aws

Guide du développeur

Amazon API Gateway

Amazon API Gateway: Guide du développeur

Table of Contents

Qu'est-ce qu'Amazon API Gateway ?

Amazon API Gateway est un AWS service de création, de publication, de maintenance, de surveillance et de sécurisation des API REST, HTTP et des WebSocket API à n'importe quelle échelle. Les développeurs d'API peuvent créer des API qui accèdent à AWS d'autres services Web, ainsi qu'à des données stockées dans le AWS cloud. En tant que développeur d'API API Gateway, vous pouvez créer des API afin de les utiliser dans vos propres applications client. Ou vous pouvez mettre vos API à la disposition de développeurs d'applications tiers. Pour de plus amples informations, veuillez consulter the section called “Qui utilise API Gateway ?”.

API Gateway crée des API RESTful qui :

- reposent sur le protocole HTTP ;
- permettent la communication client-serveur sans état ;
- mettent en œuvre les méthodes HTTP standard, telles que GET, POST, PUT, PATCH et DELETE.

Pour de plus amples informations sur les API REST et HTTP API Gateway, veuillez consulter the section called “Choix entre les API HTTP et les API REST ”, Utilisation des API HTTP, the section called “Utilisation d'API Gateway pour créer des API REST” et the section called “Développement”.

API Gateway crée des WebSocket API qui :

- Respectez le WebSocketprotocole, qui permet une communication dynamique en duplex intégral entre le client et le serveur.
- acheminent les messages entrants reposant sur le contenu du message.

Pour plus d'informations sur les API WebSocket API Gateway, consultez the section called “Utiliser API Gateway pour créer des WebSocket API” etthe section called “À propos WebSocket des API”.

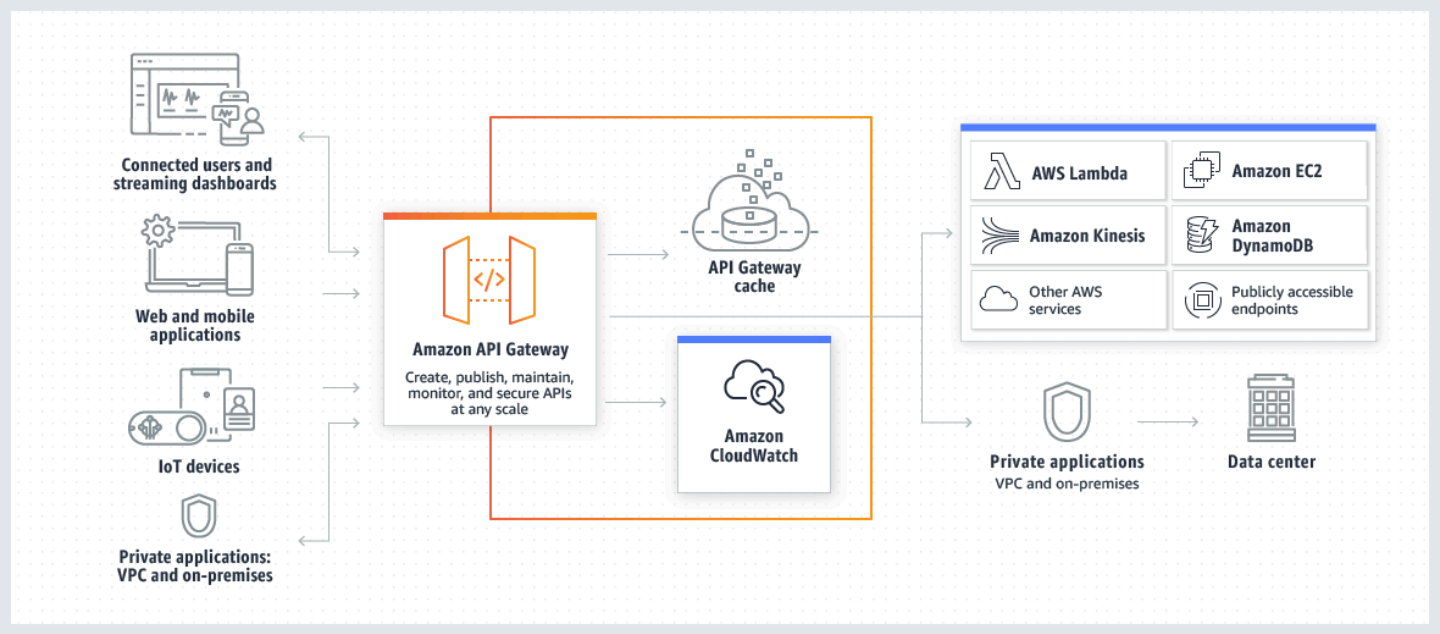
Rubriques

- Architecture API Gateway
- Fonctions d'API Gateway
- Cas d'utilisation d'API Gateway
- Accès à API Gateway
- Fait partie de l' AWS infrastructure sans serveur

- Comment démarrer avec Amazon API Gateway
- Concepts Amazon API Gateway
- Choix entre les API HTTP et les API REST
- Bien démarrer avec la console d'API REST

Architecture API Gateway

Le diagramme suivant illustre l'architecture API Gateway.



Ce diagramme illustre la manière dont les API que vous créez dans Amazon API Gateway vous offrent, à vous ou à vos clients développeurs, une expérience de développement intégrée et cohérente pour la création d'applications sans serveur AWS . API Gateway gère toutes les tâches liées à l'acceptation et au traitement de centaines de milliers d'appels d'API simultanés. Ces tâches incluent la gestion du trafic, des autorisations et du contrôle des accès, la surveillance et la gestion de la version de l'API.

API Gateway agit comme une « porte d'entrée » permettant aux applications d'accéder aux données, à la logique métier ou aux fonctionnalités de vos services principaux, tels que les charges de travail exécutées sur Amazon Elastic Compute Cloud (Amazon EC2), le code exécuté AWS Lambda sur n'importe quelle application Web ou les applications de communication en temps réel.

Fonctions d'API Gateway

Amazon API Gateway offre des fonctions telles que les suivantes :

- Support pour les API stateful (WebSocket) et stateless (HTTP et REST).
- Des mécanismes d'authentification puissants et flexibles, tels que AWS Identity and Access Management les politiques, les fonctions d'autorisation Lambda et les groupes d'utilisateurs Amazon Cognito.
- Déploiements de versions Canari pour le déploiement en toute sécurité des modifications
- CloudTrailjournalisation et surveillance de l'utilisation des API et des modifications apportées aux API.
- CloudWatch journalisation des accès et journalisation des exécutions, y compris la possibilité de définir des alarmes. Pour plus d'informations, consultez the section called “CloudWatch métriques” et the section called “Métriques”.
- Possibilité d'utiliser des AWS CloudFormation modèles pour permettre la création d'API. Pour de plus amples informations, veuillez consulter Référence des types de ressources Amazon API Gateway et Référence des types de ressources Amazon API Gateway V2.
- Prise en charge de noms de domaine personnalisés
- Intégration à AWS WAF pour protéger vos API contre les menaces web courantes
- Intégration à AWS X-Ray pour comprendre et trier les latences de performances

Pour obtenir une liste complète des nouvelles fonctions API Gateway, veuillez consulter Historique du document.

Cas d'utilisation d'API Gateway

Rubriques

- Utilisation d'API Gateway pour créer des API REST
- Utilisation d'API Gateway pour créer des API HTTP
- Utiliser API Gateway pour créer des WebSocket API
- Qui utilise API Gateway ?

Utilisation d'API Gateway pour créer des API REST

Une API REST API Gateway se compose de ressources et de méthodes. Une ressource est une entité logique à laquelle une application peut accéder via un chemin de ressource. Une méthode correspond à une demande d'API REST envoyée par l'utilisateur de l'API et à la réponse correspondante renvoyée à l'utilisateur.

Par exemple, `/incomes` peut-être le chemin de la ressource représentant les revenus de l'utilisateur de l'application. Une ressource peut comporter une ou plusieurs opérations définies par des verbes HTTP appropriés, tels que GET, POST, PUT, PATCH et DELETE. La combinaison d'un chemin de ressource et d'une opération identifie une méthode de l'API. Par exemple, la méthode `POST /incomes` ajoute les revenus générés par l'appelant, et la méthode `GET /expenses` interroge les dépenses signalées engagées par l'appelant.

L'application n'a pas besoin de savoir où sont stockées les données demandées et à partir d'où elles sont extraites sur le back-end. Dans les API REST API Gateway, le serveur frontal est encapsulé par les demandes de méthode et les réponses de méthode. L'API s'interface au backend aux moyens de demandes d'intégration et de réponses d'intégration.

Par exemple, avec DynamoDB comme backend, le développeur d'API configure la demande d'intégration pour transférer la demande de méthode entrante vers le backend choisi. La configuration inclut les spécifications d'une action DynamoDB appropriée, du rôle et des stratégies IAM requis, ainsi que de la transformation des données d'entrée nécessaires. Le backend renvoie le résultat à API Gateway sous forme de réponse d'intégration.

Pour acheminer la réponse d'intégration à une réponse de méthode appropriée (d'un code de statut HTTP donné) vers le client, vous pouvez configurer la réponse d'intégration pour mapper les paramètres de réponse requis de l'intégration à la méthode. Vous pouvez ensuite appliquer le format des données de sortie du serveur principal à celui du serveur frontal, si nécessaire. API Gateway vous permet de définir un schéma ou un modèle pour la charge utile afin de faciliter la configuration du modèle de mappage du corps.

API Gateway fournit des fonctionnalités de gestion des API REST telles que les suivantes :

- Prise en charge de la génération de kits SDK et de la création de la documentation de l'API à l'aide d'extensions API Gateway vers OpenAPI
- Limitations des demandes HTTP

Utilisation d'API Gateway pour créer des API HTTP

Les API HTTP vous permettent de créer des API RESTful avec une latence inférieure et un coût inférieur aux API REST.

Vous pouvez utiliser les API HTTP pour envoyer des demandes à des AWS Lambda fonctions ou à n'importe quel point de terminaison HTTP routable publiquement.

Par exemple, vous pouvez créer une API HTTP qui s'intègre à une fonction Lambda sur le backend. Lorsqu'un client appelle votre API, API Gateway envoie la demande à la fonction Lambda et renvoie la réponse de la fonction au client.

Les API HTTP prennent en charge OpenID Connect et l'autorisation OAuth 2.0. Ils sont fournis avec la prise en charge intégrée du partage de ressources d'origine croisée (CORS) et des déploiements automatiques.

Pour en savoir plus, consultez la section the section called “Choix entre les API HTTP et les API REST ”.

Utiliser API Gateway pour créer des WebSocket API

Dans une WebSocket API, le client et le serveur peuvent s'envoyer des messages à tout moment. Les serveurs principaux peuvent facilement transférer les données aux utilisateurs et appareils connectés, ce qui évite de devoir mettre en œuvre des mécanismes d'interrogation complexes.

Par exemple, vous pouvez créer une application sans serveur à l'aide d'une API WebSocket API Gateway et AWS Lambda pour envoyer et recevoir des messages à destination et en provenance d'utilisateurs individuels ou de groupes d'utilisateurs dans un salon de discussion. Vous pouvez également invoquer des services principaux tels qu' AWS Lambda Amazon Kinesis ou un point de terminaison HTTP en fonction du contenu du message.

Vous pouvez utiliser les API WebSocket API Gateway pour créer des applications de communication sécurisées en temps réel sans avoir à approvisionner ou à gérer des serveurs pour gérer les connexions ou les échanges de données à grande échelle. Les cas d'utilisation ciblée incluent des applications en temps réel telles que les suivantes :

- Applications de conversation
- Tableaux de bord en temps réel tels que symboles boursiers
- Alertes et notifications en temps réel

API Gateway fournit des fonctionnalités de gestion des WebSocket API telles que les suivantes :

- Surveillance et limitation des connexions et des messages
- Utilisation AWS X-Ray pour suivre les messages lorsqu'ils transitent par les API vers les services principaux
- Intégration aisée avec des points de terminaison HTTP/HTTPS

Qui utilise API Gateway ?

Deux types de développeurs utilisent API Gateway : les développeurs d'API et les développeurs d'applications.

Un développeur d'API crée et déploie une API pour activer la fonctionnalité requise dans API Gateway. Le développeur de l'API doit être un utilisateur du AWS compte propriétaire de l'API.

Un développeur d'applications crée une application fonctionnelle pour appeler AWS des services en invoquant une API WebSocket ou une API REST créée par un développeur d'API dans API Gateway.

Le développeur d'applications est le client du développeur d'API. Le développeur de l'application n'a pas besoin de AWS compte, à condition que l'API ne nécessite pas d'autorisations IAM ou prenne en charge l'autorisation des utilisateurs via des fournisseurs d'identité fédérés tiers pris en charge par la fédération d'identité du pool d'utilisateurs Amazon Cognito. Parmi ces fournisseurs d'identité, citons Amazon, les groupes d'utilisateurs Amazon Cognito, Facebook et Google.

Création et gestion d'une API API Gateway

Un développeur d'API utilise le composant de service API Gateway pour la gestion des API, nommé `apigateway`, afin de créer, configurer et déployer une API.

En tant que développeur d'API, vous pouvez créer et gérer une API à l'aide de la console API Gateway, décrite dans Mise en route avec API Gateway ou en appelant Références d'API. Il existe plusieurs méthodes pour appeler cette API. Ils incluent l'utilisation du AWS Command Line Interface (AWS CLI) ou l'utilisation d'un AWS SDK. En outre, vous pouvez activer la création d'API avec des modèles AWS CloudFormation ou (dans le cas des API REST et des API HTTP) Utilisation des extensions API Gateway vers OpenAPI.

Pour obtenir la liste des régions où API Gateway est disponible, ainsi que les points de terminaison du service de contrôle associés, veuillez consulter Points de terminaison et quotas Amazon API Gateway.

Appel d'une API API Gateway

Un développeur d'application utilise le composant de service API Gateway pour l'exécution des API, nommé `execute-api`, afin d'appeler une API créée ou déployée dans API Gateway. Les entités de programmation sous-jacentes sont exposées par l'API créée. Il existe plusieurs méthodes pour appeler une telle API. Pour en savoir plus, consultez Appel d'une API REST dans Amazon API Gateway et Invoquer une API WebSocket .

Accès à API Gateway

Vous pouvez accéder à Amazon API Gateway de la manière suivante :

- AWS Management Console— AWS Management Console Fournit une interface Web pour créer et gérer des API. Après avoir suivi les étapes de Prérequis, vous pouvez accéder à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
- AWS SDK — Si vous utilisez un langage de programmation qui AWS fournit un SDK pour, vous pouvez utiliser un SDK pour accéder à API Gateway. Les kits SDK simplifient l'authentification, s'intègrent facilement à votre environnement de développement et permettent d'accéder facilement aux commandes API Gateway. Pour de plus amples informations, veuillez consulter Outils pour Amazon Web Services.
- API API Gateway V1 et V2 : si vous utilisez un langage de programmation pour lequel aucun SDK n'est disponible, veuillez consulter Référence des API Amazon API Gateway Version 1 et Référence des API Amazon API Gateway Version 2.
- AWS Command Line Interface – Pour plus d'informations, veuillez consulter Préparation de l'installation de AWS Command Line Interface dans le Guide de l'utilisateur AWS Command Line Interface .
- AWS Tools for Windows PowerShell – Pour plus d'informations, veuillez consulter Configuration de AWS Tools for Windows PowerShell dans le Guide de l'utilisateur AWS Tools for Windows PowerShell .

Fait partie de l' AWS infrastructure sans serveur

API Gateway constitue avec AWS Lambdaelle la partie de l'infrastructure AWS sans serveur orientée vers les applications. Pour savoir comment démarrer avec la technologie sans serveur, consultez le Guide du développeur sans serveur.

Pour qu'une application appelle des AWS services accessibles au public, vous pouvez utiliser Lambda pour interagir avec les services requis et exposer les fonctions Lambda via des méthodes d'API dans API Gateway. AWS Lambda exécute votre code sur une infrastructure informatique à haute disponibilité. Il procède à l'exécution et à l'administration nécessaires des ressources de calcul. Pour activer les applications sans serveur, API Gateway prend en charge les intégrations de proxy rationalisées avec les points de AWS Lambda terminaison HTTP.

Comment démarrer avec Amazon API Gateway

Pour une présentation d'Amazon API Gateway, consultez les sections suivantes :

- Mise en route, qui fournit une procédure pas à pas pour créer une API HTTP.
- Serverless land, qui fournit des vidéos pédagogiques.
- Happy Little API Shorts, qui est une série de brèves vidéos pédagogiques.

Concepts Amazon API Gateway

API Gateway

API Gateway est un AWS service qui prend en charge les éléments suivants :

- Création, déploiement et gestion d'une interface de programmation d'applications (API) RESTful pour exposer les points de terminaison, AWS Lambda fonctions ou autres services HTTP du backend. AWS
- Création, déploiement et gestion d'une WebSocketAPI pour exposer des AWS Lambda fonctions ou d'autres AWS services.
- Invocation de méthodes d'API exposées via le protocole HTTP frontal et WebSocket les points de terminaison.

API REST API Gateway

Ensemble de ressources et de méthodes HTTP intégrées aux points de terminaison HTTP principaux, aux fonctions Lambda ou à d'autres services. AWS Vous pouvez déployer cette collection en une ou plusieurs étapes. En règle générale, les ressources API sont organisées dans une arborescence des ressources selon la logique de l'application. Chaque ressource API peut exposer une ou plusieurs méthodes d'API comportant des verbes HTTP uniques pris en charge par API Gateway. Pour plus d’informations, consultez the section called “Choix entre les API HTTP et les API REST ”.

API HTTP API Gateway

Collection de routes et de méthodes qui sont intégrées aux points de terminaison HTTP du backend ou aux fonctions Lambda. Vous pouvez déployer cette collection en une ou plusieurs étapes. Chaque route peut exposer une ou plusieurs méthodes d'API comportant des verbes HTTP uniques pris en charge par API Gateway. Pour plus d'informations, consultez the section called “Choix entre les API HTTP et les API REST ”.

WebSocket API Gateway

Ensemble de WebSocket routes et de clés de route intégrées aux points de terminaison HTTP du backend, aux fonctions Lambda ou à d'autres services. AWS Vous pouvez déployer cette collection en une ou plusieurs étapes. Les méthodes d'API sont invoquées via WebSocket des connexions frontales que vous pouvez associer à un nom de domaine personnalisé enregistré.

Déploiement de l'API

Un point-in-time aperçu de votre API API Gateway. Pour que les clients puissent accéder au déploiement et l'utiliser, il doit être associé à une ou plusieurs étapes d'API.

Développeur de l'API

Votre AWS compte propriétaire d'un déploiement d'API Gateway (par exemple, un fournisseur de services qui prend également en charge l'accès par programmation).

Point de terminaison d'API

Nom d'hôte d'une API dans API Gateway déployée dans une région spécifique. Le nom d'hôte se présente sous la forme `{api-id}.execute-api.{region}.amazonaws.com`. Les types de points de terminaison d'API suivants sont pris en charge :

- Point de terminaison d'API optimisée pour les périphériques
- Point de terminaison de l'API privée
- Point de terminaison de l'API régionale

Clé API

Chaîne alphanumérique utilisée par API Gateway pour identifier un développeur d'applications qui utilise votre REST ou votre WebSocket API. API Gateway peut générer des clés d'API pour vous, ou vous pouvez les importer à partir d'un fichier CSV. Vous pouvez utiliser des clés d'API avec des mécanismes d'autorisation Lambda ou des plans d'utilisation pour contrôler l'accès à vos API.

Consultez l'entrée Points de terminaison d'API.

Propriétaire de l'API

Voir Développeur de l'API.

Étape de l'API

Référence logique à un état du cycle de vie de votre API (par exemple, « dev », « prod », « bêta », « v2 »). Les étapes d'API sont identifiées par l'ID de l'API et un nom d'étape.

Développeur d'applications

Un créateur d'application qui possède ou non un AWS compte et qui interagit avec l'API que vous, le développeur de l'API, avez déployée. Les développeurs d'applications sont vos clients. Un développeur d'applications est généralement identifié par une clé d'API.

URL de rappel

Lorsqu'un nouveau client est connecté via une WebSocket connexion, vous pouvez appeler une intégration dans API Gateway pour stocker l'URL de rappel du client. Vous pouvez ensuite utiliser cette URL de rappel pour envoyer des messages au client à partir du système backend.

Portail des développeurs

Application qui permet à vos clients d'enregistrer, de découvrir et de s'abonner à vos produits API (plans d'utilisation API Gateway), de gérer leurs clés API et de consulter leurs métriques d'utilisation de vos API.

Point de terminaison d'API optimisée pour les périphériques

Le nom d'hôte par défaut d'une API API Gateway déployée dans la région spécifiée tout en utilisant une CloudFront distribution pour faciliter l'accès des clients, généralement depuis plusieurs AWS régions. Les demandes d'API sont acheminées vers le CloudFront point de présence (POP) le plus proche, ce qui améliore généralement le temps de connexion pour les clients géographiquement divers.

Consultez l'entrée Points de terminaison d'API.

Demande d'intégration

Interface interne d'une route d' WebSocket API ou d'une méthode d'API REST dans API Gateway, dans laquelle vous mappez le corps d'une demande de route ou les paramètres et le corps d'une demande de méthode aux formats requis par le backend.

Réponse d'intégration

Interface interne d'une route d' WebSocket API ou d'une méthode d'API REST dans API Gateway, dans laquelle vous mappez les codes d'état, les en-têtes et la charge utile reçus du backend au format de réponse renvoyé à une application cliente.

Modèle de mappage

Script en langage VTL (Velocity Template Language), permettant de convertir le corps d'une demande du format de données frontend au format de données backend, ou de convertir le corps d'une réponse du format de données backend au format de données frontend. Les modèles de mappage peuvent être spécifiés dans la demande ou la réponse d'intégration. Ils peuvent faire référence aux données rendues accessibles au moment de l'exécution en tant que variables de contexte et d'étape.

Le mappage peut être aussi simple qu'une transformation d'identité qui transmet tels quels les en-têtes ou le corps via l'intégration depuis le client vers le serveur principal pour une demande. Cela s'applique également à une réponse, la charge utile étant transmise du serveur principal au client.

Demande de méthode

Interface publique d'une méthode d'API dans API Gateway qui définit les paramètres et le corps qu'un développeur d'application doit envoyer dans des demandes pour accéder au backend via l'API.

Réponse de méthode

Interface publique d'une API REST qui définit les codes de statut, les en-têtes et les modèles de corps qu'un développeur d'application doit attendre des réponses de l'API.

Intégration fictive

Dans une intégration fictive, des réponses d'API sont générées directement depuis API Gateway, sans recourir à un backend d'intégration. En tant que développeur d'API, vous décidez de la façon dont API Gateway répond à une demande d'intégration fictive. Pour cela, vous configurez la demande d'intégration et la réponse d'intégration de la méthode pour associer une réponse à un code d'état donné.

Modèle

Schéma de données spécifiant la structure de données de la charge utile d'une demande ou d'une réponse. Un modèle est nécessaire pour générer un kit SDK fortement typé d'une API. Il est également utilisé pour valider des charges utiles. Un modèle est pratique pour générer un

exemple de modèle de mappage afin d'initier la création d'un modèle de mappage de production. Bien qu'utile, un modèle n'est pas obligatoire pour créer un modèle de mappage.

API privée

Consultez l'entrée Point de terminaison de l'API privée.

Point de terminaison de l'API privée

Point de terminaison d'API qui est exposé via des points de terminaison de VPC d'interface et qui permet à un client d'accéder en toute sécurité aux ressources de l'API privée à l'intérieur d'un VPC. Les API privées sont isolées de l'Internet public et il n'est possible d'y accéder qu'à l'aide de points de terminaison d'un VPC pour API Gateway auxquels un accès a été accordé.

Intégration privée

Type d'intégration API Gateway permettant à un client d'accéder à des ressources dans le VPC d'un client via un point de terminaison d'API REST privé sans exposer les ressources à l'Internet public.

Intégration de proxy

Configuration d'intégration API Gateway simplifiée. Vous pouvez configurer une intégration de proxy en tant qu'intégration de proxy HTTP ou intégration de proxy Lambda.

Pour l'intégration de proxy HTTP, API Gateway transfère l'intégralité de la demande et de la réponse entre le serveur frontal et un backend HTTP. Pour l'intégration de proxy Lambda, API Gateway envoie l'intégralité de la demande sous forme d'entrée à une fonction Lambda du backend. Ensuite, API Gateway transforme la sortie de la fonction Lambda en une réponse HTTP du serveur frontal.

Dans les API REST, l'intégration de proxy est plus couramment utilisée avec une ressource de proxy, qui est représentée par une variable de chemin gourmande (par exemple, `{proxy+}`) combinée à la méthode `ANY` fourre-tout.

Création rapide

Vous pouvez utiliser la fonction de création rapide pour simplifier la création d'une API HTTP. La création rapide crée une API avec une intégration Lambda ou HTTP, une route fourre-tout par défaut et une étape par défaut configurée pour déployer automatiquement les modifications. Pour de plus amples informations, veuillez consulter the section called “Création d'une API HTTP à l'aide de la AWS CLI”.

Point de terminaison d'API régional

Le nom d'hôte d'une API déployée dans la région spécifiée et destinée à servir des clients, tels que des instances EC2, dans la même AWS région. Les demandes d'API ciblent directement l'API API Gateway spécifique à la région sans passer par aucune CloudFront distribution. Pour les demandes internes à la région, un point de terminaison régional évite les allers-retours inutiles vers une CloudFront distribution.

De plus, vous pouvez appliquer le routage basé sur la latence aux points de terminaison régionaux afin de déployer une API dans plusieurs régions à l'aide de la même configuration de point de terminaison d'API régionale. Pour ce faire, définissez le même nom de domaine personnalisé pour chaque API déployée et configurez des enregistrements DNS basés sur la latence dans Route 53 afin d'acheminer les demandes des clients vers la région ayant la plus faible latence.

Consultez l'entrée Points de terminaison d'API.

Route

Une WebSocket route dans API Gateway est utilisée pour diriger les messages entrants vers une intégration spécifique, telle qu'une AWS Lambda fonction, en fonction du contenu du message. Lorsque vous définissez votre WebSocket API, vous spécifiez une clé de route et un backend d'intégration. La clé de routage est un attribut du corps du message. Lorsque la clé de routage est mise en correspondance dans un message entrant, le serveur principal d'intégration est invoqué.

Une route par défaut peut également être définie pour des clés de routage qui ne correspondent pas ou pour spécifier un modèle de proxy qui transmet le message tel quel à des composants du serveur principal qui effectuent le routage et traitent la demande.

Demande de routage

Interface publique d'une méthode d' WebSocket API dans API Gateway qui définit le corps qu'un développeur d'applications doit envoyer dans les demandes pour accéder au backend via l'API.

Réponse de routage

Interface publique d'une WebSocket API qui définit les codes de statut, les en-têtes et les modèles corporels auxquels un développeur d'applications doit s'attendre de la part d'API Gateway.

Plan d'utilisation

Un plan d'utilisation permet à des clients d'API sélectionnés d'accéder à une ou plusieurs WebSocket API ou REST déployées. Vous pouvez utiliser un plan d'utilisation pour configurer des limitations et des quotas, qui sont appliquées individuellement à chaque clé d'API client.

WebSocket connexion

API Gateway maintient une connexion persistante entre les clients et API Gateway lui-même. Il n'y a pas de connexion persistante entre API Gateway et les intégrations backend telles que les fonctions Lambda. Les services backend sont appelés selon les besoins, en fonction du contenu des messages reçus des clients.

Choix entre les API HTTP et les API REST

Les API REST et les API HTTP sont toutes des produits d'API RESTful. Les API REST prennent en charge plus de fonctionnalités que les API HTTP, tandis que les API HTTP sont conçues avec un minimum de fonctionnalités afin de pouvoir être proposées à un prix inférieur. Choisissez les API REST si vous avez besoin de fonctionnalités telles que les clés API, la limitation par client, la validation des demandes, l'intégration de AWS WAF ou les points de terminaison d'API privés. Choisissez les API HTTP si vous n'avez pas besoin des fonctionnalités incluses dans les API REST.

Les tableaux suivants résument les fonctions principales disponibles dans les API HTTP et les API REST.

Type de point de terminaison

Le type de point de terminaison fait référence au point de terminaison créé par API Gateway pour votre API. Pour plus d'informations, consultez the section called “Types de points de terminaison API Gateway”.

Types de point de terminaison	API REST	API HTTP
Optimisés pour les périphériques	✓	
Régional	✓	✓
Privé	✓	

Sécurité

API Gateway fournit un certain nombre de façons de protéger votre API contre certaines menaces, comme les acteurs malveillants ou les pics de trafic. Pour en savoir plus, consultez the section called “Protéger” et the section called “Protéger”.

Fonctions de sécurité	API REST	API HTTP
Authentification TLS mutuelle	✓	✓
Certificats pour l'authentification backend	✓	
AWS WAF	✓	

Autorisation

API Gateway prend en charge plusieurs mécanismes pour contrôler et gérer l'accès à votre API. Pour plus d’informations, consultez the section called “Contrôle d'accès” et the section called “Contrôle d'accès”.

Options d'autorisation	API REST	API HTTP
IAM	✓	✓
Politiques basées sur une ressource	✓	
Amazon Cognito	✓	✓[1]
Autorisation personnalisée avec une AWS Lambda fonction	✓	✓
Jeton JWT (JSON Web Token) [2]		✓

[1]Vous pouvez utiliser Amazon Cognito avec un mécanisme d'autorisation JWT.

[2] Vous pouvez utiliser unmécanisme d'autorisation Lambda pour valider les JWT pour les API REST.

Gestion des API

Choisissez les API REST si vous avez besoin de fonctionnalités de gestion d'API telles que les clés API et la limitation de débit par client. Pour plus d'informations, consultez the section called "Distribuer", the section called "Noms de domaine personnalisés" et the section called "Noms de domaine personnalisés".

Fonctionnalités	API REST	API HTTP
Domaines personnalisés	✓	✓
Clés API	✓	
Limitation de débit par client	✓	
Limitation de l'utilisation par client	✓	

Développement

Au fur et à mesure que vous développez votre API API Gateway, vous décidez d'un certain nombre de caractéristiques de votre API. Ces caractéristiques dépendent du cas d'utilisation de votre API. Pour plus d'informations, consultez the section called "Développement" et the section called "Développement".

Fonctionnalités	API REST	API HTTP
Configuration CORS	✓	✓
Invocations de tests	✓	
Mise en cache	✓	
Déploiements contrôlés par l'utilisateur	✓	✓
Déploiements automatiques		✓

Fonctionnalités	API REST	API HTTP
Réponses de passerelle personnalisées	✓	
Déploiement d'une version Canary	✓	
Validation des demandes	✓	
Transformation des paramètres de demande	✓	✓
Transformation du corps de la demande	✓	

Surveillance

API Gateway prend en charge plusieurs options pour consigner les demandes d'API et surveiller vos API. Pour plus d'informations, consultez the section called “Contrôle” et the section called “Contrôle”.

Fonctionnalité	API REST	API HTTP
CloudWatch Métriques Amazon	✓	✓
Logs d'accès aux CloudWatch logs	✓	✓
Journaux d'accès à Amazon Data Firehose	✓	
Journaux d'exécution	✓	
AWS X-Ray traçage	✓	

Intégrations

Les intégrations connectent votre API Gateway aux ressources backend. Pour plus d'informations, consultez the section called "Intégrations" et the section called "Intégrations".

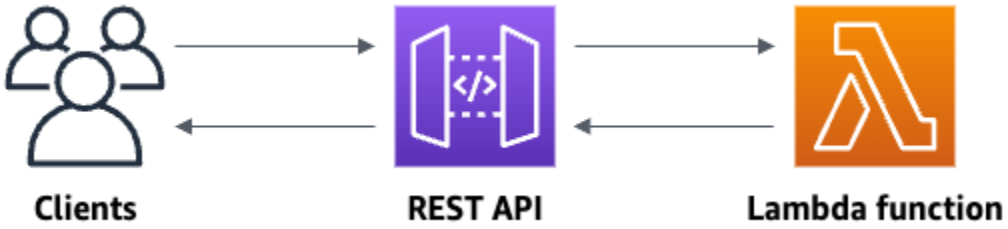
Fonctionnalité	API REST	API HTTP
Points de terminaison HTTP publics	✓	✓
AWS services	✓	✓
AWS Lambda fonctions	✓	✓
Intégrations privées avec instances de Network Load Balancer	✓	✓
Intégrations privées avec instances d'Application Load Balancer		✓
Intégrations privées avec AWS Cloud Map		✓
Intégrations fictives	✓	

Bien démarrer avec la console d'API REST

Dans cet exercice de prise en main, vous allez créer une API REST sans serveur à l'aide de la console d'API REST API Gateway. Les API sans serveur vous permettent de vous concentrer sur vos applications, plutôt que de passer du temps à provisionner et gérer des serveurs. Cet exercice devrait prendre moins de 20 minutes et peut être réalisé avec l'offre gratuite AWS.

Vous commencez par créer une fonction Lambda à l'aide de la console Lambda. Ensuite, vous créez une API REST à l'aide de la console d'API REST API Gateway. Ensuite, vous créez une méthode d'API et vous l'intégrez à une fonction Lambda à l'aide d'une intégration de proxy Lambda. Enfin, vous allez déployer et invoquer votre API.

Lorsque vous invoquez votre API REST, API Gateway route la demande vers votre fonction Lambda. Lambda exécute la fonction et renvoie une réponse à API Gateway. API Gateway vous renvoie ensuite cette réponse.



Pour effectuer cet exercice, vous avez besoin d'un utilisateur Compte AWS et d'un AWS Identity and Access Management (IAM) ayant accès à la console. Pour plus d'informations, consultez Prérequis pour démarrer avec API Gateway.

Rubriques

- Étape 1 : Créer une fonction Lambda
- Étape 2 : Créer une API REST
- Étape 3 : Créer une intégration de proxy Lambda
- Étape 4 : Déployer votre API
- Étape 5 : Invoquer l'API
- (Facultatif) Étape 6 : nettoyer

Étape 1 : Créer une fonction Lambda

Vous utilisez une fonction Lambda pour le backend de votre API. Lambda exécute le code uniquement lorsque cela est nécessaire et se met à l'échelle automatiquement, qu'il s'agisse de quelques requêtes par jour ou de milliers de requêtes par seconde.

Pour cet exercice, vous utilisez une fonction Node.js par défaut dans la console Lambda.

Pour créer une fonction Lambda

1. Connectez-vous à la console Lambda à l'adresse https://console.aws.amazon.com/lambda/.
2. Sélectionnez Create function (Créer une fonction).
3. Sous Informations de base, pour Nom de la fonction, entrez **`my-function`**.
4. Choisissez Créer une fonction.

Le code de la fonction Lambda par défaut devrait ressembler à ce qui suit :

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

Vous pouvez modifier votre fonction Lambda pour cet exercice, tant que la réponse de la fonction s'aligne sur le format requis par API Gateway.

Remplacez le corps de réponse par défaut (`Hello from Lambda!`) par `The API Gateway REST API console is great!`. Lorsque vous invoquez l'exemple de fonction, il renvoie une réponse `200` aux clients, ainsi que la réponse mise à jour.

Étape 2 : Créer une API REST

Ensuite, vous créez une API REST avec une ressource racine (`/`).

Pour créer une API REST

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Effectuez l'une des actions suivantes :
   - Pour créer votre première API, pour API REST, choisissez Création.
   - Si vous avez déjà créé une API, choisissez Créer une API, puis Création pour API REST.
3. Sous API name (Nom de l'API), saisissez **`my-rest-api`**.
4. (Facultatif) Sous Description, entrez une description.
5. Laissez Type de point de terminaison d'API défini sur Régional.
6. Sélectionnez Create API (Créer une API).

Étape 3 : Créer une intégration de proxy Lambda

Ensuite, vous devez créer une méthode d'API pour votre API REST sur la ressource racine (`/`) et intégrer la méthode à votre fonction Lambda à l'aide d'une intégration de proxy. Dans une intégration

de proxy Lambda, API Gateway transmet la demande entrante du client directement à la fonction Lambda.

Pour créer une intégration de proxy Lambda

1. Sélectionnez la ressource `/`, puis choisissez Créer une méthode.
2. Pour Type de méthode, sélectionnez `ANY`.
3. Pour Type d'intégration, sélectionnez Lambda
4. Activez Intégration de proxy Lambda.
5. Pour Fonction Lambda, entrez **`my-function`**, puis sélectionnez votre fonction Lambda.
6. Choisissez Créer une méthode.

Étape 4 : Déployer votre API

Ensuite, vous devez créer un déploiement d'API et l'associer à une étape.

Pour déployer votre API

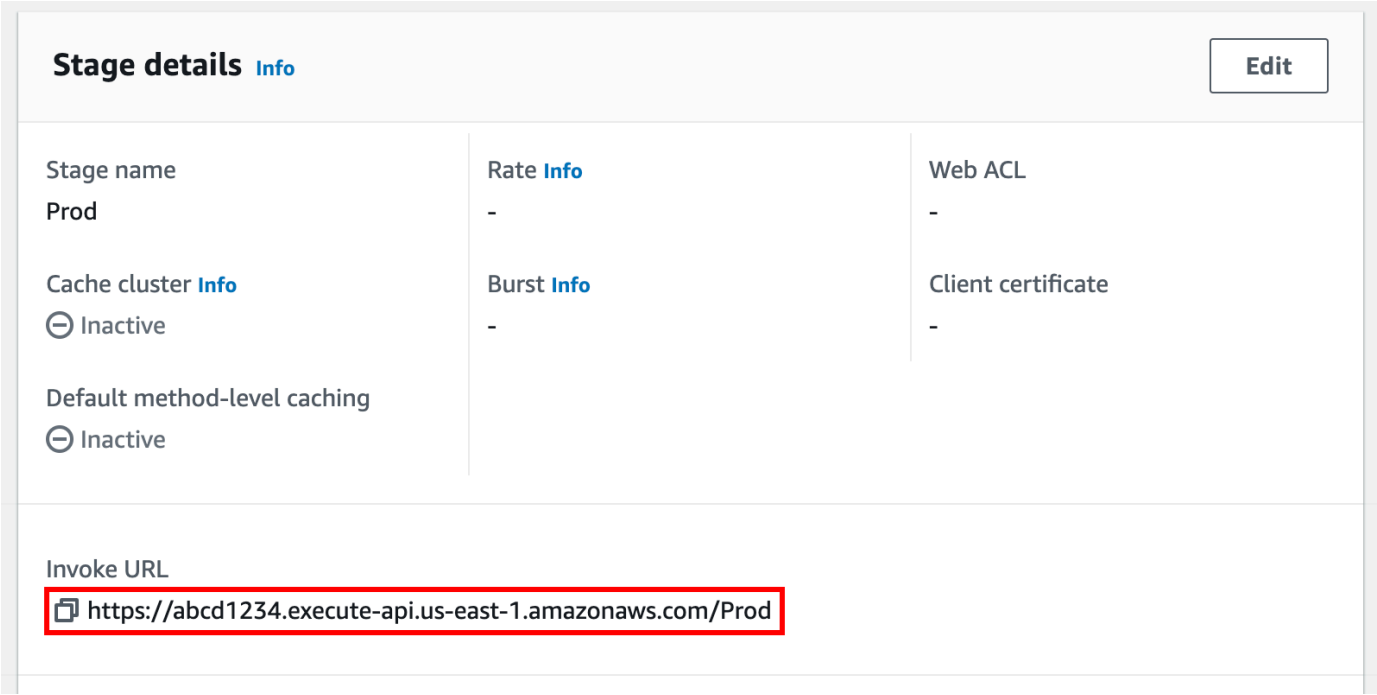
1. Sélectionnez Deploy API (Déployer une API).
2. Pour Étape, sélectionnez Nouvelle étape.
3. Sous Stage name (Nom de l'étape), entrez **`Prod`**.
4. (Facultatif) Sous Description, entrez une description.
5. Choisissez Deploy (Déployer).

Les clients peuvent désormais appeler votre API. Pour tester votre API avant de la déployer, vous pouvez éventuellement choisir la méthode ANY, accéder à l'onglet Tester, puis choisir Tester.

Étape 5 : Invoquer l'API

Pour invoquer votre API

1. Dans le panneau de navigation principal, choisissez Étape.
2. Sous Détails de l'étape, choisissez l'icône de copie pour copier l'URL d'invocation de votre API.



3. Entrez cette URL d'invocation dans un navigateur Web.

   L'URL complète doit ressembler à `https://abcd123.execute-api.us-east-2.amazonaws.com/Prod`.

   Votre navigateur envoie une requête `GET` à l'API.

4. Vérifiez la réponse de votre API. Vous devriez voir le texte `"The API Gateway REST API console is great!"` dans votre navigateur.

(Facultatif) Étape 6 : nettoyer

Pour éviter de vous faire supporter des coûts inutiles Compte AWS, supprimez les ressources que vous avez créées dans le cadre de cet exercice. Les étapes suivantes suppriment votre API REST, votre fonction Lambda, ainsi que les ressources associées.

Pour supprimer votre API REST

1. Dans le volet Ressources, choisissez Actions d'API, puis Supprimer l'API.
2. Dans la boîte de dialogue Supprimer l'API, entrez confirm, puis choisissez Supprimer.

Pour supprimer votre fonction Lambda

1. Connectez-vous à la console Lambda à l'adresse https://console.aws.amazon.com/lambda/.
2. Sur la page Fonctions, sélectionnez votre fonction. Sélectionnez Actions, Supprimer.
3. Dans la boîte de dialogue Supprimer 1 fonctions, entrez **`delete`**, puis choisissez Supprimer.

Pour supprimer le groupe de journaux de votre fonction Lambda

1. Ouvrez la page Log groups de la CloudWatch console Amazon.
2. Sur la page Groupes de journaux, sélectionnez le groupe de journaux de votre fonction (`/aws/lambda/my-function`). Ensuite, pour Actions, choisissez Supprimer le ou les groupes de journaux.
3. Dans la boîte de dialogue Supprimer le ou les groupes de journaux, choisissez Supprimer.

Pour supprimer le rôle d'exécution de votre fonction Lambda

1. Ouvrez la page Roles (Rôles) de la console IAM.
2. (Facultatif) Sur la page Rôles, dans la zone de recherche, entrez **`my-function`**.
3. Sélectionnez le rôle de votre fonction (par exemple, `my-function-31exxmpl`), puis choisissez Supprimer.
4. Dans la boîte de dialogue Supprimer **`my-function-31exxmpl`** ?, entrez le nom du rôle, puis choisissez Supprimer.

> **Tip**
>
> Vous pouvez automatiser la création et le nettoyage des AWS ressources en utilisant AWS CloudFormation or AWS Serverless Application Model (AWS SAM). Pour des exemples de AWS CloudFormation modèles, consultez les exemples de modèles pour API Gateway dans le référentiel awsdocs GitHub .

Prérequis pour démarrer avec API Gateway

Avant d'utiliser Amazon API Gateway pour la première fois, exécutez les tâches suivantes.

Inscrivez-vous pour un Compte AWS

Si vous n'en avez pas Compte AWS, procédez comme suit pour en créer un.

Pour vous inscrire à un Compte AWS

1. Ouvrez https://portal.aws.amazon.com/billing/signup.
2. Suivez les instructions en ligne.

   Dans le cadre de la procédure d'inscription, vous recevrez un appel téléphonique et vous saisirez un code de vérification en utilisant le clavier numérique du téléphone.

   Lorsque vous vous inscrivez à un Compte AWS, un Utilisateur racine d'un compte AWSest créé. Par défaut, seul l'utilisateur racine a accès à l'ensemble des Services AWS et des ressources de ce compte. La meilleure pratique en matière de sécurité consiste à attribuer un accès administratif à un utilisateur et à n'utiliser que l'utilisateur root pour effectuer les tâches nécessitant un accès utilisateur root.

AWS vous envoie un e-mail de confirmation une fois le processus d'inscription terminé. Vous pouvez afficher l'activité en cours de votre compte et gérer votre compte à tout moment en accédant à https://aws.amazon.com/ et en choisissant Mon compte.

Création d'un utilisateur doté d'un accès administratif

Une fois que vous vous êtes inscrit à un utilisateur administratif Compte AWS, que vous Utilisateur racine d'un compte AWS l'avez sécurisé AWS IAM Identity Center, que vous l'avez activé et que vous en avez créé un, afin de ne pas utiliser l'utilisateur root pour les tâches quotidiennes.

Sécurisez votre Utilisateur racine d'un compte AWS

1. Connectez-vous en AWS Management Consoletant que propriétaire du compte en choisissant Utilisateur root et en saisissant votre adresse Compte AWS e-mail. Sur la page suivante, saisissez votre mot de passe.

Pour obtenir de l'aide pour vous connecter en utilisant l'utilisateur racine, consultez Connexion en tant qu'utilisateur racine dans le Guide de l'utilisateur Connexion à AWS .

2. Activez l'authentification multifactorielle (MFA) pour votre utilisateur racine.

   Pour obtenir des instructions, voir Activer un périphérique MFA virtuel pour votre utilisateur Compte AWS root (console) dans le guide de l'utilisateur IAM.

Création d'un utilisateur doté d'un accès administratif

1. Activez IAM Identity Center.

   Pour obtenir des instructions, consultez Activation d' AWS IAM Identity Center dans le Guide de l'utilisateur AWS IAM Identity Center .

2. Dans IAM Identity Center, accordez un accès administratif à un utilisateur.

   Pour un didacticiel sur l'utilisation du Répertoire IAM Identity Center comme source d'identité, voir Configurer l'accès utilisateur par défaut Répertoire IAM Identity Center dans le Guide de AWS IAM Identity Center l'utilisateur.

Connectez-vous en tant qu'utilisateur disposant d'un accès administratif

- Pour vous connecter avec votre utilisateur IAM Identity Center, utilisez l'URL de connexion qui a été envoyée à votre adresse e-mail lorsque vous avez créé l'utilisateur IAM Identity Center.

  Pour obtenir de l'aide pour vous connecter en utilisant un utilisateur d'IAM Identity Center, consultez la section Connexion au portail AWS d'accès dans le guide de l'Connexion à AWS utilisateur.

Attribuer l'accès à des utilisateurs supplémentaires

1. Dans IAM Identity Center, créez un ensemble d'autorisations conforme aux meilleures pratiques en matière d'application des autorisations du moindre privilège.

   Pour obtenir des instructions, voir Création d'un ensemble d'autorisations dans le guide de AWS IAM Identity Center l'utilisateur.

2. Affectez des utilisateurs à un groupe, puis attribuez un accès d'authentification unique au groupe.

Pour obtenir des instructions, voir Ajouter des groupes dans le guide de AWS IAM Identity Center l'utilisateur.

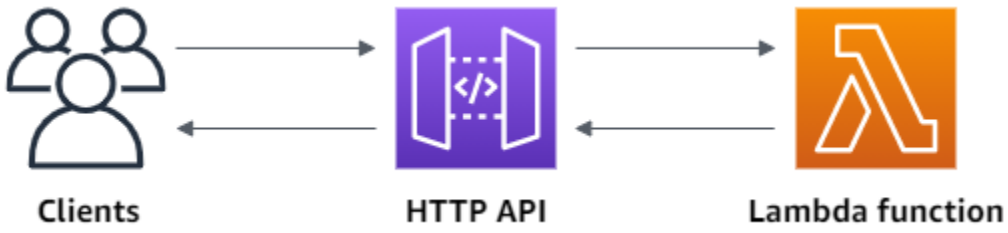
Mise en route avec API Gateway

Dans le cadre de cet exercice de mise en route, vous créez une API sans serveur. Les API sans serveur vous permettent de vous concentrer sur vos applications, plutôt que de passer du temps à mettre en service et gérer des serveurs. Cet exercice prend moins de 20 minutes et peut être réalisé avec AWS l'offre gratuite.

Vous devez d'abord créer une fonction Lambda à l'aide de la AWS Lambda console. Ensuite, vous créez une API HTTP à l'aide de la console API Gateway. Ensuite, vous appelez votre API.

> **Note**
>
> Cet exercice utilise une API HTTP. API Gateway prend également en charge les API REST, qui incluent davantage de fonctions. Pour un didacticiel utilisant une API REST, voirthe section called “Bien démarrer avec la console d'API REST”.
> Pour plus d'informations sur la différence entre les API HTTP et les API REST, consultezthe section called “Choix entre les API HTTP et les API REST ”.

Lorsque vous appelez votre API HTTP, API Gateway achemine la requête vers votre fonction Lambda. Lambda exécute la fonction Lambda et renvoie une réponse à API Gateway. API Gateway vous renvoie ensuite une réponse.



Pour effectuer cet exercice, vous avez besoin d'un AWS compte et d'un AWS Identity and Access Management utilisateur ayant accès à la console. Pour plus d’informations, consultez Prérequis.

Rubriques

- Étape 1 : Créer une fonction Lambda
- Étape 2 : Créer une API HTTP
- Étape 3 : Tester votre API

Étape 1 : Créer une fonction Lambda

Vous utilisez une fonction Lambda pour le backend de votre API. Lambda exécute le code uniquement lorsque cela est nécessaire et se met à l'échelle automatiquement, qu'il s'agisse de quelques requêtes par jour ou de milliers de requêtes par seconde.

Pour cet exemple, vous utilisez la fonction Node.js par défaut de la console Lambda.

Pour créer une fonction Lambda

1. Connectez-vous à la console Lambda à l'adresse https://console.aws.amazon.com/lambda/.
2. Choisissez Créer une fonction.
3. Sous Function name (Nom de la fonction), saisissez **`my-function`**.
4. Choisissez Créer une fonction.

L'exemple de fonction renvoie la réponse `200` aux clients, ainsi que le texte `Hello from Lambda!`.

Vous pouvez modifier votre fonction Lambda, tant que la réponse de la fonction correspond au format requis par API Gateway.

Le code de la fonction Lambda par défaut devrait ressembler à ce qui suit :

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

Étape 2 : Créer une API HTTP

Ensuite, vous créez une API HTTP. API Gateway prend également en charge les API WebSocket REST et les API, mais une API HTTP est le meilleur choix pour cet exercice. Les API REST prennent

en charge davantage de fonctions que les API HTTP, mais nous n'avons pas besoin de ces fonctions pour cet exercice. Les API HTTP sont conçues avec un minimum de fonctionnalités afin de pouvoir être proposées à un prix inférieur. WebSocket Les API maintiennent des connexions persistantes avec les clients pour les communications en duplex intégral, ce qui n'est pas obligatoire dans cet exemple.

L'API HTTP fournit un point de terminaison HTTP pour votre fonction Lambda. API Gateway achemine les requêtes vers votre fonction Lambda, puis renvoie la réponse de la fonction aux clients.

Pour créer une API HTTP

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Effectuez l'une des actions suivantes :
   - Pour créer votre première API, sous HTTP API (API HTTP), choisissez Build (Génération).
   - Si vous avez déjà créé une API, choisissez Create API (Créer une API), puis Build (Génération) sous HTTP API (API HTTP).
3. Sous Integrations (Intégrations), choisissez Add integration (Ajouter une intégration).
4. Choisissez Lambda.
5. Sous Lambda function (Fonction Lambda), saisissez **`my-function`**.
6. Sous API name (Nom de l'API), saisissez **`my-http-api`**.
7. Choisissez Suivant.
8. Consultez le routage créé par API Gateway pour vous, puis choisissez Next (Suivant).
9. Consultez l'étape créée par API Gateway pour vous, puis choisissez Next (Suivant).
10. Choisissez Créer.

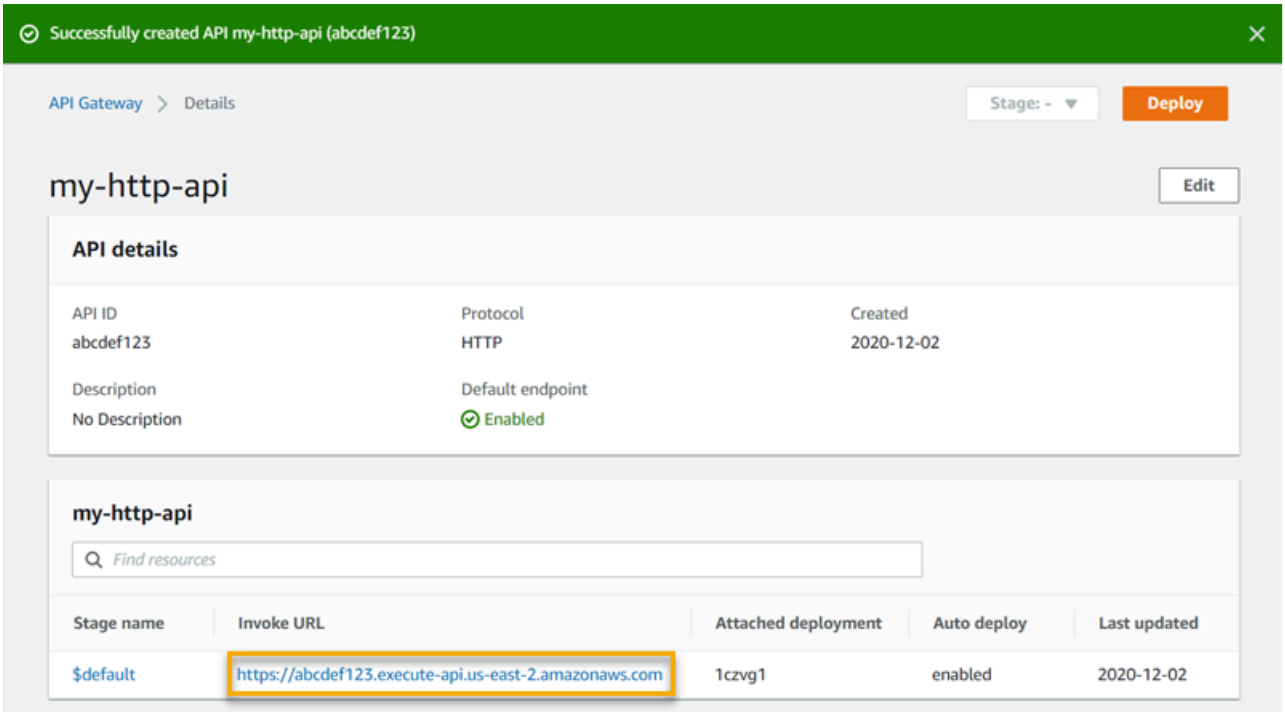
Vous avez maintenant créé une API HTTP avec une intégration Lambda, prête à recevoir des demandes de clients.

Étape 3 : Tester votre API

Ensuite, vous testez votre API pour vous assurer qu'elle fonctionne. Pour plus de simplicité, appelez votre API à l'aide d'un navigateur Web.

Pour tester votre API

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Choisissez votre API.
3. Notez l'URL d'appel de votre API.



4. Copiez l'URL d'appel de votre API et saisissez-la dans un navigateur Web. Ajoutez le nom de votre fonction Lambda à votre URL d'appel pour appeler votre fonction Lambda. Par défaut, la console API Gateway crée un routage portant le même nom que votre fonction Lambda, à savoi, `my-function`.

   L'URL complète doit ressembler à `https://`*`abcdef123`*`.execute-api.`*`us-east-2`*`.amazonaws.com/`*`my-function`*.

   Votre navigateur envoie une requête `GET` à l'API.
5. Vérifiez la réponse de votre API. Vous devriez voir le texte `"Hello from Lambda!"` dans votre navigateur.

(Facultatif) Étape 4 : Nettoyer

Pour éviter des coûts inutiles, supprimez les ressources que vous avez créées dans le cadre de cet exercice de démarrage. Les étapes suivantes suppriment votre API HTTP, votre fonction Lambda, ainsi que les ressources associées.

Pour supprimer une API HTTP

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Sur la page APIs (API) , sélectionnez une API. Choisissez Actions, puis Supprimer.
3. Sélectionnez Delete.

Pour supprimer une fonction Lambda

1. Connectez-vous à la console Lambda à l'adresse https://console.aws.amazon.com/lambda/.
2. Sur la page Functions (Fonctions), sélectionnez une fonction. Choisissez Actions, puis Supprimer.
3. Sélectionnez Delete.

Pour supprimer le groupe de journaux d'une fonction Lambda

1. Dans la CloudWatch console Amazon, ouvrez la page Log groups.
2. Sur la page Log groups (Groupes de journaux), sélectionnez le groupe de journaux de la fonction (`/aws/lambda/my-function`). Choisissez Actions, puis Supprimer le groupe de journaux.
3. Sélectionnez Delete.

Pour supprimer le rôle d'exécution d'une fonction Lambda

1. Dans la AWS Identity and Access Management console, ouvrez la page Rôles.
2. Sélectionnez le rôle de la fonction, par exempl, `my-function-31exxmpl`.
3. Choisissez Supprimer le rôle.
4. Choisissez Oui, supprimer.

Vous pouvez automatiser la création et le nettoyage des AWS ressources en utilisant AWS CloudFormation ou AWS SAM. Pour des exemples de modèles AWS CloudFormation , veuillez consulter exemples de modèles AWS CloudFormation.

Étapes suivantes

Dans cet exemple, vous avez utilisé le AWS Management Console pour créer une API HTTP simple. L'API HTTP appelle une fonction Lambda et renvoie une réponse aux clients.

Vous trouverez ci-dessous de prochaines étapes à suivre si vous souhaitez continuer à travailler avec API Gateway.

- Configurez d'autres types d'intégrations d'API, notamment les suivants :
  - Points de terminaison HTTP
  - Ressources privées dans un VPC, telles que les services Amazon ECS
  - AWS des services tels qu'Amazon Simple Queue Service et Kinesis Data Streams AWS Step Functions
- Contrôlez l'accès à vos API.
- Activez la journalisation pour vos API.
- Configurez la limitation pour vos API.
- Configurez des domaines personnalisés pour vos API.

Pour obtenir de l'aide sur Amazon API Gateway auprès de la communauté, veuillez consulter le forum de discussion API Gateway. Lorsque vous accédez à ce forum, vous devrez AWS peut-être vous connecter.

Pour obtenir de l'aide sur API Gateway directement auprès d'API Gateway AWS, consultez les options d'assistance sur la page AWS Support.

Consultez aussi nos questions fréquentes (FAQ) ou contactez-nous directement.

Didacticiels et ateliers Amazon API Gateway

Les didacticiels et ateliers suivants proposent des exercices pratiques qui vous permettent de découvrir API Gateway.

Didacticiels sur l'API REST

- Choisissez un didacticiel AWS Lambda d'intégration
- Tutoriel : Création d'une API REST par l'importation d'un exemple
- Choisissez un didacticiel d'intégration HTTP
- Tutoriel : Création d'une API REST avec l'intégration privée API Gateway
- Tutoriel : Création d'une API REST API Gateway avec AWS intégration
- Tutoriel : Création d'une API REST de calculatrice avec deux intégrations AWS de services et une intégration Lambda sans proxy
- Tutoriel : Création d'une API REST en tant que proxy Amazon S3 dans API Gateway
- Tutoriel : Création d'une API REST en tant que proxy Amazon Kinesis dans API Gateway
- Tutoriel : Création d'une API optimisée pour les périphériques à l'aide AWS de SDK ou AWS CLI
- Tutoriel : Création d'une API REST privée

Didacticiels sur l'API HTTP

- Didacticiel : création d'une API CRUD avec Lambda et DynamoDB
- Tutoriel : création d'une API HTTP avec une intégration privée à un Amazon ECS service

WebSocket Tutoriels d'API

- Tutoriel : Création d'une application de chat sans serveur avec une WebSocket API, Lambda et DynamoDB

Ateliers

- Création d'une application web serverless
- CI/CD pour les applications serverless

- Atelier sur la sécurité serverless
- Gestion des identités, authentification et autorisation serverless
- Atelier Amazon API Gateway

Didacticiels sur l'API REST Amazon API Gateway

Les didacticiels suivants proposent des exercices pratiques qui vous permettent de découvrir les API REST API Gateway.

Rubriques

- Choisissez un didacticiel AWS Lambda d'intégration
- Tutoriel : Création d'une API REST par l'importation d'un exemple
- Choisissez un didacticiel d'intégration HTTP
- Tutoriel : Création d'une API REST avec l'intégration privée API Gateway
- Tutoriel : Création d'une API REST API Gateway avec AWS intégration
- Tutoriel : Création d'une API REST de calculatrice avec deux intégrations AWS de services et une intégration Lambda sans proxy
- Tutoriel : Création d'une API REST en tant que proxy Amazon S3 dans API Gateway
- Tutoriel : Création d'une API REST en tant que proxy Amazon Kinesis dans API Gateway
- Tutoriel : Création d'une API optimisée pour les périphériques à l'aide AWS de SDK ou AWS CLI
- Tutoriel : Création d'une API REST privée

Choisissez un didacticiel AWS Lambda d'intégration

Pour créer une API avec des intégrations Lambda, vous pouvez utiliser l'intégration de proxy Lambda ou l'intégration non proxy Lambda.

Dans l'intégration du proxy Lambda, l'entrée de la fonction Lambda peut être exprimée sous la forme de n'importe quelle combinaison d'en-têtes de demande, de variables de chemin, de paramètres de chaîne de requête, de corps et de données de configuration d'API. Il suffit de choisir une fonction Lambda. API Gateway configure automatiquement la demande d'intégration et la réponse d'intégration. Une fois configurée, votre méthode d'API peut évoluer sans modifier les paramètres existants. Cela est possible car la fonction Lambda du backend analyse les données des demandes entrantes et répond au client.

Dans l'intégration autre que de proxy Lambda, vous devez vous assurer que les entrées de la fonction Lambda sont fournies en tant que charge utile de la demande d'intégration. Vous devez mapper toutes les données d'entrée fournies par le client en tant que paramètres de demande dans le corps de la demande d'intégration approprié. Vous pouvez également avoir besoin de traduire le corps de la demande fournie par le client en un format reconnu par la Fonction Lambda.

Dans le cadre d'un proxy Lambda ou d'une intégration Lambda sans proxy, vous pouvez utiliser une fonction Lambda dans un compte différent de celui sur lequel vous avez créé votre API.

Rubriques

- Tutoriel : Création d'une API REST Hello World avec l'intégration de proxy Lambda
- Tutoriel : Création d'une API REST API Gateway avec intégration non proxy Lambda
- Tutoriel : Création d'une API REST Gateway avec intégration de proxy Lambda entre comptes

Tutoriel : Création d'une API REST Hello World avec l'intégration de proxy Lambda

L'intégration de proxy Lambda est un type d'intégration d'API avec API Amazon Gateway léger et flexible qui permet d'intégrer une méthode d'API, ou une API complète, à une fonction Lambda. La fonction Lambda peut être écrite dans n'importe quelle langue prise en charge par Lambda. Puisqu'il s'agit d'une intégration de proxy, vous pouvez modifier la mise en œuvre de la fonction Lambda à tout moment, sans avoir à redéployer votre API.

Dans ce didacticiel, vous effectuez les opérations suivantes :

- Création d'une fonction Fonction Lambda pour être le backend de l'API.
- Créez et testez une API « Hello, World! » API avec intégration de proxy Lambda.

Rubriques

- Création d'une fonction Fonction Lambda
- Création d'une fonction API
- Déploiement et test de l'API

Création d'une fonction Fonction Lambda

Pour créer un « Hello, World! » Fonction Lambda dans la console Lambda

1. Connectez-vous à la console Lambda à l'adresse https://console.aws.amazon.com/lambda/.

2. Dans la barre AWS de navigation, choisissez un Région AWS.

> **Note**
>
> Notez la région où vous avez créé la fonction Lambda. Vous en aurez besoin lors de la création de l'API.

3. Choisissez Fonctions dans le panneau de navigation.
4. Choisissez Créer une fonction.
5. Choisissez Créer à partir de zéro.
6. Sous Informations de base, procédez comme suit :
   a. Sous Nom de la fonction, entrez **`GetStartedLambdaProxyIntegration`**.
   b. Pour Exécution, choisissez le dernier environnement d'exécution Node.js ou Python compatible.
   c. Sous Permissions (Autorisations), développez Change default execution role (Modifier le rôle d'exécution par défaut). Dans la liste déroulante Rôle d'exécution, choisissez Créer un nouveau rôle à partir de modèles de stratégie AWS .
   d. Sous Nom du rôle, entrez **`GetStartedLambdaBasicExecutionRole`**.
   e. Laissez vide le champ Modèles de stratégie.
   f. Choisissez Créer une fonction.
7. Sous Function code (Code de fonction), dans l'éditeur de code en ligne, copiez-collez le code suivant :

   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. Choisissez Deploy (Déployer).

Création d'une fonction API

Maintenant, créez une API pour votre « Hello, World !« Fonction Lambda à l'aide de la console API Gateway.

Pour créer un « Hello, World! » API

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Si vous utilisez API Gateway pour la première fois, vous voyez une page qui vous présente les fonctions du service. Sous REST API (API REST), choisissez Build (Création). Lorsque la fenêtre contextuelle Create Example API (Créer API exemple) s'affiche, cliquez sur OK.

Si ce n'est pas la première fois que vous utilisez API Gateway, choisissez Create API (Créer une API). Sous REST API (API REST), choisissez Build (Création).

3. Sous API name (Nom de l'API), saisissez **`LambdaProxyAPI`**.
4. (Facultatif) Sous Description, entrez une description.
5. Laissez Type de point de terminaison d'API défini sur Régional.
6. Sélectionnez Create API (Créer une API).

Une fois que vous avez créé une API, vous créez une ressource. En règle générale, les ressources API sont organisées dans une arborescence des ressources selon la logique de l'application. Pour cet exemple, vous créez une ressource /helloworld.

Pour créer une ressource

1. Sélectionnez la ressource /, puis choisissez Créer une ressource.
2. Maintenez Ressource proxy désactivée.
3. Conservez Chemin de la ressource sous la forme `/`.
4. Sous Resource Name (Nom de la ressource), entrez **`helloworld`**.
5. Maintenez CORS (Partage des ressources entre origines multiples) désactivé.
6. Choisissez Créer une ressource.

