新しく作成されたクエリ文字列パラメータがリクエスト検証に使用される場合は、[必須] を選択します。リクエスト検証の詳細については、「API Gateway でリクエストの検証を使用する」を参照してください。

   d. 新しく作成されたクエリ文字列パラメータがキャッシングキーの一部として使用される場合は、[キャッシュ] を選択します。キャッシングの詳細については、「メソッドパラメータまたは統合パラメータをキャッシュキーとして使用して、キャッシュされたレスポンスにインデックスを付ける」を参照してください。

   クエリ文字列パラメータを削除するには、[削除] を選択します。

8. メソッドにヘッダーパラメータを追加するには、以下の操作を実行します。

   a. [HTTP リクエストヘッダー] を選択した後、[ヘッダーの追加] を選択します。

   b. [名前] に、ヘッダーの名前を入力します。

   c. 新しく作成されたヘッダーがリクエスト検証に使用される場合は、[必須] を選択します。リクエスト検証の詳細については、「API Gateway でリクエストの検証を使用する」を参照してください。

   d. 新しく作成されたヘッダーがキャッシングキーの一部として使用される場合は、[キャッシュ] を選択します。キャッシングの詳細については、「メソッドパラメータまたは統合パラメータをキャッシュキーとして使用して、キャッシュされたレスポンスにインデックスを付ける」を参照してください。

   ヘッダーを削除するには、[削除] を選択します。

9. POST、PUT、または PATCH HTTP 動詞でメソッドリクエストのペイロード形式を宣言するには、[リクエスト本文] を選択して以下を実行します。

   a. [モデルの追加] を選択します。

   b. [コンテンツタイプ] に MIME タイプ (`application/json` など) を入力します。

   c. [モデル] では、ドロップダウンメニューからモデルを選択します。API で現在使用可能なモデルには、すでに作成して API の [モデル] コレクションに追加しているモデルに加え、デフォルトの `Empty` および `Error` モデルが含まれます。モデル作成についての詳細は、データモデルを理解する を参照してください。

   > **Note**
   >
   > モデルはペイロードの予測されるデータ形式をクライアントに通知するのに便利です。スケルトンベースのマッピングテンプレートを作成するのに役立ちます。API の厳密に型指定された SDK を Java、C#、Objective-C、および Swift などの言語で作成することが重要です。ペイロードに対するリクエスト検証が有効になっている場合にのみ必要です。

10. [Save] を選択します。

API Gateway コンソールで API Gateway メソッドレスポンスをセットアップする

API メソッドには、1 つ以上のレスポンスを含めることができます。各レスポンスは HTTP ステータスコードでインデックス作成されます。デフォルトでは、API Gateway コンソールはメソッドレスポンスに `200` レスポンスを追加します。たとえばこれを修正し、メソッドが `201` を返すように設定できます。アクセス拒否の `409`や、初期化されていないステージ変数が使用されている場合の `500` など、その他のレスポンスを追加することもできます。

API Gateway コンソールを使用してレスポンスを変更、削除、または API メソッドに追加するには、次の手順に従います。

1. [リソース] ペインで、メソッドを選択し、[メソッドレスポンス] タブを選択します。タブを表示するには、右矢印ボタンを選択する必要がある場合があります。
2. [メソッドレスポンスの設定] セクションで、[レスポンスを作成] を選択します。
3. [HTTP ステータスコード] には、`200`、`400`、または `500` などの HTTP ステータスコードを入力します。

   バックエンドが返したレスポンスに対応するメソッドレスポンスが定義されていない場合、API Gateway はクライアントにレスポンスを返しません。代わりに、`500 Internal server error` エラーレスポンスを返します。
4. [ヘッダーの追加] を選択します。
5. [ヘッダー名] に名前を入力します。

   バックエンドからクライアントにヘッダーを返すには、メソッドレスポンスにヘッダーを追加します。
6. [モデルを追加] を選択して、メソッドレスポンス本文の形式を定義します。

   [コンテンツタイプ] にレスポンスペイロードのメディアタイプを入力し、[モデル] ドロップダウンメニューからモデルを選択します。
7. [Save] を選択します。

既存のレスポンスを変更するには、メソッドレスポンスに移動し、[編集] を選択します。HTTP ステータスコードを変更するには、[削除] を選択し、新しいメソッドレスポンスを作成します。

バックエンドから返されたすべてのレスポンスで、互換性のあるレスポンスをメソッドレスポンスとして設定する必要があります。ただし、バックエンドの結果がクライアントに返される前にメソッドレスポンスにマッピングされない場合を除き、メソッドレスポンスのヘッダーとペイロードモデルの

設定はオプションです。API のために厳密に型指定された SDK を作成している場合は、メソッドレスポンスペイロードモデルも重要です。

API Gateway での REST API へのアクセスの制御と管理

API Gateway は API へのアクセスを制御し管理する複数のメカニズムをサポートしています。

認証と認可に次のメカニズムを使用することができます。

- リソースポリシーを使用して、特定のソース IP アドレスまたは VPC エンドポイントから、API およびメソッドへのアクセスを許可、または拒否するリソースベースのポリシーを作成できます。詳細については、「the section called “API Gateway リソースポリシーの使用”」を参照してください。
- 標準 AWS IAM ロールとポリシーは、API 全体または個々のメソッドに適用できる柔軟で堅牢なアクセスコントロールを提供します。IAM ロールとポリシーを使用して、API を作成および管理できるユーザーと、API を呼び出すことができるユーザーを制御できます。詳細については、「the section called “IAM アクセス許可を使用する”」を参照してください。
- IAM タグは、アクセスをコントロールするために、IAM ポリシーと共に使用できます。詳細については、「the section called “単一ドメイン内の属性ベースの”」を参照してください。
- インターフェイス VPC エンドポイント用のエンドポイントポリシー では、 IAM リソースポリシーをインターフェイス VPC エンドポイントにアタッチして、プライベート API のセキュリティを向上させることができます。詳細については、「the section called “プライベート API 用の VPC エンドポイントポリシーを使用する”」を参照してください。
- Lambda オーソライザーは、ヘッダー、パス、クエリ文字列、ステージ変数、コンテキスト変数のリクエストパラメータで記述される情報と同様に、ベアラートークン認証を使用して REST API メソッドへのアクセスを制御する Lambda 関数です。Lambda 認証は、REST API メソッドを呼び出すことができるユーザーを制御するために使用されます。詳細については、「the section called “Lambda 認証を使用する”」を参照してください。
- Amazon Cognito ユーザープールを使用して、REST API に関するカスタマイズ可能な認証と認可のソリューションを作成できます。Amazon Cognito ユーザープールは、REST API メソッドを呼び出すことができるユーザーを制御するために使用されます。詳細については、「the section called “REST API のオーソライザーとして Amazon Cognito ユーザープールを使用する”」を参照してください。

アクセスコントロールに関連する他のタスクを実行するために、以下のメカニズムを使用できます。

- Cross-Origin Resource Sharing (CORS) を使用して、クロスドメインのリソースリクエストへの REST API の応答を制御できます。詳細については、「the section called “CORS”」を参照してください。
- クライアント側 SSL 証明書を使用して、バックエンドシステムへの HTTP リクエストが API Gateway からのものであることを確認します。詳細については、「the section called “クライアント証明書”」を参照してください。
- AWS WAF を使用して、API Gateway API を一般的なウェブの脆弱性から保護することができます。詳細については、「the section called “AWS WAF”」を参照してください。

次のメカニズムを使用して、承認済みクライアントに付与したアクセス許可を追跡して制限できます。

- 使用量プランによって、顧客に API キーを提供でき、各 API キーの API ステージとメソッドの使用状況を追跡および制限できます。詳細については、「the section called “使用量プラン”」を参照してください。

API Gateway リソースポリシーを使用して API へのアクセスを制御する

Amazon API Gateway のリソースポリシーは、JSON ポリシードキュメントです。指定されたプリンシパル (通常、IAM ロールまたはグループ) で API を呼び出せるかどうかにかかわらず、制御する API にアタッチします。API Gateway リソースポリシーを使用すると、API を以下から安全に呼び出すことができます。

- 指定された AWS アカウントのユーザー
- 指定されたソース IP アドレス範囲または CIDR ブロック
- 指定された Virtual Private Cloud (VPC) または VPC エンドポイント (任意のアカウント)

リソースポリシーを API Gateway の API エンドポイントタイプにアタッチするには、AWS Management Console、AWS CLI、または AWS SDK を使用できます。プライベート API の場合は、リソースポリシーを VPC エンドポイントポリシーとともに使用して、どのプリンシパルがどのリソースやアクションにアクセスできるかを制御することができます。詳細については、「the section called “プライベート API 用の VPC エンドポイントポリシーを使用する”」を参照してください。

API Gateway リソースポリシーは、IAM アイデンティティベースのポリシーとは異なります。IAM アイデンティティベースのポリシーは、IAM ユーザー、グループ、またはロールにアタッチされ、

実行できるアクションとリソースを定義します。API Gateway リソースポリシーはリソースにアタッチされます。API Gateway リソースポリシーは、IAM ポリシーと組み合わせて使用できます。詳細については、「アイデンティティベースのポリシーおよびリソースベースのポリシー」を参照してください。

トピック

- Amazon API Gateway のアクセスポリシー言語の概要
- API Gateway リソースポリシーが認証ワークフローに与える影響
- API Gateway リソースポリシーの例
- API Gateway リソースポリシーを作成して API にアタッチする
- API Gateway リソースポリシーで使用できる AWS の条件キー

Amazon API Gateway のアクセスポリシー言語の概要

このページでは、Amazon API Gateway リソースポリシーで使用される基本的な要素について説明します。

リソースポリシーは、IAM ポリシーと同じ構文を使用して指定されます。ポリシー言語の詳細については、IAM ユーザーガイドの「IAM ポリシーの概要」と「AWS Identity and Access Management ポリシーリファレンス」を参照してください。

指定されたリクエストを AWS のサービスが許可または拒否する方法については、「リクエストの許可または拒否を決定する」を参照してください。

アクセスポリシーの一般的なエレメント

基本的に、リソースポリシーには以下のエレメントが含まれます。

- リソース – アクセス許可を許可または拒否できる Amazon API Gateway のリソースが API です。ポリシーでは、Amazon リソースネーム (ARN) を使用して、リソースを識別します。略式構文を使用することもできます。この構文はリソースポリシーの保存時に API Gateway によって完全な ARN に自動的に展開されます。詳細については、「API Gateway リソースポリシーの例」を参照してください。

  完全な `Resource` 要素の形式については、「API Gateway で API を実行するためのアクセス許可のリソース形式」を参照してください。

- アクション – Amazon API Gateway により、リソースごとに一連のオペレーションがサポートされています。許可 (または拒否) するリソースのオペレーションは、アクションキーワードを使用して識別します。

  たとえば、`execute-api:Invoke` アクセス権限は、クライアントリクエスト時の API の呼び出しをユーザーに許可します。

  `Action` エレメントの形式については、「API Gateway で API を実行するためのアクセス許可のアクション形式」を参照してください。
- エフェクト – ユーザーが特定のアクションをリクエストする際のエフェクト – 値は `Allow` または `Deny` のいずれかになります。また、明示的にリソースへのアクセスを拒否すると、別のポリシーによってアクセスが許可されている場合でも、ユーザーはそのリソースにアクセスできなくなります。

  > **Note**
  >
  > 「暗黙的な拒否」は、「デフォルトでは拒否」と同じ意味です。
  > 「暗黙的な拒否」は「明示的な拒否」とは異なります。詳細については、「デフォルトによる拒否と明示的な拒否の違い」を参照してください。
- プリンシパル – ステートメントのアクションやリソースへのアクセスが許可されているアカウントまたはユーザーを指します。リソースポリシーでは、プリンシパルは、このアクセス許可を受け取るユーザーまたはアカウントを指します。

以下のリソースポリシーの例は、前の一般的なポリシーのエレメントを示しています。このポリシーは、指定した*#####*で指定した *account-id* の API へのアクセスを、送信元 IP アドレスがアドレスブロック *123.4.5.6/24* にあるすべてのユーザーに許可します。このポリシーでは、ユーザーの送信元 IP がこの範囲外の場合、API へのアクセスはすべて拒否されます。

```
{
    "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 リソースポリシーが認証ワークフローに与える影響

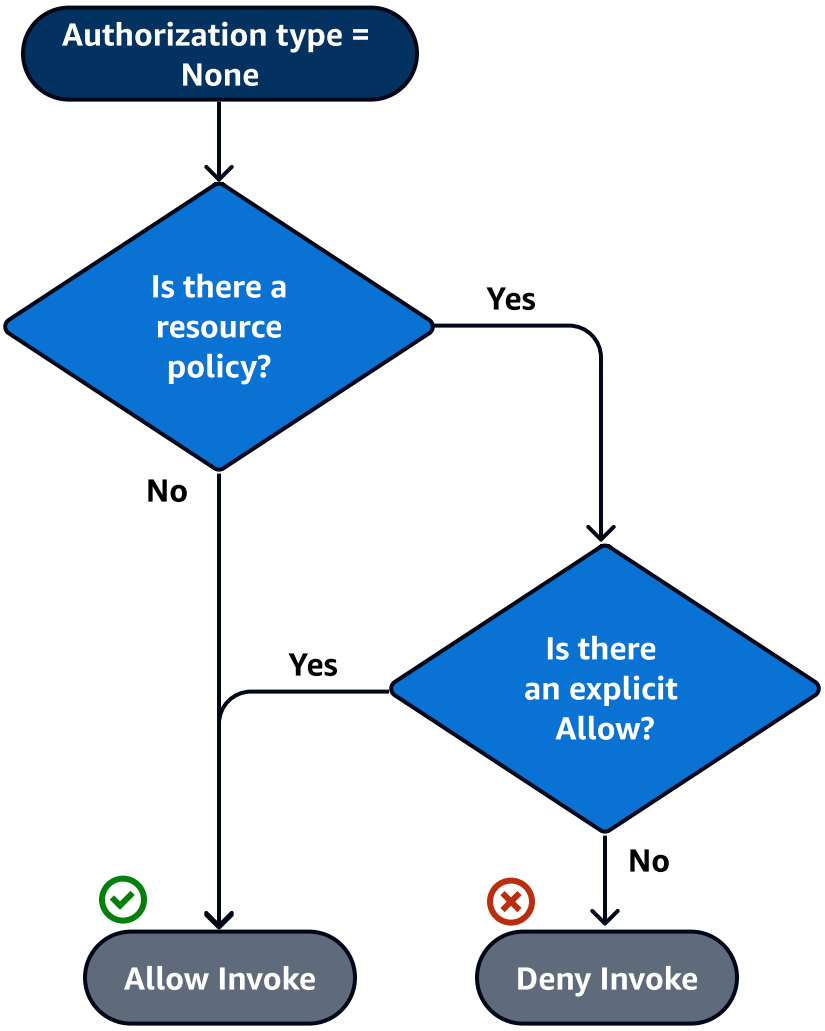
API にアタッチされたリソースポリシーを API Gateway が評価すると、以降のセクションのフローチャートに示すように、結果は API に対して定義した認証タイプによって影響を受けます。

トピック

- API Gateway リソースポリシーのみ
- Lambda オーソライザーとリソースポリシー
- IAM 認証とリソースポリシー
- Amazon Cognito 認証とリソースポリシー
- ポリシー評価の結果のテーブル

API Gateway リソースポリシーのみ

このワークフローでは、API Gateway リソースポリシーは API にアタッチされますが、認証タイプは API に対して定義されません。ポリシーの評価により、発信者のインバウンド条件に基づいて、明示的な許可の検索が呼び出されます。暗黙的な拒否または明示的な拒否により、発信者が拒否されます。



以下に、このようなリソースポリシーの例を示します。

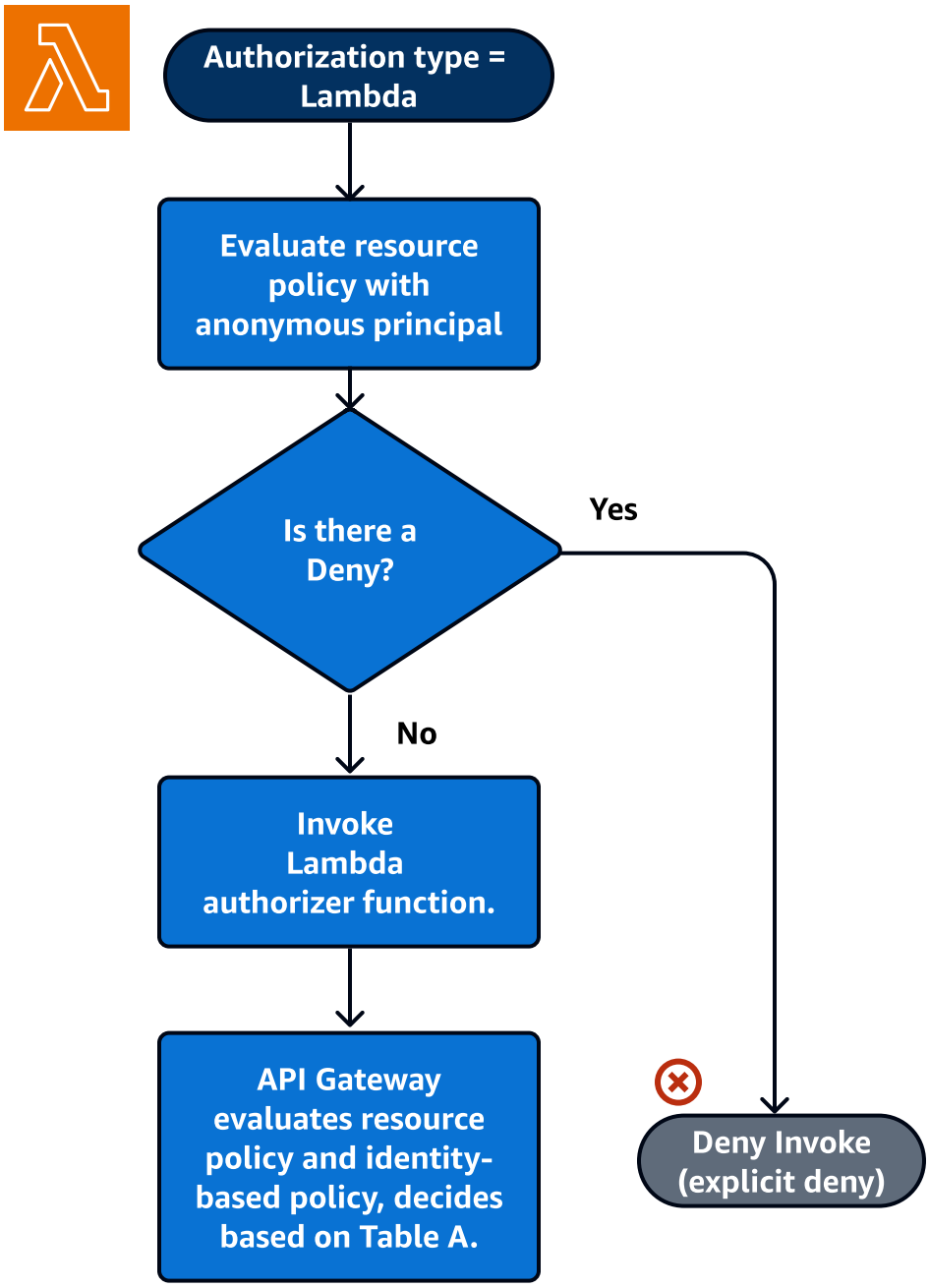
```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": "*",
            "Action": "execute-api:Invoke",
            "Resource": "arn:aws:execute-api:region:account-id:api-id/",
            "Condition": {
                "IpAddress": {
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

```
        }
    ]
}
```

Lambda オーソライザーとリソースポリシー

このワークフローでは、Lambda オーソライザーはリソースポリシーに加えて API に対して設定されます。リソースポリシーは 2 つの段階で評価されます。Lambda オーソライザーを呼び出す前に、API Gateway はまずポリシーを評価し、明示的な拒否をチェックします。見つかった場合、呼び出し元は即座にアクセスを拒否されます。それ以外の場合、Lambda オーソライザーが呼び出され、ポリシードキュメントを返します。これはリソースポリシーと一緒に評価されます。結果は、[テーブル A] に基づいて決定されます。

次のリソースポリシー例では、VPC エンドポイント ID が *vpce-1a2b3c4d* である VPC エンドポイントからのみ、呼び出しを許可します。認証前の評価中は、下例の VPC エンドポイントからの呼び出しのみが、Lambda オーソライザーを評価することを許可されます。残りの呼び出しはすべてブロックされます。




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

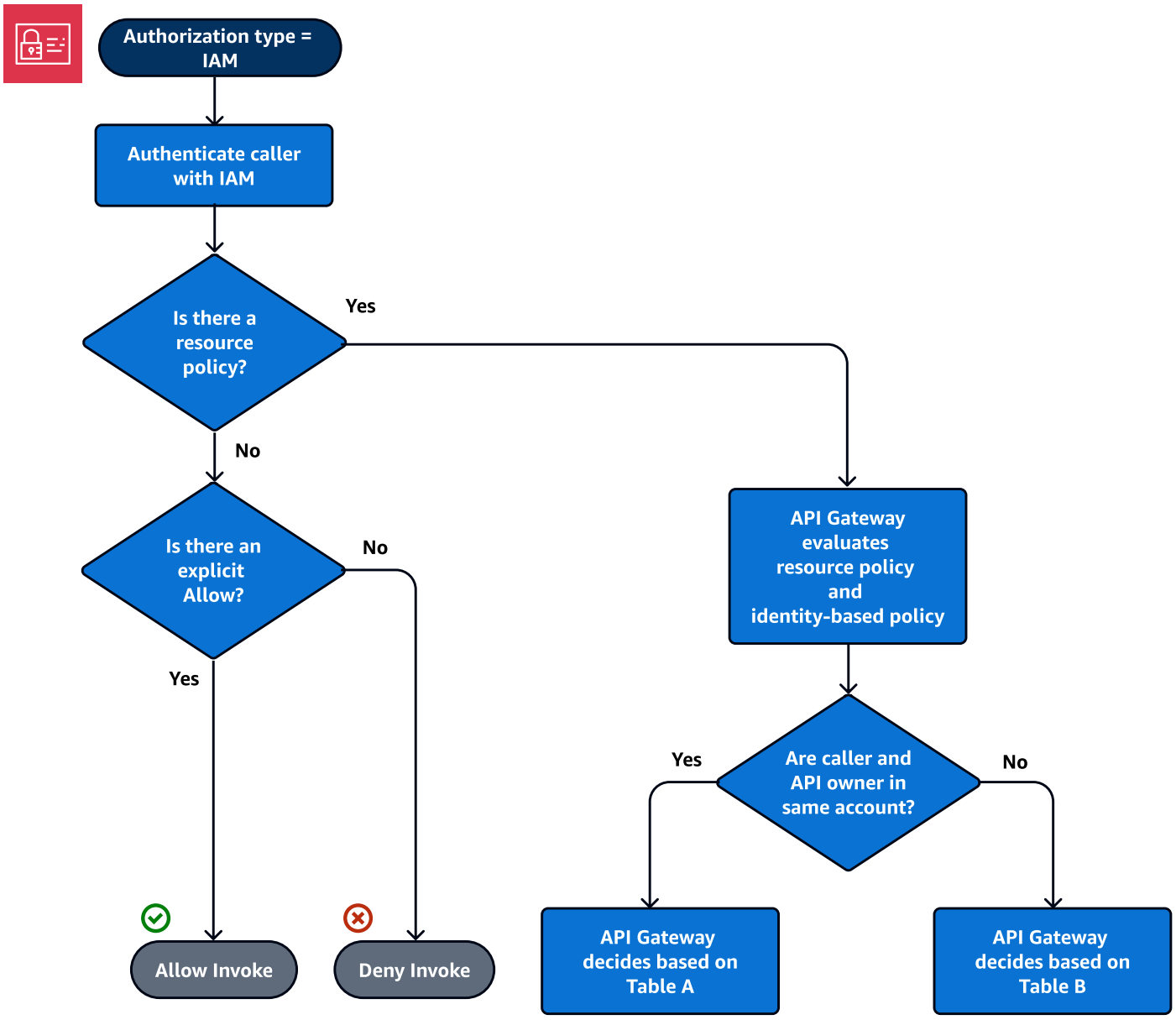
```
                "arn:aws:execute-api:region:account-id:api-id/"
            ],
            "Condition" : {
                "StringNotEquals": {
                    "aws:SourceVpce": "vpce-1a2b3c4d"
                }
            }
        }
    ]
}
```

IAM 認証とリソースポリシー

このワークフローでは、リソースポリシーに加えて API に対して IAM 認証を設定します。IAM サービスを使用してユーザーを認証した後、API は、ユーザーにアタッチされたポリシーとリソースポリシーの両方を評価します。この結果は、発信者が API 所有者と同じ AWS アカウント にいるか、API 所有者とは別の AWS アカウント にいるかに応じて異なります。

呼び出し元と API 所有者が別のアカウントである場合、IAM ポリシーとリソースポリシーの両方により、呼び出し元は明示的に続行が許可されます 詳細については、[テーブル B] を参照してください。

ただし、呼び出し元および API 所有者が同じ AWS アカウント である場合、IAM ユーザーポリシーまたはリソースポリシーで、呼び出し元の続行を明示的に許可する必要があります 詳細については、[テーブル A] を参照してください。



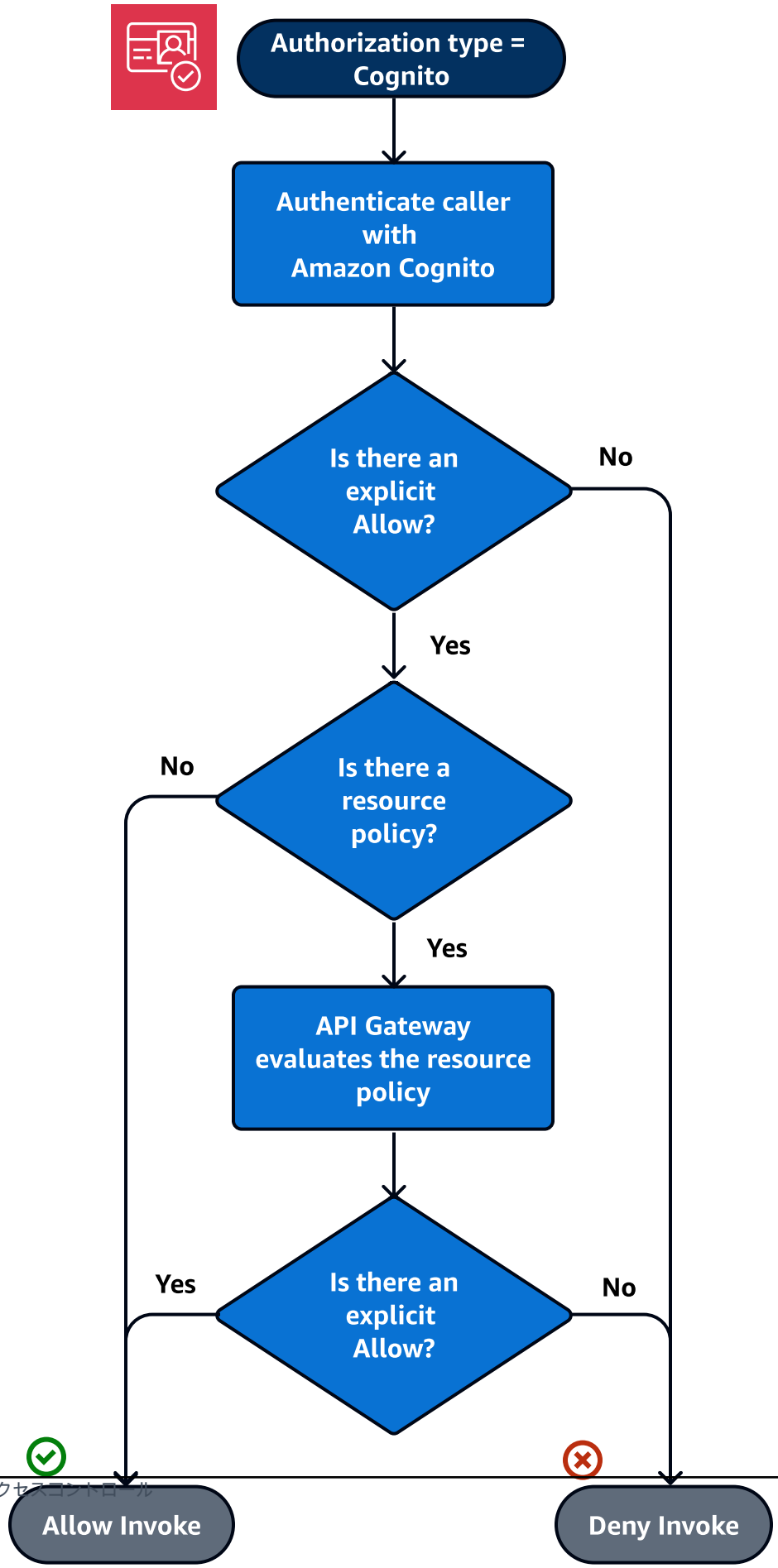
以下に、クロスアカウントリソースポリシーの例を示します。このリソースポリシーでは、IAM ポリシーに Allow Effect が含まれていることを想定し、VPC ID が *vpc-2f09a348* である VPC からの呼び出しのみが許可されます。詳細については、[テーブル B] を参照してください。

```
{
    "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 認証とリソースポリシー

このワークフローでは、リソースポリシーに加えて、API 用に Amazon Cognito ユーザープールが設定されます。API Gateway は、最初に Amazon Cognito を介して発信者の認証を試みます。これは通常、発信者から提供された JWT トークンを介して実行されます。認証が成功した場合、リソースポリシーは個別に評価され、明示的な許可が必要です。拒否の場合は拒否になり、「許可も拒否もしない」でも拒否になります。Amazon Cognito ユーザープールと一緒に使用される可能性のあるリソースポリシーの例を以下に示します。


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

次の例では、指定されたソース IP からのみ呼び出しを許可するリソースポリシーの例を示します。Amazon Cognito 認証トークンには許可が含まれていると想定します 詳細については、[テーブル B] を参照してください。

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

ポリシー評価の結果のテーブル

テーブル A に結果の動作が表示されるのは、API Gateway API へのアクセスが IAM ポリシーによって制御されているか、同じ AWS アカウント 内にある Lambda オーソライザーと API Gateway リソースポリシーによって制御されている場合です。

テーブル A: アカウント A が所有する API をアカウント A が呼び出す

IAM ポリシー (または Lambda オーソライザー)	API Gateway リソースポリシー	結果として生じる動作
許可	許可	許可
許可	許可も拒否もしない	許可
許可	拒否	明示的な拒否
許可も拒否もしない	許可	許可
許可も拒否もしない	許可も拒否もしない	暗黙的な拒否

IAM ポリシー (または Lambda オーソライザー)	API Gateway リソースポリシー	結果として生じる動作
許可も拒否もしない	拒否	明示的な拒否
拒否	許可	明示的な拒否
拒否	許可も拒否もしない	明示的な拒否
拒否	拒否	明示的な拒否

テーブル B に結果の動作が表示されるのは、API Gateway API へのアクセスが IAM ポリシーによって制御されているか、異なる AWS アカウント 内にある Amazon Cognito ユーザープールオーソライザーと API Gateway リソースポリシーによって制御されている場合です。どちらかがサイレントである (許可でも拒否でもない) 場合、クロスアカウントアクセスは拒否されます。これは、クロスアカウントアクセスでは、リソースポリシーと IAM ポリシーの両方、または Amazon Cognito ユーザープールオーソライザーが明示的にアクセス権を付与する必要があるためです。

テーブル B: アカウント A が所有する API をアカウント B が呼び出す

IAM ポリシー (または Amazon Cognito ユーザープールオーソライザー)	API Gateway リソースポリシー	結果として生じる動作
許可	許可	許可
許可	許可も拒否もしない	暗黙的な拒否
許可	拒否	明示的な拒否
許可も拒否もしない	許可	暗黙的な拒否
許可も拒否もしない	許可も拒否もしない	暗黙的な拒否
許可も拒否もしない	拒否	明示的な拒否
拒否	許可	明示的な拒否
拒否	許可も拒否もしない	明示的な拒否

IAM ポリシー (または Amazon Cognito ユーザープールオーソライザー)	API Gateway リソースポリシー	結果として生じる動作
拒否	拒否	明示的な拒否

API Gateway リソースポリシーの例

このページでは、API Gateway リソースポリシーの一般的なユースケースの例をいくつか紹介します。

以下のポリシー例では、API リソースの指定に略式構文を使用しています。この略式構文では、完全な Amazon リソースネーム (ARN) を指定する代わりに、簡略化された方法を使用して API リソースを参照できます。API Gateway がポリシーを保存するとき、簡略化された構文を完全な ARN に変換します。たとえば、リソースポリシー `execute-api:/stage-name/GET/pets` でリソースを指定できます。API Gateway がリソースポリシーを保存する際に、リソースを `arn:aws:execute-api:us-east-2:123456789012:aabbccddee/stage-name/GET/pets` に変換します。API Gateway は、現在のリージョン、AWS アカウント ID、およびリソースポリシーが関連付けられている REST API の ID を使用して、完全な ARN を構築します。`execute-api:/*` を使用して、現在の API のすべてのステージ、メソッド、パスを表すことができます。アクセスポリシー言語の詳細については、Amazon API Gateway のアクセスポリシー言語の概要 を参照してください。

トピック

- 例: 別の AWS アカウントのロールによる API の使用を許可する
- 例: 送信元の IP アドレスまたは IP アドレスの範囲に基づき、API トラフィックを拒否する
- 例: プライベート API の使用時に、送信元 IP アドレスまたは範囲に基づいて API トラフィックを拒否する
- 例: ソース VPC または VPC エンドポイントに基づいてプライベート API トラフィックを許可する

例: 別の AWS アカウントのロールによる API の使用を許可する

次のリソースポリシーの例は、Signature Version 4 (SigV4) プロトコルを使用して、1 つの AWS アカウントの API アクセス権を異なる AWS アカウントの 2 つのロールに付与します。具体的には、*account-id-2* によって識別される AWS アカウントのデベロッパーおよび管理者ロールでは、AWS アカウントの `pets` リソース (API) に対して `GET` アクションを実行する `execute-api:Invoke` アクションが許可されます。

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

例: 送信元の IP アドレスまたは IP アドレスの範囲に基づき、API トラフィックを拒否する

以下のリソースポリシーの例は、指定された 2 つの送信元 IP アドレスブロックから API への受信トラフィックを拒否 (ブロック) します。

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

例: プライベート API の使用時に、送信元 IP アドレスまたは範囲に基づいて API トラフィックを拒否する

次のリソースポリシーの例は、指定された 2 つの送信元 IP アドレスブロックからプライベート API への受信トラフィックを拒否 (ブロック) します。プライベート API を使用する場合、execute-api の VPC エンドポイントは元のソース IP アドレスを 書き換えます。aws:VpcSourceIp 条件は、元のリクエスタ IP アドレスに対してリクエストをフィルタリングします。

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

例: ソース VPC または VPC エンドポイントに基づいてプライベート API トラフィックを許可する

次のリソースポリシーの例では、指定された Virtual Private Cloud (VPC) または VPC エンドポイントからのみ、プライベート API への受信トラフィックを許可します。

このリソースポリシーの例では、ソース VPC を指定します。

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

このリソースポリシーの例では、ソース VPC エンドポイントを指定します。

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

API Gateway リソースポリシーを作成して API にアタッチする

API 実行サービスを呼び出して API にアクセスすることをユーザーに許可するには、API Gateway リソースポリシーを作成して API にアタッチする必要があります。ポリシーを API にアタッチすると、ポリシー内のアクセス許可が API のメソッドに適用されます。リソースポリシーを更新する場合は、API をデプロイする必要があります。

トピック

- 前提条件
- リソースポリシーを API Gateway API にアタッチする
- リソースポリシーのトラブルシューティング

前提条件

API Gateway リソースポリシーを更新するには、`apigateway:UpdateRestApiPolicy` アクセス許可と `apigateway:PATCH` アクセス許可が必要です。

エッジ最適化 API またはリージョン API の場合、リソースポリシーを作成時またはデプロイ後に API にアタッチできます。プライベート API の場合、リソースポリシーなしで API をデプロイする

ことはできません。詳細については、「the section called “プライベート REST API”」を参照してください。

リソースポリシーを API Gateway API にアタッチする

次の手順は、リソースポリシーを API Gateway API にアタッチする方法を示しています。

AWS Management Console

リソースポリシーを API Gateway API にアタッチするには

1. https://console.aws.amazon.com/apigateway で API Gateway コンソールにサインインします。
2. REST API を選択します。
3. 左のナビゲーションペインで、[リソースポリシー] を選択します。
4. [Create policy] を選択します。
5. (オプション) [テンプレートを選択] を選択してサンプルポリシーを生成します。

   ポリシーの例では、プレースホルダーは、二重波括弧 ("{{*placeholder*}}") で囲まれています。各プレースホルダー (波括弧を含む) を必要な情報に置き換えます。
6. どのテンプレート例も使用しない場合は、リソースポリシーを入力します。
7. [Save changes] (変更の保存) をクリックします。

API が API Gateway コンソールで以前にデプロイ済みの場合、リソースポリシーを有効にするには再デプロイする必要があります。

AWS CLI

AWS CLI を使用し、新しい API を作成してこれにリソースポリシーをアタッチするには、次のように create-rest-api コマンドを呼び出します。

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

AWS CLI を使用して、リソースポリシーを既存の API にアタッチするには、次のように update-rest-api コマンドを呼び出します。

```
aws apigateway update-rest-api \
```

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

AWS CloudFormation

AWS CloudFormation を使用して、リソースポリシーを持つ API を作成できます。次の例では、サンプルリソースポリシー the section called “例: 送信元の IP アドレスまたは IP アドレスの範囲に基づき、API トラフィックを拒否する” を使用して REST API を作成します。

```
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
```

リソースポリシーのトラブルシューティング

次のトラブルシューティングガイダンスは、リソースポリシーの問題を解決するのに役立ちます。

API が {"Message":"User: anonymous is not authorized to perform: execute-api:Invoke on resource: arn:aws:execute-api:us-east-1:********/****/****/"} を返します

リソースポリシーで、プリンシパルを次のように AWS プリンシパルに設定する場合:

```
{
    "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:/*"
            ]
        },
        ...
        }
```

API のすべてのメソッドに AWS_IAM 認証を使用する必要があります。使用しない場合、API は前のエラーメッセージを返します。メソッドの AWS_IAM 認証を有効にする方法の詳細については、「the section called “方法”」を参照してください。

リソースポリシーが更新されません

API 作成後にリソースポリシーを更新する場合は、更新後のポリシーをアタッチしてから API をデプロイし、変更を伝達する必要があります。ポリシーのみ、更新または保存した場合、API のランタイム動作が変更されることはありません。API のデプロイの詳細については、「the section called “REST API のデプロイ”」を参照してください。

API Gateway リソースポリシーで使用できる AWS の条件キー

以下のテーブルに、API Gateway の API のリソースポリシーで使用できる AWS 条件キーを認証タイプごとに示します。

AWS の条件キーの詳細については、「AWS グローバル条件コンテキストキー」を参照してください。

条件キーのテーブル

条件キー	条件	**AuthN** が必要ですか?	認証タイプ
aws:CurrentTime	なし	いいえ	すべて
aws:EpochTime	なし	いいえ	すべて
aws:TokenIssueTime	キーは、一時的セキュリティ認証情報を使用して署名されているリクエストにのみ存在します。	はい	IAM
aws:MultiFactorAuthPresent	キーは、一時的セキュリティ認証情報を使用して署名されているリクエストにのみ存在します。	はい	IAM
aws:MultiFactorAuthAge	キーは、MFA がリクエストに存在する	はい	IAM

条件キー	条件	AuthN が必要ですか?	認証タイプ
	場合にのみ存在します。		
aws:PrincipalAccount	なし	はい	IAM
aws:PrincipalArn	なし	はい	IAM
aws:PrincipalOrgID	このキーは、プリンシパルが組織のメンバーである場合にのみリクエストコンテキストに含まれます。	はい	IAM
aws:PrincipalOrgPaths	このキーは、プリンシパルが組織のメンバーである場合にのみリクエストコンテキストに含まれます。	はい	IAM
aws:PrincipalTag	このキーは、プリンシパルが、タグがアタッチされた IAM ユーザーである場合にリクエストコンテキストに含まれます。これは、タグまたはセッションタグがアタッチされた IAM ロールを使用するプリンシパルのために含まれます。	はい	IAM

条件キー	条件	**AuthN** が必要ですか?	認証タイプ
aws:Principa lType	なし	はい	IAM
aws:Referer	キーは、値が HTTP ヘッダーの呼び出し元によって渡されている場合にのみ存在します。	いいえ	すべて
aws:SecureTransport	なし	いいえ	すべて
aws:SourceArn	なし	いいえ	すべて
aws:SourceIp	なし	いいえ	すべて
aws:SourceVpc	このキーはプライベート API でのみ使用できます。	いいえ	すべて
aws:SourceVpce	このキーはプライベート API でのみ使用できます。	いいえ	すべて
aws:VpcSourceIp	このキーはプライベート API でのみ使用できます。	いいえ	すべて
aws:UserAgent	キーは、値が HTTP ヘッダーの呼び出し元によって渡されている場合にのみ存在します。	いいえ	すべて
aws:userid	なし	はい	IAM
aws:username	なし	はい	IAM

IAM アクセス許可により API へのアクセスを制御する

IAM アクセス許可 を使用して Amazon API Gateway API へのアクセスを制御するには、次の 2 つの API Gateway コンポーネントプロセスへのアクセスを制御します。

- API Gateway で API の作成、デプロイ、管理を行うには、API Gateway の API 管理コンポーネントでサポートされている必要なアクションを行うためのアクセス許可を API デベロッパーに付与する必要があります。
- デプロイされた API を呼び出したり、API キャッシュを更新したりするには、API Gateway の API 実行コンポーネントでサポートされている必要な IAM アクションを行うためのアクセス許可を API の発信者に付与する必要があります。

2 つのプロセスのアクセス制御には、以下で説明するように、異なるアクセス権限モデルが必要です。

API を作成および管理するための API Gateway アクセス許可モデル

API デベロッパーが API Gateway で API を作成および管理することを許可するには、指定した API デベロッパーが必要な API エントリを作成、更新、デプロイ、表示、または削除することを許可する IAM アクセス許可ポリシーを作成する必要があります。アクセス許可ポリシーをユーザー、ロール、またはグループにアタッチします。

アクセス権限を付与するには、ユーザー、グループ、またはロールにアクセス許可を追加します。

- AWS IAM Identity Center のユーザーとグループ:

  アクセス許可セットを作成します。「AWS IAM Identity Center ユーザーガイド」の「権限設定を作成する」の手順に従ってください。
- IAM 内で、ID プロバイダーによって管理されているユーザー:

  ID フェデレーションのロールを作成します。詳細については、「IAM ユーザーガイド」の「サードパーティー ID プロバイダー (フェデレーション) 用のロールの作成」を参照してください。
- IAM ユーザー:
  - ユーザーが担当できるロールを作成します。手順については、「IAM ユーザーガイド」の「IAM ユーザー用ロールの作成」を参照してください。
  - (お奨めできない方法) ポリシーをユーザーに直接アタッチするか、ユーザーをユーザーグループに追加する。詳細については、「IAM ユーザーガイド」の「ユーザー (コンソール) へのアクセス権限の追加」を参照してください。

アクセス権限モデルを使用する方法の詳細については、「the section called “API Gateway のアイデンティティベースのポリシー”」を参照してください。

API を呼び出すための API Gateway アクセス許可モデル

API の呼び出しや API キャッシュの更新を API 発信者に許可するには、ユーザー認証が有効な API メソッドを呼び出すことを、特定の API 発信者に許可する IAM ポリシーを作成する必要があります。API デベロッパーは、メソッドの `authorizationType` プロパティを `AWS_IAM` に設定して、発信者が認証対象のユーザーの認証情報を送信することを要求します。次に、ユーザー、グループ、またはロールにポリシーをアタッチします。

この IAM アクセス許可ポリシーのステートメントで、IAM `Resource`エレメントには、特定の HTTP 動詞および API Gateway リソースパスによって識別されるデプロイ API されたメソッドのリストが含まれます。IAM `Action` エレメントには、API Gateway の必要な API 実行アクションが含まれます。これらのアクションには、`execute-api:Invoke` や `execute-api:InvalidateCache` が含まれます。`execute-api` は、API Gateway の基盤となる API 実行コンポーネントを示します。

アクセス権限モデルを使用する方法の詳細については、「 API を呼び出すためのアクセスの制御」を参照してください。

API をバックエンドの AWS サービス (AWS Lambda など) と統合すると、API Gateway には、API 呼び出し元に代わって AWS の統合されたリソースにアクセスする (Lambda 関数を呼び出すなど) ための許可も必要です。これらのアクセス許可を付与するには、API Gateway 用の AWS のサービスタイプの IAM ロールを作成します。IAM 管理コンソールでこのロールを作成すると、API Gateway がこのロールを引き受けることを許可された信頼されたエンティティであることを宣言する以下の IAM 信頼ポリシーがロールに含まれます。

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

```
}
```

CLI の create-role コマンドまたは対応する SDK メソッドを呼び出して IAM ロールを作成する場合、上記の信頼ポリシーを `assume-role-policy-document` の入力パラメータとして指定する必要があります。このようなポリシーを、IAM マネジメントコンソールで直接作成、または AWS CLI create-policy コマンド、もしくは対応する SDK メソッドを呼び出して作成しないでください。

API Gateway で統合された AWS のサービスを呼び出すには、このロールに、統合された AWS のサービスを呼び出すための適切な IAM アクセス許可ポリシーもアタッチする必要があります。たとえば、Lambda 関数を呼び出すには、IAM ロールに次の IAM アクセス許可ポリシーを含める必要があります。

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

Lambda がサポートするリソースベースのアクセスポリシーは、信頼ポリシーとアクセス許可ポリシーの両方を組み合わせたものです。API Gateway コンソールを使用して API を Lambda 関数と統合すると、コンソールがユーザーに代わって (ユーザーの合意を得て) Lambda 関数でリソースベースのアクセス許可を設定するため、この IAM ロールを明示的に設定することは要求されません。

> **Note**
>
> AWS のサービスへのアクセス制御を有効にするには、発信者ベースのアクセス許可モデルを使用して、発信者のユーザーまたはグループにアクセス許可ポリシーを直接アタッチするか、ロールベースのアクセス許可モデルを使用して、API Gateway が引き受けることができる IAM ロールにアクセス許可ポリシーをアタッチすることができます。アクセス権限ポリシーは 2 つのモデル間で異なる場合があります。たとえば、呼び出し元ベースのポリシーはアクセスをブロックしますが、ロールベースのポリシーはアクセスを許可します。この差異を利用して、ユーザーが API Gateway API 経由でのみ AWS のサービスにアクセスするよう要求できます。

API を呼び出すためのアクセスの制御

このセクションでは、API Gateway でのデプロイされた API の呼び出しを誰に許可するかを制御するための IAM ポリシーステートメントを記述する方法について説明します。また、API 実行サービスに関連する `Action` フィールドと `Resource` フィールドの形式など、ポリシーステートメントのリファレンスも提供します。the section called “リソースポリシーが認証ワークフローに与える影響” の IAM セクションも学習する必要があります。

プライベート API の場合、API Gateway リソースポリシーと VPC エンドポイントポリシーを組み合わせて使用する必要があります。詳細については、次のトピックを参照してください。

- the section called “API Gateway リソースポリシーの使用”
- the section called “プライベート API 用の VPC エンドポイントポリシーを使用する”

IAM ポリシーを使用した API Gateway API メソッドの呼び出しを誰に許可するかを制御する

デプロイされた API を IAM のアクセス許可を使用して呼び出すことを誰に許可するか誰に許可しないかを制御するには、必要なアクセス許可を設定した IAM ポリシードキュメントを作成します。そのようなポリシードキュメントのテンプレートを以下に示します。

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

この場合、含まれているアクセス許可を付与または拒否するかどうかに応じて、*Permission* は `Allow` または `Deny` に置き換えられます。*Execution-operation* は、API 実行サービスでサポートされているオペレーションに置き換えられます。*METHOD_HTTP_VERB* は、指定されたリソースによってサポートされている HTTP 動詞を表します。*Resource-path* は、*METHOD_HTTP_VERB* を

サポートしている、デプロイされた API Resource インスタンスの URL パスのプレースホルダーです。詳細については、「API Gateway で API を実行するための IAM ポリシーのステートメントの参照」を参照してください。

> **Note**
>
> IAM ポリシーが有効になるには、API メソッドで IAM 認証を有効にしている必要があり、そのためには API メソッドの authorizationType プロパティとして AWS_IAM を設定します。認証を有効にしない場合、これらの API メソッドはパブリックにアクセス可能になります。

たとえば、指定された API で公開されているペットのリストを表示するユーザーアクセス許可を付与するものの、リストにペットを追加するユーザーアクセス許可を拒否する場合は、次のステートメントを IAM ポリシーに含めます。

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

GET /pets/*{petId}* として設定された API によって公開されている特定のペットを表示するユーザーアクセス許可を付与するには、IAM ポリシーに以下のステートメントを含めることができます。

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

API Gateway で API を実行するための IAM ポリシーのステートメントの参照

次の情報で API を実行するためのアクセス許可の IAM ポリシーステートメントのアクションおよびリソース形式について説明します。

API Gateway で API を実行するためのアクセス許可のアクション形式

API 実行の Action 式には、以下の一般的な形式があります。

```
execute-api:action
```

ここで、*action* は使用可能な API; 実行アクションです。

- *。以下のすべてのアクションを表します。
- [呼び出し] は、クライアントリクエストに応じて、API を呼び出すために使用されます。
- [InvalidateCache] は、クライアントリクエストに応じて、API キャッシュを無効にするために使用されます。

API Gateway で API を実行するためのアクセス許可のリソース形式

API 実行の Resource 式には、以下の一般的な形式があります。

```
arn:aws:execute-api:region:account-id:api-id/stage-name/HTTP-VERB/resource-path-specifier
```

各パラメータの意味は次のとおりです。

- *region* は、メソッドのデプロイされた API に対応する AWS リージョン (**us-east-1**、またはすべての * リージョンを表す AWS) です。
- *account-id* は、REST API 所有者の 12 桁の AWS アカウント ID です。
- *api-id* は、メソッドの API に割り当てられた識別子 API Gateway です。
- *stage-name* は、メソッドに関連付けられたステージの名前です。
- *HTTP-VERB* はメソッドの HTTP 動詞です。GET、POST、PUT、DELETE、PATCH のいずれかになります。
- *resource-path-specifier* は、目的のメソッドへのパスです

> **Note**
>
> ワイルドカード (*) を指定すると、Resource 式はそのワイルドカードを式の残りの部分に適用します。

以下に示しているのは、いくつかのリソース式の例です。

- **arn:aws:execute-api:*:*:***。任意の AWS リージョンの、任意の API の、任意のステージの、任意のリソースパスを表します。
- **arn:aws:execute-api:us-east-1:*:***。us-east-1 の AWS リージョンにおける、任意の API の、任意のステージの、任意のリソースパスを表します。
- 任意のステージの任意のリソースパス、AWS リージョンが us-east-1 で識別子が *api-id* の API には **arn:aws:execute-api:us-east-1:*:*api-id*/***。
- ステージ test のリソースパス、AWS リージョンが us-east-1 で識別子が *api-id* の API には **arn:aws:execute-api:us-east-1:*:*api-id*/test/***。

詳細については、「API Gateway Amazon リソースネーム (ARN) リファレンス」を参照してください。

API 実行アクセス許可の IAM ポリシーの例

アクセス権限モデルおよび他の背景情報については、「API を呼び出すためのアクセスの制御」を参照してください。

以下のポリシーステートメントでは、識別子が a123456789 の API で、ステージが test で、パスが mydemoresource の、任意の Post メソッドを呼び出すアクセス許可をユーザーに付与します。ただし、該当する API が AWS リージョン us-east-1 にデプロイされているとします。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "execute-api:Invoke"
      ],
      "Resource": [
        "arn:aws:execute-api:us-east-1:*:a123456789/test/POST/mydemoresource/*"
      ]
    }
  ]
}
```

以下のポリシーステートメントの例では、任意の petstorewalkthrough/pets リージョンの、識別子が a123456789 の API の、任意のステージの、リソースパスが AWS の、任意のメソッドを呼び出すアクセス権限をユーザーに付与します。ただし、該当する API がそのリージョンにデプロイされているとします。

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

```
}
```

ポリシーを作成してユーザーにアタッチする

ユーザーが API 管理サービスまたは API 実行サービスを呼び出せるようにするには、API Gateway エンティティへのアクセスを制御する IAM ポリシーを作成する必要があります。

JSON ポリシーエディタでポリシーを作成するには

1. AWS Management Console にサインインして、IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. 左側のナビゲーションペインで、[ポリシー] を選択します。

   初めて [ポリシー] を選択する場合には、[管理ポリシーにようこそ] ページが表示されます。[今すぐ始める] を選択します。
3. ページの上部で、[ポリシーを作成] を選択します。
4. [ポリシーエディタ] セクションで、[JSON] オプションを選択します。
5. 次の JSON ポリシードキュメントを入力します。

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

```
}
```

6. [次へ] をクリックします。

> **Note**
>
> いつでも [Visual] と [JSON] エディタオプションを切り替えることができます。ただし、[Visual] エディタで [次] に変更または選択した場合、IAM はポリシーを再構成して visual エディタに合わせて最適化することがあります。詳細については、「IAM ユーザーガイド」の「ポリシーの再構成」を参照してください。

7. [確認と作成] ページで、作成するポリシーの [ポリシー名] と [説明] (オプション) を入力します。[このポリシーで定義されているアクセス許可] を確認して、ポリシーによって付与されたアクセス許可を確認します。
8. [ポリシーの作成] をクリックして、新しいポリシーを保存します。

このステートメントで、必要に応じて *action-statement* と *resource-statement* を置き換えます。また、ユーザーに管理を許可する API Gateway エンティティ、ユーザーが呼び出せる API メソッド、または両方を指定するための他のステートメントを追加します。デフォルトでは、対応する明示的な `Allow` ステートメントがない限り、 ユーザーにアクセス許可はありません。

以上で、IAM ポリシーを作成しました。アタッチするまで効果はありません。

アクセス権限を付与するには、ユーザー、グループ、またはロールにアクセス許可を追加します。

- AWS IAM Identity Center のユーザーとグループ:

  アクセス許可セットを作成します。「AWS IAM Identity Center ユーザーガイド」の「権限設定を作成する」の手順に従ってください。
- IAM 内で、ID プロバイダーによって管理されているユーザー:

  ID フェデレーションのロールを作成します。詳細については、「IAM ユーザーガイド」の「サードパーティー ID プロバイダー (フェデレーション) 用のロールの作成」を参照してください。
- IAM ユーザー:
  - ユーザーが担当できるロールを作成します。手順については、「IAM ユーザーガイド」の「IAM ユーザー用ロールの作成」を参照してください。

- (お奨めできない方法) ポリシーをユーザーに直接アタッチするか、ユーザーをユーザーグループに追加する。詳細については、「IAM ユーザーガイド」の「ユーザー (コンソール) へのアクセス権限の追加」を参照してください。

IAM ポリシードキュメントを IAM グループにアタッチするには

1. メインのナビゲーションペインで、[グループ] を選択します。
2. 選択したグループの下で、[Permissions (アクセス許可)] タブを選択します。
3. [Attach policy] を選択します。
4. 前に作成したポリシードキュメントを選択して、[ポリシーのアタッチ] を選択します。

API Gateway がユーザーに代わって AWS の他のサービスを呼び出すには、Amazon API Gateway タイプの IAM ロールを作成します。

Amazon API Gateway タイプのロールを作成するには

1. メインのナビゲーションペインで、[ロール] を選択します。
2. [Create New Role] を選択します。
3. [ロール名] にロールの名前を入力し、[次のステップ] を選択します。
4. [AWS Service Roles] (AWS のサービスロール) の [Select Role Type] (ロールタイプの選択) で、[Amazon API Gateway] の横にある [Select] (選択) を選択します。
5. API Gateway で CloudWatch にメトリクスを記録する場合は、[ポリシーのアタッチ] で、使用可能な IAM 管理対象アクセス許可ポリシー ([AmazonAPIGatewayPushToCloudWatchLog] など) を選択し、[次のステップ] を選択します。
6. [信頼されたエンティティ] で、[apigateway.amazonaws.com] がエントリとして表示されていることを確認し、[ロールの作成] を選択します。
7. 新しく作成したロールで、[Permissions (アクセス許可)] タブ、[ポリシーのアタッチ] の順に選択します。
8. 前に作成したカスタムの IAM ポリシードキュメントを選択し、[ポリシーのアタッチ] を選択します。

API Gateway でプライベート API 用の VPC エンドポイントポリシーを使用する

プライベート API のセキュリティを向上させるには、VPC エンドポイントポリシーを作成できます。VPC エンドポイントポリシーは、VPC エンドポイントにアタッチする IAM リソースポリシーで

す。詳細については、「VPC エンドポイントによるサービスのアクセスコントロール」を参照してください。

VPC エンドポイントポリシーを作成して、以下のことを行うことができます。

- 特定の組織やリソースにのみ、VPC エンドポイントにアクセスして API を呼び出すことを許可します。
- API へのトラフィックを制御するために、セッションベースやロールベースのポリシーは使用しないで、単一のポリシーを使用します。
- オンプレミスから AWS に移行する際に、アプリケーションのセキュリティ境界を厳しくします。

VPC エンドポイントのポリシーに関する注意事項

- 呼び出し元の ID は、`Authorization` ヘッダー値に基づいて評価されます。`authorizationType` によっては、これによって `403 IncompleteSignatureException` または `403 InvalidSignatureException` エラーが発生する可能性があります。次の表は、各 `authorizationType` の `Authorization` ヘッダー値を示しています。

authorizationType	**Authorization** ヘッダーは評価されましたか?	許可される **Authorization** ヘッダー値
デフォルトでフルアクセスポリシーを持つ NONE	いいえ	不合格
カスタムアクセスポリシーを持つ NONE	はい	有効な SigV4 値である必要があります
IAM	はい	有効な SigV4 値である必要があります
CUSTOM または COGNITO_USER_POOLS	いいえ	不合格

- 特定の IAM プリンシパルへのアクセスをポリシーで制限する場合は (`arn:aws:iam::account-id:role/developer` など)、API のメソッドの `authorizationType` を `AWS_IAM` または `NONE` に設定する必要があります。メソッドの `authorizationType` を設定する方法の詳細については、「the section called “方法”」を参照してください。

- VPC エンドポイントポリシーは、API Gateway リソースポリシーと一緒に使用できます。API Gateway リソースポリシーは、どのプリンシパルが API にアクセスできるかを指定します。エンドポイントポリシーは、誰が VPC にアクセスできるか、どの API を VPC エンドポイントから呼び出せるかを指定します。プライベート API にはリソースポリシーが必要ですが、カスタム VPC エンドポイントポリシーを作成する必要はありません。

VPC エンドポイントのポリシーの例

Amazon API Gateway 用の Amazon Virtual Private Cloud クラウドエンドポイントのポリシーを作成できます。ポリシーでは、以下を指定できます。

- アクションを実行できるプリンシパル。
- 実行可能なアクション。
- 自身に対してアクションを実行できたリソース。

VPC エンドポイントにポリシーをアタッチするには 、VPC コンソールを使用する必要があります。詳細については、「VPC エンドポイントによるサービスのアクセスコントロール」を参照してください。

例 1: 2 つの API へのアクセスを許可する VPC エンドポイントポリシー

次のポリシー例では、ポリシーが関連付けられている VPC エンドポイントを経由して 2 つの特定の API へのアクセスを許可しています。

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

例 2: GET メソッドへのアクセスを許可する VPC エンドポイントポリシー

次のポリシー例では、ポリシーが関連付けられている VPC エンドポイントを経由して特定の API のGET メソッドへのアクセスを許可しています。

```
{
    "Statement": [
        {
            "Principal": "*",
            "Action": [
                "execute-api:Invoke"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:execute-api:us-east-1:123412341234:a1b2c3d4e5/stageName/GET/*"
            ]
        }
    ]
}
```

例 3: 特定の API への特定のユーザーアクセスを許可する VPC エンドポイントポリシー

次のポリシー例では、ポリシーが関連付けられている VPC エンドポイントを経由して特定の API への特定のユーザーアクセスを許可しています。

この場合、特定の IAM プリンシパルへのアクセスをポリシーで制限するため、メソッドの authorizationType を AWS_IAM または NONE に設定する必要があります。

```
{
    "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/*"
```

```
            ]
        }
    ]
}
```

API Gateway でタグを使用して REST API へのアクセスをコントロールする

REST API に対する許可は、IAM ポリシーの属性ベースのアクセスコントロールを使用して微調整できます。

詳細については、「the section called “単一ドメイン内の属性ベースの”」を参照してください。

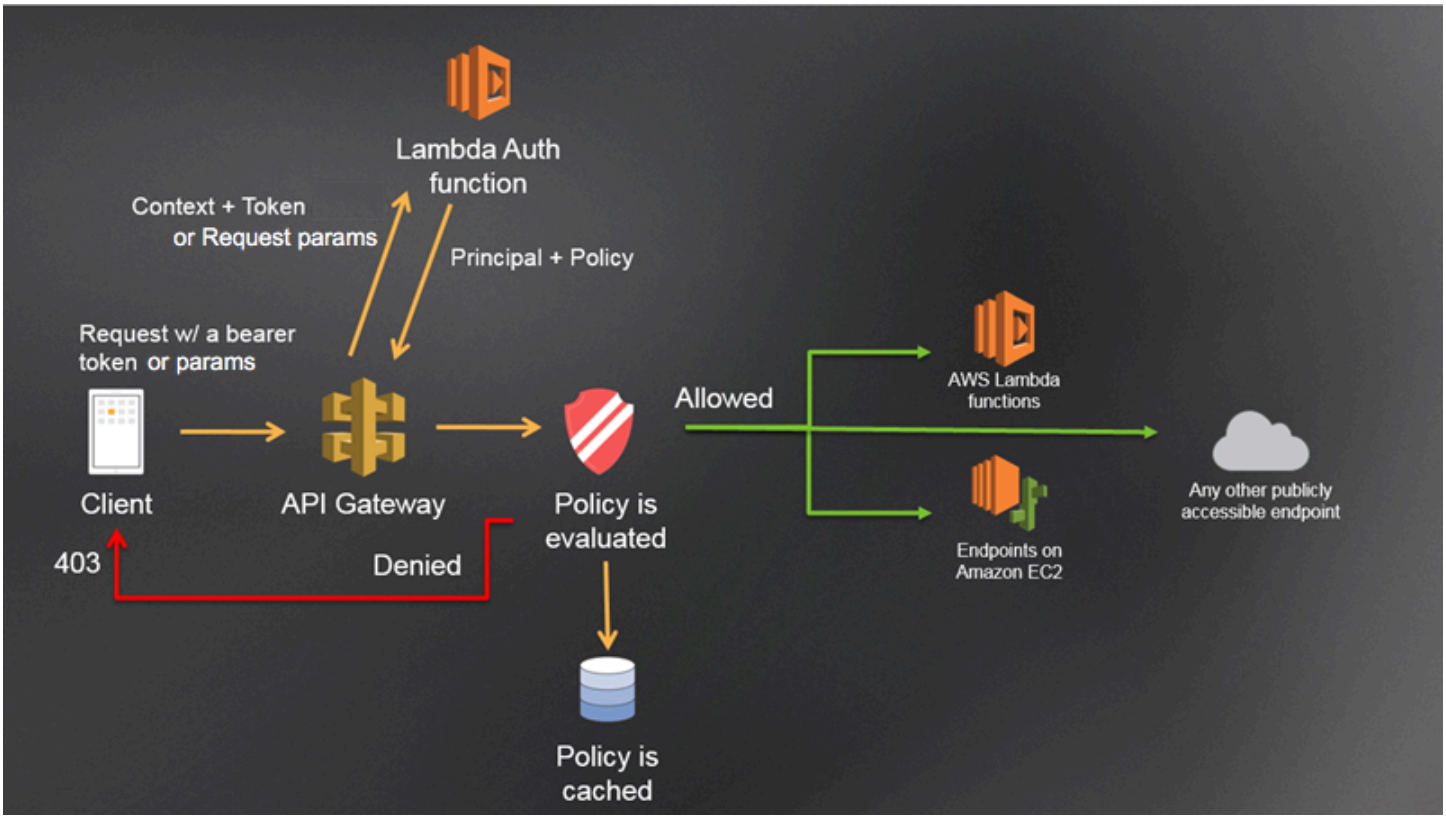
API Gateway Lambda オーソライザーを使用する

Lambda オーソライザー (以前はカスタムオーソライザーと呼ばれていました) は、API へのアクセスを制御するために使用します。クライアントが API の メソッドをリクエストすると、API Gateway は Lambda オーソライザーを呼び出します。Lambda オーソライザーは、発信者の ID を入力として受け取り、IAM ポリシーを出力として返します。

Lambda オーソライザーを使用して、カスタム認証スキームを実装します。スキームでは、リクエストパラメータを使用して、発信者のアイデンティティを判断したり、OAuth や SAML などのベアラートークン認証戦略を使用したりできます。Lambda オーソライザーは、API Gateway REST API コンソール、AWS CLI、または AWS SDK を使用して作成します。

Lambda オーソライザーの認証ワークフロー

次の図は、Lambda オーソライザーの認証ワークフローを示しています。



API Gateway Lambda 認証ワークフロー

1. クライアントは、API Gateway API でメソッドを呼び出し、ベアラートークンまたはリクエストパラメータを渡します。
2. API Gateway は、メソッドリクエストが Lambda オーソライザーで設定されているかどうかを確認します。存在する場合、API Gateway は Lambda 関数を呼び出します。
3. Lambda 関数は発信者を認証します。関数は次の方法で認証できます。
   - OAuth プロバイダーを呼び出して OAuth アクセストークンを取得します。
   - SAML プロバイダーを呼び出して SAML アサーションを取得します。
   - リクエストパラメータ値に基づいて IAM ポリシーを生成します。
   - データベースから認証情報を取得します。
4. Lambda 関数は、IAM ポリシーとプリンシパル識別子を返します。Lambda 関数がこの情報を返さない場合、呼び出しは失敗します。
5. API Gateway は IAM ポリシーを評価します。
   - アクセスが拒否された場合、API Gateway は `403 ACCESS_DENIED` などの適切な HTTP ステータスコードを返します。

- アクセスが許可されている場合、API Gateway はメソッドを呼び出します。

  認可のキャッシュを有効にすると、API Gateway でポリシーがキャッシュされるため、Lambda オーソライザー関数は再度呼び出されません。

`403 ACCESS_DENIED` または `401 UNAUTHORIZED` ゲートウェイレスポンスはカスタマイズできます。詳細については、「the section called “ゲートウェイレスポンス”」を参照してください。

Lambda オーソライザーのタイプの選択

Lambda オーソライザーには 2 種類あります。

リクエストパラメータベースの Lambda オーソライザー (**`REQUEST`** オーソライザー)

`REQUEST` オーソライザーは、発信者 ID をヘッダー、クエリ文字列パラメータ、`stageVariables`、`$context` 変数の組み合わせとして受け取ります。`REQUEST` オーソライザーを使用して、`$context.path` や `$context.httpMethod` コンテキスト変数など、複数の ID ソースからの情報に基づいてきめ細かなポリシーを作成できます。

`REQUEST` オーソライザーの認可のキャッシュを有効にすると、API Gateway は、指定されたすべての ID ソースがリクエスト内に存在することを確認します。指定された ID ソースが欠落しているか、null または空である場合、API Gateway は、Lambda オーソライザー関数を呼び出すことなく `401 Unauthorized` HTTP レスポンスを返します。複数の ID ソースが定義されている場合は、すべての ID ソースがオーソライザーのキャッシュキーを取得するために使用されます (順序は保持されます)。複数の ID ソースを使用して、きめ細かなキャッシュキーを定義できます。

キャッシュキーのいずれかの部分を変更して API を再デプロイすると、オーソライザーは、キャッシュされたポリシードキュメントを破棄して新しいドキュメントを作成します。

`REQUEST` オーソライザーの認可のキャッシュを無効にすると、API Gateway はリクエストを Lambda 関数に直接渡します。

トークンベースの Lambda オーソライザー (**`TOKEN`** オーソライザー)

`TOKEN` オーソライザーは、JSON ウェブトークン (JWT) や OAuth トークンなどのベアラートークンで発信者 ID を受け取ります。

`TOKEN` オーソライザーの認可のキャッシュを有効にすると、トークンソースに指定されているヘッダー名がキャッシュキーになります。

さらに、トークン検証を使用して RegEx ステートメントを入力できます。API Gateway は、この式に対して入力トークンの初期検証を実行し、検証が成功すると Lambda オーソライザー関数を呼び出します。これにより API への呼び出しを減らすことができます。

`IdentityValidationExpression` プロパティは `TOKEN` オーソライザーでのみサポートされています。詳細については、「the section called “x-amazon-apigateway-authorizer”」を参照してください。

> **Note**
>
> `REQUEST` オーソライザーを使用して API へのアクセスを制御することをお勧めします。`REQUEST` オーソライザーを使用すると、(`TOKEN` オーソライザーを使用した場合の単一の ID ソースと比べて) 複数の ID ソースに基づく API へのアクセスを制御できます。さらに、`REQUEST` オーソライザーでは複数の ID ソースを使用してキャッシュキーを分離できます。

`REQUEST` オーソライザー Lambda 関数の例

次のコード例で作成する Lambda オーソライザー関数では、クライアントが提供した `HeaderAuth1` ヘッダー、`QueryString1` クエリパラメータ、ステージ変数 `StageVar1` のすべてが、それぞれ `headerValue1`、`queryValue1`、`stageValue1` の指定された値と一致する場合に、リクエストを許可します。

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)
```

この例では、Lambda オーソライザー関数は入力パラメータをチェックして次のように動作します。

- 必要なすべてのパラメータ値が予想値と一致する場合、オーソライザー関数は `200 OK` HTTP レスポンスと次のような IAM ポリシーを返し、メソッドリクエストは成功します。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": "execute-api:Invoke",
      "Effect": "Allow",
      "Resource": "arn:aws:execute-api:us-east-1:123456789012:ivdtdhp7b5/
ESTestInvoke-stage/GET/"
    }
  ]
}
```

- 一致しない場合、オーソライザー関数は `401 Unauthorized` HTTP レスポンスを返し、メソッドリクエストは失敗します。

Lambda オーソライザー関数は、IAM ポリシーだけでなく、発信者のプリンシパル ID を返す必要があります。オプションで、`context` オブジェクトを返すこともできます。このオブジェクトには、

統合バックエンドに渡すことができる追加情報が含まれています。詳細については、「API Gateway Lambda オーソライザーからの出力」を参照してください。

本番稼働コードでは、権限を付与する前にユーザーの認証が必要になる場合があります。この場合、認証プロバイダーを (関連ドキュメントの指示に従って) 呼び出すことで、Lambda 関数に認証ロジックを追加できます。

TOKEN オーソライザー Lambda 関数の例

次のコード例で作成する TOKEN Lambda オーソライザー関数では、クライアントが提供したトークン値が allow の場合に、メソッドを呼び出すことを発信者に許可します。トークン値が deny の場合、発信者はリクエストを呼び出すことを許可されません。トークン値が unauthorized または空の文字列の場合、オーソライザー関数は 401 UNAUTHORIZED レスポンスを返します。

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
```

この例では、API がメソッドリクエストを受信すると、API Gateway は `event.authorizationToken` 属性でこの Lambda オーソライザー関数にソーストークンを渡します。Lambda オーソライザー関数はトークンを読み取り、次のように動作します。

- トークン値が `allow` の場合、オーソライザー関数は `200 OK` HTTP レスポンスと次のような IAM ポリシーを返し、メソッドリクエストは成功します。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": "execute-api:Invoke",
      "Effect": "Allow",
      "Resource": "arn:aws:execute-api:us-east-1:123456789012:ivdtdhp7b5/
ESTestInvoke-stage/GET/"
    }
  ]
}
```

- トークン値が `deny` の場合、オーソライザー関数は `200 OK` HTTP レスポンスと次のような `Deny` IAM ポリシーを返し、メソッドリクエストは失敗します。

```
{
  "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**
>
> テスト環境以外では、API Gateway は `403 Forbidden` HTTP レスポンスを返し、メソッドリクエストは失敗します。

- トークン値が `unauthorized` または空の文字列の場合、オーソライザー関数は `401 Unauthorized` HTTP レスポンスを返し、メソッド呼び出しは失敗します。

- それ以外のトークンの場合、クライアントは `500 Invalid token` レスポンスを受け取り、メソッド呼び出しは失敗します。

Lambda オーソライザー関数は、IAM ポリシーだけでなく、発信者のプリンシパル ID を返す必要があります。オプションで、`context` オブジェクトを返すこともできます。このオブジェクトには、統合バックエンドに渡すことができる追加情報が含まれています。詳細については、「API Gateway Lambda オーソライザーからの出力」を参照してください。

本番稼働コードでは、権限を付与する前にユーザーの認証が必要になる場合があります。この場合、認証プロバイダーを (関連ドキュメントの指示に従って) 呼び出すことで、Lambda 関数に認証ロジックを追加できます。

その他の Lambda オーソライザー関数の例

次のリストは、その他の Lambda オーソライザー関数の例を示しています。Lambda 関数は、API を作成したのと同じアカウントまたは別のアカウントで作成できます。

前の例に示した Lambda 関数の場合は、他の AWS のサービスを呼び出さないため、組み込みの AWSLambdaBasicExecutionRole を使用できます。Lambda 関数から他の AWS のサービスを呼び出す場合は、Lambda 関数に IAM 実行ロールを割り当てる必要があります。ロールを作成するには、「AWS Lambda 実行ロール」の手順に従ってください。

その他の Lambda オーソライザー関数の例

- サンプルアプリケーションについては、GitHub の「Open Banking Brazil - Authorization Samples」を参照してください。
- その他のサンプルの Lambda 関数については、GitHub の「aws-apigateway-lambda-authorizer-blueprints」を参照してください。
- Amazon Cognito ユーザープールを使用してユーザーを認証するとともに Verified Permissions を使用してポリシーストアに基づいて発信者を認証する Lambda オーソライザーを作成できます。詳細については、「Amazon Verified Permissions ユーザーガイド」の「接続された API と ID プロバイダーを使用してポリシーストアを作成する」を参照してください。
- Lambda コンソールには Python の設計図が用意されています。[(設計図を使用する] を選択して [api-gateway-authorizer-python] 設計図を選択すると使用できます。

Lambda オーソライザーを設定する

Lambda 関数を作成したら、Lambda 関数を API のオーソライザーとして設定します。次に、Lambda オーソライザーを呼び出すようにメソッドを設定し、発信者がメソッドを呼び出せるかどうかを判断します。Lambda 関数は、API を作成したのと同じアカウントまたは別のアカウントで作成できます。

API Gateway コンソールの組み込みツールまたは Postman を使用して、Lambda オーソライザーをテストできます。Postman を使用して Lambda オーソライザー関数をテストする方法については、「the section called “Lambda オーソライザーで API を呼び出す”」を参照してください。

Lambda オーソライザーを設定する (コンソール)

次の手順は、API Gateway REST API コンソールで Lambda オーソライザーを作成する方法を示しています。Lambda オーソライザーのタイプ別の詳細については、「the section called “Lambda オーソライザーのタイプの選択”」を参照してください。

REQUEST authorizer

Lambda オーソライザー **REQUEST** を設定するには

1. https://console.aws.amazon.com/apigateway で API Gateway コンソールにサインインします。
2. API を選択し、[オーソライザー] を選択します。
3. [オーソライザーの作成] を選択します。
4. [オーソライザー名] で、オーソライザー名を入力します。
5. [オーソライザータイプ] には Lambda を選択します。
6. [Lambda 関数] で、Lambda オーソライザー関数を作成した場所 AWS リージョン を選択し、関数名を入力します。
7. [Lambda 呼び出しロール] は空白のままにして、API Gateway REST API コンソールでリソースベースのポリシーを設定できるようにします。このポリシーは、Lambda オーソライザー関数を呼び出すアクセス許可を API Gateway に付与します。IAM ロールの名前を入力して、Lambda オーソライザー関数を呼び出すことを API Gateway に許可することもできます。ロールの例については、「引き受け可能な IAM ロールを作成する」を参照してください。
8. [Lambda イベントペイロード] の場合は、[リクエスト] を選択します。

9. [ID ソースタイプ] では、パラメータータイプを選択します。サポートされているパラメータタイプは、`Header`、`Query string`、`Stage variable`、および `Context` です。ID ソースをさらに追加するには、[パラメーターの追加] を選択します。
10. オーソライザーが生成した認証ポリシーをキャッシュするには、[認証キャッシュ] をオンのままにします。ポリシーのキャッシュを有効にすると、[TTL] 値を変更できます。[TTL] を 0 に設定すると、ポリシーのキャッシュは無効になります。

    キャッシュを有効にすると、オーソライザーは、API 全体のすべてのメソッドに適用されるポリシーを返す必要があります。メソッド固有のポリシーを適用するには、コンテキスト変数の `$context.path` および `$context.httpMethod` を使用します。
11. [オーソライザーの作成] を選択します。

TOKEN authorizer

`TOKEN` Lambda オーソライザーを設定するには

1. https://console.aws.amazon.com/apigateway で API Gateway コンソールにサインインします。
2. API を選択し、[オーソライザー] を選択します。
3. [オーソライザーの作成] を選択します。
4. [オーソライザー名] で、オーソライザー名を入力します。
5. [オーソライザータイプ] には Lambda を選択します。
6. [Lambda 関数] で、Lambda オーソライザー関数を作成した場所 AWS リージョン を選択し、関数名を入力します。
7. [Lambda 呼び出しロール] は空白のままにして、API Gateway REST API コンソールでリソースベースのポリシーを設定できるようにします。このポリシーは、Lambda オーソライザー関数を呼び出すアクセス許可を API Gateway に付与します。IAM ロールの名前を入力して、Lambda オーソライザー関数を呼び出すことを API Gateway に許可することもできます。ロールの例については、「引き受け可能な IAM ロールを作成する」を参照してください。
8. [Lambda イベントペイロード] の場合は、[トークン] を選択します。
9. [トークンソース] には、認証トークンを含むヘッダー名を入力します。認証トークンを Lambda オーソライザーに送信するには、発信者がこの名前のヘッダーが含める必要があります。

10. (オプション) [トークンの検証] に、RegEx ステートメントを入力します。API Gateway は、この式に対して、入力トークンの初期検証を実行し、認証が成功するとオーソライザーを呼び出します。
11. オーソライザーが生成した認証ポリシーをキャッシュするには、[認証キャッシュ] をオンのままにします。ポリシーのキャッシュが有効の場合、[トークンのソース] で指定されているヘッダー名はキャッシュキーになります。ポリシーのキャッシュを有効にすると、[TTL] 値を変更できます。[TTL] を 0 に設定すると、ポリシーのキャッシュは無効になります。

    キャッシュを有効にすると、オーソライザーは、API 全体のすべてのメソッドに適用されるポリシーを返す必要があります。メソッド固有のポリシーを適用するには、[認可のキャッシュ] をオフにすることができます。
12. [オーソライザーの作成] を選択します。

Lambda オーソライザーを作成したら、これをテストできます。次の手順は、Lambda オーソライザーをテストする方法を示しています。

REQUEST authorizer

REQUEST Lambda オーソライザーをテストするには

1. https://console.aws.amazon.com/apigateway で API Gateway コンソールにサインインします。
2. オーソライザーの名前を選択します。
3. [オーソライザーをテスト] に、ID ソースの値を入力します。

    the section called “REQUEST オーソライザー Lambda 関数の例” を使用している場合は、次の手順に従います。

    a. [ヘッダー] を選択して「**headerValue1**」と入力し、[パラメーターの追加] を選択します。
    b. [ID ソースタイプ] で [クエリ文字列] を選択して「**queryValue1**」と入力し、パ[ラメータの追加] を選択します。
    c. [ID ソースタイプ] で、[ステージ変数] を選択し、「**stageValue1**」と入力します。

    テスト呼び出しのコンテキスト変数は変更できませんが、Lambda 関数の API Gateway オーソライザーのテストイベントテンプレートは変更できます。次に、変更したコンテキスト変数を使用して Lambda オーソライザー関数をテストできます。詳細については、「AWS

Lambda デベロッパーガイド」の「コンソールでの Lambda 関数のテスト」を参照してください。

4. [オーソライザーをテスト] を選択します。

TOKEN authorizer

`TOKEN` Lambda オーソライザーをテストするには

1. https://console.aws.amazon.com/apigateway で API Gateway コンソールにサインインします。
2. オーソライザーの名前を選択します。
3. [オーソライザーをテスト] に、トークンの値を入力します。

   the section called “TOKEN オーソライザー Lambda 関数の例” を使用している場合は、次の手順に従います。

   - [authorizationToken] に、「**`allow`**」と入力します。
4. [オーソライザーをテスト] を選択します。

   Lambda オーソライザーがテスト環境でリクエストを正常に拒否すると、テストは `200 OK` HTTP レスポンスで応答します。ただし、テスト環境以外では、API Gateway は `403 Forbidden` HTTP レスポンスを返し、メソッドリクエストは失敗します。

Lambda オーソライザーを設定する (AWS CLI)

次の create-authorizer コマンドは、AWS CLI を使用して Lambda オーソライザーを作成する方法を示しています。

REQUEST authorizer

次の例では、`REQUEST` オーソライザーを作成し、`Authorizer` ヘッダーと `accountId` コンテキスト変数 を ID ソースとして使用します。

```
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

次の例では、TOKEN オーソライザーを作成し、Authorization ヘッダーを ID ソースとして使用します。

```
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
```

Lambda オーソライザーを作成したら、これをテストできます。次の test-invoke-authorizer コマンドは、Lambda オーソライザーをテストする方法を示しています。

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

Lambda オーソライザーを使用するようにメソッドを設定する (コンソール)

Lambda オーソライザーを設定したら、これを API のメソッドにアタッチする必要があります。

API メソッドを設定して Lambda オーソライザーを使用するには

1. https://console.aws.amazon.com/apigateway で API Gateway コンソールにサインインします。
2. API を選択します。
3. [リソース] を選択し、新しいメソッドを選択するか、既存のメソッドを選択します。
4. [メソッドリクエスト] タブの [メソッドリクエスト設定] で、[編集] を選択します。
5. [オーソライザー] では、ドロップダウンメニューから、先ほど作成した Lambda オーソライザーを選択します。
6. (オプション) オーソライザートークンをバックエンドに渡す場合は、[HTTP リクエストヘッダー] を選択します。[ヘッダーを追加] を選択し、認証ヘッダーの名前を追加します。[名前]

に、API の Lambda オーソライザーを作成した際に指定した [トークンソース] の名前と一致するヘッダー名を入力します。このステップは [REQUEST] オーソライザーには適用されません。

7. [Save] を選択します。
8. [API のデプロイ] を選択して、API をステージにデプロイします。ステージ変数を使用した REQUEST オーソライザーの場合は、必要なステージ変数を定義して、[ステージ] ページで値を指定する必要があります。

Lambda オーソライザーを使用するように API のメソッドを設定する (AWS CLI)

Lambda オーソライザーを設定したら、これを API のメソッドにアタッチする必要があります。新しいメソッドを作成するか、パッチオペレーションを使用してオーソライザーを既存のメソッドにアタッチできます。

次の put-method コマンドは、Lambda オーソライザーを使用する新しいメソッドを作成する方法を示しています。

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

次の update-method コマンドは、Lambda オーソライザーを使用するように既存のメソッドを更新する方法を示しています。

```
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"
```

Amazon API Gateway Lambda オーソライザーへの入力

TOKEN の入力形式

TOKEN タイプの Lambda オーソライザー (以前のカスタムオーソライザー) の場合は、API のオーソライザーを設定する際、[トークンのソース] としてカスタムヘッダーを指定する必要があります。API クライアントは、受信リクエストでそのヘッダーに必要な認証トークンを渡す必要があります。API Gateway は、受信メソッドリクエストを受け取った時点で、カスタムヘッダーからトーク

ンを抽出します。その後、authorizationToken プロパティとしてメソッド ARN を渡すだけでなく、Lambda 関数の event オブジェクトの methodArn プロパティとしてトークンを渡します。

```
{
    "type":"TOKEN",
    "authorizationToken":"{caller-supplied-token}",
    "methodArn":"arn:aws:execute-api:{regionId}:{accountId}:{apiId}/{stage}/{httpVerb}/
[{resource}/[{child-resources}]]"
}
```

この例では、type プロパティは、オーソライザーのタイプ (TOKEN オーソライザー) を指定します。*{caller-supplied-token}* は、クライアントリクエストの認証ヘッダーの値であり、任意の文字列値にすることができます。methodArn は、受信するメソッドリクエストの ARN であり、Lambda オーソライザーの設定に従って API Gateway により自動入力されます。

REQUEST の入力形式

REQUEST タイプの Lambda オーソライザーの場合、API Gateway は、event オブジェクトの一部として、オーソライザーの Lambda 関数にリクエストパラメータを渡します。このリクエストパラメータには、ヘッダー、パスパラメータ、クエリ文字列パラメータ、ステージ変数、一部のリクエストコンテキスト変数などがあります。API 発信者は、パスパラメータ、ヘッダー、クエリ文字列パラメータを設定できます。API デベロッパーは API デプロイ時にステージ変数を設定する必要があります。API Gateway ランタイムに、このリクエストコンテキストが指定されます。

> **Note**
>
> パスパラメータは、リクエストパラメータとして Lambda オーソライザー関数に渡すことができますが、ID ソースとして使用することはできません。

次の例では、API メソッド (REQUEST) をプロキシ統合に設定する GET /request オーソライザーへの入力を表します。

```
{
  "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 はキーと値のペアのマップであり、$context 変数に対応します。その結果は API に依存します。

API Gateway は、マップに新しいキーを追加する場合があります。Lambda プロキシ統合での Lambda 関数の入力の詳細については、「プロキシ統合のための Lambda 関数の入力形式」を参照してください。

API Gateway Lambda オーソライザーからの出力

Lambda オーソライザー関数の出力はディクショナリのようなオブジェクトです。プリンシパル ID (principalId) と、ポリシーステートメントのリストを含むポリシードキュメント (policyDocument) を含む必要があります。出力には、キー/値ペアを含む context マップも含まることがあります。API が使用量プランを使用する (apiKeySource が AUTHORIZER に設定されている) 場合、Lambda オーソライザー関数は usageIdentifierKey プロパティ値として、使用量プランの API キーのいずれかを返す必要があります。

この出力の例を以下に示します。

```
{
  "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}"
}
```

ここで、ポリシーステートメントは、指定された API メソッド (Effect) を呼び出す (Action) ことを API Gateway 実行サービスに許可するか拒否するか (Resource) を指定しています。ワイルド

カード (*) を使ってリソースタイプ (メソッド) を指定できます。API を呼び出す有効なポリシーの設定の詳細については、「API Gateway で API を実行するための IAM ポリシーのステートメントの参照」を参照してください。

権限付与対応のメソッド ARN (`arn:aws:execute-api:{regionId}:{accountId}:{apiId}/{stage}/{httpVerb}/[{resource}/[{child-resources}]]` など) の場合、最大長は 1600 バイトです。パスパラメータの値 (そのサイズは実行時に決定されます) によっては、ARN の長さが制限を超えることがあります。これが発生した場合、API クライアントは `414 Request URI too long` レスポンスを受け取ります。

さらに、リソース ARN は、承認者によって出力されたポリシーステートメントに示されているように、現在 512 文字に制限されています。このため、JWT トークンが長すぎる URI をリクエスト URI に使用しないでください。代わりに、JWT トークンはリクエストヘッダーで安全に渡すことができます。

`principalId` 値には、マッピングテンプレートで `$context.authorizer.principalId` 変数を使ってアクセスできます。これはバックエンドに値を渡す場合に便利です。詳細については、「データモデル、オーソライザー、マッピングテンプレート、および CloudWatch アクセスログ記録用の `$context` 変数」を参照してください。

マッピングテンプレート内の `stringKey` マップの `numberKey`、`booleanKey`、または `"value"` 値 (例: `"1"`、`"true"`、または `context`) には、それぞれ `$context.authorizer.stringKey`、`$context.authorizer.numberKey`、または `$context.authorizer.booleanKey` を呼び出すことによりアクセスできます。返される値は、すべてが文字列化されます。`context` マップでキーの有効な値として JSON オブジェクトまたは配列を設定することはできません。

オーソライザーからバックエンドに、キャッシュされた認証情報を返すには、`context` マップを使用します。この際、統合リクエストのマッピングテンプレートを使用します。これにより、バックエンドのユーザーエクスペリエンスを強化するには、キャッシュされた認証情報を使用して、シークレットキーにアクセスする必要性を抑え、リクエストごとに認証トークンを開きます。

Lambda プロキシ統合の場合、API Gateway は、Lambda オーソライザーの `context` オブジェクトを、入力 `event` の一部としてバックエンドの Lambda 関数に直接渡します。`context` のキー/値ペアは、Lambda 関数で `$event.requestContext.authorizer.key` を呼び出して取得できます。

`{api-key}` は、API ステージの使用量プランの API キーを表します。詳細については、「the section called “使用量プラン”」を参照してください。

Lambda オーソライザー例からの出力例を次に示します。この出力例は、AWS アカウント (123456789012) の API (ymy8tbxw7b) の dev ステージの GET メソッドに対する呼び出しをブロック (Deny) するポリシーステートメントを示しています。

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

API Gateway Lambda オーソライザーで API を呼び出す

Lambda オーソライザー (以前のカスタムオーソライザー) を設定し、API をデプロイしている場合は、Lambda オーソライザーを有効にして API を検証する必要があります。そのためには、cURL または Postman などの REST クライアントが必要です。以下の例では、Postman を使用します。

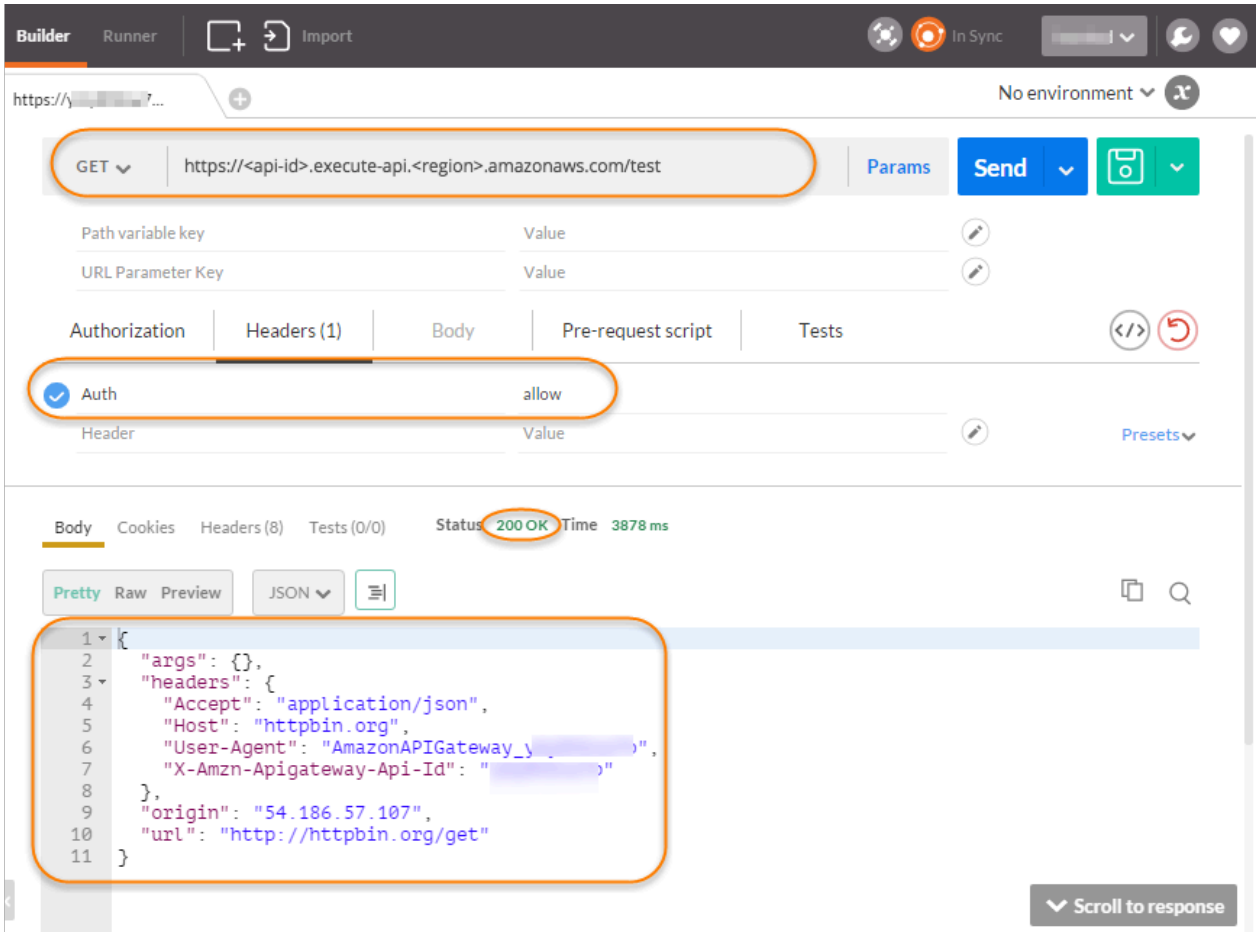
> **Note**
>
> オーソライザーが有効なメソッドを呼び出すとき、TOKEN オーソライザーに必要なトークンが設定されていない、Null である、または指定された [トークン検証式] によって無効にされている場合、API Gateway は CloudWatch への呼び出しを記録しません。同様に、REQUEST オーソライザーに必要な ID ソースのいずれかが設定されていない、Null である、または空の場合、API Gateway は CloudWatch への呼び出しを記録しません。

以下では、Postman を使用して Lambda TOKEN オーソライザーで API の呼び出しまたはテストを行う方法を説明します。必要なパス、ヘッダー、またはクエリ文字列パラメータを明示的に指定している場合は、このメソッドを Lambda REQUEST オーソライザーを使用した API の呼び出しに適用できます。

カスタムの **TOKEN** オーソライザーで API を呼び出すには

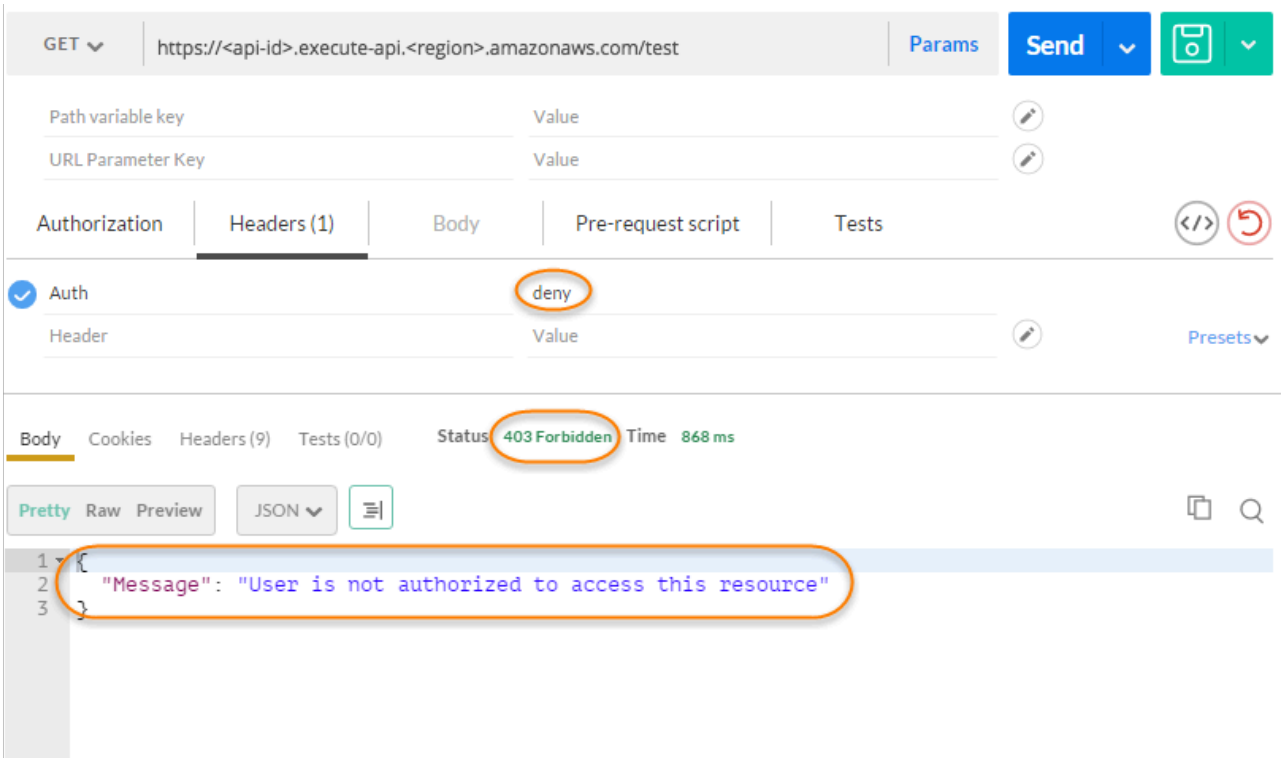
1. [Postman] を開き、[GET] メソッドを選択して、API の [呼び出し URL] を、隣の URL フィールド内に貼り付けます。

   Lambda 認証トークンヘッダーを追加し、その値を `allow` に設定します。[Send (送信)] を選択します。



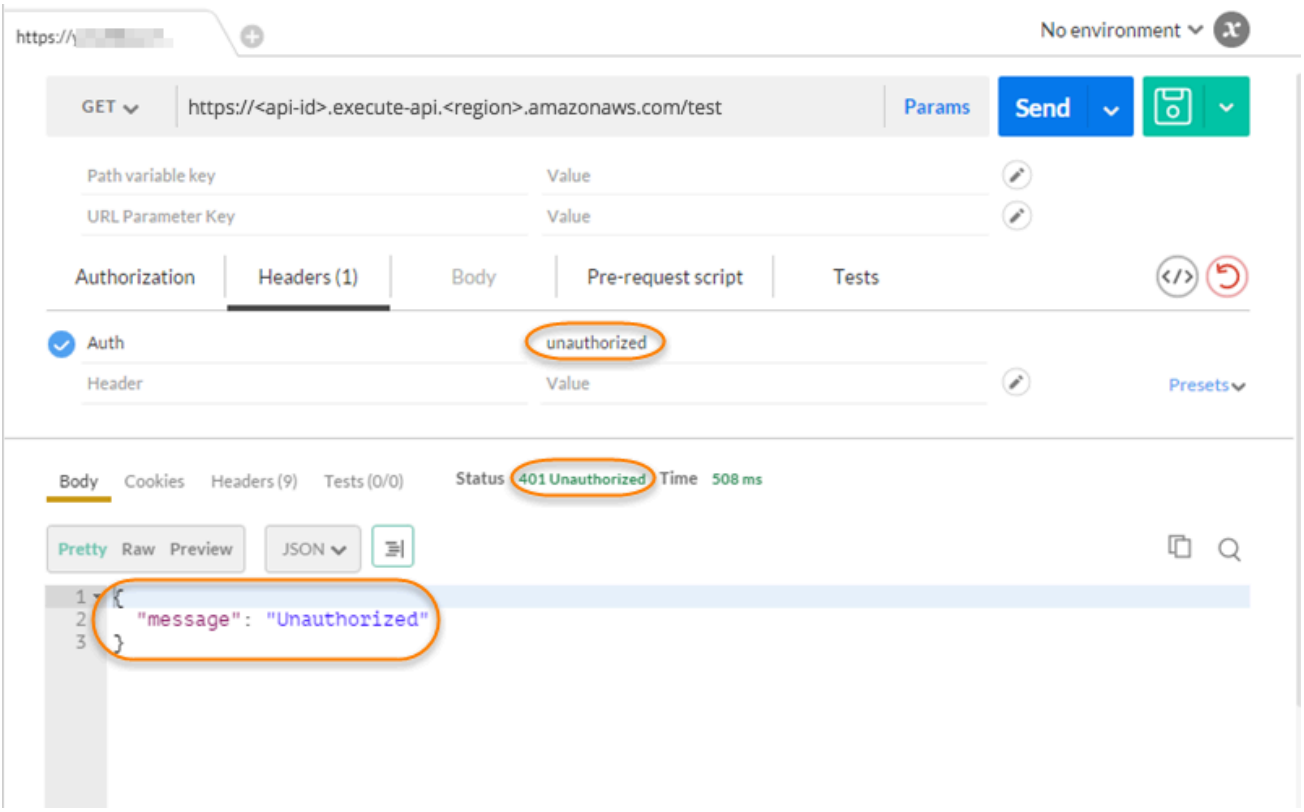
   レスポンスは、API Gateway Lambda オーソライザーが [200 OK] レスポンスを返し、メソッドに関連付けられた HTTP エンドポイント (http://httpbin.org/get) にアクセスすることを適切に呼び出しに許可していることを示しています。

2. Postman で、Lambda 認証トークンヘッダー値を `deny` に変更します。[Send (送信)] を選択します。



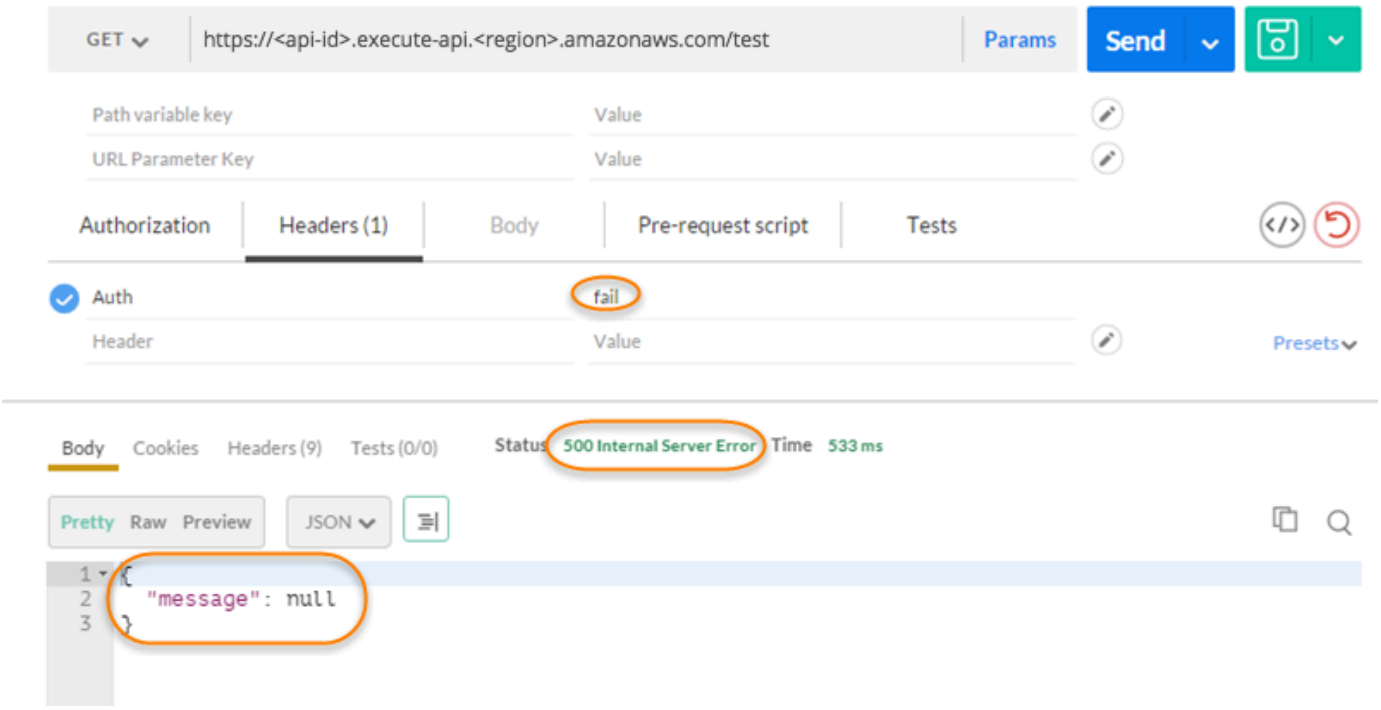
レスポンスは、API Gateway Lambda カスタム認証が [403 Forbidden] レスポンスを返し、HTTP エンドポイントにアクセスすることを呼び出しに許可しないことを示しています。

3. Postman で、Lambda 認証トークンヘッダー値を `unauthorized` に変更し、[送信] を選択します。



レスポンスは、API Gateway が [401 Unauthorized] レスポンスを返し、HTTP エンドポイントにアクセスすることを呼び出しに許可しないことを示しています。

4. ここで、Lambda 認証トークンヘッダー値を `fail` に変更します。[Send (送信)] を選択します。



レスポンスは、API Gateway が [500 Internal Server Error] レスポンスを返し、HTTP エンドポイントにアクセスすることを呼び出しに許可しないことを示しています。

クロスアカウントの Lambda オーソライザーを設定する

AWS Lambda 関数は、API オーソライザー関数とは異なる AWS アカウントからも使用できるようになりました。各アカウントは、Amazon API Gateway を利用できるリージョンであればどのリージョンでもかまいません。Lambda オーソライザー関数では、OAuth や SAML などのベアラートークン認証戦略を使用できます。これにより、複数の API Gateway API 間で簡単に一元管理し、主要な Lambda オーソライザー関数を共有できるようになります。

このセクションでは、Amazon API Gateway コンソールを使用してクロスアカウント Lambda オーソライザー関数を設定する方法を示します。

これらの手順は、AWS アカウントに API Gateway API、別のアカウントに Lambda オーソライザー関数が設定されていることを前提としています。

API Gateway コンソールを使用して クロスアカウント Lambda オーソライザーを設定する

API を設定しているアカウントで Amazon API Gateway コンソールにサインインし、以下の操作を行います。

1. API を選択し、メインナビゲーションペインで [オーソライザー] を選択します。
2. [オーソライザーの作成] を選択します。
3. [オーソライザー名] で、オーソライザー名を入力します。
4. [オーソライザータイプ] には Lambda を選択します。
5. [Lambda 関数] で、2 番目のアカウントで作成した Lambda オーソライザー関数の完全な ARN を入力します。

   > **Note**
   >
   > Lambda コンソールのコンソールウィンドウの右上隅で、関数の ARN を検索できます。

6. `aws lambda add-permission` コマンド文字列を含む警告が表示されます。このポリシーは、API Gateway にオーソライザー Lambda 関数を呼び出すアクセス許可を付与します。後ほど使用するために、コマンドをコピーして保存しておきます。コマンドは、オーソライザーの作成後に実行します。

7. [Lambda 呼び出しロール] を空白のままにすると、API Gateway コンソールでリソースベースのポリシーが設定されます。ポリシーは、API Gateway に認可の Lambda 関数を呼び出すアクセス許可を付与します。また、IAM ロールを入力して、API Gateway でオーソライザーの Lambda 関数を呼び出せるようにします。このようなロールの例については、「引き受け可能な IAM ロールを作成する」を参照してください。
8. [Lambda イベントペイロード] には、TOKEN オーソライザーの [トークン]、または REQUEST オーソライザーの [リクエスト] を選択します。
9. 前のステップの選択内容に応じて、次のいずれかを実行します。

   a. [トークン] オプションで、以下の操作を行います。

      - [トークンソース] には、認証トークンを含むヘッダー名を入力します。認証トークンを Lambda オーソライザーに送信するには、この名前のヘッダーが API クライアントに含まれている必要があります。
      - オプションで、[トークンの検証] に RegEx ステートメントを入力します。API Gateway は、この式に対して、入力トークンの初期検証を実行し、認証が成功するとオーソライザーを呼び出します。これにより API への呼び出しを減らすことができます。
      - オーソライザーが生成した認証ポリシーをキャッシュするには、[認証キャッシュ] をオンのままにします。ポリシーのキャッシュが有効の場合、[TTL] 値を変更できます。[TTL] を 0 に設定すると、ポリシーのキャッシュは無効になります。ポリシーのキャッシュが有効の場合、[トークンのソース] で指定されているヘッダー名はキャッシュキーになります。リクエストでこのヘッダーに複数の値が渡された場合、すべての値がキャッシュキーになり、順序は保持されます。

      > **Note**
      >
      > デフォルトの [TTL] 値は 300 秒です。最大値は 3600 秒で、この制限値を増やすことはできません。

   b. [リクエスト] オプションで、以下の操作を行います。

      - [ID ソースタイプ] では、パラメータータイプを選択します。サポートされているパラメータタイプは、Header、Query string、Stage variable、および Context です。ID ソースをさらに追加するには、[パラメーターの追加] を選択します。
      - オーソライザーが生成した認証ポリシーをキャッシュするには、[認証キャッシュ] をオンのままにします。ポリシーのキャッシュが有効の場合、[TTL] 値を変更できます。[TTL] を 0 に設定すると、ポリシーのキャッシュは無効になります。

API Gateway では、リクエストオーソライザーのキャッシングキーとして、指定された ID ソースを使用します。キャッシュが有効の場合、API Gateway は、指定されているすべての ID ソースがランタイムに存在していることを確認できた場合のみ、Lambda 関数を呼び出します。指定された ID ソースが欠落しているか、null、または空の場合、API Gateway は、`401 Unauthorized` レスポンスを返します。オーソライザーの Lambda 関数を呼び出すことはありません。

複数の ID ソースが定義されている場合、それらはすべて、オーソライザーのキャッシュキーを取得するために使用されます。キャッシュキー部分のいずれかを変更すると、オーソライザーは、キャッシュされたポリシードキュメントを破棄し、新しいドキュメントを作成します。リクエストで複数の値を含むヘッダーが渡された場合、すべての値がキャッシュキーの一部になり、順序は保持されます。

- キャッシングがオフの場合は、ID ソースを指定する必要はありません。

> **Note**
>
> キャッシングを有効にするには、認証は API 全体のすべてのメソッドに適用されるポリシーを返す必要があります。メソッド固有のポリシーを適用するには、[認証キャッシュ] をオフにします。

10. [オーソライザーの作成] を選択します。
11. 前のステップでコピーした `aws lambda add-permission` コマンド文字列を、2 番目のアカウント用に設定された AWS CLI ウィンドウに貼り付けます。`AUTHORIZER_ID` をユーザーのオーソライザー ID に置き換えます。これにより、最初のアカウントから 2 番目のアカウントの Lambda オーソライザー関数へのアクセスが許可されます。

Amazon Cognito ユーザープールをオーソライザーとして使用して REST API へのアクセスを制御する

IAM ロールとポリシーまたは Lambda オーソライザー (以前のカスタムオーソライザー) の代わりに、Amazon Cognito ユーザープールを使用して、Amazon API Gateway の API にアクセスできるユーザーを制御します。

API で Amazon Cognito ユーザープールを使用するには、`COGNITO_USER_POOLS` タイプのオーソライザーを作成してから、そのオーソライザーを使用する API メソッドを構成する必要があります。API がデプロイされた後、クライアントはまずユーザーをユーザープールに署名し、

ユーザーの ID またはアクセストークンを取得してから、トークンの 1 つ (通常はリクエストの `Authorization` ヘッダーに設定されている) で API メソッドを呼び出す必要があります。API 呼び出しは、必要なトークンが提供され、提供されたトークンが有効な場合にのみ成功します。そうでない場合、クライアントは認証された認証情報を持たないため呼び出しを許可されません。

ID トークンは、サインインされたユーザーの ID リクエストに基づいて API 呼び出しを承認するために使用されます。アクセストークンは、指定されたアクセス保護されたリソースのカスタムスコープに基づいて API 呼び出しを承認するために使用されます。詳細については、「ユーザープールのトークンの使用」および「リソースサーバーおよびカスタムスコープの管理」を参照してください。

API 用の Amazon Cognito ユーザープールを作成および設定するには、次のタスクを実行します。

- Amazon Cognito コンソール、CLI/SDK、または API を使用して、ユーザープールを作成するか、別の AWS アカウントが所有するものを使用します。
- API Gateway コンソール、CLI/SDK、または API を使用して、選択したユーザープールで API Gateway オーソライザーを作成します。
- API Gateway コンソール、CLI/SDK、または API を使用して、選択した API メソッドでオーソライザーを有効にします。

ユーザープールを有効にして API メソッドを呼び出すには、API クライアントで次のタスクを実行します。

- Amazon Cognito CLI/SDK、または API を使用して、選択したユーザープールにユーザーをサインインし、ID トークンまたはアクセストークンを取得します。SDK の使用方法の詳細については、「AWS SDK を使用した Amazon Cognito のコード例」を参照してください。
- クライアント固有のフレームワークを使用して、デプロイされた API Gateway API を呼び出し、`Authorization` ヘッダーに適切なトークンを指定します。

API 開発者は、クライアント開発者に、ユーザープール ID、クライアント ID、および場合によってはユーザープールの一部として定義されている関連クライアントのシークレットを提供する必要があります。

> **Note**
>
> ユーザーが Amazon Cognito の認証情報を使用してサインインし、IAM ロールのアクセス許可を使用するための一時的な認証情報を取得するには、Amazon Cognito フェデレーテッド

アイデンティティを使用します。API リソースエンドポイントの HTTP メソッドごとに、認証タイプ、カテゴリ Method Execution を AWS_IAM に設定します。

このセクションでは、ユーザープールの作成方法、API Gateway API をユーザープールと統合する方法、ユーザープールと統合された API を呼び出す方法を説明します。

トピック

- REST API 用の Amazon Cognito ユーザープールオーソライザーを作成するためのアクセス許可を取得する
- REST API 用の Amazon Cognito ユーザープールを作成する
- REST API と Amazon Cognito ユーザープールを統合する
- Amazon Cognito ユーザープールと統合された REST API を呼び出す
- API Gateway コンソールを使用して REST API 用のクロスアカウントの Amazon Cognito オーソライザーを設定する
- AWS CloudFormation を使用して REST API の Amazon Cognito オーソライザーを作成する

REST API 用の Amazon Cognito ユーザープールオーソライザーを作成するためのアクセス許可を取得する

Amazon Cognito のユーザープールを使用してオーソライザーを作成するには、選択した Amazon Cognito ユーザープールでオーソライザーを作成または更新するための Allow アクセス許可が必要です。以下の IAM ポリシードキュメントに示しているのは、そのようなアクセス許可の例です。

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

ポリシーが、ユーザーが属している IAM グループ、または割り当てられている IAM ロールにアタッチされていることを確認します。

上記のポリシードキュメントでは、apigateway:POST アクションは新しいオーソライザーを作成するためのものであり、apigateway:PATCH アクションは既存のオーソライザーを更新するためのものです。最初の 2 つのワイルドカード (*) をそれぞれ Resource の値で上書きすることで、ポリシーを特定のリージョンまたは特定の API に制限できます。

ここで使用されている Condition 句は、Allowed アクセス権限を指定されたユーザープールに制限します。Condition 句がある場合、条件に一致しないユーザープールへのアクセスは拒否されます。アクセス権限に Condition 句がない場合、すべてのユーザープールへのアクセスが許可されます。

Condition 句の設定には、次のオプションがあります。

- `ArnLike` または `ArnEquals` 条件式を設定して、指定されたユーザープールのみを使用する `COGNITO_USER_POOLS` オーソライザーの作成または更新を許可できます。
- `ArnNotLike` または `ArnNotEquals` 条件式を設定して、式に指定されていないユーザープールを使用する `COGNITO_USER_POOLS` オーソライザーの作成または更新を許可できます。
- 任意の AWS アカウントおよび任意のリージョンの任意のユーザープールを使用する `COGNITO_USER_POOLS` オーソライザーの作成または更新を許可するには、`Condition` 句を省略できます。

Amazon リソースネーム (ARN) の条件式の詳細については、Amazon リソースネームの条件演算子を参照してください。次の例に示すように、`apigateway:CognitoUserPoolProviderArn` は、`COGNITO_USER_POOLS` タイプの API Gateway オーソライザーで使用できる、または使用できない `COGNITO_USER_POOLS` ユーザープールの ARN のリストです。

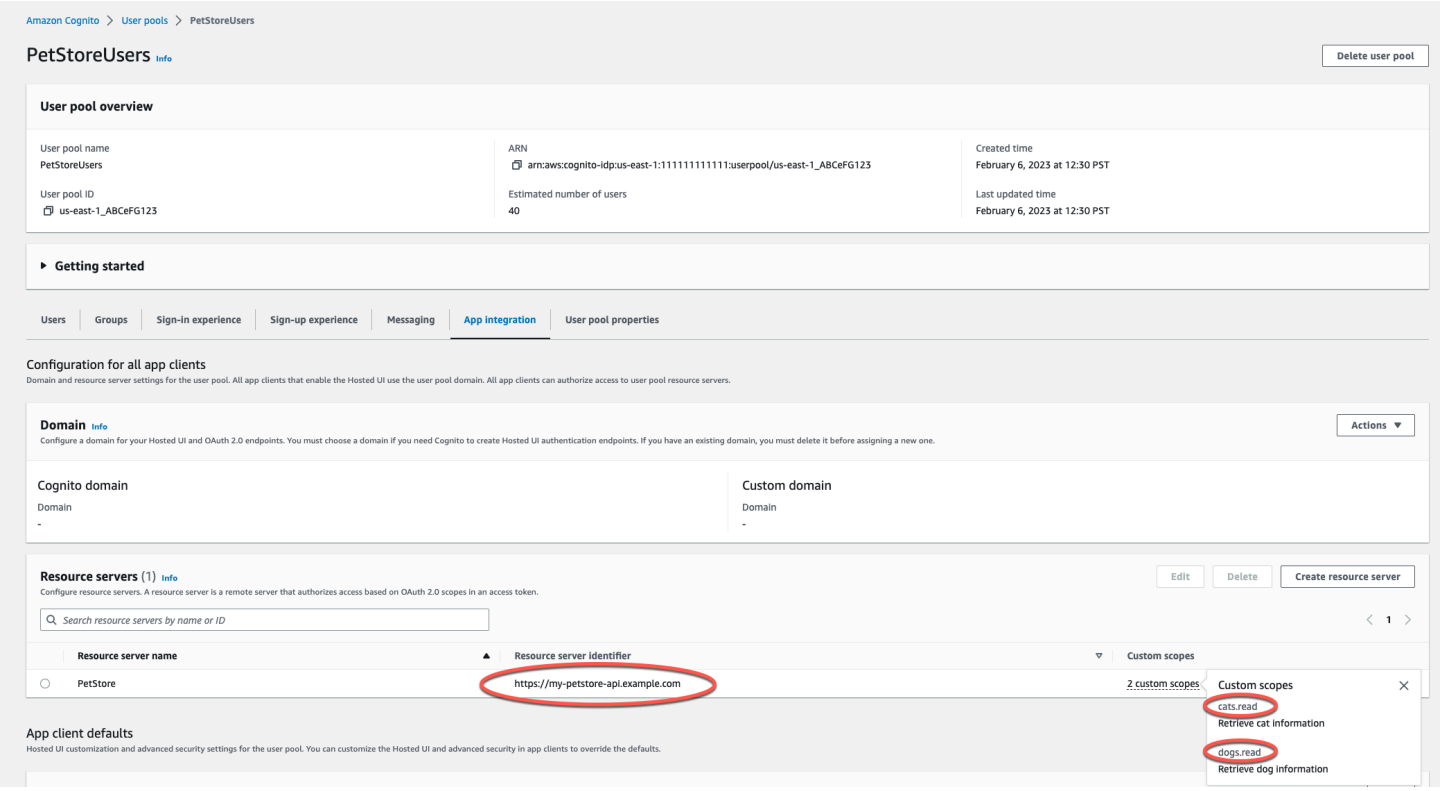
REST API 用の Amazon Cognito ユーザープールを作成する

API をユーザープールと統合する前に、Amazon Cognito でユーザープールを作成する必要があります。ユーザープールの設定は、Amazon Cognito のすべてのリソースクォータに従う必要があります。グループ、ユーザー、ロールなどのユーザー定義の Amazon Cognito 変数に使用できるのは、英数字だけです。ユーザープールを作成する方法については、Amazon Cognito デベロッパーガイドの「チュートリアル: ユーザープールを作成する」を参照してください。

ユーザープール ID、クライアント ID、クライアントシークレットをメモします。クライアントは、ユーザーがユーザープールに登録、サインインし、ユーザープールを使用して設定された API メソッドを呼び出すリクエストに含めるための ID またはアクセストークンを取得するために、それらを Amazon Cognito に提供する必要があります。また、API Gateway のオーソライザーとしてユーザープールを設定する場合、以下に説明するようにユーザープール名を指定する必要があります。

アクセストークンを使用して API メソッド呼び出しを承認している場合は、特定のリソースサーバーに必要なカスタムスコープを設定するために、ユーザープールとのアプリケーション統合を設定するようにしてください。Amazon Cognito ユーザープールでトークンを使用する方法の詳細については、「ユーザープールでトークンを使用する」を参照してください。リソースサーバーの詳細については、「ユーザープールのリソースサーバーを定義する」を参照してください。

設定されたリソースサーバー ID とカスタムスコープ名をメモしておきます。これらは、`COGNITO_USER_POOLS` オーソライザーが使用する [OAuth スコープ] のアクセススコープのフルネームを作成するために必要です。



REST API と Amazon Cognito ユーザープールを統合する

API Gateway で Amazon Cognito ユーザープールを作成し、そのユーザープールを使用する COGNITO_USER_POOLS オーソライザーを作成する必要があります。次の手順では、この操作を API Gateway コンソールを使用して行う方法について説明します。

> **Note**
>
> [CreateAuthorizer](#) アクションを使用して、複数のユーザープールを使用する COGNITO_USER_POOLS オーソライザーを作成できます。1 つの COGNITO_USER_POOLS オーソライザーには最大 1,000 のユーザープールを使用できます。この制限を増やすことはできません。

> **Important**
>
> 以下の手順の実行後、API をデプロイまたは再デプロイして変更を伝達する必要があります。API のデプロイの詳細については、「[Amazon API Gateway での REST API のデプロイ](#)」を参照してください。

API Gateway コンソールを使用して **COGNITO_USER_POOLS** 認証を作成するには

1. 新しい API を作成、または API Gateway に既存の API を選択します。
2. メインナビゲーションペインで、[オーソライザー] を選択します。
3. [オーソライザーの作成] を選択します。
4. ユーザープールを使用するように新しいオーソライザーを設定するには、次の手順を実行します。

   a. [オーソライザー名] に名前を入力します。

   b. [オーソライザータイプ] には、[Cognito] を選択します。

   c. [Cognito ユーザープール] では、Amazon Cognito を作成した場所 AWS リージョン を選択し、使用可能なユーザープールを選択します。

      ユーザープールを定義するには、ステージ変数を使用できます。ユーザープールには、形式として `arn:aws:cognito-idp:`*`us-east-2`*`:`*`111122223333`*`:userpool/${stageVariables.`*`MyUserPool`*`}` を使用します。

   d. ユーザーがサインインした際に Amazon Cognito が返す ID またはアクセストークンを渡すように、[トークンソース] には、ヘッダー名として **Authorization** と入力します。

   e. (オプション) 必要に応じて [トークン検証] フィールドに正規表現を入力して、リクエストが Amazon Cognito で承認される前に ID トークンの aud (対象者) フィールドを検証します。アクセストークンを使用する場合、この検証では、アクセストークンが aud フィールドを含まないために要求が拒否されることに注意してください。

   f. [オーソライザーの作成] を選択します。

5. COGNITO_USER_POOLS オーソライザーを作成した後、必要に応じて、ユーザープールからプロビジョニングされた ID トークンを指定することで、呼び出しをテストできます。Amazon Cognito ID SDK を呼び出してユーザーサインインを実行することで、この ID トークンを取得できます。`InitiateAuth` アクションも使用できます。[認証スコープ] を設定しない場合、API Gateway は指定されたトークンを ID トークンとして扱います。

前述の手順では、新しく作成した Amazon Cognito ユーザープールを使用する COGNITO_USER_POOLS オーソライザーを作成します。API メソッドでオーソライザーを有効にした方法に応じて、統合トークンまたは統合ユーザープールからプロビジョニングされたアクセストークンを使用できます。

メソッドで **COGNITO_USER_POOLS** オーソライザーを設定するには

1. [リソース] をクリックします。新しいメソッドを選択するか、既存のメソッドを選択します。必要に応じて、リソースを作成します。
2. [メソッドリクエスト] タブの [メソッドリクエスト設定] で、[編集] を選択します。
3. [オーソライザー] には、ドロップダウンメニューから、先ほど作成した Amazon Cognito ユーザープールオーソライザーを選択します。
4. ID トークンを使用するには、次の操作を行います。

   a. [認証スコープ] は空のままにしておきます。

   b. 必要に応じて、[統合リクエスト] で、ユーザープールからバックエンドに特定の ID クレームプロパティを渡すために、本文マッピングテンプレートに `$context.authorizer.claims['property-name']` 式または `$context.authorizer.claims.property-name` 式を追加します。`sub` や`custom-sub` などの簡単なプロパティ名については、2 つの表記法は同じです。`custom:role` などの複雑なプロパティ名の場合、ドット表記は使用できません。たとえば、以下のマッピング式は、バックエンドに、クレームの `sub` および `email` の標準フィールドを渡します。

   ```
   {
    "context" : {
     "sub" : "$context.authorizer.claims.sub",
     "email" : "$context.authorizer.claims.email"
    }
   }
   ```

   ユーザープールを設定するときにカスタムクレームフィールドを宣言した場合は、同じパターンに従ってカスタムフィールドにアクセスできます。次の例では、クレームの `role` カスタムフィールドを取得します。

   ```
   {
    "context" : {
     "role" : "$context.authorizer.claims.role"
       }
   }
   ```

   カスタムクレームフィールドが `custom:role` として宣言されている場合は、以下の例を使用してクレームのプロパティを取得します。

```
{
 "context" : {
  "role" : "$context.authorizer.claims['custom:role']"
    }
}
```

5. アクセストークンを使用するには、次の操作を行います。

   a. [認証スコープ] には、Amazon Cognito ユーザープールが作成された際に設定されたスコープのフルネームを 1 つ以上入力します。たとえば、「REST API 用の Amazon Cognito ユーザープールを作成する」の例では、スコープの 1 つは `https://my-petstore-api.example.com/cats.read` です。

      実行時に、このステップのメソッドで指定されたスコープが、受信トークンで要求されているスコープと一致する場合、メソッド呼び出しは成功します。それ以外の場合、`401 Unauthorized` レスポンスでコールが失敗します。

   b. [Save] を選択します。

6. 選択した他のメソッドにこれらの手順を繰り返してください。

`COGNITO_USER_POOLS` オーソライザーで、[OAuth Scopes] オプションが指定されていない場合、API Gateway は、指定されたトークンを ID トークンとして処理し、ユーザープールからのトークンに対して、要求された ID を検証します。それ以外の場合、API Gateway は提供されたトークンをアクセストークンとして処理し、トークンで要求されたアクセススコープをメソッドで宣言されている承認スコープと照合して検証します。

API Gateway コンソールを使う代わりに、OpenAPI 定義ファイル内で指定してメソッド上での Amazon Cognito ユーザープールを有効化し、API 定義を API Gateway にインポートすることもできます。

OpenAPI 定義ファイルを使って COGNITO_USER_POOLS オーソライザーをインポートするには

1. API 用の OpenAPI 定義ファイルを作成 (またはエクスポート) します。
2. OpenAPI 3.0 の `securitySchemes` セクション、または Open API 2.0 の `securityDefinitions` セクションの一部として、`COGNITO_USER_POOLS` オーソライザー (`MyUserPool`) JSON を次のように指定します。

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. メソッドの認可にアイデンティティトークンを使用するには、ルートリソースの次の GET メソッドに示すように、メソッドの `{ "MyUserPool": [] }` 定義に `security` を追加します。

```
"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. メソッドの認可にアクセストークンを使用するには、上記のセキュリティ定義を `{ "MyUserPool": [resource-server/scope, ...] }` に変更します。

```
  "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. 必要に応じて、OpenAPI 定義または拡張機能を使用し、他の API 設定を指定します。詳細については、「OpenAPI への API Gateway 拡張機能の使用」を参照してください。

Amazon Cognito ユーザープールと統合された REST API を呼び出す

設定されたユーザープールオーソライザーを使用して、メソッドを呼び出すには、クライアントは以下を実行する必要があります。

- ユーザープールにサインアップすることを可能にします。
- ユーザープールにサインインすることを可能にします。
- ユーザープールからサインインしているユーザーの ID またはアクセストークンを取得します。
- `Authorization` ヘッダー (または、オーソライザー作成時に指定した別のヘッダー) にトークンを含めます。

これらのタスクは、AWS Amplify Amplify を使用して実行できます。詳細については、「Integrating Amazon Cognito With Web and Mobile Apps (Amazon Cognito をウェブアプリおよびモバイルアプリとの統合)」を参照してください。

- Android の場合は、「 Getting Started with Amplify for Android (Android 用 Amplify の使用開始) 」を参照してください。
- iOS を使用するには、「Getting started with Amplify for iOS (iOS 版 Amplify の使用開始) 」を参照してください。
- JavaScript を使用するには、「Getting Started with Amplify for Javascript (Javascript のアンプリファイの使用開始) 」を参照してください。

API Gateway コンソールを使用して REST API 用のクロスアカウントの Amazon Cognito オーソライザーを設定する

Amazon Cognito ユーザープールは、API オーソライザーとは異なる AWS アカウントからも使用できるようになりました。Amazon Cognito ユーザープールでは、OAuth や SAML などのベアラートークン認証戦略を使用できます。これにより、複数の API Gateway API 間で簡単に一元管理し、主要な Amazon Cognito ユーザープールオーソライザーを共有できるようになります。

このセクションでは、Amazon API Gateway コンソールを使用して、クロスアカウント Amazon Cognito ユーザープールを設定する方法について説明します。

これらの手順は、AWS アカウントに API Gateway API、別のアカウントに Amazon Cognito ユーザープールが設定されていることを前提としています。

API Gateway コンソールを使用してクロスアカウントの Amazon Cognito オーソライザーを設定する

API を設定しているアカウントで Amazon API Gateway コンソールにサインインし、以下の操作を行います。

1. 新しい API を作成、または API Gateway に既存の API を選択します。
2. メインナビゲーションペインで、[オーソライザー] を選択します。
3. [オーソライザーの作成] を選択します。
4. ユーザープールを使用するように新しいオーソライザーを設定するには、次の手順を実行します。

   a. [オーソライザー名] に名前を入力します。

   b. [オーソライザータイプ] には、[Cognito] を選択します。

   c. [Cognito ユーザープール] で、2 番目のアカウントで作成したユーザープールの完全な ARN を入力します。

   > **Note**
   >
   > Amazon Cognito コンソールでは、ユーザープールの ARN は、[全般設定] ペインの [プール ARN] フィールドにあります。

   d. ユーザーがサインインした際に Amazon Cognito が返す ID またはアクセストークンを渡すように、[トークンソース] には、ヘッダー名として **`Authorization`** と入力します。

   e. (オプション) 必要に応じて [トークン検証] フィールドに正規表現を入力して、リクエストが Amazon Cognito で承認される前に ID トークンの aud (対象者) フィールドを検証します。アクセストークンを使用する場合、この検証では、アクセストークンが aud フィールドを含まないために要求が拒否されることに注意してください。

   f. [オーソライザーの作成] を選択します。

AWS CloudFormation を使用して REST API の Amazon Cognito オーソライザーを作成する

AWS CloudFormation を使用して、Amazon Cognito ユーザープールと Amazon Cognito オーソライザーを作成できます。この例の AWS CloudFormation テンプレートでは、次のような処理を実行します。

- Amazon Cognito ユーザープールを作成します。クライアントは、まずユーザーをユーザープールにサインインし、ID トークンまたはアクセストークンを取得する必要があります。アクセストークンを使用して API メソッド呼び出しを承認している場合は、特定のリソースサーバーに必要なカスタムスコープを設定するために、ユーザープールとのアプリケーション統合を設定するようにしてください。
- GET メソッドを使用して API Gateway API を作成します。
- Authorization ヘッダーをトークンソースとして使用する Amazon Cognito オーソライザーを作成します。

```
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
          - /
```

REST API 統合の設定

API メソッドを設定したら、バックエンドのエンドポイントに統合する必要があります。バックエンドのエンドポイントは、統合エンドポイントとも呼ばれ、Lambda 関数、HTTP ウェブページ、または AWS のサービスアクションとして使用できます。

API メソッドと同様に、API 統合には統合リクエストと統合レスポンスがあります。統合リクエストは、バックエンドが受け取った HTTP リクエストをカプセル化します。これは、クライアントが送ったメソッドリクエストと同じ場合もあれば異なる場合もあります。統合レスポンスは、バックエンドが返した出力をカプセル化する HTTP レスポンスです。

統合リクエストの設定には、クライアントから送信されたメソッドリクエストをバックエンドに渡す方法の設定、必要に応じてリクエストデータを統合リクエストデータに変換する方法の設定、呼び出す Lambda 関数の指定、受け取ったリクエストを転送する HTTP サーバーの指定、呼び出す AWS のサービスアクションの指定が含まれます。

統合レスポンスの設定には、(非プロキシ統合の場合のみ) バックエンドが返す結果を特定のステータスコードのメソッドレスポンスに渡す方法の設定、指定した統合レスポンスパラメータを事前設定されたメソッドレスポンスパラメータに変換する方法の設定、指定した本文マッピングテンプレートに従って統合レスポンス本文をメソッドレスポンス本文にマップする方法の設定が含まれます。

プログラムで、統合リクエストは `Integration` リソースにカプセル化し、統合レスポンスは API Gateway の `IntegrationResponse` リソースにカプセル化します。

統合リクエストを設定するには、`Integration` リソースを作成し、そのリソースを使用して統合エンドポイント URL を設定します。次に、バックエンドにアクセスするための IAM アクセス許可を設定し、受け取ったリクエストデータをバックエンドに渡す前に変換するためのマッピングを指定します。非プロキシ統合の場合、統合レスポンスを設定するには、`IntegrationResponse` リソースを作成し、そのリソースを使用してターゲットメソッドレスポンスを設定します。その後、バックエンド出力をメソッドレスポンスにマップする方法を設定します。

トピック

API Gateway で統合リクエストを設定する

統合リクエストを設定するには、以下の必須およびオプションのタスクを実行します。

1. 統合タイプを選択します。このタイプによってメソッドリクエストデータをバックエンドに渡す方法が決まります。
2. 非モック統合の場合 (MOCK 統合以外の場合)、HTTP メソッドとターゲット統合エンドポイントの URI を指定します。
3. Lambda 関数および他の AWS のサービスアクションとの統合の場合、API Gateway がお客様に代わってバックエンドを呼び出すために必要なアクセス許可を持つように IAM ロールを設定します。
4. 非プロキシ統合の場合、事前定義されたメソッドリクエストパラメータを該当する統合リクエストパラメータにマップするために必要なパラメータマッピングを設定します。
5. 非プロキシ統合の場合、受け取った特定のコンテンツタイプのメソッドリクエストボディをマップするために必要な本文マッピングを、指定したマッピングテンプレートに従って設定します。
6. 非プロキシ統合の場合、受け取ったメソッドリクエストデータをそのままバックエンドに渡す条件を指定します。
7. オプションで、バイナリペイロードの型変換の処理方法を指定します。
8. オプションで、キャッシュ名前空間名とキャッシュキーパラメータを宣言して API キャッシングを有効にします。

これらのタスクを実行するには、API Gateway の統合リソースを作成し、適切なプロパティ値を設定する必要があります。これを行うには、API Gateway コンソール、AWS CLI コマンド、AWS SDK、または API Gateway REST API を使用します。

トピック

- API 統合リクエストの基本タスク
- API Gateway API 統合タイプの選択
- プロキシリソースとのプロキシ統合を設定する
- API Gateway コンソールを使用して API 統合リクエストを設定する

API 統合リクエストの基本タスク

統合リクエストは、API Gateway がバックエンドに送る HTTP リクエストであり、クライアントから送信されたリクエストデータを渡し、必要に応じて変換します。統合リクエストの HTTP メソッド (つまり動詞) と URI は、バックエンド (つまり統合エンドポイント) が指定します。HTTP メソッドと URI はそれぞれ、メソッドリクエストのものと同じ場合もあれば異なる場合もあります。

たとえば、Lambda 関数が、Amazon S3 からフェッチされたファイルを返す場合、このオペレーションを直観的に GET メソッドリクエストとしてクライアントに公開できます。これは、対応する統合リクエストで POST リクエストを使用して Lambda 関数を呼び出す必要があっても同じです。HTTP エンドポイントの場合は、メソッドリクエストおよび対応する統合リクエストの両方で同じ HTTP 動詞を使用することがよくあります。ただし、これは必須ではありません。以下のメソッドリクエストがあるとします。

```
GET /{var}?query=value
Host: api.domain.net
```

これを以下の統合リクエストに統合できます。

```
POST /
Host: service.domain.com
Content-Type: application/json
Content-Length: ...