Dans une intégration de proxy, la totalité de la demande est envoyée à la fonction Lambda du backend en l'état, via une méthode `ANY` fourre-tout qui représente n'importe quelle méthode HTTP. La méthode HTTP concrète est spécifiée par le client au moment de l'exécution. La méthode `ANY` vous permet d'utiliser une seule configuration de méthode d'API pour toutes les méthodes HTTP prises en charge, `DELETE`, `GET`, `HEAD`, `OPTIONS`, `PATCH`, `POST` et `PUT`.

Pour créer une méthode **`ANY`**

1. Sélectionnez la ressource /helloworld, puis choisissez Créer une méthode.
2. Pour Type de méthode, sélectionnez ANY.
3. Pour Type d'intégration, sélectionnez Fonction Lambda.
4. Activez Intégration de proxy Lambda.
5. Pour la fonction Lambda, sélectionnez l' Région AWS endroit où vous avez créé votre fonction Lambda, puis entrez le nom de la fonction.

6. Pour utiliser la valeur de délai d'expiration par défaut de 29 secondes, gardez Délai d'expiration activé. Pour définir un délai d'expiration personnalisé, choisissez Délai d'expiration et entrez une valeur de délai d'expiration comprise entre `50` et `29000` millisecondes.
7. Choisissez Créer une méthode.

Déploiement et test de l'API

Pour déployer votre API

1. Sélectionnez Deploy API (Déployer une API).
2. Pour Étape, sélectionnez Nouvelle étape.
3. Sous Stage name (Nom de l'étape), entrez **`test`**.
4. (Facultatif) Sous Description, entrez une description.
5. Choisissez Deploy (Déployer).
6. Sous Détails de l'étape, choisissez l'icône de copie pour copier l'URL d'invocation de votre API.

Utilisation du navigateur et de cURL pour tester une API avec l'intégration de proxy Lambda

Pour tester votre API, vous pouvez utiliser un navigateur ou cURL.

Pour tester les `GET` requêtes en utilisant uniquement des paramètres de chaîne de requête, vous pouvez saisir l'URL de la `helloworld` ressource de l'API dans la barre d'adresse du navigateur.

Pour créer l'URL de la `helloworld` ressource de l'API, ajoutez la ressource `helloworld` et le paramètre de chaîne de requête `?greeter=John` à votre URL d'appel. Votre URL doit ressembler à ce qui suit.

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

Pour les autres méthodes, vous devez utiliser les utilitaires de test plus avancés de l'API REST, tels que POSTMAN ou cURL. Dans ce tutoriel, on utilise cURL. Dans les exemples de commande cURL ci-dessous, on suppose que cURL est installé sur votre ordinateur.

Pour tester l’API déployée à l’aide de cURL :

1. Ouvrez une fenêtre du terminal.

2. Copiez la commande cURL suivante et collez-la dans la fenêtre du terminal, et remplacez l'URL d'invocation par celle que vous avez copiée à l'étape précédente et ajoutez **`/helloworld`** à la fin de l'URL.

> **Note**
>
> Si vous exécutez la commande sous Windows, utilisez plutôt cette syntaxe :
>
> ```
> curl -v -X POST "https://r275xc9bmd.execute-api.us-east-1.amazonaws.com/
> test/helloworld" -H "content-type: application/json" -d "{ \"greeter\":
>  \"John\" }"
> ```

a. Pour appeler l'API avec le paramètre de chaîne de requête de `?greeter=John` :

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

b. Pour appeler l'API avec un paramètre d'en-tête de `greeter:John` :

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

c. Pour appeler l'API avec un corps `{"greeter":"John"}` :

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

Dans tous les cas, la sortie est une réponse 200 avec le corps de réponse suivant :

```
Hello, John!
```

Tutoriel : Création d'une API REST API Gateway avec intégration non proxy Lambda

Dans cette présentation, nous utilisons la console API Gateway pour créer une API qui permet à un client d'appeler des fonctions Lambda via l'intégration non proxy Lambda (également désignée sous le nom d'intégration personnalisée). Pour en savoir plus sur AWS Lambda et les fonctions Lambda, veuillez consulter le Guide du développeur AWS Lambda.

Pour faciliter l'apprentissage, nous avons choisi une fonction Lambda simple avec une configuration d'API minimale pour vous guider à travers les étapes de création d'une API Amazon API Gateway avec l'intégration personnalisée Lambda. Lorsque nécessaire, nous décrivons un aspect de la logique. Pour un exemple plus détaillé de l'intégration personnalisée Lambda, veuillez consulter Tutoriel : Création d'une API REST de calculatrice avec deux intégrations AWS de services et une intégration Lambda sans proxy.

Avant de créer l'API, configurez le backend Lambda en créant une fonction Lambda dans AWS Lambda, comme décrit ci-après.

Rubriques

Création d'une fonction Lambda pour l'intégration non proxy Lambda

> **Note**
>
> La création de fonctions Lambda peut entraîner des frais sur votre AWS compte.

Au cours de cette étape, vous allez créer une fonction Lambda de type « Hello, World ! » pour l'intégration Lambda personnalisée. Tout au long de cette procédure, la fonction est appelée `GetStartedLambdaIntegration`.

La mise en œuvre de cette fonction Lambda `GetStartedLambdaIntegration` est la suivante :

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}
```

Pour l'intégration personnalisée Lambda, API Gateway transmet les entrées à la fonction Lambda depuis le client en tant que corps de la demande d'intégration. L'objet `event` du gestionnaire de la fonction Lambda correspond à l'entrée.

Notre fonction Lambda est simple. Elle analyse l'objet `event` en entrée par rapport aux propriétés `name`, `city`, `time` et `day`. Elle renvoie ensuite un message d'accueil, en tant qu'objet JSON `{"message":greeting}`, à l'appelant. Le message suit le modèle `"Good [morning|afternoon|day], [name|you] in [city|World]. Happy day!"`. Il est supposé que l'entrée de la fonction Lambda est l'objet JSON suivant :

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

Pour plus d'informations, consultez le Manuel du développeur AWS Lambda.

En outre, la fonction enregistre son exécution sur Amazon CloudWatch en appelant`console.log(...)`. Cet aspect est précieux pour suivre les appels lors du débogage de la fonction. Pour permettre à la `GetStartedLambdaIntegration` fonction d'enregistrer l'appel, définissez un rôle IAM avec des politiques appropriées pour que la fonction Lambda crée CloudWatch les flux et ajoute des entrées de journal aux flux. La console Lambda vous guide tout au long de la création des rôles et stratégies IAM requis.

Si vous configurez l'API sans utiliser la console API Gateway, par exemple en importing une API depuis un fichier OpenAPI, vous devez créer explicitement, si nécessaire, et configurer une stratégie et un rôle d'appel pour qu'API Gateway puisse appeler les fonctions Lambda. Pour de plus amples informations sur la configuration des rôles d'appel et d'exécution Lambda pour une API Amazon API Gateway, veuillez consulter Contrôle de l'accès à une API avec des autorisations IAM.

Par rapport à `GetStartedLambdaProxyIntegration`, la fonction Lambda pour l'intégration de proxy Lambda, la fonction `GetStartedLambdaIntegration` Lambda pour l'intégration personnalisée Lambda prend uniquement les entrées du corps de la requête d'intégration d'API Amazon API Gateway. La fonction peut renvoyer une sortie sous forme d'un objet JSON, d'une chaîne, d'un nombre, d'une valeur booléenne ou même d'un blob binaire. La fonction Lambda de l'intégration de proxy Lambda, en revanche, peut accepter les entrées de toutes données de la demande, mais doit renvoyer en sortie un objet JSON particulier. La fonction `GetStartedLambdaIntegration` pour l'intégration personnalisée Lambda peut avoir comme entrées les paramètres de la demande d'API, à condition qu'API Gateway mappe les paramètres de la demande d'API au corps de demande d'intégration avant de transférer la demande du client au

backend. Pour que cela se produise, le développeur de l'API doit créer un modèle de mappage et le configurer sur la méthode de l'API lors de la création de l'API.

Maintenant, créez la fonction Lambda `GetStartedLambdaIntegration`.

Pour créer la fonction Lambda **`GetStartedLambdaIntegration`** pour l'intégration personnalisée Lambda

1. Ouvrez la AWS Lambda console à l'adresse https://console.aws.amazon.com/lambda/.
2. Effectuez l’une des actions suivantes :
   - Si la page de bienvenue s'affiche, choisissez Get Started Now (Commencer maintenant) puis Create function (Créer une fonction).
   - Si la liste Lambda > Functions (Fonctions) apparaît, choisissez Create function (Créer une fonction).
3. Choisissez Créer à partir de zéro.
4. Dans le volet Author from scratch (Créer à partir de zéro), procédez comme suit :
   a. Dans Name (Nom), entrez **`GetStartedLambdaIntegration`** comme nom de la fonction Lambda.
   b. Pour Exécution, choisissez le dernier environnement d’exécution Node.js ou Python compatible.
   c. Sous Permissions (Autorisations), développez Change default execution role (Modifier le rôle d'exécution par défaut). Dans la liste déroulante Rôle d’exécution, choisissez Créer un nouveau rôle à partir de modèles de stratégie AWS .
   d. Pour Role name (Nom du rôle), indiquez le nom de votre rôle, par exemple **`GetStartedLambdaIntegrationRole`**.
   e. Pour Policy templates (Modèles de stratégie), choisissez Simple microservice permissions (Autorisations Microservice simples).
   f. Choisissez Créer une fonction.
5. Dans le volet Configure function (Configurer la fonction), sous Function code (Code de la fonction), suivez les étapes suivantes :
   a. Copiez le code de la fonction Lambda affiché au début de cette section et collez-le dans l'éditeur de code en ligne.
   b. Acceptez les choix par défaut pour tous les autres champs de cette section.

c. Choisissez Deploy (Déployer).

6. Pour tester la fonction nouvellement créée, cliquez sur l'onglet Test.

   a. Dans Event name (Nom de l'événement), saisissez **`HelloWorldTest`**.

   b. Pour JSON d'événement, remplacez le code par défaut par le code suivant.

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

   c. Sélectionnez Test pour appeler la fonction. La section Execution result: succeeded s'affiche. Développez Détails pour voir la sortie suivante.

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

   La sortie est également écrite dans CloudWatch Logs.

À titre d'exercice, vous pouvez utiliser la console IAM pour afficher le rôle IAM (`GetStartedLambdaIntegrationRole`) créé dans le cadre de la création de la fonction Lambda. Deux stratégies en ligne sont attachées à ce rôle IAM. L'une stipule les autorisations les plus élémentaires pour l'exécution Lambda. Cela permet d'appeler toutes CloudWatch `CreateLogGroup` les CloudWatch ressources de votre compte dans la région où la fonction Lambda est créée. Cette politique permet également de créer les CloudWatch flux et de consigner les événements pour la fonction `GetStartedLambdaIntegration` Lambda.

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

L'autre document de politique s'applique à l'appel d'un autre AWS service qui n'est pas utilisé dans cet exemple. Vous pouvez l'ignorer pour l'instant.

Associée au rôle IAM se trouve une entité de confiance, qui est `lambda.amazonaws.com`. Voici la relation d'approbation :

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

La combinaison de cette relation de confiance et de la politique en ligne permet à la fonction Lambda d'appeler `console.log()` une fonction pour consigner les événements CloudWatch dans Logs.

Création d'une API avec l'intégration non proxy Lambda

Avec la fonction Lambda (`GetStartedLambdaIntegration`) créée et testée, vous êtes prêt à exposer la fonction via une API Amazon API Gateway. À des fins d'illustration, nous exposons la fonction Lambda avec une méthode HTTP générique. Nous utilisons le corps de la demande, une variable de chemin d'URL, une chaîne de requête et un en-tête pour recevoir les données d'entrée

requises du client. Nous activons le validateur de demande API Gateway pour l'API afin de garantir que toutes les données requises sont correctement définies et spécifiées. Nous configurons un modèle de mappage pour qu'API Gateway transforme les données de la demande fournie par le client au format valide comme requis par la fonction Lambda du backend.

Pour créer une API avec une intégration non proxy Lambda

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Si vous utilisez API Gateway pour la première fois, vous voyez une page qui vous présente les fonctions du service. Sous REST API (API REST), choisissez Build (Création). Lorsque la fenêtre contextuelle Create Example API (Créer API exemple) s'affiche, cliquez sur OK.

   Si ce n'est pas la première fois que vous utilisez API Gateway, choisissez Create API (Créer une API). Sous REST API (API REST), choisissez Build (Création).
3. Sous API name (Nom de l'API), saisissez **`LambdaNonProxyAPI`**.
4. (Facultatif) Sous Description, entrez une description.
5. Laissez Type de point de terminaison d'API défini sur Régional.
6. Sélectionnez Create API (Créer une API).

Après avoir créé votre API, vous créez une ressource /{city}. Ceci est un exemple de ressource avec une variable de chemin qui accepte une entrée du client. Par la suite, vous mapperez cette variable de chemin avec l'entrée de la fonction Lambda à l'aide d'un modèle de mappage.

Pour créer une ressource

1. Choisissez Créer une ressource.
2. Maintenez Ressource proxy désactivée.
3. Conservez Chemin de la ressource sous la forme `/`.
4. Sous Resource Name (Nom de la ressource), entrez **`{city}`**.
5. Maintenez CORS (Partage des ressources entre origines multiples) désactivé.
6. Choisissez Créer une ressource.

Après avoir créé votre ressource /{city}, vous créez une méthode `ANY`. Le verbe HTTP `ANY` est un espace réservé pour une méthode HTTP valide qu'un client soumet au moment de l'exécution. Cet

exemple montre que la méthode ANY peut être utilisée pour l'intégration personnalisée Lambda aussi bien que pour l'intégration de proxy Lambda.

Pour créer une méthode **ANY**

1. Sélectionnez la ressource /{city}, puis choisissez Créer une méthode.
2. Pour Type de méthode, sélectionnez ANY.
3. Pour Type d'intégration, sélectionnez Fonction Lambda.
4. Maintenez Intégration proxy Lambda désactivée.
5. Pour la fonction Lambda, sélectionnez l' Région AWS endroit où vous avez créé votre fonction Lambda, puis entrez le nom de la fonction.
6. Choisissez les paramètres de demande de méthode.

   À présent, vous activez un validateur de demande pour une variable de chemin d'URL, un paramètre de chaîne de requête et un en-tête afin de vous assurer que toutes les données requises sont définies. Dans cet exemple, vous créez un paramètre de chaîne de requête `time` et un en-tête `day`.
7. Pour Validateur de requête, sélectionnez Valider les paramètres de chaîne de requête et les en-têtes.
8. Choisissez Paramètres de chaîne de requête d'URL et procédez comme suit :

   a. Sélectionnez Add query string (Ajouter une chaîne de requêtes).

   b. Pour Name (Nom), saisissez **`time`**.

   c. Activez Obligatoire.

   d. Maintenez Mise en cache désactivée.
9. Choisissez En-têtes de requête HTTP et procédez comme suit :

   a. Sélectionnez Add header.

   b. Pour Name (Nom), saisissez **`day`**.

   c. Activez Obligatoire.

   d. Maintenez Mise en cache désactivée.
10. Choisissez Créer une méthode.

Après avoir activé un validateur de requête, vous configurez la requête d'intégration pour la méthode ANY en ajoutant un modèle de mappage corporel pour transformer la requête entrante en une charge utile JSON, comme l'exige la fonction Lambda du backend.

Pour configurer la requête d'intégration

1. Dans l'onglet Demande d'intégration, sous les paramètres de la demande d'intégration, choisissez Modifier.
2. Pour Transmission du corps de requête, sélectionnez Lorsqu'aucun modèle n'est défini (recommandé).
3. Choisissez Modèles de mappage.
4. Sélectionnez Add mapping template.
5. Pour Type de contenu, entrez **`application/json`**.
6. Pour Corps du modèle, entrez le code suivant :

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

7. Choisissez Enregistrer.

Test de l'appel de la méthode d'API

La console API Gateway fournit une installation de test qui vous permet de tester l'appel de l'API avant qu'elle ne soit déployée. Vous utilisez la fonctionnalité Test de la console pour tester l'API en soumettant la demande suivante :

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

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

Dans cette demande de test, vous allez définir ANY sur POST et {city} sur Seattle, attribuer Wednesday comme valeur de l'en-tête day et attribuer "John" comme valeur de callerName.

Pour tester la méthode **ANY**

1. Choisissez l'onglet Test. Vous devrez peut-être choisir la flèche droite pour afficher l'onglet.
2. Pour Type de méthode, sélectionnez POST.
3. Pour Chemin, sous city, entrez **Seattle**.
4. Pour Chaînes de requête, saisissez **time=morning**.
5. Pour En-têtes, saisissez **day:Wednesday**.
6. Pour Corps de la requête, saisissez **{ "callerName": "John" }**.
7. Sélectionnez Test.

Vérifiez que la charge utile de la réponse retournée se présente comme suit :

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

Vous pouvez également consulter les journaux pour examiner comment API Gateway traite la demande et la réponse.

```
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
```

Les journaux présentent la demande entrante avant le mappage et la demande d'intégration après le mappage. Lorsqu'un test échoue, les journaux sont utiles pour évaluer si l'entrée d'origine est correcte ou si le modèle de mappage fonctionne correctement.

Déploiement de l'API

L'appel du test est une simulation et possède ses limites. Par exemple, il contourne tout mécanisme d'autorisation activé sur l'API. Pour tester l'exécution de l'API en temps réel, vous devez d'abord déployer l'API. Pour déployer une API, vous créez une phase afin de créer un instantané de l'API à ce moment-là. Le nom de la phase définit également le chemin de base après le nom d'hôte par défaut de l'API. La ressource racine de l'API est ajoutée après le nom de la phase. Lorsque

vous modifiez l'API, vous devez la redéployer vers une phase nouvelle ou existante pour que les modifications prennent effet.

Pour déployer l'API sur une phase

1. Sélectionnez Deploy API (Déployer une API).
2. Pour Étape, sélectionnez Nouvelle étape.
3. Sous Stage name (Nom de l'étape), entrez **`test`**.

   > **Note**
   >
   > L'entrée doit être du texte codé en UTF-8 (par exemple, non localisé).

4. (Facultatif) Sous Description, entrez une description.
5. Choisissez Deploy (Déployer).

Sous Détails de l'étape, choisissez l'icône de copie pour copier l'URL d'invocation de votre API. Le modèle général de l'URL de base de l'API est `https://`*`api-id`*`.`*`region`*`.amazonaws.com/`*`stageName`*. Par exemple, l'URL de base de l'API (`beags1mnid`) créée dans la région `us-west-2` et déployée sur la phase `test` est `https://beags1mnid.execute-api.us-west-2.amazonaws.com/test`.

Test de l'API dans une phase de déploiement

Il existe plusieurs façons de tester une API déployée. Pour les requêtes GET n'utilisant que les variables de chemin d'URL ou les paramètres de chaîne de requête, vous pouvez saisir l'URL de la ressource d'API dans un navigateur. Pour les autres méthodes, vous devez utiliser les utilitaires de test plus avancés de l'API REST, tels que POSTMAN ou cURL.

Pour tester l'API à l'aide de cURL

1. Ouvrez une fenêtre de terminal sur votre ordinateur local connecté à Internet.
2. Pour tester `POST /Seattle?time=evening`:

   Copiez la commande cURL suivante et collez-la dans la fenêtre du terminal.

```
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"
}'
```

Vous devriez obtenir une réponse positive avec la charge utile suivante :

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

Si vous modifiez `POST` en `PUT` dans la demande de cette méthode, vous obtenez la même réponse.

Nettoyage

Si vous n'avez plus besoin des fonctions Lambda que vous avez créées pour cette procédure, vous pouvez maintenant les supprimer. Vous pouvez également supprimer les ressources IAM associées.

> **Warning**
>
> Si vous avez l'intention d'effectuer les autres procédures de cette série, ne supprimez pas le rôle d'exécution Lambda, ni le rôle d'appel Lambda. Si vous supprimez une fonction Lambda sur laquelle vos API reposent, ces API ne fonctionneront plus. La suppression d'une fonction Lambda ne peut pas être annulée. Si vous souhaitez utiliser à nouveau cette fonction Lambda, vous devez la recréer.
> Si vous supprimez une ressource IAM sur laquelle repose une fonction Lambda, cette fonction Lambda et les API reposant sur elle ne fonctionneront plus. La suppression d'une ressource IAM ne peut pas être annulée. Si vous souhaitez utiliser à nouveau cette ressource IAM, vous devez la recréer.

Pour supprimer la fonction Lambda

1. Connectez-vous à la AWS Lambda console AWS Management Console et ouvrez-la à l'adresse https://console.aws.amazon.com/lambda/.
2. Dans la liste des fonctions GetStartedLambdaIntegration, choisissez Actions, puis sélectionnez Supprimer la fonction. A l'invite, sélectionnez à nouveau Delete.

Pour supprimer les ressources IAM associées

1. Ouvrez la console IAM à l'adresse https://console.aws.amazon.com/iam/.
2. Dans le volet Details, sélectionnez Roles.
3. Dans la liste des rôles GetStartedLambdaIntegrationRole, choisissez Actions de rôle, puis choisissez Supprimer le rôle. Suivez les étapes indiquées dans la console pour supprimer le rôle.

Tutoriel : Création d'une API REST Gateway avec intégration de proxy Lambda entre comptes

Vous pouvez désormais utiliser une AWS Lambda fonction d'un autre AWS compte comme backend d'intégration d'API. Chaque compte peut se trouver dans n'importe quelle région où Amazon API Gateway est disponible. Cela facilite la gestion centralisée et le partage des fonctions de backend Lambda entre plusieurs API.

Dans cette section, nous montrons comment configurer l'intégration de proxy Lambda entre comptes à l'aide de la console Amazon API Gateway.

Création d'une API pour l'intégration Lambda entre comptes d'API Gateway

Pour créer une API

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Si vous utilisez API Gateway pour la première fois, vous voyez une page qui vous présente les fonctions du service. Sous REST API (API REST), choisissez Build (Création). Lorsque la fenêtre contextuelle Create Example API (Créer API exemple) s'affiche, cliquez sur OK.

   Si ce n'est pas la première fois que vous utilisez API Gateway, choisissez Create API (Créer une API). Sous REST API (API REST), choisissez Build (Création).
3. Sous API name (Nom de l'API), saisissez **`CrossAccountLambdaAPI`**.
4. (Facultatif) Sous Description, entrez une description.
5. Laissez Type de point de terminaison d'API défini sur Régional.
6. Sélectionnez Create API (Créer une API).

Création d'une fonction d'intégration Lambda dans un autre compte

Maintenant, vous allez créer une fonction Lambda dans un compte différent de celui dans lequel vous avez créé l'exemple d'API.

Création d'une fonction Lambda dans un autre compte

1. Connectez-vous à la console Lambda dans un compte différent de celui dans lequel vous avez créé votre API Amazon API Gateway.
2. Choisissez Créer une fonction.
3. Choisissez Créer à partir de zéro.
4. Sous Author from scratch, effectuez les étapes suivantes :
    a. Pour Nom de la fonction, entrez un nom.
    b. Dans la liste déroulante Exécution, choisissez un environnement d'exécution Node.js pris en charge.
    c. Sous Permissions (Autorisations), développez Choose or create an execution role (Choisir ou créer un rôle d'exécution). Vous pouvez créer un rôle ou choisir un rôle existant.
    d. Sélectionnez Create function pour continuer.
5. Faites défiler l'écran vers le bas jusqu'au volet Function code (Code de fonction).
6. Copiez et collez l'implémentation de la fonction Node.js depuis the section called "Tutoriel : API Hello World avec l'intégration de proxy Lambda".
7. Choisissez Deploy (Déployer).
8. Notez l'ARN complet de votre fonction (dans le coin supérieur droit du volet de la fonction Lambda). Vous en aurez besoin lors de la création de votre intégration Lambda entre comptes.

Configuration de l'intégration Lambda entre comptes

Une fois que vous disposez d'une fonction d'intégration Lambda dans un autre compte, vous pouvez utiliser la console API Gateway pour l'ajouter à votre API dans votre premier compte.

> **Note**
>
> Si vous configurez un mécanisme d'autorisation inter-compte entre régions, le `sourceArn` qui est ajouté à la fonction cible doit utiliser la région de la fonction, pas la région de l'API.

Une fois que vous avez créé une API, vous créez une ressource. En règle générale, les ressources API sont organisées dans une arborescence des ressources selon la logique de l'application. Pour cet exemple, vous créez une ressource /helloworld.

Pour créer une ressource

1. Sélectionnez la ressource /, puis choisissez Créer une ressource.
2. Maintenez Ressource proxy désactivée.
3. Conservez Chemin de la ressource sous la forme `/`.
4. Sous Resource Name (Nom de la ressource), entrez **`helloworld`**.
5. Maintenez CORS (Partage des ressources entre origines multiples) désactivé.
6. Choisissez Créer une ressource.

Après avoir créé une ressource, vous créez une méthode `GET`. Vous intégrez la méthode `GET` à une fonction Lambda dans un autre compte.

Pour créer une méthode **`GET`**

1. Sélectionnez la ressource /helloworld, puis choisissez Créer une méthode.
2. Pour Type de méthode, sélectionnez GET.
3. Pour Type d'intégration, sélectionnez Fonction Lambda.
4. Activez Intégration de proxy Lambda.
5. Pour la fonction Lambda, entrez l’ARN complet de votre fonction Lambda à l’étape 1.

   Dans la console Lambda, vous trouverez l'ARN de votre fonction dans le coin supérieur droit de la fenêtre de la console.
6. Lorsque vous entrez l’ARN, une chaîne de commande `aws lambda add-permission` apparaît. Cette politique permet d’accorder à votre premier compte l’accès à la fonction Lambda de votre second compte. Copiez et collez la chaîne de `aws lambda add-permission` commande dans une AWS CLI fenêtre configurée pour votre deuxième compte.
7. Choisissez Créer une méthode.

Vous pouvez afficher la politique de mise à jour de votre fonction dans la console Lambda.

(Facultatif) Pour consulter votre politique mise à jour

1. Connectez-vous à la AWS Lambda console AWS Management Console et ouvrez-la à l'adresse https://console.aws.amazon.com/lambda/.
2. Choisissez votre fonction Lambda.
3. Choisissez Permissions.

   Vous devriez voir une stratégie `Allow` avec une clause `Condition` dans laquelle l'`AWS:SourceArn` est l'ARN de la méthode `GET` de votre API.

Tutoriel : Création d'une API REST par l'importation d'un exemple

Vous pouvez utiliser la console Amazon API Gateway pour créer et tester une API REST simple avec intégration HTTP pour un PetStore site Web. La définition d'API est préconfigurée comme fichier OpenAPI 2.0. Après le chargement de la définition d'API dans API Gateway, vous pouvez utiliser la console API Gateway pour examiner la structure de base de l'API ou simplement déployer et tester l'API.

L' PetStore exemple d'API prend en charge les méthodes suivantes permettant à un client d'accéder au site Web principal HTTP de`http://petstore-demo-endpoint.execute-api.com/petstore/pets`.

> **Note**
>
> Ce didacticiel utilise un point de terminaison HTTP comme exemple. Lorsque vous créez vos propres API, nous vous recommandons d'utiliser des points de terminaison HTTPS pour vos intégrations HTTP.

- `GET /` : pour l'accès en lecture de la ressource racine de l'API qui n'est pas intégrée à un point de terminaison du serveur principal. API Gateway répond par un aperçu du PetStore site Web. Il s'agit d'un exemple du type d'intégration `MOCK`.
- `GET /pets` : pour l'accès en lecture de la ressource `/pets` de l'API intégrée à la ressource `/pets` du serveur principal de même nom. Le backend renvoie une page des animaux de compagnie disponibles dans le PetStore. Il s'agit d'un exemple du type d'intégration `HTTP`. L'URL du point de terminaison de l'intégration est `http://petstore-demo-endpoint.execute-api.com/petstore/pets`.

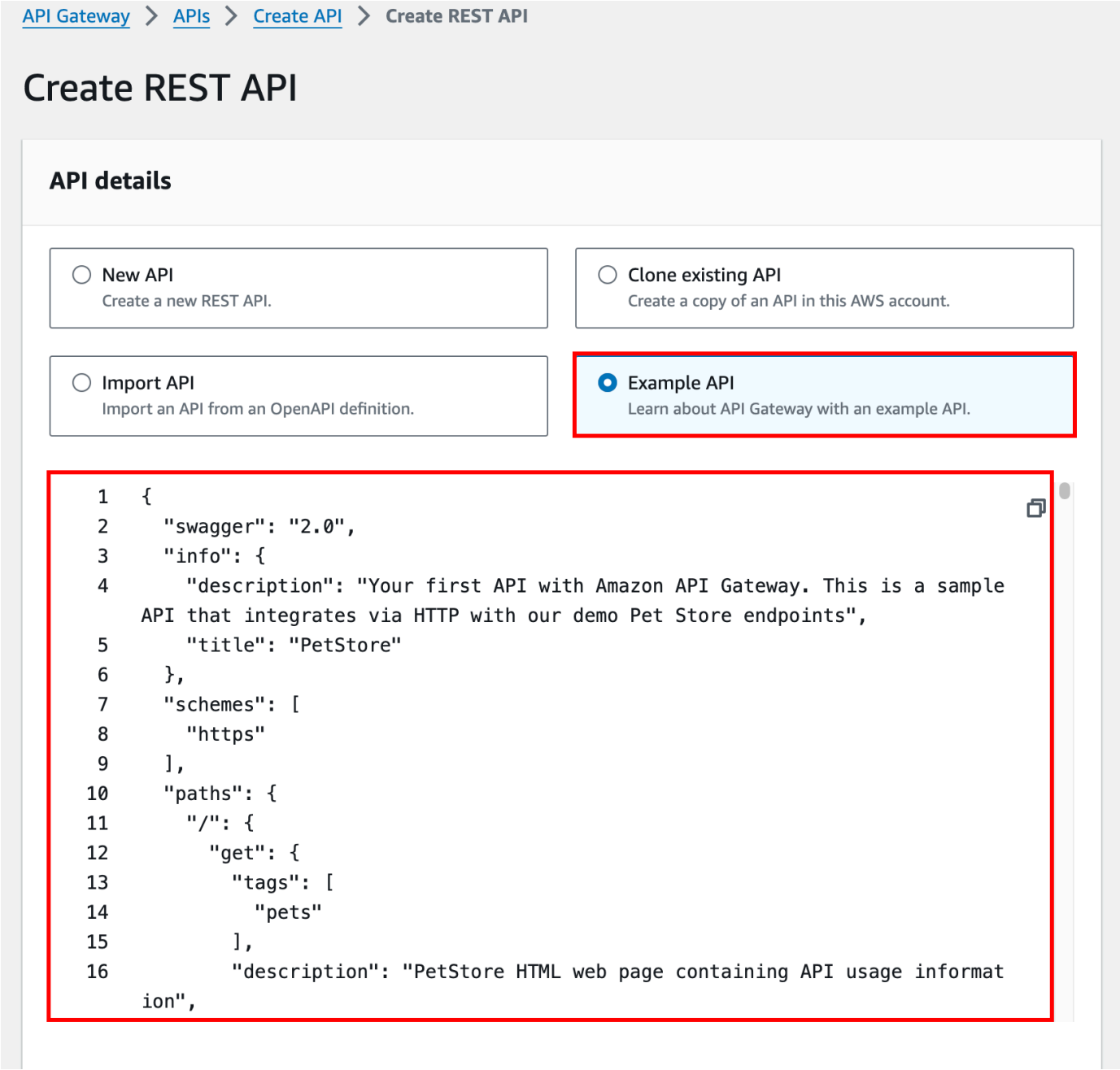
- `POST /pets` : pour l'accès en écriture de la ressource `/pets` de l'API intégrée à la ressource `/petstore/pets` du serveur principal. Dès réception d'une demande correcte, le backend ajoute l'animal spécifié au PetStore et renvoie le résultat à l'appelant. Il s'agit aussi d'une intégration `HTTP`.
- `GET /pets/{petId}` : pour l'accès en lecture à un animal identifié par une valeur `petId` telle que spécifiée comme variable du chemin d'accès de l'URL de la demande entrante. Cette méthode possède également le type d'intégration `HTTP`. Le backend renvoie l'animal de compagnie spécifié trouvé dans le PetStore. L'URL du point de terminaison HTTP du serveur principal est `http://petstore-demo-endpoint.execute-api.com/petstore/pets/`*n*, où n désigne un nombre entier comme identifiant de l'animal recherché.

L'API prend en charge l'accès CORS via les méthodes `OPTIONS` du type d'intégration `MOCK`. API Gateway renvoie les en-têtes requis prenant en charge l'accès CORS.

La procédure suivante vous guide à travers les étapes de création et de test d'une API à partir d'un exemple à l'aide de la console API Gateway.

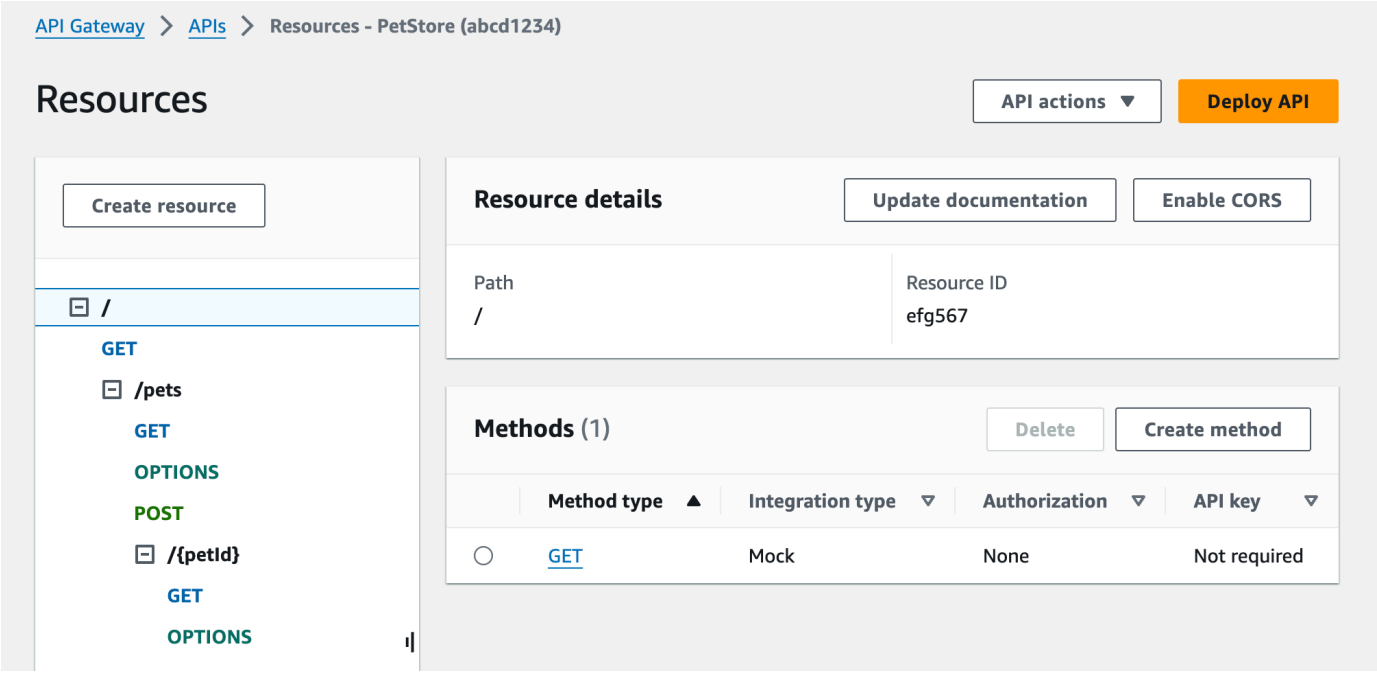
Pour importer, créer et tester l'exemple d'API

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Effectuez l’une des actions suivantes :
   - Pour créer votre première API, pour API REST, choisissez Création.
   - Si vous avez déjà créé une API, choisissez Créer une API, puis Création pour API REST.
3. Sous Créer une API REST, choisissez Exemple d'API, puis Créer une API pour créer l'exemple d'API.



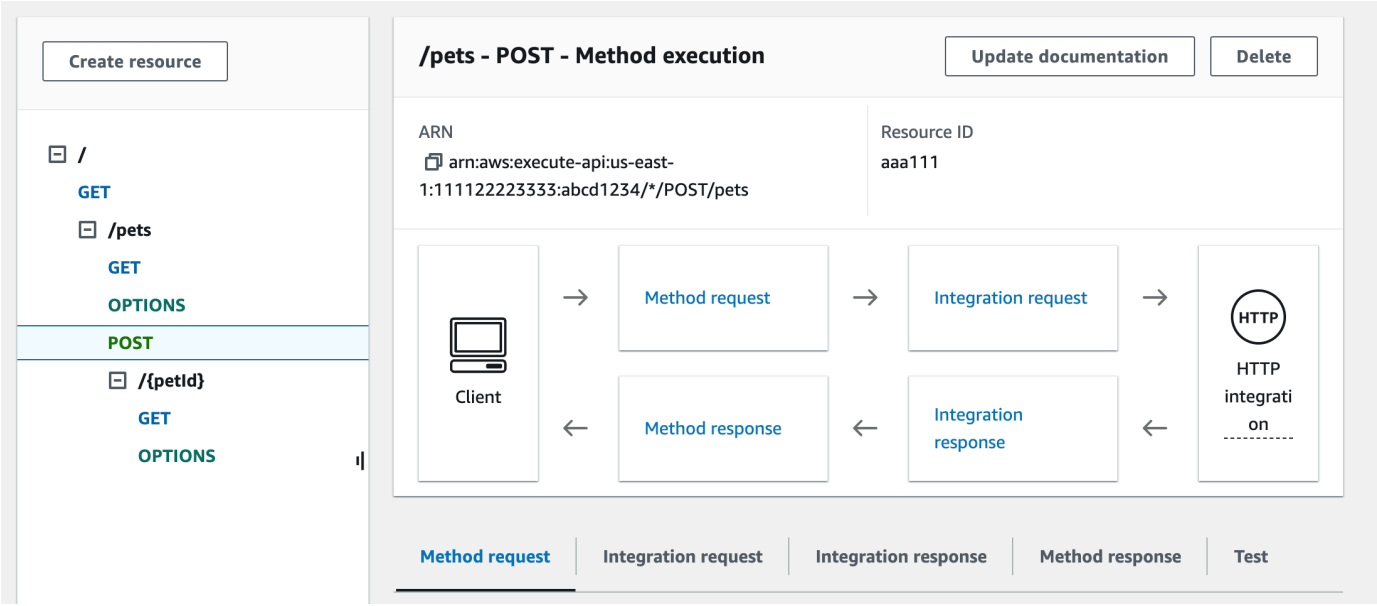
Vous pouvez faire défiler la définition OpenAPI vers le bas pour voir les détails de cet exemple d'API avant de choisir Créer une API.

4. Dans le volet de navigation principal, choisissez Ressources. L'API nouvellement créée est affichée comme suit :



Le volet Resources illustre la structure de l'API créée sous la forme d'une arborescence de nœuds. Les méthodes d'API définies sur chaque ressource sont les arêtes de l'arborescence. Lorsqu'une ressource est sélectionnée, toutes ses méthodes sont répertoriées dans la table Méthodes de droite. Le type de méthode, le type d'intégration, le type d'autorisation et la clé d'API requise sont affichés avec chaque méthode.

5. Pour afficher les détails d'une méthode, modifier sa configuration ou tester l'appel de cette méthode, sélectionnez le nom de la méthode dans la liste des méthodes ou dans l'arborescence des ressources. Ici, nous avons choisi la méthode `POST /pets` comme illustration :



Le volet Exécution de la méthode qui en résulte présente une vue logique de la structure et des comportements de la méthode choisie (`POST /pets`).

Demande de méthode et Réponse de méthode représentent l'interface de l'API avec le frontend, et Demande d'intégration et Réponse d'intégration représentent l'interface de l'API avec le backend.

Un client utilise l'API pour accéder à une fonctionnalité backend via Demande de méthode. Si nécessaire, API Gateway convertit la demande du client au format acceptable pour le backend dans Demande d'intégration avant de la transférer vers ce backend. La demande transformée est connue en tant que demande d'intégration. De même, le backend renvoie la réponse à API Gateway dans Réponse d'intégration. Ensuite, API Gateway l'achemine vers Method Response (Réponse de méthode) avant de l'envoyer au client. Là encore, si nécessaire, API Gateway peut mapper les données de réponse du backend au format attendu par le client.

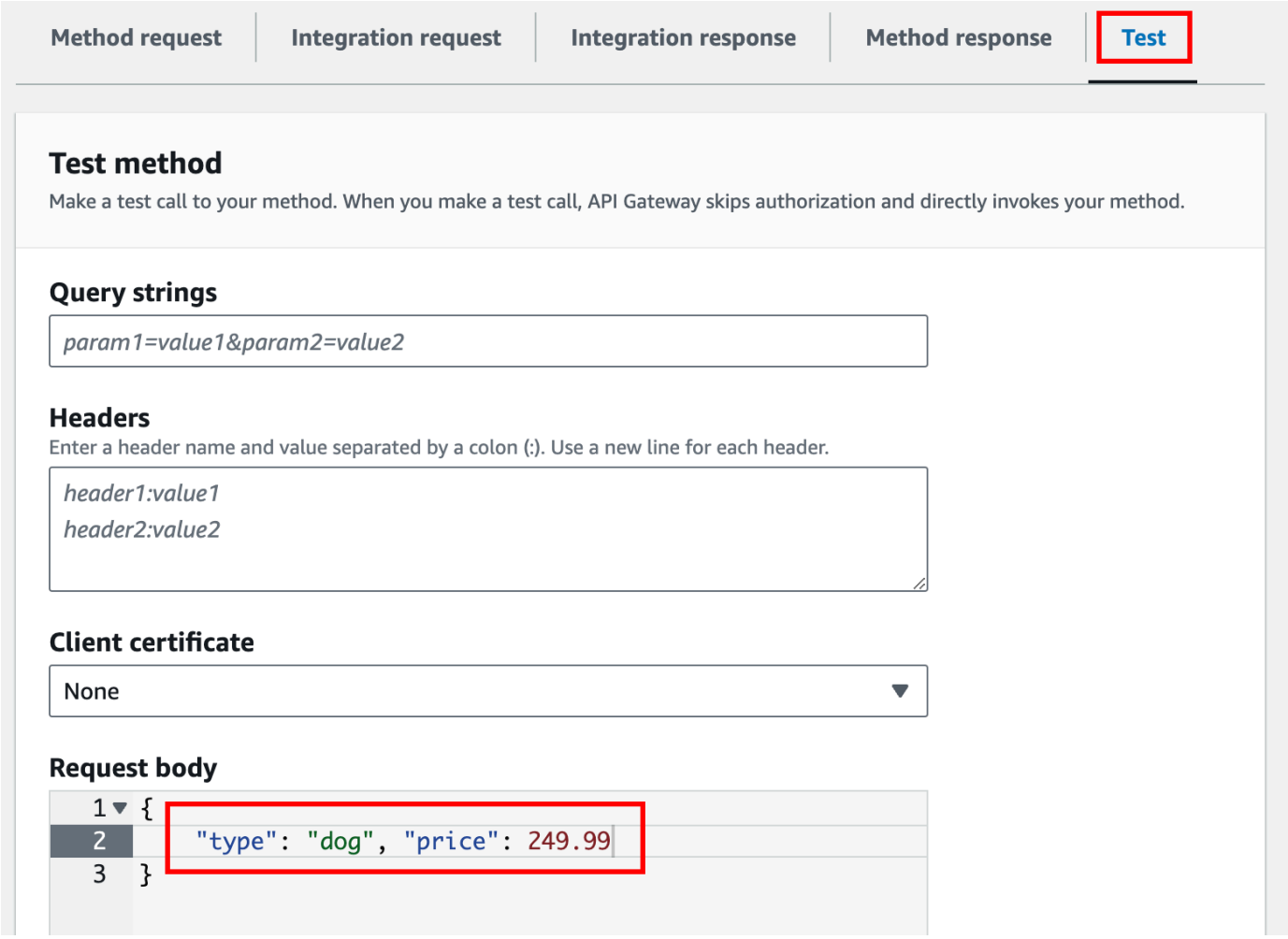
Pour la méthode `POST` sur une ressource d'API, la charge utile de la demande de méthode peut être transmise via la demande d'intégration sans modification, si la charge utile de la demande de méthode est au même format que charge utile de la demande d'intégration.

La demande de méthode `GET /` utilise le type d'intégration `MOCK` et n'est liée à aucun point de terminaison du serveur principal réel. La Réponse d'intégration correspondante est configurée pour renvoyer une page HTML statique. Lorsque la méthode est appelée, API Gateway accepte simplement la demande et renvoie immédiatement la réponse d'intégration configurée au client via Réponse de méthode. Vous pouvez utiliser l'intégration fictive pour tester une API sans avoir

besoin d'un point de terminaison sur le serveur principal. Vous pouvez également l'utiliser pour servir une réponse locale, générée à partir d'un modèle de mappage de corps de réponse.

En tant que développeur d'API, vous contrôlez les comportements des interactions frontales de votre API en configurant la demande de méthode et une réponse de méthode. Vous contrôlez les comportements des interactions principales de votre API en configurant la demande d'intégration et la réponse d'intégration. Cela implique des mappages de données entre une méthode et l'intégration correspondante. Pour le moment, nous nous concentrons sur le test de l'API afin d'offrir une expérience end-to-end utilisateur.

6. Sélectionnez l'onglet Test. Vous devrez peut-être choisir la flèche droite pour afficher l'onglet.
7. Par exemple, pour tester la méthode `POST /pets`, entrez la charge utile **`{"type": "dog","price": 249.99}`** suivante dans Corps de la demande avant de choisir Test.



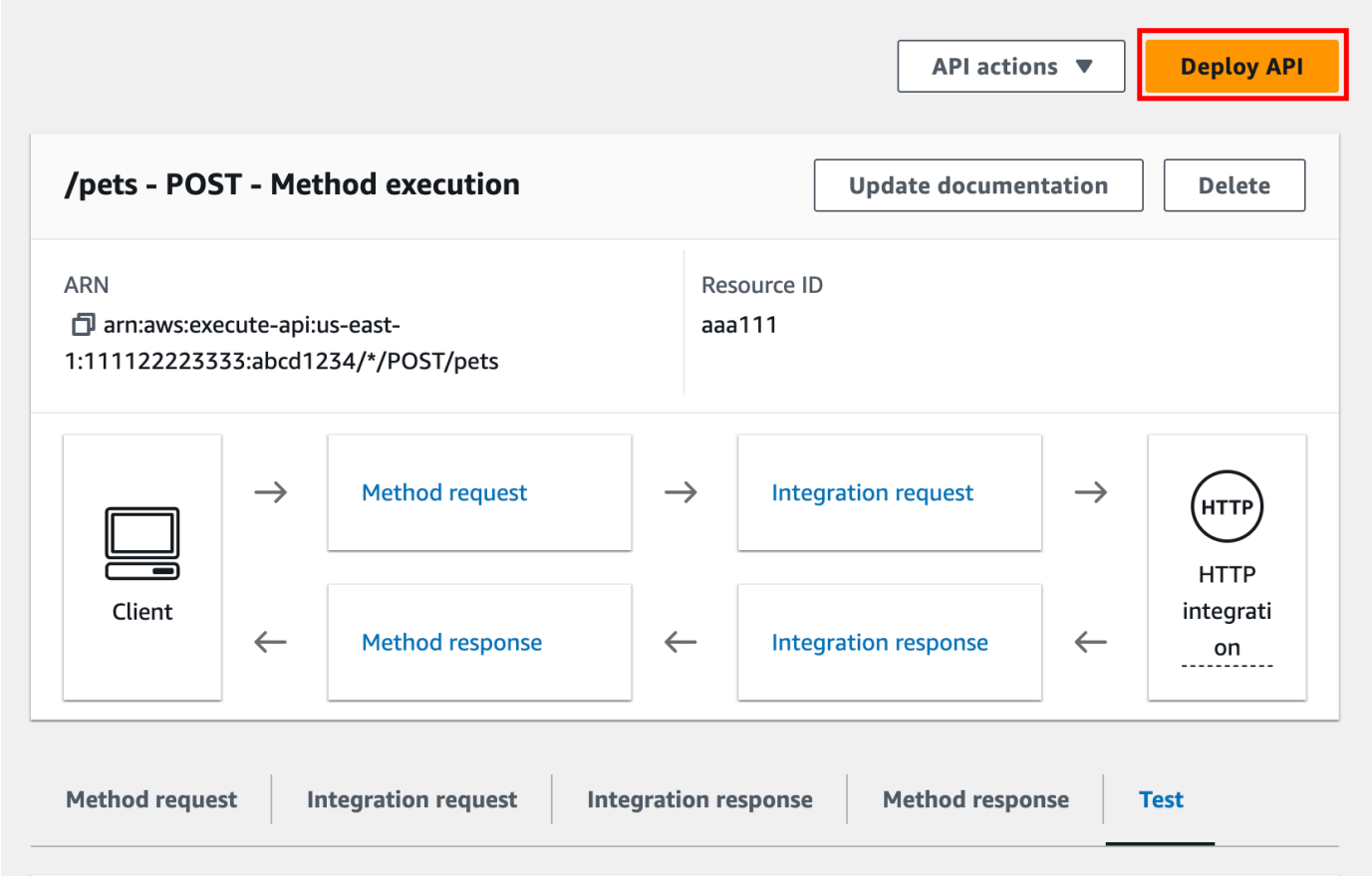
L'entrée indique les attributs de l'animal que nous voulons ajouter à la liste des animaux de compagnie sur le PetStore site Web.

8. Les résultats s'affichent comme suit :

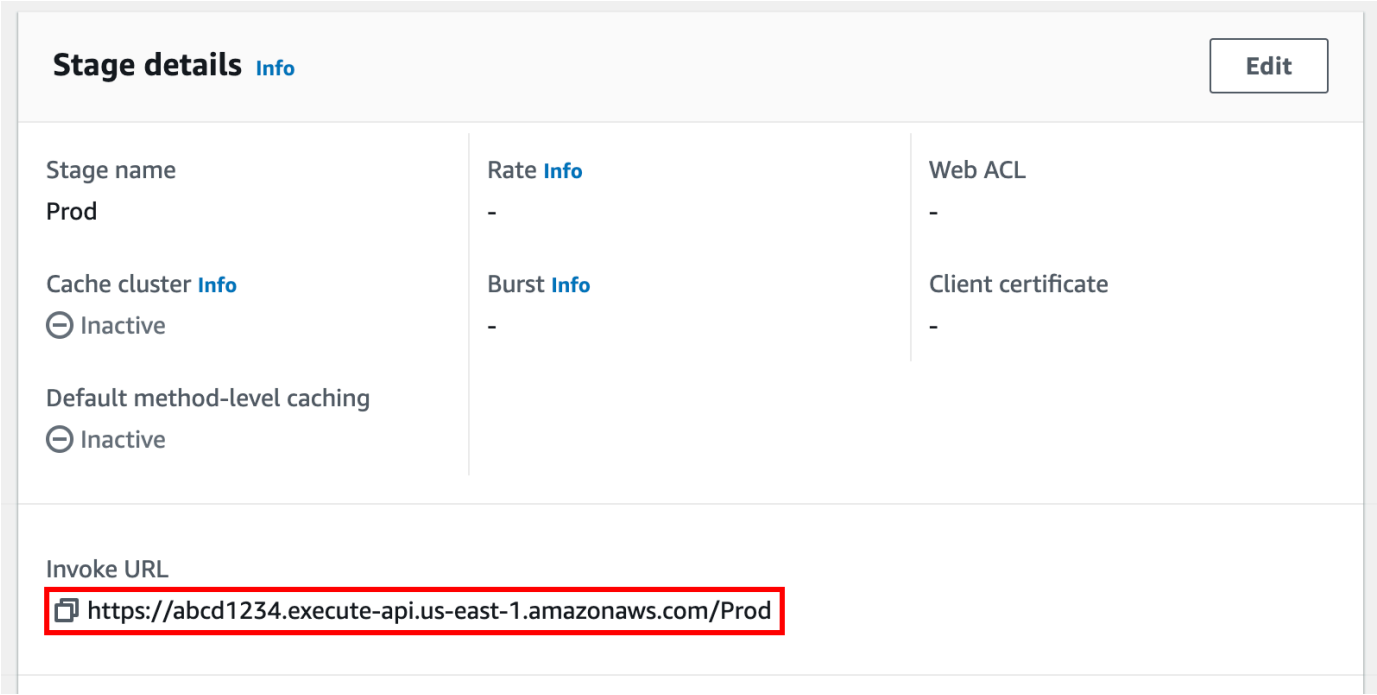
L'entrée Journal de la sortie indique les changements d'état entre la demande de la méthode et la demande d'intégration, et entre la réponse d'intégration et la réponse de la méthode. Cela peut être utile pour résoudre les éventuelles erreurs de mappage susceptibles d'entraîner l'échec de la demande. Dans cet exemple, aucun mappage n'est appliqué : la charge utile de la demande de méthode charge est transmise via la demande d'intégration au serveur principal et, de même la réponse du serveur principal est transmise via la réponse d'intégration à la réponse de la méthode.

Pour tester l'API à l'aide d'un client autre que la test-invoke-request fonctionnalité API Gateway, vous devez d'abord déployer l'API sur une étape.

9. Pour déployer l'exemple d'API jusqu'à une étape, choisissez Déployer l'API.



10. Pour Étape, sélectionnez Nouvelle étape, puis entrez **`test`**.
11. (Facultatif) Sous Description, entrez une description.
12. Choisissez Deploy (Déployer).
13. Dans le volet Étapes qui s'affiche, sous Détails de l'étape, Appeler une URL affiche l'URL permettant d'invoquer la demande de méthode `GET /` de l'API.



14. Choisissez l'icône de copie pour copier l'URL d'invocation de votre API, puis saisissez l'URL d'invocation de votre API dans un navigateur Web. Une réponse positive renvoie le résultat, généré à partir du modèle de mappage dans la réponse d'intégration.
15. Dans le panneau de navigation Stages (Étapes), développez l'étape test, sélectionnez GET sur `/pets/{petId}`, puis copiez la valeur du champ Invoke URL (Appeler une URL) de `https://`*`api-id`*`.execute-api.`*`region`*`.amazonaws.com/test/pets/{petId}`. `{petId}` représente une variable du chemin d'accès.

    Collez la valeur du champ Invoke URL (obtenue à l'étape précédente) dans la barre d'adresse d'un navigateur, en remplaçant `{petId}` par 1, par exemple, puis appuyez sur Entrée pour envoyer la demande. Une réponse 200 OK doit être renvoyée avec la charge utile JSON suivante :

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

    L'appel de la méthode d'API de cette façon est possible, car le type d'autorisation défini dans le champ Authorization est `NONE`. Si l'autorisation `AWS_IAM` était utilisée, vous devriez signer la demande à l'aide des protocoles Signature Version 4 (SigV4). Pour voir un exemple de

ce type de demande, consultez the section called “Tutoriel : Création d'une API avec une intégration HTTP autre que de proxy”.

Choisissez un didacticiel d'intégration HTTP

Pour créer une API avec intégration HTTP, vous pouvez utiliser une intégration de proxy HTTP ou une intégration HTTP personnalisée.

Dans le cadre d'une intégration de proxy HTTP, il vous suffit de définir la méthode HTTP et l'URI du point de terminaison HTTP, conformément aux exigences du backend. Nous vous recommandons d'utiliser l'intégration du proxy HTTP, dans la mesure du possible, pour tirer parti de la configuration simplifiée de l'API.

Vous pouvez utiliser une intégration HTTP personnalisée si vous devez transformer les données de demande client pour le backend ou transformer les données de réponse du backend pour le client.

Rubriques

- Tutoriel : Création d'une API REST avec une intégration de proxy HTTP
- Tutoriel : Création d'une API REST avec une intégration HTTP autre que de proxy

Tutoriel : Création d'une API REST avec une intégration de proxy HTTP

L'intégration de proxy HTTP est un mécanisme simple, puissant et polyvalent pour créer une API qui permet à une application web d'accéder à plusieurs ressources ou fonctionnalités du point de terminaison HTTP intégré, comme la totalité du site web, à l'aide d'une configuration rationalisée d'une seule méthode d'API. Dans l'intégration de proxy HTTP, API Gateway transmet au backend la demande de méthode envoyée par le client. Les données de la demande transmises incluent les en-têtes de demande, les paramètres de chaîne de requête, les variables du chemin de l'URL et la charge utile. Le point de terminaison HTTP du serveur principal ou le serveur web analyse les données de la demande entrante pour déterminer la réponse qu'il renvoie. L'intégration de proxy HTTP permet l'interaction directe entre le client et le backend sans intervention d'API Gateway après la configuration de la méthode d'API, sauf pour les problèmes connus comme les caractères non pris en charge, qui sont énumérés dans les the section called “Remarques importantes”.

Grâce à la ressource de proxy globale `{proxy+}` et au verbe fourre-tout `ANY` de la méthode HTTP, vous pouvez utiliser une intégration de proxy HTTP pour créer une API d'une seule méthode d'API. La méthode expose l'ensemble des ressources et opérations HTTP accessibles publiquement d'un

site web. Lorsque le serveur web principal ouvre de nouvelles ressources pour l'accès public, le client peut utiliser ces nouvelles ressources avec la même configuration d'API. Pour ce faire, le développeur du site web doit communiquer clairement au développeur du client quelles sont les nouvelles ressources et quelles opérations sont applicables pour chacune d'entre elles.

À titre de présentation rapide, le didacticiel suivant illustre l'intégration de proxy HTTP. Dans le didacticiel, nous créons une API à l'aide de la console API Gateway pour l'intégrer au PetStore site Web via une ressource proxy générique`{proxy+}`, et nous créons l'espace réservé à la méthode HTTP de`ANY`.

Rubriques

- Création d'une API avec l'intégration de proxy HTTP à l'aide de la console API Gateway
- Test d'une API avec l'intégration de proxy HTTP

Création d'une API avec l'intégration de proxy HTTP à l'aide de la console API Gateway

La procédure suivante explique comment créer et tester une API avec une ressource de proxy pour un backend HTTP à l'aide de la console API Gateway. Le backend HTTP est le site web `PetStore` (`http://petstore-demo-endpoint.execute-api.com/petstore/pets`) provenant du Tutoriel : Création d'une API REST avec une intégration HTTP autre que de proxy, dans lequel les captures d'écran sont utilisées comme aides visuelles pour illustrer les éléments de l'interface utilisateur API Gateway. Si c'est la première fois que vous utilisez la console API Gateway pour créer une API, il se peut que vous souhaitiez suivre d'abord les instructions de cette section.

Pour créer une API

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Si vous utilisez API Gateway pour la première fois, vous voyez une page qui vous présente les fonctions du service. Sous REST API (API REST), choisissez Build (Création). Lorsque la fenêtre contextuelle Create Example API (Créer API exemple) s'affiche, cliquez sur OK.

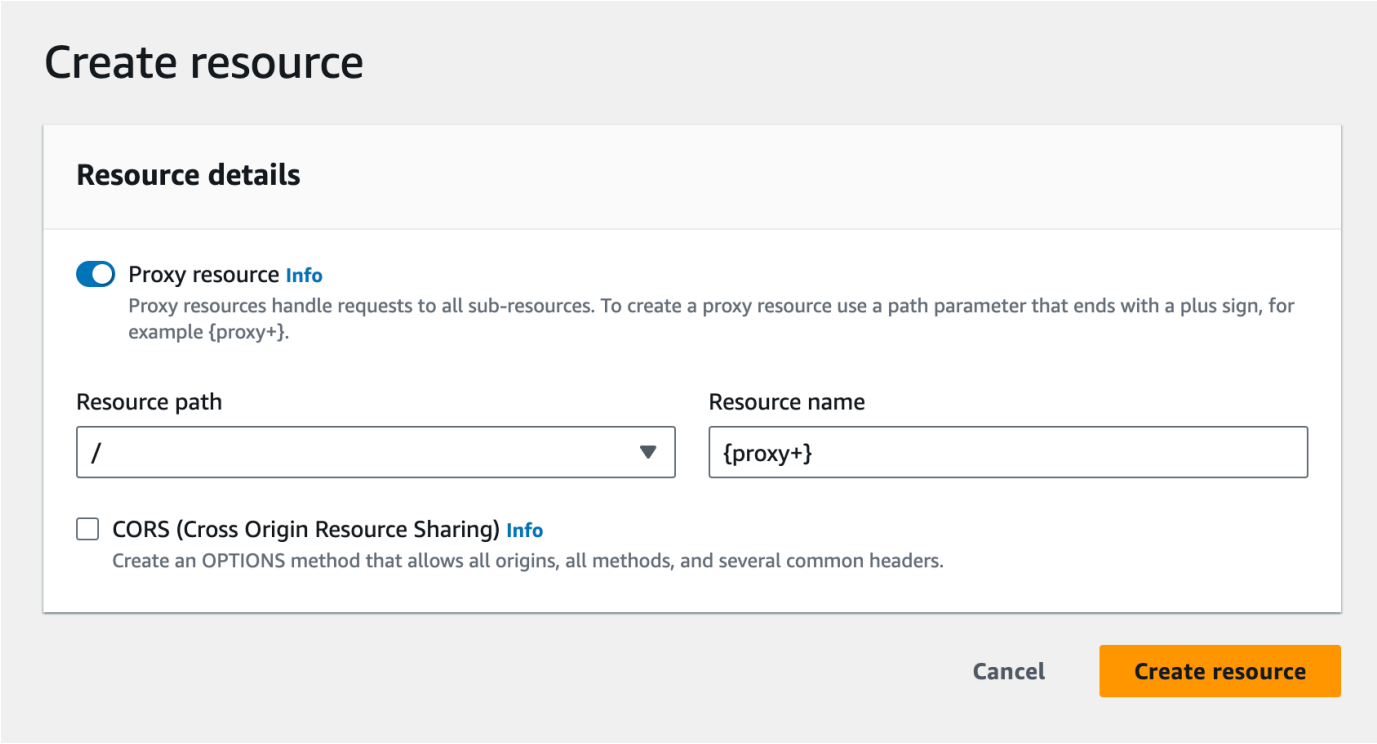
   Si ce n'est pas la première fois que vous utilisez API Gateway, choisissez Create API (Créer une API). Sous REST API (API REST), choisissez Build (Création).
3. Sous API name (Nom de l'API), saisissez **`HTTPProxyAPI`**.
4. (Facultatif) Sous Description, entrez une description.
5. Laissez Type de point de terminaison d'API défini sur Régional.

6. Sélectionnez Create API (Créer une API).

Dans cette étape, vous créez un chemin de ressource de proxy de `{proxy+}`. Il s'agit de l'espace réservé de chacun des points de terminaison du backend sous `http://petstore-demo-endpoint.execute-api.com/`. Par exemple, il peut être `petstore`, `petstore/pets` et `petstore/pets/{petId}`. API Gateway crée la méthode ANY lorsque vous créez la ressource `{proxy+}` et sert d'espace réservé pour chacun des verbes HTTP pris en charge au moment de l'exécution.

Pour créer une ressource/{proxy+}

1. Choisissez votre API.
2. Dans le volet de navigation principal, choisissez Ressources.
3. Choisissez Créer une ressource.
4. Activez Ressource proxy.
5. Conservez Chemin de la ressource sous la forme `/`.
6. Sous Resource Name (Nom de la ressource), entrez **`{proxy+}`**.
7. Maintenez CORS (Partage des ressources entre origines multiples) désactivé.
8. Choisissez Créer une ressource.



Au cours de cette étape, vous intégrez la méthode ANY à un point de terminaison HTTP de backend à l'aide d'une intégration par proxy. Dans l'intégration de proxy HTTP, API Gateway transmet au backend la requête de méthode envoyée par le client sans intervention d'API Gateway.

Pour créer une méthode **ANY**

1. Choisissez la ressource /{proxy+}.
2. Choisissez la méthode ANY.
3. Sous le symbole d'avertissement, choisissez Modifier l'intégration. Vous ne pouvez pas déployer une API dotée d'une méthode sans intégration.
4. Pour Type d'intégration, sélectionnez HTTP.
5. Activez Intégration de proxy HTTP.
6. Pour Méthode HTTP, sélectionnez ANY.
7. Pour URL du point de terminaison, saisissez **`http://petstore-demo-endpoint.execute-api.com/{proxy}`**.
8. Choisissez Enregistrer.

Test d'une API avec l'intégration de proxy HTTP

Indique si la réussite d'une demande client particulière dépend de ce qui suit :

- Si le serveur principal a rendu disponible le point de terminaison du serveur principal correspondant et, si tel est le cas, a accordé les autorisations d'accès requises.
- Si le client fournit la bonne entrée.

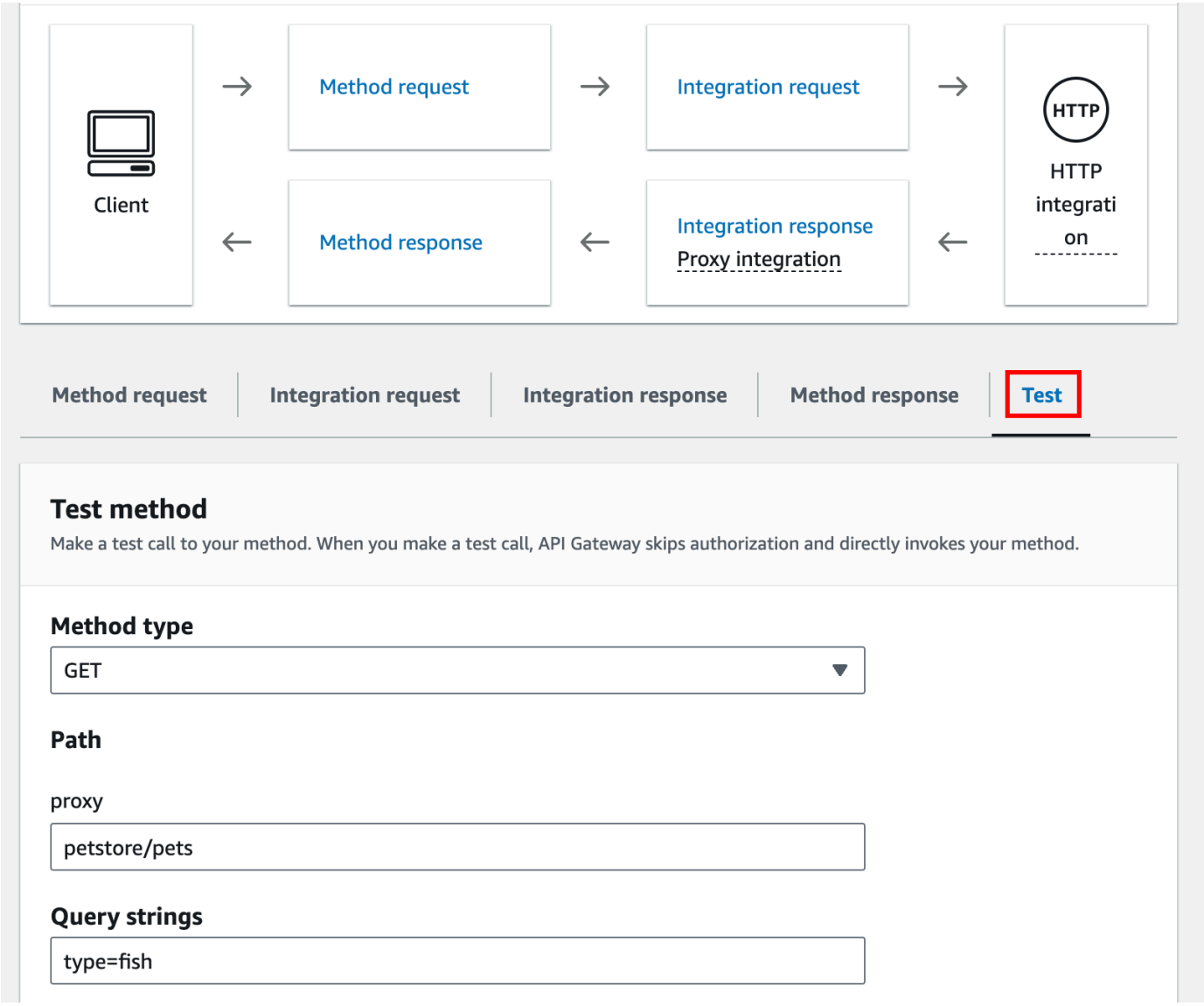
Par exemple, l' PetStore API utilisée ici n'expose pas la `/petstore` ressource. À ce titre, vous obtenez une réponse `404 Resource Not Found` contenant le message d'erreur `Cannot GET /petstore`.

De plus, le client doit être en mesure de gérer le format de sortie du serveur principal afin de pouvoir analyser le résultat correctement. API Gateway n'intervient pas pour faciliter les interactions entre le client et le backend.

Pour tester une API intégrée au PetStore site Web à l'aide de l'intégration d'un proxy HTTP via la ressource proxy

1. Sélectionnez l'onglet Test. Vous devrez peut-être choisir la flèche droite pour afficher l'onglet.

2. Pour Type de méthode, sélectionnez `GET`.
3. Pour Chemin, sous proxy, saisissez **`petstore/pets`**.
4. Pour Chaînes de requête, saisissez **`type=fish`**.
5. Sélectionnez Tester).



Comme le site web du serveur principal prend en charge la demande `GET /petstore/pets?type=fish`, il renvoie une réponse positive similaire à ce qui suit :

```
[
  {
    "id": 1,
    "type": "fish",
```

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

Si vous essayez d'appeler `GET /petstore`, vous obtenez une réponse `404` avec le message d'erreur `Cannot GET /petstore`. Cela est dû au fait que le serveur principal ne prend pas en charge l'opération spécifiée. Si vous appelez`GET /petstore/pets/1`, vous obtenez une `200 OK` réponse avec la charge utile suivante, car la demande est prise en charge par le PetStore site Web.

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

Vous pouvez utiliser un navigateur pour tester votre API. Déployez votre API et associez-la à une étape pour créer l'URL d'invocation de votre API.

Pour déployer votre API

1. Sélectionnez Deploy API (Déployer une API).
2. Pour Étape, sélectionnez Nouvelle étape.
3. Sous Stage name (Nom de l'étape), entrez **`test`**.
4. (Facultatif) Sous Description, entrez une description.
5. Choisissez Deploy (Déployer).

Les clients peuvent désormais appeler votre API.

Pour invoquer votre API

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Choisissez votre API.
3. Dans le volet de navigation principal, choisissez Étape.
4. Sous Détails de l'étape, choisissez l'icône de copie pour copier l'URL d'invocation de votre API.

   Saisissez l'URL d'invocation de votre API dans un navigateur Web.

   L'URL complète doit ressembler à `https://abcdef123.execute-api.us-east-2.amazonaws.com/test/petstore/pets?type=fish`.

   Votre navigateur envoie une requête `GET` à l'API.
5. Le résultat doit être identique à celui renvoyé lorsque vous utilisez Test depuis la console API Gateway.

Tutoriel : Création d'une API REST avec une intégration HTTP autre que de proxy

Dans ce tutoriel, vous créez une API à partir de zéro à l'aide de la console Amazon API Gateway. Vous pouvez envisager la console comme un studio de conception d'API et l'utiliser pour affiner les fonctions d'API, tester ses comportements, créer l'API et la déployer par étapes.

Rubriques

- Création d'une API avec l'intégration HTTP personnalisée
- (Facultatif) Mapper les paramètres de requête.

Création d'une API avec l'intégration HTTP personnalisée

Cette section vous guide tout au long des procédures de création de ressources, d'exposition de méthodes sur une ressource, de configuration d'une méthode pour obtenir les comportements d'API souhaités, et de test et de déploiement de l'API.

Au cours de cette étape, vous créez une API vide. Dans les étapes suivantes, vous allez créer des ressources et des méthodes pour connecter votre API au point de terminaison `http://petstore-demo-endpoint.execute-api.com/petstore/pets`, à l'aide d'une intégration HTTP sans proxy.

Pour créer une API

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Si vous utilisez API Gateway pour la première fois, vous voyez une page qui vous présente les fonctions du service. Sous REST API (API REST), choisissez Build (Création). Lorsque la fenêtre contextuelle Create Example API (Créer API exemple) s'affiche, cliquez sur OK.

   Si ce n'est pas la première fois que vous utilisez API Gateway, choisissez Create API (Créer une API). Sous REST API (API REST), choisissez Build (Création).
3. Sous API name (Nom de l'API), saisissez **`HTTPNonProxyAPI`**.
4. (Facultatif) Sous Description, entrez une description.
5. Laissez Type de point de terminaison d'API défini sur Régional.
6. Sélectionnez Create API (Créer une API).

L'arborescence Resources affiche la ressource racine (`/`) sans aucune méthode. Dans cet exercice, nous allons créer l'API avec l'intégration HTTP personnalisée du PetStore site Web (http ://petstore-demo-endpoint.execute-api.com/petstore/pets.) À des fins d'illustration, nous allons créer une `/pets` ressource en tant qu'enfant de la racine et exposer une méthode GET sur cette ressource afin qu'un client puisse récupérer une liste des articles Pets disponibles sur le PetStore site Web.

Pour créer une ressource /pets

1. Sélectionnez la ressource /, puis choisissez Créer une ressource.
2. Maintenez Ressource proxy désactivée.
3. Conservez Chemin de la ressource sous la forme `/`.
4. Sous Resource Name (Nom de la ressource), entrez **`pets`**.
5. Maintenez CORS (Partage des ressources entre origines multiples) désactivé.
6. Choisissez Créer une ressource.

Dans cette étape, vous créez une méthode `GET` pour la ressource /pets. La méthode `GET` est intégrée au site web `http://petstore-demo-endpoint.execute-api.com/petstore/pets`. Les autres options disponibles pour une méthode d'API sont les suivantes :

- POST, principalement utilisée pour créer des ressources enfants.

- PUT, principalement utilisée pour mettre à jour les ressources existantes (et, bien que ce ne soit pas recommandé, peut être utilisée pour créer des ressources enfants).
- DELETE, utilisée pour supprimer des ressources.
- PATCH, utilisée pour mettre à jour les ressources.
- HEAD, principalement utilisée dans les scénarios de test. Cette option est comparable à GET, mais ne renvoie pas la représentation de la ressource.
- OPTIONS, qui peut être utilisée par les appelants pour obtenir des informations sur les options de communication disponibles pour le service cible.

Dans la liste HTTP method, vous devez choisir pour la demande d'intégration une méthode HTTP prise en charge par le backend. Pour `HTTP` ou `Mock integration`, il est logique que la demande de méthode et la demande d'intégration utilisent le même verbe HTTP. Pour les autres types d'intégration, la demande de méthode utilisera probablement un autre verbe HTTP que la demande d'intégration. Par exemple, pour appeler une fonction Lambda, la demande d'intégration doit utiliser `POST` pour appeler la fonction, tandis que la demande de méthode peut utiliser n'importe quel verbe HTTP en fonction de la logique de la fonction Lambda.

Pour créer une méthode **`GET`** dans la ressource /pets

1. Sélectionnez la ressource /pets.
2. Choisissez Créer une méthode.
3. Pour Type de méthode, sélectionnez GET.
4. Pour Type d’intégration, sélectionnez Intégration HTTP.
5. Maintenez Intégration de proxy HTTP désactivée.
6. Pour Méthode HTTP, sélectionnez GET.
7. Pour URL du point de terminaison, saisissez **`http://petstore-demo-endpoint.execute-api.com/petstore/pets`**.

   Le PetStore site Web vous permet de récupérer une liste d'`Pet`articles par type d'animal de compagnie, comme « chien » ou « chat », sur une page donnée.
8. Pour Gestion de contenu, sélectionnez Transmettre.
9. Choisissez Paramètres de chaîne de requête d’URL.

   Le PetStore site Web utilise les paramètres de chaîne de `page` requête `type` et pour accepter une entrée. Vous ajoutez des paramètres de chaîne de requête à la demande de méthode et

vous les mappez dans les paramètres de chaîne de requête correspondants de la demande d'intégration.

10. Pour ajouter les paramètres de chaîne de requête, procédez comme suit :

    a. Sélectionnez Add query string (Ajouter une chaîne de requêtes).

    b. Pour Nom, saisissez **`type`**

    c. Gardez Obligatoire et Mise en cache désactivés.

    Répétez les étapes précédentes pour créer une chaîne de requête supplémentaire avec le nom **`page`**.

11. Choisissez Créer une méthode.

Le client peut désormais fournir un type d'animal et un numéro de page comme paramètres de chaîne de requête lorsqu'il envoie une demande. Ces paramètres d'entrée doivent être mappés dans les paramètres de chaîne de requête de l'intégration pour transmettre les valeurs d'entrée à notre PetStore site Web dans le backend.

Pour mapper les paramètres d’entrée à la requête d’intégration

1. Dans l’onglet Requête d’intégration, sous Paramètres de requête d’intégration, choisissez Modifier.
2. Choisissez Paramètres de chaîne de requête d’URL, puis procédez comme suit :

    a. Choisissez Ajouter un paramètre de chaîne de requête d’URL.

    b. Pour Name (Nom), saisissez **`type`**.

    c. Pour Mappage à partir de, saisissez **`method.request.querystring.type`**.

    d. Maintenez Mise en cache désactivée.

    e. Choisissez Ajouter un paramètre de chaîne de requête d’URL.

    f. Pour Name (Nom), saisissez **`page`**.

    g. Pour Mappage à partir de, saisissez **`method.request.querystring.page`**.

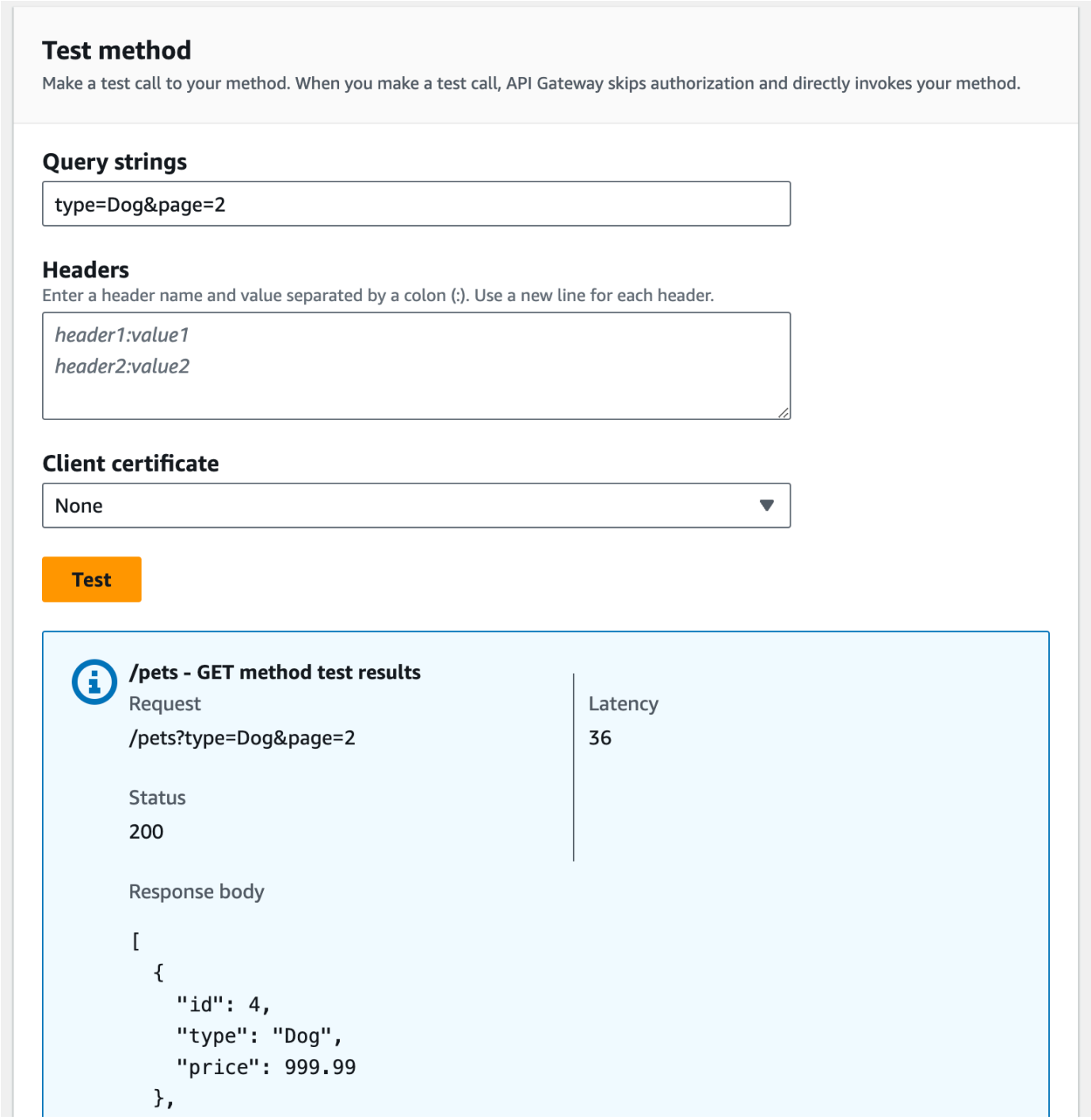
    h. Maintenez Mise en cache désactivée.

3. Choisissez Enregistrer.

Pour tester l'API

1. Choisissez l’onglet Test. Vous devrez peut-être choisir la flèche droite pour afficher l'onglet.
2. Pour Chaînes de requête, saisissez **`type=Dog&page=2`**.
3. Sélectionnez Tester).

   Le résultat est similaire à ce qui suit :



Maintenant que le test est réussi, nous pouvons déployer l'API pour la rendre disponible publiquement.

4. Sélectionnez Deploy API (Déployer une API).
5. Pour Étape, sélectionnez Nouvelle étape.
6. Sous Stage name (Nom de l'étape), entrez **`Prod`**.

7. (Facultatif) Sous Description, entrez une description.
8. Choisissez Deploy (Déployer).
9. (Facultatif) Sous Détails de l'étape, pour Invoquer une URL, vous pouvez choisir l'icône de copie pour copier l'URL d'invocation de votre API. Vous pouvez l'utiliser avec des outils tels que Postman et cURL pour tester votre API.

Si vous utilisez un kit SDK pour créer un client, vous pouvez appeler les méthodes exposées par ce kit SDK pour signer la demande. Pour en savoir plus sur la mise en œuvre, veuillez consulter le kit SDK AWS de votre choix.

> **Note**
>
> Lorsque des modifications sont apportées à votre API, vous devez la redéployer pour rendre les nouvelles fonctions ou celles mises à jour disponibles avant d'appeler à nouveau l'URL de demande.

(Facultatif) Mapper les paramètres de requête.

Paramètres de demande de mappage pour une API API Gateway

Ce didacticiel montre comment créer un paramètre de chemin `{petId}` sur l'URL de requête de méthode de l'API pour spécifier un ID d'élément, mappage de celui-ci au paramètre de chemin `{id}` dans l'URL de requête d'intégration, et envoi de la requête au point de terminaison HTTP.

> **Note**
>
> La saisie d'une lettre majuscule (ou vice versa) peut entraîner des erreurs ultérieurement dans la procédure.

Étape 1 : Créer des ressources

Au cours de cette étape, vous créez une ressource avec un paramètre de chemin {petId}.

Pour créer la ressource {petId}

1. Sélectionnez la ressource /pets, puis choisissez Créer une ressource.
2. Maintenez Ressource proxy désactivée.

3. Pour Chemin de ressource, sélectionnez /pets/.
4. Sous Resource Name (Nom de la ressource), entrez **`{petId}`**.

   Encadrez `petId` par des accolades (`{ }`) pour afficher /pets/{petId}.
5. Maintenez CORS (Partage des ressources entre origines multiples) désactivé.
6. Choisissez Créer une ressource.

Étape 2 : Créer et tester les méthodes

Au cours de cette étape, vous créez une méthode `GET` avec un paramètre de chemin `{petId}`.

Pour configurer la méthode GET

1. Sélectionnez la ressource /{petId}, puis choisissez Créer une méthode.
2. Pour Type de méthode, sélectionnez GET.
3. Pour Type d'intégration, sélectionnez Intégration HTTP.
4. Maintenez Intégration de proxy HTTP désactivée.
5. Pour Méthode HTTP, sélectionnez GET.
6. Pour URL du point de terminaison, saisissez **`http://petstore-demo-endpoint.execute-api.com/petstore/pets/{id}`**
7. Pour Gestion de contenu, sélectionnez Transmettre.
8. Maintenez Délai d'expiration par défaut activé.
9. Choisissez Créer une méthode.

Vous devez maintenant mapper le paramètre de chemin `{petId}` au paramètre de chemin `{id}` dans le point de terminaison HTTP.

Pour mapper le paramètre de chemin **`{petId}`**

1. Dans l'onglet Requête d'intégration, sous Paramètres de requête d'intégration, choisissez Modifier.
2. Choisissez paramètres de chemin d'URL.
3. API Gateway crée un paramètre de chemin pour la requête d'intégration nommé petId. Cela ne fonctionne pas pour votre backend. Le point de terminaison HTTP utilise `{id}` comme paramètre de chemin. Renommez petId en **`id`**.

Cela mappe le paramètre de chemin `petId` de la demande de méthode au paramètre de chemin `id` de la demande d'intégration.

4. Choisissez Enregistrer.

Maintenant, testez la méthode.

Pour tester la méthode

1. Choisissez l'onglet Test. Vous devrez peut-être choisir la flèche droite pour afficher l'onglet.
2. Sous Chemin pour petId entrez **4**.
3. Sélectionnez Test.

   Si le test aboutit, Corps de la réponse affiche les informations suivantes :

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

Étape 3 : Déploiement de l'API

Dans cette étape, vous allez déployer l'API afin de pouvoir commencer à l'appeler en dehors de la console API Gateway.

Pour déployer l'API

1. Sélectionnez Deploy API (Déployer une API).
2. Pour Étape, sélectionnez Prod.
3. (Facultatif) Sous Description, entrez une description.
4. Choisissez Deploy (Déployer).

Étape 4 : Test de l'API

Dans cette étape, vous allez sortir de la console API Gateway et utiliser votre API pour accéder au point de terminaison HTTP.

1. Dans le volet de navigation principal, choisissez Étape.
2. Sous Détails de l'étape, choisissez l'icône de copie pour copier l'URL d'invocation de votre API.

   Elle doit se présenter comme suit :

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

3. Entrez cette URL dans la zone d'adresse d'un nouvel onglet du navigateur et ajoutez `/pets/4` à l'URL avant d'envoyer votre requête.
4. Le navigateur retourne ce qui suit :

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

Étapes suivantes

Vous pouvez personnaliser davantage votre API en activant la validation des requêtes, en transformant les données ou en créant des réponses de passerelle client.

Pour découvrir d'autres moyens de personnaliser votre API, consultez les didacticiels suivants :

- Pour plus d'informations sur la validation de demande, consultez Configuration de la validation de demande de base dans API Gateway.
- Pour plus d'informations sur la transformation des charges utiles de requête et de réponse, consultez Configuration de transformations de données dans API Gateway.
- Pour plus d'informations sur la création d'une passerelle client, consultez Configurer une réponse de passerelle pour une API REST à l'aide de la console API Gateway.

Tutoriel : Création d'une API REST avec l'intégration privée API Gateway

Vous pouvez créer une API API Gateway avec intégration privée pour permettre à vos clients d'accéder aux ressources HTTP/HTTPS dans votre Amazon Virtual Private Cloud (Amazon VPC). Ces ressources VPC sont des points de terminaison HTTP/HTTPS sur une instance EC2 derrière un Network Load Balancer dans le VPC. Le Network Load Balancer encapsule la ressource VPC et achemine les demandes entrantes vers la ressource ciblée.

Lorsqu'un client appelle l'API, API Gateway se connecte au Network Load Balancer par le biais du lien VPC préconfiguré. Un lien VPC est encapsulé par une ressource API Gateway de. VpcLink Il est responsable de la transmission des demandes de méthode d'API aux ressources VPC et renvoie les réponses du backend à l'appelant. Pour un développeur d'API, un `VpcLink` est une fonctionnalité équivalente à un point de terminaison d'intégration.

Pour créer une API avec intégration privée, vous devez créer ou choisir un `VpcLink` existant connecté à un Network Load Balancer qui cible les ressources VPC souhaitées. Vous devez avoir les autorisations appropriées pour créer et gérer un `VpcLink`. Ensuite, vous configurez une méthode d'API et vous l'intégrez au `VpcLink` en définissant `HTTP` ou `HTTP_PROXY` comme type d'intégration, en définissant `VPC_LINK` comme type de connexion d'intégration, et en définissant l'identifiant `VpcLink` sur l'élément `connectionId` de l'intégration.

> **Note**
>
> Le Network Load Balancer et l'API doivent appartenir au même AWS compte.

Pour commencer rapidement à créer une API pour accéder aux ressources VPC, nous allons passer en revue les étapes essentielles de création d'une API avec intégration privée, à l'aide de la console API Gateway. Avant de créer l'API, procédez comme suit :

1. Créez une ressource VPC, créez ou choisissez un Network Load Balancer sous votre compte dans la même région, et ajoutez l'instance EC2 hébergeant la ressource en tant que cible du Network Load Balancer. Pour de plus amples informations, veuillez consulter Configuration d'un équilibreur Network Load Balancer pour les intégrations privées API Gateway.
2. Accordez des autorisations pour créer les liens VPC pour les intégrations privées. Pour de plus amples informations, veuillez consulter Octroi d'autorisations pour créer un lien VPC.

Après avoir créé votre ressource VPC et votre Network Load Balancer avec votre ressource VPC configurée dans ses groupes cibles, suivez les instructions ci-dessous pour créer une API et l'intégrer à la ressource VPC via un `VpcLink` dans une intégration privée.

Pour créer une API avec une intégration privée

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.

2. Si vous utilisez API Gateway pour la première fois, vous voyez une page qui vous présente les fonctions du service. Sous REST API (API REST), choisissez Build (Création). Lorsque la fenêtre contextuelle Create Example API (Créer API exemple) s'affiche, cliquez sur OK.

   Si ce n'est pas la première fois que vous utilisez API Gateway, choisissez Create API (Créer une API). Sous REST API (API REST), choisissez Build (Création).
3. Créez une API REST régionale ou optimisée pour la périphérie.
4. Sélectionnez votre API.
5. Choisissez Créer une méthode puis procédez comme suit :

   a. Pour Type de méthode, sélectionnez `GET`.
   b. Pour Type d'intégration, sélectionnez Lien VPC.
   c. Activez Intégration de proxy VPC.
   d. Pour Méthode HTTP, sélectionnez `GET`.
   e. Pour Lien VPC, sélectionnez [Utilisation de variables d'étape] et saisissez **`${stageVariables.vpcLinkId}`** dans la zone de texte ci-dessous.

      Vous pouvez définir la variable d'étape `vpcLinkId` après le déploiement de l'API à une étape et définir sa valeur sur l'ID du `VpcLink`.
   f. Pour URL de point de terminaison, entrez une URL, par exemple `http://myApi.example.com`.

      Ici, le nom d'hôte (par exemple, `myApi.example.com`) est utilisé pour définir l'en-tête `Host` de la demande d'intégration.
   g. Choisissez Créer une méthode.

      Avec l'intégration de proxy, l'API est prête pour le déploiement. Sinon, vous devez continuer à configurer les réponses de méthode et les réponses d'intégration appropriées.
6. Choisissez Déployer une API, puis procédez comme suit :

   a. Pour Étape, sélectionnez Nouvelle étape.
   b. Pour Nom de l'étape, entrez un nom d'étape.
   c. (Facultatif) Sous Description, entrez une description.
   d. Choisissez Deploy (Déployer).
7. Dans la section Détails de l'étape, notez l'URL d'invocation qui en résulte. Vous en aurez besoin pour appeler l'API. Avant cela, vous devez configurer la variable d'étape `vpcLinkId`.

8. Dans le volet Étapes, choisissez l'onglet Variables d'étape, puis procédez comme suit :

   a. Choisissez Gérer les variables, puis choisissez Ajouter une variable d'étape.

   b. Pour Name (Nom), saisissez **`vpcLinkId`**.

   c. Pour Valeur, entrez l'ID de `VPC_LINK`, par exemple, *`gix6s7`*.

   d. Choisissez Enregistrer.

      À l'aide de la variable d'étape, vous pouvez facilement passer à différents liens VPC pour l'API en modifiant la valeur de la variable d'étape.

Tutoriel : Création d'une API REST API Gateway avec AWS intégration

Les deux rubriques Tutoriel : Création d'une API REST Hello World avec l'intégration de proxy Lambda et Choisissez un didacticiel AWS Lambda d'intégration décrivent comment créer une API Gateway pour exposer la fonction intégrée Lambda. En outre, vous pouvez créer une API API Gateway pour exposer d'autres AWS services, tels qu'Amazon SNS, Amazon S3, Amazon Kinesis, etc. AWS Lambda Ceci est rendu possible par l'intégration `AWS`. L'intégration Lambda ou l'intégration proxy Lambda est un cas particulier, où l'invocation de la fonction Lambda est exposée via l'API Gateway.

Tous les AWS services prennent en charge des API dédiées pour exposer leurs fonctionnalités. Cependant, les protocoles d'application ou les interfaces de programmation sont susceptibles de différer d'un service à un autre. Une API API Gateway `AWS` intégrée présente l'avantage de fournir un protocole d'application cohérent permettant à votre client d'accéder à différents AWS services.

Dans cette procédure, nous avons créé une API pour exposer Amazon SNS. Pour d'autres exemples d'intégration d'une API à d'autres AWS services, consultezDidacticiels et ateliers Amazon API Gateway.

Contrairement à l'intégration de proxy Lambda, il n'y a pas d'intégration de proxy correspondante pour d'autres services AWS . Ainsi, une méthode d'API est intégrée à une seule AWS action. Pour plus de flexibilité, vous pouvez configurer une intégration de proxy Lambda, similaire à l'intégration de proxy. La fonction Lambda analyse et traite ensuite les demandes d'autres actions. AWS

API Gateway ne fait pas une nouvelle tentative lorsque le délai est dépassé pour le point de terminaison. L'appelant de l'API doit implémenter une logique de nouvelle tentative pour gérer les dépassements de délai du point de terminaison.

Cette procédure repose sur les instructions et les concepts décrits dans Choisissez un didacticiel AWS Lambda d'intégration. Si vous n'avez pas encore effectué cette procédure, nous vous conseillons de le faire en premier.

Rubriques

Prérequis

Avant de commencer cette procédure, exécutez les étapes suivantes :

1. Suivez les étapes de Prérequis pour démarrer avec API Gateway.
2. Créez une nouvelle API nommée `MyDemoAPI`. Pour de plus amples informations, veuillez consulter Tutoriel : Création d'une API REST avec une intégration HTTP autre que de proxy.
3. Déployez l'API au moins une fois jusqu'à une étape nommée `test`. Pour plus d'informations, consultez Déployer l'API dans Choisissez un didacticiel AWS Lambda d'intégration.
4. Suivez le reste des étapes dans Choisissez un didacticiel AWS Lambda d'intégration.
5. Créez au moins une rubrique dans Amazon Simple Notification Service (Amazon SNS). Vous utiliserez l'API déployée pour obtenir une liste des sujets associés à votre AWS compte sur Amazon SNS. Pour apprendre à créer une rubrique dans Amazon SNS, consultez Création d'une rubrique. (Vous n'avez pas besoin de copier l'ARN de la rubrique mentionné à l'étape 5.)

Étape 1 : créer le rôle d'exécution du proxy de AWS service

Pour autoriser l’API à invoquer les actions Amazon SNS requises, vous devez avoir attaché les politiques IAM appropriées à un rôle IAM.

Pour créer le rôle d'exécution du proxy de AWS service

1. Connectez-vous à la console IAM AWS Management Console et ouvrez-la à l'adresse https://console.aws.amazon.com/iam/.
2. Sélectionnez Roles.
3. Sélectionnez Créer un rôle.
4. Choisissez le AWS service sous Sélectionner le type d'entité de confiance, puis sélectionnez API Gateway et sélectionnez Allows API Gateway to push CloudWatch logs vers Logs.
5. Choisissez Suivant, puis Suivant.
6. Pour Nom du rôle, saisissez **`APIGatewaySNSProxyPolicy`**, puis choisissez Créer un rôle.
7. Dans la liste Roles, choisissez le rôle que vous venez de créer. Vous devrez peut-être faire défiler la page ou utiliser la barre de recherche pour rechercher le rôle.
8. Pour le rôle sélectionné, sélectionnez l'onglet Ajouter des autorisations.
9. Choisissez Attacher des politiques dans la liste déroulante.
10. Dans la barre de recherche, saisissez **`AmazonSNSReadOnlyAccess`** et choisissez Ajouter des autorisations.

    > **Note**
    >
    > Ce didacticiel utilise une stratégie gérée pour plus de simplicité. Il est recommandé de créer votre propre stratégie IAM afin d'accorder les autorisations minimales requises.

11. Notez l'ARN du rôle nouvellement créé. Vous l'utiliserez ultérieurement.

Étape 2 : Créer la ressource

Au cours de cette étape, vous créez une ressource qui permet au proxy de AWS service d'interagir avec le AWS service.

Pour créer la ressource

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Choisissez votre API.
3. Sélectionnez la racine de la ressource, /, représentée par une seule barre oblique (/), puis choisissez Créer une ressource.

4. Maintenez Ressource proxy désactivée.
5. Conservez Chemin de la ressource sous la forme `/`.
6. Sous Resource Name (Nom de la ressource), entrez **`mydemoawsproxy`**.
7. Maintenez CORS (Partage des ressources entre origines multiples) désactivé.
8. Choisissez Créer une ressource.

Étape 3 : Créer la méthode GET

Au cours de cette étape, vous créez une méthode GET qui permet au proxy de AWS service d'interagir avec le AWS service.

Pour créer la méthode **`GET`**

1. Sélectionnez la ressource /mydemoawsproxy, puis choisissez Créer une méthode.
2. Pour le type de méthode, sélectionnez GET.
3. Pour Type d’intégration, sélectionnez Service AWS.
4. Pour Région AWS, sélectionnez l' Région AWS endroit où vous avez créé votre rubrique Amazon SNS.
5. Pour Service AWS, sélectionnez Amazon SNS.
6. Laissez Sous-domaine AWS vide.
7. Pour Méthode HTTP, sélectionnez GET.
8. Pour Type d’action, sélectionnez Utiliser un nom d’action.
9. Pour Nom de l’action, saisissez **`ListTopics`**.
10. Pour Rôle d’exécution, saisissez l’ARN de rôle pour **`APIGatewaySNSProxyPolicy`**.
11. Choisissez Créer une méthode.

Étape 4 : Spécification des paramètres et test de la méthode

Vous pouvez maintenant tester votre méthode GET pour vérifier qu’elle a été correctement configurée pour répertorier vos rubriques Amazon SNS.

Pour tester la méthode **`GET`**

1. Choisissez l’onglet Test. Vous devrez peut-être choisir la flèche droite pour afficher l'onglet.

2. Sélectionnez Tester).

   Le résultat affiche une réponse semblable à ce qui suit :

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

Étape 5 : Déploiement de l'API

Dans cette étape, vous allez déployer l'API afin de pouvoir l'appeler en dehors de la console API Gateway.

Pour déployer l’API

1. Sélectionnez Deploy API (Déployer une API).
2. Pour Étape, sélectionnez Nouvelle étape.
3. Sous Stage name (Nom de l'étape), entrez **`test`**.
4. (Facultatif) Sous Description, entrez une description.
5. Choisissez Deploy (Déployer).

Étape 6 : Test de l'API

Au cours de cette étape, vous sortez de la console API Gateway et vous utilisez votre proxy de AWS service pour interagir avec le service Amazon SNS.

1. Dans le volet de navigation principal, choisissez Étape.
2. Sous Détails de l'étape, choisissez l'icône de copie pour copier l'URL d'invocation de votre API.

   Elle doit ressembler à ce qui suit :

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

3. Collez l'URL dans la zone d'adresse d'un nouvel onglet du navigateur.
4. Ajoutez `/mydemoawsproxy` afin que l'URL ressemble à ce qui suit :

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

   Accédez à l'URL. Des informations similaires à celles qui suivent doivent s'afficher :

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

Étape 7 : nettoyer

Vous pouvez supprimer les ressources IAM dont le proxy de AWS service a besoin pour fonctionner.

> **Warning**
>
> Si vous supprimez une ressource IAM sur laquelle repose un proxy de AWS service, ce proxy de AWS service et les API qui en dépendent ne fonctionneront plus. La suppression d'une ressource IAM ne peut pas être annulée. Si vous souhaitez réutiliser cette ressource IAM, vous devez la recréer.

Pour supprimer les ressources IAM associées

1. Ouvrez la console IAM à l'adresse https://console.aws.amazon.com/iam/.
2. Dans la zone Details, sélectionnez Roles.
3. Sélectionnez APIGateway AWSProxyExecRole, puis choisissez Actions de rôle, Supprimer le rôle. A l'invite, sélectionnez Yes, Delete.
4. Dans la zone Details, sélectionnez Policies.
5. Sélectionnez APIGateway AWSProxyExecPolicy, puis choisissez Policy Actions, Supprimer. A l'invite, sélectionnez Delete (Supprimer).

Vous avez terminé cette procédure. Pour des discussions plus détaillées sur la création d'une API en tant que proxy de AWS serviceTutoriel : Création d'une API REST en tant que proxy Amazon S3 dans API Gateway, consultezTutoriel : Création d'une API REST de calculatrice avec deux intégrations AWS de services et une intégration Lambda sans proxy, ouTutoriel : Création d'une API REST en tant que proxy Amazon Kinesis dans API Gateway.

Tutoriel : Création d'une API REST de calculatrice avec deux intégrations AWS de services et une intégration Lambda sans proxy

Le didacticiel Démarrez avec les intégrations non-proxy utilise l'intégration `Lambda Function` exclusivement. L'intégration `Lambda Function` est un cas particulier du type d'intégration `AWS Service` qui effectue une grande partie de la configuration de l'intégration pour vous, comme l'ajout automatique des autorisations basées sur les ressources requises pour appeler la fonction Lambda. Ici, deux des trois intégrations utilisent l'intégration `AWS Service`. Ce type d'intégration offre davantage de contrôle, mais vous devez exécuter manuellement les tâches telles que la création et la spécification d'un rôle IAM contenant les autorisations appropriées.

Dans ce tutoriel, vous allez créer une fonction Lambda `Calc` pour implémenter les opérations arithmétiques de base, en acceptant et en renvoyant des données d'entrée et de sortie au format JSON. Ensuite, vous allez créer une API REST et l'intégrer à la fonction Lambda de différentes manières :

1. En exposant une méthode `GET` sur la ressource `/calc` pour appeler la fonction Lambda, en fournissant l'entrée en tant que paramètres de chaîne de requête. (Intégration `AWS Service`)
2. En exposant une méthode `POST` sur la ressource `/calc` pour appeler la fonction Lambda, en fournissant l'entrée dans la charge utile de la demande de méthode. (Intégration `AWS Service`)

3. En exposant une méthode `GET` sur les ressources `/calc/{operand1}/{operand2}/{operator}` imbriquées pour appeler la fonction Lambda, en fournissant les données d'entrée en tant que paramètres de chemin. (Intégration `Lambda Function`)

Outre ce tutoriel, vous pouvez vous reporter au fichier de définitions OpenAPI pour l'API `Calc`, que vous pouvez importer dans API Gateway en suivant les instructions de the section called “OpenAPI”.

Rubriques

- Création d'un rôle IAM assumable
- Création d'une fonction Lambda Calc
- Test de la fonction Lambda Calc
- Création d'une API Calc
- Intégration 1 : Création d'une méthode GET avec des paramètres de requête pour appeler la fonction Lambda
- Intégration 2 : Création d'une méthode POST avec une charge utile JSON pour appeler la fonction Lambda
- Intégration 3 : Création d'une méthode GET avec des paramètres de chemin pour appeler la fonction Lambda
- Définitions OpenAPI de l'exemple d'API intégré à une fonction Lambda

Création d'un rôle IAM assumable

Pour que votre API invoque votre fonction Lambda `Calc`, vous devez disposer d'un rôle IAM assumable API Gateway, qui est un rôle IAM ayant la relation de confiance suivante :

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

```
}
```

Le rôle que vous créez doit disposer de l'autorisation Lambda InvokeFunction. Dans le cas contraire, l'appelant d'API recevra une réponse `500 Internal Server Error`. Pour accorder cette autorisation au rôle, vous devez lui attacher la stratégie IAM suivante :

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

Voici comment procéder :

Création d'un rôle IAM assumable API Gateway

1. Connectez-vous à la console IAM.
2. Sélectionnez Roles.
3. Choisissez Create Role (Créer le rôle).
4. Sous Sélectionner un type d’entité de confiance, choisissez AWS Service.
5. Sous Choose the service that will use this role (Choisir le service qui utilisera ce rôle), choisissez Lambda.
6. Sélectionnez Next: Permissions (Suivant : autorisations).
7. Choisissez Create Policy (Créer une politique).

   Une nouvelle fenêtre de console Create Policy (Créer une politique) s'ouvre. Dans cette fenêtre, procédez de la façon suivante :

   a. Dans l'onglet JSON, remplacez la politique existante par la suivante :

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

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

b. Choisissez Examiner une politique.

c. Sous Review Policy (Examiner une politique), procédez comme suit :

i. Pour Nom, tapez un nom, par exemple **`lambda_execute`**.

ii. Choisissez Créer une politique.

8. Dans la fenêtre de console d'origine Create Role (Créer un rôle), procédez comme suit :

a. Sous Attach permissions policies (Attacher des politiques d'autorisations), choisissez votre politique **`lambda_execute`** dans la liste déroulante.

Si votre politique n'est pas répertoriée dans la liste, cliquez sur le bouton d'actualisation en haut de la liste. (N'actualisez pas la page du navigateur.)

b. Choisissez Next:Tags (Suivant : balises).

c. Choisissez Next: Review (Suivant : vérifier).

d. Sous Role name (Nom du rôle), saisissez un nom tel que **`lambda_invoke_function_assume_apigw_role`**.

e. Sélectionnez Créer un rôle.

9. Choisissez votre rôle **`lambda_invoke_function_assume_apigw_role`** dans la liste.

10. Choisissez l'onglet Trust relationships.

11. Choisissez Modifier la relation d'approbation.

12. Remplacez la stratégie existante comme suit :

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

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

13. Choisissez Update Trust Policy.
14. Notez l'ARN du rôle que vous venez de créer. Vous en aurez besoin ultérieurement.

Création d'une fonction Lambda **`Calc`**

Vous créerez ensuite une fonction Lambda à l'aide de la console Lambda.

1. Dans la console Lambda, choisissez Create function (Créer une fonction).
2. Choisissez Créer à partir de zéro.
3. Pour Nom, saisissez **`Calc`**.
4. Pour Exécution, choisissez le dernier environnement d’exécution Node.js ou Python compatible.
5. Choisissez Créer une fonction.
6. Copiez la fonction Lambda suivante dans votre environnement d’exécution préféré et collez-la dans l’éditeur de code de la console Lambda.

   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. Sous Execution role (Rôle d'exécution), choisissez Choose an existing role (Choisir un rôle existant).
8. Entrez l'ARN du rôle **`lambda_invoke_function_assume_apigw_role`** que vous avez créé précédemment.
9. Choisissez Deploy (Déployer).

Cette fonction nécessite deux opérandes (a et b) et un opérateur (op) à partir du paramètre d'entrée `event`. L'entrée est un objet JSON au format suivant :

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

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

Cette fonction renvoie le résultat calculé (`c`) et les données d'entrée. En cas d'entrée non valide, la fonction renvoie la valeur null ou la chaîne « Opération non valide » en tant que résultat. La sortie est au format JSON suivant :

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

Vous devez tester la fonction dans la console Lambda avant de l'intégrer à l'API dans l'étape suivante.

Test de la fonction Lambda **`Calc`**

Voici comment tester la fonction `Calc` dans la console Lambda :

1. Choisissez l'onglet Test.
2. Entrez comme nom de l'événement de tes **`calc2plus5`**.
3. Remplacez la définition de l'événement de test par ce qui suit :

   ```
   {
     "a": "2",
     "b": "5",
     "op": "+"
   }
   ```
4. Choisissez Save (Enregistrer).
5. Sélectionnez Test (Tester).
6. Développez Résultat de l'exécution : réussite. Vous devez voir ce qui suit :

   ```
   {
   ```

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

Création d'une API **`Calc`**

La procédure suivante explique comment créer une API pour la fonction Lambda `Calc` que vous venez de créer. Dans les sections suivantes, vous allez y ajouter des ressources et des méthodes.

Pour créer une API

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Si vous utilisez API Gateway pour la première fois, vous voyez une page qui vous présente les fonctions du service. Sous REST API (API REST), choisissez Build (Création). Lorsque la fenêtre contextuelle Create Example API (Créer API exemple) s'affiche, cliquez sur OK.

   Si ce n'est pas la première fois que vous utilisez API Gateway, choisissez Create API (Créer une API). Sous REST API (API REST), choisissez Build (Création).
3. Sous API name (Nom de l'API), saisissez **`LambdaCalc`**.
4. (Facultatif) Sous Description, entrez une description.
5. Laissez Type de point de terminaison d'API défini sur Régional.
6. Sélectionnez Create API (Créer une API).

Intégration 1 : Création d'une méthode **`GET`** avec des paramètres de requête pour appeler la fonction Lambda

Lorsque vous créez une méthode `GET` qui transmet les paramètres de chaîne de requête à la fonction Lambda, vous permettez à l'API d'être appelée à partir d'un navigateur. Cette approche peut s'avérer utile, en particulier pour les API qui autorisent un accès ouvert.

Une fois que vous avez créé une API, vous créez une ressource. En règle générale, les ressources API sont organisées dans une arborescence des ressources selon la logique de l'application. Pour cette étape, vous créez une ressource /calc.

Pour créer une ressource /calc

1. Choisissez Créer une ressource.
2. Maintenez Ressource proxy désactivée.
3. Conservez Chemin de la ressource sous la forme `/`.
4. Sous Resource Name (Nom de la ressource), entrez **`calc`**.
5. Maintenez CORS (Partage des ressources entre origines multiples) désactivé.
6. Choisissez Créer une ressource.

Lorsque vous créez une méthode `GET` qui transmet les paramètres de chaîne de requête à la fonction Lambda, vous permettez à l'API d'être appelée à partir d'un navigateur. Cette approche peut s'avérer utile, en particulier pour les API qui autorisent un accès ouvert.

Dans cette méthode, Lambda exige que la requête `POST` soit utilisée pour invoquer une fonction Lambda. Cet exemple montre que la méthode HTTP dans une demande de méthode de serveur frontal peut être différente de la demande d'intégration sur le backend.

Pour créer une méthode **`GET`**

1. Sélectionnez la ressource /calc, puis choisissez Créer une méthode.
2. Pour Type de méthode, sélectionnez GET.
3. Pour Type d'intégration, sélectionnez Service AWS.
4. Pour Région AWS, sélectionnez l' Région AWS endroit où vous avez créé votre fonction Lambda.
5. Pour Service AWS, sélectionnez Lambda.
6. Laissez Sous-domaine AWS vide.
7. Pour Méthode HTTP, sélectionnez POST.
8. Pour Type d'action, sélectionnez Utiliser un remplacement de chemin. Cette option nous permet de spécifier l'ARN de l'action Invoke (Appel) pour exécuter notre fonction `Calc`.
9. Pour Remplacement de chemin, saisissez **`2015-03-31/functions/arn:aws:lambda:`*`us-east-2`*`:`*`account-id`*`:function:Calc/invocations`**. Pour *`account-id`*, entrez votre Compte AWS identifiant. Pour*`us-east-2`*, entrez l' Région AWS endroit où vous avez créé votre fonction Lambda.
10. Pour Rôle d'exécution, saisissez l'ARN de rôle pour **`lambda_invoke_function_assume_apigw_role`**.

11. Ne modifiez pas les paramètres de Cache d'informations d'identification et de Délai d'attente par défaut.
12. Choisissez les paramètres de demande de méthode.
13. Pour Validateur de requête, sélectionnez Valider les paramètres de chaîne de requête et les en-têtes.

    Ce paramètre renvoie un message d'erreur si le client ne spécifie pas les paramètres requis.
14. Choisissez Paramètres de chaîne de requête d'URL.

    Vous configurez maintenant les paramètres de chaîne de requête pour la méthode GET sur la ressource /calc afin qu'elle puisse recevoir des entrées au nom de la fonction Lambda principale.

    Pour créer les paramètres de chaîne de requête, procédez comme suit :

    a. Sélectionnez Add query string (Ajouter une chaîne de requêtes).
    b. Pour Name (Nom), saisissez **`operand1`**.
    c. Activez Obligatoire.
    d. Maintenez Mise en cache désactivée.

    Répétez les mêmes étapes et créez une chaîne de requête nommée **`operand2`** et une chaîne de requête nommée **`operator`**.
15. Choisissez Créer une méthode.

Maintenant, vous créez un modèle de mappage afin de convertir les chaînes de requête fournies par le client en la charge utile de requête d'intégration, comme requis par la fonction `Calc`. Ce modèle mappe les trois paramètres de chaîne de requête déclarés dans Requête de méthode aux valeurs de propriété désignées de l'objet JSON en tant que données d'entrée de la fonction Lambda du backend. L'objet JSON transformé sera inclus en tant que charge utile de la demande d'intégration.

Pour mapper les paramètres d'entrée à la requête d'intégration

1. Dans l'onglet Requête d'intégration, sous Paramètres de requête d'intégration, choisissez Modifier.
2. Pour Transmission du corps de requête, sélectionnez Lorsqu'aucun modèle n'est défini (recommandé).
3. Choisissez Modèles de mappage.

4. Sélectionnez Add mapping template.
5. Pour Type de contenu, entrez **`application/json`**.
6. Pour Corps du modèle, entrez le code suivant :

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

7. Choisissez Enregistrer.

Vous pouvez maintenant tester votre méthode GET pour vérifier qu'elle a été correctement configurée pour invoquer la fonction Lambda.

Pour tester la méthode **`GET`**

1. Choisissez l'onglet Test. Vous devrez peut-être choisir la flèche droite pour afficher l'onglet.
2. Pour Chaînes de requête, saisissez **`operand1=2&operand2=3&operator=+`**.
3. Sélectionnez Test.

   Les résultats doivent ressembler à ce qui suit :

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

operand1=2&operand2=3&operator=+

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

/?
operand1=2&operand2=3&operator=+

Status

200

Latency

414

Response body

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

Intégration 2 : Création d'une méthode **POST** avec une charge utile JSON pour appeler la fonction Lambda

En créant une méthode POST avec une charge utile JSON pour appeler la fonction Lambda, vous faites en sorte que le client doive fournir les données d'entrée nécessaires à la fonction du backend dans le corps de la demande. Pour vous assurer que le client charge les données d'entrée correctes, vous allez activer la validation de demande sur la charge utile.

Pour créer une méthode **POST** avec une charge utile JSON

1. Sélectionnez la ressource /calc, puis choisissez Créer une méthode.
2. Pour Type de méthode, sélectionnez POST.
3. Pour Type d'intégration, sélectionnez Service AWS.
4. Pour Région AWS, sélectionnez l' Région AWS endroit où vous avez créé votre fonction Lambda.
5. Pour Service AWS, sélectionnez Lambda.
6. Laissez Sous-domaine AWS vide.
7. Pour Méthode HTTP, sélectionnez POST.
8. Pour Type d'action, sélectionnez Utiliser un remplacement de chemin. Cette option nous permet de spécifier l'ARN de l'action Invoke (Appel) pour exécuter notre fonction `Calc`.
9. Pour Remplacement de chemin, saisissez **`2015-03-31/functions/arn:aws:lambda:`*`us-east-2`*`:`*`account-id`*`:function:Calc/invocations`**. Pour *`account-id`*, entrez votre Compte AWS identifiant. Pour*`us-east-2`*, entrez l' Région AWS endroit où vous avez créé votre fonction Lambda.
10. Pour Rôle d'exécution, saisissez l'ARN de rôle pour **`lambda_invoke_function_assume_apigw_role`**.
11. Ne modifiez pas les paramètres de Cache d'informations d'identification et de Délai d'attente par défaut.
12. Choisissez Créer une méthode.

Maintenant, vous créez un modèle d'entrée pour décrire la structure des données d'entrée et valider le corps de la requête entrante.

Pour créer un modèle d'entrée

1. Dans le volet de navigation principal, choisissez Modèles.
2. Sélectionnez Create model.
3. Pour Name (Nom), saisissez **`input`**.
4. Pour Type de contenu, entrez **`application/json`**.

   Si aucun type de contenu correspondant n'est trouvé, la validation de demande n'est pas effectuée. Pour utiliser le même modèle quel que soit le type de contenu, saisissez **`$default`**.
5. Pour Schéma du modèle, saisissez le modèle qui suit :

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

6. Sélectionnez Create model.

Vous créez maintenant un modèle de sortie. Ce modèle décrit la structure de données de la sortie calculée à partir du serveur principal. Elle peut être utilisée pour mapper les données de réponse d'intégration à un autre modèle. Ce didacticiel s'appuie sur le comportement de transmission et n'utilise pas ce modèle.

Pour créer un modèle de sortie

1. Sélectionnez Create model.
2. Pour Name (Nom), saisissez **`output`**.
3. Pour Type de contenu, entrez **`application/json`**.

   Si aucun type de contenu correspondant n'est trouvé, la validation de demande n'est pas effectuée. Pour utiliser le même modèle quel que soit le type de contenu, saisissez **`$default`**.
4. Pour Schéma du modèle, saisissez le modèle qui suit :

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

5. Sélectionnez Create model.

Vous créez maintenant un modèle de résultat. Ce modèle décrit la structure de données des données de réponse renvoyées. Il fait référence aux schémas d'entrée et de sortie définis dans votre API.

Pour créer un modèle de résultat

1. Sélectionnez Create model.
2. Pour Name (Nom), saisissez **`result`**.
3. Pour Type de contenu, entrez **`application/json`**.

   Si aucun type de contenu correspondant n'est trouvé, la validation de demande n'est pas effectuée. Pour utiliser le même modèle quel que soit le type de contenu, saisissez **`$default`**.
4. Pour Schéma du modèle, entrez le modèle suivant avec votre *`restapi-id`*. Votre *`restapi-id`* est indiqué entre parenthèses en haut de la console dans le flux suivant : `API Gateway > APIs > LambdaCalc (`*`abc123`*`).`

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

5. Sélectionnez Create model.

Vous configurez maintenant la requête de méthode de votre méthode POST pour activer la validation de requête sur le corps de requête entrante.

Pour activer la validation des demandes sur la méthode POST

1. Dans le volet de navigation principal, choisissez Ressources, puis sélectionnez la méthode POST dans l'arborescence de ressources.
2. Dans l'onglet Demande de méthode, sous Paramètres de demande de méthode, choisissez Modifier.
3. Pour Validateur de requête, sélectionnez Valider le corps.
4. Choisissez Corps de la requête, puis choisissez Ajouter un modèle.
5. Pour Type de contenu, entrez **`application/json`**.

   Si aucun type de contenu correspondant n'est trouvé, la validation de demande n'est pas effectuée. Pour utiliser le même modèle quel que soit le type de contenu, saisissez **`$default`**.
6. Pour Modèle, sélectionnez entrée.
7. Choisissez Enregistrer.

Vous pouvez maintenant tester votre méthode POST pour vérifier qu'elle a été correctement configurée pour invoquer la fonction Lambda.

Pour tester la méthode **`POST`**

1. Choisissez l'onglet Test. Vous devrez peut-être choisir la flèche droite pour afficher l'onglet.
2. Pour Corps de la requête, entrez la charge utile de JSON suivante.

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

3. Sélectionnez Tester).

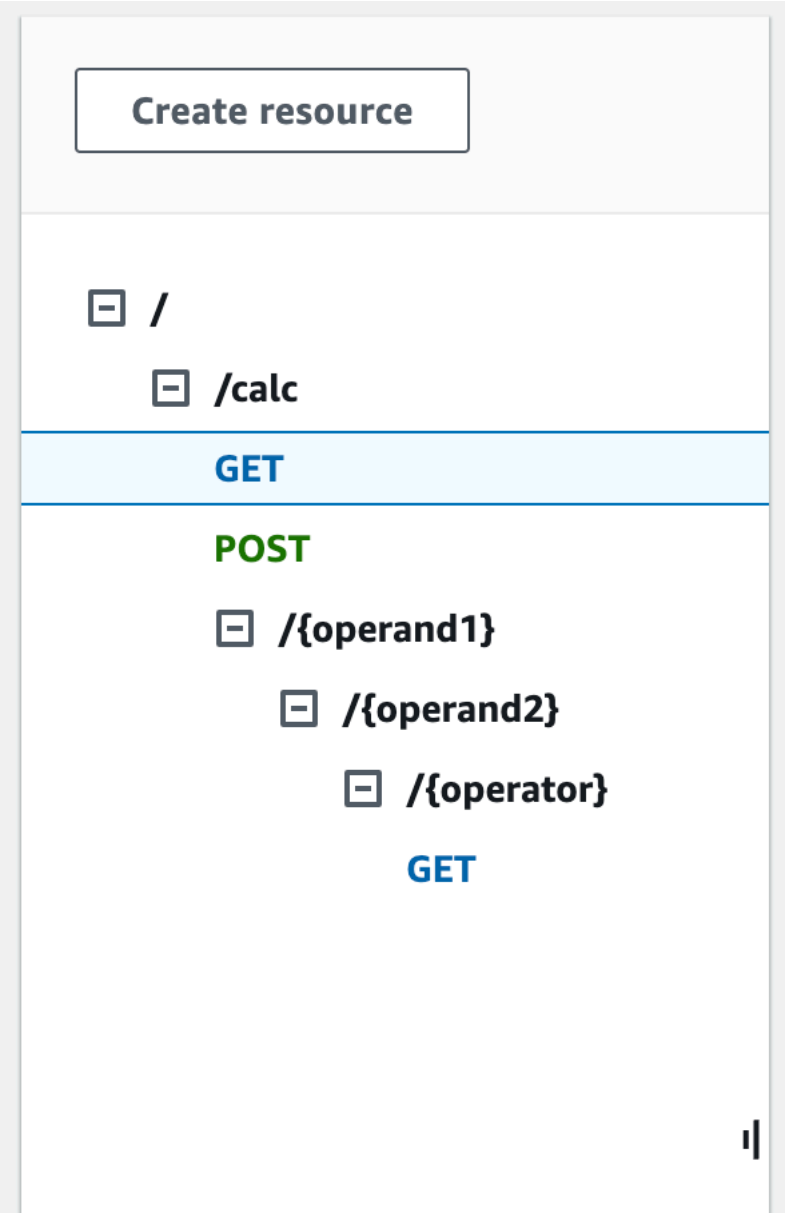
Vous devriez voir la sortie suivante :

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

Intégration 3 : Création d'une méthode **GET** avec des paramètres de chemin pour appeler la fonction Lambda

Vous allez maintenant créer une méthode GET sur une ressource spécifiée par une séquence de paramètres de chemin pour appeler la fonction Lambda du backend. Les valeurs de paramètre de chemin spécifient les données d'entrée de la fonction Lambda. Vous allez utiliser un modèle de mappage afin de mapper les valeurs de paramètres de chemin entrantes à la charge utile de demande d'intégration requise.

La structure de ressource d'API qui en résulte ressemble à ce qui suit :



Pour créer une ressource /{operand1}/{operand2}/{operator}

1. Choisissez Créer une ressource.
2. Pour Chemin de ressource, sélectionnez `/calc`.
3. Sous Resource Name (Nom de la ressource), entrez **`{operand1}`**.
4. Maintenez CORS (Partage des ressources entre origines multiples) désactivé.
5. Choisissez Créer une ressource.
6. Pour Chemin de ressource, sélectionnez `/calc/{operand1}/`.
7. Sous Resource Name (Nom de la ressource), entrez **`{operand2}`**.

8. Maintenez CORS (Partage des ressources entre origines multiples) désactivé.
9. Choisissez Créer une ressource.
10. Pour Chemin de ressource, sélectionnez `/calc/{operand1}/{operand2}/`.
11. Sous Resource Name (Nom de la ressource), entrez **`{operator}`**.
12. Maintenez CORS (Partage des ressources entre origines multiples) désactivé.
13. Choisissez Créer une ressource.

Cette fois, vous allez utiliser l'intégration Lambda intégrée dans la console API Gateway afin de configurer l'intégration de la méthode.

Pour configurer une intégration de méthode

1. Sélectionnez la ressource/{operand1}/{operand2}/{operator}, puis choisissez Créer une méthode.
2. Pour Type de méthode, sélectionnez GET.
3. Pour Type d'intégration, sélectionnez Lambda
4. Maintenez Intégration proxy Lambda désactivée.
5. Pour la fonction Lambda, sélectionnez l' Région AWS endroit où vous avez créé votre fonction Lambda et entrez. **`Calc`**
6. Maintenez Délai d'expiration par défaut activé.
7. Choisissez Créer une méthode.

Vous créez maintenant un modèle de mappage pour mapper les trois paramètres de chemin d'URL, déclarés lors de la création de la ressource /calc/{operand1}/{operand2}/{operator}, aux valeurs de propriété désignées de l'objet JSON. Etant donné que les chemins d'URL doivent être codés en URL, l'opérateur de division doit être spécifié sous la forme `%2F` au lieu de `/`. Ce modèle convertit `%2F` en `'/'` avant de le transmettre à la fonction Lambda.

Pour créer un modèle de mappage

1. Dans l'onglet Requête d'intégration, sous Paramètres de requête d'intégration, choisissez Modifier.
2. Pour Transmission du corps de requête, sélectionnez Lorsqu'aucun modèle n'est défini (recommandé).
3. Choisissez Modèles de mappage.

4. Pour Type de contenu, entrez **`application/json`**.
5. Pour Corps du modèle, entrez le code suivant :

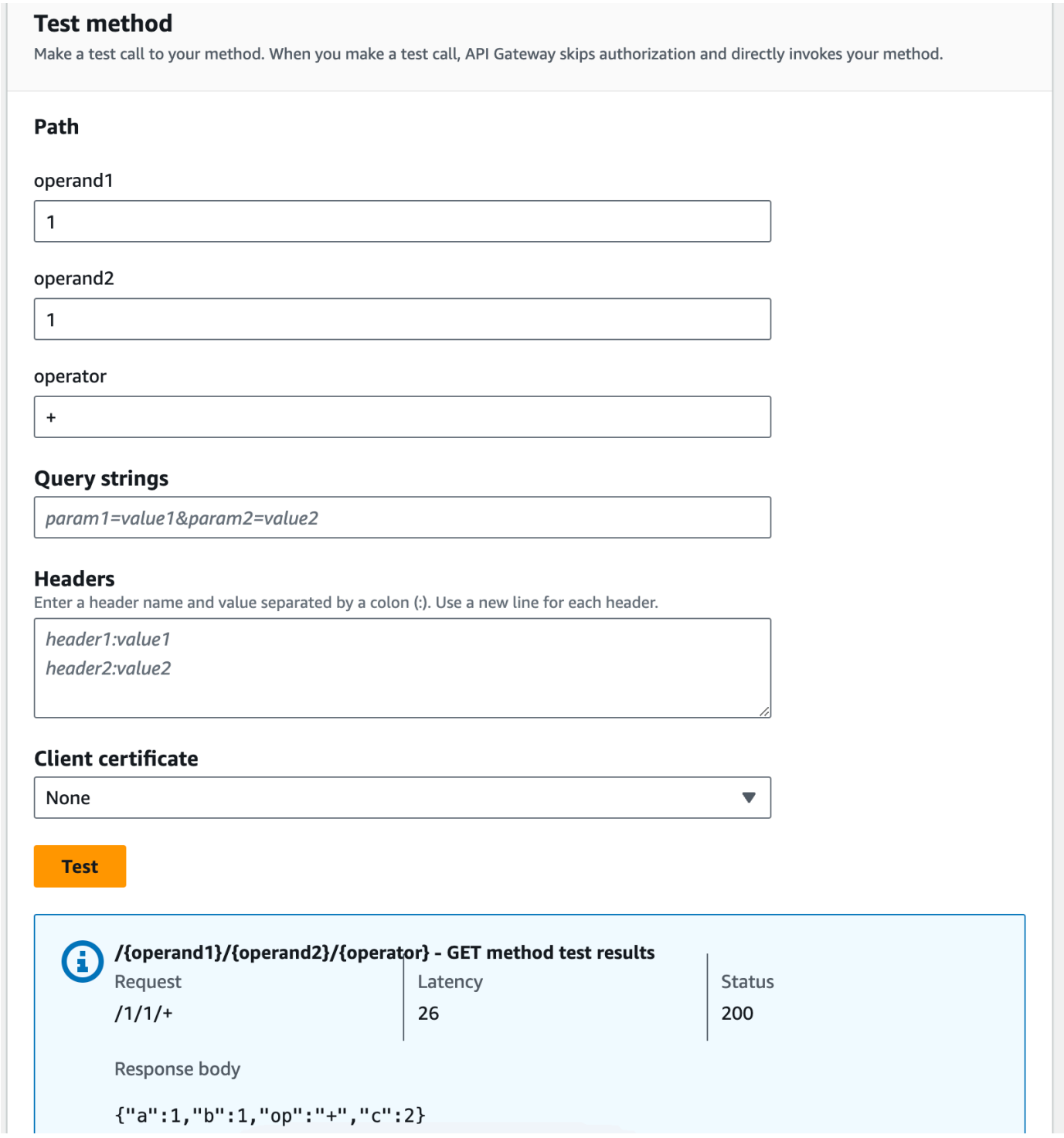
```
{
   "a": "$input.params('operand1')",
   "b": "$input.params('operand2')",
   "op":
 #if($input.params('operator')=='%2F')"/"#{else}"$input.params('operator')"#end
}
```

6. Choisissez Enregistrer.

Vous pouvez maintenant tester votre méthode GET pour vérifier qu'elle a été correctement configurée pour invoquer la fonction Lambda et transmettre le résultat d'origine via la réponse d'intégration sans mappage.

Pour tester la méthode **`GET`**

1. Choisissez l'onglet Test. Vous devrez peut-être choisir la flèche droite pour afficher l'onglet.
2. Pour Chemin, procédez comme suit :
   a. Pour opérande 1, saisissez **1**.
   b. Pour opérande 2, saisissez **1**.
   c. Pour opérateur, saisissez **+**.
3. Sélectionnez Test.
4. Le résultat doit se présenter comme suit :



Ensuite, vous modélisez la structure de données de la charge utile de réponse de méthode après le schéma `result`.

Par défaut, un modèle vide est attribué au corps de réponse de méthode. Cela entraînera la transmission du corps de réponse d'intégration sans mappage. Cependant, lorsque vous générez un

kit SDK pour l'un des langages fortement typés, comme Java ou Objective-C, les utilisateurs de votre kit SDK recevront un objet vide comme résultat. Pour vous assurer que le client REST et les clients SDK reçoivent les résultats escomptés, vous devez modéliser les données de réponse à l'aide d'un schéma prédéfini. Ici, vous allez définir un modèle pour le corps de réponse de méthode et construire un modèle de mappage pour traduire le corps de réponse d'intégration en corps de réponse de méthode.

Pour créer une réponse de méthode

1. Dans l'onglet Méthode de réponse, sous Réponse 200, choisissez Modifier.
2. Sous Corps de la réponse, choisissez Ajouter un modèle.
3. Pour Type de contenu, entrez **`application/json`**.
4. Pour Modèle, sélectionnez Résultat.
5. Choisissez Enregistrer.

La définition du modèle pour le corps de réponse de méthode garantit que les données de réponse seront intégrées dans l'objet `result` d'un kit SDK donné. Pour vous assurer que les données de réponse d'intégration sont mappées en conséquence, un modèle de mappage est nécessaire.

Pour créer un modèle de mappage

1. Dans l'onglet Réponse d'intégration, sous Par défaut - Réponse, choisissez Modifier.
2. Choisissez Modèles de mappage.
3. Pour Type de contenu, entrez **`application/json`**.
4. Pour Corps du modèle, entrez le code suivant :

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

5. Choisissez Enregistrer.

Pour tester le modèle de mappage

1. Choisissez l’onglet Test. Vous devrez peut-être choisir la flèche droite pour afficher l'onglet.
2. Pour Chemin, procédez comme suit :
   a. Pour opérande 1, saisissez **1**.
   b. Pour opérande 2, saisissez **2**.
   c. Pour opérateur, saisissez **+**.
3. Sélectionnez Tester).
4. Le résultat ressemble à ce qui suit :

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

À ce stade, vous pouvez appeler l’API uniquement à l’aide de la fonctionnalité Tester dans la console API Gateway. Afin de la rendre disponible votre API pour les clients, vous devez la déployer. Veillez toujours à redéployer votre API chaque fois que vous ajoutez, modifiez ou supprimez une ressource ou une méthode, mettez à jour un mappage de données ou mettez à jour des paramètres d'étape. Sinon, les nouvelles fonctionnalités ou les mises à jour ne seront pas disponibles pour les clients de votre API, comme suit :

Pour déployer l’API

1. Sélectionnez Deploy API (Déployer une API).
2. Pour Étape, sélectionnez Nouvelle étape.
3. Sous Stage name (Nom de l'étape), entrez **`Prod`**.
4. (Facultatif) Sous Description, entrez une description.

5. Choisissez Deploy (Déployer).
6. (Facultatif) Sous Détails de l'étape, pour Invoquer une URL, vous pouvez choisir l'icône de copie pour copier l'URL d'invocation de votre API. Vous pouvez l'utiliser avec des outils tels que Postman et cURL pour tester votre API.

> **Note**
>
> Veillez toujours à redéployer votre API chaque fois que vous ajoutez, modifiez ou supprimez une ressource ou une méthode, mettez à jour un mappage de données ou mettez à jour des paramètres d'étape. Sinon, les nouvelles fonctions ou les mises à jour ne seront pas disponibles pour les clients de votre API.

Définitions OpenAPI de l'exemple d'API intégré à une fonction Lambda

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
    }
  }
}
```

Tutoriel : Création d'une API REST en tant que proxy Amazon S3 dans API Gateway

À titre d'exemple pour présenter l'utilisation d'une API REST dans API Gateway comme proxy Amazon S3, cette section explique comment créer et configurer une API REST pour exposer les opérations Amazon S3 suivantes :

- Exposition de la méthode GET sur la ressource racine de l'API pour afficher la liste de tous les compartiments Amazon S3 d'un appelant.
- Exposition de la méthode GET sur une ressource Folder pour afficher la liste de tous les objets d'un compartiment Amazon S3.
- Exposition de la méthode GET sur une ressource Folder/Item pour afficher ou télécharger un objet à partir d'un compartiment Amazon S3.