{
   path: "{var}'s value",
   type: "value"
}
```

API 開発者は、要件に合わせてどのような HTTP 動詞と URI でもメソッドリクエストに使用できます。しかし、統合エンドポイントの要件に従う必要があります。メソッドリクエストデータが統合リクエストデータと異なる場合、メソッドリクエストデータから統合リクエストデータへのマッピングを提供することで、その差異を調整できます。

ここまでの例では、マッピングにより、{var} メソッドリクエストのパス変数 (query) とクエリパラメータ (GET) の値が、統合リクエストのペイロードプロパティ path と type の値に変換されます。その他のマップ可能なリクエストデータとしては、リクエストのヘッダーと本文があります。これらについては、「API Gateway コンソールを使用してリクエストとレスポンスのデータマッピングを設定する」で説明しています。

HTTP または HTTP プロキシ統合リクエストを設定するときは、バックエンド HTTP エンドポイント URL を統合リクエスト URI 値として割り当てます。たとえば、PetStore API では、pets のページを取得するメソッドリクエストに以下の統合リクエスト URI があります。

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

Lambda または Lambda プロキシ統合を設定するときは、Lambda 関数を呼び出すための Amazon リソースネーム (ARN) を統合リクエスト URI 値として割り当てます。この ARN は以下の形式になります。

```
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/ の後の部分、つまり、/2015-03-31/functions/arn:aws:lambda:*lambda-region*:*account-id*:function:*lambda-function-name*/invocations は、Lambda の Invoke アクションの REST API URI パスです。API Gateway コンソールを使用して Lambda 統合を設定する場合は、API Gateway によって ARN が作成され、統合 URI に割り当てられます。ただし、その前にリージョンから *lambda-function-name* を選択するように求められます。

別の AWS のサービスアクションとの統合リクエストを設定する場合、統合リクエスト URI は ARN でもあります。これは、Lambda Invoke アクションとの統合と同様です。たとえば、Amazon S3 の GetBucket アクションとの統合の場合、統合リクエスト URI は以下の形式の ARN です。

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

統合リクエスト URI は、アクションを指定するためのパス規約です。ここで、*{bucket}* はバケット名のプレースホルダーです。または、AWS のサービスアクションをその名前で参照することもできます。アクション名を使用すると、Amazon S3 の GetBucket アクションの統合リクエスト URI は以下のようになります。

```
arn:aws:apigateway:api-region:s3:action/GetBucket
```

アクションベースの統合リクエスト URI では、*{bucket}* アクションの入力形式に従って、バケット名 ({ Bucket: "*{bucket}*" }) を統合リクエストボディ (GetBucket) で指定する必要があります。

AWS 統合の場合、認証情報を設定して、API Gateway が統合アクションを呼び出せるようにする必要もあります。新しい IAM ロールを作成するか既存のものを選択して、API Gateway がそのアクションを呼び出せるようにしたら、そのロールを ARN で指定できます。以下に示しているのは、この ARN の例です。

```
arn:aws:iam::account-id:role/iam-role-name
```

この IAM ロールには、アクションの実行を許可するポリシーが含まれている必要があります。また、そのロールを引き受ける信頼されたエンティティとして API Gateway を (ロールの信頼関係で) 宣言する必要があります。そのようなアクセス権限はアクション自体に対して付与できます。それらはリソースベースのアクセス権限と呼ばれます。Lambda 統合の場合、Lambda の addPermission アクションを呼び出して、リソースベースのアクセス許可を設定したら、API Gateway 統合リクエストで credentials を null に設定できます。

統合の基本設定については説明しました。詳細設定には、メソッドリクエストデータから統合リクエストデータへのマッピングが含まれます。統合レスポンスの基本設定について説明した後、「API Gateway コンソールを使用してリクエストとレスポンスのデータマッピングを設定する」で詳細設定を取り上げています。また、ペイロードを渡す方法とコンテンツエンコードを処理する方法も取り上げています。

API Gateway API 統合タイプの選択

使用する統合エンドポイントのタイプと、統合エンドポイントに対するデータの受け渡し方法に応じて、API 統合タイプを選択します。Lambda 関数の場合は、Lambda プロキシ統合または Lambda カスタム統合を使用できます。HTTP エンドポイントの場合、HTTP プロキシ統合または HTTP カスタム統合を使用できます。AWS サービスアクションの場合、非プロキシタイプの AWS 統合のみを使用します。API Gateway はモック統合もサポートしています。この場合、API Gateway はメソッドリクエストに応答する統合エンドポイントとして機能します。

Lambda カスタム統合は、AWS 統合の特殊なケースであり、統合エンドポイントが Lambda サービスの function-invoking アクションに対応します。

Integration リソースで type プロパティを設定することによって、プログラムで統合タイプを選択します。Lambda プロキシ統合の場合、その値は AWS_PROXY です。Lambda カスタム統合と他の

すべての AWS 統合の場合、その値は AWS です。HTTP プロキシ統合と HTTP 統合の場合、その値はそれぞれ HTTP_PROXY と HTTP です。Mock 統合の場合、type 値は MOCK です。

Lambda プロキシ統合は、1 つの Lambda 関数との合理化された統合設定をサポートしています。設定はシンプルで、既存の設定を破棄することなくバックエンドで拡張できます。このような理由から、Lambda 関数との統合を強くお勧めします。

これとは対照的に、Lambda カスタム統合では、入出力データ形式の要件が同様のさまざまな統合エンドポイントに対して、設定済みのマッピングテンプレートを再利用できます。この統合ではより詳細な設定が可能なため、より高度なアプリケーションシナリオにお勧めします。

同様に、HTTP プロキシ統合にも合理化された統合設定があり、既存の設定を破棄することなくバックエンドで拡張できます。HTTP カスタム統合では設定はより詳細になりますが、設定済みのマッピングテンプレートは他の統合エンドポイントに再利用できます。

以下のリストは、サポートされている統合タイプをまとめたものです。

- AWS: このタイプの統合では、API は AWS のサービスアクションを公開します。AWS 統合では、統合リクエストと統合レスポンスの両方を設定し、メソッドリクエストから統合リクエストへの、また統合レスポンスからメソッドレスポンスへの、データマッピングを設定する必要があります。
- AWS_PROXY: このタイプの統合では、さまざまな用途に柔軟に利用できる合理化された統合設定があり、API メソッドを Lambda 関数呼び出しアクションと統合できます。この統合は、クライアントと統合 Lambda 関数との間の直接的なやり取りに依存します。

  このタイプの統合は、Lambda プロキシ統合とも呼ばれ、お客様が統合リクエストまたは統合レスポンスを設定することはありません。API Gateway は、クライアントから受け取ったリクエストをバックエンドの Lambda 関数への入力として渡します。統合 Lambda 関数は、この形式の入力を受け取り、使用可能なすべてのソースからの入力 (リクエストヘッダー、URL パス変数、クエリ文字列パラメータ、該当する本文など) を解析します。その後、こちらの出力形式に従って結果を返します。

  これは、API Gateway を介した Lambda 関数呼び出しに適した統合タイプであり、関数呼び出しアクション以外の Lambda アクションなど、他の AWS のサービスアクションには適用されません。
- HTTP: このタイプの統合は、API がバックエンドの HTTP エンドポイントを公開することを可能にします。HTTP 統合 (HTTP カスタム統合とも呼ばれます) では、統合リクエストと統合レスポンスの両方を設定する必要があります。メソッドリクエストから統合リクエストへの、また統合レスポンスからメソッドレスポンスへの、データマッピングを設定する必要があります。

- HTTP_PROXY: HTTP プロキシ統合では、クライアントは 1 つの API メソッドで合理化された統合設定を使用して、バックエンド HTTP エンドポイントにアクセスできます。この場合、お客様が統合リクエストまたは統合レスポンスを設定することはありません。API Gateway は、クライアントから受け取ったリクエストを HTTP エンドポイントに渡し、HTTP エンドポイントから送り出されたレスポンスをクライアントに渡します。
- MOCK: このタイプの統合では、API Gateway はリクエストをさらにバックエンドに送信することなく、レスポンスを返します。このタイプの統合は、API のテストに便利です。バックエンドの使用料金が発生することなく、統合設定のテストに使用したり、API の共同開発に使用したりできるためです。

  共同開発では、チームは MOCK 統合を使用して、他のチームが所有する API コンポーネントのシミュレーションを設定することで、自分たちの開発成果を区分することもできます。また、CORS 関連のヘッダーを返して、API メソッドが CORS へのアクセスを許可するようにもできます。実際に、API Gateway コンソールは Mock 統合で OPTIONS メソッドを統合して CORS をサポートします。ゲートウェイレスポンス は Mock 統合の他の例です。

プロキシリソースとのプロキシ統合を設定する

API Gateway API でプロキシリソースとのプロキシ統合を設定するには、以下のタスクを実行します。

- greedy パス変数 {*proxy*+} を使用してプロキシリソースを作成します。
- プロキシリソースに ANY メソッドを設定します。
- HTTP または Lambda 統合タイプを使用してリソースおよびメソッドをバックエンドに統合します。

> **Note**
>
> greedy パス変数、ANY メソッド、およびプロキシ統合タイプは、よく一緒に使用されますが独立した機能です。特定の HTTP メソッドを greedy リソースに設定することも、プロキシ以外の統合タイプをプロキシリソースに適用することもできます。

API Gateway は、Lambda プロキシ統合または HTTP プロキシ統合を使用した処理方法に一定の制約と制限を設定しています。詳細については、「the section called “重要な注意点”」を参照してください。

> **Note**
>
> パススルーによるプロキシの統合の使用時に、ペイロードのコンテンツタイプを指定していない場合は、API Gateway からデフォルトの `Content-Type:application/json` ヘッダーが返されます。

プロキシリソースが最も便利なのは、HTTP プロキシ統合または Lambda プロキシ統合のいずれかを使用してバックエンドと統合されている場合です。

プロキシリソースとの HTTP プロキシ統合

API Gateway REST API の `HTTP_PROXY` によって指定される HTTP プロキシ統合は、メソッドリクエストをバックエンドの HTTP エンドポイントと統合するために使用します。この統合タイプでは、API Gateway は、一定の制約と制限に従って、フロントエンドとバックエンドとの間でリクエストとレスポンスの全体を渡すだけです。

> **Note**
>
> HTTP プロキシ統合では、複数の値を持つヘッダーやクエリ文字列がサポートされます。

HTTP プロキシ統合をプロキシリソースに適用するときは、単一の統合セットアップを使用して、HTTP バックエンドのエンドポイント階層の一部または全体を公開するように API を設定できます。たとえば、バックエンドのウェブサイトがルートノード (`/site`) からツリーノードで複数のブランチとして編成されているとします (`/site/`a_0`/`a_1`/.../`a_N、`/site/`b_0`/`b_1`/.../`b_M など)。`ANY` の URL パスを使用して `/api/{proxy+}` のプロキシリソースで `/site/{proxy}` メソッドをバックエンドのエンドポイントと統合した場合、単一の統合リクエストで、[a_0, a_1, ..., a_N, b_0, b_1, ...b_M, ...] のいずれかで HTTP オペレーション (GET、POST など) をサポートできます。代わりに、特定の HTTP メソッド (たとえば `GET`) にプロキシ統合を適用した場合、返される統合リクエストは、それらの任意のバックエンドノードに対して指定したオペレーション (つまり `GET`) で有効です。

プロキシリソースとの Lambda プロキシ統合

API Gateway REST API の `AWS_PROXY` によって指定される Lambda プロキシ統合は、メソッドリクエストをバックエンドの Lambda 関数と統合するために使用します。この統合タイプでは、API Gateway はデフォルトのマッピングテンプレートを適用してリクエスト全体を Lambda 関数に送信し、Lambda 関数の出力を HTTP レスポンスに変換します。

同様に、Lambda プロキシ統合を /api/{proxy+} のプロキシリソースに適用して単一の統合を設定し、バックエンドの Lambda 関数を /api の下位にある任意の API リソースでの変更に個別に対応させることができます。

API Gateway コンソールを使用して API 統合リクエストを設定する

API メソッドの設定でメソッドとその動作を定義します。メソッドを設定するには、メソッドが公開されるリソース (ルート ("/") を含む)、HTTP メソッド (GET、POST など)、ターゲットバックエンドとの統合方法を指定する必要があります。メソッドのリクエストと応答によって、APIはどのパラメータを受け取ることができ、応答はどのようなものかを明記し、呼び出し元アプリケーションとの取り決めが指定されます。

以下の手順では、API Gateway コンソールを使用して統合リクエストを作成する方法を示します。

トピック

- Lambda 統合をセットアップする
- HTTP 統合をセットアップする
- AWS のサービス統合をセットアップする
- Mock 統合をセットアップする

Lambda 統合をセットアップする

Lambda 関数統合を使用して、API を Lambda 関数と統合します。API レベルで、非プロキシ統合を作成する場合はこれが AWS 統合タイプになり、AWS_PROXYプロキシ統合を作成する場合は 統合タイプになります。

Lambda 統合をセットアップするには

1. [リソース] ペインで、[メソッドの作成] を選択します。
2. [メソッドタイプ] には [HTTP メソッド] を選択します。
3. [統合タイプ] で、[Lambda 関数] を選択します。
4. Lambda プロキシ統合を使用するには、[Lambda プロキシ統合] をオンにします。Lambda プロキシ統合の詳細については、「the section called “ Lambda プロキシ統合を理解する ”」を参照してください。
5. [Lambda 関数] に、Lambda 関数の名前を入力します。

API とは異なるリージョンで Lambda 関数を使用している場合は、ドロップダウンメニューからリージョンを選択し、Lambda 関数の名前を入力します。クロスアカウントの Lambda 関数を使用している場合は、関数 ARN を入力します。

6. 29 秒のデフォルトのタイムアウト値を使用するには、[デフォルトタイムアウト] をオンのままにします。カスタムのタイムアウトを設定するには、[デフォルトタイムアウト] を選択してから、タイムアウト値を `50 ~ 29000` ミリ秒の間で入力します。
7. (オプション) 以下のドロップダウンメニューを使用して、メソッドリクエストの設定を構成できます。[メソッドリクエストの設定] を選択し、メソッドリクエストを設定します。詳細については、「the section called “コンソールでメソッドリクエストを編集する”」のステップ 3 を参照してください。

   メソッドを作成した後で、メソッドリクエストの設定を構成することもできます。
8. [メソッドの作成] を選択します。

HTTP 統合をセットアップする

HTTP 統合を使用して、API を HTTP エンドポイントと統合します。API レベルで、これは HTTP 統合タイプです。

HTTP 統合をセットアップするには

1. [リソース] ペインで、[メソッドの作成] を選択します。
2. [メソッドタイプ] には [HTTP メソッド] を選択します。
3. [統合タイプ] で、[HTTP] を選択します。
4. HTTP プロキシ統合を使用するには、[HTTP プロキシ統合] をオンにします。HTTP プロキシ統合の詳細については、「the section called “API Gateway の HTTP プロキシ統合を設定する”」を参照してください。
5. [HTTP メソッド] で、HTTP バックエンドのメソッドに最も厳密に一致する HTTP メソッドタイプを選択します。
6. [エンドポイント URL] に、メソッドで使用する HTTP バックエンドの URL を入力します。
7. [コンテンツ処理] には、コンテンツ処理動作を選択します。
8. 29 秒のデフォルトのタイムアウト値を使用するには、[デフォルトタイムアウト] をオンのままにします。カスタムのタイムアウトを設定するには、[デフォルトタイムアウト] を選択してから、タイムアウト値を `50 ~ 29000` ミリ秒の間で入力します。

9. (オプション) 以下のドロップダウンメニューを使用して、メソッドリクエストの設定を構成できます。[メソッドリクエストの設定] を選択し、メソッドリクエストを設定します。詳細については、「the section called “コンソールでメソッドリクエストを編集する”」のステップ 3 を参照してください。

   メソッドを作成した後で、メソッドリクエストの設定を構成することもできます。

10. [メソッドの作成] を選択します。

AWS のサービス統合をセットアップする

AWS のサービス統合を使用して、API を AWS のサービスと直接統合します。API レベルで、これは AWS 統合タイプです。

次のいずれかを実行するために IAM API Gateway を作成するには:

- 新しい Lambda 関数の作成
- Lambda 関数へのリソース許可を設定します。
- その他の Lambda サービスアクションを実行します。

[AWS のサービス] を選択する必要があります。

AWS のサービス統合をセットアップするには

1. [リソース] ペインで、[メソッドの作成] を選択します。
2. [メソッドタイプ] には [HTTP メソッド] を選択します。
3. [統合タイプ] で、[AWS のサービス] を選択します。
4. [AWS リージョン] で、このメソッドがアクションの呼び出しに使用する AWS リージョンを選択します。
5. [AWS のサービス] で、このメソッドが呼び出す AWS のサービスを選択します。
6. [AWS サブドメイン] に、AWS のサービスで使用されるサブドメインを入力します。通常、このフィールドは空欄にします。一部の AWS のサービスでは、ホストの一部としてサブドメインをサポートすることができます。可用性と詳細は、サービスのドキュメントを参照してください。
7. [HTTP メソッド] で、アクションに対応する HTTP メソッドタイプを選択します。適切な HTTP メソッドタイプについては、[AWS のサービス] で選択した AWS サービスの API リファレンスドキュメントを参照してください。

8. [アクションタイプ] には、API アクションを使用する場合は [アクション名の使用] を、カスタムリソースパスを使用する場合は [パス上書きの使用] を選択します。利用できるアクションとカスタムリソースパスについては、[AWS のサービス] で選択した AWS のサービスに関する API リファレンスドキュメントを参照してください。
9. [アクション名] または [パス上書き] のいずれかを入力します。
10. [実行ロール] には、メソッドがアクションの呼び出しに使用する IAM ロールの ARN を入力します。

    IAM ロールを作成するには、the section called “ステップ 1: AWS のサービスプロキシの実行ロールを作成する” の指示を使用できます。以下の形式のアクセスポリシーを、必要な数のアクションおよびリソースステートメントと共に指定します。

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

    アクションおよびリソースステートメントの構文については、[AWS のサービス] で選択した AWS サービスのドキュメントを参照してください。

    IAM ロールの信頼関係で、以下のように指定して、API Gateway が AWS アカウントに代わってアクションを実行できるようにます。

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

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

11. 29 秒のデフォルトのタイムアウト値を使用するには、[デフォルトタイムアウト] をオンのままにします。カスタムのタイムアウトを設定するには、[デフォルトタイムアウト] を選択してから、タイムアウト値を `50 ~ 29000` ミリ秒の間で入力します。
12. (オプション) 以下のドロップダウンメニューを使用して、メソッドリクエストの設定を構成できます。[メソッドリクエストの設定] を選択し、メソッドリクエストを設定します。詳細については、「the section called “コンソールでメソッドリクエストを編集する”」のステップ 3 を参照してください。

    メソッドを作成した後で、メソッドリクエストの設定を構成することもできます。
13. [メソッドの作成] を選択します。

Mock 統合をセットアップする

API Gateway をバックエンドとして使用して静的レスポンスを返す場合は、Mock 統合を使用します。API レベルで、これは `MOCK` 統合タイプです。一般的に、API が最終版ではないものの、テスト用に API レスポンスを生成して関連チームのブロックを解除する場合は、`MOCK` 統合を使用します。`OPTION` メソッドの場合、API Gateway は、適用された API リソースに CORS が有効のヘッダーを返すように `MOCK` 統合をデフォルトに設定します。

Mock 統合をセットアップするには

1. [リソース] ペインで、[メソッドの作成] を選択します。
2. [メソッドタイプ] には [HTTP メソッド] を選択します。
3. [統合タイプ] で、[Mock] を選択します。
4. (オプション) 以下のドロップダウンメニューを使用して、メソッドリクエストの設定を構成できます。[メソッドリクエストの設定] を選択し、メソッドリクエストを設定します。詳細については、「the section called “コンソールでメソッドリクエストを編集する”」のステップ 3 を参照してください。

    メソッドを作成した後で、メソッドリクエストの設定を構成することもできます。

5. [メソッドの作成] を選択します。

API Gateway で統合レスポンスを設定する

非プロキシ統合の場合、1 つ以上の統合レスポンスを設定し、デフォルトのレスポンスにしたら、バックエンドから返される結果をクライアントに渡すように定義する必要があります。統合レスポンスデータとメソッドレスポンスデータで形式が異なる場合は、結果をそのまま渡すか、統合レスポンスデータをメソッドレスポンスデータに変換するかを選択できます。

プロキシ統合の場合、API Gateway は自動的にバックエンド出力を HTTP レスポンスとしてクライアントに渡します。この場合、お客様が統合レスポンスもメソッドレスポンスも設定することはありません。

統合レスポンスを設定するには、以下の必須およびオプションのタスクを実行します。

1. 統合レスポンスデータがマップされるメソッドレスポンスの HTTP ステータスコードを指定します。これは必須です。
2. この統合レスポンスが表すバックエンド出力に一致する正規表現を定義します。この項目を空のままにした場合、レスポンスは、未設定のレスポンスのキャッチに使用されるデフォルトのレスポンスになります。
3. 必要に応じて、指定した統合レスポンスパラメータを特定のメソッドレスポンスパラメータにマップするための、キーと値のペアで構成されるマッピングを宣言します。
4. 必要に応じて、特定の統合レスポンスペイロードを、指定したメソッドレスポンスペイロードに変換するための、本文マッピングテンプレートを追加します。
5. 必要に応じて、バイナリペイロードの型変換の処理方法を指定します。

統合レスポンスは、バックエンドレスポンスをカプセル化する HTTP レスポンスです。HTTP エンドポイントの場合、バックエンドレスポンスは HTTP レスポンスです。統合レスポンスのステータスコードは、バックエンドが返すステータスコードであり、統合レスポンスの本文は、バックエンドが返すペイロードです。Lambda エンドポイントの場合、バックエンドレスポンスは Lambda 関数から返される出力です。Lambda 統合では、Lambda 関数出力は `200 OK` レスポンスとして返されます。ペイロードには、結果が JSON データ (JSON 文字列または JSON オブジェクト) として含まれたり、エラーメッセージが JSON オブジェクトとして含まれたりします。[selectionPattern] プロパティに正規表現を割り当てて、エラーレスポンスを該当する HTTP エラーレスポンスにマップできます。Lambda 関数のエラーレスポンスの詳細については、「API Gateway で Lambda エラーを処理する」を参照してください。Lambda プロキシ統合では、Lambda 関数は以下の形式で出力を返す必要があります。

```
{
    statusCode: "...",            // a valid HTTP status code
    headers: {
        custom-header: "..."      // any API-specific custom header
    },
    body: "...",                  // a JSON string.
    isBase64Encoded:  true|false  // for binary support
}
```

Lambda 関数レスポンスを適切な HTTP レスポンスにマップする必要はありません。

結果をクライアントに返すには、エンドポイントレスポンスを対応するメソッドレスポンスにそのまま渡すように、統合レスポンスを設定します。または、エンドポイントレスポンスデータをメソッドレスポンスデータにマップすることもできます。マップできるレスポンスデータとしては、レスポンスステータスコード、レスポンスヘッダーパラメータ、レスポンス本文などがあります。返されたステータスコードに対してメソッドレスポンスが定義されていない場合、API Gateway は 500 エラーを返します。詳細については、「マッピングテンプレートを使用して、API のリクエストおよびレスポンスパラメータとステータスコードをオーバーライドする」を参照してください。

API Gateway で Lambda 統合を設定する

Lambda プロキシ統合または Lambda 非プロキシ (カスタム) 統合を使用して、API メソッドを Lambda 関数に統合できます。

Lambda プロキシ統合では、必要なセットアップを簡単に行えます。統合の HTTP メソッドを POST に設定し、統合エンドポイント URI を特定の Lambda 関数のアクションを呼び出す Lambda 関数の ARN に設定して、ユーザーに代わって Lambda 関数を呼び出す許可を API Gateway に付与します。

Lambda 非プロキシ統合では、プロキシ統合のセットアップステップに加えて、受信リクエストデータがどのように統合リクエストにマッピングされるか、統合レスポンスデータの結果がメソッドレスポンスにどのようにマッピングされるかを指定します。

トピック

- API Gateway で Lambda プロキシ統合を設定する
- API Gateway で Lambda カスタム統合を設定する
- バックエンド Lambda 関数の非同期呼び出しをセットアップする
- API Gateway で Lambda エラーを処理する

API Gateway で Lambda プロキシ統合を設定する

トピック

API Gateway Lambda プロキシの統合について理解する

Amazon API Gateway Lambda プロキシ統合は、単一の API メソッドのセットアップで API を構築するシンプル、強力、高速なメカニズムです。Lambda プロキシ統合は、クライアントが単一の Lambda 関数をバックエンドで呼び出すことを可能にします。この関数は、他の Lambda 関数の呼び出しを含め、他の AWS のサービスのさまざまなリソースや機能にアクセスします。

Lambda プロキシ統合では、クライアントが API リクエストを送信すると、API Gateway は、統合された Lambda 関数にイベントオブジェクトを渡します。ただし、リクエストパラメータの順序は保持されません。このリクエストデータには、リクエストヘッダー、クエリ文字列パラメータ、URL パス変数、ペイロード、および API 設定データが含まれます。設定データには、現在のデプロイステージ名、ステージ変数、ユーザー ID、または承認コンテキスト (存在する場合) を含めることができます。バックエンド Lambda 関数では、受信リクエストデータを解析して、返すレスポンスを決定します。API Gateway が Lambda 出力を API レスポンスとしてクライアントに渡すには、Lambda 関数は結果をこの形式で返す必要があります。

API Gateway は Lambda プロキシ統合でクライアントとバックエンド Lambda 関数間にあまり介入しないため、クライアントと統合された Lambda 関数は、API の既存の 統合セットアップを損なうことなく、相互の変更に適応できます。これを有効にするには、クライアントはバックエンド Lambda 関数が制定したアプリケーションプロトコルに従う必要があります。

任意の API メソッドで Lambda プロキシ統合をセットアップできます。しかし、Lambda プロキシ統合は、汎用的なプロキシリソースが含まれる API メソッドに設定されていると、より強力です。汎用的なプロキシリソースは、特別なテンプレートパス変数である `{proxy+}` 、キャッチオールである `ANY` メソッドプレースホルダー、または両方によって示すことができます。クライアントは、受信リクエストのバックエンド Lambda 関数に、入力をリクエストパラメータまたは適切なペイ

ロードとして渡すことができます。リクエストパラメータには、ヘッダー、URL パス変数、クエリ文字列パラメータ、および適切なペイロードが含まれます。統合された Lambda 関数は、リクエストを処理して、必要な入力がない場合に意味のあるエラーメッセージでクライアントに応答する前に、すべての入力ソースを検証します。

ANY の汎用的な HTTP メソッドと {proxy+} の汎用的なリソースと統合されている API メソッドを呼び出す際、クライアントは ANY の代わりに特定の HTTP メソッドを持つリクエストを送信します。クライアントはさらに、{proxy+} の代わりに特定の URL パスを指定し、要求されるヘッダー、クエリ文字列パラメータ、または適切なペイロードを含めます。

次のリストは、Lambda プロキシ統合での異なる API メソッドのランタイム動作を要約しています。

- ANY /{proxy+}: クライアントは特定の HTTP メソッドを選択し、特定のリソースパス階層を設定する必要があり、Lambda 関数にデータを入力として渡すために任意のヘッダー、クエリ文字列パラメータ、および適切なペイロードを設定できます。
- ANY /res: クライアントは特定の HTTP メソッドを選択する必要があり、Lambda 関数にデータを入力として渡すために任意のヘッダー、クエリ文字列パラメータ、および適切なペイロードを設定できます。
- GET|POST|PUT|... /{proxy+}: クライアントは特定のリソースパス階層を設定することができ、Lambda 関数にデータを入力として渡すために任意のヘッダー、クエリ文字列パラメータ、および適切なペイロードを設定できます。
- GET|POST|PUT|... /res/{path}/...: クライアントは ({path} 変数に) 特定のパスセグメントを選択する必要があり、Lambda 関数に入力データを渡すために任意のリクエストヘッダー、クエリ文字列パラメータ、および適切なペイロードを設定できます。
- GET|POST|PUT|... /res: クライアントは Lambda 関数に入力データを渡すために任意のリクエストヘッダー、クエリ文字列パラメータ、および適切なペイロードを設定できます。

{proxy+} のプロキシリソースと {custom} のカスタムリソースの両方が、テンプレートパス変数として表現されています。ただし、{proxy+} はパス階層に沿った任意のリソースを参照できますが、{custom} は特定のパスセグメントのみを参照します。たとえば、食料品店はオンラインの製品インベントリを部門名、農産物カテゴリ、および製品タイプで整理しています。食料品店のウェブサイトは、カスタムリソースの次のテンプレートパス変数によって、利用可能な製品を表すことができます: /{department}/{produce-category}/{product-type}。たとえば、リンゴは /produce/fruit/apple で表され、ニンジンは /produce/vegetables/carrot で表されます。また、/{proxy+} を使用して、顧客がオンラインストアで買い物中に検索できる任意の部門、農産

物カテゴリ、または製品タイプを表すことができます。たとえば、/{proxy+} は次のいずれかのアイテムを参照できます。

- /produce
- /produce/fruit
- /produce/vegetables/carrot

顧客が利用可能な製品、その農産物カテゴリ、および関連するストア部門を検索できるようにするには、GET /{proxy+} の単一メソッドを読み取り専用アクセス権限で公開することができます。同様に、責任者が produce 部門のインベントリをアップデートできるようにするには、別の PUT /produce/{proxy+} の単一メソッドを読み取り/書き込みアクセス権限でセットアップできます。レジ係が野菜の合計を更新できるようにするには、POST /produce/vegetables/{proxy+} メッセージを読み取り/書き込みアクセス権限でセットアップできます。店長が任意のアクションを任意の利用可能な製品に実行できるようにするには、オンラインストア開発者は ANY /{proxy+} メソッドを読み取り/書き込みアクセス権限で公開できます。いずれの場合も、実行時には、顧客または従業員は選択された部門のあるタイプの特定の製品、製品部門の特定の農産物カテゴリ、または特定の部門を選択する必要があります。

API Gateway プロキシ統合のセットアップの詳細については、「プロキシリソースとのプロキシ統合を設定する」を参照してください。

プロキシ統合は、クライアントがバックエンド要件のより詳細な知識を持っていることを要求します。したがって、最適なアプリケーションパフォーマンスとユーザーエクスペリエンスを確保するために、バックエンド開発者はバックエンドの要件をクライアント開発者に明確に伝え、要件が満たされない場合は堅牢なエラーフィードバックメカニズムを提供する必要があります。

複数値のヘッダーとクエリ文字列パラメータのサポート

API Gateway は、同じ名前を持つ複数のヘッダーやクエリ文字列パラメータをサポートするようになりました。複数値ヘッダーや単一値ヘッダー、およびパラメータを、同じリクエストとレスポンスで組み合わせることができます。詳細については、プロキシ統合のための Lambda 関数の入力形式およびプロキシ統合のための Lambda 関数の出力形式を参照してください。

プロキシリソースに Lambda プロキシ統合をセットアップする

プロキシリソースに Lambda プロキシ統合タイプをセットアップするには、greedy パスパラメータ (/parent/{proxy+} など) を使用して API リソースを作成し、このリソースを arn:aws:lambda:us-west-2:123456789012:function:SimpleLambda4ProxyResource

メソッドで Lambda 関数のバックエンド (ANY など) と統合します。greedy パスパラメータは、API リソースパスの末尾にある必要があります。プロキシ以外のリソースと同様に、API Gateway コンソールを使用するか、OpenAPI 定義ファイルをインポートするか、API Gateway REST API を直接呼び出すことによって、プロキシリソースをセットアップできます。

以下の OpenAPI API 定義ファイルは、SimpleLambda4ProxyResource という名前の Lambda 関数を使用して統合された API とプロキシリソースの例を示しています。

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"
        }
      }
    }
  }
}
```

Lambda プロキシ統合では、ランタイムに API Gateway は受信リクエストを Lambda 関数の入力 `event` パラメータにマッピングします。入力にはリクエストメソッド、パス、ヘッダー、クエリ文字列パラメータ、ペイロード、関連コンテキスト、定義済みステージ変数が含まれています。入力形式については「プロキシ統合のための Lambda 関数の入力形式」で説明しています。API Gateway が Lambda 出力を HTTP レスポンスに正常にマッピングするには、Lambda 関数は、「プロキシ統合のための Lambda 関数の出力形式」で説明されている形式で結果を出力する必要があります。

ANY メソッドによるプロキシリソースの Lambda プロキシ統合では、単一のバックエンド Lambda 関数が、プロキシリソースを通じてすべてのリクエストのイベントハンドラーの役割を果たします。たとえば、トラフィックパターンを記録するために、プロキシリソースの URL パスで `/state/city/street/house` を使用してリクエストを送信することによって、モバイルデバイスから州、市、番地、建物などの場所情報を送信できます。バックエンドの Lambda 関数は、URL パスを解析し、場所情報のタプルを DynamoDB テーブルに挿入できます。

AWS CLI を使用して Lambda プロキシ統合をセットアップする

このセクションでは、AWS CLI を使用して Lambda プロキシ統合で API をセットアップする方法について説明します。

> **Note**
>
> API Gateway コンソールを使用してプロキシリソースに Lambda プロキシ統合を設定する詳しい手順については、「チュートリアル: Lambda プロキシ統合を使用した Hello World REST API の構築」を参照してください。

たとえば、次のサンプル Lambda 関数を API のバックエンドとして使用します。

```
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);
```

```
};
```

これをカスタムの Lambda 統合のセットアップと比較すると、この Lambda 関数への入力はリクエストパラメータと本文で表現できます。クライアントが同じ入力データを渡すことができるように、自由度は高くなっています。ここで、クライアントは、クエリ文字列パラメータ、ヘッダ、または本文プロパティとして Greeter の名前を渡すことができます。この関数は、カスタムの Lambda 統合をサポートすることもできます。API のセットアップはより簡単です。メソッドレスポンスまたは統合レスポンスを設定することはありません。

AWS CLI を使用して Lambda プロキシ統合をセットアップするには

1. `create-rest-api` コマンドを呼び出して API を作成します。

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

   レスポンスの作成された API の `id` 値 (`te6si5ach7`) を書き留めます。

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

   このセクションでは、API `id` が必要です。

2. `get-resources` コマンドを呼び出して、ルートリソース `id` を取得します。

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

   正常なレスポンスは次のように表示されます。

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

ルートリソース id 値 (krznpq9xpg) を書き留めます。これは次のステップおよび後で必要になります。

3. create-resource を呼び出して、/greeting の API Gateway リソースを作成します。

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

正常に終了すると、レスポンスは以下のようになります。

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

作成された {proxy+} リソースの id 値 (2jf6xt) を書き留めます。次のステップでは、/{proxy+} リソース上にメソッドを作成する必要があります。

4. put-method を呼び出して、ANY の ANY /{proxy+} メソッドリクエストを作成します。

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

正常に終了すると、レスポンスは以下のようになります。

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

この API メソッドでは、クライアントはバックエンドの Lambda 関数からお知らせを受信または送信できます。

5. put-integration を呼び出して、ANY /{proxy+} という名前の Lambda 関数で HelloWorld メソッドの統合をセットアップします。この関数は、"Hello, {name}!" パラメータが提供されている場合は greeter、クエリ文字列パラメータが設定されていない場合は "Hello, World!" のメッセージでリクエストに応答します。

```
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**
>
> Lambda 統合では、関数呼び出しの Lambda サービスアクションの仕様に従って、統合リクエストに POST の HTTP メソッドを使用する必要があります。apigAwsProxyRole の IAM ロールは、apigateway サービスが Lambda 関数を呼び出せるようにするポリシーが必要です。IAM 許可の詳細については、「the section called “ API を呼び出すための API Gateway アクセス許可モデル”」を参照してください。

正しい出力は次の例のようになります。

```
{
    "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"
}
```

credentials の IAM ロールを指定する代わりに、add-permission コマンドを呼び出して、リソースベースのアクセス許可を追加することができます。これは、API Gateway コンソールが行うことです。

6. create-deployment を呼び出して、API を test ステージにデプロイします。

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

7. ターミナルで次の cURL コマンドを使用して API をテストします。

?greeter=jane のクエリ文字列パラメータを使用して API を呼び出します。

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

greeter:jane のヘッダーパラメータを使用して API を呼び出します。

```
curl -X GET https://te6si5ach7.execute-api.us-west-2.amazonaws.com/test/hi \
  -H 'content-type: application/json' \
  -H 'greeter: jane'
```

{"greeter":"jane"} の本文を使用して API を呼び出します。

```
curl -X POST https://te6si5ach7.execute-api.us-west-2.amazonaws.com/test/hi \
  -H 'content-type: application/json' \
  -d '{ "greeter": "jane" }'
```

すべてのケースで、出力は、次のレスポンス本文を持つ 200 レスポンスです。

```
Hello, jane!
```

プロキシ統合のための Lambda 関数の入力形式

Lambda プロキシ統合では、API Gateway がクライアントリクエスト全体をバックエンド Lambda 関数の入力 event パラメータにマップします。次の例は、API Gateway が Lambda プロキシ統合に送信するイベントの構造を示しています。

```
{
```

```
  "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**
>
> 入力値は、次のようになります。
>
> - `headers` キーには、単一値のヘッダーのみを含めることができます。
> - `multiValueHeaders` キーには、複数値のヘッダーや単一値のヘッダーを含めることができます。

- `headers` と `multiValueHeaders` の両方の値を指定した場合、API Gateway はそれらを単一のリストにマージします。同じキーと値のペアが両方で指定された場合にのみ、`multiValueHeaders` の値が、マージされたリストに表示されます。

バックエンド Lambda 関数への入力では、`requestContext` オブジェクトはキーと値のペアのマップです。各ペアにおいて、キーは $context 変数プロパティの名前であり、値はそのプロパティの値です。API Gateway は、マップに新しいキーを追加する場合があります。

有効になっている機能に応じて、`requestContext` マップは API ごとに異なる場合があります。たとえば、前述の例では、認証タイプが指定されていないため、`$context.authorizer.*` や `$context.identity.*` プロパティは存在しません。認証タイプが指定されると、API Gateway は、認証されたユーザー情報を `requestContext.identity` オブジェクト内の統合エンドポイントに次のように渡します。

- 認証タイプが `AWS_IAM` の場合、認証されるユーザー情報には `$context.identity.*` プロパティが含まれます。
- 認証タイプが `COGNITO_USER_POOLS` (Amazon Cognito オーソライザー) の場合、認証されるユーザー情報には `$context.identity.cognito*` および `$context.authorizer.claims.*` プロパティが含まれます。
- 認証タイプが `CUSTOM` (Lambda オーソライザー) の場合、認証されるユーザー情報には `$context.authorizer.principalId` およびその他の該当する `$context.authorizer.*` プロパティが含まれます。

プロキシ統合のための Lambda 関数の出力形式

Lambda プロキシ統合では、以下の JSON 形式に従って出力を返すために、API Gateway はバックエンドの Lambda 関数を必要とします。

```
{
    "isBase64Encoded": true|false,
    "statusCode": httpStatusCode,
    "headers": { "headerName": "headerValue", ... },
    "multiValueHeaders": { "headerName": ["headerValue", "headerValue2", ...], ... },
    "body": "..."
}
```

出力では、次のようになります。

- 余分なレスポンスヘッダーが返されない場合、`headers` および `multiValueHeaders` キーは指定されません。
- `headers` キーには、単一値のヘッダーのみを含めることができます。
- `multiValueHeaders` キーには、複数値のヘッダーや単一値のヘッダーを含めることができます。`multiValueHeaders` キーを使用して、単一値のヘッダーを含めて、すべてのヘッダーを指定することができます。
- `headers` と `multiValueHeaders` の両方の値を指定した場合、API Gateway はそれらを単一のリストにマージします。同じキーと値のペアが両方で指定された場合にのみ、`multiValueHeaders` の値が、マージされたリストに表示されます。

Lambda プロキシ統合に対して CORS を有効にするには、`Access-Control-Allow-Origin:`*`domain-name`* を出力 `headers` に追加する必要があります。*`domain-name`* は、任意のドメイン名に対して `*` にすることができます。出力 `body` は、メソッドレスポンスペイロードとしてフロントエンドにマーシャリングされます。`body` がバイナリ BLOB の場合、`isBase64Encoded` を `true` に設定し、`*/*` を [バイナリメディアタイプ] に設定することで、Base64 エンコード文字列としてエンコードできます。それ以外の場合は、`false` に設定するか、指定しないでおくことができます。

> **Note**
>
> バイナリサポートの有効化に関する詳細については、API Gateway コンソールを使用したバイナリサポートの有効化 を参照してください。サンプルの Lambda 関数については、「Lambda プロキシ統合からバイナリメディアを返す」を参照してください。

関数出力が別の形式である場合、API Gateway は `502 Bad Gateway` エラーレスポンスを返します。

Node.js での Lambda 関数でレスポンスを返すには、次のようなコマンドを使用できます。

- 成功した場合の結果を返すには、`callback(null, {"statusCode": 200, "body": "results"})` を呼び出します。
- 例外をスローするには、`callback(new Error('internal server error'))` を呼び出します。

- クライアント側エラーの場合 (必要なパラメータがない場合など)、callback(null, {"statusCode": 400, "body": "Missing parameters of ..."}) を呼び出して、例外をスローせずにエラーを返すことができます。

Node.js の Lambda async 関数では、同等の構文は次のようになります。

- 成功した場合の結果を返すには、return {"statusCode": 200, "body": "results"} を呼び出します。
- 例外をスローするには、throw new Error("internal server error") を呼び出します。
- クライアント側エラーの場合 (必要なパラメータがない場合など)、return {"statusCode": 400, "body": "Missing parameters of ..."} を呼び出して、例外をスローせずにエラーを返すことができます。

API Gateway で Lambda カスタム統合を設定する

カスタムの Lambda 統合をセットアップする方法を表示するには、GET /greeting?greeter={name} メソッドを公開する API Gateway API を作成して、Lambda 関数を呼び出します。API には、次の Lambda 関数の例のいずれかを使用します。

次の Lambda 関数の例のいずれかを使用します。

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
```

この関数は、"Hello, {name}!" パラメータの値が空白でない文字列の場合は、greeter のメッセージで応答します。"Hello, World!" の値が空の文字列の場合は、greeter のメッセージを返します。着信リクエストで greeter パラメータが設定されていない場合、この関数は "Missing the required greeter parameter." のエラーメッセージを返します。関数名を HelloWorld とします。

これは、Lambda コンソールで、または AWS CLI を使用して作成できます。このセクションでは、次の ARN を使用してこの関数を参照します。

```
arn:aws:lambda:us-east-1:123456789012:function:HelloWorld
```

バックエンドに Lambda 関数を設定すると、API のセットアップに進みます。

AWS CLI を使用して Lambda カスタム統合をセットアップするには

1. create-rest-api コマンドを呼び出して API を作成します。

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

レスポンスの作成された API の id 値 (te6si5ach7) を書き留めます。

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

このセクションでは、API id が必要です。

2. get-resources コマンドを呼び出して、ルートリソース id を取得します。

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

正常なレスポンスは次のとおりです。

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

ルートリソース id 値 (krznpq9xpg) を書き留めます。これは次のステップおよび後で必要になります。

3. create-resource を呼び出して、/greeting の API Gateway リソースを作成します。

```
aws apigateway create-resource --rest-api-id te6si5ach7 \
      --region us-west-2 \
      --parent-id krznpq9xpg \
      --path-part greeting
```

正常に終了すると、レスポンスは以下のようになります。

```
{
```

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

作成された greeting リソースの id 値 (2jf6xt) を書き留めます。次のステップでは、/greeting リソース上にメソッドを作成する必要があります。

4. put-method を呼び出して、GET /greeting?greeter={name} の API メソッドリクエストを作成します。

```
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
```

正常に終了すると、レスポンスは以下のようになります。

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

この API メソッドでは、クライアントはバックエンドの Lambda 関数からお知らせを受信できます。バックエンドは匿名の発信者または匿名でない発信者のいずれかを処理する必要があるため、この greeter パラメータはオプションです。

5. put-method-response を呼び出して、200 OK のメソッドリクエストに対する GET /greeting?greeter={name} レスポンスをセットアップします。

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

```
        --status-code 200
```

6. put-integration を呼び出して、GET /greeting?greeter={name} という名前の Lambda 関数で HelloWorld メソッドの統合をセットアップします。この関数は、"Hello, {name}!" パラメータが提供されている場合は greeter、クエリ文字列パラメータが設定されていない場合は "Hello, World!" のメッセージでリクエストに応答します。

```
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
```

ここで指定されたマッピングテンプレートは、greeter クエリ文字列パラメータを JSON ペイロードの greeter プロパティに変換します。Lambda 関数への入力を本文で表現する必要があるため、これが必要になります。

> **Important**
>
> Lambda 統合では、関数呼び出しの Lambda サービスアクションの仕様に従って、統合リクエストに POST の HTTP メソッドを使用する必要があります。uri パラメータは、関数呼び出しアクションの ARN です。
> 正しい出力は次の例のようになります。

```
{
    "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"
}
```

apigAwsProxyRole の IAM ロールは、apigateway サービスが Lambda 関数を呼び出せるようにするポリシーが必要です。credentials の IAM ロールを指定する代わりに、add-permission コマンドを呼び出して、リソースベースのアクセス許可を追加することができます。これは、API Gateway コンソールがこれらのアクセス許可を追加する方法です。

7. put-integration-response を呼び出して、Lambda 関数の出力を 200 OK メソッドレスポンスとしてクライアントに渡すように統合レスポンスをセットアップします。

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

選択パターンを空の文字列に設定すると、200 OK レスポンスがデフォルトになります。

正常なレスポンスは次のようになります。

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

8. create-deployment を呼び出して、API を test ステージにデプロイします。

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

9. ターミナルで次の cURL コマンドを使用して API をテストします。

```
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'
```

バックエンド Lambda 関数の非同期呼び出しをセットアップする

Lambda 非プロキシ (カスタム) 統合では、デフォルトでバックエンド Lambda 関数が同期的に呼び出されます。これは、ほとんどの REST API 操作に必要な動作です。ただし、一部のアプリケーションでは非同期で作業を実行する必要があります (バッチオペレーションまたは長時間レイテンシーオペレーションとして実行)。通常、これは別々のバックエンドコンポーネントで実行されます。この場合は、バックエンド Lambda 関数は非同期に呼び出され、フロントエンドの REST API メソッドは結果を返しません。

`'Event'` を Lambda 呼び出しタイプとして指定することで、Lambda 非プロキシ統合用の Lambda 関数を非同期的に呼び出すように設定できます。これは次のように行います。

API Gateway コンソールで Lambda 非同期呼び出しを設定する

すべての呼び出しを非同期にするには：

- [統合リクエスト] で、静的な値として `'Event'` を使用して `X-Amz-Invocation-Type` ヘッダーを追加します。

クライアントが、呼び出しが非同期か同期かを判断するには、次のようにします。

1. [メソッドリクエスト] で、`InvocationType` ヘッダーを追加します。
2. [統合リクエスト] で、マッピング式として `method.request.header.InvocationType` を使用して `X-Amz-Invocation-Type` ヘッダーを追加します。
3. クライアントは、非同期呼び出しには API リクエストに`InvocationType: Event` ヘッダーを含めるか、同期呼び出しには `InvocationType: RequestResponse` を含めることができます。

OpenAPI を使用した Lambda 非同期呼び出しの設定

すべての呼び出しを非同期にするには：

- `X-Amz-Invocation-Type` ヘッダーを x-amazon-apigateway-integration セクションに追加します。

```
"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"
        }
```

クライアントが、呼び出しが非同期か同期かを判断するには、次のようにします。

1. 任意の OpenAPI Path Item オブジェクトに次のヘッダーを追加します。

```
"parameters" : [ {
"name" : "InvocationType",
"in" : "header",
"schema" : {
  "type" : "string"
}
} ]
```

2. X-Amz-Invocation-Type ヘッダーを x-amazon-apigateway-integration セクションに追加します。

```
"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. クライアントは、非同期呼び出しには API リクエストに`InvocationType: Event` ヘッダーを含めるか、同期呼び出しには `InvocationType: RequestResponse` を含めることができます。

AWS CloudFormation を使用して Lambda 非同期呼び出しを設定する

以下の AWS CloudFormation テンプレートは、非同期呼び出し用に `AWS::ApiGateway::Method` を設定する方法を示しています。

すべての呼び出しを非同期にするには：

```
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'
```

クライアントが、呼び出しが非同期か同期かを判断するには、次のようにします。

```
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'
```

クライアントは、非同期呼び出しには API リクエストに`InvocationType: Event` ヘッダーを含めるか、同期呼び出しには `InvocationType: RequestResponse` を含めることができます。

API Gateway で Lambda エラーを処理する

Lambda カスタム統合の場合、統合レスポンスで Lambda によって返されたエラーを、クライアントの標準 HTTP エラーレスポンスにマップする必要があります。そうしないと、Lambda のエラーはデフォルトで `200 OK` レスポンスとして返されるため、API ユーザーは直感的に理解できません。

Lambda から返されるエラーには、標準エラーとカスタムエラーの 2 種類があります。API では、これらを異なる方法で処理する必要があります。

Lambda プロキシ統合では、Lambda は次の形式で出力を返す必要があります。

```
{
  "isBase64Encoded" : "boolean",
  "statusCode": "number",
```

```
  "headers": { ... },
  "body": "JSON string"
}
```

この出力で、通常、statusCode の 4XX はクライアントエラー、5XX はサーバーエラーです。API Gateway では、これらのエラーを処理する方法として、指定された statusCode に従って Lambda エラーを HTTP エラーレスポンスにマッピングします。API Gateway がクライアントへのレスポンスの一部としてエラータイプ (InvalidParameterException など) を渡すには、Lambda 関数がヘッダー ("X-Amzn-ErrorType":"InvalidParameterException" など) を headers プロパティに含める必要があります。

トピック

- API Gateway で標準の Lambda エラーを処理する
- API Gateway でカスタム Lambda エラーを処理する

API Gateway で標準の Lambda エラーを処理する

AWS Lambda の標準エラーは次の形式になります。

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

ここで、errorMessage はエラーの文字列式です。errorType は、言語に依存するエラーまたは例外タイプです。stackTrace は、エラーの発生につながったスタックトレースを示す文字列式のリストです。

たとえば、次の JavaScript (Node.js) Lambda 関数について考えてみます。

```
export const handler = function(event, context, callback) {
    callback(new Error("Malformed input ..."));
};
```

この関数は、`Malformed input ...` をエラーメッセージとする、次の Lambda 標準エラーを返します。

```
{
  "errorMessage": "Malformed input ...",
  "errorType": "Error",
  "stackTrace": [
    "export const handler (/var/task/index.js:3:14)"
  ]
}
```

同様に、次の例は同じ `Exception` をエラーメッセージとし、`Malformed input ...` をスローする Python Lambda 関数です。

```
def lambda_handler(event, context):
    raise Exception('Malformed input ...')
```

この関数は、次の Lambda 標準エラーを返します。

```
{
  "stackTrace": [
    [
      "/var/task/lambda_function.py",
      3,
      "lambda_handler",
      "raise Exception('Malformed input ...')"
    ]
  ],
  "errorType": "Exception",
  "errorMessage": "Malformed input ..."
}
```

`errorType` プロパティと `stackTrace` プロパティの値は、言語に依存します。この標準エラーは、`Error` オブジェクトの拡張または `Exception` クラスのサブクラスである各エラーオブジェクトにも適用されます。

Lambda の標準エラーをメソッドレスポンスにマッピングするには、まず該当する Lambda エラーの HTTP スタータスコードを確認する必要があります。次に、その HTTP ステータスコードに関連付けられた IntegrationResponse の `selectionPattern` プロパティで正規表現パターンを設定します。API Gateway コンソールの場合、この `selectionPattern` は各統合レスポンスの下の [統合レスポンス] セクションに、[Lambda エラーの正規表現] として表示されます。

> **Note**
>
> API Gateway は、レスポンスマッピングに Java パターンスタイルの正規表現を使用します。詳細については、Oracle ドキュメントの「パターン」を参照してください。

たとえば、新しい `selectionPattern` 式を設定するには、AWS CLI を使用して、次の put-integration-response コマンドを呼び出します。

```
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
```

メソッドレスポンスで対応するエラーコード (`400`) も必ず設定します。そうしないと、ランタイムに API Gateway から無効な設定エラーレスポンスがスローされます。

> **Note**
>
> ランタイムに、API Gateway は Lambda エラーの `errorMessage` を `selectionPattern` プロパティの正規表現のパターンと照合します。一致すると、API Gateway は Lambda エラーを、対応する HTTP ステータスコードの HTTP レスポンスとして返します。一致しない場合、API Gateway はエラーをデフォルトレスポンスとして返すか、デフォルトレスポンスが設定されていなければ、無効な設定例外をスローします。
>
> 特定のレスポンスで `selectionPattern` の値を `.*` に設定すると、このレスポンスはデフォルトのレスポンスとしてリセットされます。このような選択パターンは null (任意の未指定のエラーメッセージ) を含むすべてのエラーメッセージに一致するためです。その結果のマッピングにより、デフォルトのマッピングが上書きされます。

`selectionPattern` を使用して既存の AWS CLI 値を更新するには、update-integration-response オペレーションを呼び出し、`/selectionPattern` のパス値を `Malformed*` パターンの指定された正規表現式に置き換えます。

API Gateway コンソールを使用して `selectionPattern` 式を設定するには、指定した HTTP ステータスコードの統合レスポンスを設定または更新するときに、[Lambda エラーの正規表現] テキストボックスに式を入力します。

API Gateway でカスタム Lambda エラーを処理する

AWS Lambda では、前のセクションで説明した標準エラーの代わりにカスタムエラーオブジェクトを JSON 文字列として返すことができます。エラーは、任意の有効な JSON オブジェクトです。たとえば、次の JavaScript (Node.js) Lambda 関数はカスタムエラーを返します。

```
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));
};
```

`callback` を呼び出して関数を終了する前に、`myErrorObj` オブジェクトを JSON 文字列に変換する必要があります。そうしないと、`myErrorObj` は `"[object Object]"` の文字列として返されます。API のメソッドを前の Lambda 関数と統合すると、API Gateway は以下の内容をペイロードとする統合レスポンスを受け取ります。

```
{
    "errorMessage": "{\"errorType\":\"InternalServerError\",\"httpStatus\":500,
\"requestId\":\"e5849002-39a0-11e7-a419-5bb5807c9fb2\",\"trace\":{\"function\":
\"abc()\",\"line\":123,\"file\":\"abc.js\"}}"
}
```

通常の統合レスポンスと同様に、このエラーレスポンスはそのままメソッドレスポンスに渡すことができます。または、マッピングテンプレートを使用してペイロードを別の形式に変換することもできます。次の例は、ステータスコードが `500` のメソッドレスポンスの本文マッピングテンプレートです。

```
{
    errorMessage: $input.path('$.errorMessage');
}
```

このテンプレートは、カスタムエラーの JSON 文字列が含まれている統合レスポンス本文を、次のメソッドレスポンス本文に変換します。このメソッドレスポンス本文には、カスタムエラーの JSON オブジェクトが含まれています。

```
{
    "errorMessage" : {
        errorType : "InternalServerError",
        httpStatus : 500,
        requestId : context.awsRequestId,
        trace : {
            "function": "abc()",
            "line": 123,
            "file": "abc.js"
        }
    }
};
```

API の要件によっては、カスタムエラープロパティの一部または全部をメソッドレスポンスのヘッダーパラメータとして渡す必要があります。そのためには、統合レスポンス本文からメソッドレスポンスヘッダーにカスタムエラーマッピングを適用します。

たとえば、次の OpenAPI 拡張は、`errorMessage.errorType`、`errorMessage.httpStatus`、`errorMessage.trace.function`、および `errorMessage.trace` プロパティから、それぞれ `error_type`、`error_status`、`error_trace_function`、および `error_trace` ヘッダーへのマッピングを定義します。

```
"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"
          },
          ...
```

```
        }
    }
}
```

ランタイムに、API Gateway は integration.response.body パラメータを逆シリアル化してヘッダーマッピングを行います。ただし、この逆シリアル化は、Lambda のカスタムエラーレスポンスの本文からヘッダーへのマッピングにのみ適用され、$input.body を使用した本文から本文へのマッピングには適用されません。これらのカスタムエラーの本文からヘッダーへのマッピングでは、クライアントはメソッドレスポンスの一部として以下のヘッダーを受け取ります (error_status、error_trace、error_trace_function、および error_type をメソッドリクエストで宣言している場合)。

```
"error_status":"500",
"error_trace":"{\"function\":\"abc()\",\"line\":123,\"file\":\"abc.js\"}",
"error_trace_function":"abc()",
"error_type":"InternalServerError"
```

統合レスポンス本文の errorMessage.trace プロパティは複合プロパティです。これは JSON 文字列として error_trace ヘッダーにマッピングされます。

API Gateway で HTTP 統合を設定する

HTTP プロキシ統合または HTTP カスタム統合を使用して、API メソッドを HTTP エンドポイントに統合できます。

API Gateway は、次のエンドポイントポートをサポートします。80、443、および 1024-65535 です。

プロキシ統合では、セットアップは簡単です。コンテンツのエンコーディングやキャッシングが不要な場合は、バックエンド要件に従って HTTP メソッドと HTTP エンドポイント URI を設定するだけで済みます。

カスタム統合では、セットアップは複雑になります。プロキシ統合のセットアップ手順に加えて、受信リクエストデータがどのように統合リクエストにマッピングされるか、統合レスポンスデータの結果がメソッド応答にどのようにマッピングされるかを指定する必要があります。

トピック

- API Gateway の HTTP プロキシ統合を設定する
- API Gateway の HTTP カスタム統合をセットアップする

API Gateway の HTTP プロキシ統合を設定する

HTTP プロキシ統合タイプを使用してプロキシリソースをセットアップするには、greedy パスパラメータ (`/parent/{proxy+}` など) を使用して API リソースを作成し、このリソースを `https://petstore-demo-endpoint.execute-api.com/petstore/{proxy}` メソッドで HTTP バックエンドのエンドポイント (ANY など) と統合します。greedy パスパラメーターは、リソースパスの末尾にある必要があります。

非プロキシリソースと同様に、API Gateway コンソールを使用するか、OpenAPI 定義ファイルをインポートするか、API Gateway REST API を直接呼び出すことによって、プロキシリソースに HTTP プロキシ統合をセットアップできます。API Gateway コンソールを使用して HTTP 統合でプロキシリソースを設定する詳しい手順については、「チュートリアル: HTTP プロキシ統合を使用して REST API をビルドする」を参照してください。

以下の OpenAPI 定義ファイルは、PetStore ウェブサイトに統合された API とプロキシリソースの例を示しています。

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"
        }
      }
    }
  }
}
```

この例では、キャッシュのキーは、プロキシリソースの `method.request.path.proxy` パスパラメータで宣言されます。これにより、API Gateway コンソールを使用して API を作成するときのデフォルト設定です。API ベースパス (`/test`、ステージに対応) はウェブサイトの PetStore ページ (/

petstore) にマッピングされます。単一の統合リクエストは、API の greedy パス変数とキャッチオールの ANY メソッドを使用して、PetStore ウェブサイト全体をミラーリング処理します。以下の例に、このミラーリングを示しています。

- **ANY** を **GET**、**{proxy+}** を **pets** に設定

  フロントエンドから開始されたメソッドリクエスト:

  ```
  GET https://4z9giyi2c1.execute-api.us-west-2.amazonaws.com/test/pets HTTP/1.1
  ```

  バックエンドに送信された統合リクエスト:

  ```
  GET http://petstore-demo-endpoint.execute-api.com/petstore/pets HTTP/1.1
  ```

  ANY メソッドの実行時インスタンスとプロキシリソースの両方が有効です。呼び出しは 200 OK レスポンスと、バックエンドから返されたペットの最初のバッチが含まれるペイロードを返します。

- **ANY** を **GET**、**{proxy+}** を **pets?type=dog** に設定

  ```
  GET https://4z9giyi2c1.execute-api.us-west-2.amazonaws.com/test/pets?type=dog
   HTTP/1.1
  ```

  バックエンドに送信された統合リクエスト:

  ```
  GET http://petstore-demo-endpoint.execute-api.com/petstore/pets?type=dog HTTP/1.1
  ```

  ANY メソッドの実行時インスタンスとプロキシリソースの両方が有効です。呼び出しは 200 OK レスポンスと、バックエンドから返された特定の犬の最初のバッチが含まれるペイロードを返します。

- **ANY** を **GET**、**{proxy+}** を **pets/{petId}** に設定

  フロントエンドから開始されたメソッドリクエスト:

  ```
  GET https://4z9giyi2c1.execute-api.us-west-2.amazonaws.com/test/pets/1 HTTP/1.1
  ```

  バックエンドに送信された統合リクエスト:

```
GET http://petstore-demo-endpoint.execute-api.com/petstore/pets/1 HTTP/1.1
```

ANY メソッドの実行時インスタンスとプロキシリソースの両方が有効です。呼び出しは 200 OK レスポンスと、バックエンドから返された特定のペットが含まれるペイロードを返します。

- **ANY を POST、{proxy+} を pets に設定**

  フロントエンドから開始されたメソッドリクエスト:

```
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
}
```

  バックエンドに送信された統合リクエスト:

```
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 メソッドの実行時インスタンスとプロキシリソースの両方が有効です。呼び出しは 200 OK レスポンスと、バックエンドから返された新しく作成したペットが含まれるペイロードを返します。

- **ANY を GET、{proxy+} を pets/cat に設定**

  フロントエンドから開始されたメソッドリクエスト:

```
GET https://4z9giyi2c1.execute-api.us-west-2.amazonaws.com/test/pets/cat
```

  バックエンドに送信された統合リクエスト:

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

プロキシリソースパスの実行時インスタンスが、バックエンドのエンドポイントに対応しておらず、結果としいて生成されるリクエストは無効です。その結果、400 Bad Request レスポンスが次のエラーメッセージとともに返されます。

```
{
  "errors": [
    {
      "key": "Pet2.type",
      "message": "Missing required field"
    },
    {
      "key": "Pet2.price",
      "message": "Missing required field"
    }
  ]
}
```

- **ANY を GET、{proxy+} を null に設定**

  フロントエンドから開始されたメソッドリクエスト:

  ```
  GET https://4z9giyi2c1.execute-api.us-west-2.amazonaws.com/test
  ```

  バックエンドに送信された統合リクエスト:

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

  対象のリソースはプロキシリソースの親ですが、ANY メソッドを実行時インスタンスは、そのリソースの API に定義されていません。その結果、この GET リクエストは、403 Forbidden レスポンスと、API Gateway から返された Missing Authentication Token エラーを返します。API が親リソース (ANY) で GET または / メソッドを公開している場合、呼び出しからは 404 Not Found レスポンスと、バックエンドから返された Cannot GET /petstore メッセージが返されます。

クライアントリクエストの場合、対象のエンドポイント URL が無効であるか、HTTP 動詞が有効であってもサポートされていない場合、バックエンドは `404 Not Found` レスポンスを返します。サポートされていない HTTP メソッドの場合は、`403 Forbidden` レスポンスが返されます。

API Gateway の HTTP カスタム統合をセットアップする

HTTP カスタム統合により、API メソッドと API 統合の間を行き来するデータと行き来の方法をより詳細に制御できます。これにはデータマッピングを使用します。

メソッドリクエストのセットアップの一環として、Method リソースの requestParameters プロパティを設定します。これにより、クライアントからプロビジョニングされるメソッドリクエストパラメーターのうち、バックエンドにディスパッチされる前に統合リクエストパラメーターや該当する本文プロパティにマッピングされるものが宣言されます。次に、統合リクエストのセットアップの一環として、対応する Integration リソースで requestParameters プロパティを設定し、パラメータ間のマッピングを指定します。また、requestTemplates プロパティを設定し、サポートされているコンテンツタイプにつき 1 つずつ、マッピングテンプレートを指定します。マッピングテンプレートは、メソッドリクエストパラメーターや本文を統合リクエストボディにマッピングします。

同様に、メソッドレスポンスのセットアップの一環として、MethodResponse リソースで responseParameters プロパティを設定します。これにより、クライアントにディスパッチされるメソッドレスポンスパラメーターのうち、統合レスポンスパラメーターからまたはバックエンドから返された該当する本文プロパティからマッピングされるものが宣言されます。次に、統合レスポンスのセットアップの一環として、対応する IntegrationResponse リソースで responseParameters プロパティを設定し、パラメータからパラメータへのマッピングを指定します。また、responseTemplates マップを設定し、サポートされているコンテンツタイプごとにマッピングテンプレートを 1 つずつ指定します。マッピングテンプレートは、統合レスポンスパラメーターや統合レスポンス本文のプロパティをメソッドレスポンス本文にマッピングします。

マッピングテンプレートの設定の詳細については、「REST API のデータ変換の設定」を参照してください。

API Gateway プライベート統合の設定

API Gateway のプライベートな統合により、VPC 内にある HTTP/HTTPS リソースを VPC 外のクライアントがアクセスできるように簡単に公開できます。プライベート VPC リソースへのアクセスを VPC 境界を超えて拡張するために、プライベート統合で API を作成できます。API Gateway がサポートする認証方法より、API へのアクセスを制御できます。

プライベート統合を作成するには、まず Network Load Balancer を作成する必要があります。Network Load Balancer には、VPC 内のリソースにリクエストをルーティングするリスナーが

必要です。API の可用性を高めるには、Network Load Balancer が、AWS リージョン内の複数のアベイラビリティーゾーンにあるリソースにトラフィックをルーティングするようにします。その後、API と Network Load Balancer を接続するために使用する VPC リンクを作成します。VPC リンクを作成したら、プライベート統合を作成して、VPC リンクと Network Load Balancer を介して API から VPC 内のリソースにトラフィックをルーティングします。

> **Note**
>
> Network Load Balancer と API は、同じ AWS アカウントによって所有されている必要があります。

API Gateway のプライベート統合を使用すると、プライベートネットワーク設定やテクノロジー固有のアプライアンスの詳細な知識がなくても、VPC 内の HTTP/HTTPS リソースへのアクセスを有効にできます。

トピック

- API Gateway のプライベート統合の Network Load Balancer を設定する
- VPC リンクを作成するためのアクセス許可の付与
- API Gateway コンソールを使用して、プライベート統合を使用する API Gateway API を設定する
- AWS CLI を使用してプライベート統合で API Gateway API をセットアップする
- OpenAPI でプライベート統合を使用して API を設定する
- プライベート統合用の API Gateway アカウント

API Gateway のプライベート統合の Network Load Balancer を設定する

次の手順では、Amazon EC2 コンソールを使用して API Gateway のプライベート統合用に Network Load Balancer (NLB) を設定するステップを説明し、各ステップの詳細な手順の参照を提供します。

各 VPC にリソースがあり、1 つの NLB と 1 つ VPCLink 設定するだけで済みます。NLB は、NLB あたり複数のリスナーとターゲットグループをサポートしています。各サービスを NLB の特定のリスナーとして設定でき、単一の VPCLink を使用して NLB に接続できます。API Gateway でプライベート統合を作成するときに、各サービスに割り当てられた特定のポートを使用して、各サービスを定義します。詳細については、「the section called “チュートリアル: プライベート統合を使用して API をビルドする”」を参照してください。

> **Note**
>
> Network Load Balancer と API は、同じ AWS アカウントによって所有されている必要があります。

API Gateway コンソールを使用してプライベート統合用の Network Load Balancer を作成するには

1. AWS Management Console にサインインし、Amazon EC2 コンソール (https://console.aws.amazon.com/ec2/) を開きます。
2. Amazon EC2 インスタンスでウェブサーバーを設定します。設定例については、「Amazon Linux 2 への LAMP ウェブサーバーのインストール」を参照してください。
3. Network Load Balancer を作成し、ターゲットグループに EC2 インスタンスを登録し、ターゲットグループを Network Load Balancer のリスナーに追加します。詳細については、「Network Load Balancer の使用開始」の手順に従います。
4. Network Load Balancer を作成したら、次の操作を行います。

    a. Network Load Balancer の ARN をメモします。API を Network Load Balancer の背後にある VPC リソースと統合するために、API Gateway に VPC リンクを作成する際にそれが必要になります。

    b. PrivateLink のセキュリティグループ評価をオフにします。

    - コンソールを使用して PrivateLink トラフィックのセキュリティグループ評価をオフにするには、[セキュリティ] タブ、[編集] の順に選択します。[セキュリティ設定] で、[PrivateLink トラフィックにインバウンドルールを適用する] をオフにします。
    - AWS CLI を使用して PrivateLink トラフィックのセキュリティグループ評価をオフにするには、次のコマンドを使用します。

```
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**
>
> API Gateway CIDR は予告なしに変更されることがありますので、依存関係を追加しないでください。

VPC リンクを作成するためのアクセス許可の付与

VPC リンクを作成および維持するためのアカウント内のユーザーは、VPC エンドポイントサービス設定の作成、削除、および表示、VPC エンドポイントサービスのアクセス許可の変更、およびロードバランサーの検証の権限を持っている必要があります。このような権限を付与するには、次のステップを使用します。

VPC リンクを作成、更新、削除するためのアクセス許可を付与するには

1. 次のような IAM ポリシーを作成します。

```
{
    "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. IAM ロールを作成または選択し、前述のポリシーをロールにアタッチします。
3. 自分または VPC リンクを作成しているアカウントのユーザーに IAM ロールを割り当てます。

API Gateway コンソールを使用して、プライベート統合を使用する API Gateway API を設定する

API Gateway コンソールでプライベート統合を使用する API を設定する手順については、「チュートリアル: API Gateway のプライベート統合を使用して REST API をビルドする」を参照してください。

AWS CLI を使用してプライベート統合で API Gateway API をセットアップする

プライベート統合を使用して API を作成する前に、VPC リソースをセットアップする必要があります。また、Network Load Balancer を作成し、VPC ソースをターゲットとして設定する必要があります。要件が満たされていない場合は、「API Gateway のプライベート統合の Network Load Balancer を設定する」に従って VPC リソースをインストールし、Network Load Balancer を作成して、Network Load Balancer のターゲットとして VPC リソースを設定します。

> **Note**
>
> Network Load Balancer と API は、同じ AWS アカウントによって所有されている必要があります。

`VpcLink` を作成および管理するには、適切なアクセス権限が設定されている必要もあります。詳細については、「VPC リンクを作成するためのアクセス許可の付与」を参照してください。

> **Note**
>
> API で VpcLink を作成するアクセス権限のみが必要です。VpcLink を使用するアクセス権限は必要ありません。

Network Load Balancer が作成されたら、その ARN に注意してください。プライベート統合のために VPC リンクを作成する際にそれが必要になります。

AWS CLI を使用して、プライベート統合で API を設定するには

1. 指定された Network Load Balancer を対象とする VpcLink を作成します。

```
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
```

このコマンドの出力は、リクエストの受信を確認し、作成中の VpcLink のステータス PENDING を示します。

```
{
    "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 が VpcLink の作成を完了するまでに 2〜4 分かかります。操作が正常に終了すると、status は AVAILABLE になります。次の CLI コマンドを呼び出すことで、これを確認できます。

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

操作が失敗した場合、エラーメッセージを含む FAILED と共に statusMessage ステータスが表示されます。たとえば、すでに VPC エンドポイントに関連付けられている Network Load

Balancer を使用して VpcLink を作成しようとすると、statusMessage プロパティで次のように表示されます。

```
"NLB is already associated with another VPC Endpoint Service"
```

VpcLink が正常に作成された後は、API を作成し、VpcLink を通じて VPC リソースと統合することができます。

新しく作成された id (前述の出力の VpcLink) の *gim7c3* 値に注意してください。プライベート統合を設定する際にそれが必要になります。

2. API Gateway RestApi リソースを作成して API を設定します。

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

返された結果の RestApi の id 値に注意してください。API でさらにオペレーションを実行するには、この値が必要です。

例として、ルートリソース (GET) に / メソッドのみを持つ API を作成し、そのメソッドを VpcLink と統合します。

3. GET / メソッドを設定します。最初に、ルートリソース (/) の識別子を取得します。

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

出力では、id パスの / 値に注意してください。この例では、それが、*skpp60rab7* であることを前提としています。

GET / の API メソッドに対するメソッドリクエストを設定します。

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

VpcLink とのプロキシ統合を使用しない場合は、少なくとも 200 ステータスコードのメソッドレスポンスも設定する必要があります。ここでプロキシ統合を使用します。

4. HTTP_PROXY 型のプライベート統合を設定し、次のように put-integration コマンドを呼び出します。

```
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
```

プライベート統合の場合は、connection-type を VPC_LINK に設定し、connection-id を VpcLink の識別子、または、VpcLink ID を参照するステージ変数に設定します。uri パラメータは、エンドポイントへのルーティングリクエストには使用されませんが、Host ヘッダーの設定および証明書の検証に使用されます。

このコマンドは、以下の出力を返します。

```
{
    "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": []
}
```

ステージ変数を使用して、統合を作成するときに connectionId プロパティを設定します。

```
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}"
```

ステージ変数表現 (${stageVariables.vpcLinkId}) を二重引用符で囲み、$ 文字をエスケープします。

または、統合を更新して connectionId 値をステージ変数を使用するように設定し直すこともできます。。

```
 aws apigateway update-integration \
    --rest-api-id abcdef123 \
    --resource-id skpp60rab7 \
    --http-method GET \
    --patch-operations '[{"op":"replace","path":"/
connectionId","value":"${stageVariables.vpcLinkId}"}]'
```

文字列化された JSON リストを patch-operations パラメータ値として使用するようにしてください。

ステージ変数を使用して VpcLinks ステージ変数の値をリセットすることで、API を別の VPC または Network Load Balancer と統合できます。

プライベートプロキシ統合を使用していたため、API はデプロイメントおよびテスト実行をする準備が整っています。非プロキシ統合では、API を HTTP カスタム統合を使用して設定するのと同じように、メソッドのレスポンスと統合レスポンスも設定する必要があります。

5. API をテストするには、API をデプロイします。これは、ステージ変数を VpcLink ID のプレースホルダーとして使用した場合に必要です。ステージ変数を使用して、API をデプロイするには、次のように create-deployment コマンドを呼び出します。

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

ステージ変数を別の VpcLink ID (例: *asf9d7*) で更新するには、update-stage コマンドを呼び出します。

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

次のコマンドを使用して API を呼び出します。

```
curl -X GET https://abcdef123.execute-api.us-east-2.amazonaws.com/test
```

または、ウェブブラウザに API の invoke-URL を入力して結果を表示することもできます。

`connection-id` ID リテラルを使用して `VpcLink` プロパティをハードコードする場合は、`test-invoke-method` を呼び出して、API をデプロイする前にその呼び出しをテストすることもできます。

OpenAPI でプライベート統合を使用して API を設定する

API の OpenAPI ファイルをインポートすることで、プライベート統合を使用して API を設定することができます。これらの設定は、HTTP 統合を使用する API の OpenAPI 定義と似ていますが、次のような例外があります。

- `connectionType` を明示的に `VPC_LINK` に設定する必要があります。
- `connectionId` を `VpcLink` の ID または `VpcLink` の ID を参照するステージ変数に明示的に設定する必要があります。
- プライベート統合の `uri` パラメータは、VPC 内の HTTP/HTTPS エンドポイントを指しますが、代わりに統合リクエストの `Host` ヘッダーを設定するために使用されます。
- VPC 内の HTTPS エンドポイントとのプライベート統合における `uri` パラメータは、記載されているドメイン名を VPC エンドポイントにインストールされている証明書内のドメイン名と照合するために使用されます。

ステージ変数を使用して `VpcLink` ID を参照することができます。または、ID 値を `connectionId` に直接割り当てることもできます。

次の JSON 形式の OpenAPI ファイルは、ステージ変数 (`${stageVariables.vpcLinkId}`) によって参照された VPC リンクが設定された API の例を示しています。

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"
    }
  }
```

```
}
```

プライベート統合用の API Gateway アカウント

次のリージョン固有の API Gateway アカウント ID は、`VpcLink` の作成時に `AllowedPrincipals` として VPC エンドポイントサービスに自動的に追加されます。

リージョン	アカウント 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

リージョン	アカウント ID
ap-southeast-1	195145609632
ap-southeast-2	798376113853
ap-southeast-3	652364314486
ap-southeast-4	849137399833
ap-south-1	507069717855
ap-south-2	644042651268
ap-east-1	174803364771
sa-east-1	287228555773
me-south-1	855739686837
me-central-1	614065512851

API Gateway でモック 統合を設定する

Amazon API Gateway は、API メソッドのモック統合をサポートしています。この機能により、API デベロッパーはバックエンドを統合することなく、API Gateway から直接 API レスポンスを生成できます。API 開発者がこの機能を使用すると、プロジェクト開発の完了前に、API を操作する必要がある他の依存チームのブロックを解除できます。また、この機能を活用して、API の概要や API へのナビゲーションを提供できる API のランディングページをプロビジョニングすることができます。そのようなランディングページの例として、「チュートリアル: サンプルをインポートして REST API を作成する」で説明されている API 例のルートリソースで、GET メソッドの統合リクエストとレスポンスを参照してください。

API デベロッパーは、API Gateway がモック統合リクエストにどのように応答するかを決定します。そのため、特定のステータスコードとレスポンスを関連付ける、メソッドの統合リクエストと統合レスポンスを設定します。モック統合で `200` レスポンスを返すメソッドの場合は、以下を返すように統合リクエストボディマッピングテンプレートを設定します。

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

次のようなボディマッピングテンプレートを持つように 200 統合レスポンスを設定します。

```
{
    "statusCode": 200,
    "message": "Go ahead without me."
}
```

同様に、メソッドがたとえば 500 エラーレスポンスを返す場合は、統合リクエストボディマッピングテンプレートを設定して以下を返します。

```
{"statusCode": 500}
```

たとえば、次のマッピングテンプレートを使用して 500 統合レスポンスを設定します。

```
{
    "statusCode": 500,
    "message": "The invoked method is not supported on the API resource."
}
```

あるいは、統合リクエストマッピングテンプレートを定義せずに、モック統合のメソッドがデフォルトの統合レスポンスを返すこともできます。デフォルトの統合レスポンスは、未定義の [HTTP status regex (HTTP ステータスの正規表現)] を持つレスポンスです。適切なパススルー動作が設定されていることを確認します。

> **Note**
>
> モック統合は、大規模なレスポンステンプレートをサポートするためのものではありません。お客様のユースケースにモック統合が必要な場合は、代わりに Lambda 統合を使用することを検討してください。

統合リクエストマッピングテンプレートを使用して、アプリケーションロジックを挿入して、特定の条件に基づいて返すモック統合レスポンスを決定することができます。たとえば、受信リクエストに対して scope クエリパラメータを使用して、成功レスポンスまたはエラーレスポンスを返すかどうかを判断できます。

```
{
  #if( $input.params('scope') == "internal" )
```

```
    "statusCode": 200
  #else
    "statusCode": 500
  #end
}
```

このように、モック統合の方法では、エラーレスポンスを伴う他のタイプの呼び出しを拒否しながら、内部呼び出しを通過させることができます。

このセクションでは、API Gateway コンソールを使用して、API メソッドのモック統合を有効にする方法を説明します。

トピック

- API Gateway コンソールを使用したモック統合の有効化

API Gateway コンソールを使用したモック統合の有効化

メソッドが API Gateway で使用可能であることが必要です。「チュートリアル: HTTP 非プロキシ統合を使用して REST API をビルドする」の手順に従います

1. API リソースを選択し、[メソッドを作成] を選択します。

   メソッドを作成するには、次の操作を行います。

   a. [メソッドタイプ] で、メソッドを選択します。
   b. [統合タイプ] で、[Mock] を選択します。
   c. [メソッドの作成] を選択します。
   d. [メソッドリクエスト] タブの [メソッドリクエストの設定] で、[編集] を選択します。
   e. [URL クエリ文字列パラメータ] を選択します。[クエリ文字列を追加] を選択し、[名前] に「**scope**」と入力します。このクエリパラメータは、呼び出し元が内部かどうかを決定します。
   f. [Save] を選択します。
2. [メソッドレスポンス] タブで [レスポンスを作成] を選択し、次の操作を行います。

   a. [HTTP ステータス] に「**500**」と入力します。
   b. [Save] を選択します。
3. [統合リクエスト] タブの [統合リクエストの設定] で、[編集] を選択します。

4. [マッピングテンプレート] を選択し、次の操作を行います。

   a. [マッピングテンプレートの追加] を選択します。

   b. [コンテンツタイプ] に、「**application/json**」と入力します。

   c. [テンプレート本文] で次のように入力します。

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

   d. [Save] を選択します。

5. [統合レスポンス] タブの [デフォルト - レスポンス] で、[編集] を選択します。

6. [マッピングテンプレート] を選択し、次の操作を行います。

   a. [コンテンツタイプ] に、「**application/json**」と入力します。

   b. [テンプレート本文] で次のように入力します。

```
{
    "statusCode": 200,
    "message": "Go ahead without me"
}
```

   c. [Save] を選択します。

7. [レスポンスの作成] を選択します。

   500 レスポンスを作成するには、次の操作を行います。

   a. [HTTP status regex (HTTP ステータスの正規表現)]に「**5\d{2}**」と入力します。

   b. [メソッドレスポンスのステータス] で、[**500**] を選択します。

   c. [Save] を選択します。

   d. [5\d{2} - レスポンス] で、[編集] を選択します。

   e. [マッピングテンプレート]、[マッピングテンプレートの追加] の順に選択します。

   f. [コンテンツタイプ] に、「**application/json**」と入力します。

   g. [テンプレート本文] で次のように入力します。

```
{
    "statusCode": 500,
    "message": "The invoked method is not supported on the API resource."
}
```

h. [Save] を選択します。

8. [テスト] タブを選択します。タブを表示するには、右矢印ボタンを選択する必要がある場合があります。Mock 統合をテストするには、次の操作を行います。

a. [クエリ文字列] に「scope=internal」と入力します。[Test (テスト)] を選択します。テストの結果が表示されます。

```
Request: /?scope=internal
Status: 200
Latency: 26 ms
Response Body

{
  "statusCode": 200,
  "message": "Go ahead without me"
}

Response Headers

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

b. Query strings に「scope=public」と入力するか、空白のままにします。[Test (テスト)] を選択します。テストの結果が表示されます。

```
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"}
```

また、メソッドレスポンスにヘッダーを追加してから、統合レスポンスでヘッダーマッピングを設定することによって、モック統合レスポンスでヘッダーを返すこともできます。実際に、これは API Gateway コンソールで CORS の必要なヘッダーを返すことによって CORS サポートを有効にする方法です。

API Gateway でリクエストの検証を使用する

統合リクエストを進める前に API リクエストの基本的な検証を実行するよう API Gateway を設定できます。検証に失敗した場合、API Gateway はすぐにリクエストに失敗して、400 個のエラーレスポンスを発信者に返し、CloudWatch Logs で検証結果を発行します。これによりバックエンドへの不必要な呼び出すが減少します。さらに重要な点として、アプリケーション固有の検証作業に集中することができます。リクエスト本文を検証するには、必要なリクエストパラメータが有効で null でないことを確認するか、より複雑なデータ検証用のモデルスキーマを指定します。

トピック

- API Gateway における基本的なリクエストの検証の概要
- データモデルを理解する
- API Gateway で基本的なリクエストの検証を設定する
- 基本的なリクエストの検証を使用したサンプル API の OpenAPI 定義
- 基本的なリクエスト検証を含むサンプル API の AWS CloudFormation テンプレート

API Gateway における基本的なリクエストの検証の概要

基本的なリクエストの検証は API Gateway で処理できるため、ユーザーはバックエンドでのアプリケーション固有の検証に集中できます。検証のために、API Gateway は以下の条件のどちらかまたは両方を確認します。

- 受信リクエストの URI、クエリ文字列、ヘッダーに必要なリクエストパラメータが含まれており、空白ではない。
- 該当するリクエストペイロードが、メソッドの設定済みの JSON スキーマリクエストに準拠している。

検証を有効にするには、リクエストの検証により検証ルールを指定し、この検証を API のリクエストの検証のマップに追加し、検証を個々の API メソッドに割り当てます。

> **Note**
>
> リクエスト本文の検証と統合パススルーの動作は、2 つの異なるトピックです。リクエストペイロードと一致するモデルスキーマがない場合、元のペイロードをパススルーするかブロックするかを選択できます。詳細については、「統合パススルーの動作」を参照してください。

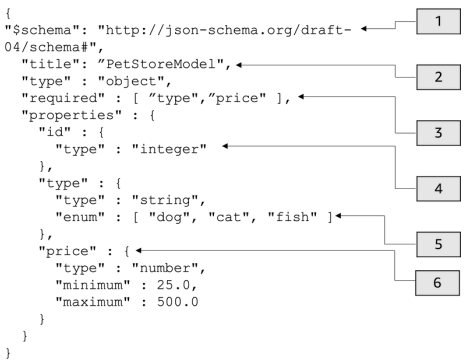
データモデルを理解する

API Gateway では、モデルはペイロードのデータ構造を定義します。API Gateway では、JSON スキーマのドラフト 4 を使用してモデルを定義します。次の JSON オブジェクトは、PetStore の例のサンプルデータです。

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

データには、ペットの `id`、`type`、`price` が含まれています。このデータのモデルにより、以下のことが可能になります。

- 基本的なリクエストの検証を使用する。
- データ変換用のマッピングテンプレートを作成する。
- SDK を生成するときに、ユーザー定義データ型 (UDT) を作成する。



このモデルの内容は以下のとおりです。

1. $schema オブジェクトは、有効な JSON スキーマのバージョン識別子を表します。このスキーマは JSON スキーマのドラフト v4 です。
2. title オブジェクトは、人間が読めるモデルの識別子です。このタイトルは PetStoreModel です。
3. required 検証キーワードには、基本的なリクエストの検証用の type と price が必要です。
4. モデルの properties は、id、type、price です。各オブジェクトには、モデルに記述されているプロパティがあります。
5. オブジェクト type は、値として dog、cat、fish のいずれかのみを持つことができます。
6. オブジェクト price は数値で、minimum が 25、maximum が 500 に制限されます。

PetStore モデル

```
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 }
```

このモデルの内容は以下のとおりです。

1. 2 行目で、$schema オブジェクトは有効な JSON スキーマのバージョン識別子を表します。このスキーマは JSON スキーマのドラフト v4 です。

2. 3 行目で、title オブジェクトは、人間が読めるモデルの識別子です。このタイトルは PetStoreModel です。
3. 5 行目で、required 検証キーワードには、基本的なリクエストの検証用の type と price が必要です。
4. 6 ~ 17 行目で、モデルの properties は id、type、price です。各オブジェクトには、モデルに記述されているプロパティがあります。
5. 12 行目で、オブジェクト type は値として dog、cat、fish のいずれかのみを持つことができます。
6. 14 ~ 17 行目で、オブジェクト price は数値で、minimum が 25、maximum が 500 に制限されます。

より複雑なモデルの作成

$ref プリミティブを使用すると、より長いモデルの再利用可能な定義を作成できます。例えば、price オブジェクトを記述する definitions セクションで、Price という定義を作成できます。$ref の値は Price 定義です。

```
{
  "$schema" : "http://json-schema.org/draft-04/schema#",
  "title" : "PetStoreModelReUsableRef",
  "required" : ["price", "type" ],
  "type" : "object",
  "properties" : {
    "id" : {
      "type" : "integer"
    },
    "type" : {
      "type" : "string",
      "enum" : [ "dog", "cat", "fish" ]
    },
    "price" : {
        "$ref": "#/definitions/Price"
    }
  },
  "definitions" : {
      "Price": {
        "type" : "number",
        "minimum" : 25.0,
        "maximum" : 500.0
            }
```

```
      }
}
```

外部モデルファイルで定義された別のモデルスキーマを参照することもできます。$ref プロパティの値をモデルの場所に設定します。次の例では、Price モデルは API a1234 の PetStorePrice モデルに定義されています。

```
{
  "$schema" : "http://json-schema.org/draft-04/schema#",
  "title" : "PetStorePrice",
  "type": "number",
  "minimum": 25,
  "maximum": 500
}
```

より長いモデルは PetStorePrice モデルを参照できます。

```
{
  "$schema" : "http://json-schema.org/draft-04/schema#",
  "title" : "PetStoreModelReusableRefAPI",
  "required" : [ "price", "type" ],
  "type" : "object",
  "properties" : {
    "id" : {
      "type" : "integer"
    },
    "type" : {
      "type" : "string",
      "enum" : [ "dog", "cat", "fish" ]
    },
    "price" : {
        "$ref": "https://apigateway.amazonaws.com/restapis/a1234/models/PetStorePrice"
    }
  }
}
```

出力データモデルを使用する

データを変換する場合、統合レスポンスでペイロードモデルを定義できます。ペイロードモデルは SDK を生成するときに使用できます。Java、Objective-C、Swift などの厳密に型指定された言語では、オブジェクトはユーザー定義データ型 (UDT) に対応します。SDK の生成時にデータモデルで

UDT を指定すると、API Gateway は UDT を作成します。データ変換の詳細については、「マッピングテンプレートについて」を参照してください。

出力データ

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

出力モデル

```
{
"$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
    }
  }
}
```

このモデルでは、SDK を呼び出し、`PetStoreOutputModel[i].description` プロパティと `PetStoreOutputModel[i].askingPrice` プロパティを読み取ることで、`description` と `askingPrice` のプロパティ値を取得できます。モデルが指定されていない場合、API Gateway は空のモデルを使用してデフォルト UDT を作成します。

次のステップ

- このセクションでは、このトピックで紹介した概念について理解を深めるために役立つリソースを紹介します。

  以下のリクエストの検証チュートリアルを参考にしてください。

  - API Gateway コンソールを使用してリクエストの検証を設定する
  - AWS CLI を使用して基本的なリクエストの検証を設定する
  - OpenAPI 定義を使用して基本的なリクエストの検証を設定する
- データ変換とマッピングテンプレートの詳細については、「マッピングテンプレートについて」を参照してください。
- また、より複雑なデータモデルを表示することもできます。「API Gateway のデータモデルとマッピングテンプレートの例」を参照してください。

API Gateway で基本的なリクエストの検証を設定する

このセクションでは、コンソール、AWS CLI、OpenAPI 定義を使用して API Gateway のリクエストの検証を設定する方法を示します。

トピック

- API Gateway コンソールを使用してリクエストの検証を設定する
- AWS CLI を使用して基本的なリクエストの検証を設定する
- OpenAPI 定義を使用して基本的なリクエストの検証を設定する

API Gateway コンソールを使用してリクエストの検証を設定する

API Gateway コンソールで API リクエストの 3 つの検証から 1 つを選択して、リクエストを検証できます。

- 本体の検証。
- クエリ文字列パラメータとヘッダーの検証。

- 本文、クエリ文字列パラメータ、ヘッダーの検証。

上記のいずれかの検証を API メソッドに適用すると、API Gateway コンソールは検証を API の RequestValidators マップに追加します。

このチュートリアルに従うには、AWS CloudFormation テンプレートを使用して不完全な API Gateway API を作成します。この API には、GET メソッドと POST メソッドを持つ /validator リソースがあります。どちらのメソッドも http://petstore-demo-endpoint.execute-api.com/petstore/pets HTTP エンドポイントと統合されています。次の 2 種類のリクエストの検証を設定します。

- GET メソッドでは、URL クエリ文字列パラメータに対するリクエストの検証を設定します。
- POST メソッドでは、リクエスト本文に対するリクエストの検証を設定します。

これにより、特定の API コールのみを API にパススルーできます。

AWS CloudFormation のアプリケーション作成テンプレートをダウンロードして解凍します。このテンプレートを使用して不完全な API を作成します。残りのステップは API Gateway コンソールで完了します。

AWS CloudFormation スタックを作成するには

1. https://console.aws.amazon.com/cloudformation で AWS CloudFormation コンソール を開きます。
2. [スタックの作成] を選択し、[With new resources (standard) 新しいリソースを使用 (標準)] を選択します。
3. [Specify template (テンプレートの指定)] で、[Upload a template file (テンプレートファイルのアップロード)] を選択します。
4. ダウンロードしたテンプレートを選択します。
5. [Next (次へ)] を選択します。
6. [Stack name] (スタックの名前) で、**request-validation-tutorial-console** と入力し、[Next] (次へ) を選択します。
7. [Configure stack options] (スタックオプションの設定) で、[Next] (次へ) を選択します。
8. [Capabilities] (機能) で、AWS CloudFormation がアカウントに IAM リソースを作成できることを承認します。

9. 送信 を選択します。

AWS CloudFormation は、テンプレートで指定されたリソースをプロビジョニングします。リソースのプロビジョニングには数分かかることがあります。AWS CloudFormation スタックのステータスが CREATE_COMPLETE の場合は、次のステップに進む準備ができています。

新しく作成した API を選択するには

1. 新しく作成した **request-validation-tutorial-console** スタックを選択します。
2. [リソース] をクリックします。
3. [物理 ID] で API を選択します。このリンクを使用して API Gateway コンソールに移動します。

GET メソッドと POST メソッドを変更する前に、モデルを作成する必要があります。

モデルを作成するには

1. 受信リクエストの本文でリクエストの検証を使用するには、モデルが必要です。モデルを作成するには、メインナビゲーションペインで [モデル] を選択します。
2. [モデルの作成] を選択します。
3. [名前] に**PetStoreModel**と入力します。
4. [コンテンツタイプ] として、「**application/json**」と入力します。一致するコンテンツタイプが見つからない場合、リクエストの検証は実行されません。コンテンツタイプに関係なく同じモデルを使用するには、「**$default**」と入力します。
5. [説明] に、モデルの説明として「**My PetStore Model**」と入力します。
6. [モデルスキーマ] で、次のモデルをコードエディタに貼り付け、[作成] を選択します。

```
{
  "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
    }
  }
}
```

これらのモデルの詳細については、「データモデルを理解する」を参照してください。

GET メソッドのリクエスト検証を設定するには

1. メインナビゲーションペインで [リソース]、[GET] メソッドの順に選択します。
2. [メソッドリクエスト] タブの [メソッドリクエスト設定] で、[編集] を選択します。
3. [リクエストの検証] で、[クエリ文字列パラメータおよびヘッダーを検証] を選択します。
4. [URL クエリ文字列パラメータ] で、次の操作を行います。
   a. [クエリ文字列の追加] を選択します。
   b. [名前] に**petType**と入力します。
   c. [必須] をオンにします。
   d. [キャッシュ] はオフのままにします。
5. [Save] を選択します。
6. [統合リクエスト] タブの [統合リクエストの設定] で、[編集] を選択します。
7. [URL クエリ文字列パラメータ] で、次の操作を行います。
   a. [クエリ文字列の追加] を選択します。
   b. [名前] に**petType**と入力します。
   c. [マッピング元] として「**method.request.querystring.petType**」と入力します。これは、**petType** をペットの種類にマッピングします。

      データマッピングの詳細については、データマッピングチュートリアルを参照してください。
   d. [キャッシュ] はオフのままにします。
8. [Save] を選択します。

GET メソッドのリクエスト検証をテストするには

1. [テスト] タブを選択します。タブを表示するには、右矢印ボタンを選択する必要がある場合があります。
2. [クエリ文字列] に「**petType=dog**」と入力し、[テスト] を選択します。
3. メソッドテストが `200 OK` と犬のリストを返します。

   この出力データを変換する方法については、データマッピングチュートリアルを参照してください。
4. **petType=dog** を削除して [テスト] を選択します。
5. メソッドテストは、`400` エラーを返し、次のエラーメッセージを表示します。

```
{
  "message": "Missing required request parameters: [petType]"
}
```

POST メソッドのリクエスト検証を設定するには

1. メインナビゲーションペインで、[ステージ]、[POST] メソッドの順に選択します。
2. [メソッドリクエスト] タブの [メソッドリクエスト設定] で、[編集] を選択します。
3. [リクエストの検証] で [本文を検証] を選択します。
4. [リクエスト本文] で、[モデルを追加] を選択します。
5. [コンテンツタイプ] に「**application/json**」と入力し、[モデル] で [PetStoreModel] を選択します。
6. [Save] を選択します。

POST メソッドのリクエスト検証をテストするには

1. [テスト] タブを選択します。タブを表示するには、右矢印ボタンを選択する必要がある場合があります。
2. [リクエスト本文] で、次の内容をコードエディタに貼り付けます。

```
{
  "id": 2,
  "name": "Bella",
  "type": "dog",
```

```
  "price": 400
}
```

[テスト] を選択します。

3. メソッドテストは、200 OK と成功メッセージを返します。
4. [リクエスト本文] で、次の内容をコードエディタに貼り付けます。

```
{
  "id": 2,
  "name": "Bella",
  "type": "dog",
  "price": 4000
}
```

[テスト] を選択します。

5. メソッドテストは、400 エラーを返し、次のエラーメッセージを表示します。

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

テストログの一番下に、無効なリクエスト本文の理由が表示されます。この場合、ペットの価格はモデルに指定されている上限を超えていました。

AWS CloudFormation スタックを削除するには

1. AWS CloudFormation コンソール (https://console.aws.amazon.com/cloudformation) を開きます。
2. AWS CloudFormation スタックを選択します。
3. [Delete] (削除) を選択し、選択を確定します。

次のステップ

- 出力データを変換して、より多くのデータマッピングを実行する方法については、データマッピングチュートリアルを参照してください。
- AWS CLI を使用して基本的なリクエストの検証を設定するチュートリアルに従い、AWS CLI を使用して同様の手順を実行します。

AWS CLI を使用して基本的なリクエストの検証を設定する

AWS CLI を使用してリクエストの検証を設定するための検証を作成できます。このチュートリアルに従うには、AWS CloudFormation テンプレートを使用して不完全な API Gateway API を作成します。

> **Note**
>
> これはコンソールチュートリアルと同じ AWS CloudFormation テンプレートではありません。

事前に公開されている `/validator` リソースを使用して `GET` メソッドと `POST` メソッドを作成します。どちらのメソッドも `http://petstore-demo-endpoint.execute-api.com/petstore/pets` HTTP エンドポイントと統合されます。次の 2 つのリクエストの検証を設定します。

- `GET` メソッドでは、`params-only` 検証を作成して URL クエリ文字列パラメータを検証します。
- `POST` メソッドでは、`body-only` 検証を作成してリクエスト本文を検証します。

これにより、特定の API コールのみを API にパススルーできます。

AWS CloudFormation スタックを作成するには

AWS CloudFormation のアプリケーション作成テンプレートをダウンロードして解凍します。

次のチュートリアルを完了するには、AWS Command Line Interface (AWS CLI) バージョン 2 が必要です。

コマンドが長い場合は、エスケープ文字 (\) を使用してコマンドを複数行に分割します。

> **Note**
>
> Windows では、一般的に使用する Bash CLI コマンドの一部 (`zip` など) が、オペレーティングシステムの組み込みターミナルでサポートされていません。Ubuntu および Bash の Windows 統合バージョンを取得するには、Windows Subsystem for Linux をインストールします。このガイドの CLI コマンドの例では、Linux フォーマットを使用しています。Windows CLI を使用している場合、インライン JSON ドキュメントを含むコマンドを再フォーマットする必要があります。

1. 次のコマンドを使用して AWS CloudFormation スタックを作成します。

```
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 は、テンプレートで指定されたリソースをプロビジョニングします。リソースのプロビジョニングには数分かかることがあります。次のコマンドを使用して AWS CloudFormation スタックのステータスを確認します。

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

3. AWS CloudFormation スタックのステータスが `StackStatus: "CREATE_COMPLETE"` になったら、次のコマンドを使用して関連する出力値を取得し、以後のステップで使用します。

```
 aws cloudformation describe-stacks --stack-name request-validation-tutorial-cli
 --query "Stacks[*].Outputs[*].{OutputKey: OutputKey, OutputValue: OutputValue,
 Description: Description}"
```

出力値は以下のとおりです。

- ApiId。API の ID です。このチュートリアルの場合、API ID は `abc123` です。
- ResourceId。`GET` メソッドと `POST` メソッドが公開されている検証リソースの ID です。このチュートリアルの場合、リソース ID は `efg456` です。

リクエストの検証を作成してモデルをインポートするには

1. AWS CLI でリクエストの検証を行うには、検証が必要です。リクエストパラメータのみを検証する検証を作成するには、次のコマンドを使用します。

```
aws apigateway create-request-validator --rest-api-id abc123 \
      --no-validate-request-body \
      --validate-request-parameters \
      --name params-only
```

`params-only` 検証の ID をメモしておきます。

2. リクエスト本文のみを検証する検証を作成するには、次のコマンドを使用します。

```
aws apigateway create-request-validator --rest-api-id abc123 \
```

```
      --validate-request-body \
      --no-validate-request-parameters \
      --name body-only
```

body-only 検証の ID をメモしておきます。

3. 受信リクエストの本文でリクエストの検証を行うには、モデルが必要です。モデルをインポートするには、以下のコマンドを使用します。

```
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}}}}'
```

一致するコンテンツタイプが見つからない場合、リクエストの検証は実行されません。コンテンツタイプに関係なく同じモデルを使用するには、キーとして $default を指定します。

GET メソッドと POST メソッドを作成するには

1. 次のコマンドを使用して、GET HTTP メソッドを /validate リソースに追加します。このコマンドは、GET メソッドを作成して、params-only 検証を追加し、必要に応じてクエリ文字列 petType を設定します。

```
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"
```

次のコマンドを使用し、POST HTTP メソッドを /validate リソースに追加します。このコマンドは、POST メソッドを作成して、body-only 検証を追加し、モデルを本文専用の検証にアタッチします。

```
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. 次のコマンドを使用して、GET /validate メソッドの 200 OK レスポンスを設定します。

```
aws apigateway put-method-response --rest-api-id abc123  \
            --resource-id efg456 \
            --http-method GET \
            --status-code 200
```

次のコマンドを使用して、POST /validate メソッドの 200 OK レスポンスを設定します。

```
aws apigateway put-method-response --rest-api-id abc123  \
            --resource-id efg456 \
            --http-method POST \
            --status-code 200
```

3. 次のコマンドを使用して、GET /validation メソッドの指定された HTTP エンドポイントを Integration に設定します。

```
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'
```

次のコマンドを使用して、POST /validation メソッドの指定された HTTP エンドポイントを Integration に設定します。

```
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. 次のコマンドを使用して、GET /validation メソッドの統合レスポンスを設定します。

```
aws apigateway put-integration-response --rest-api-id abc123 \
              --resource-id efg456\
              --http-method GET \
              --status-code 200 \
              --selection-pattern ""
```

次のコマンドを使用して、POST /validation メソッドの統合レスポンスを設定します。

```
aws apigateway put-integration-response --rest-api-id abc123 \
            --resource-id efg456 \
            --http-method POST \
            --status-code 200 \
            --selection-pattern ""
```

API をテストするには

1. クエリ文字列に対するリクエストの検証を実行する GET メソッドをテストするには、次のコマンドを使用します。

```
aws apigateway test-invoke-method --rest-api-id abc123 \
            --resource-id efg456 \
            --http-method GET \
            --path-with-query-string '/validate?petType=dog'
```

結果として、200 OK と犬のリストが返されます。

2. 次のコマンドを使用して、クエリ文字列 petType を含めずにテストします。

```
aws apigateway test-invoke-method --rest-api-id abc123 \
            --resource-id efg456 \
            --http-method GET
```

結果として 400 エラーが返されます。

3. リクエスト本文に対するリクエストの検証を実行する POST メソッドをテストするには、次のコマンドを使用します。

```
 aws apigateway test-invoke-method --rest-api-id abc123 \
            --resource-id efg456 \
```

```
            --http-method POST \
            --body '{"id": 1, "name": "bella", "type": "dog", "price" : 400 }'
```

結果として、200 OK と成功メッセージが返されます。

4. 次のコマンドを実行し、無効な本文を使用してテストします。

```
aws apigateway test-invoke-method --rest-api-id abc123 \
            --resource-id efg456 \
            --http-method POST \
            --body '{"id": 1, "name": "bella", "type": "dog", "price" : 1000 }'
```

犬の価格がモデルで定義されている最大価格を超えているため、結果として 400 エラーが返されます。

AWS CloudFormation スタックを削除するには

- AWS CloudFormation リソースを削除するには、次のコマンドを使用します。

```
aws cloudformation delete-stack  --stack-name request-validation-tutorial-cli
```

OpenAPI 定義を使用して基本的なリクエストの検証を設定する

API レベルでリクエストの検証を宣言するには、x-amazon-apigateway-request-validators オブジェクト マップ内の x-amazon-apigateway-request-validators.requestValidator オブジェクト オブジェクトのセットを指定して、リクエストのどの部分を検証するかを選択します。OpenAPI 定義の例では、次の 2 つの検証があります。

- all 検証では、本文 (RequestBodyModel データモデルを使用) とパラメータの両方を検証します。
- param-only 検証では、パラメータのみを検証します。

API のすべてのメソッドでリクエストの検証を有効にするには、OpenAPI 定義の API レベルで x-amazon-apigateway-request-validator プロパティ プロパティを指定します。OpenAPI 定義の例の場合、all 検証は、オーバーライドされない限り、すべての API メソッドで使用されます。モデルを使用して本文を検証する際、一致するコンテンツタイプが見つからない場合、リクエストの検証は実行されません。コンテンツタイプに関係なく同じモデルを使用するには、キーとして $default を指定します。

個々のメソッドでリクエストの検証を有効にするには、メソッドレベルで `x-amazon-apigateway-request-validator` プロパティを指定します。OpenAPI 定義の例の場合、`param-only` 検証は `GET` メソッドの `all` 検証を上書きします。

OpenAPI のサンプルを API Gateway にインポートするには、リージョン API を API Gateway にインポートする または エッジ最適化 API を API Gateway にインポートする に対する次の手順を参照してください。

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
    }
  }
}
```

基本的なリクエストの検証を使用したサンプル API の OpenAPI 定義

以下の OpenAPI 定義は、リクエストの検証が有効なサンプル API を定義します。API は、PetStore API のサブセットです。POST メソッドを開示し、ペットを pets コレクションおよび GET メソッドに追加して、指定されたタイプによりペットのクエリを実行します。

x-amazon-apigateway-request-validators マップでは、2 つのリクエストの検証が API レベルで宣言されています。params-only の検証は API で有効になり、GET メソッドにより継承されます。この検証により、必須のクエリパラメータ (q1) が受信リクエストに含まれており、空白でないことを API Gateway が確認できるようになります。all の検証は、POST メソッドで有効になります。この検証は、必須のヘッダーパラメーター (h1) が設定されており、空白でないことを検証します。また、ペイロード形式が、指定された RequestBodyModel 形式に準拠していることも検証します。一致するコンテンツタイプが見つからない場合、リクエストの検証は実行されません。モデルを使用して本文を検証する際、一致するコンテンツタイプが見つからない場合、リクエストの検証は実行されません。コンテンツタイプに関係なく同じモデルを使用するには、キーとして $default を指定します。

このモデルでは、入力 JSON オブジェクトに name、type、price の各プロパティが含まれている必要があります。name プロパティは任意の文字列にすることができ、type は指定された列挙フィールド (["dog", "cat", "fish"]) のいずれかでなければなりません。また、price は 25 から 500 の範囲にする必要があります。id パラメータは必須ではありません。

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": {

      }
    }
  }
}
```

基本的なリクエスト検証を含むサンプル API の AWS CloudFormation テンプレート

以下の AWS CloudFormation テンプレート定義の例は、リクエストの検証が有効なサンプル API を定義します。API は、PetStore API のサブセットです。POST メソッドを開示し、ペットを pets コレクションおよび GET メソッドに追加して、指定されたタイプによりペットのクエリを実行します。

宣言されているリクエスト検証は 2 つあります。

GETValidator

この検証は、GET メソッドで有効になります。この検証により、必須のクエリパラメータ (q1) が受信リクエストに含まれており、空白でないことを API Gateway が確認できるようになります。

POSTValidator

この検証は、POST メソッドで有効になります。これにより、API Gateway は、コンテンツタイプが application/json のとき、ペイロードリクエスト形式が指定され

た RequestBodyModel に準拠していることを検証できます。一致するコンテンツタイプが見つからない場合、リクエストの検証は実行されません。コンテンツタイプに関係なく同じモデルを使用するには、$default を指定します。RequestBodyModel にはペット ID を定義する追加のモデル RequestBodyModelId が含まれています。

```
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}'
```

REST API のデータ変換の設定

API Gateway の場合、API のメソッドリクエストは、統合リクエストペイロードとは異なる形式のペイロードを受け取ることができます。同様に、バックエンドは、メソッドレスポンスペイロードとは異なる統合レスポンスペイロードを返す場合があります。マッピングテンプレートを使用して、URL

パスパラメータ、URL クエリ文字列パラメータ、HTTP ヘッダー、およびリクエスト本文を API Gateway 全体にマッピングできます。

マッピングテンプレートは、Velocity Template Language (VTL) で表現されるスクリプトであり、JSONPath 式を使用してペイロードに適用されます。

ペイロードでは、JSON スキーマのドラフト 4 に基づくデータモデルを使用できます。モデルの詳細については、「データモデルを理解する」を参照してください。

> **Note**
>
> マッピングテンプレートを作成するためにモデルを定義する必要はありませんが、API Gateway で SDK を生成したり、API のリクエスト本文の検証を有効にしたりするには、モデルを定義する必要があります。

トピック

- マッピングテンプレートについて
- API Gateway でのデータ変換の設定
- マッピングテンプレートを使用して、API のリクエストおよびレスポンスパラメータとステータスコードをオーバーライドする
- API Gateway コンソールを使用してリクエストとレスポンスのデータマッピングを設定する
- API Gateway のデータモデルとマッピングテンプレートの例
- Amazon API Gateway API リクエストおよびレスポンスデータマッピングリファレンス
- API Gateway マッピングテンプレートとアクセスのログ記録の変数リファレンス

マッピングテンプレートについて

API Gateway では、API のメソッドリクエストまたはレスポンスは、統合リクエストまたはレスポンスとは異なる形式のペイロードを受け取ることができます。

データを次のように変換できます。

- ペイロードを API 指定の形式に一致させます。
- API のリクエストおよびレスポンスパラメータとステータスコードを上書きします。

- クライアントが選択したレスポンスヘッダーを返します。
- HTTP プロキシまたは AWS のサービスプロキシのメソッドリクエストにパスパラメータ、クエリ文字列パラメータ、またはヘッダーパラメータを関連付けます。
- Amazon DynamoDB や Lambda 関数などの AWS のサービス、または HTTP エンドポイントとの統合を使用して送信するデータを選択します。

データを変換するには、マッピングテンプレートを使用できます。マッピングテンプレートは、Velocity Template Language (VTL) で表現されるスクリプトであり、JSONPath を使用してペイロードに適用されます。

次の例は、PetStore のデータを変換するための入力データ、マッピングテンプレート、および出力データを示しています。

入力データ

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

マッピングテンプレート

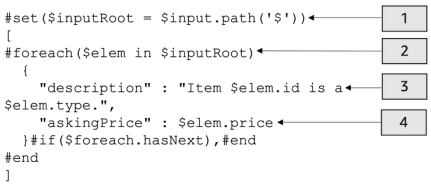
```
#set($inputRoot = $input.path('$'))
[
#foreach($elem in $inputRoot)
  {
    "description" : "Item $elem.id is a $elem.type.",
    "askingPrice" : $elem.price
  }#if($foreach.hasNext),#end
```

```
#end
]
```

出力データ

```
[
  {
    "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
  }
]
```

次の図は、このマッピングテンプレートの詳細を示しています。



1. $inputRoot 変数は、前のセクションにおける元の JSON データのルートオブジェクトです。ディレクティブは # 記号で始まります。
2. foreach ループは、元の JSON データ内の各オブジェクトを反復処理します。
3. description は、元の JSON データからのペットの id と type を連結したものです。
4. askingPrice は元の JSON データからの価格を示す price です。

PetStore マッピングテンプレート

```
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 ]
```

このマッピングテンプレートの内容は以下のとおりです。

1. 1 行目で、`$inputRoot` 変数は前のセクションからの元の JSON データのルートオブジェクトを表します。ディレクティブは # 記号で始まります。
2. 3 行目で、`foreach` ループは元の JSON データ内の各オブジェクトを反復処理します。
3. 5 行目で、`description` は、元の JSON データからのペットの `id` と `type` を連結したものです。
4. 6 行目で、`askingPrice` は元の JSON データからの価格を示す `price` です。

Velocity Template Language の詳細については、「Apache Velocity - VTL Reference」を参照してください。JSONPath の詳細については、「JSONPath - XPath for JSON」を参照してください。

マッピングテンプレートでは、基盤となるデータが JSON オブジェクトであることを前提としています。データのモデルを定義する必要はありません。ただし、出力データのモデルを使用すると、先行データを言語固有のオブジェクトとして返すことができます。詳細については、「データモデルを理解する」を参照してください。

複雑なマッピングテンプレート

また、より複雑なマッピングテンプレートを作成することもできます。次の例は、参照の連結と 100 のカットオフを使用して、ペットが手頃な価格かどうかを判断する方法を示しています。

入力データ

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

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

マッピングテンプレート

```
#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
]
```

出力データ

```
[
  {
    "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
  }
]
```

また、より複雑なデータモデルを表示することもできます。「API Gateway のデータモデルとマッピングテンプレートの例」を参照してください。

API Gateway でのデータ変換の設定

このセクションでは、コンソールと AWS CLI を使用して統合リクエストおよびレスポンスを変換するマッピングテンプレートを設定する方法を示します。

トピック

- API Gateway コンソールを使用してデータ変換を設定する
- AWS CLI を使用してデータ変換を設定する
- 完成したデータ変換の AWS CloudFormation テンプレート
- 次のステップ

API Gateway コンソールを使用してデータ変換を設定する

このチュートリアルでは、.zip ファイル (data-transformation-tutorial-console.zip) を使用して不完全な API と DynamoDB テーブルを作成します。この不完全な API には、`GET` メソッドと `POST` メソッドを持つ `/pets` リソースがあります。

- `GET` メソッドは `http://petstore-demo-endpoint.execute-api.com/petstore/pets` HTTP エンドポイントからデータを取得します。出力データは、PetStore マッピングテンプレート のマッピングテンプレートに従って変換されます。
- `POST` メソッドを使用すると、ユーザーはマッピングテンプレートを使用して Amazon DynamoDB テーブルにペット情報を `POST` できます。

AWS CloudFormation のアプリケーション作成テンプレートをダウンロードして解凍します。このテンプレートを使用して、ペット情報と不完全な API を投稿するための DynamoDB テーブルを作成します。残りのステップは API Gateway コンソールで完了します。

AWS CloudFormation スタックを作成するには

1. https://console.aws.amazon.com/cloudformation で AWS CloudFormation コンソール を開きます。
2. [スタックの作成] を選択し、[With new resources (standard) 新しいリソースを使用 (標準)] を選択します。

3. [Specify template (テンプレートの指定)] で、[Upload a template file (テンプレートファイルのアップロード)] を選択します。
4. ダウンロードしたテンプレートを選択します。
5. [Next (次へ)] を選択します。
6. [Stack name] (スタックの名前) で、**data-transformation-tutorial-console** と入力し、[Next] (次へ) を選択します。
7. [Configure stack options] (スタックオプションの設定) で、[Next] (次へ) を選択します。
8. [Capabilities] (機能) で、AWS CloudFormation がアカウントに IAM リソースを作成できることを承認します。
9. 送信 を選択します。

AWS CloudFormation は、テンプレートで指定されたリソースをプロビジョニングします。リソースのプロビジョニングには数分かかることがあります。AWS CloudFormation スタックのステータスが CREATE_COMPLETE の場合は、次のステップに進む準備ができています。

GET 統合レスポンスをテストするには

1. **data-transformation-tutorial-console** の AWS CloudFormation スタックの [リソース] タブで、API の物理 ID を選択します。
2. メインナビゲーションペインで [リソース]、[GET] メソッドの順に選択します。
3. [テスト] タブを選択します。タブを表示するには、右矢印ボタンを選択する必要がある場合があります。

   テストの出力には、次の内容が表示されます。

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

```
    "type": "fish",
    "price": 0.99
  }
]
```

この出力を PetStore マッピングテンプレート のマッピングテンプレートに従って変換します。

GET 統合レスポンスを変換するには

1. [統合レスポンス] タブを選択します。

   現在、マッピングテンプレートは定義されていないため、統合レスポンスは変換されません。
2. [デフォルト - レスポンス] で [編集] を選択します。
3. [マッピングテンプレート] を選択し、次の操作を行います。

   a. [マッピングテンプレートの追加] を選択します。

   b. [コンテンツタイプ] に、「**application/json**」と入力します。

   c. [テンプレート本文] で次のように入力します。

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

#end
]
```

   [Save] を選択します。

GET 統合レスポンスをテストするには

- [テスト] タブ、[テスト] の順に選択します。

  テストの出力に、変換されたレスポンスが表示されます。

```
[
  {
    "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
  }
]
```

POST メソッドからの入力データを変換するには

1. [POST] メソッドを選択します。
2. [統合リクエスト] タブを選択し、[統合リクエストの設定] で、[編集] を選択します。

   AWS CloudFormation テンプレートでは、いくつかの統合リクエストフィールドが入力されています。

   - 統合タイプは AWS のサービスです。
   - AWS のサービスは DynamoDB です。
   - HTTP メソッドは `POST` です。
   - アクションは `PutItem` です。
   - API Gateway が DynamoDB テーブルに項目を入力できるようにする実行ロールは `data-transformation-tutorial-console-APIGatewayRole` です。AWS CloudFormation は、API Gateway が DynamoDB とやり取りするための最小限のアクセス許可を持つように、このロールを作成済みです。

   DynamoDB テーブルの名前は未指定です。次の手順に従って名前を指定します。
3. [リクエスト本文のパススルー] で、[なし] を選択します。

   つまり、API はマッピングテンプレートを持たない Content-Type のデータを拒否します。
4. [マッピングテンプレート] を選択します。

5. [コンテンツタイプ] は application/json に設定されます。つまり、application/json 以外のコンテンツタイプは API によって拒否されます。統合パススルーの動作の詳細については、「統合パススルーの動作」を参照してください。
6. テキストエディタに次のコードを入力します。

```
{
    "TableName":"data-transformation-tutorial-console-ddb",
    "Item": {
        "id": {
            "N": $input.json("$.id")
        },
        "type": {
            "S": $input.json("$.type")
        },
        "price": {
            "N": $input.json("$.price")
        }
    }
}
```

このテンプレートでは、テーブルを data-transformation-tutorial-console-ddb として指定し、項目を id、type、price として設定します。項目は、POST メソッドの本体から取得されます。データモデルを使用してマッピングテンプレートを作成することもできます。詳細については、「API Gateway でリクエストの検証を使用する」を参照してください。

7. [保存] ボタンを選択し、マッピングテンプレートを保存します。

POST メソッドからメソッドと統合レスポンスを追加するには

AWS CloudFormation は、空白のメソッドと統合レスポンスを作成済みです。このレスポンスを編集して詳細情報を入力します。レスポンスの編集方法の詳細については、「Amazon API Gateway API リクエストおよびレスポンスデータマッピングリファレンス」を参照してください。

1. [統合レスポンス] タブの [デフォルト - レスポンス] で、[編集] を選択します。
2. [マッピングテンプレート]、[マッピングテンプレートの追加] の順に選択します。
3. [コンテンツタイプ] に「**application/json**」と入力します。
4. コードエディタで、次の出力マッピングテンプレートを入力し、出力メッセージを送信します。

```
{ "message" : "Your response was recorded at $context.requestTime" }
```

コンテキスト変数の詳細については、「データモデル、オーソライザー、マッピングテンプレート、および CloudWatch アクセスログ記録用の $context 変数」を参照してください。

5. [保存] ボタンを選択し、マッピングテンプレートを保存します。

POST メソッドをテストする

[テスト] タブを選択します。タブを表示するには、右矢印ボタンを選択する必要がある場合があります。

1. リクエスト本文に、次の例を入力します。

```
{
          "id": "4",
          "type" : "dog",
          "price": "321"
}
```

2. [テスト] を選択します。

   出力に成功メッセージが表示されるはずです。

   DynamoDB コンソール (https://console.aws.amazon.com/dynamodb/) を開いて、サンプル項目がテーブルにあることを確認できます。

AWS CloudFormation スタックを削除するには

1. AWS CloudFormation コンソール (https://console.aws.amazon.com/cloudformation) を開きます。
2. AWS CloudFormation スタックを選択します。
3. [Delete] (削除) を選択し、選択を確定します。

AWS CLI を使用してデータ変換を設定する

このチュートリアルでは、.zip ファイル (data-transformation-tutorial-cli.zip} を使用して不完全な API と DynamoDB テーブルを作成します。この不完全な API には、`http://petstore-demo-endpoint.execute-api.com/petstore/pets` HTTP エンドポイントと統合された `GET` メソッドを持つ `/pets` リソースがあります。`POST` メソッドを作成して DynamoDB テーブルに接続し、マッピングテンプレートを使用して DynamoDB テーブルにデータを入力します。

- 出力データは、PetStore マッピングテンプレート のマッピングテンプレートに従って変換します。
- POST メソッドを作成し、ユーザーがマッピングテンプレートを使用して Amazon DynamoDB テーブルにペット情報を POST できるようにします。

AWS CloudFormation スタックを作成するには

AWS CloudFormation のアプリケーション作成テンプレートをダウンロードして解凍します。

次のチュートリアルを完了するには、AWS Command Line Interface (AWS CLI) バージョン 2 が必要です。

コマンドが長い場合は、エスケープ文字 (\) を使用してコマンドを複数行に分割します。

> **Note**
>
> Windows では、一般的に使用する Bash CLI コマンドの一部 (zip など) が、オペレーティングシステムの組み込みターミナルでサポートされていません。Ubuntu および Bash の Windows 統合バージョンを取得するには、Windows Subsystem for Linux をインストールします。このガイドの CLI コマンドの例では、Linux フォーマットを使用しています。Windows CLI を使用している場合、インライン JSON ドキュメントを含むコマンドを再フォーマットする必要があります。

1. 次のコマンドを使用して AWS CloudFormation スタックを作成します。

```
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 は、テンプレートで指定されたリソースをプロビジョニングします。リソースのプロビジョニングには数分かかることがあります。次のコマンドを使用して AWS CloudFormation スタックのステータスを確認します。

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

3. AWS CloudFormation スタックのステータスが StackStatus: "CREATE_COMPLETE" になったら、次のコマンドを使用して関連する出力値を取得し、以後のステップで使用します。

```
aws cloudformation describe-stacks --stack-name data-transformation-tutorial-cli
--query "Stacks[*].Outputs[*].{OutputKey: OutputKey, OutputValue: OutputValue,
Description: Description}"
```

出力値は以下のとおりです。

- ApiRole。API Gateway が DynamoDB テーブルに項目を配置できるようにするロールの名前です。このチュートリアルの場合、ロール名は `data-transformation-tutorial-cli-APIGatewayRole-ABCDEFG` です。
- DDBTableName。DynamoDB テーブルの名前です。このチュートリアルの場合、インスタンス名は `data-transformation-tutorial-cli-ddb` です。
- ResourceId。GET メソッドと POST メソッドを公開するペットリソースの ID です。このチュートリアルの場合、リソース ID は `efg456` です。
- ApiId。API の ID です。このチュートリアルの場合、API の ID は `abc123` です。

データ変換の前に **GET** メソッドをテストするには

- 次のコマンドを使用して、GET メソッドをテストします。

```
aws apigateway test-invoke-method --rest-api-id abc123 \
          --resource-id efg456 \
          --http-method GET
```

テストの出力には、次の内容が表示されます。

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

```
    "type": "fish",
    "price": 0.99
  }
]
```

この出力を PetStore マッピングテンプレート のマッピングテンプレートに従って変換します。

GET 統合レスポンスを変換するには

- 次のコマンドを使用して、GET メソッドの統合レスポンスを更新します。*rest-api-id* と *resource-id* を実際の値に置き換えます。

  次のコマンドを使用して統合レスポンスを作成します。

```
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]"}'
```

GET メソッドをテストするには

- 次のコマンドを使用して GET メソッドをテストします。

```
aws apigateway test-invoke-method --rest-api-id abc123 \
  --resource-id efg456 \
  --http-method GET \
```

テストの出力に、変換されたレスポンスが表示されます。

```
[
  {
    "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
  }
]
```

POST メソッドを作成するには

1. 次のコマンドを使用して、/pets リソースで新しいメソッドを作成します。

```
aws apigateway put-method --rest-api-id abc123 \
  --resource-id efg456 \
  --http-method POST \
  --authorization-type "NONE" \
```

このメソッドを使用すると、AWS CloudFormation スタックで作成した DynamoDB テーブルにペット情報を送信できます。

2. 次のコマンドを使用して、POST メソッドで AWS のサービス統合を作成します。

```
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. 次のコマンドを使用して、POST メソッドの呼び出しが成功した場合のメソッドレスポンスを作成します。

```
aws apigateway put-method-response --rest-api-id abc123 \
  --resource-id efg456 \
  --http-method POST \
```

```
  --status-code 200
```

4. 次のコマンドを使用して、POST メソッドの呼び出しが成功した場合の統合レスポンスを作成します。

```
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\"}"}'
```

POST メソッドをテストするには

- 次のコマンドを使用して、POST メソッドをテストします。

```
aws apigateway test-invoke-method --rest-api-id abc123 \
  --resource-id efg456 \
  --http-method POST \
  --body '{\"id\": \"4\", \"type\": \"dog\", \"price\": \"321\"}'
```

出力に、成功のメッセージが表示されます。

AWS CloudFormation スタックを削除するには

- 次のコマンドを使用して AWS CloudFormation リソースを削除します。

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

完成したデータ変換の AWS CloudFormation テンプレート

次の例は、完成した AWS CloudFormation テンプレートです。これにより、API と、GET メソッドと POST メソッドを持つ /pets リソースを作成します。

- GET メソッドは、http://petstore-demo-endpoint.execute-api.com/petstore/pets HTTP エンドポイントからデータを取得します。出力データは、PetStore マッピングテンプレート のマッピングテンプレートに従って変換されます。

- POST メソッドを使用すると、ユーザーはマッピングテンプレートを使用して DynamoDB テーブルにペット情報を POST できます。

```
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
```

次のステップ

より複雑なマッピングテンプレートを試すには、以下の例を参照してください。

- サンプルのフォトアルバム「フォトアルバムの例」で、より複雑なモデルやマッピングテンプレートを参照してください。
- モデルの詳細については、「データモデルを理解する」を参照してください。
- さまざまなレスポンスコード出力をマッピングする方法については、「API Gateway コンソールを使用してリクエストとレスポンスのデータマッピングを設定する」を参照してください。
- API のメソッドリクエストデータからデータマッピングを設定する方法については、「API Gateway マッピングテンプレートとアクセスのログ記録の変数リファレンス」を参照してください。

マッピングテンプレートを使用して、API のリクエストおよびレスポンスパラメータとステータスコードをオーバーライドする

標準 API Gateway パラメータとレスポンスコードのマッピングテンプレートを使用して、パラメータの 1 対 1 のマッピング、および統合レスポンスステータスコードの一群 (正規表現と一致する) を単一のレスポンスステータスコードにマッピングすることができます。マッピングテンプレートオーバーライドにより多対 1 のマッピングを実行する柔軟性が提供されます。標準 API Gateway マッピング後のパラメータのオーバーライドが適用されます。条件付きでマップパラメータは本文の内容または他のパラメータ値に基づきます。プログラムによってオンザフライで新しいパラメータを作成し、統合エンドポイントによりオーバーライドステータスコードが返されます。あらゆるタイプのリクエストパラメータ、レスポンスヘッダー、またはレスポンスステータスコードはオーバーライドされます。

マッピングテンプレートオーバーライドの使用例を以下に示します。

- 2 つのパラメータを連結した新しいヘッダーを作成する (または既存のヘッダーをオーバライドする) には
- 本文の内容に基づく成功または失敗コードにレスポンスコードをオーバーライドするには
- パラメータまたはその他のパラメータの内容に基づいて、条件付きでパラメータを再マッピングするには
- JSON 本文の内容を反復処理し、ヘッダーやクエリ文字列にキーと値のペアを再マッピングするには

マッピングテンプレートオーバーライドを作成するには、マッピングテンプレートで以下のいずれかの $context 変数を使用します。

リクエストボディのマッピングテンプレート	レスポンスボディのマッピングテンプレート
`$context.requestOverride.header. header_name`	`$context.responseOverride.header. header_name`
`$context.requestOverride.path. path_name`	`$context.responseOverride.status`
`$context.requestOverride.querystring. querystring_name`	

> **Note**
>
> マッピングテンプレートオーバーライドは、データマッピングが不足しているため、プロキシ統合エンドポイントで使用することはできません。統合の詳細については、API Gateway API 統合タイプの選択 を参照してください。

> **Important**
>
> オーバーライドが終了します。オーバーライドは各パラメータに一度だけ適用されます。同じパラメータを複数回オーバーライドしようとすると、Amazon API Gateway から 5XX レスポンスが発生します。テンプレート全体で同じパラメータを複数回オーバーライドする必要がある場合は、変数を作成してテンプレートの終了時にオーバーライドを適用することをお勧めします。テンプレートはテンプレート全体が解析された後にのみ適用されることに注意してください。「チュートリアル: API Gateway コンソールを使用して API のリクエストパラメータとヘッダーをオーバーライドする」を参照してください。

以下のチュートリアルでは、API Gateway コンソールでマッピングテンプレートオーバーライドを作成およびテストする方法を示します。これらのチュートリアルでは、開始点として PetStore サンプル API を使用します。両方のチュートリアルは、PetStore サンプル API がすでに作成されていることを前提としています。

トピック

- チュートリアル: API Gateway コンソールを使用して、API の応答ステータスコードをオーバーライドする
- チュートリアル: API Gateway コンソールを使用して API のリクエストパラメータとヘッダーをオーバーライドする
- 例: API Gateway CLI を使用して API のリクエストパラメータとヘッダーをオーバーライドする
- 例: SDK for JavaScript を使用して API のリクエストパラメータとヘッダーをオーバーライドする

チュートリアル: API Gateway コンソールを使用して、API の応答ステータスコードをオーバーライドする

PetStore サンプル API を使用してペットを取得するには、`GET /pets/{petId}` の API メソッドリクエストを使用します。`{petId}` は、実行時に数字をとることができるパスパラメータです。

このチュートリアルでは、エラー状態が検出された場合、`GET` を `$context.responseOverride.status` にマッピングするマッピングテンプレートを作成することで、この `400` メソッドのレスポンスコードをオーバーライドします。

1. https://console.aws.amazon.com/apigateway で API Gateway コンソールにサインインします。
2. [API] で PetStore API を選択し、[リソース] を選択します。
3. [リソース] ツリーで、`/{petId}` の下の `GET` メソッドを選択します。
4. [テスト] タブを選択します。タブを表示するには、右矢印ボタンを選択する必要がある場合があります。
5. [petId] に「**-1**」と入力し、[テスト] を選択します。

   結果として、2 つのことが分かります。

   まず、[レスポンス本文] は範囲外エラーを示します。

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

次に、[ログ] ボックスの最後の行は `Method completed with status: 200` で終わります。

6. [統合レスポンス] タブの [デフォルト - レスポンス] で、[編集] を選択します。
7. [マッピングテンプレート] を選択します。
8. [マッピングテンプレートの追加] を選択します。
9. [コンテンツタイプ] に、「**application/json**」と入力します。
10. [テンプレート本文] で次のように入力します。

```
#set($inputRoot = $input.path('$'))
$input.json("$")
#if($inputRoot.toString().contains("error"))
#set($context.responseOverride.status = 400)
#end
```

11. [Save] を選択します。
12. [テスト] タブを選択します。
13. [petId] に「**-1**」と入力します。
14. 結果として、[Response Body (レスポンス本文)] は範囲外エラーを示します。

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

ただし、[Logs (ログ)] ボックスの最後の行は、`Method completed with status: 400` で終わるようになりました。

チュートリアル: API Gateway コンソールを使用して API のリクエストパラメータとヘッダーをオーバーライドする

このチュートリアルでは、GET を他の 2 つのヘッダーを組み合わせる新しいヘッダーにマッピングするマッピングテンプレートを作成すること

で、$context.requestOverride.header.*header_name* メソッドのリクエストヘッダーコードをオーバーライドします。

1. https://console.aws.amazon.com/apigateway で API Gateway コンソールにサインインします。
2. [API] で、PetStore API を選択します。
3. [リソース] ツリーで、/pet の下の GET メソッドを選択します。
4. [メソッドリクエスト] タブの [メソッドリクエストの設定] で、[編集] を選択します。
5. [HTTP リクエストヘッダー] を選択した後、[ヘッダーの追加] を選択します。
6. [名前] に**header1**と入力します。
7. [ヘッダーを追加] を選択し、**header2** という名前の 2 つ目のヘッダーを作成します。
8. [Save] を選択します。
9. [統合リクエスト] タブの [統合リクエストの設定] で、[編集] を選択します。
10. [リクエスト本文のパススルー] で、[テンプレートが定義されていない場合 (推奨)] を選択します。
11. [マッピングテンプレート] を選択し、次の操作を行います。
    a. [マッピングテンプレートの追加] を選択します。
    b. [コンテンツタイプ] に、「**application/json**」と入力します。
    c. [テンプレート本文] で次のように入力します。

```
#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. [Save] を選択します。
13. [テスト] タブを選択します。
14. [Headers (ヘッダー)] で、[{pets}] に、次のコードをコピーします。

```
header1:header1Val
header2:header2Val
```

15. [Test (テスト)] を選択します。

[ログ] に、このテキストを含むエントリが表示されます。

```
Endpoint request headers: {header3=header1Valheader2Val,
header2=header2Val, header1=foo, x-amzn-apigateway-api-id=<api-id>,
Accept=application/json, multivalueheader=foo,header1Valheader2Val}
```

例: API Gateway CLI を使用して API のリクエストパラメータとヘッダーをオーバーライドする

次の CLI の例では、put-integration コマンドを使用してレスポンスコードをオーバーライドする方法を示しています。

```
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>
```

ここで、*<REQUEST_TEMPLATE_MAP>* は、コンテンツタイプから適用されるテンプレートの文字列へのマッピングです。マッピングは以下のような構造になっています。

```
Content_type1=template_string,Content_type2=template_string
```

または、JSON 構文です。

```
{"content_type1": "template_string"
...}
```

次の例では、put-integration-response コマンドを使用して API レスポンスコードをオーバーライドする方法を示しています。

```
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>
```

<RESPONSE_TEMPLATE_MAP> は先ほどの *<REQUEST_TEMPLATE_MAP>* と同じ形式です。

例: SDK for JavaScript を使用して API のリクエストパラメータとヘッダーをオーバーライドする

次の例では、put-integration コマンドを使用してレスポンスコードをオーバーライドする方法を示しています。

リクエスト：

```
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
});
```

レスポンス：

```
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
});
```

API Gateway コンソールを使用してリクエストとレスポンスのデータマッピングを設定する

API Gateway コンソールを使用して API の統合リクエスト/応答を定義するには、以下の手順に従います。

> **Note**
>
> これらの手順は、すでに「API Gateway コンソールを使用して API 統合リクエストを設定する」のステップを完了していることを前提としています。

1. [リソース] ペインで、メソッドを選択します。
2. [統合リクエスト] タブの [統合リクエストの設定] で、[編集] を選択します。
3. [リクエスト本文のパススルー] のオプションを選択し、マッピングされていないコンテンツタイプのメソッドリクエスト本文を、変換することなく統合リクエストを通じて Lambda 関数、HTTP プロキシ、または AWS のサービスプロキシに渡す方法を設定します。3 つのオプションがあります。

   - 次のステップで定義しているように、メソッドリクエストのコンテンツタイプが、マッピングテンプレートに関連付けられたコンテンツタイプのいずれとも一致しないときに、統合リクエストをパススルーしてメソッドリクエスト本文を変換せずにバックエンドに渡す場合は、[リクエストの Content-Type ヘッダーに一致するテンプレートがない場合] を選択します。

     > **Note**
     >
     > API Gateway API を呼び出すときは、Integration リソースの `WHEN_NO_MATCH` プロパティの値として `passthroughBehavior` を設定して、このオプションを選択します。

   - マッピングテンプレートが統合リクエストに定義されていないときに、統合リクエストをパススルーしてメソッドリクエストボディを変換せずにバックエンドに渡す場合は、[テンプレートが定義されていない場合 (推奨)] を選択します。このオプションを選択したときにテンプレートが定義されている場合、マッピングされていないコンテンツタイプのメソッドリクエストは、HTTP 415 Unsupported Media Type レスポンスで拒否されます。

     > **Note**
     >
     > API Gateway API を呼び出すときは、Integration リソースの `WHEN_NO_TEMPLATE` プロパティの値として `passthroughBehavior` を設定して、このオプションを選択します。

- メソッドリクエストのコンテンツタイプが、統合リクエストに定義されたマッピングテンプレートと関連付けられたコンテンツタイプと一致しないか、統合リクエストにマッピングテンプレートが定義されていない場合に、メソッドリクエストを渡さないようにするには、[なし] を選択します。マッピングされていないコンテンツタイプのメソッドリクエストは、HTTP 415 Unsupported Media Type レスポンスで拒否されます。

> **Note**
>
> API Gateway API を呼び出すときは、Integration リソースの NEVER プロパティの値として `passthroughBehavior` を設定して、このオプションを選択します。

統合のパススルー動作の詳細については、「統合パススルーの動作」を参照してください。

4. HTTP プロキシまたは AWS サービスプロキシでは、統合リクエストに定義されたパスパラメータ、クエリ文字列パラメータ、またはヘッダーパラメータを対応する HTTP プロキシまたは AWS サービスプロキシのメソッドリクエスト内のパスパラメータ、クエリ文字列パラメータ、またはヘッダーパラメータに関連付けるため、以下の操作を行います。

   a. [URL パスパラメータ]、[URL クエリ文字列パラメータ]、または [HTTP ヘッダー] をそれぞれ選択し、[パスの追加]、[クエリ文字列の追加]]、または [ヘッダーの追加] をそれぞれ選択します。

   b. [名前] に、HTTP プロキシまたは AWS サービスプロキシのパスパラメータ、クエリ文字列パラメータ、またはヘッダーパラメータの名前を入力します。

   c. [マッピング元] に、パスパラメータ、クエリ文字列パラメータ、またはヘッダーパラメータのマッピング値を入力します。以下のいずれかの形式を使用します。

      - [メソッドリクエスト] ページの定義に従って *parameter-name* と名付けられたパスパラメータの場合は、**method.request.path.*parameter-name***。
      - [メソッドリクエスト] ページの定義に従って *parameter-name* と名付けられたクエリ文字列パラメータの場合は、**method.request.querystring.*parameter-name***。
      - [メソッドリクエスト] ページの定義に従って *parameter-name* と名付けられた複数値のクエリ文字列パラメータの場合は、**method.request.multivaluequerystring.*parameter-name***。
      - [メソッドリクエスト] ページの定義に従って *parameter-name* と名付けられたヘッダーパラメータの場合は、**method.request.header.*parameter-name***。

または、リテラル文字列値 (一重引用符のペアで囲まれた値) を統合ヘッダーに設定できます。

- [メソッドリクエスト] ページの定義に従って *parameter-name* と名付けられた複数値のヘッダーパラメータの場合は、**method.request.multivalueheader.*parameter-name***。

d. 別のパラメーターを追加するには、[追加] ボタンを選択します。

5. マッピングテンプレートを追加するには、[マッピングテンプレート] を選択します。
6. 入力リクエストのマッピングテンプレートを定義するには、[マッピングテンプレートの追加] を選択します。[コンテンツタイプ] にコンテンツタイプ (例: **application/json**) を入力します。次に、マッピングテンプレートを入力します。詳細については、「マッピングテンプレートについて」を参照してください。
7. [Save] を選択します。
8. バックエンドからの統合応答を、呼び出し元のアプリケーションへ返される API のメソッド応答とマッピングできます。これには、バックエンドの使用可能なヘッダーからクライアントが選択した応答ヘッダーへ戻り、バックエンドの応答ペイロードのデータ形式を API 指定の形式に変換します。このようなマッピングは、[メソッドレスポンス] および [統合レスポンス] を設定することで指定できます。

   Lambda 関数、HTTP プロキシ、または AWS のサービスプロキシから返される HTTP ステータスコードに基づいてカスタムレスポンスデータ形式をメソッドで受け取るには、以下を実行します。

   a. [統合レスポンス] を選択します。[デフォルト - レスポンス] で [編集] を選択してメソッドから 200 HTTP レスポンスコードの設定を指定するか、[レスポンスを作成] を選択してメソッドからその他の任意の HTTP レスポンスステータスコードの設定を指定します。

   b. [Lambda エラーの正規表現] (Lambda 関数の場合) または [HTTP ステータスの正規表現] (HTTP プロキシまたは AWS のサービスプロキシの場合) に正規表現を入力し、どの Lambda 関数エラー文字列 (Lambda 関数の場合) または HTTP レスポンスステータスコード (HTTP プロキシまたは AWS のサービスプロキシの場合) をこの出力マッピングにマップするかを指定します。たとえば、すべての 2xx HTTP 応答ステータスコードを HTTP プロキシからこの出力マッピングにマッピングするには、[HTTP status regex (HTTP ステータス正規表現)] に「**2\d{2}**」と入力します。「Invalid Request」を含むエラーメッセージを Lambda 関数から `400 Bad Request` レスポンスに返すには、[Lambda エラーの正規表現] として「**.*Invalid request.***」と入力します。一方、すべてのマッピングされていないエラーメッセージで `400 Bad Request` を Lambda から返すには、[Lambda エラーの正

規表現] に「**(\n|.)+**」と入力します。この最後の正規表現は、API のデフォルトのエラーのレスポンスに使用できます。

> **Note**
>
> API Gateway は、レスポンスマッピングに Java パターンスタイルの正規表現を使用します。詳細については、Oracle ドキュメントの「パターン」を参照してください。
> エラーパターンは Lambda レスポンスの `errorMessage` プロパティの文字列全体に対してマッチングされます。これは、Node.js の場合は `callback(errorMessage)`、Java の場合は `throw new MyException(errorMessage)` によって設定されます。また、エスケープ文字は正規表現が適用されるまではエスケープされません。
> '.+' を選択パターンとして使用してレスポンスをフィルタする場合は、改行 ('\n') 文字を含むレスポンスには一致しない可能性があることに注意してください。

c. 有効になっている場合は、[メソッドレスポンスのステータス] で、[メソッドレスポンス] ページで定義した HTTP レスポンスステータスコードを選択します。

d. [ヘッダーマッピング] で、[メソッドレスポンス] ページの HTTP レスポンスステータスコードで定義したヘッダーごとに、マッピング値を指定します。[マッピングの値] で、以下のいずれかの形式を使用します。

- **`integration.response.multivalueheaders.`*`header-name`*** のようにします。*`header-name`* はバックエンドからの複数値の応答ヘッダーの名前です。

  たとえば、バックエンド応答の `Date` ヘッダーを API メソッドの応答の `Timestamp` ヘッダーとして返すには、[レスポンスヘッダー] 列に [Timestamp (タイムスタンプ)] エントリを含め、関連する [マッピング値] は [integration.response.multivalueheaders.Date] に設定します。

- **`integration.response.header.`*`header-name`*** のようにします。*`header-name`* はバックエンドからの単一値の応答ヘッダーの名前です。

  たとえば、バックエンド応答の `Date` ヘッダーを API メソッドの応答の `Timestamp` ヘッダーとして返すには、[レスポンスヘッダー] 列に [Timestamp (タイムスタンプ)] エントリを含め、関連する [マッピング値] は [integration.response.header.Date] に設定します。

e. [マッピングテンプレート]、[マッピングテンプレートの追加] の順に選択します。[コンテンツタイプ] ボックスに、Lambda 関数、HTTP プロキシ、または AWS のサービスプロキシからメソッドに渡すデータのコンテンツタイプを入力します。次に、マッピングテンプレートを入力します。詳細については、「マッピングテンプレートについて」を参照してください。

f. [Save] を選択します。

API Gateway のデータモデルとマッピングテンプレートの例

以下のセクションでは、API Gateway で独自の API の出発点として使用できるモデルとマッピングテンプレートの例を示します。データ変換の詳細については、「マッピングテンプレートについて」を参照してください。データモデルの詳細については、「the section called “データモデルを理解する”」を参照してください。

トピック

- フォトアルバムの例
- ニュース記事サンプル

フォトアルバムの例

次の例は、API Gateway のフォトアルバム API を示しています。データ変換の例、追加のモデル、マッピングテンプレートを提供します。

トピック

- データ変換の例
- 写真データの入力モデル
- 写真データの出力モデル
- 写真データの入力マッピングテンプレート

データ変換の例

次の例は、Velocity Template Language (VTL) マッピングテンプレートを使用して写真に関する入力データを変換する方法を示しています。Velocity Template Language の詳細については、Apache Velocity - VTL リファレンスを参照してください。

入力データ

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

出力マッピングテ

```
#set($inputRoot = $input.path('$'))
{
  "photos": [
#foreach($elem in $inputRoot.photos.photo)
    {
```

ンプレート

```
      "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
  ]
}
```

出力データ

```
{
  "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
    }
  ]
}
```

写真データの入力モデル

入力データのモデルを定義できます。この入力モデルでは、写真 1 枚をアップロードする必要があります。また、ページごとに最低 10 枚の写真を指定します。この入力モデルを使用して SDK を生成したり、API のリクエストの検証を有効にしたりできます。

```
{
```

```
  "$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" }
            }
          }
        }
      }
    }
  }
}
```

写真データの出力モデル

出力データのモデルを定義できます。このモデルはメソッドレスポンスモデルに使用できます。これは、API 用に厳密に型指定した SDK を生成するときに必要です。これにより、出力が Java や Objective-C の適切なクラスにキャストされます。

```
{
  "$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" }
        }
      }
    }
  }
}
```

写真データの入力マッピングテンプレート

マッピングテンプレートを定義して入力データを変更できます。入力データを変更して、さらに関数統合や統合レスポンスを行うことができます。

```
#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
    ]
  }
}
```

ニュース記事サンプル

次の例は、API Gateway のニュース記事 API を示しています。データ変換の例、追加のモデル、マッピングテンプレートを提供します。

トピック

- データ変換の例
- ニュースデータの入力モデル
- ニュースデータの出力モデル
- ニュースデータの入力マッピングテンプレート

データ変換の例

次の例は、Velocity Template Language (VTL) マッピングテンプレートを使用してニュース記事に関する入力データをどのように変換できるかを示しています。Velocity Template Language の詳細については、Apache Velocity - VTL リファレンスを参照してください。

入力データ

```
{
  "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
}
```

出力マッピングテンプレート

```
#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
}
```

出力データ

```
{
  "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
}
```

ニュースデータの入力モデル

入力データのモデルを定義できます。この入力モデルでは、ニュース記事に URL、タイトル、本文を含める必要があります。この入力モデルを使用して SDK を生成したり、API のリクエストの検証を有効にしたりできます。

```
{
  "$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" }
  }
}
```

ニュースデータの出力モデル

出力データのモデルを定義できます。このモデルはメソッドレスポンスモデルに使用できます。これは、API 用に厳密に型指定した SDK を生成するときに必要です。これにより、出力が Java や Objective-C の適切なクラスにキャストされます。

```
{
  "$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" }
        }
      }
    }
  }
}
```

ニュースデータの入力マッピングテンプレート

マッピングテンプレートを定義して入力データを変更できます。入力データを変更して、さらに関数統合や統合レスポンスを行うことができます。

```
#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 リクエストおよびレスポンスデータマッピングリファレンス

このセクションでは、API のメソッドリクエストデータ (`context`、`stage`、または `util` 変数に保存された他のデータも含む) から対応する統合リクエストパラメータへのデータマッピングと、統合レスポンスデータ (他のデータも含む) からメソッドレスポンスパラメータへのデータマッピングを設定する方法について説明します。メソッドリクエストデータには、リクエストパラメータ (パス、クエリ文字列、ヘッダー) と本文が含まれます。統合レスポンスデータには、レスポンスパラメータ (ヘッダー) と本文が含まれます。ステージ変数の使用の詳細については、「Amazon API Gateway のステージ変数リファレンス」を参照してください。

トピック

- メソッドリクエストデータを統合リクエストパラメータにマッピングする
- 統合レスポンスデータをメソッドレスポンスヘッダーにマッピングする
- メソッドと統合との間でリクエストとレスポンスのペイロードをマッピングする
- 統合パススルーの動作

メソッドリクエストデータを統合リクエストパラメータにマッピングする

パス変数、クエリ文字列、またはヘッダーの形式の統合リクエストパラメータは、定義されたどのメソッドリクエストパラメータとペイロードからもマッピングできます。

次の表で、*`PARAM_NAME`* は、特定のパラメータ型のメソッドリクエストパラメータの名前です。正規表現パターン `'^[a-zA-Z0-9._$-]+$]'` に一致する必要があります。これは参照される前に定義済みである必要があります。*`JSONPath_EXPRESSION`* は、リクエストまたはレスポンスの本文の JSON フィールドの JSONPath 式です。

> **Note**
>
> "$" プレフィックスは、この構文では省略されます。

統合リクエストデータのマッピング式

マッピングされたデータソース	マッピング式
メソッドリクエストのパス	`method.request.path.` *PARAM_NAME*
メソッドリクエストのクエリ文字列	`method.request.querystring.` *PARAM_NAME*
複数値メソッドリクエストのクエリ文字列	`method.request.multivaluequerystring.` *PARAM_NAME*
メソッドリクエストのヘッダー	`method.request.header.` *PARAM_NAME*
複数値メソッドリクエストのヘッダー	`method.request.multivalueheader.` *PARAM_NAME*
メソッドリクエストボディ	`method.request.body`
メソッドリクエストボディ (JsonPath)	`method.request.body.` *JSONPath_EXPRESSION* .
ステージ変数	`stageVariables.` *VARIABLE_NAME*
コンテキスト変数	サポートされるコンテキスト変数の 1 つである必要がある `context.`*VARIABLE_NAME* 。
静的な値	*'STATIC_VALUE'* 。*STATIC_VALUE* はリテラル文字列で、単一引用符のペアで囲まれている必要があります。

Example OpenAPI におけるメソッドリクエストパラメータからのマッピング

次の例は、以下のマッピングを行う OpenAPI スニペットです。

- `methodRequestHeaderParam` という名前のメソッドリクエストのヘッダーから、`integrationPathParam` という名前の統合リクエストパスパラメータへのマッピング
- `methodRequestQueryParam` という名前の複数値のメソッドリクエストから、`integrationQueryParam` という名前の統合リクエストのクエリ文字列へのマッピング

```
...
"requestParameters" : {

    "integration.request.path.integrationPathParam" :
 "method.request.header.methodRequestHeaderParam",
    "integration.request.querystring.integrationQueryParam" :
 "method.request.multivaluequerystring.methodRequestQueryParam"

}
...
```

統合リクエストのパラメータは、JSONPath 式 を使用して JSON リクエストボディのフィールドからマッピングすることもできます。次の表は、メソッドリクエストの本文および JSON フィールドのマッピング式を示しています。

Example OpenAPI におけるメソッドリクエストボディからのマッピング

次の例は、1) メソッドリクエストボディを body-header という名前の統合リクエストのヘッダーにマッピングし、2) 本文の JSON フィールドを JSON 式 (petstore.pets[0].name。$. プレフィックスなし) で示されるようにマッピングする OpenAPI スニペットを示しています。

```
...
"requestParameters" : {

    "integration.request.header.body-header" : "method.request.body",
    "integration.request.path.pet-name" : "method.request.body.petstore.pets[0].name",

}
...
```

統合レスポンスデータをメソッドレスポンスヘッダーにマッピングする

メソッドレスポンスヘッダーのパラメータは、任意の統合レスポンスヘッダー/統合レスポンス本文、$context 変数、または静的な値からマッピングできます。

メソッドレスポンスヘッダーのマッピング式

マッピングされたデータソース	マッピング式
統合レスポンスのヘッダー	`integration.response.header.`*PARAM_NAME*
統合レスポンスのヘッダー	`integration.response.multivalueheader.`*PARAM_NAME*
統合レスポンスの本文	`integration.response.body`
統合レスポンスの本文 (JsonPath)	`integration.response.body.`*JSONPath_EXPRESSION*
ステージ変数	`stageVariables.`*VARIABLE_NAME*
コンテキスト変数	サポートされるコンテキスト変数の 1 つである必要がある `context.`*VARIABLE_NAME* 。
静的な値	*'STATIC_VALUE'* 。*STATIC_VALUE* はリテラル文字列で、単一引用符のペアで囲まれている必要があります。

Example OpenAPI における統合レスポンスからのデータマッピング

次の例は、1) 統合レスポンスの `redirect.url` JSONPath フィールドをリクエストレスポンスの `location` ヘッダーにマッピングし、2) 統合レスポンスの `x-app-id` ヘッダーをメソッドレスポンスの `id` ヘッダーにマッピングする OpenAPI スニペットを示しています。

```
...
"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",

}
...
```

メソッドと統合との間でリクエストとレスポンスのペイロードをマッピングする

API Gateway では、Velocity Template Language (VTL) エンジンを使用して、統合リクエストと統合レスポンスの本文マッピングテンプレートを処理します。マッピングテンプレートは、メソッドリクエストのペイロードを対応する統合リクエストのペイロードに変換し、統合レスポンスの本文をメソッドレスポンスの本文に変換します。

VTL テンプレートは、JSONPath 式、呼び出しコンテキストやステージ変数などの他のパラメータ、およびユーティリティ関数を使用して JSON データを処理します。

ペイロードのデータ構造を記述するモデルが定義されている場合、API Gateway はそのモデルを使用して統合リクエストや統合レスポンスのスケルトンマッピングテンプレートを生成できます。スケルトンテンプレートを利用してマッピング VTL スクリプトをカスタマイズしたり拡張したりできます。ただし、ペイロードのデータ構造のモデルを定義せずに、マッピングテンプレートを最初から作成することもできます。

VTL マッピングテンプレートを選択する

API Gateway は、以下のロジックを使用して、Velocity Template Language (VTL) のマッピングテンプレートの選択、メソッドリクエストから対応する統合リクエストへのペイロードのマッピング、統合レスポンスから対応するメソッドレスポンスへのペイロードのマッピングを行います。

リクエストペイロードの場合、API Gateway はリクエストの `Content-Type` ヘッダー値をキーとして使用して、リクエストペイロードのマッピングテンプレートを選択します。レスポンスペイロードの場合、API Gateway は受信リクエストの `Accept` ヘッダー値をキーとして使用して、マッピングテンプレートを選択します。

`Content-Type` ヘッダーがリクエストにない場合、API Gateway はデフォルト値が `application/json` であると見なします。そのようなリクエストでは、API Gateway は `application/json` をデフォルトキーとして使用してマッピングテンプレートを選択します (定義されている場合)。このキーに一致するテンプレートがない場合、passthroughBehavior プロパティが `WHEN_NO_MATCH` または `WHEN_NO_TEMPLATES` に設定されていれば、API Gateway はペイロードをマッピングせずに渡します。

`Accept` ヘッダーがリクエストで指定されていない場合、API Gateway はそのデフォルト値が `application/json` であると見なします。この場合、API Gateway は `application/json` の既存のマッピングテンプレートを選択してレスポンスペイロードをマッピングしま

す。application/json のテンプレートが定義されていない場合、API Gateway は最初の既存のテンプレートを選択してデフォルトとして使用し、レスポンスペイロードをマッピングします。同様に、指定された Accept ヘッダー値が既存のテンプレートキーに一致しない場合、API Gateway は最初の既存のテンプレートを使用します。テンプレートが定義されていない場合、API Gateway はレスポンスペイロードをマッピングしないまま渡します。

たとえば、API でリクエストペイロードの application/json テンプレートが定義されており、レスポンスペイロードの application/xml テンプレートが定義されているとします。クライアントが "Content-Type : application/json" と、 "Accept : application/xml" ヘッダーをリクエストに設定している場合、リクエストペイロードとレスポンスペイロードの両方が対応するマッピングテンプレートを使用して処理されます。Accept:application/xml ヘッダーがない場合、application/xml マッピングテンプレートを使用してレスポンスペイロードがマッピングされます。マッピングされていないレスポンスペイロードを代わりに返すには、application/json の空のテンプレートを設定する必要があります。

マッピングテンプレートを選択するとき、Accept ヘッダーおよび Content-Type ヘッダーから使用されるのは MIME タイプだけです。たとえば、"Content-Type: application/json; charset=UTF-8" のヘッダーの場合、リクエストテンプレートで application/json キーが選択されます。

統合パススルーの動作

非プロキシ統合の場合は、メソッドリクエストにペイロードが含まれ、Content-Type ヘッダーが、指定されたマッピングテンプレートに一致しないか、マッピングテンプレートが定義されていない場合は、クライアントで提供されたリクエストペイロードを、統合リクエストを通じて変換せずにバックエンドに渡す選択ができます。このプロセスは統合パススルーと呼ばれます。

プロキシ統合の場合は、API Gateway がリクエスト全体をバックエンドに渡します。パススルー動作を変更するオプションはありません。

入力リクエストの実際のパススルー動作は、統合リクエストの設定中に、指定されたマッピングテンプレートに選択したオプションと、クライアントが受信リクエストで設定する Content Type ヘッダーによって決まります。3 つのオプションがあります。

リクエストした Content-Type ヘッダーと一致するテンプレートがない場合

メソッドリクエストのコンテンツタイプが、マッピングテンプレートと関連付けられたコンテンツタイプに一致しない場合に、統合リクエストをパススルーしてメソッドリクエスト本文を変換せずにバックエンドに渡す場合は、このオプションを選択します。

API Gateway API を呼び出すときは、[統合] の passthroughBehavior プロパティの値として WHEN_NO_MATCH を設定して、このオプションを選択します。

テンプレートが定義されていない場合 (推奨)

統合リクエストでマッピングテンプレートが定義されていないときに、統合リクエストをパススルーしてメソッドリクエスト本文を変換せずにバックエンドに渡す場合は、このオプションを選択します。このオプションを選択したときにテンプレートが定義されている場合、マッピングされていないコンテンツタイプのメソッドリクエストは、HTTP 415 Unsupported Media Type レスポンスで拒否されます。

API Gateway API を呼び出すときは、[統合] の passthroughBehavior プロパティの値として WHEN_NO_TEMPLATES を設定して、このオプションを選択します。

なし

統合リクエストでマッピングテンプレートが定義されていないときに、統合リクエストをパススルーしてメソッドリクエストボディを変換せずにバックエンドに渡さない場合は、このオプションを選択します。このオプションを選択したときにテンプレートが定義されている場合、マッピングされていないコンテンツタイプのメソッドリクエストは、HTTP 415 Unsupported Media Type レスポンスで拒否されます。

API Gateway API を呼び出すときは、[統合] の passthroughBehavior プロパティの値として NEVER を設定して、このオプションを選択します。

次の例では、可能なパススルー動作について示します。

例 1: application/json コンテンツタイプで 1 つのマッピングテンプレートが統合リクエストで定義されている。

Content-Type ヘッダー\選択されたパススルーオプション	**WHEN_NO_MATCH**	**WHEN_NO_TEMPLATES**	**NEVER**
なし (デフォルト: application/json)	リクエストペイロードはテンプレートを使用して変換されます。	リクエストペイロードはテンプレートを使用して変換されます。	リクエストペイロードはテンプレートを使用して変換されます。

Content-Type ヘッダー\選択されたパススルーオプション	**WHEN_NO_MATCH**	**WHEN_NO_TEMPLATES**	**NEVER**
application/json	リクエストペイロードはテンプレートを使用して変換されます。	リクエストペイロードはテンプレートを使用して変換されます。	リクエストペイロードはテンプレートを使用して変換されます。
application/xml	リクエストペイロードは変換されず、そのままバックエンドに送信されます。	リクエストは、HTTP 415 Unsupported Media Type レスポンスで拒否されます。	リクエストは、HTTP 415 Unsupported Media Type レスポンスで拒否されます。

例 2: application/xml コンテンツタイプで 1 つのマッピングテンプレートが統合リクエストで定義されている。

Content-Type ヘッダー\選択されたパススルーオプション	**WHEN_NO_MATCH**	**WHEN_NO_TEMPLATES**	**NEVER**
なし (デフォルト: application/json)	リクエストペイロードは変換されず、そのままバックエンドに送信されます。	リクエストは、HTTP 415 Unsupported Media Type レスポンスで拒否されます。	リクエストは、HTTP 415 Unsupported Media Type レスポンスで拒否されます。
application/json	リクエストペイロードは変換されず、そのままバックエンドに送信されます。	リクエストは、HTTP 415 Unsupported Media Type レスポンスで拒否されます。	リクエストは、HTTP 415 Unsupported Media Type レスポンスで拒否されます。
application/xml	リクエストペイロードはテンプレートを	リクエストペイロードはテンプレートを	リクエストペイロードはテンプレートを

Content-Type ヘッダー\選択されたパススルーオプション	WHEN_NO_MATCH	WHEN_NO_TEMPLATES	NEVER
	使用して変換されます。	使用して変換されます。	使用して変換されます。

API Gateway マッピングテンプレートとアクセスのログ記録の変数リファレンス

このセクションでは、Amazon API Gateway がデータモデル、オーソライザー、マッピングテンプレート、および CloudWatch アクセスログ記録での使用に定義している変数と関数についてのリファレンス情報を提供します。これらの変数と関数の使用方法の詳細については、「マッピングテンプレートについて」を参照してください。Velocity Template Language (VTL) の詳細については、「VTL Reference」を参照してください。

トピック

- データモデル、オーソライザー、マッピングテンプレート、および CloudWatch アクセスログ記録用の $context 変数
- $context 変数テンプレートの例
- アクセスログ記録専用の $context 変数
- $input 変数
- $input 変数テンプレートの例
- $stageVariables
- $util 変数

> **Note**
>
> `$method` と `$integration` の変数については、「the section called “リクエストおよびレスポンスデータマッピングのリファレンス”」を参照してください。

データモデル、オーソライザー、マッピングテンプレート、および CloudWatch アクセスログ記録用の **$context** 変数

以下の $context 変数は、データモデル、オーソライザー、マッピングテンプレート、そして CloudWatch アクセスログ記録に使用することができます。API Gateway は、コンテキスト変数を追加する場合があります。

CloudWatch アクセスログ記録にのみ使用できる $context 変数については、「the section called "アクセスログ記録専用の $context 変数"」を参照してください。

パラメータ	説明
$context.accountId	API 所有者の AWS アカウント ID。
$context.apiId	API Gateway が API に割り当てる識別子。
$context.authorizer.claims.*property*	メソッドの呼び出し側が認証に成功した後に Amazon Cognito ユーザープールから返されるクレームのプロパティ。詳細については、「the section called "REST API のオーソライザーとして Amazon Cognito ユーザープールを使用する"」を参照してください。 **Note** $context.authorizer.claims を呼び出すと NULL が返されます。
$context.authorizer.principalId	クライアントにより送信され、API Gateway Lambda オーソライザー (以前のカスタムオーソライザー) から返されたトークンと関連付けられたプリンシパルユーザー ID。詳細については、「the section called "Lambda 認証を使用する"」を参照してください。
$context.authorizer.*property*	API Gateway Lambda オーソライザーの関数から返された context マップの指定されたキー/値ペアの文字列化された値。たとえば、オーソ

パラメータ	説明
	ライザーが次の context マップを返すとします。 `"context" : { "key": "value", "numKey": 1, "boolKey": true }` $context.authorizer.key の呼び出しでは "value" 文字列が返され、$context.authorizer.numKey の呼び出しでは "1" 文字列が返され、$context.authorizer.boolKey の呼び出しでは "true" 文字列が返されます。 詳細については、「the section called “Lambda 認証を使用する”」を参照してください。
$context.awsEndpointRequestId	AWS エンドポイントのリクエスト ID。
$context.deploymentId	API デプロイの ID。
$context.domainName	API の呼び出しに使用された完全ドメイン名。これは、受信 Host ヘッダーと同じである必要があります。
$context.domainPrefix	$context.domainName の 1 つ目のラベル。

パラメータ	説明
`$context.error.message`	API Gateway のエラーメッセージを含む文字列。この変数は、Velocity Template Language エンジン、およびアクセスログ記録では処理されない、GatewayResponse 本文マッピングテンプレートの単純な変数の置換でのみ使用できます。詳細については、the section called “メトリクス”および the section called “エラーレスポンスをカスタマイズするためのゲートウェイレスポンスのセットアップ”を参照してください。
`$context.error.messageString`	`$context.error.message` を引用符で囲んだ値、つまり `"$context.error.message"`。
`$context.error.responseType`	GatewayResponse の type。この変数は、Velocity Template Language エンジン、およびアクセスログ記録では処理されない、GatewayResponse 本文マッピングテンプレートの単純な変数の置換でのみ使用できます。詳細については、the section called “メトリクス”および the section called “エラーレスポンスをカスタマイズするためのゲートウェイレスポンスのセットアップ”を参照してください。
`$context.error.validationErrorString`	詳細な検証エラーメッセージを含む文字列。
`$context.extendedRequestId`	API Gateway が生成して API リクエストに割り当てる拡張 ID。拡張リクエスト ID には、デバッグとトラブルシューティングに役立つ情報が含まれています。

パラメータ	説明
`$context.httpMethod`	使用される HTTP メソッドです。有効な値には、DELETE、GET、HEAD、OPTIONS、PATCH、POST および PUT があります。
`$context.identity.accountId`	リクエストに関連付けられた AWS アカウント ID です。
`$context.identity.apiKey`	API キーを必要とする API メソッドの場合、この変数はメソッドリクエストに関連付けられている API キーです。API キーを必要としないメソッドの場合、この変数は null になります。詳細については、「the section called “使用量プラン”」を参照してください。
`$context.identity.apiKeyId`	API キーを必要とする API リクエストに関連付けられた API キー ID。
`$context.identity.caller`	リクエストに署名した発信者のプリンシパル ID。IAM 認証を使用するリソースでサポートされています。
`$context.identity.cognitoAuthenticationProvider`	リクエストを行う発信者が使用する Amazon Cognito 認証プロバイダーのカンマ区切りのリスト。リクエストが Amazon Cognito 認証情報で署名されている場合にのみ使用できます。 たとえば、Amazon Cognito ユーザープールのアイデンティティの場合、`cognito-idp.`*`region`*`.amazonaws.com/`*`user_pool_id`*` ,cognito-idp.`*`region`*`.amazonaws.com/`*`user_pool_id`*`:CognitoSignIn:`*`token subject claim`* 詳しくは、Amazon Cognito デベロッパーガイドの「Amazon Cognito ID プール (フェデレーティッド ID)」を参照してください。

パラメータ	説明
`$context.identity.cognitoAuthenticationType`	リクエストを行う発信者の Amazon Cognito 認証タイプ。リクエストが Amazon Cognito 認証情報で署名されている場合にのみ使用できます。有効な値は、認証されたアイデンティティ`authenticated` および認証されていないアイデンティティ`unauthenticated` です。
`$context.identity.cognitoIdentityId`	リクエストを行う発信者の Amazon Cognito ID。リクエストが Amazon Cognito 認証情報で署名されている場合にのみ使用できます。
`$context.identity.cognitoIdentityPoolId`	リクエストを行う発信者の Amazon Cognito ID プール ID。リクエストが Amazon Cognito 認証情報で署名されている場合にのみ使用できます。
`$context.identity.principalOrgId`	AWS 組織 ID。
`$context.identity.sourceIp`	API Gateway エンドポイントへのリクエストを行う即時 TCP 接続のソース IP アドレス。
`$context.identity.clientCert.clientCertPem`	クライアントが相互 TLS 認証中に提示した PEM エンコードされたクライアント証明書。相互 TLS が有効なカスタムドメイン名を使用してクライアントが API にアクセスすると、アクセスログに存在します。相互 TLS 認証が失敗した場合にのみ、アクセスログに存在します。
`$context.identity.clientCert.subjectDN`	クライアントが提示する証明書のサブジェクトの識別名。相互 TLS が有効なカスタムドメイン名を使用してクライアントが API にアクセスすると、アクセスログに存在します。相互 TLS 認証が失敗した場合にのみ、アクセスログに存在します。

パラメータ	説明
$context.identity.clientCert.issuerDN	クライアントが提示する証明書の発行者の識別名。相互 TLS が有効なカスタムドメイン名を使用してクライアントが API にアクセスすると、アクセスログに存在します。相互 TLS 認証が失敗した場合にのみ、アクセスログに存在します。
$context.identity.clientCert.serialNumber	証明書のシリアル番号。相互 TLS が有効なカスタムドメイン名を使用してクライアントが API にアクセスすると、アクセスログに存在します。相互 TLS 認証が失敗した場合にのみ、アクセスログに存在します。
$context.identity.clientCert.validity.notBefore	証明書が無効になる前の日付。相互 TLS が有効なカスタムドメイン名を使用してクライアントが API にアクセスすると、アクセスログに存在します。相互 TLS 認証が失敗した場合にのみ、アクセスログに存在します。
$context.identity.clientCert.validity.notAfter	証明書が無効になった日付。相互 TLS が有効なカスタムドメイン名を使用してクライアントが API にアクセスすると、アクセスログに存在します。相互 TLS 認証が失敗した場合にのみ、アクセスログに存在します。
$context.identity.vpcId	API Gateway エンドポイントへのリクエストを行う VPC の VPC ID。
$context.identity.vpceId	API Gateway エンドポイントへのリクエストを行う VPC エンドポイントの VPC エンドポイント ID。プライベート API がある場合にのみ表示されます。
$context.identity.user	リソースアクセスに対して許可されるユーザーのプリンシパル識別子。IAM 認証を使用するリソースでサポートされています。

パラメータ	説明
`$context.identity.userAgent`	API 発信者の `User-Agent` ヘッダー。
`$context.identity.userArn`	認証後に識別された有効ユーザーの Amazon リソースネーム (ARN) です。詳細については、「https://docs.aws.amazon.com/IAM/latest/UserGuide/id_users.html」を参照してください。
`$context.isCanaryRequest`	リクエストが canary に送信された場合は `true` を返し、リクエストが canary に送信されなかった場合は `false` を返します。canary が有効になっている場合にのみ表示されます。
`$context.path`	リクエストパス。たとえば、`https://{rest-api-id}.execute-api.{region}.amazonaws.com/{stage}/root/child` の非プロキシリクエスト URL の場合、`$context.path` 値は `/{stage}/root/child` 。
`$context.protocol`	`HTTP/1.1` などのリクエストプロトコル。 **Note** API Gateway API は HTTP/2 リクエストを受け入れることができますが、API Gateway は HTTP/1.1 を使用してバックエンド統合にリクエストを送信します。その結果、クライアントが HTTP/2 を使用するリクエストを送信した場合でも、リクエストプロトコルは HTTP/1.1 として記録されます。

パラメータ	説明
`$context.requestId`	リクエストの ID。クライアントは、このリクエスト ID を上書きできます。API Gateway が生成する一意のリクエスト ID に `$context.extendedRequestId` を使用します。
`$context.requestOverride.header.` *header_name*	リクエストヘッダーオーバーライド。このパラメータが定義されている場合、[Integration Request (統合リクエスト)] ペインで定義されている [HTTP Headers (HTTP ヘッダー)] の代わりに使用されるヘッダーが含まれます。詳細については、「マッピングテンプレートを使用して、API のリクエストおよびレスポンスパラメータとステータスコードをオーバーライドする」を参照してください。
`$context.requestOverride.path.` *path_name*	リクエストパスオーバーライド。このパラメータが定義されている場合、[Integration Request (統合リクエスト)] ペインで定義されている [URL Path Parameters (URL パスパラメータ)] の代わりに使用されるリクエストパスが含まれます。詳細については、「マッピングテンプレートを使用して、API のリクエストおよびレスポンスパラメータとステータスコードをオーバーライドする」を参照してください。
`$context.requestOverride.querystring.` *querystring_name*	リクエストクエリ文字列オーバーライド。このパラメータが定義されている場合、[Integration Request (統合リクエスト)] ペインで定義されている [URL Query String Parameters (URL クエリ文字列パラメータ)] の代わりに使用されるリクエストクエリ文字列が含まれます。詳細については、「マッピングテンプレートを使用して、API のリクエストおよびレスポンスパラメータとステータスコードをオーバーライドする」を参照してください。

パラメータ	説明
`$context.responseOverride.header. header_name`	レスポンスヘッダーオーバーライド。このパラメータが定義されている場合、[Integration Response (統合レスポンス)] ペインの [Default mapping (デフォルトのマッピング)] として定義されている [Response header (レスポンスヘッダー)] の代わりに返されるヘッダーが含まれます。詳細については、「マッピングテンプレートを使用して、API のリクエストおよびレスポンスパラメータとステータスコードをオーバーライドする」を参照してください。
`$context.responseOverride.status`	レスポンスステータスコードオーバーライド。このパラメータが定義されている場合、[Integration Response (統合レスポンス)] ペインの [Default mapping (デフォルトのマッピング)] として定義されている [Method response status (メソッドレスポンスのステータス)] の代わりに返されるステータスコードが含まれます。詳細については、「マッピングテンプレートを使用して、API のリクエストおよびレスポンスパラメータとステータスコードをオーバーライドする」を参照してください。
`$context.requestTime`	CLF 形式の要求時間 (`dd/MMM/yyyy:HH:mm:ss +-hhmm`)。
`$context.requestTimeEpoch`	エポック形式のリクエスト時間 (ミリ秒単位)。
`$context.resourceId`	API Gateway がリソースに割り当てる識別子です。

パラメータ	説明
$context.resourcePath	リソースへのパスです。たとえば、https://{rest-api-id}.execute-api.{region}.amazonaws.com/{stage}/root/child の非プロキシリクエスト URI の場合、$context.resourcePath 値は /root/child 。詳細については、「チュートリアル: HTTP 非プロキシ統合を使用して REST API をビルドする」を参照してください。
$context.stage	API リクエストのデプロイステージ (Beta、Prod など)。
$context.wafResponseCode	AWS WAF から受け取ったレスポンス: WAF_ALLOW または WAF_BLOCK 。ステージがウェブ ACL に関連付けられていない場合は、設定されません。詳細については、「the section called “AWS WAF”」を参照してください。
$context.webaclArn	リクエストを許可するかブロックするかを決定するために使用されるウェブ ACL の完全な ARN。ステージがウェブ ACL に関連付けられていない場合は、設定されません。詳細については、「the section called “AWS WAF”」を参照してください。

$context 変数テンプレートの例

API メソッドが、構造化データを特定のフォーマットにする必要があるバックエンドにデータを渡す場合、マッピングテンプレートで $context 変数を使用することをお勧めします。

次の例は、統合リクエストペイロード内で、受信 $context 変数をわずかに異なる名前のバックエンド変数にマッピングするマッピングテンプレートを示しています。

> **Note**
>
> 変数の 1 つは API キーです。この例では、メソッドが 1 つの API キーを要求することを前提としています。

```
{
    "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"
}
```

このマッピングテンプレートの出力は、次のようになります。

```
{
  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'
}
```

アクセスログ記録専用の $context 変数

以下の $context 変数は、アクセスログ記録でのみ使用できます。詳細については、「the section called “CloudWatch Logs”」を参照してください。(WebSocket API については、「the section called “メトリクス”」を参照してください。)

パラメータ	説明
$context.authorize.error	認可エラーメッセージ。
$context.authorize.latency	認可レイテンシー (ミリ秒単位)。
$context.authorize.status	認可の試行から返されたステータスコード。
$context.authorizer.error	オーソライザーから返されたエラーメッセージ。
$context.authorizer.integrationLatency	オーソライザーのレイテンシー (ミリ秒単位)。
$context.authorizer.integrationStatus	Lambda オーソライザーから返されたステータスコード。
$context.authorizer.latency	オーソライザーのレイテンシー (ミリ秒単位)。
$context.authorizer.requestId	AWS エンドポイントのリクエスト ID。
$context.authorizer.status	オーソライザーから返されたステータスコード。
$context.authenticate.error	認証の試行から返されたエラーメッセージ。
$context.authenticate.latency	認証レイテンシー (ミリ秒単位)。
$context.authenticate.status	認証の試行から返されたステータスコード。
$context.customDomain.basePathMatched	受信リクエストが一致した API マッピングのパス。クライアントがカスタムドメイン名を使用して API にアクセスする場合に適用されます。たとえば、クライアントがリクエストを https://api.example.com/v1/

パラメータ	説明
	orders/1234 に送信し、リクエストがパス v1/orders を持つ API マッピングと一致する場合 、値は v1/orders になります。詳細については、「the section called “API マッピング”」を参照してください。
$context.endpointType	API のエンドポイントタイプ。
$context.integration.error	統合から返されたエラーメッセージ。
$context.integration.integrationStatus	Lambda プロキシ統合の場合、バックエンドの Lambda 関数コードからではなく、AWS Lambda から返されるステータスコード。
$context.integration.latency	統合レイテンシー (ミリ秒)。$context.integrationLatency と同等です。
$context.integration.requestId	AWS エンドポイントのリクエスト ID。$context.awsEndpointRequestId と同等です。
$context.integration.status	統合から返されたステータスコード。Lambda プロキシ統合では、これは Lambda 関数コードから返されたステータスコードです。
$context.integrationLatency	統合レイテンシー (ミリ秒)。
$context.integrationStatus	Lambda プロキシ統合の場合、このパラメータはバックエンド Lambda 関数コードからではなく、AWS Lambda から返されるステータスコードを表します。
$context.responseLatency	レスポンスレイテンシー (ミリ秒)。
$context.responseLength	レスポンスペイロードの長さ (バイト単位)。
$context.status	メソッドレスポンスのステータス。

パラメータ	説明
`$context.waf.error`	から返されたエラーメッセージAWS WAF
`$context.waf.latency`	AWS WAF レイテンシー (ミリ秒単位)。
`$context.waf.status`	から返されたステータスコードAWS WAF
`$context.xrayTraceId`	X-Rayトレースのトレース ID。詳細については、「the section called “AWS X-Ray のセットアップ”」を参照してください。

`$input` 変数

`$input` 変数は、マッピングテンプレートによって処理されるメソッドリクエストペイロードとパラメータを示します。提供される関数は以下のとおりです。

変数と関数	説明
`$input.body`	文字列として raw リクエストペイロードを返します。
`$input.json(x)`	この関数は、JSONPath の式を評価し、結果を JSON 文字列で返します。 たとえば `$input.json('$.pets')` は、`pets` 構造を表す JSON 文字列を返します。 JSONPath の詳細については、JSONPath または JSONPath for Java を参照してください。
`$input.params()`	すべてのリクエストパラメータのマップを返します。インジェクション攻撃の可能性を避けるため、`$util.escapeJavaScript` を使用して結果をサニタイズすることをお勧めします。リクエストのサニタイズを完全に制御するには、テンプレートなしでプロキシ統合を使用

変数と関数	説明
	し、統合でリクエストのサニタイズを処理します。
`$input.params(x)`	パラメータ名文字列 x が指定された場合に、パス、クエリ文字列、またはヘッダー値から (この順番で検索される) メソッドリクエストパラメータの値を返します。インジェクション攻撃の可能性を避けるため、`$util.escapeJavaScript` を使用してパラメータをサニタイズすることをお勧めします。パラメータのサニタイズを完全に制御するには、テンプレートなしでプロキシ統合を使用し、統合でリクエストのサニタイズを処理します。

変数と関数	説明
`$input.path(x)`	JSONPath 式文字列 (x) を受け取り、結果の JSON オブジェクト表現を返します。これにより、Apache Velocity Template Language (VTL) でペイロード要素にネイティブにアクセスして操作できます。 たとえば、式 `$input.path('$.pets')` が次のようにオブジェクトを返すとします。 `[ { "id": 1, "type": "dog", "price": 249.99 }, { "id": 2, "type": "cat", "price": 124.99 }, { "id": 3, "type": "fish", "price": 0.99 } ]` `$input.path('$.pets').count()` は "3" を返します。 JSONPath の詳細については、JSONPath または JSONPath for Java を参照してください。

`$input` 変数テンプレートの例

以下の例は、マッピングテンプレートで `$input` 変数を使用する方法を示しています。これらの例を試すには、入力イベントを API Gateway に返す Mock 統合または Lambda 非プロキシ統合を使用できます。

パラメータマッピングテンプレートの例

次の例では、path、querystring、header を含むすべてのリクエストパラメータを JSON ペイロードを介して統合エンドポイントに渡します。

```
#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
  }
}
```

以下の入力パラメータを含むリクエストの場合:

- myparam という名前のパスパラメータ
- クエリ文字列パラメータ querystring1=value1,value2&querystring2=value3
- ヘッダー "header1" : "value1"、"header2" : "value2"、"header3" : "value3"

このマッピングテンプレートの出力は、次のようになります。

```
{
  "params" : {
        "path" : {
            "path" : "myparam"
                }
    ,        "querystring" : {
            "querystring1" : "value1,value2"
      ,           "querystring2" : "value3"
                }
    ,        "header" : {
            "header3" : "value3"
      ,           "header2" : "value2"
```

```
      ,            "header1" : "value1"
                }
          }
}
```

JSON マッピングテンプレートの例

クエリ文字列を取得する $input 変数やモデルを使用するまたは使用しないリクエストボディを使用することが必要な場合があります。さらに、パラメータとペイロードまたはペイロードのサブセクションを取得することもできます。以下の 3 つ例は、これを行う方法を示しています。

次の例では、マッピングテンプレートを使用してペイロードのサブセクションを取得します。この例では、入力パラメータ name と、さらに POST 本文全体を取得します。

```
{
    "name" : "$input.params('name')",
    "body" : $input.json('$')
}
```

クエリ文字列パラメータ name=Bella&type=dog と次の本文を含むリクエストの場合:

```
{
    "Price" : "249.99",
    "Age": "6"
}
```

このマッピングテンプレートの出力は、次のようになります。

```
{
    "name" : "Bella",
    "body" : {"Price":"249.99","Age":"6"}
}
```

JSON の入力に JavaScript で解析できない文字がエスケープされずに含まれている場合、API Gateway は 400 レスポンスを返すことがあります。JSON の入力を正しく解析できるようにするには、$util.escapeJavaScript($input.json('$')) を適用します。

前の例に $util.escapeJavaScript($input.json('$')) を適用した結果は次のとおりです。

```
{
    "name" : "$input.params('name')",
```

```
    "body" : $util.escapeJavaScript($input.json('$'))
}
```

この場合、このマッピングテンプレートの出力は、次のようになります。

```
{
    "name" : "Bella",
    "body": {\"Price\":\"249.99\",\"Age\":\"6\"}
}
```

JSONPath 式の例

次の例に、JSONPath 式を `json()` メソッドに渡す方法を示します。さらに、ピリオド (`.`) を使用してプロパティを指定することで、リクエスト本文オブジェクトのサブセクションを読み取ることもできます。

```
{
    "name" : "$input.params('name')",
    "body" : $input.json('$.Age')
}
```

クエリ文字列パラメータ `name=Bella&type=dog` と次の本文を含むリクエストの場合:

```
{
    "Price" : "249.99",
    "Age": "6"
}
```

このマッピングテンプレートの出力は、次のようになります。

```
{
    "name" : "Bella",
    "body" : "6"
}
```

メソッドリクエストのペイロードに JavaScript で解析できない文字がエスケープされずに含まれている場合、API Gateway は `400` レスポンスを返すことがあります。JSON の入力を正しく解析できるようにするには、`$util.escapeJavaScript()` を適用します。

前の例に `$util.escapeJavaScript($input.json('$.Age'))` を適用した結果は次のとおりです。

```
{
    "name" : "$input.params('name')",
    "body" : "$util.escapeJavaScript($input.json('$.Age'))"
}
```

この場合、このマッピングテンプレートの出力は、次のようになります。

```
{
    "name" : "Bella",
    "body": "\"6\""
}
```

リクエストとレスポンスの例

次の例では、パス /things/{id} でリソースに $input.params()、$input.path()、$input.json() を使用しています。

```
{
    "id" : "$input.params('id')",
    "count" : "$input.path('$.things').size()",
    "things" : $input.json('$.things')"
}
```

パスパラメータ 123 と次の本文を含むリクエストの場合:

```
{
      "things": {
            "1": {},
            "2": {},
            "3": {}
      }
}
```

このマッピングテンプレートの出力は、次のようになります。

```
{"id":"123","count":"3","things":{"1":{},"2":{},"3":{}}}
```

メソッドリクエストのペイロードに JavaScript で解析できない文字がエスケープされずに含まれている場合、API Gateway は 400 レスポンスを返すことがあります。JSON の入力を正しく解析できるようにするには、$util.escapeJavaScript() を適用します。

前の例に $util.escapeJavaScript($input.json('$.things')) を適用した結果は次のとおりです。

```
{
     "id" : "$input.params('id')",
     "count" : "$input.path('$.things').size()",
     "things" : "$util.escapeJavaScript($input.json('$.things'))"
}
```

このマッピングテンプレートの出力は、次のようになります。

```
{"id":"123","count":"3","things":"{\"1\":{},\"2\":{},\"3\":{}}"}
```

マッピングのその他の例については、マッピングテンプレートについて を参照してください。

$stageVariables

ステージ変数は、パラメータマッピングおよびマッピングテンプレートで使用できます。また、メソッド統合で使用される ARN および URL のプレースホルダーとして使用できます。詳細については、「the section called “ステージ変数のセットアップ”」を参照してください。

構文	説明
$stageVariables. *<variable_name>* , $stageVariables[' *<variable_name>* '], または ${stageVariables[' *<variable_name>* ']}	*<variable_name>* はステージ変数名を表します。

$util 変数

$util 変数には、マッピングテンプレートで使用するための効用関数が含まれます。

> **Note**
>
> 別に指定されていない限り、デフォルトの文字は UTF-8 に設定されます。

関数	説明
`$util.escapeJavaScript()`	JavaScript 文字列ルールを使用して文字列内の文字をエスケープします。 **Note** この関数は、通常の一重引用符 (') をエスケープした一重引用符 (\') に変換します。ただし、エスケープした一重引用符は JSON で有効ではありません。したがって、この関数からの出力を JSON のプロパティで使用する場合、エスケープした一重引用符 (\') を通常の一重引用符 (') に戻す必要があります。これを次の例で示します: `"input" : "$util.escapeJavaScript( data).replaceAll("\\'",""')"`

`$util.parseJson()`	"stringified" JSON を受け取り、結果のオブジェクト表現を返します。この関数の結果を使用して、Apache Velocity Template Language (VTL) でペイロード要素にネイティブにアクセスしてこれらの要素を操作できるようになります。たとえば、次のペイロードがあるとします。
	
	`{"errorMessage":"{\"key1\":\"var1\", \"key2\":{\"arr\":[1,2,3]}}"}`
	
	さらに、次のマッピングテンプレートを使用するとします。
	
	`#set ($errorMessageObj = $util.parseJson($input.path('$.errorMessage')))`

関数	説明

	<pre>{
	  "errorMessageObjKey2ArrVal" :
	$errorMessageObj.key2.arr[0]
	}</pre>
	この場合、次の出力が返されます。
	<pre>{
	  "errorMessageObjKey2ArrVal" : 1
	}</pre>

`$util.urlEncode()`	文字列を「application/x-www-form-urlencoded」形式に変換します。
`$util.urlDecode()`	「application/x-www-form-urlencoded」文字列をデコードします。
`$util.base64Encode()`	データを base64 エンコードされた文字列にエンコードします。
`$util.base64Decode()`	base64 エンコードされた文字列からデータをデコードします。

API Gateway でのゲートウェイレスポンス

ゲートウェイレスポンスは、API Gateway によって定義されたレスポンスタイプで識別されます。レスポンスは、HTTP ステータスコード、パラメータマッピングで指定される追加のヘッダーのセット、および VTL 以外のマッピングテンプレートで生成されるペイロードで構成されます。

API Gateway REST API では、ゲートウェイレスポンスは GatewayResponse によって表されます。OpenAPI では、`GatewayResponse` インスタンスは x-amazon-apigateway-gateway-responses.gatewayResponse 拡張子で記述されます。

ゲートウェイレスポンスを有効にするには、API レベルでサポートされているレスポンスタイプのゲートウェイレスポンスを設定します。API Gateway からこのタイプのレスポンスが返されるたびに、ゲートウェイレスポンスに定義されているヘッダーマッピングとペイロードマッピングテンプレートに適用されてマッピングされた結果が API 発信者に返されます。

次のセクションでは、API Gateway コンソールと API Gateway REST API を使用してゲートウェイレスポンスを設定する方法を示します。

エラーレスポンスをカスタマイズするためのゲートウェイレスポンスのセットアップ

API Gateway は、受信リクエストを処理できない場合、統合バックエンドにリクエストを転送せずにクライアントにエラーレスポンスを返します。デフォルトでは、エラーレスポンスにエラーを説明する短いメッセージが含まれます。たとえば、未定義の API リソースに対してオペレーションを呼び出そうとすると、`{ "message": "Missing Authentication Token" }` というメッセージが含まれたエラーレスポンスが返されます。API Gateway に慣れていないユーザーには、メッセージの意味がわかりにくい場合があります。

一部のエラーレスポンスについては、API デベロッパーがカスタマイズして異なる形式で返すことが API Gateway で許可されています。たとえば、`Missing Authentication Token` の場合は、次の例に示すように、元のレスポンスペイロードにヒントを追加し、考えられる原因を説明できます: `{"message":"Missing Authentication Token", "hint":"The HTTP method or resources may not be supported."}`。

API が外部交換と AWS クラウドの間を仲介する場合は、統合リクエストまたは統合レスポンスの VTL マッピングテンプレートを使用して、ペイロードを 1 つの形式から別の形式にマッピングします。ただし、VTL マッピングテンプレートはレスポンスが正常に返される有効なリクエストに対してのみ使用できます。

無効なリクエストに対しては、API Gateway は統合を完全にバイパスしてエラーレスポンスを返します。エラーレスポンスを交換に準拠した形式にするには、カスタマイズを使用する必要があります。カスタマイズは、単純な変数の置換のみをサポートする VTL 以外のマッピングテンプレートでレンダリングされます。

API Gateway で生成されたエラーレスポンスを API Gateway で生成される任意のレスポンスに一般化したものは、ゲートウェイレスポンスと呼ばれます。これにより、API Gateway で生成されたレスポンスは統合レスポンスから区別されます。ゲートウェイレスポンスのマッピングテンプレートは、`$context` 変数の値と `$stageVariables` プロパティの値にアクセスできます。また、`method.request.`*`param-position`*`.`*`param-name`* 形式のメソッドリクエストのパラメータにもアクセスできます。

`$context` 変数の詳細については、「データモデル、オーソライザー、マッピングテンプレート、および CloudWatch アクセスログ記録用の `$context` 変数」を参照してください。の詳細については、「`$stageVariables`」を参照してください。`$stageVariables`。メソッドリクエストパラメータの詳細については、「the section called “`$input` 変数”」を参照してください。

トピック

- API Gateway コンソールを使用して REST API のゲートウェイレスポンスをセットアップする
- API Gateway REST API を使用してゲートウェイレスポンスを設定する
- OpenAPI でゲートウェイレスポンスのカスタマイズを設定する
- ゲートウェイレスポンスのタイプ

API Gateway コンソールを使用して REST API のゲートウェイレスポンスをセットアップする

API Gateway コンソールを使用してゲートウェイレスポンスをカスタマイズするには

1. https://console.aws.amazon.com/apigateway で API Gateway コンソールにサインインします。
2. REST API を選択します。
3. メインナビゲーションペインで、[ゲートウェイレスポンス] を選択します。
4. レスポンスタイプを選択し、[編集] を選択します。このチュートリアルでは、[認証トークンが見つかりません] を例として使用します。
5. API Gateway で生成された [ステータスコード] を変更し、API の要件を満たす別のステータスコードを返すことができます。この例では、カスタマイズにより、ステータスコードがデフォルト値 (403) から 404 に変更されます。これは、見つからないと見なすことができるサポートされていないリソースや無効なリソースをクライアントが呼び出したときに、このエラーメッセージが発生するためです。
6. カスタムヘッダーを返すには、[レスポンスヘッダー] で [ヘッダーの追加] を選択します。例として、以下のカスタムヘッダーを追加します。

```
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
```

前述のヘッダーマッピングで、静的ドメイン名 (`'a.b.c'`) は `Allow-Control-Allow-Origin` ヘッダーにマッピングされて、CORS から API へのアクセスが許可されます。`x-amzn-RequestId` パス変数はレスポンスの `request-id` にマッピングされます。受信リクエストの `petId` パス変数はレスポンスの `request-path` ヘッダーにマッピングされます。元のリクエストの `q` クエリパラメータはレスポンスの `request-query` ヘッダーにマッピングされます。

7. [レスポンステンプレート] で、[コンテンツタイプ] を application/json にしたまま、[テンプレートの本文] エディタに次の本文マッピングテンプレートを入力します。

```
{
     "message":"$context.error.messageString",
     "type": "$context.error.responseType",
     "statusCode": "'404'",
     "stage": "$context.stage",
     "resourcePath": "$context.resourcePath",
     "stageVariables.a": "$stageVariables.a"
}
```

   この例では、$context プロパティと $stageVariables プロパティを、ゲートウェイレスポンス本文のプロパティにマッピングする方法を示しています。

8. [Save changes] (変更の保存) をクリックします。
9. 新規または既存のステージに API をデプロイします。

該当する API メソッドの呼び出し URL は https://*o81lxisefl*.execute-api.us-east-1.amazonaws.com/custErr/pets/{petId} であると仮定して、次の CURL コマンドを呼び出してゲートウェイレスポンスをテストします。

```
curl -v -H 'x-amzn-RequestId:123344566' https://o81lxisefl.execute-api.us-
east-1.amazonaws.com/custErr/pets/5/type?q=1
```

追加のクエリ文字列パラメーター q=1 は API と互換性がないため、指定されたゲートウェイのレスポンスをトリガーするために、エラーが返されます。次のようなゲートウェイレスポンスが返されます。

```
> 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
}
```

前述の例では、API バックエンドが Pet Store であること、さらに API にステージ変数 a が定義されていることを前提としています。

API Gateway REST API を使用してゲートウェイレスポンスを設定する

API Gateway REST API を使用してゲートウェイレスポンスをカスタマイズする前に、API が作成済みで、その識別子を取得済みであることが必要です。API 識別子を取得するには、restapi:gateway-responses リンクリレーションに従い、その結果を確認します。

API Gateway REST API を使用してゲートウェイレスポンスをカスタマイズするには

1. ゲートウェイレスポンスインスタンス全体を上書きするには、gatewayresponse:put アクションを呼び出します。URL パスパラメータで目的の responseType を指定し、リクエストペイロードに statusCode、responseParameters、および responseTemplates マッピングを指定します。
2. GatewayResponse インスタンスの一部を更新するには、gatewayresponse:update アクションを呼び出します。URL パラメータに responseType を指定し、リクエストペイロードに、たとえば GatewayResponse や responseParameters マッピングなど、必要な個々の responseTemplates プロパティを指定します。

OpenAPI でゲートウェイレスポンスのカスタマイズを設定する

API ルートレベルで x-amazon-apigateway-gateway-responses 拡張子を使用し、OpenAPI でゲートウェイレスポンスをカスタマイズできます。次の OpenAPI の定義は、MISSING_AUTHENTICATION_TOKEN タイプの GatewayResponse をカスタマイズする例を示しています。

```
  "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}"
      }
    }
```

この例では、カスタマイズによってステータスコードをデフォルト (403) から 404 に変更します。また、ゲートウェイレスポンスに 4 つのヘッダーパラメータと、application/json メディアタイプの 1 つの本文マッピングテンプレートを追加します。

ゲートウェイレスポンスのタイプ

API Gateway では、API デベロッパーがカスタマイズできる以下のゲートウェイレスポンスを公開しています。

ゲートウェイレスポンスのタイプ	デフォルトのステータスコード	説明
ACCESS_DENIED	403	認証が失敗した場合のゲートウェイレスポンス—たとえば、カスタムまたは Amazon Cognito オーソライザーによ

ゲートウェイレスポンスのタイプ	デフォルトのステータスコード	説明
		ってアクセスが拒否された場合などが該当します。レスポンスタイプが未指定の場合、このレスポンスはデフォルトで DEFAULT_4XX タイプになります。
API_CONFIGURATION_ERROR	500	API 設定が無効な場合のゲートウェイレスポンス — たとえば、無効なエンドポイントアドレスが送信された場合、バイナリサポートが有効になっているときにバイナリデータに対する Base64 デコーディングが失敗した場合、統合レスポンスマッピングがいずれのテンプレートとも一致せず、デフォルトテンプレートも設定されていない場合などが該当します。レスポンスタイプが未指定の場合、このレスポンスはデフォルトで DEFAULT_5XX タイプになります。
AUTHORIZER_CONFIGURATION_ERROR	500	カスタムまたは Amazon Cognito オーソライザーへの接続が失敗した場合のゲートウェイレスポンス。レスポンスタイプが未指定の場合、このレスポンスはデフォルトで DEFAULT_5XX タイプになります。

ゲートウェイレスポンスのタイプ	デフォルトのステータスコード	説明
AUTHORIZER_FAILURE	500	カスタムまたは Amazon Cognito オーソライザーが発信者の認証に失敗した場合のゲートウェイレスポンス。レスポンスタイプが未指定の場合、このレスポンスはデフォルトで DEFAULT_5XX タイプになります。
BAD_REQUEST_PARAMETERS	400	有効になっているリクエストの検証に基づいてリクエストパラメータを検証できない場合のゲートウェイレスポンス。レスポンスタイプが未指定の場合、このレスポンスはデフォルトで DEFAULT_4XX タイプになります。
BAD_REQUEST_BODY	400	有効になっているリクエストの検証に基づいてリクエストボディを検証できない場合のゲートウェイレスポンス。レスポンスタイプが未指定の場合、このレスポンスはデフォルトで DEFAULT_4XX タイプになります。

ゲートウェイレスポンスのタイプ	デフォルトのステータスコード	説明
DEFAULT_4XX	Null	レスポンスタイプが未指定で、ステータスコードが 4XX のデフォルトのゲートウェイレスポンス。このフォールバックゲートウェイレスポンスのステータスコードを変更すると、他のすべての 4XX レスポンスのステータスコードが新しい値に変更されます。このステータスコードを null にリセットすると、他のすべての 4XX レスポンスのステータスコードが元の値に戻ります。 **Note** AWS WAF カスタムレスポンスは、カスタムゲートウェイレスポンスよりも優先されます。

ゲートウェイレスポンスのタイプ	デフォルトのステータスコード	説明
DEFAULT_5XX	Null	レスポンスタイプが未指定で、ステータスコードが 5XX のデフォルトのゲートウェイレスポンス。このフォールバックゲートウェイレスポンスのステータスコードを変更すると、他のすべての 5XX レスポンスのステータスコードが新しい値に変更されます。このステータスコードを null にリセットすると、他のすべての 5XX レスポンスのステータスコードが元の値に戻ります。
EXPIRED_TOKEN	403	AWS 認証トークンの有効期限が切れた場合のゲートウェイレスポンス。レスポンスタイプが未指定の場合、このレスポンスはデフォルトで DEFAULT_4XX タイプになります。
INTEGRATION_FAILURE	504	統合が失敗した場合のゲートウェイレスポンス。レスポンスタイプが未指定の場合、このレスポンスはデフォルトで DEFAULT_5XX タイプになります。

ゲートウェイレスポンスのタイプ	デフォルトのステータスコード	説明
INTEGRATION_TIMEOUT	504	統合がタイムアウトした場合のゲートウェイレスポンス。レスポンスタイプが未指定の場合、このレスポンスはデフォルトで DEFAULT_5XX タイプになります。
INVALID_API_KEY	403	API キーを必要としているメソッドに対して無効な API キーが送信された場合のゲートウェイレスポンス。レスポンスタイプが未指定の場合、このレスポンスはデフォルトで DEFAULT_4XX タイプになります。
INVALID_SIGNATURE	403	AWS 署名が無効な場合のゲートウェイレスポンス。レスポンスタイプが未指定の場合、このレスポンスはデフォルトで DEFAULT_4XX タイプになります。
MISSING_AUTHENTICATION_TOKEN	403	認証トークンが見つからない場合のゲートウェイレスポンス。サポートされていない API メソッドやリソースをクライアントが呼び出そうとした場合などが該当します。レスポンスタイプが未指定の場合、このレスポンスはデフォルトで DEFAULT_4XX タイプになります。

ゲートウェイレスポンスのタイプ	デフォルトのステータスコード	説明
QUOTA_EXCEEDED	429	使用量プランのクォータが超過した場合のゲートウェイレスポンス。レスポンスタイプが未指定の場合、このレスポンスはデフォルトで DEFAULT_4XX タイプになります。
REQUEST_TOO_LARGE	413	リクエストが大きすぎる場合のゲートウェイレスポンス。レスポンスタイプが未指定の場合、このレスポンスはデフォルトで HTTP content length exceeded 10485760 bytes になります。
RESOURCE_NOT_FOUND	404	API リクエストが認証および認可 (API キー認証および認可を除く) に合格した後で、指定されたリソースを API Gateway で見つけることができない場合のゲートウェイレスポンス。レスポンスタイプが未指定の場合、このレスポンスはデフォルトで DEFAULT_4XX タイプになります。

ゲートウェイレスポンスのタイプ	デフォルトのステータスコード	説明
THROTTLED	429	使用量プランレベル、メソッドレベル、ステージレベル、またはアカウントレベルのスロットリング制限を超えた場合のゲートウェイレスポンス。レスポンスタイプが未指定の場合、このレスポンスはデフォルトで DEFAULT_4XX タイプになります。
UNAUTHORIZED	401	カスタムまたは Amazon Cognito オーソライザーが発信者の認証に失敗した場合のゲートウェイレスポンス。
UNSUPPORTED_MEDIA_TYPE	415	厳格なパススルー動作が有効になっているときに、ペイロードがサポートされていないメディアタイプである場合のゲートウェイレスポンス。レスポンスタイプが未指定の場合、このレスポンスはデフォルトで DEFAULT_4XX タイプになります。

ゲートウェイレスポンスのタイプ	デフォルトのステータスコード	説明
WAF_FILTERED	403	リクエストが AWS WAF によってブロックされた場合のゲートウェイレスポンス。レスポンスタイプが未指定の場合、このレスポンスはデフォルトで DEFAULT_4XX タイプになります。 Note AWS WAF カスタムレスポンスは、カスタムゲートウェイレスポンスよりも優先されます。

REST API リソースの CORS を有効にする

Cross-origin resource sharing (CORS) は、ブラウザで実行されているスクリプトから開始されるクロスオリジン HTTP リクエストを制限するブラウザのセキュリティ機能です。詳細については、「CORS とは」を参照してください。

CORS サポートを有効にするかどうかを決定する

クロスオリジン HTTP リクエストは、以下に対して行われます。

- 別のドメイン (例: example.com から amazondomains.com へ)
- 別のサブドメイン (例: example.com から petstore.example.com へ)
- 別のポート (例: example.com から example.com:10777 へ)
- 別のプロトコル (例: https://example.com から http://example.com へ)

API にアクセスできず、Cross-Origin Request Blocked を含むエラーメッセージが表示される場合は、CORS を有効にする必要がある場合があります。

クロスオリジン HTTP リクエストは、シンプルなリクエスト、およびシンプルではないリクエストの 2 種類に分類できます。

シンプルなリクエストの CORS を有効にする

以下の条件がすべて当てはまる場合、HTTP リクエストはシンプルです。

- GET、HEAD、および POST のリクエストのみを許可する API リソースに対して発行されます。
- それが POST メソッドリクエストの場合、Origin ヘッダーを含める必要があります。
- リクエストのペイロードコンテンツタイプが text/plain、multipart/form-data、または application/x-www-form-urlencoded の場合。
- リクエストにカスタムヘッダーが含まれていません。
- シンプルなリクエストに関する Mozilla CORS のドキュメントに一覧表示されている追加要件。

シンプルなクロスオリジンの POST メソッドリクエストの場合、リソースからのレスポンスにはヘッダー `Access-Control-Allow-Origin: '*'` または `Access-Control-Allow-Origin:'origin'` を含める必要があります。

他のすべてのクロスオリジン HTTP リクエストはシンプルではないリクエストです。

シンプルではないリクエストの CORS を有効にする

API のリソースがシンプルではないリクエストを受け取った場合は、統合タイプに応じて追加の CORS サポートを有効にする必要があります。

非プロキシ統合の CORS を有効にする

これらの統合の場合、CORS プロトコルは、実際のリクエストを送信する前に、ブラウザからサーバーにプリフライトリクエストを送信し、サーバーからの承認 (または認証情報のリクエスト) を待つことを要求します。プリフライトリクエストに適切なレスポンスを送信するように API を設定する必要があります。

プリフライトレスポンスを作成するには

1. モック統合の OPTIONS メソッドを作成します。
2. 以下のレスポンスヘッダーを 200 メソッドレスポンスに追加します。
    - Access-Control-Allow-Headers

- Access-Control-Allow-Methods
- Access-Control-Allow-Origin

3. 統合パススルーの動作を NEVER に設定します。この場合、マッピングされていないコンテンツタイプのメソッドリクエストは、HTTP 415 Unsupported Media Type レスポンスで拒否されます。詳細については、「統合パススルーの動作」を参照してください。
4. レスポンスヘッダーの値を入力します。すべてのオリジン、すべてのメソッド、および共通のヘッダーを許可するには、以下のヘッダー値を使用します。

   - 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: '*'

プリフライトリクエストを作成したら、少なくとも 200 個すべてのレスポンスに対して、すべての CORS 対応メソッドの Access-Control-Allow-Origin: '*' ヘッダーまたは Access-Control-Allow-Origin:*'origin'* ヘッダーを返す必要があります。

AWS Management Consoleを使用して非プロキシ統合の CORS を有効にする

AWS Management Consoleを使用して CORS を有効にすることができます。API Gateway は、OPTIONS メソッドを作成し、Access-Control-Allow-Origin ヘッダーを既存のメソッド統合レスポンスに追加します。これは常に機能するとは限りません。場合によっては、少なくとも 200 個すべてのレスポンスに対して、すべての CORS 対応メソッドの Access-Control-Allow-Origin ヘッダーを返すように統合レスポンスを手動で変更する必要があります。

プロキシ統合の CORS サポートを有効にする

Lambda プロキシ統合または HTTP プロキシ統合の場合、プロキシ統合は統合レスポンスを返さないため、バックエンドが Access-Control-Allow-Origin ヘッダー、Access-Control-Allow-Methods ヘッダー、Access-Control-Allow-Headers ヘッダーを返す必要があります。

以下の Lambda 関数の例は、必要な CORS ヘッダーを返します。

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!')
    }
```

トピック

- API Gateway コンソールを使用してリソースで CORS を有効にする
- API Gateway のインポート API を使用して、リソースで CORS を有効にする
- CORS のテスト

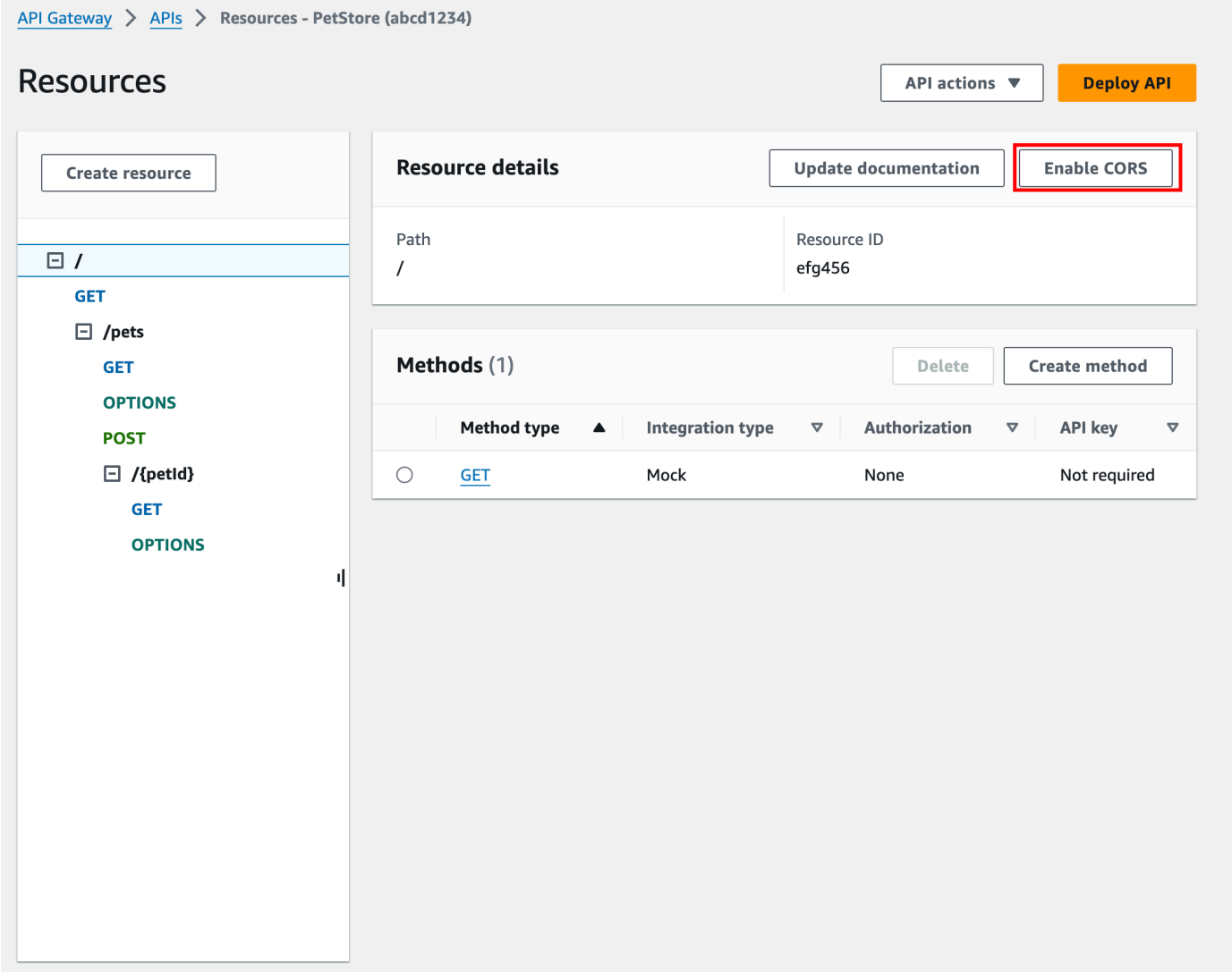
API Gateway コンソールを使用してリソースで CORS を有効にする

API Gateway コンソールを使用して、作成した REST API リソース上の 1 つまたはすべてのメソッドに対する CORS サポートを有効にできます。COR サポートを有効にしたら、統合パススルーの動作を NEVER に設定します。この場合、マッピングされていないコンテンツタイプのメソッドリクエストは、HTTP 415 Unsupported Media Type レスポンスで拒否されます。詳細については、「統合パススルーの動作」を参照してください。

> **Important**
>
> リソースには子リソースを含めることができます。リソースおよびそのメソッドに対する CORS サポートを有効にしても、子リソースおよびそのメソッドに対して再帰的に有効になるわけではありません。

REST API リソースで CORS サポートを有効にするには

1. https://console.aws.amazon.com/apigateway で API Gateway コンソールにサインインします。
2. APIを選択します。
3. [リソース] のリソースを選択します。
4. [リソースの詳細] セクションで、[CORS の有効化] を選択します。



5. [CORS の有効化] フォームで、以下の操作を行います。

   a. (オプション) カスタムゲートウェイレスポンスを作成し、そのレスポンスの CORS サポートを有効にする場合は、ゲートウェイレスポンスを選択します。

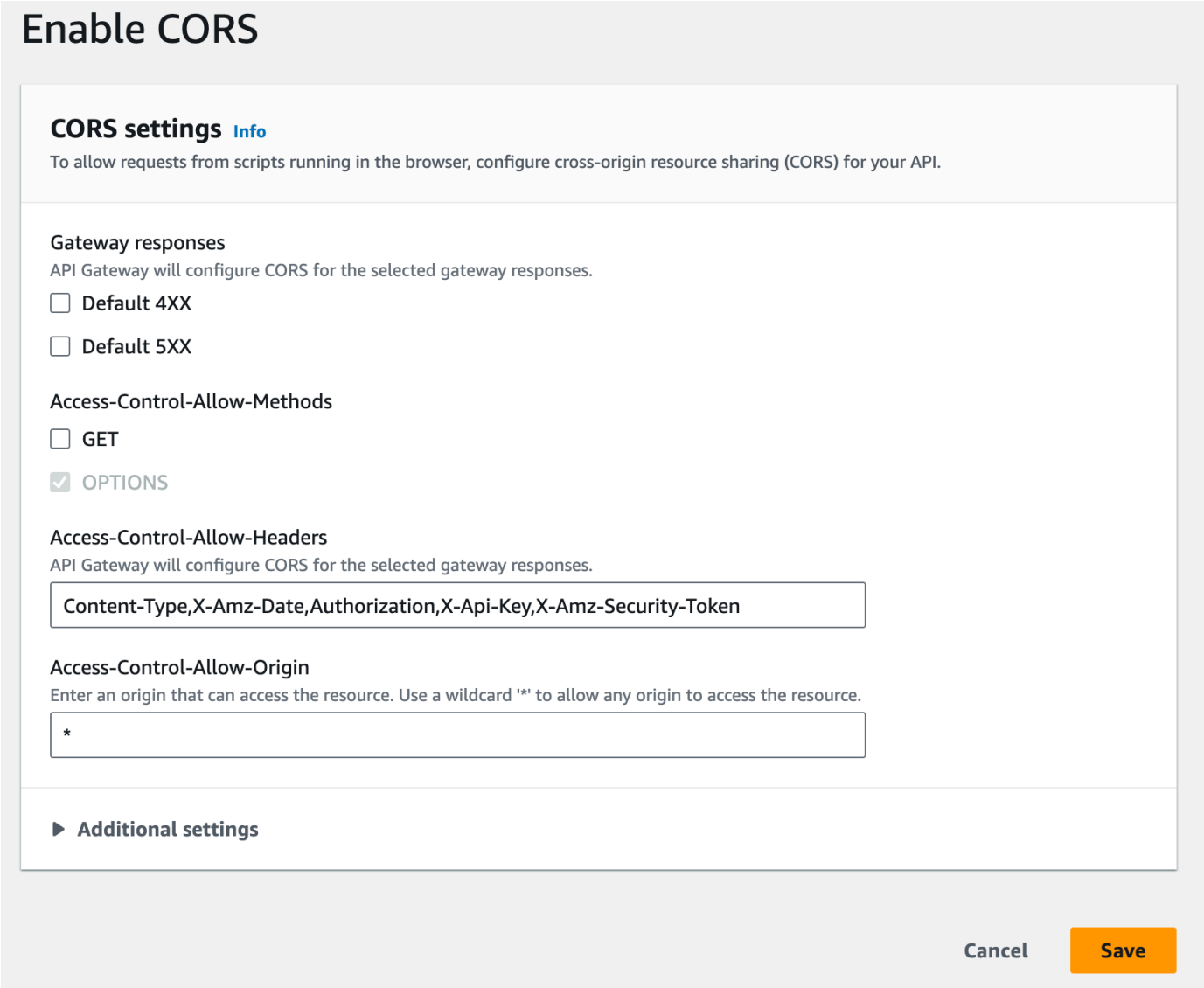
   b. それぞれのメソッドを選択して CORS サポートを有効にします。OPTION メソッドでは CORS が有効になっている必要があります。

      ANY メソッドの CORS サポートを有効にすると、すべてのメソッドで CORS が有効になります。

   c. [Access-Control-Allow-Headers] 入力フィールドに、クライアントがリソースの実際のリクエストで送信する必要があるヘッダーのカンマ区切りリストの静的な文字列を入力します。コンソールで提供されたヘッダーのリスト `'Content-Type,X-Amz-Date,Authorization,X-Api-Key,X-Amz-Security-Token'` を使用するか、独自のヘッダーを指定します。

   d. コンソールで提供された値 `'*'` を [Access-Control-Allow-Origin] ヘッダー値として使用してすべてのオリジンからのアクセスリクエストを許可するか、リソースへのアクセスを許可するオリジンを指定します。

   e. [Save] を選択します。



> **Important**
>
> 上記の手順をプロキシ統合の ANY メソッドに適用すると、適切な CORS ヘッダーは設定されません。代わりに、バックエンドは `Access-Control-Allow-Origin` などの適切な CORS ヘッダーを返す必要があります。

GET メソッドで CORS を有効にすると、すでに追加されていない場合は OPTIONS メソッドがリソースに追加されます。200 メソッドの OPTIONS レスポンスは、プリフライトハンドシェイクを満たすため 3 つの `Access-Control-Allow-*` ヘッダーを返すよう自動的に設定されます。さらに、

実際の (GET) メソッドは、デフォルトでその 200 レスポンスで Access-Control-Allow-Origin ヘッダーを返すように設定されます。他の種類のレスポンスでは、Cross-origin access エラーが発生しないようにする場合、'*' または特定のオリジンを使って Access-Control-Allow-Origin' ヘッダーを返すよう手動で設定する必要があります。

リソースで CORS サポートを有効にした後、新しい設定を有効にするには API をデプロイまたは再デプロイする必要があります。詳細については、「the section called “REST API をデプロイする (コンソール)”」を参照してください。

> **Note**
>
> 手順を実行してもリソースで CORS サポートを有効にできない場合は、CORS 設定をサンプルの API /pets リソースと比較することをお勧めします。サンプル API の作成方法については、「the section called “チュートリアル: サンプルをインポートして REST API を作成する”」を参照してください。

API Gateway のインポート API を使用して、リソースで CORS を有効にする

API Gateway の API のインポートを使用している場合、OpenAPI ファイルを使用して CORS サポートをセットアップできます。最初に、必要なヘッダーを返すリソースの、OPTIONS メソッドを定義する必要があります。

> **Note**
>
> ウェブブラウザは、Access-Control-Allow-Headers ヘッダーおよび Access-Control-Allow-Origin ヘッダーが、CORS リクエストを受け入れる各 API メソッドでセットアップされると想定します。また、一部のブラウザは、同じリソースの OPTIONS メソッドに対して HTTP リクエストを行ってから、同じヘッダーを受け取ることを想定します。

Options メソッドの例

次の例では、モック統合の OPTIONS メソッドを作成します。

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"
```

リソースに OPTIONS メソッドを設定したら、CORS リクエストを受け入れる必要がある同じリソースのその他のメソッドに、必要なヘッダーを追加できます。

1. Access-Control-Allow-Origin と Access-Control-Allow-Origin を応答のタイプに対して宣言します。

   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. x-amazon-apigateway-integration タグで、これらのヘッダーのマッピングを静的な値にセットアップします。

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 の例

次の例では、OPTIONS メソッド、および GET メソッドと HTTP 統合を含む完全な API を作成します。

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"
```

CORS のテスト

API を呼び出し、レスポンスの CORS ヘッダーを確認することで、API の CORS 設定をテストできます。次の curl コマンドは、デプロイされた API に OPTIONS リクエストを送信します。

```
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
```

レスポンスの Access-Control-Allow-Origin、Access-Control-Allow-Headers、および Access-Control-Allow-Methods ヘッダーは、API が CORS をサポートすることを示しています。詳細については、「REST API リソースの CORS を有効にする」を参照してください。

REST API のバイナリメディアタイプの使用

API Gateway では、API リクエストおよびレスポンスにテキストまたはバイナリペイロードがあります。テキストペイロードは UTF-8 でエンコードされた JSON 文字列です。テキストペイロード以外のすべてはバイナリペイロードです。バイナリペイロードの例には、JPEG ファイル、GZip ファイル、XML ファイルなどがあります。バイナリメディアをサポートするために必要な API 設定は、API がプロキシ統合を使用するか非プロキシ統合を使用するかによって異なります。

AWS Lambda プロキシ統合

AWS Lambda プロキシ統合のバイナリペイロードを処理するには、関数のレスポンスを base64 でエンコードする必要があります。また、API の binaryMediaTypes を設定する必要があります。API の `binaryMediaTypes` 設定は、API がバイナリデータとして扱うコンテンツタイプのリストです。バイナリメディアタイプの例には、`image/png` または `application/octet-stream` が含まれます。ワイルドカード文字 (*) を使用して、複数のメディアタイプを対象にすることができます。例えば、`*/*` にはすべてのコンテンツタイプが含まれます。

サンプルコードについては、「the section called “Lambda プロキシ統合からバイナリメディアを返す”」を参照してください。

非プロキシ統合

非プロキシ統合のバイナリペイロードを処理するには、メディアタイプを `RestApi` リソースの binaryMediaTypes リストに追加します。API の `binaryMediaTypes` 設定は、API がバイナリデータとして扱うコンテンツタイプのリストです。Integration リソースと IntegrationResponse リソースに contentHandling プロパティを設定することもできます。`contentHandling` 値は、`CONVERT_TO_BINARY`、`CONVERT_TO_TEXT`、または未定義にすることができます。

`contentHandling` 値の内容と、レスポンスの `Content-Type` ヘッダーと受信リクエストの `Accept` ヘッダーのどちらが `binaryMediaTypes` リストのエントリに一致するかどうかに応じて、API Gateway は raw バイナリバイトを base64 でエンコードされた文字列としてエンコードする、base64 でエンコードされた文字列をその raw バイトに戻す、または変更を加えずに本文を渡すことができます。

API Gateway で API のバイナリペイロードをサポートするには、次のように API を設定する必要があります。

- RestApi リソースの `binaryMediaTypes` リストに必要なメディアタイプを追加します。このプロパティと `contentHandling` プロパティが定義されていない場合、ペイロードは UTF-8 でエンコードされた JSON 設定として処理されます。

- Integration リソースの contentHandling プロパティを指定します。
  - リクエストペイロードを base64 でエンコードされた文字列からそのバイナリ BLOB に変換するには、プロパティを CONVERT_TO_BINARY に設定します。
  - リクエストペイロードをバイナリ BLOB から base64 でエンコードされた文字列に変換するには、プロパティを CONVERT_TO_TEXT に設定します。
  - ペイロードを変更せずにパススルーするには、プロパティを未定義のままにします。バイナリペイロードを変更せずにパススルーするには、Content-Type がいずれかの binaryMediaTypes エントリに一致し、API でパススルー動作が有効になっていることも必要です。
- IntegrationResponse リソースの contentHandling プロパティを設定します。contentHandling プロパティ、クライアントリクエストの Accept ヘッダー、API の binaryMediaTypes の組み合わせによって、API Gateway がコンテンツタイプの変換を処理する方法が決まります。詳細については、「the section called “API Gateway でのコンテンツタイプの変換”」を参照してください。

> **Important**
>
> リクエストの Accept ヘッダーに複数のメディアタイプが含まれている場合、API Gateway は最初の Accept メディアタイプのみ受け入れます。Accept メディアタイプの順序を制御できず、バイナリコンテンツのメディアタイプがリストの先頭にない場合は、API の binaryMediaTypes リストに最初の Accept メディアタイプを追加します。API Gateway は、このリスト内のすべてのコンテンツタイプをバイナリとして処理します。
> たとえば、ブラウザで <img> 要素を使用して JPEG ファイルを送信するため、ブラウザがリクエストで Accept:image/webp,image/*,*/*;q=0.8 を送信することがあります。image/webp を binaryMediaTypes リストに追加することで、エンドポイントは JPEG ファイルをバイナリとして受け取ります。

API Gateway がテキストとバイナリペイロードを処理する方法の詳細については、「API Gateway でのコンテンツタイプの変換」を参照してください。

API Gateway でのコンテンツタイプの変換

API の binaryMediaTypes、クライアントリクエストのヘッダー、統合の contentHandling プロパティの組み合わせによって、API Gateway がペイロードをエンコードする方法が決まります。

次の表に、API Gateway がリクエストの `Content-Type` ヘッダーの特定の設定のリクエストペイロード、RestApi リソースの `binaryMediaTypes` リスト、Integration リソースの `contentHandling` プロパティ値を変換する方法を示します。

API Gateway での API リクエストコンテンツタイプ変換

メソッドリクエストペイロード	リクエスト **Content-Type** ヘッダー	**binaryMediaTypes**	**contentHandling**	統合リクエストペイロード
テキストデータ	任意のデータ型	未定義	未定義	UTF8 でエンコードされた文字列
テキストデータ	任意のデータ型	未定義	CONVERT_TO_BINARY	Base64 でデコードされたバイナリ BLOB
テキストデータ	任意のデータ型	未定義	CONVERT_TO_TEXT	UTF8 でエンコードされた文字列
テキストデータ	テキストデータ型	一致するメディアタイプで設定	未定義	テキストデータ
テキストデータ	テキストデータ型	一致するメディアタイプで設定	CONVERT_TO_BINARY	Base64 でデコードされたバイナリ BLOB
テキストデータ	テキストデータ型	一致するメディアタイプで設定	CONVERT_TO_TEXT	テキストデータ
バイナリデータ	バイナリデータ型	一致するメディアタイプで設定	未定義	バイナリデータ
バイナリデータ	バイナリデータ型	一致するメディアタイプで設定	CONVERT_TO_BINARY	バイナリデータ

メソッドリクエストペイロード	リクエスト **Content-Type** ヘッダー	**binaryMediaTypes**	**contentHandling**	統合リクエストペイロード
バイナリデータ	バイナリデータ型	一致するメディアタイプで設定	CONVERT_TO_TEXT	Base64 でエンコードされた文字列

次の表に、API Gateway がリクエストの Accept ヘッダーの特定の設定のレスポンスペイロード、RestApi リソースの binaryMediaTypes リスト、IntegrationResponse リソースの contentHandling プロパティ値を変換する方法を示します。

> **Important**
>
> リクエストの Accept ヘッダーに複数のメディアタイプが含まれている場合、API Gateway は最初の Accept メディアタイプのみ受け入れます。Accept メディアタイプの順序を制御できず、バイナリコンテンツのメディアタイプがリストの先頭にない場合は、API の binaryMediaTypes リストに最初の Accept メディアタイプを追加します。API Gateway は、このリスト内のすべてのコンテンツタイプをバイナリとして処理します。
> たとえば、ブラウザで <img> 要素を使用して JPEG ファイルを送信するため、ブラウザがリクエストで Accept:image/webp,image/*,*/*;q=0.8 を送信することがあります。image/webp を binaryMediaTypes リストに追加することで、エンドポイントは JPEG ファイルをバイナリとして受け取ります。

API Gateway レスポンスコンテンツタイプの変換

統合レスポンスペイロード	リクエスト **Accept** ヘッダー	**binaryMediaTypes**	**contentHandling**	メソッドレスポンスペイロード
テキストまたはバイナリデータ	テキスト型	未定義	未定義	UTF8 でエンコードされた文字列
テキストまたはバイナリデータ	テキスト型	未定義	CONVERT_TO_BINARY	Base64 でデコードされた BLOB

統合レスポンスペイロード	リクエスト **Accept** ヘッダー	**binaryMediaTypes**	**contentHandling**	メソッドレスポンスペイロード
テキストまたはバイナリデータ	テキスト型	未定義	CONVERT_TO_TEXT	UTF8 でエンコードされた文字列
テキストデータ	テキスト型	一致するメディアタイプで設定	未定義	テキストデータ
テキストデータ	テキスト型	一致するメディアタイプで設定	CONVERT_TO_BINARY	Base64 でデコードされた BLOB
テキストデータ	テキスト型	一致するメディアタイプで設定	CONVERT_TO_TEXT	UTF8 でエンコードされた文字列
テキストデータ	バイナリ型	一致するメディアタイプで設定	未定義	Base64 でデコードされた BLOB
テキストデータ	バイナリ型	一致するメディアタイプで設定	CONVERT_TO_BINARY	Base64 でデコードされた BLOB
テキストデータ	バイナリ型	一致するメディアタイプで設定	CONVERT_TO_TEXT	UTF8 でエンコードされた文字列
バイナリデータ	テキスト型	一致するメディアタイプで設定	未定義	Base64 でエンコードされた文字列
バイナリデータ	テキスト型	一致するメディアタイプで設定	CONVERT_TO_BINARY	バイナリデータ
バイナリデータ	テキスト型	一致するメディアタイプで設定	CONVERT_TO_TEXT	Base64 でエンコードされた文字列

統合レスポンスペイロード	リクエスト **Accept** ヘッダー	**binaryMediaTypes**	**contentHandling**	メソッドレスポンスペイロード
バイナリデータ	バイナリ型	一致するメディアタイプで設定	未定義	バイナリデータ
バイナリデータ	バイナリ型	一致するメディアタイプで設定	CONVERT_TO_BINARY	バイナリデータ
バイナリデータ	バイナリ型	一致するメディアタイプで設定	CONVERT_TO_TEXT	Base64 でエンコードされた文字列

テキストペイロードをバイナリ BLOB に変換するとき、API Gateway はテキストデータが Base64 でエンコードされた文字列であるとみなし、バイナリデータを Base64 でデコードされた BLOB として出力します。変換に失敗した場合、API 設定エラーを示す 500 レスポンスを返します。そのような変換のマッピングテンプレートは提供しませんが、API でパススルー動作を有効にする必要があります。

バイナリペイロードをテキスト文字列に変換した場合、API Gateway は常にバイナリデータに Base64 エンコードを適用します。そのようなペイロードのマッピングテンプレートを定義できますが、サンプルマッピングテンプレートの次の抜粋に示すように、マッピングテンプレート内の Base64 でエンコードされた文字列には `$input.body` を通じてのみアクセスできます。

```
{
    "data": "$input.body"
}
```

バイナリペイロードが変更されずに渡されるようにするには、API でパススルー動作を有効にする必要があります。

API Gateway コンソールを使用したバイナリサポートの有効化

このセクションでは、API Gateway コンソールを使用してバイナリサポートを有効にする方法ついて説明します。例として、Amazon S3 と統合された API を使用します。サポートされるメディアタイプを設定するタスクと、ペイロードの処理方法を指定する方法に焦点を当てます。Amazon S3 と

統合された API を作成する方法の詳細については、「チュートリアル: API Gateway で REST API を Amazon S3 のプロキシとして作成する」を参照してください。

API Gateway コンソールを使用してバイナリサポートを有効にするには

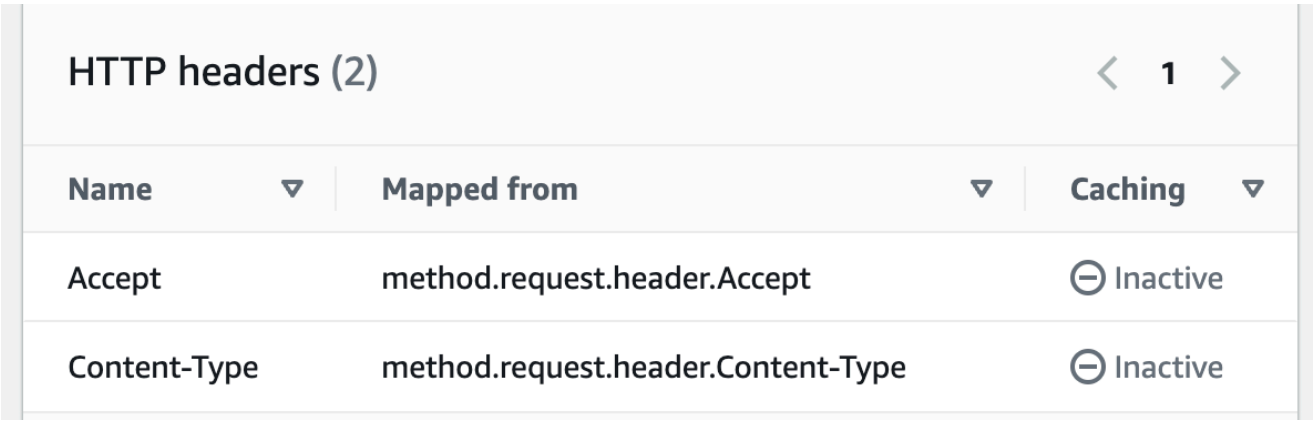
1. API のバイナリメディアタイプの設定
   a. 新しい API を作成するか、既存の API を選択します。この例では、API に `FileMan` という名前を付けます。
   b. プライマリナビゲーションパネルで選択した API で、[API の設定] を選択します。
   c. [API の設定] ペインで、[バイナリメディアタイプ] セクションの [メディアタイプを管理] を選択します。
   d. [バイナリメディアタイプを追加] を選択します。
   e. 必要なメディアタイプ (例: **`image/png`**) を入力テキストフィールドに入力します。必要に応じて、このステップを繰り返して他のメディアタイプを追加します。すべてのバイナリメディアタイプをサポートするには、`*/*` を指定します。
   f. [Save changes] (変更の保存) をクリックします。
2. API メソッドのメッセージペイロードを処理する方法を設定します。
   a. API で新しいリソースを作成するか、既存のリソースを選択します。この例では、`/{folder}/{item}` リソースを使用します。
   b. リソースで新しいメソッドを作成するか、既存のメソッドを選択します。例として、Amazon S3 で `GET /{folder}/{item}` アクションと統合された `Object GET` メソッドを使用します。
   c. [コンテンツの処理] で、オプションを選択します。

クライアントとバックエンドが同じバイナリ形式を受け入れるときに本文を変換しない場合は、[Passthrough (パススルー)] を選択します。バイナリリクエストペイロードを JSON プロパティとして渡すことがバックエンドで要求される場合などに、バイナリ本文を Base64 でエンコードされた文字列に変換するには、[テキストに変換] を選択します。また、クライアントが Base64 でエンコードされた文字列を送信してバックエンドが元のバイナリ形式を要求する場合や、エンドポイントが Base64 でエンコードされた文字列を返してクライアントがバイナリ出力のみ受け入れる場合は、[バイナリに変換] を選択します。

d. [リクエスト本文のパススルー] で、[テンプレートが定義されていない場合 (推奨)] を選択し、リクエスト本文に対するパススルー動作を有効にします。

   [なし] を選択することもできます。つまり、API はマッピングテンプレートを持たない Content-Type のデータを拒否します。

e. 統合リクエストで、受信リクエストの `Accept` ヘッダーを保持します。これは、`contentHandling` を `passthrough` に設定し、ランタイムにその設定を上書きする場合に行う必要があります。



f. テキストに変換する場合は、Base64 でエンコードされたバイナリデータを必要な形式にするマッピングテンプレートを定義します。

マッピングテンプレートをテキストに変換する例は次のとおりです。

```
{
  "operation": "thumbnail",
  "base64Image": "$input.body"
}
```

このマッピングテンプレートの形式は、入力のエンドポイント要件によって異なります。

g. [Save] を選択します。

API Gateway REST API を使用したバイナリサポートの有効化

次のタスクは、API Gateway REST API コールを使用してバイナリサポートを有効にする方法を示します。

トピック

- サポートされるバイナリメディアタイプの API への追加と更新
- リクエストペイロード変換の設定
- レスポンスペイロード変換の設定
- バイナリデータをテキストデータに変換する
- テキストデータをバイナリペイロードに変換する
- バイナリペイロードを渡す

サポートされるバイナリメディアタイプの API への追加と更新

API Gateway が新しいバイナリメディアタイプをサポートするようにするには、RestApi リソースの binaryMediaTypes リストにバイナリメディアタイプを追加する必要があります。たとえば、API Gateway が JPEG イメージを処理するようにするには、PATCH リクエストを RestApi リソースに送信します。

```
PATCH /restapis/<restapi_id>

{
  "patchOperations" : [ {
    "op" : "add",
    "path" : "/binaryMediaTypes/image~1jpeg"
  }
 ]
}
```

image/jpeg プロパティ値の一部となっている path の MIME タイプの指定は、image~1jpeg としてエスケープされます。

サポートされるバイナリメディアタイプを更新するには、binaryMediaTypes リソースの RestApi リストにあるメディアタイプを置き換えるか、削除します。たとえば、バイナリサポートを JPEG ファイルから raw バイトに変更するには、次のように PATCH リクエストを RestApi リソースに送信します。

```
PATCH /restapis/<restapi_id>

{
  "patchOperations" : [{
    "op" : "replace",
    "path" : "/binaryMediaTypes/image~1jpeg",
    "value" : "application/octet-stream"
  },
  {
    "op" : "remove",
    "path" : "/binaryMediaTypes/image~1jpeg"
  }]
}
```

リクエストペイロード変換の設定

エンドポイントにバイナリ入力が必要な場合、contentHandling リソースの Integration プロパティを CONVERT_TO_BINARY に設定します。これを行うには、次のように PATCH リクエストを送信します。

```
PATCH /restapis/<restapi_id>/resources/<resource_id>/methods/<http_method>/integration

{
  "patchOperations" : [ {
    "op" : "replace",
    "path" : "/contentHandling",
    "value" : "CONVERT_TO_BINARY"
  }]
}
```

レスポンスペイロード変換の設定

クライアントが、エンドポイントから返された Base64 でエンコードされたペイロードの代わりにバイナリ BLOB として結果を受け入れる場合、IntegrationResponse リソースの contentHandling プロパティを CONVERT_TO_BINARY に設定します。これを行うには、次のように PATCH リクエストを送信します。

```
PATCH /restapis/<restapi_id>/resources/<resource_id>/methods/<http_method>/integration/
responses/<status_code>

{
  "patchOperations" : [ {
    "op" : "replace",
    "path" : "/contentHandling",
    "value" : "CONVERT_TO_BINARY"
  }]
}
```

バイナリデータをテキストデータに変換する

バイナリデータを、API Gateway を通じて AWS Lambda または Kinesis への入力の JSON プロパティとして送信するには、以下の操作を実行します。

1. application/octet-stream の新しいバイナリメディアタイプを API の binaryMediaTypes リストに追加することで、API のバイナリペイロードサポートを有効にします。

```
PATCH /restapis/<restapi_id>

{
  "patchOperations" : [ {
    "op" : "add",
    "path" : "/binaryMediaTypes/application~1octet-stream"
  }
 ]
}
```

2. Integration リソースの contentHandling プロパティで CONVERT_TO_TEXT を設定し、バイナリデータの Base64 でエンコードされた文字列を JSON プロパティに割り当てるマッピングテンプレートを提供します。次の例では、JSON プロパティが body であり、$input.body には Base64 でエンコードされた文字列が保持されています。

```
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\"}"
    }
  ]
}
```

テキストデータをバイナリペイロードに変換する

Lambda 関数がイメージファイルを base64 でエンコードされた文字列として返すとします。API Gateway を通じてこのバイナリ出力をクライアントに渡すには、以下の操作を実行します。

1. binaryMediaTypes のバイナリメディアを追加することで API の application/octet-stream リストを更新します (まだリストにない場合)。

```
PATCH /restapis/<restapi_id>

{
  "patchOperations" : [ {
    "op" : "add",
    "path" : "/binaryMediaTypes/application~1octet-stream",
  }]
}
```

2. contentHandling リソースの Integration プロパティを CONVERT_TO_BINARY に設定します。マッピングテンプレートを定義しないでください。マッピングテンプレートを定義しない場合、API Gateway はパススルーテンプレートを呼び出して、Base64 でデコードされたバイナリ BLOB をイメージファイルとしてクライアントに返します。

```
PATCH /restapis/<restapi_id>/resources/<resource_id>/methods/<http_method>/
integration/responses/<status_code>

{
  "patchOperations" : [
    {
      "op" : "replace",
      "path" : "/contentHandling",
      "value" : "CONVERT_TO_BINARY"
    }
  ]
}
```

バイナリペイロードを渡す

API Gateway を使用して Amazon S3 バケットにイメージを保存するには、以下の操作を実行します。

1. binaryMediaTypes のバイナリメディアを追加することで API の application/octet-stream リストを更新します (まだリストにない場合)。

```
PATCH /restapis/<restapi_id>

{
  "patchOperations" : [ {
    "op" : "add",
```

```
    "path" : "/binaryMediaTypes/application~1octet-stream"
  }
 ]
}
```

2. contentHandling リソースの Integration プロパティで CONVERT_TO_BINARY を設定します。マッピングテンプレートを定義せずに、WHEN_NO_MATCH を passthroughBehavior プロパティとして設定します。これにより、API Gateway はパススルーテンプレートを呼び出すことができます。

```
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"
    }
  ]
}
```

コンテンツエンコードのインポートとエクスポート

RestApi の binaryMediaTypes リストをインポートするには、API の OpenAPI 定義ファイルへの次の API Gateway 拡張を使用します。この拡張は、API 設定のエクスポートにも使用します。

- x-amazon-apigateway-binary-media-types のプロパティ

Integration または IntegrationResponse リソースで contentHandling プロパティ値をインポートおよびエクスポートするには、OpenAPI 定義への次の API Gateway 拡張を使用します。

- x-amazon-apigateway-integration オブジェクト
- x-amazon-apigateway-integration.response オブジェクト

Lambda プロキシ統合からバイナリメディアを返す

AWS Lambda プロキシ統合からバイナリメディアを返すには、Lambda 関数からのレスポンスを base64 でエンコードします。また、API のバイナリメディアタイプを設定する必要があります。ペイロードサイズの上限は 10 MB です。

> **Note**
>
> この統合例でウェブブラウザを使用して API を呼び出すには、API のバイナリメディアタイプを */* に設定します 。API Gateway は、クライアントからの最初の Accept ヘッダーを使用して、レスポンスがバイナリメディアを返すかどうかを判断します。ブラウザからのリクエストなど、Accept ヘッダー値の順序を制御できない場合に、バイナリメディアを返すには、API のバイナリメディアタイプを */* (すべてのコンテンツタイプ) に設定します。

以下のサンプルの Lambda 関数は、Amazon S3 からのバイナリイメージまたはテキストをクライアントに返すことができます。関数のレスポンスには、返されるデータのタイプをクライアントに示す Content-Type ヘッダーが含まれています。この関数は、返すデータのタイプに応じて、レスポンスで isBase64Encoded プロパティを条件付きで設定します。

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>",
        }
```

バイナリメディアタイプの詳細については、「REST API のバイナリメディアタイプの使用」を参照してください。

API Gateway API を介して Amazon S3 のバイナリファイルにアクセスする

次の例は、Amazon S3 でのイメージのアクセスに使用される OpenAPI ファイル、Amazon S3 からイメージをダウンロードする方法、イメージを Amazon S3 にアップロードする方法を示しています。

トピック

- Amazon S3 でイメージにアクセスするためのサンプル API の OpenAPI ファイル
- Amazon S3 からイメージをダウンロードする
- Amazon S3 にイメージをアップロードする

Amazon S3 でイメージにアクセスするためのサンプル API の OpenAPI ファイル

次の OpenAPI ファイルは、Amazon S3 からのイメージファイルのダウンロードと Amazon S3 へのイメージファイルのアップロードを示すサンプル API を示しています。この API は、指定されたイメージファイルをダウンロードおよびアップロードするための `GET /s3?key={file-name}` メソッドと `PUT /s3?key={file-name}` メソッドを開示します。`GET` メソッドは、200 OK レスポンスにおいて、提供されたマッピングテンプレートに従って、JSON 出力の一部である Base64 でエンコードされた文字列としてイメージファイルを返します。`PUT` メソッドは、入力として raw バイナリ BLOB を受け取り、200 OK レスポンスを空のペイロードで返します。

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"
    }
  }
}
```

Amazon S3 からイメージをダウンロードする

イメージファイル (image.jpg) を Amazon S3 からバイナリ BLOB としてダウンロードするには

```
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
```

正常な応答は次のようになります。

```
200 OK HTTP/1.1

[raw bytes]
```

Accept ヘッダーが application/octet-stream のバイナリメディアタイプとして設定されており、API のバイナリサポートが有効になっているため、raw バイトが返されます。

または、イメージファイル (image.jpg) を、Amazon S3 から、(JSON プロパティとしてフォーマットされた) Base64 でエンコードされた文字列としてダウンロードするには、次の太字の OpenAPI 定義ブロックに示されているように、200 の統合レスポンスにレスポンステンプレートを追加します。

```
        "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}"
              }
            }
          },
```

イメージファイルをダウンロードするリクエストは、次のようになります。

```
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
```

正常なレスポンスは次のようになります。

```
200 OK HTTP/1.1

{
  "image": "W3JhdyBieXRlc10="
}
```

Amazon S3 にイメージをアップロードする

イメージファイル (`image.jpg`) をバイナリ BLOB として Amazon S3 にアップロードするには

```
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]
```

正常なレスポンスは次のようになります。

```
200 OK HTTP/1.1
```

イメージファイル (image.jpg) を Base64 でエンコードされた文字列として Amazon S3 にアップロードするには

```
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=
```

Content-Type ヘッダー値が application/json に設定されているため、入力ペイロードは Base64 でエンコードされた文字列でなければならない点に注意してください。正常なレスポンスは次のようになります。

```
200 OK HTTP/1.1
```

API Gateway API を使用して Lambda のバイナリファイルにアクセスする

次の OpenAPI の例は、API Gateway API を通じて AWS Lambda のバイナリファイルにアクセスする方法を示しています。この API は、指定したイメージファイルをダウンロードおよびアップロードするための GET /lambda?key={file-name} メソッドと PUT /lambda?key={file-name} メソッドを公開します。GET メソッドは、200 OK レスポンスにおいて、提供されたマッピングテンプレートに従って、JSON 出力の一部である Base64 でエンコードされた文字列としてイメージファイルを返します。PUT メソッドは、入力として raw バイナリ BLOB を受け取り、200 OK レスポンスを空のペイロードで返します。

API が呼び出す Lambda 関数を作成します。この Lambda 関数は、application/json の Content-Type ヘッダーを含む文字列を base64 でエンコードして返す必要があります。

トピック

- Lambda でイメージにアクセスするためのサンプル API の OpenAPI ファイル
- Lambda からイメージをダウンロードする
- Lambda へイメージをアップロードする

Lambda でイメージにアクセスするためのサンプル API の OpenAPI ファイル

次の OpenAPI ファイルは、Lambda からのイメージファイルのダウンロードと Lambda へのイメージファイルのアップロードを示す API 例を示しています。

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"
    }
  }
}
```

Lambda からイメージをダウンロードする

イメージファイル (image.jpg) を Lambda からバイナリ BLOB としてダウンロードするには

```
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
```

正常なレスポンスは次のようになります。

```
200 OK HTTP/1.1

[raw bytes]
```

イメージファイル (image.jpg) を、JSON プロパティとしてフォーマットして base64 でエンコードされた文字列として Lambda からダウンロードするには

```
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
```

正常なレスポンスは次のようになります。

```
200 OK HTTP/1.1

{
  "image": "W3JhdyBieXRlc10="
}
```

Lambda へイメージをアップロードする

イメージファイル (image.jpg) をバイナリ BLOB として Lambda にアップロードするには

```
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]
```

正常なレスポンスは次のようになります。

```
200 OK
```

イメージファイル (image.jpg) を base64 でエンコードされた文字列として Lambda にアップロードするには

```
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=
```

正常なレスポンスは次のようになります。

```
200 OK
```

Amazon API Gateway での REST API の呼び出し

デプロイされた API を呼び出すには、API 実行の `execute-api` コンポーネントサービス (API Gateway と呼ばれます) の URL にクライアントがリクエストを送信します。

REST API のベース URL の形式は以下のとおりです。

```
https://restapi_id.execute-api.region.amazonaws.com/stage_name/
```

`restapi_id` は API ID、*`region`* は AWS リージョン、*`stage_name`* は API デプロイのステージ名です。

> **Important**
>
> API を呼び出す前に、API Gateway でデプロイする必要があります。API をデプロイする手順については、「Amazon API Gateway での REST API のデプロイ」を参照してください。

トピック

- API の呼び出し URL の取得
- API の呼び出し
- API Gateway コンソールを使用して REST API メソッドをテストする
- REST API 用に API Gateway で生成された Java SDK を使用する
- REST API 用に API Gateway で生成された Android SDK を使用する
- REST API 用に API Gateway で生成された JavaScript SDK を使用する

- API Gateway によって生成された Ruby SDK を REST API で使用する
- Objective-C または Swift で REST API 用に API Gateway で生成された iOS SDK を使用する

API の呼び出し URL の取得

API の呼び出し URL を取得するには、コンソール、AWS CLI、またはエクスポートされた OpenAPI 定義を使用できます。

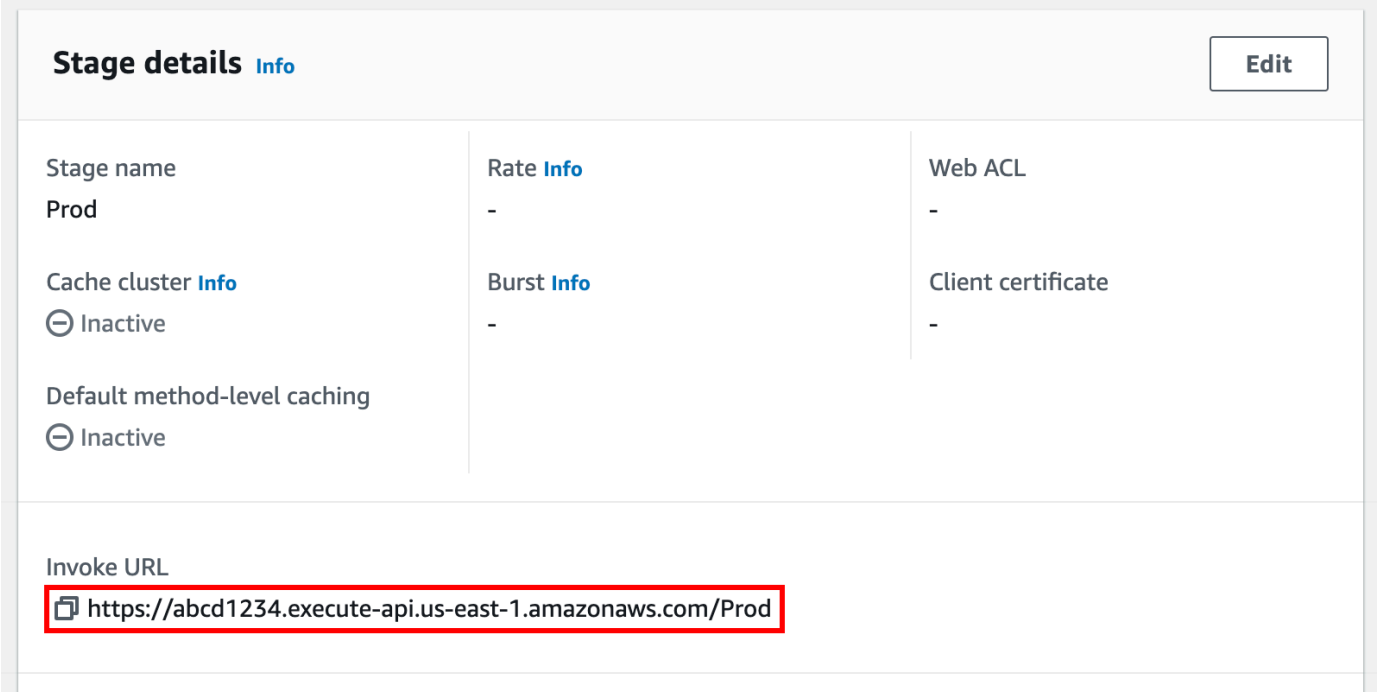
コンソールを使用した API の 呼び出し URL の取得

次の手順では、REST API コンソールで API の呼び出し URL を取得する方法を示します。

REST API コンソールを使用して API の 呼び出し URL を取得するには

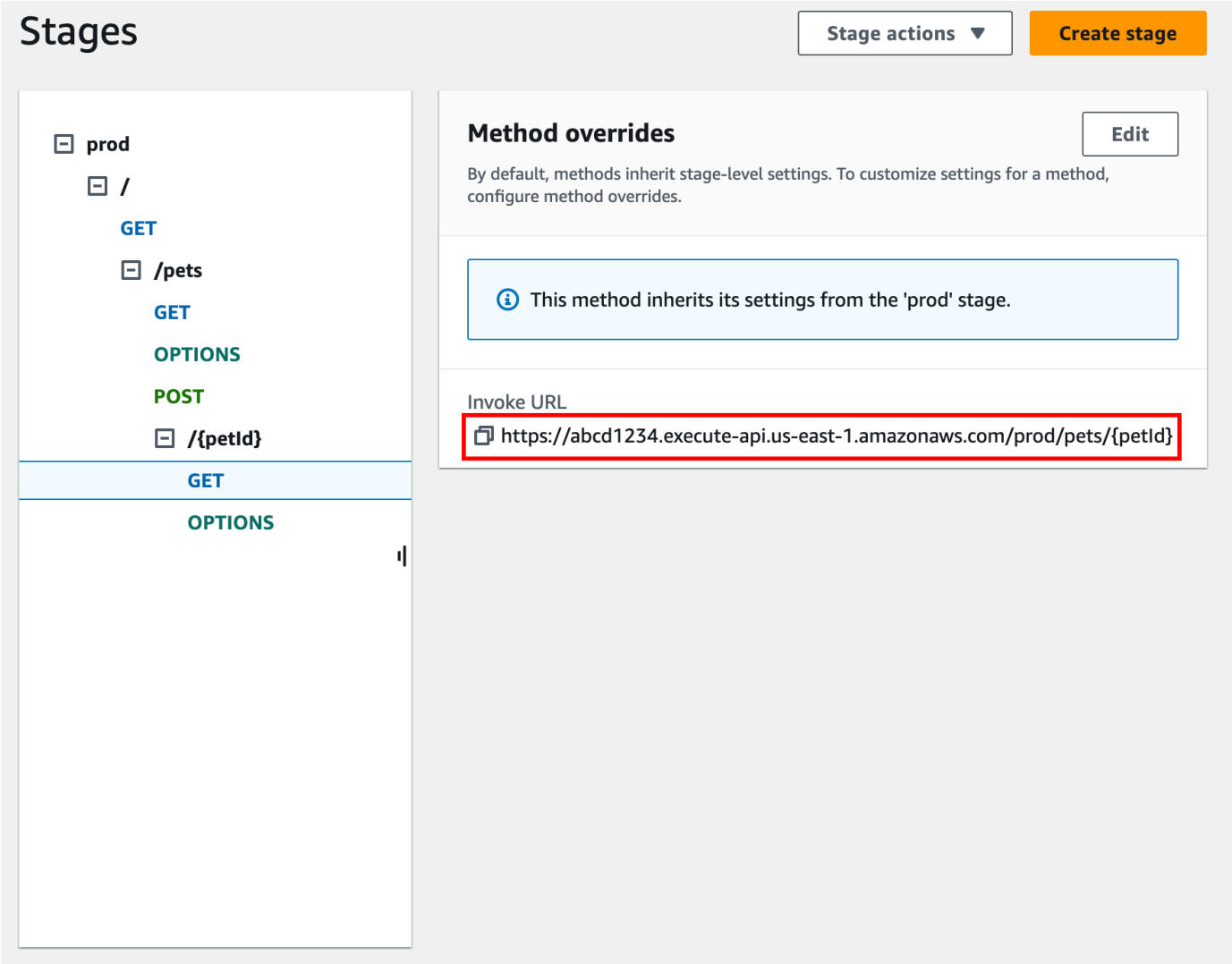
1. https://console.aws.amazon.com/apigateway で API Gateway コンソールにサインインします。
2. デプロイされた API を選択します。
3. メインのナビゲーションペインで、[ステージ] を選択します。
4. [ステージの詳細] で、コピーアイコンを選択して API の呼び出し URL をコピーします。

   この URL は API のルートリソース用です。



5. API で別のリソース用の API の呼び出し URL を取得するには、セカンダリナビゲーションペインでステージを展開し、メソッドを選択します。

6. コピーアイコンを選択して API のリソースレベルの呼び出し URL をコピーします。



AWS CLI を使用した API の呼び出し URL の取得

次の手順では、AWS CLI を使用して API の呼び出し URL を取得する方法を示します。

AWS CLI を使用した API の 呼び出し URL の取得

1. `rest-api-id` を取得するには、次のコマンドを使用します。このコマンドは、リージョンのすべての `rest-api-id` 値を返します。詳細については、「get-rest-apis」を参照してください。

```
aws apigateway get-rest-apis
```

2. 例の `rest-api-id` を独自の `rest-api-id`、例の *`{stage-name}`* を独自の *`{stage-name}`*、*`{region}`* を独自のリージョンに置き換えます。

```
https://{restapi_id}.execute-api.{region}.amazonaws.com/{stage_name}/
```

API のエクスポートされた OpenAPI 定義ファイルを使用した API の呼び出し URL の取得

API のエクスポートされた OpenAPI 定義ファイルの host フィールドと basePath フィールドを組み合わせてルート URL を構築することもできます。API のエクスポート方法の手順については、「the section called “REST API をエクスポートする”」を参照してください。

API の呼び出し

デプロイした API は、ブラウザ、curl、または他のアプリケーション (Postman など) を使用して呼び出すことができます。

さらに、API Gateway コンソールを使用して API コールをテストすることもできます。テストでは API Gateway の TestInvoke 機能を使用します。これにより、API をデプロイする前に API をテストできます。詳細については、「the section called “コンソールを使用した REST API メソッドのテスト”」を参照してください。

> **Note**
>
> 呼び出し URL のクエリ文字列パラメーターの値に %% を含めることはできません。

ウェブブラウザを使用した API の呼び出し

API が匿名アクセスを許可する場合は、任意のウェブブラウザを使用して GET メソッドを呼び出すことができます。ブラウザのアドレスバーに完全な呼び出し URL を入力します。

他のメソッドや認証が必要な呼び出しについては、ペイロードを指定するか、リクエストに署名する必要があります。HTML ページの裏のスクリプトで、または AWS SDK の 1 つを使用して、クライアントのアプリケーションで処理することができます。

curl を使用した API の呼び出し

ターミナルで curl などのツールを使用して API を呼び出すことができます。次の curl コマンドの例では、API の prod ステージの getUsers リソースで GET メソッドを呼び出します。

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"
```

API Gateway コンソールを使用して REST API メソッドをテストする

API Gateway コンソールを使用して REST API メソッドをテストします。

トピック

- 前提条件
- API Gateway コンソールを使用してメソッドをテストする

前提条件

- テストするメソッドの設定を指定する必要があります。「API Gateway の REST API のメソッド」の手順に従います

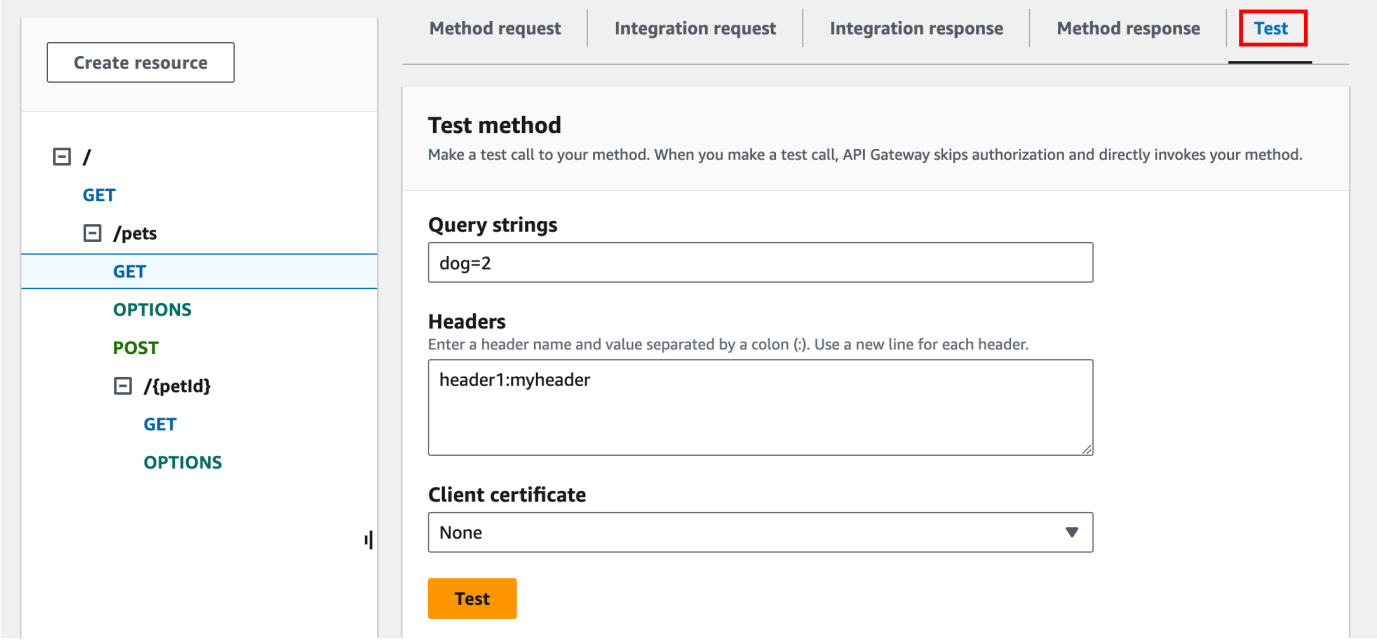
API Gateway コンソールを使用してメソッドをテストする

> **Important**
>
> API Gateway コンソールでメソッドをテストすると、リソースが変更され、元に戻せなくなる場合があります。API Gateway コンソールでメソッドをテストすることは、API Gateway コンソール外でメソッドを呼び出すことと同じです。たとえば、API Gateway コンソールを使用して API リソースを削除するメソッドを呼び出した場合、メソッドの呼び出しが成功すると、API のリソースは削除されます。

メソッドをテストするには

1. https://console.aws.amazon.com/apigateway で API Gateway コンソールにサインインします。
2. REST API を選択します。
3. [リソース] ペインで、テストするメソッドを選択します。

4. [テスト] タブを選択します。タブを表示するには、右矢印ボタンを選択する必要がある場合があります。



表示されたいずれかのボックスに値を入力します ([クエリ文字列]、[ヘッダー]、[リクエスト本文] など)。コンソールには、メソッドリクエストにこれらの値がデフォルトの application/json 形式で含まれています。

指定する必要がある追加オプションについては、API 所有者までお問い合わせください。

5. [テスト] を選択します。次の情報が表示されます。

- [リクエスト]: メソッド用に呼び出されたリソースのパスです。
- [ステータス]: 応答の HTTP ステータスコードです。
- [レイテンシー (ミリ秒)]: 発信者からのリクエストを受信してからレスポンスを返すまでの時間です。
- [レスポンス本文] は HTTP レスポンスの本文です。
- [レスポンスヘッダー] は HTTP レスポンスのヘッダーです。

> **Tip**
>
> マッピングによって、HTTP ステータスコード、応答本文、応答ヘッダーは、Lambda 関数、HTTP プロキシ、または AWS のサービスプロキシから送信されたものと異なる場合があります。

- [ログ] はシミュレートされた Amazon CloudWatch Logs エントリで、このメソッドが API Gateway コンソール外で呼び出された場合は書き込まれています。

> **Note**
>
> CloudWatch Logs エントリはシミュレートされていますが、メソッドの呼び出しの結果は現実のものです。

API Gateway コンソールの使用に加えて、AWS CLI、または API Gateway 用の AWS SDK を使用してメソッドの呼び出しをテストすることもできます。AWS CLI を使用してこれを行う方法については、「test-invoke-method」を参照してください。

REST API 用に API Gateway で生成された Java SDK を使用する

このセクションでは、単純な電卓の API を例として使用し、REST API に対して API Gateway で生成された Java SDK を使用するステップを示します。先に進む前に、「API Gateway で REST API 用 SDK を生成する」のステップを完了する必要があります。

API Gateway で生成した Java SDK をインストールして使用するには

1. ダウンロード済みの API Gateway で生成された .zip ファイルのコンテンツを抽出します。
2. Apache Maven (バージョン 3.5 以降が必要です) をダウンロードしてインストールします。
3. JDK 8 をダウンロードしてインストールします。
4. `JAVA_HOME` 環境変数を設定します。
5. 解凍した SDK フォルダ内の pom.xml ファイルに移動します。このフォルダーは、デフォルトでは `generated-code` です。mvn install コマンドを実行し、コンパイルされたアーティファクトファイルをローカルの Maven リポジトリにインストールします。これにより、コンパイル済み SDK ライブラリを含む `target` フォルダーが作成されます。
6. 空のディレクトリで次のコマンドを入力し、インストールした SDK ライブラリを使用して API を呼び出すためのクライアントプロジェクトスタブを作成します。

```
mvn -B archetype:generate \
    -DarchetypeGroupdId=org.apache.maven.archetypes \
    -DgroupId=examples.aws.apig.simpleCalc.sdk.app \
    -DartifactId=SimpleCalc-sdkClient
```

> **Note**
>
> 上のコマンドで区切り記号の \ は読みやすくするために使用しています。コマンド全体は区切り記号がない単一の行となります。

このコマンドでは、アプリケーションスタブを作成します。アプリケーションスタブには、プロジェクトのルートディレクトリ (上のコマンドの pom.xmlSimpleCalc-sdkClientsrc) の *#####* フォルダが含まれます。当初は、ソースファイルとして src/main/java/*{package-path}*/App.java と src/test/java/*{package-path}*/AppTest.java の 2 つがあります。この例では、*{package-path}* は examples/aws/apig/simpleCalc/sdk/app です。このパッケージパスは DarchetypeGroupdId の値から取得されます。App.java ファイルは、クライアントアプリケーションのテンプレートとして使用できます。また、必要に応じて同じフォルダーに他のファイルを追加できます。AppTest.java ファイルは、アプリケーションのユニットテストテンプレートとして使用できます。また、必要に応じて同じテストフォルダーに他のテストコードファイルを追加できます。

7. 生成された pom.xml ファイルのパッケージ依存関係を以下のように更新し、必要に応じてプロジェクトの groupId、artifactId、version、name の各プロパティを置き換えます。

```
<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**
>
> aws-java-sdk-core の依存アーティファクトの新しいバージョンが上記で指定されたバージョン (1.11.94) と互換性がない場合は、<version> タグを新しいバージョンに更新する必要があります。

8. 次に、SDK の getABOp(GetABOpRequest req)、getApiRoot(GetApiRootRequest req)、postApiRoot(PostApiRootRequest req) の各メソッドを呼び出して、SDK で API を呼び出す方法を示します。これらのメソッドは、GET /{a}/{b}/{op}、GET /?

a={x}&b={y}&op={operator}、POST / の各メソッドに対応します。API リクエストのペイロードはそれぞれ {"a": x, "b": y, "op": "operator"} です。

次のように App.java ファイルを更新します。

```
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());
```

```
    }
}
```

上の例で、SDK クライアントをインスタンス化するために使用している設定はサンプルであり、必ずしも推奨されるベストプラクティスではありません。また、`sdkClient.shutdown()` の呼び出しはオプションです。リソースを解放するタイミングを正確に制御する場合などに役立ちます。

Java SDK を使用して API を呼び出すための基本的なパターンを示しました。必要に応じて手順を拡張し、他の API メソッドを呼び出すことができます。

REST API 用に API Gateway で生成された Android SDK を使用する

このセクションでは、REST API 用に API Gateway で生成された Android SDK を使用するステップを説明します。先に進む前に、「API Gateway で REST API 用 SDK を生成する」のステップを済ませている必要があります。

> **Note**
>
> 生成された SDK は、Android 4.4 以前とは互換性がありません。詳細については、「the section called “重要な注意点”」を参照してください。

API Gateway で生成された Android SDK をインストールして使用するには

1. ダウンロード済みの API Gateway で生成された .zip ファイルのコンテンツを抽出します。
2. Apache Maven をダウンロードしてインストールします (バージョン 3.x を推奨)。
3. JDK 8 をダウンロードしてインストールします。
4. `JAVA_HOME` 環境変数を設定します。
5. mvn install コマンドを実行し、コンパイルされたアーティファクトファイルをローカルの Maven リポジトリにインストールします。これにより、コンパイル済み SDK ライブラリを含む `target` フォルダーが作成されます。
6. `target` フォルダからの SDK ファイル (ファイルの名前は、 `simple-calcsdk-1.0.0.jar` など、SDK の生成時に指定した Artifact ID および Artifact バージョンに由来するものです)

を、target/lib フォルダからのその他すべてのライブラリとともに、プロジェクトの lib フォルダにコピーします。

Andriod Studio を使用している場合は、クライアントアプリケーションモジュールで libs フォルダを作成し、このフォルダ内に必要な .jar ファイルをコピーします。モジュールの gradle ファイルの依存関係セクションに、以下が含まれていることを確認します。

```
    compile fileTree(include: ['*.jar'], dir: 'libs')
    compile fileTree(include: ['*.jar'], dir: 'app/libs')
```

重複した .jar ファイルが宣言されていないことを確認します。

7. ApiClientFactory クラスを使用して、API Gateway で生成された SDK を初期化します。例:

```
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. Amazon Cognito 認証情報プロバイダーを使用して API への呼び出しを認証するには、次の例で示すように API Gateway で生成された SDK を使用して `ApiClientFactory` クラスで AWS 認証情報のセットを渡します。

```
// 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. API Gateway で生成された SDK で API キーを設定するには、次のようなコードを使用します。

```
ApiClientFactory factory = new ApiClientFactory()
  .apiKey("YOUR_API_KEY");
```

REST API 用に API Gateway で生成された JavaScript SDK を使用する

> **Note**
>
> これらの手順は、すでに「API Gateway で REST API 用 SDK を生成する」の手順を完了していることを前提としています。

> **Important**
>
> API に ANY メソッドのみが定義された場合、生成された SDK パッケージに `apigClient.js` ファイルは含まれず、ANY メソッドを自分で定義する必要があります。

REST API に対して API Gateway で生成された JavaScript SDK をインストールし、開始して呼び出すには

1. ダウンロード済みの API Gateway で生成された .zip ファイルのコンテンツを抽出します。
2. API Gateway で生成された SDK で呼び出すすべてのメソッドで Cross-Origin Resource Sharing (CORS) を有効にします。手順については、REST API リソースの CORS を有効にする を参照してください。
3. ウェブページで、以下のスクリプトへの参照を含めます。

```
<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. コードで、API Gateway で生成された SDK を初期化します。次のようなコードを使用します。

```
var apigClient = apigClientFactory.newClient();
```

API Gateway で生成された SDK を AWS 認証情報で初期化するには、次のようなコードを使用します。AWS 認証情報を使用すると、API に対するすべてのリクエストに署名が付与されます。

```
var apigClient = apigClientFactory.newClient({
  accessKey: 'ACCESS_KEY',
  secretKey: 'SECRET_KEY',
});
```

API Gateway で生成された SDK で API キーを使用するには、次のようなコードを使用し、API キーをパラメータとして Factory オブジェクトに渡すことができます。API キーを使用する場合は、x-api-key ヘッダーの一部として指定され、API に対するすべてのリクエストに署名が付与されます。これは、各リクエストに適切な CORS Accept ヘッダーを設定する必要があることを意味します。

```
var apigClient = apigClientFactory.newClient({
  apiKey: 'API_KEY'
});
```

5. 以下のようなコードを使用して API Gateway で API メソッドを呼び出します。各呼び出しから、成功または失敗のコールバックとともに promise が返されます。

```
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.
    });
```

ここで、*methodName* は、メソッドリクエストのリソースパスと HTTP 動詞から作成されます。SimpleCalc API の場合、次の API メソッドの SDK メソッド

```
1. GET /?a=...&b=...&op=...
2. POST /

   { "a": ..., "b": ..., "op": ...}
3. GET /{a}/{b}/{op}
```

対応する SDK メソッドは次のとおりです。

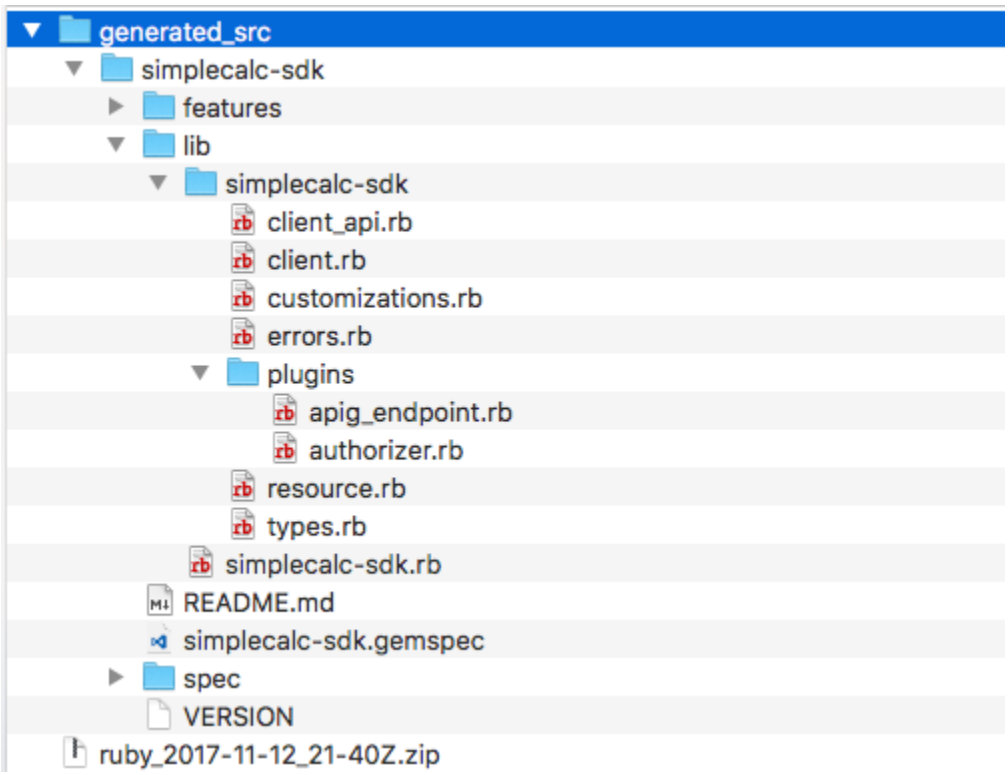
```
1. rootGet(params);      // where params={"a": ..., "b": ..., "op": ...} is
 resolved to the query parameters
2. rootPost(null, body); // where body={"a": ..., "b": ..., "op": ...}
3. aBOpGet(params);      // where params={"a": ..., "b": ..., "op": ...} is
 resolved to the path parameters
```

API Gateway によって生成された Ruby SDK を REST API で使用する

> **Note**
>
> これらの手順は、すでに「API Gateway で REST API 用 SDK を生成する」の手順を完了していることを前提としています。

REST API に対して API Gateway で生成された Ruby SDK をインストールし、開始して呼び出すには

1. ダウンロードした Ruby SDK ファイルを解凍します。生成された SDK ソースは、次のように表示されます。



2. ターミナルウィンドウで次のシェルコマンドを使用して、生成された SDK ソースから Ruby Gem を構築します。

```
# change to /simplecalc-sdk directory
cd simplecalc-sdk

# build the generated gem
gem build simplecalc-sdk.gemspec
```

この後、[simplecalc-sdk-1.0.0.gem] が利用可能になります。

3. gem をインストールします。

```
gem install simplecalc-sdk-1.0.0.gem
```

4. クライアントアプリケーションを作成します。アプリで Ruby SDK クライアントをインスタンス化して初期化します。

```
require 'simplecalc-sdk'
client = SimpleCalc::Client.new(
    http_wire_trace: true,
    retry_limit: 5,
    http_read_timeout: 50
)
```

API に AWS_IAM タイプの許可が設定されている場合、初期化中に accessKey と secretKey を指定して、呼び出し元の AWS 認証情報を含めることができます。

```
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. アプリで SDK を使用して API 呼び出しを行います。

> **Tip**
>
> SDK のメソッド呼び出し規則に精通していない場合は、生成された SDK の client.rb フォルダの lib ファイルを確認できます。このフォルダには、サポートされている各 API メソッド呼び出しのドキュメントが含まれています。

サポートされているオペレーションを検出するには:

```
# to show supported operations:
```

```
puts client.operation_names
```

これにより、GET /?a={.}&b={.}&op={.}、GET /{a}/{b}/{op}、および POST / の API メソッドに対応する次のような表示になります。必要に応じて {a:"…", b:"…", op:"…"} 形式のペイロードを加えます。

```
[:get_api_root, :get_ab_op, :post_api_root]
```

GET /?a=1&b=2&op=+API メソッドを呼び出すには、以下の Ruby SDK メソッドを呼び出します。

```
resp = client.get_api_root({a:"1", b:"2", op:"+"})
```

POST / を使用して、{a: "1", b: "2", "op": "+"} API メソッドを呼び出すには、以下の Ruby SDK メソッドを呼び出します。

```
resp = client.post_api_root(input: {a:"1", b:"2", op:"+"})
```

GET /1/2/+API メソッドを呼び出すには、以下の Ruby SDK メソッドを呼び出します。

```
resp = client.get_ab_op({a:"1", b:"2", op:"+"})
```

SDK メソッド呼び出しが正常に終了すると次のレスポンスを返します。

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

Objective-C または Swift で REST API 用に API Gateway で生成された iOS SDK を使用する

このチュートリアルでは、Objective-C または Swift アプリケーションにおいて、REST API の API Gateway で生成された iOS SDK を使用して基盤となる API を呼び出す方法について説明します。以下のトピックでは、SimpleCalc API を例として使います。

- AWS Mobile SDK の必要なコンポーネントを Xcode プロジェクト内にインストールする方法
- API のメソッドを呼び出す前に API クライアントオブジェクトを作成する方法
- API クライアントオブジェクトで対応する SDK メソッドを介して API のメソッドを呼び出す方法
- SDK の対応するモデルクラスを使用してメソッドの入力を準備し、その結果を解析する方法

トピック

- 生成された iOS SDK (Objective-C) を使用して API を呼び出す
- 生成された iOS SDK (Swift) を使用して API を呼び出す

生成された iOS SDK (Objective-C) を使用して API を呼び出す

次の手順を開始する前に、Objective-C で iOS 用の「API Gateway で REST API 用 SDK を生成する」のステップを実行し、生成された SDK の .zip ファイルをダウンロードする必要があります。

API Gateway で生成された AWS Mobile SDK と iOS SDK を Objective-C プロジェクトでインストールする

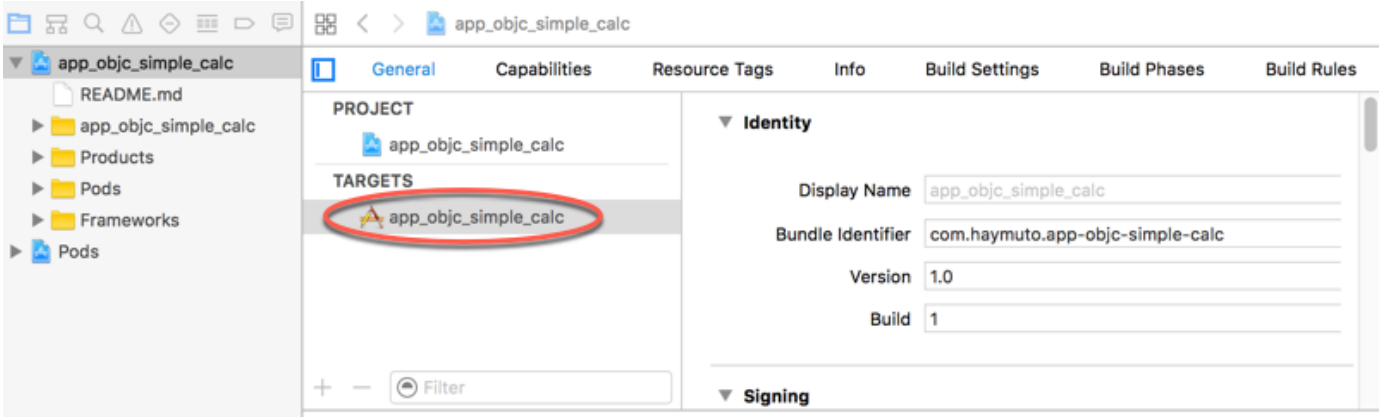
次の手順では、SDK をインストールする方法を説明します。

API Gateway で生成された iOS SDK を Objective-C でインストールして使用するには

1. ダウンロード済みの API Gateway で生成された .zip ファイルのコンテンツを抽出します。SimpleCalc API を使用して、解凍した SDK フォルダの名前を **sdk_objc_simple_calc** などに変更することもできます。この SDK フォルダーには、`README.md` ファイルと `Podfile` ファイルが含まれています。`README.md` ファイルには、SDK をインストールして使用するための手順が記載されています。このチュートリアルでは、この手順について詳しく説明します。インストールでは、CocoaPods を使用して必要な API Gateway ライブラリとその他の依存する AWS Mobile SDK コンポーネントをインポートします。SDK をアプリケーションの XCode プロジェクト内にインポートするには、`Podfile` を更新する必要があります。解凍した SDK

フォルダー内には、API の生成された SDK のソースコードを含む `generated-src` フォルダーもあります。

2. Xcode を起動し、新しい iOS Objective-C プロジェクトを作成します。プロジェクトのターゲットを書き留めておきます。それを `Podfile` に設定する必要があります。



3. CocoaPods を使用して AWS Mobile SDK for iOS を Xcode プロジェクト内にインポートするには、以下の操作を行います。

   a. ターミナルウィンドウで次のコマンドを実行して、CocoaPods をインストールします。

   ```
   sudo gem install cocoapods
   pod setup
   ```

   b. 抽出した SDK フォルダー内にある `Podfile` ファイルを、Xcode プロジェクトファイルがある同じディレクトリ内にコピーします。次のブロックのターゲット名を

   ```
   target '<YourXcodeTarget>' do
       pod 'AWSAPIGateway', '~> 2.4.7'
   end
   ```

   次のようにプロジェクトのターゲット名に置き換えます。

   ```
   target 'app_objc_simple_calc' do
       pod 'AWSAPIGateway', '~> 2.4.7'
   end
   ```

   Xcode プロジェクトに `Podfile` という名前のファイルが既に含まれている場合は、次のコード行を追加します。

```
pod 'AWSAPIGateway', '~> 2.4.7'
```

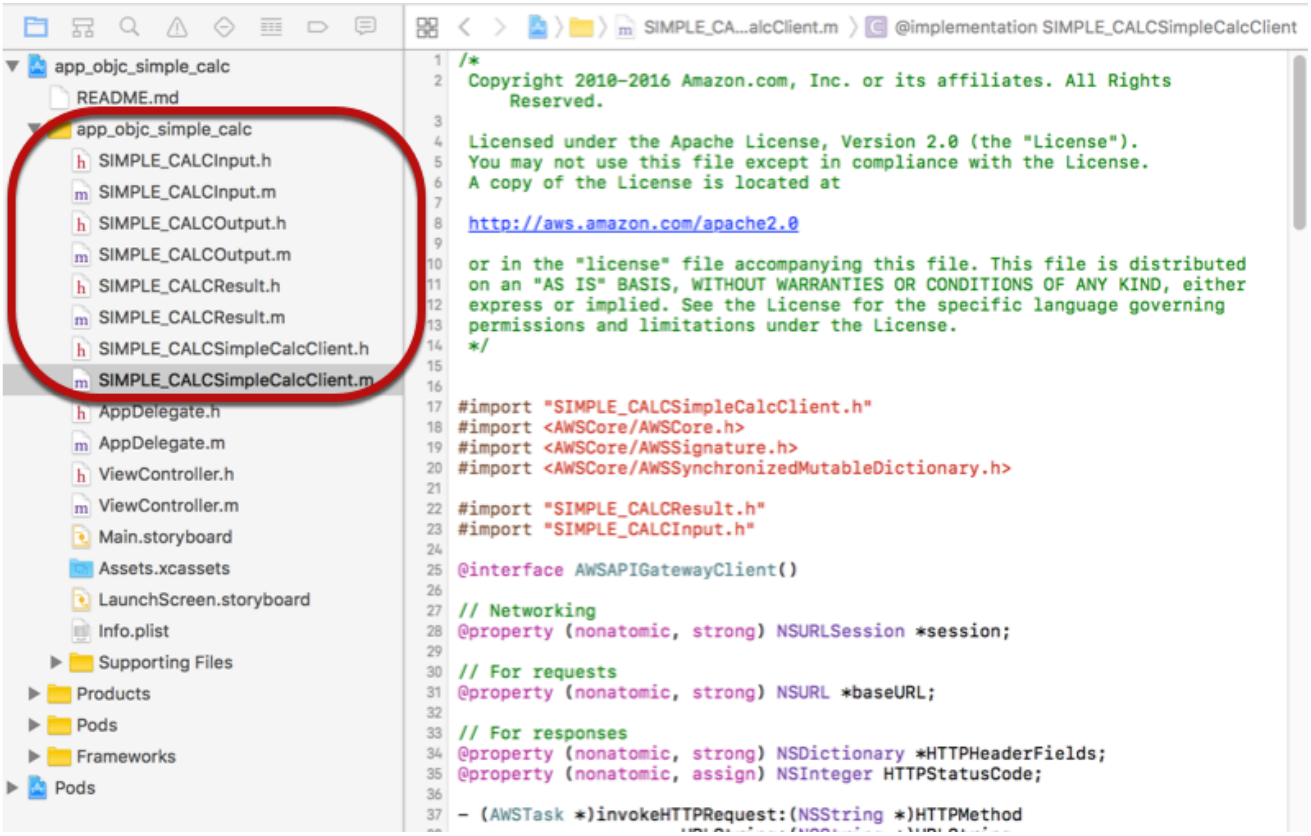
c. ターミナルウィンドウを開いて、次のコマンドを実行します。

```
pod install
```

これにより、API Gateway コンポーネント とその他の依存する AWS Mobile SDK コンポーネントがインストールされます。

d. Xcode プロジェクトを閉じて `.xcworkspace` ファイルを開き、Xcode を再起動します。

e. 抽出した SDK の `.h` ディレクトリから、すべての `.m` ファイルと `generated-src` ファイルを Xcode プロジェクト内に追加します。



AWS Mobile SDK for iOS Objective-C をプロジェクト内にインポートするために、明示的に AWS Mobile SDK をダウンロードするか、Carthage を使用する場合は、README.md ファイルの手順に従ってください。この 2 つのオプションのうち、必ず 1 つのみを使用して AWS Mobile SDK をインポートしてください。

API Gateway で生成された iOS SDK を使用して Objective-C オブジェクトで API メソッドを呼び出す

SDK の生成時に、メソッドの入力 (Input) と出力 (Result) を 2 つのモデルとする SimpleCalc API のプレフィックスとして SIMPLE_CALC を使用すると、結果の API クライアントクラスは SIMPLE_CALCSimpleCalcClient となり、対応するデータクラスはそれぞれ SIMPLE_CALCInput と SIMPLE_CALCResult になります。API のリクエストとレスポンスは、次のように SDK メソッドにマッピングされます。

- 次の API リクエストは

  ```
  GET /?a=...&b=...&op=...
  ```

  次のような SDK メソッドになります。

  ```
  (AWSTask *)rootGet:(NSString *)op a:(NSString *)a b:(NSString *)b
  ```

  AWSTask.result モデルをメソッドのレスポンスに追加した場合、SIMPLE_CALCResult プロパティのタイプは Result です。それ以外の場合、プロパティは NSDictionary タイプになります。
- 次の API リクエストは

  ```
  POST /

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

  次のような SDK メソッドになります。

  ```
  (AWSTask *)rootPost:(SIMPLE_CALCInput *)body
  ```
- 次の API リクエストは

  ```
  GET /{a}/{b}/{op}
  ```

  次のような SDK メソッドになります。

```
(AWSTask *)aBOpGet:(NSString *)a b:(NSString *)b op:(NSString *)op
```

次の手順では、Objective-C アプリケーションのソースコードで API メソッドを呼び出す方法を説明します。たとえば、viewDidLoad ファイルで ViewController.m デリゲートの一部として呼び出します。

API Gateway で生成された iOS SDK を介して API を呼び出すには

1. API クライアントクラスのヘッダーファイルをインポートして、API クライアントクラスをアプリケーションで呼び出せるようにします。

   ```
   #import "SIMPLE_CALCSimpleCalc.h"
   ```

   #import ステートメントにより、2 つのモデルクラスとして SIMPLE_CALCInput.h と SIMPLE_CALCResult.h もインポートされます。

2. API クライアントクラスをインスタンス化します。

   ```
   SIMPLE_CALCSimpleCalcClient *apiInstance = [SIMPLE_CALCSimpleCalcClient
    defaultClient];
   ```

   API で Amazon Cognito を使用するには、defaultClient メソッドを呼び出して API クライアントオブジェクトを作成する (前の例を参照) 前に、次に示すようにデフォルトの AWSServiceManager オブジェクトで defaultServiceConfiguration プロパティを設定します。

   ```
   AWSCognitoCredentialsProvider *creds = [[AWSCognitoCredentialsProvider alloc]
    initWithRegionType:AWSRegionUSEast1 identityPoolId:your_cognito_pool_id];
   AWSServiceConfiguration *configuration = [[AWSServiceConfiguration alloc]
    initWithRegion:AWSRegionUSEast1 credentialsProvider:creds];
   AWSServiceManager.defaultServiceManager.defaultServiceConfiguration =
    configuration;
   ```

3. GET /?a=1&b=2&op=+ メソッドを呼び出して、1+2 を実行します。

   ```
   [[apiInstance rootGet: @"+" a:@"1" b:@"2"] continueWithBlock:^id _Nullable(AWSTask
    * _Nonnull task) {
       _textField1.text = [self handleApiResponse:task];
   ```

```
    return nil;
}];
```

ヘルパー関数の handleApiResponse:task は、結果を文字列としてフォーマットし、テキストフィールド (_textField1) に表示します。

```
- (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;
}
```

結果として 1 + 2 = 3 と表示されます。

4. ペイロードを渡す POST / を呼び出して、1-2 を実行します。

```
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;
    }];
```

結果として 1 - 2 = -1 と表示されます。

5. GET /{a}/{b}/{op} を呼び出して、1/2 を実行します。

```
[[apiInstance aBOpGet:@"1" b:@"2" op:@"div"] continueWithBlock:^id
 _Nullable(AWSTask * _Nonnull task) {
    _textField3.text = [self handleApiResponse:task];
    return nil;
}];
```

結果として `1 div 2 = 0.5` と表示されます。ここで `div` が `/` の代わりに使用されているのは、バックエンドのシンプルな Lambda 関数では URL エンコードされたパス変数が処理されないためです。

生成された iOS SDK (Swift) を使用して API を呼び出す

次の手順を開始する前に、Swift で iOS 用の「API Gateway で REST API 用 SDK を生成する」のステップを実行し、生成された SDK の .zip ファイルをダウンロードする必要があります。

トピック

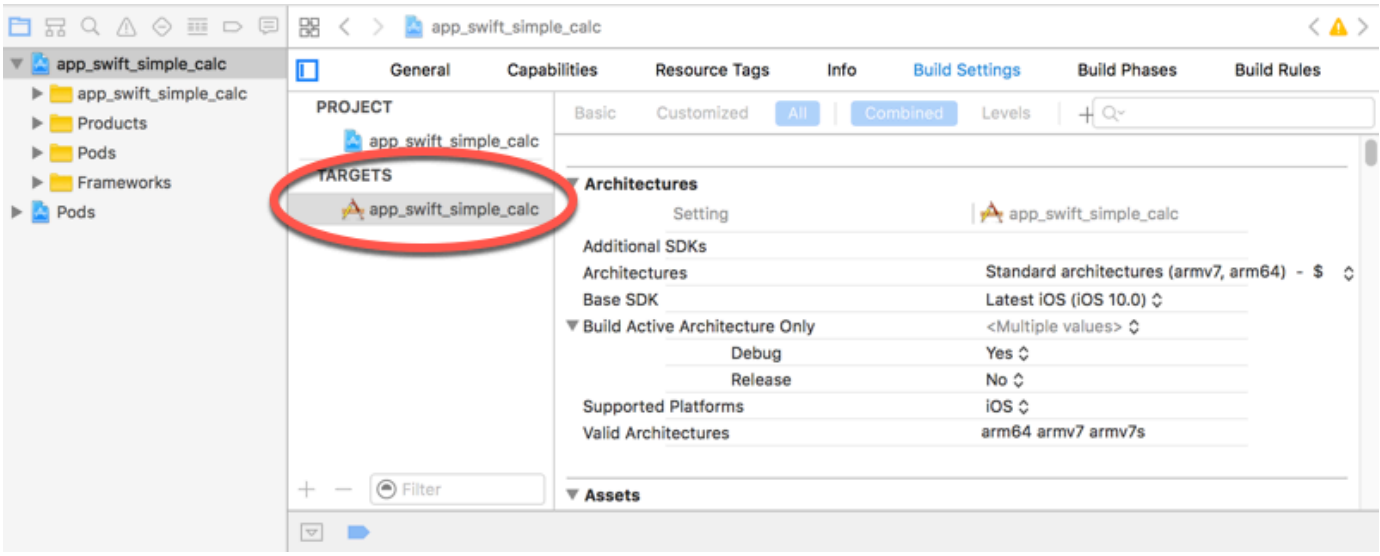
- AWS Mobile SDK と API Gateway で生成された SDK を Swift プロジェクトでインストールする
- API Gateway で生成された iOS SDK を介して Swift プロジェクトで API メソッドを呼び出す

AWS Mobile SDK と API Gateway で生成された SDK を Swift プロジェクトでインストールする

次の手順では、SDK をインストールする方法を説明します。

API Gateway で生成された iOS SDK を Swift にインストールして使用するには

1. ダウンロード済みの API Gateway で生成された .zip ファイルのコンテンツを抽出します。SimpleCalc API を使用して、解凍した SDK フォルダの名前を **`sdk_swift_simple_calc`** などに変更することもできます。この SDK フォルダーには、`README.md` ファイルと `Podfile` ファイルが含まれています。`README.md` ファイルには、SDK をインストールして使用するための手順が記載されています。このチュートリアルでは、この手順について詳しく説明します。インストールでは、CocoaPods を使用して AWS Mobile SDK の必要なコンポーネントをインポートします。SDK を Swift アプリケーションの XCode プロジェクト内にインポートするには、`Podfile` を更新する必要があります。解凍した SDK フォルダー内には、API の生成された SDK のソースコードを含む `generated-src` フォルダーもあります。
2. Xcode を起動し、新しい iOS Swift プロジェクトを作成します。プロジェクトのターゲットを書き留めておきます。それを `Podfile` に設定する必要があります。



3. CocoaPods を使用して AWS Mobile SDK の必要なコンポーネントを Xcode プロジェクト内にインポートするには、以下の操作を実行します。

   a. ターミナルウィンドウで次のコマンドを実行して CocoaPods をインストールします (まだインストールしていない場合)。

   ```
   sudo gem install cocoapods
   pod setup
   ```

   b. 抽出した SDK フォルダー内にある Podfile ファイルを、Xcode プロジェクトファイルがある同じディレクトリ内にコピーします。次のブロックのターゲット名を

   ```
   target '<YourXcodeTarget>' do
       pod 'AWSAPIGateway', '~> 2.4.7'
   end
   ```

   次のようにプロジェクトのターゲット名に置き換えます。

   ```
   target 'app_swift_simple_calc' do
       pod 'AWSAPIGateway', '~> 2.4.7'
   end
   ```

   Xcode プロジェクト内の Podfile に正しいターゲットが既に設定されている場合は、次のコード行を do ... end ループに追加するだけで済みます。

   ```
   pod 'AWSAPIGateway', '~> 2.4.7'
   ```

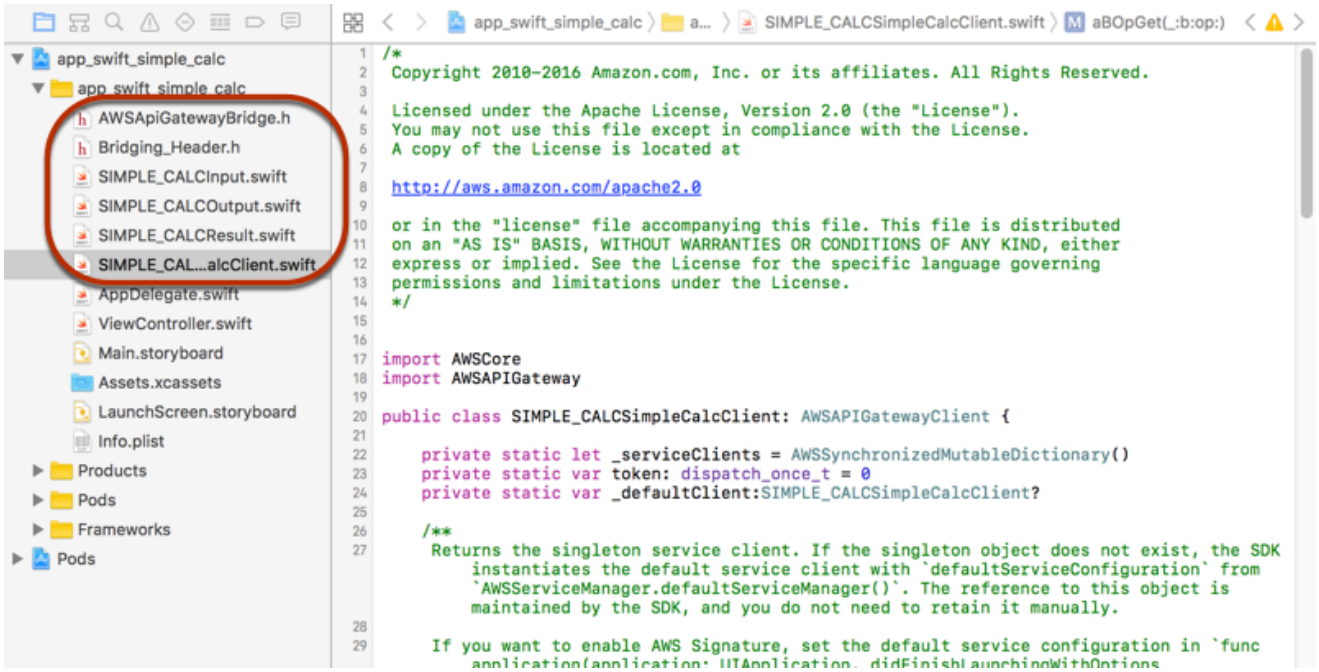
c. ターミナルウィンドウを開き、アプリケーションのディレクトリで次のコマンドを実行します。

```
pod install
```

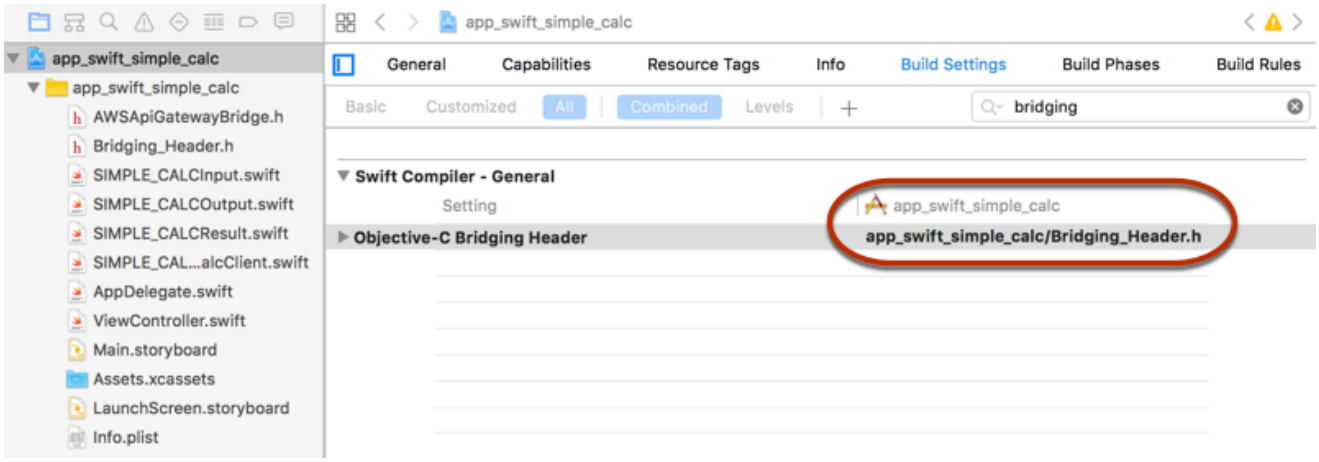
これにより、API Gateway コンポーネント とその他の依存する AWS Mobile SDK コンポーネントがアプリケーションのプロジェクト内にインストールされます。

d. Xcode プロジェクトを閉じて `*.xcworkspace` ファイルを開き、Xcode を再起動します。

e. 抽出した `.h` ディレクトリから、SDK のヘッダーファイル (`.swift`) と Swift ソースコードファイル (`generated-src`) のすべてを Xcode プロジェクトに追加します。

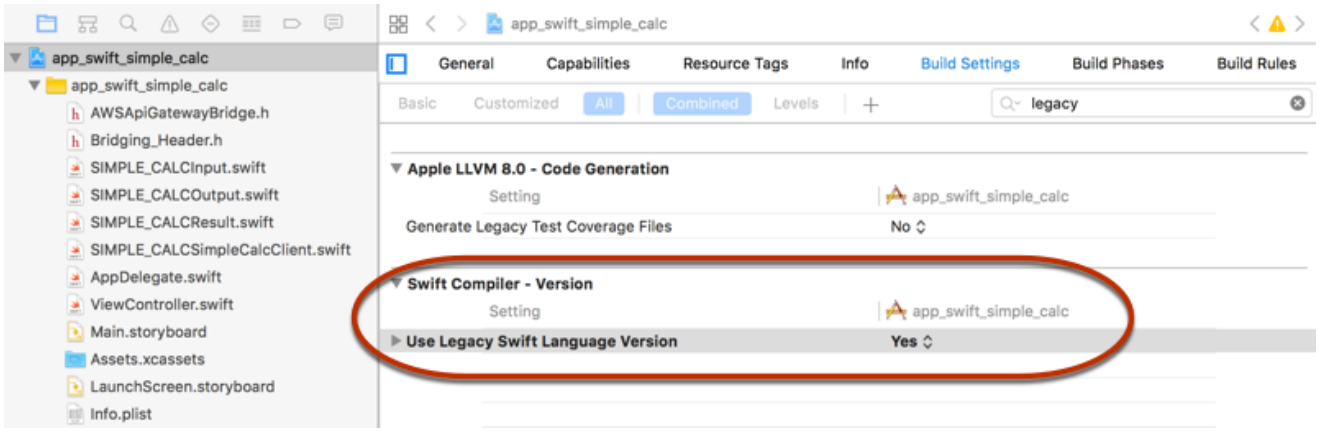


f. Swift コードプロジェクトから AWS Mobile SDK の Objective-C ライブラリを呼び出せるようにするには、Xcode プロジェクト設定の [`Bridging_Header.h`Swift Compiler - General] オプションの [Objective-C Bridging Header] プロパティで ファイルのパスを設定します。



> **Tip**
>
> Xcode の検索ボックスに「**`bridging`**」と入力し、[Objective-C Bridging Header] プロパティを見つけます。

g. Xcode プロジェクトを構築し、それが適切に設定されていることを確認してから先に進みます。Xcode で使用している Swift のバージョンが AWS Mobile SDK でサポートされているものより新しい場合は、Swift コンパイラエラーが発生します。この場合は、[Swift Compiler - Version (Swift コンパイラ - バージョン)] 設定の [Use Legacy Swift Language Version] プロパティを [Yes] に設定します。



Swift で の AWS Mobile SDK for iOS をプロジェクト内にインポートするために、明示的に AWS Mobile SDK をダウンロードするか、Carthage を使用する場合は、SDK パッケージに含まれている `README.md` ファイルに記載されている手順に従います。この 2 つのオプションのうち、必ず 1 つのみを使用して AWS Mobile SDK をインポートしてください。

API Gateway で生成された iOS SDK を介して Swift プロジェクトで API メソッドを呼び出す

SDK の生成時に、API のリクエストとレスポンスの入力 (Input) と出力 (Result) を記述する 2 つのモデルがあるこの SimpleCalc API のプレフィックスとして SIMPLE_CALC を使用すると、結果の API クライアントクラスは SIMPLE_CALCSimpleCalcClient となり、対応するデータクラスはそれぞれ SIMPLE_CALCInput と SIMPLE_CALCResult になります。API のリクエストとレスポンスは、次のように SDK メソッドにマッピングされます。

- 次の API リクエストは

  ```
  GET /?a=...&b=...&op=...
  ```

  次のような SDK メソッドになります。

  ```
  public func rootGet(op: String?, a: String?, b: String?) -> AWSTask
  ```

  AWSTask.result モデルをメソッドのレスポンスに追加した場合、SIMPLE_CALCResult プロパティのタイプは Result です。それ以外の場合は、NSDictionary タイプになります。

- 次の API リクエストは

  ```
  POST /

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

  次のような SDK メソッドになります。

  ```
  public func rootPost(body: SIMPLE_CALCInput) -> AWSTask
  ```

- 次の API リクエストは

  ```
  GET /{a}/{b}/{op}
  ```

  次のような SDK メソッドになります。

```
public func aBOpGet(a: String, b: String, op: String) -> AWSTask
```

次の手順では、Swift アプリケーションのソースコードで API メソッドを呼び出す方法を示します。たとえば、viewDidLoad() ファイルで ViewController.m デリゲートの一部として呼び出します。

API Gateway で生成された iOS SDK を介して API を呼び出すには

1. API クライアントクラスをインスタンス化します。

   ```
   let client = SIMPLE_CALCSimpleCalcClient.default()
   ```

   API で Amazon Cognito を使用するには、default メソッド (前出) を取得する前にデフォルトの AWS のサービス設定 (後出) を設定します。

   ```
   let credentialsProvider =
    AWSCognitoCredentialsProvider(regionType: AWSRegionType.USEast1, identityPoolId:
    "my_pool_id")
   let configuration = AWSServiceConfiguration(region: AWSRegionType.USEast1,
    credentialsProvider: credentialsProvider)
   AWSServiceManager.defaultServiceManager().defaultServiceConfiguration =
    configuration
   ```

2. GET /?a=1&b=2&op=+ メソッドを呼び出して、1+2 を実行します。

   ```
   client.rootGet("+", a: "1", b:"2").continueWithBlock {(task: AWSTask) -> AnyObject?
    in
       self.showResult(task)
       return nil
   }
   ```

   上のヘルパー関数 self.showResult(task) は結果またはエラーをコンソールに表示します。次に例を示します。

   ```
   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)")
        }
    }
}
```

本稼働アプリケーションでは、結果またはエラーをテキストフィールドに表示できます。結果として 1 + 2 = 3 と表示されます。

3. ペイロードを渡す POST / を呼び出して、1-2 を実行します。

```
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
}
```

結果として 1 - 2 = -1 と表示されます。

4. GET /{a}/{b}/{op} を呼び出して、1/2 を実行します。

```
client.aBOpGet("1", b:"2", op:"div").continueWithBlock {(task: AWSTask) ->
 AnyObject? in
    self.showResult(task)
    return nil
}
```

結果として 1 div 2 = 0.5 と表示されます。ここで div が / の代わりに使用されているのは、バックエンドのシンプルな Lambda 関数では URL エンコードされたパス変数が処理されないためです。

OpenAPI を使用した REST API の設定

API Gateway を使用して、REST API を外部定義ファイルから API Gateway にインポートできます。現在、API Gateway は OpenAPI v2.0 および OpenAPI v3.0 定義ファイルをサポートしています。この例外が「Amazon API Gateway の REST API に関する重要な注意点」に記載されています。新しい定義で上書きして API を更新したり、既存の API と定義をマージしたりできます。リクエスト URL で mode クエリパラメータを使用して、オプションを指定します。

API Gateway コンソールからサンプル API 機能を使用するチュートリアルについては、「チュートリアル: サンプルをインポートして REST API を作成する」を参照してください。

トピック

- エッジ最適化 API を API Gateway にインポートする
- リージョン API を API Gateway にインポートする
- OpenAPI ファイルをインポートして既存の API 定義を更新する
- OpenAPIbasePath プロパティを設定
- OpenAPI インポート用の AWS 変数
- インポート中のエラーと警告
- API Gateway から REST API をエクスポートする

エッジ最適化 API を API Gateway にインポートする

API OpenAPI 定義ファイルをインポートして、新しいエッジ最適化 API を作成できます。そのためには、OpenAPI ファイルに加えて EDGE エンドポイントタイプをインポートオペレーションへの入力として指定します。これは、API Gateway コンソール、AWS CLI、または AWS SDK を使用して行うこともできます。

API Gateway コンソールからサンプル API 機能を使用するチュートリアルについては、「チュートリアル: サンプルをインポートして REST API を作成する」を参照してください。

トピック

- API Gateway コンソールを使用してエッジ最適化 API をインポートする
- AWS CLI を使用してエッジ最適化 API をインポートする

API Gateway コンソールを使用してエッジ最適化 API をインポートする

API Gateway コンソールを使用してエッジ最適化 API をインポートするには、次の操作を行います。

1. API Gateway コンソール (https://console.aws.amazon.com/apigateway) にサインインします。
2. [API の作成] を選択します。
3. [REST API] で、[インポート] を選択します。
4. API OpenAPI 定義をコピーしてコードエディタに貼り付けるか、[ファイルの選択] を選択してローカルドライブから OpenAPI ファイルを読み込みます。
5. [エンドポイントタイプ] で、[エッジ最適化] を選択します。
6. [API の作成] を選択して OpenAPI 定義のインポートを開始します。

AWS CLI を使用してエッジ最適化 API をインポートする

AWS CLI を使用して、OpenAPI 定義ファイルから API をインポートすることで新しいエッジ最適化 API を作成するには、`import-rest-api` コマンドを以下のように使用します。

```
aws apigateway import-rest-api \
    --fail-on-warnings \
    --body 'file://path/to/API_OpenAPI_template.json'
```

または、`endpointConfigurationTypes` クエリ文字列パラメータを `EDGE` に明示的に指定します。

```
aws apigateway import-rest-api \
    --parameters endpointConfigurationTypes=EDGE \
    --fail-on-warnings \
    --body 'file://path/to/API_OpenAPI_template.json'
```

リージョン API を API Gateway にインポートする

API のインポート時、API のリージョンのエンドポイント設定を選択できます。API Gateway コンソール、AWS CLI、または AWS SDK を使用できます。

API のエクスポート時、エクスポートされた API 定義に API エンドポイント設定は含まれません。

API Gateway コンソールからサンプル API 機能を使用するチュートリアルについては、「チュートリアル: サンプルをインポートして REST API を作成する」を参照してください。

トピック

- API Gateway コンソールを使用してリージョン API をインポートする
- AWS CLI を使用してリージョン API をインポートする

API Gateway コンソールを使用してリージョン API をインポートする

API Gateway コンソールを使用してリージョンのエンドポイントの API をインポートするには、以下の手順を実行します。

1. https://console.aws.amazon.com/apigateway で API Gateway コンソールにサインインします。
2. [API の作成] を選択します。
3. [REST API] で、[インポート] を選択します。
4. API OpenAPI 定義をコピーしてコードエディタに貼り付けるか、[ファイルの選択] を選択してローカルドライブから OpenAPI ファイルを読み込みます。
5. [API エンドポイントタイプ] で、[リージョン別] を選択します。
6. [API の作成] を選択して OpenAPI 定義のインポートを開始します。

AWS CLI を使用してリージョン API をインポートする

AWS CLI を使用して OpenAPI 定義ファイルから API をインポートするには、`import-rest-api` コマンドを使用します。

```
aws apigateway import-rest-api \
    --parameters endpointConfigurationTypes=REGIONAL \
    --fail-on-warnings \
    --body 'file://path/to/API_OpenAPI_template.json'
```

OpenAPI ファイルをインポートして既存の API 定義を更新する

API 定義をインポートできるのは、エンドポイント設定、ステージおよびステージ変数、または API キーへの参照を変更せずに既存の API を更新する場合のみです。

インポートから更新への操作は、マージまたは上書きという 2 つのモードで行うことができます。

API (A) が別の API (B) にマージされると、作成された API では、2 つの API で矛盾する定義が共有されていない限り、A と B の両方の定義が保持されます。矛盾が発生した場合、マージする API (A) のメソッド定義によってマージ先の API (B) の対応するメソッド定義がオーバーライドされます。たとえば、B が 200 および 206 レスポンスを返す次のメソッドを宣言しているとします。

```
GET /a
POST /a
```

A は 200 および 400 レスポンスを返す次のメソッドを宣言しています。

```
GET /a
```

A が B にマージされると、作成される API は次のメソッドを生成します。

```
GET /a
```

200 レスポンスと 400 レスポンスを返すメソッド。

```
POST /a
```

200 レスポンスと 206 レスポンスを返すメソッド。

API のマージは、外部の API 定義を複数の小さな部分に分解し、それらの部分の変更を一度に 1 つのみ適用する場合に役立ちます。たとえば、複数のチームが API のさまざまな部分を担当し、異なるレートで変更を可能にする場合です。このモードでは、インポートされた定義で明確に定義されていない既存の API の項目はそのまま残されます。

API (A) によって別の API (B) がオーバーライドされ、作成される API には API (A) の定義が採用されます。API の上書きは、外部の API 定義に、API の完全な定義が含まれているときに役立ちます。このモードでは、インポートされた定義で明確に定義されていない既存の API の項目は削除されます。

API をマージするには、PUT リクエストを `https://apigateway.<region>.amazonaws.com/restapis/<restapi_id>?mode=merge` に送信します。`restapi_id` パスのパラメータ値は、指定された API 定義のマージ先となる API を示します。

次のコードスニペットは、指定された API がすでに API Gateway にあり、JSON の OpenAPI API 定義をペイロードとしてマージする PUT リクエストの例を示しています。

```
PUT /restapis/<restapi_id>?mode=merge
Host:apigateway.<region>.amazonaws.com
Content-Type: application/json
Content-Length: ...

An OpenAPI API definition in JSON
```

更新オペレーションのマージでは、2 つの完全な API 定義を使用し、それらをマージします。小さい増分変更の場合は、リソースの更新オペレーションを使用できます。

API をオーバーライドするには、PUT リクエストを `https://apigateway.<region>.amazonaws.com/restapis/<restapi_id>?mode=overwrite` に送信します。`restapi_id` パスパラメータは、指定された API 定義で上書きされる API を示します。

次のコードスニペットは、JSON 形式の OpenAPI 定義のペイロードでリクエストを上書きする例を示しています。

```
PUT /restapis/<restapi_id>?mode=overwrite
Host:apigateway.<region>.amazonaws.com
Content-Type: application/json
Content-Length: ...

An OpenAPI API definition in JSON
```

`mode` クエリパラメータを指定しないと、マージが想定されます。

> **Note**
>
> PUT オペレーションはべき等ですが、アトミックではありません。つまり、処理中にシステムエラーが発生した場合、API は不正な状態になる可能性があります。ただし、オペレーションを正常に繰り返すと、API は最初のオペレーションが成功した場合と同じ最終状態になります。

OpenAPI**basePath** プロパティを設定

OpenAPI 2.0 では、basePath プロパティを使用して、paths プロパティに定義された各パスに先行する 1 つ以上のパス部分を提供できます。API Gateway にはリソースのパスを表現する複数の方法があるため、API のインポート機能には、インポート中に basePath プロパティを解釈するための次のオプションが用意されています。ignore、prepend、および split です。

「OpenAPI 3.0」では、basePath は、最上位のプロパティではありません。代わりに、API Gateway は規則としてサーバー変数を使用します。API のインポート機能には、インポート中に基本パスを解釈するための同じオプションが用意されています。基本パスは次のように識別されます。

- API に basePath 変数が含まれていない場合、API のインポート機能は server.url 文字列を確認して、"/" 以外のパスが含まれているかどうかを確認します。含まれている場合は、そのパスが基本パスとして使用されます。
- API に含まれる basePath 変数が 1 つだけの場合、API のインポート機能は server.url で参照されていなくても、それを基本パスとして使用します。
- API に複数の basePath 変数が含まれている場合、API のインポート機能は最初の変数のみを基本パスとして使用します。

Ignore (無視)

OpenAPI ファイルの basePath の値が /a/b/c で、paths プロパティに /e および /f が含まれる場合に、次の POST または PUT リクエストがあるとします。

```
POST /restapis?mode=import&basepath=ignore
```

```
PUT /restapis/api_id?basepath=ignore
```

この場合、API で次のリソースが発生します。

- /
- /e
- /f

その効果として、basePath を、これが存在しなかったかのように扱い、宣言された API のすべてのリソースは、ホストに対して相対的に提供されます。これを使用できるのは、たとえば、基本パ

スを含まない API マッピングや、本番ステージを参照するステージ値を持つカスタムドメイン名がある場合です。

> **Note**
>
> API Gateway は、定義ファイルに明示的に宣言されていない場合でも、自動的にルートリソースを作成します。

指定しない場合、basePath はデフォルトで ignore を受け取ります。

Prepend

OpenAPI ファイルの basePath の値が /a/b/c で、paths プロパティに /e および /f が含まれる場合に、次の POST または PUT リクエストがあるとします。

```
POST /restapis?mode=import&basepath=prepend
```

```
PUT /restapis/api_id?basepath=prepend
```

この場合、API で次のリソースが発生します。

- /
- /a
- /a/b
- /a/b/c
- /a/b/c/e
- /a/b/c/f

その効果として、(メソッドなしで) 追加のリソースとして basePath を処理し、宣言されたリソースセットに追加します。これを使用できるのは、たとえば、さまざまなチームが API パートの異なる部分を担当し、basePath が各チームの API 部分のパスの場所を参照できる場合です。

> **Note**
>
> API Gateway は、定義に明示的に宣言されていない場合でも、自動的に中間リソースを作成します。

Split

OpenAPI ファイルの basePath の値が /a/b/c で、paths プロパティに /e および /f が含まれる場合に、次の POST または PUT リクエストがあるとします。

```
POST /restapis?mode=import&basepath=split
```

```
PUT /restapis/api_id?basepath=split
```

この場合、API で次のリソースが発生します。

- /
- /b
- /b/c
- /b/c/e
- /b/c/f

その効果として、最上位のパス部分 /a を、各リソースのパスの先頭として扱い、API 内で (メソッドなしで) 追加のリソースを作成します。これを使用できるのは、たとえば、a が、API の一部として公開するステージ名である場合です。

OpenAPI インポート用の AWS 変数

OpenAPI 定義では、次の AWS 変数を使用できます。API Gateway は、API のインポート時に変数を解決します。変数を指定するには、${*variable-name*} を使用します。

AWS 変数

変数名。	説明	
AWS::AccountId	API をインポートする AWS アカウント ID (例えば、123456789012)。	
AWS::Partition	API のインポート先の AWS パーティション。標準の AWS リージョンの場合、パーティションは aws です。	
AWS::Region	API のインポート先の AWS リージョン — 例えば、us-east-2 。	

AWS 変数の例

次の例では、AWS 変数を使用して統合用の AWS Lambda 関数を指定します。

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
```

インポート中のエラーと警告

インポート中のエラー

インポート中に、無効な OpenAPI ドキュメントなど、大きな問題に対してエラーが生成される場合があります。エラーは、失敗のレスポンスの例外 (例: BadRequestException) として返されます。エラーが発生した場合、新しい API 定義は破棄され、既存の API は変更されません。

インポート中の警告

インポート中に、モデル参照の不足など、小規模な問題に対して警告が生成される場合があります。警告が発生した場合、リクエスト URL に failonwarnings=false クエリ式が追加されている場合、オペレーションは続行します。それ以外の場合、更新はロールバックされます。デフォルトで、failonwarnings は false に設定されています。このような場合、警告は RestApi リソースのフィールドとして返されます。それ以外の場合、警告は例外のメッセージとして返されます。

API Gateway から REST API をエクスポートする

API Gateway で REST API を作成および設定したら、API Gateway コンソールなどから API Gateway Export API (Amazon API Gateway Control Service の一部です) を使用して、API を OpenAPI ファイルにエクスポートできます。API Gateway Export API を使用するには、API リクエストに署名する必要があります。リクエストの署名の詳細については、「IAM ユーザーガイド」の「AWS API リクエストの署名」を参照してください。エクスポートされた OpenAPI 定義ファイルに、API Gateway 統合の拡張機能と、Postman 拡張機能を含めるオプションがあります。

> **Note**
>
> AWS CLI を使用して API をエクスポートする場合、次の例に示すように extensions パラメータを必ず含めて、`x-amazon-apigateway-request-validator` 拡張子が含まれるようにします。
>
> ```
> aws apigateway get-export --parameters extensions='apigateway' --rest-api-id
>  abcdefg123 --stage-name dev --export-type swagger latestswagger2.json
> ```

ペイロードが `application/json` 型でない場合、API をエクスポートすることはできません。エクスポートを試みると、JSON 本文モデルが見つからないことを示すエラーレスポンスが返されます。

REST API をエクスポートするリクエスト

Export API を使用すると、GET リクエストを送信し、URL パスの一部としてエクスポートされる API を指定することにより、既存の REST API をエクスポートします。リクエストの URL は次の形式です。

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
```

`extensions` クエリ文字列を追加して、(値 `integration` を使用) API Gateway 拡張を含めるか、(値 `postman` を使用) Postman 拡張を含めるかを指定できます。

さらに、`Accept` ヘッダーを `application/json` または `application/yaml` に設定して、それぞれ JSON 形式または YAML 形式で API 定義の出力を受け取ることができます。

API Gateway Export API を使用して GET リクエストを送信する詳細については、「GetExport」を参照してください。

> **Note**
>
> API でモデルを定義する場合、モデルをエクスポートする API Gateway の "application/json" のコンテンツタイプである必要があります。それ以外の場合、API Gateway は「Only found non-JSON body models for ...」というエラーメッセージとともに例外をスローします。モデルはプロパティを含むか、特定の JSONSchema 型として定義される必要があります。

REST API OpenAPI 定義を JSON でダウンロードする

OpenAPI 定義を JSON 形式にして REST API をエクスポートおよびダウンロードするには、以下のようにします。

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
```

ここで、*<region>* はたとえば us-east-1 にできます。API Gateway を利用できるすべてのリージョンについては、「リージョンとエンドポイント」を参照してください。

REST API OpenAPI 定義を YAML でダウンロードする

OpenAPI 定義を YAML 形式にして REST API をエクスポートおよびダウンロードするには、以下のようにします。

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
```

Postman 拡張機能を使用して REST API OpenAPI 定義を JSON でダウンロードする

Postman を使用して OpenAPI 定義を JSON 形式にして REST API をエクスポートおよびダウンロードするには、以下のようにします。

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
```

API Gateway 統合を使用して REST API OpenAPI 定義ファイルを YAML でダウンロードする

OpenAPI 定義を YAML 形式にして API Gateway 統合を使用して REST API をエクスポートおよびダウンロードするには、以下のようにします。

OpenAPI 3.0

```
GET /restapis/<restapi_id>/stages/<stage_name>/exports/oas30?extensions=integrations
Host: apigateway.<region>.amazonaws.com
Accept: application/yaml
```

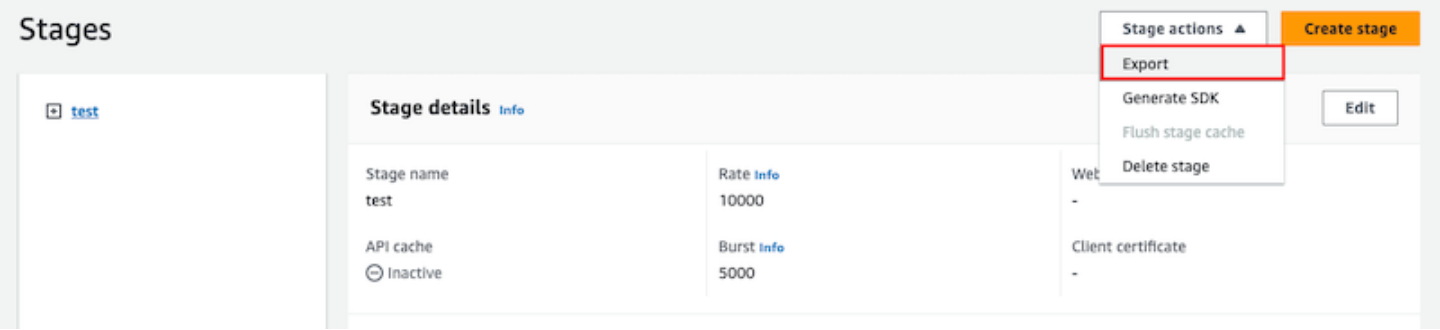
OpenAPI 2.0

```
GET /restapis/<restapi_id>/stages/<stage_name>/exports/swagger?
extensions=integrations
Host: apigateway.<region>.amazonaws.com
Accept: application/yaml
```

API Gateway コンソールを使用して REST API をエクスポートする

REST API をステージにデプロイしたら、次に API Gateway コンソールを使用してステージ内の API を OpenAPI ファイルにエクスポートすることができます。

API Gateway コンソールの [ステージ] ペインで、[ステージアクション]、[エクスポート] を選択します。



[API 仕様タイプ]、[フォーマット]、[拡張機能] を指定して API の OpenAPI 定義をダウンロードします。

ユーザーが呼び出せるように REST API を公開する

API Gateway API を作成して開発するだけでは、ユーザーが自動的に呼び出せるようにはなりません。呼び出し可能にするには、API をステージにデプロイする必要があります。さらに、ユーザーが API にアクセスするために使用する URL をカスタマイズすることもできます。ブランドと一致するドメインや、API のデフォルト URL よりも記憶に残るドメインを指定できます。

このセクションでは、API をデプロイし、アクセスするためにユーザーに提供する URL をカスタマイズする方法を説明しています。

> **Note**
>
> API Gateway API のセキュリティを強化するため、`execute-api.{region}.amazonaws.com` ドメインはパブリックサフィックスリスト (PSL) に登録されます。セキュリティ強化のため、API Gateway API のデフォルトドメイン名に機密な Cookie を設定する必要が生じた場合は、`__Host-` プレフィックスの付いた Cookie の使用をお勧めします。このプラクティスは、クロスサイトリクエストフォージェリ (CSRF) 攻撃からドメインを防ぐ際に役立ちます。詳細については、Mozilla 開発者ネットワークの「Set-Cookie」ページを参照してください。

トピック

- Amazon API Gateway での REST API のデプロイ
- REST API のカスタムドメイン名を設定する

Amazon API Gateway での REST API のデプロイ

API を作成したら、この API をデプロイしてユーザーが呼び出せるようにする必要があります。

API をデプロイするには、API デプロイを作成し、それを ステージに関連付けます。ステージは、API のライフサイクル状態への論理的なリファレンスです (例:`dev` 、`prod`、`beta`、`v2`) 。API ステージは API ID とステージ名によって識別されます。これらは、API を呼び出すために使用する URL に含まれています。各ステージは、API のデプロイの名前付きリファレンスで、クライアントアプリケーションから呼び出すことができます。

> **Important**
>
> API を更新するたびに、API を既存のステージまたは新しいステージに再デプロイする必要があります。API の更新には、ルート、メソッド、統合、オーソライザー、リソースポリシーなど、ステージ設定以外のすべての変更が含まれます。

API が進化するにつれて、API の異なるバージョンとしてさまざまなステージにデプロイし続けることができます。API アップデートは、Canary リリースデプロイとしてデプロイすることもできます。これにより、API クライアントは、同じステージで、プロダクションリリースからプロダクションバージョン、および Canary リリースから更新されたバージョンにアクセスできます。

デプロイされた API を呼び出すために、クライアントは API の URL に対してリクエストを送信します。URL は、API のプロトコル (HTTP(S) または (WSS))、ホスト名、ステージ名、および (REST API の場合) リソースパスによって決定されます。ホスト名とステージ名によって、API のベース URL が決まります。

API のデフォルトドメイン名を使用すると、特定のステージ (*`{stageName}`*) の REST API のベース URL (たとえば) は、次の形式になります。

```
https://{restapi-id}.execute-api.{region}.amazonaws.com/{stageName}
```

API のデフォルトのベース URL をよりユーザーフレンドリなものにするには、カスタムドメイン名 (たとえば、`api.example.com`) を作成し、API のデフォルトのホスト名と置き換えることができます。カスタムドメイン名で複数の API をサポートするには、API ステージをベースパスにマッピングする必要があります。

`{api.example.com}` のカスタムドメイン名と API ステージがカスタムドメイン名の下の (*`{basePath}`*) ベースパスにマップされると、REST API のベース URL は次のようになります。

```
https://{api.example.com}/{basePath}
```

ステージごとに、アカウントレベルのデフォルトのリクエストスロットリング制限を調整し、API キャッシュを有効にすることで、API のパフォーマンスを最適化できます。また、API コールのログを CloudTrail や CloudWatch に記録し、バックエンドで API リクエストを認証するためのクライエント証明書を選択することもできます。さらに、ランタイムに、ステージ固有の環境コンテキストを API 統合に渡すために、個々のメソッドのステージレベル設定を上書きし、ステージ変数を定義することができます。

ステージを使用すると、API の堅牢なバージョン管理が可能になります。たとえば、API を `test` ステージと、`prod` ステージにデプロイし、`test` ステージをテストビルドとして使用し、`prod` ステージを安定したビルドとして使用できます。更新がテストに合格したら、`test` ステージを `prod` ステージに昇格させることができます。昇格は、API を `prod` 本稼働ステージに再デプロイするか、ステージ変数値を `test` ステージ名から `prod` のステージ変数に更新することによって行うことができます。

このセクションでは、API Gateway コンソールを使用するか、API Gateway REST API を呼び出して API をデプロイする方法について説明します。他のツールを使用するには、AWS CLI または AWS SDK のドキュメントを参照してください。

トピック

- API Gateway で REST API をデプロイします。
- REST API のステージのセットアップ
- API Gateway の Canary リリースデプロイの設定
- 再デプロイが必要な REST API の更新

API Gateway で REST API をデプロイします。

API Gateway では、REST API のデプロイは Deployment リソースにより表現されます。これは、RestApi リソースによって表される API の実行可能ファイルと似ています。

クライアントが API を呼び出すには、デプロイを作成してステージを関連付ける必要があります。ステージは、Stage リソースによって表されます。これは、メソッド、統合、モデル、マッピングテンプレート、Lambda オーソライザー (以前のカスタムオーソライザー) を含む API のスナップショットを表します。API を更新すると、新しいステージを既存のステージに関連付けることによって API を再デプロイできます。ステージの作成については、the section called “ステージのセットアップ” で説明されています。

トピック

- AWS CLI を使用してデプロイを作成する
- API Gateway コンソールから REST API をデプロイする

AWS CLI を使用してデプロイを作成する

デプロイを作成するときは、Deployment リソースをインスタンス化します。API Gateway コンソール、AWS CLI、AWS SDK、または API Gateway REST API を使用して、デプロイを作成できます。

CLI を使用してデプロイを作成するには、create-deployment コマンドを使用します。

```
aws apigateway create-deployment --rest-api-id <rest-api-id> --region <region>
```

このデプロイをステージに関連付けるまで、API は呼び出し可能ではありません。既存のステージでは、ステージの `deploymentId` プロパティを新しく作成されたデプロイ ID (`<deployment-id>`) で更新することで実行できます。

```
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>'
```

初めて API をデプロイする場合は、ステージの作成とデプロイの作成を同時に行うことができます。