Vous pouvez importer l'exemple d'API en tant que proxy Amazon S3, comme illustré dans Définitions OpenAPI de l'exemple d'API en tant que proxy Amazon S3. Cet échantillon contient des méthodes plus exposées. Pour de plus amples informations sur l'importation d'une API à l'aide de la définition OpenAPI, veuillez consulter Configuration d'une API REST à l'aide d'OpenAPI.

> **Note**
>
> Pour intégrer votre API Amazon API Gateway à Amazon S3, vous devez choisir une région où les services API Gateway et Amazon S3 sont disponibles. Pour connaître la disponibilité dans les régions, veuillez consulter les points de terminaison et quotas Amazon API Gateway.

Rubriques

- Configuration d'autorisations IAM pour l'API afin d'appeler des actions Amazon S3
- Création de ressources d'API pour représenter des ressources Amazon S3
- Exposition d'une méthode d'API pour afficher la liste des compartiments Amazon S3 de l'appelant
- Exposition de méthodes d'API pour accéder à un compartiment Amazon S3
- Exposition de méthodes d'API pour accéder à un objet Amazon S3 dans un compartiment
- Définitions OpenAPI de l'exemple d'API en tant que proxy Amazon S3
- Appel de l'API à l'aide d'un client d'API REST

Configuration d'autorisations IAM pour l'API afin d'appeler des actions Amazon S3

Pour autoriser l'API à invoquer les actions Amazon S3 requises, vous devez avoir attaché les politiques IAM appropriées à un rôle IAM.

Pour créer le rôle d'exécution du proxy de AWS service

1. Connectez-vous à la console IAM AWS Management Console et ouvrez-la à l'adresse https://console.aws.amazon.com/iam/.
2. Sélectionnez Roles.
3. Sélectionnez Créer un rôle.
4. Choisissez le AWS service sous Sélectionner le type d'entité de confiance, puis sélectionnez API Gateway et sélectionnez Allows API Gateway to push CloudWatch logs vers Logs.
5. Choisissez Suivant, puis Suivant.
6. Pour Nom du rôle, saisissez **`APIGatewayS3ProxyPolicy`**, puis choisissez Créer un rôle.
7. Dans la liste Roles, choisissez le rôle que vous venez de créer. Vous devrez peut-être faire défiler la page ou utiliser la barre de recherche pour rechercher le rôle.

8. Pour le rôle sélectionné, sélectionnez l'onglet Ajouter des autorisations.
9. Choisissez Attacher des politiques dans la liste déroulante.
10. Dans la barre de recherche, saisissez **`AmazonS3FullAccess`** et choisissez Ajouter des autorisations.

> **Note**
>
> Ce didacticiel utilise une stratégie gérée pour plus de simplicité. Il est recommandé de créer votre propre stratégie IAM afin d'accorder les autorisations minimales requises.

11. Notez l'ARN du rôle nouvellement créé. Vous l'utiliserez ultérieurement.

Création de ressources d'API pour représenter des ressources Amazon S3

Vous utilisez la ressource root (`/`) de l'API comme conteneur des compartiments Amazon S3 d'un appelant authentifié. Vous créez également `Item` des ressources `Folder` et pour représenter respectivement un compartiment Amazon S3 et un objet Amazon S3 particuliers. Le nom du dossier et la clé de l'objet seront spécifiés sous la forme de paramètres de chemin dans le cadre d'une URL de demande, par l'appelant.

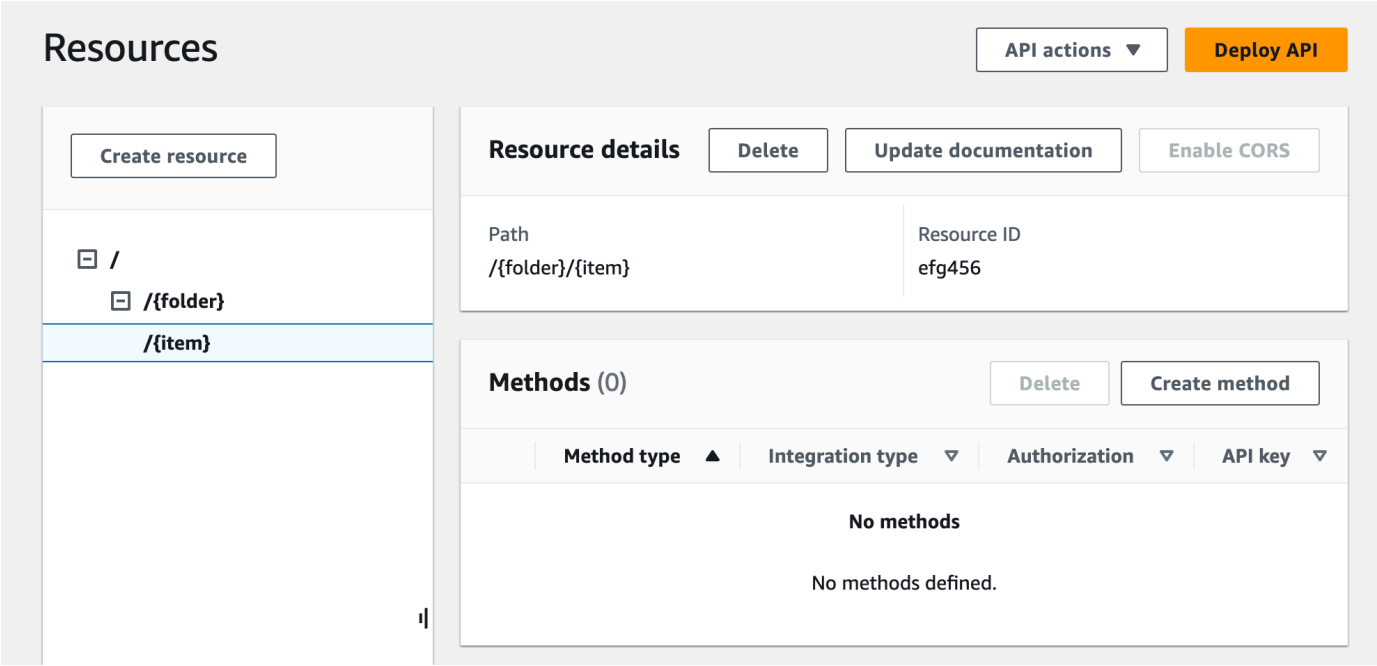
> **Note**
>
> Lorsque vous accédez à des objets dont la clé d'objet inclut `/` ou n'importe quel autre caractère spécial, ce caractère doit être encodé en URL. Par exemple, `test/test.txt` doit être codé en `test%2Ftest.txt`.

Pour créer une ressource d'API qui expose les fonctions de service Amazon S3

1. Dans le même temps Région AWS que vous avez créé votre compartiment Amazon S3, créez une API nommée MyS3. La ressource racine de cette API (/) représente le service Amazon S3. Dans cette étape, vous créez deux ressources supplémentaires /{folder} et /{item}.
2. Sélectionnez la ressource racine de l'API, puis choisissez Créer une ressource.
3. Maintenez Ressource proxy désactivée.
4. Pour Chemin de ressource, sélectionnez `/`.
5. Sous Resource Name (Nom de la ressource), entrez **`{folder}`**.

6. Gardez CORS (Partage des ressources entre origines multiples) non coché.
7. Choisissez Créer une ressource.
8. Sélectionnez la ressource /{folder}, puis choisissez Créer une ressource.
9. Suivez les étapes précédentes pour créer une ressource enfant de /{folder} nommée **`{item}`**.

   Votre API finale doit ressembler à ce qui suit :



Exposition d'une méthode d'API pour afficher la liste des compartiments Amazon S3 de l'appelant

L'obtention de la liste des compartiments Amazon S3 de l'appelant implique l'appel de l'action GET Service sur Amazon S3. Sur la ressource racine de l'API (/), créez la méthode GET. Configurez la méthode GET de façon à ce qu'elle s'intègre à Amazon S3, comme suit.

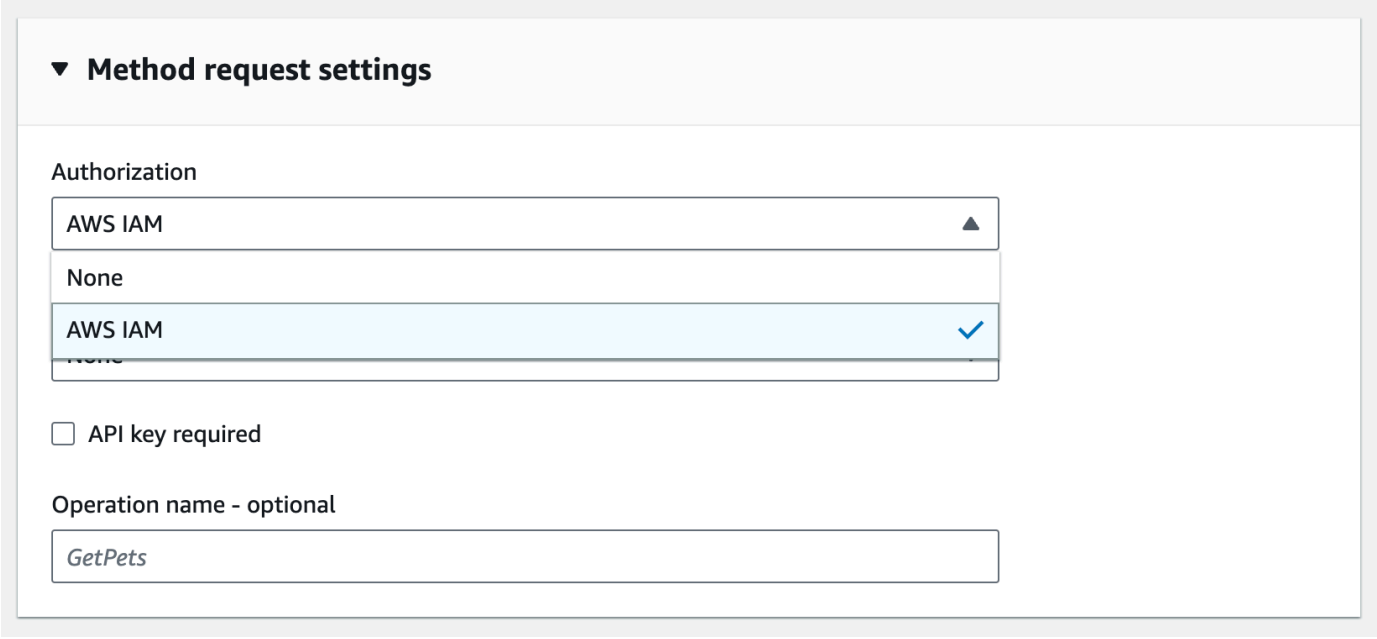
Pour créer et initialiser la méthode **`GET /`** de l'API

1. Sélectionnez la ressource /, puis choisissez Créer une méthode.
2. Pour le type de méthode, sélectionnez GET.
3. Pour Type d'intégration, sélectionnez Service AWS.
4. Pour Région AWS, sélectionnez l' Région AWS endroit où vous avez créé votre compartiment Amazon S3.

5. Pour Service AWS, sélectionnez Amazon Simple Storage Service.
6. Laissez Sous-domaine AWS vide.
7. Pour Méthode HTTP, sélectionnez GET.
8. Pour Type d'action, sélectionnez Utiliser un remplacement de chemin.

   Avec le remplacement du chemin, API Gateway transfère la demande client à Amazon S3 en tant que demande de type chemin d'API REST Amazon S3, dans laquelle une ressource Amazon S3 est exprimée par le chemin de ressource du modèle `s3-host-name/bucket/key`. API Gateway définit le `s3-host-name` et transmet les valeurs spécifiées par le client `bucket` et `key` du client vers Amazon S3.
9. Pour Remplacement de chemin, saisissez /.
10. Pour Rôle d'exécution, saisissez l'ARN de rôle pour **`APIGatewayS3ProxyPolicy`**.
11. Choisissez les paramètres de demande de méthode.

    Vous utilisez les paramètres de demande de méthode pour contrôler qui peut appeler cette méthode de votre API.
12. Pour Autorisation, dans le menu déroulant, sélectionnez `AWS_IAM`.



13. Choisissez Créer une méthode.

Cette configuration intègre la demande `GET https://your-api-host/stage/` frontale à la méthode `GET https://your-s3-host/` principale.

Pour que votre API renvoie correctement les réponses réussies et les exceptions à l'appelant, vous devez déclarer les réponses 200, 400 et 500 dans Method response. Vous utilisez le mappage par défaut pour 200 réponses afin que les réponses du backend dont le code de statut n'est pas déclaré ici soient renvoyées à l'appelant sous la forme de 200 réponses.

Pour déclarer les types de réponse pour la méthode **`GET /`**

1. Dans l’onglet Méthode de réponse, sous Réponse 200, choisissez Modifier.
2. Choisissez Ajouter un en-tête, puis procédez comme suit :

   a. Pour Nom de l’en-tête, saisissez **`Content-Type`**.

   b. Sélectionnez Add header.

   Répétez ces étapes pour créer un en-tête **`Timestamp`** et un en-tête **`Content-Length`**.
3. Choisissez Enregistrer.
4. Dans l’onglet Réponse de méthode, sous Réponses de méthode, choisissez Créer une réponse.
5. Pour Code de statut HTTP, saisissez 400.

   Vous ne définissez aucun en-tête pour cette réponse.
6. Choisissez Enregistrer.
7. Répétez les étapes suivantes pour créer la réponse 500.

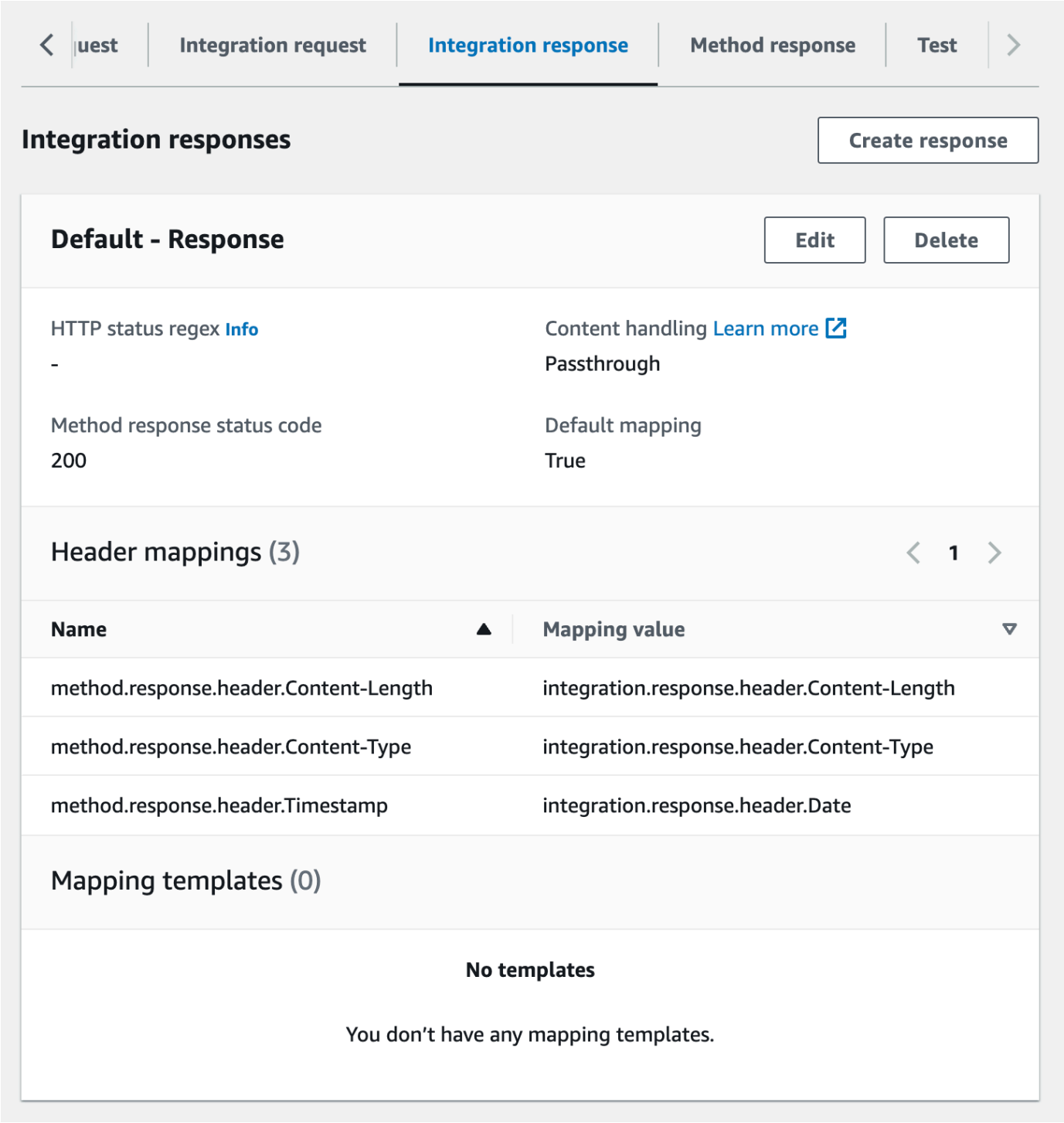
   Vous ne définissez aucun en-tête pour cette réponse.

Étant donné que la réponse d'intégration réussie d'Amazon S3 renvoie la liste de compartiments sous forme de charge utile XML et que la réponse de méthode par défaut d'API Gateway renvoie une charge utile JSON, vous devez mapper la valeur du paramètre d'en-tête Content-Type du backend à son homologue du frontend. Sinon, le client recevra `application/json` pour le type de contenu quand le corps de la réponse est en fait une chaîne XML. La procédure suivante montre comment effectuer cette configuration. En outre, vous souhaitez également afficher au client d'autres paramètres d'en-tête, tels que la date et la longueur du contenu.

Pour configurer des mappages d'en-tête de réponse pour la méthode GET /

1. Dans l’onglet Réponse d’intégration, sous Par défaut - Réponse, choisissez Modifier.

2. Pour l'en-tête Content-Length, saisissez **`integration.response.header.Content-Length`** pour la valeur de mappage.
3. Pour l'en-tête Content-Type, saisissez **`integration.response.header.Content-Type`** pour la valeur de mappage.
4. Pour l'en-tête Timestamp, saisissez **`integration.response.header.Date`** pour la valeur de mappage.
5. Choisissez Enregistrer. Le résultat devrait ressembler à ce qui suit :



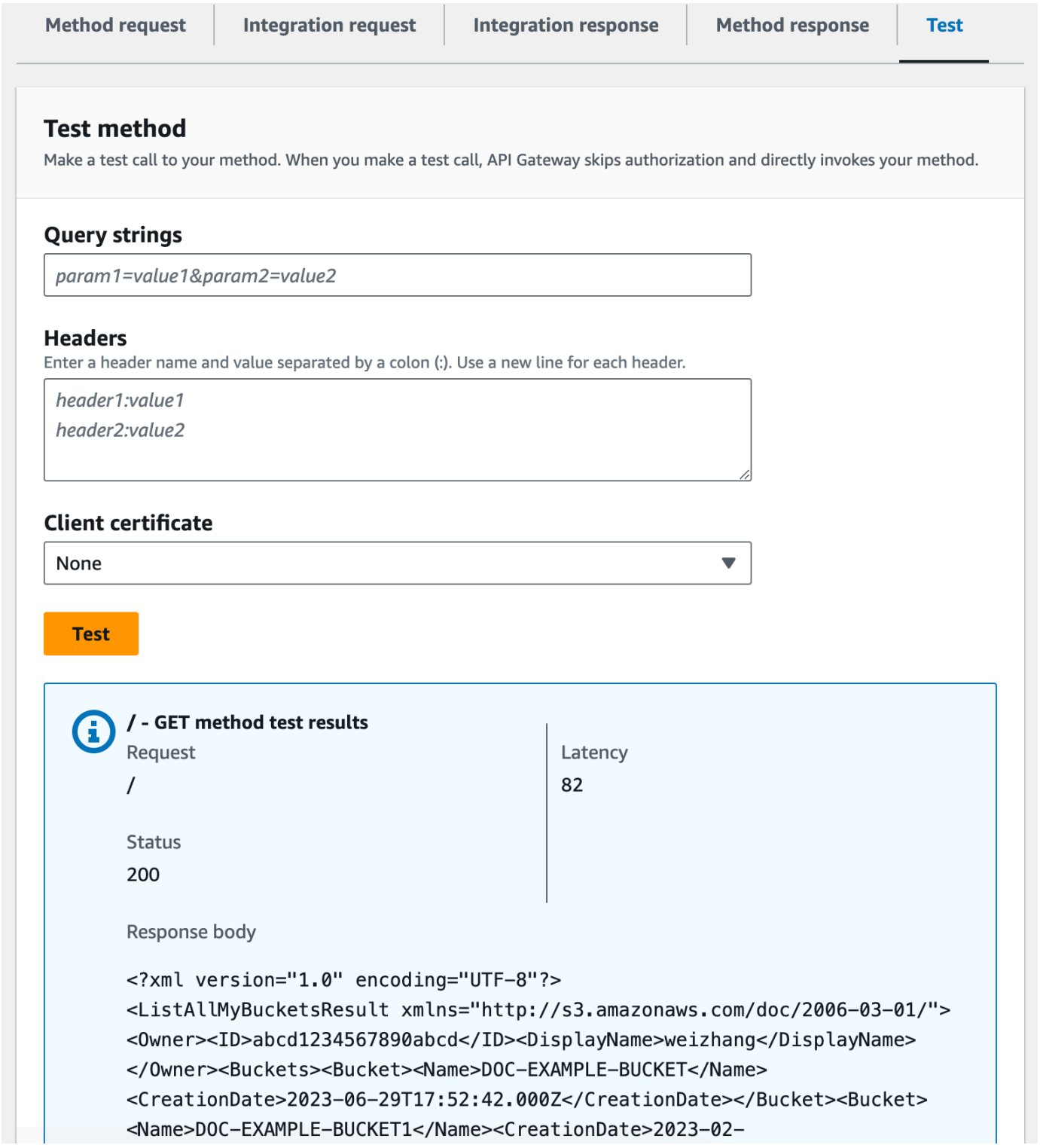
6. Dans l'onglet Réponse d'intégration, sous Réponses d'intégration, choisissez Créer une réponse.
7. Pour HTTP status regex (Regex statut HTTP), saisissez **`4\d{2}`**. Cela mappe tous les codes de statut de réponse HTTP 4xx à la réponse de la méthode.
8. Pour Code de statut de la réponse de méthode, sélectionnez **`400`**.

9. Choisissez Créer.
10. Répétez les étapes suivantes afin de créer une réponse d’intégration pour la réponse de la méthode 500. Pour HTTP status regex (Regex statut HTTP), saisissez **`5\d{2}`**.

Comme bonne pratique, vous pouvez tester l'API que vous avez configurée jusqu'à présent.

Pour tester la méthode **`GET /`**

1. Choisissez l’onglet Test. Vous devrez peut-être choisir la flèche droite pour afficher l'onglet.
2. Sélectionnez Tester). Le résultat doit ressembler à l’image suivante :


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-

Exposition de méthodes d'API pour accéder à un compartiment Amazon S3

Pour utiliser un compartiment Amazon S3, vous devez exposer la `GET` méthode sur la ressource/{folder} pour répertorier les objets d'un compartiment. Les instructions sont similaires à celles décrites dans Exposition d'une méthode d'API pour afficher la liste des compartiments Amazon S3 de l'appelant. Pour d'autres méthodes, vous pouvez importer l'exemple d'API ici, Définitions OpenAPI de l'exemple d'API en tant que proxy Amazon S3.

Pour exposer la méthode GET sur une ressource de dossier

1. Sélectionnez la ressource /{folder}, puis choisissez Créer une méthode.
2. Pour le type de méthode, sélectionnez GET.
3. Pour Type d'intégration, sélectionnez Service AWS.
4. Pour Région AWS, sélectionnez l' Région AWS endroit où vous avez créé votre compartiment Amazon S3.
5. Pour Service AWS, sélectionnez Amazon Simple Storage Service.
6. Laissez Sous-domaine AWS vide.
7. Pour Méthode HTTP, sélectionnez GET.
8. Pour Type d'action, sélectionnez Utiliser un remplacement de chemin.
9. Pour Remplacement de chemin, saisissez **`{bucket}`**.
10. Pour Rôle d'exécution, saisissez l'ARN de rôle pour **`APIGatewayS3ProxyPolicy`**.
11. Choisissez Créer une méthode.

Vous définissez le paramètre de chemin `{folder}` dans l'URL de point de terminaison Amazon S3. Vous devez mapper le paramètre de chemin `{folder}` de la requête de méthode au paramètre de chemin `{bucket}` de la requête d'intégration.

Pour mapper **`{folder}`** à **`{bucket}`**

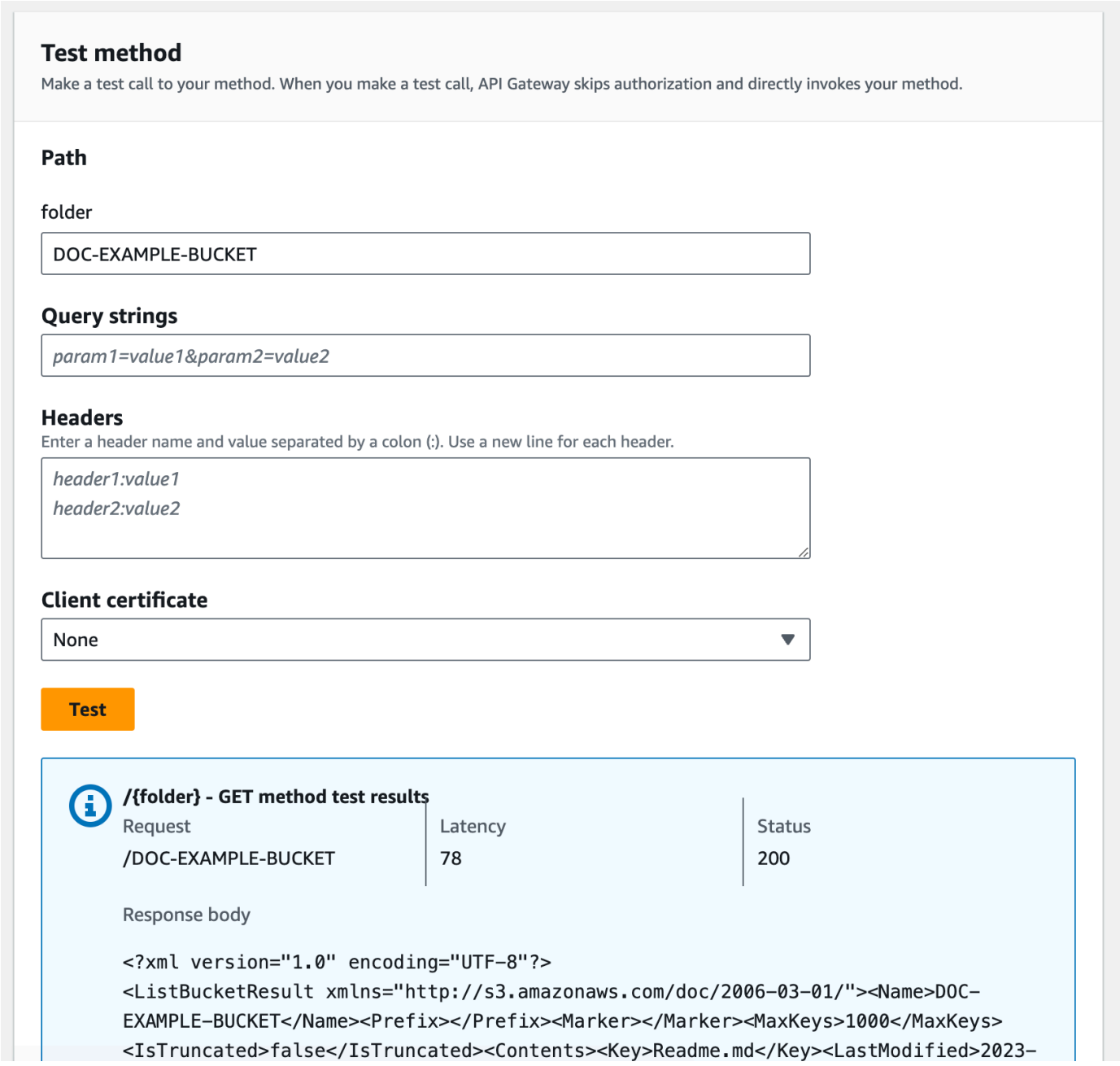
1. Dans l'onglet Requête d'intégration, sous Paramètres de requête d'intégration, choisissez Modifier.
2. Choisissez Paramètres de chemin d'URL, puis choisissez Ajouter un paramètre de chemin.
3. Pour Name (Nom), saisissez **`bucket`**.
4. Pour Mappage à partir de, entrez **`method.request.path.folder`**.
5. Choisissez Enregistrer.

Maintenant, vous testez votre API.

Pour tester la méthode **`/{folder} GET`**.

1. Choisissez l'onglet Test. Vous devrez peut-être choisir la flèche droite pour afficher l'onglet.
2. Sous Chemin, pour le dossier, saisissez le nom de votre compartiment.
3. Sélectionnez Tester).

   Le résultat du test contient une liste des objets contenus dans votre compartiment.



Exposition de méthodes d'API pour accéder à un objet Amazon S3 dans un compartiment

Amazon S3 prend en charge les actions GET, DELETE, HEAD, OPTIONS, POST et PUT pour accéder aux objets d'un compartiment donné et les gérer. Dans ce didacticiel, vous allez exposer une méthode `GET` sur la ressource `{folder}/{item}` pour obtenir une image à partir d'un compartiment. Pour d'autres applications de la ressource `{folder}/{item}`, consultez l'exemple d'API,Définitions OpenAPI de l'exemple d'API en tant que proxy Amazon S3.

Pour exposer la méthode GET sur une ressource d'élément

1. Sélectionnez la ressource /{item}, puis choisissez Créer une méthode.
2. Pour le type de méthode, sélectionnez GET.
3. Pour Type d'intégration, sélectionnez Service AWS.
4. Pour Région AWS, sélectionnez l' Région AWS endroit où vous avez créé votre compartiment Amazon S3.
5. Pour Service AWS, sélectionnez Amazon Simple Storage Service.
6. Laissez Sous-domaine AWS vide.
7. Pour Méthode HTTP, sélectionnez GET.
8. Pour Type d'action, sélectionnez Utiliser un remplacement de chemin.
9. Pour Remplacement de chemin, saisissez {bucket}/{object}.
10. Pour Rôle d'exécution, saisissez l'ARN de rôle pour **`APIGatewayS3ProxyPolicy`**.
11. Choisissez Créer une méthode.

Vous définissez les paramètres de chemin `{folder}` et `{item}` dans l'URL de point de terminaison Amazon S3. Vous devez mapper le paramètre de chemin de la requête de méthode au paramètre de chemin de la requête d'intégration.

Dans cette étape, vous effectuez les opérations suivantes :

- Mappez le paramètre de chemin `{folder}` de la demande de méthode au paramètre de chemin `{bucket}` de la demande d'intégration.
- Mappez le paramètre de chemin `{item}` de la demande de méthode au paramètre de chemin `{object}` de la demande d'intégration.

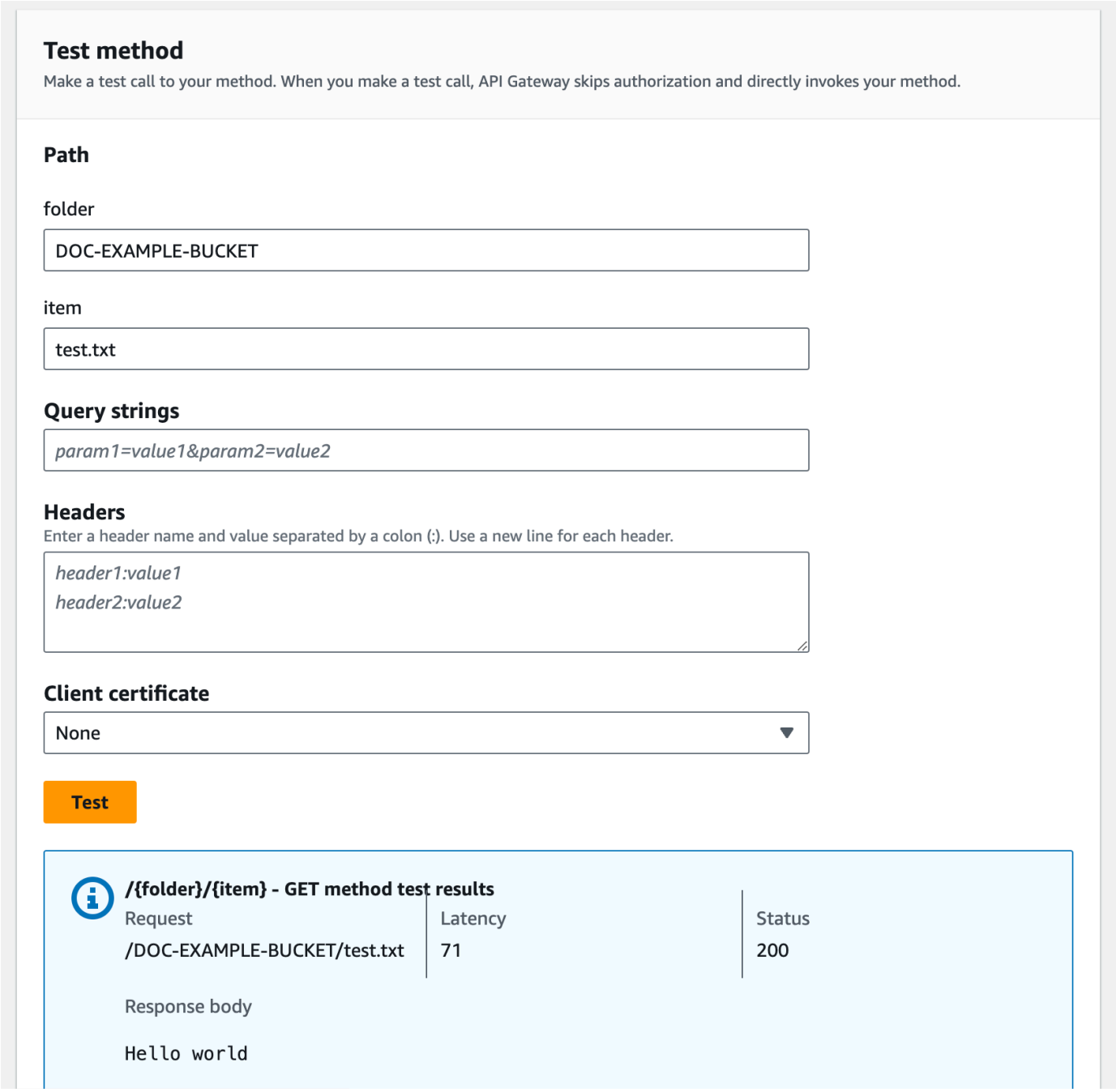
Pour mapper **{folder}** à **{bucket}** et **{item}** à **{object}**

1. Dans l'onglet Requête d'intégration, sous Paramètres de requête d'intégration, choisissez Modifier.
2. Choisissez paramètres de chemin d'URL.
3. Choisissez Ajouter un paramètre de chemin.
4. Pour Name (Nom), saisissez **bucket**.
5. Pour Mappage à partir de, entrez **method.request.path.folder**.
6. Choisissez Ajouter un paramètre de chemin.
7. Pour Name (Nom), saisissez **object**.
8. Pour Mappage à partir de, entrez **method.request.path.item**.
9. Choisissez Enregistrer.

Pour tester la méthode **/{folder}/{object} GET**.

1. Choisissez l'onglet Test. Vous devrez peut-être choisir la flèche droite pour afficher l'onglet.
2. Sous Chemin, pour le dossier, saisissez le nom de votre compartiment.
3. Sous Chemin, pour l'élément, saisissez le nom d'un élément.
4. Sélectionnez Tester).

   Le corps de la réponse contiendra le contenu de l'élément.



La requête renvoie correctement le texte brut (« Hello world ») comme contenu du fichier spécifié (test.txt) dans le compartiment Amazon S3 donné (DOC-EXAMPLE-BUCKET).

Pour télécharger ou charger des fichiers binaires (dans API Gateway, tout contenu autre que du contenu JSON codé en UTF-8), des paramètres d'API supplémentaires sont requis. La procédure à suivre est expliquée ci-après :

Pour télécharger ou charger des fichiers binaires à partir de S3

1. Enregistrez les types de média du fichier concerné dans les API binaryMediaTypes. Vous pouvez réaliser cette opération dans la console :

   a. Choisissez Paramètres de l'API pour l'API.
   b. Sous Types de médias binaires, choisissez Ajouter un type de média binaire.
   c. Choisissez Ajouter un type de média binaire, puis saisissez le type de média requis, par exemple `image/png`.
   d. Choisissez Save changes (Enregistrer les modifications) pour sauvegarder le paramètre.
2. Ajoutez l'en-tête `Content-Type` (pour charger) et/ou `Accept` (pour télécharger) à la demande de méthode pour exiger que le client indique le type de média binaire requis et l'associe à la demande d'intégration.
3. Définissez Traitement du contenu sur `Passthrough` dans la demande d'intégration (pour charger) et dans une réponse d'intégration (pour télécharger). Assurez-vous qu'aucun modèle de mappage n'est défini pour le type de contenu en question. Pour plus d'informations, consultez Comportements de transfert direct et Sélection d'un modèle de mappage VTL.

La taille de la charge utile ne doit pas dépasser 10 Mo. Voir Quotas API Gateway pour la configuration et l'exécution d'une API REST.

Assurez-vous que les types de contenu appropriés ont été ajoutés dans les métadonnées des fichiers sur Amazon S3. Pour du contenu multimédia lisible en streaming, `Content-Disposition:inline` peut également être ajouté aux métadonnées.

Pour de plus amples informations sur la prise en charge des fichiers binaires dans API Gateway, veuillez consulter Conversions du type de contenu dans API Gateway.

Définitions OpenAPI de l'exemple d'API en tant que proxy Amazon S3

Les définitions d'OpenAPI suivantes décrivent une API qui fonctionne comme un proxy Amazon S3. Cette API contient plus d'opérations Amazon S3 que l'API que vous avez créée dans le didacticiel. Les méthodes suivantes sont exposées dans les définitions d'OpenAPI :

- Exposition de la méthode GET sur la ressource racine de l'API pour afficher la liste de tous les compartiments Amazon S3 d'un appelant.
- Exposition de la méthode GET sur une ressource Folder pour afficher la liste de tous les objets d'un compartiment Amazon S3.

- Exposition de la méthode PUT sur une ressource Folder pour ajouter un compartiment à Amazon S3.
- Exposition de la méthode DELETE sur une ressource Folder pour supprimer un compartiment d'Amazon S3.
- Exposition de la méthode GET sur une ressource Folder/Item pour afficher ou télécharger un objet à partir d'un compartiment Amazon S3.
- Exposition de la méthode PUT sur une ressource Folder/Item pour charger un objet dans un compartiment Amazon S3.
- Exposition de la méthode HEAD sur une ressource Folder/Item pour obtenir les métadonnées d'objet d'un compartiment Amazon S3.
- Exposition de la méthode DELETE sur une ressource Folder/Item pour supprimer un objet d'un compartiment Amazon S3.

Pour de plus amples informations sur l'importation d'une API à l'aide de la définition OpenAPI, veuillez consulter Configuration d'une API REST à l'aide d'OpenAPI.

Pour obtenir des instructions sur la création d'une API similaire, consultez Tutoriel : Création d'une API REST en tant que proxy Amazon S3 dans API Gateway.

Pour savoir comment invoquer cette API à l'aide de Postman, qui prend en charge l'autorisation AWS IAM, consultez. Appel de l'API à l'aide d'un client d'API REST

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"
      }
    }
  }
}
```

Appel de l'API à l'aide d'un client d'API REST

Pour fournir un end-to-end didacticiel, nous montrons maintenant comment appeler l'API à l'aide de Postman, qui prend en charge l'autorisation AWS IAM.

Pour appeler notre API proxy Amazon S3 à l'aide de Postman

1. Déployez ou redéployez l'API. Notez l'URL de base de l'API qui s'affiche à côté de Invoke URL en haut de Stage Editor.
2. Lancez Postman.
3. Choisissez Autorisation, puis choisissez `AWS Signature`. Entrez l'ID de clé d'accès et la clé d'accès secrète de votre utilisateur IAM dans les champs de SecretKeysaisie AccessKeyet, respectivement. Entrez la AWS région dans laquelle votre API est déployée dans la AWS zone de texte Région. Tapez `execute-api` dans le champ de saisie Service Name.

   Vous pouvez créer une paire de clés à partir de l'onglet Security Credentials (Informations d'identification de sécurité de votre compte d'utilisateur IAM dans IAM Management Console.
4. Pour ajouter un compartiment nommé `apig-demo-5` à votre compte Amazon S3 dans la région *`{region}`* :

   > **Note**
   >
   > Veillez à ce que le nom du compartiment soit globalement unique.

   a. Choisissez PUT dans la liste déroulante de méthodes et le tapez l'URL de la méthode (`https://`*`api-id`*`.execute-api.`*`aws-region`*`.amazonaws.com/`*`stage`*`/`*`folder-name`*)

   b. Définissez la valeur de l'en-tête `Content-Type` sur `application/xml`. Vous devrez peut-être supprimer des en-têtes existants avant de définir le type de contenu.

   c. Choisissez l'élément de menu Body et tapez le fragment XML suivant comme corps de la demande :

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

   d. Choisissez Send pour envoyer la demande. Si l'opération aboutit, vous devez recevoir une réponse `200 OK` avec une charge utile vide.
5. Pour ajouter un fichier texte à un compartiment, suivez les instructions ci-dessus. Si vous spécifiez le nom de compartiment **`apig-demo-5`** pour `{folder}` et le nom de fichier **`Readme.txt`** pour `{item}` dans l'URL, et si vous fournissez la chaîne de texte **`Hello,`**

`World!` comme contenu de fichier (qui devient de ce fait la charge utile de la demande), la demande devient :

```
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!
```

Si tout se passe bien, vous devez recevoir une réponse `200 OK` avec une charge utile vide.

6. Pour obtenir le contenu du fichier `Readme.txt` que nous venons d'ajouter au compartiment `apig-demo-5`, exécutez une demande GET comme suit :

```
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
```

Si la demande aboutit, vous devez recevoir une réponse `200 OK` avec la chaîne de texte `Hello, World!` comme charge utile.

7. Pour répertorier les éléments du compartiment `apig-demo-5`, envoyez la demande suivante :

```
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
```

Si la demande aboutit, vous devez recevoir une réponse `200 OK` avec une charge utile XML montrant un seul élément dans le compartiment spécifié, sauf si vous avez ajouté d'autres fichiers au compartiment avant d'envoyer cette demande.

```
<?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**
>
> Pour charger ou télécharger une image, vous devez affecter la valeur CONVERT_TO_BINARY au paramètre de gestion de contenu.

Tutoriel : Création d'une API REST en tant que proxy Amazon Kinesis dans API Gateway

Cette page explique comment créer et configurer une API REST avec une intégration du type `AWS` pour accéder à Kinesis.

> **Note**
>
> Pour intégrer votre API API Gateway à Kinesis, vous devez choisir une région où les services API Gateway et Kinesis sont disponibles. Pour connaître la disponibilité dans les régions, veuillez consulter Points de terminaison de service et quotas.

A titre d'illustration, nous allons créer un exemple d'API pour permettre à un client d'effectuer les tâches suivantes :

1. Affichage de la liste des flux disponibles de l'utilisateur dans Kinesis
2. Création, description ou suppression d'un flux spécifique
3. Lecture ou écriture des enregistrements de données du flux spécifié

Pour effectuer les tâches précédentes, l'API expose les méthodes sur diverses ressources pour appeler les actions suivantes, respectivement :

1. L'action `ListStreams` dans Kinesis
2. L'action `CreateStream`, `DescribeStream` ou `DeleteStream`
3. L'action `GetRecords` ou `PutRecords` (y compris `PutRecord`) dans Kinesis

Plus précisément, nous allons concevoir l'API comme suit :

- Exposez une méthode HTTP GET sur la `/streams` ressource de l'API et intégrez-la à l'ListStreamsaction dans Kinesis pour répertorier les flux du compte de l'appelant.
- Exposez une méthode HTTP POST sur la `/streams/{stream-name}` ressource de l'API et intégrez-la à l'CreateStreamaction dans Kinesis pour créer un flux nommé dans le compte de l'appelant.
- Exposez une méthode HTTP GET sur la `/streams/{stream-name}` ressource de l'API et intégrez-la à l'DescribeStreamaction dans Kinesis pour décrire un flux nommé dans le compte de l'appelant.
- Exposez une méthode HTTP DELETE sur la `/streams/{stream-name}` ressource de l'API et intégrez-la à l'DeleteStreamaction dans Kinesis pour supprimer un flux dans le compte de l'appelant.

- Exposez une méthode HTTP PUT sur la `/streams/{stream-name}/record` ressource de l'API et intégrez-la à l'PutRecordaction dans Kinesis. Cela permet au client d'ajouter un enregistrement de données unique au flux nommé.
- Exposez une méthode HTTP PUT sur la `/streams/{stream-name}/records` ressource de l'API et intégrez-la à l'PutRecordsaction dans Kinesis. Cela permet au client d'ajouter une liste d'enregistrements de données au flux nommé.
- Exposez une méthode HTTP GET sur la `/streams/{stream-name}/records` ressource de l'API et intégrez-la à l'GetRecordsaction dans Kinesis. Cela permet au client d'afficher la liste des enregistrements de données du flux nommé, avec un itérateur de partition spécifique. Un itérateur de partition spécifie la position de la partition à partir de laquelle la lecture séquentielle des enregistrements de données doit commencer.
- Exposez une méthode HTTP GET sur la `/streams/{stream-name}/sharditerator` ressource de l'API et intégrez-la à l'GetShardIteratoraction dans Kinesis. Cette méthode d'assistance doit être fournie à l'action `ListStreams` dans Kinesis.

Vous pouvez appliquer les instructions présentées ici aux autres actions Kinesis. Pour obtenir la liste complète des actions Kinesis, veuillez consulter la Référence d'API Amazon Kinesis.

Au lieu d'utiliser la console API Gateway pour créer l'exemple d'API, vous pouvez importer l'exemple d'API dans API Gateway à l'aide de l'API d'importation d'API Gateway. Pour plus d'informations sur l'utilisation de la fonction d'importation d'API, consultez Configuration d'une API REST à l'aide d'OpenAPI.

Création d'un rôle et d'une stratégie IAM pour permettre à l'API d'accéder à Kinesis

Pour autoriser l’API à invoquer des actions Kinesis, vous devez avoir attaché les politiques IAM appropriées à un rôle IAM.

Pour créer le rôle d'exécution du proxy de AWS service

1. Connectez-vous à la console IAM AWS Management Console et ouvrez-la à l'adresse https://console.aws.amazon.com/iam/.
2. Sélectionnez Roles.
3. Sélectionnez Créer un rôle.
4. Choisissez le AWS service sous Sélectionner le type d'entité de confiance, puis sélectionnez API Gateway et sélectionnez Allows API Gateway to push CloudWatch logs vers Logs.

5. Choisissez Suivant, puis Suivant.
6. Pour Nom du rôle, saisissez **`APIGatewayKinesisProxyPolicy`**, puis choisissez Créer un rôle.
7. Dans la liste Roles, choisissez le rôle que vous venez de créer. Vous devrez peut-être faire défiler la page ou utiliser la barre de recherche pour rechercher le rôle.
8. Pour le rôle sélectionné, sélectionnez l'onglet Ajouter des autorisations.
9. Choisissez Attacher des politiques dans la liste déroulante.
10. Dans la barre de recherche, saisissez **`AmazonKinesisFullAccess`** et choisissez Ajouter des autorisations.

> **Note**
>
> Ce didacticiel utilise une stratégie gérée pour plus de simplicité. Il est recommandé de créer votre propre stratégie IAM afin d'accorder les autorisations minimales requises.

11. Notez l'ARN du rôle nouvellement créé. Vous l'utiliserez ultérieurement.

Création d'une API en tant que proxy Kinesis

Procédez comme suit pour créer l'API dans la console API Gateway.

Pour créer une API en tant que proxy AWS de service pour Kinesis

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Si vous utilisez API Gateway pour la première fois, vous voyez une page qui vous présente les fonctions du service. Sous REST API (API REST), choisissez Build (Création). Lorsque la fenêtre contextuelle Create Example API (Créer API exemple) s'affiche, cliquez sur OK.

   Si ce n'est pas la première fois que vous utilisez API Gateway, choisissez Create API (Créer une API). Sous REST API (API REST), choisissez Build (Création).
3. Choisissez New API (Nouvelle API).
4. Sous API name (Nom de l'API), saisissez **`KinesisProxy`**. Conservez les valeurs par défaut pour tous les autres champs.
5. (Facultatif) Sous Description, entrez une description.
6. Sélectionnez Create API (Créer une API).

Une fois l'API créée, la console API Gateway affiche la page Resources, qui contient uniquement la ressource racine (`/`) de cette API.

Liste des flux dans Kinesis

Kinesis prend en charge l'action `ListStreams` avec l'appel API REST suivant :

```
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>

{
   ...
}
```

Dans la demande API REST ci-dessus, l'action est spécifiée dans le paramètre de requête `Action`. Vous pouvez également spécifier l'action dans l'en-tête `X-Amz-Target`, plutôt que :

```
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
{
   ...
}
```

Dans ce didacticiel, nous utilisons le paramètre de requête pour spécifier l'action.

Pour exposer une action Kinesis dans l'API, ajoutez une ressource `/streams` à la racine de l'API. Ensuite, définissez une méthode `GET` sur la ressource et intégrez-la à l'action `ListStreams` de Kinesis.

La procédure suivante décrit comment répertorier les flux Kinesis à l'aide de la console API Gateway.

Pour répertorier les flux Kinesis à l'aide de la console API Gateway

1. Sélectionnez la ressource `/`, puis choisissez Créer une ressource.
2. Sous Resource Name (Nom de la ressource), entrez **`streams`**.
3. Maintenez CORS (Partage des ressources entre origines multiples) désactivé.
4. Choisissez Créer une ressource.
5. Choisissez la ressource `/streams`, puis choisissez Créer une méthode et procédez comme suit :

   a. Pour Type de méthode, sélectionnez GET.

   > **Note**
   >
   > Le verbe HTTP pour une méthode appelée par un client peut différer du verbe HTTP pour une intégration requise par le serveur principal. Nous sélectionnons `GET` dans ce cas, car l'affichage de la liste des flux est intuitivement une opération READ.

   b. Pour Type d'intégration, sélectionnez Service AWS .

   c. Pour Région AWS, sélectionnez l' Région AWS endroit où vous avez créé votre flux Kinesis.

   d. Pour Service AWS, sélectionnez Kinesis.

   e. Laissez Sous-domaine AWS vide.

   f. Dans le champ HTTP Method, sélectionnez POST.

   > **Note**
   >
   > Nous avons choisi `POST` dans ce cas, car Kinesis exige que l'action `ListStreams` soit appelée avec lui.

   g. Pour Type d'action, choisissez Utiliser un nom d'action.

   h. Pour Nom de l'action, saisissez **`ListStreams`**.

   i. Pour Rôle d'exécution, saisissez l'ARN de votre rôle d'exécution.

   j. Conservez la valeur par défaut de Transmettre pour Gestion de contenu.

   k. Choisissez Créer une méthode.

6. Dans l'onglet Requête d'intégration, sous Paramètres de requête d'intégration, choisissez Modifier.

7. Pour Transmission du corps de requête, sélectionnez Lorsqu'aucun modèle n'est défini (recommandé).
8. Choisissez Paramètres des en-têtes de requête d'URL et procédez comme suit :

   a. Choisissez le paramètre Ajouter des en-têtes de requête.

   b. Pour Name (Nom), saisissez **`Content-Type`**.

   c. Pour Mappage à partir de, entrez **`'application/x-amz-json-1.1'`**.

   Nous avons utilisé un mappage de paramètre de requête pour définir l'en-tête `Content-Type` sur la valeur statique de `'application/x-amz-json-1.1'` afin d'informer Kinesis que l'entrée est une version spécifique de JSON.
9. Choisissez Modèles de mappage, puis choisissez Ajouter un modèle de mappage, puis procédez comme suit :

   a. Pour Type de contenu, saisissez **`application/json`**.

   b. Pour Corps du modèle, saisissez **`{}`**.

   c. Choisissez Enregistrer.

   La ListStreamsdemande prend une charge utile au format JSON suivant :

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

   Cependant, les propriétés sont facultatives. Pour utiliser les valeurs par défaut, nous avons choisi une charge utile JSON vide dans ce cas.
10. Testez la méthode GET sur la ressource /streams pour invoquer l'action `ListStreams` dans Kinesis :

    Choisissez l'onglet Test. Vous devrez peut-être choisir la flèche droite pour afficher l'onglet.

    Choisissez Tester pour tester votre méthode.

Si vous avez déjà créé deux flux nommés « myStream » et « yourStream » dans Kinesis, le test réussi renvoie une réponse 200 OK contenant la charge utile suivante :

```
{
     "HasMoreStreams": false,
     "StreamNames": [
         "myStream",
         "yourStream"
     ]
}
```

Création, description et suppression d'un flux dans Kinesis

La création, la description et la suppression d'un flux dans Kinesis impliquent respectivement d'effectuer les demandes d'API REST Kinesis suivantes :

```
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"
}
```

Nous pouvons créer l'API pour qu'elle accepte l'entrée requises en tant que charge utile JSON de la demande de méthode et la transmette via la demande d'intégration. Cependant, pour fournir davantage d'exemples de mappage de données entre les demandes de méthode et d'intégration, et entre les réponses de méthode et d'intégration, nous allons créer notre API légèrement différemment.

Nous exposons les méthodes `GETPOST`,, et `Delete` HTTP sur une to-be-named `Stream` ressource. Nous utilisons la variable de chemin `{stream-name}` comme espace réservé de la ressource de flux et intégrons respectivement ces méthodes d'API aux actions `DescribeStream`, `CreateStream` et `DeleteStream`, respectivement. Nous exigeons que le client transmette les autres données d'entrée, telles que les en-têtes, les paramètres de requête ou la charge utile d'une demande de méthode. Nous fournissons des modèles de mappage pour transformer ces données en charge utile de la demande d'intégration requise.

Pour créer la ressource {stream-name}

1. Choisissez la ressource /streams, puis choisissez Create resource.
2. Maintenez Ressource proxy désactivée.
3. Pour Chemin de ressource, sélectionnez `/streams`.
4. Sous Resource Name (Nom de la ressource), entrez **`{stream-name}`**.
5. Maintenez CORS (Partage des ressources entre origines multiples) désactivé.
6. Choisissez Créer une ressource.

Pour configurer et tester la méthode GET sur une ressource de flux

1. Choisissez la ressource/{stream-name}, puis choisissez Create method.
2. Pour Type de méthode, sélectionnez GET.
3. Pour Type d'intégration, sélectionnez Service AWS .
4. Pour Région AWS, sélectionnez l' Région AWS endroit où vous avez créé votre flux Kinesis.
5. Pour Service AWS, sélectionnez Kinesis.
6. Laissez Sous-domaine AWS vide.
7. Dans le champ HTTP Method, sélectionnez POST.
8. Pour Type d'action, choisissez Utiliser un nom d'action.
9. Pour Nom de l'action, saisissez **`DescribeStream`**.
10. Pour Rôle d'exécution, saisissez l'ARN de votre rôle d'exécution.
11. Conservez la valeur par défaut de Transmettre pour Gestion de contenu.
12. Choisissez Créer une méthode.
13. Dans la section Requête d'intégration, ajoutez les Paramètres d'en-tête de requête d'URL suivants :

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

    La tâche suit la même procédure pour configurer le mappage de paramètre de demande pour la méthode `GET /streams`.
14. Ajoutez le modèle de mappage de corps suivant pour mapper des données à partir de la demande de méthode `GET /streams/{stream-name}` à la demande d'intégration `POST /?Action=DescribeStream` :

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

    Ce modèle de mappage génère la charge utile de demande d'intégration requise pour l'action `DescribeStream` de Kinesis à partir de la valeur du paramètre de chemin `stream-name` de la demande de méthode.
15. Pour tester la méthode `GET /stream/{stream-name}` permettant d'invoquer l'action `DescribeStream` dans Kinesis, cliquez sur l'onglet Tester.

16. Pour Chemin, sous nom du flux, entrez le nom d'un flux Kinesis existant.
17. Sélectionnez Tester). Si le test aboutit, une réponse 200 OK est renvoyée avec une charge utile similaire à celle qui suit :


```
{
  "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"
  }
}
```


Après avoir déployé l'API, vous pouvez exécuter une demande REST sur cette méthode d'API :

```
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
```

Pour configurer et tester la méthode POST sur une ressource de flux

1. Choisissez la ressource/{stream-name}, puis choisissez Create method.
2. Pour Type de méthode, sélectionnez POST.
3. Pour Type d'intégration, sélectionnez Service AWS .
4. Pour Région AWS, sélectionnez l' Région AWS endroit où vous avez créé votre flux Kinesis.
5. Pour Service AWS, sélectionnez Kinesis.
6. Laissez Sous-domaine AWS vide.
7. Dans le champ HTTP Method, sélectionnez POST.
8. Pour Type d'action, choisissez Utiliser un nom d'action.
9. Pour Nom de l'action, saisissez **`CreateStream`**.
10. Pour Rôle d'exécution, saisissez l'ARN de votre rôle d'exécution.
11. Conservez la valeur par défaut de Transmettre pour Gestion de contenu.
12. Choisissez Créer une méthode.
13. Dans la section Requête d'intégration, ajoutez les Paramètres d'en-tête de requête d'URL suivants :

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

    La tâche suit la même procédure pour configurer le mappage de paramètre de demande pour la méthode `GET /streams`.
14. Ajoutez le modèle de mappage de corps suivant pour mapper des données à partir de la demande de méthode `POST /streams/{stream-name}` à la demande d'intégration `POST /?Action=CreateStream` :

    ```
    {
    ```

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

Dans le modèle de mappage précédent, nous configurons `ShardCount` sur une valeur fixe de 5, si le client ne spécifie aucune valeur dans la charge utile de la demande de méthode.

15. Pour tester la méthode `POST /stream/{stream-name}` permettant d'invoquer l'action `CreateStream` dans Kinesis, cliquez sur l'onglet Tester.
16. Pour Chemin, sous nom du flux, entrez le nom d'un nouveau flux Kinesis.
17. Sélectionnez Tester). Si le test aboutit, une réponse 200 OK est renvoyée sans aucune donnée.

Après avoir déployé l'API, vous pouvez également exécuter une demande d'API REST avec la méthode POST sur une ressource de flux pour appeler l'action `CreateStream` dans Kinesis :

```
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
}
```

Configuration et test de la méthode DELETE sur une ressource de flux

1. Choisissez la ressource/{stream-name}, puis choisissez Create method.
2. Pour Type de méthode, sélectionnez DELETE.
3. Pour Type d'intégration, sélectionnez Service AWS .
4. Pour Région AWS, sélectionnez l' Région AWS endroit où vous avez créé votre flux Kinesis.
5. Pour Service AWS, sélectionnez Kinesis.
6. Laissez Sous-domaine AWS vide.
7. Dans le champ HTTP Method, sélectionnez POST.

8. Pour Type d'action, choisissez Utiliser un nom d'action.
9. Pour Nom de l'action, saisissez **`DeleteStream`**.
10. Pour Rôle d'exécution, saisissez l'ARN de votre rôle d'exécution.
11. Conservez la valeur par défaut de Transmettre pour Gestion de contenu.
12. Choisissez Créer une méthode.
13. Dans la section Requête d'intégration, ajoutez les Paramètres d'en-tête de requête d'URL suivants :

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

    La tâche suit la même procédure pour configurer le mappage de paramètre de demande pour la méthode `GET /streams`.
14. Ajoutez le modèle de mappage de corps suivant pour mapper des données à partir de la demande de méthode `DELETE /streams/{stream-name}` à la demande d'intégration correspondante de `POST /?Action=DeleteStream` :

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

    Ce modèle de mappage génère l'entrée requise pour l'action `DELETE /streams/{stream-name}` à partir du nom du chemin de l'URL fourni par le client de `stream-name`.
15. Pour tester la méthode `DELETE /stream/{stream-name}` permettant d'invoquer l'action `DeleteStream` dans Kinesis, cliquez sur l'onglet Tester.
16. Pour Chemin, sous nom du flux, entrez le nom d'un flux Kinesis existant.
17. Sélectionnez Tester). Si le test aboutit, une réponse 200 OK est renvoyée sans aucune donnée.

    Après avoir déployé l'API, vous pouvez également exécuter la demande d'API REST suivante avec la méthode DELETE sur la ressource de flux pour appeler l'action `DeleteStream` dans Kinesis :

    ```
    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

{}
```

Obtention d'enregistrements d'un flux et ajout d'enregistrements à un flux dans Kinesis

Après avoir créé un flux dans Kinesis, vous pouvez ajouter des enregistrements de données à ce flux et lire les données du flux. L'ajout d'enregistrements de données implique d'appeler l'PutRecordaction PutRecordsou dans Kinesis. La première ajoute plusieurs enregistrements, tandis que la seconde ajoute un seul enregistrement au flux.