```
aws apigateway create-deployment --region <region> \
   --rest-api-id <rest-api-id> \
   --stage-name <stage-name>
```

これは、初めて API をデプロイするとき、または API を新しいステージに再デプロイするときに、API Gateway コンソールの背後で行われる処理です。

API Gateway コンソールから REST API をデプロイする

REST API は、初めてデプロイする前に作成済みであることが必要です。詳細については、「API Gateway での REST API の開発」を参照してください。

トピック

- REST API をステージにデプロイする
- REST API をステージに再デプロイする
- REST API デプロイのステージ設定を更新する
- REST API デプロイのステージ変数を設定する
- ステージを別の REST API デプロイと関連付ける

REST API をステージにデプロイする

API Gateway コンソールでは、デプロイを作成して新規または既存のステージに関連付けることで API をデプロイできます。

> **Note**
>
> API Gateway のステージを異なるデプロイに関連付けるには、代わりに「ステージを別の REST API デプロイと関連付ける」を参照してください。

1. https://console.aws.amazon.com/apigateway で API Gateway コンソールにサインインします。
2. [API] ナビゲーションペインで、デプロイする API を選択します。
3. [リソース] ペインで、[API のデプロイ] を選択します。
4. [ステージ] では、次の中から選択します。
   a. 新しいステージを作成するには、[新規ステージ] を選択し、[ステージ名] に名前を入力します。オプションで [デプロイの説明] にこのデプロイの説明を入力できます。
   b. 既存のステージを選択するには、ドロップダウンメニューからステージ名を選択します。[デプロイの説明] に新しいデプロイの説明を入力することもできます。
   c. ステージに関連付けられていないデプロイを作成するには、[ステージなし] を選択します。後で、このデプロイをステージに関連付けることができます。
5. [デプロイ] を選択します。

REST API をステージに再デプロイする

API を再デプロイするには、「the section called “REST API をステージにデプロイする”」と同じ手順を実行します。同じステージを必要に応じて何度でも再利用することができます。

REST API デプロイのステージ設定を更新する

API のデプロイ後に、ステージ設定を変更して API キャッシュ、ログ記録、またはリクエストのスロットリングを有効または無効にすることができます。また、バックエンドで API Gateway を検証するためのクライアント証明書を選択したり、ランタイム時に API 統合にデプロイコンテキストを渡すようにステージ変数を設定したりすることもできます。詳細については、「ステージ設定の更新」を参照してください。

> **Important**
>
> ステージ設定を変更したら、変更を有効にするために API を再デプロイする必要があります。

> **Note**
>
> ログ記録の有効化など、更新した設定が新しい IAM ロールを必要とする場合、API を再デプロイせずに必要な IAM ロールを追加できます。ただし新しい IAM ロールが有効になるまでには、数分かかる場合があります。有効になるまでは、ログ作成オプションを有効にしていたとしても、API 呼び出しのトレースは記録されません。

REST API デプロイのステージ変数を設定する

デプロイで、ランタイムに API 統合に対してデプロイ固有のデータを渡すようにステージ変数を設定または変更できます。この操作は、[ステージエディター] の [ステージ変数] タブで行うことができます。詳細については、「REST API デプロイのステージ変数のセットアップ」の手順を参照してください。

ステージを別の REST API デプロイと関連付ける

デプロイは API スナップショットを表し、ステージはスナップショットへのパスを定義するため、別のデプロイとステージの組み合わせを選択して、ユーザーが API の異なるバージョンを呼び出す方法を制御できます。これは、API のステージを前のデプロイにロールバックしたり、API の「プライベートブランチ」をパブリックブランチにマージしたりする場合に役立ちます。

次の手順では、この操作を API Gateway コンソールの [Stage Editor (ステージエディター)] を使用して行う方法について説明します。以下では、API を複数回デプロイした経験があることを前提としています。

1. まだ [ステージ] ペインを開いていない場合は、メインナビゲーションペインで [ステージ] を選択します。
2. 更新するステージを選択します。
3. [デプロイ履歴] タブで、ステージに使用するデプロイを選択します。
4. [アクティブなデプロイの変更] を選択します。
5. アクティブなデプロイを変更することを確認し、[アクティブなデプロイを作成] ダイアログボックスの [アクティブなデプロイを変更] を選択します。

REST API のステージのセットアップ

ステージは、デプロイに対する名前付きのリファレンスで、API のスナップショットです。Stage (ステージ)を使用して、特定のデプロイを管理および最適化します。たとえば、ステージ設定を設定して、キャッシングを有効にしたり、リクエストスロットリングをカスタマイズしたり、ログ記録を設定したり、ステージ変数を定義したり、テストのために Canary リリースをアタッチしたりすることができます。

トピック

- API Gateway コンソールを使用したステージのセットアップ
- API Gateway で API ステージのタグをセットアップする
- REST API デプロイのステージ変数のセットアップ

API Gateway コンソールを使用したステージのセットアップ

トピック

- 新しいステージを作成する
- ステージ設定の更新
- ステージレベルの設定のオーバーライド
- ステージを削除する

新しいステージを作成する

最初のデプロイ後に、さらにステージを追加して既存のデプロイに関連付けることができます。API Gateway コンソールを使用して新しいステージを作成して使用するか、API をデプロイするときに既存のステージを選択できます。通常は、API を再デプロイする前に、API デプロイに新しいステー

ジを追加できます。API Gateway コンソールを使用して新しいステージを作成するには、次の手順に従います。

1. https://console.aws.amazon.com/apigateway で API Gateway コンソールにサインインします。
2. REST API を選択します。
3. メインナビゲーションペインで、API の下にある [ステージ] を選択します。
4. [ステージ] ナビゲーションペインから、[ステージの作成] を選択します。
5. [ステージ名] に、名前を入力します (例: **`prod`**)。

   > **Note**
   >
   > ステージ名には、英数字、ハイフン、およびアンダースコアのみ含めることができます。最大長は 128 文字です。

6. (オプション)。[説明] に、ステージの説明を入力します。
7. [デプロイメント] には、このステージに関連付ける既存の API デプロイの日付と時刻を選択します。
8. [追加設定] では、ステージの追加設定を指定できます。
9. [テーブルの作成] を選択します。

ステージ設定の更新

API が正常にデプロイされると、ステージにデフォルト設定が入力されます。API キャッシュやログ記録などのステージ設定は、コンソールまたは API Gateway REST API を使用して変更できます。次の手順では、API Gateway コンソールのステージエディタを使用してその方法を示します。

API Gateway コンソールを使用したステージ設定の更新

これらのステップでは、すでに API をステージにデプロイしていることを前提としています。

1. https://console.aws.amazon.com/apigateway で API Gateway コンソールにサインインします。
2. REST API を選択します。
3. メインナビゲーションペインで、API の下にある [ステージ] を選択します。
4. [Stages] (ステージ) ペインで、ステージの名前を選択します。
5. [ステージの詳細] セクションで、[編集] を選択します。
6. (オプション) [ステージの説明] で、説明を編集します。

7. [その他の設定] で、以下の設定を変更します。

キャッシュ設定

ステージの API キャッシュを有効にするには、[API キャッシュをプロビジョニング] をオンにします。次に、[デフォルトのメソッドレベルのキャッシュ]、[キャッシュキャパシティ]、[キャッシュデータを暗号化]、[キャッシュの有効期限 (TTL)]、およびキーごとのキャッシュの無効化の要件を設定します。

デフォルトのメソッドレベルのキャッシュをオンにするか、特定のメソッドについてメソッドレベルのキャッシュをオンにするまで、キャッシュはアクティブになりません。

キャッシュ設定の詳細については、「API キャッシュを有効にして応答性を強化する」を参照してください。

> **Note**
>
> API ステージの API キャッシュを有効にすると、AWS アカウントに対して API キャッシュの使用料金が発生することがあります。キャッシュは AWS 無料利用枠の対象ではありません。

スロットリング設定

この API に関連付けられたすべてのメソッドに対してステージレベルのスロットリング目標を設定するには、[スロットリング] をオンにします。

[Rate] (レート) に目標レートを入力します。これは、トークンバケットにトークンを追加するレート (秒あたりのリクエスト数) です。ステージレベルのレートは、アカウントレベルのレート (REST API の設定および実行に関する API Gateway クォータ で指定) を超えないものとします。

[バースト] に目標バーストレートを入力します。バーストレートとは、トークンバケットの容量です。これにより、目標レートよりも多くのリクエストが一定の期間にわたって許可されます。このステージレベルのバーストレートは、アカウントレベルのバーストレート (REST API の設定および実行に関する API Gateway クォータ で指定) を超えないものとします。

> **Note**
>
> スロットリングレートはハードリミットではなく、ベストエフォートベースで適用されます。場合によっては、クライアントは設定されている目標を超えることがあります。コストの管理や API へのアクセスのブロックを行う際にスロットリングに依存しないでください。AWS Budgets を使用してコストをモニタリングすること、および AWS WAF を使用して API リクエストを管理することを検討してください。

ファイアウォールと証明書の設定

AWS WAF ウェブ ACL をステージに関連付けるには、[ウェブ ACL] ドロップダウンリストからウェブ ACL を選択します。必要に応じて、[Block API Request if WebACL cannot be evaluated (Fail- Close) (WebACL を評価できない場合は API リクエストをブロックする (フェイルクローズ))] を選択します。

ステージのクライアント証明書を選択するには、[クライアント証明書] ドロップダウンメニューから証明書を選択します。

8. [Save] を選択します。
9. この API Gateway API のこのステージに関連付けられているすべてのメソッドで Amazon CloudWatch Logs を有効にするには、[ログとトレース] セクションで [編集] を選択します。

> **Note**
>
> CloudWatch Logs を有効にするには、ユーザーの代わりに API Gateway が CloudWatch Logs に情報を書き込むことを可能にする IAM ロールの ARN も指定する必要があります。そのためには、[API] メインナビゲーションペインから [設定] を選択します。次に、[CloudWatch ログロール] に IAM ロールの ARN を入力します。
> 一般的なアプリケーションのシナリオでは、IAM ロールは次のアクセスポリシーステートメントを含む `AmazonAPIGatewayPushToCloudWatchLogs` のマネージドポリシーをアタッチできます。

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

```
            "Action": [
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:DescribeLogGroups",
                "logs:DescribeLogStreams",
                "logs:PutLogEvents",
                "logs:GetLogEvents",
                "logs:FilterLogEvents"
            ],
            "Resource": "*"
        }
    ]
}
```

IAM ロールには、以下の信頼関係ステートメントも含まれている必要があります。

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

Amazon CloudWatch の詳細については、Amazon CloudWatch ユーザーガイドを参照してください。

10. [CloudWatch Logs] ドロップダウンメニューからログ記録レベルを選択します。ログ記録レベルは、以下のとおりです。

    - オフ — この段階ではログ記録はオンになっていません。
    - エラーのみ — ログ記録はエラーに対してのみ有効になっています。
    - エラーと情報ログ — ログ記録はすべてのイベントに対して有効になっています。

- リクエストとレスポンスの完全なログ — 詳細なログ記録がすべてのイベントに対して有効になっています。このログは API のトラブルシューティングに役立ちますが、機密データが記録される可能性があります。

> **Note**
>
> 本稼働用 API には、[リクエストとレスポンスの完全なログ] を有効にしないことをお勧めします。

11. API Gateway から CloudWatch に `API calls`、`Latency`、`Integration latency`、`400 errors`、`500 errors` の API メトリクスをレポートするには、[詳細なメトリクス] を選択します。CloudWatch の詳細については、「Amazon CloudWatch ユーザーガイド」の「基本モニタリングと詳細モニタリング」を参照してください。

> **Important**
>
> アカウントではメソッドレベルの CloudWatch メトリクスへのアクセスに対して課金されますが、API レベルまたはステージレベルのメトリクスでは課金されません。

12. 送信先へのアクセスログを有効にするには、[カスタムのアクセスログ] をオンにします。
13. [アクセスログの送信先 ARN] に、ロググループまたは Firehose ストリームの ARN を入力します。

    Firehose の ARN 形式は `arn:aws:firehose:`*`{region}`*`:`*`{account-id}`*`:deliverystream/amazon-apigateway-`*`{your-stream-name}`* です。Firehose ストリームの名前は `amazon-apigateway-`*`{your-stream-name}`* にする必要があります。
14. [ログの形式] にログの形式を入力します。ログ形式の例について詳しくは、「the section called “API Gateway での CloudWatch によるログの形式”」を参照してください。
15. API ステージで AWS X-Ray トレースを有効にするには、[X-Ray トレース] を選択します。詳細については、「X-Ray を使用した REST API へのユーザーリクエストのトレース」を参照してください。
16. [Save changes] (変更の保存) をクリックします。API を再デプロイして新しい設定を有効にします。

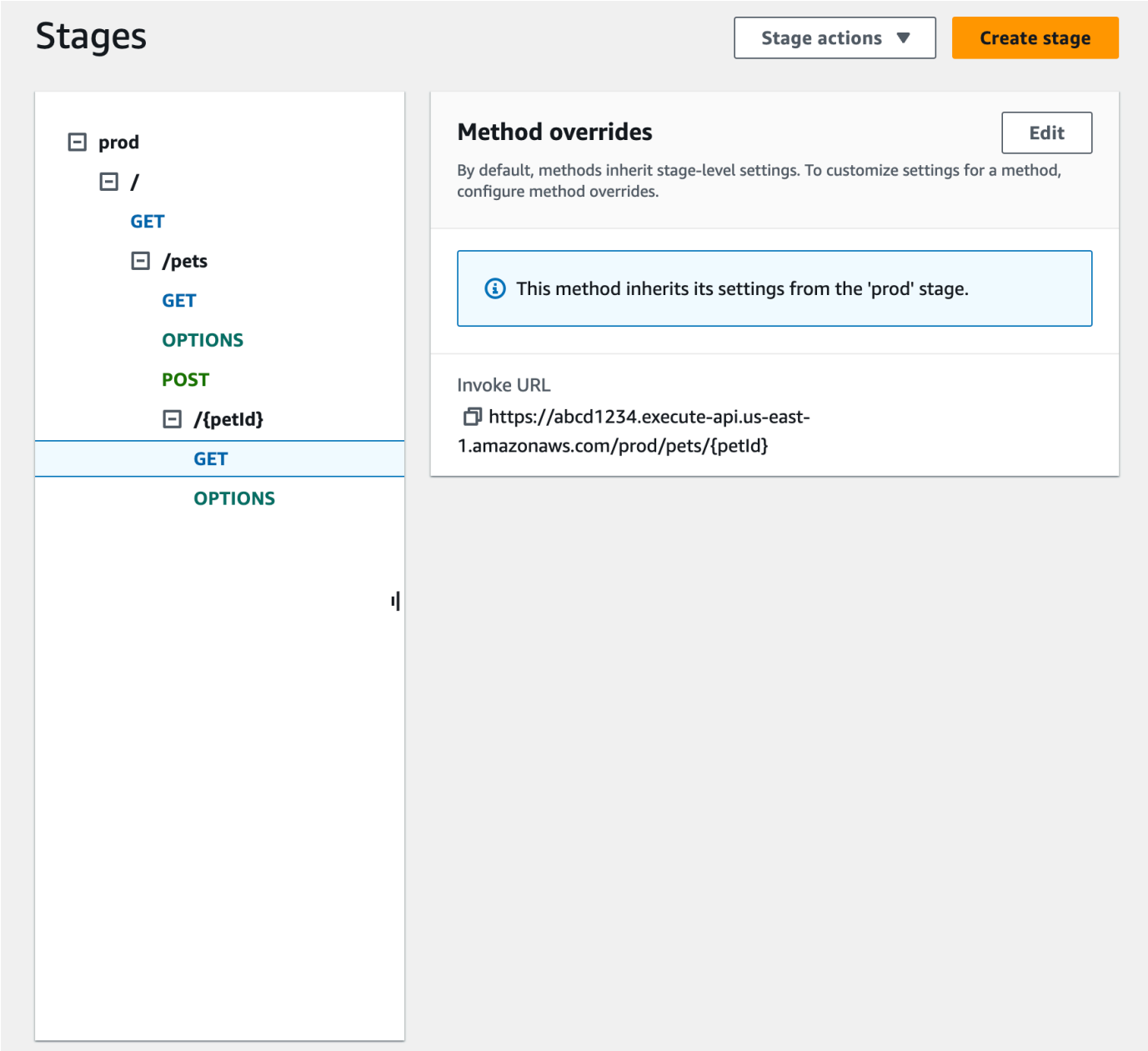
ステージレベルの設定のオーバーライド

以下の有効になっているステージレベルの設定をオーバーライドできます。オプションによっては、AWS アカウントに追加料金がかかる場合があります。

API Gateway コンソールを使用したステージレベルの設定のオーバーライド

API Gateway コンソールを使用してステージレベルの設定をオーバーライドするには

1. メソッドのオーバーライドを設定するには、セカンダリナビゲーションペインでステージを展開し、メソッドを選択します。



2. 次に、[メソッドオーバーライド] で [編集] を選択します。
3. メソッドレベルの CloudWatch 設定を有効にするには、CloudWatch Logs でログ記録レベルを選択します。
4. メソッドレベルの詳細メトリクスを有効にするには、[詳細なメトリクス] を選択します。アカウントではメソッドレベルの CloudWatch メトリクスへのアクセスに対して課金されますが、API レベルまたはステージレベルのメトリクスでは課金されません。
5. メソッドレベルのスロットリングを有効にするには、[スロットリング] を選択します。適切なメソッドレベルのオプションを入力します。スロットリングの詳細については、「the section called “スロットリング”」を参照してください。
6. メソッドレベルのキャッシュを設定するには、[メソッドキャッシュを有効にする] を選択します。[ステージの詳細] でデフォルトのメソッドレベルのキャッシュ設定を変更しても、この設定には影響しません。
7. [Save] を選択します。

ステージを削除する

ステージが不要になったら、それを削除して未使用のリソースに対する請求を避けることができます。次の手順は、API Gateway コンソールを使用してステージを削除する方法を示しています。

> **Warning**
>
> ステージを削除すると、対応する API の一部または全部を API 発信者が使用できなくなる場合があります。ステージの削除は元に戻すことができません。ただし、ステージを再作成して同じデプロイに関連付けることができます。

API Gateway コンソールを使用したステージの削除

1. https://console.aws.amazon.com/apigateway で API Gateway コンソールにサインインします。
2. REST API を選択します。
3. メインナビゲーションペインで、[ステージ] を選択します。
4. [ステージ] ペインで、削除するステージを選択してから [ステージアクション]、[ステージの削除] を選択します。
5. プロンプトが表示されたら、「**`confirm`**」と入力し、[削除] を選択します。

API Gateway で API ステージのタグをセットアップする

API Gateway では、API ステージにタグを追加したり、ステージからタグを削除したり、タグを表示したりすることができます。これを行うには、API Gateway コンソール、AWS CLI/SDK、または API Gateway REST API を使用できます。

ステージは、その親 REST API からタグを継承することもできます。詳細については、「the section called “Amazon API Gateway V1 API でのタグの継承”」を参照してください。

API Gateway リソースのタグ付けの詳細については、「タグ付け」を参照してください。

トピック

- API Gateway コンソールを使用した API ステージのタグのセットアップ
- AWS CLI を使用して API ステージのタグを設定する
- API Gateway REST API を使用した API ステージのタグのセットアップ

API Gateway コンソールを使用した API ステージのタグのセットアップ

次の手順では、API ステージのタグを設定する方法を説明します。

API Gateway コンソールを使用して API ステージのタグを設定するには

1. [API Gateway コンソール] にサインインします。
2. 既存の API を選択するか、リソース、メソッド、および対応する統合を含む新しい API を作成します。
3. ステージを選択するか、新しいステージに API をデプロイします。
4. メインナビゲーションペインで、[ステージ] を選択します。
5. [タグ] タブを選択します。タブを表示するには、右矢印ボタンを選択する必要がある場合があります。
6. [Manage tags (タグの管理)] を選択します。
7. [タグエディター] で、[新しいタグの追加] を選択します。[キー] フィールドにタグキー (`Department` など) を入力し、[値] フィールドにタグ値 (`Sales` など) を入力します。[保存] を選択してタグを保存します。
8. 必要に応じて、ステップ 5 を繰り返して API ステージにさらにタグを追加します。ステージあたりのタグの最大数は 50 です。
9. 既存のステージからタグを削除するには、[削除] を選択します。

10. API が API Gateway コンソールで既にデプロイされているという場合は、それを再度デプロイして変更を有効にする必要があります。

AWS CLI を使用して API ステージのタグを設定する

AWS CLI で create-stage コマンドまたは tag-resource コマンドを使用して API ステージのタグを設定できます。API ステージから 1 つ以上のタグを削除するには、untag-resource コマンドを使用できます。

次の例では、test ステージの作成時にタグを追加します。

```
aws apigateway create-stage --rest-api-id abc1234 --stage-name test --description
 'Testing stage' --deployment-id efg456 --tag Department=Sales
```

次の例では、prod ステージにタグを追加します。

```
aws apigateway tag-resource --resource-arn arn:aws:apigateway:us-east-2::/
restapis/abc123/stages/prod --tags Department=Sales
```

次の例では、test ステージから Department=Sales タグを削除します。

```
aws apigateway untag-resource --resource-arn arn:aws:apigateway:us-east-2::/
restapis/abc123/stages/test --tag-keys Department
```

API Gateway REST API を使用した API ステージのタグのセットアップ

API Gateway REST API を使用して API ステージのタグを設定するには、次のいずれかの操作を行います。

- API ステージにタグ付けするには tags:tag を呼び出します。
- API ステージから 1 つまたは複数のタグを削除するには tags:untag を呼び出します。
- 作成する API ステージに 1 つまたは複数のタグを追加するには stage:create を呼び出します。

また、API ステージでタグを記述するには tags:get を呼び出します。

API ステージのタグ付け

ステージ (m5zr3vnks7) に API (test) をデプロイした後は、tags:tag を呼び出すことで、ステージをタグ付けします。必須のステージである Amazon リソースネーム (ARN) (arn:aws:apigateway:us-east-1::/restapis/m5zr3vnks7/stages/test) は URL エンコードされている必要があります (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"
  }
}
```

前回のリクエストを使用して既存のタグを新しい値で更新することもできます。

stage:create を呼び出してステージを作成するときに、ステージにタグ付けできます。

```
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"
  }
}
```

API ステージのタグ解除

ステージから Department タグを削除するには、tags:untag を呼び出します。

```
DELETE /tags/arn%3Aaws%3Aapigateway%3Aus-east-1%3A%3A%2Frestapis%2Fm5zr3vnks7%2Fstages
%2Ftest?tagKeys=Department
Host: apigateway.us-east-1.amazonaws.com
Authorization: ...
```

複数のタグを削除するには、クエリ式でタグキーのカンマ区切りのリストを使用します (例: ?tagKeys=Department,Division,…)。

API ステージのタグを説明する

特定のステージで既存のタグを記述するには、tags:get を呼び出します。

```
GET /tags/arn%3Aaws%3Aapigateway%3Aus-east-1%3A%3A%2Frestapis%2Fm5zr3vnks7%2Fstages
%2Ftags
Host: apigateway.us-east-1.amazonaws.com
Authorization: ...
```

正常に終了すると、レスポンスは以下のようになります。

```
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"
    }
}
```

REST API デプロイのステージ変数のセットアップ

ステージ変数は、REST API のデプロイステージと関連付けられた設定属性として定義できる名前と値のペアです。環境変数と同様に機能し、API のセットアップやマッピングテンプレートで使用できます。

たとえば、ステージ設定でステージ変数を定義し、REST API のメソッド用に、HTTP 統合の URL 文字列としてその値を設定できます。後で、API のセットアップから関連するステージの変数名を使用して URL 文字列を参照できます。これによって、ステージ変数の値を対応する URL にリセットすることで、各ステージで異なるエンドポイントとともに同じ API セットアップを使用することができます。

また、マッピングテンプレートでステージ変数にアクセスしたり、AWS Lambda または HTTP バックエンドに設定パラメータを渡したりできます。

マッピングテンプレートの詳細については、「API Gateway マッピングテンプレートとアクセスのログ記録の変数リファレンス」を参照してください。

> **Note**
>
> ステージ変数は、認証情報などの機密データに使用されることを意図していません。機密データを統合に渡すには、AWS Lambda オーソライザーを使用します。Lambda オーソライザーの出力では、機密データを統合に渡すことができます。詳細については、「the section called “API Gateway Lambda オーソライザーからの出力”」を参照してください。

ユースケース

API Gateway のデプロイステージでは、アルファ、ベータ、本番稼働など、各 API 用の複数のリリースステージを管理できます。ステージ変数を使用することで、異なるバックエンドのエンドポイントとやり取りするよう API デプロイステージを設定できます。

たとえば、API は HTTP プロキシとして GET リクエストをバックエンドウェブホスト (`http://example.com` など) に渡すことができます。この場合、バックエンドウェブホストはステージ変

数で設定されるため、デベロッパーが本番稼働エンドポイントを呼び出した場合、API Gateway は example.com を呼び出します。ベータエンドポイントを呼び出す場合、API Gateway は、ベータステージ用にステージ変数で設定された値を使用し、別のウェブホスト (例: `beta.example.com`) を呼び出します。同様に、ステージ変数を使用して、API で各ステージに別の AWS Lambda 関数名を指定することができます。

また、ステージ変数を使用して、マッピングテンプレートを通じて Lambda 関数に設定パラメータを渡すこともできます。たとえば、API で複数のステージ用に同じ Lambda 関数を再利用するが、呼び出しているステージによって、関数が別の Amazon DynamoDB テーブルからデータを読み取るようにしたい場合があります。Lambda 関数のリクエストを生成するマッピングテンプレートで、ステージ変数を使用してテーブル名を Lambda に渡すことができます。

例

ステージ変数を使用して HTTP 統合エンドポイントをカスタマイズするには、最初に指定された名前 (**`url`** など) のステージ変数を設定し、次に値 (**`example.com`** など) を割り当てます。次に、メソッド設定から HTTP プロキシ統合をセットアップします。エンドポイントの URL を入力する代わりに、ステージ変数の値、**`http://${stageVariables.url}`** を使用するように API Gateway に指示できます 。この値により、API が実行中のステージに基づいて、ランタイムにステージ変数 `${}` を置き換えるよう API Gateway が指示されます。

ステージ変数は、認証情報フィールドに Lambda 関数名、AWS サービスプロキシのパス、または AWS ロール ARN を指定する方法と同様に参照できます。

ステージ変数値として Lambda 関数名を指定する場合は、その Lambda 関数に対するアクセス許可を手動で設定する必要があります。API Gateway コンソールで Lambda 関数を指定すると、AWS CLI コマンドがポップアップ表示され、適切なアクセス許可を設定できるようになります。AWS Command Line Interface (AWS CLI) を使用してこれを行うこともできます。

```
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
```

Amazon API Gateway コンソールを使用したステージ変数の設定

このチュートリアルでは、Amazon API Gateway コンソールを使用してサンプル API の 2 つのデプロイステージのステージ変数を設定する方法について説明します。開始する前に、以下の前提条件を満たしていることを確認します。

- API が API Gateway で使用可能であることが必要です。「」の手順に従いますAPI Gateway での REST API の開発
- API は少なくとも 1 度はデプロイする必要があります。「」の手順に従いますAmazon API Gateway での REST API のデプロイ
- デプロイされた API の最初のステージを作成済みである必要があります。「」の手順に従いますす新しいステージを作成する

API Gateway コンソールを使用してステージ変数を宣言するには

1. https://console.aws.amazon.com/apigateway で API Gateway コンソールにサインインします。
2. API を作成し、API のルートリソースで GET メソッドを作成します。統合タイプを HTTP に設定し、[エンドポイント URL] を **`http://${stageVariables.url}`** に設定します。
3. **`beta`** という名前の新しいステージに API をデプロイします。
4. メインナビゲーションペインで、[ステージ] を選択してから、beta ステージを選択します。
5. [ステージ変数] タブで [編集] を選択します。
6. [ステージ変数を追加] を選択します。
7. [Name (名前)] に **`url`** と入力します。[値] に「**`httpbin.org/get`**」と入力します。
8. [ステージ変数の追加] を選択し、次の操作を行います。

   [Name (名前)] に **`stageName`** と入力します。[値] に「**`beta`**」と入力します。
9. [ステージ変数の追加] を選択し、次の操作を行います。

   [Name (名前)] に **`function`** と入力します。[値] に「**`HelloWorld`**」と入力します。

   > **Note**
   >
   > Lambda 関数をステージ変数の値に設定するときは、関数のローカル名を使用します。エイリアスまたはバージョン仕様は、**`HelloWorld`**、**`HelloWorld:1`**、**`HelloWorld:alpha`** のように含めます。関数の ARN を使用しないでください (例: **`arn:aws:lambda:us-east-1:123456789012:function:HelloWorld`**)。API Gateway コンソールは、Lambda 関数のステージ変数値が非修飾関数名であると想定し、指定されたステージ変数を ARN に展開します。
10. [Save] を選択します。

11. 次に、2 つ目のステージを作成します。[ステージ] ナビゲーションペインから、[ステージの作成] を選択します。[Stage name (ステージ名)] に **prod** と入力します。[デプロイメント] から最近のデプロイを選択し、[ステージの作成] を選択します。
12. beta ステージと同じように、3 つのステージ変数 (url、stageName、function) をそれぞれ異なる値 (**petstore-demo-endpoint.execute-api.com/petstore/pets**、**prod**、**HelloEveryone**) に設定します。

    ステージ変数の使用方法については、「the section called “ステージ変数の使用 ”」を参照してください。

Amazon API Gateway のステージ変数の使用

API Gateway ステージ変数を使用して、さまざまな API デプロイステージの HTTP と Lambda バックエンドにアクセスできます。ステージ変数を使用して、ステージ固有の設定メタデータをクエリパラメータとして HTTP バックエンドに渡し、入力マッピングテンプレートで生成されるペイロードとして Lambda 関数に渡すこともできます。

前提条件

2 つの異なる HTTP エンドポイント (2 つの異なる Lambda 関数に割り当てられた function ステージ変数、およびステージ固有のメタデータを含む stageName ステージ変数) に設定された url ステージ変数を持つ 2 つのステージを作成する必要があります。

ステージ変数を使用し、API を通じて HTTP エンドポイントにアクセスする

1. [ステージ] ナビゲーションペインで、[beta] を選択します。[ステージの詳細] で、コピーアイコンを選択して API の呼び出し URL をコピーし、Web ブラウザに API の呼び出し URL を入力します。これにより、API のルートリソースで beta ステージの `GET` リクエストが開始されます。

> **Note**
>
> [Invoke URL] リンクは、beta ステージの API のルートリソースを指します。Web ブラウザに URL を入力すると、ルートリソースで beta ステージの `GET` メソッドが呼び出されます。ルートリソースそのものではなく子リソースでメソッドが定義されている場合は、Web ブラウザに URL を入力すると、`{"message":"Missing Authentication Token"}` エラーレスポンスが返されます。この場合、特定の子リソースの名前を [呼び出し URL] リンクに追加する必要があります。

2. beta ステージの GET リクエストから取得するレスポンスを次に示します。また、ブラウザを使用し、http://httpbin.org/get に移動して結果を確認することもできます。この値は beta ステージの `url` 変数に割り当てられました。2 つのレスポンスは同一です。
3. [ステージ] ナビゲーションペインで、[prod] ステージを選択します。[ステージの詳細] で、コピーアイコンを選択して API の呼び出し URL をコピーし、Web ブラウザに API の呼び出し URL を入力します。これにより、API のルートリソースで prod ステージの GET リクエストが開始されます。
4. prod ステージの GET リクエストから取得するレスポンスを次に示します。ブラウザを使用し、http://petstore-demo-endpoint.execute-api.com/petstore/pets に移動して結果を確認できます。この値は prod ステージの `url` 変数に割り当てられました。2 つのレスポンスは同一です。

クエリパラメータ式でステージ変数を使用してステージ固有のメタデータを HTTP バックエンドに渡す

この手順では、クエリパラメータ式でステージ変数値を使用して、ステージ固有のメタデータを HTTP バックエンドに渡す方法について説明します。「`stageName`」で宣言された Amazon API Gateway コンソールを使用したステージ変数の設定 ステージ変数を使用します。

1. [リソース] ナビゲーションペインで、GET メソッドを選択します。

   メソッドの URL にクエリ文字列パラメータを追加するには、[メソッドリクエスト] タブを選択し、[メソッドリクエストの設定] セクションで、[編集] を選択します。
2. [URL クエリ文字列パラメータ] を選択してから、次の操作を行います。

   a. [クエリ文字列の追加] を選択します。

   b. [名前] に**`stageName`**と入力します。

   c. [必須] と [キャッシュ] はオフのままにしておきます。
3. [Save] を選択します。
4. [統合リクエスト] タブを選択し、[統合リクエスト設定] セクションで [編集] を選択します。
5. [エンドポイント URL] では、以前に定義した URL 値に **`?stageName=${stageVariables.stageName}`** を追加し、エンドポイント URL 全体が **`http://${stageVariables.url}?stageName=${stageVariables.stageName}`** になるようにします。
6. [API をデプロイ] を選択し、beta ステージを選択します。

7. メインナビゲーションペインで、[ステージ] を選択します。[ステージ] ナビゲーションペインで、[beta] を選択します。[ステージの詳細] で、コピーアイコンを選択して API の呼び出し URL をコピーし、Web ブラウザに API の呼び出し URL を入力します。

> **Note**
>
> ここでベータステージを使用するのは、(`url` 変数「http://httpbin.org/get」によって指定される) HTTP エンドポイントがクエリパラメータ式を受け取り、レスポンスで `args` オブジェクトとしてそれらを返すためです。

8. 次のレスポンスが返されます。`beta` ステージ変数に割り当てられた `stageName` は、`stageName` 引数としてバックエンドで渡されることに注意してください。

```
{
  "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"
}
```

ステージ変数を使用して API 経由で Lambda 関数を呼び出す

この手順では、ステージ変数を使用して API のバックエンドとして Lambda 関数を呼び出す方法について説明します。前に宣言した `function` ステージ変数を使用します。詳細については、「Amazon API Gateway コンソールを使用したステージ変数の設定」を参照してください。

1. デフォルトの Node.js ランタイムを使用して、**`HelloWorld`** という名前の Lambda 関数を作成します。コードには次が含まれている必要があります。

```
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...');
```

```
};
```

Lambda 関数の作成方法の詳細については、「REST API コンソール入門」を参照してください。

2. [リソース] ペインで [リソースの作成] を選択し、次の操作を行います。

   a. [リソースパス] には、/を選択します。

   b. [リソース名] に「**lambdav1**」と入力します。

   c. [リソースの作成] を選択します。

3. [/lambdav1] リソースを選択し、[メソッドを作成] を選択します。

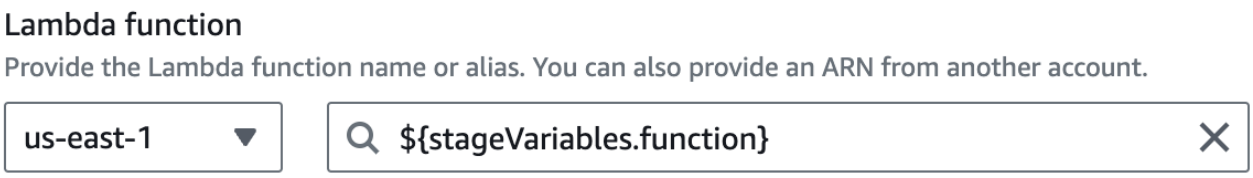
   次に、以下の操作を実行します。

   a. [メソッドタイプ] には、GET を選択します。

   b. [統合タイプ] で、[Lambda 関数] を選択します。

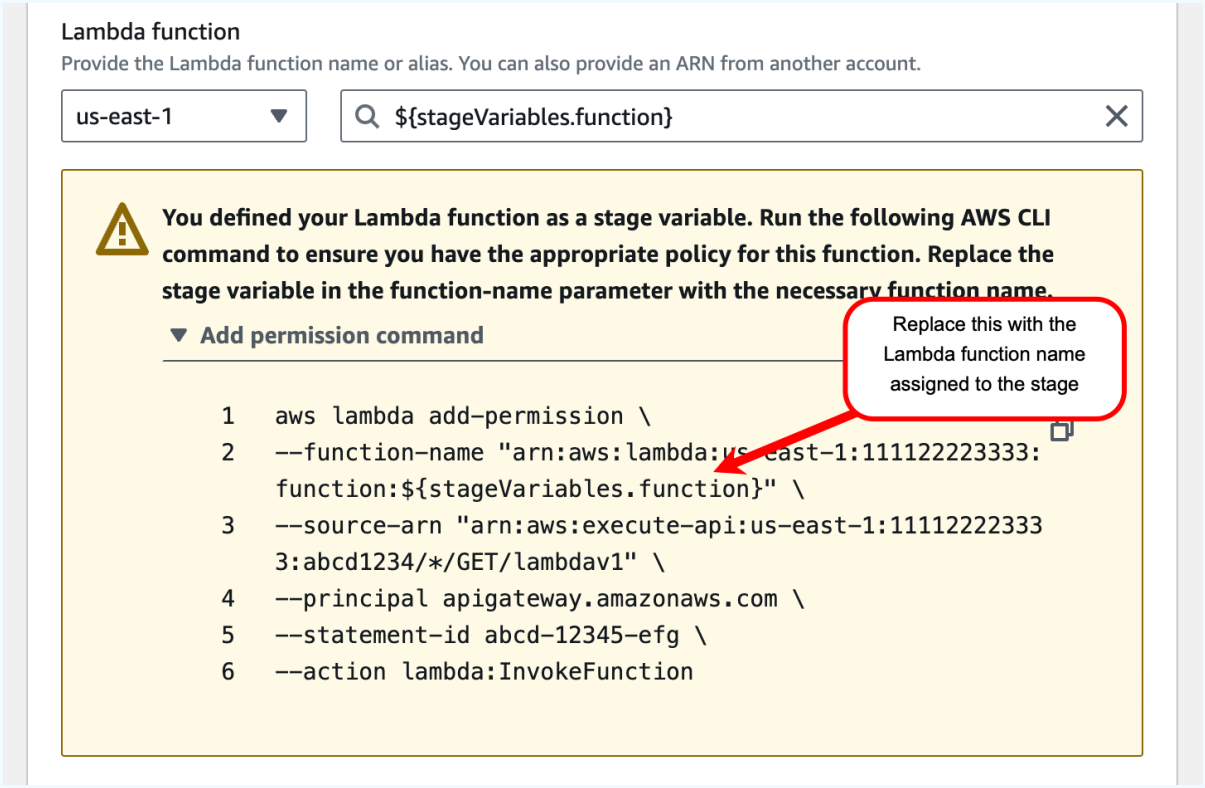
   c. [Lambda プロキシ統合] はオフのままにしておきます。

   d. [Lambda 関数] に「`${stageVariables.function}`」と入力します。



> **Tip**
>
> [アクセス許可を追加] コマンドのプロンプトが表示されたら、AWS CLI コマンドをコピーします。`function` ステージ変数に今後割り当てられる各 Lambda 関数でコマンドを実行します。たとえば、`$stageVariables.function` 値が `HelloWorld` の場合、以下の AWS CLI コマンドを実行します。
>
> ```
> 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
> ```

これを行わなかった場合、メソッドを呼び出すと `500 Internal Server Error` レスポンスが発生します。ステージ変数に割り当てられる Lambda 関数で `${stageVariables.function}` を置き換えます。



e. [メソッドの作成] を選択します。

4. API を、**prod** と **beta** ステージの両方にデプロイします。
5. メインナビゲーションペインで、[ステージ] を選択します。[ステージ] ナビゲーションペインで、[beta] を選択します。[ステージの詳細] で、コピーアイコンを選択して API の呼び出し URL をコピーし、Web ブラウザに API の呼び出し URL を入力します。Enter キーを押す前に **/lambdav1** を URL に追加します。

   次のレスポンスが返されます。

```
"Hello, World! I'm not sure where I'm calling from..."
```

ステージ変数を使用してステージ固有のメタデータを Lambda 関数に渡す

この手順では、ステージ変数を使用して、ステージ固有の設定メタデータを Lambda 関数に渡す方法を示します。POST メソッドと入力マッピングテンプレートを使用して、前に宣言した `stageName` ステージ変数でペイロードを生成します。

1. [/lambdav1] リソースを選択し、[メソッドを作成] を選択します。

   次に、以下の操作を実行します。

   a. [メソッドタイプ] では、POST を選択します。
   b. [統合タイプ] で、[Lambda 関数] を選択します。
   c. [Lambda プロキシ統合] はオフのままにしておきます。
   d. [Lambda 関数] に「`${stageVariables.function}`」と入力します。
   e. [アクセス許可を追加] コマンドのプロンプトが表示されたら、AWS CLI コマンドをコピーします。`function` ステージ変数に今後割り当てられる各 Lambda 関数でコマンドを実行します。
   f. [メソッドの作成] を選択します。
2. [統合リクエスト] タブを選択し、[統合リクエスト設定] セクションで [編集] を選択します。
3. [マッピングテンプレート]、[マッピングテンプレートの追加] の順に選択します。
4. [コンテンツタイプ] に、「**`application/json`**」と入力します。
5. [テンプレート本文] には、次のテンプレートを入力します。

```
#set($inputRoot = $input.path('$'))
{
    "stageName" : "$stageVariables.stageName"
}
```

> **Note**
>
> マッピングテンプレートでは、ステージ変数を引用符で囲んで参照する必要があります (`"$stageVariables.stageName"` または `"${stageVariables.stageName}"` のように)。他の場所では、引用符なしで参照する必要があります (`${stageVariables.function}` のように)。

6. [Save] を選択します。
7. API を、**`beta`** と **`prod`** ステージの両方にデプロイします。

8. REST API クライアントを使用してステージ固有のメタデータを渡すには、以下を実行します。

   a. [ステージ] ナビゲーションペインで、[beta] を選択します。[ステージの詳細] で、コピーアイコンを選択して API の呼び出し URL をコピーし、REST API クライアントの入力フィールドに API の呼び出し URL を入力します。リクエストを送信する前に **/lambdav1** を追加します。

      次のレスポンスが返されます。

      ```
      "Hello, World! I'm calling from the beta stage."
      ```

   b. [ステージ] ナビゲーションペインで、prod を選択します。[ステージの詳細] で、コピーアイコンを選択して API の呼び出し URL をコピーし、REST API クライアントの入力フィールドに API の呼び出し URL を入力します。リクエストを送信する前に **/lambdav1** を追加します。

      次のレスポンスが返されます。

      ```
      "Hello, World! I'm calling from the prod stage."
      ```

9. テスト機能を使用してステージ固有のメタデータを渡すには、以下を実行します。

   a. [リソース] ナビゲーションペインで、[テスト] タブを選択します。タブを表示するには、右矢印ボタンを選択する必要がある場合があります。

   b. [関数] に「**HelloWorld**」と入力します。

   c. StageName に「**beta**」と入力します。

   d. [テスト] を選択します。POST リクエストに本文を追加する必要はありません。

      次のレスポンスが返されます。

      ```
      "Hello, World! I'm calling from the beta stage."
      ```

   e. 前の手順を繰り返して、Prod ステージをテストできます。StageName に「**Prod**」と入力します。

      次のレスポンスが返されます。

      ```
      "Hello, World! I'm calling from the prod stage."
      ```

Amazon API Gateway のステージ変数リファレンス

次のような場合に、API Gateway ステージ変数を使用できます。

パラメータマッピング式

ステージ変数は、API メソッドのリクエストまたはレスポンスヘッダーパラメータ用のパラメータマッピング式で、一部を置き換えることなく使用できます。次の例では、$ を使用したり、{...} で囲むことなくステージ変数を参照しています。

- `stageVariables.<variable_name>`

マッピングテンプレート

ステージ変数は、次の例に示すように、マッピングテンプレートのどこでも使用できます。

- `{ "name" : "$stageVariables.<variable_name>"}`
- `{ "name" : "${stageVariables.<variable_name>}"}`

HTTP 統合 URI

ステージ変数は、次の例に示すように、HTTP 統合 URLの一部として使用できます。

- プロトコルのない完全な URI – `http://${stageVariables.<variable_name>}`
- 完全なドメイン – `http://${stageVariables.<variable_name>}/resource/operation`
- サブドメイン – `http://${stageVariables.<variable_name>}.example.com/resource/operation`
- パス – `http://example.com/${stageVariables.<variable_name>}/bar`
- クエリ文字列 – `http://example.com/foo?q=${stageVariables.<variable_name>}`

AWS 統合 URI

ステージ変数は、次の例に示すように、AWS URI アクションまたはパスコンポーネントの一部として使用できます。

- `arn:aws:apigateway:<region>:<service>:${stageVariables.<variable_name>}`

AWS 統合 URI (Lambda 関数)

ステージ変数は、次の例に示すように、 Lambda 関数名またはバージョン/エイリアスの代わりに使用できます。

- `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**
>
> Lambda 関数にステージ変数を使用するには、関数が API と同じアカウントにある必要があります。ステージ変数は、クロスアカウント Lambda 関数をサポートしていません。

Amazon Cognito ユーザープール

ステージ変数は、COGNITO_USER_POOLS オーソライザーの Amazon Cognito ユーザープールの代わりに使用できます。

- `arn:aws:cognito-idp:<region>:<account_id>:userpool/${stageVariables.<variable_name>}`

AWS 統合認証情報

ステージ変数は、次の例に示すように、AWS ユーザー/ロールの認証情報 ARN の一部として使用できます。

- `arn:aws:iam::<account_id>:${stageVariables.<variable_name>}`

API Gateway の Canary リリースデプロイの設定

Canary リリースは、新しいバージョンの API (および他のソフトウェア) をテスト目的でデプロイするソフトウェア開発戦略であり、ベースバージョンは同じステージで通常のオペレーション用の本稼働リリースとしてデプロイされたままになります。説明のため、このドキュメントではベースバー

ジョンを本稼働リリースとします。これは合理的ですが、テストのために Canary リリースを非本稼働バージョンにも自由に適用できます。

Canary リリースのデプロイでは、すべての API トラフィックはランダムに区切られて事前に設定された比率で本稼働リリースと Canary リリースに送られます。通常、Canary リリースは低い割合の API トラフィックを受け取り、残りは本稼働リリースが受け取ります。更新された API 機能は、Canary を介した API トラフィックのみに認識されます。Canary トラフィックの割合を調整してテストカバレッジやパフォーマンスを最適化できます。

Canary トラフィックを低く保ち、選択をランダムにすることにより、どのような時でもほとんどのユーザーは新しいバージョンの潜在的なバグに悪影響を受けず、また、常に悪影響を受け続けるユーザーもいません。

テストメトリクスが要件を満たしたら、Canary リリースを本稼働リリースに昇格させ、Canary をデプロイから無効にします。これにより、本稼働ステージで新機能が使用可能になります。

トピック

- API Gateway での Canary リリースのデプロイ
- Canary リリースのデプロイを作成する
- Canary リリースの更新
- Canary リリースの昇格
- Canary リリースをオフにする

API Gateway での Canary リリースのデプロイ

API Gateway で、Canary リリースのデプロイでは、API のベースバージョンの本番稼働リリース用のデプロイステージを使用し、その API のベースバージョンに関連した新しいバージョンの Canary リリースへアタッチします。ステージは初期のデプロイと関連付けられ、Canary は後続のデプロイに関連付けられます。最初は、ステージと Canary ポイントの両方が同じ API バージョンを指します。このセクションでは、ステージと本稼働リリースを同じ意味で使用し、また、Canary と Canary リリースを同じ意味で使用します。

Canary リリースで API をデプロイするには、Canary 設定を通常のデプロイのステージに追加して Canary リリースのデプロイを作成します。Canary の設定は基礎となる Canary リリースを表し、ステージはこのデプロイ内の API の本稼働リリースを表します。Canary の設定を追加するには、デプロイステージの `canarySettings` を設定して以下を指定します。

- デプロイ ID。最初はステージで設定されるベースバージョンのデプロイの ID と同じ。

- 0.0 から 100.0 までの間の、Canary リリース用のAPI トラフィックの割合。
- 本番稼働リリースのステージ変数を上書きできるCanary リリースのステージ変数。
- Canary リクエストのステージキャッシュの使用。useStageCache が設定され API キャッシュがステージで有効になっている場合。

Canary リリースが有効にされると、Canary リリースが無効にされて Canary 設定がステージから削除されるまで、デプロイステージは別の非 Canary リリースデプロイに関連付けることができなくなります。

API 実行のログ作成を有効にすると、Canary リリースはすべての Canary リクエストに生成される独自のログとメトリクスを持つようになります。これらは、本番稼働ステージの CloudWatch Logs グループおよび Canary 固有の CloudWatch Logs グループにレポートされます。アクセスログ記録も同じです。個別の Canary 固有のログは新しい API の変更を検証し、変更を受け入れて Canary リリースを本稼働ステージへ昇格させるか、または、変更を破棄して Canary リリースを本稼働ステージから戻すかを決定するのに役立ちます。

本稼働ステージの実行ロググループの名前は `API-Gateway-Execution-Logs/{rest-api-id}/{stage-name}` で、Canary リリースの実行ロググループの名前は `API-Gateway-Execution-Logs/{rest-api-id}/{stage-name}/Canary` です。アクセスログの記録には、新しいロググループを作成する、または、既存のものを選択する必要があります。Canary リリースのアクセスロググループ名には、選択されたロググループ名に `/Canary` サフィックスが付加されます。

Canary リリースはステージキャッシュを使用でき、有効にされると、レスポンスを保存しキャッシュされたエントリを使用して、事前に設定された有効期限 (TTL) 内に次の Canary リクエストへ結果を返します。

Canary リリースのデプロイでは、本稼働リリースと Canary リリースの API は同じバージョンまたは異なるバージョンに関連付けることができます。異なるバージョンに関連付けられている場合、本稼働と Canary リクエストへのレスポンスは別々にキャッシュされ、ステージキャッシュは本稼働と Canary リクエストへ対応する結果を返します。本稼働リリースと Canary リリースが同じデプロイに関連付けられている場合、ステージキャッシュは両方のタイプのリクエストに単一のキャッシュキーを使用し、本稼働リリースと Canary リリースからの同じリクエストに同じレスポンスを返します。

Canary リリースのデプロイを作成する

Canary 設定でデプロイ作成オペレーションへの追加の入力として API をデプロイする場合は、Canary リリースのデプロイを作成します。

また、Canary 設定をステージに追加する `stage:update` リクエストを行うことで、既存の非 Canary デプロイから Canary リリースのデプロイを作成することもできます。

非 Canary リリースのデプロイを作成する場合、存在しないステージ名を指定できます。指定されたステージが存在しない場合、API Gateway がそれを作成します。ただし、Canary リリースのデプロイを作成する場合は、存在しないステージ名を指定することはできません。エラーが表示され、API Gateway は Canary リリースのデプロイを作成しません。

API Gateway コンソール、AWS CLI、または AWS SDK を使用して、API Gateway で Canary リリースデプロイを作成できます。

トピック

- API Gateway コンソールを使用した Canary デプロイメントの作成
- AWS CLI を使用して Canary デプロイを作成する

API Gateway コンソールを使用した Canary デプロイメントの作成

API Gateway コンソールを使用して Canary デプロイのリリースを作成するには、以下の手順に従います。

最初の Canary リリースのデプロイを作成するには

1. [API Gateway コンソール] にサインインします。
2. 既存の REST API を選択するか、新しい REST API を作成します。
3. メインナビゲーションペインで、[リソース] を選択してから [API をデプロイ] を選択します。[API のデプロイ] にある画面の指示に従って API を新しいステージにデプロイします。

   ここまでで、API を本稼働リリースステージにデプロイしました。次に、ステージの Canary 設定を編集し、必要に応じて、キャッシュの有効化、ステージ変数の設定、または API 実行またはアクセスログの設定を行います。
4. API キャッシュを有効にするか、AWS WAF Web ACL をステージに関連付けるには、[ステージの詳細] セクションで [編集] を選択します。詳細については、「the section called “キャッシュ設定”」または「the section called “API Gateway コンソールを使用して AWS WAF ウェブ ACL を API Gateway API ステージに関連付けるには”」を参照してください。
5. 実行またはアクセスログを設定するには、[ログとトレース] セクションで、[編集] を選択して画面の手順に従います。詳細については、「API Gateway での CloudWatch による REST API のログの設定」を参照してください。

6. ステージ変数を設定するには、[ステージ変数] タブを選択し、画面の手順に従ってステージ変数を追加または変更します。詳細については、「the section called “ステージ変数のセットアップ”」を参照してください。
7. [Canary] タブを選択し、[Canary の作成] を選択します。[Canary] タブを表示するには、右矢印ボタンを選択する必要がある場合があります。
8. [Canary 設定] の [Canary] で、canary に転送されるリクエストの割合を入力します。
9. 必要に応じて、[ステージキャッシュ] を選択し、Canary リリースのキャッシュを有効にします。API キャッシュが有効になるまでは Canary リリースのキャッシュは使用できません。
10. 既存のステージ変数を上書きするには、[Canary の上書き] に新しいステージ変数値を入力します。

Canary リリースがデプロイステージで初期化された後、API を変更して変更をテストできます。更新されたバージョンとベースバージョンの両方に同じステージからアクセスできるように、同じステージに API を再デプロイできます。以下のステップでは、その方法について説明します。

最新の API バージョンを Canary にデプロイするには

1. 各 API の更新で、[API のデプロイ] を選択します。
2. [API のデプロイ] で、[デプロイされるステージ] ドロップダウンリストから Canary を含むステージを選択します。
3. (オプション)[デプロイの説明] に、デプロイの説明を入力します。
4. [デプロイ] を選択し、最新の API バージョンを Canary リリースにプッシュします。
5. 必要に応じて、最初の Canary リリースのデプロイを作成するには の説明にあるとおり、ステージ設定、ログ、または Canary 設定を再設定します。

その結果、Canary リリースは最新バージョンを指し、本稼働リリースは API の最初のバージョンを指します。[canarySettings] には新しい [deploymentId] 値ができていますが、ステージには引き続き当初の [deploymentId] があります。内部では、コンソールは [stage:update] を呼び出します。

AWS CLI を使用して Canary デプロイを作成する

まず、2 つのステージ変数で、Canary なしでベースラインデプロイを作成します。

```
aws apigateway create-deployment \
    --variables sv0=val0,sv1=val1 \
    --rest-api-id abcd1234 \
```

```
    --stage-name 'prod' \
```

このコマンドは、次のような Deployment を結果とする表現を返します。

```
{
    "id": "du4ot1",
    "createdDate": 1511379050
}
```

結果的に生じるデプロイ id により API のスナップショット (またはバージョン) が識別されます。

次に、prod ステージの Canary デプロイを作成します。

```
aws apigateway create-deployment  --rest-api-id abcd1234 \
    --canary-settings \
    '{
        "percentTraffic":10.5,
        "useStageCache":false,
        "stageVariableOverrides":{
            "sv1":"val2",
            "sv2":"val3"
        }
    }' \
    --stage-name 'prod'
```

指定されたステージ (prod) が存在しない場合、前述のコマンドはエラーを返します。それ以外の場合は、次のような新しく作成したデプロイリソースの表現を返します。

```
{
    "id": "a6rox0",
    "createdDate": 1511379433
}
```

結果的に生じるデプロイ id により Canary リリースの API のテストバージョンが識別されます。その結果、関連付けられたステージは Canary が有効になっています。get-stage コマンドを呼び出して、このステージの表現を表示できます。次のようになります。

```
aws apigateway get-stage --rest-api-id acbd1234 --stage-name prod
```

以下に示すのは、コマンドの出力としての Stage の表現です。

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

この例では、ベースバージョンの API は {"sv0":val0", "sv1":val1"} のステージ変数を使用し、テストバージョンではステージ変数 {"sv1":val2", "sv2":val3"} を使用します。本稼働リリースと Canary リリースの両方が同じステージ変数 sv1 を使用しますが、それぞれ別の値、val1 および val2 をそれぞれ使用します。ステージ変数 sv0 は本稼働リリースでのみ使用され、ステージ変数 sv2 は Canary リリースでのみ使用されます。

既存の通常のデプロイから Canary リリースのデプロイを作成するには、Canary が有効になるようにステージを更新します。これを示すため、まず通常のデプロイを作成します。

```
aws apigateway create-deployment \
    --variables sv0=val0,sv1=val1 \
    --rest-api-id abcd1234 \
    --stage-name 'beta'
```

このコマンドは、ベースバージョンのデプロイの表現を返します。

```
{
    "id": "cifeiw",
```

```
    "createdDate": 1511380879
}
```

関連付けられたベータステージには Canary 設定がありません。

```
{
    "stageName": "beta",
    "variables": {
        "sv0": "val0",
        "sv1": "val1"
    },
    "cacheClusterEnabled": false,
    "cacheClusterStatus": "NOT_AVAILABLE",
    "deploymentId": "cifeiw",
    "lastUpdatedDate": 1511380879,
    "createdDate": 1511380879,
    "methodSettings": {}
}
```

次に、ステージの Canary をアタッチすることで、新しい Canary リリースのデプロイを作成します。

```
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"
        }]'
```

更新されたステージの表現は次のようになります。

```
{
```

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

既存のバージョンの API で Canary を先ほど有効にしたので、本稼働リリース (Stage) および Canary リリース (canarySettings) 両方が同じデプロイ、つまり、同じバージョンの API (deploymentId) を指します。API を変更して再度このステージにデプロイした後、新しいバージョンは Canary リリースに入り、ベースバージョンは本稼働リリースに残ります。これは、ステージ進化で明らかになります。Canary リリースの deploymentId は新しいデプロイ id に更新され、本稼働リリースの deploymentId は変更されません。

Canary リリースの更新

Canary リリースがデプロイされた後、テストパフォーマンスを最適化するため、Canary トラフィックの割合を調整したり、ステージキャッシュの使用を有効または無効にしたりするかもしれません。また、実行コンテキストが更新されたとき、Canary リリースで使用されているステージ変数を変更することもできます。このような更新を行うには、stage:update オペレーションを呼び出すとき canarySettings に新しい値を入れます。

Canary リリースは、API Gateway コンソール、AWS CLI update-stage コマンド、または AWS SDK を使用して更新できます。

トピック

- API Gateway コンソールを使用して Canary リリースを更新する
- AWS CLI を使用して Canary リリースを更新する

API Gateway コンソールを使用して Canary リリースを更新する

API Gateway コンソールを使用してステージで既存の Canary 設定を更新するには、次の操作を行います。

既存の canary 設定を更新するには

1. API Gateway コンソールにサインインし、既存の REST API を選択します。
2. メインナビゲーションペインで、[ステージ] を選択してから、既存のステージを選択します。
3. [canary] タブを選択してから、[編集] をクリックします。[Canary] タブを表示するには、右矢印ボタンを選択する必要がある場合があります。
4. 0.0 ~ 100.0 の間でパーセント数を調整して、[リクエストの割合 (%)] を更新します。
5. [ステージキャッシュ] チェックボックスを選択または選択解除します。
6. Canary ステージ変数を追加、削除、または変更します。
7. [Save] を選択します。

AWS CLI を使用して Canary リリースを更新する

AWS CLI を使用して Canary を更新するには、`update-stage` コマンドを呼び出します。

Canary のステージキャッシュの使用を有効または無効にするには、次のように `update-stage` コマンドを呼び出します。

```
aws apigateway update-stage \
    --rest-api-id {rest-api-id} \
    --stage-name '{stage-name}'  \
    --patch-operations op=replace,path=/canarySettings/useStageCache,value=true
```

Canary トラフィックの割合を調整するには、update-stage を呼び出し、ステージの `/canarySettings/percentTraffic` の値を置き換えます。

```
aws apigateway update-stage \
    --rest-api-id {rest-api-id} \
    --stage-name '{stage-name}' \
```

```
    --patch-operations op=replace,path=/canarySettings/percentTraffic,value=25.0
```

Canary ステージの変数を追加、置き換え、または削除して Canary ステージの変数を更新するには

```
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"
      }]'
```

上記のすべてを更新するには、オペレーションを 1 つの patch-operations 値にまとめます。

```
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"
      }]'
```

Canary リリースの昇格

Canary リリースを昇格させると、本稼働ステージでテスト中の API バージョンを使用可能にできます。オペレーションには次のタスクが含まれます。

- ステージのデプロイ ID を Canary のデプロイ ID 設定でリセットします。これによりステージの API スナップショットが Canary のスナップショットで更新され、テストバージョンを本稼働リリースにもします。
- Canary のステージ変数でステージ変数を更新する (存在する場合) これによりステージの API 実行コンテキストが Canary のもので更新されます。この更新をしないと、テストバージョンでさまざまなステージ変数やさまざまな既存のステージの値を使用している場合、新しい API バージョンで予期しない結果が生じる場合があります。
- Canary トラフィックの割合を 0.0% に設定します。

Canary リリースが昇格してもステージで Canary は無効になりません。Canary を無効にするには、ステージで Canary 設定を削除する必要があります。

トピック

- API Gateway コンソールを使用して Canary リリースを昇格させる
- AWS CLI を使用して Canary リリースを昇格させる

API Gateway コンソールを使用して Canary リリースを昇格させる

API Gateway コンソールを使用して Canary リリースのデプロイを昇格させるには、以下を実行します。

Canary リリースのデプロイを昇格させるには

1. API Gateway コンソールにサインインし、プライマリナビゲーションペインで既存の API を選択します。
2. メインナビゲーションペインで、[ステージ] を選択してから、既存のステージを選択します。
3. [Canary]タブを選択します。
4. [Canary の昇格] を選択します。

5. 変更内容を確認し、[Canary の昇格] を選択します。

昇格後、本稼働リリースは Canary リリースと同じ API バージョン (deploymentId) を参照します。AWS CLI を使用してこれを検証できます。例については、「the section called “AWS CLI を使用して Canary リリースを昇格させる”」を参照してください。

AWS CLI を使用して Canary リリースを昇格させる

AWS CLI コマンドを使用して Canary リリースを本稼働リリースに昇格させるには、`update-stage` コマンドを呼び出して Canary に関連付けられた `deploymentId` をステージに関連付けられた `deploymentId` にコピーし、Canary トラフィックの割合をゼロ (`0.0`) にリセットして、Canary にバインドされたステージ変数を対応するステージにバインドされた変数にコピーします。

次のようなステージで示された、Canary リリースのデプロイがあるとします。

```
{
    "_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"
```

```
    }
}
```

以下の update-stage リクエストを呼び出して昇格させることができます。

```
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"
      }]'
```

昇格後、ステージは次のようになります。

```
{
    "_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"
    }
}
```

ご覧のように、Canary リリースをステージに昇格させても Canary は無効にならず、デプロイは Canary リリースのデプロイのままになります。通常の本番稼働用デプロイにするには、Canary 設定を無効にする必要があります。Canary リリースのデプロイを無効にする方法については、「the section called “Canary リリースをオフにする”」を参照してください。

Canary リリースをオフにする

Canary リリースのデプロイをオフにするには、`canarySettings` を null に設定してステージから削除します。

API Gateway コンソール、AWS CLI、または AWS SDK を使用して、Canary リリースのデプロイを無効にすることができます。

トピック

- API Gateway コンソールを使用して Canary リリースをオフにする
- AWS CLI を使用して Canary リリースをオフにする

API Gateway コンソールを使用して Canary リリースをオフにする

API Gateway コンソールを使用して Canary リリースのデプロイをオフにするには、以下のステップを使用します。

Canary リリースのデプロイをオフにするには

1. API Gateway コンソールにサインインし、メインナビゲーションペインで既存の API を選択します。
2. メインナビゲーションペインで、[ステージ] を選択してから、既存のステージを選択します。
3. [Canary]タブを選択します。

4. [削除] を選択します。
5. [削除] を選択して Canary を削除することを確認します。

その結果、canarySettings プロパティは null となり、デプロイされるステージから削除されます。AWS CLI を使用してこれを検証できます。例については、「the section called “AWS CLI を使用して Canary リリースをオフにする”」を参照してください。

AWS CLI を使用して Canary リリースをオフにする

AWS CLI を使用して Canary リリースのデプロイをオフにするには、以下のように update-stage コマンドを呼び出します。

```
aws apigateway update-stage \
    --rest-api-id abcd1234 \
    --stage-name canary \
    --patch-operations '[{"op":"remove", "path":"/canarySettings"}]'
```

レスポンスが成功すると、次のようなペイロードを返します。

```
{
    "stageName": "prod",
    "accessLogSettings": {
        ...
    },
    "cacheClusterEnabled": false,
    "cacheClusterStatus": "NOT_AVAILABLE",
    "deploymentId": "nfcn0x",
    "lastUpdatedDate": 1511309280,
    "createdDate": 1511152939,
    "methodSettings": {
        ...
    }
}
```

出力で示されているように、canarySettings プロパティは、Canary が無効なデプロイのステージには存在しなくなりました。

再デプロイが必要な REST API の更新

API の管理は、既存の API セットアップを表示、更新、削除することを意味します。API Gateway コンソール、AWS CLI、SDK、または API Gateway REST API を使用して、API を維持できま

す。API の更新には、特定のリソースプロパティまたは API の構成設定の変更が含まれます。リソースの更新には API の再デプロイが必要です。設定の更新の場合は必要ありません。

以下の表に更新できる API リソースの詳細を示します。

API の再デプロイを必要とする API リソースの更新

リソース	解説
ApiKey	適用可能なプロパティとサポートされているオペレーションについては、「apikey:update」を参照してください。更新では API を再デプロイする必要があります。
オーソライザー	適用可能なプロパティとサポートされているオペレーションについては、「authorizer:update」を参照してください。更新では API を再デプロイする必要があります。
DocumentationPart	適用可能なプロパティとサポートされているオペレーションについては、「documentationpart:update」を参照してください。更新では API を再デプロイする必要があります。
DocumentationVersion	適用可能なプロパティとサポートされているオペレーションについては、「documentationversion:update」を参照してください。更新では API を再デプロイする必要があります。
GatewayResponse	適用可能なプロパティとサポートされているオペレーションについては、「gatewayresponse:update」を参照してください。更新では API を再デプロイする必要があります。
Integration	適用可能なプロパティとサポートされているオペレーションについては、「integration:update」を参照してください。更新では API を再デプロイする必要があります。
IntegrationResponse	適用可能なプロパティとサポートされているオペレーションについては、「integrationresponse:update」を参照してください。更新では API を再デプロイする必要があります。
方法	適用可能なプロパティとサポートされているオペレーションについては、「method:update」を参照してください。更新では API を再デプロイする必要があります。

リソース	解説
MethodResponse	適用可能なプロパティとサポートされているオペレーションについては、「methodresponse:update」を参照してください。更新では API を再デプロイする必要があります。
[Model] (モデル)	適用可能なプロパティとサポートされているオペレーションについては、「model:update」を参照してください。更新では API を再デプロイする必要があります。
RequestValidator	適用可能なプロパティとサポートされているオペレーションについては、「requestvalidator:update」を参照してください。更新では API を再デプロイする必要があります。
[リソース]	適用可能なプロパティとサポートされているオペレーションについては、「resource:update」を参照してください。更新では API を再デプロイする必要があります。
RestApi 数	適用可能なプロパティとサポートされているオペレーションについては、「restapi:update」を参照してください。更新では API を再デプロイする必要があります。
VpcLink	適用可能なプロパティとサポートされているオペレーションについては、「vpclink:update」を参照してください。更新では API を再デプロイする必要があります。

以下の表に更新できる API 設定の詳細を示します。

API の再デプロイを必要としない API 設定の更新

設定	解説
アカウント	適用可能なプロパティとサポートされているオペレーションについては、「account:update」を参照してください。更新では API を再デプロイする必要はありません。
デプロイメント	適用可能なプロパティとサポートされているオペレーションについては、「deployment:update」を参照してください。

設定	解説
DomainName	適用可能なプロパティとサポートされているオペレーションについては、「domainname:update」を参照してください。更新では API を再デプロイする必要はありません。
BasePathMapping	適用可能なプロパティとサポートされているオペレーションについては、「basepathmapping:update」を参照してください。更新では API を再デプロイする必要はありません。
ステージ	適用可能なプロパティとサポートされているオペレーションについては、「stage:update」を参照してください。更新では API を再デプロイする必要はありません。
使用方法	適用可能なプロパティとサポートされているオペレーションについては、「usage:update」を参照してください。更新では API を再デプロイする必要はありません。
UsagePlan	適用可能なプロパティとサポートされているオペレーションについては、「usageplan:update」を参照してください。更新では API を再デプロイする必要はありません。

REST API のカスタムドメイン名を設定する

カスタムドメイン名は、API ユーザーに提供できる、よりシンプルで直感的な URL です。

API のデプロイ後、お客様 (およびその顧客) は、以下の形式のデフォルトのベース URL を使用して API を呼び出すことができます。

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

`api-id` は API Gateway によって生成され、`region` (AWS リージョン) は API の作成時に、`stage` は API のデプロイ時に、ユーザーが指定します。

URL のホスト名の部分 (つまり *api-id*`.execute-api.`*region*`.amazonaws.com`) は API エンドポイントを参照します。デフォルトの API エンドポイントは再呼び出しが難しく、ユーザーフレンドリではありません。

カスタムドメイン名を使用すると、API のホスト名を設定し、代替パスを API にマッピングするための基本パス (myservice など) を選択できます。たとえば、API のよりわかりやすい ベース URL は以下のようになります。

```
https://api.example.com/myservice
```

> **Note**
>
> ただし、リージョン別カスタムドメインは REST API と HTTP API に関連付けることができます。API Gateway バージョン 2 API を使用して、REST API のリージョン別カスタムドメイン名を作成および管理することができます。
> カスタムドメイン名はプライベート API ではサポートされていません。
> REST API がサポートする最低限の TLS バージョンを選択できます。REST API には、TLS 1.2 または TLS 1.0 を選択できます。

ドメイン名を登録する

API のカスタムドメイン名を設定するには、登録されたインターネットドメイン名が必要です。ドメイン名は RFC 1035 仕様に準拠している必要があり、ラベルあたり最大 63オクテット、合計 255 オクテットを含めることができます。必要に応じて、Amazon Route 53 を使用するか、任意のサードパーティーのドメインレジストラを使用して、インターネットドメインを登録できます。API のカスタムドメイン名は、登録されたインターネットドメインのサブドメイン名またはルートドメイン名 (「Zone Apex」など) にすることができます。

カスタムドメイン名が API Gateway で作成されたら、API エンドポイントにマッピングするために DNS プロバイダーのリソースレコードを作成または更新する必要があります。このマッピングを行わないと、カスタムドメイン名宛ての API リクエストが API Gateway に届きません。

> **Note**
>
> カスタムドメイン名は、すべての AWS アカウントにおいてリージョン内で一意である必要があります。
> リージョン間または AWS アカウント間でエッジ最適化カスタムドメイン名を移動するには、既存の CloudFront ディストリビューションを削除して新規に作成する必要があります。新しいカスタムドメイン名が使用できるようになるまで、約 30 分かかる場合があります。詳細については、「CloudFront ディストリビューションの更新」を参照してください。

エッジ最適化カスタムドメイン名

エッジ最適化 API をデプロイすると、API Gateway は Amazon CloudFront ディストリビューションと DNS レコードを設定し、API ドメイン名を CloudFront ディストリビューションドメイン名にマッピングします。API のリクエストは、マッピングされた CloudFront ディストリビューションを介して API Gateway にルーティングされます。

エッジ最適化 API のカスタムドメイン名を作成すると、API Gateway によって CloudFront ディストリビューションがセットアップされます。ただし、カスタムドメイン名を CloudFront ディストリビューションドメイン名にマッピングするには、DNS レコードを設定する必要があります。このマッピングは、カスタムドメイン名宛ての API リクエストに使用され、そのリクエストは、マッピング先の CloudFront ディストリビューションを介して API Gateway にルーティングされます。カスタムドメイン名の証明書を提供する必要もあります。

> **Note**
>
> API Gateway によって作成された CloudFront ディストリビューションは、API Gateway と提携しているリージョン固有のアカウントによって所有されています。CloudWatch Logs でこのような CloudFront ディストリビューションを作成および更新するためのオペレーションをトレースするときは、この API Gateway アカウント ID を使用する必要があります。詳細については、「CloudTrail におけるカスタムドメイン名の作成のログ記録」を参照してください。

エッジ最適化のカスタムドメイン名を設定したり、その証明書を更新したりするには、CloudFront ディストリビューションを更新するためのアクセス許可が必要です。

アクセス権限を付与するには、ユーザー、グループ、またはロールにアクセス許可を追加します。

- AWS IAM Identity Center のユーザーとグループ:

  アクセス許可セットを作成します。「AWS IAM Identity Center ユーザーガイド」の「権限設定を作成する」の手順に従ってください。
- IAM 内で、ID プロバイダーによって管理されているユーザー:

  ID フェデレーションのロールを作成します。詳細については、「IAM ユーザーガイド」の「サードパーティー ID プロバイダー (フェデレーション) 用のロールの作成」を参照してください。
- IAM ユーザー:

- ユーザーが担当できるロールを作成します。手順については、「IAM ユーザーガイド」の「IAM ユーザー用ロールの作成」を参照してください。
- (お奨めできない方法) ポリシーをユーザーに直接アタッチするか、ユーザーをユーザーグループに追加する。詳細については、「IAM ユーザーガイド」の「ユーザー (コンソール) へのアクセス権限の追加」を参照してください。

CloudFront ディストリビューションを更新するには、以下のアクセス許可が必要です。

```
{
    "Version": "2012-10-17",
    "Statement": [
         {
            "Sid": "AllowCloudFrontUpdateDistribution",
            "Effect": "Allow",
            "Action": [
                "cloudfront:updateDistribution"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

API Gateway は、CloudFront ディストリビューションで Server Name Indication (SNI) を利用することで、エッジ最適化のカスタムドメイン名をサポートしています。証明書の必須の形式や証明書の最大キー長など、CloudFront ディストリビューションでのカスタムドメイン名の使用の詳細については、Amazon CloudFront デベロッパーガイドの「代替ドメイン名と HTTPS の使用」を参照してください。

API のホスト名としてカスタムドメイン名を設定する場合、API 所有者はカスタムドメイン名の SSL/TLS 証明書を提供する必要があります。

エッジ最適化のカスタムドメイン名の証明書を提供するには、AWS Certificate Manager (ACM) にリクエストして新しい証明書を ACM で生成するか、us-east-1 リージョン (米国東部 (バージニア北部)) でサードパーティー認証機関から発行された証明書を ACM にインポートできます。

リージョン別カスタムドメイン名

特定のリージョンの API のカスタムドメイン名を作成すると、API Gateway は API のリージョン別ドメイン名を作成します。カスタムドメイン名をリージョン別ドメイン名にマッピングするように、DNS レコードを設定する必要があります。カスタムドメイン名の証明書を提供する必要もあります。

ワイルドカードカスタムドメイン名

ワイルドカードカスタムドメイン名を使用すると、デフォルトのクォータを超えずにほぼ無数のドメイン名をサポートできます。たとえば、各お客様に個別のドメイン名を付けることができます *customername*.api.example.com。

ワイルドカードカスタムドメイン名を制作するためには、ルートドメインの可能なすべてのサブドメインを表すカスタムドメインの最初のサブドメインとして、ワイルドカード (*) を指定します。

たとえば、ワイルドカードカスタムドメイン名として *.example.com を使用すると、a.example.com、b.example.com、c.example.com などのサブドメインが生成され、これらはすべて同じドメインにルーティングされます。

ワイルドカードカスタムドメイン名は、API Gateway の標準のカスタムドメイン名とは異なる設定をサポートします。たとえば、1 つの AWS アカウントで、*.example.com と a.example.com を異なる動作に設定できます。

コンテキスト変数 $context.domainName と $context.domainPrefix コンテキスト変数を使用して、クライアントが API を呼び出すために使用したドメイン名を判断できます。コンテキスト変数の詳細については、「API Gateway マッピングテンプレートとアクセスのログ記録の変数リファレンス」を参照してください。

ワイルドカードカスタムドメイン名を作成するには、DNS または E メール検証方法を使用して検証された証明書を ACM から発行してもらう必要があります。

> **Note**
>
> 別の AWS アカウントで作成済みのカスタムドメイン名と競合するようなワイルドカードカスタムドメイン名を作成することはできません。たとえば、アカウント A で a.example.com が作成済みである場合、アカウント B はワイルドカードカスタムドメイン名として *.example.com を作成できません。
> アカウント A とアカウント B の所有者が同じである場合は、AWS サポートセンターに連絡して例外をリクエストできます。

カスタムドメイン名の証明書

> **Important**
>
> カスタムドメイン名の証明書を指定します。アプリケーションで証明書ピンニング (SSL ピンニングとも呼ばれる) を使用して ACM 証明書を固定すると、AWS が証明書を更新した後にアプリケーションがドメインに接続できないことがあります。詳細については、「AWS Certificate Manager ユーザーガイド」の「証明書のピンニングの問題」を参照してください。

ACM がサポートされているリージョンでカスタムドメイン名の証明書を提供するには、ACM に証明書をリクエストする必要があります。ACM がサポートされていないリージョンで、リージョン別カスタムドメイン名の証明書を提供するには、そのリージョン内の API Gateway に証明書をインポートする必要があります。

SSL/TLS 証明書をインポートするには、カスタムドメイン名の PEM 形式の SSL/TLS 認証本文、そのプライベートキー、およびカスタムドメイン名の証明書チェーンを提供する必要があります。ACM に保存された各証明書は ARN によって識別されます。AWS で管理された証明書をドメイン名で使用するには、その ARN を単に参照します。

ACM を使用すると、API のカスタムドメイン名を簡単に設定して使用できます。特定のドメイン名の証明書を作成 (または証明書をインポート) し、ACM が提供する証明書の ARN を使用して API Gateway でドメイン名を設定します。次に、カスタムドメイン名のベースパスを、デプロイされた API のステージにマッピングします。ACM 発行の証明書により、プライベートキーなど証明書の機密の詳細が漏れる心配はありません。

トピック

- での証明書の準備AWS Certificate Manager
- API Gateway におけるカスタムドメインのセキュリティポリシーの選択
- エッジ最適化カスタムドメイン名の作成
- API Gateway でのリージョン別カスタムドメイン名の設定
- カスタムドメイン名を別の API エンドポイントに移行する
- REST API の API マッピングの使用
- REST API のデフォルトエンドポイントの無効化
- DNS フェイルオーバーのカスタムヘルスチェックの設定

での証明書の準備AWS Certificate Manager

API のカスタムドメイン名を設定する前に、AWS Certificate Managerで SSL/TLS 証明書を準備する必要があります。次のステップでそのやり方を説明します。詳細については、AWS Certificate Manager ユーザーガイドをご参照ください。

> **Note**
>
> API Gateway のエッジ最適化されたカスタムドメイン名を持つ ACM 証明書を使用するには、米国東部 (バージニア北部) (us-east-1) リージョンで証明書をリクエストまたはインポートする必要があります。API Gateway リージョン別カスタムドメイン名については、API と同じリージョンで証明書をリクエストまたはインポートする必要があります。証明書は、信頼された公的認証機関によって署名され、当該カスタムドメイン名を対象としている必要があります。

まず、インターネットドメインを登録します (例: *example*.com)。Amazon Route 53 または認定されているサードパーティーのドメインレジストラを使用できます。そのようなレジストラの一覧については、ICANN のウェブサイトの認定レジストラディレクトリを参照してください。

ドメイン名の SSL/TLS 証明書を ACM で作成またはインポートするには、次のいずれかを行います。

ACM により提供されたドメイン名の証明書をリクエストするには

1. AWS Certificate Manager コンソールにサインインします。
2. [証明書のリクエスト] を選択します。
3. API のカスタムドメイン名 (例: api.example.com) を [ドメイン名] に入力します。
4. 必要に応じて、[この証明書に別の名前を追加] を選択します。
5. [Review and request] を選択します。
6. [Confirm and request] を選択します。
7. リクエストが有効であるためには、ACM が証明書を発行する前に、インターネットドメインの登録された所有者がリクエストに同意する必要があります。

ACM にドメイン名の証明書をインポートするには

1. 証明機関からカスタムドメイン名の PEM エンコード SSL/TLS 証明書を取得します。このような CA のリストの一部については、「Mozilla Included CA List」を参照してください。

   a. 証明書のプライベートキーを生成し、OpenSSL ウェブサイトの OpenSSL ツールキットを使用して出力をファイルに保存します。

   ```
   openssl genrsa -out private-key-file 2048
   ```

   > **Note**
   >
   > Amazon API Gateway は、Amazon CloudFront を活用して、カスタムドメイン名の証明書をサポートします。そのため、カスタムドメイン名の SSL/TLS 証明書の要件と制約は CloudFront によって指定されます。たとえば、パブリックキーの最大サイズは 2048 で、プライベートキーのサイズは 1024、2048、または 4096 とすることができます。パブリックキーのサイズは、使用している認証機関によって決まります。デフォルトの長さと異なるサイズのキーを返すよう認証機関に依頼します。詳細については、「オブジェクトへのセキュアなアクセス」および「署名付き URL と署名付き Cookie」を参照してください。

   b. OpenSSL を使用して、以前に作成されたプライベートキーで証明書署名リクエスト (CSR) を生成します。

   ```
   openssl req -new -sha256 -key private-key-file -out CSR-file
   ```

   c. CSR を認証機関に送信し、結果として生じる証明書を保存します。

   d. 認証機関から証明書チェーンをダウンロードします。

   > **Note**
   >
   > 別の方法でプライベートキーを取得し、キーが暗号化されている場合は、次のコマンドを使用してキーを復号してから、カスタムドメイン名を設定するためにキーを API Gateway に送信できます。

```
openssl pkcs8 -topk8 -inform pem -in MyEncryptedKey.pem -outform pem -
nocrypt -out MyDecryptedKey.pem
```

2. 証明書を AWS Certificate Manager へアップロードします

   a. AWS Certificate Manager コンソールにサインインします。

   b. [Import a certificate] を選択します。

   c. [証明書本文] に、証明機関からの PEM 形式のサーバー証明書の本文を入力するか貼り付けます。このような証明書の省略された例を次に示します。

```
-----BEGIN CERTIFICATE-----
EXAMPLECA+KgAwIBAgIQJ1XxJ8Pl++gOfQtj0IBoqDANBgkqhkiG9w0BAQUFADBB
...
az8Cg1aicxLBQ7EaWIhhgEXAMPLE
-----END CERTIFICATE-----
```

   d. [証明書のプライベートキー] に、PEM 形式の証明書のプライベートキーを入力するか貼り付けます。このようなキーの省略された例を次に示します。

```
-----BEGIN RSA PRIVATE KEY-----
EXAMPLEBAAKCAQEA2Qb3LDHD7StY7Wj6U2/opV6Xu37qUCCkeDWhwpZMYJ9/nETO
...
1qGvJ3u04vdnzaYN5WoyN5LFckrlA71+CszD1CGSqbVDWEXAMPLE
-----END RSA PRIVATE KEY-----
```

   e. [証明書チェーン] に、PEM 形式の中間証明書を (必要に応じてルート証明書も) 空白行なしに連続して入力するか貼り付けます。ルート証明書を含める場合は、証明書チェーンの先頭は中間証明書で、末尾がルート証明書である必要があります。証明機関によって提供された中間証明書を使用します。信頼パスのチェーン内に存在しない中間証明書は含めないでください。次の例で省略された例を示します。

```
-----BEGIN CERTIFICATE-----
EXAMPLECA4ugAwIBAgIQWrYdrB5NogYUx1U9Pamy3DANBgkqhkiG9w0BAQUFADCB
...
8/ifBlIK3se2e4/hEfcEejX/arxbx1BJCHBvlEPNnsdw8EXAMPLE
-----END CERTIFICATE-----
```

   別の例を示します。

```
-----BEGIN CERTIFICATE-----
Intermediate certificate 2
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
Intermediate certificate 1
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
Optional: Root certificate
-----END CERTIFICATE-----
```

f. [Review and import] を選択します。

証明書が正常に作成またはインポートされた後、証明書の ARN をメモします。カスタムドメイン名を設定する際に必要になります。

API Gateway におけるカスタムドメインのセキュリティポリシーの選択

Amazon API Gateway カスタムドメインのセキュリティを強化するために、API Gateway コンソール、AWS CLI、または AWS SDK でセキュリティポリシーを選択できます。

セキュリティポリシーは、API Gateway が提供する TLS の最小バージョンと暗号スイートの事前定義された組み合わせです。TLS バージョン 1.2 または TLS バージョン 1.0 のセキュリティポリシーを選択できます。TLS プロトコルは、クライアントとサーバーの間の改ざんや傍受などのネットワークセキュリティの問題に対処します。クライアントがカスタムドメインを介して API に TLS ハンドシェイクを確立すると、セキュリティポリシーにより、TLS バージョンと暗号スイートのオプションが適用されます。ここで使用するオプションは、クライアントが選択できます。

カスタムドメインの設定では、セキュリティポリシーによって 2 つの設定が決定されます。

- API クライアントとの通信に API Gateway が使用する TLS の最小バージョン
- API クライアントに返すコンテンツを暗号化するために API Gateway が使用する暗号化方式

TLS 1.0 のセキュリティポリシーを選択した場合、セキュリティポリシーは、TLS 1.0、TLS 1.2、TLS 1.3 のトラフィックを受け入れます。TLS 1.2 セキュリティポリシーを選択した場合、セキュリティポリシーは TLS 1.2 と TLS 1.3 のトラフィックを受け入れ、TLS 1.0 のトラフィックを拒否します。

> **Note**
>
> セキュリティポリシーは、カスタムドメインにのみ指定できます。デフォルトエンドポイントを使用する API の場合、API Gateway は次のセキュリティポリシーを使用します。
>
> - エッジ最適化 API の場合: TLS-1-0
> - リージョン API の場合: TLS-1-0
> - プライベート API の場合: TLS-1-2

トピック

カスタムドメインのセキュリティポリシーの指定方法

カスタムドメイン名を作成するときに、セキュリティポリシーを指定します。カスタムドメインの作成方法については、「the section called “エッジ最適化カスタムドメイン名の作成”」または「the section called “リージョン別カスタムドメイン名の設定”」を参照してください。

カスタムドメイン名のセキュリティポリシーを変更するには、カスタムドメイン設定を更新します。カスタムドメイン名の設定を更新するには、AWS Management Console、AWS CLI、または AWS SDK を使用できます。

API Gateway REST API または AWS CLI を使用する場合は、`securityPolicy` パラメータで新しい TLS バージョンとして `TLS_1_0` または `TLS_1_2` を指定します。詳細については、「Amazon API Gateway REST API リファレンス」の「domainname:update」または「AWS CLI リファレンス」の「update-domain-name」を参照してください。

更新オペレーションは、完了するまで数分かかることがあります。

エッジ最適化カスタムドメインで、セキュリティポリシー、TLS プロトコルバージョン、暗号をサポート

次の表では、エッジ最適化カスタムドメイン名に指定できるセキュリティポリシーについて説明します。

セキュリティポリシー	TLS_1_0	TLS_1_2
TLS プロトコル		
TLSv1.3	♦	♦
TLSv1.2	♦	♦
TLSv1.1	♦	
TLSv1	♦	
TLS 暗号		
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	♦	♦

セキュリティポリシー	TLS_1_0	TLS_1_2
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	♦	♦
ECDHE-RSA-AES256-SHA384	♦	♦
ECDHE-RSA-AES256-SHA	♦	
AES128-GCM-SHA256	♦	
AES256-GCM-SHA384	♦	♦
AES128-SHA256	♦	♦
AES256-SHA	♦	
AES128-SHA	♦	
DES-CBC3-SHA	♦	

リージョンカスタムドメインで、セキュリティポリシー、TLS プロトコルバージョン、暗号をサポート

次の表では、リージョンカスタムドメイン名に指定できるセキュリティポリシーについて説明します。

セキュリティポリシー	TLS_1_0	TLS_1_2
TLS プロトコル		
TLSv1.3	♦	♦
TLSv1.2	♦	♦
TLSv1.1	♦	
TLSv1	♦	
TLS 暗号		
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-AES128-SHA	♦	

セキュリティポリシー	TLS_1_0	TLS_1_2
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	♦	♦
AES128-SHA	♦	
AES256-GCM-SHA384	♦	♦
AES256-SHA256	♦	♦
AES256-SHA	♦	

プライベート API で、TLS プロトコルバージョンと暗号をサポート

次の表では、プライベート API でサポートされている TLS プロトコルと暗号について説明します。プライベート API でのセキュリティポリシーの指定はサポートされていません。

セキュリティポリシー	TLS_1_2
TLS プロトコル	

セキュリティポリシー	TLS_1_2
TLSv1.2	♦
TLS 暗号	
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 および RFC の暗号名

OpenSSL と IETF RFC 5246 では、同じ暗号に異なる名前を使用します。以下の表では、各暗号化方式の OpenSSL 名 から RFC 名までを示しています。

OpenSSL の暗号名	RFC の暗号名
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
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

OpenSSL の暗号名	RFC の暗号名
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

HTTP API と WebSocket API に関する情報

HTTP API と WebSocket API の詳細については、「the section called “HTTP API のセキュリティポリシー”」と「the section called “WebSocket API のセキュリティポリシー”」を参照してください。

エッジ最適化カスタムドメイン名の作成

トピック

- API Gateway API のエッジ最適化カスタムドメイン名を設定する
- CloudTrail におけるカスタムドメイン名の作成の口グ記録
- カスタムドメイン名をホスト名として API のベースパスマッピングを設定する
- ACM にインポートされた証明書を更新
- カスタムドメイン名を使用して API を呼び出す

API Gateway API のエッジ最適化カスタムドメイン名を設定する

次の手順では、API Gateway コンソールを使用して API のカスタムドメイン名を作成する方法を説明します。

API Gateway コンソールを使用してカスタムドメイン名を作成するには

1. https://console.aws.amazon.com/apigateway で API Gateway コンソールにサインインします。
2. メインのナビゲーションペインから [Custom Domain Names (カスタムドメイン名)] を選択します。

3. [Create] を選択します。
4. [Domain name (ドメイン名)] には、ドメイン名を入力します。
5. [設定] で、[Edge-optimized (エッジ最適化)] を選択します。
6. TLS の最小バージョンを選択します。
7. ACM 証明書を選択します。

> **Note**
>
> API Gateway のエッジ最適化されたカスタムドメイン名を持つ ACM 証明書を使用するには、`us-east-1` (米国東部 (バージニア北部)) リージョンで証明書をリクエストまたはインポートする必要があります。

8. [ドメイン名の作成] を選択します。
9. カスタムドメイン名を作成したら、コンソールは関連する CloudFront ディストリビューションドメイン名を証明書 ARN と一緒に *`distribution-id`*`.cloudfront.net` の形式で表示します。出力に表示される CloudFront ディストリビューションドメイン名をメモしておきます。次のステップで、DNS でカスタムドメインの CNAME 値または A レコードのエイリアスターゲットを設定するために、これが必要になります。

> **Note**
>
> 新しく作成したカスタムドメイン名は使用できるまでに約 40 分かかります。カスタムドメイン名が初期化されている間に、カスタムドメイン名を関連する CloudFront ディストリビューションドメイン名にマッピングし、カスタムドメイン名のベースパスマッピングを設定するために、DNS レコードのエイリアスを設定できます。

10. 次に、DNS プロバイダーで DNS レコードを設定して、カスタムドメイン名を関連する CloudFront ディストリビューションにマッピングします。Amazon Route 53 の手順については、Amazon Route 53 デベロッパーガイドの「ドメイン名を使用してトラフィックを Amazon API Gateway API にルーティングする 」を参照してください。

    ほとんどの DNS プロバイダーでは、カスタムドメイン名は、CNAME リソースレコードセットとしてホストゾーンに追加されます。CNAME レコードの名前では、[Domain Name (ドメイン名)] に先ほど入力したカスタムドメイン名 (`api.example.com` など) を指定します。CNAME レコードの値では、CloudFront ディストリビューションのドメイン名を指定します。ただし、カスタムドメインが Zone Apex である (つまり、`example.com` ではなく `api.example.com` である) 場合、CNAME レコードは動作しません。一般的に、Zone Apex は組織のルートドメイ

ンとしても知られています。zone apex には、A レコードのエイリアスを使用する必要があります (DNS プロバイダーによってサポートされている場合)。

Route 53 では、カスタムドメイン名の A レコードのエイリアスを作成し、エイリアス先として CloudFront ディストリビューションドメイン名を指定することができます。その結果、カスタムドメイン名が Zone Apex であっても、Route 53 はカスタムドメイン名をルーティングすることができます。詳細については、Amazon Route 53 デベロッパーガイドの「エイリアスリソースレコードセットと非エイリアスリソースレコードセットの選択」を参照してください。

A レコードエイリアスを使用するときも、ドメイン名のマッピングは Route 53 内でのみ起こるため、基盤となる CloudFront ディストリビューションのドメイン名で公開する必要はなくなります。これらの理由により、可能な限りは Route 53 A レコードエイリアスの使用をお勧めします。

API のためのカスタムドメイン名のセットアップには、API Gateway コンソールの使用に加えて、API Gateway REST API、AWS CLI、または AWS SDK の 1 つを使用できます。例として、次の手順では REST API 呼び出しを使用したステップを簡単に示しています。

API Gateway REST API を使用してカスタムドメイン名を設定するには

1. domainname:create を呼び出し、カスタムドメイン名と AWS Certificate Manager に保存してある証明書の ARN を指定します。

   API コールが成功すると、証明書 ARN、および関連する CloudFront ディストリビューション名をペイロードに含む `201 Created` レスポンスを返します。
2. 出力に表示される CloudFront ディストリビューションドメイン名をメモしておきます。次のステップで、DNS でカスタムドメインの CNAME 値または A レコードのエイリアスターゲットを設定するために、これが必要になります。
3. 前の手順を実行して、A レコードエイリアスをセットアップして CloudFront ディストリビューション名にカスタムドメイン名をマッピングします。

この REST API コールのコード例については、「domainname:create」を参照してください。

CloudTrail におけるカスタムドメイン名の作成のログ記録

アカウントによる API Gateway コールのログ記録に CloudTrail が有効になっている場合、API Gateway は、API のカスタムドメイン名が作成または更新されたときに、関連付けられた CloudFront ディストリビューションの更新を記録します。これらの CloudFront ディストリビュー

ションは API Gateway が所有しているため、報告された各 CloudFront ディストリビューションは、API 所有者のアカウント ID ではなく、次のいずれかのリージョン固有の API Gateway アカウント ID によって識別されます。

リージョン	アカウント 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

リージョン	アカウント ID
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

カスタムドメイン名をホスト名として API のベースパスマッピングを設定する

複数の API のホスト名として、単一のカスタムドメイン名を使用できます。カスタムドメイン名でベースパスマッピングを設定することにより、これを実現できます。ベースパスマッピングにより、カスタムドメインの API は、カスタムドメイン名と関連するベースパスの組み合わせによりアクセス可能になります。

たとえば、`PetStore` という名前の API と `PetShop` という名前の別の API を作成し、`api.example.com` のカスタムドメイン名を API Gateway で設定した場合、`PetStore` API の URL を、`https://api.example.com` または `https://api.example.com/myPetStore` として設定できます。`PetStore` API は、カスタムドメイン名 `myPetStore` で、空の文字列または `api.example.com` というベースパスと関連付けられます。同様に、`yourPetShop` API にベースパス `PetShop` を割り当てることができます。URL `https://api.example.com/yourPetShop` は、`PetShop` API のルート URL となります。

API にベースパスを設定する前に、「」のステップを完了してくださいAPI Gateway API のエッジ最適化カスタムドメイン名を設定する

以下の手順では、カスタムドメイン名から API ステージにパスをマップするための API マッピングをセットアップします。

API Gateway コンソールを使用して API マッピングを作成するには

1. API Gateway コンソール (https://console.aws.amazon.com/apigateway) にサインインします。
2. カスタムドメイン名を選択します。
3. [Configure API mappings (API マッピングの設定)] を選択します。
4. [Add new mapping (新しいマッピングを追加)] を選択します。
5. マッピングの API、ステージ、パス (オプション) を指定します。
6. [保存] を選択します。

また、カスタムドメイン名をホスト名として使用して API のベースパスマッピングをセットアップするために、API Gateway REST API、AWS CLI、または AWS SDK の 1 つを呼び出すこともできます。例として、次の手順では REST API 呼び出しを使用したステップを簡単に示しています。

API Gateway REST API を使用して API のベースパスマッピングを設定するには

- 特定のカスタムドメイン名で basepathmapping:create を呼び出し、リクエストペイロードの `basePath`、`restApiId`、およびデプロイ `stage` プロパティを指定します。

  API 呼び出しが成功すると、`201 Created` レスポンスを返します。

REST API コールのコード例については、basepathmapping:create を参照してください。

ACM にインポートされた証明書を更新

ACM は使用する証明書の更新を自動的に処理します。カスタムドメイン名に ACM 使用の証明書を更新する必要はありません。CloudFront はユーザーに代わって処理します。

ただし、証明書を ACM にインポートし、カスタムドメイン名に使用すると、期限切れ前に証明書を更新する必要があります。これにはドメイン名に関する新しいサードパーティーの証明書のインポートも含まれ、既存の証明書は新規更新されます。期限切れの証明書を新しくインポートするときはそのプロセスを繰り返す必要があります。また、ACM にドメイン名に関する新しい証明書を発行するようリクエストもでき、既存のものを ACM 発行の新しい証明書に更新します。その後で、ACM と CloudFront により証明書の更新を自動的に処理するようにできます。新しい ACM 証明書を作成またはインポートするには、指定されたドメイン名に「新しい ACM 証明書をリクエストまたはインポートする」のステップを実行してください。

ドメイン名の証明書のローテーションを行うには、API Gateway コンソール、API Gateway REST API、AWS CLI、または AWS SDK のいずれかを使用できます。

API Gateway コンソールを使って ACM にインポートした、期限切れとなる証明書を更新するには

1. ACM に証明書をリクエストするかインポートします。
2. API Gateway コンソールに戻ります。
3. API Gateway コンソールのナビゲーションペインから [Custom Domain Names (カスタムドメイン名)] を選択します。
4. カスタムドメイン名を選択します。
5. [Edit] を選択します。
6. ドロップダウンリストの [ACM certificate (ACM 証明書)] から希望する証明書を選択します。
7. [保存] を選択し、カスタムドメイン名の証明書の更新を開始します。

> **Note**
>
> プロセスが終了するには約40分かかります。更新完了後、[ACM 証明書] の隣にある対面矢印アイコンを選択すると、オリジナルの証明書に戻せます。

カスタムドメイン名にインポートした証明書をプログラムを使って更新する方法として、API Gateway REST API を使用するステップを簡単に説明します。

API Gateway REST APIを使用してインポートした証明書を更新する

- domainname:update アクションを呼び出し、指定したドメイン名の新しい ACM 証明書の ARN を指定します。

カスタムドメイン名を使用して API を呼び出す

カスタムドメイン名で API を呼び出すことは、正しい URL を使用する場合、デフォルトのドメイン名で API を呼び出すことと同じです。

次の例では、指定されたリージョン (udxjef) で、指定されたカスタムドメイン名 (qf3duz) の 2 つの API (us-east-1 および api.example.com) のデフォルトの URL と対応するカスタム URL を比較し、違いを示します。

デフォルトおよびカスタムドメイン名を持つ API のルート URL

API ID	ステージ	デフォルト URL	基本パス	カスタム URL
udxjef	prod	https://udxjef.execute-api.us-east-1.amazonaws.com/prod	/petstore	https://api.example.com/petstore
udxjef	tst	https://udxjef.execute-api.us-east-1.amazonaws.com/tst	/petdepot	https://api.example.com/petdepot
qf3duz	dev	https://qf3duz.execute-api.us-east-1.amazonaws.com/dev	/bookstore	https://api.example.com/bookstore
qf3duz	tst	https://qf3duz.execute-api.us-east-1.amazonaws.com/tst	/bookstand	https://api.example.com/bookstand

API Gateway では、Server Name Indication (SNI) を使用した、API のカスタムドメイン名がサポートされています。SNI をサポートするブラウザまたはクライアントライブラリを使用して、カスタムドメイン名の API を呼び出すことができます。

API Gateway では、CloudFront ディストリビューションで SNI を実施します。CloudFront でのカスタムドメイン名の使用の詳細については、「Amazon CloudFront カスタム SSL」を参照してください。

API Gateway でのリージョン別カスタムドメイン名の設定

リージョン別 API エンドポイント (AWS リージョン用) のカスタムドメイン名を作成できます。カスタムドメイン名を作成するには、リージョン固有の ACM 証明書を指定する必要があります。カスタムドメイン名の証明書を作成またはアップロードする方法の詳細については、「 での証明書の準備 AWS Certificate Manager」を参照してください。

> **Important**
>
> API Gateway リージョン別カスタムドメイン名については、API と同じリージョンで証明書をリクエストまたはインポートする必要があります。

ACM 証明書でリージョン別カスタムドメイン名を作成または移行する際、アカウント内にすでにロールが存在していない場合、 API Gateway はサービスにリンクされたロールをアカウント内に作成します。サービスにリンクされたロールは、ACM 証明書をリージョン別エンドポイントにアタッチするのに必要です。ロールの名前は AWSServiceRoleForAPIGateway です。また、管理ポリシーの APIGatewayServiceRolePolicy がアタッチされます。サービスにリンクされたロールの詳細な使用方法については、「サービスにリンクされたロールの使用」を参照してください。

> **Important**
>
> DNS レコードを作成し、カスタムドメイン名をリージョン別ドメイン名にポイントする必要があります。これにより、カスタムドメイン名にバインドされるトラフィックが、API のリージョン別ホスト名にルーティングされます。DNS レコードは、CNAME または A タイプにできます。

トピック

- API Gateway コンソールで ACM 証明書を使用してリージョン別カスタムドメイン名を設定する
- AWS CLI で ACM 証明書を使用してリージョン別カスタムドメイン名を設定する

API Gateway コンソールで ACM 証明書を使用してリージョン別カスタムドメイン名を設定する

API Gateway コンソールを使用してリージョン別カスタムドメイン名を設定するには、以下の手順を実行します。

API Gateway コンソールを使用してリージョン別カスタムドメイン名を設定する

1. https://console.aws.amazon.com/apigateway で API Gateway コンソールにサインインします。
2. メインのナビゲーションペインから [Custom Domain Names (カスタムドメイン名)] を選択します。
3. [Create] を選択します。
4. [Domain name (ドメイン名)] には、ドメイン名を入力します。
5. [設定] で、[Regional (リージョン)] を選択します。
6. TLS の最小バージョンを選択します。
7. ACM 証明書を選択します。証明書は API と同じリージョンに存在する必要があります。
8. [Create] を選択します。
9. トラフィックを API Gateway にルーティングするように Route 53 を設定する方法については、Route 53 のドキュメントを参照してください。

以下の手順では、カスタムドメイン名から API ステージにパスをマップするための API マッピングをセットアップします。

API Gateway コンソールを使用して API マッピングを作成するには

1. API Gateway コンソール (https://console.aws.amazon.com/apigateway) にサインインします。
2. カスタムドメイン名を選択します。
3. [Configure API mappings (API マッピングの設定)] を選択します。
4. [Add new mapping (新しいマッピングを追加)] を選択します。
5. マッピングの API、ステージ、パスを指定します。
6. [保存] を選択します。

カスタムドメインの basepath マッピングの設定については、「カスタムドメイン名をホスト名として API のベースパスマッピングを設定する」を参照してください。

AWS CLI で ACM 証明書を使用してリージョン別カスタムドメイン名を設定する

AWS CLI を使用してリージョン別 API のカスタムドメイン名を設定するには、次の手順を実行します。

1. カスタムドメイン名とリージョン別証明書の ARN を指定する create-domain-name を呼び出します。

```
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
```

指定された証明書は us-west-2 リージョンのものであり、この例では基盤となる API も同じリージョンのものであることを前提としていることに注意してください。

成功すると、この呼び出しは以下のような結果を返します。

```
{
    "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 プロパティ値は、リージョン別 API のホスト名を返します。DNS レコードを作成し、カスタムドメイン名をこのリージョン別ドメイン名にポイントする必要があります。これにより、カスタムドメイン名にバインドされるトラフィックが、このリージョン別 API のホスト名にルーティングされます。

2. DNS レコードを作成し、カスタムドメイン名とリージョン別ドメイン名を関連付けます。これにより、カスタムドメイン名にバインドされるリクエストが、API のリージョン別ホスト名にルーティングされます。

3. 基本パスマッピングを追加し、指定のカスタムドメイン名 (0qzs2sy7bh など) のデプロイステージ (test など) で指定の API (regional.example.com など) を公開します。

```
aws apigatewayv2 create-api-mapping \
```

```
    --domain-name 'regional.example.com' \
    --api-mapping-key 'myApi' \
    --api-id 0qzs2sy7bh \
    --stage 'test'
```

その結果、ステージにデプロイされる API のカスタムドメイン名を使用するベース URL は `https://regional.example.com/myAPI` になります。

4. DNS レコードを設定して、リージョン別カスタムドメイン名を指定されたホストゾーン ID のホスト名にマッピングします。まず、リージョン別ドメイン名の DNS レコードを設定するための設定を含む JSON ファイルを作成します。次の例に、カスタムドメイン名の作成時にプロビジョニングされたリージョン別のホスト名 (A) にリージョン別のカスタムドメイン名 (`regional.example.com`) をマッピングする DNS `d-numh1z56v6.execute-api.us-west-2.amazonaws.com` レコードの作成方法を示します。DNSName の HostedZoneId プロパティと AliasTarget プロパティは、カスタムドメイン名の `regionalDomainName` と `regionalHostedZoneId` の値をそれぞれ示しています。また、リージョン別 Route 53 ホストゾーン ID は、「Amazon API Gateway エンドポイントとクォータ」でも取得できます。

```
{
  "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. 次の CLI コマンドを実行します。

```
aws route53 change-resource-record-sets \
    --hosted-zone-id {your-hosted-zone-id} \
    --change-batch file://path/to/your/setup-dns-record.json
```

ここで *{your-hosted-zone-id}* は、アカウントに設定された DNS レコードの Route 53 ホストゾーン ID です。change-batch パラメータ値は、フォルダ (*path/to/your*) 内の JSON ファイル (*setup-dns-record.json*) を指しています。

カスタムドメイン名を別の API エンドポイントに移行する

エッジ最適化のエンドポイントとリージョン別エンドポイントの間で、カスタムドメイン名を移行できます。まず、新しいエンドポイント設定タイプをカスタムドメイン名の既存の endpointConfiguration.types リストに追加します。次に、カスタムドメイン名が、新しくプロビジョンされたエンドポイントを参照するように、DNS レコードを設定します。オプションで最後に、不要なカスタムドメイン名設定データを削除します。

移行を計画するとき、エッジ最適化 API のカスタムドメイン名の場合は、ACM によって提供される必須の証明書は、米国東部 (バージニア北部) リージョン (us-east-1) から取得する必要があります。この証明書はすべての地理的場所に配布されます。ただし、リージョン API の場合、リージョン別ドメイン名の ACM 証明書は、API をホストする同じリージョンから取得する必要があります。us-east-1 リージョンにないエッジ最適化のカスタムドメイン名をリージョン別カスタムドメイン名に移行するには、まず、API のローカルリージョンに新しい ACM 証明書をリクエストする必要があります。

API Gateway でエッジ最適化のカスタムドメイン名とリージョン別カスタムドメイン名との間で移行が完了するまでに最大 60 秒かかることがあります。新しく作成されたエンドポイントがトラフィックを受け入れ可能になるまでの時間を含める場合、移行にかかる時間は DNS レコードの更新時期によっても異なります。

トピック

- AWS CLI を使用してカスタムドメイン名を移行する

AWS CLI を使用してカスタムドメイン名を移行する

AWS CLI を使用して、エッジ最適化エンドポイントからリージョンエンドポイント (またはその逆) にカスタムドメイン名を移行するには、 update-domain-nameコマンドを呼び出して新しいエンドポイントタイプを追加します。必要に応じて、update-domain-name コマンドを呼び出して古いエンドポイントタイプを削除します。

トピック

- エッジ最適化のカスタムドメイン名からリージョンのカスタムドメイン名に移行する

- リージョンのカスタムドメイン名をエッジ最適化のカスタムドメイン名に移行する

エッジ最適化のカスタムドメイン名からリージョンのカスタムドメイン名に移行する

エッジ最適化のカスタムドメイン名をリージョン別カスタムドメイン名に移行するには、以下のように update-domain-name CLI コマンドを呼び出します。

```
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' } \
      ]
```

リージョン別証明書はリージョン別 API と同じリージョンである必要があります。

成功のレスポンスには、200 OK ステータスコードと以下のような本文が含まれます。

```
{
    "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"
}
```

移行されたリージョンのカスタムドメイン名の場合、リージョン別 API のホスト名が regionalDomainName プロパティとして返されます。リージョン別カスタムドメイン名がこのリージョン別ホスト名を参照するように DNS レコードを設定する必要があります。これにより、カスタムドメイン名宛てのトラフィックがリージョン別ホストにルーティングされるようになります。

DNS レコードが設定されたら、update-domain-name の AWS CLI コマンドを呼び出して、エッジ最適化のカスタムドメイン名を削除できます。

```
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'} \
        ]
```

リージョンのカスタムドメイン名をエッジ最適化のカスタムドメイン名に移行する

リージョン別カスタムドメイン名をエッジ最適化のカスタムドメイン名に移行するには、以下のように update-domain-name の AWS CLI コマンドを呼び出します。

```
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' } \
        ]
```

エッジ最適化のドメイン証明書は、us-east-1 リージョンに作成する必要があります。

成功のレスポンスには、200 OK ステータスコードと以下のような本文が含まれます。

```
{
    "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"
}
```

指定したカスタムドメイン名の場合、API Gateway はエッジ最適化 API のホスト名を distributionDomainName プロパティ値として返します。エッジ最適化のカスタムドメイン名がこのディストリビューションドメイン名を参照するように DNS レコードを設定する必要があります。これにより、エッジ最適化のカスタムドメイン名宛てのトラフィックがエッジ最適化の API ホスト名にルーティングされるようになります。

DNS レコードが設定されたら、カスタムドメイン名の REGION エンドポイントタイプを削除できます。

```
aws apigateway update-domain-name \
    --domain-name api.example.com \
    --patch-operations [ \
        {op:'remove', path:'/endpointConfiguration/types', value:'REGIONAL'}, \
        {op:'remove', path:'regionalCertificateArn'} \
      ]
```

このコマンドの結果は以下のような出力になります。この出力には、エッジ最適化のドメイン名の設定データのみが含まれています。

```
{
    "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"
    }
}
```

REST API の API マッピングの使用

API マッピングを使用して、API ステージをカスタムドメイン名に接続します。ドメイン名を作成し、DNS レコードを設定したら、API マッピングを使用して、カスタムドメイン名を使用して API にトラフィックを送信します。

API マッピングは、API、ステージ、およびオプションでマッピングに使用するパスを指定します。たとえば、API の `production` ステージを `https://api.example.com/orders` にマッピングできます。

HTTP API と REST API ステージを同じカスタムドメイン名にマッピングできます。

API マッピングを作成する前に、API、ステージ、およびカスタムドメイン名が必要です。カスタムドメイン名の作成と設定の詳細については、「the section called “リージョン別カスタムドメイン名の設定”」を参照してください。

API リクエストのルーティング

API マッピングは、例えば `orders/v1/items` と `orders/v2/items` のように、複数のレベルで設定できます。

> **Note**
>
> 複数のレベルで API マッピングを設定するには、カスタムドメイン名をリージョン別とし、TLS 1.2 セキュリティポリシーを使用する必要があります。

複数のレベルを持つ API マッピングの場合、API Gateway は、一致するパスが最も長い API マッピングにリクエストをルーティングします。API Gateway は、API マッピング用に設定されたパスだけを考慮し、呼び出す API を選択します。API ルートは考慮しません。リクエストに一致するパスがない場合、API Gateway は空のパス (`none`) にマッピングした API にリクエストを送信します。

複数のレベルの API マッピングを使用するカスタムドメイン名の場合、API Gateway は、一致するプレフィックスが最も長い API マッピングにリクエストをルーティングします。

たとえば、次の API マッピングを持つカスタムドメイン名 `https://api.example.com` を考えてます。

1. API 1 にマッピングされている (`none`)。
2. API 2 にマッピングされている `orders`。
3. API 3 にマッピングされている `orders/v1/items`。
4. API 4 にマッピングされている `orders/v2/items`。
5. API 5 にマッピングされている `orders/v2/items/categories`。

リクエスト	選択した API	説明
https://api.example.com/orders	API 2	リクエストは、この API マッピングと完全に一致します。
https://api.example.com/orders/v1/items	API 3	リクエストは、この API マッピングと完全に一致します。
https://api.example.com/orders/v2/items	API 4	リクエストは、この API マッピングと完全に一致します。
https://api.example.com/orders/v1/items/123	API 3	API Gateway は、最も長い一致パスを持つ API マッピングを選択します。リクエストの最後にある 123 は、選択には影響しません。
https://api.example.com/orders/v2/items/categories/5	API 5	API Gateway は、最も長い一致パスを持つ API マッピングを選択します。
https://api.example.com/customers	API 1	API Gateway は、空のマッピングをキャッチオールとして使用します。
https://api.example.com/ordersandmore	API 2	API Gateway は、一致するプレフィックスが最も長い API マッピングを選択します。単一レベルのマッピングで設定されたカスタムドメイン名の場合 (https://api.example.com/orders とhttps://api.example.com/ のみなど)、API ゲートウェイは、ordersandmore と一致

リクエスト	選択した API	説明
		するパスがないため、API 1 を選択します。

制限事項

- API マッピングでは、カスタムドメイン名とマップされた API が同じ AWS アカウントにある必要があります。
- API マッピングに含めることができるのは、文字、数字、および $-_.+!*'()/ の文字だけです。
- API マッピングのパスの最大文字数は 300 文字です。
- ドメイン名ごとに、複数のレベルで 200 個の API マッピングを設定できます。
- TLS 1.2 セキュリティポリシーでは、HTTP API をリージョン別カスタムドメイン名にだけマッピングできます。
- WebSocket API を HTTP API または REST API と同じカスタムドメイン名にマッピングすることはできません。

API マッピングを作成する

API マッピングを作成するには、最初にカスタムドメイン名、API、およびステージを作成する必要があります。カスタムドメイン名の作成方法については、「the section called “リージョン別カスタムドメイン名の設定”」を参照してください。

例えば、すべてのリソースを作成する AWS Serverless Application Model テンプレートについては、GitHub で「Sessions With SAM」を参照してください。

AWS Management Console

API マッピングを作成するには

1. API Gateway コンソール (https://console.aws.amazon.com/apigateway) にサインインします。
2. [カスタムドメイン名] を選択します。
3. 既に作成したカスタムドメイン名を選択します。
4. [API マッピング] を選択します。

5. [Configure API mappings (API マッピングの設定)] を選択します。
6. [Add new mapping (新しいマッピングを追加)] を選択します。
7. API、Stage、必要に応じて Path を入力します。
8. [保存] を選択します。

AWS CLI

次の AWS CLI コマンドは、API マッピングを作成します。この例では、API Gateway が指定された API およびステージに `api.example.com/v1/orders` に対するリクエストを送信します。

> **Note**
>
> 複数のレベルで API マッピングを作成するには、`apigatewayv2` を使用する必要があります。

```
aws apigatewayv2 create-api-mapping \
    --domain-name api.example.com \
    --api-mapping-key v1/orders \
    --api-id a1b2c3d4 \
    --stage test
```

AWS CloudFormation

次の AWS CloudFormation 例は、API マッピングを作成します。

> **Note**
>
> 複数のレベルで API マッピングを作成するには、`AWS::ApiGatewayV2` を使用する必要があります。

```
MyApiMapping:
  Type: 'AWS::ApiGatewayV2::ApiMapping'
  Properties:
    DomainName: api.example.com
    ApiMappingKey: 'orders/v2/items'
```

```
    ApiId: !Ref MyApi
    Stage: !Ref MyStage
```

REST API のデフォルトエンドポイントの無効化

デフォルトでは、クライアントは、API Gateway が API 用に生成する execute-api エンドポイントを使用して API を呼び出すことができます。クライアントがカスタムドメイン名を使用した場合のみ API にアクセスできるようにするには、デフォルトの execute-api エンドポイントを無効にします。クライアントは引き続きデフォルトのエンドポイントに接続できますが、403 Forbidden ステータスコードを受け取ります。

> **Note**
>
> デフォルトのエンドポイントを無効にすると、API のすべてのステージに影響します。

次の AWS CLI コマンドは、REST API のデフォルトエンドポイントを無効にします。

```
aws apigateway update-rest-api \
    --rest-api-id abcdef123 \
    --patch-operations op=replace,path=/disableExecuteApiEndpoint,value='True'
```

デフォルトのエンドポイントを無効にした後で、変更を有効にするには、API をデプロイする必要があります。

次の AWS CLI コマンドは、デプロイを作成します。

```
aws apigateway create-deployment \
    --rest-api-id abcdef123 \
    --stage-name dev
```

DNS フェイルオーバーのカスタムヘルスチェックの設定

Amazon Route 53 ヘルスチェックを使用して、プライマリ AWS リージョンの API Gateway API からセカンダリリージョンの API Gateway API への DNS フェイルオーバーを制御できます。これは、リージョンの問題が発生した場合の影響を軽減するのに役立ちます。カスタムドメインを使用すると、クライアントが API エンドポイントを変更しなくてもフェイルオーバーを実行できます。

エイリアスレコードに対して [ターゲットのヘルスの評価] を選択した場合、それらのレコードが失敗するのは API Gateway サービスがリージョンで利用できない場合だけです。場合によっては、その時間より前に独自の API Gateway API で中断が発生する可能性があります。DNS フェイルオーバーを直接制御するには、API Gateway API のカスタム Route 53 ヘルスチェックを設定します。この例では、オペレーターが DNS フェイルオーバーを制御するのに役立つ CloudWatch アラームを使用します。フェイルオーバーを設定する際のその他の例や考慮事項については、「Route 53 を使用したディザスタリカバリメカニズムの作成」と「AWS Lambdaと CloudWatch を使用して VPC 内のプライベートリソースで Route 53 のヘルスチェックを実行する」を参照してください。

トピック

前提条件

この手順を完了するには、次のリソースを作成して設定する必要があります。

- 所有するドメイン名。
- 2 つの AWS リージョンにある、そのドメイン名の ACM 証明書。詳細については、the section called “ での証明書の準備AWS Certificate Manager” を参照してください。
- ドメイン名の Route 53 ホストゾーン。詳細については、Amazon Route 53 デベロッパーガイドの「ホストゾーンの使用」を参照してください。

ドメイン名の Route 53 フェイルオーバー DNS レコードを作成する方法の詳細については、「Amazon Route 53 デベロッパーガイド」の「ルーティングポリシーの選択」を参照してください。CloudWatch アラームをモニタリングする方法の詳細については、「Amazon Route 53 デベロッパーガイド」の「CloudWatch アラームのモニタリング」を参照してください。

ステップ 1 : リソースを設定する

この例では、以下のリソースを作成して、ドメイン名の DNS フェイルオーバーを設定します。

- 2 つの AWS リージョンにある API Gateway API

- 2 つの AWS リージョンで同じ名前を持つ API Gateway カスタムドメイン名
- API Gateway API をカスタムドメイン名に接続する API Gateway API マッピング
- ドメイン名の Route 53 フェイルオーバー DNS レコード
- セカンダリリージョンの CloudWatch アラーム
- セカンダリリージョンの CloudWatch アラームに基づく Route 53 ヘルスチェック

まず、プライマリおよびセカンダリリージョンで、必要なすべてのリソースがあることを確認します。セカンダリリージョンにはアラームとヘルスチェックが含まれている必要があります。これにより、フェイルオーバーの実行についてプライマリリージョンに依存する必要がなくなります。これらのリソースを作成する AWS CloudFormation テンプレートの例については、「primary.yaml」と「secondary.yaml」を参照してください。

> **Important**
>
> セカンダリリージョンにフェイルオーバーする前に、必要なリソースがすべて利用可能であることを確認してください。利用可能でない場合、API はセカンダリリージョンのトラフィックに対応することができません。

ステップ 2: セカンダリリージョンへのフェイルオーバーを開始する

次の例では、スタンバイリージョンが CloudWatch メトリクスを受け取り、フェイルオーバーを開始します。フェイルオーバーを開始するにはオペレーターの介入を必要とするカスタムメトリクスを使用します。

```
aws cloudwatch put-metric-data \
    --metric-name Failover \
    --namespace HealthCheck \
    --unit Count \
    --value 1 \
    --region us-west-1
```

メトリクスデータを、設定した CloudWatch アラームの対応するデータに置き換えます。

ステップ 3: フェイルオーバーをテストする

API を呼び出し、セカンダリリージョンからレスポンスがあることを確認します。ステップ 1 でサンプルテンプレートを使用した場合、レスポンスはフェイルオーバー後に {"message":

"Hello from the primary Region!"} から {"message": "Hello from the secondary Region!"} に変わります。

```
curl https://my-api.example.com

{"message": "Hello from the secondary Region!"}
```

ステップ 4: プライマリリージョンに戻る

プライマリリージョンに戻るには、ヘルスチェックに合格する CloudWatch メトリクスを送信します。

```
aws cloudwatch put-metric-data \
    --metric-name Failover \
    --namespace HealthCheck \
    --unit Count \
    --value 0 \
    --region us-west-1
```

メトリクスデータを、設定した CloudWatch アラームの対応するデータに置き換えます。

API を呼び出し、プライマリリージョンからレスポンスがあることを確認します。ステップ 1 でサンプルテンプレートを使用した場合、レスポンスは {"message": "Hello from the secondary Region!"} から {"message": "Hello from the primary Region!"} に変わります。

```
curl https://my-api.example.com

{"message": "Hello from the primary Region!"}
```

次のステップ: 定期的にカスタマイズしてテストする

この例は、DNS フェイルオーバーを設定する 1 つの方法を示しています。フェイルオーバーを管理するヘルスチェックには、さまざまな CloudWatch メトリクスまたは HTTP エンドポイントを使用できます。フェイルオーバーメカニズムを定期的にテストして、期待どおりに機能すること、およびオペレーターがフェイルオーバー手順に精通していることを確認します。

REST API のパフォーマンスの最適化

API を呼び出せるようにしたら、応答性を向上させるために最適化する必要があることに気づくかもしれません。API Gateway には、レスポンスキャッシュやペイロード圧縮など、API を最適化するた

めのいくつかの戦略が用意されています。このセクションでは、これらの機能を有効にする方法を説明しています。

トピック

- API キャッシュを有効にして応答性を強化する
- API のペイロードの圧縮を有効にする

API キャッシュを有効にして応答性を強化する

Amazon API Gateway で API キャッシュを有効にして、エンドポイントのレスポンスがキャッシュされるようにできます。キャッシュを有効にすると、エンドポイントへの呼び出しの数を減らすことができ、また、API へのリクエストのレイテンシーを短くすることもできます。

ステージに対してキャッシュを有効にすると、API Gateway は、秒単位で指定した有効期限 (TTL) が切れるまで、エンドポイントからのレスポンスをキャッシュします。その後、API Gateway は、エンドポイントへのリクエストを行う代わりに、キャッシュからのエンドポイントのレスポンスを調べます。API キャッシュのデフォルトの TTL 値は 300 秒です。最大の TTL 値は 3600 秒です。TTL=0 は、キャッシュが無効なことを意味します。

> **Note**
>
> キャッシュはベストエフォートです。Amazon CloudWatch の `CacheHitCount` および `CacheMissCount` メトリクスを使用して、API Gateway が API キャッシュから提供するリクエストをモニタリングできます。

キャッシュが可能なレスポンスの最大サイズは 1048576 バイトです。キャッシュデータの暗号化は、キャッシュされているときのレスポンスのサイズを増やす可能性があります。

これは HIPAA 対象サービスです。AWS、1996 年制定の医療保険の相互運用性と説明責任に関する法律 (HIPAA)、および AWS のサービスを使用した保護対象医療情報 (PHI) の処理、保存、転送に関する詳細については、HIPAA の概要を参照してください。

> **Important**
>
> ステージに対してキャッシュを有効にすると、デフォルトでは`GET` メソッドのみでキャッシュが有効になります。これは、API の安全性と可用性を確保するのに役立ちます。オーバーライドするメソッドの設定により、他のメソッドのキャッシュを有効にできます。

> **Important**
>
> キャッシュでは、選択したキャッシュサイズに応じて、時間単価で料金が発生します。キャッシュは AWS 無料利用枠の対象ではありません。詳細については、「API Gateway の料金」を参照してください。

Amazon API Gateway のキャッシュを有効にする

API Gateway では、ステージ別にキャッシュを有効にすることができます。

キャッシュを有効にするときは、キャッシュ容量を選択する必要があります。一般的に、容量を大きくすると、パフォーマンスは良くなりますが、コストは増えます。サポートされているキャッシュサイズについては、API Gateway API リファレンスの「cacheClusterSize」を参照してください。

API Gateway で、キャッシュを有効にするには、専用のキャッシュインスタンスを作成します。このプロセスには最長 4 分かかることがあります。

API Gateway で、キャッシュの容量を変更するには、既存のキャッシュインスタンスを削除してから、変更した容量でキャッシュインスタンスを作成します。既存のキャッシュされたデータはすべて削除されます。

> **Note**
>
> キャッシュ容量は、キャッシュインスタンスの CPU、メモリ、およびネットワーク帯域幅に影響を及ぼします。その結果、キャッシュ容量はキャッシュのパフォーマンスに影響を与える可能性があります。
> API Gateway では、10 分のロードテストを実行して、キャッシュ容量がワークロードに適していることを確認することをお勧めします。ロードテスト中のトラフィックが本番トラフィックを反映していることを確認します。例えば、ランプアップ、一定のトラフィック、およびトラフィックのスパイクを含めます。ロードテストには、キャッシュから提供できるレスポンスと、キャッシュに項目を追加する一意のレスポンスを含める必要があります。ロードテスト中に、レイテンシー、4xx、5xx、キャッシュヒット、キャッシュミスメトリクスをモニタリングします。これらのメトリクスに基づいて、必要に応じてキャッシュ容量を調整します。負荷テストの詳細については、「レート制限に達しないように、最適な Amazon API Gateway キャッシュ容量を選択するにはどうすればよいですか？」を参照してください。

API Gateway コンソールの [ステージ] ページで、キャッシュを設定します。ステージキャッシュをプロビジョニングし、デフォルトのメソッドレベルのキャッシュ設定を指定します。デフォルトのメソッドレベルのキャッシュをオンにすると、メソッドにメソッドオーバーライドがある場合を除き、ステージのすべての GET メソッドでメソッドレベルのキャッシュが有効になります。ステージにデプロイした追加の GET メソッドでは、メソッドレベルのキャッシュが有効になります。ステージの特定のメソッドに対してメソッドレベルのキャッシュ設定を構成するには、メソッドオーバーライドを使用できます。メソッドオーバーライドの詳細については、「the section called “ステージキャッシュをメソッドキャッシュで上書きする”」を参照してください。

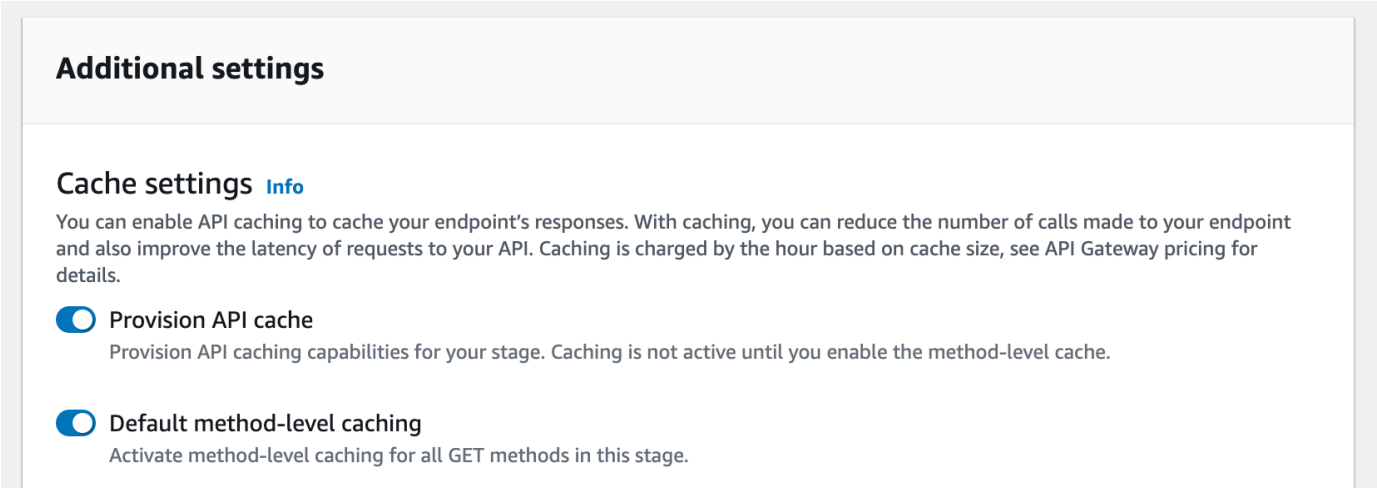
特定のステージの API キャッシュを設定するには

1. https://console.aws.amazon.com/apigateway で API Gateway コンソールにサインインします。
2. [ステージ] を選択します。
3. API の [ステージ] リストで、ステージを選択します。
4. [ステージの詳細] セクションで、[編集] を選択します。
5. [その他の設定] の [キャッシュ設定] で、[API キャッシュをプロビジョニング] をオンにします。

   これで、ステージのキャッシュクラスターがプロビジョニングされます。
6. ステージのキャッシュを有効にするには、[デフォルトのメソッドレベルのキャッシュ] をオンにします。

   これにより、ステージのすべての GET メソッドに対してメソッドレベルのキャッシュが有効になります。このステージにデプロイした追加の GET メソッドでは、メソッドレベルのキャッシュが有効になります。

> **Note**
>
> メソッドレベルのキャッシュに関する既存の設定がある場合は、デフォルトのメソッドレベルのキャッシュ設定を変更しても、既存の設定には影響しません。



7. [Save changes] (変更の保存) をクリックします。

> **Note**
>
> キャッシュの作成または削除は、API Gateway が完了するまで約 4 分かかります。
> キャッシュを作成すると、[キャッシュクラスター]] の値は `Create in progress` から `Active` に変わります。キャッシュの削除が完了すると、[キャッシュクラスター] の値は `Delete in progress` から `Inactive` に変わります。
> ステージのすべてのメソッドに対してメソッドレベルのキャッシュをオンにすると、[デフォルトのメソッドレベルのキャッシュ] の値は `Active` に変わります。ステージのすべてのメソッドに対してメソッドレベルのキャッシュをオフにすると、[デフォルトのメソッドレベルのキャッシュ] の値は `Inactive` に変わります。メソッドレベルのキャッシュに関する既存の設定がある場合は、キャッシュのステータスを変更しても、既存の設定には影響しません。

ステージの [キャッシュ設定] 内でキャッシュを有効にすると、`GET` メソッドのみがキャッシュされます。API の安全性と可用性を確保するため、この設定を変更しないことをお勧めします。ただし、オーバーライドするメソッドの設定により、他のメソッドのキャッシュを有効にできます。

キャッシュが想定どおり機能していることを確認するために、2 つの全般的なオプションがあります。

- API とステージの CacheHitCount および CacheMissCount の CloudWatch メトリクスを調べる。
- レスポンスにタイムスタンプを入力する。

> **Note**
>
> API が API Gateway キャッシュのインスタンスから供給されているかどうか確認するために、CloudFront レスポンスの `X-Cache` ヘッダーを使用しないでください。

API Gateway のステージレベルのキャッシュをメソッドレベルのキャッシュで上書きする

特定のメソッドのキャッシュをオンまたはオフにすることで、ステージレベルのキャッシュ設定を上書きできます。TTL 期間を変更したり、キャッシュされたレスポンスの暗号化のオン/オフを切り替えたりすることもできます。

[ステージの詳細] でデフォルトのメソッドレベルのキャッシュ設定を変更しても、オーバーライドが設定されたメソッドレベルのキャッシュ設定には影響しません。

キャッシュ中のメソッドがそのレスポンスで機密データを受け取ることが予想される場合は、[キャッシュ設定] で [キャッシュデータを暗号化する] を選択します。

コンソールを使用してメソッド別の API キャッシュを設定するには

1. https://console.aws.amazon.com/apigateway で API Gateway コンソールにサインインします。
2. [API] を選択します。
3. [ステージ] を選択します。
4. API の [ステージ] リストで、ステージを展開し、API のメソッドを選択します。
5. [メソッドオーバーライド] セクションで、[編集] を選択します。
6. [メソッド設定] セクションで、[メソッドキャッシュを有効にする] をオンまたはオフにするか、その他の必要なオプションをカスタマイズします。

   > **Note**
   >
   > ステージのキャッシュクラスターをプロビジョニングするまで、キャッシュはアクティブになりません。

7. [Save] を選択します。

メソッドパラメータまたは統合パラメータをキャッシュキーとして使用して、キャッシュされたレスポンスにインデックスを付ける

キャッシュされたメソッドや統合に、カスタムヘッダー、URL パス、またはクエリ文字列の形式でパラメーターを渡せる場合、パラメーターの一部またはすべてを使用してキャッシュキーを作成できます。API Gateway は、使用されたパラメータ値に応じて、メソッドレスポンスをキャッシュできます。

> **Note**
>
> キャッシュキーはリソースにキャッシュを設定するときに必要です。

たとえば、以下の形式のリクエストがあるとします。

```
GET /users?type=... HTTP/1.1
host: example.com
...
```

このリクエストでは、type は admin または regular の値を受け取ることができます。キャッシュキーに type パラメーターを含めた場合、GET /users?type=admin からのレスポンスと GET /users?type=regular からのレスポンスは別々にキャッシュされます。

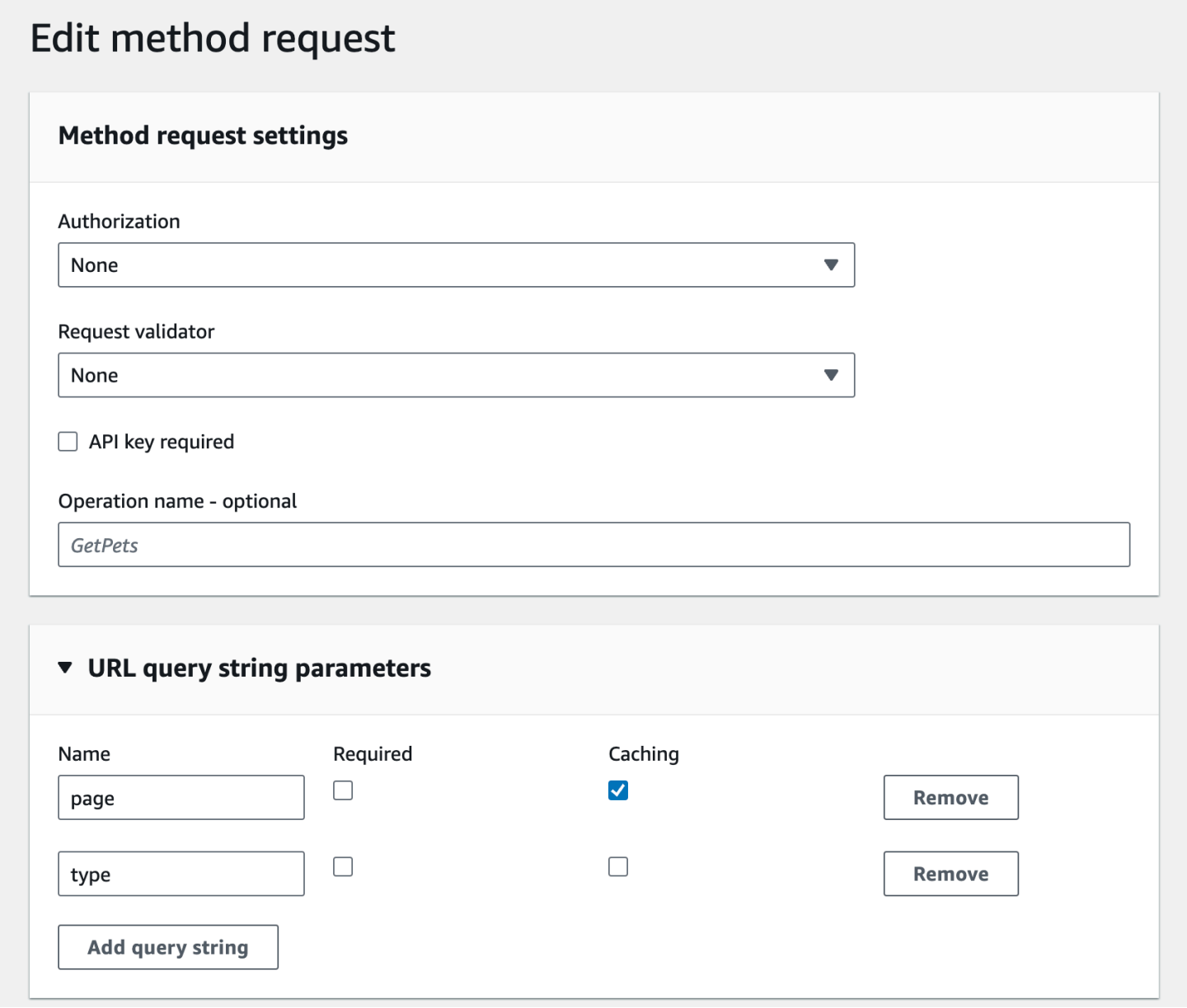
メソッドリクエストまたは統合リクエストが複数のパラメーターを受け取るときは、パラメーターの一部またはすべてを含めてキャッシュキーを作成するように選択できます。たとえば、指定した順に TTL 期間内に行われる以下のリクエストに対して、キャッシュキーに type パラメータのみを含めることができます。

```
GET /users?type=admin&department=A HTTP/1.1
host: example.com
...
```

このリクエストからのレスポンスがキャッシュされ、以下のリクエストの処理に使用されます。

```
GET /users?type=admin&department=B HTTP/1.1
host: example.com
...
```

API Gateway コンソールで、キャッシュキーにメソッドリクエストまたは統合リクエストのパラメータを含めるには、パラメータを追加した後に [Caching (キャッシュ)] を選択します。



API Gateway で API ステージキャッシュをフラッシュする

API キャッシュが有効になったら、API ステージのキャッシュをフラッシュして、API のクライアントが統合エンドポイントから最新のレスポンスを取得できるようにします。

API ステージキャッシュをフラッシュするには、[ステージアクション] メニューを選択し、[ステージキャッシュをフラッシュする] を選択します。

> **Note**
>
> キャッシュがフラッシュされると、再度キャッシュが作成されるまで、統合エンドポイントからレスポンスが処理されます。この間に、統合エンドポイントに送信されるリクエストの数が増加する場合があります。これにより、API のレイテンシー全体が一時的に増える可能性があります。

API Gateway のキャッシュエントリの無効化

API のクライアントは既存のキャッシュエントリを無効化し、個別のリクエストに対して統合エンドポイントからそのエントリを再ロードできます。クライアントは、`Cache-Control: max-age=0` ヘッダーを含むリクエストを送信する必要があります。クライアントは、クライアントが許可されている場合、キャッシュの代わりに統合エンドポイントから直接レスポンスを受け取ります。これは既存のキャッシュエントリを、統合エンドポイントから取得される新しいレスポンスで置き換えます。

クライアントのアクセス許可を付与するには、次の形式のポリシーを、ユーザーの IAM 実行ロールにアタッチします。

> **Note**
>
> クロスアカウントのキャッシュ無効化はサポートされていません。

```
{
  "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"
```

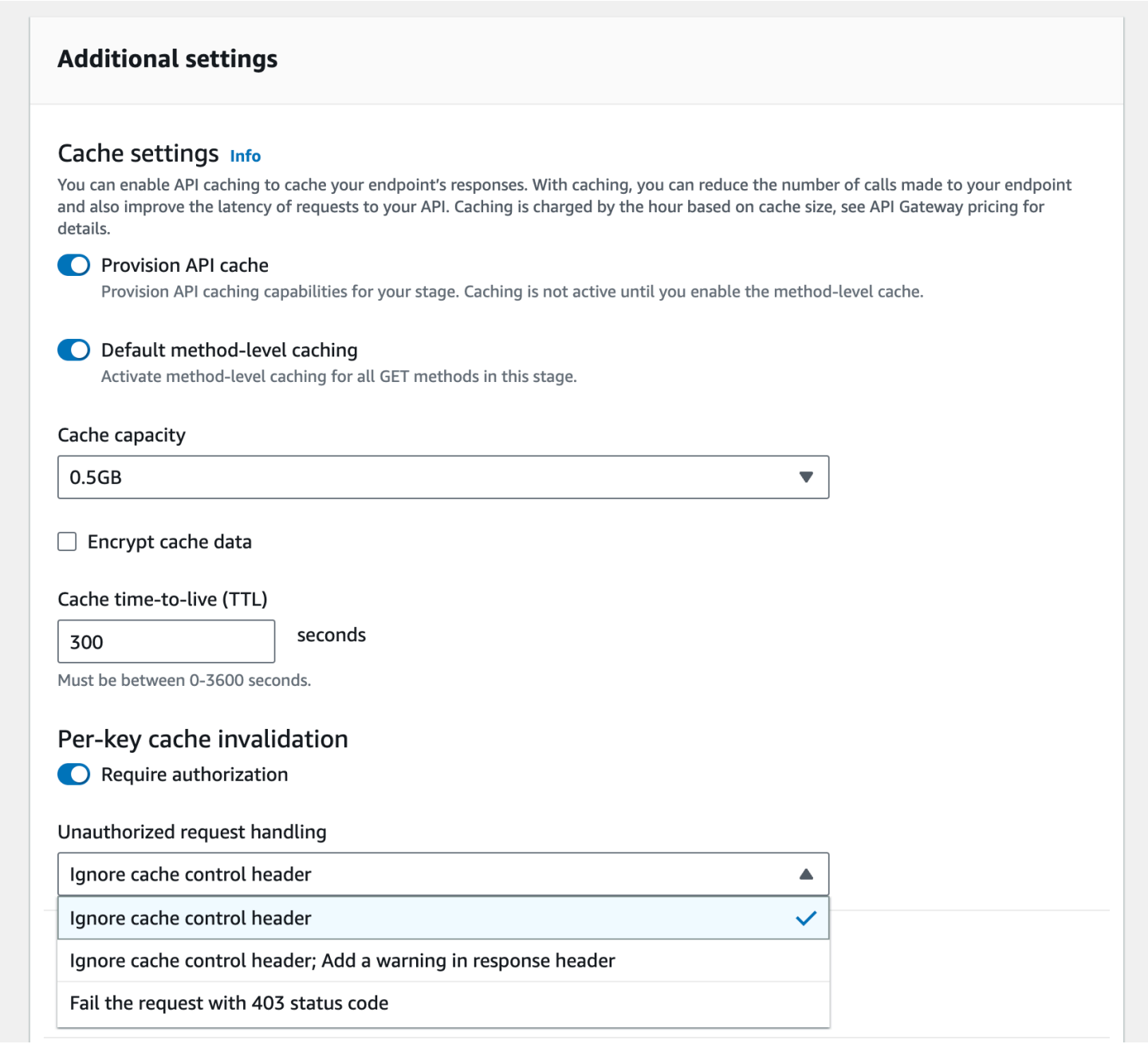
```
      ]
    }
  ]
}
```

このポリシーでは、API Gateway 実行サービスが指定された 1 つまたは複数のリソースのリクエストに対してキャッシュを無効にします。対象リソースのグループを指定するには、`account-id`、`api-id`、および他のエントリの ARN 値 `Resource` に対してワイルドカード文字 (*) を使用します。API Gateway 実行サービスのアクセス許可の設定方法については、「IAM アクセス許可により API へのアクセスを制御する」を参照してください。

`InvalidateCache` ポリシーを適用しない場合 (またはコンソールで [Require authorization (認証が必要)] チェックボックスをオンにした場合)、すべてのクライアントが API キャッシュを無効にできます。すべてまたはほとんどのクライアントが API キャッシュを無効にする場合、API のレイテンシーが非常に大きくなる可能性があります。

ポリシーが設定されると、キャッシュが有効になり、承認が必要になります。

API Gateway コンソールで [不正なリクエスト処理] のオプションを選択して、不正なリクエストの処理方法を制御できます。



3 つのオプションにより、次の動作が発生します。

- 403 ステータスコードでリクエストに失敗する: 403 Unauthorized レスポンスが返されます。

  API を使用してこのオプションを設定するには、`FAIL_WITH_403` を使用します。
- キャッシュコントロールヘッダーを無視し、レスポンスヘッダーに警告を追加する: リクエストを処理し、レスポンスに警告ヘッダーを追加します。

API を使用してこのオプションを設定するには、SUCCEED_WITH_RESPONSE_HEADER を使用します。

- キャッシュコントロールヘッダーを無視する: リクエストを処理し、レスポンスで警告ヘッダーを追加しません。

  API を使用してこのオプションを設定するには、SUCCEED_WITHOUT_RESPONSE_HEADER を使用します。

API のペイロードの圧縮を有効にする

API Gateway を使用すると、サポートされているコンテンツコーディングのいずれかを使用して、圧縮されたペイロードで API を呼び出すことができます。デフォルトでは、API Gateway はメソッドリクエストペイロードの解凍をサポートしています。ただし、メソッドレスポンスペイロードの圧縮を有効にするように API を設定する必要があります。

API で圧縮を有効にするには、API を作成するとき、または API を作成した後に、minimumCompressionsSize プロパティを 0 から 10485760 (10M バイト) の間の負でない整数に設定します。API で圧縮を無効にするには、minimumCompressionSize を null に設定するか、または完全に削除します。API Gateway コンソール、AWS CLI、または API Gateway REST API を使用して、API の圧縮を有効または無効にすることができます。

任意のサイズのペイロードに圧縮を適用する場合は、minimumCompressionSize 値をゼロに設定します。ただし、小さいサイズのデータを圧縮すると、実際にはデータサイズが大きくなる可能性があります。さらに、API Gateway での圧縮とクライアントでの解凍は、全体のレイテンシーを増加させ、より多くの計算時間を必要とする可能性があります。API に対してテストケースを実行して、最適な値を決定する必要があります。

クライアントは、API Gateway に対して、圧縮ペイロードと適切な Content-Encoding ヘッダーを含む API リクエストを送信して、統合エンドポイントにリクエストを渡す前に、解凍して適用可能なマッピングテンプレートを適用できます。圧縮が有効になって API がデプロイされた後、メソッドリクエストに適切な Accept-Encoding ヘッダーが指定されている場合、クライアントは圧縮されたペイロードで API レスポンスを受け取ることができます。

統合エンドポイントが圧縮されていない JSON ペイロードを想定して返すとき、圧縮されていない JSON ペイロード用に設定されたマッピングテンプレートは、圧縮されたペイロードに適用されます。圧縮されたメソッドリクエストペイロードの場合、API Gateway はペイロードを解凍し、マッピングテンプレートを適用して、マップされたリクエストを統合エンドポイントに渡します。圧縮さ

れていない統合レスポンスペイロードの場合、API Gateway はマッピングテンプレートを適用し、マップされたペイロードを圧縮し、圧縮されたペイロードをクライアントに返します。

トピック

- API のペイロードの圧縮を有効にする
- 圧縮されたペイロードで API メソッドを呼び出す
- 圧縮されたペイロードで API レスポンスを受信する

API のペイロードの圧縮を有効にする

API Gateway コンソール、AWS CLI、または AWS SDK を使用して、API の圧縮を有効にできます。

既存の API では、圧縮を有効にした後、API をデプロイして変更を有効にする必要があります。新しい API の場合、API のセットアップ完了後に API をデプロイします。

> **Note**
>
> 最も優先順位の高いコンテンツのエンコードは、API Gateway によってサポートされている必要があります。サポートされていない場合は、レスポンスペイロードに圧縮が適用されません。

トピック

- API Gateway コンソールを使用して API のペイロードの圧縮を有効にする
- AWS CLI を使用して API のペイロードの圧縮を有効にする
- API Gateway でサポートされるコンテンツコーディング

API Gateway コンソールを使用して API のペイロードの圧縮を有効にする

次の手順では、API のペイロードの圧縮を有効にする方法について説明します。

API Gateway コンソールを使用してペイロードの圧縮を有効にするには

1. https://console.aws.amazon.com/apigateway で API Gateway コンソールにサインインします。
2. 既存の API を選択するか、新しい API を作成します。

3. メインナビゲーションペインで、[API キー] を選択します。
4. [API の詳細] セクションで [編集] を選択します。
5. [コンテンツエンコーディング] をオンにして、ペイロード圧縮を有効にします。[本文の最小サイズ] には、最小圧縮サイズ (バイト単位) の数を入力します。圧縮をオフにするには、[コンテンツエンコーディング] オプションをオフにします。
6. [Save changes] (変更の保存) をクリックします。

AWS CLI を使用して API のペイロードの圧縮を有効にする

AWS CLI を使用して、新しい API を作成し、圧縮を有効にするには、次のように create-rest-api コマンドを呼び出します。

```
aws apigateway create-rest-api \
    --name "My test API" \
    --minimum-compression-size 0
```

AWS CLI を使用して、既存の API で圧縮を有効にするには、次のように update-rest-api コマンドを呼び出します。

```
aws apigateway update-rest-api \
    --rest-api-id 1234567890 \
    --patch-operations op=replace,path=/minimumCompressionSize,value=0
```

minimumCompressionSize プロパティには、0 ~ 10485760 (10M バイト) の間の負でない整数値があります。これは圧縮のしきい値を測定します。ペイロードサイズがこの値よりも小さい場合、圧縮または解凍はペイロードに適用されません。ゼロに設定すると、任意のペイロードサイズの圧縮を許可します。

AWS CLI を使用して、圧縮を無効にするには、次のように update-rest-api コマンドを呼び出します。

```
aws apigateway update-rest-api \
    --rest-api-id 1234567890 \
    --patch-operations op=replace,path=/minimumCompressionSize,value=
```

value を空の文字列 "" に設定するか、前の呼び出しで value プロパティを完全に省略することができます。

API Gateway でサポートされるコンテンツコーディング

API Gateway は、次のコンテンツコーディングをサポートしています。

- `deflate`
- `gzip`
- `identity`

API Gateway は RFC 7231 仕様に従って、次の `Accept-Encoding` ヘッダー形式もサポートしています。

- `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`

圧縮されたペイロードで API メソッドを呼び出す

圧縮されたペイロードで API リクエストを行うには、クライアントは `Content-Encoding` ヘッダーをサポートされているコンテンツコーディングの 1 つで設定する必要があります。

API クライアントで、PetStore API メソッド (`POST /pets`) を呼び出すとします。次の JSON 出力を使用してメソッドを呼び出さないでください。

```
POST /pets
Host: {petstore-api-id}.execute-api.{region}.amazonaws.com
Content-Length: ...

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

代わりに、GZIP コーディングを使用して圧縮された同じペイロードでメソッドを呼び出すことができます。

```
POST /pets
```

```
Host: {petstore-api-id}.execute-api.{region}.amazonaws.com
Content-Encoding:gzip
Content-Length: ...

◆◆◆RPP*◆,HU◆RPJ◆OW◆◆e&◆◆◆L,◆,-y◆j
```

API Gateway はリクエストを受け取ると、指定されたコンテンツコーディングがサポートされているかどうかを確認します。次に、指定されたコンテンツコーディングでペイロードを解凍しようとします。解凍が成功すると、リクエストは統合エンドポイントにディスパッチされます。指定されたコーディングがサポートされていないか、または指定されたコーディングで指定されたペイロードが圧縮されていない場合、API Gateway は `415 Unsupported Media Type` エラーレスポンスを返します。このエラーは、API およびステージが識別される前の解凍の早い段階で発生した場合は、CloudWatch Logs に記録されません。

圧縮されたペイロードで API レスポンスを受信する

圧縮が有効な API を作成する場合、クライアントは `Accept-Encoding` ヘッダーをサポートされているコンテンツコーディングで指定することにより、圧縮されたレスポンスのペイロードを指定の形式で受信することを選択できます。

API Gateway は、次の条件が満たされた場合にのみレスポンスのペイロードを圧縮します。

- 着信リクエストには、サポートされているコンテンツコーディングと形式の `Accept-Encoding` ヘッダーがあります。

  > **Note**
  >
  > ヘッダーが設定されていない場合、デフォルト値は RFC 7231 で定義された * です。このような場合、API Gateway はペイロードを圧縮しません。一部のブラウザまたはクライアントでは、`Accept-Encoding` (`Accept-Encoding:gzip, deflate, br` など) が、圧縮が有効なリクエストに自動的に追加されます。これにより、API Gateway でペイロードの圧縮をトリガーできます。サポートされている `Accept-Encoding` ヘッダー値の明示的な仕様がない場合、API Gateway はペイロードを圧縮しません。

- `minimumCompressionSize` は、圧縮を有効にするために API で設定されています。
- 統合レスポンスには `Content-Encoding` ヘッダーはありません。
- 統合レスポンスのペイロードのサイズは、適用可能なマッピングテンプレートが適用されると、指定された `minimumCompressionSize` 値以上になります。

API Gateway はペイロードを圧縮する前に、統合レスポンス用に設定されたマッピングテンプレートを適用します。統合レスポンスに Content-Encoding ヘッダーが含まれている場合、API Gateway は、統合レスポンスのペイロードはすでに圧縮されているとみなし、圧縮処理がスキップされます。

たとえば、PetStore API の例と次のリクエストがあります。

```
GET /pets
Host: {petstore-api-id}.execute-api.{region}.amazonaws.com
Accept: application/json
```

バックエンドはリクエストに次のような圧縮されていない JSON ペイロードで応答します。

```
200 OK

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

この出力を圧縮されたペイロードとして受け取るために、API クライアントは次のようにリクエストを送信できます。

```
GET /pets
Host: {petstore-api-id}.execute-api.{region}.amazonaws.com
Accept-Encoding:gzip
```

クライアントは、Content-Encoding ヘッダーと次のような GZIP エンコードされたペイロードでレスポンスを受信します。

```
200 OK
Content-Encoding:gzip
...