```
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"
}
```

ou

```
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"
}
```

Ici, `StreamName` identifie le flux cible pour ajouter des enregistrements. `StreamName`, `Data`, et `PartitionKey` sont des données d'entrée requises. Dans notre exemple, nous utilisons les valeurs par défaut de toutes les données d'entrée facultatives et nous ne spécifierons pas explicitement leurs valeurs dans les données d'entrée de la demande de méthode.

Lire des données dans Kinesis revient à lancer l'action suivante : GetRecords

```
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
}
```

Ici, le flux source à partir duquel nous obtenons les enregistrements est spécifié dans la valeur requise `ShardIterator`, comme indiqué dans l'action Kinesis suivante pour obtenir un itérateur de partition :

```
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
```

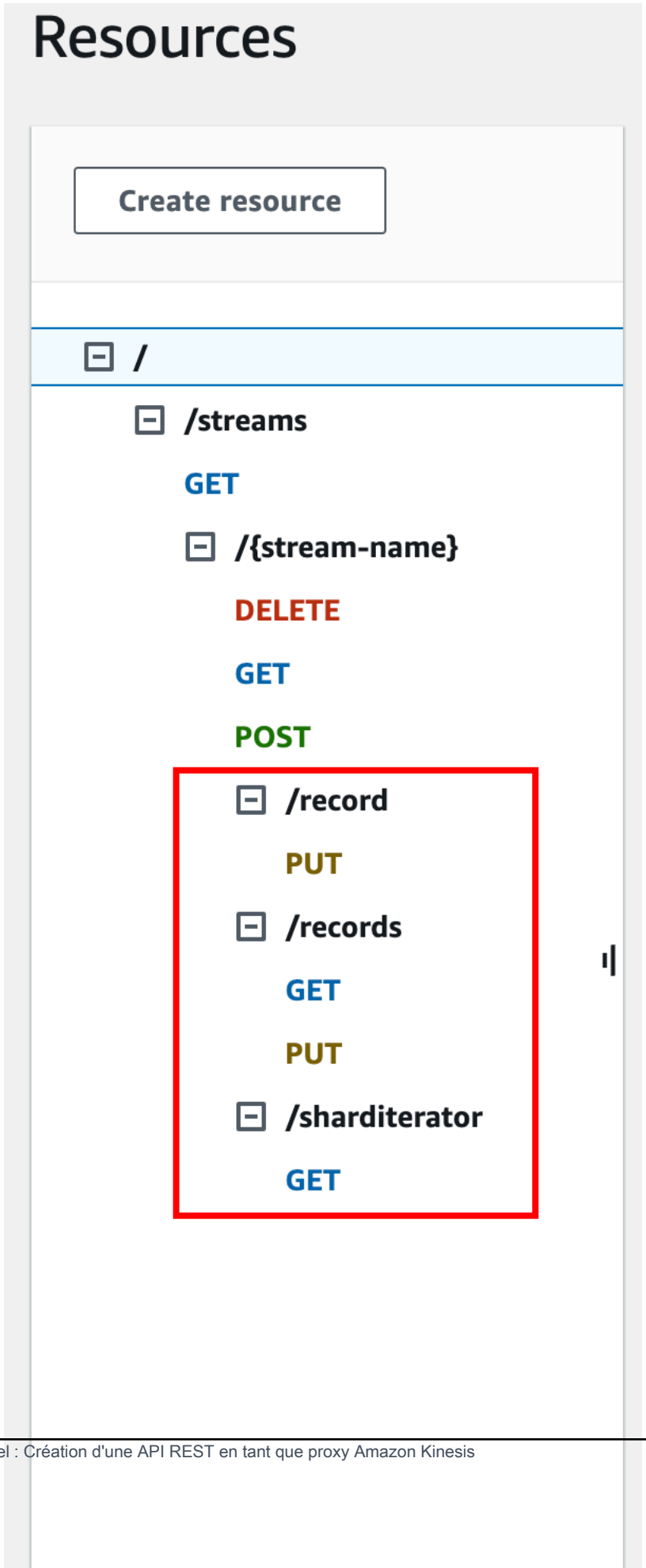
```
{
    "ShardId": "string",
    "ShardIteratorType": "string",
    "StartingSequenceNumber": "string",
    "StreamName": "string"
}
```

Pour les actions `GetRecords` et `PutRecords`, nous exposons respectivement les méthodes `GET` et `PUT` sur une ressource `/records` qui est ajoutée à une ressource de flux nommé (`/{stream-name}`). De même, nous exposons l'action `PutRecord` en tant que méthode `PUT` sur une ressource `/record`.

Etant donné que l'action `GetRecords` utilise comme données d'entrée une valeur `ShardIterator`, qui est obtenue en appelant l'action d'assistance `GetShardIterator`, nous exposons une méthode d'assistance `GET` sur une ressource `ShardIterator` (`/sharditerator`).

Pour créer les ressources /record, /records et /sharditerator

1. Choisissez la ressource/{stream-name}, puis choisissez Create resource.
2. Maintenez Ressource proxy désactivée.
3. Pour Chemin de ressource, sélectionnez `/{stream-name}`.
4. Sous Resource Name (Nom de la ressource), entrez **`record`**.
5. Maintenez CORS (Partage des ressources entre origines multiples) désactivé.
6. Choisissez Créer une ressource.
7. Répétez les étapes précédentes pour créer une ressource /records et une ressource / sharditerator. L’API finale doit ressembler à ce qui suit :


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

Les quatre procédures suivantes expliquent comment configurer chacune des méthodes, comment mapper les données des demandes de méthode aux demandes d'intégration et comment tester les méthodes.

Pour configurer et tester la méthode **`PUT /streams/{stream-name}/record`** de façon à ce qu'elle appelle **`PutRecord`** dans Kinesis

1. Choisissez /record, puis choisissez Create method.
2. Pour Type de méthode, sélectionnez PUT.
3. Pour Type d'intégration, sélectionnez Service AWS .
4. Pour Région AWS, sélectionnez l' Région AWS endroit où vous avez créé votre flux Kinesis.
5. Pour Service AWS, sélectionnez Kinesis.
6. Laissez Sous-domaine AWS vide.
7. Dans le champ HTTP Method, sélectionnez POST.
8. Pour Type d'action, choisissez Utiliser un nom d'action.
9. Pour Nom de l'action, saisissez **`PutRecord`**.
10. Pour Rôle d'exécution, saisissez l'ARN de votre rôle d'exécution.
11. Conservez la valeur par défaut de Transmettre pour Gestion de contenu.
12. Choisissez Créer une méthode.
13. Dans la section Requête d'intégration, ajoutez les Paramètres d'en-tête de requête d'URL suivants :

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

    La tâche suit la même procédure pour configurer le mappage de paramètre de demande pour la méthode `GET /streams`.
14. Ajoutez le modèle de mappage de corps suivant pour mapper des données à partir de la demande de méthode `PUT /streams/{stream-name}/record` à la demande d'intégration correspondante de `POST /?Action=PutRecord` :

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

Ce modèle de mappage suppose que la charge utile de la demande de méthode est au format suivant :

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

Ces données peuvent être modélisées par le schéma JSON suivant :

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

Vous pouvez créer un modèle pour inclure ce schéma et utiliser ce modèle pour faciliter la génération du modèle de mappage. Cependant, vous pouvez également générer un modèle de mappage sans utiliser aucun modèle.

15. Pour tester la méthode `PUT /streams/{stream-name}/record`, définissez la variable de chemin `stream-name` sur le nom d'un flux existant, fournissez une charge utile au format requis, puis envoyez la demande de méthode. Si le test aboutit, le résultat est une réponse `200 OK` avec une charge utile au format suivant :

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

Pour configurer et tester la méthode **`PUT /streams/{stream-name}/records`** de façon à ce qu'elle appelle **`PutRecords`** dans Kinesis

1. Choisissez la ressource /records, puis choisissez Create method.
2. Pour Type de méthode, sélectionnez PUT.
3. Pour Type d'intégration, sélectionnez Service AWS .
4. Pour Région AWS, sélectionnez l' Région AWS endroit où vous avez créé votre flux Kinesis.
5. Pour Service AWS, sélectionnez Kinesis.
6. Laissez Sous-domaine AWS vide.
7. Dans le champ HTTP Method, sélectionnez POST.
8. Pour Type d'action, choisissez Utiliser un nom d'action.
9. Pour Nom de l'action, saisissez **`PutRecords`**.
10. Pour Rôle d'exécution, saisissez l'ARN de votre rôle d'exécution.
11. Conservez la valeur par défaut de Transmettre pour Gestion de contenu.
12. Choisissez Créer une méthode.
13. Dans la section Requête d'intégration, ajoutez les Paramètres d'en-tête de requête d'URL suivants :

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

    La tâche suit la même procédure pour configurer le mappage de paramètre de demande pour la méthode `GET /streams`.
14. Ajoutez le modèle de mappage de corps suivant pour mapper des données à partir de la requête de méthode `PUT /streams/{stream-name}/records` à la requête d'intégration correspondante de `POST /?Action=PutRecords` :

    ```
    {
        "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
        ]
    ```

```
}
```

Ce modèle de mappage suppose que la charge utile de la demande de méthode puisse être modélisée par le schéma JSON suivant :

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

Vous pouvez créer un modèle pour inclure ce schéma et utiliser ce modèle pour faciliter la génération du modèle de mappage. Cependant, vous pouvez également générer un modèle de mappage sans utiliser aucun modèle.

Dans ce didacticiel, nous avons utilisé deux formats de charge utile légèrement différents pour illustrer qu'un développeur d'API peut choisir d'exposer le format de données principal au client ou de lui cacher. Un format concerne la méthode PUT `/streams/{stream-name}/records` (ci-dessus). Un autre format est utilisé pour la méthode PUT `/streams/{stream-name}/record` (dans la procédure précédente). Dans un environnement de production, vous devez maintenir la cohérence des deux formats.

15. Pour tester la méthode PUT `/streams/{stream-name}/records`, définissez la variable de chemin `stream-name` sur un flux existant, fournissez la charge utile suivante, puis envoyez la demande de méthode.

```
{
```

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

Si le test aboutit, le résultat est une réponse 200 OK avec une charge utile semblable à la sortie suivante :

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

Pour configurer et tester la méthode **`GET /streams/{stream-name}/sharditerator`**, appelez **`GetShardIterator`** dans Kinesis

La méthode `GET /streams/{stream-name}/sharditerator` est une méthode d'assistance permettant d'acquérir un itérateur de partition requis avant d'appeler la méthode `GET /streams/{stream-name}/records`.

1. Choisissez la ressource /sharditerator, puis choisissez Create method.
2. Pour Type de méthode, sélectionnez GET.
3. Pour Type d’intégration, sélectionnez Service AWS .

4. Pour Région AWS, sélectionnez l' Région AWS endroit où vous avez créé votre flux Kinesis.
5. Pour Service AWS, sélectionnez Kinesis.
6. Laissez Sous-domaine AWS vide.
7. Dans le champ HTTP Method, sélectionnez POST.
8. Pour Type d'action, choisissez Utiliser un nom d'action.
9. Pour Nom de l'action, saisissez **`GetShardIterator`**.
10. Pour Rôle d'exécution, saisissez l'ARN de votre rôle d'exécution.
11. Conservez la valeur par défaut de Transmettre pour Gestion de contenu.
12. Choisissez Paramètres de chaîne de requête d'URL.

    L'`GetShardIterator`action nécessite la saisie d'une ShardId valeur. Pour transmettre une valeur `ShardId` fournie par le client, nous ajoutons un paramètre de requête `shard-id` à la requête de méthode, comme indiqué dans l'étape suivante.
13. Sélectionnez Add query string (Ajouter une chaîne de requêtes).
14. Pour Name (Nom), saisissez **`shard-id`**.
15. Gardez Obligatoire et Mise en cache désactivés.
16. Choisissez Créer une méthode.
17. Dans la section Requête d'intégration, ajoutez le modèle de mappage pour générer l'entrée requise (`ShardId` et `StreamName`) pour l'action `GetShardIterator` à partir des paramètres `shard-id` et `stream-name` de la requête de méthode. En outre, le modèle de mappage définit également `ShardIteratorType` à `TRIM_HORIZON` en tant que valeur par défaut.

    ```
    {
        "ShardId": "$input.params('shard-id')",
        "ShardIteratorType": "TRIM_HORIZON",
        "StreamName": "$input.params('stream-name')"
    }
    ```
18. À l'aide de l'option Test de la console API Gateway, entrez un nom de flux existant en tant que valeur de variable `stream-name` Path (Chemin), définissez la `shard-id` chaîne de requête sur une valeur `ShardId` existante (par exemple, `shard-000000000004`) et choisissez Test.

    Si le test aboutit, la charge utile de la réponse est similaire à la sortie suivante :

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

```
}
```

Notez la valeur `ShardIterator`. Vous en avez besoin pour obtenir les enregistrements d'un flux.

Pour configurer et tester la méthode **`GET /streams/{stream-name}/records`** pour appeler l'action **`GetRecords`** dans Kinesis

1. Choisissez la ressource /records, puis choisissez Create method.
2. Pour Type de méthode, sélectionnez GET.
3. Pour Type d'intégration, sélectionnez Service AWS .
4. Pour Région AWS, sélectionnez l' Région AWS endroit où vous avez créé votre flux Kinesis.
5. Pour Service AWS, sélectionnez Kinesis.
6. Laissez Sous-domaine AWS vide.
7. Dans le champ HTTP Method, sélectionnez POST.
8. Pour Type d'action, choisissez Utiliser un nom d'action.
9. Pour Nom de l'action, saisissez **`GetRecords`**.
10. Pour Rôle d'exécution, saisissez l'ARN de votre rôle d'exécution.
11. Conservez la valeur par défaut de Transmettre pour Gestion de contenu.
12. Choisissez les en-têtes de requête HTTP.

    L'action `GetRecords` nécessite une valeur `ShardIterator` en entrée. Pour transmettre une `ShardIterator` valeur fournie par le client, nous ajoutons un paramètre d'`Shard-Iterator`en-tête à la demande de méthode.
13. Sélectionnez Add header.
14. Pour Name (Nom), saisissez **`Shard-Iterator`**.
15. Gardez Obligatoire et Mise en cache désactivés.
16. Choisissez Créer une méthode.
17. Dans la section Requête d'intégration, ajoutez le modèle de mappage suivant pour mapper la valeur du paramètre d'en-tête `Shard-Iterator` à la valeur de propriété `ShardIterator` de la charge utile JSON pour l'action `GetRecords` dans Kinesis.

```
{
    "ShardIterator": "$input.params('Shard-Iterator')"
```

```
}
```

18. À l'aide de l'option Tester dans la console API Gateway, saisissez un nom de flux existant comme valeur de variable `stream-name` Chemin, définissez l'en-tête `Shard-Iterator` sur la valeur `ShardIterator` obtenue à partir de l'exécution de test de la méthode `GET /streams/{stream-name}/sharditerator` (ci-dessus) et choisissez Tester.

    Si le test aboutit, la charge utile de la réponse est similaire à la sortie suivante :

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

Définitions OpenAPI d'un exemple d'API en tant que proxy Kinesis

Vous trouverez ci-après des définitions OpenAPI pour l'exemple d'API en tant que proxy Kinesis utilisé dans ce tutoriel.

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"
              }
            }
          }
        }
      }
```

```
    }
  }
}
```

Tutoriel : Création d'une API optimisée pour les périphériques à l'aide AWS de SDK ou AWS CLI

Le didacticiel suivant montre comment créer une PetStore API prenant en charge les `GET /pets/{petId}` méthodes `GET /pets` et. Les méthodes sont intégrées à un point de terminaison HTTP. Vous pouvez suivre ce didacticiel en utilisant le AWS SDK pour JavaScript, le SDK pour Python (Boto3) ou le. AWS CLI Vous utilisez les fonctions ou commandes suivantes pour configurer votre API :

JavaScript v3

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

Python

- create_rest_api
- créer_ressource
- méthode de saisie
- put_method_response
- put_integration
- put_integration_response
- créer_déploiement

AWS CLI

- create-rest-api
- créer une ressource
- méthode put
- put-method-response
- intégration de PUT
- put-integration-response
- créer-déploiement

Pour plus d'informations sur le AWS SDK pour la JavaScript version 3, voir À quoi sert le AWS SDK ? JavaScript . Pour plus d'informations sur le SDK pour Python (Boto3), consultez. AWS SDK for Python (Boto3) Pour plus d'informations sur le AWS CLI, voir Qu'est-ce que le AWS CLI ? .

Configuration d'une API optimisée pour les périphériques PetStore

Dans ce didacticiel, les exemples de commandes utilisent des valeurs d'espace réservé pour les ID de valeur tels que l'ID d'API et l'ID de ressource. Au fur et à mesure que vous avez terminé le didacticiel, remplacez ces valeurs par les vôtres.

Pour configurer une PetStore API optimisée pour les périphériques à l'aide de SDK AWS

1. Utilisez l'exemple suivant pour créer une `RestApi` entité :

   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)
}
})();
```

Un appel réussi renvoie votre ID d'API et l'ID de ressource racine de votre API dans une sortie similaire à la suivante :

```
{
  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)
```

Un appel réussi renvoie votre ID d'API et l'ID de ressource racine de votre API dans une sortie similaire à la suivante :

```
{'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
```

Voici la sortie de cette commande :

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

L'API que vous avez créée possède un ID d'API `abcd1234` et un ID de ressource racine de`efg567`. Vous utilisez ces valeurs dans la configuration de votre API.

2. Ensuite, vous ajoutez une ressource enfant sous la racine, que vous spécifiez `RootResourceId` comme valeur de `parentId` propriété. Utilisez l'exemple suivant pour créer une `/pets` ressource pour votre API :

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)
}
})();
```

Un appel réussi renvoie des informations sur votre ressource sous forme de sortie comme suit :

```
{
    "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)
```

Un appel réussi renvoie des informations sur votre ressource sous forme de sortie comme suit :

```
{'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
```

Voici la sortie de cette commande :

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

La `/pets` ressource que vous avez créée possède un ID de ressource de`aaa111`. Vous utilisez cette valeur lors de la configuration de votre API.

3. Ensuite, vous ajoutez une ressource enfant sous la `/pets` ressource. Cette ressource `/{petId}` possède un paramètre de chemin pour le `{petId}` .Pour transformer une partie du chemin en paramètre de chemin, placez-la entre deux crochets,. `{ }` Utilisez l'exemple suivant pour créer une `/pets/{petId}` ressource pour votre API :

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)
}
})();
```

Un appel réussi renvoie des informations sur votre ressource sous forme de sortie comme suit :

```
{
    "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)
```

Un appel réussi renvoie des informations sur votre ressource sous forme de sortie comme suit :

```
{'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}'
```

Si elle aboutit, cette commande renvoie la réponse suivante :

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

La `/pets/{petId}` ressource que vous avez créée possède un ID de ressource de`bbb222`. Vous utilisez cette valeur lors de la configuration de votre API.

4. Au cours des deux étapes suivantes, vous allez ajouter des méthodes HTTP à vos ressources. Dans ce didacticiel, vous définissez les méthodes de libre accès en réglant la valeur `authorization-type` à`NONE`. Pour autoriser uniquement les utilisateurs authentifiés à appeler la méthode, vous pouvez utiliser des rôles et des stratégies IAM, un mécanisme d'autorisation

Lambda (anciennement appelé mécanisme Custom Authorizer) ou un groupe d'utilisateurs Amazon Cognito. Pour plus d'informations, consultez the section called "Contrôle d'accès".

L'exemple suivant ajoute la méthode `GET` HTTP à la `/pets` ressource :

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)
}
})();
```

Un appel réussi renvoie le résultat suivant :

```
{
    "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)
```

Un appel réussi renvoie le résultat suivant :

```
{'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
```

Voici un exemple de la sortie positive de cette commande :

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

5. L'exemple suivant ajoute la méthode GET HTTP à la `/pets/{petId}` ressource et définit la `requestParameters` propriété pour transmettre la `petId` valeur fournie par le client au backend :

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)
}
})();
```

Un appel réussi renvoie le résultat suivant :

```
{
    "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)
```

Un appel réussi renvoie le résultat suivant :

```
{'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
```

Voici un exemple de la sortie positive de cette commande :

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

```
}
```

6. Utilisez l'exemple suivant pour ajouter la réponse de la méthode 200 OK pour la `GET /pets` méthode :

   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)
   }
   })();
   ```

   Un appel réussi renvoie le résultat suivant :

   ```
   {
       "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)
```

Un appel réussi renvoie le résultat suivant :

```
{'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
```

Voici la sortie de cette commande :

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

7. Utilisez l'exemple suivant pour ajouter la réponse de la méthode 200 OK pour la `GET /pets/{petId}` méthode :

   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)
}
})();
```

Un appel réussi renvoie le résultat suivant :

```
{
    "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)
```

Un appel réussi renvoie le résultat suivant :

```
{'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
```

Voici la sortie de cette commande :

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

8. L'exemple suivant configure une intégration de la `GET /pets` méthode avec un point de terminaison HTTP. Le point de terminaison HTTP est`http://petstore-demo-endpoint.execute-api.com/petstore/pets`.

   JavaScript v3

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

```
})();
```

Un appel réussi renvoie le résultat suivant :

```
{
    "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)
```

Un appel réussi renvoie le résultat suivant :

```
{'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
```

Voici la sortie de cette commande :

```
{
    "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. L'exemple suivant configure une intégration de la `GET /pets/{petId}` méthode avec un point de terminaison HTTP. Le point de terminaison HTTP est`http://petstore-demo-endpoint.execute-api.com/petstore/pets/{id}`. Au cours de cette étape, vous mappez le paramètre de chemin `petId` au paramètre de chemin du point de terminaison d'intégration de`id`.

   JavaScript v3

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

```
    httpMethod: 'GET',
    type: 'HTTP',
    integrationHttpMethod: 'GET',
    uri: 'http://petstore-demo-endpoint.execute-api.com/petstore/pets/{id}'
    requestParameters: {
        "integration.request.path.id": "method.request.path.petId"
     }
});
try {
    const results = await apig.send(command)
    console.log(results)
} catch (err) {
    console.log("Set up the integration of the 'GET /pets/{petId}' method of the
 API failed:\n", err)
}
})();
```

Un appel réussi renvoie le résultat suivant :

```
{
    "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)
```

Un appel réussi renvoie le résultat suivant :

```
{'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
```

Voici la sortie de cette commande :

```
{
    "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. L'exemple suivant ajoute la réponse d'intégration pour l'`GET /pets`intégration :

    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)
}
})();
```

    Un appel réussi renvoie le résultat suivant :

```
{
    "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)
```

Un appel réussi renvoie le résultat suivant :

```
{'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
```

Voici la sortie de cette commande :

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

```
}
```

11. L'exemple suivant ajoute la réponse d'intégration pour l'`GET /pets/{petId}`intégration :

    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)
    }
    })();
    ```

    Un appel réussi renvoie le résultat suivant :

    ```
    {
        "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)
```

Un appel réussi renvoie le résultat suivant :

```
{'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
```

Voici la sortie de cette commande :

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

Une fois que vous avez créé la réponse d'intégration, votre API peut interroger les animaux de compagnie disponibles sur le PetStore site Web et consulter un animal individuel avec un identifiant spécifique. Avant que votre API ne soit appelable par vos clients, vous devez la déployer. Avant de déployer votre API, nous vous recommandons de la tester.

12. L'exemple suivant teste la `GET /pets` méthode :

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. L'exemple suivant teste la `GET /pets/{petId}` méthode avec une valeur `petId` de 3 :

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'
```

Après avoir testé avec succès votre API, vous pouvez la déployer sur une phase.

14. L'exemple suivant déploie votre API sur une étape nomméetest. Lorsque vous déployez votre API sur une étape, les appelants peuvent invoquer votre API.

JavaScript v3

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

Python

```
import botocore
import boto3
import logging

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

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

AWS CLI

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

Voici la sortie de cette commande :

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

Votre API est désormais appelable par les clients. Vous pouvez tester cette API en saisissant l'`https://abcd1234.execute-api.us-west-2.amazonaws.com/test/pets`URL dans un navigateur et en la `abcd1234` remplaçant par l'identifiant de votre API.

Pour d'autres exemples de création ou de mise à jour d'une API à l'aide de AWS SDK ou du AWS CLI, consultez la section Actions pour API Gateway à l'aide de AWS SDK.

Automatisez la configuration de votre API

Au lieu de créer votre API step-by-step, vous pouvez automatiser la création et le nettoyage des AWS ressources en utilisant OpenAPI AWS CloudFormation ou Terraform pour créer votre API.

Définition d'OpenAPI 3.0

Vous pouvez importer une définition d'OpenAPI dans API Gateway. Pour plus d'informations, consultez the section called "OpenAPI".

```
{
  "openapi" : "3.0.1",
  "info" : {
    "title" : "Simple PetStore (OpenAPI)",
    "description" : "Demo API created using OpenAPI",
    "version" : "2024-05-24T20:39:34Z"
  },
  "servers" : [ {
    "url" : "{basePath}",
    "variables" : {
      "basePath" : {
        "default" : "Prod"
      }
    }
  } ],
  "paths" : {
    "/pets" : {
      "get" : {
        "responses" : {
          "200" : {
            "description" : "200 response",
            "content" : { }
          }
        },
        "x-amazon-apigateway-integration" : {
          "type" : "http",
          "httpMethod" : "GET",
          "uri" : "http://petstore-demo-endpoint.execute-api.com/petstore/pets",
          "responses" : {
            "default" : {
```

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

```
}
```

AWS CloudFormation modèle

Pour déployer votre AWS CloudFormation modèle, consultez la section Création d'une pile sur la AWS CloudFormation console.

```
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'
```

Configuration de Terraform

Pour plus d'informations sur Terraform, consultez Terraform.

```
provider "aws" {
  region = "us-east-1" # Update with your desired region
}
resource "aws_api_gateway_rest_api" "Api" {
  name        = "Simple PetStore (Terraform)"
  description = "Demo API created using Terraform"
}
resource "aws_api_gateway_resource" "petsResource"{
    rest_api_id = aws_api_gateway_rest_api.Api.id
    parent_id = aws_api_gateway_rest_api.Api.root_resource_id
    path_part = "pets"
}
```

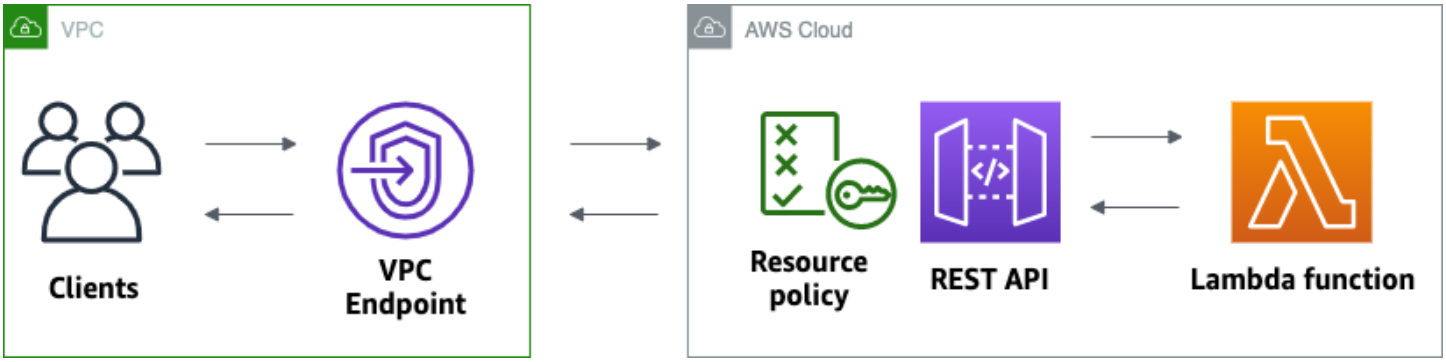
```
resource "aws_api_gateway_resource" "petIdResource"{
    rest_api_id = aws_api_gateway_rest_api.Api.id
    parent_id = aws_api_gateway_resource.petsResource.id
    path_part = "{petId}"
}
resource "aws_api_gateway_method" "petsMethodGet" {
  rest_api_id   = aws_api_gateway_rest_api.Api.id
  resource_id   = aws_api_gateway_resource.petsResource.id
  http_method   = "GET"
  authorization = "NONE"
}


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

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

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

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

resource "aws_api_gateway_method" "petIdMethodGet" {
    rest_api_id   = aws_api_gateway_rest_api.Api.id
    resource_id   = aws_api_gateway_resource.petIdResource.id
    http_method   = "GET"
    authorization = "NONE"
```

```
    request_parameters = {"method.request.path.petId" = true}
}

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


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

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


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

Tutoriel : Création d'une API REST privée

Dans ce tutoriel, vous créez une API REST privée. Les clients peuvent accéder à l'API uniquement à partir de votre Amazon VPC. L'API est isolée de l'Internet public, ce qui est une exigence de sécurité commune.

Ce didacticiel vous prendra environ 30 minutes. Tout d'abord, vous utilisez un AWS CloudFormation modèle pour créer un Amazon VPC, un point de terminaison VPC, une AWS Lambda fonction, puis vous lancez une instance Amazon EC2 que vous utiliserez pour tester votre API. Ensuite, vous l'utilisez AWS Management Console pour créer une API privée et y associer une politique de ressources qui autorise l'accès uniquement depuis votre point de terminaison VPC. Enfin, vous testez votre API.



Pour suivre ce didacticiel, vous avez besoin d'un AWS compte et d'un AWS Identity and Access Management utilisateur disposant d'un accès à la console. Pour plus d'informations, consultez Prérequis.

Dans ce tutoriel, vous utiliserez l AWS Management Console. Pour un AWS CloudFormation modèle qui crée cette API et toutes les ressources associées, consultez template.yaml.

Rubriques

- Étape 1 : création de dépendances
- Étape 2 : Créer une API privée
- Étape 3 : création d'une méthode et intégration
- Étape 4 : attacher une stratégie de ressources
- Étape 5 : déploiement de votre API
- Étape 6 : vérifiez que votre API n'est pas accessible publiquement
- Étape 7 : connectez-vous à une instance dans votre VPC et appelez votre API
- Étape 8 : Nettoyage

- Prochaines étapes : Automatisez avec AWS CloudFormation

Étape 1 : création de dépendances

Téléchargez et décompressez ce AWS CloudFormation modèle. Vous utilisez le modèle pour créer toutes les dépendances de votre API privée, y compris un Amazon VPC, un point de terminaison d'un VPC et une fonction Lambda qui sert de backend de votre API. Vous créerez l'API privée ultérieurement.

Pour créer une AWS CloudFormation pile

1. Ouvrez la AWS CloudFormation console à l'adresse https://console.aws.amazon.com/cloudformation.
2. Choisissez Créer une pile, puis choisissez Avec de nouvelles ressources (standard).
3. Dans Spécifier le modèle, choisissez Charger un modèle de fichier.
4. Sélectionnez le modèle que vous avez téléchargé.
5. Choisissez Suivant.
6. Pour Nom de la pile, saisissez **`private-api-tutorial`**, puis choisissez Suivant.
7. Pour Configurer les options de pile, choisissez Suivant.
8. Pour les fonctionnalités, reconnaissez que AWS CloudFormation vous pouvez créer des ressources IAM dans votre compte.
9. Sélectionnez Envoyer.

AWS CloudFormation fournit les dépendances de votre API, ce qui peut prendre quelques minutes. Lorsque le statut de votre AWS CloudFormation pile est CREATE_COMPLETE, choisissez Outputs. Notez votre ID de point de terminaison d'un VPC. Vous en avez besoin pour les étapes ultérieures de ce tutoriel.

Étape 2 : Créer une API privée

Vous créez une API privée pour autoriser uniquement les clients de votre VPC à y accéder.

Pour créer une API privée

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.

2. Choisissez Créer une API, puis pour API REST, choisissez Créer.
3. Sous API name (Nom de l'API), saisissez **`private-api-tutorial`**.
4. Pour Type de point de terminaison d'API, sélectionnez Privé.
5. Pour les ID de point de terminaison VPC, entrez l'ID de point de terminaison VPC dans les sorties de votre pile. AWS CloudFormation
6. Sélectionnez Create API (Créer une API).

Étape 3 : création d'une méthode et intégration

Vous créez une méthode `GET` et l'intégration Lambda pour gérer les demandes `GET` à votre API. Lorsqu'un client appelle votre API, API Gateway envoie la demande à la fonction Lambda que vous avez créée à l'étape 1, puis renvoie une réponse au client.

Pour créer une méthode et une intégration

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Choisissez votre API.
3. Sélectionnez la ressource /, puis choisissez Créer une méthode.
4. Pour Type de méthode, sélectionnez `GET`.
5. Pour Type d'intégration, sélectionnez Fonction Lambda.
6. Activez Intégration de proxy Lambda. Avec une intégration de proxy Lambda, API Gateway envoie un événement à Lambda avec une structure définie, et transforme la réponse de votre fonction Lambda en réponse HTTP.
7. Pour la fonction Lambda, choisissez la fonction que vous avez créée avec le AWS CloudFormation modèle à l'étape 1. Le nom de la fonction commence par **`private-api-tutorial`**.
8. Choisissez Créer une méthode.

Étape 4 : attacher une stratégie de ressources

Vous attachez une stratégie de ressources à votre API qui permet aux clients d'appeler votre API uniquement via votre point de terminaison d'un VPC. Pour restreindre davantage l'accès à votre API, vous pouvez également configurer une stratégie de point de terminaison d'un VPC pour votre point de terminaison d'un VPC, ce n'est cependant pas nécessaire pour ce didacticiel.

Pour attacher une stratégie de ressources

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Choisissez votre API.
3. Choisissez Stratégie de ressources, puis Créer une stratégie.
4. Saisissez la stratégie suivante. Remplacez *VPCEid* par votre identifiant de point de terminaison VPC dans les sorties de votre stack. AWS CloudFormation

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

5. Sélectionnez Enregistrer les modifications.

Étape 5 : déploiement de votre API

Ensuite, vous déployez votre API pour la rendre disponible pour les clients de votre Amazon VPC.

Pour déployer une API

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Choisissez votre API.
3. Sélectionnez Deploy API (Déployer une API).
4. Pour Étape, sélectionnez Nouvelle étape.
5. Sous Stage name (Nom de l'étape), entrez **`test`**.
6. (Facultatif) Sous Description, entrez une description.
7. Choisissez Deploy (Déployer).

Vous êtes désormais prêt à tester votre API.

Étape 6 : vérifiez que votre API n'est pas accessible publiquement

Utilisez `curl` pour vérifier que vous ne pouvez pas appeler votre API depuis l'extérieur de votre Amazon VPC.

Pour tester votre API

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Choisissez votre API.
3. Dans le volet de navigation principal, choisissez Étapes, puis choisissez l’étape de test.
4. Sous Détails de l'étape, choisissez l'icône de copie pour copier l'URL d'invocation de votre API. L'URL ressemble à `https://abcdef123.execute-api.us-west-2.amazonaws.com/test`. Le point de terminaison d'un VPC que vous avez créé à l'étape 1 a un DNS privé activé de sorte que vous pouvez utiliser l'URL fournie pour appeler votre API.
5. Utilisez curl pour tenter d'appeler votre API depuis l'extérieur de votre VPC.

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

   Curl indique que le point de terminaison de votre API ne peut pas être résolu. Si vous obtenez une réponse différente, revenez à l'étape 2 et assurez-vous que vous choisissez Privé pour le type de point de terminaison de votre API.

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

Ensuite, vous vous connectez à une instance Amazon EC2 dans votre VPC pour appeler votre API.

Étape 7 : connectez-vous à une instance dans votre VPC et appelez votre API

Ensuite, vous testez votre API depuis votre Amazon VPC. Pour accéder à votre API privée, vous vous connectez à une instance Amazon EC2 dans votre VPC, puis utilisez curl pour appeler votre API. Vous utilisez le Gestionnaire de session Systems Manager pour vous connecter à votre instance dans le navigateur.

Pour tester votre API

1. Ouvrez la console Amazon EC2 à l'adresse https://console.aws.amazon.com/ec2/.
2. Choisissez Instances.
3. Choisissez l'instance nommée private-api-tutorialque vous avez créée avec le AWS CloudFormation modèle à l'étape 1.
4. Choisissez Connecter, puis Session Manager.
5. Choisissez Connecter pour lancer une session basée sur un navigateur vers votre instance.
6. Dans votre session Session Manager, utilisez curl pour appeler votre API. Vous pouvez appeler votre API car vous utilisez une instance dans votre Amazon VPC.

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

   Vérifiez que vous obtenez la réponse `Hello from Lambda!`.

Vous avez créé avec succès une API accessible uniquement à partir de votre Amazon VPC, puis vous avez vérifié qu'elle fonctionne.

Étape 8 : Nettoyage

Pour éviter des coûts inutiles, supprimez les ressources que vous avez créées dans le cadre de ce tutoriel. Les étapes suivantes suppriment votre API REST et votre AWS CloudFormation stack.

Pour supprimer une API REST

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Sur la page APIs (API) , sélectionnez une API. Choisissez Actions d'API, choisissez Supprimer l'API, puis confirmez votre choix.

Pour supprimer une AWS CloudFormation pile

1. Ouvrez la AWS CloudFormation console à l'adresse https://console.aws.amazon.com/cloudformation.
2. Sélectionnez votre AWS CloudFormation pile.
3. Choisissez Supprimer, puis confirmez votre choix.

Prochaines étapes : Automatisez avec AWS CloudFormation

Vous pouvez automatiser la création et le nettoyage de toutes les AWS ressources impliquées dans ce didacticiel. Pour un exemple complet de modèle AWS CloudFormation , veuillez consulter template.yaml.

Didacticiels sur l'API HTTP Amazon API Gateway

Les didacticiels suivants proposent des exercices pratiques qui vous permettent de découvrir les API HTTP API Gateway.

Rubriques

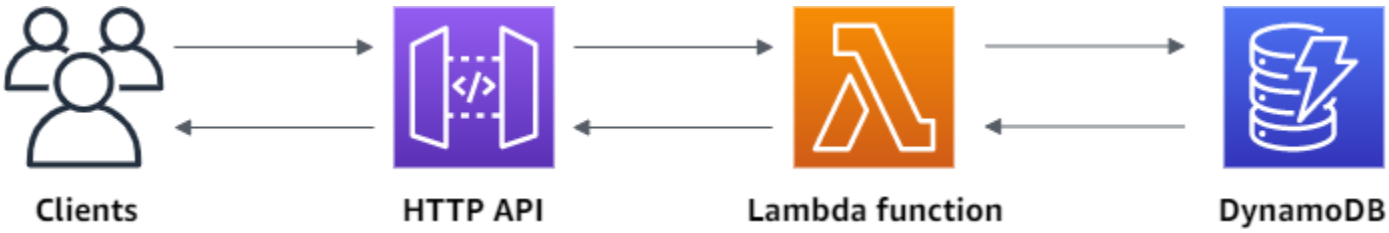
- Didacticiel : création d'une API CRUD avec Lambda et DynamoDB
- Tutoriel : création d'une API HTTP avec une intégration privée à un Amazon ECS service

Didacticiel : création d'une API CRUD avec Lambda et DynamoDB

Dans ce didacticiel, vous créez une API serverless qui crée, lit, met à jour et supprime des éléments d'une table DynamoDB. DynamoDB est un service de base de données NoSQL entièrement géré offrant des performances rapides et prévisibles avec une scalabilité simple. Ce tutoriel dure environ 30 minutes, et vous pouvez le suivre dans l'offre gratuite AWS.

Tout d'abord, vous créez une table DynamoDB à l'aide de la console DynamoDB. Vous créez ensuite une fonction Lambda à l'aide de la AWS Lambda console. Ensuite, vous créez une API HTTP à l'aide de la console API Gateway. Enfin, vous testez votre API.

Lorsque vous appelez votre API HTTP, API Gateway achemine la requête vers votre fonction Lambda. La fonction Lambda interagit avec DynamoDB et renvoie une réponse à API Gateway. API Gateway vous renvoie ensuite une réponse.



Pour effectuer cet exercice, vous avez besoin d'un AWS compte et d'un AWS Identity and Access Management utilisateur disposant d'un accès à la console. Pour plus d'informations, consultez Prérequis.

Dans ce tutoriel, vous utiliserez l AWS Management Console. Pour un AWS SAM modèle qui crée cette API et toutes les ressources associées, consultez `template.yaml`.

Rubriques

- Étape 1 : création d'une table DynamoDB

Étape 1 : création d'une table DynamoDB

Vous utilisez une table DynamoDB afin de stocker les données de votre API.

Chaque élément a un ID unique, que nous utilisons en tant que clé de partition pour la table.

Pour créer une table DynamoDB

1. Ouvrez la console DynamoDB à l'adresse https://console.aws.amazon.com/dynamodb/.
2. Choisissez Create table (Créer une table).
3. Sous Table name (Nom de la table), saisissez **`http-crud-tutorial-items`**.
4. Pour Clé de partition, saisissez **`id`**.
5. Choisissez Créer un tableau.

Étape 2 : création d'une fonction Lambda

Vous utilisez une fonction Lambda pour le backend de votre API. Cette fonction Lambda crée, lit, met à jour et supprime des éléments de DynamoDB. La fonction utilise des événements d'API Gateway afin de déterminer l'interaction avec DynamoDB. Pour plus de simplicité, ce didacticiel utilise une seule fonction Lambda. Il est recommandé de créer des fonctions distinctes pour chaque route.

Pour créer une fonction Lambda

1. Connectez-vous à la console Lambda à l'adresse https://console.aws.amazon.com/lambda/.
2. Choisissez Créer une fonction.
3. Sous Function name (Nom de la fonction), saisissez **`http-crud-tutorial-function`**.

4. Pour Exécution, choisissez le dernier environnement d'exécution Node.js ou Python compatible.
5. Sous Permissions (Autorisations), choisissez Change default execution role (Modifier le rôle d'exécution par défaut).
6. Sélectionnez Créer un nouveau rôle à partir de modèles de AWS politique.
7. Sous Role name (Nom du rôle), saisissez **`http-crud-tutorial-role`**.
8. Sous Policy templates (Modèles de stratégie), choisissez **`Simple microservice permissions`**. Cette stratégie accorde à la fonction Lambda l'autorisation d'interagir avec DynamoDB.

   > **Note**
   >
   > Ce didacticiel utilise une stratégie gérée pour plus de simplicité. Il est recommandé de créer votre propre stratégie IAM afin d'accorder les autorisations minimales requises.

9. Choisissez Créer une fonction.
10. Ouvrez la fonction Lambda dans l'éditeur de code de la console et remplacez son contenu par le code suivant. Choisissez Deploy (Déployer) afin de mettre à jour votre fonction.

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
```

Étape 3 : création d'une API HTTP

L'API HTTP fournit un point de terminaison HTTP pour votre fonction Lambda. Au cours de cette étape, vous créez une API vide. Dans les étapes suivantes, vous configurez des itinéraires et des intégrations afin de connecter votre API et votre fonction Lambda.

Pour créer une API HTTP

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Choisissez Create API (Créer une API). Ensuite, sous HTTP API (API HTTP), choisissez Build (Créer).
3. Sous API name (Nom de l'API), saisissez **`http-crud-tutorial-api`**.
4. Choisissez Suivant.
5. Sous Configure routes (Configurer les itinéraires), choisissez Next (Suivant) afin d'ignorer la création d'itinéraires. Vous créerez des itinéraires ultérieurement.
6. Consultez l'étape créée par API Gateway pour vous, puis choisissez Next (Suivant).
7. Choisissez Créer.

Étape 4 : création d'itinéraires

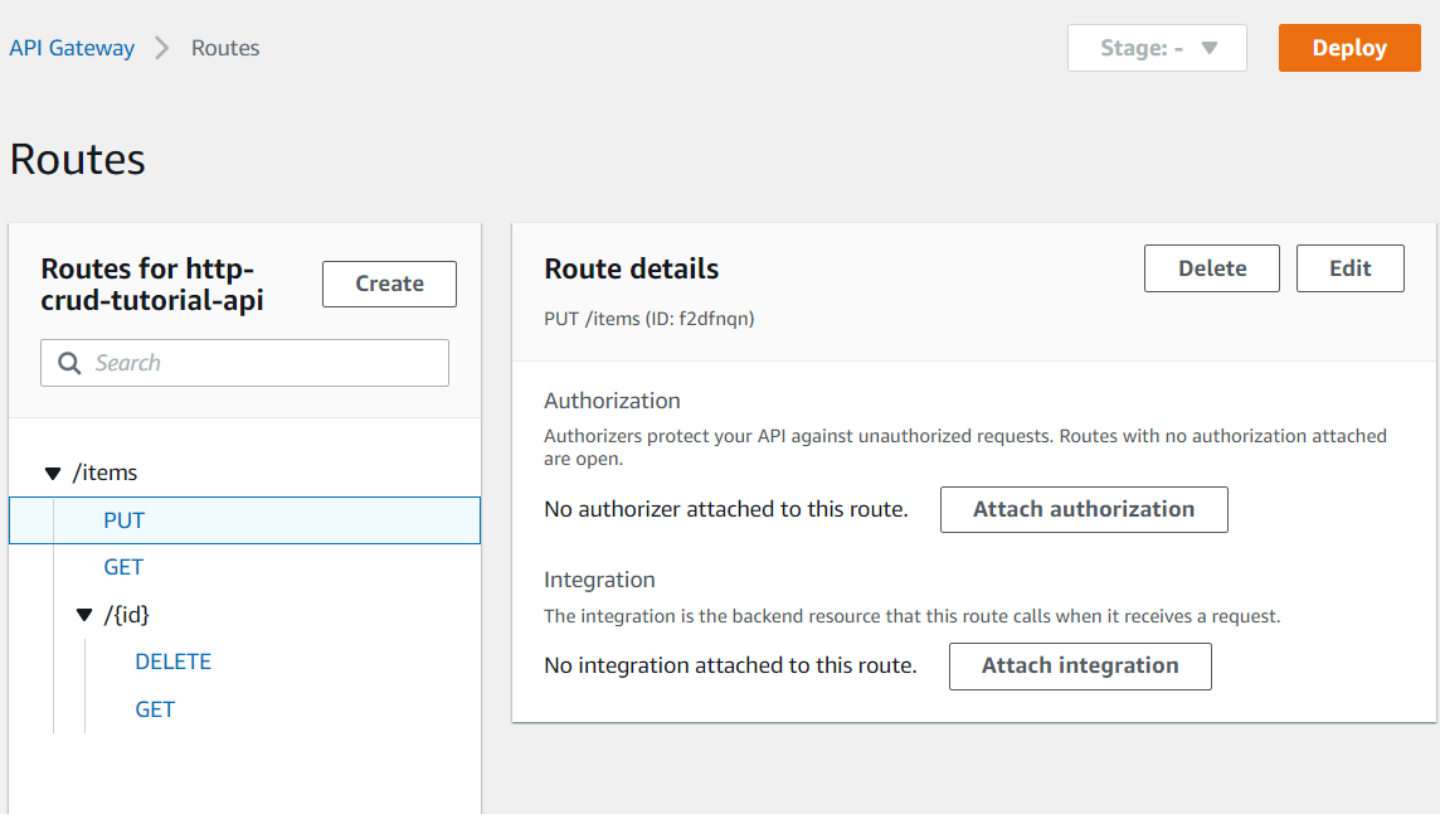
Les itinéraires permettent d'envoyer les demandes d'API entrantes aux ressources backend. Les itinéraires se composent de deux parties : une méthode HTTP et un chemin de ressource (par exemple,, `GET /items`. Pour cet exemple d'API, nous créons quatre itinéraires :

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

Pour créer des itinéraires

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Choisissez votre API.

3. Choisissez Itinéraires.
4. Choisissez Créer.
5. Pour Méthode, choisissez **`GET`**.
6. Pour le chemin, saisissez **`/items/{id}`**. Le `{id}` à la fin du chemin est un paramètre de chemin d'accès qu'API Gateway extrait du chemin d'accès de la demande lorsqu'un client effectue une demande.
7. Choisissez Créer.
8. Répétez les étapes 4 à 7 pour `GET /items`, `DELETE /items/{id}` et `PUT /items`.



Étape 5 : création d'une intégration

Vous créez une intégration afin de connecter un itinéraire aux ressources backend. Pour cet exemple d'API, vous créez une intégration Lambda que vous utilisez pour tous les itinéraires.

Pour créer une intégration

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.

2. Choisissez votre API.
3. Choisissez Integrations (Intégrations).
4. Choisissez Manage integrations (Gérer les intégrations), puis Create (Créer).
5. Ignorez Attach this integration to a route (Attacher cette intégration à un itinéraire). Vous vous en occuperez ultérieurement.
6. Sous Integration type (Type d'intégration), choisissez Lambda function (Fonction Lambda).
7. Sous Lambda function (Fonction Lambda), saisissez **`http-crud-tutorial-function`**.
8. Choisissez Créer.

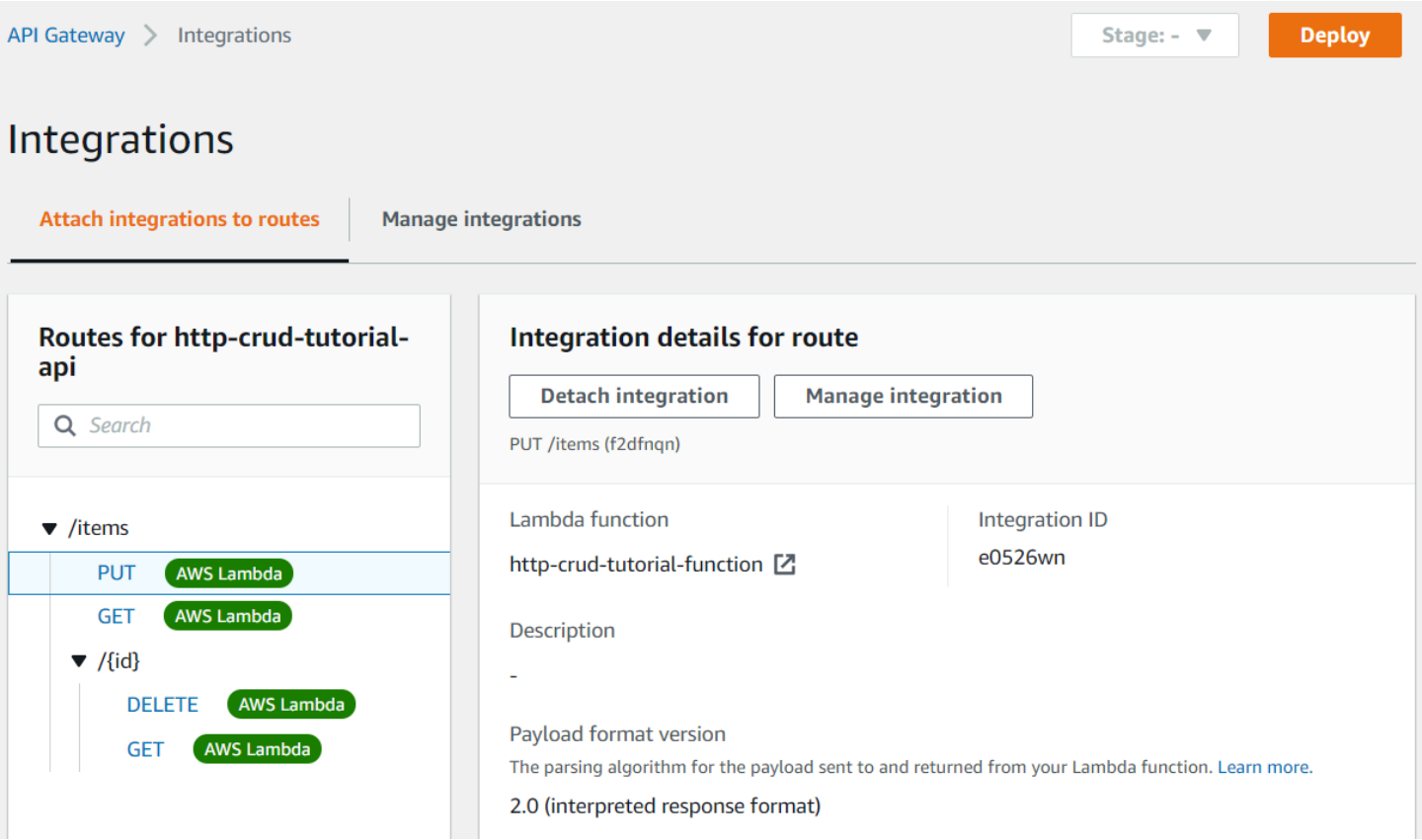
Étape 6 : attachement de votre intégration aux itinéraires

Pour cet exemple d'API, vous utilisez la même intégration Lambda pour tous les itinéraires. Après que l'intégration a été attachée à l'ensemble des itinéraires de l'API, votre fonction Lambda est appelée lorsqu'un client appelle l'un de vos itinéraires.

Pour attacher des intégrations à des itinéraires

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Choisissez votre API.
3. Choisissez Integrations (Intégrations).
4. Choisissez un itinéraire.
5. Sous Choose an existing integration (Choisir une intégration existante), choisissez **`http-crud-tutorial-function`**.
6. Choisissez Attach integration (Attacher l'intégration).
7. Répétez les étapes 4 à 6 pour tous les itinéraires.

Tous les itinéraires indiquent qu'une AWS Lambda intégration est attachée.



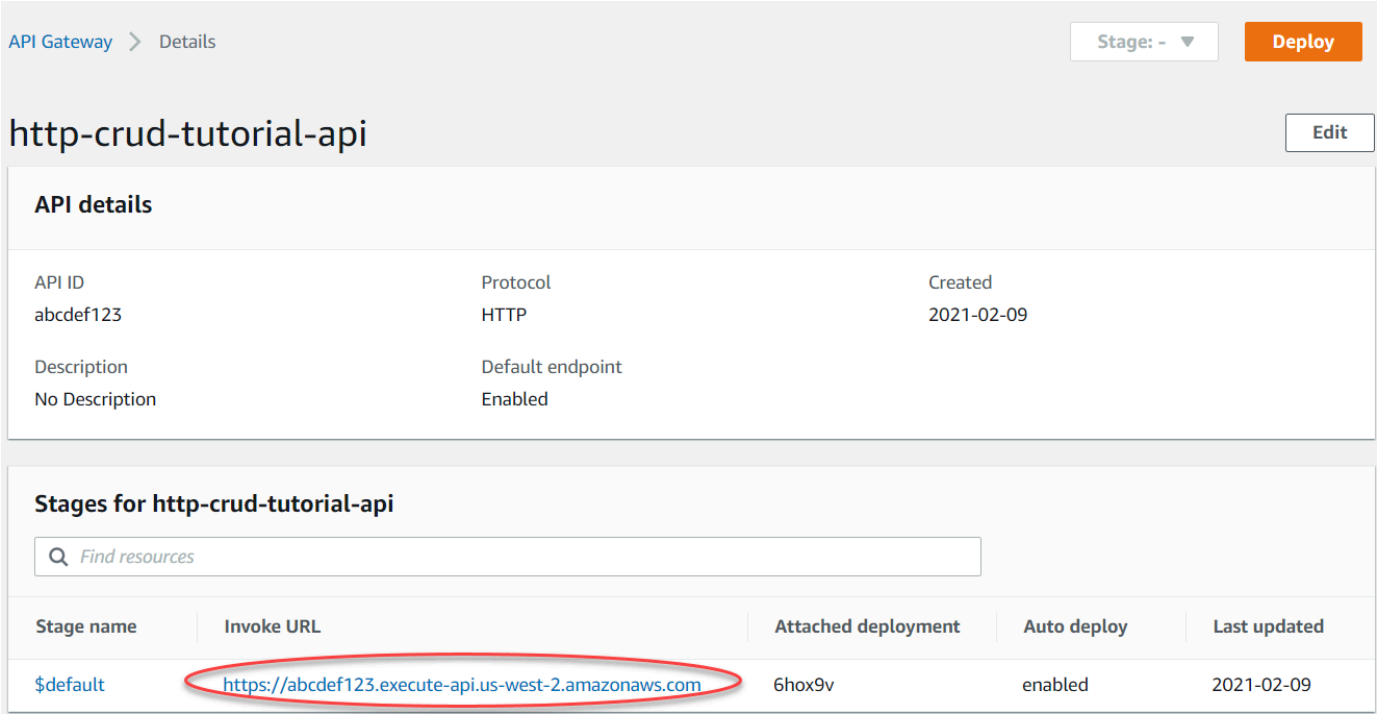
Maintenant que vous disposez d'une API HTTP avec des itinéraires et des intégrations, vous pouvez tester votre API.

Étape 7 : test de votre API

Afin de vous assurer que votre API fonctionne, vous utilisez curl.

Pour obtenir l'URL pour appeler votre API

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Choisissez votre API.
3. Notez l'URL d'appel de votre API. Elle apparaît sous Invoke URL (Appeler l'URL) sur la page Details (Détails).



4. Copiez l'URL d'appel de votre API.

   L'URL complète se présente sous la forme `https://abcdef123.execute-api.us-west-2.amazonaws.com`.

Pour créer ou mettre à jour un élément

- Utilisez la commande suivante afin de créer ou de mettre à jour un élément. La commande inclut un corps de demande et l'ID, le prix et le nom de l'élément.