[binary]

[binary]
```

レスポンスのペイロードが圧縮されると、圧縮されたデータサイズのみがデータ転送に対して課金されます。

REST API をクライアントに配布する

このセクションでは、API Gateway API を顧客に配布する方法について詳しく説明します。API の配布には、顧客がクライアントアプリケーションをダウンロードして統合するための SDK の生成、顧客がクライアントアプリケーションからそれを呼び出す方法を顧客に知らせるための API のドキュメント化、製品提供の一部としての API の利用が含まれます。

トピック

- API キーを使用した使用量プランの作成と使用
- REST API をドキュメント化する
- API Gateway で REST API 用 SDK を生成する
- AWS Marketplace で API Gateway API を販売する

API キーを使用した使用量プランの作成と使用

API を作成、テストして、デプロイすると、API Gateway 使用量プランを使用して、顧客への提供商品として使用できるようになります。選択した API へのアクセスを顧客に許可する使用量プランと API キーを設定し、定義した制限とクォータに基づいてこれらの API へのリクエストのスロットリングを開始できます。これらは API、または API メソッドレベルで設定できます。

使用量プランおよび API キーとは

使用量プランは、デプロイ済みの API ステージとメソッドにアクセスできるユーザーを指定します。リクエストのスロットリングを開始するターゲットリクエストレートを設定することもできます (オプション)。このプランは、API キーを使用して、各キーの関連付けられた API ステージにアクセスできる API クライアントとユーザーを識別します。

API キーは、API へのアクセスを付与するために顧客のアプリケーションデベロッパーに配布する英数字の文字列値です。API キーと Lambda オーソライザー、IAM ロール、または Amazon Cognito を一緒に使用して API へのアクセスを制御できます。ユーザーに代わって API Gateway が API キーを生成することも、CSV ファイルからインポートすることもできます。API Gateway で API キーを生成することも、外部ソースから API Gateway にインポートすることもできます。詳細については、「the section called “API Gateway コンソールを使用して API キーをセットアップする”」を参照してください。

API キーには名前と値があります。(「API キー」と「API キー値」という用語は、しばしば同じ意味で使用されます。) この名前は 1024 文字を超えることはできません。値は、20〜128 文字の英数字の文字列です。例えば、`apikey1234abcdefghij0123456789` です。

> **Important**
>
> API キー値は一意である必要があります。異なる名前で同じ値の 2 つの API キーを作成しようとすると、API Gateway はそれらを同じ API キーと見なします。
> API キーを複数の使用量プランと関連付けることができます。使用量プランを複数のステージと関連付けることができます。ただし、指定された API キーは API の各ステージの 1 つの使用量プランにのみ関連付けることができます。

スロットリングの制限は、リクエストスロットリングを開始するターゲットポイントを設定します。これは API、または API メソッドレベルで設定できます。

クォータ制限は、指定した期間内に送信できる API キーを持つリクエストの目標最大数を設定します。個別の API メソッドを、使用量プラン設定に基づく API キー認証を要求するように設定できます。

スロットリングとクォータ制限は、使用プランのすべての API ステージについて集約される個々の API キーのリクエストに適用されます。

> **Note**
>
> 使用量プランのスロットリングとクォータはハードリミットではなく、ベストエフォートベースで適用されます。場合によっては、クライアントは設定されているクォータを超えることがあります。コストの管理や API へのアクセスのブロックを行う際に使用プランのクォータやスロットリングに依存しないでください。AWS Budgets を使用してコストをモニタリングすること、および AWS WAF を使用して API リクエストを管理することを検討してください。

API キーと使用量プランのベストプラクティス

以下は、API キーおよび使用量プランを使用する場合の推奨されるベストプラクティスです。

> **Important**
>
> - API キーを、API へのアクセスを制御するための認証または承認に使用しないでください。使用量プランに複数の API がある場合、その使用量プランの 1 つの API に対して有効な API キーを持つユーザーは、その使用量プランのすべての API にアクセスできます。代わりに、API へのアクセスを制御するには、IAM ロール、Lambda オーソライザー、または Amazon Cognito ユーザープールを使用します。
> - API Gateway が生成する API キーを使用します。API キーには機密情報を含めないでください。クライアントは通常、ログに記録できるヘッダーで機密情報を送信します。

- デベロッパーポータルを使用して API を公開している場合は、顧客に表示していなくても、特定の使用量プランのすべての API は顧客からサブスクライブ可能であることに注意してください。
- 場合によっては、クライアントは設定されているクォータを超えることがあります。コストを制御するために使用計画に依存しないでください。AWS Budgets を使用してコストをモニタリングすること、および AWS WAF を使用して API リクエストを管理することを検討してください。
- API キーを使用量プランに追加した後で、更新オペレーションが完了するまでに数分かかる場合があります。

使用量プランを設定するステップ

以下のステップは、API 所有者として、顧客のために使用量プランを作成し、設定する方法を示します。

使用プランを設定するには

1. 1 つ以上の API を作成し、API キーを要求するメソッドを設定して、ステージに API をデプロイします。
2. API を使用するアプリケーションの開発者 (顧客) に配布する API キーを生成またはインポートします。
3. 目的のスロットリングとクォータ制限を持つ使用プランを作成します。
4. 使用量プランに、API ステージと API キーを関連付けます。

API の呼び出し元は、API へのリクエストの `x-api-key` ヘッダーで、割り当てられた API キーを指定する必要があります。

> **Note**
>
> 使用量プランに API メソッドを含めるには、個別の API メソッドを、API キーを要求するように設定する必要があります。考慮すべきベストプラクティスについては、「the section called “API キーと使用量プランのベストプラクティス”」を参照してください。

API キーのソースを選択する

使用量プランを API と関連付けて API メソッドで API キーを有効にする場合、API への各受信リクエストには API キーが含まれている必要があります。API Gateway はキーを読み取り、使用量プランのキーと照合します。一致する場合、API Gateway は、プランのリクエスト制限とクォータに従ってリクエストを調整します。それ以外の場合は、`InvalidKeyParameter` 例外がスローされます。その結果、発信者は `403 Forbidden` レスポンスを受け取ります。

API Gateway は 2 つのソースのいずれかから API キーを受け取ることができます。

`HEADER`

API キーを顧客に配布して、各受信リクエストの `X-API-Key` ヘッダーとして API キーを渡す必要があります。

AUTHORIZER

この認可レスポンスの一部として API キーを返す Lambda オーソライザーを使用することができます。認証レスポンスの詳細については、「the section called “API Gateway Lambda オーソライザーからの出力”」を参照してください。

> **Note**
>
> 考慮すべきベストプラクティスについては、「the section called “API キーと使用量プランのベストプラクティス”」を参照してください。

API Gateway コンソールを使用して API の API キーソースを選択するには、次の作業を行います。

1. [API Gateway コンソール] にサインインします。
2. 既存の API を選択するか、新しい API を作成します。
3. メインナビゲーションペインで、[API キー] を選択します。
4. [API の詳細] セクションで [編集] を選択します。
5. API キーソースで、ドロップダウンリストから `Header` または `Authorizer` を選択します。
6. [Save changes] (変更の保存) をクリックします。

AWS CLI を使用して API の API キーのソースを選択するには、次のように `update-rest-api` コマンドを呼び出します。

```
aws apigateway update-rest-api --rest-api-id 1234123412 --patch-operations
 op=replace,path=/apiKeySource,value=AUTHORIZER
```

クライアントが API キーを送信するには、上記の CLI コマンドで `value` を `HEADER` に設定します。

API Gateway REST API を使用して API の API キーのソースを選択するには、次のように `restapi:update` を呼び出します。

```
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"
    }
  ]
}
```

オーソライザーに API キーを戻すには、前の value 入力で AUTHORIZER を patchOperations に設定します。

選択した API キーのソースタイプに応じて、以下のいずれかの手順を使用してヘッダーソースの API キー、またはメソッド呼び出しでオーソライザーから返される API キーを使用します。

ヘッダーソースの API キーを使用するには

1. 必要な API メソッドで API を作成し、その API をステージにデプロイします。
2. 新しい使用プランを作成するか、既存のものから選択します。デプロイされた API ステージを使用プランに追加します。使用プランに API キーをアタッチするか、プラン内の既存の API キーを選択します。選択した API キーの値を書き留めます。
3. API キーを要求するよう API メソッドをセットアップします。
4. 同じステージに API を再デプロイします。新しいステージに API をデプロイする場合は、必ず新しい API ステージをアタッチするように使用プランを更新します。

これでクライアントから、選択した API キーをヘッダー値とする x-api-key ヘッダーを指定しながら API メソッドを呼び出すことができるようになりました。

オーソライザーソースの API キーを使用するには

1. 必要な API メソッドで API を作成し、その API をステージにデプロイします。
2. 新しい使用プランを作成するか、既存のものから選択します。デプロイされた API ステージを使用プランに追加します。使用プランに API キーをアタッチするか、プラン内の既存の API キーを選択します。選択した API キーの値を書き留めます。

3. トークンベースの Lambda オーソライザーを作成します。認証レスポンスのルートレベルのプロパティとして `usageIdentifierKey:{api-key}` を含めます。トークンベースのオーソライザーを作成する手順については、「the section called “TOKEN オーソライザー Lambda 関数の例”」を参照してください。
4. API キーを要求するように API メソッドを設定し、このメソッドでも Lambda オーソライザーを有効にします。
5. 同じステージに API を再デプロイします。新しいステージに API をデプロイする場合は、必ず新しい API ステージをアタッチするように使用プランを更新します。

これでクライアントから、明示的に API キーを指定せずに、API キーを要求するメソッドを呼び出すことができるようになりました。オーソライザーから返される API キーは自動的に使用されます。

API Gateway コンソールを使用して API キーをセットアップする

API キーを設定するには、以下の作業を行います。

- API キーを要求するよう API メソッドを設定します。
- リージョンの API 用に API キーを作成またはインポートします。

API キーを設定するには、事前に API を設定し、それをステージにデプロイしている必要があります。API キー値は、作成後に変更することはできません。

API Gateway コンソールを使用して API を作成し、デプロイする方法については、「API Gateway での REST API の開発」および「Amazon API Gateway での REST API のデプロイ」をそれぞれ参照してください。

API キーを作成したら、これを使用量プランに関連付ける必要があります。詳細については、「API Gateway コンソールで使用量プランを作成、設定、テストする」を参照してください。

> **Note**
>
> 考慮すべきベストプラクティスについては、「the section called “API キーと使用量プランのベストプラクティス”」を参照してください。

トピック

- メソッドで API キーを要求する

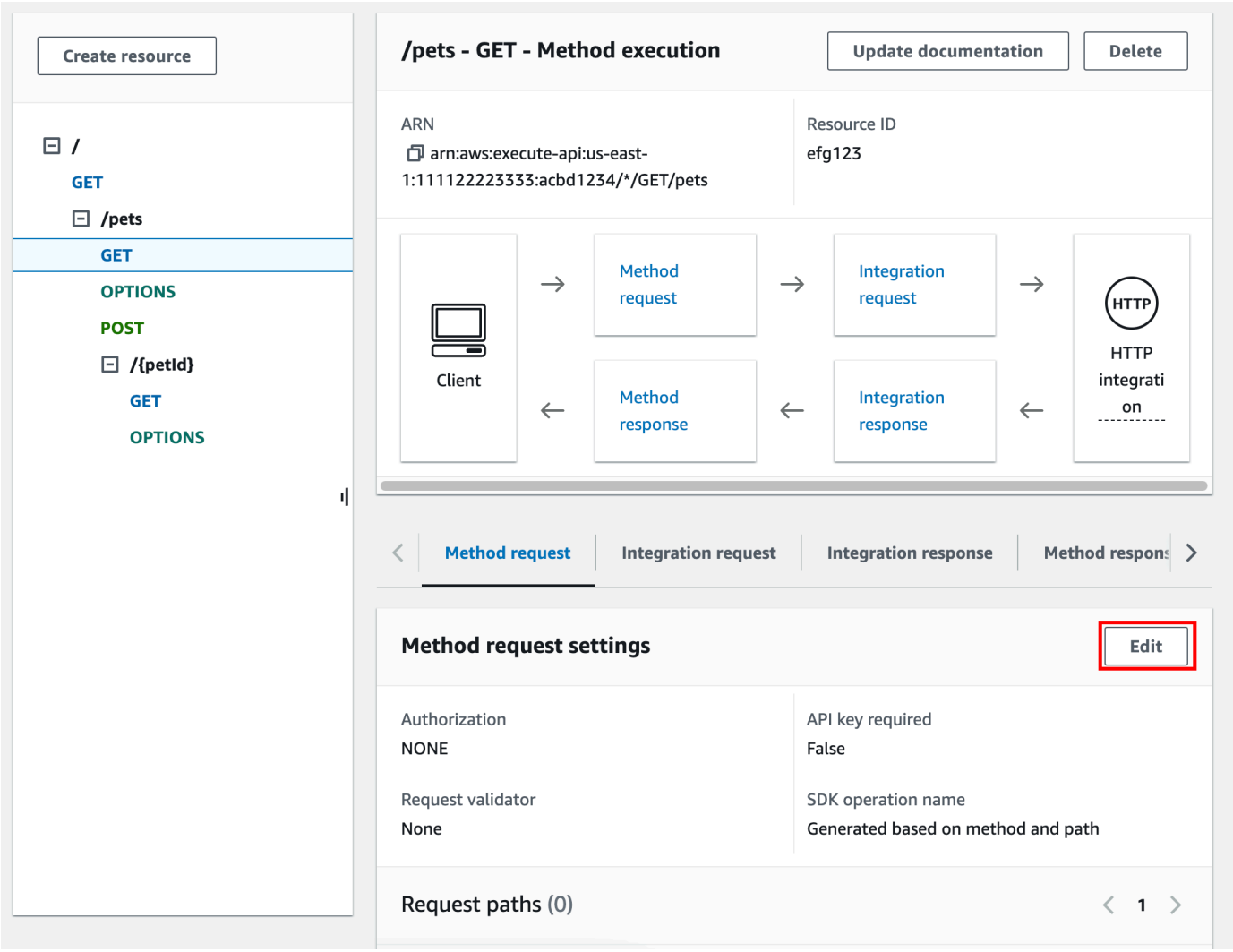
- API キーを作成する
- API キーをインポートする

メソッドで API キーを要求する

次の手順では、API キーを要求する API メソッドを設定する方法について説明します。

API キーを要求する API メソッドを設定するには

1. https://console.aws.amazon.com/apigateway で API Gateway コンソールにサインインします。
2. REST API を選択します。
3. API Gateway のメインナビゲーションペインで、[Resources (リソース)] を選択します。
4. [リソース] で、新しいメソッドを作成するか、既存のメソッドを選択します。
5. [メソッドリクエスト] タブの [メソッドリクエスト設定] で、[編集] を選択します。



6. [API キーの必要性] を選択します。
7. [Save] を選択します。
8. API をデプロイまたは再デプロイして要件を満たします。

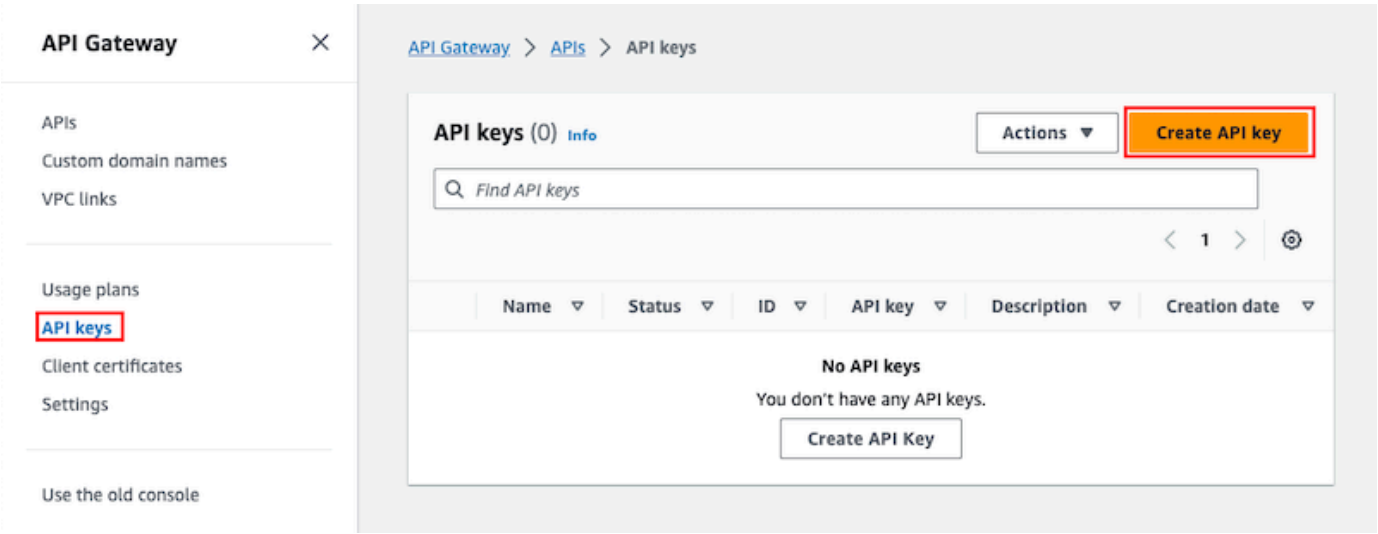
必要な API キーのオプションを `false` に設定し、上記の手順を実行しない場合、そのメソッドに対しては API ステージに関連付けられている API キーが使用されません。

API キーを作成する

使用量プランで使う API キーをすでに作成またはインポートした場合、この手順および次の手順を省略できます。

API キーを作成するには

1. https://console.aws.amazon.com/apigateway で API Gateway コンソールにサインインします。
2. REST API を選択します。
3. API Gateway のメインナビゲーションペインで、[API キー] を選択します。
4. [API の作成] を選択します。



5. [ロール名] に名前を入力します。
6. (オプション) [説明] に説明を入力します。
7. [API キー] では、[自動生成] を選択して API Gateway にキー値を生成させるか、[カスタム] を選択して独自のキー値を作成します。
8. [Save] を選択します。

API キーをインポートする

次の手順は、使用プランで API キーをインポートする方法について説明します。

API キーをインポートするには

1. https://console.aws.amazon.com/apigateway で API Gateway コンソールにサインインします。
2. REST API を選択します。
3. メインナビゲーションペインで、[API キー] を選択します。
4. [アクション] ドロップダウンメニューから、[API キーのインポート] を選択します。
5. カンマ区切りのキーファイルをロードするには、[ファイルの選択] を選択します。テキストエディターにキーを入力します。ファイル形式の詳細については、「the section called “API Gateway API キーファイルの形式”」を参照してください。
6. 警告が発生した場合、[警告を失敗とみなす] を選択してインポートを停止するか、[警告を無視する] を選択して、エラーが発生した場合に、有効なキーエントリのインポートを継続します。
7. [インポート] を選択して API キーをインポートします。

API Gateway コンソールで使用量プランを作成、設定、テストする

使用量プランを作成する前に、該当する API キーがセットアップされていることを確認します。詳細については、「API Gateway コンソールを使用して API キーをセットアップする」を参照してください。

このセクションでは、API Gateway コンソールを使って使用量プランを作成、使用するための方法ついて説明します。

トピック

- API をデフォルトの使用量プランに移行する (必要な場合)
- 使用量プランを作成する
- 使用量プランをテストする
- 使用量プランをメンテナンスする

API をデフォルトの使用量プランに移行する (必要な場合)

使用量プラン機能が展開された 2016 年 8 月 11 日以降に API Gateway の使用を開始した場合は、サポートされているすべてのリージョンで自動的に使用量プランが有効になっています。

それ以前に API Gateway の使用を開始した場合は、デフォルトの使用量プランへの移行が必要な場合もあります。選択したリージョンで使用量プランを初めて使用する前に、使用量プランの有効化オプションが表示されます。このオプションを有効化すると、既存の API キーに関連付けられた一意の API ステージすべてに、デフォルトの使用量プランが作成されます。デフォルトの使用量プランでは最初にスロットリングやクォータ制限が設定されていません。API キーと API ステージの関連付けは使用量プランにコピーされます。API は以前と同じように動作します。ただし、指定された API ステージの値 (`apiId` および `stage`) を含まれている API キーに関連付ける (`UsagePlanKey` を使用) には、`ApiKey` `stageKeys` プロパティを使用する代わりに、`UsagePlan` `apiStages` プロパティを使用する必要があります。

既にデフォルトの使用量プランに移行しているかどうかを確認するには、`get-account` CLI コマンドを使用します。使用量プランが有効になっている場合は、コマンド出力で `features` リストに `"UsagePlans"` のエントリが含まれます。

次のように AWS CLI を使用して、デフォルトの使用量プランに API を移行することもできます。

AWS CLI を使用してデフォルトの使用量プランに移行する

1. この CLI コマンド `update-account` を呼び出します。
2. `cli-input-json` パラメータには次の JSON を使用します。

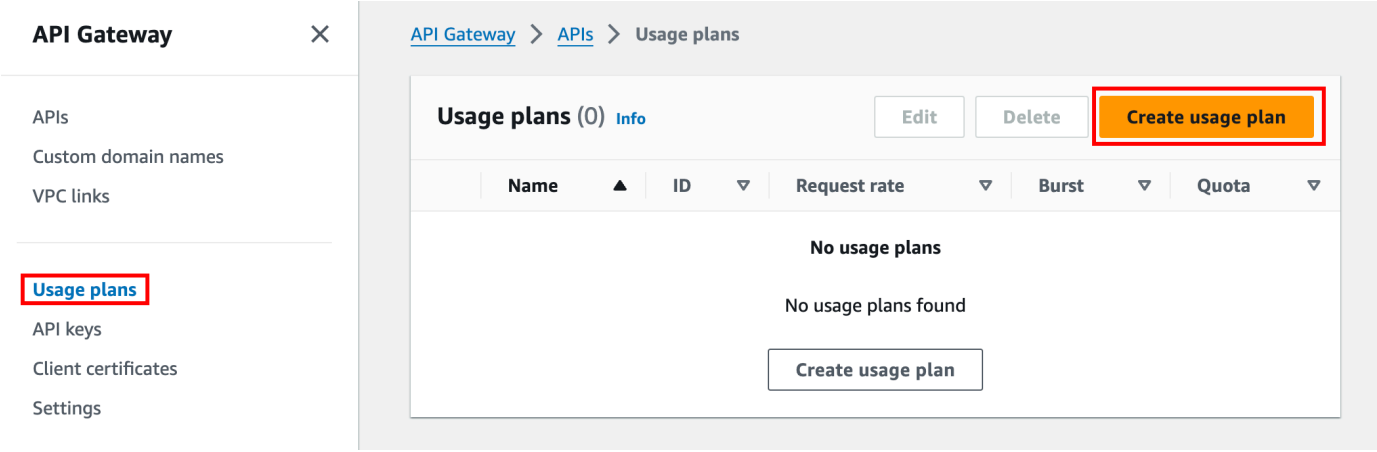
```
[
    {
        "op": "add",
        "path": "/features",
        "value": "UsagePlans"
    }
]
```

使用量プランを作成する

次の手順は、使用量プランを作成する方法を説明します。

使用量プランを作成するには

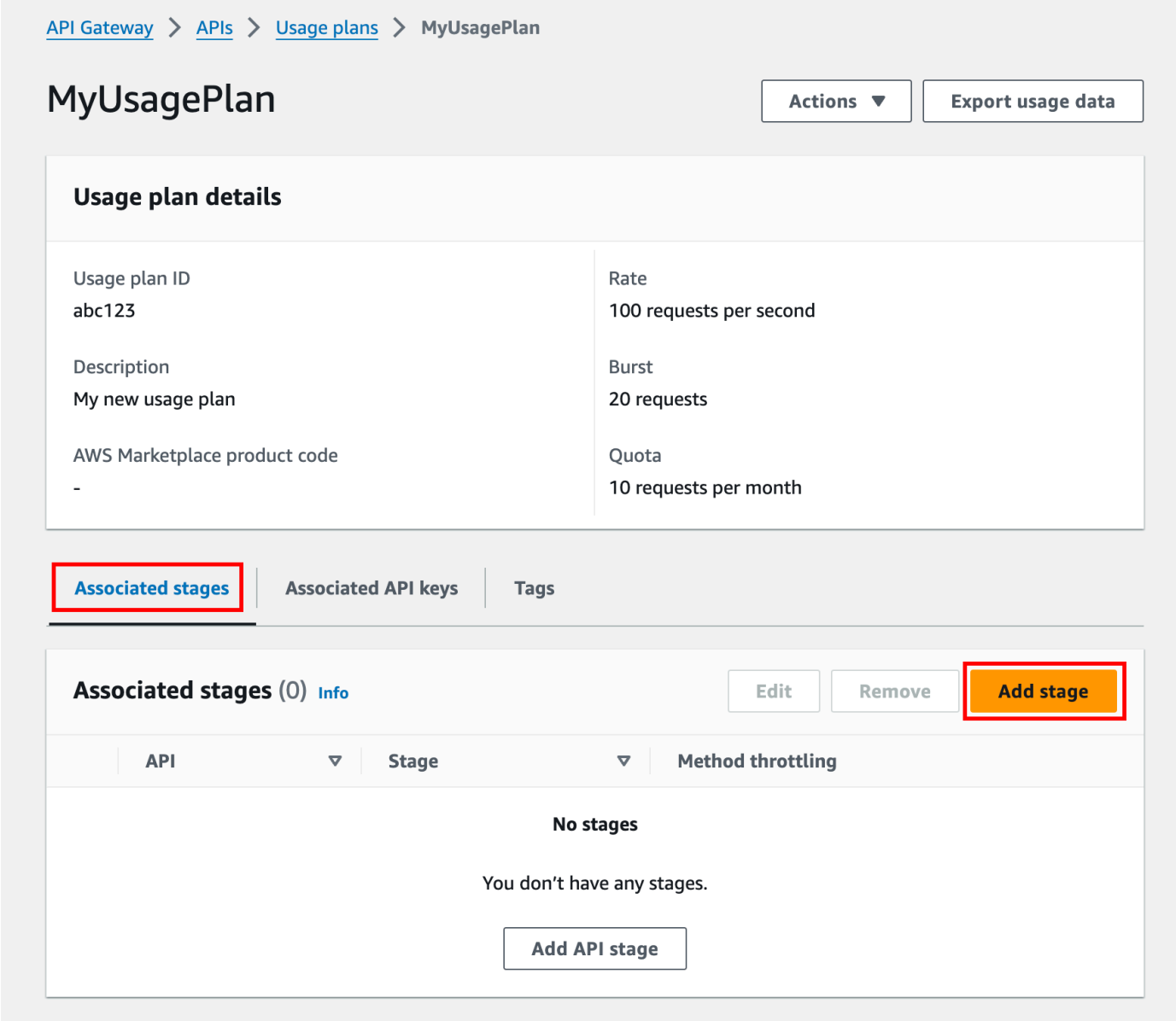
1. https://console.aws.amazon.com/apigateway で API Gateway コンソールにサインインします。
2. Amazon API Gateway のメインナビゲーションペインで、[使用量プラン] を選択し、次に [使用量プランの作成] を選択します。



3. [ロール名] に名前を入力します。
4. (オプション) [説明] に説明を入力します。
5. デフォルトでは、使用量プランではスロットリングが有効になっています。使用量プランの [レート] と [バースト] を入力します。[スロットリング] を選択してスロットリングをオフにします。
6. デフォルトでは、使用量プランでは一定期間のクォータが有効になっています。[リクエスト] には、使用プランの期間中にユーザーが実行できるリクエストの合計数を入力します。[クォータ] を選択してクォータを無効にします。
7. [使用量プランの作成] を選択します。

使用プランにステージを追加するには

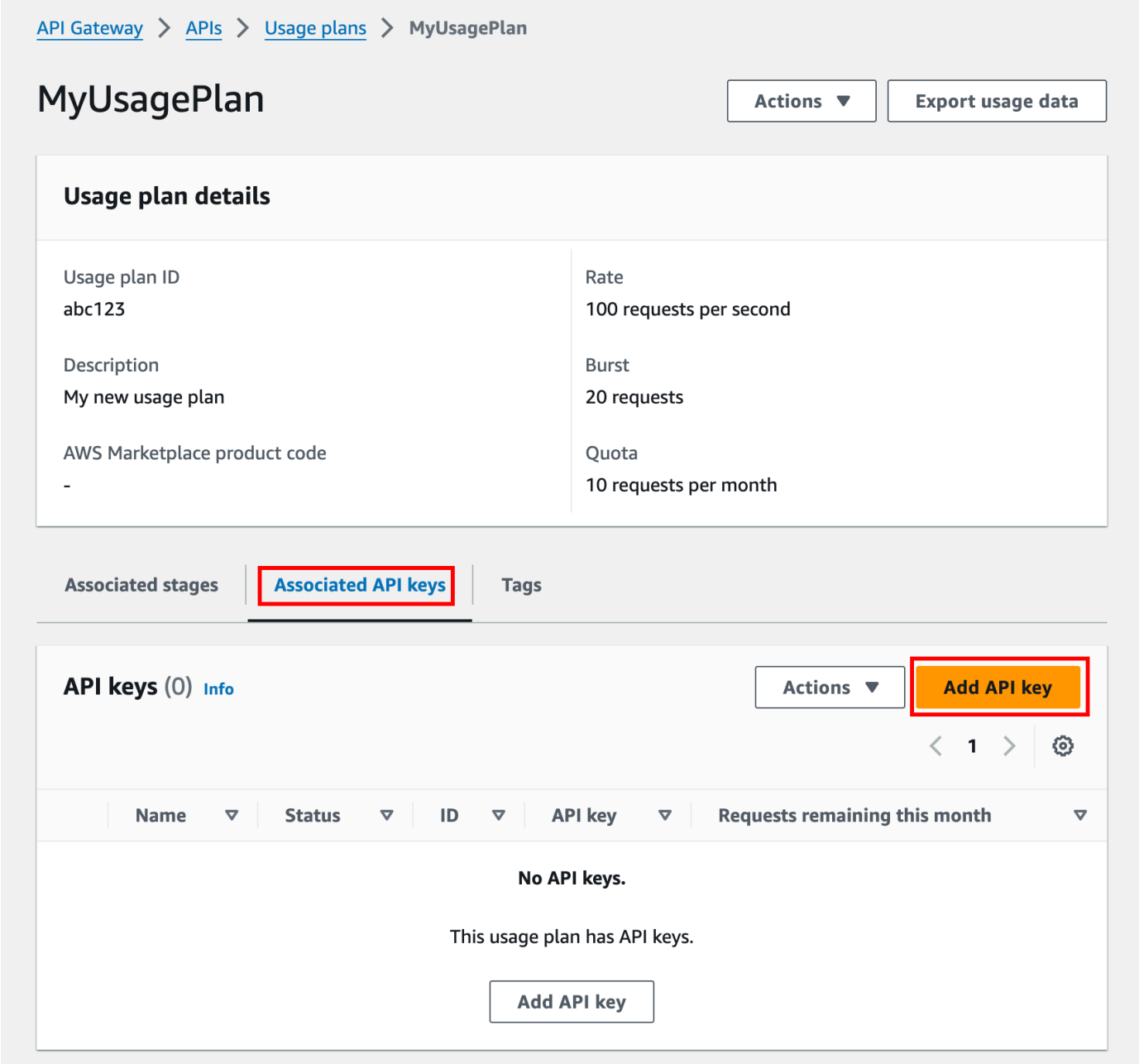
1. 使用プランを選択します。
2. [関連ステージ] タブの [ステージを追加する] を選択します。



3. [API] には API を選択します。
4. [ステージ] では、ステージを選択します。
5. (オプション) メソッドレベルのスロットリングをオンにするには、次の手順を実行します。
   a. [メソッドレベルのスロットリング] を選択し、[メソッドを追加] を選択します。
   b. [リソース] では、API からリソースを選択します。
   c. [メソッド] では、API からメソッドを選択します。
   d. 使用量プランの [レート] と [バースト] を入力します。
6. [使用量プランに追加] を選択します。

使用プランにキーを追加するには

1. [関連付けられた API キー] で、[API キーを追加] を選択します。



2. a. 既存のキーを使用プランに関連付けるには、[既存のキーを追加] を選択し、ドロップダウンメニューから既存のキーを選択します。

   b. 新しい API キーを作成するには、[新しいキーを作成して追加] を選択し、新しいキーを作成します。新しいキーの作成方法の詳細については、「API キーを作成する」を参照してください。

3. [API キーの追加] を選択します。

使用量プランをテストする

使用量プランをテストするには、AWS SDK、AWS CLI、または REST API クライアント (Postman など) を使用できます。使用量プランのテストに Postman を使用する例については、「使用量プランのテスト」を参照してください。

使用量プランをメンテナンスする

使用量プランのメンテナンスには、特定の期間の使用されたクォータおよび残りのクォータのモニタリング、および、必要に応じて、指定された量の残りのクォータの拡張が含まれます。次の手順は、クォータを監視する方法について説明しています。

使用されたクォータおよび残りのクォータを監視するには

1. https://console.aws.amazon.com/apigateway で API Gateway コンソールにサインインします。
2. API Gateway のメインナビゲーションペインで、[使用量プラン] を選択します。
3. 使用プランを選択します。
4. [関連付けられた API キー] タブを選択すると、各キーの期間に残っているリクエスト数が表示されます。
5. (オプション) [使用量データのエクスポート] を選択し、[開始] と [終了] を選択します。次に、エクスポートするデータ形式として [JSON] または [CSV] を選択し、[エクスポート] を選択します。

   次の例では、エクスポートされたファイルの例を示します。

```
{
  "thisPeriod": {
    "px1KW6...qBazOJH": [
      [
        0,
        5000
      ],
      [
        0,
        5000
      ],
      [
        0,
        10
      ]
```

```
    ]
  },
  "startDate": "2016-08-01",
  "endDate": "2016-08-03"
}
```

この例の使用状況データは、API キー (`px1KW6...qBazOJH`) によって識別される、2016 年 8 月 1 日から 2016 年 8 月 3 日の、API クライアントの毎日の使用状況データです。それぞれの毎日の使用状況データは使用されたクォータおよび残りのクォータを表示します。この例では、サブスクライバは割り当てられたクォータをまだ使用していないため、API 所有者または管理者は残りのクォータを 3 日目に 5000 から 10 に削減しました。

次の手順は、クォータを修正する方法を説明します。

残りのクォータを拡張するには

1. https://console.aws.amazon.com/apigateway で API Gateway コンソールにサインインします。
2. API Gateway のメインナビゲーションペインで、[使用量プラン] を選択します。
3. 使用プランを選択します。
4. [関連付けられた API キー] タブを選択すると、各キーの期間に残っているリクエスト数が表示されます。
5. API キーを選択し、[使用期限の延長を付与] を選択します。
6. [残りリクエスト] クォータの数を入力します。使用プランの期間中は、残りのリクエストを増やすことも、残りのリクエストを減らすこともできます。
7. [クォータの更新] を選択します。

API Gateway REST API を使用して API キーをセットアップする

API キーを設定するには、以下の作業を行います。

- API キーを要求するよう API メソッドを設定します。
- リージョンの API 用に API キーを作成またはインポートします。

API キーを設定するには、事前に API を設定し、それをステージにデプロイしている必要があります。API キー値は、作成後に変更することはできません。

API の作成およびデプロイのための REST API コールについては、「restapi:create」および「deployment:create」をそれぞれ参照してください。

> **Note**
>
> 考慮すべきベストプラクティスについては、「the section called “API キーと使用量プランのベストプラクティス”」を参照してください。

トピック

- メソッドで API キーを要求する
- API キーの作成またはインポート

メソッドで API キーを要求する

メソッドの API キーを要求するには、次のいずれかを実行します。

- method:put を呼び出してメソッドを作成します。リクエストペイロードで apiKeyRequired を true に設定します。
- method:update を呼び出して apiKeyRequired を true に設定します。

API キーの作成またはインポート

API キーを作成またはインポートするには、次のいずれかを実行します。

- apikey:create を呼び出して API キーを作成します。
- apikey:import を呼び出して、ファイルから API キーをインポートします。ファイル形式については、「API Gateway API キーファイルの形式」を参照してください。

新しい API キーの値を変更することはできません。使用量プランを設定する方法については、「API Gateway CLI および REST API を使用してテスト使用量プランを作成、設定、およびテストする」を参照してください。

API Gateway CLI および REST API を使用してテスト使用量プランを作成、設定、およびテストする

使用量プランを設定する前に、以下がすでに完了している必要があります。選択した API のメソッドで API キーが要求されるようセットアップ、API をステージにデプロイまたは再デプロイ、1 つ以上の API キーを作成またはインポート。詳細については、「API Gateway REST API を使用して API キーをセットアップする」を参照してください。

API Gateway REST API を使用して使用量プランを設定するには、使用量プランに追加する API を既に作成したものと仮定して、次の手順を実行します。

トピック

- デフォルトの使用量プランへの移行
- 使用量プランを作成する
- AWS CLI を使用して使用量プランを管理する
- 使用量プランのテスト

デフォルトの使用量プランへの移行

最初に使用量プランを作成する場合は、`account:update` を次の本文と共に呼び出して、選択した API キーに関連付けられている既存の API ステージを使用量プランに移行できます。

```
{
  "patchOperations" : [ {
    "op" : "add",
    "path" : "/features",
    "value" : "UsagePlans"
  } ]
}
```

API キーに関連付けられている API ステージの移行については、「API Gateway コンソールでデフォルトの使用量プランに移行する」を参照してください。

使用量プランを作成する

次の手順は、使用量プランを作成する方法を説明します。

REST API を使用して、使用量プランを作成するには

1. usageplan:create を呼び出して、使用量プランを作成します。ペイロードで、プランの名前と説明、関連付けられた API ステージ、レート制限、およびクォータを指定します。

   結果として生じる使用プランの識別子を書き留めます。これは次の手順で必要です。

2. 次のいずれかを行ってください。

   a. usageplankey:create を呼び出して、使用量プランに API キーを追加します。ペイロードで keyId と keyType を指定します。

      使用量プランに API キーを追加するには、一度に 1 つの API キーずつ、前の呼び出しを繰り返します。

   b. apikey:import を呼び出して、指定した使用量プランに 1 つ以上の API キーを直接追加します。リクエストペイロードは API キー値、関連付けられた使用量プランの識別子、キーが使用量プランに有効であることを示すブーリアン型フラグ、および、場合によっては、API キーの名前と説明を含む必要があります。

      apikey:import リクエストの次の例では、3 つの API キー (key、name、および description により識別される) が、1 つの使用量プラン (usageplanIds により識別される) に追加されます。

```
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
```

      その結果、3 つの UsagePlanKey リソースが作成されて、UsagePlan に追加されます。

      この方法で API キーを複数の使用プランに追加することもできます。これを行うには、各 usageplanIds 列の値を、選択した使用量プランの識別子を含むカンマ区切り文字列に変更し、引用符で囲みます ("n371pt,m282qs" または 'n371pt,m282qs')。

> **Note**
>
> API キーを複数の使用量プランと関連付けることができます。使用量プランを複数のステージと関連付けることができます。ただし、指定された API キーは API の各ステージの 1 つの使用量プランにのみ関連付けることができます。

AWS CLI を使用して使用量プランを管理する

次のコード例は、update-usage-plan コマンドを呼び出して、使用量プランのメソッドレベルのスロットリング設定を追加、削除、または変更する方法を示します。

> **Note**
>
> us-east-1 を変更して、API に適切なリージョン値を指定してください。

個々のリソースとメソッドのスロットリングのレート制限を追加または置換するには。

```
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"
```

個々のリソースとメソッドのスロットリングのバースト制限を追加または置換するには。

```
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"
```

個々のリソースとメソッドのメソッドレベルのスロットリング設定を追加または置換するには。

```
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=""
```

API のメソッドレベルのスロットリング設定をすべて削除するには。

```
aws apigateway --region us-east-1 update-usage-plan --usage-plan-id <planId> --patch-
operations op="remove",path="/apiStages/<apiId>:<stage>/throttle ",value=""
```

Pet Store サンプル API を使用する例を次に示します。

```
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}}"'
```

使用量プランのテスト

例として、チュートリアル: サンプルをインポートして REST API を作成する で作成される PetStore API を使用します。API は Hiorr45VR...c4GJc の API キーを使用するように設定されると仮定します。以下の手順では、使用プランをテストする方法について説明します。

使用プランをテストするには

- 使用量プランの API (例: GET) の、/pets クエリパラメータを使用して、Pets リソース (?type=...&page=...) で、xbvxlpijch リクエストを作成します。

  ```
  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**
  >
  > API Gateway の execute-api コンポーネントにこのリクエストを送信して、必要な Hiorr45VR...c4GJc ヘッダーで、必要な API キー (例: x-api-key) を提供する必要があります。

  正常なレスポンスでは、200 OK ステータスコード、およびバックエンドからリクエストされた結果を含むペイロードが返されます。x-api-key ヘッダーの設定を忘れるか、不正なキー

を設定した場合は、403 Forbidden レスポンスが表示されます。ただし、必要な API キーにメソッドを設定しなかった場合は、200 OK ヘッダーを正しく設定したかどうかにかかわらず、x-api-key レスポンスが表示され、使用量プランのスロットリングとクォータ制限はバイパスされます。

API Gateway がリクエストに使用量プランスロットリング制限またはクォータを適用できない内部エラーが発生することがあります。この場合、API Gateway では使用量プランで指定されたスロットリング制限またはクォータを適用せずにリクエストを処理します。ただし、CloudWatch に Usage Plan check failed due to an internal error のエラーメッセージが記録されます。ときどき起こるこのようなエラーは無視してかまいません。

AWS CloudFormation を使用した API キーと使用量プランの作成および設定

AWS CloudFormation を使用して API メソッドで API キーを要求したり、API の使用量プランを作成したりできます。この例の AWS CloudFormation テンプレートでは、次のような処理を実行します。

- GET および POST メソッドを使用して API Gateway API を作成します。
- GET および POST メソッドの API キーが必要です。この API は、受信する各リクエストの X-API-KEY ヘッダーからキーを受け取ります。
- API キーを作成します。
- 毎月のクォータを毎月 1,000 リクエスト、スロットリングレート制限を 1 秒あたり 100 リクエスト、スロットリングバースト制限を 1 秒あたり 200 リクエストに指定する使用量プランを作成します。
- GET メソッドに対して、メソッドレベルのスロットリングレート制限を 1 秒あたり 50 リクエストに、メソッドレベルのスロットリングバースト制限を 1 秒あたり 100 リクエストに指定します。
- 使用量プランに、API ステージと API キーを関連付けます。

```
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}'
```

OpenAPI 定義で API キーを使用するようにメソッドを設定する

OpenAPI 定義を使用してメソッドで API キーを要求できます。

メソッドごとに、メソッドを呼び出す API キーを要求するためのセキュリティ要件オブジェクトを作成します。次に、セキュリティ定義で api_key を定義します。API キーを作成したら、新しい API ステージを使用量プランに追加します。

次の例では API を作成し、POST メソッドと GET メソッドで API キーを要求します。

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 キーファイルの形式

API Gateway はカンマ区切り値 (CSV) 形式の外部ファイルから API キーをインポートでき、インポートされたキーを 1 つ以上の使用量プランに関連付けます。インポートされたファイルには、Name および Key 列が含まれている必要があります。次の例のように、列のヘッダー名は大文字小文字を区別せず、順序は任意です。

```
Key,name
apikey1234abcdefghij0123456789,MyFirstApiKey
```

Key 値は、20 ~ 128 文字にする必要があります。Name 値は 1024 文字を超えることはできません。

次の例のように、API キーファイルには、Description、Enabled、または UsagePlanIds 列もあります。

```
Name,key,description,Enabled,usageplanIds
MyFirstApiKey,apikey1234abcdefghij0123456789,An imported key,TRUE,c7y23b
```

次の例のように、キーが複数の使用量プランに関連付けられる場合、UsagePlanIds 値は、二重引用符または一重引用符で囲まれたカンマ区切り文字列の使用量プラン ID です。

```
Enabled,Name,key,UsageplanIds
true,MyFirstApiKey,apikey1234abcdefghij0123456789,"c7y23b,glvrsr"
```

認識されない列は許可されますが、無視されます。デフォルト値は、空の文字列または `true` ブール値です。

同じ API キーを複数回インポートすることができ、最新バージョンは前のキーを上書きします。`key` 値が同じ場合、2 つの API キーは同一です。

> **Note**
>
> 考慮すべきベストプラクティスについては、「the section called “API キーと使用量プランのベストプラクティス”」を参照してください。

REST API をドキュメント化する

顧客が API について理解して使いやすくするため、API をドキュメント化してください。API をドキュメント化しやすいように、API Gateway では API 開発プロセスの不可欠な部分として個々の API エンティティのヘルプコンテンツを追加および更新できます。API Gateway にはソースコンテンツが保存され、さまざまなバージョンのドキュメントをアーカイブできます。ドキュメントバージョンと API ステージを関連付けたり、ステージ固有のドキュメントスナップショットを外部 OpenAPI ファイルにエクスポートしたり、ファイルをドキュメントの出版物として配布したりすることができます。

API をドキュメント化するには、API Gateway REST API を呼び出すか、AWS SDK のいずれかを使用するか、API Gateway 用の AWS CLI を使用するか、API Gateway コンソールを使用できます。加えて、外部 OpenAPI ファイルで定義されたドキュメントパーツをインポートまたはエクスポートすることができます。

API ドキュメントをデベロッパーと共有するには、デベロッパーポータルを使用できます。例については、AWS パートナーネットワーク (APN) ブログの「Integrating ReadMe with API Gateway to Keep Your Developer Hub Up to Date」を参照してください。

トピック

- API Gateway での API ドキュメントの表現
- API Gateway コンソールを使用して API を文書化する
- API Gateway コンソールを使用して API ドキュメントを公開する

- API Gateway REST API を使用して API を文書化する
- API Gateway REST API を使用した API ドキュメントを公開する
- API ドキュメントのインポート
- API ドキュメントへのアクセスの制御

API Gateway での API ドキュメントの表現

API Gateway API ドキュメントは、API、リソース、メソッド、リクエスト、レスポンス、メッセージパラメータ (つまり、パス、クエリ、ヘッダー) と、オーソライザーおよびモデルを含む特定の API エンティティと関連付けられた個々のドキュメントパートで構成されます。

API Gateway では、ドキュメントパートは DocumentationPart リソースにより表現されます。API ドキュメント全体は、DocumentationParts コレクションとして表現されます。

API をドキュメント化するには、`DocumentationPart` インスタンスを作成して `DocumentationParts` コレクションに追加し、API の展開に応じてドキュメントパーツのバージョンを維持することが必要です。

トピック

- ドキュメントパーツ
- ドキュメントバージョン

ドキュメントパーツ

DocumentationPart リソースは、個々の API エンティティに適用されるドキュメントコンテンツが保存される JSON オブジェクトです。その `properties` フィールドには、ドキュメントコンテンツがキー/値ペアのマップとして含まれています。その `location` プロパティは、関連付けられた API エンティティを識別します。

コンテンツマップのシェイプは、API デベロッパーであるお客様が決定します。キー/値ペアの値は、文字列、数値、ブール値、オブジェクト、配列のいずれかです。`location` オブジェクトのシェイプは、ターゲットとなるエンティティタイプによって異なります。

`DocumentationPart` リソースでは、コンテンツの継承がサポートされます。API エンティティのドキュメントコンテンツは、その API エンティティの子に適用されます。子エンティティとコンテンツ継承の定義の詳細については、「より一般的な仕様の API エンティティからのコンテンツの継承」を参照してください。

ドキュメントパーツの場所

DocumentationPart インスタンスの location プロパティは、関連付けられたコンテンツが適用される API エンティティを識別します。API エンティティは、RestApi、Resource、Method、MethodResponse、Authorizer、Model などの API Gateway REST API リソースです。エンティティは、URL パスパラメーター、クエリ文字列パラメーター、リクエストまたはレスポンスヘッダーパラメーター、リクエストまたはレスポンス本部、レスポンスステータスコードなどのメッセージパラメーターでもかまいません。

API エンティティを指定するには、location オブジェクトの type 属性を API、AUTHORIZER、MODEL、RESOURCE、METHOD、PATH_PARAMETER、QUERY_PARAMETER、REQUEST_ のいずれかに設定します。

API エンティティの type によっては、method、name、path、statusCode などの他の location 属性を指定する必要があります。これらのすべての属性が特定の API エンティティで有効なわけではありません。たとえば、type, path、name、statusCode は RESPONSE エンティティの有効な属性です。type エンティティの有効な location 属性は path と RESOURCE だけです。特定の API エンティティの location の DocumentationPart に無効なフィールドを含めるとエラーになります。

有効な location フィールドがすべて必須なわけではありません。たとえば、type は、すべての API エンティティで有効かつ必須の location フィールドです。一方、method、path、statusCode は、RESPONSE エンティティで有効ですが、必須の属性ではありません。明示的に指定されていない場合、有効な location フィールドではそのデフォルト値が使用されます。path のデフォルト値は /、つまり API のルートリソースです。method または statusCode のデフォルト値は * です。これは、ぞれぞれ任意のメソッドまたはステータスコードを意味します。

ドキュメントパーツのコンテンツ

properties 値は、JSON 文字列としてエンコードされます。properties 値には、ドキュメント要件を満たすために選択した情報がすべて含まれます。たとえば、有効なコンテンツマップを以下に示します。

```
{
  "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."
```

```
}
```

API Gateway は任意の有効な JSON 文字列をコンテンツマップとして受け入れますが、コンテンツ属性は 2 つのカテゴリ (OpenAPI が認識できるカテゴリと認識できないカテゴリ) として扱われます。前の例では、`info`、`description`、`x-custom-info` が OpenAPI により標準の OpenAPI オブジェクト、プロパティ、または拡張として認識されます。一方、`my-info` は OpenAPI 仕様に準拠していません。API Gateway は、OpenAPI 準拠のコンテンツ属性を、関連付けられた `DocumentationPart` インスタンスから API エンティティ定義に伝達します。API Gateway は、非準拠のコンテンツ属性を API エンティティ定義に伝達しません。

別の例として、`DocumentationPart` エンティティのターゲットとなった `Resource` を示します。

```
{
    "location" : {
        "type" : "RESOURCE",
        "path": "/pets"
    },
    "properties" : {
        "summary" : "The /pets resource represents a collection of pets in PetStore.",
        "description": "... a child resource under the root...",
    }
}
```

ここでは、`type` と `path` の両方が `RESOURCE` タイプを識別するための有効なフィールドです。ルートリソース (`/`) では、`path` フィールドを省略できます。

```
{
    "location" : {
        "type" : "RESOURCE"
    },
    "properties" : {
        "description" : "The root resource with the default path specification."
    }
}
```

これは、次の `DocumentationPart` インスタンスと同じです。

```
{
    "location" : {
        "type" : "RESOURCE",
        "path": "/"
```

```
    },
    "properties" : {
        "description" : "The root resource with an explicit path specification"
    }
}
```

より一般的な仕様の API エンティティからのコンテンツの継承

オプションの location フィールドのデフォルト値では、API エンティティのパターン化された説明が提供されます。location オブジェクトでデフォルト値を使用すると、properties マップ内の全般的な説明を、このタイプの DocumentationPart パターンを持つ location インスタンスに追加できます。API Gateway は、汎用 API エンティティの DocumentationPart から該当する OpenAPI ドキュメント属性を抽出し、一般的な location パターンに一致するか、正確な値に一致する location フィールドを持つ特定の API エンティティに挿入します。ただし、特定のエンティティに DocumentationPart インスタンスが既に関連付けられている場合を除きます。この動作は、より全般的な仕様の API エンティティからのコンテンツ継承とも呼ばれます。

コンテンツ継承は、特定の API エンティティのタイプには適用されません。詳細については以下の表を参照してください。

API エンティティが複数の DocumentationPart の位置パターンに一致する場合、そのエンティティは優先度と具体性が最も高い位置フィールドを持つドキュメントパーツを継承します。優先順位は、path、statusCode の順です。path フィールドと一致させるため、API Gateway は最も具体的なパス値を持つエンティティを選択します。次の表に、いくつかの例とともにこれを示します。

ケース	path	statusCode	name	解説
1	/pets	*	id	この位置パターンに関連付けられたドキ

ケース	path	statusCode	name	解説	
				ユメントは、位置パターンに一致するエンティティによって継承されます。	

ケース	path	statusCode	name	解説	
2	/pets	200	id	この位置パターンに関連付けられたドキュメントは、ケース 1 とケース 2 の両方が一致する場合に位置パターンに一致するエンティティに	

ケース	path	statusCode	name	解説	
				よって継承されます。ケース 2 の方がケース 1 より具体的だからです。	

ケース	path	statusCode	name	解説
3	/pets/petId	*	id	この位置パターンに関連付けられたドキュメントは、ケース 1、2、および 3 が一致する場合に位置パターンに一致するエンティティによっ

ケース	path	statusCode	name	解説
				て継承されます。ケース 3 の方がケース 2 より優先度が高く、ケース 1 より具体的だからです。

ここでは、汎用的な DocumentationPart インスタンスと具体的なインスタンスの違いを別の例で示します。上書きされていない限り、全般的なエラーメッセージ "Invalid request error" が 400 エラーレスポンスの OpenAPI 定義に挿入されます。

```
{
    "location" : {
        "type" : "RESPONSE",
        "statusCode": "400"
    },
    "properties" : {
```

```
        "description" : "Invalid request error."
    }"
}
```

次のように上書きすると、400 リソースでのすべてのメソッドに対する /pets レスポンスに、説明 "Invalid petId specified" が代わりに挿入されます。

```
{
    "location" : {
        "type" : "RESPONSE",
        "path": "/pets",
        "statusCode": "400"
    },
    "properties" : "{
        "description" : "Invalid petId specified."
    }"
}
```

DocumentationPart の有効な位置フィールド

次の表に、特定のタイプの API エンティティに関連付けられた DocumentationPart リソースの有効なフィールド、必須のフィールド、および適用可能なデフォルト値を示します。

API エンティティ	有効な位置フィールド	必須の位置フィールド	デフォルトフィールド値	継承可能なコンテンツ
API	`{ "location": { "type": "API" }, ... }`	type	該当なし	いいえ
[リソース]	`{ "location": { "type": "RESOURCE", "path": "resource_path " }, ...`	type	path の初期値はです。/	いいえ

API エンティティ	有効な位置フィールド	必須の位置フィールド	デフォルトフィールド値	継承可能なコンテンツ
	... }			

方法	{
	"location": {
	"type": "METHOD",
	"path": "*resource_path* ",
	"method": "*http_verb* "
	},
	...
	}
		type	path と method のデフォルト値は、それぞれ / と * です。	はい。プレフィックスにより path に一致し、任意の値の method に一致します。

クエリパラメーター	{
	"location": {
	"type": "QUERY_PARAMETER",
	"path": "*resource_path* ",
	"method": "*HTTP_verb* ",
	"name": "*query_parameter_name* "
	},
	...
	}
		type	path と method のデフォルト値は、それぞれ / と * です。	はい。プレフィックスにより path に一致し、正確な値により method に一致します。

API エンティティ	有効な位置フィールド	必須の位置フィールド	デフォルトフィールド値	継承可能なコンテンツ
リクエストボディ	`{ "location": { "type": "REQUEST_BODY", "path": "resource_path ", "method": "http_verb " }, ... }`	type	path と method のデフォルト値は、それぞれ / と * です。	はい。プレフィックスにより path に一致し、正確な値により method に一致します。
リクエストヘッダーパラメーター	`{ "location": { "type": "REQUEST_HEADER", "path": "resource_path ", "method": "HTTP_verb ", "name": "header_name " }, ... }`	type, name	path と method のデフォルト値は、それぞれ / と * です。	はい。プレフィックスにより path に一致し、正確な値により method に一致します。

API エンティティ	有効な位置フィールド	必須の位置フィールド	デフォルトフィールド値	継承可能なコンテンツ

リクエストパスパラメーター	{
	"location": {
	"type": "PATH_PARAMETER",
	"path": "*resource*/{*path_parameter_name* }",
	"method": "*HTTP_verb* ",
	"name": "*path_parameter_name* "
	},
	...
	}
		type, name	path と method のデフォルト値は、それぞれ / と * です。	はい。プレフィックスにより path に一致し、正確な値により method に一致します。

レスポンス	{
	"location": {
	"type": "RESPONSE",
	"path": "*resource_path* ",
	"method": "*http_verb* ",
	"statusCode": "*status_code* "
	},
	...
	}
		type	path、method、statusCode のデフォルト値は、それぞれ /、*、* です。	はい。プレフィックスにより path に一致し、正確な値により method と statusCode に一致します。

API エンティティ	有効な位置フィールド	必須の位置フィールド	デフォルトフィールド値	継承可能なコンテンツ
レスポンスヘッダー	{ "location": { "type": "RESPONSE_HEADER", "path": "*resource_path* ", "method": "*http_verb* ", "statusCode": "*status_code* ", "name": "*header_name* " }, ... }	type, name	path、method、status のデフォルト値は、それぞれ /、*、* です。	はい。プレフィックスにより path に一致し、正確な値により method と statusCode に一致します。
レスポンス本文	{ "location": { "type": "RESPONSE_BODY", "path": "*resource_path* ", "method": "*http_verb* ", "statusCode": "*status_code* " }, ... }	type	path、method、status のデフォルト値は、それぞれ /、*、* です。	はい。プレフィックスにより path に一致し、正確な値により method と statusCode に一致します。

API エンティティ	有効な位置フィールド	必須の位置フィールド	デフォルトフィールド値	継承可能なコンテンツ
オーソライザー	`{ "location": { "type": "AUTHORIZER", "name": "authorizer_name " }, ... }`	type	該当なし	いいえ
[Model] (モデル)	`{ "location": { "type": "MODEL", "name": "model_name " }, ... }`	type	該当なし	いいえ

ドキュメントバージョン

ドキュメントバージョンは、API の DocumentationParts コレクションのスナップショットであり、バージョン識別子でタグ付けされます。API のドキュメントを発行するには、ドキュメントバージョンを作成して API ステージに関連付け、そのステージ固有のバージョンの API ドキュメントを外部 OpenAPI ファイルにエクスポートします。API Gateway では、ドキュメントスナップショットが DocumentationVersion リソースとして表現されます。

API を更新するとき、新しいバージョンの API を作成します。API Gateway では、DocumentationVersions コレクションを使用してすべてのドキュメントバージョンを維持します。

API Gateway コンソールを使用して API を文書化する

このセクションでは、API Gateway コンソールを使用して API のドキュメントパートを作成および維持する方法について説明します。

API のドキュメントを作成および編集するための前提条件は、すでに API を作成していることです。このセクションでは、例として PetStore API を使用します。API Gateway コンソールを使用して API を作成するには、「チュートリアル: サンプルをインポートして REST API を作成する」の手順に従います。

トピック

- API エンティティのドキュメント化
- RESOURCE エンティティのドキュメント化
- METHOD エンティティのドキュメント化
- QUERY_PARAMETER エンティティのドキュメント化
- PATH_PARAMETER エンティティのドキュメント化
- REQUEST_HEADER エンティティのドキュメント化
- REQUEST_BODY エンティティのドキュメント化
- RESPONSE エンティティのドキュメント化
- RESPONSE_HEADER エンティティのドキュメント化
- RESPONSE_BODY エンティティのドキュメント化
- MODEL エンティティのドキュメント化
- AUTHORIZER エンティティのドキュメント化

API エンティティのドキュメント化

API エンティティの新しいドキュメントパーツを追加するには、以下を実行します。

1. メインナビゲーションペインで [ドキュメント] を選択し、[ドキュメントパーツの作成] を選択します。
2. [ドキュメントタイプ] には [API] を選択します。

   API のドキュメントパーツが作成されなかった場合、ドキュメントパーツの `properties` マップエディターを取得します。テキストエディタに次の `properties` マップを入力します。

```
{
  "info": {
    "description": "Your first API Gateway API.",
    "contact": {
        "name": "John Doe",
        "email": "john.doe@api.com"
```

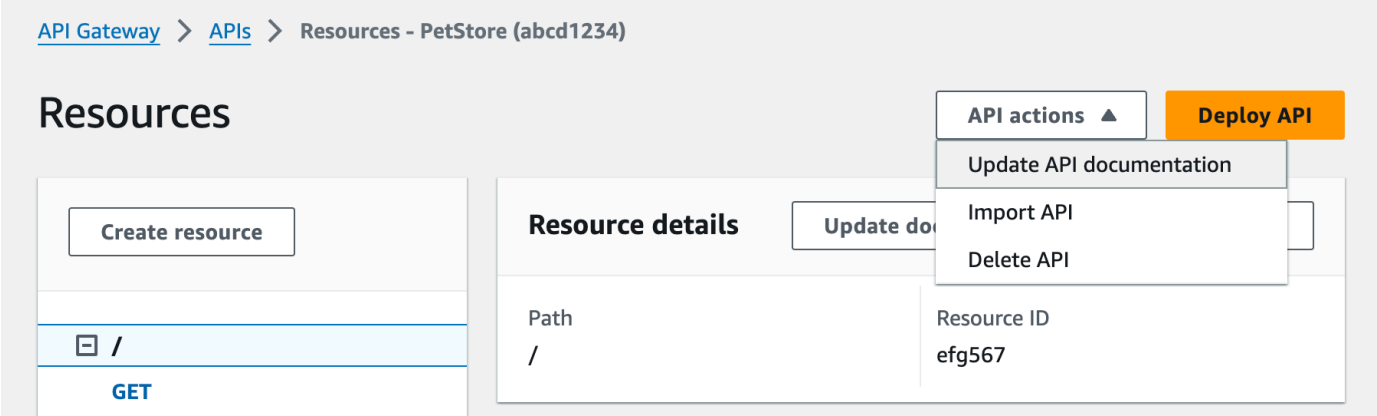
```
    }
  }
}
```

> **Note**
>
> `properties` マップを JSON 文字列にエンコードする必要はありません。API Gateway コンソールにより、JSON オブジェクトが自動的に stringify されます。

3. [ドキュメントパーツの作成] を選択します。

[リソース] ペインに API エンティティの新しいドキュメントパーツを追加するには、以下を実行します。

1. メインナビゲーションペインで、[リソース] を選択します。
2. [API アクション] メニューを選択し、[API ドキュメントの更新] を選択します。



既存のドキュメントパーツを編集するには、以下を実行します。

1. [ドキュメント] ペインで、[リソースとメソッド] タブを選択します。
2. API の名前を選択し、API カードで [編集] を選択します。

RESOURCE エンティティのドキュメント化

RESOURCE エンティティの新しいドキュメントパーツを追加するには、以下を実行します。

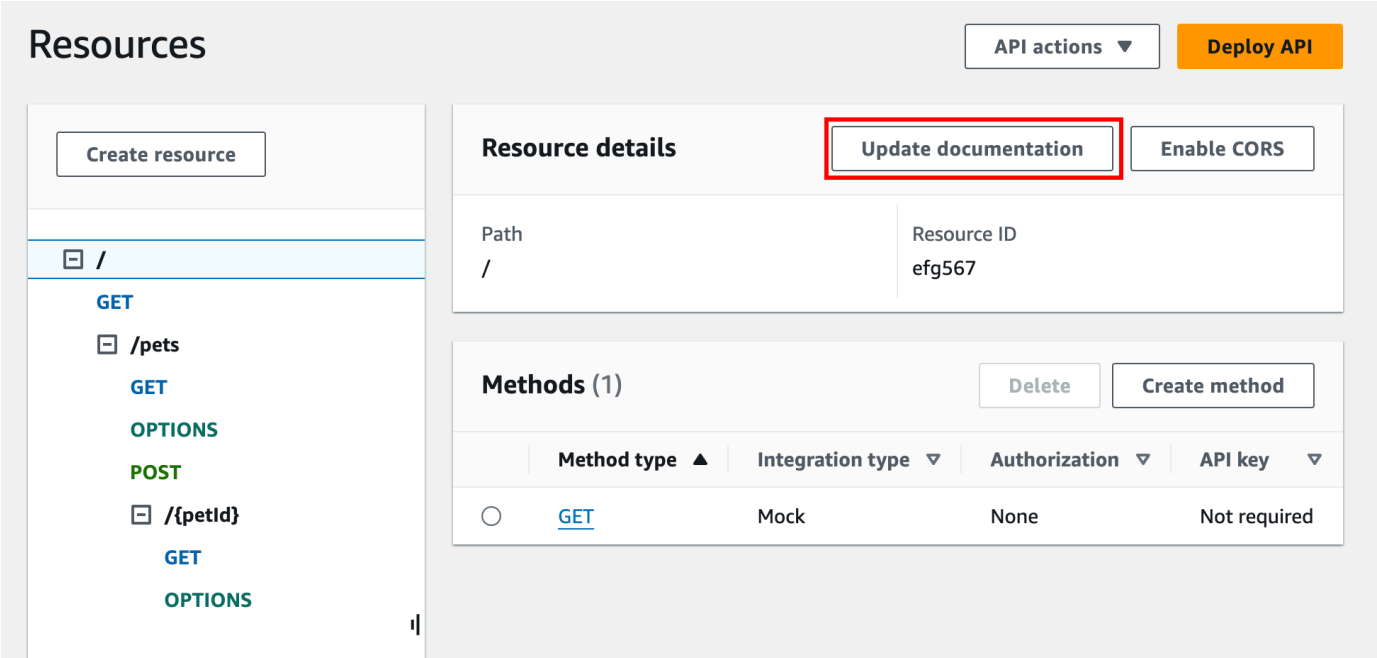
1. メインナビゲーションペインで [ドキュメント] を選択し、[ドキュメントパーツの作成] を選択します。
2. [ドキュメントタイプ] には [リソース] を選択します。
3. [パス] にはパスを入力します。
4. テキストエディタに説明を入力します。例:

```
{
    "description": "The PetStore's root resource."
}
```

5. [ドキュメントパーツの作成] を選択します。リストにないリソースのドキュメントを作成できます。
6. 必要に応じて、これらのステップを繰り返し、他のドキュメントパーツを追加するか、編集します。

[リソース] ペインに RESOURCE エンティティの新しいドキュメントパーツを追加するには、以下を実行します。

1. メインナビゲーションペインで、[リソース] を選択します。
2. リソースを選択し、[ドキュメントの更新] を選択します。



既存のドキュメントパーツを編集するには、以下を実行します。

1. [ドキュメント] ペインで、[リソースとメソッド] タブを選択します。
2. ドキュメントパーツを含むリソースを選択し、[編集] を選択します。

METHOD エンティティのドキュメント化

METHOD エンティティの新しいドキュメントパーツを追加するには、以下を実行します。

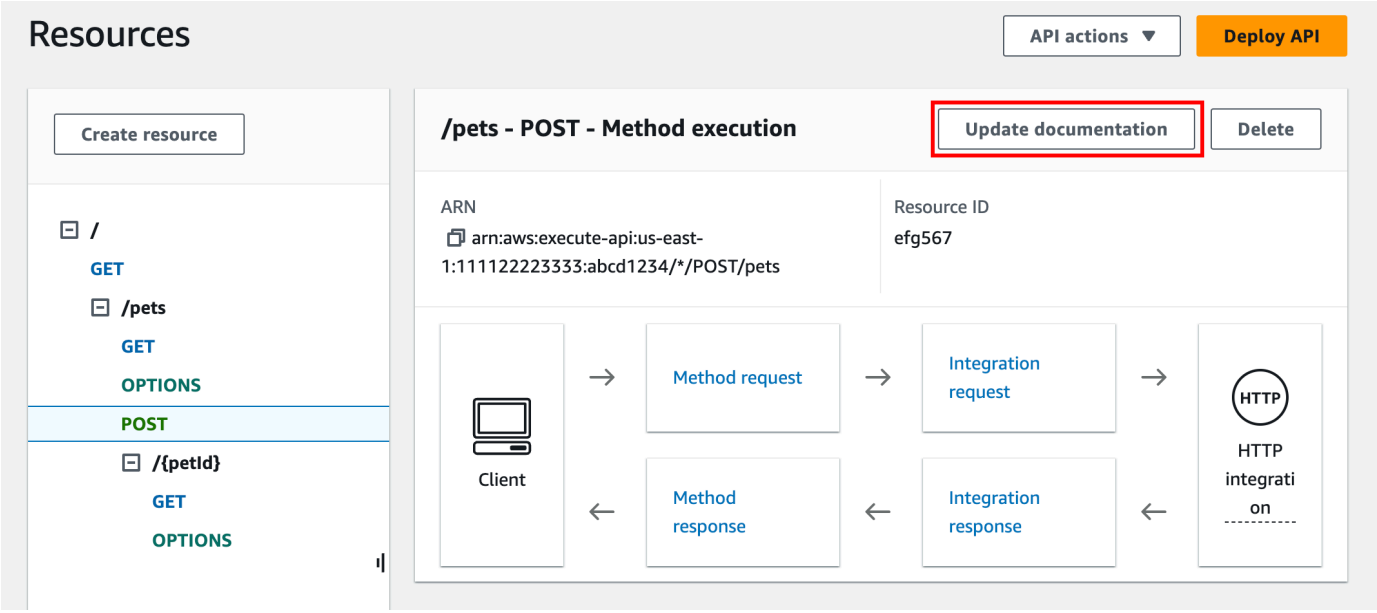
1. メインナビゲーションペインで [ドキュメント] を選択し、[ドキュメントパーツの作成] を選択します。
2. [ドキュメントタイプ] には [メソッド] を選択します。
3. [パス] にはパスを入力します。
4. [メソッド] で、HTTP 動詞を選択します。
5. テキストエディタに説明を入力します。例:

```
{
  "tags" : [ "pets" ],
  "summary" : "List all pets"
}
```

6. [ドキュメントパーツの作成] を選択します。リストにないメソッドのドキュメントを作成できます。
7. 必要に応じて、これらのステップを繰り返し、他のドキュメントパーツを追加するか、編集します。

[リソース] ペインに METHOD エンティティの新しいドキュメントパーツを追加するには、以下を実行します。

1. メインナビゲーションペインで、[リソース] を選択します。
2. メソッドを選択し、[ドキュメントの更新] を選択します。



既存のドキュメントパーツを編集するには、以下を実行します。

1. [ドキュメント] ペインで、[リソースとメソッド] タブを選択します。
2. メソッドを選択するか、メソッドを含むリソースを選択してから、検索バーを使用してドキュメントパーツを検索および選択できます。
3. [編集] を選択します。

QUERY_PARAMETER エンティティのドキュメント化

QUERY_PARAMETER エンティティの新しいドキュメントパーツを追加するには、以下を実行します。

1. メインナビゲーションペインで [ドキュメント] を選択し、[ドキュメントパーツの作成] を選択します。
2. [ドキュメントタイプ] には [クエリパラメータ] を選択します。
3. [パス] にはパスを入力します。
4. [メソッド] で、HTTP 動詞を選択します。
5. [ロール名] に名前を入力します。
6. テキストエディタに説明を入力します。
7. [ドキュメントパーツの作成] を選択します。リストにないクエリパラメータのドキュメントを作成できます。

8. 必要に応じて、これらのステップを繰り返し、他のドキュメントパーツを追加するか、編集します。

既存のドキュメントパーツを編集するには、以下を実行します。

1. [ドキュメント] ペインで、[リソースとメソッド] タブを選択します。
2. クエリパラメータを選択するか、クエリパラメータを含むリソースを選択してから、検索バーを使用してドキュメントパーツを検索および選択できます。
3. [編集] を選択します。

PATH_PARAMETER エンティティのドキュメント化

PATH_PARAMETER エンティティの新しいドキュメントパーツを追加するには、以下を実行します。

1. メインナビゲーションペインで [ドキュメント] を選択し、[ドキュメントパーツの作成] を選択します。
2. [ドキュメントタイプ] には [パスパラメータ] を選択します。
3. [パス] にはパスを入力します。
4. [メソッド] で、HTTP 動詞を選択します。
5. [ロール名] に名前を入力します。
6. テキストエディタに説明を入力します。
7. [ドキュメントパーツの作成] を選択します。リストにないパスパラメータのドキュメントを作成できます。
8. 必要に応じて、これらのステップを繰り返し、他のドキュメントパーツを追加するか、編集します。

既存のドキュメントパーツを編集するには、以下を実行します。

1. [ドキュメント] ペインで、[リソースとメソッド] タブを選択します。
2. パスパラメータを選択するか、パスパラメータを含むリソースを選択してから、検索バーを使用してドキュメントパーツを検索および選択できます。
3. [編集] を選択します。

REQUEST_HEADER エンティティのドキュメント化

REQUEST_HEADER エンティティの新しいドキュメントパーツを追加するには、以下を実行します。

1. メインナビゲーションペインで [ドキュメント] を選択し、[ドキュメントパーツの作成] を選択します。
2. [ドキュメントタイプ] には [リクエストヘッダー] を選択します。
3. [パス] には、リクエストヘッダーのパスを入力します。
4. [メソッド] で、HTTP 動詞を選択します。
5. [ロール名] に名前を入力します。
6. テキストエディタに説明を入力します。
7. [ドキュメントパーツの作成] を選択します。リストにないリクエストヘッダーのドキュメントを作成できます。
8. 必要に応じて、これらのステップを繰り返し、他のドキュメントパーツを追加するか、編集します。

既存のドキュメントパーツを編集するには、以下を実行します。

1. [ドキュメント] ペインで、[リソースとメソッド] タブを選択します。
2. リクエストヘッダーを選択するか、リクエストヘッダーを含むリソースを選択してから、検索バーを使用してドキュメントパーツを検索および選択できます。
3. [編集] を選択します。

REQUEST_BODY エンティティのドキュメント化

REQUEST_BODY エンティティの新しいドキュメントパーツを追加するには、以下を実行します。

1. メインナビゲーションペインで [ドキュメント] を選択し、[ドキュメントパーツの作成] を選択します。
2. [ドキュメントタイプ] には [リクエスト本文] を選択します。
3. [パス] には、リクエスト本文のパスを入力します。
4. [メソッド] で、HTTP 動詞を選択します。
5. テキストエディタに説明を入力します。
6. [ドキュメントパーツの作成] を選択します。リストにないリクエスト本文のドキュメントを作成できます。

7. 必要に応じて、これらのステップを繰り返し、他のドキュメントパーツを追加するか、編集します。

既存のドキュメントパーツを編集するには、以下を実行します。

1. [ドキュメント] ペインで、[リソースとメソッド] タブを選択します。
2. リクエスト本文を選択するか、リクエスト本文を含むリソースを選択してから、検索バーを使用してドキュメントパーツを検索および選択できます。
3. [編集] を選択します。

RESPONSE エンティティのドキュメント化

RESPONSE エンティティの新しいドキュメントパーツを追加するには、以下を実行します。

1. メインナビゲーションペインで [ドキュメント] を選択し、[ドキュメントパーツの作成] を選択します。
2. [ドキュメントタイプ] には、[レスポンス (ステータスコード)] を選択します。
3. [パス] には、レスポンスのパスを入力します。
4. [メソッド] で、HTTP 動詞を選択します。
5. [ステータスコード] には、HTTP ステータスコードを入力します。
6. テキストエディタに説明を入力します。
7. [ドキュメントパーツの作成] を選択します。リストにないレスポンスステータスコードのドキュメントを作成できます。
8. 必要に応じて、これらのステップを繰り返し、他のドキュメントパーツを追加するか、編集します。

既存のドキュメントパーツを編集するには、以下を実行します。

1. [ドキュメント] ペインで、[リソースとメソッド] タブを選択します。
2. レスポンスステータスコードを選択するか、レスポンスステータスコードを含むリソースを選択してから、検索バーを使用してドキュメントパーツを検索および選択できます。
3. [編集] を選択します。

RESPONSE_HEADER エンティティのドキュメント化

RESPONSE_HEADER エンティティの新しいドキュメントパーツを追加するには、以下を実行します。

1. メインナビゲーションペインで [ドキュメント] を選択し、[ドキュメントパーツの作成] を選択します。
2. [ドキュメントタイプ] には [レスポンスヘッダー] を選択します。
3. [パス] には、レスポンスヘッダーのパスを入力します。
4. [メソッド] で、HTTP 動詞を選択します。
5. [ステータスコード] には、HTTP ステータスコードを入力します。
6. テキストエディタに説明を入力します。
7. [ドキュメントパーツの作成] を選択します。リストにないレスポンスヘッダーのドキュメントを作成できます。
8. 必要に応じて、これらのステップを繰り返し、他のドキュメントパーツを追加するか、編集します。

既存のドキュメントパーツを編集するには、以下を実行します。

1. [ドキュメント] ペインで、[リソースとメソッド] タブを選択します。
2. レスポンスヘッダーを選択するか、レスポンスヘッダーを含むリソースを選択してから、検索バーを使用してドキュメントパーツを検索および選択できます。
3. [編集] を選択します。

RESPONSE_BODY エンティティのドキュメント化

RESPONSE_BODY エンティティの新しいドキュメントパーツを追加するには、以下を実行します。

1. メインナビゲーションペインで [ドキュメント] を選択し、[ドキュメントパーツの作成] を選択します。
2. [ドキュメントタイプ] には [レスポンス本文] を選択します。
3. [パス] には、レスポンス本文のパスを入力します。
4. [メソッド] で、HTTP 動詞を選択します。
5. [ステータスコード] には、HTTP ステータスコードを入力します。
6. テキストエディタに説明を入力します。

7. [ドキュメントパーツの作成] を選択します。リストにないレスポンス本文のドキュメントを作成できます。
8. 必要に応じて、これらのステップを繰り返し、他のドキュメントパーツを追加するか、編集します。

既存のドキュメントパーツを編集するには、以下を実行します。

1. [ドキュメント] ペインで、[リソースとメソッド] タブを選択します。
2. レスポンス本文を選択するか、レスポンス本文を含むリソースを選択してから、検索バーを使用してドキュメントパーツを検索および選択できます。
3. [編集] を選択します。

MODEL エンティティのドキュメント化

MODEL エンティティをドキュメント化するには、モデルの DocumentPart インスタンスと、モデルの各 properties' を作成および管理する必要があります。たとえば、各 API に付属する Error モデルには、デフォルトでスキーマ定義

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

があり、2 つの DocumentationPart インスタンス (1 つは Model 用、もう 1 つはその message プロパティ用) が必要です。

```
{
  "location": {
    "type": "MODEL",
    "name": "Error"
  },
  "properties": {
    "title": "Error Schema",
    "description": "A description of the Error model"
  }
```

```
}
```

および

```
{
  "location": {
    "type": "MODEL",
    "name": "Error.message"
  },
  "properties": {
    "description": "An error message."
  }
}
```

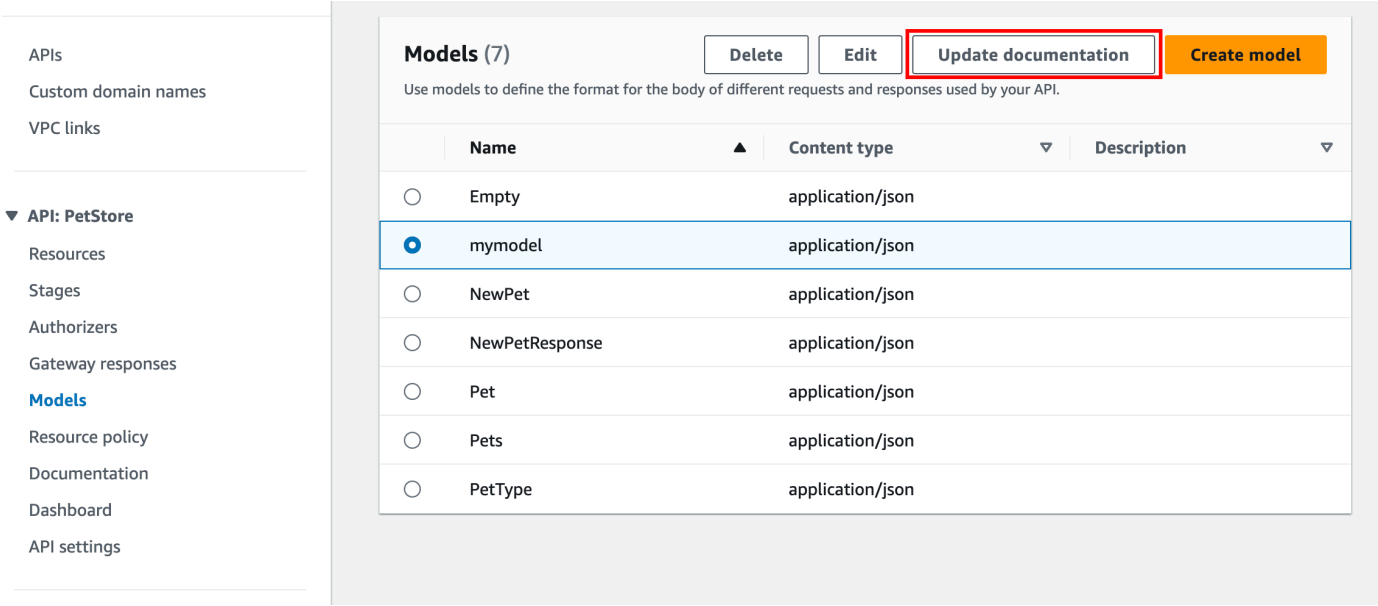
API がエクスポートされると、DocumentationPart のプロパティにより元のスキーマの値が上書きされます。

MODEL エンティティの新しいドキュメントパーツを追加するには、以下を実行します。

1. メインナビゲーションペインで [ドキュメント] を選択し、[ドキュメントパーツの作成] を選択します。
2. [ドキュメントタイプ] には [モデル] を選択します。
3. [名前] に、モデルの名前を入力します。
4. テキストエディタに説明を入力します。
5. [ドキュメントパーツの作成] を選択します。リストにないモデルのドキュメントを作成できます。
6. 必要に応じて、これらのステップを繰り返し、他のモデルにドキュメントパーツを追加するか、編集します。

[モデル] ペインに MODEL エンティティの新しいドキュメントパーツを追加するには、以下を実行します。

1. ナビゲーションペインで、[モデル] を選択します。
2. モデルを選択し、[ドキュメントの更新] を選択します。



既存のドキュメントパーツを編集するには、以下を実行します。

1. [ドキュメント] ペインで [モデル] タブを選択します。
2. 検索バーを使用するかモデルを選択し、[編集] を選択します。

AUTHORIZER エンティティのドキュメント化

AUTHORIZER エンティティの新しいドキュメントパーツを追加するには、以下を実行します。

1. メインナビゲーションペインで [ドキュメント] を選択し、[ドキュメントパーツの作成] を選択します。
2. [ドキュメントタイプ] には [オーソライザー] を選択します。
3. [名前] に、オーソライザーの名前を入力します。
4. テキストエディタに説明を入力します。オーソライザーの有効な [location] フィールドで値を指定します。
5. [ドキュメントパーツの作成] を選択します。リストにないオーソライザーのドキュメントを作成できます。
6. 必要に応じて、これらのステップを繰り返し、他のオーソライザーにドキュメントパーツを追加するか、編集します。

既存のドキュメントパーツを編集するには、以下を実行します。

1. [ドキュメント] ペインで [オーソライザー] タブを選択します。
2. 検索バーを使用するかオーソライザーを選択し、[編集] を選択します。

API Gateway コンソールを使用して API ドキュメントを公開する

次の手順では、ドキュメントバージョンを発行する方法について説明します。

API Gateway コンソールを使用してドキュメントバージョンを公開するには

1. メインナビゲーションペインで、[ドキュメント] を選択します。
2. [ドキュメントの発行] を選択します。
3. 出版物をセットアップします。

    a. [ステージ] では、ステージを選択します。
    b. [バージョン] には、バージョン識別子を入力します (例: `1.0.0`)。
    c. (オプション) [説明] に説明を入力します。
4. [Publish] を選択します。

これで、ドキュメントを OpenAPI ファイルにエクスポートすることにより、発行されたドキュメントのダウンロードに進むことができるようになります。詳細については、「the section called “REST API をエクスポートする”」を参照してください。

API Gateway REST API を使用して API を文書化する

このセクションでは、API Gateway REST API を使用して API のドキュメントパートを作成および維持する方法について説明します。

API のドキュメントを作成および編集する前に、まず API を作成します。このセクションでは、例として PetStore API を使用します。API Gateway コンソールを使用して API を作成するには、「チュートリアル: サンプルをインポートして REST API を作成する」の手順に従います。

トピック

- API エンティティのドキュメント化
- RESOURCE エンティティのドキュメント化
- METHOD エンティティのドキュメント化
- QUERY_PARAMETER エンティティのドキュメント化

- PATH_PARAMETER エンティティのドキュメント化
- REQUEST_BODY エンティティのドキュメント化
- REQUEST_HEADER エンティティのドキュメント化
- RESPONSE エンティティのドキュメント化
- RESPONSE_HEADER エンティティのドキュメント化
- AUTHORIZER エンティティのドキュメント化
- MODEL エンティティのドキュメント化
- ドキュメントパーツの更新
- ドキュメントパーツの一覧表示

API エンティティのドキュメント化

API のドキュメントを追加するには、API エンティティの DocumentationPart リソースを追加します。

```
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}"
}
```

成功した場合、オペレーションは新しく作成された 201 Created インスタンスをペイロードに含む、DocumentationPart レスポンスを返します。例:

```
{
  ...
  "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}"
}
```

ドキュメントパートが既に追加されている場合、エラーメッセージ 409 Conflict を含む Documentation part already exists for the specified location: type 'API'." レスポンスが返されます。この場合、documentationpart:update オペレーションを呼び出す必要があります。

```
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}"
  } ]
}
```

正常なレスポンスでは、更新された 200 OK インスタンスをペイロードに含む DocumentationPart ステータスコードが返されます。

RESOURCE エンティティのドキュメント化

API のルートリソースのドキュメントを追加するには、対応する Resource リソースをターゲットとした DocumentationPart リソースを追加します。

```
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}"
}
```

成功した場合、オペレーションは新しく作成された 201 Created インスタンスをペイロードに含む、DocumentationPart レスポンスを返します。例:

```
{
  "_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}"
}
```

リソースパスが指定されていない場合、リソースはルートリソースとみなされます。"path": "/" を properties に追加すると、明示的に指定できます。

API の子リソースのドキュメントを作成するには、対応する Resource リソースをターゲットとした DocumentationPart リソースを追加します。

```
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}"
}
```

成功した場合、オペレーションは新しく作成された 201 Created インスタンスをペイロードに含む、DocumentationPart レスポンスを返します。例:

```
{
  "_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}"
}
```

パスパラメータにより指定された子リソースのドキュメントを追加するには、Resource リソースをターゲットとした DocumentationPart リソースを追加します。

```
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}"
}
```

成功した場合、オペレーションは新しく作成された 201 Created インスタンスをペイロードに含む、DocumentationPart レスポンスを返します。例:

```
{
  "_links": {
    "curies": {
      "href": "http://docs.aws.amazon.com/apigateway/latest/developerguide/restapi-
documentationpart-{rel}.html",
      "name": "documentationpart",
```

```
      "templated": true
    },
    "self": {
      "href": "/restapis/4wk1k4onj3/documentation/parts/k6fpwb"
    },
    "documentationpart:delete": {
      "href": "/restapis/4wk1k4onj3/documentation/parts/k6fpwb"
    },
    "documentationpart:update": {
      "href": "/restapis/4wk1k4onj3/documentation/parts/k6fpwb"
    }
  },
  "id": "k6fpwb",
  "location": {
    "path": "/pets/{petId}",
    "method": null,
    "name": null,
    "statusCode": null,
    "type": "RESOURCE"
  },
  "properties": "{\n\t\"description\" : \"A child resource specified by the petId path
 parameter.\"\n}"
}
```

> **Note**
>
> RESOURCE エンティティの DocumentationPart インスタンスは、どの子リソースも継承することができません。

METHOD エンティティのドキュメント化

API のメソッドのドキュメントを追加するには、対応する Method リソースをターゲットとした DocumentationPart リソースを追加します。

```
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}"
}
```

成功した場合、オペレーションは新しく作成された 201 Created インスタンスをペイロードに含む、DocumentationPart レスポンスを返します。例:

```
{
  "_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}"
}
```

成功した場合、オペレーションは新しく作成された 201 Created インスタンスをペイロードに含む、DocumentationPart レスポンスを返します。例:

```
{
  "_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}"
}
```

前のリクエストで location.method フィールドが指定されていない場合、ワイルドカード文字 ANY で表される * であるとみなされます。

METHOD エンティティのドキュメントのコンテンツを更新するには、新しい properties マップを指定して documentationpart:update オペレーションを呼び出します。

```
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}"
  } ]
}
```

正常なレスポンスでは、更新された 200 OK インスタンスをペイロードに含む DocumentationPart ステータスコードが返されます。例:

```
{
  "_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}"
}
```

QUERY_PARAMETER エンティティのドキュメント化

リクエストクエリパラメータのドキュメントを追加するには、QUERY_PARAMETER タイプをターゲットとした、有効なフィールド path および name を含む DocumentationPart リソースを追加します。

```
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}"
}
```

成功した場合、オペレーションは新しく作成された 201 Created インスタンスをペイロードに含む、DocumentationPart レスポンスを返します。例:

```
{
  "_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 エンティティのドキュメントパーツの QUERY_PARAMETER マップは、いずれかの子 QUERY_PARAMETER エンティティが継承できます。たとえば、treats の後に /pets/{petId} リソースを追加して、GET で /pets/{petId}/treats メソッドを有効にし、page クエリパラメータを開示した場合、この子クエリパラメータは DocumentationPart メソッドの同様の名前のクエリパラメータから properties の GET /pets マップを継承します。ただし、DocumentationPart メソッドの page クエリパラメータに GET /pets/{petId}/treats リソースを明示的に追加した場合を除きます。

PATH_PARAMETER エンティティのドキュメント化

パスパラメータのドキュメントを追加するには、PATH_PARAMETER エンティティの DocumentationPart リソースを追加します。

```
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}"
```

```
}
```

成功した場合、オペレーションは新しく作成された 201 Created インスタンスをペイロードに含む、DocumentationPart レスポンスを返します。例:

```
{
  "_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}"
}
```

REQUEST_BODY エンティティのドキュメント化

リクエストボディのドキュメントを追加するには、リクエストボディの DocumentationPart リソースを追加します。

```
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}"
}
```

成功した場合、オペレーションは新しく作成された 201 Created インスタンスをペイロードに含む、DocumentationPart レスポンスを返します。例:

```
{
  "_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}"
}
```

REQUEST_HEADER エンティティのドキュメント化

リクエストヘッダーのドキュメントを追加するには、リクエストヘッダーの DocumentationPart リソースを追加します。

```
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}"
}
```

成功した場合、オペレーションは新しく作成された 201 Created インスタンスをペイロードに含む、DocumentationPart レスポンスを返します。例:

```
{
  "_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}"
}
```

RESPONSE エンティティのドキュメント化

ステータスコードのレスポンスのドキュメントを追加するには、対応する MethodResponse リソースをターゲットとする DocumentationPart リソースを追加します。

```
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}"
}
```

成功した場合、オペレーションは新しく作成された 201 Created インスタンスをペイロードに含む、DocumentationPart レスポンスを返します。例:

```
{
    "_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}"
}
```

RESPONSE_HEADER エンティティのドキュメント化

レスポンスヘッダーのドキュメントを追加するには、レスポンスヘッダーの DocumentationPart リソースを追加します。

```
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}"
```

成功した場合、オペレーションは新しく作成された 201 Created インスタンスをペイロードに含む、DocumentationPart レスポンスを返します。例:

```
{
  "_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}"
}
```

この Content-Type レスポンスヘッダーのドキュメントは、API の任意のレスポンスの Content-Type ヘッダーのデフォルトドキュメントです。

AUTHORIZER エンティティのドキュメント化

API オーソライザーのドキュメントを追加するには、指定されたオーソライザーをターゲットとする DocumentationPart リソースを追加します。

```
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}"
}
```

成功した場合、オペレーションは新しく作成された 201 Created インスタンスをペイロードに含む、DocumentationPart レスポンスを返します。例:

```
{
  "_links": {
    "curies": {
      "href": "http://docs.aws.amazon.com/apigateway/latest/developerguide/restapi-
documentationpart-{rel}.html",
      "name": "documentationpart",
      "templated": true
    },
    "self": {
      "href": "/restapis/4wk1k4onj3/documentation/parts/pw3qw3"
    },
    "documentationpart:delete": {
      "href": "/restapis/4wk1k4onj3/documentation/parts/pw3qw3"
    },
    "documentationpart:update": {
      "href": "/restapis/4wk1k4onj3/documentation/parts/pw3qw3"
    }
  },
  "id": "pw3qw3",
  "location": {
    "path": null,
    "method": null,
    "name": "myAuthorizer",
    "statusCode": null,
```

```
    "type": "AUTHORIZER"
  },
  "properties": "{\n\t\"description\" : \"Authorizes invocations of configured methods.
\"\n}"
}
```

> **Note**
>
> AUTHORIZER エンティティの DocumentationPart インスタンスは、どの子リソースも継承することができません。

MODEL エンティティのドキュメント化

MODEL エンティティをドキュメント化するには、モデルの DocumentPart インスタンスと、モデルの各 properties' を作成および管理する必要があります。たとえば、各 API に付属する Error モデルには、デフォルトでスキーマ定義

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

があり、2 つの DocumentationPart インスタンス (1 つは Model 用、もう 1 つはその message プロパティ用) が必要です。

```
{
  "location": {
    "type": "MODEL",
    "name": "Error"
  },
  "properties": {
    "title": "Error Schema",
    "description": "A description of the Error model"
  }
}
```

および

```
{
  "location": {
    "type": "MODEL",
    "name": "Error.message"
  },
  "properties": {
    "description": "An error message."
  }
}
```

API がエクスポートされると、DocumentationPart のプロパティにより元のスキーマの値が上書きされます。

API モデルのドキュメントを追加するには、指定されたモデルをターゲットとする DocumentationPart リソースを追加します。

```
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}"
}
```

成功した場合、オペレーションは新しく作成された 201 Created インスタンスをペイロードに含む、DocumentationPart レスポンスを返します。例:

```
{
  "_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}"
}
```

同じステップを繰り返して、いずれかのモデルのプロパティの DocumentationPart インスタンスを作成します。

> **Note**
>
> MODEL エンティティの DocumentationPart インスタンスは、どの子リソースも継承することができません。

ドキュメントパーツの更新

任意のタイプの API エンティティのドキュメントパートを更新するには、指定されたパート識別子の DocumentationPart インスタンスで PATCH リクエストを送信し、既存の `properties` マップを新しいマップに置き換えます。

```
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"
  } ]
}
```

正常なレスポンスでは、更新された 200 OK インスタンスをペイロードに含む DocumentationPart ステータスコードが返されます。

複数のドキュメントパーツを 1 回の PATCH リクエストで更新できます。

ドキュメントパーツの一覧表示

任意のタイプの API エンティティのドキュメントパートを一覧表示するには、DocumentationParts コレクションで GET リクエストを送信します。

```
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
```

正常なレスポンスでは、利用可能な 200 OK インスタンスをペイロードに含む DocumentationPart ステータスコードが返されます。

API Gateway REST API を使用した API ドキュメントを公開する

API のドキュメントを発行するには、ドキュメントスナップショットを作成、更新、または取得した後、ドキュメントスナップショットを API ステージに関連付けます。ドキュメントスナップショットを作成するとき、同時に API ステージに関連付けることもできます。

トピック

- ドキュメントスナップショットの作成と API ステージへの関連付け
- ドキュメントスナップショットの作成
- ドキュメントスナップショットの更新
- ドキュメントスナップショットの取得
- API ステージへのドキュメントスナップショットの関連付け
- ステージに関連付けられたドキュメントスナップショットのダウンロード

ドキュメントスナップショットの作成と API ステージへの関連付け

API のドキュメントパーツのスナップショットを作成し、同時に API ステージと関連付けるには、次の POST リクエストを送信します。

```
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"
}
```

成功した場合、オペレーションは新しく作成された 200 OK インスタンスをペイロードとして含む、DocumentationVersion レスポンスを返します。

または、最初は API ステージに関連付けずにドキュメントスナップショットを作成してからrestapi:update を呼び出し、指定した API ステージにスナップショットを関連付けることもできます。既存のドキュメントスナップショットの更新またはクエリを実行してから、そのステージの関連付けを更新することもできます。次の 4 つのセクションでは、そのステップを示します。

ドキュメントスナップショットの作成

API のドキュメントパートのスナップショットを作成するには、新しい DocumentationVersion リソースを作成し、API の DocumentationVersions コレクションに追加します。

```
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"
}
```

成功した場合、オペレーションは新しく作成された 200 OK インスタンスをペイロードとして含む、DocumentationVersion レスポンスを返します。

ドキュメントスナップショットの更新

ドキュメントスナップショットは、対応する DocumentationVersion リソースの description プロパティを変更することによってのみ更新できます。次の例は、そのバージョン識別子 *version* (1.0.0 など) によって識別されたとおりに、ドキュメントスナップショットの説明を更新する方法を示しています。

```
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."
    }]
}
```

成功した場合、オペレーションは更新された 200 OK インスタンスをペイロードとして含む、DocumentationVersion レスポンスを返します。

ドキュメントスナップショットの取得

ドキュメントスナップショットを取得するには、指定された DocumentationVersion リソースに対して GET リクエストを送信します。次の例は、特定のバージョン識別子 1.0.0 のドキュメントスナップショットを取得する方法を示しています。

```
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
```

API ステージへのドキュメントスナップショットの関連付け

API ドキュメントを発行するには、ドキュメントスナップショットを API ステージに関連付けます。ドキュメントバージョンをステージに関連付ける前に、API ステージをすでに作成している必要があります。

API Gateway REST API を使用してドキュメントスナップショットを API ステージに関連付けるには、stage:update オペレーションを呼び出して stage.documentationVersion プロパティに必要なドキュメントバージョンを設定します。

```
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"
    }]
}
```

ステージに関連付けられたドキュメントスナップショットのダウンロード

ドキュメントパートのバージョンがステージに関連付けられたら、API Gateway コンソール、API Gateway REST API、その SDK のいずれか、または API Gateway 用の AWS CLI を使用して、ドキュメントパートと API エンティティ定義を外部ファイルにエクスポートできます。プロセスは、API のエクスポートと同じです。エクスポートされるファイルの形式は、JSON または YAML です。

API Gateway REST API を使用すると、API ドキュメントパート、API 統合およびオーソライザーが API エクスポートに含められるように `extension=documentation,integrations,authorizers` クエリパラメータを明示的に設定することもできます。デフォルトでは、API のエクスポート時、ドキュメントパーツは含められますが、統合とオーソライザーは含められません。API エクスポートからのデフォルト出力は、ドキュメントの配付に適しています。

API Gateway REST API を使用して外部 JSON OpenAPI ファイルに API ドキュメントをエクスポートするには、次の GET リクエストを送信します。

```
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
```

ここでは、`x-amazon-apigateway-documentation` オブジェクトにドキュメントパーツが含まれており、API エンティティ定義に OpenAPI によりサポートされているドキュメントプロパティが含まれています。この出力には、統合や Lambda オーソライザー (以前のカスタムオーソライザー) の詳細は含まれません。両方の詳細を含めるには、`extensions=integrations,authorizers,documentation` を設定します。オーソライザーの詳細は含めず統合の詳細を含めるには、`extensions=integrations,documentation` を設定します。

JSON ファイルで結果を出力するには、リクエストで `Accept:application/json` ヘッダーを設定する必要があります。YAML 出力を生成するには、リクエストヘッダーを `Accept:application/yaml` に変更します。

例として、ルートリソース (GET) でシンプルな / メソッドを開示する API について調べます。この API には、OpenAPI 定義ファイルで定義された 4 つの API エンティティ (それぞれ API、MODEL、METHOD、および RESPONSE タイプ用) があります。ドキュメントパーツは、API、METHOD、RESPONSE の各エンティティに追加されています。前の documentation-exporting コマンドを呼び出すと、次の出力が生成され、標準 OpenAPI ファイルへの拡張として x-amazon-apigateway-documentation オブジェクト内にドキュメントパーツがリストされます。

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"
}
```

ドキュメントパートの `properties` マップで定義された OpenAPI に準拠する属性の場合、API Gateway は関連付けられた API エンティティ定義に属性を挿入します。`x-something` の属性は、標準 OpenAPI 拡張です。この拡張は、API エンティティ定義に伝達されます。たとえば、`x-example` メソッドの `GET` 属性を参照してください。`foo` などの属性は、OpenAPI 仕様の一部ではないため、関連付けられた API エンティティ定義には挿入されません。

ドキュメントレンダリングツール (OpenAPI UI など) が API エンティティ定義を解析してドキュメント属性を抽出する場合、`properties` インスタンスの OpenAPI に準拠していない `DocumentationPart` 属性はどれもツールで使用できません。一方、ドキュメントレンダリングツールが `x-amazon-apigateway-documentation` オブジェクトを解析してコンテンツを取得するか、ツールが restapi:documentation-parts および documenationpart:by-id を呼び出して API

Gateway からドキュメントパートを取得する場合、ツールでの表示にすべてのドキュメント属性を使用できます。

ドキュメントと、統合の詳細を含む API エンティティ定義を JSON OpenAPI ファイルにエクスポートするには、次の GET リクエストを送信します。

```
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
```

ドキュメントと、統合およびオーソライザーの詳細を含む API エンティティ定義を YAML OpenAPI ファイルにエクスポートするには、次の GET リクエストを送信します。

```
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
```

API Gateway コンソールを使用して、API の発行済みドキュメントをエクスポートおよびダウロードするには、「API Gateway コンソールを使用して REST API をエクスポートする」の手順に従います。

API ドキュメントのインポート

API エンティティ定義のインポートと同様、ドキュメントパートは API Gateway で外部 OpenAPI ファイルから API にインポートできます。インポートするドキュメントパートは、有効な OpenAPI 定義ファイルの x-amazon-apigateway-documentation オブジェクト 拡張内で指定します。ドキュメントをインポートしても、既存の API エンティティ定義は変更されません。

API Gateway で新しく指定されたドキュメントパートを既存のドキュメントパートにマージしたり、既存のドキュメントパートを上書きしたりするオプションがあります。MERGE モードでは、OpenAPI ファイルで定義された新しいドキュメントパーツが API の DocumentationParts コレクションに追加されます。インポートする DocumentationPart がすでに存在する場合、2 つの属性が異なる場合はインポートされた属性によって既存の属性が置き換えられます。他の既存のドキュメント属性は影響を受けません。OVERWRITE モードでは、インポートされた OpenAPI 定義ファイルに従って DocumentationParts コレクション全体が置き換えられます。

API Gateway REST API を使用してドキュメントパートをインポートする

API Gateway REST API を使用して API ドキュメントをインポートするには、documentationpart:import オペレーションを呼び出します。次の例は、API の既存のドキュメントパーツを単一の GET / メソッドに置き換えて、成功時は 200 OK レスポンスを返す方法を示しています。

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"
        }
      }
    ]
  }
}
```

成功すると、このリクエストはペイロードにインポートされた DocumentationPartId を含む 200 OK レスポンスを返します。

```
{
  "ids": [
    "kg3mth",
    "796rtf",
    "zhek4p",
    "5ukm9s"
  ]
}
```

加えて、API 定義の入力 OpenAPI ファイルの一部として `x-amazon-apigateway-documentation` オブジェクトでドキュメントパートを指定して、restapi:import または restapi:put を呼び出すこともできます。API インポートからドキュメントパーツを除外するには、リクエストクエリパラメータで `ignore=documentation` を設定します。

API Gateway コンソールを使用してドキュメントパートをインポートする

次の手順では、ドキュメントパーツをインポートする方法について説明します。

コンソールを使用して API のドキュメントパーツを外部ファイルからインポートするには

1. メインナビゲーションペインで、[ドキュメント] を選択します。
2. Import (インポート) を選択します。
3. 既存のドキュメントがある場合は、新しいドキュメントを [上書き] するか [マージ] するかを選択します。
4. [ファイルの選択] を選択してドライブからファイルをロードするか、ファイルのコンテンツをファイルビューに入力します。例として、「API Gateway REST API を使用してドキュメントパートをインポートする」にあるサンプルリクエストのペイロードを参照してください。
5. インポート時の警告の処理方法を選択します。[警告を失敗とみなす] または [警告を無視する] を選択します。詳細については、「the section called “インポート中のエラーと警告”」を参照してください。
6. [Import] を選択します。

API ドキュメントへのアクセスの制御

API ドキュメントの執筆と編集を行う専用ドキュメントチームを持っている場合、デベロッパー (API 開発用) と執筆者および編集者 (コンテンツ開発用) に別個のアクセス権限を設定することができます。これは特に、サードパーティベンダーがドキュメントの作成に関与している場合に適しています。

API ドキュメントを作成、更新、発行するアクセス許可をドキュメントチームに付与するには、次の IAM ポリシーを持つ IAM ロールをドキュメントチームに割り当てることができます。ここで、*account_id* はドキュメントチームの AWS アカウント ID です。

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

API Gateway リソースにアクセスするためのアクセス許可の設定については、「the section called "Amazon API Gateway と IAM の連携方法"」を参照してください。

API Gateway で REST API 用 SDK を生成する

プラットフォームおよび言語に固有な方法で REST API を呼び出すには、プラットフォームおよび言語に固有な SDK を API 用に生成する必要があります。API を作成およびテストし、ステージにデプロイした後で SDK を生成します。現在、API Gateway は、Java、JavaScript、Java for Android、iOS 用の Objective-C または Swift、Ruby において、API の SDK の生成をサポートしています。

このセクションでは、API Gateway API の SDK を生成する方法について説明します。また、生成された SDK を Java、Java for Android、iOS 用の Objective-C と Swift、JavaScript の各アプリで使用する方法についても説明します。

わかりやすく説明するために、この単純な電卓の Lambda 関数を公開する、この API Gateway API を使用します。

先に進む前に、API Gateway で少なくとも 1 回、API を作成またはインポートし、デプロイしてください。手順については、「Amazon API Gateway での REST API のデプロイ」を参照してください。

トピック

- 単純な電卓の Lambda 関数
- API Gateway の単純な電卓の API
- 単純な電卓の API OpenAPI 定義
- API の Java SDK の生成
- API の Android SDK の生成
- API の iOS SDK の生成
- REST API の JavaScript SDK の生成
- API の Ruby SDK の生成
- AWS CLI コマンドを使用して API 用の SDK を生成する

単純な電卓の Lambda 関数

例として加減乗除のバイナリ演算を行う Node.js Lambda 関数を使用します。

トピック

- 単純な電卓の Lambda 関数の入力形式
- 単純な電卓の Lambda 関数の出力形式
- 単純な電卓の Lambda 関数の実装

単純な電卓の Lambda 関数の入力形式

この関数の入力形式は次のとおりです。

```
{ "a": "Number", "b": "Number", "op": "string"}
```

op は (+, -, *, /, add, sub, mul, div) のいずれかです。

単純な電卓の Lambda 関数の出力形式

演算が成功すると、次の形式の結果が返されます。

```
{ "a": "Number", "b": "Number", "op": "string", "c": "Number"}
```

c には計算結果が入ります。

単純な電卓の Lambda 関数の実装

Lambda 関数の実装は次のとおりです。

```
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 の単純な電卓の API

Amazon の単純な電卓の API では、3 つのメソッド (GET、POST、GET) を公開して the section called “単純な電卓の Lambda 関数” を呼び出します。この API を図に示すと次のようになります。


Resources
Create resource
/
GET
POST
/{a}
ANY
/{b}
ANY
/{op}
GET

この 3 つのメソッドは方法は異なりますが、バックエンドの Lambda 関数に入力を渡して同じ操作を行います。

- `GET /?a=...&b=...&op=...` メソッドは、クエリパラメータを使用して入力を指定します。
- `POST /` メソッドは、`{"a":"Number", "b":"Number", "op":"string"}` の JSON ペイロードを使用して入力を指定します。
- `GET /{a}/{b}/{op}` メソッドは、パスパラメータを使用して入力を指定します。

API Gateway では、HTTP メソッドとパス部分を組み合わせて、対応する SDK メソッド名を生成します (定義されていない場合)。ルートパス部分 (/) は `Api Root` と呼ばれます。たとえば、`GET /?a=...&b=...&op=...` の API メソッドのデフォルトの Java SDK メソッド名は `getABOp`、`POST /` のデフォルトの SDK メソッド名は `postApiRoot`、 `GET /{a}/{b}/{op}` のデフォルトの SDK メソッド名は `getABOp` です。個々の SDK は規則をカスタマイズすることができます。SDK 固有のメソッド名については、生成された SDK ソースのドキュメントを参照してください。

各 API メソッドで operationName プロパティを指定すると、デフォルトの SDK メソッド名を上書きできます。API メソッドを作成する場合や、API Gateway REST API を使用して API メソッドを更新する場合は、これを行うことができます。API Swagger の定義では、同じ結果を得るために `operationId` を設定できます。

この API に対して API Gateway で生成された SDK を使用してこれらのメソッドを 呼び出す方法を示す前に、メソッドの設定方法を簡単に再確認しましょう。詳細な手順については、「」を参照してくださいAPI Gateway での REST API の開発 API Gateway を初めて使用する場合は、まず「AWS Lambda 統合を選択するチュートリアル」を参照してください。

入力と出力のモデルの作成

SDK で厳密に型指定された入力を指定するには、API の `Input` モデルを作成します。レスポンス本文のデータ型を記述するには、`Output` モデルと `Result` モデルを作成します。

入力、出力、結果のモデルを作成するには

1. ナビゲーションペインで、[モデル] を選択します。
2. [モデルの作成] を選択します。
3. [名前] に**input**と入力します。
4. [コンテンツタイプ] に、「**application/json**」と入力します。

一致するコンテンツタイプが見つからない場合、リクエストの検証は実行されません。コンテンツタイプに関係なく同じモデルを使用するには、「**$default**」と入力します。

5. [モデルのスキーマ] に次のモデルを入力します。

```
{
    "$schema" : "$schema": "http://json-schema.org/draft-04/schema#",
    "type":"object",
    "properties":{
        "a":{"type":"number"},
        "b":{"type":"number"},
        "op":{"type":"string"}
    },
    "title":"Input"
}
```

6. [モデルの作成] を選択します。

7. 次の手順を繰り返して、Output モデルと Result モデルを作成します。

   Output モデルについては、[モデルのスキーマ] に次のように入力します。

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "type": "object",
    "properties": {
        "c": {"type":"number"}
    },
    "title": "Output"
}
```

   Result モデルについては、[モデルのスキーマ] に次のように入力します。API ID abc123 を自分の API ID に置き換えます。

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "type":"object",
    "properties":{
        "input":{
            "$ref":"https://apigateway.amazonaws.com/restapis/abc123/models/Input"
        },
        "output":{
```

```
            "$ref":"https://apigateway.amazonaws.com/restapis/abc123/models/Output"
        }
    },
    "title":"Result"
}
```

GET / メソッドのクエリパラメータの設定

GET /?a=..&b=..&op=.. メソッドのクエリパラメータは [メソッドリクエスト] で宣言します。

GET / URL クエリ文字列パラメータを設定するには

1. [メソッドリクエスト] セクションのルート (/) リソースの GET メソッドで、[編集] を選択します。
2. [URL クエリ文字列パラメータ] を選択してから、次の操作を行います。

   a. [クエリ文字列の追加] を選択します。
   b. [名前] に**a**と入力します。
   c. [必須] と [キャッシュ] はオフのままにしておきます。
   d. [キャッシュ] はオフのままにします。

   同じ手順を繰り返して、**b** という名前のクエリ文字列と **op** という名前のクエリ文字列を作成します。
3. [Save] を選択します。

バックエンドへの入力としてペイロードのデータモデルを設定する

POST / メソッドについては、Input モデルを作成し、それをメソッドリクエストに追加して入力データの形式を定義します。

バックエンドへの入力としてペイロードのデータモデルを設定するには

1. [メソッドリクエスト] セクションのルート (/) リソースの POST メソッドで、[編集] を選択します。
2. [リクエスト本文] を選択します。
3. [モデルの追加] を選択します。
4. [コンテンツタイプ] に、「**application/json**」と入力します。

5. [モデル] で [入力] を選択します。
6. [保存] を選択します。

API のユーザーは、このモデルを使用して `Input` オブジェクトをインスタンス化することで、SDK を呼び出して入力を指定できます。このモデルがないと、Lambda 関数への JSON 入力を表すためにディクショナリオブジェクトの作成が必要になります。

バックエンドからの結果出力のデータモデルを設定する

3 つすべてのメソッドで、`Result` モデルを作成し、それをメソッドの `Method Response` に追加して、Lambda 関数から返される出力の形式を定義します。

バックエンドからの結果出力のデータモデルを設定するには

1. /{a}/{b}/{op} リソースを選択し、[GET] メソッドを選択します。
2. [メソッドレスポンス] タブの [レスポンス 200] で、[編集] を選択します。
3. [リクエスト本文] で、[モデルを追加] を選択します。
4. [コンテンツタイプ] に、「**`application/json`**」と入力します。
5. [モデル] で、[結果] を選択します。
6. [Save] を選択します。

API のユーザーは、このモデルを使って `Result` オブジェクトのプロパティを読み取ることで、正常な出力を解析できます。このモデルがないと、JSON 出力を表すためにディクショナリオブジェクトを作成することが必要になります。

単純な電卓の API OpenAPI 定義

次に示すのは、単純な電卓の API の OpenAPI 定義です。この定義をアカウント内にインポートできます。ただし、インポート後に Lambda 関数でリソースベースのアクセス許可をリセットする必要があります。そのためには、API Gateway コンソールの [Integration Request (統合リクエスト)] で、アカウントで作成した Lambda 関数を再選択します。これにより、必要なアクセス許可が API Gateway コンソールでリセットされます。別の方法として、add-permission の Lambda コマンドに AWS Command Line Interface を使用することもできます。

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"
          }
        }
      }
    }
  }
}
```

API の Java SDK の生成

API Gateway で API の Java SDK を生成するには

1. https://console.aws.amazon.com/apigateway で API Gateway コンソールにサインインします。
2. REST API を選択します。
3. [ステージ] を選択します。
4. [ステージ] ペインで、ステージの名前を選択します。
5. [ステージアクション] メニューを開き、[SDK を生成] を選択します。
6. [プラットフォーム] で Java プラットフォームを選択し、次の操作を行います。

   a. [サービス名] で、SDK の名前を指定します たとえば、**SimpleCalcSdk** と指定します。これは、SDK クライアントクラスの名前になります。この名前は、SDK のプロジェクトフォルダの pom.xml ファイルで、<name> の下にある <project> タグに対応します。ハイフンは含めないでください。

   b. [Java Package Name (Java パッケージ名)] で、SDK のパッケージ名を指定します たとえば、**examples.aws.apig.simpleCalc.sdk** と指定します。このパッケージ名は SDK ライブラリの名前空間として使用されます。ハイフンは含めないでください。

c. [Java Build System (Java ビルドシステム)] で「**maven**」または「**gradle**」と入力し、ビルドシステムを指定します。

d. [Java Group Id (Java グループ ID)] で、SDK プロジェクトのグループ ID を入力します たとえば、「**my-apig-api-examples**」と入力します。この ID は、SDK のプロジェクトフォルダーの `<groupId>` ファイルで、`<project>` の下にある `pom.xml` タグに対応します。

e. [Java Artifact Id (Java アーティファクト ID)] で、SDK プロジェクトのアーティファクト ID を入力します たとえば、「**simple-calc-sdk**」と入力します。この ID は、SDK のプロジェクトフォルダーの `<artifactId>` ファイルで、`<project>` の下にある `pom.xml` タグに対応します。

f. [Java Artifact Version (Java アーティファクトバージョン)] で、バージョン ID の文字列を入力します たとえば、**1.0.0** と指定します。このバージョン ID は、SDK のプロジェクトフォルダーの `<version>` ファイルで、`<project>` の下にある `pom.xml` タグに対応します。

g. [Source Code License Text (ソースコードライセンステキスト)] に、ソースコードのライセンステキスト (ある場合) を入力します。

7. [Generate SDK (SDK の生成)] を選択し、画面の指示に従って、API Gateway で生成された SDK をダウンロードします。

生成された SDK を使用するには、「REST API 用に API Gateway で生成された Java SDK を使用する」の手順に従います。

API を更新するたびに、更新を反映するために API を再デプロイして SDK を再生成する必要があります。

API の Android SDK の生成

API Gateway で API の Android SDK を生成するには

1. https://console.aws.amazon.com/apigateway で API Gateway コンソールにサインインします。
2. REST API を選択します。
3. [ステージ] を選択します。
4. [ステージ] ペインで、ステージの名前を選択します。
5. [ステージアクション] メニューを開き、[SDK を生成] を選択します。
6. [プラットフォーム] で Android プラットフォームを選択し、次の操作を行います。

a. [Group ID (グループ ID)] に、対応するプロジェクトの一意の ID を入力します。これは、`pom.xml` ファイルで使用されます (例: **`com.mycompany`**)。

b. [Invoker package (Invoker パッケージ)] に、生成されたクライアントクラスの名前空間を入力します (例: **`com.mycompany.clientsdk`**)。

c. [Artifact ID (アーティファクト ID)] に、コンパイルされた .jar ファイルの名前を、バージョンを除外して入力します。これは、`pom.xml` ファイルで使用されます (例: **`aws-apigateway-api-sdk`**)。

d. [Artifact version (アーティファクトバージョン)] に、生成されたクライアントのアーティファクトバージョンの番号を入力します。これは `pom.xml` ファイルで使用され、*`major.minor.patch`* パターンに従います (例: **`1.0.0`**)。

7. [Generate SDK (SDK の生成)] を選択し、画面の指示に従って、API Gateway で生成された SDK をダウンロードします。

生成された SDK を使用するには、「REST API 用に API Gateway で生成された Android SDK を使用する」の手順に従います。

API を更新するたびに、更新を反映するために API を再デプロイして SDK を再生成する必要があります。

API の iOS SDK の生成

API Gateway で API の iOS SDK を生成するには

1. https://console.aws.amazon.com/apigateway で API Gateway コンソールにサインインします。
2. REST API を選択します。
3. [ステージ] を選択します。
4. [ステージ] ペインで、ステージの名前を選択します。
5. [ステージアクション] メニューを開き、[SDK を生成] を選択します。
6. [プラットフォーム] で、iOS (Objective-C) または iOS (Swift) プラットフォームを選択し、次の操作を行います。

   - [プレフィックス] ボックスに一意のプレフィックスを入力します。

     プレフィックスの効果は次のとおりです。例えば、`input`、`output`、`result` をモデルとする SimpleCalc API の SDK にプレフィックスとして **`SIMPLE_CALC`** を割り当てる

場合、生成される SDK には、メソッドのリクエスト/レスポンスを含む API をカプセル化する `SIMPLE_CALCSimpleCalcClient` クラスが含まれます。さらに、生成される SDK には、リクエスト入力およびレスポンス出力の入力、出力、結果をそれぞれ表す `SIMPLE_CALCinput` クラス、`SIMPLE_CALCoutput` クラス、`SIMPLE_CALCresult` クラスが含まれます。詳細については、「Objective-C または Swift で REST API 用に API Gateway で生成された iOS SDK を使用する」を参照してください。

7. [Generate SDK (SDK の生成)] を選択し、画面の指示に従って、API Gateway で生成された SDK をダウンロードします。

生成された SDK を使用するには、「Objective-C または Swift で REST API 用に API Gateway で生成された iOS SDK を使用する」の手順に従います。

API を更新するたびに、更新を反映するために API を再デプロイして SDK を再生成する必要があります。

REST API の JavaScript SDK の生成

API Gateway で API の JavaScript SDK を生成するには

1. https://console.aws.amazon.com/apigateway で API Gateway コンソールにサインインします。
2. REST API を選択します。
3. [ステージ] を選択します。
4. [ステージ] ペインで、ステージの名前を選択します。
5. [ステージアクション] メニューを開き、[SDK を生成] を選択します。
6. [プラットフォーム] には、JavaScript プラットフォームを選択します。
7. [Generate SDK (SDK の生成)] を選択し、画面の指示に従って、API Gateway で生成された SDK をダウンロードします。

生成された SDK を使用するには、「REST API 用に API Gateway で生成された JavaScript SDK を使用する」の手順に従います。

API を更新するたびに、更新を反映するために API を再デプロイして SDK を再生成する必要があります。

API の Ruby SDK の生成

API Gateway で API の Ruby SDK を生成するには

1. https://console.aws.amazon.com/apigateway で API Gateway コンソールにサインインします。
2. REST API を選択します。
3. [ステージ] を選択します。
4. [ステージ] ペインで、ステージの名前を選択します。
5. [ステージアクション] メニューを開き、[SDK を生成] を選択します。
6. [プラットフォーム] で Ruby プラットフォームを選択し、次の操作を行います。

    a. [サービス名] で、SDK の名前を指定します (**`SimpleCalc`** など)。これは、API の Ruby Gem の名前空間を生成するために使用されます。名前はすべて文字である必要があり (`a-zA-Z`)、その他の特殊文字や数字は使用できません。

    b. [Ruby Gem Name (Ruby Gem 名)] で、Ruby Gem の名前を指定し、API 用に生成された SDK ソースコードを含めます。デフォルトでは、小文字のサービス名に `-sdk` サフィックスを加えたものです (**`simplecalc-sdk`** など)。

    c. [Ruby Gem Version (Ruby Gem バージョン)] で、生成された Ruby Gem のバージョン番号を指定します。デフォルトでは、`1.0.0` に設定されます。
7. [Generate SDK (SDK の生成)] を選択し、画面の指示に従って、API Gateway で生成された SDK をダウンロードします。

生成された SDK を使用するには、「API Gateway によって生成された Ruby SDK を REST API で使用する」の手順に従います。

API を更新するたびに、更新を反映するために API を再デプロイして SDK を再生成する必要があります。

AWS CLI コマンドを使用して API 用の SDK を生成する

AWS CLI を使用して、get-sdk コマンドを呼び出すことによって、サポートされているプラットフォーム用の API の SDK を生成およびダウンロードできます。以下では、サポートされているプラットフォームの一部について、これを示しています。

トピック

- AWS CLI を使用して Java for Android SDK を生成してダウンロードする

- AWS CLI を使用して JavaScript SDK を生成してダウンロードする
- AWS CLI を使用して Ruby SDK を生成してダウンロードする

AWS CLI を使用して Java for Android SDK を生成してダウンロードする

特定のステージ (udpuvvzbkc) で、API (test) の API Gateway で生成された Java for Android SDK を生成してダウンロードするには、次のようにコマンドを呼び出します。

```
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 の最後の入力は、myApi-android-sdk.zip という名前の、ダウンロードした SDK ファイルへのパスです。

AWS CLI を使用して JavaScript SDK を生成してダウンロードする

特定のステージ (udpuvvzbkc) で、API (test) の API Gateway で生成された JavaScript SDK を生成してダウンロードするには、次のようにコマンドを呼び出します。

```
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 の最後の入力は、myApi-js-sdk.zip という名前の、ダウンロードした SDK ファイルへのパスです。

AWS CLI を使用して Ruby SDK を生成してダウンロードする

特定のステージ (udpuvvzbkc) で、API (test) の Ruby SDK を生成してダウンロードするには、次のようにコマンドを呼び出します。

```
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` の最後の入力は、`myApi-ruby-sdk.zip` という名前の、ダウンロードした SDK ファイルへのパスです。

次に、生成された SDK を使用して基盤となる API を呼び出す方法を示します。詳細については、「Amazon API Gateway での REST API の呼び出し」を参照してください。

AWS Marketplace で API Gateway API を販売する

API を作成、テスト、デプロイしたら、それらを API Gateway 使用量プランにパッケージ化し、AWS Marketplace を通じてそのプランを SaaS (Software-as-a-Service) 製品として販売します。製品にサブスクライブする API の購入者には、使用量プランに対して行われたリクエストの数に基づいて AWS Marketplace により請求されます。

AWS Marketplace で API を販売するには、販売チャネルをセットアップして AWS Marketplace と API Gateway を統合する必要があります。一般に、これには AWS Marketplace への製品の掲載、API Gateway が使用状況メトリクスを AWS Marketplace に送信できるようにする適切なポリシーを持つ IAM ロールの設定、AWS Marketplace 製品と API Gateway 使用プランの関連付け、AWS Marketplace の購入者と API Gateway API キーの関連付けが必要です。詳細については、以降のセクションで説明します。

API を AWS Marketplace で SaaS 製品として販売する方法の詳細については、「AWS Marketplace ユーザーガイド」を参照してください。

トピック

- API Gateway を使用して AWS Marketplace 統合を初期化する
- 使用量プランへの顧客サブスクリプションの処理

API Gateway を使用して AWS Marketplace 統合を初期化する

以下のタスクは、API Gateway との AWS Marketplace の統合を初期化する 1 回限りのタスクです。これにより、API を SaaS 製品として販売できるようになります。

への製品の出品AWS Marketplace

使用量プランを SaaS 商品として出品するには、AWS Marketplace を通じて製品ロードフォームを送信します。商品には、apigateway タイプの requests というディメンションが含まれている必要があります。このディメンションは、リクエストあたりの料金を定義し、API へのリクエストを計測するために API Gateway により使用されます。

計測ロールの作成

次の実行ポリシーと信頼ポリシーを持つ ApiGatewayMarketplaceMeteringRole という名前の IAM ロールを作成します。このロールは、API Gateway が使用状況メトリクスを自動的に AWS Marketplace に送信できるようにします。

計測ロールの実行ポリシー

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "aws-marketplace:BatchMeterUsage",
        "aws-marketplace:ResolveCustomer"
      ],
      "Resource": "*",
      "Effect": "Allow"
    }
  ]
}
```

計測ロールの信頼関係ポリシー

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

```
}
```

使用量プランと AWS Marketplace 商品の関連付け

AWS Marketplace に商品を出品すると、AWS Marketplace 製品コードを受け取ります。API Gateway と AWS Marketplace を統合するには、使用量プランを AWS Marketplace 製品コードに関連付けます。関連付けは、API Gateway コンソール、API Gateway REST API、API Gateway 用の AWS CLI、または API Gateway 用の AWS SDKを使用して、API Gateway `UsagePlan` の `productCode` フィールドを AWS Marketplace 製品コードに設定することで有効化します。次のコード例では、API Gateway REST API が使用されています。

```
PATCH /usageplans/USAGE_PLAN_ID
Host: apigateway.region.amazonaws.com
Authorization: ...

{
    "patchOperations" : [{
        "path" : "/productCode",
        "value" : "MARKETPLACE_PRODUCT_CODE",
        "op" : "replace"
    }]
}
```

使用量プランへの顧客サブスクリプションの処理

以下のタスクは、デベロッパーポータルアプリケーションにより処理されます。

お客様が AWS Marketplace を通じて製品にサブスクライブすると、AWS Marketplace は、AWS Marketplace で製品を掲載したときに登録した SaaS サブスクリプション URL に `POST` リクエストを送信します。`POST` リクエストには、購入者情報を含む `x-amzn-marketplace-token` パラメータが付属しています。「SaaS の顧客のオンボーディング」の手順に従って、開発者ポータルアプリケーションでこのリダイレクトを処理します。

お客様のサブスクリプションリクエストに応答して、AWS Marketplace はサブスクライブ可能な Amazon SNS トピックに `subscribe-success` 通知を送信します。(「SaaS の顧客のオンボーディング」を参照してください)。お客様のサブスクリプションリクエストを承諾するには、お客様の API Gateway API キーを作成または取得し、AWS Marketplace によってプロビジョニングされたお客様の `customerId` を API キーに関連付けて、その API キーを使用料プランに関連付けることによって、`subscribe-success` 通知を処理します。

お客様のサブスクリプションリクエストが完了すると、デベロッパーポータルアプリケーションに、お客様および関連付けられた API キーが表示され、API へのリクエストの x-api-key ヘッダーに API キーを含める必要があることがお客様に通知されます。

顧客が使用量プランへのサブスクリプションをキャンセルすると、AWS Marketplace は unsubscribe-success 通知を SNS トピックに送信します。顧客をサブスクライブ解除するプロセスを完了するには、使用量プランから顧客の API キーを削除することで unsubscribe-success 通知を処理します。

顧客による使用量プランへのアクセスの許可

特定の顧客に使用量プランへのアクセスを許可するには、API Gateway API を使用して顧客の API キーをフェッチまたは作成し、API キーを使用量プランに追加します。

次の例は、API Gateway REST API を呼び出して、特定の AWS Marketplace customerId 値 (*MARKETPLACE_CUSTOMER_ID*) が設定された新しい API キーを作成する方法を説明するものです。

```
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"
}
```

次の例は、特定の AWS Marketplace customerId 値 (*MARKETPLACE_CUSTOMER_ID*) が設定された API キーを取得する方法を示しています。

```
GET apikeys?customerId=MARKETPLACE_CUSTOMER_ID HTTP/1.1
Host: apigateway.region.amazonaws.com
Authorization: ...
```

API キーを使用量プランに追加するには、関連する使用量プランの API キーを持つ UsagePlanKey を作成します。次の例では、API Gateway REST API を使用してこれを実現する方法を示していま

す。ここで、n371pt は使用量プラン ID、q5ugs7qjjh は前の例から返されるサンプル API keyId です。

```
POST /usageplans/n371pt/keys HTTP/1.1
Host: apigateway.region.amazonaws.com
Authorization: ...

{
    "keyId": "q5ugs7qjjh",
    "keyType": "API_KEY"
}
```

顧客と API キーの関連付け

ApiKey の customerId フィールドをお客様の AWS Marketplace お客様 ID に更新する必要があります。これにより、API キーと AWS Marketplace 顧客が関連付けられ、購入者の計測と請求が可能になります。次のコード例では、それを行うために API Gateway REST API を呼び出します。

```
PATCH /apikeys/q5ugs7qjjh
Host: apigateway.region.amazonaws.com
Authorization: ...

{
    "patchOperations" : [{
        "path" : "/customerId",
        "value" : "MARKETPLACE_CUSTOMER_ID",
        "op" : "replace"
    }]
}
```

REST API の保護

API Gateway は、悪意のあるユーザーやトラフィックの急増など、特定の脅威から API を保護するさまざまな方法を提供します。SSL 証明書の生成、ウェブアプリケーションファイアウォールの設定、スロットリング目標の設定、Virtual Private Cloud (VPC) から API へのアクセスのみの許可などの戦略を使用することで、API を保護できます。このセクションでは、API Gateway を使用してこれらの機能を有効にする方法を説明しています。

トピック

- REST API の相互 TLS 認証の設定

- バックエンド認証用 SSL 証明書の生成と設定
- AWS WAF を使用して API を保護する
- API リクエストを調整してスループットを向上させる
- Amazon API Gateway のプライベート REST API

REST API の相互 TLS 認証の設定

相互 TLS 認証には、クライアントとサーバー間の双方向認証が必要です。相互 TLS では、クライアントは X.509 証明書を提示して、API にアクセスするためのアイデンティティを検証する必要があります。相互 TLS は、モノのインターネット (IoT) および B2B アプリケーションの一般的な要件です。

相互 TLS は、API Gateway でサポートされる他の認証オペレーションおよび認可オペレーションとともに使用できます。API Gateway は、クライアントが提供する証明書を Lambda オーソライザーおよびバックエンド統合に転送します。

> **Important**
>
> デフォルトでは、クライアントは、API Gateway が API 用に生成する `execute-api` エンドポイントを使用して API を呼び出すことができます。相互 TLS でカスタムドメイン名を使用することによってのみクライアントが API にアクセスできるようにするには、デフォルトの `execute-api` エンドポイントを無効にします。詳細については、「the section called “デフォルトのエンドポイントを無効にする”」を参照してください。

トピック

- 相互 TLS の前提条件
- カスタムドメイン名の相互 TLS の設定
- 相互 TLS を必要とするカスタムドメイン名を使用して API を呼び出す
- 信頼ストアの更新
- 相互 TLS を無効にする
- 証明書の警告のトラブルシューティング
- ドメイン名の競合のトラブルシューティング
- ドメイン名のステータスメッセージのトラブルシューティング

相互 TLS の前提条件

相互 TLS を設定するには、以下が必要です。

- カスタムドメイン名
- カスタムドメイン名用の AWS Certificate Manager で構成されている少なくとも 1 つの証明書
- 設定され Amazon S3 にアップロードされた信頼ストア

カスタムドメイン名

REST API の相互 TLS を有効にするには、API のカスタムドメイン名を設定する必要があります。カスタムドメイン名の相互 TLS を有効にした後、カスタムドメイン名をクライアントに提供できます。相互 TLS が有効なカスタムドメイン名を使用して API にアクセスするには、クライアントは API リクエストで信頼できる証明書を提示する必要があります。詳細な情報は、「the section called “カスタムドメイン名”」にあります。

AWS Certificate Manager が発行した証明書を使用する

パブリックに信頼できる証明書は ACM から直接要求すること、またはパブリック証明書または自己署名証明書をインポートすることができます。ACM で証明書を設定するには、「ACM」を参照してください。証明書をインポートする場合は、次のセクションを参照してください。

インポートされた証明書、または AWS Private Certificate Authority 証明書を使用する

ACM にインポートされた証明書、または相互 TLS を用いた AWS Private Certificate Authority からの証明書を使用するには、API Gateway には ACM が発行した `ownershipVerificationCertificate` が必要です。この所有権証明書は、ドメイン名を使用する許可を持っていることを確認するためにのみ使用されます。TLS ハンドシェイクには使用されません。まだ `ownershipVerificationCertificate` を持っていない場合は、https://console.aws.amazon.com/acm/ に進み、その 1 つをセットアップします。

ドメイン名の有効期間中、この証明書を有効にしておく必要があります。証明書の有効期限が切れ、自動更新が失敗した場合、ドメイン名の更新はすべてロックされます。他の変更を加える前に、有効な `ownershipVerificationCertificate` を用いて `ownershipVerificationCertificateArn` を更新する必要があります。`ownershipVerificationCertificate` は、API Gateway の別の相互 TLS ドメインのサーバー証明書として使用できません。証明書を ACM に直接再インポートする場合、発行者は以前と同じである必要があります。

信頼ストアの設定

信頼ストアは、.pem ファイル拡張子が付いたテキストファイルです。これらは、証明機関からの証明書の信頼できるリストです。相互 TLS を使用するには、API にアクセスするために信頼する X.509 証明書の信頼ストアを作成します。

信頼ストアには、発行元の CA 証明書からルート CA 証明書までの完全な信頼チェーンを含める必要があります。API Gateway は、信頼チェーンに存在する任意の CA によって発行されたクライアント証明書を受け入れます。パブリックまたはプライベート認証機関からの証明書を使用できます。証明書チェーンの最大長は 4 です。自己署名証明書を提供することもできます。トラストストアでは、次のアルゴリズムがサポートされています。

- SHA-256 以上
- RSA-2048 以上
- ECDSA-256 または ECDSA-384

API Gateway はいくつかの証明書プロパティを検証します。Lambda オーソライザーを使用して、証明書が失効しているかどうかのチェックなど、クライアントによる API の呼び出し時に追加のチェックを実行できます。API Gateway は、次のプロパティを検証します。

検証	説明
X.509 構文	証明書は X.509 構文の要件を満たしている必要があります。
整合性	証明書の内容は、信頼ストアの認証機関によって署名された内容から変更されていないことが必要です。
Validity	証明書の有効期間は最新のものであることが必要です。
名前のチェーン/キーのチェーン	証明書の名前とサブジェクトは、途切れのないチェーンを形成する必要があります。証明書チェーンの最大長は 4 です。

信頼ストアを 1 つのファイルで Amazon S3 バケットにアップロードします。

以下に示しているのは、.pem ファイルの具体的な例です。

Example certificates.pem

```
-----BEGIN CERTIFICATE-----
<Certificate contents>
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
<Certificate contents>
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
<Certificate contents>
-----END CERTIFICATE-----
...
```

次の AWS CLI コマンドは、`certificates.pem` を Amazon S3 バケットにアップロードします。

```
aws s3 cp certificates.pem s3://bucket-name
```

API Gateway にトラストストアへのアクセスを許可するには、Amazon S3 バケットに API Gateway に対する読み取りアクセス許可が必要です。

カスタムドメイン名の相互 TLS の設定

REST API の相互 TLS を設定するには、API のリージョン別カスタムドメイン名を TLS_1_2 セキュリティポリシーで使用する必要があります。セキュリティポリシーの選択の詳細については、「the section called “セキュリティポリシーの選択”」を参照してください。

> **Note**
>
> 相互 TLS は、プライベート API ではサポートされていません。

信頼ストアを Amazon S3 にアップロードした後、相互 TLS を使用するようにカスタムドメイン名を設定できます。次のもの (スラッシュを含む) を端末に貼り付けます。

```
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
```

ドメイン名を作成したら、API オペレーション用に DNS レコードと基本パスのマッピングを設定する必要があります。詳細については、「API Gateway でのリージョン別カスタムドメイン名の設定」を参照してください。

相互 TLS を必要とするカスタムドメイン名を使用して API を呼び出す

相互 TLS が有効な API を呼び出すには、クライアントは API リクエストで信頼できる証明書を提示する必要があります。クライアントが API を呼び出そうとすると、API Gateway は信頼ストアでクライアント証明書の発行者を探します。API Gateway でリクエストを続行するには、証明書の発行者と、ルート CA 証明書までの完全な信頼チェーンが信頼ストアにある必要があります。

以下の例の `curl` コマンドは、`api.example.com`, が含まれるリクエストを `my-cert.pem` に送信します。`my-key.key` は証明書のプライベートキーです。

```
curl -v --key ./my-key.key --cert ./my-cert.pem api.example.com
```

API は、信頼ストアが証明書を信頼している場合にのみ呼び出されます。次の条件により、API Gateway が TLS ハンドシェイクに失敗し、`403` ステータスコードのリクエストが拒否されます。証明書が以下の場合:

- 信頼されていない
- 有効期限切れである
- サポートされているアルゴリズムを使用していない

> **Note**
>
> API Gateway は、証明書が失効したかどうかを検証しません。

信頼ストアの更新

信頼ストアの証明書を更新するには、新しい証明書バンドルを Amazon S3 にアップロードします。次に、カスタムドメイン名を更新して、更新された証明書を使用することができます。

Amazon S3 バージョニングを使用して、信頼ストアの複数のバージョンを維持します。新しい信頼ストアバージョンを使用するようにカスタムドメイン名を更新すると、証明書が無効な場合に API Gateway から警告が返されます。

API Gateway は、ドメイン名を更新するときにのみ、証明書の警告を生成します。API Gateway は、以前にアップロードされた証明書の有効期限が切れた場合でも、通知しません。

以下の AWS CLI コマンドは、新しい信頼ストアバージョンを使用するようにカスタムドメイン名を更新します。

```
aws apigateway update-domain-name \
    --domain-name api.example.com \
    --patch-operations op='replace',path='/mutualTlsAuthentication/
truststoreVersion',value='abcdef123'
```

相互 TLS を無効にする

カスタムドメイン名の相互 TLS を無効にするには、以下のコマンドに示すように、カスタムドメイン名から信頼ストアを削除します。

```
aws apigateway update-domain-name \
    --domain-name api.example.com \
    --patch-operations op='replace',path='/mutualTlsAuthentication/
truststoreUri',value=''
```

証明書の警告のトラブルシューティング

相互 TLS でカスタムドメイン名を作成する場合、信頼ストア内の証明書が有効でない場合に API Gateway から警告が返されます。これは、新しい信頼ストアを使用するようにカスタムドメイン名を更新するときにも発生します。警告は、証明書の問題と、警告の発生元となった証明書のサブジェクトを示します。相互 TLS は引き続き API で有効ですが、一部のクライアントは API にアクセスできない場合があります。

警告の発生元となった証明書を特定するには、信頼ストア内の証明書をデコードする必要があります。openssl などのツールを使用して、証明書をデコードし、そのサブジェクトを特定できます。

以下のコマンドは、証明書の内容 (サブジェクトなど) を表示します。

```
openssl x509 -in certificate.crt -text -noout
```

警告の発生元となった証明書を更新または削除してから、新しい信頼ストアを Amazon S3 にアップロードします。新しい信頼ストアをアップロードした後、その信頼ストアを使用するようにカスタムドメイン名を更新します。

ドメイン名の競合のトラブルシューティング

エラー `"The certificate subject <certSubject> conflicts with an existing certificate from a different issuer."` は、複数の認証機関がこのドメインの証明書を発行したことを表します。証明書の件名ごとに、相互 TLS ドメイン用の API Gateway には 1 人の発行者しか存在できません。1 つの発行者を通じて、その件名に関するすべての証明書を取得する必要があります。管理できない証明書に問題があるけれども、ドメイン名の所有権を証明できる場合は、連絡先 AWS Support からチケットを開きます。

ドメイン名のステータスメッセージのトラブルシューティング

`PENDING_CERTIFICATE_REIMPORT`: これは、証明書を ACM に再インポートし、検証に失敗したことを意味します。すなわち、新しい証明書には SAN (サブジェクトの別名) があり、`ownershipVerificationCertificate` または証明書のサブジェクトまたは SAN はドメイン名をカバーしないからです。何かが正しく設定されていないか、無効な証明書がインポートされた可能性があります。有効な証明書を ACM に再インポートする必要があります。検証の詳細については、「ドメインの所有権の検証」を参照してください。

`PENDING_OWNERSHIP_VERIFICATION`: 以前に検証した証明書の有効期限が切れ、ACM がそれを自動更新できなかったことを意味します。証明書を更新するか、新しい証明書をリクエストする必要があります。証明書の更新の詳細については、「ACM の管理された証明書の更新に関するトラブルシューティングガイド」を参照してください。

バックエンド認証用 SSL 証明書の生成と設定

API Gateway を使用して SSL 証明書を生成し、バックエンドでそのパブリックキーを使用して、バックエンドシステムへの HTTP リクエストが API Gateway からのものであることを確認できます。これにより、HTTP バックエンドは、バックエンドがパブリックにアクセス可能であっても、Amazon API Gateway から送信されるリクエストのみを制御し、受け入れることができます。

> **Note**
>
> 一部のバックエンドサーバーは、API Gateway のようには SSL クライアント認証をサポートしていない場合があり、SSL 証明書エラーを返す可能性があります。互換性のないバック

エンドサーバーのリストについては、「the section called “重要な注意点”」を参照してください。

API Gateway によって生成される SSL 証明書は自己署名されており、証明書のパブリックキーのみ API Gateway コンソールで、または API を通じて表示されます。

トピック

- API Gateway コンソールを使用してクライアント証明書を生成する
- SSL 証明書を使用するように API を設定する
- クライアント証明書の設定を確認するテスト呼び出し
- クライアント証明書を検証するようにバックエンド HTTPS サーバーを設定する
- 失効するクライアント証明書の更新
- HTTP および HTTP プロキシ統合のために API Gateway によりサポートされる証明機関

API Gateway コンソールを使用してクライアント証明書を生成する

1. API Gateway (https://console.aws.amazon.com/apigateway) コンソールを開きます。
2. REST API を選択します。
3. メインナビゲーションペインで、[クライアント証明書] を選択します。
4. [クライアント証明書] ペインから、[証明書の生成] を選択します。
5. (オプション) [説明] に説明を入力します。
6. [証明書を生成] を選択して証明書を生成します。API Gateway は新しい証明書を生成し、PEM エンコードされた公開鍵とともに新しい証明書 GUID を返します。

これで API で証明書を使用するように設定する準備が整いました。

SSL 証明書を使用するように API を設定する

これらの手順は、「API Gateway コンソールを使用してクライアント証明書を生成する」をすでに完了していることを前提としています。

1. API Gateway コンソールで、クライアント証明書を使用する API を作成するか開きます。API がステージにデプロイされていることを確認します。
2. メインナビゲーションペインで、[ステージ] を選択します。

3. [ステージの詳細] セクションで、[編集] を選択します。
4. [クライアント証明書] で、証明書を選択します。
5. [Save changes] (変更の保存) をクリックします。

   API が API Gateway コンソールで以前にデプロイ済みの場合、変更を有効にするには再デプロイが必要となります。詳細については、「the section called “REST API をステージに再デプロイする”」を参照してください。

API に対して証明書が選択され保存されると、API Gateway は API での HTTP 統合へのすべての呼び出しにこの証明書を使用します。

クライアント証明書の設定を確認するテスト呼び出し

1. API メソッドを選択します。[テスト] タブを選択します。[テスト] タブを表示するには、右矢印ボタンを選択する必要がある場合があります。
2. [クライアント証明書] で、証明書を選択します。
3. [テスト] を選択します。

API Gateway は、API を認証するために、選択された SSL 証明書を HTTP バックエンドに提示します。

クライアント証明書を検証するようにバックエンド HTTPS サーバーを設定する

以下の手順は、「API Gateway コンソールを使用してクライアント証明書を生成する」を完了し、クライアント証明書のコピーをダウンロード済みであることが前提です。クライアント証明書は、API Gateway REST API の `clientcertificate:by-id` または AWS CLI の `get-client-certificate` を呼び出すことでダウンロードできます。

API Gateway のクライアント SSL 証明書を確認するバックエンド HTTPS サーバーを設定する前に、PEM エンコードされたプライベートキーと、信頼された証明機関によって提供されたサーバー側証明書を取得しておく必要があります。

サーバーのドメイン名が `myserver.mydomain.com` の場合、サーバー証明書の CNAME 値は `myserver.mydomain.com` または `*.mydomain.com` にする必要があります。

サポートされている証明機関は、Let's Encrypt や「the section called “HTTP および HTTP プロキシ統合用にサポートされている証明機関”」のいずれかなどです。

たとえば、クライアント証明書ファイルが apig-cert.pem で、サーバーのプライベートキーと証明書ファイルはそれぞれ、server-key.pem と server-cert.pem であるとします。バックエンドの Node.js サーバーの場合、次のようなサーバーを設定できます。

```
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);
```

node-express アプリの場合、client-certificate-auth モジュールを使用して、PEM エンコードされた証明書を使用するクライアントのリクエストを認証できます。

その他の HTTPS サーバーについては、サーバーのドキュメントを参照してください。

失効するクライアント証明書の更新

API Gateway によって生成されたクライアント証明書は 365 日間有効です。API のダウンタイムを避けるため、API ステージのクライアント証明書が失効する前に、証明書を更新する必要があります。証明書の有効期限は、API Gateway REST API の clientCertificate:by-id または get-client-certificate の AWS CLI コマンドを呼び出して、返された expirationDate プロパティを調べることで確認できます。

クライアント証明書をローテーションするには、次の手順を実行します。

1. API Gateway REST API の clientcertificate:generate または generate-client-certificate の AWS CLI コマンドを呼び出して、新しいクライアント証明書を生成します。このチュートリアルでは、新しいクライアント証明書 ID を ndiqef とします。
2. バックエンドサーバーを更新して新しいクライアント証明書を含めます。既存のクライアント証明書はまだ削除しないでください。

更新を完了するためにサーバーの再起動が必要になる場合があります。更新中にサーバーを再起動する必要があるかどうかは、サーバーのドキュメントを参照してください。

3. API Gateway REST API の stage:update を呼び出し、新しいクライアント証明書 ID (ndiqef) 使用して新しいクライアント証明書を使用するように API ステージを更新します。

```
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"
    }
  ]
}
```

または、update-stage の CLI コマンドを呼び出します。

4. バックエンドサーバーを更新して古い証明書を削除します。
5. 古い証明書の clientCertificateId (a1b2c3) を指定して API Gateway REST API の clientcertificate:delete を呼び出し、古い証明書を API Gateway から削除します。

```
DELETE /clientcertificates/a1b2c3
```

または、delete-client-certificate の CLI コマンドを呼び出します。

```
aws apigateway delete-client-certificate --client-certificate-id a1b2c3
```

以前にデプロイされた API のコンソールでクライアント証明書をローテーションするには、以下のようにします。

1. メインナビゲーションペインで、[クライアント証明書] を選択します。
2. [クライアント証明書] ペインから、[証明書の生成] を選択します。

3. クライアント証明書を使用する API を開きます。
4. 選択された API で [ステージ] を選択し、ステージを選択します。
5. [ステージの詳細] セクションで、[編集] を選択します。
6. [クライアント証明書] で、証明書を選択します。
7. 設定を保存するには、[変更の保存] を選択します。

   変更を有効にするには、API を再デプロイする必要があります。詳細については、「the section called “REST API をステージに再デプロイする”」を参照してください。

HTTP および HTTP プロキシ統合のために API Gateway によりサポートされる証明機関

次のリストは、HTTP、HTTP プロキシ、およびプライベート統合のために API Gateway によりサポートされる証明機関を示しています。

```
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
```

AWS WAF を使用して API を保護する

AWS WAF は、ウェブアプリケーションと API を攻撃から保護するウェブアプリケーションファイアウォールです。お客様が定義するカスタマイズ可能なウェブセキュリティルールと条件に基づいて、ウェブリクエストを許可、ブロック、またはカウントする一連のルール (ウェブアクセスコントロールリスト (ウェブ ACL) と呼ばれます) を設定することができます。詳細については、「AWS WAF の仕組み」を参照してください。

AWS WAF を使用して、SQL インジェクションやクロスサイトスクリプティング (XSS) 攻撃などの一般的なウェブの脆弱性から API Gateway REST API を保護できます。これらは、API の可用性とパフォーマンスに影響を与え、セキュリティを侵害したり、過剰なリソースを消費したりする可能性があります。例えば、指定した IP アドレス範囲からのリクエスト、CIDR ブロックからのリクエスト、特定の国またはリージョンからのリクエスト、悪意のある SQL コードを含むリクエスト、悪意のあるスクリプトを含むリクエストを許可またはブロックするルールを作成できます。

また、HTTP ヘッダー、メソッド、クエリ文字列、URI、リクエスト本文内の指定文字列や正規表現パターン (最初の 64 KB までに制限) と一致するルールを作成することもできます。さらに、特定のユーザーエージェント、悪質なボット、またはコンテンツスクレーパーからの攻撃をブロックするルールを作成できます。たとえば、レートベースのルールを使用して、継続的に更新される後続の 5 分間で、各クライアント IP によって許可されるウェブリクエストの数を指定できます。

> **Important**
>
> AWS WAF は、ウェブの脆弱性に対する防御の最前線です。API で AWS WAF が有効になっている場合は、リソースポリシー、IAM ポリシー、Lambda オーソライザー、および Amazon Cognito オーソライザーなど他のアクセスコントロール機能の前に AWS WAF ルールが評価されます。たとえば、AWS WAF がリソースポリシーで許可されている CIDR ブロックからのアクセスをブロックした場合、AWS WAF が優先され、リソースポリシーは評価されません。

API に対して AWS WAF を有効にするには、次の操作を行う必要があります。

1. AWS WAF コンソール、AWS SDK、または CLI を使用して、AWS WAF マネージドルールと独自のカスタムルールを自由に組み合わせたウェブ ACL を作成します。詳細については、「AWS WAF の開始方法」と「ウェブアクセスコントロールリスト (ウェブ ACL)」を参照してください。

   > **Important**
   >
   > API Gateway には、リージョナルアプリケーション用の AWS WAFV2 ウェブ ACL または AWS WAF Classic Regional ウェブ ACL が必要です。

2. AWS WAF ウェブ ACL を API ステージに関連付けます。これを行うには、AWS WAF コンソール、AWS SDK、CLI、または API Gateway コンソールを使用できます。

API Gateway コンソールを使用して AWS WAF ウェブ ACL を API Gateway API ステージに関連付けるには

API Gateway コンソールを使用して AWS WAF ウェブ ACL を既存の API Gateway API ステージに関連付けるには、次の手順を使用します。

1. https://console.aws.amazon.com/apigateway で API Gateway コンソールにサインインします。

2. 既存の API を選択するか、新しい API を作成します。
3. メインナビゲーションペインで、[ステージ] を選択してから、ステージを選択します。
4. [ステージの詳細] セクションで、[編集] を選択します。
5. [ウェブアプリケーションファイアウォール (AWS WAF)] で、ウェブ ACL を選択します。

   AWS WAFV2 を使用している場合は、リージョナルアプリケーション用の AWS WAFV2 ウェブ ACL を選択します。使用するウェブ ACL やその他の AWS WAFV2 リソースは、API と同じリージョンに存在している必要があります。

   AWS WAF Classic Regional を使用している場合は、リージョナルウェブ ACL を選択します。
6. [Save changes] (変更の保存) をクリックします。

AWS CLI を使用して AWS WAF ウェブ ACL を API Gateway API ステージに関連付ける

AWS CLI を使用してリージョナルアプリケーション用の AWS WAFV2 ウェブ ACL を既存の API Gateway API ステージに関連付けるには、次の例に示すように、associate-web-acl コマンドを呼び出します。

```
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
```

AWS CLI を使用して AWS WAF Classic Regional ウェブ ACL を既存の API Gateway API ステージに関連付けるには、次の例に示すように associate-web-acl コマンドを呼び出します。

```
aws waf-regional associate-web-acl \
--web-acl-id 'aabc123a-fb4f-4fc6-becb-2b00831cadcf' \
--resource-arn 'arn:aws:apigateway:{region}::/restapis/4wk1k4onj3/stages/prod'
```

AWS WAF REST API を使用して AWS WAF ウェブ ACL を API ステージに関連付ける

AWS WAFV2 REST API を使用してリージョナルアプリケーション用の AWS WAFV2 ウェブ ACL を既存の API Gateway API ステージに関連付けるには、次の例に示すように、AssociateWebACL コマンドを呼び出します。

```
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'
)
```

AWS WAF REST API を使用して AWS WAF Classic Regional ウェブ ACL を既存の API Gateway API ステージに関連付けるには、次の例に示すように、AssociateWebACL コマンドを呼び出します。

```
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'
)
```

API リクエストを調整してスループットを向上させる

API のスロットリングおよびクォータを設定して、多すぎるリクエストで API の負荷が高くなりすぎないように保護できます。スロットルとクォータの両方はベストエフォートベースで適用されるため、これらは保証されたリクエスト上限ではなく、目標として考える必要があります。

API Gateway は、トークンバケットアルゴリズムを使用してトークンでリクエストをカウントし、API へのリクエストを調整します。特に API Gateway では、アカウントのすべての API に送信されるリクエストのレートとバーストをリージョンごとに検証します。トークンバケットアルゴリズムでは、これらの制限の事前定義されたオーバーランがバーストによって許可されますが、場合によっては、他の要因によって制限のオーバーランが発生することがあります。

リクエストの送信数がリクエストの定常レートおよびバーストを超えると、API Gateway はリクエストを調整を開始します。クライアントは、この時点で 429 Too Many Requests エラーレスポンスを受け取ることがあります。このような例外をキャッチすると、クライアントは 失敗したリクエストをレート制限する方法で再送信できます。

API デベロッパーは、API の個々のステージまたはメソッドに目標制限を設定して、アカウントのすべての API にわたるパフォーマンス全体を向上させることができます。別の方法として、使用量プランを有効にして、指定したリクエストレートおよびクォータに基づいて、クライアントのリクエスト送信数でスロットリングを設定することもできます。

トピック

- API Gateway でスロットリング制限設定が適用される方法
- リージョンごとのアカウントレベルのスロットリング
- 使用量プランの API レベルとステージレベルのスロットリング目標の設定
- ステージレベルのスロットリングターゲットの設定
- 使用量プランのメソッドレベルのスロットリング目標の設定

API Gateway でスロットリング制限設定が適用される方法

API のスロットリングおよびクォータ設定を構成する前に、Amazon API Gateway によってこれらの設定がどのように適用されるかを理解しておくことをお勧めします。

Amazon API Gateway のスロットリングに関連する設定には、4 つの基本的なタイプがあります。

- AWS スロットリングの制限は、リージョン内のすべてのアカウントとクライアントに適用されます。これらの制限設定は、リクエストが多すぎるために API —およびアカウント—が処理しきれなくなることを防ぎます。これらの制限は AWS によって設定され、お客様が変更することはできません。
- アカウントごとの制限は、指定したリージョンのアカウント内のすべての API に適用されます。このアカウントレベルのレート制限は、申請に応じて引き上げることができます。API のタイムアウトが短く、ペイロードが小さい場合、高い制限を設定できます。リージョンごとのアカウントレベルのスロットリング制限の引き上げを申請するには、AWS サポートセンターにお問い合わせください。詳細については、「クォータと重要な注意点」を参照してください。これらの制限は、AWS スロットリングの制限以上に高くすることはできません。
- API ごと、ステージごとのスロットリング制限は、ステージの API メソッドレベルで適用されます。すべてのメソッドに同じ設定を構成することも、メソッドごとに異なるスロットル設定を構成することもできます。これらの制限は、AWS スロットリングの制限以上に高くすることはできません。
- クライアントあたりのスロットリング制限は、クライアント ID として使用プランに関連付けられた API キーを使用するクライアントに適用されます。これらの制限は、 アカウントごとの制限以上に高くすることはできません。

API Gateway のスロットリングに関連する設定は、次の順序で適用されます。

1. 使用量プランで API ステージに設定したクライアントあたり、またはメソッドあたりのスロットリング制限
2. API ステージに設定したメソッドあたりのスロットリング制限
3. リージョンごとのアカウントレベルのスロットリング
4. AWS リージョンのスロットリング

リージョンごとのアカウントレベルのスロットリング

API Gateway はデフォルトで、リージョンごとに AWS アカウント内のすべての API 全体で定常状態のリクエスト/秒 (RPS) を制限します。また、リージョンごとに AWS アカウント内のすべての API にわたってバースト (最大バケットサイズ) を制限します。API Gateway では、バースト制限は、API Gateway が `429 Too Many Requests` エラーレスポンスを返す前に処理する同時リクエスト送信の目標最大数を表します。スロットリングクォータの詳細については、「クォータと重要な注意点」を参照してください。

使用量プランの API レベルとステージレベルのスロットリング目標の設定

使用量プランでは、 API またはステージレベルでのすべてのメソッドのメソッドあたりのスロットリング目標を設定できます。スロットリングレート (1 秒あたりのリクエスト数) を指定できます。スロットリングレートとは、トークンバケットにトークンが追加されるレートです。また、トークンバケットの容量であるスロットリングバーストを指定することもできます。

使用量プランを作成するには、AWS CLI、SDK、AWS Management Console を使用できます。使用量プランを作成する方法の詳細については、「???」を参照してください。

ステージレベルのスロットリングターゲットの設定

AWS CLI、SDK、および AWS Management Console を使用して、ステージレベルのスロットリングターゲットを作成できます。

AWS Management Console を使用してステージレベルのスロットリングターゲットを作成する方法の詳細については、「???」を参照してください。AWS CLI を使用してステージレベルのスロットリングターゲットを作成する方法の詳細については、「create-stage」を参照してください。

使用量プランのメソッドレベルのスロットリング目標の設定

[使用量プラン] で、使用量プランを作成する に示すように、メソッドレベルの追加のスロットリング目標を設定できます。これらは、API Gateway コンソールの、[メソッドスロットリングの設定] 設定で、`Resource=<resource>` 、`Method=<method>` を指定して設定されます。たとえば、PetStore サンプルに、`Resource=/pets`、`Method=GET` を指定できます。

Amazon API Gateway のプライベート REST API

プライベート API は、Amazon VPC 内からのみ呼び出せる REST API です。API にアクセスするには、インターフェイス VPC エンドポイントを使用します。これは、VPC で作成するエンドポイントネットワークインターフェイスです。インターフェイスエンドポイントは、プライベート IP アドレスを経由して AWS サービスにプライベートにアクセスできるテクノロジーである AWS PrivateLink により強化されています。

AWS Direct Connect を使用してオンプレミスネットワークから Amazon VPC への接続を確立し、この接続を介してプライベート API にアクセスすることも可能です。いずれの場合も、プライベート API へのトラフィックは安全な接続を使用し、パブリックインターネットからは隔離されます。トラフィックは、Amazon ネットワークを離れません。

プライベート API のベストプラクティス

プライベート API を作成するときは、以下のベストプラクティスを使用することをお勧めします。

- 単一の VPC エンドポイントを使用して複数のプライベート API にアクセスします。これにより、必要な VPC エンドポイントの数が減少します。
- VPC エンドポイントを API に関連付けます。これにより、Route 53 エイリアス DNS レコードが作成され、プライベート API の呼び出しが簡単になります。
- VPC のプライベート DNS をオンにします。これにより、ホストまたは `x-apigw-api-id` ヘッダーを渡すことなく、VPC 内で API を呼び出すことができます。プライベート DNS を有効にしないことを選択した場合、API にはパブリック DNS を経由でのみアクセスできるようになります。
- プライベート API へのアクセスを特定の VPC または VPC エンドポイントに制限します。API のリソースポリシーに `aws:SourceVpc` 条件や `aws:SourceVpce` 条件を追加して、アクセスを制限します。
- データ境界を最も安全にするには、VPC エンドポイントポリシーを作成できます。これにより、プライベート API を呼び出せる VPC エンドポイントへのアクセスを制御します。

プライベート API に関する考慮事項

以下の考慮事項は、プライベート API の使用に影響する可能性があります。

- REST API のみがサポートされています。
- カスタムドメイン名は、プライベート API ではサポートされません。
- プライベート API をエッジ最適化 API に変換することはできません。
- プライベート API は、TLS 1.2 のみをサポートします。以前の TLS バージョンはサポートされていません。
- プライベート API の VPC エンドポイントには、他のインターフェイス VPC エンドポイントと同じ制限が適用されます。詳細については、「AWS PrivateLink ガイド」の「Access an AWS service using an interface VPC endpoint」(インターフェイス VPC エンドポイントを使用してのサービスにアクセスする) を参照してください。API Gateway を共有 VPC および共有サブネットで使用する方法の詳細については、「AWS PrivateLink ガイド」の「共有サブネット」を参照してください。

プライベート API に関する次のステップ

プライベート API を作成し、VPC エンドポイントを関連付ける方法については、「the section called “プライベート API の作成”」を参照してください。チュートリアルに従って、AWS Management Console の依存関係とプライベート API を AWS CloudFormation で作成するには、「the section called “チュートリアル: プライベート REST API を構築する”」を参照してください。

プライベート API の作成

プライベート API を作成する前に、まず API Gateway 用の VPC エンドポイントを作成します。次に、プライベート API を作成してリソースポリシーをアタッチします。必要に応じて、VPC エンドポイントをプライベート API に関連付けることで、API の呼び出し方法を簡素化できます。最後に、API をデプロイします。

これを行う方法を、以下の手順で示します。プライベート REST API は、AWS Management Console、AWS CLI、または AWS SDK を使用して作成できます。

前提条件

この手順を実行するには、完全に設定された VPC が必要です。デフォルトの VPC を作成するには、「Amazon VPC ユーザーガイド」の「VPC のみを作成する」を参照してください。VPC の作成

時にすべての推奨ステップを実行するには、プライベート DNS を有効にします。これにより、ホストまたは `x-apigw-api-id` ヘッダーを渡すことなく、VPC 内で API を呼び出すことができます。

プライベート DNS を有効にするには、VPC の `enableDnsSupport` 属性と `enableDnsHostnames` 属性を `true` に設定する必要があります。詳細については、「VPC の DNS サポート」および「VPC の DNS サポートの更新」を参照してください。

ステップ 1: API Gateway 用の VPC エンドポイントを VPC に作成する

次の手順は、API Gateway 用の VPC エンドポイントを作成する方法を示しています。API Gateway 用の VPC エンドポイントを作成するには、プライベート API を作成する先の AWS リージョンで `execute-api` ドメインを指定します。`execute-api` ドメインは、API 実行用の API Gateway コンポーネントサービスです。

API Gateway 用の VPC エンドポイントを作成するときは、DNS 設定を指定します。プライベート DNS をオフにすると、パブリック DNS を介してのみ API にアクセスできます。詳細については、「the section called “問題: API Gateway VPC エンドポイントからパブリック API に接続できません”」を参照してください。

AWS Management Console

API Gateway 用のインターフェイス VPC エンドポイントを作成するには

1. AWS Management Console にサインインして、Amazon VPC コンソール (https://console.aws.amazon.com/vpc/) を開きます。
2. ナビゲーションペインの [仮想プライベートクラウド] で、[VPC] を選択します。
3. [エンドポイントの作成] を選択します。
4. (オプション) [名前タグ] に、VPC エンドポイントを識別するのに役立つ名前を入力します。
5. [Service category] (サービスカテゴリ) で、[AWS services] (のサービス) を選択します。
6. [サービス] で、検索バーに「**`execute-api`**」と入力します。次に、API を作成する先の AWS リージョンで API Gateway サービスエンドポイントを選択します。サービス名は `com.amazonaws.us-east-1.execute-api` のようになり、[タイプ] は [インターフェイス] になります。
7. [VPC] の場合は、エンドポイントを作成する VPC を選択します。
8. (オプション) [プライベート DNS 名を有効化] をオフにするには、[その他の設定] を選択し、[プライベート DNS 名を有効化] をオフにします。
9. [サブネット] で、エンドポイントネットワークインターフェイスを作成したアベイラビリティーゾーンを選択します。API の可用性を高めるには、複数のサブネットを選択します。

10. [セキュリティグループ] で、VPC エンドポイントネットワークインターフェイスに関連付けるセキュリティグループを選択します。

    選択したセキュリティグループでは、VPC の IP 範囲または VPC 内の別のセキュリティグループのいずれかからの TCP ポート 443 インバウンド HTTPS トラフィックを許可するように設定する必要があります。

11. [ポリシー] で、以下のいずれかを実行します。
    - プライベート API を作成していないか、カスタム VPC エンドポイントポリシーを設定しない場合は、[フルアクセス] を選択します。
    - プライベート API を既に作成していて、カスタム VPC エンドポイントポリシーを設定する場合は、カスタム VPC エンドポイントポリシーを入力できます。詳細については、「the section called “プライベート API 用の VPC エンドポイントポリシーを使用する”」を参照してください。

    VPC エンドポイントポリシーは、VPC エンドポイントの作成後に更新できます。詳細については、「VPC エンドポイントポリシーを更新する」を参照してください。

12. [エンドポイントの作成] を選択します。
13. 結果の VPC エンドポイント ID をコピーしておきます。これは、以降のステップで使用する場合があります。

AWS CLI

次の create-vpc-endpoint コマンドを使用して VPC エンドポイントを作成できます。