```
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
```

Pour obtenir tous les éléments

- Utilisez la commande suivante afin de répertorier tous les éléments.

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

Pour obtenir un élément

- Utilisez la commande suivante afin d'obtenir un élément par son ID.

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

Pour supprimer un article

1. Utilisez la commande suivante afin de supprimer un élément.

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

2. Obtenez tous les éléments afin de vérifier que l'élément a été supprimé.

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

Étape 8 : Nettoyage

Pour éviter des coûts inutiles, supprimez les ressources que vous avez créées dans le cadre de cet exercice de démarrage. Les étapes suivantes suppriment votre API HTTP, votre fonction Lambda, ainsi que les ressources associées.

Pour supprimer une table DynamoDB

1. Ouvrez la console DynamoDB à l'adresse https://console.aws.amazon.com/dynamodb/.
2. Sélectionnez votre table.
3. Choisissez Supprimer la table.
4. Confirmez votre choix et choisissez Delete (Supprimer).

Pour supprimer une API HTTP

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Sur la page APIs (API) , sélectionnez une API. Choisissez Actions, puis Supprimer.
3. Sélectionnez Delete.

Pour supprimer une fonction Lambda

1. Connectez-vous à la console Lambda à l'adresse https://console.aws.amazon.com/lambda/.
2. Sur la page Functions (Fonctions), sélectionnez une fonction. Choisissez Actions, puis Supprimer.
3. Sélectionnez Delete.

Pour supprimer le groupe de journaux d'une fonction Lambda

1. Dans la CloudWatch console Amazon, ouvrez la page Log groups.
2. Sur la page Log groups (Groupes de journaux), sélectionnez le groupe de journaux de la fonction (`/aws/lambda/http-crud-tutorial-function`). Choisissez Actions, puis Supprimer le groupe de journaux.
3. Sélectionnez Delete.

Pour supprimer le rôle d'exécution d'une fonction Lambda

1. Dans la AWS Identity and Access Management console, ouvrez la page Rôles.
2. Sélectionnez le rôle de la fonction, par exempl, `http-crud-tutorial-role`.
3. Choisissez Supprimer le rôle.
4. Choisissez Oui, supprimer.

Prochaines étapes : Automatiser avec AWS SAM ou AWS CloudFormation

Vous pouvez automatiser la création et le nettoyage des AWS ressources en utilisant AWS CloudFormation ou AWS SAM. Afin d'obtenir un exemple de modèle AWS SAM pour ce tutoriel, veuillez consulter `template.yaml`.

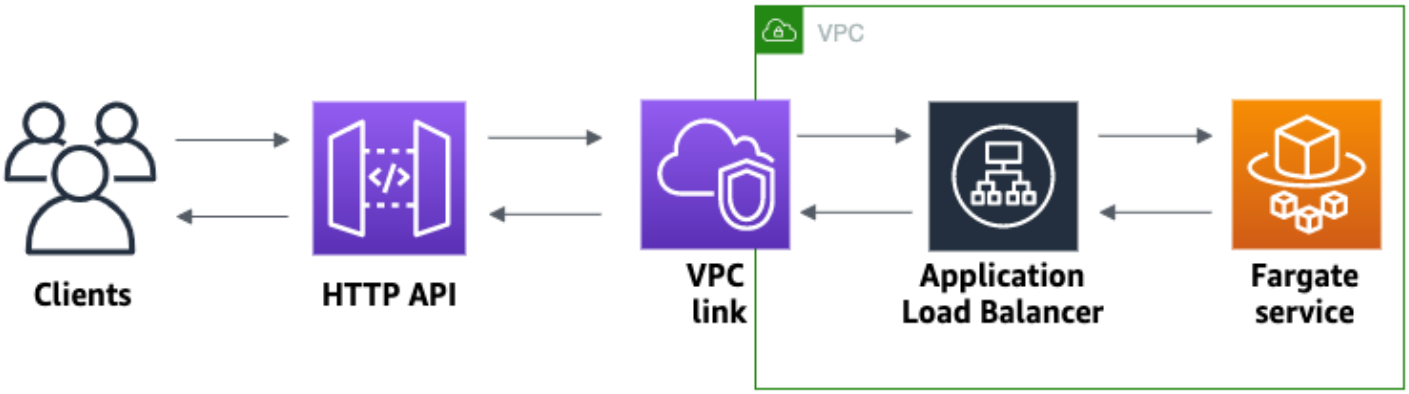
Pour des exemples AWS CloudFormation de modèles, voir les exemples AWS CloudFormation de modèles.

Tutoriel : création d'une API HTTP avec une intégration privée à un Amazon ECS service

Dans ce tutoriel, vous créerez une API sans serveur qui se connecte à un Amazon ECS service qui s'exécute dans un VPC Amazon. Les clients en dehors de votre VPC Amazon peuvent utiliser l'API pour accéder à votre Amazon ECS service.

Ce tutoriel vous prendra environ 1 heure. Tout d'abord, vous utilisez un AWS CloudFormation modèle pour créer un service Amazon VPC et Amazon ECS. Ensuite, vous utilisez la console API Gateway pour créer un lien VPC. Le lien VPC permet à API Gateway d'accéder à l'Amazon ECS service qui s'exécute dans votre VPC Amazon. Ensuite, vous créez une API HTTP qui utilise le lien VPC pour vous connecter à votre Amazon ECS service. Enfin, vous testez votre API.

Lorsque vous appelez votre API HTTP, API Gateway achemine la demande vers votre Amazon ECS service via votre lien VPC, puis renvoie la réponse du service.



Pour suivre ce didacticiel, vous avez besoin d'un AWS compte et d'un AWS Identity and Access Management utilisateur disposant d'un accès à la console. Pour plus d'informations, consultez Prérequis.

Dans ce tutoriel, vous utiliserez l AWS Management Console. Pour un AWS CloudFormation modèle qui crée cette API et toutes les ressources associées, consultez template.yaml.

Rubriques

- Étape 1 : créer un service Amazon ECS service
- Étape 2 : création d'un lien VPC
- Étape 3 : création d'une API HTTP
- Étape 4 : création d'une route

Étape 1 : créer un service Amazon ECS service

Amazon ECS est un service de gestion de conteneurs qui facilite l'exécution, l'arrêt et la gestion des conteneurs Docker sur un cluster. Dans ce tutoriel, vous exécuterez votre cluster sur une infrastructure sans serveur gérée par Amazon ECS.

Téléchargez et décompressez ce AWS CloudFormation modèle, qui crée toutes les dépendances du service, y compris un Amazon VPC. Vous utilisez le modèle pour créer un Amazon ECS service qui utilise un Application Load Balancer.

Pour créer une AWS CloudFormation pile

1. Ouvrez la AWS CloudFormation console à l'adresse https://console.aws.amazon.com/cloudformation.
2. Choisissez Créer une pile, puis choisissez Avec de nouvelles ressources (standard).
3. Dans Spécifier le modèle, choisissez Charger un modèle de fichier.
4. Sélectionnez le modèle que vous avez téléchargé.
5. Choisissez Suivant.
6. Pour Nom de la pile, saisissez **`http-api-private-integrations-tutorial`**, puis choisissez Suivant.
7. Pour Configurer les options de pile, choisissez Suivant.
8. Pour les fonctionnalités, reconnaissez que AWS CloudFormation vous pouvez créer des ressources IAM dans votre compte.
9. Sélectionnez Envoyer.

AWS CloudFormation fournit le service ECS, ce qui peut prendre quelques minutes. Lorsque le statut de votre AWS CloudFormation pile est CREATE_COMPLETE, vous êtes prêt à passer à l'étape suivante.

Étape 2 : création d'un lien VPC

Un lien VPC permet à API Gateway d'accéder aux ressources privées dans un VPC Amazon. Vous utilisez un lien VPC pour permettre aux clients d'accéder à votre Amazon ECS service via votre API HTTP.

Pour créer un lien VPC

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Dans le volet de navigation principal, choisissez VPC links, puis Create.

   Vous devrez peut-être choisir l'icône du menu pour ouvrir le volet de navigation principal.
3. Pour Choisir une version de lien VPC, sélectionnez Lien VPC pour des API HTTP.
4. Pour Nom, saisissez **`private-integrations-tutorial`**.
5. Pour VPC, choisissez le VPC que vous avez créé à l'étape 1. Le nom doit commencer par PrivateIntegrationsStack.
6. Pour Sous-réseaux, sélectionnez les deux sous-réseaux privés de votre VPC. Leurs noms finissent par `PrivateSubnet`.
7. Choisissez Créer.

Après avoir créé votre lien VPC, API Gateway alloue aux interfaces réseau Elastic l'accès à votre VPC. Ce processus peut prendre quelques minutes. En attendant, vous pouvez créer votre API.

Étape 3 : création d'une API HTTP

L'API HTTP fournit un point de terminaison HTTP pour votre Amazon ECS service. Au cours de cette étape, vous créez une API vide. Dans les étapes 4 et 5, vous configurerez une route et une intégration pour connecter votre API et votre Amazon ECS service.

Pour créer une API HTTP

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Choisissez Create API (Créer une API). Ensuite, sous HTTP API (API HTTP), choisissez Build (Créer).

3. Sous API name (Nom de l'API), saisissez **`http-private-integrations-tutorial`**.
4. Choisissez Suivant.
5. Sous Configure routes (Configurer les itinéraires), choisissez Next (Suivant) afin d'ignorer la création d'itinéraires. Vous créerez des itinéraires ultérieurement.
6. Passez en revue l'étape créée par API Gateway pour vous. API Gateway crée une étape `$default` avec les déploiements automatiques activés, ce qui constitue le meilleur choix pour ce tutoriel. Choisissez Suivant.
7. Choisissez Créer.

Étape 4 : création d'une route

Les itinéraires permettent d'envoyer les demandes d'API entrantes aux ressources backend. Les itinéraires se composent de deux parties : une méthode HTTP et un chemin de ressource (par exemple,, `GET /items`. Pour cet exemple d'API, nous créerons une route.

Pour créer une route

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Choisissez votre API.
3. Choisissez Itinéraires.
4. Choisissez Créer.
5. Pour Méthode, choisissez **`ANY`**.
6. Pour le chemin, saisissez **`/{proxy+}`**. Le `{proxy+}` à la fin du chemin est une variable de chemin gourmand. API Gateway envoie toutes les requêtes à votre API à cette route.
7. Choisissez Créer.

Étape 5 : création d'une intégration

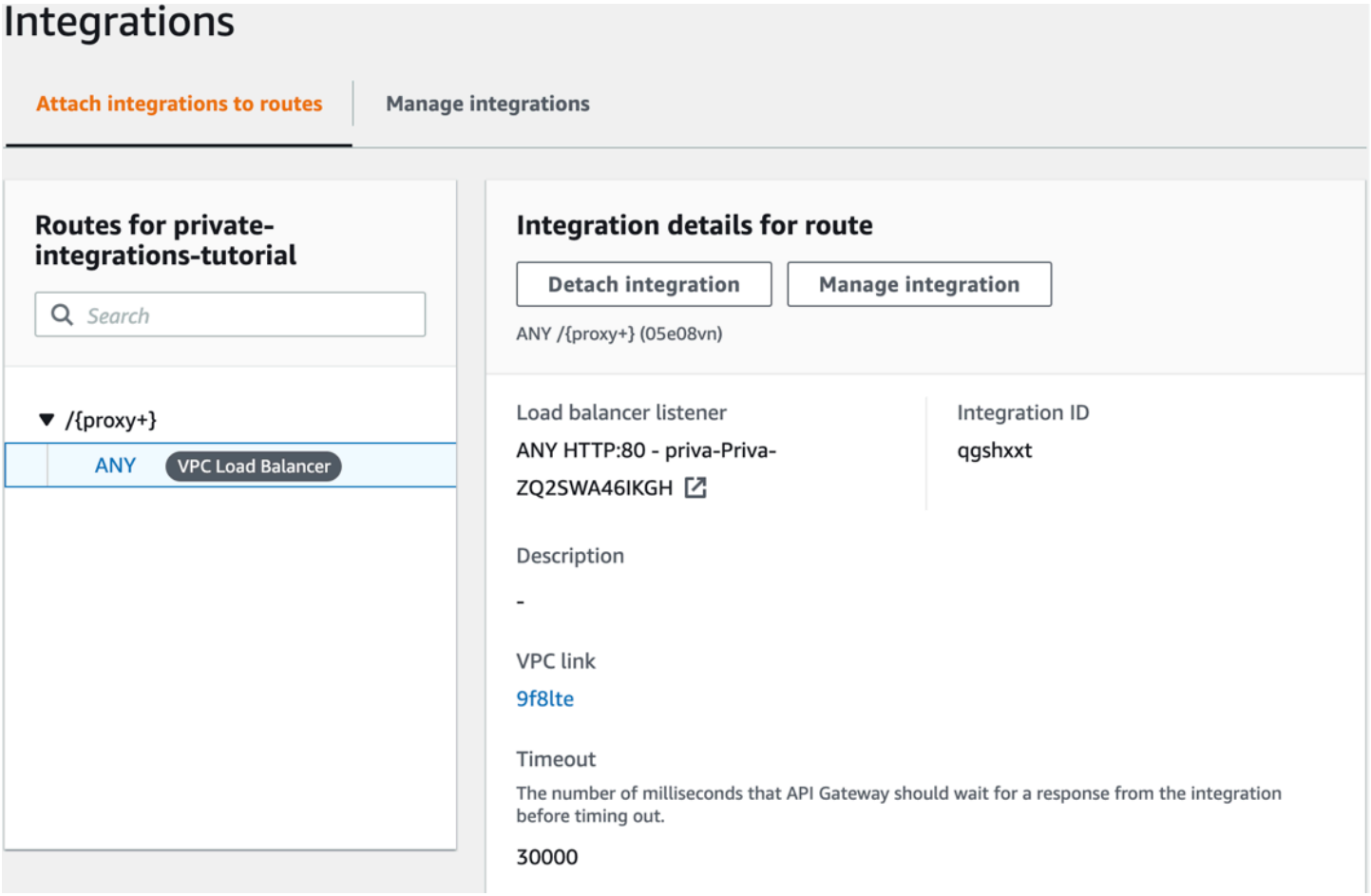
Vous créez une intégration afin de connecter un itinéraire aux ressources backend.

Pour créer une intégration

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.

2. Choisissez votre API.
3. Choisissez Integrations (Intégrations).
4. Choisissez Manage integrations (Gérer les intégrations), puis Create (Créer).
5. Pour Attacher cette intégration à une route, sélectionnez la route ANY/{proxy+} que vous avez créé précédemment.
6. Pour Type d'intégration, choisissez Ressource privée.
7. Pour les Détails de l'intégration, choisissez Sélectionner manuellement.
8. Pour Service cible, choisissez ALB/NLB.
9. Pour Équilibreur de charge, choisissez l'équilibreur de charge que vous avez créé avec le modèle AWS CloudFormation à l'étape 1. Son nom devrait commencer par HTTP-Priva.
10. Pour Écouteur, choisissez **`HTTP 80`**.
11. Pour Lien VPC, choisissez le lien VPC que vous avez créé à l'étape 2. Son nom devrait être `private-integrations-tutorial`.
12. Choisissez Créer.

Pour vérifier que votre route et votre intégration sont correctement configurés, sélectionnez Attacher les intégrations aux routes. La console indique que vous disposez d'une route`ANY /{proxy+}` avec une intégration à un équilibreur de charge VPC.



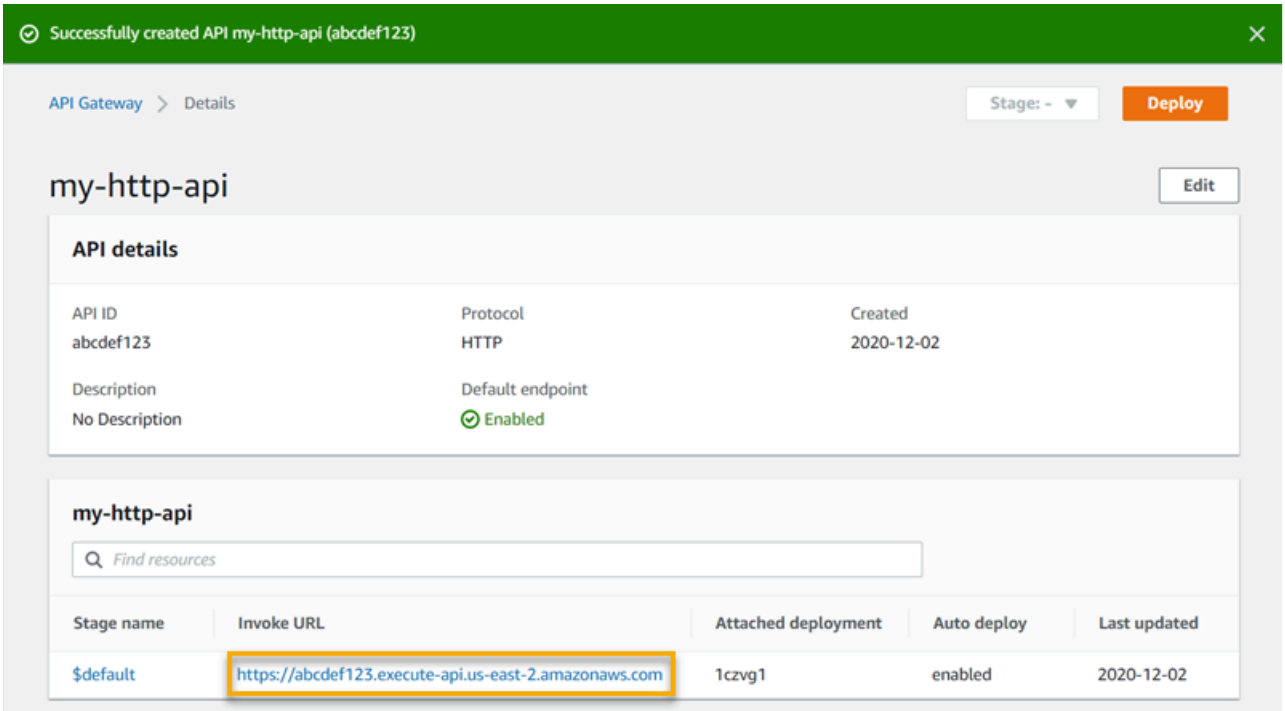
Vous êtes désormais prêt à tester votre API.

Étape 6 : tester votre API

Ensuite, vous testez votre API pour vous assurer qu'elle fonctionne. Pour plus de simplicité, appelez votre API à l'aide d'un navigateur Web.

Pour tester votre API

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Choisissez votre API.
3. Notez l'URL d'appel de votre API.



4. Dans un navigateur Web, accédez à l'URL d'appel de votre API.

   L'URL complète doit ressembler à `https://abcdef123.execute-api.us-east-2.amazonaws.com`.

   Votre navigateur envoie une requête `GET` à l'API.

5. Vérifiez que la réponse de votre API est un message de bienvenue qui vous indique que votre application est en cours d'exécution sur Amazon ECS.

   Si le message de bienvenue s'affiche, vous avez créé avec succès un Amazon ECS service qui s'exécute dans un VPC Amazon et vous avez utilisé une API HTTP API Gateway avec un lien VPC pour accéder à l'Amazon ECS service.

Étape 7 : nettoyer

Pour éviter des coûts inutiles, supprimez les ressources que vous avez créées dans le cadre de ce tutoriel. Les étapes suivantes suppriment votre lien VPC, votre AWS CloudFormation stack et votre API HTTP.

Pour supprimer une API HTTP

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Sur la page APIs (API) , sélectionnez une API. Choisissez Actions, choisissez Supprimer, puis confirmez votre choix.

Pour supprimer un lien VPC

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Choisissez Lien VPC.
3. Sélectionnez votre lien VPC, choisissez Supprimer, puis confirmez votre choix.

Pour supprimer une AWS CloudFormation pile

1. Ouvrez la AWS CloudFormation console à l'adresse https://console.aws.amazon.com/cloudformation.
2. Sélectionnez votre AWS CloudFormation pile.
3. Choisissez Supprimer, puis confirmez votre choix.

Prochaines étapes : Automatisez avec AWS CloudFormation

Vous pouvez automatiser la création et le nettoyage de toutes les AWS ressources impliquées dans ce didacticiel. Pour un exemple complet de modèle AWS CloudFormation , veuillez consulter template.yaml.

Tutoriels sur l'API Amazon WebSocket API Gateway

Les didacticiels suivants proposent un exercice pratique destiné à vous aider à en savoir plus sur les API WebSocket API Gateway.

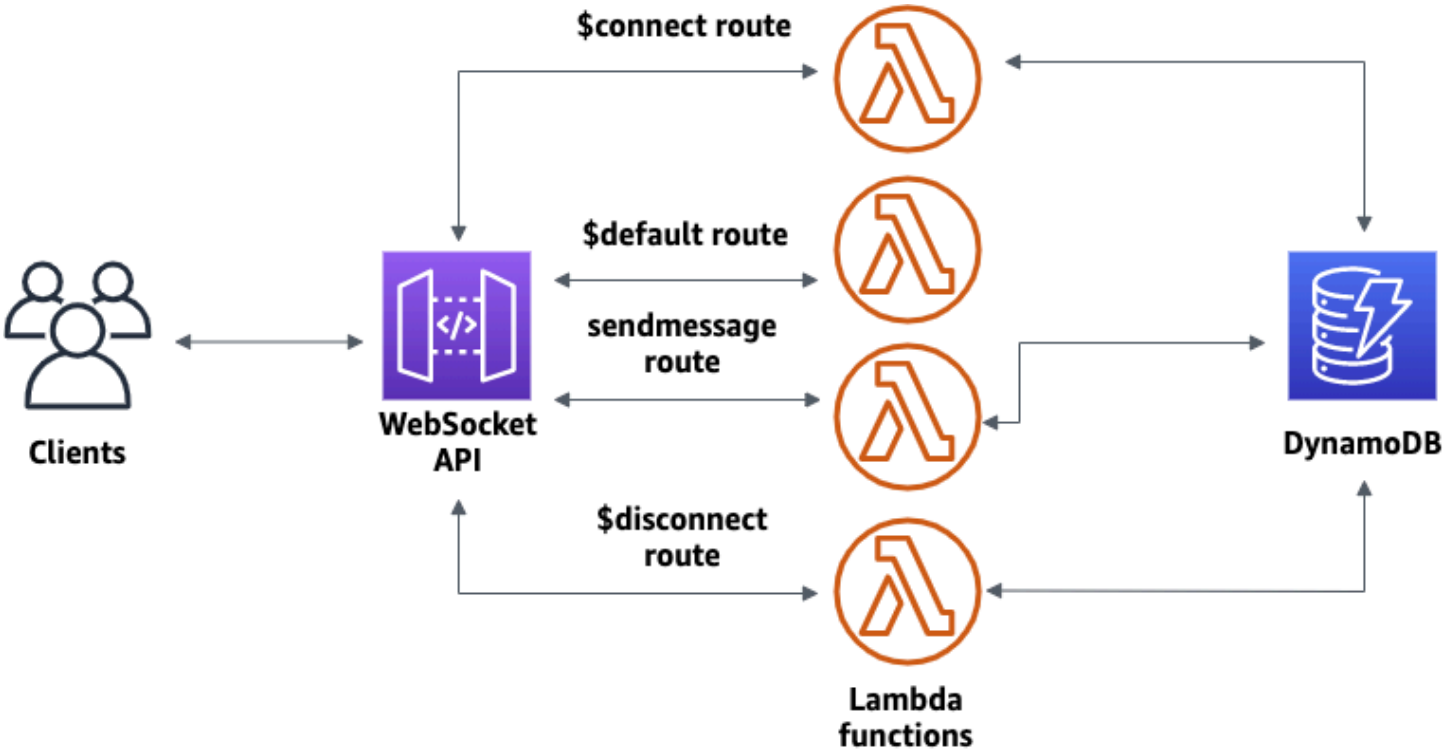
Rubriques

- Tutoriel : Création d'une application de chat sans serveur avec une WebSocket API, Lambda et DynamoDB
- Tutoriel : Création d'une application sans serveur avec trois types d'intégration

Tutoriel : Création d'une application de chat sans serveur avec une WebSocket API, Lambda et DynamoDB

Dans ce didacticiel, vous allez créer une application de chat sans serveur avec une WebSocket API. Avec une WebSocket API, vous pouvez prendre en charge la communication bidirectionnelle entre les clients. Les clients peuvent recevoir des messages sans avoir à rechercher les mises à jour.

Ce didacticiel vous prendra environ 30 minutes. Tout d'abord, vous allez utiliser un AWS CloudFormation modèle pour créer des fonctions Lambda qui géreront les demandes d'API, ainsi qu'une table DynamoDB qui stocke les identifiants de vos clients. Vous allez ensuite utiliser la console API Gateway pour créer une WebSocket API qui s'intègre à vos fonctions Lambda. Enfin, vous allez tester votre API pour vérifier que les messages sont envoyés et reçus.



Pour suivre ce didacticiel, vous avez besoin d'un AWS compte et d'un AWS Identity and Access Management utilisateur disposant d'un accès à la console. Pour plus d'informations, consultez Prérequis.

Vous avez également besoin de `wscat` pour vous connecter à votre API. Pour plus d'informations, consultez the section called "`wscat`À utiliser pour se connecter à une WebSocket API et y envoyer des messages".

Rubriques

Étape 1 : Création de fonctions Lambda et d'une table DynamoDB

Téléchargez et décompressez le modèle de création d'application pour AWS CloudFormation. Vous allez utiliser ce modèle pour créer une table Amazon DynamoDB afin de stocker les ID client de votre application. Chaque client connecté possède un identifiant unique que nous utiliserons comme clé de partition de la table. Ce modèle crée également des fonctions Lambda qui mettent à jour vos connexions client dans DynamoDB et gèrent l'envoi de messages aux clients connectés.

Pour créer une AWS CloudFormation pile

1. Ouvrez la AWS CloudFormation console à l'adresse https://console.aws.amazon.com/cloudformation.
2. Choisissez Créer une pile, puis choisissez Avec de nouvelles ressources (standard).
3. Dans Spécifier le modèle, choisissez Charger un modèle de fichier.
4. Sélectionnez le modèle que vous avez téléchargé.
5. Choisissez Suivant.
6. Pour Nom de la pile, saisissez **`websocket-api-chat-app-tutorial`**, puis choisissez Suivant.
7. Pour Configurer les options de pile, choisissez Suivant.
8. Pour les fonctionnalités, reconnaissez que AWS CloudFormation vous pouvez créer des ressources IAM dans votre compte.
9. Sélectionnez Envoyer.

AWS CloudFormation fournit les ressources spécifiées dans le modèle. La fin du provisionnement de vos ressources peut prendre quelques minutes. Lorsque le statut de votre AWS CloudFormation pile est CREATE_COMPLETE, vous êtes prêt à passer à l'étape suivante.

Étape 2 : Création d'une WebSocket API

Vous allez créer une WebSocket API pour gérer les connexions client et acheminer les demandes vers les fonctions Lambda que vous avez créées à l'étape 1.

Pour créer une WebSocket API

1. Connectez-vous à la console API Gateway à l'adresse : https://console.aws.amazon.com/apigateway.
2. Sélectionnez Create API (Créer une API). Ensuite, pour WebSocket API, choisissez Build.
3. Sous API name (Nom de l'API), saisissez **`websocket-chat-app-tutorial`**.
4. Pour Route selction expression (Expression de sélection d'itinéraire), entrez **`request.body.action`**. Expression de sélection de route pour déterminer la route à appeler lorsqu'un client envoie un message.
5. Choisissez Suivant.
6. Pour Predefined routes (Routes prédéfinies), choisissez Add $connect (Ajouter $connect), Add $disconnect (Ajouter $disconnect) et Add $default (Ajouter $default). Les routes $connect et $disconnect sont des itinéraires spéciaux qu'API Gateway appelle automatiquement lorsqu'un client se connecte à une API ou se déconnecte de celle-ci. API Gateway appelle la route `$default` lorsqu'aucune autre route ne correspond à une demande.
7. Pour Custom routes (Routes personnalisées), choisissez Add custom route (Ajouter une route personnalisée). Pour Route key (Clé de route), entrez **`sendmessage`**. Cette route personnalisée gère les messages envoyés aux clients connectés.
8. Choisissez Suivant.
9. Sous Attach integrations (Attacher les intégrations), pour chaque route et type d'intégration, choisissez Lambda.

   Pour Lambda, choisissez la fonction Lambda correspondante que vous avez créée à l'étape 1. AWS CloudFormation Le nom de chaque fonction correspond à une route. Par exemple, pour la route $connect, choisissez la fonction nommée **`websocket-chat-app-tutorial-ConnectHandler`**.
10. Passez en revue l'étape créée par API Gateway pour vous. Par défaut, API Gateway crée un nom d'étape `production` et déploie automatiquement votre API à ce stade. Choisissez Suivant.
11. Choisissez Create and deploy (Créer et déployer).

Étape 3 : Tester votre API

Ensuite, vous testez votre API pour vérifier qu'elle fonctionne. Pour vous connecter à l'API, utilisez la commande `wscat`.

Pour obtenir l'URL d'appel de votre API

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Choisissez votre API.
3. Choisissez Stages (Étapes), puis production.
4. Notez l'WebSocket URL de votre API. L'URL doit ressembler à `wss://abcdef123.execute-api.us-east-2.amazonaws.com/production`.

Pour vous connecter à votre API

1. Utilisez la commande suivante pour vous connecter à votre API. Lorsque vous vous connectez à votre API, API Gateway appelle la route `$connect`. Lorsque cette route est appelée, il appelle une fonction Lambda qui stocke votre ID de connexion dans DynamoDB.

   ```
   wscat -c wss://abcdef123.execute-api.us-west-2.amazonaws.com/production
   ```

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

2. Ouvrez un nouveau terminal et exécutez à nouveau la commande wscat avec les paramètres suivants.

   ```
   wscat -c wss://abcdef123.execute-api.us-west-2.amazonaws.com/production
   ```

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

   Vous disposez ainsi de deux clients connectés capables d'échanger des messages.

Pour envoyer un message

- API Gateway détermine la route à appeler en fonction de l'expression de sélection de route de votre API. L'expression de sélection de route de votre API est `$request.body.action`. Par

conséquent, API Gateway appelle la route `sendmessage` lorsque vous envoyez le message suivant :

```
{"action": "sendmessage", "message": "hello, everyone!"}
```

La fonction Lambda associée à l'itinéraire appelé collecte les ID client de DynamoDB. Ensuite, la fonction appelle API Gateway Management et envoie le message à ces clients. Tous les clients connectés reçoivent le message suivant :

```
< hello, everyone!
```

Pour appeler la route $ par défaut de votre API

- API Gateway appelle la route par défaut de votre API lorsqu'un client envoie un message qui ne correspond pas à vos routes définies. La fonction Lambda associée à la route `$default` utilise API Gateway Management pour envoyer au client des informations relatives à leur connexion.

```
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
```

Pour vous déconnecter de votre API

- Pour vous déconnecter de votre API, appuyez sur **CTRL+C**. Lorsqu'un client se déconnecte de votre API, API Gateway appelle la route `$disconnect`. L'intégration Lambda pour la route `$disconnect` de votre API supprime l'ID de connexion de DynamoDB.

Étape 4 : Nettoyer

Pour éviter des coûts inutiles, supprimez les ressources que vous avez créées dans le cadre de ce tutoriel. Les étapes suivantes suppriment votre AWS CloudFormation stack et votre WebSocket API.

Pour supprimer une WebSocket API

1. Connectez-vous à la console API Gateway à l'adresse : https://console.aws.amazon.com/apigateway.
2. Sur la page APIs (API), sélectionnez votre API `websocket-chat-app-tutorial`. Choisissez Actions, choisissez Supprimer, puis confirmez votre choix.

Pour supprimer une AWS CloudFormation pile

1. Ouvrez la AWS CloudFormation console à l'adresse https://console.aws.amazon.com/cloudformation.
2. Sélectionnez votre AWS CloudFormation pile.
3. Choisissez Supprimer, puis confirmez votre choix.

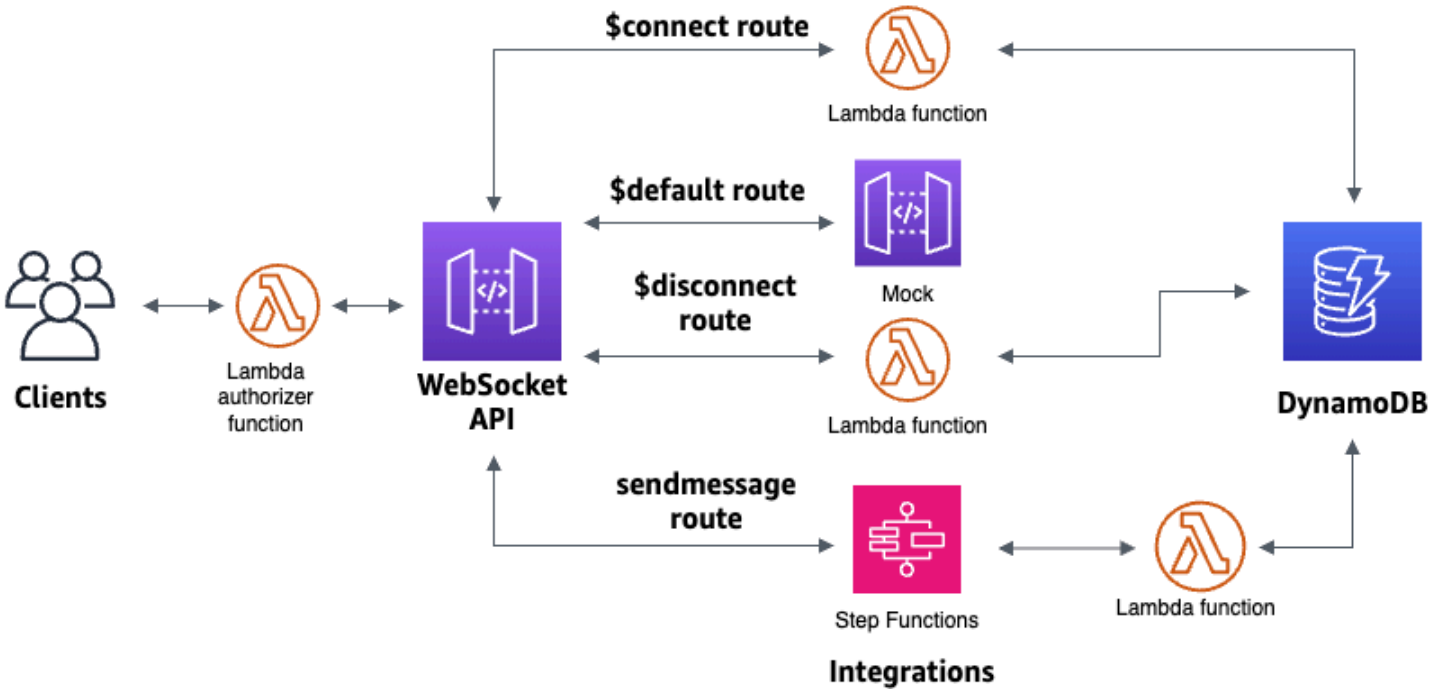
Prochaines étapes : Automatisez avec AWS CloudFormation

Vous pouvez automatiser la création et le nettoyage de toutes les AWS ressources impliquées dans ce didacticiel. Pour un AWS CloudFormation modèle qui crée cette API et toutes les ressources associées, consultez chat-app.yaml.

Tutoriel : Création d'une application sans serveur avec trois types d'intégration

Dans ce didacticiel, vous allez créer une application de diffusion sans serveur avec une WebSocket API. Les clients peuvent recevoir des messages sans avoir à rechercher les mises à jour.

Ce didacticiel explique comment diffuser des messages aux clients connectés et inclut un exemple d'autorisateur Lambda, une intégration fictive et une intégration sans proxy à Step Functions.



Après avoir créé vos ressources à l'aide d'un AWS CloudFormation modèle, vous utiliserez la console API Gateway pour créer une WebSocket API qui s'intègre à vos AWS ressources. Vous allez associer un autorisateur Lambda à votre API et créer une intégration de AWS service avec Step Functions pour démarrer une exécution par machine à états. La machine d'état Step Functions invoquera une fonction Lambda qui envoie un message à tous les clients connectés.

Après avoir créé votre API, vous testerez votre connexion à votre API et vérifierez que les messages sont envoyés et reçus. Ce didacticiel dure environ 45 minutes.

Rubriques

Prérequis

Vous avez besoin des prérequis suivants :

- Un AWS compte et un AWS Identity and Access Management utilisateur ayant accès à la console. Pour plus d'informations, consultez Prérequis.
- `wscat`pour vous connecter à votre API. Pour plus d'informations, consultez the section called "`wscat`À utiliser pour se connecter à une WebSocket API et y envoyer des messages".

Nous vous recommandons de suivre le didacticiel de l'application de WebSocket chat avant de commencer ce didacticiel. Pour terminer le didacticiel de l'application de WebSocket chat, voirthe section called "WebSocket application de chat".

Étape 1 : Créer des ressources

Téléchargez et décompressez le modèle de création d'application pour AWS CloudFormation. Vous allez utiliser ce modèle pour créer les éléments suivants :

- Fonctions Lambda qui gèrent les demandes d'API et autorisent l'accès à votre API.
- Une table DynamoDB pour stocker les identifiants des clients et l'identification de l'utilisateur principal renvoyés par l'autorisateur Lambda.
- Une machine d'état Step Functions pour envoyer des messages aux clients connectés.

Pour créer une AWS CloudFormation pile

1. Ouvrez la AWS CloudFormation console à l'adresse https://console.aws.amazon.com/cloudformation.
2. Choisissez Créer une pile, puis choisissez Avec de nouvelles ressources (standard).
3. Dans Spécifier le modèle, choisissez Charger un modèle de fichier.
4. Sélectionnez le modèle que vous avez téléchargé.
5. Choisissez Suivant.
6. Pour Nom de la pile, saisissez **`websocket-step-functions-tutorial`**, puis choisissez Suivant.
7. Pour Configurer les options de pile, choisissez Suivant.
8. Pour les fonctionnalités, reconnaissez que AWS CloudFormation vous pouvez créer des ressources IAM dans votre compte.

9. Sélectionnez Envoyer.

AWS CloudFormation fournit les ressources spécifiées dans le modèle. La fin du provisionnement de vos ressources peut prendre quelques minutes. Choisissez l'onglet Sorties pour voir les ressources que vous avez créées et leurs ARN. Lorsque le statut de votre AWS CloudFormation pile est CREATE_COMPLETE, vous êtes prêt à passer à l'étape suivante.

Étape 2 : Création d'une WebSocket API

Vous allez créer une WebSocket API pour gérer les connexions clients et acheminer les demandes vers les ressources que vous avez créées à l'étape 1.

Pour créer une WebSocket API

1. Connectez-vous à la console API Gateway à l'adresse : https://console.aws.amazon.com/apigateway.
2. Sélectionnez Create API (Créer une API). Ensuite, pour WebSocket API, choisissez Build.
3. Sous API name (Nom de l'API), saisissez **`websocket-step-functions-tutorial`**.
4. Pour Route selction expression (Expression de sélection d'itinéraire), entrez **`request.body.action`**.

   Expression de sélection de route pour déterminer la route à appeler lorsqu'un client envoie un message.
5. Choisissez Suivant.
6. Pour les itinéraires prédéfinis, choisissez Ajouter $connect, Ajouter $disconnect, Ajouter $default.

   Les routes $connect et $disconnect sont des itinéraires spéciaux qu'API Gateway appelle automatiquement lorsqu'un client se connecte à une API ou se déconnecte de celle-ci. API Gateway invoque la route $default lorsqu'aucune autre route ne correspond à une demande. Vous allez créer un itinéraire personnalisé pour vous connecter à Step Functions après avoir créé votre API.
7. Choisissez Suivant.
8. Pour l'intégration de $connect, procédez comme suit :

   a. Pour Type d'intégration, choisissez Lambda.

b. Pour la fonction Lambda, choisissez la fonction Lambda $connect correspondante que vous avez créée AWS CloudFormation à l'étape 1. Le nom de la fonction Lambda doit commencer par. **`websocket-step`**

9. Pour l'intégration de $disconnect, procédez comme suit :

   a. Pour Type d'intégration, choisissez Lambda.

   b. Pour la fonction Lambda, choisissez la fonction Lambda $disconnect correspondante que vous avez créée AWS CloudFormation à l'étape 1. Le nom de la fonction Lambda doit commencer par. **`websocket-step`**

10. Pour Integration for $default, choisissez mock.

    Dans une intégration fictive, API Gateway gère la réponse de l'itinéraire sans recourir à un backend d'intégration.

11. Choisissez Suivant.

12. Passez en revue l'étape créée par API Gateway pour vous. Par défaut, API Gateway crée une étape nommée production et déploie automatiquement votre API à cette étape. Choisissez Suivant.

13. Choisissez Create and deploy (Créer et déployer).

Étape 3 : créer un autorisateur Lambda

Pour contrôler l'accès à votre WebSocket API, vous devez créer un autorisateur Lambda. Le AWS CloudFormation modèle a créé la fonction d'autorisation Lambda pour vous. Vous pouvez voir la fonction Lambda dans la console Lambda. Le nom doit commencer par**`websocket-step-functions-tutorial-AuthorizerHandler`**. Cette fonction Lambda refuse tous les appels à l' WebSocket API sauf si l'`Authorization`en-tête l'est. `Allow` La fonction Lambda transmet également la `$context.authorizer.principalId` variable à votre API, qui est ensuite utilisée dans la table DynamoDB pour identifier les appelants d'API.

Au cours de cette étape, vous configurez la route $connect pour utiliser l'autorisateur Lambda.

Pour créer un autorisateur Lambda

1. Connectez-vous à la console API Gateway à l'adresse : https://console.aws.amazon.com/apigateway.

2. Dans le panneau de navigation principal, choisissez Mécanismes d'autorisation.

3. Choisissez Créer un autorisateur.
4. Dans le champ Nom de l'autorisateur, entrez**`LambdaAuthorizer`**.
5. Pour Authorizer ARN, entrez le nom de l'autorisateur créé par le AWS CloudFormation modèle. Le nom doit commencer par**`websocket-step-functions-tutorial-AuthorizerHandler`**.

   > **Note**
   >
   > Nous vous recommandons de ne pas utiliser cet exemple d'autorisateur pour vos API de production.

6. Pour le type de source d'identité, choisissez En-tête. Pour Key (Clé), saisissez **`Authorization`**.
7. Choisissez Créer un mécanisme d'autorisation.

Après avoir créé votre autorisateur, vous l'attachez à la route $connect de votre API.

Pour associer un autorisateur à la route $connect

1. Dans le volet de navigation principal, choisissez Routes.
2. Choisissez la route $connect.
3. Dans la section Paramètres de la demande de route, choisissez Modifier.
4. Pour Autorisation, choisissez le menu déroulant, puis sélectionnez l'autorisateur de votre demande.
5. Sélectionnez Enregistrer les modifications.

Étape 4 : Création d'une intégration bidirectionnelle fictive

Ensuite, vous créez l'intégration fictive bidirectionnelle pour la route $default. Une intégration fictive vous permet d'envoyer une réponse au client sans utiliser de backend. Lorsque vous créez une intégration pour la route $default, vous pouvez montrer aux clients comment interagir avec votre API.

Vous configurez la route $default pour indiquer aux clients d'utiliser la route sendmessage.

Pour créer une intégration fictive

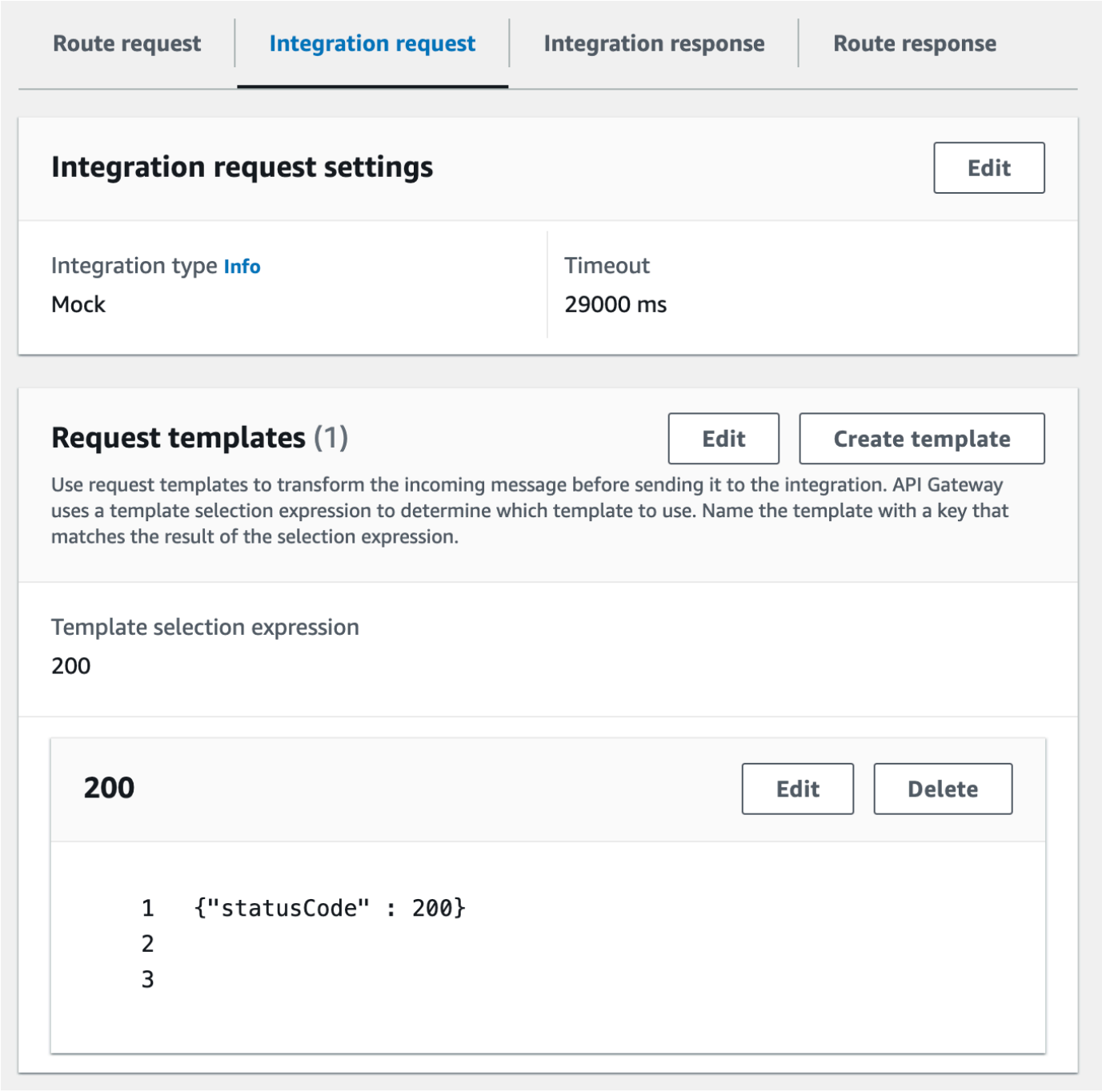
1. Connectez-vous à la console API Gateway à l'adresse : https://console.aws.amazon.com/apigateway.

2. Choisissez l'itinéraire $default, puis cliquez sur l'onglet Demande d'intégration.
3. Pour les modèles de demande, choisissez Modifier.
4. Pour Expression de sélection du modèle**200**, entrez, puis choisissez Modifier.
5. Dans l'onglet Demande d'intégration, pour Modèles de demande, choisissez Créer un modèle.
6. Dans le champ Clé du modèle, entrez**200**.
7. Pour Générer un modèle, entrez le modèle de mappage suivant :

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

   Sélectionnez Create template (Créer un modèle).

   Le résultat doit avoir l'aspect suivant :



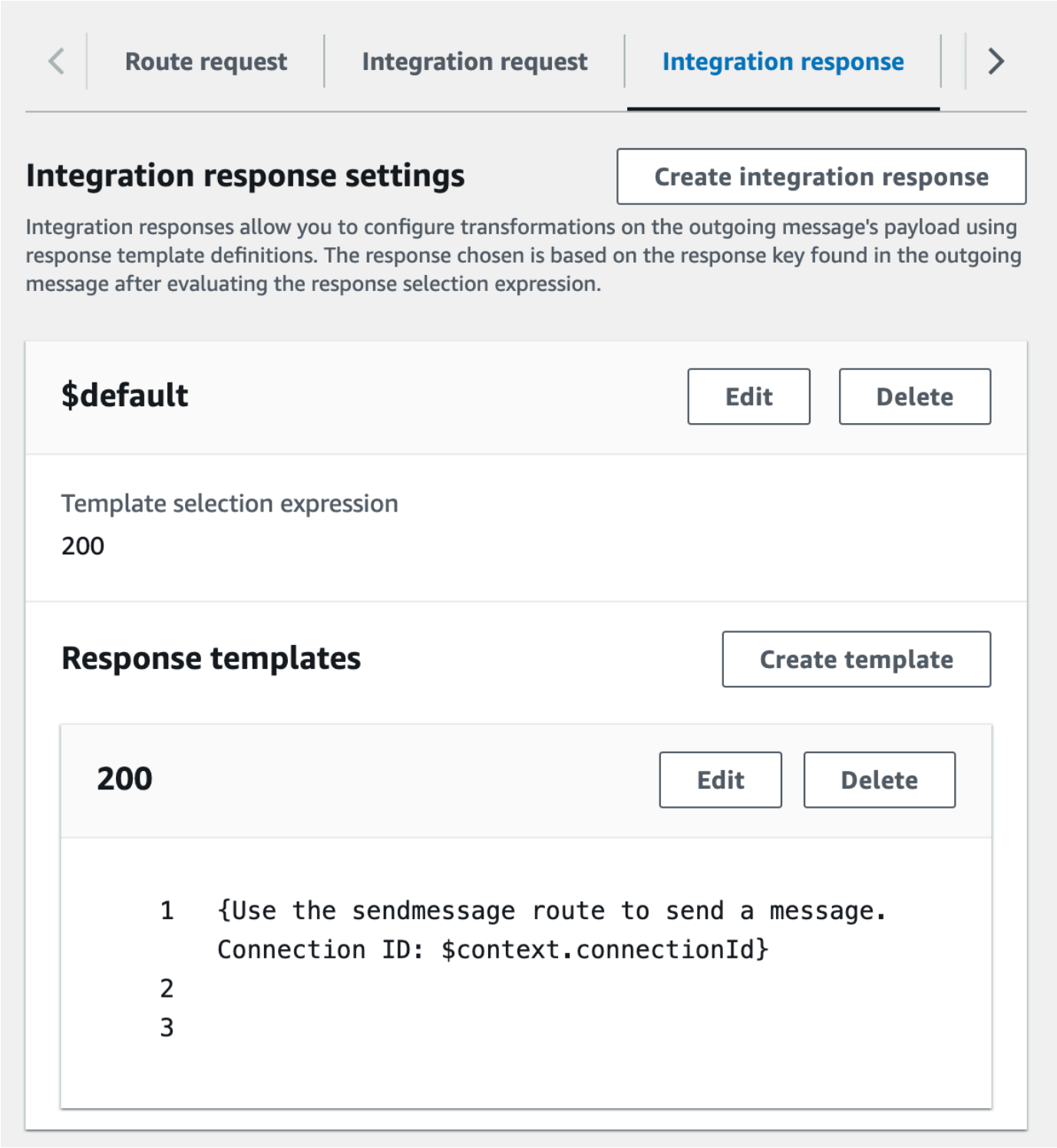
8. Dans le volet de routage $default, choisissez Activer la communication bidirectionnelle.
9. Choisissez l'onglet Réponse d'intégration, puis choisissez Créer une réponse d'intégration.
10. Dans le champ Clé de réponse, entrez**`$default`**.
11. Pour Expression de sélection du modèle, entrez**`200`**.
12. Choisissez Créer une réponse.
13. Sous Modèles de réponse, choisissez Créer un modèle.

14. Dans le champ Clé du modèle, entrez**200**.
15. Pour Modèle de réponse, entrez le modèle de mappage suivant :

```
{"Use the sendmessage route to send a message. Connection ID:
 $context.connectionId"}
```

16. Sélectionnez Create template (Créer un modèle).

    Le résultat doit avoir l'aspect suivant :



Étape 5 : Création d'une intégration sans proxy avec Step Functions

Ensuite, vous créez un itinéraire d'envoi de message. Les clients peuvent invoquer la route sendmessage pour diffuser un message à tous les clients connectés. La route sendmessage

est intégrée à un AWS service non proxy avec. AWS Step Functions L'intégration invoque la StartExecutioncommande pour la machine à états Step Functions créée pour vous par le AWS CloudFormation modèle.

Pour créer une intégration sans proxy

1. Connectez-vous à la console API Gateway à l'adresse : https://console.aws.amazon.com/apigateway.
2. Choisissez Create Route (Créer un itinéraire).
3. Pour Route key (Clé de route), entrez **`sendmessage`**.
4. Dans Type d'intégration, sélectionnez AWS service.
5. Pour AWS Région, entrez la région dans laquelle vous avez déployé votre AWS CloudFormation modèle.
6. Pour le AWS service, choisissez Step Functions.
7. Dans le champ HTTP Method, sélectionnez POST.
8. Pour Nom de l'action, saisissez **`StartExecution`**.
9. Pour Rôle d'exécution, entrez le rôle d'exécution créé par le AWS CloudFormation modèle. Le nom doit être WebsocketTutorialApiRole.
10. Choisissez Create Route (Créer un itinéraire).

Ensuite, vous créez un modèle de mappage pour envoyer les paramètres de demande à la machine d'état Step Functions.

Pour créer un modèle de mappage

1. Choisissez l'itinéraire d'envoi du message, puis cliquez sur l'onglet Demande d'intégration.
2. Dans la section Modèles de demande, choisissez Modifier.
3. Pour Expression de sélection du modèle, entrez**`\$default`**.
4. Choisissez Modifier.
5. Dans la section Modèles de demande, choisissez Créer un modèle.
6. Dans le champ Clé du modèle, entrez**`\$default`**.
7. Pour Générer un modèle, entrez le modèle de mappage suivant :

```
#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"
}
```

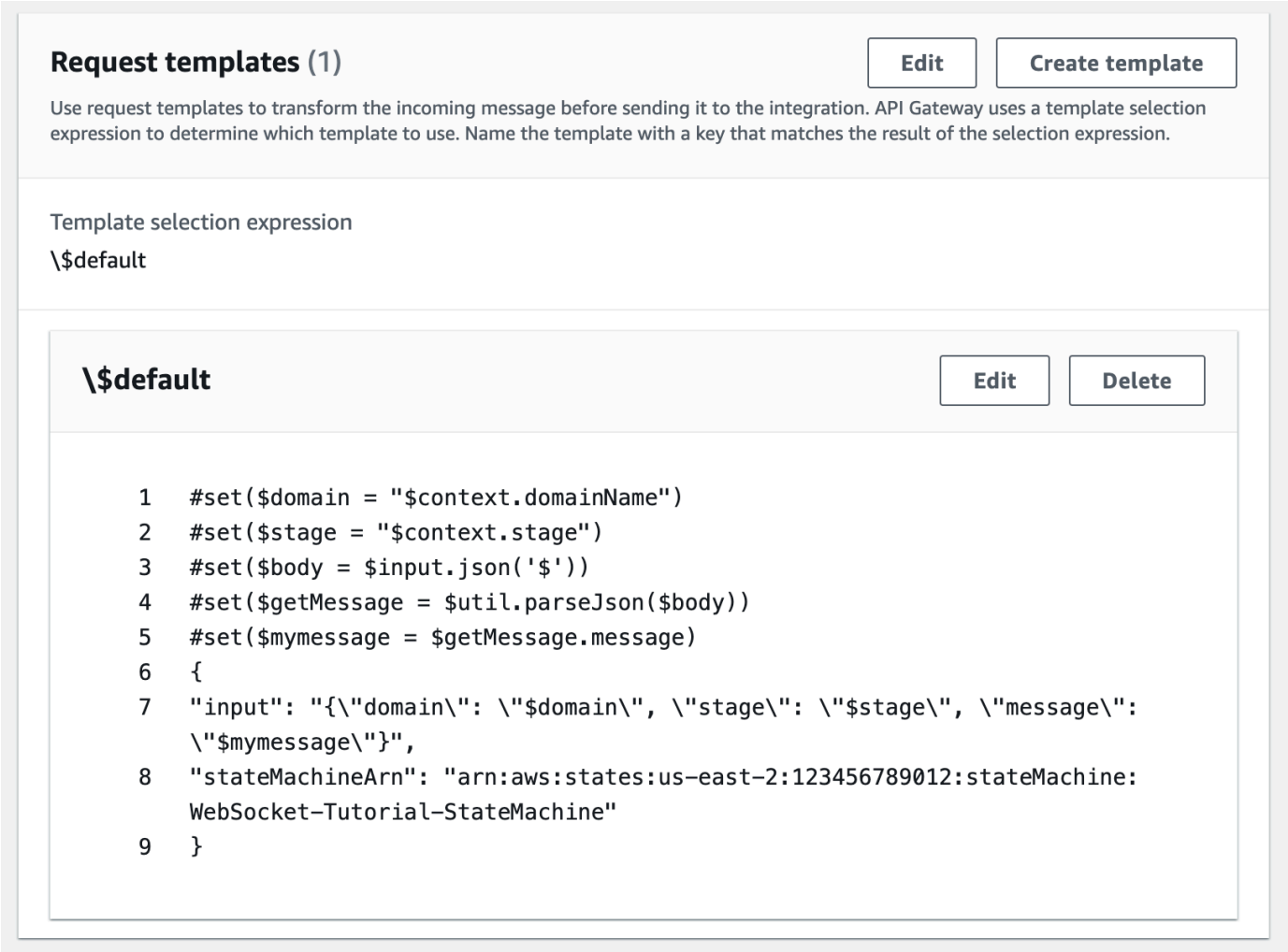
Remplacez le *stateMachineArn*par l'ARN de la machine à états créée par AWS CloudFormation.

Le modèle de mappage effectue les opérations suivantes :

- Crée la variable `$domain` à l'aide de la variable de contexte`domainName`.
- Crée la variable `$stage` à l'aide de la variable de contexte`stage`.

  Les `$stage` variables `$domain` et sont nécessaires pour créer une URL de rappel.
- Prend en compte le message `sendmessage` JSON entrant et extrait la `message` propriété.
- Crée l'entrée pour la machine à états. L'entrée correspond au domaine et à l'étape de l' WebSocket API, ainsi qu'au message provenant de l'`sendmessage`itinéraire.

8. Sélectionnez Create template (Créer un modèle).



Vous pouvez créer une intégration sans proxy sur les routes $connect ou $disconnect, pour ajouter ou supprimer directement un ID de connexion dans la table DynamoDB, sans appeler de fonction Lambda.

Étape 6 : tester votre API

Vous allez ensuite déployer et tester votre API pour vous assurer qu'elle fonctionne correctement. Vous utiliserez la `wscat` commande pour vous connecter à l'API, puis vous utiliserez une commande slash pour envoyer un cadre ping afin de vérifier la connexion à l' WebSocket API.

Pour déployer votre API

1. Connectez-vous à la console API Gateway à l'adresse : https://console.aws.amazon.com/apigateway.
2. Dans le volet de navigation principal, choisissez Routes.

3. Sélectionnez Deploy API (Déployer une API).
4. Pour Stage, sélectionnez production.
5. (Facultatif) Pour la description du déploiement, entrez une description.
6. Choisissez Deploy (Déployer).

Après avoir déployé votre API, vous pouvez l'invoquer. Utilisez l'URL d'appel pour appeler votre API.

Pour obtenir l'URL d'appel de votre API

1. Choisissez votre API.
2. Choisissez Stages (Étapes), puis production.
3. Notez l'WebSocket URL de votre API. L'URL doit ressembler à `wss://abcdef123.execute-api.us-east-2.amazonaws.com/production`.

Maintenant que vous avez votre URL d'appel, vous pouvez tester la connexion à votre WebSocket API.

Pour tester la connexion à votre API

1. Utilisez la commande suivante pour vous connecter à votre API. Tout d'abord, vous testez la connexion en invoquant le `/ping` chemin.

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

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

2. Entrez la commande suivante pour envoyer un ping au cadre de contrôle. Vous pouvez utiliser un cadre de contrôle à des fins de maintien en vie du côté client.

```
/ping
```

   Le résultat doit avoir l'aspect suivant :

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

Maintenant que vous avez testé la connexion, vous pouvez vérifier que votre API fonctionne correctement. Au cours de cette étape, vous ouvrez une nouvelle fenêtre de terminal afin que l'WebSocket API puisse envoyer un message à tous les clients connectés.

Pour tester votre API

1. Ouvrez un nouveau terminal et exécutez à nouveau la commande `wscat` avec les paramètres suivants.

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

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

2. API Gateway détermine la route à invoquer en fonction de l'expression de sélection de la demande d'itinéraire de votre API. L'expression de sélection de route de votre API est`$request.body.action`. Par conséquent, API Gateway appelle la route `sendmessage` lorsque vous envoyez le message suivant :

   ```
   {"action": "sendmessage", "message": "hello, from Step Functions!"}
   ```

   La machine d'état Step Functions associée à la route appelle une fonction Lambda avec le message et l'URL de rappel. La fonction Lambda appelle l'API API Gateway Management et envoie le message à tous les clients connectés. Tous les clients reçoivent le message suivant :

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

Maintenant que vous avez testé votre WebSocket API, vous pouvez vous déconnecter de votre API.

Pour vous déconnecter de votre API

- Pour vous déconnecter de votre API, appuyez sur `CTRL+C`.

  Lorsqu'un client se déconnecte de votre API, API Gateway invoque la route $disconnect de votre API. L'intégration Lambda pour la route $disconnect de votre API supprime l'ID de connexion de DynamoDB.

Étape 7 : nettoyer

Pour éviter des coûts inutiles, supprimez les ressources que vous avez créées dans le cadre de ce tutoriel. Les étapes suivantes suppriment votre AWS CloudFormation stack et votre WebSocket API.

Pour supprimer une WebSocket API

1. Connectez-vous à la console API Gateway à l'adresse : https://console.aws.amazon.com/apigateway.
2. Sur la page APIs, sélectionnez votre websocket-api.
3. Choisissez Actions, choisissez Supprimer, puis confirmez votre choix.

Pour supprimer une AWS CloudFormation pile

1. Ouvrez la AWS CloudFormation console à l'adresse https://console.aws.amazon.com/cloudformation.
2. Sélectionnez votre AWS CloudFormation pile.
3. Choisissez Supprimer, puis confirmez votre choix.

Étapes suivantes

Vous pouvez automatiser la création et le nettoyage de toutes les AWS ressources impliquées dans ce didacticiel. Pour obtenir un exemple de AWS CloudFormation modèle qui automatise ces actions pour ce didacticiel, consultez le fichier ws-sfn.zip.

Utilisation des API REST

Une API REST dans API Gateway est un ensemble de ressources et de méthodes intégrées aux points de terminaison HTTP principaux, aux fonctions Lambda ou à d'autres services. AWS Vous pouvez utiliser les fonctions API Gateway pour vous aider dans tous les aspects du cycle de vie de l'API, de la création à la surveillance de vos API de production.

Les API REST API Gateway utilisent un modèle de demande/réponse dans lequel un client envoie une demande à un service et le service répond de manière synchrone. Ce type de modèle convient à de nombreux types d'applications qui dépendent de la communication synchrone.

Rubriques

Développement d'une API REST dans API Gateway

Dans Amazon API Gateway, vous créez une API REST sous la forme d'un ensemble d'entités programmables, appelées ressources. Par exemple, vous utilisez une RestApiressource pour représenter une API qui peut contenir un ensemble d'entités Resource.

Chaque `Resource` entité peut avoir une ou plusieurs ressources de méthode. A `Method` est une demande entrante soumise par le client et est exprimée dans les paramètres et le corps de la demande. Il définit l'interface de programmation de l'application permettant au client d'accéder à l'exposé`Resource`. Pour intégrer le `Method` à un point de terminaison principal, également appelé point de terminaison d'intégration, vous créez une ressource d'intégration. Cela transmet la demande entrante à un URI de point de terminaison d'intégration spécifié. Si nécessaire, vous pouvez transformer les paramètres ou le corps de la demande pour répondre aux exigences du backend.

Pour les réponses, vous pouvez créer une MethodResponseressource pour représenter une réponse à une demande reçue par le client et vous créer une IntegrationResponseressource pour représenter

la réponse à la demande renvoyée par le backend. Vous pouvez configurer la réponse d'intégration pour transformer les données de la réponse du backend avant de les renvoyer au client ou pour transmettre la réponse du backend telle quelle au client.

Pour aider vos clients à comprendre votre API, vous pouvez également fournir une documentation sur l'API, dans le cadre de la création de l'API ou une fois l'API créée. Pour ce faire, ajoutez une DocumentationPartressource pour une entité d'API prise en charge.

Pour contrôler la manière dont les clients appellent une API, utilisez des autorisations IAM, un mécanisme d'autorisation Lambda ou un groupe d'utilisateurs Amazon Cognito. Pour mesurer l'utilisation de votre API, configurez plans d'utilisation pour limiter les demandes d'API. Vous pouvez les activer lors de la création ou de la mise à jour de votre API.

Pour une introduction sur la création d'une API, consultezthe section called “Tutoriel : API Hello World avec l'intégration de proxy Lambda”. Pour en savoir plus sur les fonctionnalités d'API Gateway que vous pouvez utiliser lors du développement d'une API REST, consultez les rubriques suivantes. Ces rubriques contiennent des informations conceptuelles et des procédures que vous pouvez exécuter à l'aide de la console API Gateway, de l'API REST API Gateway AWS CLI, du ou de l'un des AWS SDK.

Rubriques

- Types de points de terminaison d'API API Gateway
- Méthodes pour les API REST dans API Gateway
- Contrôle et gestion de l'accès à une API REST dans API Gateway
- Configuration des intégrations d'API REST
- Utilisation de la validation des demandes dans API Gateway
- Configuration des transformations de données pour les API REST
- Réponses de passerelle dans API Gateway
- Activation de CORS pour une ressource de l'API REST
- Utilisation des types de médias binaires pour les API REST
- Appel d'une API REST dans Amazon API Gateway
- Configuration d'une API REST à l'aide d'OpenAPI

Types de points de terminaison d'API API Gateway

Un type de point de terminaison d'API fait référence au nom d'hôte de l'API. Le type de point de terminaison de l'API peut être optimisé pour les périphériques, régional ou privé, en fonction de l'emplacement duquel provient la majorité de votre trafic d'API.

Points de terminaison d'API optimisées pour les périphériques

Un point de terminaison d'API optimisé pour les périphériques achemine généralement les demandes vers le CloudFront point de présence (POP) le plus proche, ce qui peut être utile dans les cas où vos clients sont répartis géographiquement. Il s'agit du type de point de terminaison par défaut pour les API REST API Gateway.

Les API optimisées pour les périphériques utilisent une lettre majuscule pour les noms des en-têtes HTTP (par exemple, `Cookie`).

CloudFront trie les cookies HTTP dans l'ordre naturel par nom de cookie avant de transmettre la demande à votre source. Pour plus d'informations sur la manière dont les cookies CloudFront sont traités, consultez la section Mise en cache du contenu basé sur les cookies.

Un nom de domaine personnalisé que vous utilisez pour une API optimisée pour les périphériques s'applique à toutes les régions.

Points de terminaison d'API régionale

Un point de terminaison d'API régional est destiné aux clients de la même région. Lorsqu'un client exécuté sur une instance EC2 appelle une API dans la même région, ou lorsqu'une API est destinée à servir un petit nombre de clients ayant des exigences élevées, une API régionale réduit la charge de connexion.

Pour une API régionale, tout nom de domaine personnalisé que vous utilisez est spécifique à la région où l'API est déployée. Si vous déployez une API régionale dans plusieurs régions, elle pourra avoir le même nom de domaine personnalisé dans toutes les régions. Vous pouvez utiliser des domaines personnalisés avec Amazon Route 53 pour effectuer des tâches telles que le routage basé sur la latence. Pour de plus amples informations, veuillez consulter the section called “Configuration d'un nom de domaine personnalisé régional” et the section called “Création d'un nom de domaine personnalisé optimisé pour la périphérie”.

Les points de terminaison d'API régionale transmettent tous les noms d'en-tête en l'état.

> **Note**
>
> Dans les cas où les clients d'API sont géographiquement dispersés, il peut toujours être judicieux d'utiliser un point de terminaison d'API régional, associé à votre propre CloudFront distribution Amazon, afin de garantir qu'API Gateway n'associe pas l'API à des distributions contrôlées par les services CloudFront . Pour plus d'informations sur ce cas d'utilisation, consultez Comment configurer API Gateway avec ma propre CloudFront distribution ? .

Point de terminaison d'API privée

Un point de terminaison d'API privée est un point de terminaison d'API qui est uniquement accessible à partir de votre Amazon Virtual Private Cloud (VPC) à l'aide d'un point de terminaison d'un VPC d'interface, qui est une interface réseau de point de terminaison (ENI) créée dans votre VPC. Pour de plus amples informations, veuillez consulter the section called “API REST privées”.

Les points de terminaison d’API privée transmettent tous les noms d'en-tête à travers en l'état.

Modification d'un type de point de terminaison d'API publique ou privée dans API Gateway

La modification d'un type de point de terminaison d'API nécessite que vous mettiez à jour la configuration de l'API. Vous pouvez modifier un type d'API existant à l'aide de la console API Gateway AWS CLI, du ou d'un AWS SDK pour API Gateway. Le type de point de terminaison ne peut pas être modifié de nouveau tant que la modification en cours n'est pas terminée, mais votre API sera disponible.

Les modifications de types de points de terminaison suivantes sont prises en charge :

- De l'optimisation périphérique à l'optimisation régionale ou privée
- Du régional à l'optimisation périphérique ou au privé
- Du privé au régional

Vous ne pouvez pas modifier une API privée en une API optimisée pour les périphériques.

Si vous passez d'une API publique optimisée pour les périphériques à une API régionale ou vice versa, notez qu'une API optimisée pour les périphériques peut avoir des comportements différents de ceux d'une API régionale. Par exemple, une API optimisée pour les périphériques supprime l'en-

tête `Content-MD5`. Toute valeur de hachage MD5 transmise au backend peut être exprimée dans un paramètre de chaîne de demande ou une propriété du corps. Cependant, l'API régionale transmet cet en-tête, bien qu'elle puisse le renommer en un autre nom. Comprendre les différences vous aide à décider comment mettre à jour une API optimisée pour les périphériques vers une API régionale ou d'une API régionale vers une API optimisée pour les périphériques.

Rubriques

- Utilisation de la console API Gateway pour modifier un type de point de terminaison d'API
- Utilisez le AWS CLI pour modifier le type de point de terminaison d'une API

Utilisation de la console API Gateway pour modifier un type de point de terminaison d'API

Pour modifier le type de point de terminaison d'API de votre API, effectuez l'une des étapes suivantes :

Pour convertir un point de terminaison public de « régional » ou « optimisé pour la périphérie » et vice-versa

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Choisissez une API REST.
3. Choisissez Paramètres de l'API.
4. Dans la section Détails de l'API, choisissez Modifier.
5. Pour Type de point de terminaison d'API, sélectionnez Optimisé pour la périphérie ou Régional.
6. Sélectionnez Enregistrer les modifications.
7. Redéployez votre API afin que les modifications prennent effet.

Pour convertir un point de terminaison privé en point de terminaison régional

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Choisissez une API REST.
3. Modifiez la stratégie de ressources de votre API afin de supprimer toute mention des VPC ou des points de terminaison d'un VPC afin que les appels d'API provenant de l'extérieur de votre VPC ainsi que de l'intérieur de votre VPC aboutissent.

4. Choisissez Paramètres de l'API.
5. Dans la section Détails de l'API, choisissez Modifier.
6. Pour Type de point de terminaison d'API, sélectionnez Régional.
7. Sélectionnez Enregistrer les modifications.
8. Supprimez la stratégie de ressources de votre API.
9. Redéployez votre API afin que les modifications prennent effet.

Pour convertir un point de terminaison régional en point de terminaison privé

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Choisissez une API REST.
3. Créez une politique de ressources qui accorde l'accès à votre VPC ou à votre point de terminaison VPC. Pour plus d'informations, consultez ???.
4. Choisissez Paramètres de l'API.
5. Dans la section Détails de l'API, choisissez Modifier.
6. Pour Type de point de terminaison d'API, sélectionnez Privé.
7. (Facultatif) Pour les ID de point de terminaison VPC, sélectionnez les ID de point de terminaison VPC que vous souhaitez associer à votre API privée.
8. Sélectionnez Enregistrer les modifications.
9. Redéployez votre API afin que les modifications prennent effet.

Utilisez le AWS CLI pour modifier le type de point de terminaison d'une API

Pour utiliser le AWS CLI pour mettre à jour une API optimisée pour les périphériques dont l'ID d'API est le suivant*{api-id}*, appelez update-rest-apicomme suit :

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

La réponse positive contient un code de statut `200 OK` et une charge utile similaire à ce qui suit :

```
{
```

```
    "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"
}
```

Inversement, mettez à jour une API régionale vers une API optimisée pour les périphériques comme suit :

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

Comme il put-rest-apis'agit de mettre à jour les définitions d'API, il ne s'applique pas à la mise à jour d'un type de point de terminaison d'API.

Méthodes pour les API REST dans API Gateway

Dans API Gateway, une méthode d'API incarne une demande de méthode et une réponse de méthode. La configuration d'une méthode d'API vous permet de définir les actions qu'un client doit ou devrait exécuter pour soumettre une demande d'accès au serveur principal et les réponses que le client reçoit en retour. Pour l'entrée, vous pouvez choisir les paramètres de demande de méthode (ou une charge utile applicable) pour que le client fournisse les données obligatoires ou facultatives au moment de l'exécution. Pour la sortie, vous déterminez le code de statut, les en-têtes et le corps applicable de la réponse de la méthode en tant que cibles auxquelles mapper les données de réponse du backend, avant qu'elles ne soient renvoyés au client. Pour aider le développeur client à comprendre les comportements et les formats d'entrée et de sortie de votre API, vous pouvez documenter votre API et fournir des messages d'erreur appropriés pour les demandes non valides.

Une demande de méthode d'API de demande HTTP. Pour configurer la demande de méthode, vous configurez une méthode HTTP (ou verbe), le chemin d'accès à une ressource d'API, les en-têtes et les paramètres de chaîne de requête applicables. Vous configurez également une charge utile lorsque la méthode HTTP est `POST`, `PUT`ou `PATCH`. Par exemple, pour récupérer un animal de compagnie à l'aide de l'PetStore exemple d'API, vous définissez la demande de méthode d'API`GET /pets/{petId}`, où `{petId}` est un paramètre de chemin pouvant prendre un nombre au moment de l'exécution.

```
GET /pets/1
Host: apigateway.us-east-1.amazonaws.com
...
```

Si le client spécifie un chemin d'accès incorrect, par exemple `/pet/1` ou `/pets/one` au lieu de `/pets/1`, une exception est renvoyée.

Une réponse de méthode d'API est une réponse HTTP avec un code de statut donné. Dans le cas d'une intégration autre que de proxy, vous devez configurer les réponses de méthode afin de spécifier les cibles obligatoires ou facultatives des mappages. Cela permet de transformer les en-têtes ou corps de la réponse d'intégration en en-têtes ou corps de la réponse de méthode associée. Le mappage peut être aussi simple qu'une transformation d'identité qui transmet les en-têtes ou le corps via l'intégration en l'état. Par exemple, la réponse de méthode `200` suivante illustre un exemple de transfert d'une réponse d'intégration positive telle quelle.

```
200 OK
Content-Type: application/json
...

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

En principe, vous pouvez définir une réponse de méthode correspondant à une réponse spécifique à partir du backend. En général, cela implique n'importe quelle réponse 2XX, 4XX et 5XX. Toutefois, il se peut que ce ne soit pas pratique, car souvent, vous ne connaissez pas forcément à l'avance toutes les réponses qu'un backend peut renvoyer. En pratique, vous pouvez désigner une réponse de méthode par défaut pour gérer l'inconnu ou des réponses non mappées depuis le backend. Il est recommandé de désigner la réponse 500 comme valeur par défaut. Dans tous les cas, vous devez configurer au moins une réponse de méthode pour les intégrations autres que de proxy. Sinon, API Gateway renvoie une réponse d'erreur 500 au client, même lorsque la demande aboutit au niveau du backend.

Pour prendre en charge un kit SDK fortement typé, tel qu'un kit SDK Java, pour votre API, vous devez définir le modèle de données de l'entrée de la demande de méthode, ainsi que le modèle de données de la sortie de la réponse de méthode.