```
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
```

結果の VPC エンドポイント ID をコピーしておきます。これは、以降のステップで使用する場合があります。

ステップ 2: プライベート API を作成する

VPC エンドポイントを作成したら、プライベート REST API を作成します。次の手順で、プライベート API を作成する方法を示します。

AWS Management Console

プライベート API を作成するには

1. https://console.aws.amazon.com/apigateway で API Gateway コンソールにサインインします。
2. [API の作成] を選択します。
3. [REST API] で、[構築] を選択します。
4. [ロール名] に名前を入力します。
5. (オプション) [説明] に説明を入力します。
6. [API エンドポイントタイプ] で、[プライベート] を選択します。
7. (オプション) [VPC エンドポイント ID] に、VPC エンドポイント ID を入力します。

   VPC エンドポイント ID をプライベート API に関連付ける場合は、`Host` ヘッダーを上書きしたり、`x-apigw-api-id header` を渡したりすることなく、VPC 内から API を呼び出すことができます。詳細については、「the section called “(オプション) VPC エンドポイントとプライベート API の関連付けまたは関連付けの解除”」を参照してください。
8. API の作成 を選択します。

ここまでの手順を完了したら、「the section called “REST API コンソールの開始方法”」の手順に従って、この API のメソッドと統合を設定できます。ただし、API をデプロイすることはできません。API をデプロイするには、ステップ 3 に従い、リソースポリシーを API にアタッチします。

AWS CLI

次の update-rest-api コマンドは、プライベート API を作成する方法を示しています。

```
aws apigateway create-rest-api \
        --name 'Simple PetStore (AWS CLI, Private)' \
        --description 'Simple private PetStore API' \
        --region us-west-2 \
        --endpoint-configuration '{ "types": ["PRIVATE"] }'
```

呼び出しに成功すると、次のような出力が返されます。

```
{
    "createdDate": "2017-10-13T18:41:39Z",
    "description": "Simple private PetStore API",
```

```
    "endpointConfiguration": {
        "types": "PRIVATE"
    },
    "id": "0qzs2sy7bh",
    "name": "Simple PetStore (AWS CLI, Private)"
}
```

ここまでの手順を完了したら、「the section called “チュートリアル: AWS SDKまたは AWS CLI を使用してエッジ最適化 API を作成する”」の手順に従って、この API のメソッドと統合を設定できます。ただし、API をデプロイすることはできません。API をデプロイするには、ステップ 3 に従い、リソースポリシーを API にアタッチします。

SDK JavaScript v3

次の例は、AWS SDK for JavaScript v3 を使用してプライベート API を作成する方法を示しています。

```
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)
 }
};
```

呼び出しに成功すると、次のような出力が返されます。

```
{
  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'
}
```

ここまでの手順を完了したら、「the section called "チュートリアル: AWS SDKまたは AWS CLI を使用して エッジ最適化 API を作成する"」の手順に従って、この API のメソッドと統合を設定できます。ただし、API をデプロイすることはできません。API をデプロイするには、ステップ 3 に従い、リソースポリシーを API にアタッチします。

Python SDK

次の例は、 AWS SDK for Python を使用してプライベート API を作成する方法を示しています。

```
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
```

呼び出しに成功すると、次のような出力が返されます。

```
"{\"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\"}"
```

ここまでの手順を完了したら、「the section called “チュートリアル: AWS SDKまたは AWS CLI を使用して エッジ最適化 API を作成する”」の手順に従って、この API のメソッドと統合を設定できます。ただし、API をデプロイすることはできません。API をデプロイするには、ステップ 3 に従い、リソースポリシーを API にアタッチします。

ステップ 3: プライベート API のリソースポリシーをセットアップする

現在のプライベート API は、すべての VPC からアクセスできるわけではありません。リソースポリシーを使用して、VPC と VPC エンドポイントにプライベート API へのアクセスを許可します。任意の AWS アカウントで VPC エンドポイントへのアクセスを許可できます。

リソースポリシーには、アクセスを制限するための `aws:SourceVpc` 条件や `aws:SourceVpce` 条件を含める必要があります。特定の VPC と VPC エンドポイントに限定することにして、すべての VPC や VPC エンドポイントにアクセスを許可するリソースポリシーは作成しないようお勧めします。

次の手順は、リソースポリシーを API にアタッチする方法を示しています。

AWS Management Console

1. https://console.aws.amazon.com/apigateway で API Gateway コンソールにサインインします。
2. REST API を選択します。
3. 左のナビゲーションペインで、[リソースポリシー] を選択します。
4. [Create policy] を選択します。
5. [テンプレートを選択] を選択し、[ソース VPC] を選択します。
6. `{{vpceID}}` (中括弧を含む) を VPC エンドポイント ID に置き換えます。
7. [Save changes] (変更の保存) をクリックします。

AWS CLI

次の update-rest-api コマンドは、リソースポリシーを既存の API にアタッチする方法を示しています。

```
aws apigateway update-rest-api \
    --rest-api-id a1b2c3 \
    --patch-operations op=replace,path=/
policy,value='"{\"jsonEscapedPolicyDocument\"}"'
```

また、VPC エンドポイントにアクセスできるリソースを制御することもできます。VPC エンドポイントにアクセスできるリソースを制御するには、エンドポイントポリシーを VPC エンドポイントにアタッチします。詳細については、「the section called “プライベート API 用の VPC エンドポイントポリシーを使用する”」を参照してください。

(オプション) VPC エンドポイントとプライベート API の関連付けまたは関連付けの解除

VPC エンドポイントをプライベート API に関連付けると、API Gateway は新しい Route 53 エイリアス DNS レコードを生成します。このレコードを使用して、`Host` ヘッダーを上書きしたり、`x-apigw-api-id` ヘッダーを渡したりすることなく、パブリック API を呼び出す場合と同じように、プライベート API を呼び出すことができます。

生成されたベース URL は次の形式になります。

```
https://{rest-api-id}-{vpce-id}.execute-api.{region}.amazonaws.com/{stage}
```

Associate a VPC endpoint (AWS Management Console)

VPC エンドポイントは、作成時または作成後にプライベート API に関連付けることができます。次の手順は、VPC エンドポイントを以前に作成した API に関連付ける方法を示しています。

VPC エンドポイントをプライベート API に関連付けるには

1. https://console.aws.amazon.com/apigateway で API Gateway コンソールにサインインします。
2. プライベート API を選択します。
3. 左のナビゲーションペインで、[リソースポリシー] を選択します。
4. 追加の VPC エンドポイントからの呼び出しを許可するようにリソースポリシーを編集します。

5. メインナビゲーションペインで、[API キー] を選択します。
6. [API の詳細] セクションで [編集] を選択します。
7. [VPC エンドポイント ID] で、追加の VPC エンドポイント ID を選択します。
8. [Save] を選択します。
9. 変更を有効にするには、API を再デプロイします。

Dissociate a VPC endpoint (AWS Management Console)

プライベート REST API から VPC エンドポイントの関連付けを解除するには

1. https://console.aws.amazon.com/apigateway で API Gateway コンソールにサインインします。
2. プライベート API を選択します。
3. 左のナビゲーションペインで、[リソースポリシー] を選択します。
4. リソースポリシーを編集して、プライベート API との関連付けを解除する VPC エンドポイントへの言及を削除します。
5. メインナビゲーションペインで、[API キー] を選択します。
6. [API の詳細] セクションで [編集] を選択します。
7. [VPC エンドポイント ID] で、[X] を選択して VPC エンドポイントの関連付けを解除します。
8. [Save] を選択します。
9. 変更を有効にするには、API を再デプロイします。

Associate a VPC endpoint (AWS CLI)

次の create-rest-api コマンドは、API の作成時に VPC エンドポイントを関連付ける方法を示しています。

```
aws apigateway create-rest-api \
    --name Petstore \
    --endpoint-configuration '{ "types": ["PRIVATE"], "vpcEndpointIds" :
 ["vpce-0212a4ababd5b8c3e", "vpce-0393a628149c867ee"] }' \
    --region us-west-2
```

出力は次のようになります。

```
{
    "apiKeySource": "HEADER",
    "endpointConfiguration": {
        "types": [
            "PRIVATE"
        ],
        "vpcEndpointIds": [
            "vpce-0212a4ababd5b8c3e",
            "vpce-0393a628149c867ee"
        ]
    },
    "id": "u67n3ov968",
    "createdDate": 1565718256,
    "name": "Petstore"
}
```

次の update-rest-api コマンドは、VPC エンドポイントを作成済みの API に関連付ける方法を示しています。

```
aws apigateway update-rest-api \
    --rest-api-id u67n3ov968 \
    --patch-operations "op='add',path='/endpointConfiguration/
vpcEndpointIds',value='vpce-01d622316a7df47f9'" \
    --region us-west-2
```

出力は次のようになります。

```
{
    "name": "Petstore",
    "apiKeySource": "1565718256",
    "tags": {},
    "createdDate": 1565718256,
    "endpointConfiguration": {
        "vpcEndpointIds": [
            "vpce-0212a4ababd5b8c3e",
            "vpce-0393a628149c867ee",
            "vpce-01d622316a7df47f9"
        ],
        "types": [
            "PRIVATE"
        ]
```

```
    },
    "id": "u67n3ov968"
}
```

変更を有効にするには、API を再デプロイします。

Disassociate a VPC endpoint (AWS CLI)

次の update-rest-api コマンドは、プライベート API から VPC エンドポイントの関連付けを解除する方法を示しています。

```
aws apigateway update-rest-api \
    --rest-api-id u67n3ov968 \
    --patch-operations "op='remove',path='/endpointConfiguration/
vpcEndpointIds',value='vpce-0393a628149c867ee'" \
    --region us-west-2
```

出力は次のようになります。

```
{
    "name": "Petstore",
    "apiKeySource": "1565718256",
    "tags": {},
    "createdDate": 1565718256,
    "endpointConfiguration": {
        "vpcEndpointIds": [
            "vpce-0212a4ababd5b8c3e",
            "vpce-01d622316a7df47f9"
        ],
        "types": [
            "PRIVATE"
        ]
    },
    "id": "u67n3ov968"
}
```

変更を有効にするには、API を再デプロイします。

ステップ 4: プライベート API をデプロイする

API をデプロイするには、API デプロイを作成してステージに関連付けます。次の手順は、プライベート API をデプロイする方法を示しています。

AWS Management Console

プライベート API をデプロイするには

1. API を選択します。
2. [API のデプロイ] を選択します。
3. [ステージ] で [新規ステージ] を選択します。
4. [ステージ名] に、ステージ名を入力します。
5. (オプション) [説明] に説明を入力します。
6. [デプロイ] を選択します。

AWS CLI

次の create-deployment コマンドは、プライベート API をデプロイする方法を示しています。

```
aws apigateway create-deployment --rest-api-id a1b2c3 \
  --stage-name test \
  --stage-description 'Private API test stage' \
  --description 'First deployment'
```

プライベート API のトラブルシューティング

次に、プライベート API の作成時に発生する可能性があるエラーや問題に関するトラブルシューティングのアドバイスを示します。

問題: API Gateway VPC エンドポイントからパブリック API に接続できません

VPC を作成するときに、DNS 設定を構成できます。VPC のプライベート DNS をオンにすることをお勧めします。プライベート DNS をオフにすることを選択した場合、API にはパブリック DNS 経由でのみアクセスできます。

プライベート DNS を有効にすると、VPC エンドポイントからパブリック API Gateway API のデフォルトエンドポイントにアクセスできなくなります。カスタムドメイン名を使用して API にアクセスできます。

リージョンのカスタムドメイン名を作成する場合は、A タイプのエイリアスレコードを使用します。エッジ最適化カスタムドメイン名を作成する場合は、レコードタイプに制限はありません。これらのパブリック API には、プライベート DNS を有効にしてアクセスできます。詳細については、「問

題: API Gateway VPC エンドポイントからパブリック API に接続できません」を参照してください。

問題: API が **{"Message":"User: anonymous is not authorized to perform: execute-api:Invoke on resource: arn:aws:execute-api:us-east-1:********/****/****/"}** を返します

リソースポリシーで、プリンシパルを次のように AWS プリンシパルに設定する場合:

```
{
    "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:/*"
            ]
        },
        ...
        }
```

API のすべてのメソッドに AWS_IAM 認証を使用する必要があります。使用しない場合、API は前のエラーメッセージを返します。メソッドの AWS_IAM 認証を有効にする方法の詳細については、「the section called “方法”」を参照してください。

問題: VPC エンドポイントが API に関連付けられているかどうかわかりません

VPC エンドポイントをプライベート API に関連付けるか、関連付けを解除する場合は、API を再デプロイする必要があります。DNS 伝播のため、更新オペレーションが完了するまでに数分かかることがあります。この間、API は使用できますが、新しく生成された DNS URL の DNS 伝達がまだ進行中である可能性があります。数分経っても新しい URL が DNS に解決されない場合は、API を再デプロイすることをお勧めします。

プライベート API の呼び出し

プライベート API は VPC 内からのみ呼び出すことができます。プライベート API には、特定の VPC や VPC エンドポイントに API の呼び出しを許可するリソースポリシーが必要です。

プライベート API は、次の方法で呼び出すことができます。

- Route53 エイリアスを使用して API を呼び出します。これは、VPC エンドポイントを API に関連付けている場合にのみ使用できます。詳細については、「the section called “(オプション) VPC エンドポイントとプライベート API の関連付けまたは関連付けの解除”」を参照してください。
- プライベート DNS を使用して API を呼び出します。これは、VPC のプライベート DNS を有効にしている場合にのみ使用できます。
- AWS Direct Connect を使用して API を呼び出します。
- エンドポイント固有のパブリック DNS ホスト名を使用して API を呼び出します。

DNS 名を使用してプライベート API を呼び出すには、API の DNS 名を特定する必要があります。次の手順は、DNS 名を検索する方法を示しています。

AWS Management Console

DNS 名を見つけるには

1. AWS Management Console にサインインして、Amazon VPC コンソール (https://console.aws.amazon.com/vpc/) を開きます。
2. メインナビゲーションペインで、[エンドポイント] を選択し、API Gateway 用のインターフェイス VPC エンドポイントを選択します。
3. [詳細] ペインで、[DNS 名] フィールドに 5 つの値が表示されます。最初の 3 つは API のパブリック DNS 名です。残りの 2 つはプライベート DNS 名です。

AWS CLI

次の describe-vpc-endpoints コマンドを使用して、DNS 値を一覧表示します。

```
aws ec2 describe-vpc-endpoints --filters vpc-endpoint-id=vpce-01234567abcdef012
```

最初の 3 つは API のパブリック DNS 名です。残りの 2 つはプライベート DNS 名です。

Route53 エイリアスを使用したプライベート API の呼び出し

VPC エンドポイントとプライベート API を関連付けたり、関連付けを解除したりできます。詳細については、「the section called "(オプション) VPC エンドポイントとプライベート API の関連付けまたは関連付けの解除"」を参照してください。

VPC エンドポイントをプライベート API に関連付けると、次のベース URL を使用して API を呼び出すことができます。

```
https://{rest-api-id}-{vpce-id}.execute-api.{region}.amazonaws.com/{stage}
```

例えば、test ステージで GET /pets メソッドを設定し、REST API ID が 01234567ab、VPC エンドポイント ID が vpce-01234567abcdef012、リージョンが us-west-2 である場合は、次のように API を呼び出すことができます。

```
curl -v https://01234567ab-vpce-01234567abcdef012.execute-api.us-west-2.amazonaws.com/
test/pets
```

プライベート DNS 名を使用してプライベート API を呼び出す

プライベート DNS を有効にしている場合は、次のプライベート DNS 名を使用してプライベート API にアクセスできます。

```
{restapi-id}.execute-api.{region}.amazonaws.com
```

API を呼び出すためのベース URL は次の形式です。

```
https://{restapi-id}.execute-api.{region}.amazonaws.com/{stage}
```

例えば、test ステージで GET /pets メソッドをセットアップし、REST API ID が 01234567ab、リージョンが us-west-2 である場合は、ブラウザに次の URL を入力してプライベート API を呼び出すことができます。

```
https://01234567ab.execute-api.us-west-2.amazonaws.com/test/pets
```

または、次の cURL コマンドを使用してプライベート API を呼び出すこともできます。

```
curl -X GET https://01234567ab.execute-api.us-west-2.amazonaws.com/test/pets
```

> **Warning**
>
> VPC エンドポイントのプライベート DNS を有効にすると、パブリック API のデフォルトエンドポイントにアクセスできます。詳細については、「API Gateway VPC エンドポイントからパブリック API に接続できないのはなぜですか?」を参照してください。

AWS Direct Connect を使用してプライベート API を呼び出す

AWS Direct Connect を使用して、オンプレミスネットワークから Amazon VPC への専用のプライベート接続を確立し、この接続経由でパブリック DNS 名を使用してプライベート API エンドポイントにアクセスできます。

プライベート DNS 名を使用すると、Amazon Route 53 Resolver インバウンドエンドポイントを設定してリモートネットワークからプライベート DNS のすべての DNS クエリを転送することによって、オンプレミスネットワークからプライベート API にアクセスすることもできます。詳細については、Amazon Route 53 デベロッパーガイドの「VPC へのインバウンド DNS クエリの転送」を参照してください。

エンドポイント固有のパブリック DNS ホスト名を使用してプライベート API を呼び出す

エンドポイント固有のパブリック DNS ホスト名を使用してたプライベート API にアクセスできます。これらは、プライベート API 用の VPC エンドポイント ID または API ID を含むパブリック DNS ホスト名です。

生成されたベース URL は次の形式になります。

```
https://{public-dns-hostname}.execute-api.{region}.vpce.amazonaws.com/{stage}
```

例えば、`test` ステージで `GET /pets` メソッドを設定し、REST API ID が `abc1234`、パブリック DNS ホスト名が `vpce-def-01234567`、リージョンが `us-west-2`である場合、cURL コマンドの `Host` ヘッダーで VPCe ID を使用してプライベート API を呼び出すことができます。

```
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'
```

または、cURL コマンドで `x-apigw-api-id` ヘッダーを次の形式で使用し、API ID を経由してプライベート API を呼び出すことができます。

```
curl -v https://{public-dns-hostname}.execute-api.{region}.vpce.amazonaws.com/{stage} -
H 'x-apigw-api-id:{api-id}'
```

REST API のモニタリング

このセクションでは、CloudWatch のメトリクス、CloudWatch Logs、Firehose、AWS X-Ray を使用して API をモニタリングする方法について説明します。CloudWatch の実行ログと CloudWatch のメトリクスを組み合わせることにより、エラーの記録と実行のトレースを行って、API のパフォーマンスをモニタリングすることができます。API コールを Firehose のログに記録することもできます 。AWS X-Ray を使用して、API を構成するダウンストリームサービスを通じて呼び出しをトレースすることもできます。

> **Note**
>
> API Gateway は、次の場合にログとメトリクスが生成されない可能性があります。
>
> - 413 Request Entity Too Large エラー
> - 過剰な 429 Too Many Requests エラー
> - API マッピングを持たないカスタムドメインに送信されたリクエストからの 400 シリーズのエラー
> - 内部の障害によって発生した 500 シリーズのエラー
>
> API Gateway は、REST API メソッドをテストするときに、ログとメトリクスを生成しません。CloudWatch エントリがシミュレートされます。詳細については、「the section called “コンソールを使用した REST API メソッドのテスト”」を参照してください。

トピック

- Amazon CloudWatch のメトリクスを使用した REST API の実行のモニタリング
- API Gateway での CloudWatch による REST API のログの設定
- Amazon Data Firehose への API コールのログ記録
- X-Ray を使用した REST API へのユーザーリクエストのトレース

Amazon CloudWatch のメトリクスを使用した REST API の実行のモニタリング

CloudWatch を使用して API の実行をモニタリングすることで、API Gateway から raw データを収集し、リアルタイムに近い読み取り可能なメトリクスに加工することができます。これらの統計は 15 か月間記録されるため、履歴情報にアクセスしてウェブアプリケーションやサービスの動作をより的確に把握できます。デフォルトでは、API Gateway のメトリクスデータは 1 分間隔で自動的に CloudWatch に送信されます。詳細については、Amazon CloudWatch ユーザーガイドの「Amazon CloudWatch とは」を参照してください。

API Gateway からレポートされるメトリクスには、さまざまな方法で分析できる情報が含まれています。以下に、メトリクスの一般的な使用方法を示します。これらの使用方法を参考にしてください。

- バックエンドの応答性を測定するため、[IntegrationLatency] メトリクスをモニタリングします。
- API コールの全体的な応答性を測定するために、Latency メトリクスをモニタリングします。
- 目的のパフォーマンスの実現に向けてキャッシュ容量を最適化するために、CacheHitCount と CacheMissCount メトリクスをモニタリングします。

トピック

Amazon API Gateway のディメンションとメトリクス

API Gateway が Amazon CloudWatch に送信するメトリクスとディメンションを以下に示します。詳細については、「Amazon CloudWatch のメトリクスを使用した REST API の実行のモニタリング」を参照してください。

API Gateway のメトリクス

Amazon API Gateway は、メトリクスデータを 1 分ごとに CloudWatch に送信します。

AWS/ApiGateway 名前空間には、次のメトリクスが含まれます。

メトリクス	説明
4XXError	指定された期間に取得されたクライアント側エラーの数。 API Gateway は、変更されたゲートウェイのレスポンスステータスコードを 4xxError エラーとしてカウントします。 Sum 統計は、このメトリクス、つまり、指定された期間内の 4XXError エラーの合計数を表します。Average 統計は、4XXError のエラー率 (4XXError エラーの合計数をその期間のリクエストの合計数で割った値) を表します。分母は Count メトリクス (下記) に対応します。 Unit: Count
5XXError	指定された期間に取得されたサーバー側エラーの数。 Sum 統計は、このメトリクス、つまり、指定された期間内の 5XXError エラーの合計数を表します。Average 統計は、5XXError のエラー率 (5XXError エラーの合計数をその期間のリクエストの合計数で割った値) を表します。分母は Count メトリクス (下記) に対応します。 Unit: Count
CacheHitCount	指定された期間内に API キャッシュから配信されたリクエストの数。 Sum 統計は、このメトリクス、つまり、指定された期間内のキャッシュヒットの合計数を表します。Average 統計は、キャッシュヒット率、つまり、キャッシュヒットの合計数をその期間のリクエストの合計数で割ったものです。分母は Count メトリクス (下記) に対応します。

メトリクス	説明
	Unit: Count
CacheMissCount	API キャッシュが有効になっている特定の期間における、バックエンドから提供されたリクエストの数。 Sum 統計は、このメトリクス、つまり、指定された期間内のキャッシュミスの合計数を表します。Average 統計は、キャッシュミス率、つまり、キャッシュミスの合計数をその期間のリクエストの合計数で割ったものです。分母は Count メトリクス (下記) に対応します。 Unit: Count
Count	指定期間内の API リクエストの合計数。 SampleCount 統計は、このメトリクスを表します。 Unit: Count
IntegrationLatency	API Gateway がバックエンドにリクエストを中継してから、バックエンドからレスポンスを受け取るまでの時間。 Unit: Millisecond
Latency	API Gateway がクライアントからリクエストを受け取ってから、クライアントにレスポンスを返すまでの時間。レイテンシーには、統合のレイテンシーおよびその他の API Gateway のオーバーヘッドが含まれます。 Unit: Millisecond

メトリクスのディメンション

API Gateway のメトリクスをフィルタリングするには、次の表のディメンションを使用できます。

> **Note**
>
> API Gateway は、メトリクスを CloudWatch に送信する前に、ApiName ディメンションから ASCII 以外の文字を削除します。APIName に ASCII 文字が含まれていない場合、API ID は ApiName として使用されます。

ディメンション	説明
ApiName	指定した API 名を使用して、API Gateway の REST API のメトリクスをフィルタリングします。
ApiName, Method, Resource, Stage	指定した API 名、ステージ、リソース、メソッドを使用して、API Gateway の API のメソッドのメトリクスをフィルタリングします。 詳細な CloudWatch のメトリクスを明示的に有効にしない限り、API Gateway はこれらのメトリクスを送信しません。コンソールでステージを選択し、[ログとトレース] で [編集] を選択します。[詳細なメトリクス]、[変更を保存] の順に選択します。update-stage AWS CLI コマンドを呼び出して `metricsEnabled` プロパティを `true` に更新することもできます。 これらのメトリクスを有効にすることで、アカウントに追加料金が発生します。詳細については、Amazon CloudWatch 料金表をご覧ください。
ApiName, Stage	指定した API 名とステージを使用して、API Gateway の API のステージリソースのメトリクスをフィルタリングします。

API Gateway の API ダッシュボードで CloudWatch のメトリクスを表示する

API Gateway コンソールの API ダッシュボードを使用して、API Gateway にデプロイされた API に関する CloudWatch のメトリクスを表示できます。これは、時間の経過に伴う API アクティビティの概要として表示されます。

トピック

前提条件

1. API Gateway で API が作成済みであることが必要です。「」の手順に従いますAPI Gateway での REST API の開発
2. API は少なくとも一度はデプロイする必要があります。「Amazon API Gateway での REST API のデプロイ」の手順に従います

ダッシュボードでの API アクティビティの調査

1. https://console.aws.amazon.com/apigateway で API Gateway コンソールにサインインします。
2. APIを選択します。
3. ナビゲーションペインで、[ダッシュボード] を選択します。
4. [ステージ] では、目的のステージを選択します。
5. [日付範囲] を選択して日付の範囲を指定します。
6. [API コール]、[レイテンシー]、[統合のレイテンシー]、[レイテンシー]、[4xx エラー]、および [5xx エラー] というタイトルの個別のグラフに表示される各メトリクスを、必要に応じて更新して確認します。

> **Tip**
>
> メソッドレベルで CloudWatch のメトリクスを確認するには、メソッドレベルで CloudWatch Logs を有効にしている必要があります。メソッドレベルでログを設定する方法については、「」を参照してくださいAPI Gateway コンソールを使用したステージ設定の更新

CloudWatch コンソールで API Gateway のメトリクスを表示する

メトリクスはまずサービスの名前空間ごとにグループ化され、次に各名前空間内のさまざまなディメンションの組み合わせごとにグループ化されます。API のメトリクスレベルでメトリクスを表示するには、詳細なメトリクスをオンにします。詳細については、「the section called “ステージ設定の更新”」を参照してください。

CloudWatch コンソールを使用して API Gateway のメトリクスを表示するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. 必要に応じて AWS リージョン を変更します。ナビゲーションバーから、AWS リソースがあるリージョンを選択します。
3. ナビゲーションペインで メトリクスを選択します。
4. [すべてのメトリクス] タブで、[API Gateway] を選択します。
5. ステージ別にメトリクスを表示するには、[ステージ別] パネルを選択します。次に、API とメトリクス名を選択します。
6. API 別にメトリクスを表示するには、[API 名別] パネルを選択します。次に、API とメトリクス名を選択します。

AWS CLI を使ってメトリクスを表示するには

1. コマンドプロンプトで、以下のコマンドを使用してメトリクスを一覧表示します。

   ```
   aws cloudwatch list-metrics --namespace "AWS/ApiGateway"
   ```

   メトリクスを作成した後、メトリクスが表示されるまでに最大 15 分かかります。メトリクスの統計をより早く確認するには、get-metric-data または get-metric-statistics を使用します。
2. 特定の統計情報 (Average など) を 5 分間隔で表示するには、以下のコマンドを呼び出します。

   ```
   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
   ```

CloudWatch コンソールで API Gateway のログイベントを表示する

前提条件

1. API Gateway で API が作成済みであることが必要です。「API Gateway での REST API の開発」の手順に従います。
2. API は、少なくとも 1 回、デプロイして呼び出す必要があります。「Amazon API Gateway での REST API のデプロイ」および「Amazon API Gateway での REST API の呼び出し」の手順に従います
3. ステージで CloudWatch Logs を有効にする必要があります。「API Gateway での CloudWatch による REST API のログの設定」の手順に従います。

ログに記録された API リクエストや API レスポンスを CloudWatch コンソールで表示するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. 必要に応じて AWS リージョン を変更します。ナビゲーションバーから、AWS リソースがあるリージョンを選択します。詳細については、「リージョンとエンドポイント」を参照してください。
3. ナビゲーションペインで、[ログ]、[ロググループ] の順に選択します。
4. [ロググループ] 一覧で、"API-Gateway-Execution-Logs_{rest-api-id}/{stage-name}" という名前のロググループを選択します。
5. [ログストリーム] 一覧で、ログストリームを選択します。タイムスタンプを使用して、目的のログストリームを見つけることができます。
6. 未加工のテキストを表示するには、[テキスト] を選択します。または、行ごとにイベントを表示するには、[行] を選択します。

> **Important**
>
> CloudWatch ではロググループやログストリームを削除することができますが、API Gateway の API のロググループやログストリームは手動で削除せず、API Gateway にそれらのリソースを管理させるようにしてください。ロググループまたはストリームを手動で削除すると、API のリクエストとレスポンスがログに記録されなくなる場合があります。その場合は、API のロググループ全体を削除して API を再デプロイしてください。これは、API Gateway によって API のデプロイ時に API ステージのロググループまたはログストリームが作成されるためです。

AWS のモニタリングツール

AWS では、API Gateway のモニタリングに使用できるさまざまなツールを提供しています。これらのツールの中には、自動モニタリングを設定できるものもあれば、手操作を必要とするものもあります。モニタリングタスクをできるだけ自動化することをお勧めします。

AWS の自動モニタリングツール

以下の自動モニタリングツールを使用して、API Gateway を監視し、問題が発生したときにレポートすることができます。

- Amazon CloudWatch アラーム - 指定した期間にわたって単一のメトリクスをモニタリングし、複数の期間にわたる特定のしきい値に対するメトリクスの値に基づいて 1 つ以上のアクションを実行します。アクションは、Amazon Simple Notification Service (Amazon SNS) のトピックまたは Amazon EC2 Auto Scaling のポリシーに送信される通知です。CloudWatch アラームは、特定の状態にあるという理由だけでアクションを呼び出すことはありません。状態が変更され、指定された期間維持されている必要があります。詳細については、「Amazon CloudWatch のメトリクスを使用した REST API の実行のモニタリング」を参照してください。
- Amazon CloudWatch Logs - AWS CloudTrail またはその他の出典からログファイルをモニタリング、保存、およびアクセスします。詳細については、「Amazon CloudWatch ユーザーガイド」の「Amazon CloudWatch Logs とは」を参照してください。
- Amazon EventBridge (旧 CloudWatch Events) - イベントに一致したものを 1 つ以上のターゲットの関数やストリームにルーティングして、変更、状態の情報の収集、是正措置を行います。詳細については、「EventBridge ユーザーガイド」の「Amazon EventBridge とは」を参照してください。
- AWS CloudTrail ログモニタリング - アカウント間でログファイルを共有し、CloudTrail のログファイルを CloudWatch Logs に送信することでそれらをリアルタイムでモニタリングし、ログを処理するアプリケーションを Java で作成し、CloudTrail からの提供後にログファイルが変更されていないことを検証します。詳細については、「AWS CloudTrail ユーザーガイド」の「CloudTrail ログファイルの使用」を参照してください。

手動モニタリングツール

API Gateway のモニタリングでもう 1 つ重要な点は、CloudWatch のアラームの対象外の項目を手動でモニタリングすることです。API Gateway、CloudWatch、その他の AWS のコンソールのダッシュボードは、AWS 環境の状態を一目で把握できるビューを提供します。API 実行のログファイルを確認することもお勧めします。

- API Gateway のダッシュボードには、指定した期間の API の特定のステージに関する以下の統計情報が表示されます。
  - API 呼び出し
  - キャッシュヒット、API キャッシュが有効になっている場合のみ。
  - キャッシュミス、API キャッシュが有効になっている場合のみ。
  - レイテンシー
  - 統合のレイテンシー
  - 4XX エラー
  - 5XX エラー
- CloudWatch のホームページには、以下の情報が表示されます。
  - 現在のアラームとステータス
  - アラームとリソースのグラフ
  - サービスのヘルスステータス

  また、CloudWatch を使用して以下のことを行えます。
  - 重視するサービスをモニタリングするためのカスタマイズしたダッシュボードを作成します
  - メトリクスデータをグラフ化して、問題のトラブルシューティングを行い、傾向を確認する
  - AWS リソースのすべてのメトリクスを検索して、参照する
  - 問題があることを通知するアラームを作成/編集する

API Gateway をモニタリングする CloudWatch のアラームの作成

アラームの状態が変わったら、Amazon SNS メッセージを送信する Amazon CloudWatch のアラームを作成することができます。アラームは、指定期間にわたって単一のメトリクスを監視し、指定したしきい値に対応したメトリクスの値に基づいて、期間数にわたって 1 つ以上のアクションを実行します。アクションは、Amazon SNS のトピックまたはオートスケーリングのポリシーに送信される通知です。アラームは、持続している状態変化に対してのみアクションを呼び出します。CloudWatch のアラームは、メトリクスが特定の状態にあるだけではアクションを呼び出しません。アクションを呼び出すには、指定した期間継続している必要があります。

API Gateway での CloudWatch による REST API の口グの設定

リクエストの実行や API へのクライアントのアクセスに関連する問題のデバッグに役立てるために、Amazon CloudWatch Logs で API コールの口グを有効にすることができます。CloudWatch の詳細については、「the section called “CloudWatch メトリクス”」を参照してください。

API Gateway での CloudWatch によるログの形式

CloudWatch による API のログには、実行ログとアクセスログの 2 種類があります。実行ログでは、API Gateway によって CloudWatch Logs が管理されます。このプロセスには、ロググループとログストリームの作成、および呼び出し元のリクエストとレスポンスのログストリームへのレポートが含まれます。

ログに記録されるデータには、エラーや実行のトレース (リクエストやレスポンスのパラメータ値やペイロードなど)、Lambda オーソライザー (旧カスタムオーソライザー) が使用するデータ、API キーが必要かどうか、使用量プランが有効かどうかなどの情報が含まれます。API Gateway は、認証ヘッダー、API キー値、および同様の機密リクエストパラメータをログデータからマスキングします。

API をデプロイすると、API Gateway はロググループとそのログストリームを作成します。ロググループの名前は `API-Gateway-Execution-Logs_{rest-api-id}/{stage_name}` 形式に従います。各ロググループ内で、ログはログデータにさらに分割され、ログデータがレポートされるときに [最終のイベント時刻] によって順序付けられます。

アクセスログの作成では、API デベロッパーとして、API にアクセスしたユーザーと、呼び出し元が API にアクセスした方法を記録します。独自のロググループを作成したり、既存のロググループを選択したりすることができます。これらは、API Gateway で管理することができます。アクセスの詳細を指定するには、`$context` 変数、ログ形式、ロググループの送信先を選択します。

アクセスログには少なくとも `$context.requestId` または `$context.extendedRequestId` が含まれている必要があります。ベストプラクティスとして、`$context.requestId` と `$context.extendedRequestId` を ログ形式に含めます。

`$context.requestId`

これにより、`x-amzn-RequestId` ヘッダーに値が記録されます。クライアントは、`x-amzn-RequestId` ヘッダーの値を共通の一意の識別子 (UUID) 形式の値で上書きできます。API Gateway は、`x-amzn-RequestId` レスポンスヘッダー内のこのリクエスト ID を返します。API Gateway は、アクセスログの上書きされたリクエスト ID のうち UUID の形式ではないものを *`UUID`*`_REPLACED_INVALID_REQUEST_ID` に置き換えます。

`$context.extendedRequestId`

extendedRequestID は、API Gateway が生成する一意の ID です。API Gateway は、`x-amz-apigw-id` レスポンスヘッダー内のこのリクエスト ID を返します。API 発信者は、このリクエスト ID を提供することやオーバーライドすることはできません。API のトラブルシューティング役立てるために、必要に応じて、この値を AWS サポートに提供します。詳細については、「the

section called “データモデル、オーソライザー、マッピングテンプレート、および CloudWatch アクセスログ記録用の $context 変数”」を参照してください。

> **Note**
>
> $context 変数のみがサポートされます。

Common Log Format (CLF)、JSON、XML、CSV など、分析バックエンドでも採用されているログ形式を選択します。その後、フィードに直接アクセスログを入力して、メトリクスを計算してレンダリングすることができます。ログの形式を定義するには、ロググループの ARN をステージの accessLogSettings/destinationArn プロパティに設定します。ロググループの ARN は、CloudWatch コンソールで取得できます。アクセスログの形式を定義するには、選択した形式をステージの accessLogSetting/format プロパティに設定します。

API Gateway コンソールには、一般的に使用されるアクセスログの形式の例が表示されます。以下にもその例を示します。

- CLF (Common Log Format):

```
$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 (カンマ区切り値):

```
$context.identity.sourceIp,$context.identity.caller,$context.identity.user,
$context.requestTime,$context.httpMethod,$context.resourcePath,$context.protocol,
$context.status,$context.responseLength,$context.requestId,$context.extendedRequestId
```

CloudWatch によるログのアクセス許可

CloudWatch Logs を有効にするには、API Gateway の CloudWatch に対する口グの読み取りと書き込みのアクセス許可をアカウントに付与する必要があります。この AmazonAPIGatewayPushToCloudWatchLogs 管理ポリシー (ARN が arn:aws:iam::aws:policy/service-role/AmazonAPIGatewayPushToCloudWatchLogs) には、すべての必要なアクセス権限があります。

```
{
    "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 は IAM ロールを引き受けるために AWS Security Token Service を呼び出すため、リージョンで AWS STS が有効になっていることを確認します。詳細については、「AWS リージョンでの AWS STS の管理」を参照してください。

これらのアクセス許可をアカウントに付与するには、`apigateway.amazonaws.com` を信頼できるエンティティとして IAM ロールを作成し、前述のポリシーを IAM ロールにアタッチして、その IAM ロールの ARN をアカウントの cloudWatchRoleArn プロパティに設定します。cloudWatchRoleArn プロパティは、CloudWatch Logs を有効にする AWS リージョンごとに設定する必要があります。

IAM ロール ARN の設定時にエラーが発生した場合は、AWS Security Token Service アカウントの設定をチェックして、使用しているリージョンで AWS STS が有効になっていることを確認してください。AWS STS を有効にする方法の詳細については、IAM ユーザーガイドの「AWS リージョンでの AWS STS の管理」を参照してください。

API Gateway コンソールを使用した CloudWatch による API のログの設定

CloudWatch による API のログを設定するには、API をステージにデプロイしておく必要があります。また、アカウントに適切な CloudWatch Logs ロールの ARN を設定しておく必要があります。

1. https://console.aws.amazon.com/apigateway で API Gateway コンソールにサインインします。
2. メインナビゲーションペインで [設定] を選択し、[ログ] で [編集] を選択します。
3. [CloudWatch ログのロール ARN] に、適切な権限を持つ IAM ロールの ARN を入力します。API Gateway を使用して API を作成する AWS アカウント ごとにこれを実行する必要があります。
4. メインナビゲーションペインで、[API] を選択して、以下のいずれかを実行します。
   a. 既存の API を選択し、ステージを選択します。
   b. API を作成してから、ステージにデプロイします。
5. メインナビゲーションペインで、[ステージ] を選択します。
6. [ログおよびトレース] セクションで [編集] を選択します。
7. 実行ログ作成を有効にするには
   a. [CloudWatch Logs] ドロップダウンメニューからログ記録レベルを選択します。ログ記録レベルは、以下のとおりです。
      - オフ — この段階ではログ記録はオンになっていません。

- エラーのみ — ログ記録はエラーに対してのみ有効になっています。
- エラーと情報ログ — ログ記録はすべてのイベントに対して有効になっています。
- リクエストとレスポンスの完全なログ — 詳細なログ記録がすべてのイベントに対して有効になっています。このログは API のトラブルシューティングに役立ちますが、機密データが記録される可能性があります。

> **Note**
>
> 本稼働用 API には、[リクエストとレスポンスの完全なログ] を有効にしないことをお勧めします。

b. 必要に応じて、[詳細メトリクス] を選択して、詳細な CloudWatch メトリクスを有効にします。

CloudWatch のメトリクスの詳細については、「the section called “CloudWatch メトリクス”」を参照してください。

8. アクセスログの作成を有効にするには

   a. [カスタムアクセスロギング] を有効にします。

   b. [アクセスログの送信先 ARN] に、ロググループの ARNを入力します。ARN 形式は `arn:aws:logs:{region}:{account-id}:log-group:log-group-name` です。

   c. [ログの形式] にログの形式を入力します。[CLF]、[JSON]、[XML]、または [CSV] を選択できます。ログ形式の例について詳しくは、「the section called “API Gateway での CloudWatch による ログの形式”」を参照してください。

9. [Save changes] (変更の保存) をクリックします。

> **Note**
>
> 実行ログとアクセスログを互いに独立して有効にすることができます。

これで、API Gateway で API へのリクエストのログを記録する準備が整いました。ステージ設定、ログ、またはステージ変数を更新するときに API を再デプロイする必要はありません。

AWS CloudFormation を使用した CloudWatch API ログ記録の設定

次の AWS CloudFormation テンプレート例を使用して Amazon CloudWatch Logs ロググループを作成し、ステージの実行およびアクセスログ記録を設定します。CloudWatch Logs を有効にするには、API Gateway の CloudWatch に対するログの読み取りと書き込みのアクセス許可をアカウントに付与する必要があります。詳細については、「AWS CloudFormation ガイド」の「アカウントに IAM ロールを関連付ける」を参照してください。

```
 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
```

Amazon Data Firehose への API コールのログ記録

API へのクライアントのアクセスに関連する問題のデバッグに役立てるために、API コールのログを Amazon Data Firehose に記録できます。Firehose の詳細については、「Amazon Data Firehose とは」を参照してください。

アクセスログでは、CloudWatch または Firehose のいずれかのみを有効にできます。両方を有効にすることはできません。ただし、実行ログで CloudWatch を有効にし、アクセスログで Firehose を有効にすることはできます。

トピック

- API Gateway の Firehose ログ形式
- Firehose ログ記録のアクセス許可
- API Gateway コンソールを使用して Firehose アクセスログ記録を設定する

API Gateway の Firehose ログ形式

Firehose ログ記録では、CloudWatch ログ記録と同じ形式を使用します。

Firehose ログ記録のアクセス許可

ステージで Firehose アクセスログ記録を有効にすると、API Gateway はアカウントにサービスリンクロールを作成します (ロールがまだ存在しない場合)。このロールは `AWSServiceRoleForAPIGateway` という名前で、`APIGatewayServiceRolePolicy` マネージドポリシーがロールにアタッチされます。サービスにリンクされたロールの詳細については、「サービスにリンクされたロールの使用」を参照してください。

> **Note**
>
> Firehose ストリームの名前は `amazon-apigateway-{your-stream-name}` にする必要があります。

API Gateway コンソールを使用して Firehose アクセスログ記録を設定する

API ログ記録を設定するには、API をステージにデプロイしている必要があります。また、Firehose ストリームを作成している必要があります。

1. https://console.aws.amazon.com/apigateway で API Gateway コンソールにサインインします。
2. 次のいずれかを行います。
   a. 既存の API を選択し、ステージを選択します。
   b. API を作成し、ステージにデプロイします。
3. メインナビゲーションペインで、[ステージ] を選択します。

4. [ログおよびトレース] セクションで [編集] を選択します。
5. Firehose ストリームへのアクセスログ記録を有効にするには:

   a. [カスタムアクセスロギング] を有効にします。
   b. [アクセスログの送信先 ARN] に、Firehose ストリームの ARN を入力します。ARN 形式は `arn:aws:firehose:{region}:{account-id}:deliverystream/amazon-apigateway-{your-stream-name}` です。

      > **Note**
      >
      > Firehose ストリームの名前は `amazon-apigateway-{your-stream-name}` にする必要があります。

   c. [ログの形式] にログの形式を入力します。[CLF]、[JSON]、[XML]、または [CSV] を選択できます。ログ形式の例について詳しくは、「the section called “API Gateway での CloudWatch による ログの形式”」を参照してください。
6. [Save changes] (変更の保存) をクリックします。

これで、API Gateway で API へのリクエストのログを Firehose に記録する準備が整いました。ステージ設定、ログ、またはステージ変数を更新するときに API を再デプロイする必要はありません。

X-Ray を使用した REST API へのユーザーリクエストのトレース

AWS X-Ray を使用して、Amazon API Gateway の REST API から基盤となるサービスへのユーザーリクエストの流れをトレースして分析することができます。API Gateway では、API Gateway のすべての REST API エンドポイントタイプ (リージョン、エッジ最適化、プライベート) で X-Ray によるトレースがサポートされています。X-Ray は、X-Ray を利用できるすべての AWS リージョンで Amazon API Gateway と併用できます。

X-Ray では、リクエスト全体をエンドツーエンドで確認できるため、API とそのバックエンドのサービスのレイテンシーを分析することができます。X-Ray のサービスマップを使用して、リクエスト全体のレイテンシーや X-Ray と統合されているダウンストリームのサービスのレイテンシーを確認できます。また、サンプリングルールを設定することで、X-Ray で記録するリクエストとサンプリングレートを条件に応じて指定できます。

既にトレースされているサービスから API Gateway の API を呼び出すと、API Gateway はその API に対して X-Ray によるトレースが有効になっていない場合でもトレースを行います。

X-Ray は、API のステージに対して、API Gateway コンソールを使用するか、API Gateway の API または CLI を使用して有効にすることができます。

トピック

- API Gateway REST API を使用した AWS X-Ray のセットアップ
- API Gateway での AWS X-Ray サービスマップとトレースビューの使用
- API Gateway API の AWS X-Ray サンプリングルールの設定
- Amazon API Gateway API に対する AWS X-Ray トレースを理解する

API Gateway REST API を使用した AWS X-Ray のセットアップ

このセクションでは、API Gateway REST API を使用して AWS X-Ray をセットアップする方法について詳しく説明します。

トピック

- API Gateway での X-Ray のトレースモード
- X-Ray によるトレースのアクセス許可
- API Gateway コンソールでの X-Ray によるトレースの有効化
- API Gateway CLI を使用した AWS X-Ray トレースの有効化

API Gateway での X-Ray のトレースモード

アプリケーションを経由するリクエストのパスは、トラック ID を使用して追跡されます。トレースでは、1 つのリクエスト (通常は HTTP GET または POST リクエスト) で生成されたセグメントをすべて収集します。

API Gateway の API のトレースには、次の 2 つのモードがあります。

- パッシブ: API のステージに対して X-Ray によるトレースを有効にしていない場合は、これがデフォルトの設定です。この方法では、API Gateway の API は、アップストリームのサービスで X-Ray が有効になっている場合にのみトレースされます。
- アクティブ: API Gateway の API のステージに対してこの設定を行っている場合、API Gateway は X-Ray で指定されているサンプリングアルゴリズムに基づいて API の呼び出しのリクエストを自動でサンプリングします。

  ステージに対してアクティブトレースを有効にすると、サービスにリンクされたロールがアカウントにない場合は API Gateway によってロールが作成されます。このロールは

AWSServiceRoleForAPIGateway という名前で、APIGatewayServiceRolePolicy マネージドポリシーがロールにアタッチされます。サービスにリンクされたロールの詳細については、「サービスにリンクされたロールの使用」を参照してください。

> **Note**
>
> X-Ray はサンプリングアルゴリズムを適用することでトレースを効率的にしつつ、API が受け取るリクエストの代表的なサンプルを提供します。デフォルトのサンプリングアルゴリズムは 1 秒間につき 1 リクエストで、この制限を超えた場合はリクエストの 5 パーセントがサンプリングされます。

API のトレースモードは、API Gateway マネジメントコンソール、API Gateway CLI、または AWS SDK を使用して変更できます。

X-Ray によるトレースのアクセス許可

ステージに対して X-Ray によるトレースを有効にすると、サービスにリンクされたロールがアカウントにない場合は API Gateway によってロールが作成されます。このロールは AWSServiceRoleForAPIGateway という名前で、APIGatewayServiceRolePolicy マネージドポリシーがロールにアタッチされます。サービスにリンクされたロールの詳細については、「サービスにリンクされたロールの使用」を参照してください。

API Gateway コンソールでの X-Ray によるトレースの有効化

Amazon API Gateway コンソールを使用して、API のステージに対してアクティブトレースを有効にすることができます。

これらのステップでは、すでに API をステージにデプロイしていることを前提としています。

1. https://console.aws.amazon.com/apigateway で API Gateway コンソールにサインインします。
2. API を選択し、メインナビゲーションペインで [ステージ] を選択します。
3. [ステージ] ペインで、ステージを選択します。
4. [ログおよびトレース] セクションで [編集] を選択します。
5. X-Ray によるアクティブトレースを有効にするには、[X-Ray トレース] を選択して、X-Ray トレースを有効にします。
6. [Save changes] (変更の保存) をクリックします。

API のステージに対して X-Ray を有効にすると、X-Ray マネジメントコンソールを使用してトレースとサービスマップを表示できます。

API Gateway CLI を使用した AWS X-Ray トレースの有効化

AWS CLI を使用して、API ステージの作成時にステージに対して X-Ray のアクティブトレースを有効化するには、次の例のように create-stage コマンドを呼び出します。

```
aws apigateway create-stage \
    --rest-api-id {rest-api-id} \
    --stage-name {stage-name} \
    --deployment-id {deployment-id} \
    --region {region} \
    --tracing-enabled=true
```

呼び出しが成功したときの出力例を次に示します。

```
{
    "tracingEnabled": true,
    "stageName": {stage-name},
    "cacheClusterEnabled": false,
    "cacheClusterStatus": "NOT_AVAILABLE",
    "deploymentId": {deployment-id},
    "lastUpdatedDate": 1533849811,
    "createdDate": 1533849811,
    "methodSettings": {}
}
```

AWS CLI を使用して、API ステージの作成時にステージに対して X-Ray のアクティブトレースを無効化するには、次の例のように create-stage コマンドを呼び出します。

```
aws apigateway create-stage \
    --rest-api-id {rest-api-id} \
    --stage-name {stage-name} \
    --deployment-id {deployment-id} \
    --region {region} \
    --tracing-enabled=false
```

呼び出しが成功したときの出力例を次に示します。

```
{
    "tracingEnabled": false,
```

```
    "stageName": {stage-name},
    "cacheClusterEnabled": false,
    "cacheClusterStatus": "NOT_AVAILABLE",
    "deploymentId": {deployment-id},
    "lastUpdatedDate": 1533849811,
    "createdDate": 1533849811,
    "methodSettings": {}
}
```

AWS CLI を使用して、既にデプロイされている API に対して X-Ray のアクティブトレースを有効化するには、次のように update-stage コマンドを呼び出します。

```
aws apigateway update-stage \
    --rest-api-id {rest-api-id} \
    --stage-name {stage-name} \
    --patch-operations op=replace,path=/tracingEnabled,value=true
```

AWS CLI を使用して、既にデプロイされている API に対して X-Ray のアクティブトレースを無効化するには、次の例のように update-stage コマンドを呼び出します。

```
aws apigateway update-stage \
    --rest-api-id {rest-api-id} \
    --stage-name {stage-name} \
    --region {region} \
    --patch-operations op=replace,path=/tracingEnabled,value=false
```

呼び出しが成功したときの出力例を次に示します。

```
{
    "tracingEnabled": false,
    "stageName": {stage-name},
    "cacheClusterEnabled": false,
    "cacheClusterStatus": "NOT_AVAILABLE",
    "deploymentId": {deployment-id},
    "lastUpdatedDate": 1533850033,
    "createdDate": 1533849811,
    "methodSettings": {}
}
```

API のステージに対して X-Ray を有効にしたら、X-Ray CLI を使用してトレース情報を取得します。詳細については、「AWS CLI での AWS X-Ray API の使用」を参照してください。

API Gateway での AWS X-Ray サービスマップとトレースビューの使用

このセクションでは、AWS X-Ray サービスマップとトレースビューを API Gateway で使用する方法について詳しく説明します。

サービスマップとトレースビューの詳細、およびそれらの解釈方法については、AWS X-Ray コンソールを参照してください。

トピック

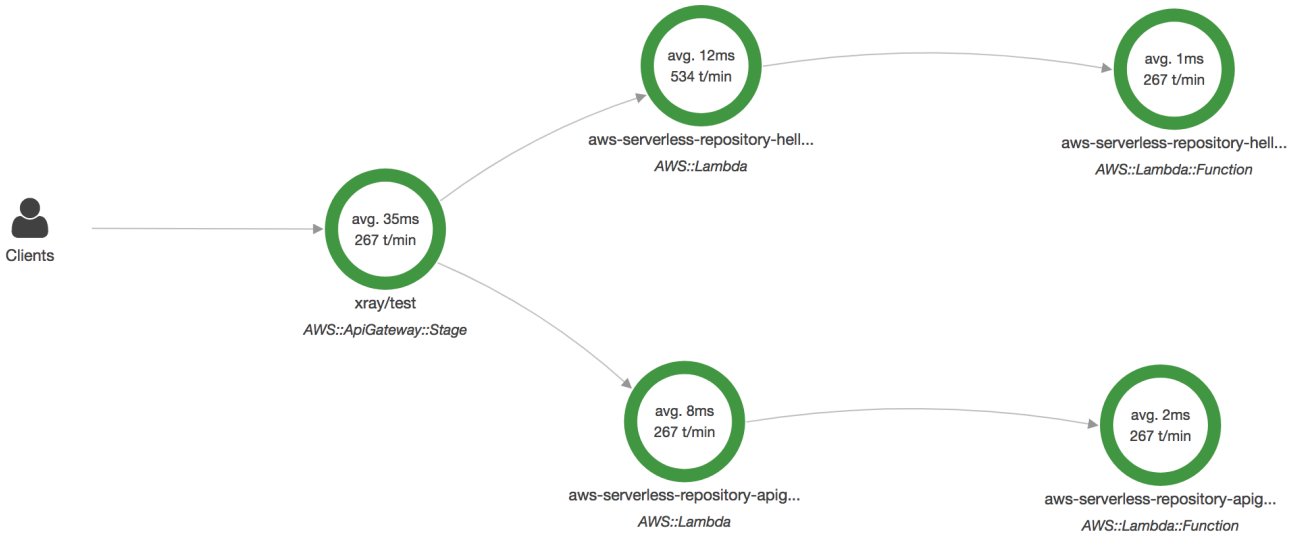
- X-Ray のサービスマップの例
- X-Ray のトレースビューの例

X-Ray のサービスマップの例

AWS X-Ray サービスマップは、API とそのすべてのダウンストリームサービスに関する情報を表示します。API Gateway で API のステージに対して X-Ray を有効にすると、サービスマップにノードが表示され、API Gateway のサービスにかかった全体の時間についての情報が示されます。レスポンスのステータスに関する詳細情報、および選択されたタイムフレームの API 応答時間のヒストグラムを取得できます。AWS Lambda や Amazon DynamoDB などの AWS のサービスと統合している API では、それらのサービスに関連するパフォーマンスメトリクスを提供するノードの数が多くなります。API ステージごとにサービスマップが作成されます。

以下の例に示しているのは、`test` という API の `xray` ステージのサービスマップです。この API は Lambda と統合されており、Lambda オーソライザー関数とバックエンドの Lambda 関数を使用しています。ノードは、API Gateway サービス、Lambda サービス、2 つの Lambda 関数を表しています。

サービスマップの構造の詳細については、「サービスマップの表示」を参照してください。



サービスマップから拡大して API ステージのトレースビューを表示できます。トレースには、API に関する詳細情報が、セグメントおよびサブセグメントとして表示されます。例えば、上記のサービスマップのトレースには、Lambda サービスと Lambda 関数のセグメントが含まれます。詳細については、AWS LambdaおよびAWS X-Rayを参照してください。

X-Ray のサービスマップでノードまたはエッジを選択すると、X-Ray コンソールにレイテンシーの分布のヒストグラムが表示されます。レイテンシーのヒストグラムを使用して、サービスでリクエストを完了するのにかかる時間を確認できます。次の図は、上記のサービスマップの `xray/test` という API Gateway のステージのヒストグラムです。レイテンシー分散ヒストグラムの詳細な説明については、「AWS X-Ray コンソールでレイテンシーのヒストグラムを使用する」を参照してください。

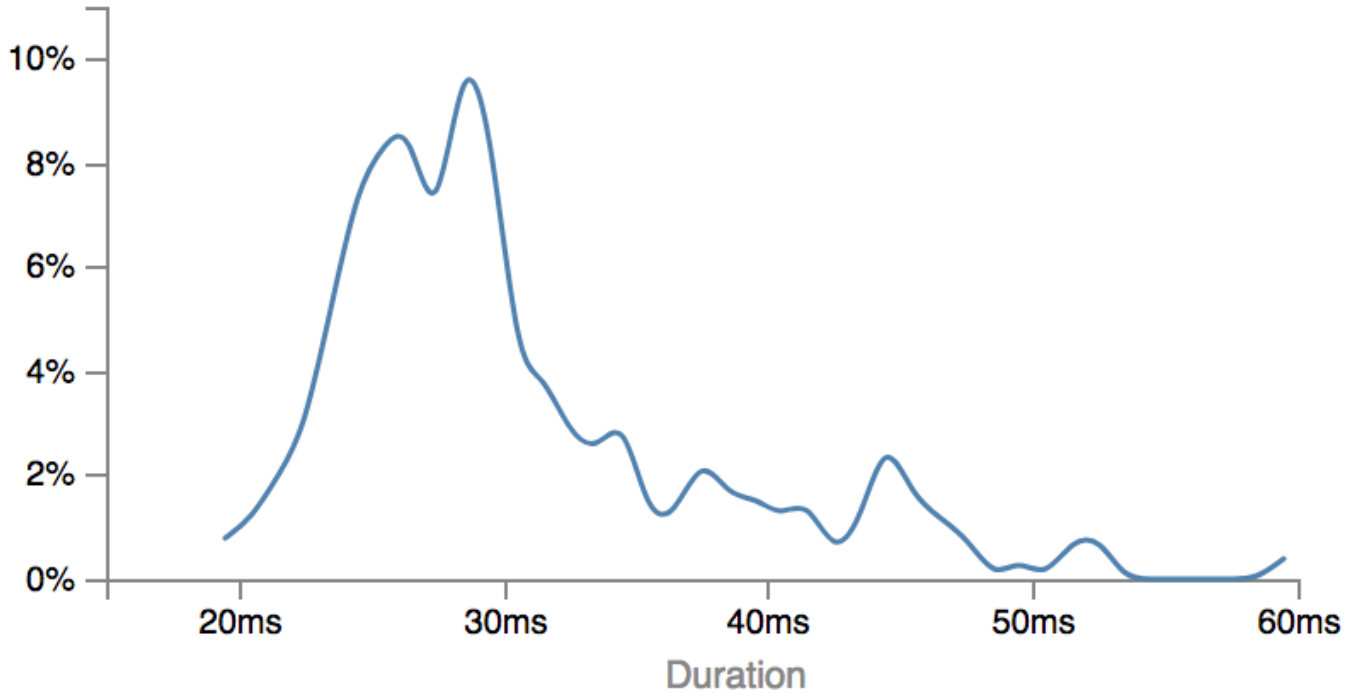
Service details

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.

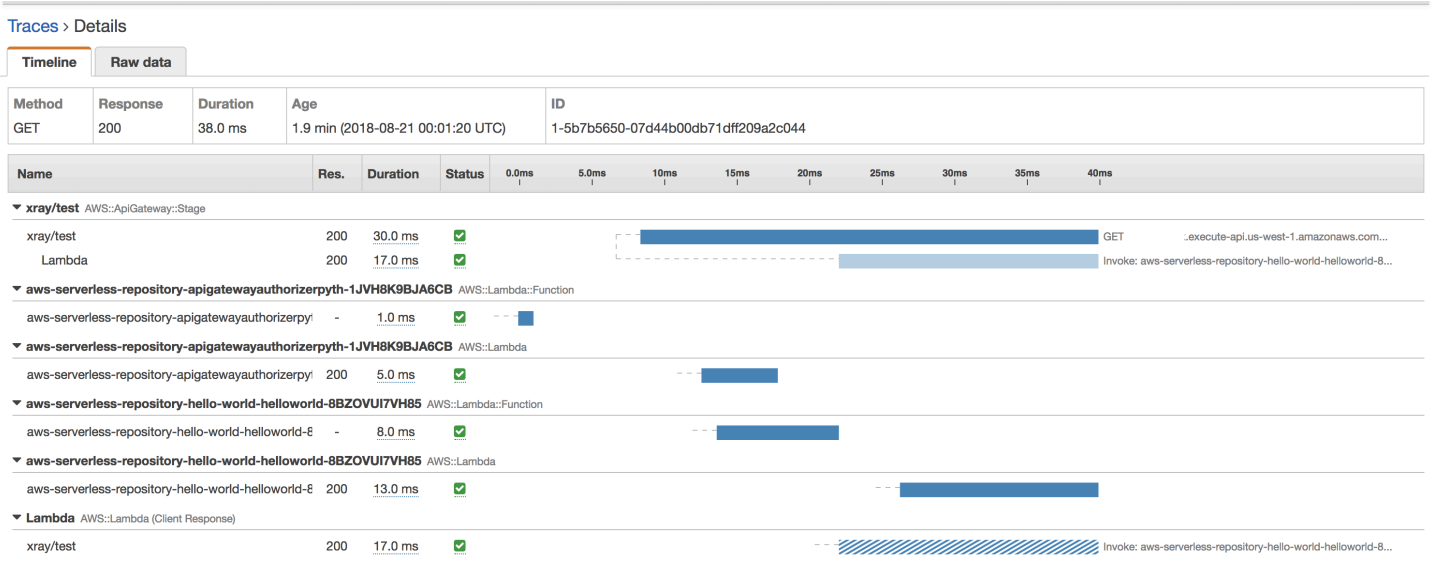
OK: 100% Error: 0%

Fault: 0% Throttle: 0%

X-Ray のトレースビューの例

次の図は、上記の API の例で生成されたトレースビューであり、バックエンドの Lambda 関数と Lambda オーソライザー関数が示されています。正常に行われた API メソッドリクエストと、レスポンスコード 200 を示しています。

トレースビューの詳細については、「トレースの表示」を参照してください。



API Gateway API の AWS X-Ray サンプリングルールの設定

AWS X-Ray コンソールまたは SDK を使用して、Amazon API Gateway API のサンプリングルールを設定できます。サンプリングルールでは、API で X-Ray が記録するリクエストを指定します。サンプリングルールをカスタマイズすることで、コードを変更または再デプロイすることなく、その場で、記録するレコードの量を制御したり、サンプリング動作を変更したりできます。

X-Ray のサンプリングルールを指定する前に、X-Ray デベロッパーガイドの以下のトピックを参照してください。

- AWS X-Ray コンソールでのサンプリングルールの設定
- X-Ray API でのサンプリングルールの使用

トピック

- API Gateway の API に関連する X-Ray のサンプリングルールのオプション値
- X-Ray のサンプリングルールの例

API Gateway の API に関連する X-Ray のサンプリングルールのオプション値

API Gateway に関連する X-Ray のサンプリングのオプションを以下に示します。文字列値では、ワイルドカードを使用して、1 つの文字列 (?) またはゼロ以上の文字 (*) に一致させることができます。[リザーバ] および [レート] 設定の使用に関する詳細な説明を含む詳細については、「AWS X-Ray コンソールでのサンプリングルールの設定」を参照してください。

- [Rule name (ルール名)] (文字列) - ルールの一意の名前。
- [Priority (優先度)] (1 ~ 9999 の整数) - サンプリングルールの優先度。サービスでは、優先度の昇順でルールが評価され、一致する最初のルールを使用してサンプリングの決定が行われます。
- [Reservoir (リザーバ)] (負ではない整数) - 固定レートを適用するまでに 1 秒あたりの一致するリクエストを計測する上限の固定数。リザーバはサービスで直接使用されませんが、ルールを一括して使用するすべてのサービスに適用されます。
- [Rate (レート)] (0 ~ 100) - リザーバの上限に達した後に一致するリクエストを計測するサンプリング率。
- [Service name (サービス名)] (文字列) - ***{api-name}/{stage-name}*** 形式の API のステージ名。たとえば、PetStore サンプル API を `test` というステージにデプロイする場合、サンプリングルールで指定する [サービス名] の値は **`pets/test`** となります。
- [Service type (サービスタイプ)] (文字列) - API Gateway の API では、**`AWS::ApiGateway::Stage`** または **`AWS::ApiGateway::*`** を指定できます。
- [Host (ホスト)] (文字列) - HTTP ヘッダーの Host のホスト名。これを **`*`** に設定すると、すべてのホスト名に対して一致します。または、一致させる完全なホスト名または部分的なホスト名を指定できます (例: **`api.example.com`** または **`*.example.com`**)。
- [Resource ARN] (リソース ARN) (文字列) - API ステージの ARN (**`arn:aws:apigateway:`*`region`*`::/restapis/`*`api-id`*`/stages/`*`stage-name`** など)。

  ステージ名は、コンソールや API Gateway の CLI または API から取得できます。ARN 形式の詳細については、「Amazon Web Services 全般のリファレンス」を参照してください。
- [HTTP method (HTTP メソッド)] (文字列) - サンプリングするメソッド (例: **`GET`**)。
- [URL path] (URL パス) (文字列) — リクエストの URL パス。
- (オプション) [Attributes (属性)] (キーと値) - 元の HTTP リクエストのヘッダー (例: **`Connection`**、**`Content-Length`**、**`Content-Type`**)。各属性の値は、最大 32 文字とすることができます。

X-Ray のサンプリングルールの例

サンプリングルールの例 1

このルールでは、GET ステージの testxray API に対するすべての test リクエストをサンプリングします。

- ルール名 - **test-sampling**
- 優先度 - **17**
- リザーバのサイズ - **10**
- 固定レート - **10**
- サービス名 - **testxray/test**
- サービスタイプ - **AWS::ApiGateway::Stage**
- HTTP メソッド - **GET**
- リソース ARN - *****
- ホスト - *****

サンプリングルールの例 2

このルールでは、testxray ステージの prod API に対するすべてのリクエストをサンプリングします。

- ルール名 - **prod-sampling**
- 優先度 - **478**
- リザーバのサイズ - **1**
- 固定レート - **60**
- サービス名 - **testxray/prod**
- サービスタイプ - **AWS::ApiGateway::Stage**
- HTTP メソッド - *****
- リソース ARN - *****
- ホスト - *****
- 属性 - **{}**

Amazon API Gateway API に対する AWS X-Ray トレースを理解する

このセクションでは、Amazon API Gateway API に対する AWS X-Ray トレースのセグメント、サブセグメント、およびその他のトレースフィールドについて説明します。

このセクションを読む前に、X-Ray デベロッパーガイドの以下のトピックを参照してください。

- AWS X-Ray コンソール
- AWS X-Ray セグメントドキュメント
- X-Ray の概念

トピック

- API Gateway の API のトレースに関連するオブジェクトの例
- トレースについて

API Gateway の API のトレースに関連するオブジェクトの例

このセクションでは、API Gateway の API のトレースに関連するいくつかのオブジェクトについて説明します。

注釈

注釈は、セグメントとサブセグメントに表示されます。これらは、トレースをフィルタリングするために、サンプリングルールでフィルタ式として使用されます。詳細については、「AWS X-Ray コンソールでのサンプリングルールの設定」を参照してください。

`annotations` オブジェクトの例を以下に示します。この例では、API のステージは、API の ID と API のステージ名によって識別されます。

```
"annotations": {
    "aws:api_id": "a1b2c3d4e5",
    "aws:api_stage": "dev"
}
```

AWS リソースデータ

`aws` オブジェクトは、セグメントでのみ使用されます。次に示す例は、デフォルトのサンプリングルールに一致する aws オブジェクトです。サンプリングルールの詳細な説明については、「AWS X-Ray コンソールのサンプリングルールの設定」を参照してください。

```
"aws": {
    "xray": {
        "sampling_rule_name": "Default"
    },
    "api_gateway": {
        "account_id": "123412341234",
        "rest_api_id": "a1b2c3d4e5",
        "stage": "dev",
        "request_id": "a1b2c3d4-a1b2-a1b2-a1b2-a1b2c3d4e5f6"
    }
}
```

トレースについて

次の例は、API Gateway のステージのトレースのセグメントを示しています。トレースセグメントを構成するフィールドの詳細な説明については、AWS X-Ray デベロッパーガイドの「AWS X-Ray セグメントドキュメント」を参照してください。

```
        {
            "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"
    }
```

HTTP API の操作

REST API と HTTP API は、いずれも RESTful API 製品です。REST API は HTTP API よりも多くの機能をサポートしていますが、HTTP API は低価格で提供できるように最小限の機能で設計されています。詳細については、「the section called “REST API と HTTP API 間で選択する ”」を参照してください。

HTTP API を使用して、AWS Lambda 関数またはルーティング可能な HTTP エンドポイントにリクエストを送信できます。たとえば、バックエンドの Lambda 関数と統合する HTTP API を作成できます。クライアントが API を呼び出すと、API Gateway はリクエストを Lambda 関数に送信し、関数のレスポンスをクライアントに返します。

HTTP API は、OpenID Connect および OAuth 2.0 認証をサポートしています。クロスオリジンリソース共有 (CORS) と自動デプロイのサポートが組み込まれています。

AWS マネジメントコンソール、AWS CLI、API、AWS CloudFormation、または SDK を使用して HTTP API を作成できます。

トピック

- API Gateway での HTTP API の開発
- 顧客が呼び出すための HTTP API の公開
- HTTP API の保護
- HTTP API のモニタリング
- HTTP API に関する問題のトラブルシューティング

API Gateway での HTTP API の開発

このセクションでは、API Gateway API の開発中に必要な API Gateway 機能について詳しく説明します。

API Gateway API の開発中、API の特性をいくつか決定することになります。これらの特性は API のユースケースによって異なります。たとえば、API を特定のクライアントのみが呼び出せるようにしたり、すべてのユーザーが利用できるようにしたりできます。API コールで Lambda 関数の実行、データベースクエリの作成やアプリケーションの呼び出しができます。

トピック

- HTTP API の作成
- HTTP API のルートの使用
- API Gateway での HTTP API へのアクセスの制御と管理
- HTTP API の統合の設定
- HTTP API の CORS の設定
- API リクエストとレスポンスの変換
- HTTP API の OpenAPI 定義の使用

HTTP API の作成

機能する API を作成するには、少なくとも 1 つのルート、統合、ステージ、およびデプロイが必要です。

次の例は、AWS Lambda または HTTP 統合、ルート、および変更を自動的にデプロイするように設定されたデフォルトステージを持つ API を作成する方法を示しています。

このガイドは、ユーザーが既に API Gateway と Lambda に関する知識を持っていることを前提としています。より詳細なガイドについては、「開始方法」を参照してください。

トピック

- を使用して HTTP API を作成するAWS Management Console
- AWS CLI を使用して HTTP API を作成する

を使用して HTTP API を作成するAWS Management Console

1. API Gateway コンソールを開きます。
2. [Create API] を選択します。
3. [HTTP API] で、[Build (ビルド)] を選択します。
4. [統合の追加] を選択し、AWS Lambda 関数を選択するか、HTTP エンドポイントを入力します。
5. [Name (名前)] に、API の名前を指定します。
6. [Review and create] を選択します。
7. [Create] を選択します。

これで、API を呼び出す準備ができました。API をテストするには、呼び出し URL をブラウザに入力するか、Curl を使用します。

```
curl https://api-id.execute-api.us-east-2.amazonaws.com
```

AWS CLI を使用して HTTP API を作成する

クイック作成を使用して、Lambda または HTTP 統合、デフォルトのキャッチオールルート、変更を自動的にデプロイするように設定されたデフォルトステージを持つ API を作成できます。次のコマンドは、クイック作成を使用して、バックエンドで Lambda 関数と統合する API を作成します。

> **Note**
>
> Lambda 統合を呼び出すには、必要なアクセス許可が API Gateway に必要です。リソースベースのポリシーまたは IAM ロールを使用して、Lambda 関数を呼び出す API Gateway のアクセス許可を付与できます。詳細については、AWS Lambda デベロッパーガイドの「AWS Lambda のアクセス許可」を参照してください。

Example

```
aws apigatewayv2 create-api --name my-api --protocol-type HTTP --target
 arn:aws:lambda:us-east-2:123456789012:function:function-name
```

これで、API を呼び出す準備ができました。API をテストするには、呼び出し URL をブラウザに入力するか、Curl を使用します。

```
curl https://api-id.execute-api.us-east-2.amazonaws.com
```

HTTP API のルートの使用

ルートは、直接の受信 API リクエストをバックエンドリソースにルーティングします。ルートは、HTTP メソッドとリソースパスという 2 つの部分で構成されます (例: `GET /pets`)。ルートに特定の HTTP メソッドを定義できます。または、`ANY` メソッドを使用して、リソースに対して定義していないすべてのメソッドを一致させることができます。他のどのルートとも一致しないリクエストのキャッチオールとして機能する `$default` ルートを作成できます。

> **Note**
>
> API Gateway は、URL エンコードされたリクエストパラメータをデコードしてからバックエンド統合に渡します。

パス変数の操作

HTTP API ルートでパス変数を使用できます。

たとえば、`GET /pets/{petID}` ルートは、クライアントが GET に送信する `https://api-id.execute-api.us-east-2.amazonaws.com/pets/6` リクエストをキャッチします。

greedy パス変数は、ルートのすべての子リソースをキャッチします。greedy パス変数を作成するには、変数名に + を追加します (例: `{proxy+}`)。greedy パス変数は、リソースパスの末尾にある必要があります。

クエリ文字列パラメータの操作

デフォルトでは、API Gateway は HTTP API へのリクエストにクエリ文字列パラメータが含まれている場合、バックエンド統合にクエリ文字列パラメータを送信します。

たとえば、クライアントが `https://api-id.execute-api.us-east-2.amazonaws.com/pets?id=4&type=dog` にリクエストを送信すると、クエリ文字列パラメータ `?id=4&type=dog` が統合に送信されます。

`$default` ルートの操作

`$default` ルートは、API 内の他のルートと明示的に一致しないリクエストをキャッチします。

`$default` ルートがリクエストを受信すると、API Gateway は完全なリクエストパスを統合に送信します。たとえば、`$default` ルートのみを持つ API を作成し、ANY メソッドで `https://petstore-demo-endpoint.execute-api.com` HTTP エンドポイントと統合できます。`https://api-id.execute-api.us-east-2.amazonaws.com/store/checkout` にリクエストを送信すると、API Gateway は `https://petstore-demo-endpoint.execute-api.com/store/checkout` にリクエストを送信します。

HTTP 統合の詳細については、「HTTP API の HTTP プロキシ統合の使用」を参照してください。

API リクエストのルーティング

クライアントが API リクエストを送信すると、API Gateway はまずリクエストをルーティングするステージを決定します。リクエストがステージと明示的に一致する場合、API Gateway はそのステージにリクエストを送信します。リクエストに完全に一致するステージがない場合、API Gateway はリクエストを `$default` ステージに送信します。`$default` ステージがない場合、API は `{"message":"Not Found"}` を返し、CloudWatch ログを生成しません。

ステージを選択した後、API Gateway はルートを選択します。API Gateway は、次の優先順位を使用して、最も具体的な一致を持つルートを選択します。

1. ルートとメソッドの完全一致。
2. greedy パス変数 (`{proxy+}`)を持つルートとメソッドを一致させます。
3. `$default` ルート。

リクエストに一致するルートがない場合、API Gateway は `{"message":"Not Found"}` をクライアントに返します。

たとえば、`$default` ステージがある API と、次のルート例を考えてみます。

1. `GET /pets/dog/1`
2. `GET /pets/dog/{id}`
3. `GET /pets/{proxy+}`
4. `ANY /{proxy+}`
5. `$default`

次の表は、API Gateway がリクエストをルート例にルーティングする方法をまとめたものです。

リクエスト	選択されたルート	説明
`GET https://api-id.execute-api. region.amazonaws.com/pets/dog/1`	`GET /pets/dog/1`	リクエストはこの静的ルートに完全に一致します。
`GET https://api-id.execute-`	`GET /pets/dog/{id}`	リクエストはこのルートに完全に一致します。

リクエスト	選択されたルート	説明
api. *region*.amazonaws.com/pets/dog/2		
GET https://*api-id*.execute-api. *region*.amazonaws.com/pets/cat/1	GET /pets/{proxy+}	リクエストがルートに完全に一致していません。GET メソッドと greedy パス変数を持つルートは、このリクエストをキャッチします。
POST https://*api-id*.execute-api. *region*.amazonaws.com/test/5	ANY /{proxy+}	ANY メソッドは、ルートに対して定義していないすべてのメソッドに一致します。greedy パス変数を持つルートの優先順位は、$default ルートよりも高くなります。

API Gateway での HTTP API へのアクセスの制御と管理

API Gateway は HTTP API へのアクセスを制御し管理する複数のメカニズムをサポートしています。

- Lambda オーソライザーは、Lambda 関数を使用して API へのアクセスを制御します。詳細については、「HTTP API の AWS Lambda オーソライザーの使用」を参照してください。
- JWT オーソライザーは、JSON ウェブトークンを使用して API へのアクセスをコントロールします。詳細については、「JWT オーソライザーを使用した HTTP API へのアクセスの制御」を参照してください。
- 標準の AWS IAM ロールとポリシーは、柔軟で堅牢なアクセス制御を提供します。IAM ロールとポリシーを使用して、どのユーザーが API を呼び出すことができるだけでなく、作成および管理することもできるかをコントロールできます。詳細については、「IAM 認証の使用」を参照してください。

HTTP API の AWS Lambda オーソライザーの使用

Lambda オーソライザーを使用し、Lambda 関数を使用して HTTP API へのアクセスを制御します。次に、クライアントが API を呼び出すと、API Gateway が Lambda 関数を呼び出します。API

Gateway は、Lambda 関数からのレスポンスを使用して、クライアントが API にアクセスできるかどうかを判断します。

ペイロード形式バージョン

オーソライザーの'ペイロード形式バージョンでは、API Gateway が Lambda 認証に送信するデータの形式と、API Gateway が Lambda からのレスポンスをどのように解釈するかを指定します。ペイロード形式バージョンを指定しない場合、AWS Management Console はデフォルトで最新バージョンを使用します。AWS CLI、AWS CloudFormation、または SDK を使用して Lambda オーソライザーを作成する場合は、`authorizerPayloadFormatVersion` を指定する必要があります。サポートされる値は `1.0` と `2.0` です。

REST APIとの互換性が必要な場合は、バージョン `1.0` を使用してください。

次の例は、各ペイロード形式バージョンの構造を示しています。

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 オーソライザーのレスポンス形式

ペイロード形式バージョンによって、Lambda 関数から返す必要のあるレスポンスの構造も決まります。

Lambda 関数レスポンス形式 1.0

`1.0` 形式バージョンを選択した場合、Lambda オーソライザーは API ルートへのアクセスを許可または拒否する IAM ポリシーを返す必要があります。ポリシーでは、標準の IAM ポリシー構文を使用できます。IAM ポリシーの例については、「the section called “ API を呼び出すためのアクセスの制御”」を参照してください。`$context.authorizer.property` を使用して、コンテキストプロパティを Lambda 統合に渡したり、ログにアクセスしたりできます。`context` オブジェクトはオプションです。`claims` は予約されたプレースホルダーであり、コンテキストオブジェクトとしては使用できません。詳細については、「the section called “ログ記録変数”」を参照してください。

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"
  }
}
```

Lambda 関数レスポンス形式 2.0

`2.0` 形式バージョンを選択した場合は、Lambda 関数から、ブール値を返すか、標準の IAM ポリシー構文を使用する IAM ポリシーを返すことができます。ブール値を返すには、オーソライザーの簡易レスポンスを有効にします。以下の例では、Lambda 関数から返すように

コーディングする必要のある形式を示しています。context オブジェクトはオプションです。$context.authorizer.*property* を使用して、コンテキストプロパティを Lambda 統合に渡したり、ログにアクセスしたりできます。詳細については、「the section called “ログ記録変数”」を参照してください。

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"
  }
}
```

Lambda オーソライザー関数の例

以下のサンプルの Node.js Lambda 関数では、2.0 ペイロード形式バージョンの場合に Lambda 関数から返す必要のあるレスポンスの形式を示しています。

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
```

ID ソース

オプションで、Lambda オーソライザーの ID ソースを指定できます。ID ソースは、リクエストを認可するために必要なデータの場所を指定します。例えば、ヘッダーまたはクエリ文字列の値を ID ソースとして指定できます。ID ソースを指定する場合、クライアントはそれらのソースをリクエストに含める必要があります。クライアントのリクエストに ID ソースが含まれていない場合、API Gateway は Lambda オーソライザーを呼び出さず、クライアントは 401 エラーを受け取ります。以下の ID ソースがサポートされています。

選択式

タイプ	例	コメント
ヘッダー値	$request.header.*name*	ヘッダー名では大文字と小文字は区別されません。
クエリ文字列値	$request.querystring.*name*	クエリ文字列名では大文字と小文字が区別されます。
コンテキスト変数	$context.*variableName*	サポートされているコンテキスト変数の値。
ステージ変数	$stageVariables.*variableName*	ステージ変数の値。

オーソライザーのレスポンスのキャッシュ

authorizerResultTtlInSeconds を指定することで、Lambda オーソライザーのキャッシュを有効にすることができます。オーソライザーのキャッシュが有効になっていると、API Gateway はオーソライザーの ID ソースをキャッシュキーとして使用します。クライアントが設定済みの TTL 内の ID ソースで同じパラメータを指定した場合、API Gateway は、Lambda 関数を呼び出すのではなく、キャッシュされたオーソライザーの結果を使用します。

キャッシュを有効にするには、オーソライザーに 1 つ以上の ID ソースが必要です。

オーソライザーの簡易レスポンスを有効にすると、オーソライザーのレスポンスは、キャッシュされた ID ソース値に一致するすべての API リクエストを完全に許可するか、拒否するかになります。より詳細なアクセス許可が必要な場合は、簡易レスポンスを無効にし、IAM ポリシーが返されるようにします。

デフォルトでは、API Gateway は、オーソライザーを使用する API のすべてのルートに対して、キャッシュされたオーソライザーのレスポンスを使用します。ルートごとにレスポンスをキャッシュするには、オーソライザーの ID ソースに `$context.routeKey` を追加します。

Lambda オーソライザーの作成

Lambda オーソライザーを作成するときは、API Gateway で使用する Lambda 関数を指定します。関数のリソースポリシーまたは IAM ロールを使用して Lambda 関数を呼び出すには、API Gateway のアクセス許可を付与する必要があります。この例では、Lambda 関数を呼び出すアクセス許可を API Gateway に付与するように関数のリソースポリシーを更新します。

```
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
```

次のコマンドは、Lambda 関数を呼び出すためのアクセス許可を API Gateway に付与します。API Gateway に関数を呼び出すアクセス許可がない場合、クライアントは `500 Internal Server Error` を受け取ります。

```
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"
```

オーソライザーを作成し、呼び出すアクセス許可を API Gateway に付与したら、オーソライザーを使用するようにルートを更新します。

```
aws apigatewayv2 update-route \
    --api-id abcdef123 \
    --route-id acd123 \
    --authorization-type CUSTOM \
    --authorizer-id def123
```

Lambda オーソライザーのトラブルシューティング

API Gateway が Lambda オーソライザーを呼び出せない場合、または Lambda オーソライザーが無効な形式のレスポンスを返す場合、クライアントは `500 Internal Server Error` を受け取ります。

エラーのトラブルシューティングを行うには、API ステージのアクセスログを有効にします。ログ形式に `$context.authorizer.error` ログ記録変数を含めます。

API Gateway に関数を呼び出すアクセス許可がないことがログに示されている場合は、関数のリソースポリシーを更新するか、IAM ロールを提供して API Gateway にオーソライザーを呼び出すアクセス許可を付与します。

Lambda 関数が無効なレスポンスを返すことがログに示されている場合は、Lambda 関数が必要な形式でレスポンスを返すことを確認してください。

JWT オーソライザーを使用した HTTP API へのアクセスの制御

OpenID Connect (OIDC) および OAuth 2.0 フレームワークの一部として JSON ウェブトークン (JWT) を使用して、API へのクライアントアクセスを制限できます。

API のルートに JWT オーソライザーを設定する場合、API Gateway はクライアントが API リクエストとともに送信する JWT を検証します。API Gateway は、トークンの検証、およびオプションでトークン内のスコープに基づいてリクエストを許可または拒否します。ルートのスコープを設定する場合、ルートのスコープを 1 つ以上、トークンに含める必要があります。

API の各ルートに個別のオーソライザーを設定することも、複数のルートに同じオーソライザーを使用することもできます。

> **Note**
>
> JWT アクセストークンを OpenID Connect ID トークンなど他のタイプの JWT と区別する標準的なメカニズムはありません。API 認証に ID トークンが必要でない限り、認可スコープを要求するようにルートを設定することをお勧めします。また、JWT アクセストークンを発行

するときにのみ ID プロバイダーが使用する発行者または対象者を要求するように JWT オーソライザーを設定することもできます。

JWT オーソライザーによる API リクエストの承認

API Gateway は、次の一般的なワークフローを使用して、JWT オーソライザーを使用するように設定されたルートへの要求を承認します。

1. identitySource でトークンを確認します。identitySource には、トークンのみを含めるか、Bearer のプレフィックスが付いたトークンのみを含めることができます。
2. トークンをデコードします。
3. 発行者の jwks_uri から取得したパブリックキーを使用して、トークンのアルゴリズムと署名を確認します。現在、RSA ベースのアルゴリズムのみがサポートされています。API Gateway は、パブリックキーを 2 時間キャッシュできます。ベストプラクティスとして、キーをローテーションするときは、古いキーと新しいキーの両方が有効な猶予期間を設けてください。
4. クレームを検証します。API Gateway は、次のトークンクレームを評価します。
   - kid – トークンには、トークンに署名した jwks_uri のキーと一致するヘッダークレームが必要です。
   - iss – オーソライザーに設定された issuer と一致する必要があります。
   - aud または client_id – オーソライザーに設定された audience エントリの 1 つと一致する必要があります。API Gateway は、aud が存在しない場合にのみ、client_id を検証します。aud と client_id の両方が存在する場合、API Gateway は aud を評価します。
   - exp – UTC の現在時刻より後にする必要があります。
   - nbf – UTC の現在時刻より前にする必要があります。
   - iat – UTC の現在時刻より前にする必要があります。
   - scope または scp – トークンには、ルートの authorizationScopes のスコープを少なくとも 1 つ含める必要があります。

これらのいずれかの手順が失敗した場合、API Gateway は API リクエストを拒否します。

API Gateway は JWT を検証した後、トークン内のクレームを API ルートの統合に渡します。JWT クレームには、Lambda 関数などのバックエンドリソースがアクセスできます。例えば、JWT に ID クレーム emailID が含まれている場合、$event.requestContext.authorizer.jwt.claims.emailID で Lambda 統合に使用で

きます。API Gateway が Lambda 統合に送信するペイロードの詳細については、「the section called "AWS Lambda の統合"」を参照してください。

JWT オーソライザーを作成する

JWT オーソライザーを作成する前に、クライアントアプリケーションを ID プロバイダーに登録する必要があります。また、HTTP API を作成しておく必要があります。HTTP API の作成例については、「HTTP API の作成」を参照してください。

コンソールを使用して JWT オーソライザーを作成する

次の手順は、コンソールを使用して JWT オーソライザーを作成する方法を示しています。

コンソールを使用して JWT オーソライザーを作成するには

1. https://console.aws.amazon.com/apigateway で API Gateway コンソールにサインインします。
2. HTTP API を選択します。
3. メインナビゲーションペインで、[認可] を選択します。
4. [オーソライザーを管理] タブを選択します。
5. [Create] (作成) を選択します。
6. [オーソライザーのタイプ] で、[JWT] を選択します。
7. JWT オーソライザーを設定し、トークンのソースを定義する ID ソースを指定します。
8. [Create] (作成) を選択します。

AWS CLI を使用して JWT オーソライザーを作成する

次の AWS CLI コマンドは、JWT オーソライザーを作成します。`jwt-configuration` には、ID プロバイダーの `Audience` と `Issuer` を指定します。Amazon Cognito を ID プロバイダーとして使用する場合、`IssuerUrl` は `https://cognito-idp.us-east-2.amazonaws.com/userPoolID` です。

```
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
```

AWS CloudFormation を使用して JWT オーソライザーを作成する

次の AWS CloudFormation テンプレートでは、Amazon Cognito を ID プロバイダーとして使用する JWT オーソライザーで HTTP API を作成します。

AWS CloudFormation テンプレートの出力は、クライアントがサインアップ/サインインして JWT を受け取ることができる、Amazon Cognito がホストする UI の URL です。クライアントは、サインインすると、URL のアクセストークンにより、HTTP API にリダイレクトされます。アクセストークンを使用して API を呼び出すには、URL の # を ? に変更し、トークンをクエリ文字列パラメータとして使用します。

AWS CloudFormation テンプレートの例

```
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
```

JWT オーソライザーを使用するようにルートを更新する

JWT オーソライザーを使用するようにルートを更新するには、コンソール、AWS CLI、または AWS SDK を使用できます。

コンソールを使用して、JWT オーソライザーを使用するようにルートを更新する

次の手順では、コンソールを使用して、JWT オーソライザーを使用するようにルートを更新する方法を示します。

コンソールを使用して JWT オーソライザーを作成するには

1. https://console.aws.amazon.com/apigateway で API Gateway コンソールにサインインします。
2. HTTP API を選択します。
3. メインナビゲーションペインで、[認可] を選択します。
4. メソッドを選択し、ドロップダウンメニューからオーソライザーを選択して、[オーソライザーをアタッチ] を選択します。

AWS CLI を使用して、JWT オーソライザーを使用するようにルートを更新する

次のコマンドでは、AWS CLI を使用して、JWT オーソライザーを使用するようにルートを更新します。

```
aws apigatewayv2 update-route \
   --api-id api-id  \
   --route-id route-id  \
   --authorization-type JWT \
   --authorizer-id authorizer-id \
   --authorization-scopes user.email
```

IAM 認証の使用

HTTP API ルートに対して IAM 認証を有効にできます。IAM 認証が有効な場合、クライアントは Signature Version 4 (SigV4) を使用して、AWS 認証情報でリクエストに署名する必要がありま

す。API Gateway は、クライアントがルートに対する execute-api アクセス許可を持っている場合にのみ、API ルートを呼び出します。

HTTP API に対する IAM 認証は、REST API に対する認証と同様です。

> **Note**
>
> リソースポリシーは、現在 HTTP API 向けにサポートされていません。

API を呼び出すアクセス許可をクライアントに付与する IAM ポリシーの例については、「the section called “ API を呼び出すためのアクセスの制御”」を参照してください。

ルートに対する IAM 認証を有効にする

次の AWS CLI コマンドは、HTTP API ルートの IAM 認証を有効にします。

```
aws apigatewayv2 update-route \
    --api-id abc123 \
    --route-id abcdef \
    --authorization-type AWS_IAM
```

HTTP API の統合の設定

統合は、ルートをバックエンドリソースに接続します。HTTP API は、Lambda プロキシ、AWS のサービス、および HTTP プロキシの統合をサポートします。たとえば、API の POST ルートへの /signup リクエストを設定して、お客様のサインアップを処理する Lambda 関数と統合できます。

トピック

- HTTP API の AWS Lambda プロキシ統合の使用
- HTTP API の HTTP プロキシ統合の使用
- HTTP API の AWS のサービス統合の使用
- HTTP API のプライベート統合の使用

HTTP API の AWS Lambda プロキシ統合の使用

Lambda プロキシ統合を使用すると、API ルートを Lambda 関数と統合できます。クライアントが API を呼び出すと、API Gateway はリクエストを Lambda 関数に送信し、関数のレスポンスをクライアントに返します。HTTP API の作成例については、「HTTP API の作成」を参照してください。

ペイロード形式バージョン

ペイロード形式バージョンは、API Gateway が Lambda 統合に送信するイベントの形式と、API Gateway が Lambda からのレスポンスをどのように解釈するかを指定します。ペイロード形式バージョンを指定しない場合、AWS Management Console はデフォルトで最新バージョンを使用します。AWS CLI、AWS CloudFormation、SDK を使用して Lambda 統合を作成する場合は、`payloadFormatVersion` を指定する必要があります。サポートされる値は `1.0` と `2.0` です。

`payloadFormatVersion` を設定する方法の詳細については、「create-integration」を参照してください。既存の統合の `payloadFormatVersion` を確認する方法の詳細については、「get-integration」を参照してください。

ペイロード形式の相違点

次のリストは、ペイロード形式の `1.0` バージョンと `2.0` バージョンの違いを示しています。

- Format `2.0` には `multiValueHeaders` または `multiValueQueryStringParameters` フィールドがありません。重複するヘッダーはコンマで結合され、`headers` フィールドに含められます。重複するクエリ文字列はコンマで結合され、`queryStringParameters` フィールドに含められます。
- 形式 `2.0` には `rawPath` があります。API マッピングを使用してステージをカスタムドメイン名に接続する場合、`rawPath` は API マッピング値を提供しません。カスタムドメイン名の API マッピングにアクセスするには、形式 `1.0` と `path` を使用します。
- 形式 `2.0` には、新しい `cookies` フィールドが含まれています。リクエスト内のすべてのクッキーヘッダーはコンマで結合され、`cookies` フィールドに追加されます。クライアントへのレスポンスでは、各クッキーは `set-cookie` ヘッダーになります。

ペイロード形式の構造

次の例は、各ペイロード形式バージョンの構造を示しています。すべてのヘッダー名は小文字です。

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 関数レスポンス形式

ペイロード形式バージョンによって、Lambda 関数が返す必要があるレスポンスの構造が決まります。

Lambda 関数レスポンス形式 1.0

1.0 形式バージョンの場合、Lambda 統合は次の JSON 形式でレスポンスを返す必要があります。

Example

```
{
    "isBase64Encoded": true|false,
    "statusCode": httpStatusCode,
    "headers": { "headername": "headervalue", ... },
    "multiValueHeaders": { "headername": ["headervalue", "headervalue2", ...], ... },
    "body": "..."
}
```

Lambda 関数レスポンス形式 2.0

2.0 形式バージョンでは、API Gateway がレスポンス形式を推測できます。API Gateway は、Lambda 関数が有効な JSON を返し、statusCode を返さない場合、次の仮定を行います。

- isBase64Encoded は false。
- statusCode は 200。

- content-type は application/json。
- body は関数のレスポンスです。

次の例は、Lambda 関数と API Gateway の解釈の出力を示しています。

Lambda 関数出力	API Gateway 解釈
`"Hello from Lambda!"`	`{ "isBase64Encoded": false, "statusCode": 200, "body": "Hello from Lambda!", "headers": { "content-type": "application/json" } }`
`{ "message": "Hello from Lambda!" }`	`{ "isBase64Encoded": false, "statusCode": 200, "body": "{ \"message\": \"Hello from Lambda!\" }", "headers": { "content-type": "application/json" } }`

レスポンスをカスタマイズするには、Lambda 関数は以下の形式でレスポンスを返す必要があります。

```
{
    "cookies" : ["cookie1", "cookie2"],
    "isBase64Encoded": true|false,
    "statusCode": httpStatusCode,
    "headers": { "headername": "headervalue", ... },
    "body": "Hello from Lambda!"
}
```

HTTP API の HTTP プロキシ統合の使用

HTTP プロキシ統合を使用すると、API ルートをパブリックにルーティング可能な HTTP エンドポイントに接続できます。この統合タイプでは、API Gateway はフロントエンドとバックエンド間でリクエストとレスポンスの全体を渡します。

HTTP プロキシ統合を作成するには、パブリックにルーティング可能な HTTP エンドポイントの URL を指定します。

パス変数と HTTP プロキシの統合

HTTP API ルートでパス変数を使用できます。

たとえば、ルート `/pets/{petID}` は `/pets/6` へのリクエストをキャッチします。統合 URI でパス変数を参照して、変数の内容を統合に送信できます。例: 「`/pets/extendedpath/{petID}`」。

greedy パス変数を使用して、ルートのすべての子リソースをキャッチできます。greedy パス変数を作成するには、変数名に + を追加します (例: `{proxy+}`)。

すべてのリクエストをキャッチする HTTP プロキシ統合でルートを設定するには、greedy パス変数 (`/parent/{proxy+}` など) を使用して API ルートを作成します。`https://petstore-demo-endpoint.execute-api.com/petstore/{proxy}` メソッドでルートを HTTP エンドポイント (ANY など) と統合します。greedy パス変数は、リソースパスの末尾にある必要があります。

HTTP API の AWS のサービス統合の使用

HTTP API と AWS のサービスは、ファーストクラス統合を使用して統合できます。ファーストクラス統合は、HTTP API ルートを AWS のサービス API に接続します。クライアントがファーストクラス統合でサポートされるルートを呼び出すと、API Gateway は AWS のサービス API を呼び出します。例えば、ファーストクラス統合を使用して、Amazon Simple Queue Service キューにメッセージを送信したり、AWS Step Functions ステートマシンを起動したりすることができます。サポートされているサービスアクションについては、「the section called “AWS のサービス統合のリファレンス”」を参照してください。

リクエストパラメータのマッピング

ファーストクラス統合には、必須およびオプションのパラメータがあります。統合を作成するには、すべての必須のパラメータを設定する必要があります。静的な値を使用するか、実行時に動的に評価

されるパラメータをマッピングできます。サポートされている統合とパラメータの完全なリストについては、「the section called “AWS のサービス統合のリファレンス”」を参照してください。

パラメータのマッピング

タイプ	例	コメント
ヘッダー値	$request.header.*name*	ヘッダー名では大文字と小文字は区別されません。API Gateway は、複数のヘッダー値をコンマで結合します (例: `"header1": "value1,value2"`)。
クエリ文字列値	$request.querystring.*name*	クエリ文字列名では大文字と小文字が区別されます。API Gateway は複数の値をコンマで結合します (例: `"querystring1": "Value1,Value2"`)。
パスパラメータ	$request.path.*name*	リクエストのパスパラメータの値。例えば、ルートが `/pets/{petId}` の場合は、*`$request.path.petId`* を使用してリクエストの `petId` パラメータをマッピングできます。
リクエストボディのパススルー	$request.body	API Gateway はリクエストボディ全体をパススルーします。
リクエストボディ	$request.body.*name*	JSON パス式。再帰下降 (`$request.body..` *`name`*) およびフィルター式 (`?(`*`expression`*` )`) はサポートされていません。

タイプ	例	コメント
		Note JSON パスを指定すると、API Gateway はリクエスト本文を 100 KB で切り捨て、選択式を適用します。100 KB を超えるペイロードを送信するには、`$request.body` を指定します。
コンテキスト変数	$context.*variableName*	サポートされているコンテキスト変数の値。
ステージ変数	$stageVariables.*variableName*	ステージ変数の値。
静的な値	*string*	定数値。

ファーストクラス統合を作成する

ファーストクラス統合を作成する前に、統合する AWS のサービスアクションを呼び出す API Gateway アクセス許可を付与する IAM ロールを作成する必要があります。詳細については、「AWS のサービスのロールの作成」を参照してください。

ファーストクラス統合を作成するには、`SQS-SendMessage` などのサポートされている AWS のサービスアクションを選択し、リクエストパラメータを設定して、統合された AWS のサービス API を呼び出すアクセス許可を API Gateway に付与するロールを提供します。統合サブタイプに応じて、異なるリクエストパラメータが必要です。詳細については、「the section called “AWS のサービス統合のリファレンス”」を参照してください。

次の AWS CLI コマンドは、Amazon SQS メッセージを送信する統合を作成します。

```
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"}'
```

AWS CloudFormation を使用してファーストクラス統合を作成する

次の例は、Amazon EventBridge とのファーストクラス統合を使用して /{source}/{detailType} ルートを作成する AWS CloudFormation スニペットを示しています。

Source パラメータは {source} パスパラメータ、DetailType は {DetailType} パスパラメータ、Detail パラメータはリクエスト本文にマップされます。

このスニペットは、PutEvents アクションを呼び出すための API Gateway アクセス許可を付与するイベントバスや IAM ロールを表示しません。

```
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"
```

統合サブタイプのリファレンス

以下の統合サブタイプは HTTP API でサポートされています。

統合サブタイプ

- EventBridge-PutEvents
- SQS-SendMessage
- SQS-ReceiveMessage
- SQS-DeleteMessage
- SQS-PurgeQueue
- AppConfig-GetConfiguration
- Kinesis-PutRecord
- StepFunctions-StartExecution
- StepFunctions-StartsyncExecution
- StepFunctions-StopExecution

EventBridge-PutEvents

カスタムイベントを、ルールとマッチングできるように Amazon EventBridge に送信します。

EventBridge-PutEvents 1.0

パラメータ	必須
詳細	True
DetailType	True
送信元	True
時間	False
EventBusName	False
リソース	False
リージョン	False

パラメータ	必須
TraceHeader	False

詳細については、 Amazon EventBridge API リファレンスの「PutEvents」を参照してください。

SQS-SendMessage

指定されたキューにメッセージを配信します。

SQS-SendMessage 1.0

パラメータ	必須
QueueUrl	True
MessageBody	True
DelaySeconds	False
MessageAttributes	False
MessageDeduplicationId	False
MessageGroupId	False
MessageSystemAttributes	False
リージョン	False

詳細については、Amazon Simple Queue Service API リファレンスの「SendMessage」を参照してください。

SQS-ReceiveMessage

指定されたキューから 1 個以上 (最大 10 個) のメッセージを取得します。

SQS-ReceiveMessage 1.0

パラメータ	必須
QueueUrl	True
AttributeNames	False
MaxNumberOfMessages	False
MessageAttributeNames	False
ReceiveRequestAttemptId	False
VisibilityTimeout	False
WaitTimeSeconds	False
リージョン	False

詳細については、Amazon Simple Queue Service API リファレンスの「ReceiveMessage」を参照してください。

SQS-DeleteMessage

指定されたキュー内の指定されたメッセージを削除します。

SQS-DeleteMessage 1.0

パラメータ	必須
ReceiptHandle	True
QueueUrl	True
リージョン	False

詳細については、Amazon Simple Queue Service API リファレンスの「DeleteMessage」を参照してください。

SQS-PurgeQueue

指定されたキュー内のすべてのメッセージを削除します。

SQS-PurgeQueue 1.0

パラメータ	必須
QueueUrl	True
リージョン	False

詳細については、Amazon Simple Queue Service API リファレンスの 「PurgeQueue」を参照してください。

AppConfig-GetConfiguration

設定に関する情報を受け取ります。

AppConfig-GetConfiguration 1.0

パラメータ	必須
アプリケーション	True
環境	True
設定	True
ClientId	True
ClientConfigurationVersion	False
リージョン	False

詳細については、AWS AppConfig API リファレンスの「GetConfiguration」を参照してください。

Kinesis-PutRecord

1 つのデータレコードを Amazon Kinesis Data Streams に書き込みます。

Kinesis-PutRecord 1.0

パラメータ	必須
StreamName	True
データ	True
PartitionKey	True
SequenceNumberForOrdering	False
ExplicitHashKey	False
リージョン	False

詳細については、Amazon Kinesis Data Streams API リファレンスの「PutRecord」を参照してください。

StepFunctions-StartExecution

ステートマシンの実行を開始します。

StepFunctions-StartExecution 1.0

パラメータ	必須
StateMachineArn	True
名前	False
Input	False
リージョン	False

詳細については、AWS Step Functions API リファレンスの「StartExecution」を参照してください。

StepFunctions-StartsyncExecution

同期状態マシンの実行を開始します。

StepFunctions-StartsyncExecution 1.0

パラメータ	必須
StateMachineArn	True
名前	False
Input	False
リージョン	False
TraceHeader	False

詳細については、AWS Step Functions API リファレンスの「StartSyncExecution」を参照してください。

StepFunctions-StopExecution

実行を停止します。

StepFunctions-StopExecution 1.0

パラメータ	必須
ExecutionArn	True
原因	False
エラー	False
リージョン	False

詳細については、AWS Step Functions API リファレンスの「StopExecution」を参照してください。

HTTP API のプライベート統合の使用

プライベート統合を使用すると、Application Load Balancer や Amazon ECS コンテナベースのアプリケーションなど、VPC 内のプライベートリソースと API 統合を作成できます。

プライベート統合を使用して、VPC 外のクライアントがアクセスできるように VPC 内のリソースを公開できます。API Gateway がサポートする認証方法より 、API へのアクセスを制御できます。

プライベート統合を作成するには、まず VPC リンクを作成する必要があります。VPC リンクの詳細については、「HTTP API の VPC リンクの使用」を参照してください。

VPC リンクを作成したら、Application Load Balancer、Network Load Balancer、または AWS Cloud Map サービスに登録されたリソースに接続するプライベート統合を設定できます。

プライベート統合を作成するには、すべてのリソースが同じ AWS アカウント (ロードバランサーまたは AWS Cloud Map サービス、VPC リンク、および HTTP API を含む) によって所有されている必要があります。

デフォルトでは、プライベート統合トラフィックは HTTP プロトコルを使用します。HTTPS を使用するためにプライベート統合トラフィックが必要な場合は、`tlsConfig` を指定できます。

> **Note**
>
> プライベート統合では、API Gateway はバックエンドリソースへのリクエストに API エンドポイントのステージ部分を含めます。例えば、API の `test` ステージへのリクエストには、プライベート統合へのリクエストに `test/route-path` が含まれます。バックエンドリソースへのリクエストからステージ名を削除するには、パラメータのマッピングを使用して、リクエストパスを `$request.path` にオーバーライドします。

Application Load Balancer または Network Load Balancer を使用したプライベート統合の作成

プライベート統合を作成する前に、VPC リンクを作成する必要があります。VPC リンクの詳細については、「HTTP API の VPC リンクの使用」を参照してください。

Application Load Balancer または Network Load Balancer とのプライベート統合を作成するには、HTTP プロキシ統合を作成し、使用する VPC リンクを指定して、ロードバランサーのリスナー ARN を指定します。

VPC リンクを使用してロードバランサーに接続するプライベート統合を作成するには、次のコマンドを使用します。

```
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
```

AWS Cloud Map サービス検出を使用したプライベート統合の作成

プライベート統合を作成する前に、VPC リンクを作成する必要があります。VPC リンクの詳細については、「HTTP API の VPC リンクの使用」を参照してください。

AWS Cloud Map との統合のために、API Gateway は `DiscoverInstances` を使用してリソースを識別します。クエリパラメータを使用して、特定のリソースをターゲットにすることができます。登録されたリソースの属性には、IP アドレスとポートを含める必要があります。API Gateway は、`DiscoverInstances` から返される正常なリソース間にリクエストを分散します。詳細については、AWS Cloud Map API リファレンスの「DiscoverInstances」を参照してください。

> **Note**
>
> Amazon ECS を使用して AWS Cloud Map にエントリを入力する場合は、Amazon ECS サービス検出で SRV レコードを使用するように Amazon ECS タスクを設定するか、Amazon ECS Service Connect を有効にする必要があります。詳細については、「Amazon Elastic Container Service デベロッパーガイド」の「相互接続サービス」を参照してください。

AWS Cloud Map とのプライベート統合を作成するには、HTTP プロキシ統合を作成し、使用する VPC リンクを指定して、AWS Cloud Map サービスの ARN を指定します。

AWS Cloud Map サービス検出を使用してリソースを識別するプライベート統合を作成するには、次のコマンドを使用します。

```
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
```

HTTP API の VPC リンクの使用

VPC リンクを使用すると、Application Load Balancer または Amazon ECS コンテナベースのアプリケーションなどの、HTTP API ルートを VPC 内のプライベートリソースに接続するプライベート統

合を作成できます。プライベート統合の作成の詳細については、「HTTP API のプライベート統合の使用」を参照してください。

プライベート統合は、VPC リンクを使用して、API Gateway と、対象となる VPC リソース間の接続をカプセル化します。異なるルートと API 間で VPC リンクを再利用できます。

VPC リンクを作成すると、API Gateway はアカウントの VPC リンク用の Elastic Network Interface を作成および管理します。このプロセスには数分かかることがあります。VPC リンクを使用する準備ができたら、状態は `PENDING` から `AVAILABLE` に移行します。

> **Note**
>
> VPC リンク経由で 60 日間トラフィックが送信されない場合は、`INACTIVE` になります。VPC リンクが `INACTIVE` 状態の場合、API Gateway は VPC リンクのネットワークインターフェイスをすべて削除します。これにより、VPC リンクに依存する API リクエストが失敗します。API リクエストが再開すると、API Gateway はネットワークインターフェイスを再プロビジョニングします。ネットワークインターフェイスを作成し、VPC リンクを再度アクティブ化するには、数分かかることがあります。VPC リンクステータスを使用して、VPC リンクの状態を監視できます。

AWS CLI を使用して VPC リンクを作成する

VPC リンクを作成するには、以下のコマンドを使用します。VPC リンクを作成するには、関連するすべてのリソースが同じ AWS アカウントによって所有されている必要があります。

```
aws apigatewayv2 create-vpc-link --name MyVpcLink \
    --subnet-ids subnet-aaaa subnet-bbbb \
    --security-group-ids sg1234 sg5678
```

> **Note**
>
> VPC リンクは変更できません。VPC リンクを作成した後は、そのサブネットやセキュリティグループを変更することはできません。

AWS CLI を使用して VPC リンクを削除する

次のコマンドを使用して、VPC リンクを削除します。

```
aws apigatewayv2 delete-vpc-link --vpc-link-id abcd123
```

リージョン別の利用可能性

HTTP API の VPC リンクは、次のリージョンとアベイラビリティーゾーンでサポートされています。

リージョン名	リージョン	サポートされているアベイラビリティーゾーン
米国東部 (オハイオ)	us-east-2	use2-az1、use2-az2、use2-az3
米国東部 (バージニア北部)	us-east-1	use1-az1、use1-az2、use1-az4、use1-az5、use1-az6
米国西部 (北カリフォルニア)	us-west-1	usw1-az1、usw1-az3
米国西部 (オレゴン)	us-west-2	usw2-az1、usw2-az2、usw2-az3、usw2-az4
アジアパシフィック (香港)	ap-east-1	ape1-az2、ape1-az3
アジアパシフィック (ムンバイ)	ap-south-1	aps1-az1、aps1-az2、aps1-az3
アジアパシフィック (ソウル)	ap-northeast-2	apne2-az1、apne2-az2、apne2-az3
アジアパシフィック (シンガポール)	ap-southeast-1	apse1-az1、apse1-az2、apse1-az3

リージョン名	リージョン	サポートされているアベイラビリティーゾーン
アジアパシフィック (シドニー)	ap-southeast-2	apse2-az1、apse2-az3、apse2-az2
アジアパシフィック (東京)	ap-northeast-1	apne1-az1、apne1-az2、apne1-az4
カナダ (中部)	ca-central-1	cac1-az1、cac1-az2
欧州 (フランクフルト)	eu-central-1	euc1-az1、euc1-az2、euc1-az3
欧州 (アイルランド)	eu-west-1	euw1-az1、euw1-az2、euw1-az3
欧州 (ロンドン)	eu-west-2	euw2-az1、euw2-az2、euw2-az3
欧州 (パリ)	eu-west-3	euw3-az1、euw3-az3
欧州 (ストックホルム)	eu-north-1	eun1-az1、eun1-az2、eun1-az3
中東 (バーレーン)	me-south-1	mes1-az1、mes1-az2、mes1-az3
南米 (サンパウロ)	sa-east-1	sae1-az1、sae1-az2、sae1-az3
AWS GovCloud (米国西部)	us-gov-west-1	usgw1-az1、usgw1-az2、usgw1-az3

HTTP API の CORS の設定

Cross-origin resource sharing (CORS) は、ブラウザで実行されているスクリプトから開始される HTTP リクエストを制限するブラウザのセキュリティ機能です。API にアクセスできず、`Cross-Origin Request Blocked` を含むエラーメッセージが表示される場合は、CORS を有効にする必要がある場合があります。詳細については、「CORS とは」を参照してください。

通常、CORS は、異なるドメインまたはオリジンでホストされている API にアクセスするウェブアプリケーションを構築するために必要です。CORS を有効にして、別のドメインでホストされているウェブアプリケーションから API へのリクエストを許可することができます。たとえば、API が `https://{api_id}.execute-api.{region}.amazonaws.com/` でホストされていて、`example.com` でホストされているウェブアプリケーションから API を呼び出す場合、API が CORS をサポートしている必要があります。

API に CORS を設定した場合、API Gateway は API に OPTIONS ルートが設定されていない場合でも、プリフライト OPTIONS リクエストに対するレスポンスを自動的に送信します。CORS リクエストの場合、API Gateway は設定された CORS ヘッダーを、統合からのレスポンスに追加します。

> **Note**
>
> API の CORS を設定する場合、API Gateway はバックエンド統合から返された CORS ヘッダーを無視します。

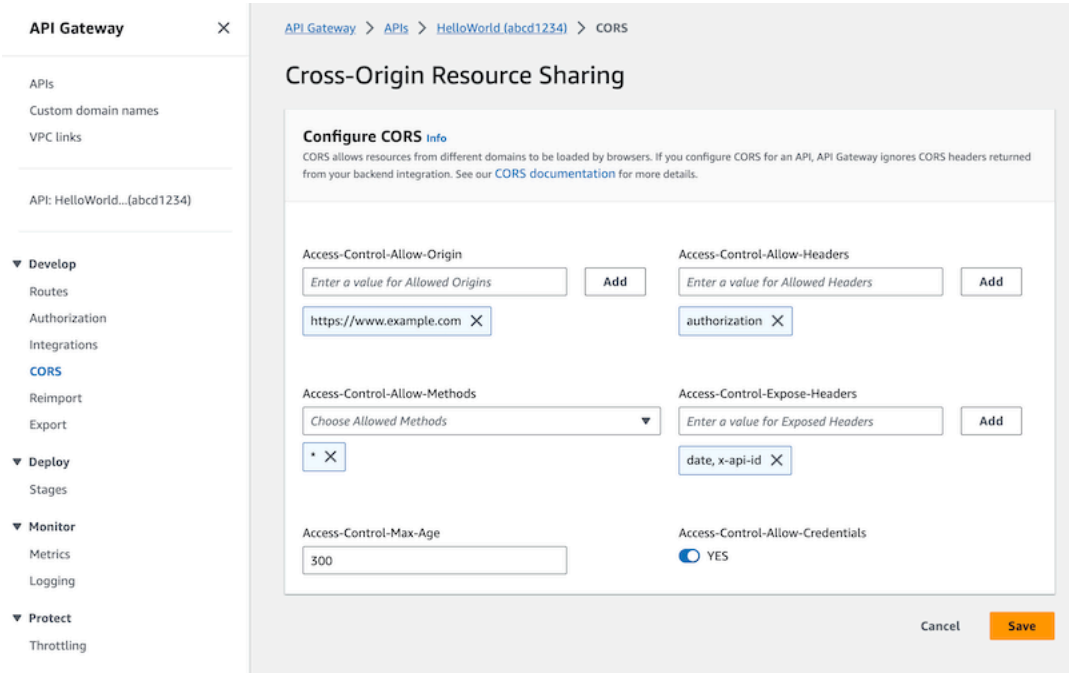
CORS 設定では、次のパラメータを指定できます。API Gateway HTTP API コンソールを使用してこれらのパラメータを追加するには、値を入力した後に [追加] を選択します。

CORS ヘッダー	CORS 設定プロパティ	値の例
Access-Control-Allow-Origin	allowOrigins	• `https://www.example.com` • `*` (すべてのオリジンを許可する) • `https://*` (https:// で始まる任意のオリジンを許可する)

CORS ヘッダー	CORS 設定プロパティ	値の例
		• `http://*` (http:// で始まる任意のオリジンを許可する)
Access-Control-Allow-Credentials	allowCredentials	true
Access-Control-Expose-Headers	exposeHeaders	日付、x-api-id、*
Access-Control-Max-Age	maxAge	300
Access-Control-Allow-Methods	allowMethods	GET, POST, DELETE , *
Access-Control-Allow-Headers	allowHeaders	Authorization, *

CORS ヘッダーを返すには、リクエストに `origin` ヘッダーが含まれている必要があります。

CORS 設定は次の例のようになります。



`$default` ルートおよびオーソライザーによる HTTP API の CORS の設定

CORS を有効にし、 HTTP API の任意のルートに対して認証を設定できます。$default ルートの CORS および認証を有効にする場合、いくつかの特別な考慮事項があります。$default ルートは、OPTIONS リクエストを含め、明示的に定義していないすべてのメソッドとルートのリクエストをキャッチします。不正な OPTIONS リクエストをサポートするには、認証を必要としない OPTIONS /{proxy+} ルートを API に追加し、ルートに統合をアタッチします。OPTIONS /{proxy+} ルートの優先順位は $default ルートよりも高くなります。その結果、クライアントは認証なしで API に OPTIONS リクエストを送信できます。ルーティングの優先順位の詳細については、「API リクエストのルーティング」を参照してください。

AWS CLI を使用して HTTP API の CORS を設定する

次の update-api コマンドを使用して、https://www.example.com から CORS リクエストを有効にすることができます。

Example

```
aws apigatewayv2 update-api --api-id api-id --cors-configuration AllowOrigins="https://
www.example.com"
```

詳細については、Amazon API Gateway バージョン 2 API リファレンスの「CORS」を参照してください。

API リクエストとレスポンスの変換

バックエンド統合に到達する前に、クライアントからの API リクエストを変更できます。API Gateway がクライアントに応答を返す前に、統合からレスポンスを変更することもできます。パラメータマッピングを使用して、HTTP API の API リクエストとレスポンスを変更します。パラメーターマッピングを使用するには、変更する API リクエストまたはレスポンスパラメーターを指定し、それらのパラメーターを変更する方法を指定します。

API リクエストの変換

リクエストパラメーターを使用して、バックエンド統合に到達する前にリクエストを変更します。ヘッダー、クエリ文字列、またはリクエストパスを変更できます。

リクエストパラメータは、キーと値のマップです。キーは、変更するリクエストパラメーターの場所とその変更方法を識別します。値は、パラメータの新しいデータを指定します。

次の表に、サポートされているキーを示します。

パラメータのマッピングキー

タイプ	構文
ヘッダー	`append|overwrite|remove:header.` *`headername`*
クエリ文字列	`append|overwrite|remove:querystring.` *`querystring-name`*
パス	`overwrite:path`

次の表に、パラメータにマップできるサポートされている値を示します。

パラメータのマッピング値をリクエストする

タイプ	構文	コメント
ヘッダー値	$request.header.*name* または ${request.header.*name*}	ヘッダー名では大文字と小文字は区別されません。API Gateway は、複数のヘッダー値をコンマで結合します (例: `"header1": "value1,value2"`)。一部のヘッダーは予約されています。詳細については、「the section called “予約済みヘッダー”」を参照してください。
クエリ文字列値	$request.querystring.*name* または ${request.querystring.*name*}	クエリ文字列名では大文字と小文字が区別されます。API Gateway は複数の値をコンマで結合します (例: `"querystring1" "Value1,Value2"`)。

タイプ	構文	コメント
リクエスト本文	$request.body.*name* または ${request.body.*name*}	JSON path 式。再帰下降 (`$request.body..name`) およびフィルタ式 (`?(expression)`) はサポートされていません。 **Note** JSON パスを指定すると、API Gateway はリクエスト本文を 100 KB で切り捨て、選択式を適用します。100 KB を超えるペイロードを送信するには、`$request.body` を指定します。
リクエストパス	$request.path または ${request.path}	ステージ名を含まないリクエストパス。
パスパラメータ	$request.path.*name* または ${request.path.*name*}	リクエストのパスパラメータの値。例えば、ルートが `/pets/{petId}` の場合は、`petId` を使用してリクエストのパラメータ`$request.path.petId` をマッピングできます。

タイプ	構文	コメント
コンテキスト変数	$context.*variableName* または ${context.*variableName*}	コンテキスト変数の値 。 **Note** 特殊文字の . と _ のみがサポートされています。
ステージ変数	$stageVariables.*variableName* または ${stageVariables.*variableName*}	ステージ変数の値。
静的な値	*string*	定数値。

Note

選択式で複数の変数を使用するには、変数を角かっこで囲みます。例えば、`${request.path.name} ${request.path.id}` と指定します。

API レスポンスの変換

レスポンスをクライアントに返す前に、レスポンスパラメーターを使用して HTTP レスポンスをバックエンド統合から変換します。API Gateway がクライアントにレスポンスを返す前に、ヘッダーまたはステータスコードを変更できます。

統合によって返されるステータスコードごとに、レスポンスパラメーターを構成します。レスポンスパラメータは、キーと値のマップです。キーは、変更するリクエストパラメーターの場所とその変更方法を識別します。値は、パラメータの新しいデータを指定します。

次の表に、サポートされているキーを示します。

応答パラメータのマッピングキー

タイプ	構文
ヘッダー	`append|overwrite|remove:header.` *`headername`*
ステータスコード	`overwrite:statuscode`

次の表に、パラメータにマップできるサポートされている値を示します。

レスポンスパラメータのマッピング値

タイプ	構文	コメント
ヘッダー値	$response.header.*name* または ${response.header.*name*}	ヘッダー名では大文字と小文字は区別されません。API Gateway は、複数のヘッダー値をコンマで結合します (例: `"header1": "value1,value2"`)。一部のヘッダーは予約されています。詳細については、「the section called "予約済みヘッダー"」を参照してください。
レスポンス本文	$response.body.*name* または ${response.body.*name*}	JSON パス式。再帰下降 (`$response.body..name`) およびフィルター式 (`?(expression)`) はサポートされていません。 **Note** JSON パスを指定すると、API Gateway はレスポンス本文を 100 KB で切り捨て、選択式を適用しま

タイプ	構文	コメント
		す。100 KB を超えるペイロードを送信するには、`$response.body` を指定します。
コンテキスト変数	$context.*variableName* または ${context.*variableName*}	サポートされているコンテキスト変数の値。
ステージ変数	$stageVariables.*variableName* または ${stageVariables.*variableName*}	ステージ変数の値。
静的な値	*string*	定数値。

> **Note**
>
> 選択式で複数の変数を使用するには、変数を角かっこで囲みます。例えば、`${request.path.name} ${request.path.id}` と指定します。

予約済みヘッダー

次のヘッダーは予約されています。これらのヘッダーには、要求またはレスポンスマッピングを構成できません。

- 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

例

以下の AWS CLI 例は、パラメータマッピングを設定します。AWS CloudFormation テンプレートの例については、GitHub を参照してください。

API リクエストへのヘッダーの追加

次の例では、バックエンド統合に到達する前に API リクエストにheader1という名前のヘッダーを追加します。API Gateway は、ヘッダーにリクエスト ID を設定します。

```
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" }'
```

リクエストヘッダーの名前を変更する

次の例では、リクエストヘッダーの名前をheader1からheader2に変更します 。

```
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": "''"}'
```

統合からのレスポンスの変更

次の例では、統合のレスポンスパラメータを設定します。インテグレーションが 500 ステータスコードを返すと、API Gateway はステータスコードを 403 に変更し、応答に header1 1 を追加します。統合によって 404 ステータスコードが返されると、API Gateway は応答にヘッダーerrorを追加します。

```
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"}  }'
```

構成されたパラメータマッピングの削除

次のコマンド例では、以前に設定されたリクエストパラメータappend:header.header1を削除します。また、200 ステータスコードに対して以前に構成されたレスポンスパラメータも削除されます。

```
aws apigatewayv2 update-integration \
    --api-id abcdef123 \
    --integration-id hijk456 \
    --request-parameters '{"append:header.header1" : ""}' \
    --response-parameters '{"200" : {}}'
```

HTTP API の OpenAPI 定義の使用

HTTP API は、OpenAPI 3.0 定義ファイルを使用して定義できます。次に、定義を API Gateway にインポートして API を作成できます。OpenAPI への API Gateway 拡張の詳細については、「OpenAPI の拡張」を参照してください。

HTTP API のインポート

HTTP API は、OpenAPI 3.0 定義ファイルをインポートすることによって作成できます。

REST API から HTTP API に移行するには、REST API を OpenAPI 3.0 定義ファイルとしてエクスポートできます。次に、API 定義を HTTP API としてインポートします。REST API のエクスポートの詳細については、「API Gateway から REST API をエクスポートする」を参照してください。

> **Note**
>
> HTTP API は、REST API と同じ AWS 変数をサポートします。詳細については、「OpenAPI インポート用の AWS 変数」を参照してください。

検証情報のインポート

API のインポート時に、API Gateway から 3 つのカテゴリの検証情報が提供されます。

Info

プロパティは OpenAPI 仕様では有効ですが、そのプロパティは HTTP API ではサポートされていません。

たとえば、次の OpenAPI 3.0 スニペットは、HTTP API がリクエストの検証をサポートしていないため、インポート時に情報を生成します。API Gateway は、`requestBody` フィールドと `schema` フィールドを無視します。

```
"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

プロパティまたは構造は、OpenAPI 仕様では無効ですが、API の作成をブロックしません。API Gateway でこれらの警告を無視して API の作成を続行するか、警告時に API の作成を停止するかを指定できます。

次の OpenAPI 3.0 ドキュメントでは、HTTP API は Lambda プロキシと HTTP プロキシの統合のみをサポートしているため、インポート時に警告が生成されます。

```
"x-amazon-apigateway-integration": {
  "type": "AWS",
  "httpMethod": "POST",
  "uri": "arn:aws:lambda:us-east-2:123456789012:function:HelloWorld",
  "payloadFormatVersion": "1.0"
}
```

エラー

OpenAPI 仕様が無効であるか、形式が正しくありません。API Gateway は、不正な形式のドキュメントからリソースを作成できません。エラーを修正してから、もう一度やり直してください。

次の API 定義では、HTTP API は OpenAPI 3.0 仕様のみをサポートしているため、インポート時にエラーが発生します。

```
{
  "swagger": "2.0.0",
  "info": {
    "title": "My API",
    "description": "An Example OpenAPI definition for Errors/Warnings/ImportInfo",
    "version": "1.0"
  }
  ...
}
```

別の例として、OpenAPI では、ユーザーが特定のオペレーションに複数のセキュリティ要件を適用した API を定義できますが、API Gateway はこれをサポートしません。各オペレーションには、IAM 認証、Lambda オーソライザー 、または JWT オーソライザーのいずれかしか持つことができません。複数のセキュリティ要件をモデル化しようとすると、エラーが発生します。

AWS CLI を使用した API のインポート

次のコマンドは、OpenAPI 3.0 定義ファイル `api-definition.json` を HTTP API としてインポートします。

Example

```
aws apigatewayv2 import-api --body file://api-definition.json
```

Example

次のサンプル OpenAPI 3.0 定義をインポートして、HTTP API を作成することができます。

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

API Gateway からの HTTP API のエクスポート

HTTP API を作成したら、API Gateway から API の OpenAPI 3.0 定義をエクスポートできます。エクスポートするステージを選択するか、API の最新の設定をエクスポートできます。エクスポートした API 定義を API Gateway にインポートして、同一の API をもう 1 つ作成することもできます。API 定義のインポートの詳細については、「HTTP API のインポート」を参照してください。

AWS CLI を使用してステージの OpenAPI 3.0 定義をエクスポートする

以下のコマンドは、prod という名前の API ステージの OpenAPI 定義を、stage-definition.yaml という名前の YAML ファイルにエクスポートします。エクスポートされた定義ファイルには、デフォルトで API Gateway 拡張が含まれます。

```
aws apigatewayv2 export-api \
    --api-id api-id  \
    --output-type YAML  \
    --specification OAS30 \
    --stage-name prod \
    stage-definition.yaml
```

AWS CLI を使用して API の最新変更の OpenAPI 3.0 定義をエクスポートする

以下のコマンドは、HTTP API の OpenAPI 定義を latest-api-definition.json という名前の JSON ファイルにエクスポートします。このコマンドはステージを指定しないため、API Gateway は、ステージにデプロイされているかどうかに関係なく、API の最新の設定をエクスポートします。エクスポートされた定義ファイルには、API Gateway 拡張は含まれません。

```
aws apigatewayv2 export-api \
    --api-id api-id  \
    --output-type JSON  \
    --specification OAS30 \
    --no-include-extensions \
    latest-api-definition.json
```

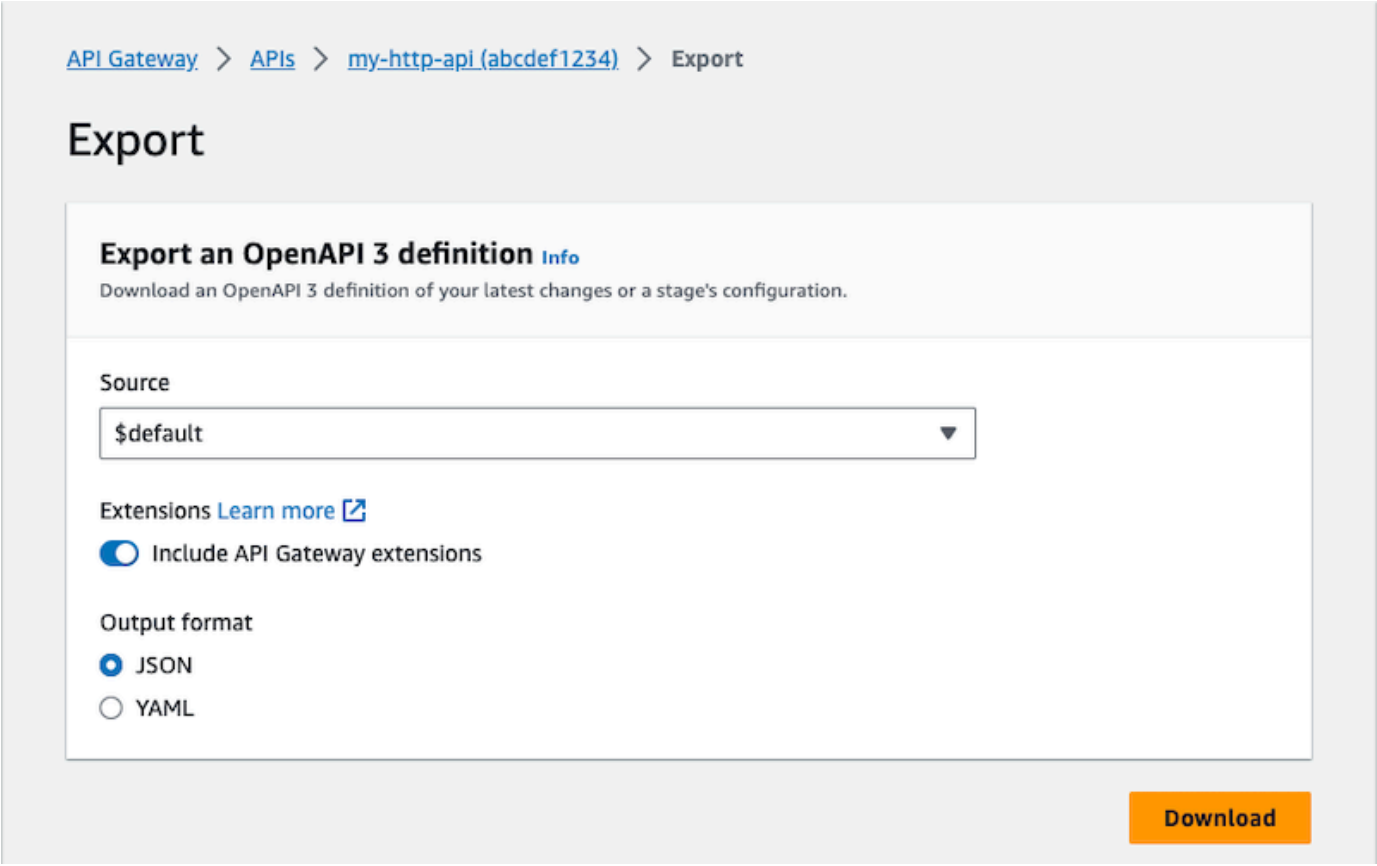
詳細については、Amazon API Gateway バージョン 2 API リファレンスの「ExportAPI」を参照してください。

API Gateway コンソールを使用して OpenAPI 3.0 定義をエクスポートする

以下の手順は、HTTP API の OpenAPI 定義をエクスポートする方法を示しています。

API Gateway コンソールを使用して OpenAPI 3.0 定義をエクスポートするには

1. https://console.aws.amazon.com/apigateway で API Gateway コンソールにサインインします。
2. HTTP API を選択します。
3. メインナビゲーションペインの [開発] で、[エクスポート] を選択します。
4. API をエクスポートするには、以下のオプションから選択してください。



a. [ソース] で、OpenAPI 3.0 定義のソースを選択します。エクスポートするステージを選択するか、API の最新の設定をエクスポートできます。

b. API Gateway 拡張機能を含める場合は、[API Gateway 拡張機能を含める] をオンにします。

c. [出力形式] では、出力形式を選択します。

5. [ダウンロード] を選択します。

顧客が呼び出すための HTTP API の公開

ステージとカスタムドメイン名を使用して、クライアントが呼び出す API を発行できます。

API ステージは、API のライフサイクル状態への論理的なリファレンスです (例: `dev`、`prod`、`beta`、`v2` など)。各ステージは、API のデプロイの名前付きリファレンスで、クライアントアプリケーションから呼び出すことができます。API の各ステージに対して、さまざまな統合と設定を構成できます。

カスタムドメイン名を使用すると、クライアントがデフォルト URL ではなく API を呼び出すための、よりシンプルで直感的な URL `https://api-id.execute-api.region.amazonaws.com/stage` を提供できます。

> **Note**
>
> API Gateway API のセキュリティを強化するため、`execute-api.{region}.amazonaws.com` ドメインはパブリックサフィックスリスト (PSL) に登録されます。セキュリティ強化のため、API Gateway API のデフォルトドメイン名に機密な Cookie を設定する必要が生じた場合は、`__Host-` プレフィックスの付いた Cookie の使用をお勧めします。このプラクティスは、クロスサイトリクエストフォージェリ (CSRF) 攻撃からドメインを防ぐ際に役立ちます。詳細については、Mozilla 開発者ネットワークの「Set-Cookie」ページを参照してください。

トピック

- HTTP API のステージの使用
- HTTP API のセキュリティポリシー
- HTTP API のカスタムドメイン名の設定

HTTP API のステージの使用

API ステージは、API のライフサイクル状態への論理的なリファレンスです (例: `dev`、`prod`、`beta`、`v2` など)。API ステージは API ID とステージ名で識別され、API の呼び出しに使用する URL に含まれます。各ステージは、API のデプロイの名前付きリファレンスで、クライアントアプリケーションから呼び出すことができます。

ステージには、API のベース URL から呼び出される `$default` のステージを作成することができます。ベース URL は、`https://{api_id}.execute-api.{region}.amazonaws.com/` のような形式になります。この URL を使用して API ステージを呼び出します。

デプロイは、API 設定のスナップショットです。ステージに API をデプロイすると、クライアントがその API を呼び出すことができます。変更を有効にするには、API をデプロイする必要があります。自動デプロイを有効にすると、API に対する変更が自動的にリリースされます。

ステージ変数

ステージ変数は、HTTP API のステージに対して定義できるキーと値のペアです。環境変数と同様に機能し、API のセットアップで使用できます。

たとえば、ステージ変数を定義し、その値を HTTP プロキシ統合の HTTP エンドポイントとして設定することができます。後で、関連付けられたステージ変数名を使用してエンドポイントを参照できます。これにより、各ステージで異なるエンドポイントで同じ API セットアップを使用できます。同様に、ステージ変数を使用して、API の各ステージに異なる AWS Lambda 関数の統合を指定できます。

> **Note**
>
> ステージ変数は、認証情報などの機密データに使用されることを意図していません。機密データを統合に渡すには、AWS Lambda オーソライザーを使用します。Lambda オーソライザーの出力では、機密データを統合に渡すことができます。詳細については、「the section called “Lambda オーソライザーのレスポンス形式”」を参照してください。

例

ステージ変数を使用して HTTP 統合エンドポイントをカスタマイズするには、まずステージ変数の名前と値 (`url` など) を `example.com` の値で設定する必要があります。次に、HTTP プロキシ統合を設定します。エンドポイントの URL を入力する代わりに、ステージ変数の値、**`http://${stageVariables.url}`** を使用するように API Gateway に指示できます 。この値を指定すると、API Gateway は、ランタイムに API のステージに応じてステージ変数の `${}` を置き換えます。

同様に、ステージ変数を参照して Lambda 関数名や AWS のロールの ARN を指定することができます。

ステージ変数値として Lambda 関数名を指定する場合は、その Lambda 関数に対するアクセス許可を手動で設定する必要があります。これを行うには、AWS Command Line Interface (AWS CLI) を使用できます。

```
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 のステージ変数のリファレンス

HTTP 統合 URI

ステージ変数は、次の例に示すように、HTTP 統合 URI の一部として使用できます。

- プロトコルのない完全な URI – `http://${stageVariables.<variable_name>}`
- 完全なドメイン – `http://${stageVariables.<variable_name>}/resource/operation`
- サブドメイン – `http://${stageVariables.<variable_name>}.example.com/resource/operation`
- パス – `http://example.com/${stageVariables.<variable_name>}/bar`
- クエリ文字列 – `http://example.com/foo?q=${stageVariables.<variable_name>}`

Lambda 関数

ステージ変数は、次の例に示すように、Lambda 関数の統合名やエイリアスの代わりに使用できます。

- `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**
>
> Lambda 関数にステージ変数を使用するには、関数が API と同じアカウントにある必要があります。ステージ変数は、クロスアカウント Lambda 関数をサポートしていません。

AWS 統合認証情報

次の例に示すように、ステージ変数を AWS ユーザーまたはロールの認証情報 ARN の一部として使用できます。

- `arn:aws:iam::<account_id>:${stageVariables.<variable_name>}`

HTTP API のセキュリティポリシー

API Gateway は、すべての HTTP API エンドポイントに TLS_1_2 のセキュリティポリシーを適用します。

セキュリティポリシーとは、Amazon API Gateway が提供する TLS の最小バージョンと暗号スイートの事前定義された組み合わせです。TLS プロトコルは、クライアントとサーバーの間の改ざんや傍受などのネットワークセキュリティの問題に対処します。クライアントがカスタムドメインを介して API に TLS ハンドシェイクを確立すると、セキュリティポリシーにより、TLS バージョンと暗号スイートのオプションが適用されます。ここで使用するオプションは、クライアントが選択できます。このセキュリティポリシーは、TLS 1.2 および TLS 1.3 トラフィックを受け入れ、TLS 1.0 トラフィックを拒否します。

HTTP API でサポートされている TLS プロトコルと暗号

次の表は、HTTP API でサポートされている TLS プロトコルと暗号を示しています。

セキュリティポリシー	TLS_1_2
TLS プロトコル	
TLSv1.3	♦
TLSv1.2	♦
TLS 暗号	
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	♦

セキュリティポリシー	TLS_1_2
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 および RFC の暗号名

OpenSSL と IETF RFC 5246 では、同じ暗号に異なる名前を使用します。暗号名のリストについては、「the section called “OpenSSL および RFC の暗号名”」を参照してください。

REST API と WebSocket API に関する情報

REST API と WebSocket API の詳細については、「the section called “セキュリティポリシーの選択”」と「the section called “WebSocket API のセキュリティポリシー”」を参照してください。

HTTP API のカスタムドメイン名の設定

カスタムドメイン名は、API ユーザーに提供できる、よりシンプルで直感的な URL です。

API のデプロイ後、お客様 (およびその顧客) は、以下の形式のデフォルトのベース URL を使用して API を呼び出すことができます。

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

`api-id` は API Gateway によって生成され、`region` (AWS リージョン) は API の作成時に、`stage` は API のデプロイ時に、ユーザーが指定します。

URL のホスト名の部分 (つまり *api-id*.execute-api.*region*.amazonaws.com) は API エンドポイントを参照します。デフォルトの API エンドポイントは再呼び出しが難しく、ユーザーフレンドリではありません。

カスタムドメイン名を使用すると、API のホスト名を設定し、代替パスを API にマッピングするための基本パス (myservice など) を選択できます。たとえば、API のよりわかりやすい ベース URL は以下のようになります。

```
https://api.example.com/myservice
```

> **Note**
>
> カスタムドメインは、REST API と HTTP API に関連付けることができます。API Gateway バージョン 2 API を使用すると、REST API と HTTP API のリージョン別カスタムドメイン名を作成および管理することができます。
> HTTP API の場合、サポートされる TLS バージョンは TLS 1.2 のみです。

ドメイン名を登録する

API のカスタムドメイン名を設定するには、登録されたインターネットドメイン名が必要です。ドメイン名は RFC 1035 仕様に準拠している必要があり、ラベルあたり最大 63オクテット、合計 255 オクテットを含めることができます。必要に応じて、Amazon Route 53 を使用するか、任意のサードパーティーのドメインレジストラを使用して、インターネットドメインを登録できます。API のカスタムドメイン名は、登録されたインターネットドメインのサブドメイン名またはルートドメイン名 (「Zone Apex」など) にすることができます。

カスタムドメイン名が API Gateway で作成されたら、API エンドポイントにマッピングするために DNS プロバイダーのリソースレコードを作成または更新する必要があります。このマッピングを行わないと、カスタムドメイン名宛ての API リクエストが API Gateway に届きません。

リージョン別カスタムドメイン名

特定のリージョンの API のカスタムドメイン名を作成すると、API Gateway は API のリージョン別ドメイン名を作成します。カスタムドメイン名をリージョン別ドメイン名にマッピングするように、DNS レコードを設定する必要があります。カスタムドメイン名の証明書を提供する必要もあります。

ワイルドカードカスタムドメイン名

ワイルドカードカスタムドメイン名を使用すると、デフォルトのクォータを超えずにほぼ無数のドメイン名をサポートできます。たとえば、各お客様に個別のドメイン名を付けることができます *customername*.api.example.com。

ワイルドカードカスタムドメイン名を制作するためには、ルートドメインの可能なすべてのサブドメインを表すカスタムドメインの最初のサブドメインとして、ワイルドカード (*) を指定します。

たとえば、ワイルドカードカスタムドメイン名として *.example.com を使用すると、a.example.com、b.example.com、c.example.com などのサブドメインが生成され、これらはすべて同じドメインにルーティングされます。

ワイルドカードカスタムドメイン名は、API Gateway の標準のカスタムドメイン名とは異なる設定をサポートします。たとえば、1 つの AWS アカウントで、*.example.com と a.example.com を異なる動作に設定できます。

ワイルドカードカスタムドメイン名を作成するには、DNS または E メール検証方法を使用して検証された証明書を ACM から発行してもらう必要があります。

> **Note**
>
> 別の AWS アカウントで作成済みのカスタムドメイン名と競合するようなワイルドカードカスタムドメイン名を作成することはできません。たとえば、アカウント A で a.example.com が作成済みである場合、アカウント B はワイルドカードカスタムドメイン名として *.example.com を作成できません。
> アカウント A とアカウント B の所有者が同じである場合は、AWS サポートセンターに連絡して例外をリクエストできます。

カスタムドメイン名の証明書

> **Important**
>
> カスタムドメイン名の証明書を指定します。アプリケーションで証明書ピンニング (SSL ピンニングとも呼ばれる) を使用して ACM 証明書を固定すると、AWS が証明書を更新した後にアプリケーションがドメインに接続できないことがあります。詳細については、「AWS Certificate Manager ユーザーガイド」の「証明書のピンニングの問題」を参照してください。

ACM がサポートされているリージョンでカスタムドメイン名の証明書を提供するには、ACM に証明書をリクエストする必要があります。ACM がサポートされていないリージョンで、リージョン別カスタムドメイン名の証明書を提供するには、そのリージョン内の API Gateway に証明書をインポートする必要があります。

SSL/TLS 証明書をインポートするには、カスタムドメイン名の PEM 形式の SSL/TLS 認証本文、そのプライベートキー、およびカスタムドメイン名の証明書チェーンを提供する必要があります。ACM に保存された各証明書は ARN によって識別されます。AWS で管理された証明書をドメイン名で使用するには、その ARN を単に参照します。

ACM を使用すると、API のカスタムドメイン名を簡単に設定して使用できます。特定のドメイン名の証明書を作成 (または証明書をインポート) し、ACM が提供する証明書の ARN を使用して API Gateway でドメイン名を設定します。次に、カスタムドメイン名のベースパスを、デプロイされた API のステージにマッピングします。ACM 発行の証明書により、プライベートキーなど証明書の機密の詳細が漏れる心配はありません。

カスタムドメイン名の設定の詳細については、「 での証明書の準備AWS Certificate Manager」および「API Gateway でのリージョン別カスタムドメイン名の設定」を参照してください。

HTTP API の API マッピングの使用

API マッピングを使用して、API ステージをカスタムドメイン名に接続します。ドメイン名を作成し、DNS レコードを設定したら、API マッピングを使用して、カスタムドメイン名を使用して API にトラフィックを送信します。

API マッピングは、API、ステージ、およびオプションでマッピングに使用するパスを指定します。たとえば、API の `production` ステージを `https://api.example.com/orders` にマッピングできます。

HTTP API と REST API ステージを同じカスタムドメイン名にマッピングできます。

API マッピングを作成する前に、API、ステージ、およびカスタムドメイン名が必要です。カスタムドメイン名の作成と設定の詳細については、「the section called “リージョン別カスタムドメイン名の設定”」を参照してください。

API リクエストのルーティング

API マッピングは、例えば `orders/v1/items` と `orders/v2/items` のように、複数のレベルで設定できます。

複数のレベルを持つ API マッピングの場合、API Gateway は、一致するパスが最も長い API マッピングにリクエストをルーティングします。API Gateway は、API マッピング用に設定されたパスだけを考慮し、呼び出す API を選択します。API ルートは考慮しません。リクエストに一致するパスがない場合、API Gateway は空のパス (none) にマッピングした API にリクエストを送信します。

複数のレベルの API マッピングを使用するカスタムドメイン名の場合、API Gateway は、一致するプレフィックスが最も長い API マッピングにリクエストをルーティングします。

たとえば、次の API マッピングを持つカスタムドメイン名 `https://api.example.com` を考えてます。

1. API 1 にマッピングされている (none)。
2. API 2 にマッピングされている `orders`。
3. API 3 にマッピングされている `orders/v1/items`。
4. API 4 にマッピングされている `orders/v2/items`。
5. API 5 にマッピングされている `orders/v2/items/categories`。

リクエスト	選択した API	説明
`https://api.example.com/orders`	`API 2`	リクエストは、この API マッピングと完全に一致します。
`https://api.example.com/orders/v1/items`	`API 3`	リクエストは、この API マッピングと完全に一致します。
`https://api.example.com/orders/v2/items`	`API 4`	リクエストは、この API マッピングと完全に一致します。
`https://api.example.com/orders/v1/items/123`	`API 3`	API Gateway は、最も長い一致パスを持つ API マッピングを選択します。リクエストの最後にある 123 は、選択には影響しません。

リクエスト	選択した API	説明
`https://api.example.com/orders/v2/items/categories/5`	`API 5`	API Gateway は、最も長い一致パスを持つ API マッピングを選択します。
`https://api.example.com/customers`	`API 1`	API Gateway は、空のマッピングをキャッチオールとして使用します。
`https://api.example.com/ordersandmore`	`API 2`	API Gateway は、一致するプレフィックスが最も長い API マッピングを選択します。単一レベルのマッピングで設定されたカスタムドメイン名の場合 (`https://api.example.com/orders` と`https://api.example.com/` のみなど)、API ゲートウェイは、`ordersandmore` と一致するパスがないため、`API 1` を選択します。

制限事項

- API マッピングでは、カスタムドメイン名とマップされた API が同じ AWS アカウントにある必要があります。
- API マッピングに含めることができるのは、文字、数字、および `$-_.+!*'()/` の文字だけです。
- API マッピングのパスの最大文字数は 300 文字です。
- ドメイン名ごとに、複数のレベルで 200 個の API マッピングを設定できます。
- TLS 1.2 セキュリティポリシーでは、HTTP API をリージョン別カスタムドメイン名にだけマッピングできます。
- WebSocket API を HTTP API または REST API と同じカスタムドメイン名にマッピングすることはできません。

API マッピングを作成する

API マッピングを作成するには、最初にカスタムドメイン名、API、およびステージを作成する必要があります。カスタムドメイン名の作成方法については、「the section called “リージョン別カスタムドメイン名の設定”」を参照してください。

例えば、すべてのリソースを作成する AWS Serverless Application Model テンプレートについては、GitHub で「Sessions With SAM」を参照してください。

AWS Management Console

API マッピングを作成するには

1. API Gateway コンソール (https://console.aws.amazon.com/apigateway) にサインインします。
2. [カスタムドメイン名] を選択します。
3. 既に作成したカスタムドメイン名を選択します。
4. [API マッピング] を選択します。
5. [Configure API mappings (API マッピングの設定)] を選択します。
6. [Add new mapping (新しいマッピングを追加)] を選択します。
7. API、Stage、必要に応じて Path を入力します。
8. [保存] を選択します。

AWS CLI

次の AWS CLI コマンドは、API マッピングを作成します。この例では、API Gateway が指定された API およびステージに `api.example.com/v1/orders` に対するリクエストを送信します。

```
aws apigatewayv2 create-api-mapping \
    --domain-name api.example.com \
    --api-mapping-key v1/orders \
    --api-id a1b2c3d4 \
    --stage test
```

AWS CloudFormation

次の AWS CloudFormation 例は、API マッピングを作成します。

```
MyApiMapping:
  Type: 'AWS::ApiGatewayV2::ApiMapping'
  Properties:
    DomainName: api.example.com
    ApiMappingKey: 'orders/v2/items'
    ApiId: !Ref MyApi
    Stage: !Ref MyStage
```

HTTP API のデフォルトのエンドポイントの無効化

デフォルトでは、クライアントは、API Gateway が API 用に生成する `execute-api` エンドポイントを使用して API を呼び出すことができます。クライアントがカスタムドメイン名を使用した場合のみ API にアクセスできるようにするには、デフォルトの `execute-api` エンドポイントを無効にします。

> **Note**
>
> デフォルトエンドポイントを無効にすると、API のすべてのステージに影響します。

次の AWS CLI コマンドは、HTTP API のデフォルトエンドポイントを無効にします。

```
aws apigatewayv2 update-api \
    --api-id abcdef123 \
    --disable-execute-api-endpoint
```

デフォルトエンドポイントを無効にした後は、自動デプロイが有効になっている場合を除き、その変更を有効にするために API をデプロイする必要があります。

次の AWS CLI コマンドは、デプロイを作成します。

```
aws apigatewayv2 create-deployment \
    --api-id abcdef123 \
    --stage-name dev
```

HTTP API の保護

API Gateway は、悪意のあるユーザーやトラフィックの急増など、特定の脅威から API を保護するさまざまな方法を提供します。スロットリングターゲットの設定や相互 TLS の有効化などの戦略を使用して、API を保護できます。このセクションでは、API Gateway を使用してこれらの機能を有効にする方法を説明しています。

トピック

HTTP API へのリクエストの調整

API のスロットリングを設定して、多すぎるリクエストで API の負荷が高くなりすぎないように保護できます。スロットルはベストエフォートベースで適用されるため、これらは保証されたリクエスト上限ではなく、目標として考える必要があります。

API Gateway は、トークンバケットアルゴリズムを使用してトークンでリクエストをカウントし、API へのリクエストを調整します。特に API Gateway では、アカウントのすべての API に送信されるリクエストのレートとバーストをリージョンごとに検証します。トークンバケットアルゴリズムでは、これらの制限の事前定義されたオーバーランがバーストによって許可されますが、場合によっては、他の要因によって制限のオーバーランが発生することがあります。

リクエストの送信数がリクエストの定常レートおよびバーストを超えると、API Gateway はリクエストを調整を開始します。クライアントは、この時点で `429 Too Many Requests` エラーレスポンスを受け取ることがあります。このような例外をキャッチすると、クライアントは失敗したリクエストをレート制限する方法で再送信できます。

API デベロッパーは、API の個々のステージまたはルートに制限を設定して、アカウントのすべての API にわたるパフォーマンス全体を向上させることができます。

リージョンごとのアカウントレベルのスロットリング

API Gateway はデフォルトで、リージョンごとに AWS アカウント内のすべての API 全体で定常状態のリクエスト/秒 (RPS) を制限します。また、リージョンごとに AWS アカウント内のすべての API にわたってバースト (最大バケットサイズ) を制限します。API Gateway では、バースト制限は、API Gateway が `429 Too Many Requests` エラーレスポンスを返す前に処理する同時リクエ

スト送信の目標最大数を表します。スロットリングクォータの詳細については、「クォータと重要な注意点」を参照してください。

アカウントごとの制限は、指定したリージョンのアカウント内のすべての API に適用されます。このアカウントレベルのレート制限は、申請に応じて引き上げることができます。API のタイムアウトが短く、ペイロードが小さい場合、高い制限を設定できます。リージョンごとのアカウントレベルのスロットリング制限の引き上げを申請するには、AWS サポートセンターにお問い合わせください。詳細については、「クォータと重要な注意点」を参照してください。これらの制限は、AWS スロットリングの制限以上に高くすることはできません。

ルートレベルのスロットリング

特定のステージでまたは API の個々のルートでアカウントレベルのリクエストスロットリング制限を上書きするように、ルートレベルのスロットリングを設定できます。デフォルトのルートスロットリング制限は、アカウントレベルのレート制限を超えることはできません。

AWS CLI を使用して、ルートレベルのスロットリングを設定できます。次のコマンドは、API の指定されたステージとルートのカスタムスロットリングを設定します。

```
aws apigatewayv2 update-stage \
    --api-id a1b2c3d4 \
    --stage-name dev \
    --route-settings '{"GET /pets":
{"ThrottlingBurstLimit":100,"ThrottlingRateLimit":2000}}'
```

HTTP API の相互 TLS 認証の設定

相互 TLS 認証には、クライアントとサーバー間の双方向認証が必要です。相互 TLS では、クライアントは X.509 証明書を提示して、API にアクセスするためのアイデンティティを検証する必要があります。相互 TLS は、モノのインターネット (IoT) および B2B アプリケーションの一般的な要件です。

相互 TLS は、API Gateway でサポートされる他の認証オペレーションおよび認可オペレーションとともに使用できます。API Gateway は、クライアントが提供する証明書を Lambda オーソライザーおよびバックエンド統合に転送します。

> **Important**
>
> デフォルトでは、クライアントは、API Gateway が API 用に生成する `execute-api` エンドポイントを使用して API を呼び出すことができます。相互 TLS でカスタムドメイン名を使

用することによってのみクライアントが API にアクセスできるようにするには、デフォルトの execute-api エンドポイントを無効にします。詳細については、「the section called “デフォルトのエンドポイントを無効にする”」を参照してください。

相互 TLS の前提条件

相互 TLS を設定するには、以下が必要です。

- カスタムドメイン名
- カスタムドメイン名用の AWS Certificate Manager で構成されている少なくとも 1 つの証明書
- 設定され Amazon S3 にアップロードされた信頼ストア

カスタムドメイン名

HTTP API の相互 TLS を有効にするには、API のカスタムドメイン名を設定する必要があります。カスタムドメイン名の相互 TLS を有効にした後、カスタムドメイン名をクライアントに提供できます。相互 TLS が有効なカスタムドメイン名を使用して API にアクセスするには、クライアントは API リクエストで信頼できる証明書を提示する必要があります。詳細な情報は、「the section called “カスタムドメイン名”」にあります。

AWS Certificate Manager が発行した証明書を使用する

パブリックに信頼できる証明書は ACM から直接要求すること、またはパブリック証明書または自己署名証明書をインポートすることができます。ACM で証明書を設定するには、「ACM」を参照してください。証明書をインポートする場合は、次のセクションを参照してください。

インポートされた証明書、または AWS Private Certificate Authority 証明書を使用する

ACM にインポートされた証明書、または相互 TLS を用いた AWS Private Certificate Authority からの証明書を使用するには、API Gateway には ACM が発行した ownershipVerificationCertificate が必要です。この所有権証明書は、ドメイン名を使用する許可を持っていることを確認するためにのみ使用されます。TLS ハンドシェイクには使用されません。まだ ownershipVerificationCertificate を持っていない場合は、https://console.aws.amazon.com/acm/ に進み、その 1 つをセットアップします。