Prérequis

Avant de configurer une méthode d'API, vérifiez ce qui suit :

- La méthode doit être disponible dans API Gateway. Suivez les instructions de la section Tutoriel : Création d'une API REST avec une intégration HTTP autre que de proxy.
- Si vous souhaitez que la méthode communique avec une fonction Lambda, vous devez avoir déjà créé le rôle d'invocation Lambda et le rôle d'exécution Lambda dans IAM. Vous devez également avoir créé la fonction Lambda avec laquelle votre méthode pourra communiquer dans AWS Lambda. Pour créer les rôles et la fonction, suivez les instructions de Création d'une fonction Lambda pour l'intégration non proxy Lambda dans Choisissez un didacticiel AWS Lambda d'intégration.
- Si vous souhaitez que la méthode communique avec HTTP ou une intégration de proxy HTTP, vous devez déjà avoir créé l'URL de point de terminaison HTTP avec lequel votre méthode va communiquer, et y avoir accès.
- Vérifiez que vos certificats pour les points de terminaison HTTP et proxy HTTP sont pris en charge par API Gateway. Pour plus d'informations, consultez Autorités de certification prises en charge par API Gateway pour HTTP et les intégrations de proxy HTTP.

> **Note**
>
> Lorsque vous créez une méthode à l'aide de la console de l'API REST, vous configurez à la fois la demande d'intégration et la demande de méthode. Pour plus d’informations, consultez the section called “ Configuration d'une demande d'intégration à l'aide de la console”.

Rubriques

- Configuration d'une demande de méthode dans API Gateway
- Configuration des réponses de méthode dans API Gateway
- Configuration d'une méthode à l'aide de la console API Gateway

Configuration d'une demande de méthode dans API Gateway

La configuration d'une demande de méthode implique l'exécution des tâches suivantes, après avoir créé une RestApiressource :

1. Création d'une API ou sélection d'une entité Resource (Ressource) d'API existante.
2. Création d'une ressource Method d'API consistant en un verbe HTTP spécifique sur la nouvelle API ou l'entité `Resource` d'API sélectionnée. Cette tâche peut être également décomposée en plusieurs tâches secondaires comme suit :
   - Ajout d'une méthode HTTP pour la demande de méthode
   - Configuration des paramètres de la demande
   - Définition d'un modèle pour le corps de la demande
   - Application d'un schéma d'autorisation
   - Activation de la validation de la demande

Vous pouvez effectuer ces tâches à l'aide des méthodes suivantes :

- Console API Gateway
- AWS CLI commandes (create-resource et put-method)
- AWS Fonctions du SDK (par exemple, dans Node.js, CreateResource et PutMethod)
- API REST API Gateway (resource:create et method:put).

Rubriques

- Configuration des ressources API
- Configuration d'une méthode HTTP
- Configuration des paramètres de demande de méthode
- Configuration du modèle de demande de méthode
- Configuration de l'autorisation de demande de méthode
- Configuration de la validation de demande de méthode

Configuration des ressources API

Dans une API API Gateway, vous exposez les ressources adressables sous la forme d'une arborescence d'entités Resources (Ressources) de l'API, avec la ressource racine (`/`) au sommet de la hiérarchie. La ressource racine est relative à l'URL de base de l'API, qui se compose du point de terminaison de l'API et du nom de l'étape. Dans la console API Gateway, cette URI de base est appelée Invoke URI (URI d'appel) et s'affiche dans l'éditeur d'étape de l'API après le déploiement de l'API.

Le point de terminaison de l'API peut être un nom d'hôte par défaut ou un nom de domaine personnalisé. Le nom d'hôte par défaut a le format suivant :

```
{api-id}.execute-api.{region}.amazonaws.com
```

Dans ce format, *{api-id}* représente l'identificateur de l'API généré par API Gateway. La variable *{region}* représente la Région AWS (par exemple, `us-east-1`) que vous avez choisie lors de la création de l'API. Un nom de domaine personnalisé est un nom convivial sous un domaine Internet valide. Par exemple, si vous avez enregistré un domaine Internet du nom de `example.com`, n'importe quel `*.example.com` est un nom de domaine personnalisé valide. Pour plus d'informations, consultez Création d'un nom de domaine valide.

Pour l'PetStore exemple d'API, la ressource racine (`/`) expose l'animalerie. La ressource `/pets` désigne l'ensemble des animaux disponibles dans l'animalerie. `/pets/{petId}` expose un animal individuel d'un identifiant donné (`petId`). Le paramètre de chemin de `{petId}` est inclus dans les paramètres de la demande.

Pour configurer une ressource d'API, vous désignez une ressource existante comme son parent, puis vous créez la ressource enfant sous cette ressource parent. Vous commencez par la ressource racine en tant que parent, vous ajoutez une ressource à ce parent, vous ajoutez une autre ressource à cette ressource enfant en tant que nouveau parent, et ainsi de suite, jusqu'à son identificateur parent. Ensuite, vous ajoutez la ressource nommée au parent.

Avec AWS CLI, vous pouvez appeler la `get-resources` commande pour savoir quelles ressources d'une API sont disponibles :

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

Le résultat est une liste des ressources de l'API disponibles actuellement. Pour notre PetStore exemple d'API, cette liste se présente comme suit :

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

Chaque élément répertorie les identificateurs de la ressource (`id`) et, à l'exception de la ressource racine, son parent immédiat (`parentId`), ainsi que le nom de la ressource (`pathPart`). La ressource racine est spéciale, car elle n'a pas de parent. Après avoir choisi une ressource comme parent, appelez la commande suivante pour ajouter une ressource enfant.

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

Par exemple, pour ajouter de la nourriture pour animaux de compagnie à vendre sur le PetStore site Web, ajoutez une `food` ressource à la racine (/) en définissant « `path-part` à » `food` et « `parent-id` à `svzr2028x8` ». Le résultat se présente comme suit :

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

Utilisation d'une ressource de proxy pour rationnaliser la configuration de l'API

Au fur et à mesure que l'entreprise se développe, le PetStore propriétaire peut décider d'ajouter de la nourriture, des jouets et d'autres articles liés aux animaux de compagnie à vendre. Pour que cela soit pris en charge, vous pouvez ajouter `/food`, `/toys` et d'autres ressources sous la ressource racine. Sous chaque catégorie de vente, vous pouvez également ajouter d'autres ressources, telles que `/food/{type}/{item}`, `/toys/{type}/{item}`, etc. Cela peut être fastidieux. Si vous décidez d'ajouter une couche intermédiaire `{subtype}` aux chemins d'accès de la ressource afin de modifier la hiérarchie de chemin d'accès en `/food/{type}/{subtype}/{item}`, `/toys/{type}/{subtype}/{item}`, etc., les modifications rendront la configuration de l'API existante inopérante. Pour éviter cela, vous pouvez utiliser une ressource de proxy API Gateway pour exposer un jeu de ressources d'API simultanément.

API Gateway définit une ressource de proxy comme espace réservé afin qu'une ressource soit spécifiée lorsque la demande est envoyée. Une ressource de proxy est exprimée par un paramètre de chemin spécial, `{proxy+}`, souvent désigné comme un paramètre de chemin gourmand. Le signe + indique quelles ressources enfants y sont ajoutées. L'espace réservé `/parent/{proxy+}` représente toute ressource correspondant au modèle de chemin `/parent/*`. Le nom du paramètre de chemin gourmand, `proxy`, peut être remplacé par une autre chaîne comme s'il s'agissait d'un nom de paramètre de chemin standard.

À l'aide de AWS CLI, vous appelez la commande suivante pour configurer une ressource proxy sous le root (`/{proxy+}`) :

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

Le résultat est similaire à ce qui suit :

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

Pour l'exemple d'API `PetStore`, vous pouvez utiliser `/{proxy+}` pour représenter `/pets` et `/pets/{petId}`. Cette ressource proxy peut également faire référence à toute autre ressource

(existante ou to-be-added) `/food/{type}/{item}/toys/{type}/{item}`, telle que, etc.`/food/{type}/{subtype}/{item}`, ou`/toys/{type}/{subtype}/{item}`, etc. Le développeur du backend détermine la hiérarchie de ressource et le développeur client est responsable de sa compréhension. API Gateway ne fait que transmettre ce que le client a envoyé au backend.

Une API peut disposer de plus d'une ressource de proxy. Par exemple, les ressources de proxy suivantes sont autorisées au sein d'une API.

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

Lorsqu'une ressource de proxy dispose de sœurs autres que de proxy, les ressources sœurs sont exclues de la représentation de la ressource de proxy. Pour les exemples précédents, `/{proxy+}` fait référence à toutes les ressources situées sous la ressource racine, sauf les ressources `/parent[/*]`. En d'autres termes, une méthode de demande sur une ressource spécifique prime sur une méthode de demande sur une ressource générique au même niveau que la hiérarchie de ressource.

Une ressource de proxy ne peut pas disposer de ressource enfant. Toute ressource d'API située après `{proxy+}` est redondante et ambiguë. Les ressources de proxy suivantes ne sont pas autorisées au sein d'une API.

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

Configuration d'une méthode HTTP

Une demande de méthode d'API est encapsulée par la ressource Method d'API Gateway. Pour configurer la demande de méthode, vous devez d'abord instancier la ressource `Method`, et définir au moins une méthode HTTP et un type d'autorisation sur la méthode.

Étroitement associé à la ressource de proxy, API Gateway prend en charge une méthode HTTP `ANY`. Cette méthode `ANY` représente n'importe quelle méthode HTTP à fournir au moment de l'exécution. Cela vous permet d'utiliser une seule configuration de méthode d'API pour toutes les méthodes HTTP prises en charge, `DELETE`, `GET`, `HEAD`, `OPTIONS`, `PATCH`, `POST` et `PUT`.

Vous pouvez configurer la méthode `ANY` sur une ressource autre que de proxy également. L'association de la méthode `ANY` à une ressource de proxy vous permet d'obtenir une seule

configuration de méthode d'API pour toutes les méthodes HTTP prises en charge par rapport à toutes les ressources d'une API. De plus, le backend peut évoluer sans rendre la configuration d'API existante inopérante.

Avant de configurer une méthode d'API, prenez en compte qui peut appeler la méthode. Définissez le type d'autorisation en fonction de votre plan. Pour un accès ouvert, définissez-le sur `NONE`. Pour utiliser les autorisations IAM, définissez le type d'autorisation sur `AWS_IAM`. Pour utiliser une fonction du mécanisme d'autorisation Lambda, définissez cette propriété sur `CUSTOM`. Pour utiliser un groupe d'utilisateurs Amazon Cognito, définissez le type d'autorisation sur `COGNITO_USER_POOLS`.

La AWS CLI commande suivante montre comment créer une demande de méthode du `ANY` verbe pour une ressource spécifiée (`6sxz2j`), en utilisant les autorisations IAM pour contrôler son accès.

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

Pour créer une demande de méthode d'API avec un autre type d'autorisation, consultez the section called “Configuration de l'autorisation de demande de méthode”.

Configuration des paramètres de demande de méthode

Les paramètres de demande de méthode permettent à un client de fournir des données d'entrée ou un contexte d'exécution nécessaires pour terminer la demande de méthode. Un paramètre de méthode peut être un paramètre de chemin, un en-tête ou un paramètre de chaîne de requête. Dans le cadre de la configuration de la demande de méthode, vous devez déclarer les paramètres de demande obligatoires pour les rendre disponibles pour le client. Pour l'intégration autre que de proxy, vous pouvez traduire ces paramètres de demande en un formulaire compatible avec les exigences du serveur principal.

Par exemple, pour la demande de méthode `GET /pets/{petId}`, la variable de chemin d'accès `{petId}` est un paramètre de demande obligatoire. Vous pouvez déclarer ce paramètre de chemin lors de l'appel de la commande `put-method` de l' AWS CLI. Ceci est illustré comme suit :

```
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
```

Si un paramètre n'est pas obligatoire, vous pouvez le définir sur `false` dans `request-parameters`. Par exemple, si la méthode `GET /pets` utilise un paramètre de chaîne de requêtes facultatif `type` et un paramètre d'en-tête facultatif `breed`, vous pouvez les déclarer à l'aide de la commande CLI suivante, en supposant que la ressource `/pets` `id` est `6sxz2j` :

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

À la place de cette forme abrégée, vous pouvez utiliser une chaîne JSON pour définir la valeur `request-parameters` :

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

Grâce à cette configuration, le client peut interroger les animaux par type :

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

De plus, le client peut interroger les chiens de la race des caniches comme suit :

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

Pour plus d'informations sur le mappage des paramètres de demande de méthode aux paramètres de demande d'intégration, consultez the section called “Intégrations”.

Configuration du modèle de demande de méthode

Pour obtenir une méthode API pouvant accepter des données d'entrée dans une charge utile, vous pouvez utiliser un modèle. Un modèle est exprimé dans un schéma JSON version 4 et décrit la structure des données du corps de la demande. Avec un modèle, un client peut déterminer de quelle manière créer une charge utile de demande de méthode en tant qu'entrée. Qui plus est, API Gateway

utilise le modèle pour valider une demande, générer un kit SDK et initialiser un modèle de mappage pour configurer l'intégration dans la console API Gateway. Pour obtenir des informations sur la façon de créer un modèle, consultez Comprendre les modèles de données.

En fonction des différents types de contenu, une méthode de charge utile peut avoir différents formats. Un modèle est indexé en fonction du type de média de la charge utile appliquée. API Gateway utilise l'en-tête de `Content-Type` requête pour déterminer le type de contenu. Pour configurer des modèles de demande de méthode, ajoutez des paires clé-valeur du "*<media-type>*":"*<model-name>*" format à la `requestModels` carte lors de l'appel de la AWS CLI `put-method` commande.

Pour utiliser le même modèle quel que soit le type de contenu, spécifiez `$default` comme clé.

Par exemple, pour définir un modèle sur la charge utile JSON de la demande de `POST /pets` méthode de l'API d' PetStore exemple, vous pouvez appeler la AWS CLI commande suivante :

```
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"}'
```

Dans cet exemple, `petModel` correspond à la valeur de la propriété `name` d'une ressource `Model` décrivant un animal. La définition de schéma réelle est exprimée sous forme d'une valeur de chaîne JSON de la propriété `schema` de la ressource `Model`.

Dans Java, ou dans d'autres kits SDK fortement typés, de l'API, les données d'entrée sont exprimées en tant que classe `petModel` dérivée de la définition du schéma. Avec le modèle de demande, les données d'entrée dans le kit SDK généré sont intégrées à la classe `Empty`, qui est dérivée du modèle `Empty` par défaut. Dans ce cas, le client ne peut pas instancier la classe de données correcte pour fournir l'entrée requise.

Configuration de l'autorisation de demande de méthode

Pour contrôler qui peut appeler la méthode d'API, vous pouvez configurer le type d'autorisation sur la méthode. Vous pouvez utiliser ce type pour activer l'un des mécanismes d'autorisation pris en

charge, notamment les rôles et stratégies IAM (`AWS_IAM`), un groupe d'utilisateurs Amazon Cognito (`COGNITO_USER_POOLS`) ou un mécanisme d'autorisation Lambda (`CUSTOM`).

Pour utiliser les autorisations IAM afin d'autoriser l'accès à la méthode d'API, définissez la propriété d'entrée `authorization-type` sur **`AWS_IAM`**. Lorsque vous définissez cette option, API Gateway vérifie la signature de l'appelant dans la demande en fonction des informations d'identification de l'appelant. Si l'utilisateur vérifié est autorisé à appeler la méthode, la demande est acceptée. Sinon, la demande est rejetée et l'appelant reçoit une réponse d'erreur de non-autorisation. L'appel à la méthode n'aboutit pas, à moins que l'appelant soit autorisé à appeler la méthode d'API. La politique IAM suivante accorde à l'appelant l'autorisation d'appeler les méthodes d'API créées dans le même Compte AWS :

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

Pour plus d'informations, consultez the section called “Utilisation des autorisations IAM”.

Pour l'heure, vous pouvez uniquement accorder cette politique aux utilisateurs, groupes et rôles relevant du Compte AWS du propriétaire de l'API. Les utilisateurs d'un autre Compte AWS pays ne peuvent appeler les méthodes de l'API que s'ils sont autorisés à assumer un rôle au sein du propriétaire de l'API Compte AWS avec les autorisations nécessaires pour lancer l'`execute-api:Invoke`action. Pour de plus amples informations sur les autorisations entre comptes, veuillez consulter Utilisation des rôles IAM.

Vous pouvez utiliser AWS CLI un AWS SDK ou un client d'API REST, tel que Postman, qui implémente la signature Signature Version 4 (SigV4).

Pour utiliser un mécanisme d'autorisation Lambda permettant d'accéder à la méthode d'API, définissez la propriété d'entrée `authorization-type` sur `CUSTOM` et définissez la propriété d'entrée `authorizer-id` sur la valeur de propriété `id` d'un mécanisme d'autorisation Lambda qui

existe déjà. Le mécanisme d'autorisation Lambda référencé peut être de type `TOKEN` ou `REQUEST`. Pour de plus amples informations sur la création d'un mécanisme d'autorisation Lambda, veuillez consulter the section called “Utilisation des mécanismes d'autorisation Lambda”.

Pour utiliser un groupe d'utilisateurs Amazon Cognito permettant d'accéder à la méthode d'API, définissez la propriété d'entrée `authorization-type` sur `COGNITO_USER_POOLS` et définissez la propriété d'entrée `authorizer-id` sur la valeur de propriété `id` du mécanisme d'autorisation `COGNITO_USER_POOLS` qui a déjà été créé. Pour de plus amples informations sur la création d'un mécanisme d'autorisation de groupe d'utilisateurs Amazon Cognito, veuillez consulter the section called “Utilisation d'un groupe d'utilisateurs Amazon Cognito en tant que mécanisme d'autorisation pour une API REST”.

Configuration de la validation de demande de méthode

Vous pouvez activer la validation de demande lors de la configuration d'une demande de méthode d'API. Pour cela, commencez par créer un validateur de demande :

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

Cette commande CLI crée un validateur de demande de corps uniquement. L'exemple de sortie se présente comme suit :

```
{
    "validateRequestParameters": false,
    "validateRequestBody": true,
    "id": "jgpyy6",
    "name": "bodyOnlyValidator"
}
```

Grâce à ce validateur de demande, vous pouvez activer la validation de demande dans le cadre de la configuration de la demande de méthode :

```
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
```

Pour qu'il soit inclus dans la validation de demande, un paramètre de demande doit être déclaré comme obligatoire. Si le paramètre de chaîne de requête de la page est utilisé dans la validation de demande, le mappage `request-parameters` de l'exemple précédent doit être spécifié comme `'{"method.request.querystring.type": false, "method.request.querystring.page":true}'`.

Configuration des réponses de méthode dans API Gateway

Une réponse de méthode d'API encapsule la sortie d'une demande de méthode d'API que le client recevra. Les données de sortie incluent un code de statut HTTP, certains en-têtes et éventuellement un corps.

Avec les intégrations autres que de proxy, les paramètres et le corps de réponse spécifiés peuvent être mappés à partir des données de la réponse d'intégration associés ou se voir attribuer certaines valeurs statiques en fonction des mappages. Ces mappages sont spécifiés dans la réponse d'intégration. Le mappage peut être une transformation identique qui transfère la réponse d'intégration telle quelle.

Avec une intégration de proxy, API Gateway transfère automatiquement la réponse du backend par le biais de la réponse de méthode. Il n'est pas nécessaire de configurer la réponse de méthode d'API. Toutefois, avec l'intégration de proxy Lambda, la fonction Lambda doit renvoyer un résultat dans ce format de sortie pour qu'API Gateway puisse mapper avec succès la réponse d'intégration à une réponse de méthode.

Du point de vue de la programmation, la configuration de la réponse de la méthode revient à créer une MethodResponseressource d'API Gateway et à définir les propriétés de StatusCode, ResponseParameters et ResponseModels.

Lors de la définition des codes de statut pour une méthode d'API, vous devez en choisir par défaut pour gérer une réponse d'intégration quelconque de code de statut inattendu. Il est possible de définir `500` comme valeur par défaut, car cela revient à lancer des réponses non mappées comme une erreur côté serveur. Pour des raisons pédagogiques, la console API Gateway définit la réponse `200` comme valeur par défaut. Mais vous pouvez réinitialiser cette valeur sur la réponse `500`.

Pour configurer une réponse de méthode, vous devez avoir créé la demande de méthode.

Configuration du code de statut de réponse de méthode

Le code de statut d'une réponse de méthode définit un type de réponse. Par exemple, les réponses de 200, 400 et 500 indiquent des réponses positives, d'erreur côté client et d'erreur côté serveur, respectivement.

Pour configurer un code de statut de réponse de méthode, définissez la propriété `statusCode` sur un code de statut HTTP. La commande d' AWS CLI suivante crée une méthode de réponse de `200`.

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

Configuration des paramètres de réponse de méthode

Les paramètres de méthode de réponse définissent les en-têtes que le client reçoit en réponse à la demande de méthode associée. Les paramètres de réponse spécifient également une cible vers laquelle API Gateway mappe un paramètre de réponse d'intégration, en fonction des mappages prescrits dans la réponse d'intégration de la méthode d'API.

Pour configurer les paramètres de réponse de la méthode, ajoutez au mappage `responseParameters` des paires clé/valeur `MethodResponse` au format `"{parameter-name}":"{boolean}"`. La commande CLI suivante montre un exemple de définition de l'`my-header`en-tête.

```
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
```

Configuration des modèles de réponse de méthode

Un modèle de réponse de méthode définit un format pour le corps de la réponse de méthode. Avant de configurer le modèle de réponse, vous devez d'abord créer le modèle dans API Gateway. Pour cela, vous pouvez appeler la commande `create-model`. L'exemple suivant montre comment créer un modèle `PetStorePet` pour décrire le corps de la réponse dans la demande de méthode `GET /pets/{petId}`.

```
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" } \
                } \
             }'
```

Le résultat est créé en tant que ressource API Gateway `Model`.

Pour configurer les modèles de réponse de méthode afin de définir le format de charge utile, ajoutez la paire clé-valeur « application/json » : » PetStorePet "à la `requestModels`carte des ressources. `MethodResponse` La AWS CLI commande suivante `put-method-response` montre comment procéder :

```
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"}'
```

La configuration d'un modèle de réponse de méthode est nécessaire lorsque vous générez un kit SDK fortement typé pour l'API. Cela garantit que la sortie est lancée dans une classe correspondante dans Java ou Objective-C. Dans d'autres cas, la configuration d'un modèle est facultative.

Configuration d'une méthode à l'aide de la console API Gateway

Lorsque vous créez une méthode à l'aide de la console de l'API REST, vous configurez à la fois la demande d'intégration et la demande de méthode. Par défaut, API Gateway crée la réponse de `200` méthode pour votre méthode.

Les instructions suivantes indiquent comment modifier les paramètres de demande de méthode et comment créer des réponses de méthode supplémentaires pour votre méthode.

Rubriques

- Modifier une demande de méthode API Gateway dans la console API Gateway
- Configuration d'une réponse de méthode API Gateway à l'aide de la console API Gateway

Modifier une demande de méthode API Gateway dans la console API Gateway

Ces instructions supposent que vous avez déjà créé votre demande de méthode. Pour plus d'informations sur la création d'une méthode, consultezthe section called “ Configuration d'une demande d'intégration à l'aide de la console”.

1. Dans le volet Ressources, choisissez votre méthode, puis cliquez sur l'onglet Demande de méthode.
2. Dans la section Paramètres de la demande de méthode, choisissez Modifier.
3. Pour Autorisation, sélectionnez un mécanisme d'autorisation disponible.

    a. Pour activer l'accès ouvert à la méthode pour n'importe quel utilisateur, choisissez Aucun. Cette étape peut être ignorée si le paramètre par défaut n'a pas été modifié.

    b. Pour utiliser les autorisations IAM afin de contrôler l'accès client à la méthode, sélectionnez `AWS_IAM`. Avec ce choix, seuls les utilisateurs des rôles IAM avec la bonne stratégie IAM attachée sont autorisés à appeler cette méthode.

    Pour créer le rôle IAM, spécifiez une stratégie d'accès dans un format similaire à ce qui suit :

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

```
        "execute-api:Invoke"
      ],
      "Resource": [
        "resource-statement"
      ]
    }
  ]
}
```

Dans cette politique d'accès, *resource-statement* est l'ARN de votre méthode. Vous pouvez trouver l'ARN de votre méthode en sélectionnant la méthode sur la page Ressources. Pour en savoir plus sur la définition d'autorisations IAM, consultez Contrôle de l'accès à une API avec des autorisations IAM.

Pour créer le rôle IAM, vous pouvez adapter les instructions du didacticiel suivant,???.

c. Pour utiliser un mécanisme d'autorisation Lambda, sélectionnez un jeton ou un mécanisme d'autorisation de demande. Créez un mécanisme d'autorisation Lambda pour que ce choix s'affiche dans le menu déroulant. Pour de plus amples informations sur la création d'un mécanisme d'autorisation Lambda, veuillez consulter Utilisation des mécanismes d'autorisation Lambda API Gateway.

d. Pour utiliser un groupe d'utilisateurs Amazon Cognito, choisissez un groupe d'utilisateurs disponible sous Cognito user pool authorizers (Mécanismes d'autorisation du groupe d'utilisateurs Cognito). Créez un groupe d'utilisateurs dans Amazon Cognito, ainsi qu'un mécanisme d'autorisation de groupe d'utilisateurs Amazon Cognito dans API Gateway pour que ce choix s'affiche dans le menu déroulant. Pour de plus amples informations sur la manière de créer un mécanisme d'autorisation de groupe d'utilisateurs Amazon Cognito, veuillez consulter Contrôle de l'accès à une API REST à l'aide de groupes d'utilisateurs Amazon Cognito en tant que mécanisme d'autorisation.

4. Pour spécifier la validation des demandes, sélectionnez une valeur dans le menu déroulant Validateur de demande. Pour désactiver la validation des demandes, sélectionnez Aucune. Pour plus d'informations concernant chaque option, consultez Utilisation de la validation des demandes dans API Gateway.

5. Sélectionnez Clé API obligatoire pour exiger une clé API. Lorsque cette option est activée, les clés d'API sont utilisées dans les plans d'utilisation afin de limiter le trafic client.

6. (Facultatif) Pour attribuer un nom d'opération dans un kit SDK Java de cette API généré par API Gateway, entrez un nom pour Nom de l'opération. Par exemple, pour la demande de méthode `GET /pets/{petId}`, le nom d'opération du kit SDK Java est, par défaut, `GetPetsPetId`.

Ce nom est créé à partir du verbe HTTP der la méthode (`GET`) et des noms de variables du chemin de la ressource (`Pets` et `PetId`). Si vous définissez le nom de l'opération en tant que `getPetById`, le nom d'opération du kit SDK devient `GetPetById`.

7. Pour ajouter un paramètre de chaîne de requête à la méthode, procédez comme suit :

   a. Choisissez Paramètres de chaîne de requête d'URL, puis Ajouter une chaîne de requête.

   b. Pour Nom, entrez le nom du paramètre de chaîne de requête.

   c. Sélectionnez Obligatoire si le paramètre de chaîne de requête récemment créé doit être utilisé pour une validation de demande. Pour plus d'informations sur la validation de demande, consultez Utilisation de la validation des demandes dans API Gateway.

   d. Sélectionnez Mise en cache si le paramètre de chaîne de requête récemment créé doit être utilisé dans le cadre d'une clé de mise en cache. Pour plus d'informations sur la mise en cache, consultez Utilisation de paramètres de méthode ou d'intégration en tant que clés de cache pour indexer les réponses mises en cache.

   Pour supprimer le paramètre de chaîne de requête, choisissez Supprimer.

8. Pour ajouter un paramètre d'en-tête à la méthode, procédez comme suit :

   a. Choisissez En-têtes de demande HTTP, puis Ajouter un en-tête.

   b. Pour Nom, entrez le nom de l'en-tête.

   c. Sélectionnez Obligatoire si l'en-tête récemment créé doit être utilisé pour une validation de demande. Pour plus d'informations sur la validation de demande, consultez Utilisation de la validation des demandes dans API Gateway.

   d. Sélectionnez Mise en cache si l'en-tête récemment créé doit être utilisé dans le cadre d'une clé de mise en cache. Pour plus d'informations sur la mise en cache, consultez Utilisation de paramètres de méthode ou d'intégration en tant que clés de cache pour indexer les réponses mises en cache.

   Pour supprimer l'en-tête, choisissez Supprimer.

9. Pour déclarer le format de charge utile d'une demande de méthode avec le verbe HTTP `POST`, `PUT` ou `PATCH`, choisissez Corps de la demande et procédez comme suit :

   a. Choisissez Add model.

   b. Pour Content-Type, entrez un type MIME (par exemple `application/json`).

c. Pour Modèle, sélectionnez un modèle dans le menu déroulant. Les modèles disponibles actuellement pour l'API incluent les modèles `Empty` et `Error` par défaut, ainsi que tous les modèles que vous avez créés et ajoutés à la collection Models de l'API. Pour plus d'informations sur la création d'un modèle, consultez Comprendre les modèles de données.

> **Note**
>
> Le modèle est utile pour informer le client du format de données prévu pour une charge utile. Il permet également de générer un modèle de mappage squelettique. Il est important de générer un kit SDK fortement typé de l'API dans des langages comme Java, C#, Objective-C et Swift. Ceci est nécessaire uniquement si la validation de demande est activée pour la charge utile.

10. Choisissez Enregistrer.

Configuration d'une réponse de méthode API Gateway à l'aide de la console API Gateway

Une méthode d'API peut avoir un ou plusieurs réponses. Chaque réponse est indexée selon son code de statut HTTP. Par défaut, la console API Gateway ajoute la réponse `200` aux réponses de méthode. Vous pouvez modifier cela, par exemple pour que la méthode renvoie plutôt la réponse `201`. Vous pouvez ajouter d'autres réponses, par exemple, `409` en cas d'accès refusé et `500` si des variables d'étape non initialisées sont utilisées.

Pour utiliser la console API Gateway pour modifier, supprimer ou ajouter une réponse à une méthode d'API, suivez les instructions suivantes.

1. Dans le volet Ressources, choisissez votre méthode, puis cliquez sur l'onglet Réponse de la méthode. Vous devrez peut-être choisir la flèche droite pour afficher l'onglet.
2. Dans la section Paramètres de réponse de la méthode, choisissez Créer une réponse.
3. Pour Code de statut HTTP, entrez un code de statut HTTP, tel que `200`, `400` ou `500`.

   Lorsqu'une réponse renvoyée par le backend n'a pas de méthode de réponse correspondante définie, API Gateway échoue à renvoyer la réponse au client. Au lieu de cela, une réponse d'erreur `500 Internal server error` est renvoyée.
4. Sélectionnez Add header.
5. Pour Nom de l'en-tête, entrez un nom.

Pour renvoyer un en-tête du backend au client, ajoutez-le dans la réponse de la méthode.

6. Choisissez Ajouter un modèle pour définir un format pour le corps de la réponse de méthode.

   Entrez le type de média de la charge utile de la réponse pour Type de contenu et choisissez un modèle dans le menu déroulant Modèles.

7. Choisissez Enregistrer.

Pour modifier une réponse existante, accédez à la réponse de votre méthode, puis choisissez Modifier. Pour modifier le Code de statut HTTP, choisissez Supprimer et créez une nouvelle méthode de réponse.

Pour chaque réponse renvoyée par le backend, vous devez disposer d'une réponse compatible configurée comme réponse de méthode. Toutefois, le modèle de charge utile et d'en-têtes de réponse de méthode de configuration sont facultatifs, sauf si vous mappez le résultat à partir du backend à la réponse de méthode avant de revenir au client. De plus, un modèle de charge utile de réponse de méthode est important si vous générez un kit SDK fortement typé pour votre API.

Contrôle et gestion de l'accès à une API REST dans API Gateway

API Gateway prend en charge plusieurs mécanismes pour contrôler et gérer l'accès à votre API.

Les mécanismes suivants peuvent être utilisés pour l'authentification et l'autorisation :

- Les stratégies de ressources vous permettent de créer des stratégies basées sur les ressources afin d'autoriser ou de refuser l'accès à vos API et méthodes à certaines adresses IP source spécifiées ou points de terminaison d'un VPC. Pour plus d’informations, consultez the section called “Utilisation des stratégies de ressources API Gateway”.
- Les rôles et politiques AWS IAM standard offrent des contrôles d'accès flexibles et robustes qui peuvent être appliqués à une API complète ou à des méthodes individuelles. Les rôles et les stratégies IAM peuvent être utilisés afin de contrôler qui peut créer et gérer vos API, ainsi que qui peut les appeler. Pour plus d’informations, consultez the section called “Utilisation des autorisations IAM”.
- Les balises IAM peuvent être utilisées avec des stratégies IAM pour contrôler l'accès. Pour plus d’informations, consultez the section called “Contrôle d'accès basé sur les attributs”.
- Les stratégies de point de terminaison pour les points de terminaison de VPC d'interface vous permettent d'attacher des stratégies de ressources IAM à des points de terminaison de VPC

d'interface pour améliorer la sécurité de vos API privées. Pour plus d'informations, consultez the section called "Utilisation de stratégies de point de terminaison de VPC pour des API privées".

- Les mécanismes d'autorisation Lambda sont des fonctions Lambda qui contrôlent l'accès à vos méthodes d'API REST à l'aide d'authentification par jeton de porteur, ainsi qu'aux informations figurant dans les paramètres de la requête tels que les en-têtes, chemins, chaînes de requête, variables d'étape ou variables de contexte. Les autorisateurs Lambda sont utilisés pour contrôler qui peut appeler les méthodes d'API REST. Pour plus d'informations, consultez the section called "Utilisation des mécanismes d'autorisation Lambda".
- Les groupes d'utilisateurs Amazon Cognito vous permettent de créer des solutions d'authentification et d'autorisation personnalisables pour vos API REST. Les groupes d'utilisateurs Amazon Cognito sont utilisés pour contrôler qui peut appeler les méthodes d'API REST. Pour plus d'informations, consultez the section called "Utilisation d'un groupe d'utilisateurs Amazon Cognito en tant que mécanisme d'autorisation pour une API REST".

Les mécanismes suivants peuvent être utilisés pour effectuer d'autres tâches liées au contrôle de l'accès :

- Le partage des ressources cross-origin (CORS) vous permet de contrôler la façon dont votre API REST répond aux requêtes de ressources inter-domaines. Pour plus d'informations, consultez the section called "CORS".
- Les certificats SSL côté client peuvent être utilisés pour vérifier que les requêtes HTTP adressées à votre système backend proviennent d'API Gateway. Pour plus d'informations, consultez the section called "Certificats clients".
- AWS WAF peut être utilisé pour protéger votre API d'API Gateway contre les menaces web courantes. Pour plus d'informations, consultez the section called "AWS WAF".

Les mécanismes suivants peuvent être utilisés pour suivre et limiter l'accès que vous avez accordé aux clients autorisés :

- Les plans d'utilisation vous permettent de fournir des clés d'API à vos clients. Vous pouvez ensuite suivre et limiter l'utilisation de vos étapes et méthodes d'API pour chaque clé d'API. Pour plus d'informations, voir the section called "Plans d'utilisation".

Contrôle d'accès à une API avec des stratégies de ressources API Gateway

Les stratégies de ressources Amazon API Gateway sont des documents de politique JSON que vous attachez à une API pour contrôler si un principal spécifié (généralement un groupe ou un rôle IAM) peut appeler l'API. Vous pouvez utiliser des stratégies de ressources API Gateway pour autoriser l'appel sécurisé de votre API par :

- Utilisateurs d'un AWS compte spécifié.
- les plages d'adresses IP ou les blocs d'adresse CIDR d'un source spécifiée ;
- des clouds privés virtuels (VPC, Virtual Private Cloud) ou des points de terminaison d'un VPC spécifiés (dans n'importe quel compte).

Vous pouvez joindre une politique de ressources pour n'importe quel type de point de terminaison d'API dans API Gateway à l' AWS Management Console aide de la AWS CLI ou AWS des SDK. Pour les API privées, vous pouvez utiliser des stratégies de ressources avec les stratégies de point de terminaison de VPC pour contrôler l'accès des mandataires aux ressources et aux actions. Pour plus d'informations, consultez the section called “Utilisation de stratégies de point de terminaison de VPC pour des API privées”.

Les stratégies de ressources API Gateway sont différentes des stratégies basées sur l'identité IAM. Les stratégies basées sur l'identité IAM sont attachées à des utilisateurs, groupes ou rôles IAM et définissent les actions que ces identités sont capables d'exécuter, ainsi que les ressources sur lesquelles peuvent porter ces actions. Les stratégies de ressource API Gateway sont attachées à des ressources. Vous pouvez utiliser des stratégies de ressources API Gateway avec les stratégies IAM. Pour plus d'informations, consultez Stratégies basées sur l'identité et Stratégies basées sur une ressource.

Rubriques

Présentation du langage d'access policy pour Amazon API Gateway

Cette page décrit les éléments de base utilisés dans les stratégies de ressources Amazon API Gateway.

Les stratégies de ressources sont spécifiés en utilisant la même syntaxe que dans les stratégies IAM. Pour obtenir des informations complètes sur le langage de politique d'accès, veuillez consulter Présentation des politiques IAM et Référence des stratégies AWS Identity and Access Management dans le Guide de l'utilisateur IAM.

Pour plus d'informations sur la manière dont un AWS service décide si une demande donnée doit être autorisée ou refusée, voir Déterminer si une demande est autorisée ou refusée.

Éléments courants dans une stratégie d'accès

Une stratégie de ressources contient à la base les éléments suivants :

- Ressources – Les API sont les ressources Amazon API Gateway pour lesquelles vous pouvez accorder ou refuser des autorisations. Dans une stratégie, vous pouvez utiliser l'Amazon Resource Name (ARN) pour identifier la ressource. Vous pouvez également utiliser une syntaxe abrégée, qui sera automatiquement développée par API Gateway en ARN complet lors de l'enregistrement d'une stratégie de ressources. Pour en savoir plus, veuillez consulter la section Exemples de stratégies de ressources API Gateway.

  Pour connaître le format de l'élément `Resource` complet, veuillez consulter Format de l'expression Resource des autorisations d'exécution d'API dans API Gateway.
- Actions – pour chaque ressource, Amazon API Gateway prend en charge un ensemble d'opérations. Vous identifiez les opérations de ressource que vous accordez (ou refusez) en utilisant des mots clés d'action.

  Par exemple, l'autorisation `execute-api:Invoke` va permettre à l'utilisateur d'appeler une API sur demande d'un client.

  Pour connaître le format de l'élément `Action`, consultez Format de l'expression Action des autorisations d'exécution d'API dans API Gateway.
- Effet – L'effet produit lorsque l'utilisateur demande une action spécifique. Il peut s'agir de `Allow` ou `Deny`. Vous pouvez aussi explicitement refuser l'accès à une ressource, ce que vous pouvez faire afin de vous assurer qu'un utilisateur n'y a pas accès, même si une stratégie différente octroie l'accès.

> **Note**
>
> « Refus implicite » est la même chose que « Refus par défaut ».
> Un « refus implicite » est différent d'un « refus explicite ». Pour de plus amples informations, veuillez consulter Différence entre refus par défaut et refus explicite.

- Principal – Compte ou utilisateur autorisé à accéder aux actions ou ressources dans l'instruction. Dans une stratégie de ressources, le principal est l'utilisateur ou le compte qui reçoit cette autorisation.

L'exemple de stratégie de ressources suivant montre les éléments de stratégie courants précédemment évoqués. La stratégie octroie l'accès à toutes les API sous l'ID *account-id* dans la *région* spécifiée à tous les utilisateurs dont l'adresse IP source se trouve dans le bloc d'adresses *123.4.5.6/24*. La stratégie refuse tout accès à l'API si l'adresse IP source de l'utilisateur n'est pas comprise dans cette plage.

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

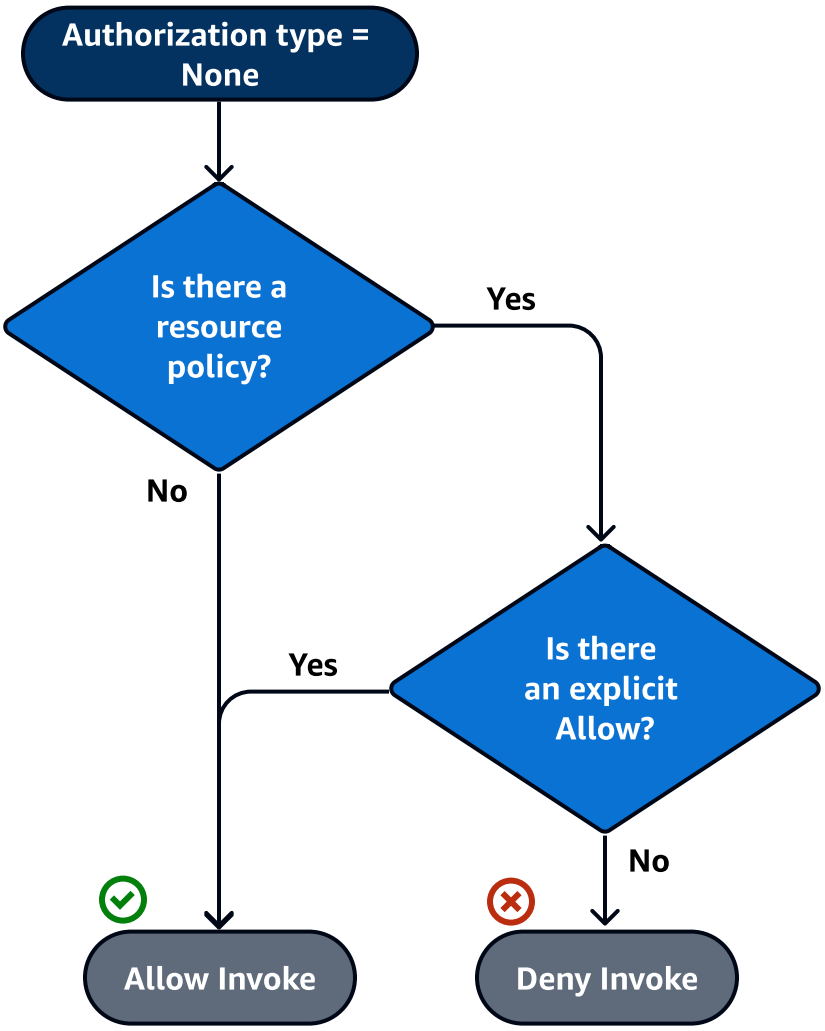
Comment les stratégies de ressources API Gateway affectent le flux de travail d'autorisation

Lorsque API Gateway évalue la stratégie de ressources attachée à votre API, le résultat est affecté par le type d'authentification que vous avez défini pour l'API, comme illustré dans les diagrammes présentés dans les sections suivantes.

Rubriques

Stratégie de ressources API Gateway uniquement

Dans ce flux de travail, une stratégie de ressources API Gateway est attachée à l'API, mais aucun type d'authentification n'est défini pour l'API. L'évaluation de la stratégie consiste à obtenir une autorisation explicite en fonction des critères entrants de l'appelant. Un refus implicite ou explicite entraîne le refus de l'appelant.



Voici un exemple d'une telle stratégie de ressources.

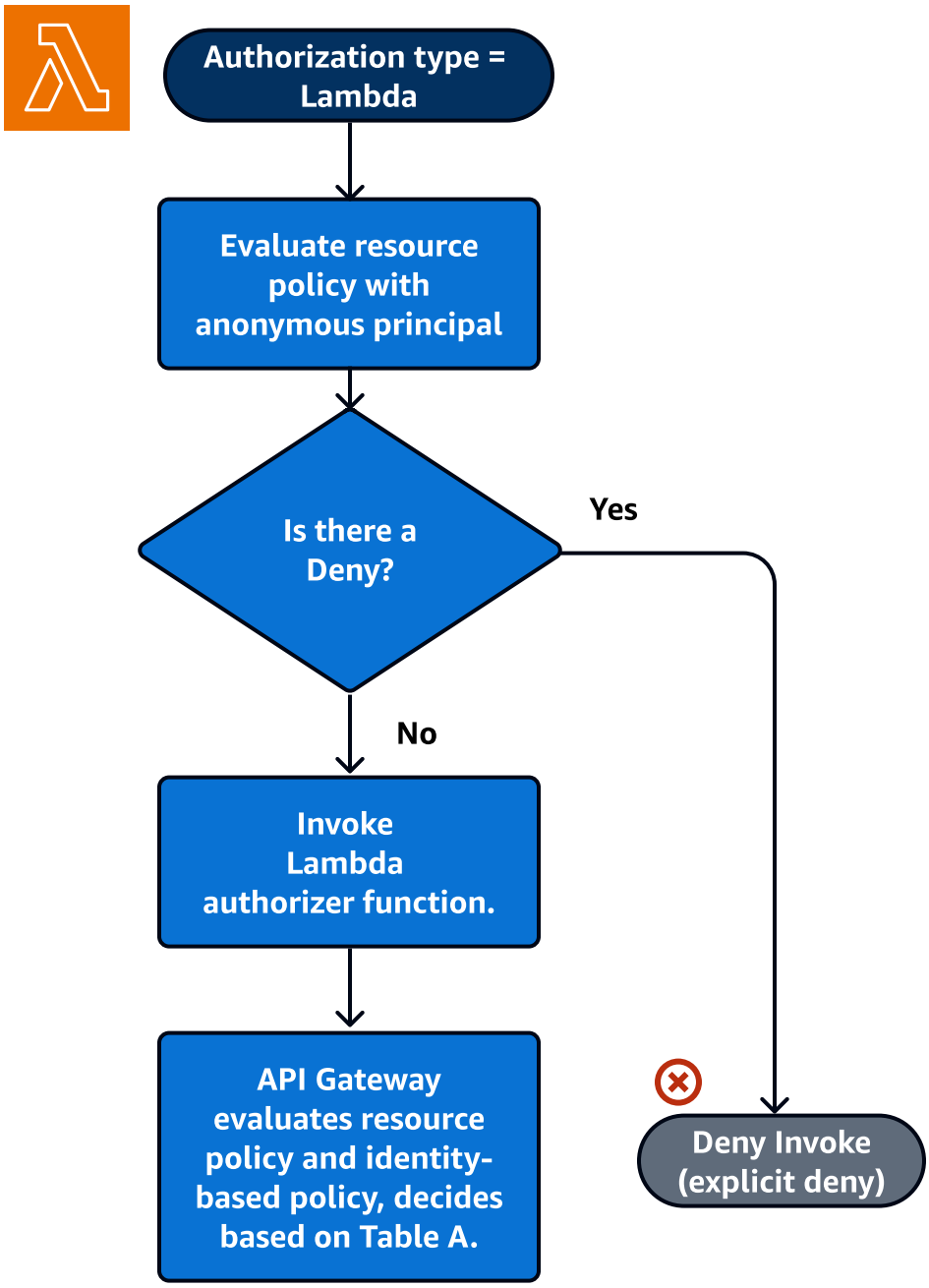
```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": "*",
            "Action": "execute-api:Invoke",
            "Resource": "arn:aws:execute-api:region:account-id:api-id/",
            "Condition": {
                "IpAddress": {
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

```
        }
    ]
}
```

Mécanisme d'autorisation et stratégie de ressources Lambda

Dans ce flux de travail, un mécanisme d'autorisation Lambda est configuré pour l'API en plus d'une stratégie de ressources. La stratégie de ressources est évaluée en deux phases. Avant d'appeler le mécanisme d'autorisation Lambda, API Gateway évalue la stratégie et vérifie tout refus explicite. S'il en trouve, l'accès sera immédiatement refusé à l'utilisateur. Sinon, le mécanisme d'autorisation Lambda est appelé et renvoie un document de stratégie évalué avec la stratégie de ressources. Le résultat est déterminé sur la base du tableau A.

Dans l'exemple de stratégie de ressources suivant, les appels sont autorisés uniquement à partir du point de terminaison de VPC dont l'ID est *vpce-1a2b3c4d*. Pendant l'évaluation d'authentification préalable, seuls les appels provenant du point de terminaison de VPC indiqué dans l'exemple peuvent continuer et évaluer le mécanisme d'autorisation Lambda. Tous les appels restants sont bloqués.




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

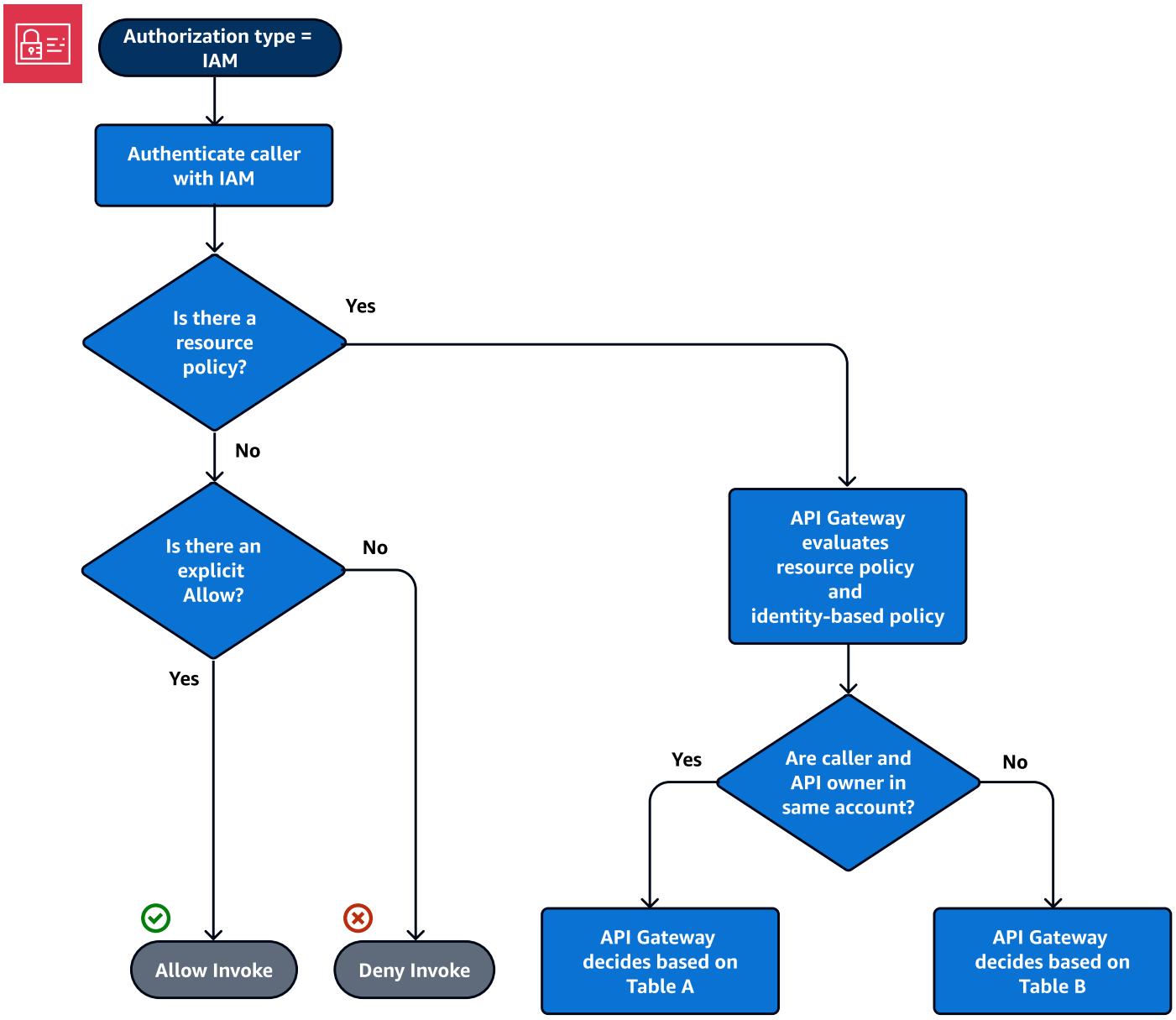
```
                "arn:aws:execute-api:region:account-id:api-id/"
            ],
            "Condition" : {
                "StringNotEquals": {
                    "aws:SourceVpce": "vpce-1a2b3c4d"
                }
            }
        }
    ]
}
```

Stratégie de ressources et d'authentification IAM

Dans ce flux de travail, vous configurez l'authentification IAM pour l'API en plus d'une stratégie de ressources. Une fois que vous avez authentifié l'utilisateur auprès du service IAM, l'API évalue à la fois les stratégies attachées à l'utilisateur et la stratégie de ressources. Le résultat varie selon que l'appelant est le même que le propriétaire de l'API Compte AWS ou qu'il est distinct Compte AWS de celui qui l'appelle.

Si l'appelant et le propriétaire de l'API relèvent de comptes distincts, les politiques IAM et la stratégie de ressources autorisent toutes deux explicitement l'appelant à continuer. Pour plus d'informations, consultez le tableau B.

Cependant, si l'appelant et le propriétaire de l'API relèvent du même Compte AWS, les politiques utilisateur IAM ou la stratégie de ressources doivent explicitement autoriser l'appelant à continuer. Pour plus d'informations, consultez le tableau A.



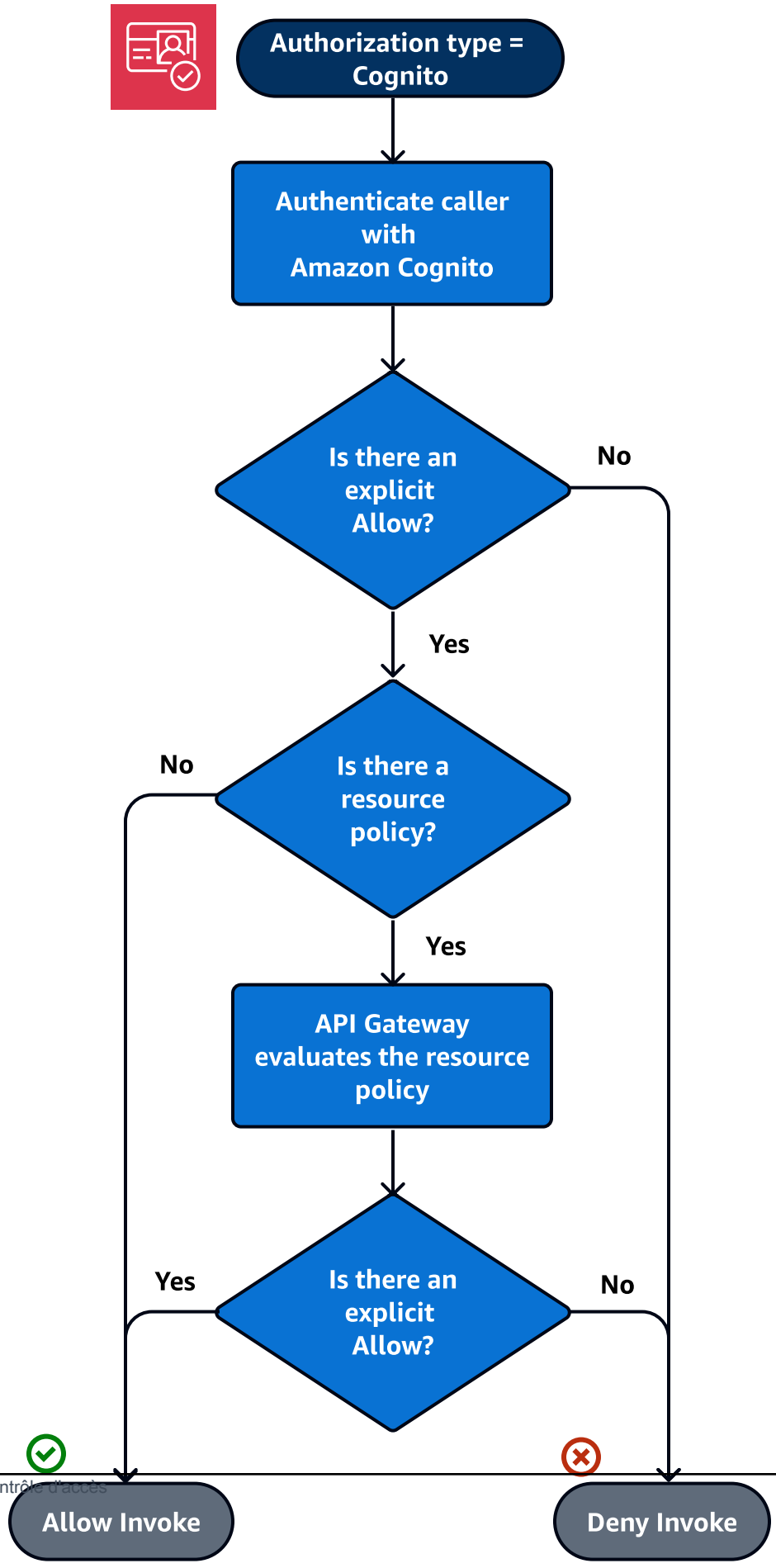
Voici un exemple de stratégie de ressources entre comptes. À supposer que la politique IAM contient un effet Allow (Autoriser), cette stratégie de ressources autorise uniquement les appels du VPC dont l'ID est *vpc-2f09a348*. Pour plus d'informations, consultez le tableau B.

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

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

Stratégie de ressources et d'authentification Amazon Cognito

Dans ce flux de travail, un groupe d'utilisateurs Amazon Cognito est configuré pour l'API en plus d'une stratégie de ressources. API Gateway tente d'abord d'authentifier l'appelant via Amazon Cognito. Ceci est généralement effectué via un jeton JWT fourni par l'appelant. Si l'authentification est réussie, la stratégie de ressources est évaluée de façon indépendante, et une autorisation explicite est requise. Un refus ou « ni autoriser ni refuser » entraîne un refus. Voici un exemple de stratégie de ressources pouvant être utilisée avec des groupes d'utilisateurs Amazon Cognito.


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

Voici un exemple de stratégie de ressources autorisant les appels uniquement à partir des adresses IP source spécifiées, en supposant que le jeton d'authentification Amazon Cognito contienne une autorisation. Pour plus d'informations, consultez le tableau B.

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

Tableaux de résultat des évaluations de stratégies

Le tableau A répertorie le comportement qui en résulte lorsque l'accès à une API API Gateway est contrôlé par une politique IAM ou un autorisateur Lambda et une politique de ressources API Gateway, qui sont toutes deux identiques. Compte AWS

Tableau A : le compte A appelle une API appartenant au compte A

Politique IAM (ou autorisateur Lambda)	Stratégie de ressources API Gateway	Comportement obtenu
Autorisation	Autorisation	Autorisation
Autorisation	Ni autorisation ni refus	Autorisation
Autorisation	Refuser	Refus explicite
Ni autorisation ni refus	Autorisation	Autorisation
Ni autorisation ni refus	Ni autorisation ni refus	Refus implicite

Politique IAM (ou autorisateur Lambda)	Stratégie de ressources API Gateway	Comportement obtenu
Ni autorisation ni refus	Refuser	Refus explicite
Refuser	Autorisation	Refus explicite
Refuser	Ni autorisation ni refus	Refus explicite
Refuser	Refuser	Refus explicite

Le tableau B répertorie le comportement qui en résulte lorsque l'accès à une API API Gateway est contrôlé par une politique IAM ou un autorisateur de groupes d'utilisateurs Amazon Cognito et par une politique de ressources API Gateway, qui sont différentes. Comptes AWS Si l'un des deux est silencieux (pas d'autorisation ni de refus), l'accès entre comptes est refusé. En effet, l'accès entre comptes nécessite que la politique de ressources et la politique IAM ou l'autorisateur de groupes d'utilisateurs Amazon Cognito accordent explicitement l'accès.

Tableau B : le compte B appelle une API appartenant au compte A

Politique IAM (ou autorisateur de groupes d'utilisateurs Amazon Cognito)	Stratégie de ressources API Gateway	Comportement obtenu
Autorisation	Autorisation	Autorisation
Autorisation	Ni autorisation ni refus	Refus implicite
Autorisation	Refuser	Refus explicite
Ni autorisation ni refus	Autorisation	Refus implicite
Ni autorisation ni refus	Ni autorisation ni refus	Refus implicite
Ni autorisation ni refus	Refuser	Refus explicite
Refuser	Autorisation	Refus explicite
Refuser	Ni autorisation ni refus	Refus explicite

Politique IAM (ou autorisateur de groupes d'utilisateurs Amazon Cognito)	Stratégie de ressources API Gateway	Comportement obtenu
Refuser	Refuser	Refus explicite

Exemples de stratégies de ressources API Gateway

Cette page présente quelques exemples de cas d'utilisation standard des stratégies de ressources API Gateway.

Les exemples de stratégie suivants utilisent une syntaxe simplifiée pour spécifier la ressource d'API. Cette syntaxe simplifiée est une façon abrégée qui vous permet de faire référence à une ressource d'API plutôt que de spécifier l'Amazon Resource Name (ARN) complet. API Gateway convertit la syntaxe abrégée en ARN complet lorsque vous enregistrez la stratégie. Par exemple, vous pouvez spécifier la ressource `execute-api:/stage-name/GET/pets` dans une stratégie de ressources. API Gateway convertit la ressource `arn:aws:execute-api:us-east-2:123456789012:aabbccddee/stage-name/GET/pets` lorsque vous enregistrez la stratégie de ressources. API Gateway crée l'ARN complet en utilisant la région actuelle, votre ID de AWS compte et l'ID de l'API REST à laquelle la politique de ressources est associée. Vous pouvez utiliser `execute-api:/*` pour représenter toutes les étapes, méthodes et chemins dans l'API actuelle. Pour en savoir plus sur le langage d'access policy, consultez Présentation du langage d'access policy pour Amazon API Gateway.

Rubriques

Exemple : autoriser les rôles d'un autre AWS compte à utiliser une API

L'exemple de politique de ressources suivant accorde l'accès à l'API dans un AWS compte à deux rôles dans un AWS compte différent via les protocoles Signature Version 4 (SigV4). Plus

précisément, le rôle de développeur et d'administrateur du AWS compte identifié par *account-id-2* est autorisé à exécuter l'`GET`action sur la `pets` ressource (API) de votre AWS compte. `execute-api:Invoke`

```
{
   "Version": "2012-10-17",
   "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": [
                    "arn:aws:iam::account-id-2:role/developer",
                    "arn:aws:iam::account-id-2:role/Admin"
                ]
            },
            "Action": "execute-api:Invoke",
            "Resource": [
                "execute-api:/stage/GET/pets"
            ]
        }
    ]
}
```

Exemple : Refuser le trafic API en fonction de l'adresse IP source ou de la plage

L'exemple de stratégie de ressources suivant refuse (bloque) le trafic entrant vers une API à partir de deux blocs d'adresses IP source spécifiés.

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

Exemple : Refuser le trafic API en fonction de l'adresse IP source ou de la plage lors de l'utilisation d'une API privée

L'exemple de stratégie de ressources suivant refuse (bloque) le trafic entrant vers une API privée à partir de deux blocs d'adresses IP source spécifiés. Lors de l'utilisation d'API privées, le point de terminaison VPC pour `execute-api` réécrit l'adresse IP source d'origine. La condition `aws:VpcSourceIp` filtre la demande par rapport à l'adresse IP du demandeur d'origine.

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

```
        }
    ]
}
```

Exemple : Autoriser le trafic de l'API privée en fonction du point de terminaison d'un VPC ou d'un VPC source

L'exemple de stratégies de ressources suivant autorise le trafic entrant vers une API privée uniquement à partir d'un VPC (Virtual Private Cloud) spécifié ou d'un point de terminaison d'un VPC.

Cet exemple de stratégie de ressources spécifie un VPC source :

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

Cet exemple de stratégie de ressources spécifie un point de terminaison d'un VPC source :

```
{
```

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

Création et attachement d'une stratégie de ressources API Gateway à une API

Pour autoriser un utilisateur à accéder à votre API en appelant le service d'exécution de l'API, vous devez créer une politique de ressources API Gateway et l'associer à l'API. Lorsque vous attachez une politique à votre API, elle applique les autorisations de la politique aux méthodes de l'API. Si vous mettez à jour la politique en matière de ressources, vous devrez déployer l'API.

Rubriques

- Prérequis
- Associer une politique de ressources à une API API Gateway
- Résoudre les problèmes liés à votre politique de ressources

Prérequis

Pour mettre à jour une politique de ressources d'API Gateway, vous aurez besoin de `apigateway:UpdateRestApiPolicy` cette autorisation et de `apigateway:PATCH` cette autorisation.

Pour une API régionale ou optimisée pour les périphériques, vous pouvez associer votre politique de ressources à votre API lors de sa création ou après son déploiement. Pour une API privée, vous ne pouvez pas déployer votre API sans politique de ressources. Pour plus d'informations, consultez the section called “API REST privées”.

Associer une politique de ressources à une API API Gateway

La procédure suivante explique comment associer une politique de ressources à une API API Gateway.

AWS Management Console

Pour attacher une stratégie de ressources à une API API Gateway

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Choisissez une API REST.
3. Dans le panneau de navigation principal, choisissez Stratégie de ressources.
4. Choisissez Créer une politique.
5. (Facultatif) Choisissez Sélectionnez un modèle pour générer un exemple de stratégie.

   Dans les exemples de stratégies, les espaces réservés sont délimitées par des doubles accolades (`"{{placeholder}}"`). Remplacez chacun des espaces réservés (y compris les accolades) par les informations nécessaires.
6. Si vous n'utilisez pas l'un des exemples de modèle, entrez votre stratégie de ressources.
7. Sélectionnez Enregistrer les modifications.

Si l'API a déjà été déployée dans la console API Gateway, vous devez la redéployer pour que la stratégie de ressources prennent effet.

AWS CLI

AWS CLI Pour créer une nouvelle API et y associer une politique de ressources, appelez la create-rest-apicommande comme suit :

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

AWS CLI Pour associer une politique de ressources à une API existante, appelez la update-rest-apicommande comme suit :

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

AWS CloudFormation

Vous pouvez l'utiliser AWS CloudFormation pour créer une API avec une politique de ressources. L'exemple suivant crée une API REST avec l'exemple de politique de ressources,the section called “Exemple : Refuser le trafic API en fonction de l'adresse IP source ou de la plage”.