ドメイン名の有効期間中、この証明書を有効にしておく必要があります。証明書の有効期限が切れ、自動更新が失敗した場合、ドメイン名の更新はすべてロックされ

ます。他の変更を加える前に、有効な ownershipVerificationCertificate を用いて ownershipVerificationCertificateArn を更新する必要があります。ownershipVerificationCertificate は、API Gateway の別の相互 TLS ドメインのサーバー証明書として使用できません。証明書を ACM に直接再インポートする場合、発行者は以前と同じである必要があります。

信頼ストアの設定

信頼ストアは、.pem ファイル拡張子が付いたテキストファイルです。これらは、証明機関からの証明書の信頼できるリストです。相互 TLS を使用するには、API にアクセスするために信頼する X.509 証明書の信頼ストアを作成します。

信頼ストアには、発行元の CA 証明書からルート CA 証明書までの完全な信頼チェーンを含める必要があります。API Gateway は、信頼チェーンに存在する任意の CA によって発行されたクライアント証明書を受け入れます。パブリックまたはプライベート認証機関からの証明書を使用できます。証明書チェーンの最大長は 4 です。自己署名証明書を提供することもできます。信頼ストアでは、次のハッシュアルゴリズムがサポートされています。

- SHA-256 以上
- RSA-2048 以上
- ECDSA-256 以上

API Gateway はいくつかの証明書プロパティを検証します。Lambda オーソライザーを使用して、証明書が失効しているかどうかのチェックなど、クライアントによる API の呼び出し時に追加のチェックを実行できます。API Gateway は、次のプロパティを検証します。

検証	説明
X.509 構文	証明書は X.509 構文の要件を満たしている必要があります。
整合性	証明書の内容は、信頼ストアの認証機関によって署名された内容から変更されていないことが必要です。
Validity	証明書の有効期間は最新のものであることが必要です。

検証	説明
名前のチェーン/キーのチェーン	証明書の名前とサブジェクトは、途切れのないチェーンを形成する必要があります。証明書チェーンの最大長は 4 です。

信頼ストアを 1 つのファイルで Amazon S3 バケットにアップロードします。

Example certificates.pem

```
-----BEGIN CERTIFICATE-----
<Certificate contents>
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
<Certificate contents>
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
<Certificate contents>
-----END CERTIFICATE-----
...
```

次の AWS CLI コマンドは、`certificates.pem` を Amazon S3 バケットにアップロードします。

```
aws s3 cp certificates.pem s3://bucket-name
```

カスタムドメイン名の相互 TLS の設定

HTTP API の相互 TLS を設定するには、API のリージョン別カスタムドメイン名を使用してください。TLS バージョンは 1.2 以上であることが必要です。カスタムドメイン名の作成と設定の詳細については、「the section called “リージョン別カスタムドメイン名の設定”」を参照してください。

> **Note**
>
> 相互 TLS は、プライベート API ではサポートされていません。

信頼ストアを Amazon S3 にアップロードした後、相互 TLS を使用するようにカスタムドメイン名を設定できます。次のもの (スラッシュを含む) を端末に貼り付けます。

```
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
```

ドメイン名を作成したら、API オペレーション用に DNS レコードと基本パスのマッピングを設定する必要があります。詳細については、「API Gateway でのリージョン別カスタムドメイン名の設定」を参照してください。

相互 TLS を必要とするカスタムドメイン名を使用して API を呼び出す

相互 TLS が有効な API を呼び出すには、クライアントは API リクエストで信頼できる証明書を提示する必要があります。クライアントが API を呼び出そうとすると、API Gateway は信頼ストアでクライアント証明書の発行者を探します。API Gateway でリクエストを続行するには、証明書の発行者と、ルート CA 証明書までの完全な信頼チェーンが信頼ストアにある必要があります。

以下の例の `curl` コマンドは、`api.example.com`, が含まれるリクエストを `my-cert.pem` に送信します。`my-key.key` は証明書のプライベートキーです。

```
curl -v --key ./my-key.key --cert ./my-cert.pem api.example.com
```

API は、信頼ストアが証明書を信頼している場合にのみ呼び出されます。次の条件により、API Gateway が TLS ハンドシェイクに失敗し、`403` ステータスコードのリクエストが拒否されます。証明書が以下の場合:

- 信頼されていない
- 有効期限切れである
- サポートされているアルゴリズムを使用していない

> **Note**
>
> API Gateway は、証明書が失効したかどうかを検証しません。

信頼ストアの更新

信頼ストアの証明書を更新するには、新しい証明書バンドルを Amazon S3 にアップロードします。次に、カスタムドメイン名を更新して、更新された証明書を使用することができます。

Amazon S3 バージョニングを使用して、信頼ストアの複数のバージョンを維持します。新しい信頼ストアバージョンを使用するようにカスタムドメイン名を更新すると、証明書が無効な場合に API Gateway から警告が返されます。

API Gateway は、ドメイン名を更新するときにのみ、証明書の警告を生成します。API Gateway は、以前にアップロードされた証明書の有効期限が切れた場合でも、通知しません。

以下の AWS CLI コマンドは、新しい信頼ストアバージョンを使用するようにカスタムドメイン名を更新します。

```
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'
```

相互 TLS を無効にする

カスタムドメイン名の相互 TLS を無効にするには、以下のコマンドに示すように、カスタムドメイン名から信頼ストアを削除します。

```
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=''
```

証明書の警告のトラブルシューティング

相互 TLS でカスタムドメイン名を作成する場合、信頼ストア内の証明書が有効でない場合に API Gateway から警告が返されます。これは、新しい信頼ストアを使用するようにカスタムドメイン名を更新するときにも発生します。警告は、証明書の問題と、警告の発生元となった証明書のサブジェクトを示します。相互 TLS は引き続き API で有効ですが、一部のクライアントは API にアクセスできない場合があります。

警告の発生元となった証明書を特定するには、信頼ストア内の証明書をデコードする必要があります。openssl などのツールを使用して、証明書をデコードし、そのサブジェクトを特定できます。

以下のコマンドは、証明書の内容 (サブジェクトなど) を表示します。

```
openssl x509 -in certificate.crt -text -noout
```

警告の発生元となった証明書を更新または削除してから、新しい信頼ストアを Amazon S3 にアップロードします。新しい信頼ストアをアップロードした後、その信頼ストアを使用するようにカスタムドメイン名を更新します。

ドメイン名の競合のトラブルシューティング

エラー "The certificate subject <certSubject> conflicts with an existing certificate from a different issuer." は、複数の認証機関がこのドメインの証明書を発行したことを表します。証明書の件名ごとに、相互 TLS ドメイン用の API Gateway には 1 人の発行者しか存在できません。1 つの発行者を通じて、その件名に関するすべての証明書を取得する必要があります。管理できない証明書に問題があるけれども、ドメイン名の所有権を証明できる場合は、連絡先 AWS Support からチケットを開きます。

ドメイン名のステータスメッセージのトラブルシューティング

PENDING_CERTIFICATE_REIMPORT: これは、証明書を ACM に再インポートし、検証に失敗したことを意味します。すなわち、新しい証明書には SAN (サブジェクトの別名) があり、ownershipVerificationCertificate または証明書のサブジェクトまたは SAN はドメイン名をカバーしないからです。何かが正しく設定されていないか、無効な証明書がインポートされた可能性があります。有効な証明書を ACM に再インポートする必要があります。検証の詳細については、「ドメインの所有権の検証」を参照してください。

PENDING_OWNERSHIP_VERIFICATION: 以前に検証した証明書の有効期限が切れ、ACM がそれを自動更新できなかったことを意味します。証明書を更新するか、新しい証明書をリクエストする必要があります。証明書の更新の詳細については、「ACM の管理された証明書の更新に関するトラブルシューティングガイド」を参照してください。

HTTP API のモニタリング

CloudWatch メトリクスと CloudWatch Logs を使用して HTTP API をモニタリングできます。ログとメトリクスを組み合わせることで、エラーをログに記録し、API のパフォーマンスを監視できます。

> **Note**
>
> API Gateway は、次の場合にログとメトリクスが生成されない可能性があります。
>
> - 413 Request Entity Too Large エラー
> - 過剰な 429 Too Many Requests エラー
> - API マッピングを持たないカスタムドメインに送信されたリクエストからの 400 シリーズのエラー
> - 内部の障害によって発生した 500 シリーズのエラー

トピック

HTTP API のメトリクスの使用

CloudWatch を使用して API の実行をモニタリングすることで、API Gateway から raw データを収集し、リアルタイムに近い読み取り可能なメトリクスに加工することができます。これらの統計は 15 か月間記録されるため、履歴情報にアクセスしてウェブアプリケーションやサービスの動作をより的確に把握できます。デフォルトでは、API Gateway のメトリクスデータは 1 分間隔で自動的に CloudWatch に送信されます。メトリクスをモニタリングするには、API 用の CloudWatch ダッシュボードを作成します。CloudWatch ダッシュボードの作成方法の詳細については、「Amazon CloudWatch ユーザーガイド」の「CloudWatch ダッシュボードの作成」を参照してください。詳細については、Amazon CloudWatch ユーザーガイドの「Amazon CloudWatch とは」を参照してください。

HTTP API では、次のメトリクスがサポートされています。ルートレベルのメトリクスを Amazon CloudWatch に書き込む詳細メトリクスを有効にすることもできます。

メトリクス	説明
4xx	指定された期間に取得されたクライアント側エラーの数。

メトリクス	説明
5xx	指定された期間に取得されたサーバー側エラーの数。
カウント	指定期間内の API リクエストの合計数。
IntegrationLatency	API Gateway がバックエンドにリクエストを中継してから、バックエンドからレスポンスを受け取るまでの時間。
レイテンシー	API Gateway がクライアントからリクエストを受け取ってから、クライアントにレスポンスを返すまでの時間。レイテンシーには、統合のレイテンシーおよびその他の API Gateway のオーバーヘッドが含まれます。
DataProcessed	処理されたデータの量 (バイト単位)。

API Gateway のメトリクスをフィルタリングするには、次の表のディメンションを使用できます。

ディメンション	説明
ApiId	指定した API ID で API の API Gateway メトリクスをフィルタリングします。
ApiId、ステージ	指定した API ID とステージ ID で API ステージの API Gateway メトリクスをフィルタリングします。

ディメンション	説明
ApiId、メソッド、リソース、ステージ	指定した API ID、ステージ ID、リソースパス、ルート ID で API メソッドの API Gateway メトリクスをフィルタリングします。 詳細な CloudWatch のメトリクスを明示的に有効にしない限り、API Gateway はこれらのメトリクスを送信しません。そのためには、API Gateway V2 REST API の UpdateStage アクションを呼び出して、`detailedMetricsEnabled` プロパティを `true` に更新します。update-stage AWS CLI コマンドを呼び出して `DetailedMetricsEnabled` プロパティを `true` に更新することもできます。このようなメトリクスを有効にすることで、アカウントに追加料金が発生します。詳細については、Amazon CloudWatch 料金表をご覧ください。

HTTP API のログ記録の設定

ログ記録を有効にして CloudWatch Logs にログを記録することができます。ログ変数を使用して、ログの内容をカスタマイズできます。

HTTP API のログ記録を有効にするには、以下を実行する必要があります。

1. ログ記録を有効にするために必要なアクセス許可がユーザーにあることを確認します。
2. CloudWatch Logs ロググループを作成します。
3. API のステージの CloudWatch Logs ロググループの ARN を指定します。

ログ記録を有効にするアクセス許可

API のログ記録を有効にするには、 ユーザーに次のアクセス許可が必要です。

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": "*"
        }
    ]
}
```

ロググループを作成し、HTTP API のログ記録を有効にする

ロググループを作成し、AWS Management Console または AWS CLI を使用してアクセスログ記録を有効化できます。

AWS Management Console

1. ロググループを作成します。

   コンソールを使用してロググループを作成する方法については、Amazon CloudWatch Logs ユーザーガイドの「ロググループとログストリームの操作」を参照してください。
2. https://console.aws.amazon.com/apigateway で API Gateway コンソールにサインインします。
3. HTTP API を選択します。
4. プライマリナビゲーションパネルの [Monitor] (モニタリング) タブで、[Logging] (ログ記録) を選択します。
5. ログ記録を有効にするステージを選択し、[Select] (選択) を選択します。
6. [Edit] (編集) を選択してアクセスログを有効にします。
7. [Access logging] (アクセスのログ記録) を有効にし、CloudWatch Logs を入力して、ログ形式を選択します。
8. [Save] を選択します。

AWS CLI

以下の AWS CLI コマンドはロググループを作成します。

```
aws logs create-log-group --log-group-name my-log-group
```

ログ記録を有効にするには、ロググループの Amazon リソースネーム (ARN) が必要です。ARN 形式は、arn:aws:logs:*region*:*account-id*:log-group:*log-group-name* です。

次の AWS CLI コマンドは、HTTP API の `$default` ステージのログ記録を有効にします。

```
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"}'
```

ログ形式の例

いくつかの一般的なアクセスログ形式の例は API Gateway コンソールで使用できます。それらの例を以下に示します。

- CLF (Common Log Format):

```
$context.identity.sourceIp - - [$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 (カンマ区切り値):

```
$context.identity.sourceIp,$context.requestTime,$context.httpMethod,
$context.routeKey,$context.protocol,$context.status,$context.responseLength,
$context.requestId,$context.extendedRequestId
```

HTTP API アクセスログのカスタマイズ

次の変数を使用して、HTTP API のアクセスログをカスタマイズできます。HTTP API のアクセスログの詳細については、「HTTP API のログ記録の設定」を参照してください。

パラメータ	説明
$context.accountId	API 所有者の AWS アカウント ID。
$context.apiId	API Gateway が API に割り当てる識別子。
$context.authorizer.claims.*property*	メソッドの呼び出し元が正常に認証された後で JSON ウェブトークン (JWT) から返されるクレームのプロパティ ($context.authorizer.claims.username など)。詳細については、「JWT オーソライザーを使用した HTTP API へのアクセスの制御」を参照してください。 **Note** $context.authorizer.claims を呼び出すと NULL が返されます。
$context.authorizer.error	オーソライザーから返されたエラーメッセージ。
$context.authorizer.principalId	Lambda オーソライザーから返されたプリンシパルユーザー ID。

$context.authorizer.*property*	API Gateway Lambda オーソライザー関数から返された context マップの指定されたキー/値ペアの値。たとえば、オーソライザーが次の context マップを返すとします。
	
	`"context" : {`
	`  "key": "value",`
	`  "numKey": 1,`
	`  "boolKey": true`
	`}`

パラメータ	説明
	$context.authorizer.key を呼び出すと "value" 文字列が返され、$context.authorizer.numKey を呼び出すと 1 が返され、$context.authorizer.boolKey を呼び出すと true が返されます。
$context.awsEndpointRequestId	x-amz-request-id または x-amzn-requestId ヘッダーからの AWS エンドポイントのリクエスト ID。
$context.awsEndpointRequestId2	x-amz-id-2 ヘッダーからの AWS エンドポイントのリクエスト ID。
$context.customDomain.basePathMatched	受信リクエストが一致した API マッピングのパス。クライアントがカスタムドメイン名を使用して API にアクセスする場合に適用されます。たとえば、クライアントがリクエストを https://api.example.com/v1/orders/1234 に送信し、リクエストがパス v1/orders を持つ API マッピングと一致する場合 、値は v1/orders になります。詳細については、「the section called “API マッピング”」を参照してください。
$context.dataProcessed	処理されたデータの量 (バイト単位)。
$context.domainName	API の呼び出しに使用された完全ドメイン名。これは、受信 Host ヘッダーと同じである必要があります。
$context.domainPrefix	$context.domainName の 1 つ目のラベル。
$context.error.message	API Gateway エラーメッセージを含む文字列。

パラメータ	説明
$context.error.messageString	$context.error.message を引用符で囲んだ値、つまり "$context.error.message" 。
$context.error.responseType	GatewayResponse のタイプ。詳細については、the section called “メトリクス”および the section called “エラーレスポンスをカスタマイズするためのゲートウェイレスポンスのセットアップ”を参照してください。
$context.extendedRequestId	$context.requestId と同等です。
$context.httpMethod	使用される HTTP メソッドです。有効な値には、DELETE、GET、HEAD、OPTIONS、PATCH、POST および PUT があります。
$context.identity.accountId	リクエストに関連付けられた AWS アカウント ID です。IAM 認証を使用するルートでサポートされています。
$context.identity.caller	リクエストに署名した発信者のプリンシパル ID。IAM 認証を使用するルートでサポートされています。

パラメータ	説明
`$context.identity.cognitoAuthenticationProvider`	リクエストを行う発信者が使用する Amazon Cognito 認証プロバイダーのカンマ区切りのリスト。リクエストが Amazon Cognito 認証情報で署名されている場合にのみ使用できます。 たとえば、Amazon Cognito ユーザープールのアイデンティティの場合、`cognito-idp.`*`region`*`.amazonaws.com/`*`user_pool_id`*`,cognito-idp.`*`region`*`.amazonaws.com/`*`user_pool_id`*`:CognitoSignIn:`*`token subject claim`* 詳しくは、Amazon Cognito デベロッパーガイドの「Amazon Cognito ID プール (フェデレーティッド ID)」を参照してください。
`$context.identity.cognitoAuthenticationType`	リクエストを行う発信者の Amazon Cognito 認証タイプ。リクエストが Amazon Cognito 認証情報で署名されている場合にのみ使用できます。有効な値は、認証されたアイデンティティ`authenticated` および認証されていないアイデンティティ`unauthenticated` です。
`$context.identity.cognitoIdentityId`	リクエストを行う発信者の Amazon Cognito ID。リクエストが Amazon Cognito 認証情報で署名されている場合にのみ使用できます。
`$context.identity.cognitoIdentityPoolId`	リクエストを行う発信者の Amazon Cognito ID プール ID。リクエストが Amazon Cognito 認証情報で署名されている場合にのみ使用できます。
`$context.identity.principalOrgId`	AWS 組織 ID。IAM 認証を使用するルートでサポートされています。

パラメータ	説明
$context.identity.clientCert.clientCertPem	クライアントが相互 TLS 認証中に提示した PEM エンコードされたクライアント証明書。相互 TLS が有効なカスタムドメイン名を使用してクライアントが API にアクセスすると、アクセスログに存在します。
$context.identity.clientCert.subjectDN	クライアントが提示する証明書のサブジェクトの識別名。相互 TLS が有効なカスタムドメイン名を使用してクライアントが API にアクセスすると、アクセスログに存在します。
$context.identity.clientCert.issuerDN	クライアントが提示する証明書の発行者の識別名。相互 TLS が有効なカスタムドメイン名を使用してクライアントが API にアクセスすると、アクセスログに存在します。
$context.identity.clientCert.serialNumber	証明書のシリアル番号。相互 TLS が有効なカスタムドメイン名を使用してクライアントが API にアクセスすると、アクセスログに存在します。
$context.identity.clientCert.validity.notBefore	証明書が無効になる前の日付。相互 TLS が有効なカスタムドメイン名を使用してクライアントが API にアクセスすると、アクセスログに存在します。
$context.identity.clientCert.validity.notAfter	証明書が無効になった日付。相互 TLS が有効なカスタムドメイン名を使用してクライアントが API にアクセスすると、アクセスログに存在します。
$context.identity.sourceIp	API Gateway エンドポイントへのリクエストを実行する即時 TCP 接続のソース IP アドレス。

パラメータ	説明
$context.identity.user	リソースアクセスに対して許可されるユーザーのプリンシパル識別子。IAM 認証を使用するルートでサポートされています。
$context.identity.userAgent	API 発信者の User-Agent ヘッダー。
$context.identity.userArn	認証後に識別された有効ユーザーの Amazon リソースネーム (ARN) です。IAM 認証を使用するルートでサポートされています。詳細については、「https://docs.aws.amazon.com/IAM/latest/UserGuide/id_users.html」を参照してください。
$context.integration.error	統合から返されたエラーメッセージ。$context.integrationErrorMessage と同等です。
$context.integration.integrationStatus	Lambda プロキシ統合の場合、バックエンドの Lambda 関数コードからではなく、AWS Lambda から返されるステータスコード。
$context.integration.latency	統合レイテンシー (ミリ秒)。$context.integrationLatency と同等です。
$context.integration.requestId	AWS エンドポイントのリクエスト ID。$context.awsEndpointRequestId と同等です。
$context.integration.status	統合から返されたステータスコード。Lambda プロキシ統合では、これは Lambda 関数コードから返されたステータスコードです。
$context.integrationErrorMessage	統合エラーメッセージを含む文字列。
$context.integrationLatency	統合レイテンシー (ミリ秒)。

パラメータ	説明
$context.integrationStatus	Lambda プロキシ統合の場合、このパラメータはバックエンド Lambda 関数からではなく、AWS Lambda から返されたステータスコードを表します。
$context.path	リクエストパス。たとえば、/{stage}/root/child と指定します。
$context.protocol	HTTP/1.1 などのリクエストプロトコル。 **Note** API Gateway API は HTTP/2 リクエストを受け入れることができますが、API Gateway は HTTP/1.1 を使用してバックエンド統合にリクエストを送信します。その結果、クライアントが HTTP/2 を使用するリクエストを送信した場合でも、リクエストプロトコルは HTTP/1.1 として記録されます。
$context.requestId	API Gateway が API リクエストに割り当てる ID。
$context.requestTime	CLF 形式の要求時間 (dd/MMM/yyyy:HH:mm:ss +-hhmm)。
$context.requestTimeEpoch	エポック形式のリクエスト時間。
$context.responseLatency	レスポンスレイテンシー (ミリ秒)。
$context.responseLength	レスポンスペイロードの長さ (バイト単位)。
$context.routeKey	API リクエストのルートキー (例:/pets)。

パラメータ	説明
$context.stage	API リクエストのデプロイステージ (beta、prod など)。
$context.status	メソッドレスポンスのステータス。

HTTP API に関する問題のトラブルシューティング

以下のトピックでは、HTTP API の使用時に発生する可能性のあるエラーや問題のトラブルシューティングに関するアドバイスを提供します。

トピック

- HTTP API Lambda 統合に関する問題のトラブルシューティング
- HTTP API JWT オーライザーに関する問題のトラブルシューティング

HTTP API Lambda 統合に関する問題のトラブルシューティング

次に、HTTP API で AWS Lambda の統合 を使用するときに発生する可能性のあるエラーや問題に関するトラブルシューティングのアドバイスを示します。

問題: Lambda 統合のある API が **{"message":"Internal Server Error"}** を返します

内部サーバーエラーのトラブルシューティングを行うには、ログ形式に $context.integrationErrorMessage ログ記録変数を追加し、HTTP API のログを表示します。これを達成するには、次の操作を行います。

を使用してロググループを作成するにはAWS Management Console

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. [ロググループ] を選択します。
3. [ロググループの作成] を選択します。
4. ロググループ名を入力し、[作成] を選択します。

5. ロググループの Amazon リソースネーム (ARN) を書き留めます。ARN 形式は、arn:aws:logs:*region*: *account-id*:log-group:*log-group-name* です。HTTP API のアクセスのログ記録を有効にするには、ロググループ ARN が必要です。

`$context.integrationErrorMessage` ロギング変数を追加するには

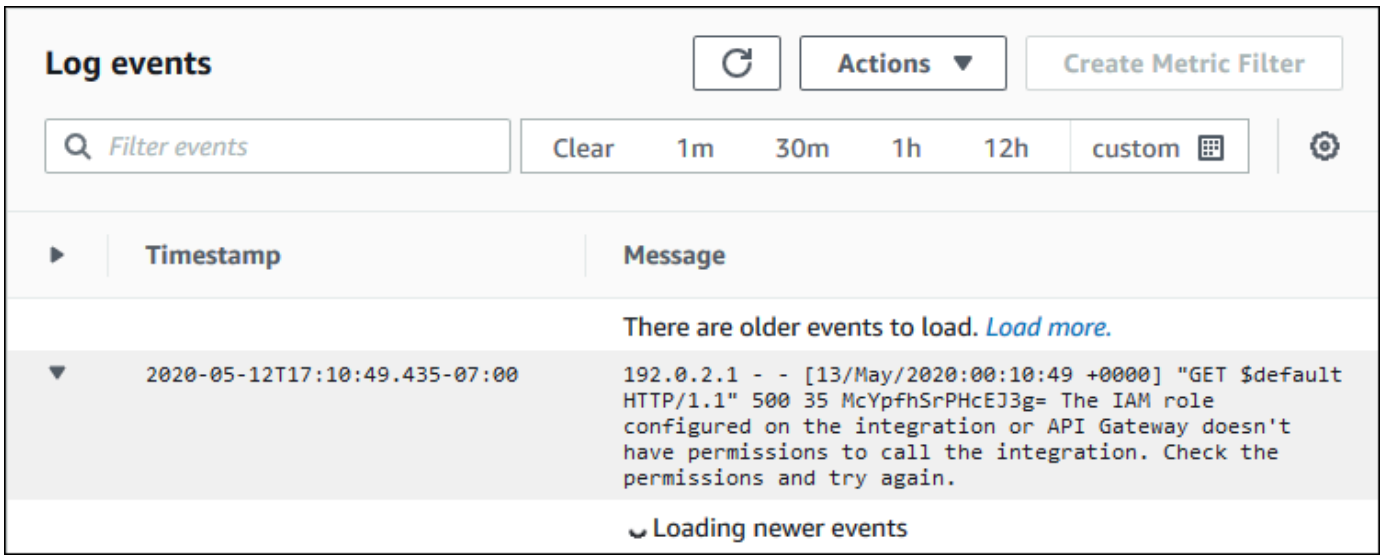
1. https://console.aws.amazon.com/apigateway で API Gateway コンソールにサインインします。
2. HTTP API を選択します。
3. [モニタリング] で、[ログ記録] を選択します。
4. API のステージを選択します。
5. [編集] を選択し、アクセスログを有効にします。
6. [Log destination] (ログの送信先) で、前のステップで作成したロググループの ARN を入力します。
7. [ログの形式] で、[CLF] を選択します。API Gateway はサンプルのログ形式を作成します。
8. ログ形式の末尾に `$context.integrationErrorMessage` を追加します。
9. [保存] を選択します。

API のログを表示するには

1. ログを生成します。ブラウザまたは `curl` を使用して API を呼び出します。

```
$curl https://api-id.execute-api.us-west-2.amazonaws.com/route
```

2. https://console.aws.amazon.com/apigateway で API Gateway コンソールにサインインします。
3. HTTP API を選択します。
4. [モニタリング] で、[ログ記録] を選択します。
5. ロギングを有効にした API のステージを選択します。
6. [CloudWatch のログを表示] を選択します。
7. 最新のログストリームを選択して、HTTP API のログを表示します。
8. ログエントリは、以下のようになります。



ログ形式に `$context.integrationErrorMessage` を追加したため、問題の概要を示すエラーメッセージがログに表示されます。

Lambda 関数コードに問題があることを示す別のエラーメッセージがログに含まれている可能性があります。その場合は、Lambda 関数コードを確認し、Lambda 関数が必要な形式でレスポンスを返すことを確認します。ログにエラーメッセージが含まれていない場合は、トラブルシューティングに役立つ詳細情報のために、ログ形式に `$context.error.message` および `$context.error.responseType` を追加します。

この場合、API Gateway には Lambda 関数を呼び出すために必要なアクセス許可がないことがログに示されます。

API Gateway コンソールで Lambda 統合を作成すると、API Gateway は Lambda 関数を呼び出すためのアクセス許可を自動的に設定します。AWS CLI、AWS CloudFormation、または SDK を使用して Lambda 統合を作成する場合、API Gateway に関数を呼び出すアクセス許可を付与する必要があります。次の例の AWS CLI コマンドは、Lambda 関数を呼び出すためのアクセス許可を複数の異なる HTTP API ルートに付与します。

Example 例 — HTTP API の **`$default`** ステージと **`$default`** ルートの場合

```
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 例 — HTTP API の **prod** ステージと **test** ルートの場合

```
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"
```

Lambda コンソールの [Permissions (アクセス権限)] タブで関数ポリシーを確認します。

API をもう一度呼び出してみてください。Lambda 関数のレスポンスが表示されるはずです。

HTTP API JWT オーライザーに関する問題のトラブルシューティング

次に、HTTP API で JSON ウェブトークン (JWT) オーソライザーを使用するときに発生する可能性のあるエラーおよび問題のトラブルシューティングに関するアドバイスを示します。

問題: API が **401 {"message":"Unauthorized"}** を返します

API からのレスポンスで www-authenticate ヘッダーを確認します。

次のコマンドは、curl を使用して、$request.header.Authorization を ID ソース として使用する JWT オーソライザーを使用して API にリクエストを送信します。

```
$curl -v -H "Authorization: token" https://api-id.execute-api.us-
west-2.amazonaws.com/route
```

API からのレスポンスには www-authenticate ヘッダーが含まれます。

```
...
< 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"}}
```

この場合、www-authenticate ヘッダーには、有効な対象者に対してトークンが発行されなかったことが示されます。API Gateway でリクエストを承認するには、JWT の aud または client_id クレームが、オーソライザー用に設定されている対象者のエントリの 1 つと一致する必要があります。API Gateway は、aud が存在しない場合にのみ、client_id を検証します。aud と client_id の両方が存在する場合、API Gateway は aud を評価します。

また、JWTをデコードし、APIが必要とする発行者、対象者、スコープと一致することを確認することもできます。ウェブサイトの jwt.io は、ブラウザで JWT をデバッグすることができます。OpenID Foundation では、JWT で使用するライブラリのリストが管理されます。

JWT オーソライザーの詳細については、「JWT オーソライザーを使用した HTTP API へのアクセスの制御」を参照してください 。

WebSocket API の操作

API Gateway の WebSocket API は、バックエンドの HTTP エンドポイント、Lambda 関数、またはその他の AWS のサービスを使用して統合されている WebSocket ルートのコレクションです。API Gateway 機能を使用すると、作成から本番稼働 API のモニタリングまで、API ライフサイクルのあらゆる側面を支援できます。

API Gateway WebSocket API は双方向です。クライアントはサービスにメッセージを送信し、サービスは個別にクライアントにメッセージを送信できます。この双方向動作により、クライアントが明示的な要求を行う必要がなく、サービスがクライアントにデータをプッシュできるため、より豊かなクライアント/サービスの対話が実現します。WebSocket API は、チャットアプリケーション、コラボレーションプラットフォーム、マルチプレイヤーゲーム、金融取引プラットフォームなどのリアルタイムアプリケーションでよく使用されます。

開始するためのサンプルアプリについては、「チュートリアル: WebSocket API、Lambda、DynamoDB を使用したサーバーレスチャットアプリケーションの構築」を参照してください。

このセクションでは、API Gateway を使用して WebSocket API を開発、公開、保護、監視する方法を学習できます。

トピック

- API Gateway での WebSocket API について
- API ゲートウェイでの WebSocket API の開発
- 顧客が呼び出す WebSocket API の発行
- WebSocket API の保護
- WebSocket API のモニタリング

API Gateway での WebSocket API について

API Gateway では、AWS のサービス (Lambda や DynamoDB など) または HTTP エンドポイントのステートフルフロントエンドとして WebSocket API を作成できます。WebSocket API は、クライアントアプリから受信するメッセージのコンテンツに基づいて、バックエンドを呼び出します。

WebSocket API は、リクエストを受け取って応答する REST API とは異なり、クライアントアプリとバックエンド間の双方向通信をサポートします。バックエンドは、接続されたクライアントにコールバックメッセージを送信できます。

WebSocket API では、受信する JSON メッセージは設定したルートに基づいてバックエンド統合に転送されます (JSON 以外のメッセージは、設定した `$default` ルートに転送されます)。

ルートにはルートキーが含まれます。これは、ルート選択式が評価されたときに予期される値です。属性 `routeSelectionExpression` は、API レベルで定義されます。メッセージペイロードに存在することが予期される JSON プロパティを指定します。ルート選択式の詳細については、「the section called “”」を参照してください。

たとえば、JSON メッセージに `action` プロパティが含まれていて、このプロパティに基づいてさまざまなアクションを実行する場合、ルート選択式は `${request.body.action}` のようになります。ルーティングテーブルでは、このテーブルで定義したカスタムルートキーの値に対して、`action` プロパティの値を一致させることによって、実行するアクションを指定します。

3 つの事前定義されたルート (`$connect`、`$disconnect`、および `$default`) を使用できます。さらに、カスタムルートを作成することができます。

- API Gateway は、クライアントと WebSocket API 間の永続的な接続が開始されたときに、`$connect` ルートを呼び出します。
- API Gateway は、クライアントまたはサーバーが API から切断したときに、`$disconnect` ルートを呼び出します。
- API Gateway は、一致するルートが見つかった場合、メッセージに対してルート選択式が評価された後でカスタムルートを呼び出します。この一致により、呼び出される統合が決まります。
- ルート選択式をメッセージに対して評価できない場合や、一致するルートが見つからない場合、API Gateway は `$default` ルートを呼び出します。

`$connect` および `$disconnect` ルートの詳細については、「the section called “接続されたユーザーおよびクライアントアプリの管理”」を参照してください。

`$default` ルートおよびカスタムルートの詳細については、「the section called “バックエンド統合の呼び出し”」を参照してください。

バックエンドサービスは、接続されたクライアントアプリにデータを送信できます。詳細については、「the section called “バックエンドサービスから接続されたクライアントへのデータの送信”」を参照してください。

接続されたユーザーおよびクライアントアプリの管理: $connect ルートおよび $disconnect ルート

トピック

$connect ルート

クライアントアプリは、WebSocket アップグレードリクエストを送信して WebSocket API に接続します。リクエストが成功すると、接続が確立されている間に $connect ルートが実行されます。

WebSocket 接続はステートフルな接続であるため、$connect ルートのみで認証を設定できます。AuthN/AuthZ は接続時にのみ実行されます。

$connect ルートに関連付けられている統合の実行が完了するまで、アップグレードリクエストは保留中になり、実際の接続は確立されません。$connect リクエストが失敗した場合 (AuthN/AuthZ の障害や統合の障害など)、接続は行われません。

> **Note**
>
> $connect で承認が失敗した場合、接続は確立されず、クライアントは 401 または 403 レスポンスを受け取ります。

$connect の統合の設定はオプションです。次の場合は $connect 統合の設定を検討してください。

- Sec-WebSocket-Protocol フィールドを使用して、クライアントがサブプロトコルを指定できるようにする。サンプルコードについては、「WebSocket サブプロトコルを必要とする $connect ルートの設定」を参照してください。
- クライアントが接続したときに通知を受ける。
- 接続をスロットリングする、または接続するユーザーを管理する。
- バックエンドで、コールバック URL を使用してメッセージをクライアントに送信する。
- 各接続 ID およびその他の情報をデータベース (例: Amazon DynamoDB) に保存する。

$connect ルートからの接続情報の受け渡し

プロキシ統合と非プロキシ統合の両方を使用して、$connect ルートからデータベースまたは他の AWS のサービス に情報を渡すことができます。

プロキシ統合を使用して接続情報を渡すには

イベントでは、Lambda プロキシ統合から接続情報にアクセスできます。別の AWS のサービス または AWS Lambda 関数を使用して接続に投稿します。

次の Lambda 関数は、requestContext オブジェクトを使用して接続 ID、ドメイン名、ステージ名、およびクエリ文字列を記録する方法を示しています。

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}
```

非プロキシ統合を使用して接続情報を渡すには

- 非プロキシ統合により接続情報にアクセスできます。統合リクエストを設定し、WebSocket API リクエストテンプレートを提供します。以下の、Velocity Template Language (VTL) マッピングテンプレートは、統合リクエストを提供します。このリクエストは、以下の詳細を非プロキシ統合に送信します。

  - 接続 ID
  - ドメイン名
  - ステージ名
  - パス
  - ヘッダー
  - クエリ文字列

  このリクエストは、接続 ID、ドメイン名、ステージ名、パス、ヘッダー、クエリ文字列を非プロキシ統合に送信します。

  ```
  {
      "connectionId": "$context.connectionId",
      "domain": "$context.domainName",
      "stage": "$context.stage",
      "params": "$input.params()"
  }
  ```

  データ変換の設定の詳細については、「the section called “データ変換”」を参照してください。

  統合リクエストを完了するには、統合レスポンスに `StatusCode: 200` を設定します。統合レスポンスの詳しい設定方法については、「API Gateway コンソールを使用した統合レスポンスの設定」を参照してください。

`$disconnect` ルート

`$disconnect` ルートは、接続を閉じた後に実行されます。

接続は、サーバーまたはクライアントによって閉じることができます。接続が実行されると、接続がすでに閉じられているため、$disconnect がベストエフォート型のイベントです。API Gateway は、統合に $disconnect イベントを配信するために最善を尽くしますが、配信を保証することはできません。

バックエンドは、@connections API を使用して切断を開始できます。詳細については、「the section called “バックエンドサービスでの @connections コマンドの使用”」を参照してください。

バックエンド統合の呼び出し: **$default** ルートおよびカスタムルート

トピック

- ルートを使用したメッセージの処理
- $default ルート
- カスタムルート
- API Gateway WebSocket API 統合を使用したビジネスロジックへの接続
- WebSocket API と REST API の重要な相違点

ルートを使用したメッセージの処理

API Gateway WebSocket API では、クライアントからバックエンドサービスに、またはその逆にメッセージを送信できます。HTTP のリクエスト/レスポンスモデルとは異なり、WebSocket ではクライアントがアクションを実行することなく、バックエンドがクライアントにメッセージを送信できます。

メッセージの形式は JSON または JSON 以外とすることができます。ただし、メッセージの内容によっては、JSON メッセージのみを特定の統合にルーティングできます。JSON 以外のメッセージは、$default ルートによってバックエンドにパススルーされます。

> **Note**
>
> API Gateway は最大 128 KB までのメッセージペイロードをサポートし、最大フレームサイズは 32 KB です。メッセージが 32 KB を超えた場合は、それぞれが 32 KB 以下の複数のフレームに分割する必要があります。大きなメッセージ (またはフレーム) が受信された場合、接続は 1009 コードで閉じられます。
> 現時点では、バイナリペイロードはサポートされていません。バイナリフレームが受信された場合、接続は 1003 コードで閉じられます。ただし、バイナリペイロードはテキストに変換できます。「the section called “バイナリメディアタイプ”」を参照してください。

API Gateway の WebSocket API では、メッセージの内容に基づいて JSON メッセージをルーティングし、特定のバックエンドサービスを実行できます。クライアントが WebSocket 接続経由でメッセージを送信すると、これが WebSocket API に対するルートリクエストになります。リクエストが、API Gateway の対応するルートキーを持つルートと照合されます。WebSocket API のルートリクエストは、API Gateway コンソールで、または AWS CLI もしくは AWS SDK を使用してセットアップできます。

> **Note**
>
> AWS CLI および AWS SDK で、統合を作成する前または後にルートを作成できます。現在のところ、コンソールは統合の再利用をサポートしていないため、最初にルートを作成してから、そのルートの統合を作成する必要があります。

統合リクエストを進める前にルートリクエストの検証を実行するよう API Gateway を設定できます。検証に失敗すると、API Gateway はバックエンドを呼び出さずにリクエストを失敗させ、次のような "Bad request body" ゲートウェイレスポンスをクライアントに送信し、CloudWatch Logs で検証の失敗を発行します。

```
{"message" : "Bad request body", "connectionId": "{connectionId}", "messageId":
 "{messageId}"}
```

これによりバックエンドへの不要な呼び出しが減り、API のその他の要件に集中することができます。

また、API のルートに対するルートのレスポンスを定義し、双方向通信を有効にすることもできます。ルートレスポンスは、特定のルートの統合の完了時にどのようなデータがクライアントに送信されるかを示します。たとえば、クライアントがレスポンスを受信せずにバックエンドにレスポンスを送信するようにする場合 (単方向通信)、ルートのレスポンスを定義する必要はありません。ただし、ルートレスポンスのルートを提供しない場合、API Gateway は統合の結果に関する情報をクライアントに送信しません。

`$default` ルート

すべての API Gateway WebSocket API は、`$default` ルートを持つことができます。これは、次の方法で使用できる特殊なルーティング値です。

- これを定義されたルートキーと一緒に使用し、定義されたいずれのキーとも一致しない受信メッセージの「フォールバック」ルート (たとえば、特定のエラーメッセージを返す汎用モック統合) を指定できます。
- 定義されたルートキーなしでこれを使用し、バックエンドコンポーネントにルーティングを委任するプロキシモデルを指定できます。
- これを使用して、JSON 以外のペイロードのルートを指定できます。

カスタムルート

メッセージの内容に基づいて特定の統合を呼び出す場合、カスタムルートを作成してこれを行うことができます。

カスタムルートは、指定されたルートキーと統合を使用します。受信メッセージにプロパティ JSON プロパティが含まれていて、そのプロパティがルートキーの値に一致する値に評価される場合、API Gateway は統合を呼び出します。(詳しくは、the section called “WebSocket API について” を参照してください)。

たとえば、チャットルームアプリケーションを作成するとします。ルート選択式が `$request.body.action` である WebSocket API を作成して開始できます。次に、`joinroom` および `sendmessage` の 2 つのルートを定義できます。クライアントアプリは、次のようなメッセージを送信して `joinroom` ルートを呼び出す場合があります。

```
{"action":"joinroom","roomname":"developers"}
```

また、次のようなメッセージを送信して `sendmessage` ルートを呼び出す場合があります。

```
{"action":"sendmessage","message":"Hello everyone"}
```

API Gateway WebSocket API 統合を使用したビジネスロジックへの接続

API Gateway WebSocket API のルートを設定したら、使用する統合を指定する必要があります。ルートリクエストとルートレスポンスを持つことができるルートと同じように、統合は統合リクエストと統合レスポンスを持つことができます。統合リクエストには、クライアントから受け取ったリクエストを処理するためにバックエンドで予期される情報が含まれます。統合レスポンスには、バックエンドが API Gateway に返すデータが含まれます。このデータは、クライアントに送信するメッセージを作成するために使用される場合があります (ルートレスポンスが定義されている場合)。

統合の設定の詳細については、「the section called “統合”」を参照してください。

WebSocket API と REST API の重要な相違点

WebSocket API の統合は REST API の統合に似ていますが、次のような違いがあります。

- 現時点では、API Gateway コンソールで最初にルートを作成してから、そのルートのターゲットとして統合を作成する必要があります。ただし、API と CLI では、ルートと統合を任意の順序で個別に作成できます。
- 複数のルートに単一の統合を使用できます。たとえば、相互に密接に関連した一連のアクションがある場合は、それらのすべてのルートを単一の Lambda 関数に移動したい場合があります。統合の詳細を複数回定義する代わりに、1 回指定して、関連する各ルートに割り当てることができます。

> **Note**
>
> 現在のところ、コンソールは統合の再利用をサポートしていないため、最初にルートを作成してから、そのルートの統合を作成する必要があります。
> AWS CLI、AWS SDK では、ルートのターゲットを "integrations/*{integration-id}*" の値に設定して統合を再利用できます。ここで *{integration-id}*" は、ルートに関連付ける統合の一意の ID です。

- API Gateway は、ルートと統合で使用できる複数の選択式を提供します。入力テンプレートまたは出力マッピングを選択するために、コンテンツタイプに依存する必要はありません。ルート選択式の場合と同様に、API Gateway で評価される選択式を定義して、適切な項目を選択できます。それらのすべては、一致するテンプレートが見つからない場合に、$default テンプレートにフォールバックされます。
  - 統合リクエストで、テンプレート選択式は $request.body.*<json_path_expression>* および静的な値をサポートしています。
  - 統合レスポンスで、テンプレート選択式は $request.body.*<json_path_expression>*、$integration.response.statuscode、$integ および静的な値をサポートします。

リクエストとレスポンスが同期的に送信される HTTP プロトコルでは、通信は基本的に一方向です。WebSocket プロトコルでは、通信は双方向です。レスポンスは非同期であり、必ずしもクライアントのメッセージの送信と同じ順序でクライアントによって受信される必要はありません。さらに、バックエンドはクライアントにメッセージを送信できます。

> **Note**
>
> AWS_PROXY または LAMBDA_PROXY 統合を使用するように設定されたルートでは、通信は一方向で、API Gateway は自動的にルートレスポンスにバックエンドレスポンスをパススルーしません。たとえば、LAMBDA_PROXY 統合の場合は、 Lambda 関数が返す本文はクライアントに返されません。クライアントが統合レスポンスを受信するには、ルートレスポンスを定義して、双方向通信を可能にする必要があります。

バックエンドサービスから接続されたクライアントへのデータの送信

API Gateway WebSocket API は、バックエンドサービスから接続されたクライアントにデータを送信するために、次の方法を提供します。

- 統合はレスポンスを送信できます。これは、定義したルートレスポンスでクライアントに返されます。
- @connections API を使用して POST リクエストを送信することができます。詳細については、「the section called “バックエンドサービスでの @connections コマンドの使用”」を参照してください。

API Gateway での WebSocket 選択式

トピック

- ルートレスポンス選択式
- API キー選択式
- API マッピング選択式
- WebSocket 選択式の概要

API Gateway はリクエストとレスポンスのコンテキストを評価し、キーを生成する方法として、選択式を使用します。次に、このキーを使用して、通常は API デベロッパーから提供される使用可能な値のセットから選択できます。サポートされている正確な変数のセットは、式によって異なります。各式については、以下で詳しく説明します。

すべての式では、言語は一連のルールに従います。

- 変数には "$" が前に付けられます。

- 中かっこを使用して、変数の境界を明示的に定義できます (例: "${request.body.version}-beta")。
- 複数の変数がサポートされていますが、評価は 1 回しか発生しません (繰り返しの評価は行われません)。
- ドル記号 ($) を使って "\" をエスケープすることができます。これは、予約された $default キー (例: "\$default") にマッピングされる式を定義するときに最も便利です。
- 場合によっては、パターン形式が必要です。この場合、式は "/" のようにスラッシュ ("/2\d\d/") で囲み、2*XX* のステータスコードに合わせる必要があります。

ルートレスポンス選択式

ルートレスポンスは、バックエンドからクライアントにレスポンスをモデル化するために使用されます。WebSocket API の場合、ルートレスポンスはオプションです。定義された場合、WebSocket メッセージを受信したときにレスポンスをクライアントに返す必要があることを API Gateway に指定します。

ルートレスポンス選択式の評価により、ルートレスポンスキーが生成されます。最終的に、このキーを使用して、API に関連付けられたいずれかの RouteResponses から選択されます。ただし、現在サポートされているのは $default キーのみです。

API キー選択式

この式は、クライアントが有効な API キーを提供した場合のみ、特定のリクエストを続行する必要があるとサービスが決定したときに評価されます。

現在サポートされている値は、$request.header.x-api-key および $context.authorizer.usageIdentifierKey の 2 つのみです。

API マッピング選択式

この式は、カスタムドメインを使用してリクエストが実行されたときに選択する API ステージを決定するために評価されます。

現在、サポートされている値は $request.basepath のみです。

WebSocket 選択式の概要

次の表は、WebSocket API の選択式のユースケースをまとめたものです。

選択式	評価結果のキー	コメント	ユースケースの例
Api.RouteSelectionExpression	Route.RouteKey	$defaul はキャッチオールルートとしてサポートされています。	クライアントリクエストのコンテキストに基づいて Route WebSocket メッセージをルーティングします。
Route.ModelSelectionExpression	Route.RequestModels のキー	オプション。 非プロキシ統合に対して指定されている場合、モデル検証が発生します。	同じルート内で動的にリクエスト検証を実行します。

選択式	評価結果のキー	コメント	ユースケースの例
		$defaul は キャッチオールとしてサポートされています。	

選択式	評価結果のキー	コメント	ユースケースの例
Integration.TemplateSelectionExpression	Integration.RequestTemplates のキー	オプション。 受信ペイロードを操作するための非プロキシ統合に提供できます。 ${reque .body.j nPath} および静的な値がサポートされています。 $defaul は キャッチオールとしてサポー	リクエストの動的プロパティに基づいて、呼び出し元のリクエストを操作します。

選択式	評価結果のキー	コメント	ユースケースの例
		トされています。	

選択式	評価結果のキー	コメント	ユースケースの例
`Integration.IntegrationResponseSelectionExpression`	`IntegrationResponse.IntegrationResponseKey`	オプション。非プロキシ統合に提供される場合があります。 エラーメッセージのパターン一致 (Lambda から) またはステータスコード (HTTP 統合から) として動作します。 `$defau` は、成功した	バックエンドからのレスポンスを操作します。 バックエンドの動的なレスポンスに基づいて発生するアクションを選択します (特定のエラーを明確に処理)。

選択式	評価結果のキー	コメント	ユースケースの例
		レスポンスのキャッチオールとして動作する非プロキシ統合で必要です。	

選択式	評価結果のキー	コメント	ユースケースの例
IntegrationResponse.TemplateSelectionExpression	IntegrationResponse.ResponseTemplates のキー	オプション。非プロキシ統合に提供される場合があります。 $default はサポートされています。	場合によっては、レスポンスの動的プロパティは、同じルートおよび関連する統合内で別の変換を指示します。 ${request.body.jsonPath} 、${integration.response.statuscode} 、${integration.response.header.headerName} 、${integration.response.multivalueheader.headerName} 、

選択式	評価結果のキー	コメント	ユースケースの例
			および静的な値がサポートされています。 `$default` はキャッチオールとしてサポートされています。

選択式	評価結果のキー	コメント	ユースケースの例
Route.RouteResponseSelectionExpression	RouteResponse.RouteResponseKey	WebSocルートの双方向通信を開始するために提供する必要があります。 現在、この値は$defaulのみに制限されています。	
RouteResponse.ModelSelectionExpression	RouteResponse.RequestModels のキー	現在サポートされていません。	

API ゲートウェイでの WebSocket API の開発

このセクションでは、API Gateway API の開発中に必要な API Gateway 機能について詳しく説明します。

API Gateway API の開発中、API の特性をいくつか決定することになります。これらの特性は API のユースケースによって異なります。たとえば、API を特定のクライアントのみが呼び出せるようにしたり、すべてのユーザーが利用できるようにしたりできます。API コールで Lambda 関数の実行、データベースクエリの作成やアプリケーションの呼び出しができます。

トピック

API Gateway での WebSocket API のデプロイ

WebSocket API は、API Gateway コンソールで AWS CLI create-api コマンドを使用するか、AWS SDK で `CreateApi` コマンドを使用して作成できます。以下の手順では、新しい WebSocket API を作成する方法を示しています。

> **Note**
>
> WebSocket API は、TLS 1.2 のみをサポートします。以前の TLS バージョンはサポートされていません。

AWS CLI コマンドを使用して WebSocket API を作成する

AWS CLI を使用して WebSocket API を作成するには、次の例に示すように create-api コマンドを呼び出す必要があります。これにより、`$request.body.action` ルート選択式で API が作成されます。

```
aws apigatewayv2 --region us-east-1 create-api --name "myWebSocketApi3" --protocol-type
 WEBSOCKET --route-selection-expression '$request.body.action'
```

出力例:

```
{
    "ApiKeySelectionExpression": "$request.header.x-api-key",
    "Name": "myWebSocketApi3",
    "CreatedDate": "2018-11-15T06:23:51Z",
    "ProtocolType": "WEBSOCKET",
    "RouteSelectionExpression": "'$request.body.action'",
    "ApiId": "aabbccddee"
}
```

API Gateway コンソールを使用した WebSocket API の作成

WebSocket プロトコルを選択し、API に名前を付けることで、コンソールで WebSocket API を作成できます。

> **Important**
>
> API を作成した後で、選択したプロトコルを変更することはできません。WebSocket API を REST API に変換することはできません。その逆も同様です。

API Gateway コンソールを使用して WebSocket API を作成するには

1. API Gateway コンソールにサインインし、[Create API (API の作成)] を選択します。
2. [WebSocket API] で [Build (ビルド)] を選択します リージョンのエンドポイントのみがサポートされます。
3. [API 名] に API の名前を入力します。
4. [ルート選択式] に値を入力します。例えば、`$request.body.action` と指定します。

   ルート選択式の詳細については、「the section called “”」を参照してください。
5. 次のいずれかを行います。
   - ルートのない API を作成するには、[空の API を作成] を選択します。
   - [次へ] を選択して API にルートをアタッチします。

   API を作成したら、ルートをアタッチできます。

WebSocket API のルートの操作

WebSocket API では、受信する JSON メッセージは設定したルートに基づいてバックエンド統合に転送されます (JSON 以外のメッセージは、設定した `$default` ルートに転送されます)。

ルートにはルートキーが含まれます。これは、ルート選択式が評価されたときに予期される値です。属性 `routeSelectionExpression` は、API レベルで定義されます。メッセージペイロードに存在することが予期される JSON プロパティを指定します。ルート選択式の詳細については、「the section called “”」を参照してください。

たとえば、JSON メッセージに `action` プロパティが含まれていて、このプロパティに基づいてさまざまなアクションを実行する場合、ルート選択式は `${request.body.action}` のようになります。ルーティングテーブルでは、このテーブルで定義したカスタムルートキーの値に対して、`action` プロパティの値を一致させることによって、実行するアクションを指定します。

3 つの事前定義されたルート (`$connect`、`$disconnect`、および `$default`) を使用できます。さらに、カスタムルートを作成することができます。

- API Gateway は、クライアントと WebSocket API 間の永続的な接続が開始されたときに、`$connect` ルートを呼び出します。
- API Gateway は、クライアントまたはサーバーが API から切断したときに、`$disconnect` ルートを呼び出します。
- API Gateway は、一致するルートが見つかった場合、メッセージに対してルート選択式が評価された後でカスタムルートを呼び出します。この一致により、呼び出される統合が決まります。
- ルート選択式をメッセージに対して評価できない場合や、一致するルートが見つからない場合、API Gateway は `$default` ルートを呼び出します。

ルート選択式

ルート選択式は、サービスが受信メッセージについてたどるルートを選択するときに評価されます。このサービスでは、`routeKey` が評価された値に完全に一致するときにルートが使用されます。何も一致せず、`$default` ルートキーを持つルートが存在する場合は、そのルートが選択されます。評価された値と一致するルートがなく、`$default` ルートもない場合、サービスからエラーが返されます。WebSocket ベースの API の場合、式の形式は `$request.body.`*`{path_to_body_element}`* である必要があります。

たとえば、次の JSON メッセージを送信するとします。

```
{
    "service" : "chat",
    "action" : "join",
    "data" : {
        "room" : "room1234"
   }
}
```

action プロパティに基づいて、API の動作を選択する必要があります。その場合は、次のルート選択式を定義できます。

```
$request.body.action
```

この例で request.body は、メッセージの JSON ペイロードを参照し、.action が JSONPath 式になります。JSON パス式は request.body の後に使用できますが、結果は文字列化されることに注意してください。たとえば、JSONPath 式は 2 つの要素の配列を返し、文字列 "[item1, item2]" として表示されます。このため、式は配列やオブジェクトではなく値に対して評価することをお勧めします。

静的な値を使用するか、複数の変数を使用できます。以下の表に示しているのは、例と上記のペイロードに対して評価された結果です。

式	評価結果	説明
$request.body.action	join	1 つのラップ解除された変数
${request.body.action}	join	1 つのラップされた変数
${request.body.service}/${r	chat/join	静的な値を持

式	評価結果	説明
`equest.body.action}`		つ複数の変数
`${request.body.action}-${request.body.invalidPath}`	`join-`	JSONPath が見つからない場合、変数は "" として解決されます。
`action`	`action`	静的な値
`\$default`	`$default`	静的な値

評価された結果は、ルートを見つけるために使用されます。一致するルートキーがあるルートがある場合、そのルートがメッセージを処理するために選択されます。一致するルートが見つからなかった場合、API Gateway は `$default` ルートを見つけようとします (利用可能な場合)。`$default` ルートが定義されていない場合、API Gateway はエラーを返します。

API Gateway で WebSocket API のルートを設定する

新しい WebSocket API を初めて作成するときは、3 つの事前定義されたルート (`$connect`、`$disconnect`、`$default`) があります。これらのルートは、コンソール、API、または AWS CLI を使って作成できます。必要に応じて、カスタムルートを作成することができます。詳細については、「the section called “WebSocket API について”」を参照してください。

> **Note**
>
> CLI では、統合を作成する前または後にルートを作成したり、複数のルートで同じ統合を再利用したりできます。

API Gateway コンソールを使用したルートの作成

API Gateway コンソールを使用してルートを作成するには

1. API Gateway コンソールにサインインし、[API]、[Routes (ルート)] の順に選択します。
2. [ルートの作成] を選択します。
3. [ルートキー] に、ルートキー名を入力します。定義済みのルート (`$connect`、`$disconnect`、および`$default`) またはカスタムルートを作成できます。

   > **Note**
   >
   > カスタムルートを作成する場合、ルートキー名に $ プレフィックスは使用しないでください。このプレフィックスは事前定義されたルートのために予約されています。

4. ルートの統合タイプを選択して設定します。詳細については、「the section called “API Gateway コンソールを使用して WebSocket API 統合リクエストを設定する”」を参照してください。

AWS CLI を使用したルートの作成

AWS CLI を使用してルートを作成するには、次の例に示すように、`create-route` を呼び出します。

```
aws apigatewayv2 --region us-east-1 create-route --api-id aabbccddee --route-key
 $default
```

出力例:

```
{
    "ApiKeyRequired": false,
    "AuthorizationType": "NONE",
    "RouteKey": "$default",
    "RouteId": "1122334"
}
```

`$connect` のルートリクエスト設定の指定

API の $connect ルートを設定するときは、以下のオプション設定を使用して、API の認証を有効にできます。詳細については、「the section called “$connect ルート”」を参照してください。

- 認可: 認証が必要ない場合は、NONE を指定できます。それ以外の場合は、以下を指定できます。
  - API へのアクセスを制御するために標準の AWS IAM ポリシーを使用するには、AWS_IAM を指定します。
  - 以前に作成した Lambda オーソライザー関数を指定して API の認証を実装するには、CUSTOM を指定します。オーソライザーは、独自の AWS アカウントまたは別の AWS アカウントに設定できます。Lambda オーソライザーの詳細については、「API Gateway Lambda オーソライザーを使用する」を参照してください。

    > **Note**
    >
    > API Gateway コンソールで、CUSTOM 設定は「the section called “Lambda オーソライザーを設定する (コンソール)”」で説明したオーソライザー関数を設定した後でのみ表示されます。

> **Important**
>
> [認証] 設定は、$connect ルートだけではなく、API 全体に適用されます。$connect ルートは、すべての接続で呼び出されるため、他のルートを保護します。

- [API キーの必要性]: 必要に応じて、API の $connect ルートの API キーを要求できます。API キーを使用量プランと一緒に使用して、API へのアクセスを制御、追跡できます。詳細については、「the section called “使用量プラン”」を参照してください。

API Gateway コンソールを使用して **$connect** ルートリクエストをセットアップする

API Gateway コンソールを使用して WebSocket API の $connect ルートリクエストを設定するには

1. API Gateway コンソールにサインインし、[API]、[Routes (ルート)] の順に選択します。
2. [ルート] で、$connect を選択するか、the section called “API Gateway コンソールを使用したルートの作成” に従って $connect ルートを作成します。
3. [ルートリクエスト設定] セクションで、[編集] を選択します。
4. [承認] では、認証タイプを選択します。
5. $connect ルートで API を必須にするには、[API キーを要求する] を選択します。
6. [Save changes] (変更の保存) をクリックします。

API Gateway で WebSocket API のルートレスポンスを設定する

WebSocket ルートは、双方向または単方向通信に対して設定できます。ユーザーがルートレスポンスを設定しない限り、API Gateway はルートレスポンスにバックエンドレスポンスを渡しません。

> **Note**
>
> WebSocket API の `$default` ルートレスポンスのみを定義できます。統合レスポンスを使用して、バックエンドサービスからのレスポンスを操作できます。詳細については、「the section called “統合レスポンスの概要”」を参照してください。

API Gateway コンソール、AWS CLI、または AWS SDK を使用して、ルートレスポンスとレスポンス選択式を設定できます。

ルートレスポンス選択式の設定の詳細については、「the section called “”」を参照してください。

トピック

- API Gateway コンソールを使用したルートレスポンスの設定
- AWS CLI を使用してルートレスポンスを設定する

API Gateway コンソールを使用したルートレスポンスの設定

WebSocket API を作成し、プロキシ Lambda 関数をデフォルトルートにアタッチしたら、API Gateway コンソールを使用してルートレスポンスを設定できます。

1. API Gateway コンソールにサインインし、`$default` ルートの Lambda 関数の統合で WebSocket API を選択します。
2. [Routes] (ルート) で、`$default` ルートを選択します。
3. [双方向通信を有効にする] を選択します。
4. [API のデプロイ] を選択します。
5. API をステージにデプロイします。

以下の wscat コマンドを使用して、API に接続します。`wscat` の詳細については、「the section called “`wscat` を使用した WebSocket API への接続とメッセージの送信”」を参照してください。

```
wscat -c wss://api-id.execute-api.us-east-2.amazonaws.com/test
```

Enter ボタンを押して、デフォルトルートを呼び出します。Lambda 関数の本体が戻るはずです。

AWS CLI を使用してルートレスポンスを設定する

AWS CLI を使用して WebSocket API のルートレスポンスを設定するには、次の例に示すように `create-route-response` コマンドを呼び出します。`get-apis` と `get-routes` を呼び出すことで、API ID とルート ID を識別できます。

```
aws apigatewayv2 create-route-response \
    --api-id aabbccddee \
    --route-id 1122334  \
    --route-response-key '$default'
```

出力例:

```
{
    "RouteResponseId": "abcdef",
    "RouteResponseKey": "$default"
}
```

WebSocket サブプロトコルを必要とする **$connect** ルートの設定

クライアントは、WebSocket API への接続中に、`Sec-WebSocket-Protocol` フィールドを使用して WebSocket サブプロトコルをリクエストできます。API がサポートするサブプロトコルをクライアントがリクエストした場合にのみ接続を許可するように、`$connect` ルートの統合を設定できます。

次の Lambda 関数の例は、`Sec-WebSocket-Protocol` ヘッダーをクライアントに返します。この関数は、クライアントが `myprotocol` サブプロトコルを指定した場合のみ API への接続を確立します。

このサンプル API と Lambda プロキシ統合を作成する AWS CloudFormation テンプレートについては、`ws-subprotocol.yaml` を参照してください。

```
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;
}
```

wscat を使用して、API がサポートするサブプロトコルをクライアントがリクエストした場合にのみ API が接続を許可することをテストできます。次のコマンドは、-s フラグを使用して、接続中にサブプロトコルを指定します。

次のコマンドは、サポートされていないサブプロトコルとの接続を試行します。クライアントが chat1 サブプロトコルを指定したため、Lambda 統合は 400 エラーを返し、接続は失敗します。

```
wscat -c wss://api-id.execute-api.region.amazonaws.com/beta -s chat1
error: Unexpected server response: 400
```

次のコマンドは、サポートされているサブプロトコルを接続リクエストに含めます。Lambda 統合により、接続が可能になります。

```
wscat -c wss://api-id.execute-api.region.amazonaws.com/beta -s chat1,myprotocol
```

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

WebSocket API の呼び出しの詳細については、「WebSocket API の呼び出し」を参照してください。

API Gateway での WebSocket API へのアクセスの制御と管理

API Gateway は、WebSocket API へのアクセスを制御および管理するための複数のメカニズムをサポートしています。

認証と認可に次のメカニズムを使用することができます。

- 標準の AWS IAM ロールとポリシーは、柔軟で堅牢なアクセス制御を提供します。IAM ロールとポリシーを使用して、API を作成および管理できるユーザーと、API を呼び出すことができるユーザーを制御できます。詳細については、「IAM 認証の使用」を参照してください。
- IAM タグは、アクセスをコントロールするために、IAM ポリシーと共に使用できます。詳細については、「タグを使用して API Gateway REST API リソースへのアクセスをコントロールする」を参照してください。
- Lambda オーソライザー は、API へのアクセスを制御する Lambda 関数です。詳細については、「Lambda REQUEST オーソライザーの関数の作成」を参照してください。

トピック

- IAM 認証の使用
- Lambda REQUEST オーソライザーの関数の作成

IAM 認証の使用

WebSocket API の IAM 認証は REST API と似ていますが、次のような例外があります。

- 既存のアクション (`execute-api`、`ManageConnections`) に加えて、`Invoke` アクションは `InvalidateCache` をサポートしています。`ManageConnections` は @connections API へのアクセスを制御します。
- WebSocket ルートは別の ARN 形式を使用します。

```
arn:aws:execute-api:region:account-id:api-id/stage-name/route-key
```

- この `@connections` API は、REST API と同じ ARN 形式を使用します。

```
arn:aws:execute-api:region:account-id:api-id/stage-name/POST/@connections
```

> **Important**
>
> IAM 認証を使用する場合は、Signature Version 4 (SigV4) でリクエストに署名する必要があります。

たとえば、クライアントに次のポリシーを設定できます。次の例では、全員がメッセージ (Invoke) を prod ステージのシークレットルートを除くすべてのルートに送信でき、全員がすべてのステージの接続されたクライアント (ManageConnections) にメッセージを送信するのを防ぎます。

```
{
  "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/*"
```

```
      ]
    }
  ]
}
```

Lambda **REQUEST** オーソライザーの関数の作成

WebSocket API の Lambda オーソライザー関数は、REST API と似ていますが、次のような例外があります。

- Lambda オーソライザー関数は、`$connect` ルートにのみ使用できます。
- パス変数 (`event.pathParameters`) を使用することはできません。これは、パスが固定されているためです。
- `event.methodArn` は、HTTP メソッドがないため、REST API の同等のメソッドとは異なります。`$connect` の場合、`methodArn` は `"$connect"` で終了します。

  ```
  arn:aws:execute-api:region:account-id:api-id/stage-name/$connect
  ```

- `event.requestContext` のコンテキスト変数は、REST API のコンテキスト変数とは異なります。

次の例では、WebSocket API の REQUEST オーソライザーへの入力を示しています。

```
{
    "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"
    }
}
```

次の例では、Lambda オーソライザー関数は、the section called “その他の Lambda オーソライザー関数の例” の REST API の Lambda オーソライザー関数の WebSocket バージョンです。

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)
```

前述の Lambda 関数を WebSocket API の REQUEST オーソライザー関数として設定する場合、REST API と同じ手順に従います。

コンソールでこの Lambda オーソライザーを使用するよう $connect ルートを設定するには、$connect ルートを選択または作成します。[ルートリクエスト設定] セクションで、[編集] を選択します。[承認] ドロップダウンメニューで承認者を選択し、[変更を保存] を選択します。

オーソライザーをテストするには、新しい接続を作成する必要があります。$connect でオーソライザーを変更しても、接続済みのクライアントに影響はありません。WebSocket API に接続するときは、設定済みの ID ソースの値を指定する必要があります。たとえば、次の例のように wscat を使用し、有効なクエリ文字列およびヘッダーを送信して接続できます。

```
wscat -c 'wss://myapi.execute-api.us-east-1.amazonaws.com/beta?
QueryString1=queryValue1' -H HeaderAuth1:headerValue1
```

有効な ID 値なしに接続しようとすると、401 レスポンスが送信されます。

```
wscat -c wss://myapi.execute-api.us-east-1.amazonaws.com/beta
error: Unexpected server response: 401
```

WebSocket API 統合の設定

API ルートを設定したら、バックエンドのエンドポイントに統合する必要があります。バックエンドエンドポイントは、統合エンドポイントとも呼ばれ、Lambda 関数、HTTP エンドポイント、または AWS のサービスアクションにすることができます。API 統合には統合リクエストと統合レスポンスがあります。

このセクションでは、 WebSocket API の統合リクエストと統合レスポンスを設定する方法を説明します。

トピック

- API Gateway での WebSocket API 統合リクエストの設定

- API Gateway での WebSocket API 統合リクエストの設定

API Gateway での WebSocket API 統合リクエストの設定

統合リクエストの設定では、次の操作が必要になります。

- バックエンドに統合するルートキーを選択します。
- 呼び出すバックエンドエンドポイントを指定します。WebSocket API は、以下の統合タイプをサポートしています。
  - `AWS_PROXY`
  - `AWS`
  - `HTTP_PROXY`
  - `HTTP`
  - `MOCK`

  統合タイプの詳細については、API Gateway V2 REST API の IntegrationType を参照してください。
- 必要に応じて、1 つ以上のリクエストテンプレートを指定して、ルートリクエストデータを統合リクエストデータに変換する方法を設定します。

API Gateway コンソールを使用して WebSocket API 統合リクエストを設定する

API Gateway コンソールを使用して統合リクエストを WebSocket API のルートに追加するには

1. API Gateway コンソールにサインインし、[API]、[Routes (ルート)] の順に選択します。
2. [ルート] で、ルートを選択します。
3. [統合リクエスト] タブを選択し、[統合リクエスト設定] セクションで [編集] を選択します。
4. [統合タイプ] で、以下のいずれかを選択します。
   - [Lambda 関数] は、API がこのアカウントまたは別のアカウントで既に作成済みの AWS Lambda 関数と統合される場合のみ選択してください。

     AWS Lambda で新しい Lambda 関数を作成する場合、Lambda 関数にリソースのアクセス許可を設定する場合、またはその他の Lambda サービスアクションを実行する場合、代わりに [AWS Service] (AWS のサービス) を選択します。

- API が既存の HTTP エンドポイントに統合される場合は、[HTTP] を選択します。詳細については、「API Gateway で HTTP 統合を設定する」を参照してください。
- 統合バックエンドを必要とすることなく、API Gateway から直接 API レスポンスを生成する場合は、[Mock] を選択します。詳細については、「API Gateway でモック 統合を設定する」を参照してください。
- API が AWS のサービスと直接統合する場合は、[AWS のサービス] を選択します。
- API でプライベート統合エンドポイントして VpcLink を使用する場合は、[VPC リンク] を選択します。詳細については、「API Gateway プライベート統合の設定」を参照してください。

5. [Lambda 関数] を選択した場合は、以下の操作を実行します。

   a. [プロキシ統合の使用] で、Lambda プロキシ統合または クロスアカウントの Lambda プロキシ統合を使用する場合は、チェックボックスをオンにします。

   b. [Lambda 関数] で、次のいずれかの方法で関数を指定します。

      - Lambda 関数が同じアカウントにある場合、関数名を入力し、表示されたドロップダウンリストから関数を選択します。

        > **Note**
        >
        > 関数名には、オプションでエイリアスまたはバージョン指定を含めることができます (`HelloWorld`、`HelloWorld:1`、または `HelloWorld:alpha`)。

      - 関数が別のアカウントにある場合は、関数の ARN を入力します。

   c. 29 秒のデフォルトのタイムアウト値を使用するには、[デフォルトタイムアウト] をオンのままにします。カスタムのタイムアウトを設定するには、[デフォルトタイムアウト] を選択してから、タイムアウト値を `50 ~ 29000` ミリ秒の間で入力します。

6. [HTTP] を選択した場合は、「the section called “ コンソールを使用して統合リクエストを設定する”」のステップ 4 に従います。
7. [Mock] を選択した場合は、[リクエストテンプレート] ステップに進みます。
8. [AWS のサービス] を選択した場合は、「the section called “ コンソールを使用して統合リクエストを設定する”」のステップ 6 に従います。
9. [VPC リンク] を選択した場合は、以下の操作を実行します。

   a. [VPC プロキシ統合の使用] で、リクエストを VPCLink のエンドポイントにプロキシする場合は、チェックボックスを選択します。

b. [HTTP メソッド] で、HTTP バックエンドのメソッドに最も厳密に一致する HTTP メソッドタイプを選択します。

c. [VPC リンク] ドロップダウンリストから、[VPC リンク] を選択します。[Use Stage Variables] を選択して、リストの下のテキストボックスに **${stageVariables.vpcLinkId}** を入力できます。

ステージに API をデプロイした後、vpcLinkId ステージ変数を定義し、その値を VpcLink の ID に定義できます。

d. [エンドポイント URL] に、この統合で使用する HTTP バックエンドの URL を入力します。

e. 29 秒のデフォルトのタイムアウト値を使用するには、[デフォルトタイムアウト] をオンのままにします。カスタムのタイムアウトを設定するには、[デフォルトタイムアウト] を選択してから、タイムアウト値を 50 ~ 29000 ミリ秒の間で入力します。

10. [Save changes] (変更の保存) をクリックします。

11. [クエストテンプレート] で、次の操作を行います。

a. テンプレート選択式を入力するには、[リクエストテンプレート] で [編集] を選択します。

b. テンプレート選択式を入力します。メッセージペイロード内で API Gateway が検索する式を使用します。見つかった式は評価され、結果はメッセージペイロード内のデータに適用されるデータのマッピングテンプレートを選択するために使用されるテンプレートキー値となります。次のステップで、データマッピングテンプレートを作成します。[はい、編集します] を選択して、変更を保存します。

c. [テンプレートの作成] を選択してデータマッピングテンプレートを作成します。[テンプレートキー] に、メッセージペイロード内のデータに適用されるデータマッピングの選択に使用するテンプレートキー値を入力します。次に、マッピングテンプレートを入力します。[テンプレートを作成] をクリックします。

テンプレート選択式の詳細については、「the section called “テンプレート選択式”」を参照してください。

AWS CLI を使用して統合リクエストを設定する

WebSocket API のルートの統合リクエストを設定するには、次の例のように、Mock 統合を作成する AWS CLI を使用します。

1. 次の内容で、integration-params.json というファイルを作成します。

```
{"PassthroughBehavior": "WHEN_NO_MATCH", "TimeoutInMillis": 29000,
 "ConnectionType": "INTERNET", "RequestTemplates": {"application/json":
 "{\"statusCode\":200}"}, "IntegrationType": "MOCK"}
```

2. 次の例に示すように create-integration コマンドを実行します。

```
aws apigatewayv2 --region us-east-1 create-integration --api-id aabbccddee --cli-
input-json file://integration-params.json
```

この例のサンプル出力を以下に示します。

```
{
    "PassthroughBehavior": "WHEN_NO_MATCH",
    "TimeoutInMillis": 29000,
    "ConnectionType": "INTERNET",
    "IntegrationResponseSelectionExpression": "${response.statuscode}",
    "RequestTemplates": {
        "application/json": "{\"statusCode\":200}"
    },
    "IntegrationId": "0abcdef",
    "IntegrationType": "MOCK"
}
```

または、次の例のように AWS CLI を使用してプロキシ統合の統合リクエストを設定することもできます。

1. Lambda コンソールで Lambda 関数を作成し、基本的な Lambda 実行ロールを提供します。
2. 次の例のように create-integration コマンドを実行します。

```
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
```

この例のサンプル出力を以下に示します。

```
{
    "PassthroughBehavior": "WHEN_NO_MATCH",
```

```
    "IntegrationMethod": "POST",
    "TimeoutInMillis": 29000,
    "ConnectionType": "INTERNET",
    "IntegrationUri": "arn:aws:apigateway:us-east-1:lambda:path/2015-03-31/functions/
arn:aws:lambda:us-east-1:123412341234:function:simpleproxy-echo-e2e/invocations",
    "IntegrationId": "abcdefg",
    "IntegrationType": "AWS_PROXY"
}
```

WebSocket API のプロキシ統合のための Lambda 関数の入力形式

Lambda プロキシ統合では、API Gateway がクライアントリクエスト全体をバックエンド Lambda 関数の入力 event パラメータにマップします。次の例は、API Gateway が Lambda プロキシ統合に送信する $connect ルートおよび $disconnect ルートの入力イベントの構造を示しています。

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
}
```

API Gateway での WebSocket API 統合リクエストの設定

トピック

- 統合レスポンスの概要
- 双方向通信の統合レスポンス
- API Gateway コンソールを使用した統合レスポンスの設定
- AWS CLI を使用して統合レスポンスを設定する

統合レスポンスの概要

API Gateway の統合レスポンスとは、バックエンドサービスからのレスポンスをモデル化および操作するための方法です。REST API と WebSocket API 統合レスポンスの設定にはいくつかの違いがありますが、概念的には同じ動作です。

WebSocket ルートは、双方向または単方向通信に対して設定できます。

- ルートが双方向通信用に設定されている場合、REST API の統合レスポンスと同様に、統合レスポンスにより、返されたメッセージペイロードで変換を設定できます。
- ルートが単方向通信用に設定されている場合、統合レスポンスの設定に関係なく、メッセージの処理後に WebSocket チャネル経由でレスポンスは返されません。

ユーザーがルートレスポンスを設定しない限り、API Gateway はルートレスポンスにバックエンドレスポンスを渡しません。ルートレスポンスの詳しい設定方法については、「the section called “WebSocket API のルートレスポンスを設定する”」を参照してください。

双方向通信の統合レスポンス

統合はプロキシ統合と非プロキシ統合に分割することができます。

> **Important**
>
> プロキシ統合では、API Gateway は自動的にバックエンド出力を完全なペイロードとして発信者に渡します。統合レスポンスは発生しません。

非プロキシ統合の場合、少なくとも 1 つの統合レスポンスを設定する必要があります。

- 明示的な選択が行われなかった場合、いずれかの統合レスポンスがキャッチオールとして機能することが理想的です。このデフォルトのケースは、統合レスポンスキー `$default` を設定することにより表されます。
- 他のいずれの場合も、統合レスポンスキーは正規表現として機能します。これは、`"/expression/"` の形式に従う必要があります。

非プロキシ HTTP 統合の場合:

- API Gateway は、バックエンドレスポンスの HTTP ステータスコードへの一致を試みます。この場合、統合レスポンスキーは正規表現として機能します。一致が見つからない場合、`$default` が統合レスポンスとして選択されます。
- テンプレート選択式も、前述したように同様に機能します。例:
  - `/2\d\d/`: 成功のレスポンスを受信して変換
  - `/4\d\d/`: 不正なリクエストエラーを受信して変換
  - `$default`: すべての予期しないレスポンスを受信して変換

テンプレート選択式の詳細については、「the section called “テンプレート選択式”」を参照してください。

API Gateway コンソールを使用した統合レスポンスの設定

API Gateway コンソールを使用して WebSocket API のルート統合レスポンスを設定するには

1. https://console.aws.amazon.com/apigateway で API Gateway コンソールにサインインします。
2. WebSocket API を選択し、ルートを選択します。
3. [統合リクエスト] タブを選択してから、[統合リクエスト設定] セクションで [統合レスポンスの作成] を選択します。

4. [レスポンスキー] に、レスポンス選択式を評価した後で、送信メッセージのレスポンスキーに表示される値を入力します。たとえば、**/4\d\d/** を入力して不正なリクエストエラーを受信して変換したり、**$default** を入力してテンプレート選択式に一致するすべてのレスポンスを受信して変換したりできます。
5. [テンプレート選択式] には、送信メッセージを評価するための選択式を入力します。
6. [レスポンスの作成] を選択します。
7. マッピングテンプレートを定義して、返されるメッセージペイロードの変換を設定することもできます。[テンプレートを作成] をクリックします。
8. キー名を入力します。デフォルトのテンプレート選択式を選択する場合は、**\$default** を入力します。
9. [レスポンステンプレート] に、コードエディタのマッピングテンプレートを入力します。
10. [テンプレートを作成] をクリックします。
11. [API をデプロイ] を選択して、API をデプロイします。

以下の wscat コマンドを使用して、API に接続します。wscat の詳細については、「the section called “wscat を使用した WebSocket API への接続とメッセージの送信”」を参照してください。

```
wscat -c wss://api-id.execute-api.us-east-2.amazonaws.com/test
```

ルートを呼び出すと、返されたメッセージペイロードが戻るはずです。

AWS CLI を使用して統合レスポンスを設定する

AWS CLI を使用して WebSocket API の統合レスポンスを設定するには、create-integration-response コマンドを呼び出します。以下の CLI コマンドは、$default 統合レスポンスを作成する例を示しています。

```
aws apigatewayv2 create-integration-response \
    --api-id vaz7da96z6 \
    --integration-id a1b2c3 \
    --integration-response-key '$default'
```

リクエストの検証

統合リクエストを進める前にルートリクエストの検証を実行するよう API Gateway を設定できます。検証に失敗すると、API ゲートウェイはバックエンドを呼び出さずにリクエストを失敗させ、「Bad request body」というゲートウェイレスポンスをクライアントに送信し、CloudWatch Logs に

検証の失敗を発行します。このように検証を使用すると、APIバックエンドへの不要な呼び出しを減らすことができます。

モデル選択式

モデル選択式を使用して、同じルート内のリクエストを動的に検証できます。モデルの検証は、プロキシ統合または非プロキシ統合のモデル選択式を指定した場合に行われます。一致するモデルが見つからない場合は、`$default` モデルをフォールバックとして定義する必要があります。一致するモデルが存在せず、`$default` が定義されていない場合、検証は失敗します。選択式は `Route.ModelSelectionExpression` のような外観で 、`Route.RequestModels` のキーに評価されます。

WebSocket API のルートを定義する場合、オプションでモデル選択式を指定できます。この式は評価され、リクエストが受信されたときに本文の検証に使用されるモデルが選択されます。この式はルートの `requestmodels` のいずれかのエントリに評価されます。

モデルは JSON スキーマとしてで表され、リクエストボディのデータ構造を説明します。この選択式の特性により、特定のルートに対して実行時に検証するモデルを動的に選択できます。モデルを作成する方法については、「the section called “データモデルを理解する”」を参照してください。

API Gateway コンソールを使用してリクエストの検証を設定する

次の例は、ルートにリクエストの検証を設定する方法を示しています。

最初にモデルを作成し、次にルートを作成します。次に、作成したルートにリクエストの検証を設定します。最後に、API をデプロイしてテストします。このチュートリアルを完了するには、`$request.body.action` をルート選択式とする WebSocket API と、新しいルートの統合エンドポイントが必要です。

また、API に接続するには `wscat` も必要です。詳細については、「the section called “`wscat` を使用した WebSocket API への接続とメッセージの送信”」を参照してください。

モデルを作成するには

1. https://console.aws.amazon.com/apigateway で API Gateway コンソールにサインインします。
2. WebSocket API を選択します。
3. ナビゲーションペインで、[モデル] を選択します。
4. [モデルの作成] を選択します。
5. [名前] に**`emailModel`**と入力します。

6. [コンテンツタイプ] に、「**application/json**」と入力します。
7. [モデルのスキーマ] に次のモデルを入力します。

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "type" : "object",
    "required" : [ "address"],
    "properties" : {
        "address": {
            "type": "string"
        }
    }
}
```

   このモデルでは、リクエストに E メールアドレスを含める必要があります。
8. [Save] を選択します。

このステップでは、WebSocket API のルートを作成します。

ルートを作成するには

1. メインナビゲーションペインで、[ルート] を選択します。
2. [ルートの作成] を選択します。
3. [Route key] (ルートキー) に「**sendMessage**」と入力します。
4. 統合タイプを選択し、統合エンドポイントを指定します。詳細については、「the section called “統合”」を参照してください。
5. [ルートの作成] を選択します。

このステップでは、sendMessage ルートに対するリクエストの検証を設定します。

リクエストの検証を設定するには

1. [ルートリクエスト] タブの [ルートリクエスト設定] で、[編集] を選択します。
2. [モデル選択式] に、「**${request.body.messageType}**」と入力します。

   API Gateway は、messageType プロパティを使用して受信リクエストを検証します。
3. [リクエストモデルを追加] を選択します。

4. [モデルキー] に、「**email**」と入力します。
5. [モデル]] で、[emailModel] を選択します。

   API Gateway は、messageType プロパティが email に設定されている受信メッセージを、このモデルと照合して検証します。

   > **Note**
   >
   > API Gateway は、モデル選択式とモデルキーを一致させることができない場合、$default モデルを選択します。$default モデルが存在しない場合、検証は失敗します。本番稼働 API の場合は、$default モデルを作成することをお勧めします。

6. [Save changes] (変更の保存) をクリックします。

このステップでは、API をデプロイしてテストします。

API をデプロイしてテストするには

1. [API のデプロイ] を選択します。
2. ドロップダウンリストから目的のステージを選択するか、新しいステージの名前を入力します。
3. [デプロイ] を選択します。
4. メインナビゲーションペインで、[ステージ] を選択します。
5. API の WebSocket URL をコピーします。URL は wss://*abcdef123*.execute-api.*us-east-2*.amazonaws.com/production のようになります。
6. 新しいターミナルを開き、次のパラメータを指定して wscat コマンドを実行します。

```
wscat -c wss://abcdef123.execute-api.us-west-2.amazonaws.com/production
```

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

7. 次のコマンドを使用して API をテストします。

```
{"action": "sendMessage", "messageType": "email"}
```

```
{"message": "Invalid request body", "connectionId":"ABCD1=234",
 "requestId":"EFGH="}
```

API Gateway はリクエストを失敗させます。

次のコマンドを使用して API に有効なリクエストを送信します。

```
{"action": "sendMessage", "messageType": "email", "address":
 "mary_major@example.com"}
```

WebSocket API のデータ変換の設定

API Gateway では、API のメソッドリクエストは、バックエンドで必要となる、該当する統合リクエストペイロードとは異なる形式のペイロードを受け取ることができます。同様に、バックエンドは、フロントエンドで予期されるメソッドレスポンスペイロードとは異なる統合レスポンスペイロードを返す場合があります。

API Gateway では、マッピングテンプレートを使用して、ペイロードをメソッドリクエストから該当する統合リクエストにマッピングしたり、統合レスポンスから該当するメソッドレスポンスにマッピングしたりできます。テンプレート選択式を指定して、必要なデータ変換の実行に使用するテンプレートを決定します。

データマッピングを使用して、ルートリクエストからバックエンド統合にデータをマッピングできます。詳細については、「the section called “データマッピング”」を参照してください。

テンプレートとモデルのマッピング

マッピングテンプレートは、Velocity Template Language (VTL) で表現されるスクリプトであり、JSONPath 式を使用してペイロードに適用されます。API Gateway マッピングテンプレートの詳細については、「マッピングテンプレートについて」を参照してください。

ペイロードでは、JSON スキーマのドラフト 4 に基づくデータモデルを使用できます。マッピングテンプレートを作成するためにモデルを定義する必要はありません。ただし、API Gateway では指定されたモデルに基づいてテンプレート設計図が生成されるため、モデルはテンプレートの作成に役立ちます。API Gateway モデルの詳細については、「データモデルを理解する」を参照してください。

テンプレート選択式

マッピングテンプレートを使用してペイロードを変換するには、統合リクエストまたは統合レスポンスで WebSocket API テンプレート選択式を指定します。この式の評価により、(入力テンプレートを

介して) リクエストボディを統合リクエストボディに変換するか、(出力テンプレートを介して) 統合レスポンスボディをルートレスポンスボディに変換するために使用される入力または出力テンプレート (存在する場合) が決定されます。

`Integration.TemplateSelectionExpression` は、`${request.body.jsonPath}` および静的な値をサポートします。

`IntegrationResponse.TemplateSelectionExpression` は、`${request.body.jsonPath}`、`${integration.response.statuscode}`、`${integration.re` および静的な値をサポートします。

統合レスポンスの選択式

WebSocket API に対して統合レスポンスを設定するときは、オプションで統合レスポンスの選択式を指定できます。この式により、統合が返されるときに選択する `IntegrationResponse` が決定されます。現在、この式の値は以下に定義するように、API Gateway によって制限されています。この式は非プロキシ統合に対してのみ適用されることに注意してください。プロキシ統合は、モデル化または変更なしに、単純にレスポンスペイロードを呼び出し元に渡すだけです。

上記に示した他の選択式とは異なり、この式はパターンマッチング形式を現在サポートしています。式はスラッシュで囲む必要があります。

現在、値は `integrationType` に応じて固定されています。

- Lambda ベースの統合の場合は、`$integration.response.body.errorMessage` です。
- `HTTP` および `MOCK` 統合の場合、その値は `$integration.response.statuscode` です。
- `HTTP_PROXY` と `AWS_PROXY` の場合、式は利用されません。これは、呼び出し元へのペイロードのパススルーをリクエストしているためです。

WebSocket API のデータマッピングの設定

データマッピングを使用すると、ルートリクエストからバックエンド統合にデータをマッピングできます。

> **Note**
>
> WebSocket API のデータマッピングは、 ではサポートされていませんAWS Management Console データマッピングを設定するには、AWS CLI、AWS CloudFormation、または SDK を使用する必要があります。

トピック

- ルートリクエストデータを統合リクエストパラメータにマッピングする
- 例

ルートリクエストデータを統合リクエストパラメータにマッピングする

統合リクエストパラメータは、定義されたルートリクエストパラメータ、リクエストボディ、`context` または `stage` 変数、静的値からマッピングできます。

以下の表で、*PARAM_NAME* は、特定のパラメータ型のルートリクエストパラメータの名前です。正規表現パターン `'^[a-zA-Z0-9._$-]+$]'` に一致する必要があります。*JSONPath_EXPRESSION* はリクエストボディの JSON フィールドの JSONPath 式です。

統合リクエストデータのマッピング式

マッピングされたデータソース	マッピング式
リクエストクエリ文字列 ($connect ルートでのみサポート)	`route.request.querystring.`*PARAM_NAME*
リクエストヘッダー ($connect ルートでのみサポート)	`route.request.header.`*PARAM_NAME*
複数値のリクエストクエリ文字列 ($connect ルートでのみサポート)	`route.request.multivaluequerystring.`*PARAM_NAME*
複数値のリクエストヘッダー ($connect ルートでのみサポート)	`route.request.multivalueheader.`*PARAM_NAME*
リクエストボディ	`route.request.body.`*JSONPath_EXPRESSION*
ステージ変数	`stageVariables.`*VARIABLE_NAME*
コンテキスト変数	サポートされるコンテキスト変数の 1 つである必要がある `context.`*VARIABLE_NAME* 。

マッピングされたデータソース	マッピング式
静的な値	*'STATIC_VALUE'* 。*STATIC_VALUE* はリテラル文字列であり、単一引用符で囲む必要があります。

例

以下の AWS CLI の例では、データマッピングを設定しています。AWS CloudFormation テンプレートの例については、「websocket-data-mapping.yaml」を参照してください。

クライアントの connectionId を統合リクエストのヘッダーにマッピングする

以下の例のコマンドは、クライアントの connectionId をバックエンド統合へのリクエストの connectionId ヘッダーにマッピングします。

```
aws apigatewayv2 update-integration \
    --integration-id abc123 \
    --api-id a1b2c3d4 \
    --request-parameters
 'integration.request.header.connectionId'='context.connectionId'
```

クエリ文字列パラメータを統合リクエストのヘッダーにマッピングする

以下の例のコマンドは、authToken クエリ文字列パラメータを統合リクエストの authToken ヘッダーにマッピングします。

まず、authToken クエリ文字列パラメータをルートのリクエストパラメータに追加します。

```
aws apigatewayv2 update-route --route-id 0abcdef \
    --api-id a1b2c3d4 \
    --request-parameters '{"route.request.querystring.authToken": {"Required": false}}'
```

次に、クエリ文字列パラメータを、バックエンド統合へのリクエストの authToken ヘッダーにマッピングします。

```
aws apigatewayv2 update-integration \
    --integration-id abc123 \
```

```
    --api-id a1b2c3d4 \
    --request-parameters
 'integration.request.header.authToken'='route.request.querystring.authToken'
```

必要に応じて、ルートのリクエストパラメータから authToken クエリ文字列パラメータを削除します。

```
aws apigatewayv2 delete-route-request-parameter \
    --route-id 0abcdef \
    --api-id a1b2c3d4 \
    --request-parameter-key 'route.request.querystring.authToken'
```

API Gateway WebSocket API のマッピングテンプレートのリファレンス

このセクションでは、API Gateway の WebSocket API で現在サポートされている一連の変数の概要を示します。

パラメータ	説明
$context.connectionId	クライアントへのコールバックを行うために使用できる接続の一意の ID。
$context.connectedAt	エポック形式の接続時間。
$context.domainName	WebSocket API のドメイン名。これは、(ハードコーディングされた値の代わりに) クライアントへのコールバックを行うために使用できます。
$context.eventType	イベントタイプ: (CONNECT、MESSAGE、または DISCONNECT)。
$context.messageId	メッセージの一意のサーバー側 ID。$context.eventType が MESSAGE である場合のみ利用できます。
$context.routeKey	選択されたルートキー。
$context.requestId	$context.extendedRequestId と同じ

パラメータ	説明
$context.extendedRequestId	自動生成された API コールの ID。デバッグとトラブルシューティングにさらに役立つ情報が含まれています。
$context.apiId	API Gateway が API に割り当てる識別子。
$context.authorizer.principalId	クライアントにより送信され、API Gateway Lambda オーソライザー (以前のカスタムオーソライザー) の Lambda 関数から返されたトークンと関連付けられたプリンシパルユーザー ID。
$context.authorizer. *property*	API Gateway Lambda オーソライザーの関数から返された context マップの指定されたキー/値ペアの文字列化された値。たとえば、オーソライザーが次の context マップを返すとします。 `"context" : { "key": "value", "numKey": 1, "boolKey": true }` $context.authorizer.key の呼び出しでは "value" 文字列が返され、$context.authorizer.numKey の呼び出しでは "1" 文字列が返され、$context.authorizer.boolKey の呼び出しでは "true" 文字列が返されます。
$context.error.messageString	$context.error.message を引用符で囲んだ値、つまり "$context.error.message" 。

パラメータ	説明
`$context.error.validationErrorString`	詳細な検証エラーメッセージを含む文字列。
`$context.identity.accountId`	リクエストに関連付けられた AWS アカウント ID です。
`$context.identity.apiKey`	キー対応 API リクエストに関連付けられた API 所有者キー。
`$context.identity.apiKeyId`	キー対応 API リクエストに関連付けられた API キー ID。
`$context.identity.caller`	リクエストを実行している発信者のプリンシパル ID です。
`$context.identity.cognitoAuthenticationProvider`	リクエストを行う発信者が使用する Amazon Cognito 認証プロバイダーのカンマ区切りのリスト。リクエストが Amazon Cognito 認証情報で署名されている場合にのみ使用できます。 たとえば、Amazon Cognito ユーザープールのアイデンティティの場合、`cognito-idp.`*`region`*`.amazonaws.com/`*`user_pool_id`*`,cognito-idp.`*`region`*`.amazonaws.com/`*`user_pool_id`*`:CognitoSignIn:`*`token subject claim`* 詳しくは、Amazon Cognito デベロッパーガイドの「Amazon Cognito ID プール (フェデレーティッド ID)」を参照してください。

パラメータ	説明
`$context.identity.cognitoAuthenticationType`	リクエストを行う発信者の Amazon Cognito 認証タイプ。リクエストが Amazon Cognito 認証情報で署名されている場合にのみ使用できます。有効な値は、認証されたアイデンティティ`authenticated` および認証されていないアイデンティティ`unauthenticated` です。
`$context.identity.cognitoIdentityId`	リクエストを行う発信者の Amazon Cognito ID。リクエストが Amazon Cognito 認証情報で署名されている場合にのみ使用できます。
`$context.identity.cognitoIdentityPoolId`	リクエストを行う発信者の Amazon Cognito ID プール ID。リクエストが Amazon Cognito 認証情報で署名されている場合にのみ使用できます。
`$context.identity.sourceIp`	API Gateway エンドポイントへのリクエストを実行する即時 TCP 接続のソース IP アドレス。
`$context.identity.user`	リクエストを実行しているユーザーのプリンシパル ID です。
`$context.identity.userAgent`	API 発信者のユーザーエージェントです。
`$context.identity.userArn`	認証後に識別された有効ユーザーの Amazon リソースネーム (ARN) です。
`$context.requestTime`	CLF 形式の要求時間 (`dd/MMM/yyyy:HH:mm:ss +-hhmm`)。
`$context.requestTimeEpoch`	エポック形式のリクエスト時間 (ミリ秒単位)。
`$context.stage`	API コールの開発ステージ (Beta、Prod など)。
`$context.status`	レスポンスステータス。

パラメータ	説明
`$input.body`	文字列として raw ペイロードを返します。
`$input.json(x)`	この関数は、JSONPath の式を評価し、結果を JSON 文字列で返します。 たとえば `$input.json('$.pets')` は、ペットの構造を表す JSON 文字列を返します。 JSONPath の詳細については、JSONPath または JSONPath for Java を参照してください。

パラメータ	説明
`$input.path(x)`	JSONPath 式文字列 (x) を受け取り、結果の JSON オブジェクト表現を返します。これにより、Apache Velocity Template Language (VTL) でペイロード要素にネイティブにアクセスして操作できます。 たとえば、式 `$input.path('$.pets')` が次のようにオブジェクトを返すとします。 `[ { "id": 1, "type": "dog", "price": 249.99 }, { "id": 2, "type": "cat", "price": 124.99 }, { "id": 3, "type": "fish", "price": 0.99 } ]` `$input.path('$.pets').count()` は "3" を返します。 JSONPath の詳細については、JSONPath または JSONPath for Java を参照してください。
`$stageVariables.` *<variable_name>*	*<variable_name>* はステージ変数名を表します。
`$stageVariables['` *<variable_name>* `']`	*<variable_name>* は任意のステージ変数名を表します。

パラメータ	説明
`${stageVariables['` *`<variable_name>`* `']}`	*`<variable_name>`* は任意のステージ変数名を表します。
`$util.escapeJavaScript()`	JavaScript 文字列ルールを使用して文字列内の文字をエスケープします。 **Note** この関数は、通常の一重引用符 (') をエスケープした一重引用符 (\') に変換します。ただし、エスケープした一重引用符は JSON で有効ではありません。したがって、この関数からの出力を JSON のプロパティで使用する場合、エスケープした一重引用符 (\') を通常の一重引用符 (') に戻す必要があります。これを次の例で示します: `$util.escapeJavaScript( data).replaceAll("\\'",'''')`

パラメータ	説明

`$util.parseJson()`	"stringified" JSON を受け取り、結果のオブジェクト表現を返します。この関数の結果を使用して、Apache Velocity Template Language (VTL) でペイロード要素にネイティブにアクセスしてこれらの要素を操作できるようになります。たとえば、次のペイロードがあるとします。
	`{"errorMessage":"{\"key1\":\"var1\", \"key2\":{\"arr\":[1,2,3]}}"}`
	さらに、次のマッピングテンプレートを使用するとします。
	`#set ($errorMessageObj = $util.parseJson($input.path('$.errorMessage')))`
	`{`
	`"errorMessageObjKey2ArrVal" : $errorMessageObj.key2.arr[0]`
	`}`
	この場合、次の出力が返されます。
	`{`
	`"errorMessageObjKey2ArrVal" : 1`
	`}`

`$util.urlEncode()`	文字列を「application/x-www-form-urlencoded」形式に変換します。
`$util.urlDecode()`	「application/x-www-form-urlencoded」文字列をデコードします。
`$util.base64Encode()`	データを base64 エンコードされた文字列にエンコードします。

パラメータ	説明
`$util.base64Decode()`	base64 エンコードされた文字列からデータをデコードします。

WebSocket API のバイナリメディアタイプの使用

現在、API Gateway WebSocket API は、受信メッセージのペイロードでバイナリフレームをサポートしていません。クライアントアプリがバイナリフレームを送信した場合、API Gateway はこれを拒否し、1003 コードでクライアントを切断します。

この動作には回避策があります。クライアントがテキストでエンコードされたバイナリデータ (Base64 など) をテキストフレームとして送信する場合、統合の `contentHandlingStrategy` プロパティを `CONVERT_TO_BINARY` に設定して、ペイロードを base64 エンコードの文字列からバイナリに変換できます。

プロキシ以外の統合でバイナリペイロードのルートレスポンスを返すには、統合レスポンスの `contentHandlingStrategy` プロパティを `CONVERT_TO_TEXT` に設定して、ペイロードをバイナリから base64 でエンコードされた文字列に変換できます。

WebSocket API の呼び出し

WebSocket API をデプロイすると、クライアントアプリケーションはそれに接続してメッセージを送信できます。また、バックエンドサービスは接続されたクライアントアプリケーションにメッセージを送信できます。

- `wscat` を使用して WebSocket API に接続し、メッセージを送信してクライアントの動作をシミュレートできます。「the section called “`wscat` を使用した WebSocket API への接続とメッセージの送信”」を参照してください。
- バックエンドサービスから @connections API を使用して、接続されたクライアントへのコールバックメッセージの送信、接続情報の取得、またはクライアントの切断を行うことができます。「the section called “バックエンドサービスでの `@connections` コマンドの使用”」を参照してください。
- クライアントアプリケーションは独自の WebSocket ライブラリを使用して、WebSocket API を呼び出すことができます。

`wscat` を使用した WebSocket API への接続とメッセージの送信

この `wscat` ユーティリティは、API Gateway で作成してデプロイした WebSocket API をテストするための便利なツールです。`wscat` は以下のようにインストールして使用できます。

1. https://www.npmjs.com/package/wscat から `wscat` をダウンロードします。
2. 次のコマンドを実行して `wscat` をインストールします。

   ```
   npm install -g wscat
   ```

3. API に接続するには、次の例のように `wscat` コマンドを実行します。この例では、`Authorization` 設定が `NONE` であることを前提としています。

   ```
   wscat -c wss://aabbccddee.execute-api.us-east-1.amazonaws.com/test/
   ```

   `aabbccddee` を実際の API ID に置き換える必要があります。これは API Gateway コンソールに表示されるか、AWS CLI `create-api` コマンドによって返されます。

   さらに、API が `us-east-1` 以外のリージョンにある場合、正しいリージョンに置き換える必要があります。

4. API をテストするには、接続中に次のようなメッセージを入力します。

   ```
   {"{jsonpath-expression}":"{route-key}"}
   ```

   ここで、*`{jsonpath-expression}`* は JSONPath 式で、*`{route-key}`* は API のルートキーです。例:

   ```
   {"action":"action1"}
   {"message":"test response body"}
   ```

   JSONPath の詳細については、JSONPath または JSONPath for Java を参照してください。

5. API から切断するには、「`ctrl-C`」と入力します。

バックエンドサービスでの @connections コマンドの使用

バックエンドサービスは次の WebSocket 接続 HTTP リクエストを使用して、接続されたクライアントへのコールバックメッセージの送信、接続情報の取得、またはクライアントの切断を行うことができます。

> **Important**
>
> これらのリクエストは IAM 認証を使用するため、署名バージョン 4 (SigV4) を使用して署名する必要があります。これを行うには、API Gateway 管理 API を使用します。詳細については、「ApiGatewayManagementApi」を参照してください。

次のコマンドでは、*{api-id}* を実際の API ID に置き換える必要があります。この ID は API Gateway コンソールに表示されるか、AWS CLI create-api コマンドから返されます。このコマンドを使用する前に接続を確立する必要があります。

コールバックメッセージをクライアントに送信するには、以下を使用します。

```
POST https://{api-id}.execute-api.us-east-1.amazonaws.com/{stage}/
@connections/{connection_id}
```

以下の例のように、Postman を使用するか、awscurl を呼び出すことで、このリクエストをテストできます。

```
awscurl --service execute-api -X POST -d "hello world" https://{prefix}.execute-api.us-
east-1.amazonaws.com/{stage}/@connections/{connection_id}
```

次の例のようにコマンドを URL でエンコードする必要があります。

```
awscurl --service execute-api -X POST -d "hello world" https://aabbccddee.execute-
api.us-east-1.amazonaws.com/prod/%40connections/R0oXAdfD0kwCH6w%3D
```

クライアントの最新の接続ステータスを取得するには、以下を使用します。

```
GET https://{api-id}.execute-api.us-east-1.amazonaws.com/{stage}/
@connections/{connection_id}
```

クライアントを切断するには、以下を使用します。

```
DELETE https://{api-id}.execute-api.us-east-1.amazonaws.com/{stage}/
@connections/{connection_id}
```

統合で $context 変数を使用して、コールバック URL を動的に構築できます。例えば、Node.js Lambda 関数で Lambda プロキシの統合を使用する場合は、次のように URL を構築し、接続されたクライアントにメッセージを送信できます。

```
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,
  };
};
```

コールバックメッセージを送信する場合、Lambda 関数には API ゲートウェイ管理 API を呼び出すアクセス許可が必要です。接続が確立される前、またはクライアントの接続が切断された後にメッセージを投稿すると、GoneException を含むエラーが表示される場合があります。

顧客が呼び出す WebSocket API の発行

API Gateway API を作成して開発するだけでは、ユーザーが自動的に呼び出せるようにはなりません。呼び出し可能にするには、API をステージにデプロイする必要があります。さらに、ユーザーが API にアクセスするために使用する URL をカスタマイズすることもできます。ブランドと一致するドメインや、API のデフォルト URL よりも記憶に残るドメインを指定できます。

このセクションでは、API をデプロイし、アクセスするためにユーザーに提供する URL をカスタマイズする方法を説明しています。

> **Note**
>
> API Gateway API のセキュリティを強化するため、`execute-api.{region}.amazonaws.com` ドメインはパブリックサフィックスリスト (PSL) に登録されます。セキュリティ強化のため、API Gateway API のデフォルトドメイン名に機密な Cookie を設定する必要が生じた場合は、`__Host-` プレフィックスの付いた Cookie の使用をお勧めします。このプラクティスは、クロスサイトリクエストフォージェリ (CSRF) 攻撃からドメインを防ぐ際に役立ちます。詳細については、Mozilla 開発者ネットワークの「Set-Cookie」ページを参照してください。

トピック

- WebSocket API のステージの操作
- API ゲートウェイでの WebSocket API のデプロイ
- WebSocket API のセキュリティポリシー
- WebSocket API のカスタムドメイン名の設定

WebSocket API のステージの操作

API ステージは、API のライフサイクル状態への論理的なリファレンスです (例: `dev`、`prod`、`beta`、`v2` など)。API ステージは API ID とステージ名で識別され、API の呼び出しに使用する URL に含まれます。各ステージは、API のデプロイの名前付きリファレンスで、クライアントアプリケーションから呼び出すことができます。

デプロイは、API 設定のスナップショットです。ステージに API をデプロイすると、クライアントがその API を呼び出すことができます。変更を有効にするには、API をデプロイする必要があります。

ステージ変数

ステージ変数は、WebSocket API のステージに対して定義できるキーと値のペアです。環境変数と同様に機能し、API のセットアップで使用できます。

たとえば、ステージ変数を定義し、その値を HTTP プロキシ統合の HTTP エンドポイントとして設定することができます。後で、関連付けられたステージ変数名を使用してエンドポイントを参照できます。これにより、各ステージで異なるエンドポイントで同じ API セットアップを使用できます。同様に、ステージ変数を使用して、API の各ステージに異なる AWS Lambda 関数の統合を指定できます。

> **Note**
>
> ステージ変数は、認証情報などの機密データに使用されることを意図していません。機密データを統合に渡すには、AWS Lambda オーソライザーを使用します。Lambda オーソライザーの出力では、機密データを統合に渡すことができます。詳細については、「the section called “Lambda オーソライザーのレスポンス形式”」を参照してください。

例

ステージ変数を使用して HTTP 統合エンドポイントをカスタマイズするには、まずステージ変数の名前と値 (`url` など) を `example.com` の値で設定する必要があります。次に、HTTP プロキシ統合を設定します。エンドポイントの URL を入力する代わりに、ステージ変数の値、**`http://${stageVariables.url}`** を使用するように API Gateway に指示できます 。この値を指定すると、API Gateway は、ランタイムに API のステージに応じてステージ変数の `${}` を置き換えます。

同様に、ステージ変数を参照して Lambda 関数名や AWS のロールの ARN を指定することができます。

ステージ変数値として Lambda 関数名を指定する場合は、その Lambda 関数に対するアクセス許可を手動で設定する必要があります。これを行うには、AWS Command Line Interface (AWS CLI) を使用できます。

```
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 のステージ変数のリファレンス

HTTP 統合 URI

ステージ変数は、次の例に示すように、HTTP 統合 URI の一部として使用できます。

- プロトコルのない完全な URI – `http://${stageVariables.<variable_name>}`
- 完全なドメイン – `http://${stageVariables.<variable_name>}/resource/operation`
- サブドメイン – `http://${stageVariables.<variable_name>}.example.com/resource/operation`
- パス – `http://example.com/${stageVariables.<variable_name>}/bar`
- クエリ文字列 – `http://example.com/foo?q=${stageVariables.<variable_name>}`

Lambda 関数

ステージ変数は、次の例に示すように、Lambda 関数名やエイリアスの代わりに使用できます。

- `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**
>
> Lambda 関数にステージ変数を使用するには、関数が API と同じアカウントにある必要があります。ステージ変数は、クロスアカウント Lambda 関数をサポートしていません。

AWS 統合認証情報

次の例に示すように、ステージ変数を AWS ユーザーまたはロールの認証情報 ARN の一部として使用できます。

- `arn:aws:iam::<account_id>:${stageVariables.<variable_name>}`

API ゲートウェイでの WebSocket API のデプロイ

WebSocket API を作成したら、この API をデプロイしてユーザーが呼び出せるようにする必要があります。

API をデプロイするには、API デプロイを作成し、それを ステージに関連付けます。各ステージは、API のスナップショットであり、クライアントアプリが呼び出し可能になります。

> **Important**
>
> API を更新するたびに、API を再デプロイする必要があります。ステージ設定以外の変更には、以下のリソースへの変更も含めて、再デプロイが必要です。
>
> - ルート
> - 統合
> - オーソライザー
>
> API ごとのステージ数は、デフォルトで 10 個に制限されています。デプロイでステージを再利用することをお勧めします。

デプロイされた WebSocket API を呼び出すため、クライアントはメッセージを API の URL に送信します。URL は、API のホスト名とステージ名によって決定されます。

> **Note**
>
> API Gateway は最大 128 KB までのペイロードをサポートし、最大フレームサイズは 32 KB です。メッセージが 32 KB を超えた場合は、それぞれを 32 KB 以下の複数のフレームに分割する必要があります。

API のデフォルトドメイン名を使用すると、特定のステージ (*{stageName}*) の WebSocket API の URL は、たとえば次の形式になります。

```
wss://{api-id}.execute-api.{region}.amazonaws.com/{stageName}
```

WebSocket API のデフォルトのベース URL をよりユーザーフレンドリなものにするには、カスタムドメイン名 (例: `api.example.com`) を作成し、API のデフォルトのホスト名と置き換えることがで

きます。設定プロセスは、REST API と同じです。詳細については、「the section called “カスタムドメイン名”」を参照してください。

ステージを使用すると、API の堅牢なバージョン管理が可能になります。たとえば、API を test ステージと、prod ステージにデプロイし、test ステージをテストビルドとして使用し、prod ステージを安定したビルドとして使用できます。更新がテストに合格したら、test ステージを prod ステージに昇格させることができます。昇格は、prod ステージに API を再デプロイすることで実行できます。ステージの詳細については、「the section called “ステージのセットアップ”」を参照してください。

トピック

- AWS CLI を使用して WebSocket API デプロイを作成する
- API Gateway コンソールを使用して WebSocket API デプロイを作成する

AWS CLI を使用して WebSocket API デプロイを作成する

AWS CLI を使用してデプロイを作成するには、次の例に示すように create-deployment コマンドを実行します。

```
aws apigatewayv2 --region us-east-1 create-deployment --api-id aabbccddee
```

出力例:

```
{
    "DeploymentId": "fedcba",
    "DeploymentStatus": "DEPLOYED",
    "CreatedDate": "2018-11-15T06:49:09Z"
}
```

このデプロイをステージに関連付けるまで、デプロイされた API は呼び出し可能ではありません。新しいステージを作成するか、以前に作成したステージを再利用できます。

新しいステージを作成してデプロイと関連付けるには、次の例に示すように create-stage コマンドを使用します。

```
aws apigatewayv2 --region us-east-1 create-stage --api-id aabbccddee --deployment-id
 fedcba --stage-name test
```

出力例:

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

既存のステージを再利用するには、update-stage コマンドを使用して、ステージの deploymentId プロパティを、新しく作成したデプロイ ID (*{deployment-id}*) で更新します。

```
aws apigatewayv2 update-stage --region {region} \
   --api-id {api-id} \
   --stage-name {stage-name} \
   --deployment-id {deployment-id}
```

API Gateway コンソールを使用して WebSocket API デプロイを作成する

API Gateway コンソールを使用して WebSocket API のデプロイを作成するには、次の作業を行います。

1. API Gateway コンソールにサインインし、API を選択します。
2. [API のデプロイ] を選択します。
3. ドロップダウンリストから目的のステージを選択するか、新しいステージの名前を入力します。

WebSocket API のセキュリティポリシー

API Gateway は、すべての WebSocket API エンドポイントに対して TLS_1_2 のセキュリティポリシーを適用します。

セキュリティポリシーとは、Amazon API Gateway が提供する TLS の最小バージョンと暗号スイートの事前定義された組み合わせです。TLS プロトコルは、クライアントとサーバーの間の改ざんや傍受などのネットワークセキュリティの問題に対処します。クライアントがカスタムドメインを介

して API に TLS ハンドシェイクを確立すると、セキュリティポリシーにより、TLS バージョンと暗号スイートのオプションが適用されます。ここで使用するオプションは、クライアントが選択できます。このセキュリティポリシーは、TLS 1.2 および TLS 1.3 トラフィックを受け入れ、TLS 1.0 トラフィックを拒否します。

WebSocket API でサポートされている TLS プロトコルと暗号

次の表では、WebSocket API でサポートされている TLS プロトコルと暗号について説明します。

セキュリティポリシー	TLS_1_2
TLS プロトコル	
TLSv1.3	♦
TLSv1.2	♦
TLS 暗号	
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	♦

セキュリティポリシー	TLS_1_2
AES128-SHA256	♦
AES256-GCM-SHA384	♦
AES256-SHA256	♦

OpenSSL および RFC の暗号名

OpenSSL と IETF RFC 5246 では、同じ暗号に異なる名前を使用します。暗号名のリストについては、「the section called “OpenSSL および RFC の暗号名”」を参照してください。

REST API と HTTP API に関する情報

REST API と HTTP API の詳細については、「the section called “セキュリティポリシーの選択”」と「the section called “HTTP API のセキュリティポリシー”」を参照してください。

WebSocket API のカスタムドメイン名の設定

カスタムドメイン名は、API ユーザーに提供できる、よりシンプルで直感的な URL です。

API のデプロイ後、お客様 (およびその顧客) は、以下の形式のデフォルトのベース URL を使用して API を呼び出すことができます。

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

`api-id` は API Gateway によって生成され、`region` (AWS リージョン) は API の作成時に、`stage` は API のデプロイ時に、ユーザーが指定します。

URL のホスト名の部分 (つまり *api-id*`.execute-api.`*region*`.amazonaws.com`) は API エンドポイントを参照します。デフォルトの API エンドポイントは再呼び出しが難しく、ユーザーフレンドリではありません。

カスタムドメイン名を使用すると、API のホスト名を設定し、代替パスを API にマッピングするための基本パス (`myservice` など) を選択できます。たとえば、API のよりわかりやすい ベース URL は以下のようになります。

```
https://api.example.com/myservice
```

> **Note**
>
> WebSocket API のカスタムドメイン名を REST API または HTTP API にマッピングすることはできません。
> WebSocket API では、リージョンのカスタムドメイン名がサポートされます。
> WebSocket API の場合、サポートされる TLS バージョンは TLS 1.2 のみです。

ドメイン名を登録する

API のカスタムドメイン名を設定するには、登録されたインターネットドメイン名が必要です。ドメイン名は RFC 1035 仕様に準拠している必要があり、ラベルあたり最大 63オクテット、合計 255 オクテットを含めることができます。必要に応じて、Amazon Route 53 を使用するか、任意のサードパーティーのドメインレジストラを使用して、インターネットドメインを登録できます。API のカスタムドメイン名は、登録されたインターネットドメインのサブドメイン名またはルートドメイン名 (「Zone Apex」など) にすることができます。

カスタムドメイン名が API Gateway で作成されたら、API エンドポイントにマッピングするために DNS プロバイダーのリソースレコードを作成または更新する必要があります。このマッピングを行わないと、カスタムドメイン名宛ての API リクエストが API Gateway に届きません。

リージョン別カスタムドメイン名

特定のリージョンの API のカスタムドメイン名を作成すると、API Gateway は API のリージョン別ドメイン名を作成します。カスタムドメイン名をリージョン別ドメイン名にマッピングするように、DNS レコードを設定する必要があります。カスタムドメイン名の証明書を提供する必要もあります。

ワイルドカードカスタムドメイン名

ワイルドカードカスタムドメイン名を使用すると、デフォルトのクォータを超えずにほぼ無数のドメイン名をサポートできます。たとえば、各お客様に個別のドメイン名を付けることができます *customername*.api.example.com。

ワイルドカードカスタムドメイン名を制作するためには、ルートドメインの可能なすべてのサブドメインを表すカスタムドメインの最初のサブドメインとして、ワイルドカード (*) を指定します。

たとえば、ワイルドカードカスタムドメイン名として *.example.com を使用すると、a.example.com、b.example.com、c.example.com などのサブドメインが生成され、これらはすべて同じドメインにルーティングされます。

ワイルドカードカスタムドメイン名は、API Gateway の標準のカスタムドメイン名とは異なる設定をサポートします。たとえば、1 つの AWS アカウントで、`*.example.com` と `a.example.com` を異なる動作に設定できます。

コンテキスト変数 `$context.domainName` と `$context.domainPrefix` コンテキスト変数を使用して、クライアントが API を呼び出すために使用したドメイン名を判断できます。コンテキスト変数の詳細については、「API Gateway マッピングテンプレートとアクセスのログ記録の変数リファレンス」を参照してください。

ワイルドカードカスタムドメイン名を作成するには、DNS または E メール検証方法を使用して検証された証明書を ACM から発行してもらう必要があります。

> **Note**
>
> 別の AWS アカウントで作成済みのカスタムドメイン名と競合するようなワイルドカードカスタムドメイン名を作成することはできません。たとえば、アカウント A で `a.example.com` が作成済みである場合、アカウント B はワイルドカードカスタムドメイン名として `*.example.com` を作成できません。
> アカウント A とアカウント B の所有者が同じである場合は、AWS サポートセンターに連絡して例外をリクエストできます。

カスタムドメイン名の証明書

> **Important**
>
> カスタムドメイン名の証明書を指定します。アプリケーションで証明書ピンニング (SSL ピンニングとも呼ばれる) を使用して ACM 証明書を固定すると、AWS が証明書を更新した後にアプリケーションがドメインに接続できないことがあります。詳細については、「AWS Certificate Manager ユーザーガイド」の「証明書のピンニングの問題」を参照してください。

ACM がサポートされているリージョンでカスタムドメイン名の証明書を提供するには、ACM に証明書をリクエストする必要があります。ACM がサポートされていないリージョンで、リージョン別カスタムドメイン名の証明書を提供するには、そのリージョン内の API Gateway に証明書をインポートする必要があります。

SSL/TLS 証明書をインポートするには、カスタムドメイン名の PEM 形式の SSL/TLS 認証本文、そのプライベートキー、およびカスタムドメイン名の証明書チェーンを提供する必要があります。ACM に保存された各証明書は ARN によって識別されます。AWS で管理された証明書をドメイン名で使用するには、その ARN を単に参照します。

ACM を使用すると、API のカスタムドメイン名を簡単に設定して使用できます。特定のドメイン名の証明書を作成 (または証明書をインポート) し、ACM が提供する証明書の ARN を使用して API Gateway でドメイン名を設定します。次に、カスタムドメイン名のベースパスを、デプロイされた API のステージにマッピングします。ACM 発行の証明書により、プライベートキーなど証明書の機密の詳細が漏れる心配はありません。

カスタムドメイン名の設定

カスタムドメイン名の設定の詳細については、「 での証明書の準備AWS Certificate Manager」および「API Gateway でのリージョン別カスタムドメイン名の設定」を参照してください。

WebSocket API 用の API マッピングの使用

API マッピングを使用して、API ステージをカスタムドメイン名に接続します。ドメイン名を作成し、DNS レコードを設定したら、API マッピングを使用して、カスタムドメイン名を使用して API にトラフィックを送信します。

API マッピングは、API、ステージ、およびオプションでマッピングに使用するパスを指定します。たとえば、API の `production` ステージを `wss://api.example.com/orders` にマッピングできます。

API マッピングを作成する前に、API、ステージ、およびカスタムドメイン名が必要です。カスタムドメイン名の作成と設定の詳細については、「the section called “リージョン別カスタムドメイン名の設定”」を参照してください。

制限事項

- API マッピングでは、カスタムドメイン名とマップされた API が同じ AWS アカウントにある必要があります。
- API マッピングに含めることができるのは、文字、数字、および `$-_.+!*'()` の文字だけです。
- API マッピングのパスの最大文字数は 300 文字です。
- WebSocket API を HTTP API または REST API と同じカスタムドメイン名にマッピングすることはできません。

API マッピングを作成する

API マッピングを作成するには、最初にカスタムドメイン名、API、およびステージを作成する必要があります。カスタムドメイン名の作成方法については、「the section called “リージョン別カスタムドメイン名の設定”」を参照してください。

AWS Management Console

API マッピングを作成するには

1. API Gateway コンソール (https://console.aws.amazon.com/apigateway) にサインインします。
2. [カスタムドメイン名] を選択します。
3. 既に作成したカスタムドメイン名を選択します。
4. [API マッピング] を選択します。
5. [Configure API mappings (API マッピングの設定)] を選択します。
6. [Add new mapping (新しいマッピングを追加)] を選択します。
7. API、Stage、必要に応じて Path を入力します。
8. [保存] を選択します。

AWS CLI

次の AWS CLI コマンドは、API マッピングを作成します。この例では、API Gateway が指定された API およびステージに `api.example.com/v1` に対するリクエストを送信します。

```
aws apigatewayv2 create-api-mapping \
    --domain-name api.example.com \
    --api-mapping-key v1 \
    --api-id a1b2c3d4 \
    --stage test
```

AWS CloudFormation

次の AWS CloudFormation 例は、API マッピングを作成します。

```
MyApiMapping:
  Type: 'AWS::ApiGatewayV2::ApiMapping'
  Properties:
```

```
    DomainName: api.example.com
    ApiMappingKey: 'v1'
    ApiId: !Ref MyApi
    Stage: !Ref MyStage
```

WebSocket API のデフォルトのエンドポイントを無効にする

デフォルトでは、クライアントは、API Gateway が API 用に生成する `execute-api` エンドポイントを使用して API を呼び出すことができます。クライアントがカスタムドメイン名を使用した場合のみ API にアクセスできるようにするには、デフォルトの `execute-api` エンドポイントを無効にします。

> **Note**
>
> デフォルトのエンドポイントを無効にすると、API のすべてのステージに影響します。

次の AWS CLI コマンドは、WebSocket API のデフォルトエンドポイントを無効にします。

```
aws apigatewayv2 update-api \
    --api-id abcdef123 \
    --disable-execute-api-endpoint
```

デフォルトのエンドポイントを無効にした後で、変更を有効にするには、API をデプロイする必要があります。

次の AWS CLI コマンドは、デプロイを作成します。

```
aws apigatewayv2 create-deployment \
    --api-id abcdef123 \
    --stage-name dev
```

WebSocket API の保護

API のスロットリングを設定して、多すぎるリクエストで API の負荷が高くなりすぎないように保護できます。スロットルはベストエフォートベースで適用されるため、これらは保証されたリクエスト上限ではなく、目標として考える必要があります。

API Gateway は、トークンバケットアルゴリズムを使用してトークンでリクエストをカウントし、API へのリクエストを調整します。特に API Gateway では、アカウントのすべての API に送信されるリクエストのレートとバーストをリージョンごとに検証します。トークンバケットアルゴリズムでは、これらの制限の事前定義されたオーバーランがバーストによって許可されますが、場合によっては、他の要因によって制限のオーバーランが発生することがあります。

リクエストの送信数がリクエストの定常レートおよびバーストを超えると、API Gateway はリクエストを調整を開始します。クライアントは、この時点で `429 Too Many Requests` エラーレスポンスを受け取ることがあります。このような例外をキャッチすると、クライアントは失敗したリクエストをレート制限する方法で再送信できます。

API デベロッパーは、API の個々のステージまたはルートに制限を設定して、アカウントのすべての API にわたるパフォーマンス全体を向上させることができます。

リージョンごとのアカウントレベルのスロットリング

API Gateway はデフォルトで、リージョンごとに AWS アカウント内のすべての API 全体で定常状態のリクエスト/秒 (RPS) を制限します。また、リージョンごとに AWS アカウント内のすべての API にわたってバースト (最大バケットサイズ) を制限します。API Gateway では、バースト制限は、API Gateway が `429 Too Many Requests` エラーレスポンスを返す前に処理する同時リクエスト送信の目標最大数を表します。スロットリングクォータの詳細については、「クォータと重要な注意点」を参照してください。

アカウントごとの制限は、指定したリージョンのアカウント内のすべての API に適用されます。このアカウントレベルのレート制限は、申請に応じて引き上げることができます。API のタイムアウトが短く、ペイロードが小さい場合、高い制限を設定できます。リージョンごとのアカウントレベルのスロットリング制限の引き上げを申請するには、AWS サポートセンターにお問い合わせください。詳細については、「クォータと重要な注意点」を参照してください。これらの制限は、AWS スロットリングの制限以上に高くすることはできません。

ルートレベルのスロットリング

特定のステージでまたは API の個々のルートでアカウントレベルのリクエストスロットリング制限を上書きするように、ルートレベルのスロットリングを設定できます。デフォルトのルートスロットリング制限は、アカウントレベルのレート制限を超えることはできません。

AWS CLI を使用して、ルートレベルのスロットリングを設定できます。次のコマンドは、API の指定されたステージとルートのカスタムスロットリングを設定します。

```
aws apigatewayv2 update-stage \
    --api-id a1b2c3d4 \
    --stage-name dev \
    --route-settings '{"messages":
{"ThrottlingBurstLimit":100,"ThrottlingRateLimit":2000}}'
```

WebSocket API のモニタリング

CloudWatch メトリクスと CloudWatch Logs を使用して、WebSocket API をモニタリングできます。ログとメトリクスを組み合わせることで、エラーをログに記録し、API のパフォーマンスを監視できます。

> **Note**
>
> API Gateway は、次の場合にログとメトリクスが生成されない可能性があります。
>
> - 413 Request Entity Too Large エラー
> - 過剰な 429 Too Many Requests エラー
> - API マッピングを持たないカスタムドメインに送信されたリクエストからの 400 シリーズのエラー
> - 内部の障害によって発生した 500 シリーズのエラー

トピック

- CloudWatch メトリクスを使用した WebSocket API の実行のモニタリング
- WebSocket API の ログ記録の設定

CloudWatch メトリクスを使用した WebSocket API の実行のモニタリング

Amazon CloudWatch メトリクスを使用して、WebSocket API をモニタリングできます。この設定は、REST API に使用される設定と似ています。詳細については、「Amazon CloudWatch のメトリクスを使用した REST API の実行のモニタリング」を参照してください。

WebSocket API では、次のメトリクスがサポートされています。

メトリクス	説明
ConnectCount	`$connect` ルート統合に送信されるメッセージの数。
MessageCount	クライアントとの間で送受信される、WebSocket API に送信されるメッセージの数。
IntegrationError	統合から 4XX/5XX レスポンスを返すリクエストの数。
ClientError	統合が呼び出される前に API Gateway によって返された 4XX レスポンスを持つリクエストの数。
ExecutionError	統合を呼び出す際に発生したエラー。
IntegrationLatency	API Gateway による統合へのリクエストの送信から、API Gateway による統合からのレスポンスの受信までの時間の差。コールバックと Mock 統合では除外されます。

API Gateway のメトリクスをフィルタリングするには、次の表のディメンションを使用できます。

ディメンション	説明
ApiId	指定した API ID で API の API Gateway メトリクスをフィルタリングします。
ApiId、ステージ	指定した API ID とステージ ID で API ステージの API

ディメンション	説明
	Gateway メトリクスをフィルタリングします。
ApiId、メソッド、リソース、ステージ	指定した API ID、ステージ ID、リソースパス、ルート ID で API メソッドの API Gateway メトリクスをフィルタリングします。 詳細な CloudWatch のメトリクスを明示的に有効にしない限り、API Gateway はこれらのメトリクスを送信しません。そのためには、API Gateway V2 REST API の UpdateStage アクションを呼び出して、`detailedMetricsEnabled` プロパティを `true` に更新します。update-stage AWS CLI コマンドを呼び出して `DetailedMetricsEnabled` プロパティを `true` に更新することもできます。このようなメトリクスを有効にすることで、アカウントに追加料金が発生します。詳細については、Amazon CloudWatch 料金表をご覧ください。

WebSocket API のログ記録の設定

ログ記録を有効にして CloudWatch Logs にログを記録することができます。CloudWatch による API のログには、実行ログとアクセスログの 2 種類があります。実行ログでは、API Gateway に

よって CloudWatch Logs が管理されます。このプロセスには、ロググループとログストリームの作成、および呼び出し元のリクエストとレスポンスのログストリームへのレポートが含まれます。

アクセスログの作成では、API デベロッパーとして、API にアクセスしたユーザーと、呼び出し元が API にアクセスした方法を記録します。独自のロググループを作成したり、既存のロググループを選択したりすることができます。これらは、API Gateway で管理することができます。アクセスの詳細を指定するには、`$context` 変数 (選択した形式で表示される) を選択し、ロググループを宛先として選択します。

CloudWatch ログ記録の設定方法については、「the section called “API Gateway コンソールを使用した CloudWatch による API のログの設定”」を参照してください。

[ログの形式] を指定する際に、記録するコンテキスト変数を選択できます。次の変数は、サポートされています。

パラメータ	説明
`$context.apiId`	API Gateway が API に割り当てる識別子。
`$context.authorize.error`	認可エラーメッセージ。
`$context.authorize.latency`	認可レイテンシー (ミリ秒単位)。
`$context.authorize.status`	認可の試行から返されたステータスコード。
`$context.authorizer.error`	オーソライザーから返されたエラーメッセージ。
`$context.authorizer.integrationLatency`	Lambda オーソライザーレイテンシー (ミリ秒)。
`$context.authorizer.integrationStatus`	オーソライザーから返されたステータスコード。
`$context.authorizer.latency`	オーソライザーのレイテンシー (ミリ秒単位)。
`$context.authorizer.requestId`	AWS エンドポイントのリクエスト ID。
`$context.authorizer.status`	オーソライザーから返されたステータスコード。

パラメータ	説明
`$context.authorizer.principalId`	クライアントにより送信され、API Gateway Lambda オーソライザー Lambda 関数から返されたトークンと関連付けられたプリンシパルユーザー ID (Lambda オーソライザーは、以前はカスタムオーソライザーと呼ばれていました)。
`$context.authorizer.`*`property`*	API Gateway Lambda オーソライザーの関数から返された `context` マップの指定されたキー/値ペアの文字列化された値。たとえば、オーソライザーが次の `context` マップを返すとします。 `"context" : { "key": "value", "numKey": 1, "boolKey": true }` `$context.authorizer.key` の呼び出しでは `"value"` 文字列が返され、`$context.authorizer.numKey` の呼び出しでは `"1"` 文字列が返され、`$context.authorizer.boolKey` の呼び出しでは `"true"` 文字列が返されます。
`$context.authenticate.error`	認証の試行から返されたエラーメッセージ。
`$context.authenticate.latency`	認証レイテンシー (ミリ秒単位)。
`$context.authenticate.status`	認証の試行から返されたステータスコード。
`$context.connectedAt`	エポック形式の接続時間。

パラメータ	説明
$context.connectionId	クライアントへのコールバックを行うために使用できる接続の一意の ID。
$context.domainName	WebSocket API のドメイン名。これは、(ハードコーディングされた値の代わりに) クライアントへのコールバックを行うために使用できます。
$context.error.message	API Gateway エラーメッセージを含む文字列。
$context.error.messageString	$context.error.message を引用符で囲んだ値、つまり "$context.error.message" 。
$context.error.responseType	エラーのレスポンスタイプ。
$context.error.validationErrorString	詳細な検証エラーメッセージを含む文字列。
$context.eventType	イベントタイプ: (CONNECT、MESSAGE、または DISCONNECT)。
$context.extendedRequestId	$context.requestId と同等です。
$context.identity.accountId	リクエストに関連付けられた AWS アカウント ID です。
$context.identity.apiKey	キー対応 API リクエストに関連付けられた API 所有者キー。
$context.identity.apiKeyId	キー対応 API リクエストに関連付けられた API キー ID。
$context.identity.caller	リクエストに署名した発信者のプリンシパル ID。IAM 認証を使用するルートでサポートされています。

パラメータ	説明
`$context.identity.cognitoAuthenticationProvider`	リクエストを行う発信者が使用する Amazon Cognito 認証プロバイダーのカンマ区切りのリスト。リクエストが Amazon Cognito 認証情報で署名されている場合にのみ使用できます。 たとえば、Amazon Cognito ユーザープールのアイデンティティの場合、`cognito-idp.`*`region`*`.amazonaws.com/`*`user_pool_id`*`,cognito-idp.`*`region`*`.amazonaws.com/`*`user_pool_id`*`:CognitoSignIn:`*`token subject claim`* 詳しくは、Amazon Cognito デベロッパーガイドの「Amazon Cognito ID プール (フェデレーティッド ID)」を参照してください。
`$context.identity.cognitoAuthenticationType`	リクエストを行う発信者の Amazon Cognito 認証タイプ。リクエストが Amazon Cognito 認証情報で署名されている場合にのみ使用できます。有効な値は、認証されたアイデンティティ`authenticated` および認証されていないアイデンティティ`unauthenticated` です。
`$context.identity.cognitoIdentityId`	リクエストを行う発信者の Amazon Cognito ID。リクエストが Amazon Cognito 認証情報で署名されている場合にのみ使用できます。
`$context.identity.cognitoIdentityPoolId`	リクエストを行う発信者の Amazon Cognito ID プール ID。リクエストが Amazon Cognito 認証情報で署名されている場合にのみ使用できます。
`$context.identity.principalOrgId`	AWS 組織 ID。IAM 認証を使用するルートでサポートされています。

パラメータ	説明
$context.identity.sourceIp	API Gateway へのリクエストを実行する TCP 接続のソース IP アドレス。
$context.identity.user	リソースアクセスに対して許可されるユーザーのプリンシパル識別子。IAM 認証を使用するルートでサポートされています。
$context.identity.userAgent	API 発信者のユーザーエージェント。
$context.identity.userArn	認証後に識別された有効ユーザーの Amazon リソースネーム (ARN) です。
$context.integration.error	統合から返されたエラーメッセージ。
$context.integration.integrationStatus	Lambda プロキシ統合の場合、バックエンドの Lambda 関数コードからではなく、AWS Lambda から返されるステータスコード。
$context.integration.latency	統合レイテンシー (ミリ秒)。$context.integrationLatency と同等です。
$context.integration.requestId	AWS エンドポイントのリクエスト ID。$context.awsEndpointRequestId と同等です。
$context.integration.status	統合から返されたステータスコード。Lambda プロキシ統合では、これは Lambda 関数コードから返されたステータスコードです。これは $context.integrationStatus と同等です。
$context.integrationLatency	統合レイテンシー (ミリ秒)。アクセスログ記録でのみ使用できます。
$context.messageId	メッセージの一意のサーバー側 ID。$context.eventType が MESSAGE である場合のみ利用できます。

パラメータ	説明
$context.requestId	$context.extendedRequestId と同じ
$context.requestTime	CLF 形式の要求時間 (dd/MMM/yyyy:HH:mm:ss +-hhmm)。
$context.requestTimeEpoch	エポック形式のリクエスト時間 (ミリ秒単位)。
$context.routeKey	選択されたルートキー。
$context.stage	API 呼び出しのデプロイステージ (Beta、Prod など) 。
$context.status	レスポンスステータス。
$context.waf.error	から返されたエラーメッセージAWS WAF
$context.waf.latency	AWS WAF レイテンシー (ミリ秒単位)。
$context.waf.status	から返されたステータスコードAWS WAF

API Gateway コンソールには、一般的に使用されるアクセスログの形式の例が表示されます。以下にもその例を示します。

- CLF (Common Log Format):

```
$context.identity.sourceIp $context.identity.caller \
$context.identity.user [$context.requestTime] "$context.eventType $context.routeKey
 $context.connectionId" \
$context.status $context.requestId
```

継続文字 (\) は、視覚補助を目的としたものです。ログ形式は 1 行である必要があります。ログ形式の末尾に改行文字 (\n) を追加して、各ログエントリの末尾に改行を含めることができます。

- 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"
}
```

継続文字 (\) は、視覚補助を目的としたものです。ログ形式は 1 行である必要があります。ログ形式の末尾に改行文字 (\n) を追加して、各ログエントリの末尾に改行を含めることができます。

- 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>
```

継続文字 (\) は、視覚補助を目的としたものです。ログ形式は 1 行である必要があります。ログ形式の末尾に改行文字 (\n) を追加して、各ログエントリの末尾に改行を含めることができます。

- CSV (カンマ区切り値):

```
$context.identity.sourceIp,$context.identity.caller, \
$context.identity.user,$context.requestTime,$context.eventType, \
$context.routeKey,$context.connectionId,$context.status, \
$context.requestId
```

継続文字 (\) は、視覚補助を目的としたものです。ログ形式は 1 行である必要があります。ログ形式の末尾に改行文字 (\n) を追加して、各ログエントリの末尾に改行を含めることができます。

API Gateway Amazon リソースネーム (ARN) リファレンス

以下の表では、API Gateway リソースの Amazon リソースネーム (ARN) を示しています。AWS Identity and Access Management ポリシーでの ARN の使用の詳細については、「Amazon API Gateway と IAM の連携方法」および「IAM アクセス許可により API へのアクセスを制御する」を参照してください。

HTTP API および WebSocket API リソース

リソース	ARN
AccessLogSettings	arn:*partition* :apigatew ay: *region*::/apis/*api-id*/ stages/*stage-name* /accesslo gsettings
Api	arn:*partition* :apigatew ay: *region*::/apis/*api-id*
API	arn:*partition* :apigatew ay: *region*::/apis
ApiMapping	arn:*partition* :apigatew ay: *region*::/domainnames/ *domain-na me* /apimappings/ *id*
ApiMappings	arn:*partition* :apigatew ay: *region*::/domainnames/ *domain-na me* /apimappings
オーソライザー	arn:*partition* :apigatew ay: *region*::/apis/*api-id*/authoriz ers/ *id*
オーソライザー	arn:*partition* :apigatew ay: *region*::/apis/*api-id*/authoriz ers

リソース	ARN
Cors	arn:*partition* :apigatew ay: *region*::/apis/*api-id*/cors
デプロイ	arn:*partition* :apigatew ay: *region*::/apis/*api-id*/deployme nts/ *id*
デプロイ	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*
Integration	arn:*partition* :apigatew ay: *region*::/apis/*api-id*/integrat ions/ *integration-id*
統合	arn:*partition* :apigatew ay: *region*::/apis/*api-id*/integrat ions
IntegrationResponse	arn:*partition* :apigatew ay: *region*::/apis/*api-id*/integrat ionresponses/ *integration-respon se*

リソース	ARN
IntegrationResponses	arn:*partition* :apigatew ay: *region*::/apis/*api-id*/integrat ionresponses
モデル	arn:*partition* :apigatew ay: *region*::/apis/*api-id*/models/*id*
モデル	arn:*partition* :apigatew ay: *region*::/apis/*api-id*/models
ModelTemplate	arn:*partition* :apigatew ay: *region*::/apis/*api-id*/models/*id*/ template
ルート	arn:*partition* :apigatew ay: *region*::/apis/*api-id*/routes/*id*
ルート	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*

リソース	ARN
ステージ	arn:*partition* :apigatew ay: *region*::/apis/*api-id*/ stages/*stage-name*
ステージ	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 リソース

リソース	ARN
アカウント	arn:*partition* :apigatew ay: *region*::/account
ApiKey	arn:*partition* :apigatew ay: *region*::/apikeys/ *id*
ApiKeys	arn:*partition* :apigatew ay: *region*::/apikeys
オーソライザー	arn:*partition* :apigatew ay: *region*::/restapis/ *api-id*/ authorizers/ *id*
オーソライザー	arn:*partition* :apigatew ay: *region*::/restapis/ *api-id*/ authorizers

リソース	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
デプロイ	arn:*partition* :apigatew ay: *region*::/restapis/ *api-id*/ deployments/ *id*
デプロイ	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*

リソース	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
Integration	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*
[メソッド]	arn:*partition* :apigatew ay: *region*::/restapis/ *api-id*/ resources/ *resource-id* /methods/ *http-method*

リソース	ARN
MethodResponse	arn:*partition* :apigatew ay: *region*::/restapis/ *api-id*/ resources/ *resource-id* /methods/ *http-method* /responses/ *status-co de*
モデル	arn:*partition* :apigatew ay: *region*::/restapis/ *api-id*/ models/*model-name*
モデル	arn:*partition* :apigatew ay: *region*::/restapis/ *api-id*/ models
RequestValidator	arn:*partition* :apigatew ay: *region*::/restapis/ *api-id*/ requestvalidators/ *id*
RequestValidators	arn:*partition* :apigatew ay: *region*::/restapis/ *api-id*/ requestvalidators
リソース	arn:*partition* :apigatew ay: *region*::/restapis/ *api-id*/ resources/ *id*
リソース	arn:*partition* :apigatew ay: *region*::/restapis/ *api-id*/ resources
RestApi 数	arn:*partition* :apigatew ay: *region*::/restapis/ *api-id*
RestApis	arn:*partition* :apigatew ay: *region*::/restapis

リソース	ARN
ステージ	arn:*partition* :apigatew ay: *region*::/restapis/ *api-id*/ stages/*stage-name*
ステージ	arn:*partition* :apigatew ay: *region*::/restapis/ *api-id*/ stages
タグ	arn:*partition* :apigatew ay: *region*::/tags/*url-encoded-resource-arn*
テンプレート	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、および REST API)

リソース	ARN
WebSocket API エンドポイント	arn:*partition* :execute-api: *region*:*account-id* :*api-id*/*stage*/*route-key*
HTTP API および REST API エンドポイント *	arn:*partition* :execute-api: *region*:*account-id* :*api-id*/*stage*/*http-method* /*resource-path*
Lambda オーソライザー **	arn:*partition* :execute-api: *region*:*account-id* :*api-id*/authorizers/ *authorizer-id*

* HTTP API の $default ルートエンドポイントの ARN は arn:*partition*:execute-api:*region*:*account-id*:*api-id*/*/$default です。

** この ARN は、Lambda オーソライザー関数でリソースポリシーの SourceArn 条件を設定する場合にのみ適用されます。例については、「the section called “Lambda オーソライザーの作成”」を参照してください。

OpenAPI への API Gateway 拡張機能の使用

API Gateway 拡張は、REST API および HTTP API に対する AWS 固有の認証および API Gateway 固有の API 統合をサポートします。このセクションでは、OpenAPI の仕様に対する API Gateway 拡張について説明します。

> **Tip**
>
> アプリケーションで API Gateway 拡張がどのように使用されているかを理解するには、API Gateway コンソールを使用して REST API または HTTP API を作成し、OpenAPI 定義ファイルにエクスポートします。API のエクスポート方法の詳細については、「API Gateway から REST API をエクスポートする」および「API Gateway からの HTTP API のエクスポート」を参照してください。

トピック

- x-amazon-apigateway-integration.requestParameters オブジェクト
- x-amazon-apigateway-integration.responses オブジェクト
- x-amazon-apigateway-integration.response オブジェクト
- x-amazon-apigateway-integration.responseTemplates オブジェクト
- x-amazon-apigateway-integration.responseParameters オブジェクト
- x-amazon-apigateway-integration.tlsConfig オブジェクト
- x-amazon-apigateway-minimum-compression-size
- x-amazon-apigateway-policy
- x-amazon-apigateway-request-validator プロパティ
- x-amazon-apigateway-request-validators オブジェクト
- x-amazon-apigateway-request-validators.requestValidator オブジェクト
- x-amazon-apigateway-tag-value プロパティ

x-amazon-apigateway-any-method オブジェクト

OpenAPI Path Item オブジェクトの API Gateway キャッチオール ANY メソッドの OpenAPI Operation オブジェクトを指定します。このオブジェクトは、他の Operation オブジェクトとともに指定でき、明示的に宣言されていない HTTP メソッドをキャッチします。

次の表は、API Gateway が拡張するプロパティを示します。他の OpenAPI Operation プロパティについては、OpenAPI の仕様を参照してください。

プロパティ

プロパティ名	タイプ	説明
`isDefaultRoute`	`Boolean`	ルートが `$default` ルートであるかどうかを指定します。HTTP API に対してのみサポートされます。詳細については、「HTTP API のルートの使用」を参照してください。
`x-amazon-apigateway-integration`	x-amazon-apigateway-integration オブジェクト	メソッドとバックエンドの統合を指定します。こ

プロパティ名	タイプ	説明
		れは、OpenAPI Operation オブジェクトの拡張プロパティです。統合のタイプは、AWS、AWS_PROXY 、HTTP、HTTP_PROXY 、または MOCK です。

x-amazon-apigateway-any-method の例

次の例では、プロキシリソース、 ANY の {proxy+} メソッドを Lambda 関数、TestSimpleProxy と統合しています。

```
    "/{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"
        }
```

次の例では、Lambda 関数、$default と統合する HTTP API の HelloWorld ルートを作成します。

```
"/$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 オブジェクト

HTTP API のクロスオリジンリソース共有 (CORS) 設定を指定します。この拡張機能は、ルートレベルの OpenAPI 構造に適用されます。詳細については、「HTTP API の CORS の設定」を参照してください。

プロパティ

プロパティ名	タイプ	説明
allowOrigins	Array	許可されたオリジンを指定します。
allowCredentials	Boolean	CORS リクエストに認証情報を含めるかどうかを指定します。
exposeHeaders	Array	公開されるヘッダーを指定します。
maxAge	Integer	ブラウザがプリフライトリクエスト結果をキャッシュする秒数を指定します。
allowMethods	Array	許可される HTTP メソッドを指定します。

プロパティ名	タイプ	説明
allowHeaders	Array	許可されるヘッダーを指定します。

x-amazon-apigateway-cors の例

HTTP API の CORS 設定の例を次に示します。

```
"x-amazon-apigateway-cors": {
    "allowOrigins": [
      "https://www.example.com"
    ],
    "allowCredentials": true,
    "exposeHeaders": [
      "x-apigateway-header",
      "x-amz-date",
      "content-type"
    ],
    "maxAge": 3600,
    "allowMethods": [
      "GET",
      "OPTIONS",
      "POST"
    ],
    "allowHeaders": [
      "x-apigateway-header",
      "x-amz-date",
      "content-type"
    ]
}
```

x-amazon-apigateway-api-key-source プロパティ

キーを要求する API メソッドを調整する API キーを受け取るソースを指定します。この API レベルのプロパティは、String タイプです。API キーを要求するメソッドの設定の詳細については、「the section called “OpenAPI 定義で API キーを使用するようにメソッドを設定する”」を参照してください。

リクエストの API キーのソースを指定します。有効な値は次のとおりです。

- リクエストの HEADER ヘッダーから API キーを受信するための X-API-Key。
- Lambda オーソライザー (以前のカスタムオーソライザー) の AUTHORIZER から API キーを受信するための UsageIdentifierKey 。

x-amazon-apigateway-api-key-source の例

次の例では、X-API-Key ヘッダーを API キーソースとして設定します。

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 オブジェクト

API Gateway でのメソッド呼び出しの認証に適用する認証タイプを定義します。

プロパティ

プロパティ名	タイプ	説明
`type`	`string`	認可タイプを指定します。オープンアクセスに対して "NONE" を指定します。IAM アクセス許可を使用するように "AWS_IAM" を指定します。値の大文字と小文字が区別されません。

x-amazon-apigateway-auth の例

次の例では、REST API メソッドの認証タイプを設定します。

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 オブジェクト

API Gateway でのメソッド呼び出しの認証に適用される Lambda オーソライザー、Amazon Cognito ユーザープール、または JWT オーソライザーを定義します。この拡張機能は、 OpenAPI 2 および OpenAPI 3 のセキュリティ定義に適用されます。

プロパティ

プロパティ名	タイプ	説明
`type`	`string`	認証のタイプ。これは必須のプロパティです。 REST API で、発信者 ID が認証トークンに埋め込まれるオーソライザーの場合は、`token` を指定します。呼び出し元 ID がリクエストパラメータに含まれるオーソライザーの場合は、`request` を指定します。Amazon Cognito ユーザープールを使用して API へのアクセスを制御するオーソライザーには、`cognito_user_pools` を指定します。 HTTP API で、発信者 ID がリクエストパラメータに含まれる Lambda オーソライザーの場合は、`request` を指定します。JWT オーソライザーの場合は、`jwt` を指定します。

プロパティ名	タイプ	説明
authorizerUri	string	オーソライザー Lambda 関数の Uniform Resource Identifier (URI)。構文は次のとおりです。 `"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	IAM 実行ロールの ARN 形式でオーソライザーを呼び出すのに必要な認証情報 (ある場合)。たとえば、"arn:aws:iam::*account-id* :*IAM_role*"。
authorizerPayloadFormatVersion	string	HTTP API の場合、API Gateway が Lambda 認証に送信するデータの形式と、API Gateway が Lambda からのレスポンスをどのように解釈するかを指定します。詳細については、「the section called “ペイロード形式バージョン”」を参照してください。

プロパティ名	タイプ	説明
enableSimpleResponses	Boolean	HTTP API では、request オーソライザーがブール値を返すか IAM ポリシーを返すかを指定します。authorizerPayloadFormatVersion が 2.0 のオーソライザーでのみサポートされます。有効にすると、Lambda オーソライザー関数はブール値を返します。詳細については、「the section called "Lambda 関数レスポンス形式 2.0"」を参照してください。
identitySource	string	ID ソースとして、リクエストパラメータの式をマッピングするカンマ区切りのリスト。request および jwt タイプのオーソライザーにのみ適用されます。
jwtConfiguration	Object	JWT オーソライザーの発行者と対象者を指定します。詳細については、API Gateway バージョン 2 API リファレンスの「JWTConfiguration」を参照してください。HTTP API に対してのみサポートされます。

プロパティ名	タイプ	説明
identityValidationExpression	string	入力のアイデンティティとしてトークンを検証するための正規表現。たとえば、"^x-[a-z]+" などです。REST API の TOKEN オーソライザーでのみサポートされています。
authorizerResultTtlInSeconds	string	オーソライザーの結果がキャッシュされる秒数。
providerARNs	string の配列	COGNITO_USER_POOLS の Amazon Cognito ユーザープールの ARN のリスト。

REST API の x-amazon-apigateway-authorizer の例

次の OpenAPI セキュリティ定義の例では、test-authorizer という名前で「token」タイプの Lambda オーソライザーを指定します。

```
 "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
      }
    }
  }
```

次の OpenAPI オペレーションオブジェクトのスニペットでは、上記の Lambda オーソライザーを使用するよう GET /http を設定します。

```
  "/http" : {
     "get" : {
       "responses" : { },
       "security" : [ {
         "test-authorizer" : [ ]
       } ],
       "x-amazon-apigateway-integration" : {
         "type" : "http",
         "responses" : {
           "default" : {
             "statusCode" : "200"
           }
         },
         "httpMethod" : "GET",
         "uri" : "http://api.example.com"
       }
     }
   }
```

次の OpenAPI セキュリティ定義の例では、単一のヘッダーパラメータ (auth) を ID ソースとして使用して、「request」タイプの Lambda オーソライザーを指定します。securityDefinitions の名前は request_authorizer_single_header です。

```
"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
      }
    }
}
```

次の OpenAPI セキュリティ定義の例では、1 つのヘッダー (HeaderAuth1) とクエリ文字列パラメータ (QueryString1) を ID ソースとして使用して、「request」タイプの Lambda オーソライザーを指定します。

```
"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
      }
    }
}
```

次の OpenAPI セキュリティ定義の例では、単一のステージ変数 (stage) を ID ソースとして使用して、「request」タイプの API Gateway Lambda オーソライザーを指定します。

```
"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
      }
    }
}
```

次の OpenAPI セキュリティ定義例では、Amazon Cognito ユーザープールをオーソライザーとして指定しています。

```
 "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"
        ]
      }
    }
```

次の OpenAPI オペレーションオブジェクトスニペットでは、カスタムスコープなしで、上記の Amazon Cognito ユーザープールをオーソライザーとして使用するよう `GET /http` を設定しています。

```
   "/http" : {
      "get" : {
        "responses" : { },
        "security" : [ {
          "cognito-pool" : [ ]
        } ],
        "x-amazon-apigateway-integration" : {
          "type" : "http",
          "responses" : {
            "default" : {
              "statusCode" : "200"
            }
          },
          "httpMethod" : "GET",
          "uri" : "http://api.example.com"
        }
      }
    }
```

HTTP API の x-amazon-apigateway-authorizer の例

次の OpenAPI 3.0 の例では、Amazon Cognito を ID プロバイダーとして使用し、Authorization ヘッダーを ID ソースとして使用する HTTP API の JWT オーソライザーを作成します。

```
"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"
    }
  }
}
```

次の OpenAPI 3.0 の例では、前の例と同じ JWT オーソライザーを作成します。ただし、この例では OpenAPI の `openIdConnectUrl` のプロパティを使用して、発行元を自動的に検出します。`openIdConnectUrl` は完全な形式になっている必要があります。

```
"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"
    }
  }
}
```

次の例では、HTTP API の Lambda オーソライザーを作成します。この例のオーソライザーは、ID ソースとして `Authorization` ヘッダーを使用します。`2.0` が `enableSimpleResponses` に設定されているため、オーソライザーは `true` ペイロード形式バージョンを使用し、ブール値を返します。

```
"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 プロパティ

REST API の場合、この拡張モジュールを使用して Lambda オーソライザーのカスタムタイプを定義できます。この場合、値は自由形式です。たとえば、API は、異なる内部スキームを使用する複数の Lambda オーソライザーを持つ場合があります。この拡張を使用して、Lambda オーソライザーの内部スキームを識別できます。

より一般的には、HTTP API と REST API では、同じセキュリティスキームを共有する複数のオペレーションにわたって IAM 認証を定義する方法としても使用できます。この場合、`awsSigv4`の接頭辞が付く用語と同様、`aws`は予約語になります。

この拡張機能は、 OpenAPI 2 および OpenAPI 3の `apiKey` タイプのセキュリティスキームに適用されます。

x-amazon-apigateway-authtype の例

次の OpenAPI 3 の例では、REST API または HTTP API の複数のリソースにわたって IAM 認証を定義します。

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

次の OpenAPI 3 の例は、REST API のカスタムスキームを使用して Lambda オーソライザーを定義します。

```
{
  "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"
}
```

以下の資料も参照してください。

authorizer.authType

x-amazon-apigateway-binary-media-types のプロパティ

API Gateway がサポートする必要があるバイナリメディアタイプのリストを指定します (application/octet-stream、image/jpeg など) この拡張子は JSON 配列です。これを OpenAPI ドキュメントの最上位ベンダー拡張子として含める必要があります。

x-amazon-apigateway-binary-media-types の例

次の例は、API のエンコード検索順序を示しています。

```
"x-amazon-apigateway-binary-media-types": [ "application/octet", "image/jpeg" ]
```

x-amazon-apigateway-documentation オブジェクト

API Gateway にインポートするドキュメントパーツを定義します。このオブジェクトは、DocumentationPart インスタンスの配列を含む JSON オブジェクトです。

プロパティ

プロパティ名	タイプ	説明
documentationParts	Array	エクスポートまたはインポートする DocumentationPart インスタンスの配列。
version	String	エクスポートするドキュメントパーツのスナップショットのバージョン識別子。

x-amazon-apigateway-documentation の例

OpenAPI への API Gateway 拡張の次の例では、API Gateway で API にインポートまたはエクスポートする DocumentationParts インスタンスを定義します。

```
{ ...
  "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-configuration オブジェクト

API のエンドポイント設定の詳細を指定します。この拡張は、OpenAPI Operation オブジェクトの拡張プロパティです。このオブジェクトは、Swagger 2.0 の最上位ベンダー拡張に存在する必要があります。OpenAPI 3.0 では、サーバーオブジェクトのベンダー拡張の下に存在する必要があります。

プロパティ

プロパティ名	タイプ	説明
`disableExecuteApiEndpoint`	ブール値	クライアントがデフォルトの `execute-api` エンドポイントを使用して API を呼び出すことができるかどうかを指定します。デフォルトでは、クライアントはデフォルトの `https://{api_id}.execute-api.{region}.amazonaws.com` エンドポイントを使用して API を呼び出すことができます。クライアントがカスタムドメイン名を使用して API を呼び出すように要求するには、`true` を指定します。
`vpcEndpointIds`	`String` の配列	REST API の Route 53 エイリアスレコードを作成する対象となる VpcEndpoint 識別子のリスト。`PRIVATE` エンドポイントタイプの REST API でのみサポートされています。

x-amazon-apigateway-endpoint-configuration の例

次の例では、指定された VPC エンドポイントを REST API に関連付けます。

```
"x-amazon-apigateway-endpoint-configuration": {
    "vpcEndpointIds": ["vpce-0212a4ababd5b8c3e", "vpce-01d622316a7df47f9"]
}
```

次の例では、API のデフォルトのエンドポイントを無効にします。

```
"x-amazon-apigateway-endpoint-configuration": {
    "disableExecuteApiEndpoint": true
}
```

x-amazon-apigateway-gateway-responses オブジェクト

API のゲートウェイレスポンスを、キーと値のペアの string-to-GatewayResponse マップとして定義します。この拡張機能は、ルートレベルの OpenAPI 構造に適用されます。

プロパティ

プロパティ名	タイプ	説明
responseType	x-amazon-apigateway-gateway-responses.gatewayResponse	指定した *responseType* の GatewayResponse です。

x-amazon-apigateway-gateway-responses の例

次の OpenAPI に対する API Gateway 拡張例では、2 つの GatewayResponse インスタンス (1 つは DEFAULT_4XX タイプ用、もう 1 つは INVALID_API_KEY タイプ用) を含む GatewayResponses マップを定義しています。

```
{
  "x-amazon-apigateway-gateway-responses": {
    "DEFAULT_4XX": {
      "responseParameters": {
        "gatewayresponse.header.Access-Control-Allow-Origin": "'domain.com'"
      },
```

```
      "responseTemplates": {
        "application/json": "{\"message\": test 4xx b }"
      }
    },
    "INVALID_API_KEY": {
      "statusCode": "429",
      "responseTemplates": {
        "application/json": "{\"message\": test forbidden }"
      }
    }
  }
}
```

x-amazon-apigateway-gateway-responses.gatewayResponse オブジェクト

ステータスコード、該当するレスポンスパラメータ、またはレスポンステンプレートを含め、特定のレスポンスタイプのゲートウェイレスポンスを定義します。

プロパティ

プロパティ名	タイプ	説明
responseParameters	x-amazon-apigateway-gateway-responses.responseParameters	GatewayResponse パラメータ (ヘッダーパラメータ) を指定します。パラメータ値は、受信リクエストパラメータ値または静的なカスタム値です。
responseTemplates	x-amazon-apigateway-gateway-responses.responseTemplates	ゲートウェイレスポンスのマッピングテンプレートを指定します。テンプレートは、VTL エンジンでは処理されません。
statusCode	string	ゲートウェイレスポンスの HTTP ステータスコードです。

x-amazon-apigateway-gateway-responses.gatewayResponse の例

次の OpenAPI に対する API Gateway 拡張例では、GatewayResponse を定義し、INVALID_API_KEY レスポンスをカスタマイズして、456 のステータスコード、受信リクエストの api-key ヘッダー値、"Bad api-key" メッセージを返しています。

```
"INVALID_API_KEY": {
  "statusCode": "456",
  "responseParameters": {
    "gatewayresponse.header.api-key": "method.request.header.api-key"
  },
  "responseTemplates": {
    "application/json": "{\"message\": \"Bad api-key\" }"
  }
}
```

x-amazon-apigateway-gateway-responses.responseParameters オブジェクト

キーと値のペアの文字列間マップを定義し、ゲートウェイレスポンスパラメータを受信リクエストパラメータから生成するか、リテラル文字列を使用して生成します。REST API でのみサポートされます。

プロパティ

プロパティ名	タイプ	説明
gatewayresponse. *param-position* .*param-name*	string	*param-position* は、header、path、querystring のいずれかです。詳細については、「メソッドリクエストデータを統合リクエストパラメータにマッピングする」を参照してください。

x-amazon-apigateway-gateway-responses.responseParameters の例

次の OpenAPI の拡張例では、GatewayResponse レスポンスパラメータマッピング式を使用して、*.example.domain ドメインのリソースに対する CORS サポートを有効にします。

```
"responseParameters": {
  "gatewayresponse.header.Access-Control-Allow-Origin": '*.example.domain',
  "gatewayresponse.header.from-request-header" : method.request.header.Accept,
  "gatewayresponse.header.from-request-path" : method.request.path.petId,
  "gatewayresponse.header.from-request-query" : method.request.querystring.qname
}
```

x-amazon-apigateway-gateway-responses.responseTemplates オブジェクト

特定のゲートウェイレスポンスの GatewayResponse マッピングテンプレートを、キーと値のペアの文字列間マップとして定義します。キーと値のペアごとに、キーはコンテンツタイプです。たとえば、「application/json」と値は、単純な変数置換のための文字列化されたマッピングテンプレートです。GatewayResponse マッピングテンプレートは、Velocity Template Language (VTL) エンジンでは処理されません。

プロパティ

プロパティ名	タイプ	説明
content-type	string	ゲートウェイレスポンス本文をカスタマイズする方法として単純な変数の置換のみをサポートする GatewayResponse 本文マッピングテンプレートです。

x-amazon-apigateway-gateway-responses.responseTemplates の例

次の OpenAPI 拡張例では、API Gateway により生成されたエラーレスポンスをアプリ固有の形式にカスタマイズする GatewayResponse マッピングテンプレートを示しています。

```
      "responseTemplates": {
        "application/json": "{ \"message\": $context.error.messageString, \"type\":
$context.error.responseType, \"statusCode\": '488' }"
      }
```

次の OpenAPI 拡張例では、API Gateway により生成されたエラーレスポンスを静的なエラーメッセージで上書きする GatewayResponse マッピングテンプレートを示しています。

```
      "responseTemplates": {
        "application/json": "{ \"message\": 'API-specific errors' }"
      }
```

x-amazon-apigateway-importexport-version

HTTP API の API Gateway のインポートおよびエクスポートアルゴリズムのバージョンを指定します。現在、サポートされている値は `1.0` のみです。詳細については、API Gateway バージョン 2 API リファレンスの「exportVersion」を参照してください。

x-amazon-apigateway-importexport-version の例

以下の例では、インポートおよびエクスポートバージョンを `1.0` に設定します。

```
{
    "openapi": "3.0.1",
    "x-amazon-apigateway-importexport-version": "1.0",
    "info": { ...
```

x-amazon-apigateway-integration オブジェクト

このメソッドのために使用するバックエンド統合の詳細を指定します。この拡張は、OpenAPI Operation オブジェクトの拡張プロパティです。その結果、 API Gateway 統合オブジェクトが作成されます。

プロパティ

プロパティ名	タイプ	説明
cacheKeyParameters	string の配列	値がキャッシュされるリクエストパラメーターのリスト。
cacheNamespace	string	キャッシュされた関連パラメーターの API 固有のタググループ。
connectionId	string	プライベート統合の VpcLink の ID。
connectionType	string	統合の接続タイプ。有効な値は、プライベート統合の場合は "VPC_LINK" 、それ以外の場合は "INTERNET" です。
credentials	string	AWS IAM ロールベースの認証情報については、該当する IAM ロールの ARN を指定します。指定しない場合、認証情報はデフォルトでリソースベースのアクセス許可に設定されます。API がリソースにアクセスできるようにするためには、リソースベースのアクセス許可を手動で追加する必要があります。詳細については、「リソースポリシーを使用してアクセス許可を付与する」を参照してください。 注意: IAM 認証情報を使用するときは、この API が最適なパフォーマンスのためにデプロイされているリージョンで

プロパティ名	タイプ	説明
		AWS STS リージョンのエンドポイントが有効化されていることを確認してください。
contentHandling	string	リクエストのペイロードエンコード変換タイプ。有効な値は、1) バイナリペイロードを Base64 でエンコードされた文字列に変換する場合、テキストペイロードを utf-8 でエンコードされた文字列に変換する場合、変更なしでテキストペイロードをネイティブに渡す場合の CONVERT_TO_TEXT と、2) テキストペイロードを Base64 でデコードされた BLOB に変換する場合、または変更なしでバイナリペイロードをネイティブに渡す場合の CONVERT_TO_BINARY です。
httpMethod	string	統合リクエストで使用される HTTP メソッドです。Lambda 関数の呼び出しでは、値は POST である必要があります。
integrationSubtype	string	AWS のサービス統合の統合サブタイプを指定します。HTTP API に対してのみサポートされます。サポートされている統合サブタイプについては、「the section called “AWS のサービス統合のリファレンス”」を参照してください。

プロパティ名	タイプ	説明
passthroughBehavior	string	マッピングされていないコンテンツタイプのリクエストペイロードを、変更なしで統合リクエストに渡す方法を指定します。サポートされる値は when_no_templates 、when_no_match 、never です。詳細については、「Integration.passthroughBehavior」を参照してください。
payloadFormatVersion	string	統合に送信されるペイロードの形式を指定します。HTTP API に必須です。HTTP API の場合、Lambda プロキシ統合でサポートされている値は 1.0 および 2.0 です。他のすべての統合では、1.0 が唯一サポートされている値です。詳細については、「the section called “AWS Lambda の統合”」と「the section called “AWS のサービス統合のリファレンス”」を参照してください。

プロパティ名	タイプ	説明
requestParameters	x-amazon-apigateway-integration.requestParameters オブジェクト	REST API では、メソッドリクエストパラメータから統合リクエストパラメータへのマッピングを指定します。サポートされているリクエストパラメーターは、querystring 、path、header、および body です。 HTTP API の場合、リクエストパラメータは、指定した AWS_PROXY と integrationSubtype の統合に渡すパラメータを指定するキーと値のマップです。静的値、マップリクエストデータ、ランタイムに評価されるステージ変数やコンテキスト変数のいずれかを指定できます。詳細については、「the section called “AWS のサービス統合”」を参照してください。
requestTemplates	x-amazon-apigateway-integration.requestTemplates オブジェクト	指定された MIME タイプのリクエストペイロード用のマッピングテンプレートです。
responses	x-amazon-apigateway-integration.responses オブジェクト	メソッドのレスポンスを定義し、統合レスポンスからメソッドレスポンスまで必要なパラメーターマッピングまたはペイロードマッピングを指定します。

プロパティ名	タイプ	説明
timeoutInMillis	integer	統合タイムアウトは 50 ミリ秒～29,000 ミリ秒です。
type	string	特定のバックエンドを持つ統合のタイプ。有効な値は次のとおりです。 • http または http_proxy (HTTP バックエンドとの統合の場合)。 • aws_proxy (AWS Lambda 関数との統合の場合)。 • aws (AWS Lambda 関数または AWS のその他サービス (Amazon DynamoDB、Amazon Simple Notification Service、Amazon Simple Queue Service など) と統合する場合)。 • mock (バックエンドを呼び出さずに API Gateway と統合する場合)。 統合タイプの詳細については、「integration:type」を参照してください。
tlsConfig	the section called “x-amazon-apigateway-integration.tlsConfig”	統合の TLS 設定を指定します。

プロパティ名	タイプ	説明
uri	string	バックエンドのエンドポイント URI。aws タイプの統合の場合、この URI は ARN 値です。HTTP 統合の場合、この URI は、https または http スキームを含む HTTP エンドポイントの URL です。

x-amazon-apigateway-integration の例

HTTP API の場合、OpenAPI 定義のコンポーネントセクションで統合を定義できます。詳細については、「x-amazon-apigateway-integrations オブジェクト」を参照してください。

```
"x-amazon-apigateway-integration": {
    "$ref": "#/components/x-amazon-apigateway-integrations/integration1"
}
```

次の例では、Lambda 関数との統合を作成します。デモンストレーションの目的で、以下の例の requestTemplates および responseTemplates のサンプルマッピングテンプレートは、{ "name":"value_1", "key":"value_2", "redirect": {"url" :"..."} } の JSON 出力または { "stage":"value_1", "user-id":"value_2" } の XML 出力を生成するために、<stage>value_1</stage> という JSON 形式のペイロードに適用されるものとします。

```
"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'"
            }
        }
    }
}
```

マッピングテンプレートの JSON 文字列の二重引用符「"」は、エスケープ文字を付けて「\"」とする必要があります。

x-amazon-apigateway-integrations オブジェクト

統合のコレクションを定義します。OpenAPI 定義のコンポーネントセクションで統合を定義し、統合を複数のルートに再利用できます。HTTP API に対してのみサポートされます。

プロパティ

プロパティ名	タイプ	説明
##	x-amazon-apigateway-integration オブジェクト	統合オブジェクトのコレクション。

x-amazon-apigateway-integrations の例

次の例では、2 つの統合を定義する HTTP API を作成し、`$ref": "#/components/x-amazon-apigateway-integrations/`*`integration-name`* を使用して統合を参照します。

```
{
  "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 オブジェクト

指定された MIME タイプのリクエストペイロード用のマッピングテンプレートを指定します。

プロパティ

プロパティ名	タイプ	説明
MIME type	string	MIME タイプの例の 1 つが application/json です。マッピングテンプレート作成の詳細については、「PetStore マッピングテンプレート」を参照してください。

x-amazon-apigateway-integration.requestTemplates の例

次の例では、application/json および application/xml の MIME タイプのリクエストペイロード用のマッピングテンプレートが設定されています。

```
"requestTemplates" : {
    "application/json" : "#set ($root=$input.path('$')) { \"stage\": \"$root.name\",
 \"user-id\": \"$root.key\" }",
```

```
    "application/xml" : "#set ($root=$input.path('$')) <stage>$root.name</stage> "
}
```

x-amazon-apigateway-integration.requestParameters オブジェクト

REST API では、名前付きメソッドリクエストパラメータから統合リクエストパラメータへのマッピングを指定します。メソッドリクエストパラメーターは、参照される前に定義済みである必要があります。

REST API では、指定した AWS_PROXY を使用して、integrationSubtype 統合に渡すパラメータを指定します。

プロパティ

プロパティ名	タイプ	説明
integration.request. *<param-type>* .*<param-name>*	string	REST API では、値は通常、事前に定義されたmethod.request. *<param-type>* .*<param-name>* 形式のメソッドリクエストパラメータです (ここで、<param-type> は querystring 、path、header、または body です)。ただし、$context. *VARIABLE_NAME* 、$stageVariables. *VARIABLE_NAME* 、および *STATIC_VALUE* も有効です。body パラメーターの場合、<param-name> は $. プレフィックスなしの JSON パス式です。
parameter	string	HTTP API の場合、リクエストパラメータは、指定した

プロパティ名	タイプ	説明
		AWS_PROXY と integrationSubtype の統合に渡すパラメータを指定するキーと値のマップです。静的値、マップリクエストデータ、ランタイムに評価されるステージ変数やコンテキスト変数のいずれかを指定できます。詳細については、「the section called “AWS のサービス統合”」を参照してください。

x-amazon-apigateway-integration.requestParameters の例

次のリクエストパラメータのマッピングの例では、メソッドリクエストのクエリ (version)、ヘッダー (x-user-id)、パス (service) の各パラメータが、統合リクエストのクエリ (stage)、ヘッダー (x-userid)、パス (op) の各パラメータにそれぞれ変換されています。

> **Note**
>
> OpenAPI または AWS CloudFormation を介してリソースを作成している場合は、静的な値は一重引用符で囲む必要があります。
> コンソールからこの値を追加するには、引用符なしでボックスに application/json と入力します。

```
"requestParameters" : {
    "integration.request.querystring.stage" : "method.request.querystring.version",
    "integration.request.header.x-userid" : "method.request.header.x-user-id",
    "integration.request.path.op" : "method.request.path.service"
},
```

x-amazon-apigateway-integration.responses オブジェクト

メソッドのレスポンスを定義し、統合レスポンスからメソッドレスポンスまでパラメーターマッピングまたはペイロードマッピングを指定します。

プロパティ

プロパティ名	タイプ	説明
###############	x-amazon-apigateway-integration.response オブジェクト	統合レスポンスとメソッドレスポンスの照合に使用される正規表現、または未設定のすべての応答をキャッチするために使用される `default` のいずれかです。HTTP 統合の場合、この正規表現が統合レスポンスのステータスコードに適用されます。Lambda の呼び出しでは、Lambda 関数の実行によって例外がスローされるときに、失敗レスポンスの本文として AWS Lambda から返されるエラー情報オブジェクトの `errorMessage` フィールドに正規表現が適用されます。 **Note** ############## のプロパティ名は、レスポンスステータスコードまたはレスポンスステータスコードのグループを表す正規表現を示しています。これは、API Gateway REST API の Integrati

プロパティ名	タイプ	説明
		onResponse リソースのどの識別子にも対応していません。

x-amazon-apigateway-integration.responses の例

次の例では、2xx から 302 までのレスポンスのリストが示されています。2xx レスポンスの場合、メソッドレスポンスは application/json または application/xml MIME タイプの統合レスポンスのペイロードからマッピングされます。このレスポンスでは、指定のマッピングテンプレートが使用されます。302 レスポンスの場合、メソッドレスポンスは Location ヘッダーを返します。ヘッダーの値は、統合レスポンスのペイロードで redirect.url プロパティから派生した値です。

```
"responses" : {
    "2\\d{2}" : {
        "statusCode" : "200",
        "responseTemplates" : {
            "application/json" : "#set ($root=$input.path('$')) { \"stage\":
 \"$root.name\", \"user-id\": \"$root.key\" }",
            "application/xml" : "#set ($root=$input.path('$')) <stage>$root.name</
stage> "
        }
    },
    "302" : {
        "statusCode" : "302",
        "responseParameters" : {
            "method.response.header.Location": "integration.response.body.redirect.url"
        }
    }
}
```

x-amazon-apigateway-integration.response オブジェクト

レスポンスを定義し、統合レスポンスからメソッドレスポンスまでパラメーターマッピングまたはペイロードマッピングを指定します。

プロパティ

プロパティ名	タイプ	説明
statusCode	string	メソッドレスポンスの HTTP ステータスコード (例: "200")。これは、OpenAPI Operation responses フィールドで一致しているレスポンスに対応している必要があります。
responseTemplates	x-amazon-apigateway-integration.responseTemplates オブジェクト	レスポンスのペイロード用に MIME タイプ固有のマッピングテンプレートを指定します。
responseParameters	x-amazon-apigateway-integration.responseParameters オブジェクト	レスポンス用にパラメーターマッピングを指定します。メソッドの header パラメータにマッピングできるのは、統合レスポンスの body パラメータおよび header パラメータだけです。
contentHandling	string	レスポンスのペイロードエンコード変換タイプ。有効な値は、1) バイナリペイロードを Base64 でエンコードされた文字列に変換する場合、テキストペイロードを utf-8 でエンコードされた文字列に変換する場合、変更なしでテ

プロパティ名	タイプ	説明
		キストペイロードをネイティブに渡す場合の CONVERT_TO_TEXT と、2) テキストペイロードを Base64 でデコードされた BLOB に変換する場合、または変更なしでバイナリペイロードをネイティブに渡す場合の CONVERT_TO_BINARY です。

x-amazon-apigateway-integration.response の例

次の例では、バックエンドから 302 または application/json の MIME タイプのペイロードを派生させるメソッド用の application/xml レスポンスが定義されています。このレスポンスでは、指定のマッピングテンプレートが使用され、メソッドの Location ヘッダーで統合レスポンスからのリダイレクト URL が返されます。

```
{
    "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 オブジェクト

指定された MIME タイプのレスポンスペイロード用のマッピングテンプレートを指定します。

プロパティ

プロパティ名	タイプ	説明
MIME type	string	指定の MIME タイプで統合レスポンス本文をメソッドレスポンス本文に変換するマッピングテンプレートを指定します。マッピングテンプレート作成の詳細については、「PetStore マッピングテンプレート」を参照してください。*MIME ###*の例の 1 つが application/json です。

x-amazon-apigateway-integration.responseTemplate の例

次の例では、application/json および application/xml の MIME タイプのリクエストペイロード用のマッピングテンプレートが設定されています。

```
"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 オブジェクト

統合メソッドレスポンスパラメーターからメソッドレスポンスパラメーターまでマッピングを指定します。header、body、または静的な値を header メソッドレスポンスのタイプにマッピングできます。REST API でのみサポートされます。

プロパティ

プロパティ名	タイプ	説明
`method.response.header. <param-name>`	`string`	名前付きパラメータ値が派生するのは、統合レスポンスパラメータの header タイプおよび body タイプからです。

x-amazon-apigateway-integration.responseParameters の例

次の例では、統合レスポンスの body および header の各パラメーターがメソッドレスポンスの 2 つの header パラメーターにマッピングされています。

```
"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 オブジェクト

統合の TLS 設定を指定します。

プロパティ

プロパティ名	タイプ	説明
`insecureSkipVerification`	`Boolean`	REST API でのみサポートされます。サポートされている

プロパティ名	タイプ	説明
		認証機関から統合エンドポイントの証明書が発行されているかどうかの検証を API Gateway がスキップするかどうかを指定します。これはお勧めしませんが、プライベート認証機関によって署名された証明書、または自己署名された証明書を使用できます。有効にした場合でも、API Gateway は証明書の基本的な検証を実行します。これには、証明書の有効期限、ホスト名、ルート認証機関の存在の確認が含まれます。民間機関に属するルート証明書は、以下の制約を満たす必要があります。 • x509 拡張子 keyUsage には keyCertSign が必要です。 • x509 拡張子 basicConstraints には CA:TRUE が必要です。 HTTP および HTTP_PROXY 統合でのみサポートされます。 ⚠ Warning insecureSkipVerification の有効化は、特にパ

プロパティ名	タイプ	説明
		ブリック HTTPS エンドポイントとの統合には推奨されません。`insecureSkipVerification` を有効にすると、中間者 (MITM) 攻撃のリスクが高くなります。
`serverNameToVerify`	`string`	HTTP API プライベート統合でのみサポートされます。サーバー名を指定すると、API Gateway ではそれを使用して統合の証明書のホスト名を検証します。サーバー名は、Server Name Indication (SNI) または仮想ホスティングをサポートするために、TLS ハンドシェイクにも含まれています。

x-amazon-apigateway-integration.tlsConfig examples

以下の OpenAPI3.0 の例では、REST API HTTP プロキシ統合に対して `insecureSkipVerification` を有効にしています。

```
"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",
}
```

以下の OpenAPI 3.0 の例では、REST API プライベート統合に対して `serverNameToVerify` を指定しています。

```
"x-amazon-apigateway-integration" : {
  "payloadFormatVersion" : "1.0",
  "connectionId" : "abc123",
  "type" : "http_proxy",
  "httpMethod" : "ANY",
  "uri" : "arn:aws:elasticloadbalancing:us-west-2:123456789012:listener/app/my-load-
balancer/50dc6c495c0c9188/0467ef3c8400ae65",
  "connectionType" : "VPC_LINK",
  "tlsConfig" : {
     "serverNameToVerify" : "example.com"
  }
}
```

x-amazon-apigateway-minimum-compression-size

REST API の最小圧縮サイズを指定します。圧縮を有効にするには、0 ~ 10485760 の整数を指定します。詳細については、「API のペイロードの圧縮を有効にする」を参照してください。

x-amazon-apigateway-minimum-compression-size の例

次の例では、REST API の最小圧縮サイズ 5242880 バイトを指定します。

```
"x-amazon-apigateway-minimum-compression-size": 5242880
```

x-amazon-apigateway-policy

REST API のリソースポリシーを指定します。リソースポリシーの詳細については、「API Gateway リソースポリシーを使用して API へのアクセスを制御する」を参照してください。リソースポリシーの例については、「API Gateway リソースポリシーの例」を参照してください。

x-amazon-apigateway-policy の例

以下の例では、REST API のリソースポリシーを指定しています。リソースポリシーでは、指定されたソース IP アドレスブロックから API への受信トラフィックを拒否 (ブロック) します。インポート時に、"execute-api:/*" は現在のリージョン、AWS アカウント ID、現在の REST API ID を使用して arn:aws:execute-api:*region*:*account-id*:*api-id*/* に変換されます。

```
"x-amazon-apigateway-policy": {
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": "*",
            "Action": "execute-api:Invoke",
            "Resource": [
                "execute-api:/*"
            ]
        },
        {
            "Effect": "Deny",
            "Principal": "*",
            "Action": "execute-api:Invoke",
            "Resource": [
               "execute-api:/*"
            ],
            "Condition" : {
                "IpAddress": {
                    "aws:SourceIp": "192.0.2.0/24"
                }
            }
        }
    ]
}
```

x-amazon-apigateway-request-validator プロパティ

request_validator_name マップの x-amazon-apigateway-request-validators オブジェクト を参照することでリクエストの検証を指定し、リクエストの検証が含まれる API またはメソッドでのリクエストの検証を有効にします。この拡張の値は、JSON 文字列です。

この拡張は、API レベルまたはメソッドレベルで指定できます。API レベルの検証は、メソッドレベルの検証によりオーバーライドされなければすべてのメソッドに適用されます。

x-amazon-apigateway-request-validator の例

次の例では、basic リクエストの検証を parameter-only リクエストに適用すると同時に、POST /validation リクエストの検証を API レベルで適用します。

OpenAPI 2.0

```
{
  "swagger": "2.0",
  "x-amazon-apigateway-request-validators" : {
    "basic" : {
      "validateRequestBody" : true,
      "validateRequestParameters" : true
    },
    "params-only" : {
      "validateRequestBody" : false,
      "validateRequestParameters" : true
    }
  },
  "x-amazon-apigateway-request-validator" : "basic",
  "paths": {
    "/validation": {
      "post": {
        "x-amazon-apigateway-request-validator" : "params-only",
       ...
     }
}
```

x-amazon-apigateway-request-validators オブジェクト

リクエストの検証が含まれる API のサポートされるリクエストの検証を、検証名と関連するリクエスト検証ルールの間のマップとして定義します。この拡張は、REST API に適用されます。

プロパティ

プロパティ名	タイプ	説明
request_validator_name	x-amazon-apigateway-request-validators.requestValidator オブジェクト	指定された検証で構成される検証ルールを指定します。例: `"basic" : { "validateRequestBody" : true, "validateRequestParameters" : true },` 特定のメソッドにこの検証を適用するには、basic プロパティの値として検証名 (x-amazon-apigateway-request-validator プロパティ) を参照してください。

x-amazon-apigateway-request-validators の例

次の例は、検証名と関連するリクエスト検証ルールの間のマップとして定義された、API のリクエストの検証のセットを示しています。

OpenAPI 2.0

```
{
  "swagger": "2.0",
  ...
  "x-amazon-apigateway-request-validators" : {
    "basic" : {
      "validateRequestBody" : true,
      "validateRequestParameters" : true
    },
    "params-only" : {
      "validateRequestBody" : false,
      "validateRequestParameters" : true
```

```
    }
  },
  ...
}
```

x-amazon-apigateway-request-validators.requestValidator オブジェクト

リクエストの検証の検証ルールを x-amazon-apigateway-request-validators オブジェクト マップ定義の一部として指定します。

プロパティ

プロパティ名	タイプ	説明
validateRequestBody	Boolean	リクエストボディを検証するか (true) しないか (false) を指定します。
validateRequestParameters	Boolean	必須のリクエストパラメータを検証するか (true) しないか (false) を指定します。

x-amazon-apigateway-request-validators.requestValidator の例

次の例は、パラメーターのみのリクエストの検証を示しています。

```
"params-only": {
    "validateRequestBody" : false,
    "validateRequestParameters" : true
}
```

x-amazon-apigateway-tag-value プロパティ

HTTP API の AWS タグの値を指定します。x-amazon-apigateway-tag-value プロパティを、ルートレベルの OpenAPI タグオブジェクトの一部として使用して、HTTP API の AWS タグを指

定できます。x-amazon-apigateway-tag-value プロパティを指定せずにタグ名を指定すると、API Gateway は値として空の文字列を持つタグを作成します。

タグ付けの詳細については、「API Gateway リソースのタグ付け」を参照してください。

プロパティ

プロパティ名	タイプ	説明
name	String	タグキーを指定します。
x-amazon-apigateway-tag-value	String	タグ値を指定します。

x-amazon-apigateway-tag-value の例

次の例では、HTTP API に 2 つのタグを指定します。

- "Owner": "Admin"
- "Prod": ""

```
"tags": [
    {
      "name": "Owner",
      "x-amazon-apigateway-tag-value": "Admin"
    },
    {
      "name": "Prod"
    }
]
```

Amazon API Gateway でのセキュリティ

AWS ではクラウドセキュリティが最優先事項です。セキュリティを最も重視する組織の要件を満たすために構築された AWS のデータセンターとネットワークアーキテクチャは、お客様に大きく貢献します。

セキュリティは、AWS とお客様とが共有する責務です。責任共有モデルでは、これをクラウドのセキュリティおよびクラウド内のセキュリティと説明しています。

- クラウドのセキュリティ - AWS は、AWS クラウドで AWS サービスを実行するインフラストラクチャを保護する責任を負います。また AWS は、安全に使用できるサービスを提供します。AWS コンプライアンスプログラムの一環として、サードパーティー監査者が定期的にセキュリティの有効性をテストおよび検証します。Amazon API Gateway に適用するコンプライアンスプログラムの詳細については、コンプライアンスプログラムによる対象範囲内の「コンプライアンスプログラムによる対象範囲内の AWS のサービス」を参照してください。
- クラウド内のセキュリティ - ユーザーの責任は、使用する AWS サービスに応じて異なります。また、お客様は、データの機密性、会社の要件、適用される法律や規制など、その他の要因についても責任を負います。

このドキュメントは、API Gateway を使用する際の責任共有モデルの適用方法を理解するのに役立ちます。以下のトピックでは、セキュリティおよびコンプライアンスの目的を達成するように API Gateway を設定する方法について説明します。また、API Gateway リソースのモニタリングや保護に役立つ他の AWS のサービスの使用方法についても説明します。

詳細については、「Amazon API Gateway のセキュリティの概要」をご参照ください。

トピック

Amazon API Gateway におけるデータ保護

AWS の責任共有モデルは、Amazon API Gateway でのデータ保護に適用されます。このモデルで説明されているように、AWS は、AWS クラウド のすべてを実行するグローバルインフラストラクチャを保護する責任を担います。お客様は、このインフラストラクチャでホストされているコンテンツに対する管理を維持する責任があります。また、使用する AWS のサービスのセキュリティ設定と管理タスクもユーザーの責任となります。データプライバシーの詳細については、「データプライバシーのよくある質問」を参照してください。欧州でのデータ保護の詳細については、AWS セキュリティブログに投稿された記事「AWS 責任共有モデルおよび GDPR」を参照してください。

データを保護するため、AWS アカウント 認証情報を保護し、AWS IAM Identity Center または AWS Identity and Access Management (IAM) を使用して個々のユーザーをセットアップすることをお勧めします。この方法により、それぞれのジョブを遂行するために必要な権限のみが各ユーザーに付与されます。また、次の方法でデータを保護することもお勧めします:

- 各アカウントで多要素認証 (MFA) を使用します。
- SSL/TLS を使用して AWS リソースと通信します。TLS 1.2 は必須であり TLS 1.3 がお勧めです。
- AWS CloudTrail で API とユーザーアクティビティロギングをセットアップします。
- AWS のサービス 内のすべてのデフォルトセキュリティ管理に加え、AWS 暗号化ソリューションを使用します。
- Amazon Macie などの高度なマネージドセキュリティサービスを使用します。これらは、Amazon S3 に保存されている機密データの検出と保護を支援します。
- コマンドラインインターフェイスまたは API により AWS にアクセスするときに FIPS 140-2 検証済み暗号化モジュールが必要な場合は、FIPS エンドポイントを使用します。利用可能な FIPS エンドポイントの詳細については、「連邦情報処理規格 (FIPS) 140-2」を参照してください。

お客様の E メールアドレスなどの極秘または機密情報は、タグ、または名前フィールドなどの自由形式のテキストフィールドに配置しないことを強くお勧めします。これには、コンソール、API、AWS CLI、または AWS SDK を使用して API Gateway やその他の AWS のサービス を使用する場合も含まれます。名前に使用する自由記述のテキストフィールドやタグに入力したデータは、課金や診断ログに使用される場合があります。外部サーバーへの URL を提供する場合は、そのサーバーへのリクエストを検証するための認証情報を URL に含めないように強くお勧めします。

Amazon API Gateway でのデータ暗号化

データ保護には、転送時 (API Gateway とのデータの送受信中) のデータ保護と、保管時 (での保存中) のデータ保護がありますAWS

Amazon API Gateway で保管時のデータ暗号化

REST API のキャッシュを有効にする場合は、キャッシュ暗号化を有効にすることができます。詳細については、「API キャッシュを有効にして応答性を強化する」を参照してください。

データ保護の詳細については、AWS セキュリティブログのブログ投稿「AWS の責任共有モデルと GDPR」を参照してください。

Amazon API Gateway で転送中のデータ暗号化

Amazon API Gateway で作成された API は、HTTPS エンドポイントのみ公開します。API Gateway は非暗号化 (HTTP) エンドポイントをサポートしません。

API Gateway はデフォルトの `execute-api` エンドポイントの証明書を管理します。カスタムドメイン名を設定する場合は、ドメイン名の証明書を指定します。ベストプラクティスとして、証明書を固定しないでください。

セキュリティを強化するために、API Gateway カスタムドメインに適用する最小バージョンの Transport Layer Security (TLS) プロトコルを選択できます。WebSocket API および HTTP API は TLS 1.2 のみをサポートします。詳細については、「API Gateway におけるカスタムドメインのセキュリティポリシーの選択」を参照してください。

また、アカウントで独自 SSL 証明書を使用して Amazon CloudFront ディストリビューションを設定し、リージョン API で使用することもできます。セキュリティとコンプライアンスの要件に基づいて、TLS 1.1 以降の CloudFront ディストリビューションにセキュリティポリシーを設定できます。

データ保護の詳細については、「REST API の保護」と AWS セキュリティブログの「The AWS Shared Responsibility Model and GDPR」を参照してください。

インターネットトラフィックのプライバシー

Amazon API Gateway を使用すると、Amazon Virtual Private Cloud (VPC) からのみアクセスできるプライベート REST API を作成できます。VPC は、インターフェイス VPC エンドポイントを使用します。これは、VPC 内に作成するエンドポイントネットワークインターフェイスです。リソースポリシーを使用して、選択した VPC および VPC エンドポイント (複数の AWS アカウント含む) から API へのアクセスを許可または拒否できます。各エンドポイントを使用して複数のプライベート API にアクセスできます。AWS Direct Connect を使用してオンプレミスネットワークから Amazon VPC への接続を確立し、その接続経由でプライベート API にアクセスすることも可能です。いずれの場合も、プライベート API へのトラフィックは安全な接続を使用し、Amazon のネットワーク

の外には出ません。パブリックインターネットからは隔離されています。詳細については、「the section called “プライベート REST API”」を参照してください。

Amazon API Gateway の Identity and Access Management

AWS Identity and Access Management (IAM) は、管理者が AWS リソースへのアクセスを安全に制御するために役立つ AWS のサービスです。IAM 管理者は、誰を認証 (サインイン) し、誰に API Gateway リソースの使用を承認する (アクセス許可を付与する) かを制御します。IAM は、追加費用なしで使用できる AWS のサービスです。

トピック

対象者

AWS Identity and Access Management (IAM) の使用方法は、API Gateway で行う作業によって異なります。

サービスユーザー – API Gateway サービスを使用してジョブを実行する場合は、必要なアクセス許可と認証情報を管理者が用意します。使用する API Gateway 機能が増えると、追加のアクセス許可が必要になることがあります。アクセスの管理方法を理解すると、管理者から適切なアクセス許可をリクエストするのに役に立ちます。API Gateway の機能にアクセスできない場合は、「Amazon API Gateway の ID とアクセスのトラブルシューティング」を参照してください。

サービス管理者 – 社内の API Gateway リソースを担当している場合は、おそらく API Gateway へのフルアクセスがあります。サービスのユーザーがどの API Gateway 機能やリソースにアクセスするかを決めるのは管理者の仕事です。その後、IAM 管理者にリクエストを送信して、サービスユーザーの権限を変更する必要があります。このページの情報を確認して、IAM の基本概念を理解して

ください。会社で API Gateway を使用して IAM を利用する方法の詳細については、「Amazon API Gateway と IAM の連携方法」を参照してください。

IAM 管理者 - 管理者は、 API Gateway へのアクセスを管理するポリシーの作成方法の詳細について確認する場合があります。IAM で使用できる API Gateway アイデンティティベースのポリシーの例を表示するには、「Amazon API Gateway のアイデンティティベースのポリシーの例」を参照してください。

アイデンティティを使用した認証

認証とは、アイデンティティ認証情報を使用して AWSにサインインする方法です。ユーザーは、AWS アカウントのルートユーザー として、IAM ユーザーとして、または IAM ロールを引き受けることによって、認証済み (AWS にサインイン済み) である必要があります。

ID ソースから提供された認証情報を使用すると、フェデレーティッドアイデンティティとして AWS にサインインできます。AWS IAM Identity Centerフェデレーティッドアイデンティティの例としては、(IAM アイデンティティセンター) ユーザー、貴社のシングルサインオン認証、Google または Facebook の認証情報などがあります。フェデレーティッドアイデンティティとしてサインインする場合、IAM ロールを使用して、前もって管理者により ID フェデレーションが設定されています。フェデレーションを使用して AWSにアクセスする場合、間接的にロールを引き受けることになります。

ユーザーのタイプに応じて、AWS Management Console または AWSアクセスポータルにサインインできます。AWS へのサインインの詳細については、『AWS サインイン ユーザーガイド』の「AWS アカウント にサインインする方法」を参照してください。

プログラムで AWS にアクセスする場合、AWS は Software Development Kit (SDK) とコマンドラインインターフェイス (CLI) を提供し、認証情報でリクエストに暗号で署名します。AWS ツールを使用しない場合は、リクエストに自分で署名する必要があります。リクエストに署名する推奨方法の使用については、『IAM ユーザーガイド』の「AWS API リクエストの署名」を参照してください。

使用する認証方法を問わず、追加セキュリティ情報の提供をリクエストされる場合もあります。例えば、AWS では、アカウントのセキュリティ強化のために多要素認証 (MFA) の使用をお勧めしています。詳細については、『AWS IAM Identity Center ユーザーガイド』の「Multi-factor authentication」(多要素認証) および『IAM ユーザーガイド』の「AWS における多要素認証 (MFA) の使用」を参照してください。

AWS アカウントのルートユーザー

AWS アカウントを作成する場合は、このアカウントのすべての AWS のサービスとリソースに対して完全なアクセス権を持つ 1 つのサインインアイデンティティから始めます。この ID は AWS アカウント ルートユーザーと呼ばれ、アカウントの作成に使用したメールアドレスとパスワードでサインインすることによってアクセスできます。日常的なタスクには、ルートユーザーを使用しないことを強くお勧めします。ルートユーザーの認証情報は保護し、ルートユーザーでしか実行できないタスクを実行するときに使用します。ルートユーザーとしてサインインする必要があるタスクの完全なリストについては、「IAM ユーザーガイド」の「ルートユーザー認証情報が必要なタスク」を参照してください。

IAM ユーザーとグループ

IAM ユーザーは、1 人のユーザーまたは 1 つのアプリケーションに対して特定の権限を持つ AWS アカウント 内のアイデンティティです。可能であれば、パスワードやアクセスキーなどの長期的な認証情報を保有する IAM ユーザーを作成する代わりに、一時認証情報を使用することをお勧めします。ただし、IAM ユーザーでの長期的な認証情報が必要な特定のユースケースがある場合は、アクセスキーをローテーションすることをお勧めします。詳細については、『IAM ユーザーガイド』の「長期的な認証情報を必要とするユースケースのためにアクセスキーを定期的にローテーションする」を参照してください。

IAM グループは、IAM ユーザーの集団を指定するアイデンティティです。グループとしてサインインすることはできません。グループを使用して、複数のユーザーに対して一度に権限を指定できます。多数のユーザーグループがある場合、グループを使用することで権限の管理が容易になります。例えば、IAMAdmins という名前のグループを設定して、そのグループに IAM リソースを管理する権限を与えることができます。

ユーザーは、ロールとは異なります。ユーザーは 1 人の人または 1 つのアプリケーションに一意に関連付けられますが、ロールはそれを必要とする任意の人が引き受けるようになっています。ユーザーには永続的な長期の認証情報がありますが、ロールでは一時的な認証情報が提供されます。詳細については、『IAM ユーザーガイド』の「IAM ユーザー (ロールではなく) の作成が適している場合」を参照してください。

IAM ロール

IAM ロールは、特定の権限を持つ、AWS アカウント 内のアイデンティティです。これは IAM ユーザーに似ていますが、特定のユーザーには関連付けられていません。ロールの切り替えによって、AWS Management Console で IAM ロールを一時的に引き受けることができます。ロールを引

き受けるには、AWS CLI または AWSAPI オペレーションを呼び出すか、カスタム URL を使用します。ロールを使用する方法の詳細については、『IAM ユーザーガイド』の「IAM ロールの使用」を参照してください。

IAM ロールと一時的な認証情報は、次の状況で役立ちます:

- フェデレーションユーザーアクセス – フェデレーティッドアイデンティティに権限を割り当てるには、ロールを作成してそのロールの権限を定義します。フェデレーティッドアイデンティティが認証されると、そのアイデンティティはロールに関連付けられ、ロールで定義されている権限が付与されます。フェデレーションの詳細については、『IAM ユーザーガイド』の「サードパーティーアイデンティティプロバイダー向けロールの作成」 を参照してください。IAM アイデンティティセンターを使用する場合、権限セットを設定します。アイデンティティが認証後にアクセスできるものを制御するため、IAM Identity Center は、権限セットを IAM のロールに関連付けます。権限セットの詳細については、『AWS IAM Identity Center ユーザーガイド』の「権限セット」を参照してください。
- 一時的な IAM ユーザー権限 - IAM ユーザーまたはロールは、特定のタスクに対して複数の異なる権限を一時的に IAM ロールで引き受けることができます。
- クロスアカウントアクセス - IAM ロールを使用して、自分のアカウントのリソースにアクセスすることを、別のアカウントの人物 (信頼済みプリンシパル) に許可できます。クロスアカウントアクセス権を付与する主な方法は、ロールを使用することです。ただし、一部の AWS のサービスでは、(ロールをプロキシとして使用する代わりに) リソースにポリシーを直接アタッチできます。クロスアカウントアクセスにおけるロールとリソースベースのポリシーの違いについては、『IAM ユーザーガイド』の「IAM ロールとリソースベースのポリシーとの相違点」を参照してください。
- クロスサービスアクセス権 - 一部の AWS のサービスでは、他の AWS のサービスの機能を使用します。例えば、あるサービスで呼び出しを行うと、通常そのサービスによって Amazon EC2 でアプリケーションが実行されたり、Amazon S3 にオブジェクトが保存されたりします。サービスでは、呼び出し元プリンシパルの権限、サービスロール、またはサービスにリンクされたロールを使用してこれを行う場合があります。
  - 転送アクセスセッション (FAS) – IAM ユーザーまたはロールを使用して AWS でアクションを実行するユーザーは、プリンシパルと見なされます。一部のサービスを使用する際に、アクションを実行することで、別のサービスの別のアクションがトリガーされることがあります。FAS は、AWS のサービスを呼び出すプリンシパルの権限を、AWS のサービスのリクエストと合わせて使用し、ダウンストリームのサービスに対してリクエストを行います。FAS リクエストは、サービスが、完了するために他の AWS のサービス またはリソースとのやりとりを必要とするリクエストを受け取ったときにのみ行われます。この場合、両方のアクションを実行するた

めのアクセス許可が必要です。FAS リクエストを行う際のポリシーの詳細については、「転送アクセスセッション」を参照してください。

- サービスロール - サービスがユーザーに代わってアクションを実行するために引き受ける IAM ロールです。IAM 管理者は、IAM 内からサービスロールを作成、変更、削除できます。詳細については、『IAM ユーザーガイド』の「AWS のサービス に権限を委任するロールの作成」を参照してください。
- サービスリンクロール - サービスリンクロールは、AWS のサービス にリンクされたサービスロールの一種です。サービスがロールを引き受け、ユーザーに代わってアクションを実行できるようになります。サービスリンクロールは、AWS アカウント に表示され、サービスによって所有されます。IAM 管理者は、サービスにリンクされたロールの権限を表示できますが、編集することはできません。

- Amazon EC2 で実行されるアプリケーション - EC2 インスタンスで実行され、AWS CLI または AWS API 要求を行っているアプリケーションの一時的な認証情報を管理するために、IAM ロールを使用できます。これは、EC2 インスタンス内でのアクセスキーの保存に推奨されます。AWS ロールを EC2 インスタンスに割り当て、そのすべてのアプリケーションで使用できるようにするには、インスタンスに添付されたインスタンスプロファイルを作成します。インスタンスプロファイルにはロールが含まれ、EC2 インスタンスで実行されるプログラムは一時的な認証情報を取得できます。詳細については、『IAM ユーザーガイド』の「Amazon EC2 インスタンスで実行されるアプリケーションに IAM ロールを使用して権限を付与する」を参照してください。

IAM ロールと IAM ユーザーのどちらを使用するかについては、『IAM ユーザーガイド』の「(IAM ユーザーではなく) IAM ロールをいつ作成したら良いのか?」を参照してください。

ポリシーを使用したアクセス権の管理

AWS でアクセス権を管理するには、ポリシーを作成して AWS アイデンティティまたはリソースにアタッチします。ポリシーは AWS のオブジェクトであり、アイデンティティやリソースに関連付けて、これらの権限を定義します。AWS は、プリンシパル (ユーザー、ルートユーザー、またはロールセッション) がリクエストを行うと、これらのポリシーを評価します。ポリシーでの権限により、リクエストが許可されるか拒否されるかが決まります。大半のポリシーは JSON ドキュメントとして AWSに保存されます。JSON ポリシードキュメントの構造と内容の詳細については、『IAM ユーザーガイド』の「JSON ポリシー概要」を参照してください。

管理者は AWSJSON ポリシーを使用して、だれが何にアクセスできるかを指定できます。つまり、どのプリンシパルがどんなリソースにどんな条件でアクションを実行できるかということです。

デフォルトでは、ユーザーやロールに権限はありません。IAM 管理者は、リソースで必要なアクションを実行するための権限をユーザーに付与する IAM ポリシーを作成できます。その後、管理者はロールに IAM ポリシーを追加し、ユーザーはロールを引き継ぐことができます。

IAM ポリシーは、オペレーションの実行方法を問わず、アクションの権限を定義します。例えば、`iam:GetRole` アクションを許可するポリシーがあるとします。このポリシーがあるユーザーは、AWS Management Console、AWS CLI、または AWS API からロール情報を取得できます。

アイデンティティベースポリシー

アイデンティティベースポリシーは、IAM ユーザー、ユーザーのグループ、ロールなど、アイデンティティにアタッチできる JSON 権限ポリシードキュメントです。これらのポリシーは、ユーザーとロールが実行できるアクション、リソース、および条件をコントロールします。アイデンティティベースのポリシーを作成する方法については、『IAM ユーザーガイド』の「IAM ポリシーの作成」を参照してください。

アイデンティティベースポリシーは、さらにインラインポリシーまたはマネージドポリシーに分類できます。インラインポリシーは、単一のユーザー、グループ、またはロールに直接埋め込まれます。管理ポリシーは、AWS アカウント 内の複数のユーザー、グループ、およびロールにアタッチできるスタンドアロンポリシーです。マネージドポリシーには、AWS マネージドポリシーおよびカスタマーマネージドポリシーがあります。マネージドポリシーまたはインラインポリシーのいずれかを選択する方法については、『IAM ユーザーガイド』の「マネージドポリシーとインラインポリシーの比較」を参照してください。

リソースベースのポリシー

リソースベースのポリシーは、リソースに添付する JSON ポリシードキュメントです。リソースベースのポリシーには例として、IAM ロールの信頼ポリシーや Amazon S3 バケットポリシーがあげられます。リソースベースのポリシーをサポートするサービスでは、サービス管理者はポリシーを使用して特定のリソースへのアクセスを制御できます。ポリシーがアタッチされているリソースの場合、指定されたプリンシパルがそのリソースに対して実行できるアクションと条件は、ポリシーによって定義されます。リソースベースのポリシーでは、プリンシパルを指定する必要があります。プリンシパルには、アカウント、ユーザー、ロール、フェデレーションユーザー、または AWS のサービス を含めることができます。

リソースベースのポリシーは、そのサービス内にあるインラインポリシーです。リソースベースのポリシーでは IAM の AWSマネージドポリシーは使用できません。

アクセスコントロールリスト (ACL)

アクセスコントロールリスト (ACL) は、どのプリンシパル (アカウントメンバー、ユーザー、またはロール) がリソースにアクセスするための権限を持つかをコントロールします。ACL はリソースベースのポリシーに似ていますが、JSON ポリシードキュメント形式は使用しません。

Amazon S3、AWS WAF、および Amazon VPC は、ACL をサポートするサービスの例です。ACL の詳細については、『Amazon Simple Storage Service デベロッパーガイド』の「アクセスコントロールリスト (ACL) の概要」を参照してください。

その他のポリシータイプ

AWS では、他の一般的ではないポリシータイプをサポートしています。これらのポリシータイプでは、より一般的なポリシータイプで付与された最大の権限を設定できます。

- アクセス許可の境界 - アクセス許可の境界は、アイデンティティベースのポリシーによって IAM エンティティ (IAM ユーザーまたはロール) に付与できる権限の上限を設定する高度な機能です。エンティティにアクセス許可の境界を設定できます。結果として得られる権限は、エンティティのアイデンティティベースポリシーとそのアクセス許可の境界の共通部分になります。`Principal` フィールドでユーザーまたはロールを指定するリソースベースのポリシーでは、アクセス許可の境界は制限されません。これらのポリシーのいずれかを明示的に拒否した場合、権限は無効になります。アクセス許可の境界界の詳細については、『IAM ユーザーガイド』の「IAM エンティティのアクセス許可の境界」を参照してください。
- サービスコントロールポリシー (SCP) - SCP は、AWS Organizations で組織や組織単位 (OU) の最大権限を指定する JSON ポリシーです。AWS Organizations は、顧客のビジネスが所有する複数の AWS アカウント をグループ化し、一元的に管理するサービスです。組織内のすべての機能を有効にすると、サービスコントロールポリシー (SCP) を一部またはすべてのアカウントに適用できます。SCP はメンバーアカウントのエンティティに対する権限を制限します (各 AWS アカウントのルートユーザー など)。Organizations と SCP の詳細については、『AWS Organizations ユーザーガイド』の「SCP の仕組み」を参照してください。
- セッションポリシー - セッションポリシーは、ロールまたはフェデレーションユーザーの一時的なセッションをプログラムで作成する際にパラメータとして渡す高度なポリシーです。結果としてセッションの権限は、ユーザーまたはロールのアイデンティティベースポリシーとセッションポリシーの共通部分になります。また、リソースベースのポリシーから権限が派生する場合もあります。これらのポリシーのいずれかを明示的に拒否した場合、権限は無効になります。詳細については、『IAM ユーザーガイド』の「セッションポリシー」を参照してください。

複数のポリシータイプ

1 つのリクエストに複数のタイプのポリシーが適用されると、結果として作成される権限を理解するのがさらに難しくなります。複数のポリシータイプが関与するときに、リクエストを許可するかどうかを AWS が決定する方法の詳細については、IAM ユーザーガイドの「ポリシーの評価ロジック」を参照してください。

Amazon API Gateway と IAM の連携方法

IAM を使用して API Gateway へのアクセスを管理する前に、API Gateway で使用できる IAM 機能について理解しておく必要があります。API Gateway およびその他の AWS のサービスが IAM と連携する方法の概要を把握するには、IAM ユーザーガイドの「IAM と連携する AWS のサービス」を参照してください。

トピック

- API Gateway のアイデンティティベースのポリシー
- API Gateway リソースベースのポリシー
- API Gateway タグに基づく認証
- API Gateway IAM ロール

API Gateway のアイデンティティベースのポリシー

IAM アイデンティティベースのポリシーでは、許可または拒否するアクションとリソース、およびアクションを許可または拒否する条件を指定できます。API Gateway は、特定のアクション、リソース、および条件キーをサポートしています。API Gateway 固有のアクション、リソース、条件キーの詳細については、「Amazon API Gateway Management のアクション、リソース、および条件キー」および「Amazon API Gateway Management V2 のアクション、リソース、および条件キー」を参照してください。JSON ポリシーで使用するすべての要素については、IAM ユーザーガイドの「IAM JSON ポリシーの要素のリファレンス」を参照してください。

次の例は、ユーザーがプライベート REST API だけを作成または更新できるようにする ID ベースのポリシーを示しています。その他の例については、「the section called “アイデンティティベースのポリシーの例”」を参照してください。

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

アクション

JSON ポリシーの Action 要素は、ポリシー内のアクセスを許可または拒否するために使用できるアクションを記述します。

API Gateway のポリシーアクションでは、アクションの前にプレフィックス apigateway: を使用します。ポリシーステートメントには、Action または NotAction 工レメントを含める必要があります。API Gateway は、このサービスで実行できるタスクを記述する独自のアクションのセットを定義します。

API が管理する Action 式の形式は apigateway:*action* であり、ここで *action* は、GET、POST、PUT、DELETE、PATCH (リソースの更新用)、または * (これまでのアクションすべて) の API Gateway アクションのいずれかになります。

Action 式のいくつかの例は以下の通りです。

- **apigateway:***すべての API Gateway アクションについては 。
- **apigateway:GET**API Gateway の GET アクションのみについては 。

単一のステートメントに複数のアクションを指定するには、次のようにコンマで区切ります。

```
"Action": [
      "apigateway:action1",
      "apigateway:action2"
```

特定の API Gateway オペレーションに使用する HTTP 動詞については、「Amazon API Gateway Version 1 API Reference (REST API) 」および「Amazon API Gateway Version 2 API Reference (WebSocket API および HTTP API) 」を参照してください。

詳細については、「the section called “アイデンティティベースのポリシーの例”」を参照してください。

リソース

管理者は AWS JSON ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどんなリソースにどんな条件でアクションを実行できるかということです。

Resource JSON ポリシー要素は、アクションが適用されるオブジェクトを指定します。ステートメントには、Resource または NotResource要素を含める必要があります。ベストプラクティスとして、Amazon リソースネーム (ARN) を使用してリソースを指定します。これは、リソースレベルの権限と呼ばれる特定のリソースタイプをサポートするアクションに対して実行できます。

操作のリスト化など、リソースレベルの許可をサポートしないアクションの場合は、ワイルドカード (*) を使用して、ステートメントがすべてのリソースに適用されることを示します。

```
"Resource": "*"
```

API Gateway リソースには、次の ARN 形式があります。

```
arn:aws:apigateway:region::resource-path-specifier
```

たとえば、ステートメントの認証者など、ID *api-id* とそのサブリソースで REST API を指定するには、次の ARN を使用します。

```
"Resource": "arn:aws:apigateway:us-east-2::/restapis/api-id/*"
```

特定のアカウントに属するすべての REST API やサブリソースを指定するには、ワイルドカード (*) を使用します。

```
"Resource": "arn:aws:apigateway:us-east-2::/restapis/*"
```

API Gateway リソースタイプとそれらの ARN の一覧については、「API Gateway Amazon リソースネーム (ARN) リファレンス」を参照してください。

条件キー

管理者は AWS JSON ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどんなリソースにどんな条件でアクションを実行できるかということです。

Condition 要素 (または Condition ブロック) を使用すると、ステートメントが有効な条件を指定できます。Condition 要素はオプションです。イコールや未満などの 条件演算子 を使用して条件式を作成することで、ポリシーの条件とリクエスト内の値を一致させることができます。

1 つのステートメントに複数の Condition 要素を指定するか、1 つの Condition 要素に複数のキーを指定すると、AWS は AND 論理演算子を使用してそれらを評価します。単一の条件キーに複数の値を指定すると、AWS は OR 論理演算子を使用して条件を評価します。ステートメントの権限が付与される前にすべての条件が満たされる必要があります。

条件を指定する際にプレースホルダー変数も使用できます。例えば IAM ユーザーに、IAM ユーザー名がタグ付けされている場合のみリソースにアクセスできる権限を付与することができます。詳細については、『IAM ユーザーガイド』の「IAM ポリシーの要素: 変数およびタグ」を参照してください。

AWS はグローバル条件キーとサービス固有の条件キーをサポートしています。すべての AWS グローバル条件キーを確認するには、IAM ユーザーガイドの「AWS グローバル条件コンテキストキー」を参照してください。

API Gateway は独自の条件キーを定義し、一部のグローバル条件キーの使用もサポートしています。API Gateway の条件キーの一覧については、IAM ユーザーガイドの「Amazon API Gateway の

条件キー」を参照してください。どのアクションおよびリソースと条件キーを使用できるかについては、「Amazon API Gateway Management V2 のアクション、リソース、および条件キー」を参照してください。

属性ベースのアクセスコントロールなど、タグ付けの詳細については、タグ付け を参照してください。

例

API Gateway アイデンティティベースのポリシーの例については、「Amazon API Gateway のアイデンティティベースのポリシーの例」を参照してください。

API Gateway リソースベースのポリシー

リソースベースのポリシーとは、 API Gateway リソース上で指定するプリンシパルとしてのどのアクションをどの条件で実行できるかを指定する JSON ポリシードキュメントです。API Gateway は、REST API のリソースベースのアクセス許可ポリシーをサポートします。リソースポリシーを使用して、REST API を呼び出すことができるユーザーを管理します。詳細については、「the section called “API Gateway リソースポリシーの使用”」を参照してください。

例

API Gateway リソースベースのポリシーの例については、「API Gateway リソースポリシーの例」を参照してください。

API Gateway タグに基づく認証

API Gateway リソースにタグをアタッチしたり、リクエスト内のタグを API Gateway に渡したりできます。タグに基づいてアクセスを制御するには、`apigateway:ResourceTag/key-name`、`aws:RequestTag/key-name`、または `aws:TagKeys` の条件キーを使用して、ポリシーの条件要素でタグ情報を提供します。API Gateway リソースのタグ付けの詳細については、「the section called “単一ドメイン内の属性ベースの”」を参照してください。

リソースのタグに基づいてリソースへのアクセスを制限するためのアイデンティティベースのポリシーの例については、「タグを使用して API Gateway REST API リソースへのアクセスをコントロールする」を参照してください。

API Gateway IAM ロール

IAM ロールは AWS アカウント内のエンティティで、特定の許可を持っています。

API Gateway での一時的な認証情報の使用

一時的な認証情報を使用して、フェデレーションでサインイン、IAM ロールを引き受ける、またはクロスアカウントロールを引き受けることができます。一時的なセキュリティ認証情報を取得するには、AssumeRole または GetFederationToken などの AWS STS API オペレーションを呼び出します。

API Gateway では、一時認証情報の使用をサポートしています。

サービスにリンクされたロール

サービスリンクロールは、AWS サービスが他のサービスのリソースにアクセスしてお客様の代わりにアクションを完了することを許可します。サービスリンクロールは IAM アカウント内に表示され、サービスによって所有されます。IAM 管理者は、サービスにリンクされたロールのアクセス許可を表示できますが、編集することはできません。

API Gateway は、サービスリンクロールをサポートします。API Gateway サービスにリンクされたロールの作成または管理の詳細については、「API Gateway でのサービスリンクロールの使用」を参照してください。

サービスロール

サービスは、ユーザーに代わってサービスロールを引き受けることができます。このサービスロールにより、サービスはユーザーに代わって他のサービスのリソースにアクセスし、アクションを完了できます。サービスロールロールは、IAM アカウントに表示され、アカウントによって所有されます。そのため、管理者はこのロールのアクセス権限を変更できます。ただし、それにより、サービスの機能が損なわれる場合があります。

API Gateway は、サービスロールをサポートします。

Amazon API Gateway のアイデンティティベースのポリシーの例

デフォルトでは、IAM ユーザーとロールには API Gateway リソースを作成または変更するためのアクセス許可はありません。AWS Management Console、AWS CLI、または AWS SDK を使用してタスクを実行することもできません。IAM 管理者は、ユーザーとロールに必要な、指定されたリソースで特定の API オペレーションを実行するアクセス許可をユーザーとロールに付与する IAM ポリシーを作成する必要があります。続いて、管理者はそれらのアクセス許可が必要な IAM ユーザーまたはグループにそのポリシーをアタッチします。

IAM ポリシーの作成方法の詳細については、 IAM ユーザーガイド の 「[JSON] タブでのポリシーの作成 」を参照してください。API Gateway 固有のアクション、リソース、条件の詳細については、

「Amazon API Gateway Management のアクション、リソース、および条件キー」および「Amazon API Gateway Management V2 のアクション、リソース、および条件キー」を参照してください。

トピック

- ポリシーのベストプラクティス
- ユーザーが自分の権限を表示できるようにする
- シンプルな読み取りアクセス許可
- REQUEST または JWT オーライザだけを作成する
- デフォルトの execute-api エンドポイントが無効になっていることを要求する
- ユーザーにプライベート REST API の作成または更新のみを許可する
- API ルートに認証を要求する
- ユーザーが VPC リンクを作成または更新することを防止する

ポリシーのベストプラクティス

ID ベースのポリシーは、あるユーザーがアカウント内で API Gateway リソースを作成、アクセス、または削除できるどうかを決定します。これらのアクションを実行すると、AWS アカウント に料金が発生する可能性があります。アイデンティティベースポリシーを作成したり編集したりする際には、以下のガイドラインと推奨事項に従ってください:

- AWS マネージドポリシーを使用して開始し、最小特権の権限に移行する – ユーザーとワークロードへの権限の付与を開始するには、多くの一般的なユースケースのために権限を付与する AWS マネージドポリシーを使用します。これらは AWS アカウントで使用できます。ユースケースに応じた AWS カスタマーマネージドポリシーを定義することで、権限をさらに減らすことをお勧めします。詳細については、『IAM ユーザーガイド』の「AWS マネージドポリシー」または「AWS ジョブ機能の管理ポリシー」を参照してください。
- 最小特権を適用する – IAM ポリシーで権限を設定するときは、タスクの実行に必要な権限のみを付与します。これを行うには、特定の条件下で特定のリソースに対して実行できるアクションを定義します。これは、最小特権権限とも呼ばれています。IAM を使用して権限を適用する方法の詳細については、『IAM ユーザーガイド』の「IAM でのポリシーと権限」を参照してください。
- IAM ポリシーで条件を使用してアクセスをさらに制限する - ポリシーに条件を追加して、アクションやリソースへのアクセスを制限できます。例えば、ポリシー条件を記述して、すべてのリクエストを SSL を使用して送信するように指定できます。また、AWS CloudFormation などの特定の AWS のサービスを介して使用する場合、条件を使ってサービスアクションへのアクセス権を付与

することもできます。詳細については、『IAM ユーザーガイド』の [IAM JSON policy elements: Condition] (IAM JSON ポリシー要素：条件) を参照してください。

- IAM Access Analyzer を使用して IAM ポリシーを検証し、安全で機能的な権限を確保する - IAM Access Analyzer は、新規および既存のポリシーを検証して、ポリシーが IAM ポリシー言語 (JSON) および IAM のベストプラクティスに準拠するようにします。IAM アクセスアナライザーは 100 を超えるポリシーチェックと実用的な推奨事項を提供し、安全で機能的なポリシーの作成をサポートします。詳細については、『IAM ユーザーガイド』の「IAM Access Analyzer ポリシーの検証」を参照してください。
- 多要素認証 (MFA) を要求する – AWS アカウント で IAM ユーザーまたはルートユーザーを要求するシナリオがある場合は、セキュリティを強化するために MFA をオンにします。API オペレーションが呼び出されるときに MFA を必須にするには、ポリシーに MFA 条件を追加します。詳細については、『IAM ユーザーガイド』の「MFA 保護 API アクセスの設定」を参照してください。

IAM でのベストプラクティスの詳細については、「IAM ユーザーガイド」の「IAM でのセキュリティベストプラクティス」を参照してください。

ユーザーが自分の権限を表示できるようにする

この例では、ユーザーアイデンティティにアタッチされたインラインおよびマネージドポリシーの表示を IAM ユーザーに許可するポリシーの作成方法を示します。このポリシーには、コンソールで、または AWS CLI か AWS API を使用してプログラム的に、このアクションを完了するアクセス許可が含まれています。

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

シンプルな読み取りアクセス許可

このポリシーの例は、AWS リージョン us-east-1 の a123456789 の識別子を使用して、HTTP または WebSocket API のすべてのリソースに関する情報を取得する許可をユーザーに付与します。リソース `arn:aws:apigateway:us-east-1::/apis/a123456789/*` には、認証者やデプロイなど、API のすべてのサブリソースが含まれます。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "apigateway:GET"
      ],
      "Resource": [
        "arn:aws:apigateway:us-east-1::/apis/a123456789/*"
      ]
    }
  ]
}
```

REQUEST または JWT オーライザだけを作成する

このポリシーの例を参考にすれば、ユーザーが インポートを含む、REQUEST または JWT オーソリザーだけを使用して API を作成できます。ポリシーの Resource セクションによると、arn:aws:apigateway:us-east-1::/apis/?????????? では、リソースは最大10 文字にする必要があります。これはAPI のサブリソースでは除外されます。この例では、ForAllValues セクションの Condition を使用します。これは、ユーザーが API をインポートすることで複数のオーソリザーを一度に作成できるためです。

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

デフォルトの **execute-api** エンドポイントが無効になっていることを要求する

このポリシーの例は、DisableExecuteApiEndpoint が trueの場合に、ユーザーが API を作成、更新、またはインポートすることを示してます。DisableExecuteApiEndpoint が true の

場合は、クライアントは API を呼び出すために、デフォルトの execute-api エンドポイントを使用できません。

BoolIfExists の条件を使用して、 DisableExecuteApiEndpoint 条件キーが入力されない API を更新する呼び出しを処理します。ユーザーが API を作成またはインポートしようとすると、DisableExecuteApiEndpoint 条件キーは常に入力されます。

apis/*リソースはオーソリザーやメソッドなどのサブリソースもキャプチャするため、明示的に Deny ステートメントを使用して API だけを範囲とします。

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

```
  ]
}
```

ユーザーにプライベート REST API の作成または更新のみを許可する

このポリシーの例は、ユーザーが PRIVATE API のみを作成することを必須とし、API を PRIVATE から別のタイプ (REGIONAL など) に変更する可能性がある更新が行われないように条件キーを使用します。

ForAllValues を使用して、API に追加されるすべての EndpointType が PRIVATE であることを義務付けています。API に対する更新はいずれも、それが PRIVATE の場合に限り、リソース条件キーを使用して許可されます。ForAllValues は、条件キーが存在する場合にのみ適用されます。

非貪欲マッチャー (?) を使用して、API IDと明示的に照合し、オーソリザーなどの非 API リソースを許可しないようにしています。

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

API ルートに認証を要求する

このポリシーにより、ルートに認証がない場合、ルートの作成または更新（インポートを含む）の試行が失敗します。ルートが作成または更新されていないときなど、キーが存在しないと、ForAnyValue は false と評価します。インポートによって複数のルートを作成することができるので、ForAnyValue を使用します。

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

ユーザーが VPC リンクを作成または更新することを防止する

このポリシーは、ユーザーが VPC リンクを作成または更新することを防止します。VPC リンクを使用すると、Amazon VPC 内のリソースを VPC 外のクライアントに公開できます。

```
{
  "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 のリソースベースのポリシーの例

リソースベースのポリシーの例については、「the section called “API Gateway リソースポリシーの例”」を参照してください。

Amazon API Gateway の ID とアクセスのトラブルシューティング

次の情報は、API Gateway と IAM の使用に伴って発生する可能性がある一般的な問題の診断や修復に役立ちます。

トピック

- API Gateway でアクションを実行する権限がない
- iam: PassRole を実行する権限がありません
- AWS アカウント外のユーザーに API Gateway リソースへのアクセスを許可したい

API Gateway でアクションを実行する権限がない

「I am not authorized to perform an action in Amazon Bedrock」というエラーが表示された場合、そのアクションを実行できるようにポリシーを更新する必要があります。

次のエラー例は、mateojackson IAM ユーザーがコンソールを使用して、ある *my-example-widget* リソースに関する詳細情報を表示しようとしたことを想定して、その際に必要なapigateway:*GetWidget* アクセス許可を持っていない場合に発生するものです。

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

この場合、apigateway:*GetWidget* アクションを使用して *my-example-widget*リソースへのアクセスを許可するように、mateojackson ユーザーのポリシーを更新する必要があります。

サポートが必要な場合は、AWS 管理者に問い合わせてください。管理者とは、サインイン認証情報を提供した担当者です。

iam: PassRole を実行する権限がありません

`iam:PassRole` アクションを実行する権限がないというエラーが表示された場合は、ポリシーを更新して API Gateway にロールを渡すことができるようにする必要があります。

一部の AWS のサービスでは、新しいサービスロールまたはサービスリンクロールを作成せずに、既存のロールをサービスに渡すことが許可されています。そのためには、サービスにロールを渡す権限が必要です。

以下の例のエラーは、`marymajor` という IAM ユーザーがコンソールを使用して API Gateway でアクションを実行しようする場合に発生します。ただし、このアクションをサービスが実行するには、サービスロールから付与された権限が必要です。Mary には、ロールをサービスに渡す権限がありません。

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

この場合、Mary のポリシーを更新してメアリーに `iam:PassRole` アクションの実行を許可する必要があります。

サポートが必要な場合は、AWS 管理者に問い合わせてください。管理者とは、サインイン認証情報を提供した担当者です。

AWS アカウント外のユーザーに API Gateway リソースへのアクセスを許可したい

他のアカウントのユーザーや組織外のユーザーが、リソースへのアクセスに使用できるロールを作成できます。ロールの引き受けを委託するユーザーを指定できます。リソースベースのポリシーまたはアクセス制御リスト (ACL) をサポートするサービスの場合、それらのポリシーを使用して、リソースへのアクセスを付与できます。

詳細については、以下を参照してください。

- API Gateway がこれらの機能をサポートしているかどうかについては、「Amazon API Gateway と IAM の連携方法」を参照してください。
- 所有している AWS アカウント全体のリソースへのアクセス権を提供する方法については、IAM ユーザーガイドの「所有している別の AWS アカウント アカウントへのアクセス権を IAM ユーザーに提供」を参照してください。

- サードパーティーの AWS アカウント にリソースへのアクセス権を提供する方法については、『IAM ユーザーガイド』の「第三者が所有する AWS アカウント へのアクセス権を付与する」を参照してください。
- ID フェデレーションを介してアクセスを提供する方法については、『IAM ユーザーガイド』の「外部で認証されたユーザー (ID フェデレーション) へのアクセス権限」を参照してください。
- クロスアカウントアクセスでのロールとリソースベースのポリシーの使用の違いの詳細については、『IAM ユーザーガイド』の「IAM ロールとリソースベースのポリシーとの相違点」を参照してください。

API Gateway でのサービスリンクロールの使用

Amazon API Gateway は、AWS Identity and Access Management (IAM) サービスリンクロールを使用します。サービスにリンクされたロールは、API Gateway に直接リンクされた一意のタイプの IAM ロールです。サービスリンクロールは API Gateway によって事前定義されており、サービスがユーザーに代わって AWS の他のサービスを呼び出すために必要なすべての許可が含まれています。

サービスにリンクされたロールを使用することで、必要なアクセス許可を手動で追加する必要がなくなるため、API Gateway の設定が簡単になります。API Gateway は、サービスにリンクされたロールのアクセス許可を定義します。別の指定がない限り、API Gateway のみがそのロールを引き受けることができます。定義される許可は、信頼ポリシーと許可ポリシーに含まれており、その許可ポリシーを他の IAM エンティティにアタッチすることはできません。

サービスにリンクされたロールを削除するには、まずその関連リソースを削除します。これにより、リソースに対するアクセス許可が誤って削除されることがなくなり、API Gateway リソースは保護されます。

サービスリンクロールをサポートする他のサービスについては、「IAM と連携する AWS のサービス」を参照して、[サービスにリンクされたロール] 列が[Yes] (はい) になっているサービスを探してください。サービスにリンクされたロールに関するドキュメントをサービスで表示するには、[はい] リンクを選択します。

API Gateway のサービスにリンクされたロールのアクセス許可

API Gateway は、AWSServiceRoleForAPIGateway という名前のサービスリンクロールを使用します。これにより、API Gateway がユーザーに代わって Elastic Load Balancing、Amazon Data Firehose、その他のサービスリソースにアクセスすることが許可されます。

AWSServiceRoleForAPIGateway サービスリンクロールは、以下のサービスを信頼してロールを引き受けます。

- ops.apigateway.amazonaws.com

ロールのアクセス許可ポリシーは、指定したリソースに対して以下のアクションを完了することを API Gateway に許可します。

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

サービスにリンクされたロールの作成、編集、削除を IAM エンティティ (ユーザー、グループ、ロールなど) に許可するには、アクセス許可を設定する必要があります。詳細については、IAM ユーザーガイドの「サービスにリンクされたロールの許可」を参照してください。

API Gateway のサービスにリンクされたロールの作成

サービスにリンクされたロールを手動で作成する必要はありません。AWS Management Console、AWS CLI、または AWS API で、API、カスタムドメイン名、または VPC リンクを作成すると、API Gateway がサービスリンクロールを作成します。

このサービスにリンクされたロールを削除した後で再度作成する必要が生じた場合は、同じ方法でアカウントにロールを再作成できます。API、カスタムドメイン名、または VPC リンクを作成すると、API Gateway によってサービスリンクされたロールが再び作成されます。

API Gateway のサービスにリンクされたロールの編集

API Gateway では、AWSServiceRoleForApigateWay サービスリンクロールを編集することはできません。サービスリンクロールを作成した後は、さまざまなエンティティがロールを参照する可能性があることから、ロール名を変更することはできません。ただし、IAM を使用したロール記述の編集はできます。詳細については、IAM ユーザーガイドのサービスにリンクされたロールの編集を参照してください。

API Gateway のサービスにリンクされたロールの削除

サービスにリンクされたロールが必要な機能またはサービスが不要になった場合には、そのロールを削除することをお勧めします。そうすることで、使用していないエンティティがアクティブにモニタリングされたり、メンテナンスされたりすることがなくなります。ただし、手動で削除する前に、サービスにリンクされたロールのリソースをクリーンアップする必要があります。

> **Note**
>
> リソースを削除する際に、API Gateway サービスでそのロールが使用されている場合、削除は失敗することがあります。失敗した場合は、数分待ってから再度オペレーションを実行してください。

AWSServiceRoleForApigateWay で使用される API Gateway リソースを削除するには

1. API Gateway (https://console.aws.amazon.com/apigateway) コンソールを開きます。
2. サービスにリンクされたロールを使用する API、カスタムドメイン名、または VPC リンクに移動します。
3. コンソールを使用してリソースを削除します。
4. この手順を繰り返して、サービスにリンクされたロールを使用するすべての API、カスタムドメイン名、または VPC リンクを削除します。

サービスにリンクされたロールを IAM で手動削除するには

IAM コンソール、AWS CLI、または AWS API を使用して、AWSServiceRoleForAPIGateway サービスリンクロールを削除します。詳細については、IAM ユーザーガイドの「サービスにリンクされたロールの削除」を参照してください。

API Gateway サービスリンクロールでサポートされるリージョン

API Gateway は、サービスを利用できるすべてのリージョンで、サービスにリンクされたロールの使用をサポートします。詳細については、「AWS サービスエンドポイント」を参照してください。

AWS 管理ポリシーに対する API Gateway の更新

API Gateway の AWS 管理ポリシーの更新に関する詳細を、このサービスがこれらの変更の追跡を開始した以降の分について表示します。このページの変更に関する自動通知については、API Gateway のドキュメントの履歴ページの RSS フィードを購読してください。

変更	説明	日付
`acm:GetCertificate` のサポートが `AWSServic`	`AWSServiceRoleForAPIGateway` ポリシーに	2021 年 7 月 12 日

変更	説明	日付
eRoleForAPIGateway ポリシーに追加されました。	ACM GetCertificate API アクションを呼び出すアクセス許可が含まれるようになりました。	
API Gateway が変更の追跡を開始しました	API Gateway が AWS 管理ポリシーの変更の追跡を開始しました。	2021 年 7 月 12 日

Amazon API Gateway でのログ記録とモニタリング

モニタリングは、API Gateway および AWS ソリューションの信頼性、可用性、およびパフォーマンスを維持する上で重要な役割を果たします。マルチポイント障害が発生した場合は、その障害をより簡単にデバッグできるように、AWS ソリューションのすべての部分からモニタリングデータを収集する必要があります。AWS は、API Gateway リソースをモニタリングし、潜在的なインシデントに対応するための複数のツールを提供します。

Amazon CloudWatch ログ

リクエストの実行や API へのクライアントアクセスに関連する問題のデバッグに役立てるために、CloudWatch Logs を有効にして API コールのログを記録できます。詳細については、「the section called “CloudWatch Logs”」を参照してください。

Amazon CloudWatch アラーム

Amazon CloudWatch アラームを使用して、指定した期間にわたって 1 つのメトリクスを確認します。メトリクスが特定のしきい値を超えると、Amazon Simple Notification Service のトピックまたは AWS Auto Scaling ポリシーに通知が送信されます。メトリクスが特定の状態にある場合、CloudWatch アラームはアクションを呼び出しません。状態が変わり、それが指定した期間だけ維持される必要があります。詳細については、「the section called “CloudWatch メトリクス”」を参照してください。

Firehose へのアクセスログ

API へのクライアントアクセスに関連する問題のデバッグに役立てるために、Firehose で API コールのログを記録できます。詳細については、「the section called “Firehose”」を参照してください。

AWS CloudTrail

CloudTrail は、API Gateway のユーザー、ロール、または AWS のサービスによって実行されたアクションの記録を提供します。CloudTrail で収集された情報を使用して、API Gateway に対するリクエスト、リクエスト元の IP アドレス、リクエストの実行者、リクエスト日時などの詳細を確認できます。詳細については、「the section called “CloudTrail の使用”」を参照してください。

AWS X-Ray

X-Ray は、アプリケーションが対応するリクエストに関するデータを収集する AWS のサービスで、このデータを使ってアプリケーションの問題や最適化の機会を識別するために使用できるサービスマップを作成します。詳細については、「the section called “AWS X-Ray のセットアップ”」を参照してください。

AWS Config

AWS Config は、アカウントにある AWS リソースの設定詳細ビューを提供します。リソース間の関係、設定変更の履歴、関係と設定の時間的な変化を確認できます。AWS Config を使用して、データコンプライアンスのためにリソース設定を評価するルールを定義できます。AWS Config ルールは、API Gateway リソースの理想的な設定を表します。リソースがルールに違反しており、非準拠としてフラグが付けられると、AWS Config は Amazon Simple Notification Service (Amazon SNS) トピックを使用してアラートを送信できます。詳細については、「the section called “AWS Config の使用”」を参照してください。

AWS CloudTrail を使用した Amazon API Gateway API コールのログ記録

Amazon API Gateway は、ユーザー、ロール、または AWS のサービス が実行したアクションの記録を提供するサービスである AWS CloudTrail と統合されています。CloudTrail は、API Gateway のすべての REST API コールをイベントとしてキャプチャします。キャプチャには、API Gateway コンソールからの呼び出しと API Gateway サービス API へのコード呼び出しが含まれます。CloudTrail で収集した情報を使用して、API Gateway へのリクエスト、リクエスト元の IP アドレス、リクエストの作成日時、その他の詳細を確認できます。

> **Note**
>
> TestInvokeAuthorizer と TestInvokeMethod は CloudTrail のログに記録されません。

各イベントまたはログエントリには、リクエストの生成者に関する情報が含まれます。アイデンティティ情報は、以下を判別するのに役立ちます:

- ルートユーザーまたはユーザー認証情報のどちらを使用してリクエストが送信されたか
- リクエストが IAM Identity Center ユーザーに代わって行われたかどうか。
- リクエストがロールまたはフェデレーションユーザーのテンポラリなセキュリティ認証情報を使用して行われたかどうか。
- リクエストが、別の AWS のサービス によって送信されたかどうか。

アカウントを作成すると、AWS アカウント で CloudTrail がアクティブになり、自動的に CloudTrail の[イベント履歴] にアクセスできるようになります。CloudTrail の [イベント履歴] では、AWS リージョン で過去 90 日間に記録された 管理イベントの表示、検索、およびダウンロードが可能で、変更不可能な記録を確認できます。詳細については、「AWS CloudTrail ユーザーガイド」の「CloudTrail イベント履歴の使用」を参照してください。[イベント履歴] の閲覧には CloudTrail の料金はかかりません。

AWS アカウント で過去 90 日間のイベントを継続的に記録するには、証跡または CloudTrail Lake イベントデータストアを作成します。

CloudTrail 証跡

証跡により、CloudTrail はログファイルを Amazon S3 バケットに配信できます。AWS Management Console を使用して作成した証跡はマルチリージョンです。AWS CLI を使用する際は、単一リージョンまたは複数リージョンの証跡を作成できます。アカウント内のすべて AWS リージョン でアクティビティを把握するため、マルチリージョン証跡を作成することをお勧めします。単一リージョンの証跡を作成する場合、証跡の AWS リージョン に記録されたイベントのみを表示できます。証跡の詳細については、「AWS CloudTrail ユーザーガイド」の「AWS アカウント の証跡の作成」および「組織の証跡の作成」を参照してください。

証跡を作成すると、進行中の管理イベントのコピーを 1 つ無料で CloudTrail から Amazon S3 バケットに配信できますが、Amazon S3 ストレージには料金がかかります。CloudTrail の料金の詳細については、「AWS CloudTrail の料金」を参照してください。Amazon S3 の料金に関する詳細については、「Amazon S3 の料金」を参照してください。

CloudTrail Lake イベントデータストア

CloudTrail Lake を使用すると、イベントに対して SQL ベースのクエリを実行できます。CloudTrail Lake は、行ベースの JSON 形式の既存のイベントを Apache ORC 形式に変換します。ORC は、データを高速に取得するために最適化された単票ストレージ形式です。イベントはイベントデータストアに集約されます。イベントデータストアは、高度なイベントセレクタを適用することによって選択する条件に基いた、イベントのイミュータブルなコレクションです。

どのイベントが存続し、クエリに使用できるかは、イベントデータストアに適用するセレクタが制御します。CloudTrail Lake の詳細については、「AWS CloudTrail ユーザーガイド」の「Lake の使用AWS CloudTrail」を参照してください。

CloudTrail Lake のイベントデータストアとクエリにはコストがかかります。イベントデータストアを作成する際に、イベントデータストアに使用する料金オプションを選択します。料金オプションによって、イベントの取り込みと保存にかかる料金、および、そのイベントデータストアのデフォルトと最長の保持期間が決まります。CloudTrail の料金の詳細については、「AWS CloudTrail の料金」を参照してください。

CloudTrail の API Gateway 管理イベント

管理イベントでは、AWS アカウント のリソースに対して実行される管理オペレーションについての情報が得られます。これらのイベントは、コントロールプレーンオペレーションとも呼ばれます。CloudTrail は、デフォルトで管理イベントをログ記録します。

Amazon API Gateway は、TestInvokeAuthorizer と TestInvokeMethod を除き、すべての API Gateway アクションを管理イベントとして記録します。API Gateway が CloudTrail に記録する Amazon API Gateway アクションのリストについては、「Amazon API Gateway API リファレンス」を参照してください。

API Gateway イベントの例

各イベントは任意の送信元からの単一のリクエストを表し、リクエストされた API オペレーション、オペレーションの日時、リクエストパラメータなどに関する情報を含みます。CloudTrail ログファイルは、パブリック API コールの順序付けられたスタックトレースではないため、イベントは特定の順序で表示されません。

次は、API Gateway の `GetResource` アクションを示す CloudTrail ログエントリの例です。

```
{
    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 ...
    ]
}
```

CloudTrail レコードの内容については、「AWS CloudTrail ユーザーガイド」の「CloudTrail record contents」を参照してください。

AWS Config による API Gateway API 設定のモニタリング

AWS Config を使用して、API Gateway API リソースに対して行われた設定の変更を記録し、リソースの変更に基づいて通知を送信することができます。API Gateway リソースの設定変更履歴を維持することは、運用上のトラブルシューティング、監査、およびコンプライアンスのユースケースに役立ちます。

AWS Config は、以下の変更を追跡することができます。

- 以下のような API ステージ設定

  - キャッシュクラスター設定
  - スロットル設定
  - アクセスログ設定
  - ステージに設定されたアクティブなデプロイ
- 以下のような API 設定
  - エンドポイント設定
  - バージョン
  - プロトコル
  - タグ

さらに、AWS Config ルール 機能を使用して設定ルールを定義し、これらのルールに対する違反を自動的に検出、追跡、および警告することができます。これらのリソース設定プロパティに対する変更を追跡することによって、変更によってトリガーされる AWS Config ルールを API Gateway リソースのために作成し、ベストプラクティスに照らしてリソース設定をテストすることもできます。

AWS Config コンソールか AWS Config を使用して、アカウントの AWS CLI を有効にすることができます。変更を追跡するリソースのタイプを選択してください。すべてのリソースタイプを記録するように AWS Config を既に設定している場合は、アカウントでこれらの API Gateway リソースが自動的に記録されます。AWS Config での Amazon API Gateway のサポートは、すべての AWS パブリックリージョンと AWS GovCloud (US) で利用できます。サポートされているリージョンの完全なリストについては、「AWS 全般のリファレンス」の「Amazon API Gateway のエンドポイントとクォータ」を参照してください。

トピック

- サポートされているリソースタイプ
- AWS Config のセットアップ
- API Gateway リソースを記録するための AWS Config の設定
- AWS Config コンソールでの API Gateway 設定の詳細の表示
- AWS Config ルールを使用した API Gateway リソースの評価

サポートされているリソースタイプ

以下の API Gateway リソースタイプは AWS Config と統合されており、AWS Config でサポートされる AWS リソースタイプとリソース関係に記載されています。

- `AWS::ApiGatewayV2::Api` (WebSocket および HTTP API)
- `AWS::ApiGateway::RestApi` (REST API)
- `AWS::ApiGatewayV2::Stage` (WebSocket および HTTP API ステージ)
- `AWS::ApiGateway::Stage` (REST API ステージ)

AWS Config の詳細については、AWS Config 開発者ガイドを参照してください。料金情報については、「AWS Config 料金表ページ」を参照してください。

> **Important**
>
> API のデプロイ後に以下の API プロパティのいずれかを変更した場合は、変更を反映するために API を再デプロイする必要があります。それ以外の場合、AWS Config コンソールに属性の変更が表示されますが、以前のプロパティ設定はまだ有効です。API のランタイム動作が変更されることはありません。
>
> - **`AWS::ApiGateway::RestApi`** – `binaryMediaTypes`, `minimumCompressionSize`, `apiKeySource`
> - **`AWS::ApiGatewayV2::Api`** – `apiKeySelectionExpression`

AWS Config のセットアップ

AWS Config を初めて設定するには、AWS Config デベロッパーガイドの以下のトピックを参照してください。

- コンソールによる AWS Config の設定
- AWS CLI による AWS Config のセットアップ

API Gateway リソースを記録するための AWS Config の設定

デフォルトでは、環境が実行されているリージョンで検出された、サポートされているすべてのタイプのリージョナルリソースについての設定の変更が AWS Config で記録されます。AWS Config をカスタマイズすることで、特定のリソースタイプのみの変更を記録するか、グローバルリソースの変更を記録できます。

リージョナルリソースとグローバルリソースの相違点、および AWS Config 設定のカスタマイズ手順の詳細については、「AWS Config で記録するリソースの選択」を参照してください。

AWS Config コンソールでの API Gateway 設定の詳細の表示

AWS Config コンソールを使用して API Gateway リソースを探し、それらの設定に関する現在および履歴的な詳細情報を取得できます。以下の手順は、API Gateway API に関する情報を見つける方法を示しています。

AWS config コンソールで API Gateway リソースを検索するには

1. AWS Config コンソールを開きます。
2. [リソース] を選択します。
3. [Resource (リソース)] のインベントリページで、[Resources (リソース)] を選択します。
4. [Resource type (リソースタイプ)] メニューを開き、APIGateway または APIGatewayV2 までスクロールし、1 つ以上の API Gateway リソースタイプを選択します。
5. [検索] を選択します。
6. AWS Config に表示されるリソースのリストからリソース ID を選択します。AWS Config では、選択したリソースに関する設定の詳細とその他の情報が表示されます。
7. 記録した設定の詳細全体を表示するには、[View Details (詳細を表示)] を選択します。

このページでリソースを検索して情報を表示する他の方法については、AWS デベロッパーガイドの「AWS Config リソースの設定および履歴の表示」を参照してください。

AWS Config ルールを使用した API Gateway リソースの評価

API Gateway リソースの理想的な設定を表す AWS Config ルールを作成できます。事前定義済みの AWS Config マネージドルールを使用するか、カスタムルールを定義することができます。AWS Config は、リソースの設定変更を継続的に追跡し、これらの変更がルールの条件に違反していないかどうかを確認します。AWS Config コンソールには、ルールとリソースのコンプライアンスステータスが表示されます。

リソースがルールに違反しており、非準拠としてフラグが付けられた場合、AWS Config は Amazon Simple Notification Service デベロッパーガイド (Amazon SNS) のトピックを使用してアラートを送信できます。これらの AWS Config アラートのデータをプログラム的に使用するには、Amazon SNS トピックの通知エンドポイントとして Amazon Simple Queue Service (Amazon SQS) キューを使用します。

ルールの設定と使用の詳細については、AWS Config デベロッパーガイドの「ルールでのリソースの評価」を参照してください。

Amazon API Gateway のコンプライアンスの検証

AWS のサービス が特定のコンプライアンスプログラムの範囲内にあるかどうかを確認するには、「コンプライアンスプログラム別の範囲」の「AWS のサービス」と「」の「AWS のサービス」を参照し、関心のあるコンプライアンスプログラムを選択してください。一般的な情報については、AWS コンプライアンスプログラム を参照してください。

AWS Artifact を使用して、サードパーティーの監査レポートをダウンロードできます。詳細については、「Downloading Reports in AWS Artifact」を参照してください。

AWS のサービスを使用する際のユーザーのコンプライアンス責任は、ユーザーのデータの機密性や貴社のコンプライアンス目的、適用される法律および規制によって決まります。AWS では、コンプライアンスに役立つ次のリソースを提供しています。

- セキュリティとコンプライアンスのクイックスタートガイド - これらのデプロイガイドでは、アーキテクチャ上の考慮事項について説明し、セキュリティとコンプライアンスに重点を置いたベースライン環境を AWS にデプロイするためのステップを示します。
- 「Amazon Web Services での HIPAA のセキュリティとコンプライアンスのためのアーキテクチャ」 - このホワイトペーパーは、企業が AWS を使用して HIPAA 対象アプリケーションを作成する方法を説明しています。

  > **Note**
  >
  > すべての AWS のサービスが HIPAA 適格であるわけではありません。詳細については、「HIPAA 対応サービスのリファレンス」を参照してください。

- AWS コンプライアンスのリソース – このワークブックおよびガイドのコレクションは、顧客の業界と拠点に適用されるものである場合があります。
- AWS Customer Compliance Guide — コンプライアンスの観点から見た責任共有モデルを理解できます。このガイドは、AWS のサービスを保護するためのベストプラクティスを要約したものであり、複数のフレームワーク (米国標準技術研究所 (NIST)、ペイメントカード業界セキュリティ標準評議会 (PCI)、国際標準化機構 (ISO) など) にわたるセキュリティ統制へのガイダンスがまとめられています。
- 「AWS Config デベロッパーガイド」の「ルールでのリソースの評価」 - AWS Config サービスは、自社のプラクティス、業界ガイドライン、および規制に対するリソースの設定の準拠状態を評価します。

- AWS Security Hub - この AWS のサービスは、AWS 内のセキュリティ状態の包括的なビューを提供します。Security Hub では、セキュリティコントロールを使用して AWS リソースを評価し、セキュリティ業界標準とベストプラクティスに対するコンプライアンスをチェックします。サポートされているサービスとコントロールのリストについては、「Security Hub のコントロールリファレンス」を参照してください。
- Amazon GuardDuty – この AWS のサービスは、環境をモニタリングして、疑わしいアクティビティや悪意のあるアクティビティがないか調べることで、AWS アカウント、ワークロード、コンテナ、データに対する潜在的な脅威を検出します。GuardDuty を使用すると、特定のコンプライアンスフレームワークで義務付けられている侵入検知要件を満たすことで、PCI DSS などのさまざまなコンプライアンス要件に対応できます。
- AWS Audit Manager - この AWS のサービスは、AWS の使用状況を継続的に監査して、リスクの管理方法や、規制および業界標準へのコンプライアンスの管理方法を簡素化するために役立ちます。

Amazon API Gateway の耐障害性

AWS のグローバルインフラストラクチャは、AWS リージョンとアベイラビリティーゾーンを中心として構築されています。AWSリージョンには、低レイテンシー、高いスループット、そして高度の冗長ネットワークで接続されている複数の物理的に独立および隔離されたアベイラビリティーゾーンがあります。アベイラビリティーゾーンでは、ゾーン間で中断することなく自動的にフェイルオーバーするアプリケーションとデータベースを設計および運用することができます。アベイラビリティーゾーンは、従来の単一または複数のデータセンターインフラストラクチャよりも可用性、耐障害性、および拡張性が優れています。

AWS リージョンとアベイラビリティーゾーンの詳細については、「AWS グローバルインフラストラクチャ」を参照してください。

API へのリクエストが過度に集中しないように、API Gateway は API へのリクエストを調整します。特に API Gateway では、アカウントのすべての API に送信されるリクエストの定常レートとバーストに対して、リージョンごとに制限を設定します。API のカスタムスロットリングを設定できます。詳細については、「API リクエストを調整してスループットを向上させる」を参照してください。

Route 53 ヘルスチェックを使用して、プライマリリージョンの API Gateway API からセカンダリリージョンの API Gateway API への DNS フェイルオーバーを制御できます。例については、「the section called “DNS フェイルオーバー”」を参照してください。

Amazon API Gateway のインフラストラクチャセキュリティ

マネージドサービスである Amazon API Gateway は、AWS グローバルネットワークセキュリティで保護されています。AWS セキュリティサービスと AWS がインフラストラクチャを保護する方法については、「AWS クラウドセキュリティ」を参照してください。インフラストラクチャセキュリティのベストプラクティスを使用して AWS 環境を設計するには、「セキュリティの柱 - AWS Well-Architected Framework」の「インフラストラクチャ保護」を参照してください。

AWS が公開した API 呼び出しを使用して、ネットワーク経由で API Gateway にアクセスします。クライアントは以下をサポートする必要があります:

- Transport Layer Security (TLS)。TLS 1.2 は必須で TLS 1.3 がお勧めです。
- DHE (楕円ディフィー・ヘルマン鍵共有) や ECDHE (楕円曲線ディフィー・ヘルマン鍵共有) などの完全前方秘匿性 (PFS) による暗号スイート。これらのモードは、Java 7 以降など、ほとんどの最新システムでサポートされています。

また、リクエストには、アクセスキー ID と、IAM プリンシパルに関連付けられているシークレットアクセスキーを使用して署名する必要があります。または、AWS Security Token Service (AWS STS) を使用して、一時的なセキュリティ認証情報を生成し、リクエストに署名することもできます。

これらの API オペレーションは任意のネットワークの場所から呼び出すことができますが、API Gateway ではリソースベースのアクセスポリシーがサポートされているため、ソース IP アドレスに基づく制限を含めることができます。また、リソースベースのポリシーを使用して、特定の Amazon Virtual Private Cloud (Amazon VPC) エンドポイントまたは特定の VPC からのアクセスを制御することもできます。これにより、実質的に AWS ネットワーク内の特定の VPC からのみ特定の API Gateway リソースへのネットワークアクセスが分離されます。

Amazon API Gateway の脆弱性分析

構成および IT 管理は、AWS、お客様および弊社のお客様の間で共有される責任です。詳細については、AWS 責任共有モデルを参照してください。

Amazon API Gateway のセキュリティのベストプラクティス

API Gateway には、独自のセキュリティポリシーを作成および実装する際に役立ついくつかのセキュリティ機能が用意されています。以下のベストプラクティスは一般的なガイドラインであり、完

全なセキュリティソリューションに相当するものではありません。これらのベストプラクティスはお客様の環境に適切ではないか、十分ではない場合があるため、これらは処方箋ではなく、有用な考慮事項と見なしてください。

最小特権アクセスの実装

IAM ポリシーを使用して、API Gateway の API の作成、読み取り、更新、削除について最小権限のアクセスを実装します。詳細については、「Amazon API Gateway の Identity and Access Management」を参照してください。API Gateway には、作成した API へのアクセスをコントロールするためのオプションがいくつか用意されています。詳細については、「API Gateway での REST API へのアクセスの制御と管理API Gateway での WebSocket API へのアクセスの制御と管理」と「」を参照してください。JWT オーソライザーを使用した HTTP API へのアクセスの制御

ログ記録の実装

CloudWatch Logs または Amazon Data Firehose を使用して、API へのリクエストをログに記録します。詳細については、「REST API のモニタリングWebSocket API の口グ記録の設定」と「」を参照してください。HTTP API の口グ記録の設定

Amazon CloudWatch のアラームの実装

Amazon CloudWatch アラームを使用して、指定した期間にわたって 1 つのメトリクスを確認します。メトリクスが特定のしきい値を超えると、Amazon Simple Notification Service のトピックまたは AWS Auto Scaling ポリシーに通知が送信されます。メトリクスが特定の状態にある場合、CloudWatch アラームはアクションを呼び出しません。その代わり、状態が変更され、指定期間にわたって維持される必要があります。詳細については、「the section called “CloudWatch メトリクス”」を参照してください。

の有効化AWS CloudTrail

CloudTrail は、API Gateway のユーザー、ロール、または AWS のサービスによって実行されたアクションの記録を提供します。CloudTrail で収集された情報を使用して、API Gateway に対するリクエスト、リクエスト元の IP アドレス、リクエストの実行者、リクエスト日時などの詳細を確認できます。詳細については、「the section called “CloudTrail の使用”」を参照してください。

の有効化AWS Config

AWS Config は、アカウントにある AWS リソースの設定詳細ビューを提供します。リソース間の関係、設定変更の履歴、関係と設定の時間的な変化を確認できます。AWS Config を使用して、データコンプライアンスのためにリソース設定を評価するルールを定義できます。AWS Config ルールは、API Gateway リソースの理想的な設定を表します。リソースがルールに違

反しており、非準拠としてフラグが付けられると、AWS Config は Amazon Simple Notification Service (Amazon SNS) トピックを使用してアラートを送信できます。詳細については、「the section called “AWS Config の使用”」を参照してください。

AWS Security Hub を使用します。

AWS Security Hub を使用して、セキュリティのベストプラクティスに関連する API Gateway の使用状況をモニタリングします。Security Hub は、セキュリティコントロールを使用してリソース設定とセキュリティ標準を評価し、お客様がさまざまなコンプライアンスフレームワークに準拠できるようサポートします。Security Hub を使用して API Gateway リソースを評価する方法の詳細については、「AWS Security Hub ユーザーガイド」の「Amazon API Gateway コントロール」を参照してください。

API Gateway リソースのタグ付け

タグとは、ユーザーまたは AWS が AWS リソースに割り当てるメタデータラベルです。各タグは 2 つの部分で構成されます。

- タグキー (例： `CostCenter`、`Environment`、または `Project`)。タグキーでは、大文字と小文字が区別されます。
- タグ値として知られるオプションのフィールド (例： `111122223333` または `Production`)。タグ値を省略すると、空の文字列を使用した場合と同じになります。タグキーとタグ値は大文字と小文字が区別されます。

タグは、以下のことに役立ちます。

- リソースに割り当てられたタグに基づいて、リソースへのアクセスをコントロールします。AWS Identity and Access Management (IAM) ポリシーの条件でキーと値を指定することによってアクセスをコントロールします。タグベースのアクセスコントロールの詳細については、IAM ユーザーガイドの「タグを使用したアクセスのコントロール」を参照してください。
- AWS のコストの追跡。これらのタグは、AWS Billing and Cost Management ダッシュボードで有効にします。AWS では、タグを使用してコストを分類し、毎月のコスト配分レポートを提供します。詳細については、AWS Billing ユーザーガイドの「コスト配分タグの使用」を参照してください。
- AWS リソースの特定と整理。多くの AWS のサービスではタグ付けがサポートされるため、さまざまなサービスからリソースに同じタグを割り当てて、リソースの関連を示すことができます。たとえば、CloudWatch イベントルールに割り当てる同じタグを API Gateway ステージに割り当てることができます。

タグの使用に関するヒントについては、ホワイトペーパー「AWS タグ付け戦略」を参照してください。

以下のセクションでは、Amazon API Gateway のタグに関する詳細を示します。

トピック

- タグ付けできる API Gateway リソース
- タグを使用して API Gateway REST API リソースへのアクセスをコントロールする

タグ付けできる API Gateway リソース

タグは、Amazon API Gateway V2 API の次の HTTP API または WebSocket APIリソースで設定できます。

- Api
- DomainName
- Stage
- VpcLink

また、タグは Amazon API Gateway V1 API の次の REST API リソースに設定することができます。

- ApiKey
- ClientCertificate
- DomainName
- RestApi
- Stage
- UsagePlan
- VpcLink

タグを他のリソースに直接設定することはできません。ただし、Amazon API Gateway V1 API では、子リソースは親リソースに設定されているタグを継承します。例:

- タグが RestApi リソースに設定されている場合、そのタグは、属性ベースのアクセスコントロールの RestApi の次の子リソースに継承されます:
  - Authorizer
  - Deployment
  - Documentation
  - GatewayResponse
  - Integration
  - Method
  - Model
  - Resource

- ResourcePolicy
- Setting
- Stage

- タグが DomainName に設定されている場合、そのタグはその下の BasePathMapping リソースに継承されます。
- タグが UsagePlan に設定されている場合、そのタグはその下の UsagePlanKey リソースに継承されます。

> **Note**
>
> タグの継承は、属性ベースのアクセスコントロールにのみ適用されます。例えば、継承されたタグを使用して AWS Cost Explorer でコストを監視することはできません。API Gateway は、リソースの GetTags の呼び出し時に継承されたタグを返しません。

Amazon API Gateway V1 API でのタグの継承

以前は、タグをステージにしか設定できませんでした。他のリソースにも設定できるようになったため、Stage は 2 つの方法でタグを受け取ることができます。

- Stage でタグが直接設定される。
- ステージがその親 RestApi からタグを継承する。

ステージが両方の方法でタグを受け取った場合は、ステージで直接設定されているタグが優先されます。たとえば、ステージがその親 REST API から以下のタグを継承するとします。

```
{
 'foo': 'bar',
 'x':'y'
}
```

ステージで以下のタグも直接設定されているとします。

```
{
 'foo': 'bar2',
 'hello': 'world'
```

```
}
```

最終的に、ステージは以下のタグと値を受け取ることになります。

```
{
 'foo': 'bar2',
 'hello': 'world'
 'x':'y'
}
```

タグの制限事項と使用規則

API Gateway リソースでのタグの使用には、以下の制限事項と使用規則が適用されます。

- 各リソースには、最大 50 個のタグを設定できます。
- タグキーは、リソースごとにそれぞれ一意である必要があります。また、各タグキーに設定できる値は 1 つのみです。
- タグキーの最大長は UTF-8 で 128 Unicode 文字です。
- タグ値の最大長は UTF-8 で 256 Unicode 文字です。
- キーと値に使用できる文字は、UTF-8 対応の文字、数字、スペースと、文字 (. : + = @ _ / -) (ハイフン) です。Amazon EC2 リソースでは、任意の文字を使用できます。
- タグのキーと値は大文字と小文字が区別されます。ベストプラクティスとして、タグを大文字にするための戦略を決定し、その戦略をすべてのリソースタイプにわたって一貫して実装します。たとえば、`Costcenter`、`costcenter`、`CostCenter` のいずれを使用するかを決定し、すべてのタグに同じ規則を使用します。大文字と小文字の扱いについて、同様のタグに整合性のない規則を使用することは避けてください。
- プレフィックス `aws:` はタグで使用することはできません。AWS 用に予約されています。このプレフィックスが含まれるタグのキーや値を編集したり削除することはできません。このプレフィックスを持つタグは、リソースあたりのタグ数の制限には計算されません。

タグを使用して API Gateway REST API リソースへのアクセスをコントロールする

AWS Identity and Access Management ポリシーの条件は、API Gateway リソースに対するアクセス許可を指定するために使用する構文の一部です。IAM ポリシーの指定の詳細については、「the

section called “IAM アクセス許可を使用する”」を参照してください。API Gateway では、リソースにタグを付けることができ、一部のアクションにタグを含めることができます。IAM ポリシーを作成するときに、タグ条件キーを使用して以下をコントロールできます。

- どのユーザーが API Gateway リソースに対してアクションを実行できるか (リソースにすでに付けられているタグに基づいて)。
- どのタグをアクションのリクエストで渡すことができるか。
- リクエストで特定のタグキーを使用できるかどうか。

属性ベースのアクセスコントロールを使用すると、API レベルのコントロールよりもきめ細かく、リソースベースのアクセスコントロールよりも動的なコントロールが可能になります。リクエストで渡されたタグ (リクエストタグ) に基づいて、または操作するリソースに付けられたタグ (リソースタグ) に基づいて、オペレーションを許可または禁止する、IAM ポリシーを作成できます。一般に、リソースタグは、既存のリソースに使用します。リクエストタグは、新しいリソースの作成時に使用します。

タグ条件キーの完全な構文とセマンティクスについては、IAM ユーザーガイドの「タグを使用したアクセスコントロール」を参照してください。

以下の例は、API Gateway ユーザー用のポリシーでタグ条件を指定する方法を示しています。

リソースタグに基づいてアクションを制限する

次のポリシー例では、値が prod である Environment タグがリソースにない限り、すべてのリソースですべてのアクションを実行するアクセス許可がユーザーに付与されます。

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

```
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/Environment": "prod"
        }
      }
    }
  ]
}
```

リソースタグに基づいてアクションを許可する

次のポリシー例では、値が Development である Environment タグがリソースにある限り、API Gateway リソースですべてのアクションの実行がユーザーに許可されます。この Deny ステートメントは、ユーザーが Environment タグの値を変更することを防ぎます。

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

タグ付けオペレーションを拒否する

次のポリシー例では、タグの変更を除くすべての API Gateway アクションの実行がユーザーに許可されます。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "apigateway:*"
      ],
      "Resource": [
        "*"
      ],
    },
    {
      "Effect": "Deny",
      "Action": [
        "apigateway:*"
      ],
      "Resource": "arn:aws:apigateway:*::/tags*",
    }
  ]
```

```
}
```

タグ付けオペレーションを許可する

次のポリシー例では、すべての API Gateway リソースの取得と、それらのリソースのタグの変更がユーザーに許可されます。リソースのタグを取得するには、ユーザーはそのリソースに対する GET アクセス許可を持っている必要があります。リソースのタグを更新するには、ユーザーはそのリソースに対する PATCH アクセス許可を持っている必要があります。

```
{
  "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 リファレンス

Amazon API Gateway は、独自の HTTP および WebSocket API を作成およびデプロイするための API を提供しています。さらに、API Gateway API は標準の AWS SDK で使用できます。

使用している言語に対応した AWS SDK がある場合は、API Gateway REST API を直接使用する代わりに、SDK を使用することをお勧めします。SDK を利用すると、認証が簡素化され、開発環境との統合が容易になり、API Gateway コマンドに簡単にアクセスできます。

AWS SDK および API Gateway REST API リファレンスドキュメントの参照先を以下に示します。

- Amazon Web Services のツール
- Amazon API Gateway REST API リファレンス
- Amazon API Gateway WebSocket および HTTP API リファレンス

Amazon API Gateway のクォータと重要な注意点

トピック

特に明記されていない限り、クォータはリクエストに応じて引き上げることができます。クォータの引き上げをリクエストするには、Service Quotas を使用するか、AWS サポートセンターにお問い合わせください。

認可がメソッドに対して有効になっていると、メソッドの ARN (たとえば、`arn:aws:execute-api:{region-id}:{account-id}:{api-id}/{stage-id}/{method}/{resource}/{path}`) の最大長は 1,600 バイトです。パスパラメータ値 (ランタイムにサイズが決定される) により、ARN の長さが制限を超過する可能性があります。この場合、API クライアントは `414 Request URI too long` レスポンスを受け取ります。

> **Note**
>
> これによって、リソースポリシーが使用される場合の URI の長さが制限されます。リソースポリシーが必要となるプライベート API の場合は、すべてのプライベート API の URI の長さが制限されます。

API Gateway アカウントレベルのクォータ (リージョンごと)

以下のクォータは、アカウントごと、Amazon API Gateway のリージョンごとに適用されます。

リソースまたはオペレーション	デフォルトのクォータ	引き上げ可能
HTTP API、REST API、WebSocket API、WebSocket コールバック API にわたってリージョンごとに適用されるアカウント単位のスロットリングクォータ	10,000 リクエスト/秒 (RPS) と、トークンバケットアルゴリズムで提供される追加のバースト容量 (最大バケット容量は 5,000 リクエスト *)。 **Note** バーストクォータは、リージョンのアカウントの全体的な RPS クォータに基づいて、API Gateway サービスチームによって決定されます。このクォータは顧客が制御したり変更をリクエストできるものではありません。	はい
リージョンの API	600	いいえ
エッジ最適化 API	120	いいえ

* 次のリージョンでは、デフォルトのスロットリングクォータは 2500 RPS、デフォルトのバーストクォータは 1250 RPS です: アフリカ (ケープタウン)、欧州 (ミラノ)、アジアパシフィック (ジャカルタ)、中東 (アラブ首長国連邦)、アジアパシフィック (ハイデラバード)、アジアパシフィック (メルボルン)、欧州 (スペイン)、欧州 (チューリッヒ)、イスラエル (テルアビブ)、カナダ西部 (カルガリー)。

HTTP API クォータ

次のクォータは、API Gateway での HTTP API の設定と実行に適用されます。

リソースまたはオペレーション	デフォルトのクォータ	引き上げ可能
API あたりのルート数	300	はい
API あたりの統合の数	300	いいえ

リソースまたはオペレーション	デフォルトのクォータ	引き上げ可能
最大統合タイムアウト	30 秒	いいえ
API あたりのステージ数	10	[Yes (はい)]
ドメインごとのマルチレベル API マッピング	200	いいえ
ステージごとのタグ	50	いいえ
リクエスト行とヘッダー値の合計サイズ	10240 バイト	いいえ
ペイロードサイズ	10 MB	いいえ
1 アカウント、1 リージョンあたりのカスタムドメイン	120	はい
アクセスログテンプレートのサイズ	3 KB	いいえ
Amazon CloudWatch Logs のログエントリ	1 MB	いいえ
API ごとのオーソライザー数	10	はい
オーソライザーあたりの対象者数	50	いいえ

リソースまたはオペレーション	デフォルトのクォータ	引き上げ可能
ルートあたりのスコープ数	10	いいえ
JSON ウェブキーセットエンドポイントのタイムアウト	1500 ms	いいえ
JSON Web Key Set エンドポイントからのレスポンスサイズ	150,000 バイト	いいえ
OpenID Connect 検出エンドポイントのタイムアウト	1500 ms	いいえ
Lambda オーソライザーレスポンスのタイムアウト	10000 ミリ秒	いいえ
1 アカウント、1 リージョンあたりの VPC リンク	10	はい
VPC リンクあたりのサブネットの数	10	はい
ステージごとのステージ変数	100	いいえ
ステージ変数内のキー長さ (文字数)	64	いいえ
ステージ変数内の値の長さ (文字数)	512	いいえ

WebSocket API の設定と実行に関する API Gateway クォータ

Amazon API Gateway での WebSocket API の設定と実行には、次のクォータが適用されます。

リソースまたはオペレーション	デフォルトのクォータ	引き上げ可能
1 秒あたり、1 アカウント (すべての WebSocket API 間) あたり、1 リージョンあたりの新しい接続数	500	はい
同時接続	対象外	該当しません
API ごとの AWS Lambda オーソライザー	10	はい
AWS Lambda オーソライザー結果サイズ	8 KB	いいえ
API あたりのルート数	300	はい
API あたりの統合の数	300	はい
統合のタイムアウト	Lambda、Lambda プロキシ、HTTP、HTTP プロキシ、AWS 統合など、すべての統合タイプで 50 ミリ秒〜29 秒。	いいえ
API あたりのステージ数	10	はい
WebSocket のフレームサイズ	32 KB	いいえ

リソースまたはオペレーション	デフォルトのクォータ	引き上げ可能
メッセージのペイロードサイズ	128 KB **	いいえ
WebSocket API の接続時間	2 時間	いいえ
アイドル接続のタイムアウト	10 分	いいえ
WebSocket API の URL の長さ (文字数)	4096	いいえ

* API Gateway は、同時接続にクォータを適用しません。同時接続の最大数は、1 秒あたりの新規接続数と 2 時間の最大接続時間によって決まります。たとえば、デフォルトのクォータが 500 新規接続/秒の場合、クライアントが 2 時間を超える最大速度で接続する場合、API Gateway は最大 3,600,000 の同時接続を提供することができます。

** WebSocket フレームサイズには 32 KB のクォータがあるため、32 KB を超えるメッセージはそれぞれが 32 KB 以下の複数のフレームに分割する必要があります。これは `@connections` のコマンドにも該当します。それより大きなメッセージ (または大きなフレームサイズ) が受信された場合、接続は 1009 コードで閉じられます。

REST API の設定および実行に関する API Gateway クォータ

Amazon API Gateway での REST API の設定と実行には、次のクォータが適用されます。restapi:import または restapi:put では、API 定義ファイルの最大サイズは、6 MB です。

API あたりのすべてのクォータは、特定の API でのみ引き上げることができます。

リソースまたはオペレーション	デフォルトのクォータ	引き上げ可能
1 アカウント、1 リージョンあたり	120	はい

リソースまたはオペレーション	デフォルトのクォータ	引き上げ可能
のカスタムドメイン名		
ドメインごとのマルチレベル API マッピング	200	いいえ
エッジ最適化 API の URLの長さ (文字数)	8192	いいえ
リージョン API の URL の長さ (文字数)	10240	いいえ
1 アカウント、1 リージョンあたりのプライベート API	600	いいえ
API Gateway リソースポリシーの長さ (文字単位)	8192	はい
1 アカウント、1 リージョンあたりの API キー	10000	いいえ
1 アカウント、1 リージョンあたりのクライアント証明書	60	はい

リソースまたはオペレーション	デフォルトのクォータ	引き上げ可能
API ごとのオーソライザー (AWS Lambda および Amazon Cognito)	10	はい
API あたりのドキュメント部分	2000	はい
API あたりのリソース数	300	はい
API あたりのステージ数	10	はい
ステージごとのステージ変数	100	いいえ
ステージ変数内のキー長さ (文字数)	64	いいえ
ステージ変数内の値の長さ (文字数)	512	いいえ
1 アカウント、1 リージョンあたりの使用量プラン	300	はい
API キーあたりの使用量プラン	10	はい
1 アカウント、1 リージョンあたりの VPC リンク	20	はい

リソースまたはオペレーション	デフォルトのクォータ	引き上げ可能
API キャッシュの TTL	デフォルトで 300 秒、API 所有者が 0～3600 の範囲で設定可能。	上限あり (3600)
キャッシュされたレスポンスサイズ	1048576 バイト。キャッシュデータの暗号化により、キャッシュされるアイテムのサイズが増える可能性があります。	いいえ
統合のタイムアウト	Lambda、Lambda プロキシ、HTTP、HTTP プロキシ、AWS 統合など、すべての統合タイプで 50 ミリ秒～29 秒。	はい *
すべてのヘッダー値の合計サイズ	10,240 バイト	いいえ
プライベート API のすべてのヘッダー値の合計サイズ	8000 バイト	いいえ
ペイロードサイズ	10 MB	いいえ
ステージごとのタグ	50	いいえ
マッピングテンプレートでの `#foreach ... #end` ループ内での反復回数	1000	いいえ
認証があるメソッドの ARN の長さ	1600 バイト	いいえ
使用量プランのステージに対するメソッドレベルのスロットリング設定	20	はい

リソースまたはオペレーション	デフォルトのクォータ	引き上げ可能
API あたりのモデルサイズ	400 KB	いいえ
トラストストアの証明書数	1,000 件の証明書 (合計オブジェクトサイズは最大 1 MB)	いいえ

* 統合タイムアウトを 50 ミリ秒未満に設定することはできません。リージョン API とプライベート API の統合タイムアウトは 29 秒超に設定できますが、これに伴ってアカウントレベルのスロットリングクォータ制限の緩和が必要になる場合があります。

API の作成、デプロイ、管理に関する API Gateway クォータ

以下の固定クォータは、AWS CLI、API Gateway コンソール、または API Gateway REST API とその SDK を使用して、API Gateway で API の作成、デプロイ、および管理に適用されます。これらのクォータを増やすことはできません。

アクション	デフォルトのクォータ	引き上げ可能
CreateApiKey	アカウントあたり 1 秒ごとに 5 リクエスト	いいえ
CreateDeployment	アカウントあたり 5 秒ごとに 1 リクエスト	いいえ
CreateDocumentationVersion	アカウントあたり 20 秒ごとに 1 リクエスト	いいえ
CreateDomainName	アカウントあたり 30 秒ごとに 1 リクエスト	いいえ
CreateResource	アカウントあたり 1 秒ごとに 5 リクエスト	いいえ

アクション	デフォルトのクォータ	引き上げ可能
CreateRestApi	リージョン API またはプライベート API • アカウントあたり 3 秒ごとに 1 リクエスト エッジ最適化 API • アカウントあたり 30 秒ごとに 1 リクエスト	いいえ
CreateVpcLink (V2)	アカウントあたり 15 秒ごとに 1 リクエスト	いいえ
DeleteApiKey	アカウントあたり 1 秒ごとに 5 リクエスト	いいえ
DeleteDomainName	アカウントあたり 30 秒ごとに 1 リクエスト	いいえ
DeleteResource	アカウントあたり 1 秒ごとに 5 リクエスト	いいえ
DeleteRestApi	アカウントあたり 30 秒ごとに 1 リクエスト	いいえ
GetResources	アカウントあたり 2 秒ごとに 5 リクエスト	いいえ
DeleteVpcLink (V2)	アカウントあたり 30 秒ごとに 1 リクエスト	いいえ
ImportDocumentationParts	アカウントあたり 30 秒ごとに 1 リクエスト	いいえ

アクション	デフォルトのクォータ	引き上げ可能
ImportRestApi	リージョン API またはプライベート API • アカウントあたり 3 秒ごとに 1 リクエスト エッジ最適化 API • アカウントあたり 30 秒ごとに 1 リクエスト	いいえ
PutRestApi	アカウントあたり 1 秒ごとに 1 リクエスト	いいえ
UpdateAccount	アカウントあたり 20 秒ごとに 1 リクエスト	いいえ
UpdateDomainName	アカウントあたり 30 秒ごとに 1 リクエスト	いいえ
UpdateUsagePlan	アカウントあたり 20 秒ごとに 1 リクエスト	いいえ
その他のオペレーション	合計アカウントクォータまでのクォータはありません。	いいえ
合計オペレーション数	1 秒あたり 10 リクエストで、バーストクォータは 1 秒あたり 40 リクエスト。	いいえ

Amazon API Gateway に関する重要な注意点

トピック

- Amazon API Gateway の REST API、HTTP API、WebSocket API に関する重要な注意点
- Amazon API Gateway の REST および WebSocket API に関する重要な注意点

- Amazon API Gateway の WebSocket API に関する重要な注意点
- Amazon API Gateway の REST API に関する重要な注意点

Amazon API Gateway の REST API、HTTP API、WebSocket API に関する重要な注意点

- 署名バージョン 4A は Amazon API Gateway では公式にサポートされていません。

Amazon API Gateway の REST および WebSocket API に関する重要な注意点

- API Gateway では、REST と WebSocket API 間でのカスタムドメイン名の共有はサポートされていません。
- ステージ名には、英数字、ハイフン、およびアンダースコアのみ含めることができます。最大長は 128 文字です。
- `/ping` および `/sping` のパスは、サービスのヘルスチェック専用です。カスタムドメインを使用した API ルートレベルでの使用は良い結果を生みません。
- API Gateway は現在、ログイベントを 1024 バイトに制限しています。リクエストやレスポンスの本文など、1024 バイトを超えるログイベントは、CloudWatch Logs への送信前に API Gateway によって切り捨てられます。
- 現在、CloudWatch メトリクスではディメンション名と値が 255 文字の有効な XML 文字に制限されています (詳細については、CloudWatch ユーザーガイドを参照してください)。ディメンション値は、API 名、ラベル (ステージ) 名、およびリソース名を含む、ユーザーが定義した名前の関数です。これらの名前を選択するときは、CloudWatch メトリクスの制限を超えないように注意してください。
- マッピングテンプレートの最大サイズは 300 KB です。

Amazon API Gateway の WebSocket API に関する重要な注意点

- API Gateway は最大 128 KB までのメッセージペイロードをサポートし、最大フレームサイズは 32 KB です。メッセージが 32 KB を超えた場合は、それぞれが 32 KB 以下の複数のフレームに分割する必要があります。大きなメッセージが受信された場合、接続は 1009 コードで閉じられます。

Amazon API Gateway の REST API に関する重要な注意点

- プレーンテキストパイプ文字 (|) はリクエスト URL クエリ文字列にサポートされておらず、URL エンコードされている必要があります。
- セミコロン文字 (;) はリクエスト URL クエリ文字列に対してサポートされておらず、この結果、データが分割されます。
- REST API は、URL エンコードされたリクエストパラメータをデコードしてからバックエンド統合に渡します。UTF-8 リクエストパラメータの場合、REST API は、パラメータをデコードしてから、バックエンド統合にユニコードとして渡します。
- API Gateway コンソールを使用して API をテストするとき、バックエンドに自己署名証明書が存在する、中間証明書が証明書チェーンにない、または他の認識されない証明書関連の例外がバックエンドからスローされた場合に「不明なエンドポイントのエラー」レスポンスが返されることがあります。
- プライベート統合の API `Resource` または `Method` エンティティの場合は、`VpcLink` のハードコーディングされたリファレンスを削除した後で、それを削除する必要があります。それ以外の場合は、ダングリング統合があり、`Resource` または `Method` エンティティが削除されても VPC リンクがまだ使用中であることを示すエラーが表示されます。プライベート統合がステージ変数を通じて `VpcLink` を参照する場合、この動作は適用されません。
- 以下のバックエンドでは、API Gateway と互換性のある方法では SSL クライアント認証がサポートされない場合があります。
  - NGINX
  - Heroku
- API Gateway は、ほとんどの OpenAPI 2.0 仕様と OpenAPI 3.0 仕様をサポートしますが、次の例外があります。
  - パスセグメントには、英数字、アンダースコア、ハイフン、ピリオド、カンマ、コロン、中括弧のみを含めることができます。パスパラメータは、別のパスセグメントである必要があります。たとえば、「resource/{path_parameter_name}」は有効です。「resource{path_parameter_name}」は無効です。
  - モデル名は、英数字のみ含めることができます。
  - 入力パラメータについては、次の属性のみがサポートされています。`name`、`in`、`required`、`type`、`description`。他の属性は無視されます。
  - `securitySchemes` タイプを使用する場合は、`apiKey` にする必要があります。ただし、OAuth 2 および HTTP Basic 認証は Lambda オーソライザーによってサポートされています。OpenAPI 設定は、ベンダー拡張機能によって実現されます。

- この deprecated フィールドはサポートされておらず、エクスポートされた API では削除されます。
- API Gateway モデルは、OpenAPI が使用する JSON スキーマではなく、JSON スキーマのドラフト 4 を使用して定義されます。
- discriminatorパラメータは、どのスキーマオブジェクトでもサポートされません。
- example タグはサポートされません。
- exclusiveMinimum は API Gateway ではサポートされていません。
- 単純なリクエストの検証には maxItems タグと minItems タグが含まれません。この問題を回避するには、インポートした後で、検証を行う前にモデルを更新します。
- oneOf は、OpenAPI 2.0 または SDK の生成ではサポートされていません。
- readOnly フィールドはサポートされません。
- $ref を使用して他のファイルを参照することはできません。
- "500": {"$ref": "#/responses/UnexpectedError"} フォームのレスポンス定義は、OpenAPI ドキュメントルートではサポートされません。回避策として、参照をインラインスキーマに置き換えます。
- Int32 タイプまたは Int64 タイプの数値はサポートされていません。例を以下に示します。

```
"elementId": {
    "description": "Working Element Id",
    "format": "int32",
    "type": "number"
}
```

- 10 進数型 ("format": "decimal") は、スキーマ定義でサポートされません。
- メソッドレスポンスでは、スキーマ定義をオブジェクト型にする必要があり、プリミティブ型にすることはできません。たとえば、"schema": { "type": "string"} はサポートされません。ただし、回避策として次のオブジェクト型を使用できます。

```
"schema": {
     "$ref": "#/definitions/StringResponse"
           }

 "definitions": {
    "StringResponse": {
      "type": "string"
    }
```

```
  }
```

- API Gateway は、OpenAPI 仕様に定義されているルートレベルのセキュリティを使用しません。したがって、オペレーションレベルでセキュリティを定義して、セキュリティが適切に適用されるようにする必要があります。
- `default` のキーワードはサポートされていません。

- API Gateway は、Lambda 統合または HTTP 統合を使用した処理方法に以下の制約と制限を設定しています。
  - ヘッダー名とクエリパラメータは大文字と小文字を区別する方法で処理されます。
  - 以下の表には、統合エンドポイントに送信されたり、統合エンドポイントから戻ったりしたときに、ドロップされたり、再マップされたり、それ以外の場合は変更されることがあるヘッダーがリストされています。この表で以下の点に注意してください。
    - `Remapped` は、ヘッダー名が *<string>* から `X-Amzn-Remapped-`*<string>* に変更されたことを意味します。

      `Remapped Overwritten` は、ヘッダー名が *<string>* から `X-Amzn-Remapped-`*<string>* に変更され、値が上書きされたことを意味します。

ヘッダー名	リクエスト (**http/http_proxy /lambda**)	レスポンス (**http/http_proxy /lambda**)
`Age`	パススルー	パススルー
`Accept`	パススルー	除外/[Passthrough]/[Passthrough]
`Accept-Charset`	パススルー	パススルー

ヘッダー名	リクエスト (**http/http_proxy /lambda**)	レスポンス (**http/http_proxy /lambda**)
Accept-Encoding	パススルー	パススルー
Authorization	パススルー *	Remapped
Connection	パススルー/パススルー/除外	Remapped
Content-Encoding	パススルー/除外/パススルー	パススルー
Content-Length	パススルー (本文に基づいて生成された)	パススルー
Content-MD5	除外	Remapped
Content-Type	パススルー	パススルー
Date	パススルー	上書きされ再マッピングされた
Expect	除外	除外
Host	統合エンドポイントに上書きされます	除外
Max-Forwards	除外	Remapped
Pragma	パススルー	パススルー

ヘッダー名	リクエスト (**http/http_proxy /lambda**)	レスポンス (**http/http_proxy /lambda**)
Proxy-Authenticate	除外	除外
Range	パススルー	パススルー
Referer	パススルー	パススルー
Server	除外	上書きされ再マッピングされた
TE	除外	除外
Transfer-Encoding	除外/除外/例外	除外
Trailer	除外	除外
Upgrade	除外	除外
User-Agent	パススルー	Remapped
Via	除外/除外/パススルー	パススルー/除外/除外
Warn	パススルー	パススルー

ヘッダー名	リクエスト (**http/http_proxy /lambda**)	レスポンス (**http/http_proxy /lambda**)
WWW-Authenticate	除外	Remapped

* Signature Version 4 の署名が含まれている場合、または AWS_IAM 認証が使用されている場合、Authorization ヘッダーは削除されます。

- API Gateway で生成された API の Android SDK では、java.net.HttpURLConnection クラスを使用します。このクラスは、Android 4.4 以前を実行するデバイスでは、WWW-Authenticate ヘッダーから X-Amzn-Remapped-WWW-Authenticate への再マッピングに伴う 401 レスポンスに対して、処理されない例外をスローします。
- API Gateway で生成された Java や、API の Android および iOS SDK とは異なり、API Gateway で生成された API の JavaScript SDK では、500 レベルのエラーの再試行はサポートされていません。
- メソッドのテスト呼び出しでは、application/json のデフォルトコンテンツタイプを使用し、その他のコンテンツタイプの使用は無視されます。
- X-HTTP-Method-Override ヘッダーを渡して API にリクエストを送信すると、API Gateway によってメソッドがオーバーライドされます。したがって、ヘッダーをバックエンドに渡すには、ヘッダーを統合リクエストに追加する必要があります。
- リクエストの Accept ヘッダーに複数のメディアタイプが含まれている場合、API Gateway は最初の Accept メディアタイプのみ受け入れます。Accept メディタイプの順序を制御できず、バイナリコンテンツのメディアタイプがリストの先頭にない場合、API の Accept リストに最初の binaryMediaTypes メディアタイプを追加できます。 API Gateway によりコンテンツがバイナリとして返されます。たとえば、ブラウザで <img> 要素を使用して JPEG ファイルを送信するため、ブラウザがリクエストで Accept:image/webp,image/*,*/*;q=0.8 を送信することがあります。image/webp を binaryMediaTypes リストに追加することで、エンドポイントは JPEG ファイルをバイナリとして受け取るようになります。
- 現在、413 REQUEST_TOO_LARGE のデフォルトゲートウェイレスポンスのカスタマイズはサポートされていません。
- API Gateway には、すべての統合レスポンスの Content-Type ヘッダーが含まれています。デフォルトでは、コンテンツタイプは application/json です。

ドキュメント履歴

次の表に、Amazon API Gateway の前回のリリース以後に行われた、ドキュメントの重要な変更を示します。このドキュメントの更新に関する通知については、トップメニューパネルの RSS ボタンを選択して RSS フィードにサブスクライブします。

- 最新のドキュメント更新: 2024 年 2 月 15 日

変更	説明	日付
TLS 1.3 のサポートを追加	API Gateway は、リージョン REST API、HTTP API、WebSocket API で TLS 1.3 をサポートするようになりました。詳細については、「API Gateway でのカスタムドメインのセキュリティポリシーの選択」を参照してください。	2024 年 2 月 15 日
REST API および WebSocket API コンソールの更新	REST API と WebSocket API のコンソール情報を更新しました。	2023 年 12 月 10 日
ドキュメントの更新	API Gateway REST API のデータ変換とリクエスト検証のトピックに関する概念情報を更新し、新しいチュートリアルを作成しました。詳細については、「API Gateway でリクエストの検証を使用する」と「REST API のデータ変換の設定」を参照してください。	2023 年 6 月 22 日

マルチリージョン API Gateway の DNS フェイルオーバーの設定	Amazon Route 53 ヘルスチェックを使用して、プライマリ AWS リージョンの API Gateway REST API からセカンダリリージョンの API Gateway REST API への DNS フェイルオーバーを制御するためのサポートを追加しました。詳細については、「DNS フェイルオーバーのカスタムヘルスチェックの設定」を参照してください。	2022 年 10 月 31 日
ドキュメントの更新	REST API と HTTP API の API に関するコア機能の概要を更新しました。詳細については、「REST API と HTTP API 間で API を選択する」を参照してください。	2022 年 5 月 31 日
管理ポリシーの更新	`acm:GetCertificate` のサポートが `AWSServiceRoleForAPIGateway` ポリシーに追加されました。詳細については、「API Gateway でのサービスリンクロールの使用」を参照してください。	2021 年 7 月 12 日
HTTP API のパラメータマッピング	HTTP API のパラメータマッピングのサポートが追加されました。詳細については、「API リクエストとレスポンスの変換」を参照してください。	2021 年 1 月 7 日

REST API のデフォルトのエンドポイントを無効にする	REST API のデフォルトのエンドポイントを無効にするサポートが追加されました。詳細については、「REST API のデフォルトエンドポイントを無効にする」を参照してください。	2020 年 10 月 29 日
相互 TLS 認証	REST API と HTTP API の相互 TLS 認証のサポートが追加されました。詳細については、「REST API の相互 TLS 認証の設定」および「HTTP API の相互 TLS 認証の設定」を参照してください。	2020 年 9 月 17 日
HTTP API AWS Lambda オーソライザー	HTTP API で AWS Lambda オーソライザーのサポートが追加されました。詳細については、「HTTP API の AWS Lambda オーソライザーの使用」を参照してください。	2020 年 9 月 9 日
HTTP API AWS のサービスの統合	HTTP API の AWS のサービス統合のサポートが追加されました。詳細については、「HTTP API の AWS のサービス統合の使用」を参照してください。	2020 年 8 月 20 日
HTTP API ワイルドカードカスタムドメイン	HTTP API でワイルドカードカスタムドメイン名のサポートが追加されました。詳細については、「ワイルドカードカスタムドメイン名」を参照してください。	2020 年 8 月 10 日

サーバーレス開発者ポータルの改善	管理者パネルにユーザー管理を追加し、API 定義のエクスポートをサポートします。詳細については、「サーバーレス開発者ポータルを使用して API Gateway API を分類する」を参照してください。	2020 年 6 月 25 日
WebSocket API の Sec-WebSocket-Protocol サポート	Sec-WebSocket-Protocol フィールドのサポートを追加しました。詳細については、「IWebSocket サブプロトコルを必要とする $connect ルートの設定」を参照してください。	2020 年 6 月 16 日
HTTP API エクスポート	HTTP API の OpenAPI 3.0 定義のエクスポートのサポートが追加されました。詳細については、「API Gateway からの HTTP API のエクスポート」を参照してください。	2020 年 4 月 20 日
セキュリティドキュメント	セキュリティドキュメントを追加しました。詳細については、「Amazon API Gateway のセキュリティ」を参照してください。	2020 年 3 月 31 日
ドキュメントが再編されました	開発者ガイドが再編されました。	2020 年 3 月 12 日
HTTP API の一般提供	一般提供の HTTP API をリリースしました。詳細については、「HTTP API の使用」を参照してください。	2020 年 3 月 12 日

HTTP API ログ記録	HTTP API ログでの `$context.integrationErrorMessage` のサポートが追加されました。詳細については、「HTTP API ログ記録変数」を参照してください。	2020 年 2 月 26 日
AWSOpenAPI インポート用の変数	OpenAPI 定義の AWS 変数のサポートを追加しました。詳細については、「OpenAPI インポート用の AWS 変数」を参照してください。	2020 年 2 月 17 日
HTTP API	HTTP API のベータ版をリリースしました。詳細については、「HTTP API」を参照してください。	2019 年 12 月 4 日
ワイルドカードカスタムドメイン名	ワイルドカードカスタムドメイン名のサポートが追加されました。詳細については、「ワイルドカードカスタムドメイン名」を参照してください。	2019 年 10 月 21 日
Amazon Data Firehose でのログ記録	アクセスログデータの送信先として Amazon Data Firehose のサポートを追加しました。詳細については、「API Gateway アクセスログ記録の送信先として Amazon Data Firehose を使用する」を参照してください。	2019 年 10 月 15 日

プライベート API を呼び出すための Route53 エイリアス	プライベート API を呼び出すための追加の Route53 エイリアス DNS レコードのサポートを追加しました。詳細については、「Route53 エイリアスを使用したプライベート API へのアクセス」を参照してください。	2019 年 9 月 18 日
WebSocket API 向けのタグベースのアクセスコントロール	WebSocket API 向けのタグベースのアクセスコントロールのサポートを追加しました。詳細については、「タグ付けできる API Gateway リソース」を参照してください。	2019 年 6 月 27 日
カスタムドメインの TLS バージョンの選択	カスタムドメインにデプロイされている API に対して Transport Layer Security (TLS) バージョンの選択のサポートを追加しました。「API Gateway のカスタムドメインの最小 TLS バージョンの選択」の注意を参照してください。	2013 年 6 月 20 日

プライベート API 用の VPC エンドポイントポリシー	エンドポイントポリシーをインターフェイスの VPC エンドポイントにアタッチすることで、プライベート API のセキュリティを向上させるためのサポートを追加しました。詳細については、「API Gateway のプライベート API 用に VPC エンドポイントポリシーを使用する」を参照してください。	2019 年 6 月 4 日
ドキュメントの更新	「Amazon API Gateway の開始方法」を書き換えました。チュートリアルを Amazon API Gateway のチュートリアルに移動しました。	2019 年 5 月 29 日
REST API 向けのタグベースのアクセスコントロール	REST API 向けのタグベースのアクセスコントロールのサポートを追加しました。詳細については、「IAM ポリシーでタグを使用して API Gateway リソースへのアクセスを制御する」を参照してください。	2019 年 5 月 23 日

ドキュメントの更新	6 つのトピック:「Amazon API Gateway とは」、「チュートリアル: HTTP プロキシ統合を使用して API をビルドする」、「チュートリアル : 3 つの非プロキシ統合を使用して Calc REST API を作成する」、「API Gateway のマッピングテンプレートとアクセスのログ記録の変数リファレンス」、「API Gateway Lambda オーソライザーの使用」、および「API Gateway REST API リソースの CORS を有効にする」を書き換えました。	2019 年 4 月 5 日
サーバーレス開発者ポータルの改善	Amazon API Gateway 開発者ポータルで API を発行しやすくするために管理者パネルおよびその他の改善が追加されました。詳細については、「サーバーレス開発者ポータルを使用して API を分類する」を参照してください。	2019 年 3 月 28 日
AWS Config のサポート	AWS Config のサポートが追加されました。詳細については、「AWS Config による API Gateway API 設定のモニタリング」を参照してください。	2019 年 3 月 20 日

AWS CloudFormation のサポート	API Gateway V2 API を AWS CloudFormation テンプレートリファレンスに追加しました。詳細については、「Amazon API Gateway V2 リソースタイプリファレンス」を参照してください。	2019 年 2 月 7 日
WebSocket API のサポート	WebSocket API のサポートを追加しました。詳細については、「Amazon API Gateway での WebSocket API の作成」を参照してください。	2018 年 12 月 18 日
サーバーレスデベロッパーポータルが次を経由して利用可能にAWS Serverless Application Repository	Amazon API Gateway デベロッパーポータルのサーバーレスアプリケーションが、AWS Serverless Application Repository (GitHub に加えて) から利用できるようになりました。詳細については、「サーバーレス開発者ポータルを使用して API Gateway API を分類する」を参照してください。	2018 年 11 月 16 日
AWS WAF のサポート	AWS WAF (ウェブアプリケーションファイアウォール) のサポートを追加しました。詳細については、「AWS WAF を使用した API へのアクセスの制御」を参照してください。	2018 年 11 月 5 日

サーバーレス開発者ポータル	Amazon API Gateway は、API Gateway API を公開するためにデプロイできるサーバーレスアプリケーションとして、完全にカスタマイズ可能な開発者ポータルを提供します。詳細については、「サーバーレス開発者ポータルを使用して API Gateway API を分類する」を参照してください。	2018 年 10 月 29 日
複数値のヘッダーとクエリ文字列パラメータのサポート	Amazon API Gateway は、同じ名前を持つ複数のヘッダーやクエリ文字列パラメータをサポートするようになりました。詳細については、「複数値のヘッダーとクエリ文字列パラメータのサポート」を参照してください。	2018 年 10 月 4 日
OpenAPI のサポート	Amazon API Gatewayは、OpenAPI (Swagger) 2.0と同様に OpenAPI 3.0 をサポートするようになりました。	2018 年 9 月 27 日
ドキュメントの更新	新しいトピックの「Amazon API Gateway リソースポリシーが認証ワークフローに与える影響」が追加されました。	2018 年 9 月 27 日

AWS X-Ray のアクティブな統合	AWS X-Ray を使用して、API を経由して基盤となるサービスにユーザーリクエストが流れる際に、そのレイテンシーを追跡し、分析できるようになりました。詳細については、「AWS X-Ray を使用した API Gateway API 実行のトレース」を参照してください。	2018 年 9 月 6 日
キャッシュの改善	API ステージに対してキャッシュを有効にすると、デフォルトでは GET メソッドのみでキャッシュが有効になります。これは、API の安全性を確保するのに役立ちます。オーバーライドするメソッドの設定により、他のメソッドのキャッシュを有効にできます。詳細については、「API キャッシュを有効にして応答性を強化する」を参照してください。	2018 年 8 月 20 日
サービスの制限の改訂	いくつかの制限が改訂されています。アカウントごとの API の数が増加しました。Create/Import/Deploy API の API レート制限が増加しました。いくつかのレートは 1 分あたりから 1 秒あたりに修正されました。詳細については、「制限」を参照してください。	2018 年 7 月 13 日

API リクエストとレスポンスパラメータとヘッダーのオーバーライド	レスポンスヘッダーとステータスコードと同様に、リクエストヘッダー、クエリ文字列、およびパスのオーバーライドのサポートが追加されました。詳細については、「マッピングテンプレートを使用して、API のリクエストおよびレスポンスパラメータとヘッダーをオーバーライドする」を参照してください。	2018 年 7 月 12 日
使用量プランのメソッドレベルのスロットリング	使用量プラン設定の個々の API メソッドのスロットリング制限と同様に、メソッドあたりのスロットリング制限のデフォルト設定のサポートが追加されました。これらの設定は、ステージ設定で設定できる既存のアカウントレベルのスロットリングとデフォルトのメソッドレベルのスロットリング制限に加えられます。詳細については、「API リクエストを調整してスループットを向上させる」を参照してください。	2018 年 7 月 11 日

API Gateway 開発者ガイドの更新通知が RSS から利用可能になりました	HTML 版の API Gateway 開発者ガイドで、このドキュメント履歴ページに説明されている更新の RSS フィードがサポートされるようになりました。RSS フィードには、2018 年 6 月 27 日以降に行われた更新が含まれています。以前に発表された更新は、このページで引き続き利用できます。このフィードにサブスクライブするには、トップメニューパネルの RSS ボタンを使用します。	2018 年 27 月 6 日

以前の更新

次の表は、2018 年 6 月 27 日以前の API Gateway デベロッパーガイドの各リリースにおける重要な変更点を説明するものです。

変更	説明	変更日
プライベート API	プライベート API のサポートが追加されました。これは、インターフェイス VPC エンドポイントを介して公開します。プライベート API へのトラフィックは、Amazon のネットワークの外には出ません。パブリックインターネットからは隔離されています。	2018 年 14 月 6 日
クロスアカウント Lambda 認証と統合およびクロスアカウント Amazon Cognito ユーザープールオーソライザー	Lambda オーソライザー関数または API 統合バックエンドとして、別の AWS アカウントから AWS Lambda 関数を使用します。または、Amazon Cognito ユーザープールをオーソライザーとして使用します。もう 1 つのアカウントは、Amazon API Gateway を利用できるリージョンであればどのリージョンでもかまいません。詳細については、the section called “クロスアカウントの Lambda オーソライザーを設定する”、the section called “チュートリアル: クロ	2018 年 4 月 2 日

変更	説明	変更日
	スアカウントの Lambda プロキシ統合を使用して API をビルドする”、およびthe section called “REST API 用のクロスアカウントの Amazon Cognito オーソライザーを設定する”を参照してください。	
API のリソースポリシー	別の AWS アカウントのユーザーが、API に安全にアクセスするか、指定された送信元 IP アドレス範囲または CIDR ブロックからのみ API を呼び出すことを許可するには、API Gateway リソースポリシーを使用します。詳細については、「the section called “API Gateway リソースポリシーの使用”」を参照してください。	2018 年 4 月 2 日
API ゲートウェイリソースのタグ付け	API リクエストのコスト配分と API Gateway でのキャッシングのために、API ステージに最大 50 個のタグを付けます。詳細については、「the section called “タグをセットアップする”」を参照してください。	2017 年 12 月 19 日
ペイロードの圧縮と解凍	サポートされているコンテンツコーディングのいずれかを使用して圧縮ペイロードで API を呼び出すことができます。本文マッピングテンプレートが指定されている場合、圧縮されたペイロードはマッピングの対象となります。詳細については、「the section called “コンテンツのエンコーディング”」を参照してください。	2017 年 12 月 19 日
カスタムオーソライザーから発生した API キー	カスタムオーソライザーから API Gateway に API キーを返して、キーを必要とする API メソッドの使用量プランを適用します。詳細については、「the section called “API キーのソースを選択する”」を参照してください。	2017 年 12 月 19 日
OAuth 2 スコープを使用した認証	`COGNITO_USER_POOLS` オーソライザーで、OAuth 2 スコープを使用してメソッド呼び出しの認証を有効にします。詳細については、「the section called “REST API のオーソライザーとして Amazon Cognito ユーザープールを使用する”」を参照してください。	2017 年 14 月 12 日

変更	説明	変更日
プライベート統合と VPC リンク	API Gateway プライベート統合を使用して、クライアントに VpcLink リソースを通じて VPC の外部から Amazon VPC 内の HTTP/HTTPS リソースへのアクセスを提供する API を作成します。詳細については、the section called “チュートリアル: プライベート統合を使用して API をビルドする”およびthe section called “プライベート統合”を参照してください。	2017 年 11 月 30 日
API テストの Canary のデプロイ	既存の API デプロイに Canary リリースを追加して、同じステージで現在のバージョンを維持しながら、新しいバージョンの API をテストします。Canary リリースのステージトラフィックの割合を設定し、個別の CloudWatch Logs ログに記録された Canary 固有の実行とアクセスを有効にすることができます。詳細については、「the section called “Canary リリースのデプロイをセットアップする”」を参照してください。	2017 年 11 月 28 日
アクセスのログ記録	$context 変数 から派生したデータを使用して、選択した形式でクライアントアクセスを API に記録します。詳細については、「the section called “CloudWatch Logs”」を参照してください。	2017 年 11 月 21 日
API の Ruby SDK	API 用の Ruby SDK を生成し、それを使用して API メソッドを呼び出します。詳細については、the section called “API の Ruby SDK の生成”およびthe section called “API Gateway によって生成された Ruby SDK を REST API で使用する”を参照してください。	2017 年 11 月 20 日

変更	説明	変更日
リージョン API エンドポイント	リージョン API エンドポイントを指定して、非モバイルクライアント用の API を作成します。EC2 インスタンスなどの非モバイルクライアントは、API がデプロイされている同じ AWS リージョンで実行されます。エッジ最適化 API と同様に、リージョン API のカスタムドメイン名を作成できます。詳細については、the section called “API Gateway エンドポイントタイプ”および the section called “リージョン別カスタムドメイン名の設定”を参照してください。	2017 年 11 月 2 日
カスタムリクエストオーソライザー	カスタムリクエストオーソライザーを使用して、リクエストパラメータにユーザー認証情報を追加して、API メソッド呼び出しを認証します。リクエストパラメータには、ヘッダーやクエリ文字列パラメータだけでなく、ステージ変数やコンテキスト変数が含まれます。詳細については、「API Gateway Lambda オーソライザーを使用する」を参照してください。	2017 年 9 月 15 日
ゲートウェイレスポンスのカスタマイズ	API リクエストが統合バックエンドに到達しなかった場合に API Gateway で生成されるゲートウェイレスポンスをカスタマイズします。ゲートウェイのメッセージをカスタマイズすることで、API 固有のカスタムエラーメッセージを発信者に提供できます。たとえば、必要な CORS ヘッダーを返したり、ゲートウェイレスポンスデータを外部交換の形式に変換したりできます。詳細については、「エラーレスポンスをカスタマイズするためのゲートウェイレスポンスのセットアップ」を参照してください。	2017 年 6 月 6 日
Lambda カスタムエラープロパティをメソッドレスポンスのヘッダーにマッピングする	`integration.response.body` パラメータを使用して、Lambda から返される個々のカスタムエラープロパティをメソッドレスポンスのヘッダーパラメータにマッピングします。文字列化されたカスタムエラーオブジェクトは、API Gateway でランタイムに逆シリアル化されます。詳細については、「API Gateway でカスタム Lambda エラーを処理する」を参照してください。	2017 年 6 月 6 日

変更	説明	変更日
スロットリング制限の引き上げ	アカウントレベルのリクエストの定常レート制限を 10,000 リクエスト/秒 (rps) に引き上げ、バースト制限を 5,000 同時リクエストに引き上げます。詳細については、「API リクエストを調整してスループットを向上させる」を参照してください。	2017 年 6 月 6 日
メソッドリクエストの検証	API レベルまたはメソッドレベルで基本的なリクエストの検証を設定し、API Gateway が受信リクエストを検証できるようにします。API Gateway は、必須のパラメータが設定されていて空白でないことを確認し、該当するペイロードが設定されたモデルに従っていることを確認します。詳細については、「API Gateway でリクエストの検証を使用する」を参照してください。	2017 年 4 月 11 日
ACM との統合	API のカスタムドメイン名には ACM 証明書を使用します。AWS Certificate Manager で証明書を作成するか、既存の PEM 形式の証明書を ACM にインポートすることができます。その後、API のカスタムドメイン名を設定するとき、証明書の ARN を参照します。詳細については、「REST API のカスタムドメイン名を設定する」を参照してください。	2017 年 3 月 9 日
API の Java SDK の生成と呼び出し	API Gateway が API の Java SDK を生成できるようにし、SDK を使用して Java クライアントで API を呼び出します。詳細については、「REST API 用に API Gateway で生成された Java SDK を使用する」を参照してください。	2017 年 1 月 13 日
AWS Marketplace との統合	AWS Marketplace を通じて、使用量プランで API を SaaS 製品として販売します。AWS Marketplace を使用して API のリーチを拡大します。顧客への請求は AWS Marketplace が自動的に行います。API Gateway がユーザー認証と使用量計測を処理できるようにします。詳細については、「SaaS としての API の販売」を参照してください。	2016 年 12 月 1 日

変更	説明	変更日
API のドキュメントサポートの有効化	API Gateway の DocumentationPart リソースに、API エンティティのドキュメントを追加します。コレクション `DocumentationPart` インスタンスのスナップショットを API ステージに関連付け、DocumentationVersion を作成します。ドキュメントバージョンを Swagger ファイルなどの外部ファイルにエクスポートすることで、API ドキュメントを発行します。詳細については、「REST API をドキュメント化する」を参照してください。	2016 年 12 月 1 日
更新されたカスタムオーソライザー	カスタムオーソライザー Lambda 関数が、発信者のプリンシパル ID を返すようになりました。この関数は、`context` マップと IAM ポリシーのキー/値ペアとして他の情報を返すこともできます。詳細については、「API Gateway Lambda オーソライザーからの出力」を参照してください。	2016 年 12 月 1 日
バイナリペイロードのサポート	API で binaryMediaTypes を設定し、リクエストまたはレスポンスのバイナリペイロードをサポートします。Integration または IntegrationResponse で `contentHandling` プロパティを設定し、ネイティブバイナリ BLOB、Base64 でエンコードされた文字列、または変更なしのパススルーのいずれのものとしてバイナリペイロードを処理するかを指定します。詳細については、「REST API のバイナリメディアタイプの使用」を参照してください。	2016 年 11 月 17 日
API のプロキシリソースを通じた HTTP または Lambda バックエンドとのプロキシ統合の有効化。	`{proxy+}` の形式の greedy パスパラメータとキャッチオールの `ANY` メソッドを使用したプロキシリソースの作成。プロキシリソースは、HTTP または Lambda プロキシ統合を使用して、それぞれ HTTP または Lambda バックエンドと統合されます。詳細については、「プロキシリソースとのプロキシ統合を設定する」を参照してください。	2016 年 9 月 20 日

変更	説明	変更日
1 つ以上の使用プランを提供し、お客様への提供商品として、API Gateway で選択した API を拡張します。	承認済みのリクエストレートとクォータで、選択した API クライアントの特定の API ステージへのアクセスを有効にする API Gateway の使用プランを作成します。詳細については、「API キーを使用した使用量プランの作成と使用」を参照してください。	2016 年 8 月 11 日
Amazon Cognito のユーザープールを使用したメソッドレベルの認証の有効化	Amazon Cognito のユーザープールを作成し、ID プロバイダーとして使用します。ユーザープールに登録されたユーザーにアクセス付与するメソッドレベルの認証としてユーザープールを設定できます。詳細については、「Amazon Cognito ユーザープールをオーソライザーとして使用して REST API へのアクセスを制御する」を参照してください。	2016 年 7 月 28 日
AWS/ApiGateway 名前空間で Amazon CloudWatch のメトリクスとディメンションを有効にします。	API Gateway のメトリクスが、AWS/ApiGateway の CloudWatch 名前空間で標準化されるようになりました。API Gateway コンソールと Amazon CloudWatch コンソールの両方で表示できます。詳細については、「Amazon API Gateway のディメンションとメトリクス」を参照してください。	2016 年 7 月 28 日
カスタムドメイン名の証明書の更新を有効にする	証明書の更新により、カスタムドメイン名の証明書の有効期限が切れる場合に、証明書をアップロードして更新することができます。詳細については、「ACM にインポートされた証明書を更新」を参照してください。	2016 年 4 月 27 日
更新された Amazon API Gateway コンソールの変更を文書化します。	更新された API Gateway コンソールを使用して API を設定する方法を説明します。詳細については、チュートリアル: サンプルをインポートして REST API を作成するおよびチュートリアル: HTTP 非プロキシ統合を使用して REST API をビルドするを参照してください。	2016 年 4 月 5 日

変更	説明	変更日
API のインポート機能を有効にして、外部の API 定義から新しい API を作成するか、既存の API を有効にします。	API のインポート機能により、新しい API を作成するか、Swagger 2.0 で表された外部の API 定義を API Gateway 拡張機能でアップロードして、既存の API を更新できます。API のインポートの詳細については、「OpenAPI を使用した REST API の設定」を参照してください。	2016 年 4 月 5 日
`$input.body` 変数を公開して、raw ペイロードに文字列としてアクセスします。また、`$util.parseJson()` 関数にアクセスして、マッピングテンプレートで JSON 文字列を JSON オブジェクトに変換します。	`$input.body` と `$util.parseJson()` の詳細については、「API Gateway マッピングテンプレートとアクセスのログ記録の変数リファレンス」を参照してください。	2016 年 4 月 5 日
メソッドレベルのキャッシュ無効化のクライアントリクエストを有効にして、リクエストのスロットリング管理を向上させます。	API ステージレベルのキャッシュをフラッシュし、個別のキャッシュエントリを無効にします。詳細については、API Gateway で API ステージキャッシュをフラッシュするおよびAPI Gateway のキャッシュエントリの無効化を参照してください。API リクエストのスロットリングを管理するためのコンソールの操作を向上させます。詳細については、「API リクエストを調整してスループットを向上させる」を参照してください。	2016 年 3 月 25 日
カスタム認証を使用した API Gateway API の有効化と呼び出し	カスタム許可を実装した AWS Lambda 関数を作成して設定します。この関数は、API Gateway API のクライアントリクエストに許可または拒否のアクセス許可を付与する IAM ポリシードキュメントを返します。詳細については、「API Gateway Lambda オーソライザーを使用する」を参照してください。	2016 年 2 月 11 日

変更	説明	変更日
Swagger 定義ファイルと拡張機能を使用した API Gateway API のインポートとエクスポート	Swagger の指定と API Gateway 拡張機能を使用して、API Gateway API を作成および更新します。API Gateway Importer を使用して Swagger 定義をインポートします。API Gateway コンソールまたは API Gateway Export API を使用して、API Gateway API を Swagger 定義ファイルにエクスポートします。詳細については、OpenAPI を使用した REST API の設定およびAPI Gateway から REST API をエクスポートするを参照してください。	2015 年 12 月 18 日
リクエストまたはレスポンス本文または本文の JSON フィールドを、リクエストまたはレスポンスパラメーターにマッピングします。	メソッドリクエストボディまたはその JSON フィールドを統合リクエストのパス、クエリ文字列、またはヘッダーにマッピングします。統合のレスポンス本文またはその JSON フィールドをリクエストレスポンスのヘッダーにマッピングします。詳細については、「Amazon API Gateway API リクエストおよびレスポンスデータマッピングリファレンス」を参照してください。	2015 年 12 月 18 日
Amazon API Gateway でのステージ変数の使用	設定属性を Amazon API Gateway の API のデプロイステージに関連付ける方法について説明します。詳細については、「REST API デプロイのステージ変数のセットアップ」を参照してください。	2015 年 11 月 5 日
方法: メソッドの CORS を有効にする	Amazon API Gateway のメソッドに CORS (Cross-Origin Resource Sharing) を簡単に有効にできるようになりました。詳細については、「REST API リソースの CORS を有効にする」を参照してください。	2015 年 3 月 11 日
方法: クライアント側 SSL 認証を使用する	Amazon API Gateway を使用して、HTTP バックエンドの呼び出しを認証するための SSL 証明書を生成します。詳細については、「バックエンド認証用 SSL 証明書の生成と設定」を参照してください。	2015 年 9 月 22 日
メソッドのモック統合	API と Amazon API Gateway のモック統合方法について説明します。この機能により、デベロッパーは最終的な統合バックエンドを事前に用意することなく直接 API Gateway から API レスポンスを生成できます。	2015 年 9 月 1 日

変更	説明	変更日
Amazon Cognito D プールのサポート	Amazon API Gateway は、`$context` 変数のスコープを拡張し、リクエストが Amazon Cognito 認証情報で署名されたときに Amazon Cognito ID に関する情報を返すようになりました。加えて、JavaScript で文字をエスケープして URL と文字列をエンコードするための `$util` 変数が追加されました。詳細については、「API Gateway マッピングテンプレートとアクセスのログ記録の変数リファレンス」を参照してください。	2015 年 8 月 28 日
Swagger 統合	GitHub の Swagger インポートツールを使用して、Amazon API Gateway に Swagger API の定義をインポートします。「OpenAPI への API Gateway 拡張機能の使用」で、インポートツールを使用した API とメソッドの作成とデプロイについて詳細を追加しました。Swagger インポートツールを使用すると、既存の API を更新できます。	2015 年 7 月 21 日
マッピングテンプレートリファレンス	「`$input`」で、API Gateway マッピングテンプレートとアクセスのログ記録の変数リファレンス パラメータとその機能について説明を追加しました。	2015 年 7 月 18 日
初回一般リリース	これは API Gateway デベロッパーガイドの最初の一般リリースです。	2015 年 7 月 9 日

AWS 用語集

AWS の最新の用語については、「AWS の用語集 リファレンス」の「AWS 用語集」を参照してください。