```
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
```

Résoudre les problèmes liés à votre politique de ressources

Les conseils de dépannage suivants peuvent vous aider à résoudre les problèmes liés à votre politique en matière de ressources.

Mon API renvoie {"Message » « L'utilisateur : anonymous n'est pas autorisé à exécuter : Execute-API:Invoke sur la ressource : arn:aws:execute-api:us-east-1 : ********/****/****/"}

Dans votre politique de ressources, si vous attribuez au AWS principal un principal, par exemple :

```
{
    "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:/*"
        ]
    },
    ...
    }
```

Vous devez utiliser `AWS_IAM` l'autorisation pour chaque méthode de votre API, sinon votre API renvoie le message d'erreur précédent. Pour plus d'instructions sur la façon d'activer `AWS_IAM` l'autorisation pour une méthode, consultezthe section called “Méthodes”.

Ma politique en matière de ressources n'est pas mise à jour

Si vous mettez à jour la stratégie de ressources une fois l'API créée, vous devez déployer l'API pour propager les changements une fois le rattachement de la stratégie mise à jour effectué. La simple mise à jour ou le simple enregistrement de la stratégie ne modifie pas le comportement d'exécution de l'API. Pour plus d'informations sur le déploiement de votre API, consultez the section called “Déploiement d'une API REST”.

AWS clés de condition pouvant être utilisées dans les politiques de ressources d'API Gateway

Le tableau suivant contient les clés de AWS condition qui peuvent être utilisées dans les politiques de ressources pour les API d'API d'API Gateway pour chaque type d'autorisation.

Pour plus d'informations sur les clés de AWS condition, consultez la section Clés contextuelles de condition AWS globales.

Tableau des clés de condition

Clés de condition	Critères	Nécessite **`AuthN`**?	Type d'autorisation
`aws:CurrentTime`	Aucun	Non	Tous
`aws:EpochTime`	Aucun	Non	Tous
`aws:Token IssueTime`	Cette clé est uniquement présente	Oui	IAM

Clés de condition	Critères	Nécessite **AuthN**?	Type d'autorisation
	dans les demandes signées à l'aide d'informations d'identification de sécurité temporaires.		
`aws:MultiFactorAuthPresent`	Cette clé est uniquement présente dans les demandes signées à l'aide d'informations d'identification de sécurité temporaires.	Oui	IAM
`aws:MultiFactorAuthAge`	La clé est présente uniquement si l'élément MFA est présent dans la demande.	Oui	IAM
`aws:PrincipalAccount`	Aucun	Oui	IAM
`aws:PrincipalArn`	Aucun	Oui	IAM
`aws:PrincipalOrgID`	Cette clé ne figure dans le contexte de la demande que si le mandataire est membre d'une organisation.	Oui	IAM

Clés de condition	Critères	Nécessite **AuthN**?	Type d'autorisation
`aws:PrincipalOrgPaths`	Cette clé ne figure dans le contexte de la demande que si le mandataire est membre d'une organisation.	Oui	IAM
`aws:PrincipalTag`	Cette clé figure dans le contexte de la demande si le mandataire utilise un utilisateur IAM avec des balises attachées . Elle est incluse pour un mandataire utilisant un rôle IAM avec des balises attachées ou des balises de session.	Oui	IAM
`aws:PrincipalType`	Aucun	Oui	IAM
`aws:Referer`	La clé est présente uniquement si la valeur est fournie par le mandataire dans l'en-tête HTTP.	Non	Tous
`aws:SecureTransport`	Aucun	Non	Tous
`aws:SourceArn`	Aucun	Non	Tous
`aws:SourceIp`	Aucun	Non	Tous

Clés de condition	Critères	Nécessite **AuthN**?	Type d'autorisation
`aws:SourceVpc`	Cette clé ne peut être utilisée que pour les API privées.	Non	Tous
`aws:SourceVpce`	Cette clé ne peut être utilisée que pour les API privées.	Non	Tous
`aws:VpcSourceIp`	Cette clé ne peut être utilisée que pour les API privées.	Non	Tous
`aws:UserAgent`	La clé est présente uniquement si la valeur est fournie par le mandataire dans l'en-tête HTTP.	Non	Tous
`aws:userid`	Aucun	Oui	IAM
`aws:username`	Aucun	Oui	IAM

Contrôle de l'accès à une API avec des autorisations IAM

Vous contrôlez l'accès à votre API Amazon API Gateway avec les autorisations IAM en contrôlant l'accès aux deux processus de composant API Gateway suivants :

- Pour créer, déployer et gérer une API dans API Gateway, vous devez accorder les autorisations aux développeurs d'API pour qu'ils exécutent les actions requises prises en charge par le composant de gestion des API API Gateway.
- Pour appeler une API déployée ou pour actualiser la mise en cache de l'API, vous devez accorder les autorisations de l'appelant d'API pour exécuter les actions IAM obligatoires prises en charge par le composant de l'exécution d'API API Gateway.

Le contrôle d'accès pour les deux processus implique différents modèles d'autorisation, expliqués plus bas.

Modèle d'autorisation API Gateway pour créer et gérer une API

Pour permettre à un développeur d'API de créer et gérer une API dans API Gateway, vous devez créer des stratégies d'autorisation IAM qui permettent à un développeur d'API spécifié de créer, mettre à jour, déployer, afficher ou supprimer les entités d'API requises. Vous attachez la politique d'autorisations à un utilisateur, un rôle ou un groupe.

Pour activer l’accès, ajoutez des autorisations à vos utilisateurs, groupes ou rôles :

- Utilisateurs et groupes dans AWS IAM Identity Center :

  Créez un jeu d’autorisations. Suivez les instructions de la rubrique Création d’un jeu d’autorisations du Guide de l’utilisateur AWS IAM Identity Center .
- Utilisateurs gérés dans IAM par un fournisseur d’identité :

  Créez un rôle pour la fédération d’identité. Pour plus d’informations, voir la rubrique Création d’un rôle pour un fournisseur d’identité tiers (fédération) du Guide de l’utilisateur IAM.
- Utilisateurs IAM :
  - Créez un rôle que votre utilisateur peut assumer. Suivez les instructions de la rubrique Création d’un rôle pour un utilisateur IAM du Guide de l’utilisateur IAM.
  - (Non recommandé) Attachez une politique directement à un utilisateur ou ajoutez un utilisateur à un groupe d’utilisateurs. Suivez les instructions de la rubrique Ajout d’autorisations à un utilisateur (console) du Guide de l’utilisateur IAM.

Pour plus d'informations sur l'utilisation de ce modèle d'autorisation, consultez the section called “Stratégies basées sur l'identité d'API Gateway”.

Modèle d'autorisation API Gateway pour l'appel d'une API

Pour permettre à un appelant d'API d'appeler l'API ou d'actualiser sa mise en cache, vous devez créer des politiques IAM qui permettent à un appelant d'API spécifié d'appeler la méthode d'API pour laquelle l'authentification d'utilisateur est activée. Le développeur d'API définit la propriété `authorizationType` de la méthode sur `AWS_IAM` pour imposer à l'appelant de soumettre les informations d'identification de l'utilisateur à l'authentification. Vous pouvez ensuite attacher la politique à un utilisateur, un groupe ou un rôle.

Dans cette déclaration de stratégie d'autorisation IAM, l'élément IAM `Resource` contient la liste des méthodes d'API déployées et identifiées par les verbes HTTP fournis et les chemins d'accès des ressources API Gateway. L'élément IAM `Action` contient les actions d'exécution requises des API API Gateway. Ces actions incluent `execute-api:Invoke` ou `execute-api:InvalidateCache`, où `execute-api` désigne le composant d'exécution des API sous-jacent d'API Gateway.

Pour plus d'informations sur l'utilisation de ce modèle d'autorisation, consultez Contrôler l'accès pour l'appel d'une API.

Lorsqu'une API est intégrée à un AWS service (par exemple, AWS Lambda) dans le back-end, API Gateway doit également être autorisée à accéder aux AWS ressources intégrées (par exemple, en invoquant une fonction Lambda) au nom de l'appelant de l'API. Pour accorder ces autorisations, créez un rôle IAM du type service AWS pour API Gateway. Lorsque vous créez ce rôle dans la console de gestion IAM, le rôle résultant contient la stratégie d'approbation IAM suivante qui déclare API Gateway en tant qu'entité de confiance autorisée à assumer le rôle :

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

Si vous créez le rôle IAM en appelant la commande create-role de la CLI ou une méthode de kit SDK correspondante, vous devez fournir la politique d'approbation ci-dessus en tant que paramètre d'entrée de `assume-role-policy-document`. N'essayez pas de créer une telle politique directement dans la console de gestion IAM ou d'appeler la commande AWS CLI create-policy ou une méthode du SDK correspondante.

Pour qu'API Gateway puisse appeler le AWS service intégré, vous devez également associer à ce rôle des politiques d'autorisation IAM appropriées pour appeler les AWS services intégrés. Par exemple, pour appeler une fonction Lambda, vous devez inclure la stratégie d'autorisations IAM suivante dans le rôle IAM :

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

Notez que Lambda prend en charge la stratégie d'accès basée sur les ressources, qui combine les stratégies d'approbation et d'autorisation. Lors de l'intégration d'une API à une fonction Lambda à l'aide de la console API Gateway, il ne vous est pas demandé de définir ce rôle IAM explicitement, car la console définit pour vous les autorisations basées sur les ressources sur la fonction Lambda, avec votre consentement.

> **Note**
>
> Pour mettre en place un contrôle d'accès à un AWS service, vous pouvez utiliser le modèle d'autorisation basé sur l'appelant, dans lequel une politique d'autorisation est directement attachée à l'utilisateur ou au groupe de l'appelant, ou le modèle d'autorisation basé sur les rôles, dans lequel une politique d'autorisation est attachée à un rôle IAM que API Gateway peut assumer. Les stratégies d'autorisation peuvent varier dans les deux modèles. Par exemple, la stratégie basée sur l'appelant bloque l'accès tandis que la stratégie basée sur les rôles l'autorise. Vous pouvez en profiter pour exiger qu'un utilisateur accède à un AWS service via une API API Gateway uniquement.

Contrôler l'accès pour l'appel d'une API

Dans cette section, vous allez apprendre à écrire les déclarations de stratégie IAM pour contrôler qui peut ou non appeler une API déployée dans API Gateway. Ici, vous trouverez également la référence de déclaration de stratégie, y compris les formats des champs `Action` et `Resource` liés au service de l'exécution des API. Vous devez également étudier la section IAM de the section called “Comment les stratégies de ressources affectent le flux de travail d'autorisation”.

Pour les API privées, vous devez utiliser une combinaison constituée d'une stratégie de ressources API Gateway et d'un point de terminaison de VPC. Pour plus d'informations, consultez les rubriques suivantes :

- the section called “Utilisation des stratégies de ressources API Gateway”
- the section called “Utilisation de stratégies de point de terminaison de VPC pour des API privées”

Contrôle des personnes habilitées à appeler une méthode d'API API Gateway avec les stratégies IAM

Pour contrôler qui peut ou non appeler une API déployée avec les autorisations IAM, créez un document de stratégie IAM avec les autorisations requises. Un modèle pour un tel document de stratégie est affiché comme suit.

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

Ici, *Permission* doit être remplacé par `Allow` ou `Deny`, selon que vous voulez accorder ou révoquer les autorisations incluses. *Execution-operation* doit être remplacé par les opérations prises en charge par le service d'exécution des API. *METHOD_HTTP_VERB* représente un verbe HTTP pris en charge par les ressources spécifiées. *Resource-path* est l'espace réservé du chemin d'URL d'une instance `Resource` d'API déployée prenant en charge ledit verbe *METHOD_HTTP_VERB*. Pour plus d’informations, consultez Référence de déclaration de stratégie IAM pour l'exécution des API dans API Gateway.

> **Note**
>
> Pour que les stratégies IAM soient efficaces, vous devez avoir activé l'authentification IAM sur les méthodes de l'API en définissant `AWS_IAM` pour la propriété `authorizationType` de la méthode. Faute de quoi ces méthodes d'API seront accessibles au public.

Par exemple, pour accorder à un utilisateur l'autorisation d'afficher la liste des « pets » exposés par une API spécifiée, mais lui refuser l'autorisation d'ajouter un « pet » à la liste, vous pouvez inclure la déclaration suivante dans la stratégie IAM :

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

Pour accorder à un utilisateur l'autorisation d'afficher un « pet » exposé spécifique par une API configurée en tant que `GET /pets/{petId}`, vous pouvez inclure la déclaration suivante dans la stratégie IAM :

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

Référence de déclaration de stratégie IAM pour l'exécution des API dans API Gateway

Les informations suivantes décrivent le format des expressions Action et Resource des déclarations de stratégie IAM d'autorisation d'accès pour l'exécution d'une API.

Format de l'expression Action des autorisations d'exécution d'API dans API Gateway

L'expression `Action` des autorisations d'exécution d'API a le format général suivant :

```
execute-api:action
```

où *`action`* est une action d'exécution d'API disponible :

- *, qui représente l'ensemble des actions suivantes.
- Invoke, utilisée pour appeler une API sur demande d'un client.
- InvalidateCache, utilisé pour invalider le cache de l'API à la demande d'un client.

Format de l'expression Resource des autorisations d'exécution d'API dans API Gateway

L'expression `Resource` des autorisations d'exécution d'API a le format général suivant :

```
arn:aws:execute-api:region:account-id:api-id/stage-name/HTTP-VERB/resource-path-
specifier
```

où :

- *`région`* est la AWS région (telle que **`us-east-1`** ou **`*`** pour toutes les AWS régions) qui correspond à l'API déployée pour la méthode.

- *account-id* est l'identifiant de AWS compte à 12 chiffres du propriétaire de l'API REST.
- *api-id* est l'identifiant qu'API Gateway a attribué à l'API pour la méthode.
- *stage-name* est le nom de l'étape associée à la méthode.
- *HTTP-VERB* est le verbe HTTP pour la méthode. Les valeurs possibles sont GET, POST, PUT, DELETE, PATCH.
- *resource-path-specifier*est le chemin d'accès à la méthode souhaitée.

> **Note**
>
> Si vous spécifiez un caractère générique (`*`), l'expression `Resource` applique le caractère générique au reste de l'expression.

Voici quelques exemples d'expression Resource :

- **arn:aws:execute-api:*:*:***pour n'importe quel chemin de ressource à n'importe quelle étape, pour n'importe quelle API dans n'importe quelle AWS région.
- **arn:aws:execute-api:us-east-1:*:***pour tout chemin de ressource à n'importe quelle étape, pour toute API de la AWS région de`us-east-1`.
- **arn:aws:execute-api:us-east-1:*:*api-id*/***pour tout chemin de ressource à n'importe quelle étape, pour l'API dont l'identifiant *api-id* se situe dans la AWS région us-east-1.
- **arn:aws:execute-api:us-east-1:*:*api-id*/test/*** pour le chemin de ressource à l'étape `test`, pour l'API avec l'identifiant *api-id* dans la région AWS us-east-1.

Pour en savoir plus, veuillez consulter la section Référence Amazon Resource Name (ARN) API Gateway.

Exemples de stratégies IAM pour les autorisations d'exécution d'API

Pour en savoir plus sur les modèles d'autorisation et pour d'autres informations générales, consultez Contrôler l'accès pour l'appel d'une API.

La déclaration de stratégie suivante donne à l'utilisateur l'autorisation d'appeler n'importe quelle méthode POST sur le chemin `mydemoresource`, à l'étape `test`, pour l'API avec l'identifiant `a123456789`, en supposant que l'API correspondante a été déployée dans la région AWS us-east-1 :

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "execute-api:Invoke"
      ],
      "Resource": [
        "arn:aws:execute-api:us-east-1:*:a123456789/test/POST/mydemoresource/*"
      ]
    }
  ]
}
```

L'exemple de déclaration de stratégie suivant accorde à l'utilisateur l'autorisation d'appeler n'importe quelle méthode sur le chemin de ressource `petstorewalkthrough/pets`, à n'importe quelle étape, pour l'API ayant l'identifiant `a123456789`, dans n'importe quelle région AWS où l'API correspondante a été déployée :

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

Création et attachement d'une stratégie à un utilisateur

Pour permettre à un utilisateur d'appeler le service de gestion des API ou le service d'exécution des API, vous devez créer une politique IAM qui contrôle l'accès aux entités API Gateway.

Pour utiliser l'éditeur de politique JSON afin de créer une politique

1. Connectez-vous à la console IAM AWS Management Console et ouvrez-la à l'adresse https://console.aws.amazon.com/iam/.
2. Dans le panneau de navigation de gauche, sélectionnez Policies (Politiques).

   Si vous sélectionnez Politiques pour la première fois, la page Bienvenue dans les politiques gérées s'affiche. Sélectionnez Mise en route.
3. En haut de la page, sélectionnez Créer une politique.
4. Dans la section Éditeur de politiques, choisissez l'option JSON.
5. Entrez le document de politique JSON suivant :

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

6. Choisissez Suivant.

> **Note**
>
> Vous pouvez basculer à tout moment entre les options des éditeurs visuel et JSON. Toutefois, si vous apportez des modifications ou si vous choisissez Suivant dans l'éditeur visuel, IAM peut restructurer votre politique afin de l'optimiser pour l'éditeur visuel. Pour de plus amples informations, consultez la page Restructuration de politique dans le Guide de l'utilisateur IAM.

7. Sur la page Vérifier et créer, saisissez un Nom de politique et une Description (facultative) pour la politique que vous créez. Vérifiez les Autorisations définies dans cette politique pour voir les autorisations accordées par votre politique.
8. Choisissez Create policy (Créer une politique) pour enregistrer votre nouvelle politique.

Dans cette déclaration, remplacez *`action-statement`* et *`resource-statement`* si nécessaire, et ajoutez d'autres déclarations pour spécifier les entités API Gateway que vous voulez autoriser l'utilisateur à gérer, les méthodes d'API que l'utilisateur peut appeler ou les deux. Par défaut, l'utilisateur ne dispose pas d'autorisations, à moins qu'il existe une déclaration `Allow` correspondante explicite.

Vous venez de créer une stratégie IAM. Elle n'aura aucun effet tant que vous ne l'aurez pas attachée.

Pour activer l'accès, ajoutez des autorisations à vos utilisateurs, groupes ou rôles :

- Utilisateurs et groupes dans AWS IAM Identity Center :

  Créez un jeu d'autorisations. Suivez les instructions de la rubrique Création d'un jeu d'autorisations du Guide de l'utilisateur AWS IAM Identity Center .
- Utilisateurs gérés dans IAM par un fournisseur d'identité :

  Créez un rôle pour la fédération d'identité. Pour plus d'informations, voir la rubrique Création d'un rôle pour un fournisseur d'identité tiers (fédération) du Guide de l'utilisateur IAM.
- Utilisateurs IAM :
  - Créez un rôle que votre utilisateur peut assumer. Suivez les instructions de la rubrique Création d'un rôle pour un utilisateur IAM du Guide de l'utilisateur IAM.
  - (Non recommandé) Attachez une politique directement à un utilisateur ou ajoutez un utilisateur à un groupe d'utilisateurs. Suivez les instructions de la rubrique Ajout d'autorisations à un utilisateur (console) du Guide de l'utilisateur IAM.

Pour attacher un document de stratégie IAM à un groupe IAM

1. Dans le volet de navigation principal, choisissez Groupes.
2. Choisissez l'onglet Autorisations sous le groupe choisi.
3. Choisissez Attach policy (Attacher une politique).
4. Choisissez le document de stratégie que vous avez précédemment créé, puis sélectionnez Attacher la stratégie.

Pour qu'API Gateway puisse appeler d'autres AWS services en votre nom, créez un rôle IAM du type Amazon API Gateway.

Pour créer un type de rôle Amazon API Gateway

1. Dans le volet de navigation principal, choisissez Rôles.
2. Choisissez Create New Role.
3. Tapez un nom pour Nom du rôle, puis choisissez Étape suivante.
4. Sous Sélectionner un type de rôle, dans Rôles de service AWS , choisissez Sélectionner à côté d'Amazon API Gateway.
5. Choisissez une politique d'autorisations IAM gérée disponible, par exemple AmazonAPI GatewayPushToCloudWatchLog si vous souhaitez qu'API Gateway enregistre les métriques CloudWatch, sous Attach Policy, puis choisissez Next Step.
6. Sous Entités de confiance, vérifiez qu'apigateway.amazonaws.com apparaît comme une entrée, puis choisissez Créer un rôle.
7. Dans le rôle nouvellement créé, sélectionnez l'onglet Autorisations, puis choisissez Attacher la stratégie.
8. Sélectionnez le document de stratégie IAM personnalisé précédemment créé, puis choisissez Attach Policy (Attacher la stratégie).

Utilisation de stratégies de point de terminaison de VPC pour des API privées dans API Gateway

Pour améliorer la sécurité de votre API privée, vous pouvez créer une politique de point de terminaison VPC. La politique de point de terminaison de VPC est une politique de ressources IAM qui est jointe à un point de terminaison d'un VPC. Pour plus d'informations, consultez Contrôle de l'accès aux services avec des points de terminaison de VPC.

Vous souhaiterez peut-être créer une politique de point de terminaison VPC pour effectuer les opérations suivantes :

- Autorisez uniquement certaines organisations ou ressources à accéder à votre point de terminaison VPC et à appeler votre API.
- Utilisez une politique unique et évitez les politiques basées sur les sessions ou les rôles pour contrôler le trafic vers votre API.
- Resserrez le périmètre de sécurité de votre application lors de la migration d'une application sur site vers AWS.

Considérations d'une politique de point de terminaison d'un VPC

- L'identité de l'invocateur est évaluée en fonction de la valeur de l'en-tête `Authorization`. Selon votre `authorizationType`, cela peut entraîner une erreur `403 IncompleteSignatureException` ou une erreur `403 InvalidSignatureException`. Le tableau suivant affiche les valeurs d'en-tête `Authorization` pour chaque `authorizationType`.

`authorizationType`	En-tête **`Authorization`** évalué ?	Valeurs d'en-tête **`Authorization`** autorisées
`NONE` avec la stratégie d'accès complet par défaut	Non	Non validé
`NONE` avec une stratégie d'accès personnalisée	Oui	Doit être une valeur SigV4 valide
`IAM`	Oui	Doit être une valeur SigV4 valide
`CUSTOM` ou `COGNITO_USER_POOLS`	Non	Non validé

- Si une politique restreint l'accès à un principal IAM spécifique, par exemple`arn:aws:iam::account-id:role/developer`, vous devez définir la méthode `authorizationType` de votre API sur ou. `AWS_IAM` `NONE` Pour plus d'instructions sur la façon de définir le `authorizationType` pour une méthode, consultezthe section called “Méthodes”.

- Les stratégies de point de terminaison de VPC peuvent être utilisées conjointement avec des stratégies de ressources API Gateway. La politique de ressources API Gateway indique quels principaux peuvent accéder à l'API. La politique de point de terminaison précise qui peut accéder au VPC et quelles API peuvent être appelées depuis le point de terminaison du VPC. Votre API privée a besoin d'une politique de ressources, mais vous n'avez pas besoin de créer une politique de point de terminaison VPC personnalisée.

Exemples de stratégie de point de terminaison de VPC

Vous pouvez créer des stratégies pour les points de terminaison Amazon Virtual Private Cloud pour Amazon API Gateway dans lesquelles vous pouvez spécifier :

- Le mandataire qui peut exécuter des actions.
- Les actions qui peuvent être effectuées.
- Les ressources qui peuvent avoir des actions exécutées sur elles.

Pour attacher la stratégie au point de terminaison de VPC, vous devez utiliser la console VPC. Pour plus d'informations, consultez Contrôle de l'accès aux services avec des points de terminaison de VPC.

Exemple 1 : Stratégie de point de terminaison de VPC accordant l'accès à deux API

L'exemple de stratégie suivant accorde l'accès à seulement deux API spécifiques via le point de terminaison de VPC auquel la stratégie est attachée.

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

```
}
```

Exemple 2 : Stratégie de point de terminaison de VPC accordant l'accès aux méthodes GET

L'exemple de stratégie suivant accorde aux utilisateurs l'accès aux méthodes `GET` pour une API spécifique via le point de terminaison de VPC auquel la stratégie est attachée.

```
{
    "Statement": [
        {
            "Principal": "*",
            "Action": [
                "execute-api:Invoke"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:execute-api:us-east-1:123412341234:a1b2c3d4e5/stageName/GET/*"
            ]
        }
    ]
}
```

Exemple 3 : Stratégie de point de terminaison de VPC accordant à un utilisateur spécifique l'accès à une API spécifique

L'exemple de stratégie suivant accorde à un utilisateur spécifique l'accès à une API spécifique via le point de terminaison de VPC auquel la stratégie est attachée.

Dans ce cas, étant donné que la politique restreint l'accès à des principes IAM spécifiques, vous devez définir `authorizationType` la méthode sur ou. `AWS_IAM` `NONE`

```
{
    "Statement": [
        {
            "Principal": {
                "AWS": [
                    "arn:aws:iam::123412341234:user/MyUser"
                ]
            },
            "Action": [
                "execute-api:Invoke"
```

```
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:execute-api:us-east-1:123412341234:a1b2c3d4e5/*"
            ]
        }
    ]
}
```

Utilisation de balises pour contrôler l'accès à une API REST dans API Gateway

L'autorisation d'accès aux API REST peut être optimisée en utilisant le contrôle d'accès basé sur les attributs dans des politiques IAM.

Pour plus d'informations, voir the section called "Contrôle d'accès basé sur les attributs".

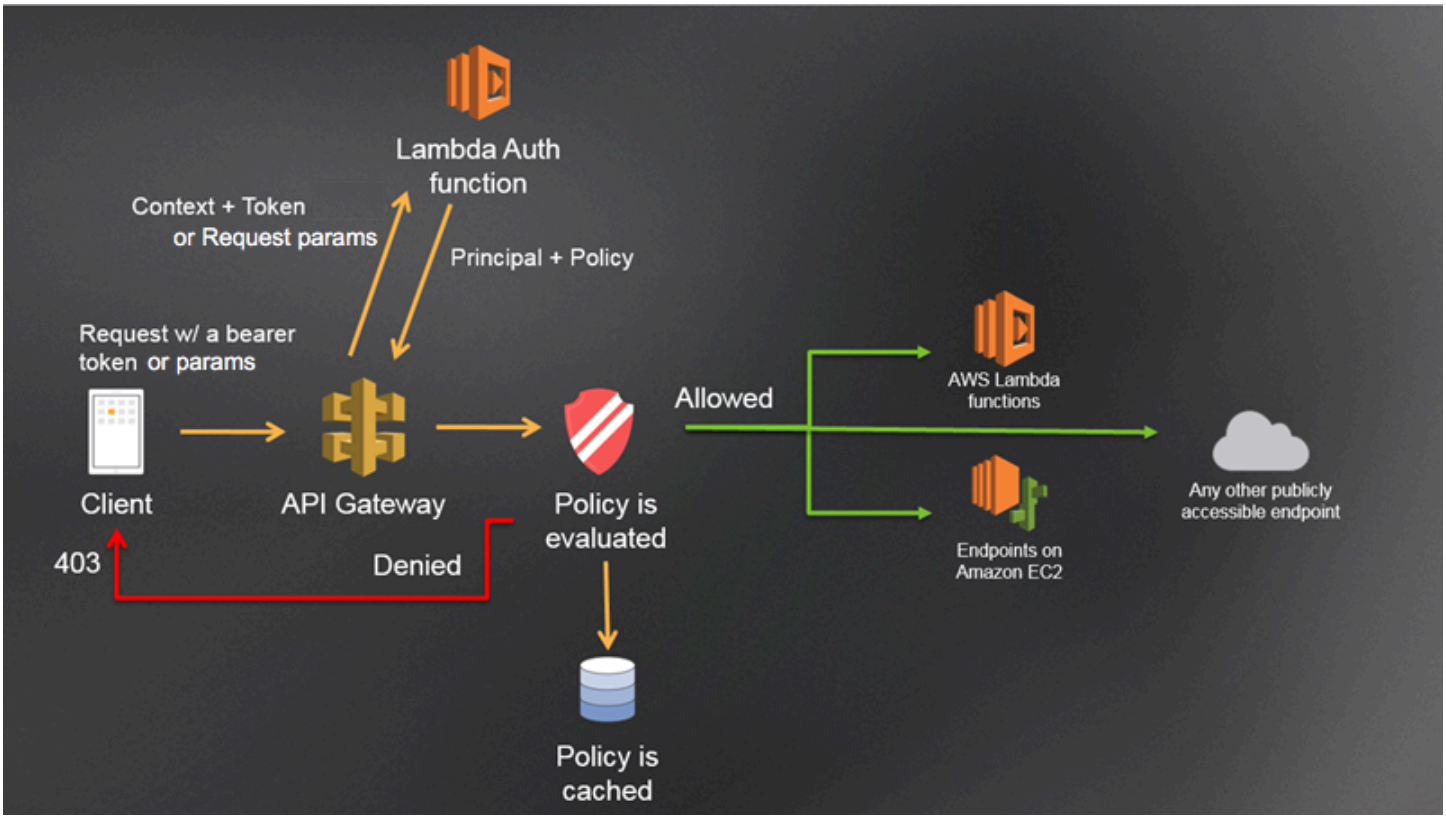
Utilisation des mécanismes d'autorisation Lambda API Gateway

Utilisez un autorisateur Lambda (anciennement connu sous le nom d'autorisateur personnalisé) pour contrôler l'accès à votre API. Lorsqu'un client demande la méthode de votre API, API Gateway appelle votre autorisateur Lambda. L'autorisateur Lambda prend l'identité de l'appelant en entrée et renvoie une politique IAM en sortie.

Utilisez un autorisateur Lambda pour implémenter un schéma d'autorisation personnalisé. Votre schéma peut utiliser des paramètres de demande pour déterminer l'identité de l'appelant ou utiliser une stratégie d'authentification par jeton porteur telle que OAuth ou SAML. Créez un autorisateur Lambda dans la console API REST API Gateway, à l'aide du ou d'un AWS CLI SDK. AWS

Flux de travail d'autorisation Lambda Authorizer

Le schéma suivant montre le flux de travail d'autorisation pour un autorisateur Lambda.



Flux de travail d'autorisation Lambda API Gateway

1. Le client appelle une méthode sur une API API Gateway, en transmettant un jeton porteur ou des paramètres de demande.
2. API Gateway vérifie si la demande de méthode est configurée avec un autorisateur Lambda. Si c'est le cas, API Gateway appelle la fonction Lambda.
3. La fonction Lambda authentifie l'appelant. La fonction peut s'authentifier de différentes manières :
   - En appelant un fournisseur OAuth pour obtenir un jeton d'accès OAuth.
   - En appelant un fournisseur SAML pour obtenir une assertion SAML.
   - En générant une politique IAM basée sur les valeurs des paramètres de demande.
   - En récupérant les informations d'identification d'une base de données.
4. La fonction Lambda renvoie une politique IAM et un identifiant principal. Si la fonction Lambda ne renvoie pas ces informations, l'appel échoue.
5. API Gateway évalue la politique IAM.

- Si l'accès est refusé, API Gateway renvoie un code de statut HTTP adapté, par exemple `403 ACCESS_DENIED`.
- Si l'accès est autorisé, API Gateway appelle la méthode.

  Si vous activez la mise en cache des autorisations, API Gateway met en cache la politique afin que la fonction d'autorisation Lambda ne soit pas invoquée à nouveau.

Vous pouvez personnaliser les réponses `403 ACCESS_DENIED` ou les réponses de la `401 UNAUTHORIZED` passerelle. Pour en savoir plus, veuillez consulter la section the section called “Réponses de passerelle”.

Choisir un type d'autorisateur Lambda

Il existe deux types d'autorisations Lambda :

Autorisateur Lambda basé sur les paramètres de demande (autorisateur) **`REQUEST`**

Un `REQUEST` autorisateur reçoit l'identité de l'appelant sous forme d'une combinaison d'en-têtes, de paramètres de chaîne de requête et de variables `stageVariables`. `$context` Vous pouvez utiliser un `REQUEST` autorisateur pour créer des politiques précises basées sur les informations provenant de plusieurs sources d'identité, telles que les variables de contexte `$context.path` et`$context.httpMethod`.

Si vous activez la mise en cache des autorisations pour un `REQUEST` autorisateur, API Gateway vérifie que toutes les sources d'identité spécifiées sont présentes dans la demande. Si une source d'identification spécifiée est manquante, nulle ou vide, API Gateway renvoie une réponse `401 Unauthorized` HTTP sans appeler la fonction d'autorisation Lambda. Lorsque plusieurs sources d'identité sont définies, elles sont toutes utilisées pour dériver la clé de cache de l'autorisateur, l'ordre étant préservé. Vous pouvez définir une clé de cache précise en utilisant plusieurs sources d'identité.

Si vous modifiez l'un des éléments clés du cache et que vous redéployez votre API, l'autorisateur supprime le document de politique mis en cache et en génère un nouveau.

Si vous désactivez la mise en cache des autorisations pour un `REQUEST` autorisateur, API Gateway transmet directement la demande à la fonction Lambda.

Autorisateur Lambda basé sur des jetons (autorisateur) **`TOKEN`**

Un `TOKEN` autorisateur reçoit l'identité de l'appelant dans un jeton porteur, tel qu'un jeton Web JSON (JWT) ou un jeton OAuth.

Si vous activez la mise en cache des autorisations pour un `TOKEN` autorisateur, le nom d'en-tête spécifié dans la source du jeton devient la clé du cache.

En outre, vous pouvez utiliser la validation par jeton pour saisir une RegEx déclaration. API Gateway effectue la validation initiale du jeton d'entrée par rapport à cette expression et invoque la fonction d'autorisation Lambda une fois la validation réussie. Cela contribue à réduire le nombre d'appels à votre API.

La `IdentityValidationExpression` propriété est prise en charge uniquement pour les `TOKEN` personnes autorisées. Pour plus d'informations, consultez the section called "`x-amazon-apigateway-authorizer`".

> **Note**
>
> Nous vous recommandons d'utiliser un `REQUEST` autorisateur pour contrôler l'accès à votre API. Vous pouvez contrôler l'accès à votre API en fonction de plusieurs sources d'identité lorsque vous utilisez un `REQUEST` autorisateur, par rapport à une source d'identité unique lorsque vous utilisez un `TOKEN` autorisateur. En outre, vous pouvez séparer les clés de cache à l'aide de plusieurs sources d'identité pour un `REQUEST` autorisateur.

Exemple de fonction **`REQUEST`** Lambda d'autorisation

L'exemple de code suivant crée une fonction d'autorisation Lambda qui autorise une demande si l'`HeaderAuth1`en-tête, le paramètre de `QueryString1` requête et la variable d'étape fournis par le client correspondent aux valeurs spécifiées de`StageVar1`, et`headerValue1`, `queryValue1` respectivement. `stageValue1`

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)
```

Dans cet exemple, la fonction du mécanisme d'autorisation Lambda vérifie les paramètres d'entrée et agit comme suit :

- Si toutes les valeurs de paramètre obligatoires correspondent aux valeurs attendues, la fonction du mécanisme d'autorisation renvoie une réponse HTTP `200 OK` et une stratégie IAM qui ressemble à ce qui suit, et la demande de la méthode aboutit :

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

```
      "Resource": "arn:aws:execute-api:us-east-1:123456789012:ivdtdhp7b5/
ESTestInvoke-stage/GET/"
    }
  ]
}
```

- Dans le cas contraire, la fonction d'autorisation renvoie une réponse `401 Unauthorized` HTTP et la demande de méthode échoue.

En plus de renvoyer une stratégie IAM, la fonction du mécanisme d'autorisation Lambda doit également renvoyer l'identifiant principal de l'appelant. Facultativement, il peut renvoyer un `context` objet contenant des informations supplémentaires qui peuvent être transmises au backend d'intégration. Pour plus d'informations, consultez Résultat d'un autorisateur Lambda API Gateway.

Dans le code de production, il se peut que vous deviez authentifier l'utilisateur avant d'accorder l'autorisation. Vous pouvez ajouter une logique d'authentification dans la fonction Lambda en appelant un fournisseur d'authentification comme indiqué dans la documentation de ce fournisseur.

Exemple de fonction **`TOKEN`** Lambda d'autorisation

L'exemple de code suivant crée une fonction d'autorisation `TOKEN` Lambda qui permet à un appelant d'invoquer une méthode si la valeur du jeton fournie par le client est. `allow` L'appelant n'est pas autorisé à invoquer la demande si la valeur du jeton est`deny`. Si la valeur du jeton est `unauthorized` ou une chaîne vide, la fonction d'autorisation renvoie une `401 UNAUTHORIZED` réponse.

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
```

Dans cet exemple, lorsque l'API reçoit une demande de méthode, API Gateway transmet le jeton source à cette fonction du mécanisme d'autorisation Lambda dans l'attribut `event.authorizationToken`. La fonction du mécanisme d'autorisation Lambda lit le jeton et agit comme suit :

- Si la valeur du jeton est `allow`, la fonction du mécanisme d'autorisation renvoie une réponse HTTP `200 OK` et une stratégie IAM qui ressemble à ce qui suit, et la demande de méthode aboutit :

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": "execute-api:Invoke",
      "Effect": "Allow",
      "Resource": "arn:aws:execute-api:us-east-1:123456789012:ivdtdhp7b5/
ESTestInvoke-stage/GET/"
    }
  ]
}
```

- Si la valeur du jeton est `deny`, la fonction du mécanisme d'autorisation renvoie une réponse HTTP `200 OK` et une stratégie IAM `Deny` qui ressemble à ce qui suit, et la demande de méthode échoue :

```
{
  "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**
>
> En dehors de l'environnement de test, API Gateway renvoie une réponse `403 Forbidden` HTTP et la demande de méthode échoue.

- Si la valeur du jeton est `unauthorized` ou une chaîne vide, la fonction du mécanisme d'autorisation renvoie une réponse HTTP `401 Unauthorized` et l'appel de la méthode échoue.
- Si le jeton a n'importe quelle autre valeur, le client reçoit une réponse `500 Invalid token` et l'appel de la méthode échoue.

En plus de renvoyer une stratégie IAM, la fonction du mécanisme d'autorisation Lambda doit également renvoyer l'identifiant principal de l'appelant. Facultativement, il peut renvoyer un `context` objet contenant des informations supplémentaires qui peuvent être transmises au backend d'intégration. Pour plus d'informations, consultez Résultat d'un autorisateur Lambda API Gateway.

Dans le code de production, il se peut que vous deviez authentifier l'utilisateur avant d'accorder l'autorisation. Vous pouvez ajouter une logique d'authentification dans la fonction Lambda en appelant un fournisseur d'authentification comme indiqué dans la documentation de ce fournisseur.

Exemples supplémentaires de fonctions d'autorisation Lambda

La liste suivante présente d'autres exemples de fonctions d'autorisation Lambda. Vous pouvez créer une fonction Lambda dans le même compte, ou dans un compte différent, à partir duquel vous avez créé votre API.

Dans l'exemple précédent, vous pouvez utiliser les fonctions Lambda intégrées AWSLambdaBasicExecutionRole, car ces fonctions n'appellent aucun autre AWS service. Si votre fonction Lambda appelle d'autres AWS services, vous devez attribuer un rôle d'exécution IAM à la fonction Lambda. Pour créer le rôle, suivez les instructions dans Rôle d'exécution AWS Lambda.

Exemples supplémentaires de fonctions d'autorisation Lambda

- Pour un exemple d'application, voir Open Banking Brazil - Authorization Samples on GitHub.
- Pour plus d'exemples de fonctions Lambda, voir aws-apigateway-lambda-authorizer-blueprints on. GitHub
- Vous pouvez créer un système d'autorisation Lambda qui authentifie les utilisateurs à l'aide des groupes d'utilisateurs Amazon Cognito et autorise les appelants sur la base d'un magasin de politiques utilisant des autorisations vérifiées. Pour plus d'informations, consultez la section Créer un magasin de politiques avec une API connectée et un fournisseur d'identité dans le guide de l'utilisateur Amazon Verified Permissions.
- La console Lambda fournit un plan Python, que vous pouvez utiliser en choisissant Utiliser un plan puis en choisissant le plan. api-gateway-authorizer-python

Configuration d'un autorisateur Lambda

Après avoir créé une fonction Lambda, vous configurez la fonction Lambda en tant qu'autorisateur pour votre API. Vous configurez ensuite votre méthode pour appeler votre autorisateur Lambda afin de déterminer si un appelant peut invoquer votre méthode. Vous pouvez créer une fonction Lambda dans le même compte, ou dans un compte différent, à partir duquel vous avez créé votre API.

Vous pouvez tester votre autorisateur Lambda à l'aide des outils intégrés à la console API Gateway ou à l'aide de Postman. Pour obtenir des instructions sur la façon d'utiliser Postman pour tester votre fonction d'autorisation Lambda, consultez. the section called “Appel d'une API avec des mécanismes d'autorisation Lambda”

Configuration d'un autorisateur Lambda (console)

La procédure suivante montre comment créer un autorisateur Lambda dans la console API REST API Gateway. Pour en savoir plus sur les différents types d'autorisateurs Lambda, consultez. the section called “Choisir un type d'autorisateur Lambda”

REQUEST authorizer

Pour configurer un autorisateur **`REQUEST`** Lambda

1. Connectez-vous à la console API Gateway à l'adresse : https://console.aws.amazon.com/apigateway.
2. Sélectionnez une API, puis choisissez Authorizers.

3. Choisissez Créer un mécanisme d'autorisation.
4. Pour Nom du mécanisme d’autorisation, entrez un nom pour le mécanisme d’autorisation.
5. Pour Type de mécanisme d'autorisation, sélectionnez Lambda.
6. Pour la fonction Lambda, sélectionnez l' Région AWS endroit où vous avez créé votre fonction d'autorisation Lambda, puis entrez le nom de la fonction.
7. Laissez le rôle d'appel Lambda vide pour permettre à la console API REST API Gateway de définir une politique basée sur les ressources. La politique accorde à API Gateway l'autorisation d'invoquer la fonction d'autorisation Lambda. Vous pouvez également choisir de saisir le nom d'un rôle IAM pour permettre à API Gateway d'appeler la fonction d'autorisation Lambda. Pour un exemple de rôle, voirCréation d'un rôle IAM assumable.
8. Pour Charge utile d'événement Lambda, sélectionnez Effectuer une demande.
9. Pour Type de source d'identité, sélectionnez un type de paramètre. Les types de paramètre pris en charge sont `Header`, `Query string`, `Stage variable` et `Context`. Pour ajouter des sources d'identité supplémentaires, choisissez Ajouter un paramètre.
10. Pour mettre en cache la politique d'autorisation générée par le mécanisme d'autorisation, maintenez active l'option Mise en cache des autorisations. Lorsque la mise en cache des politiques est activée, vous pouvez modifier la valeur TTL. Le fait de définir TTL sur zéro désactive la mise en cache de la politique.

    Si vous activez la mise en cache, votre autorisateur doit renvoyer une politique applicable à toutes les méthodes d'une API. Pour appliquer une politique spécifique à une méthode, utilisez les variables `$context.path` de contexte et. `$context.httpMethod`
11. Choisissez Créer un mécanisme d'autorisation.

TOKEN authorizer

Pour configurer un autorisateur **`TOKEN`** Lambda

1. Connectez-vous à la console API Gateway à l'adresse : https://console.aws.amazon.com/apigateway.
2. Sélectionnez une API, puis choisissez Authorizers.
3. Choisissez Créer un mécanisme d'autorisation.
4. Pour Nom du mécanisme d’autorisation, entrez un nom pour le mécanisme d’autorisation.
5. Pour Type de mécanisme d'autorisation, sélectionnez Lambda.

6. Pour la fonction Lambda, sélectionnez l' Région AWS endroit où vous avez créé votre fonction d'autorisation Lambda, puis entrez le nom de la fonction.
7. Laissez le rôle d'appel Lambda vide pour permettre à la console API REST API Gateway de définir une politique basée sur les ressources. La politique accorde à API Gateway l'autorisation d'invoquer la fonction d'autorisation Lambda. Vous pouvez également choisir de saisir le nom d'un rôle IAM pour permettre à API Gateway d'appeler la fonction d'autorisation Lambda. Pour un exemple de rôle, voirCréation d'un rôle IAM assumable.
8. Pour Charge utile d'événement Lambda, sélectionnez Jeton.
9. Pour Source de jeton, entrez le nom de l'en-tête contenant le jeton d'autorisation. L'appelant doit inclure un en-tête de ce nom pour envoyer le jeton d'autorisation à l'autorisateur Lambda.
10. (Facultatif) Pour la validation du jeton, entrez une RegEx déclaration. API Gateway effectue la validation initiale du jeton d'entrée en fonction de cette expression et appelle le mécanisme d'autorisation quand la validation aboutit.
11. Pour mettre en cache la politique d'autorisation générée par le mécanisme d'autorisation, maintenez active l'option Mise en cache des autorisations. Si la mise en cache de la politique est activée, le nom de l'en-tête spécifié dans Source de jeton devient la clé de cache. Lorsque la mise en cache des politiques est activée, vous pouvez modifier la valeur TTL. Le fait de définir TTL sur zéro désactive la mise en cache de la politique.

    Si vous activez la mise en cache, votre autorisateur doit renvoyer une politique applicable à toutes les méthodes d'une API. Pour appliquer une politique spécifique à une méthode, vous pouvez désactiver la mise en cache des autorisations.
12. Choisissez Créer un mécanisme d'autorisation.

Après avoir créé votre autorisateur Lambda, vous pouvez le tester. La procédure suivante indique comment tester votre autorisateur Lambda.

REQUEST authorizer

Pour tester un autorisateur **`REQUEST`** Lambda

1. Connectez-vous à la console API Gateway à l'adresse : https://console.aws.amazon.com/apigateway.
2. Sélectionnez le nom de votre autorisateur.
3. Sous Autorisateur de test, entrez une valeur pour votre source d'identité.

Si vous utilisez lethe section called “Exemple de fonction REQUEST Lambda d'autorisation”, procédez comme suit :

a. Sélectionnez En-tête et entrez **`headerValue1`**, puis choisissez Ajouter un paramètre.

b. Sous Type de source d'identité, sélectionnez Chaîne de requête et entrez **`queryValue1`**, puis choisissez Ajouter un paramètre.

c. Sous Type de source d'identité, sélectionnez Variable d'étape et entrez **`stageValue1`**.

Vous ne pouvez pas modifier les variables de contexte pour l'appel de test, mais vous pouvez modifier le modèle d'événement de test API Gateway Authorizer pour votre fonction Lambda. Vous pouvez ensuite tester votre fonction d'autorisation Lambda avec des variables de contexte modifiées. Pour plus d'informations, consultez la section Tester les fonctions Lambda dans la console dans le Guide du AWS Lambda développeur.

4. Choisissez Tester le mécanisme d'autorisation.

TOKEN authorizer

Pour tester un autorisateur **`TOKEN`** Lambda

1. Connectez-vous à la console API Gateway à l'adresse : https://console.aws.amazon.com/apigateway.
2. Sélectionnez le nom de votre autorisateur.
3. Sous Autorisateur de test, entrez une valeur pour votre jeton.

   Si vous utilisez lethe section called “Exemple de fonction TOKEN Lambda d'autorisation”, procédez comme suit :

   - Pour AuthorizationToken, entrez. **`allow`**
4. Choisissez Tester le mécanisme d'autorisation.

   Si votre autorisateur Lambda refuse avec succès une demande dans l'environnement de test, le test répond par une `200 OK` réponse HTTP. Toutefois, en dehors de l'environnement de test, API Gateway renvoie une réponse `403 Forbidden` HTTP et la demande de méthode échoue.

Configurer un autorisateur Lambda ()AWS CLI

La commande create-authorizer suivante montre comment créer un autorisateur Lambda à l'aide du. AWS CLI

REQUEST authorizer

L'exemple suivant crée un `REQUEST` autorisateur et utilise l'`Authorizer`en-tête et la variable de `accountId` contexte comme sources d'identité :

```
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

L'exemple suivant crée un `TOKEN` autorisateur et utilise l'`Authorization`en-tête comme source d'identité :

```
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
```

Après avoir créé votre autorisateur Lambda, vous pouvez le tester. La test-invoke-authorizercommande suivante indique comment tester votre autorisateur Lambda :

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

Configurer une méthode pour utiliser un autorisateur Lambda (console)

Après avoir configuré votre autorisateur Lambda, vous devez l'associer à une méthode pour votre API.

Pour configurer une méthode d'API pour utiliser un mécanisme d'autorisation Lambda

1. Connectez-vous à la console API Gateway à l'adresse : https://console.aws.amazon.com/apigateway.
2. Sélectionnez une API.
3. Choisissez Ressources, puis choisissez une nouvelle méthode ou choisissez une méthode existante.
4. Dans l'onglet Demande de méthode, sous Paramètres de demande de méthode, choisissez Modifier.
5. Pour Mécanisme d'autorisation, dans le menu déroulant, sélectionnez le mécanisme d'autorisation Lambda que vous venez de créer.
6. (Facultatif) Si vous souhaitez transmettre le jeton d'autorisation au backend, choisissez En-têtes de demande HTTP. Choisissez Ajouter un en-tête, puis ajoutez le nom de l'en-tête d'autorisation. Dans Nom, entrez le nom d'en-tête qui correspond au nom de la source du jeton que vous avez spécifié lors de la création de l'autorisateur Lambda pour l'API. Cette étape ne s'applique pas aux mécanismes d'autorisation `REQUEST`.
7. Choisissez Enregistrer.
8. Sélectionnez Deploy API pour déployer l'API jusqu'à une étape. Pour un mécanisme d'autorisation `REQUEST` utilisant les variables d'étape, vous devez également définir les variables d'étape requises et spécifier leurs valeurs dans la page Étapes.

Configurer la méthode d'une API pour utiliser un autorisateur Lambda ()AWS CLI

Après avoir configuré votre autorisateur Lambda, vous devez l'associer à une méthode pour votre API. Vous pouvez créer une nouvelle méthode ou utiliser une opération de correctif pour associer un autorisateur à une méthode existante.

La commande put-method suivante montre comment créer une nouvelle méthode utilisant un autorisateur Lambda :

```
aws apigateway put-method --rest-api-id 1234123412 \
```

```
  --resource-id a1b2c3 \
  --http-method PUT \
  --authorization-type CUSTOM \
  --authorizer-id efg1234
```

La commande update-method suivante montre comment mettre à jour une méthode existante pour utiliser un autorisateur Lambda :

```
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"
```

Données d'entrée d'un mécanisme d'autorisation Lambda Amazon API Gateway

Format d'entrée **`TOKEN`**

Pour un mécanisme d'autorisation Lambda (anciennement appelé Custom Authorizer) de type `TOKEN`, vous devez spécifier un en-tête personnalisé comme Token Source (Source de jeton) lorsque vous configurez le mécanisme d'autorisation de votre API. Le client de l'API doit transmettre le jeton d'autorisation requis dans l'en-tête de la demande entrante. À réception de la demande de méthode entrante, API Gateway extrait le jeton à partir de l'en-tête personnalisé. Il transmet ensuite le jeton en tant que propriété `authorizationToken` de l'objet `event` de la fonction Lambda, en plus de l'ARN de méthode via la propriété `methodArn` :

```
{
    "type":"TOKEN",
    "authorizationToken":"{caller-supplied-token}",
    "methodArn":"arn:aws:execute-api:{regionId}:{accountId}:{apiId}/{stage}/{httpVerb}/
[{resource}/[{child-resources}]]"
}
```

Dans cet exemple, la propriété `type` spécifie le type de mécanisme d'autorisation, à savoir `TOKEN`. La valeur *`{caller-supplied-token}`* provient de l'en-tête d'autorisation d'une demande client et peut correspondre à n'importe quelle valeur de chaîne. La propriété `methodArn` est l'ARN de la demande de méthode entrante et elle est renseignée par API Gateway conformément à la configuration du mécanisme d'autorisation Lambda.

Format d'entrée **`REQUEST`**

Pour un mécanisme d'autorisation Lambda de type `REQUEST`, API Gateway transmet les paramètres de demande à la fonction Lambda du mécanisme d'autorisation de l'objet `event`. Les paramètres de demande comprennent les en-têtes, les paramètres des chemins, les paramètres des chaînes d'interrogation, les variables d'étape et certaines variables de contexte de demande. L'appelant de l'API peut définir les paramètres des chemins, les en-têtes et les paramètres des chaînes d'interrogation. Le développeur de l'API doit définir les variables d'étape pendant le déploiement de l'API, et API Gateway fournit le contexte de la demande au moment de l'exécution.

> **Note**
>
> Les paramètres de chemin peuvent être transmis comme paramètres de la demande à la fonction du mécanisme d'autorisation Lambda, mais ils ne peuvent pas être utilisés comme sources d'identité.

L'exemple suivant montre les données d'entrée pour un mécanisme d'autorisation `REQUEST` et une méthode d'API (`GET /request`) avec une intégration proxy :

```
{
  "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"
  }
}
```

Le contexte `requestContext` est un mappage de paires clé-valeur et correspond à la variable $context. Son résultat est dépendant de l'API.

API Gateway peut ajouter de nouvelles clés à la carte. Pour de plus amples informations sur l'entrée de la fonction Lambda dans l'intégration de proxy Lambda, veuillez consulter Format d'entrée d'une fonction Lambda pour l'intégration proxy.

Résultat d'un autorisateur Lambda API Gateway

La sortie de la fonction du mécanisme d'autorisation Lambda est un objet de type dictionnaire, qui doit inclure l'identifiant principal (`principalId`) et un document de stratégie (`policyDocument`) contenant la liste des déclarations de stratégie. La sortie peut également inclure un mappage `context` contenant des paires clé-valeur. Si l'API applique un plan d'utilisation (`apiKeySource` est

défini sur AUTHORIZER), la fonction du mécanisme d'autorisation Lambda doit renvoyer l'une des clés d'API du plan d'utilisation comme valeur de la propriété `usageIdentifierKey`.

Voici un exemple de sortie de ce type.

```
{
  "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}"
}
```

Dans cet exemple, une déclaration de stratégie indique s'il faut permettre ou interdire (`Effect`) au service d'exécution API Gateway d'appeler (`Action`) la méthode d'API spécifiée (`Resource`). Vous pouvez utiliser un caractère générique (*) pour spécifier un type de ressource (méthode). Pour plus d'informations sur le paramétrage des stratégies valides pour appeler une API, consultez Référence de déclaration de stratégie IAM pour l'exécution des API dans API Gateway.

Pour un ARN de méthode activé par autorisation, par exemple `arn:aws:execute-api:{regionId}:{accountId}:{apiId}/{stage}/{httpVerb}/[{resource}/[{child-resources}]]`, la longueur maximale est de 1 600 octets. Les valeurs de paramètre de chemin, dont la taille est déterminée au moment de l'exécution, peuvent entraîner un dépassement de limite de la longueur de l'ARN. Dans ce cas, le client API reçoit une réponse `414 Request URI too long`.

De plus, l'ARN de ressources, comme indiqué dans la sortie de déclaration de stratégie par l'autorisateur, est actuellement limitée à 512 caractères. Pour cette raison, vous ne devez pas utiliser d'URI avec un jeton JWT d'une longueur significative dans une URI de demande. Vous pouvez plutôt transmettre le jeton JWT en toute sécurité dans un en-tête de demande.

Vous pouvez accéder à la valeur `principalId` d'un modèle de mappage à l'aide de la variable `$context.authorizer.principalId`. C'est utile si vous souhaitez transmettre la valeur au backend. Pour plus d'informations, consultez $contextVariables pour les modèles de données, les autorisateurs, les modèles de mappage et la journalisation des CloudWatch accès.

Vous pouvez accéder à la valeur `stringKey`, `numberKey` ou `booleanKey` (par exemple, `"value"`, `"1"` ou `"true"`) du mappage `context` d'un modèle de mappage en appelant `$context.authorizer.stringKey`, `$context.authorizer.numberKey` ou `$context.authorizer.booleanKey`, respectivement. Les valeurs renvoyées sont toutes obtenues à l'aide de stringify. Notez que vous ne pouvez pas définir un objet ou un tableau JSON comme valeur valide d'une clé dans le mappage `context`.

Vous pouvez utiliser le mappage `context` pour renvoyer les informations d'identification mises en cache depuis le mécanisme d'autorisation vers le backend, via un modèle de mappage de demande d'intégration. En exploitant les informations d'identification mises en cache, le backend offre une meilleure expérience utilisateur et évite ainsi d'accéder aux clés secrètes et d'ouvrir des jetons d'autorisation pour chaque demande.

Pour l'intégration du proxy Lambda, API Gateway transmet l'objet `context` depuis un mécanisme d'autorisation Lambda directement à la fonction Lambda du backend via l'entrée `event`.
Vous pouvez récupérer les paires clé-valeur `context` de la fonction Lambda en appelant `$event.requestContext.authorizer.`*`key`*.

`{api-key}` représente une clé d'API dans le plan d'utilisation de l'étape d'API. Pour plus d'informations, consultez the section called “Plans d'utilisation”.

L'exemple de mécanisme d'autorisation Lambda renvoie l'exemple de sortie suivant. L'exemple de sortie contient une déclaration de politique visant à bloquer (`Deny`) les appels à la `GET` méthode pour l'`dev`étape d'une API (`ymy8tbxw7b`) d'un AWS compte (`123456789012`).

```
{
  "principalId": "user",
  "policyDocument": {
    "Version": "2012-10-17",
```

```
    "Statement": [
      {
        "Action": "execute-api:Invoke",
        "Effect": "Deny",
        "Resource": "arn:aws:execute-api:us-west-2:123456789012:ymy8tbxw7b/dev/GET/"
      }
    ]
  }
}
```

Appel d'une API avec des mécanismes d'autorisation Lambda API Gateway

Après avoir configuré le mécanisme d'autorisation Lambda (anciennement appelé Custom Authorizer) et déployé l'API, vous devez tester l'API avec le mécanisme d'autorisation Lambda activé. Pour cela, vous avez besoin d'un client REST, tel que cURL ou Postman. Dans les exemples suivants, nous utilisons Postman.

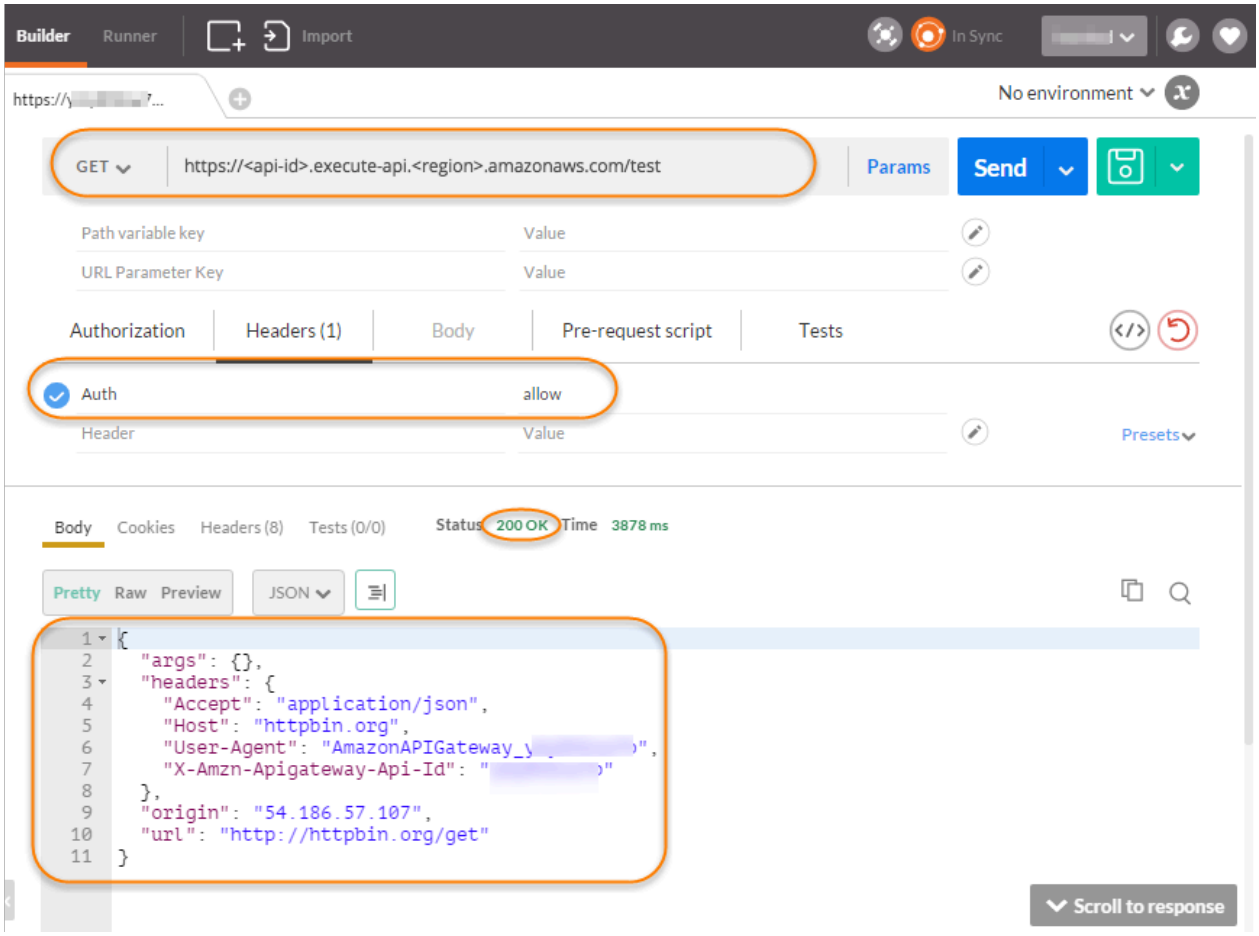
> **Note**
>
> Lors de l'appel d'une méthode activée par l'autorisateur, API Gateway n'enregistre pas l'appel CloudWatch si le jeton requis pour l'**`TOKEN`**autorisateur n'est pas défini, est nul ou est invalidé par l'expression de validation du jeton spécifiée. De même, API Gateway n'enregistre pas l'appel CloudWatch si l'une des sources d'identité requises pour l'`REQUEST`autorisateur n'est pas définie, est nulle ou est vide.

Nous détaillons ci-après comment utiliser Postman pour appeler ou tester une API avec le mécanisme d'autorisation Lambda `TOKEN`. La méthode peut servir à appeler une API avec un mécanisme d'autorisation Lambda `REQUEST` si vous spécifiez le chemin, l'en-tête ou les paramètres des chaînes d'interrogation requis de façon explicite.

Pour appeler une API avec le Custom **`TOKEN`** Authorizer

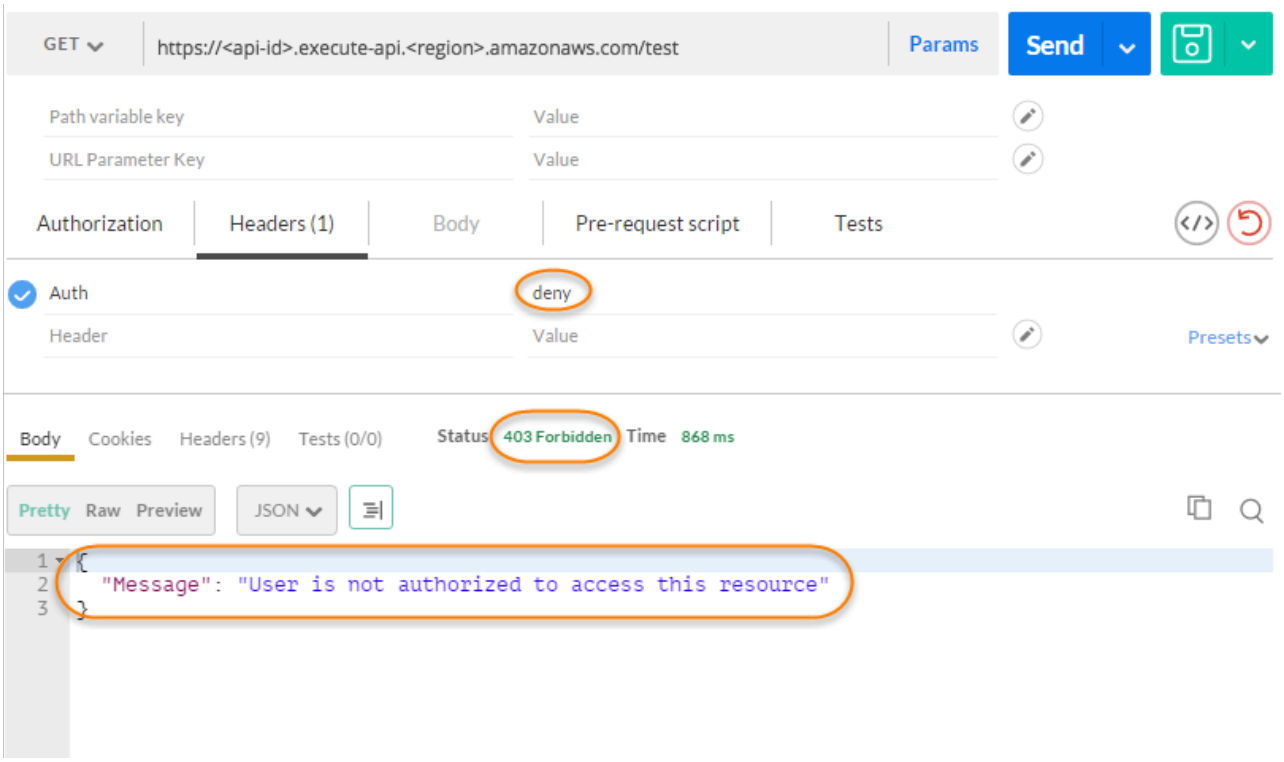
1. Ouvrez Postman, sélectionnez la méthode GET et collez la valeur du champ Invoke URL de l'API dans le champ d'URL adjacent.

   Ajoutez l'en-tête de jeton d'autorisation Lambda et définissez la valeur `allow`. Sélectionnez Send (Envoyer).



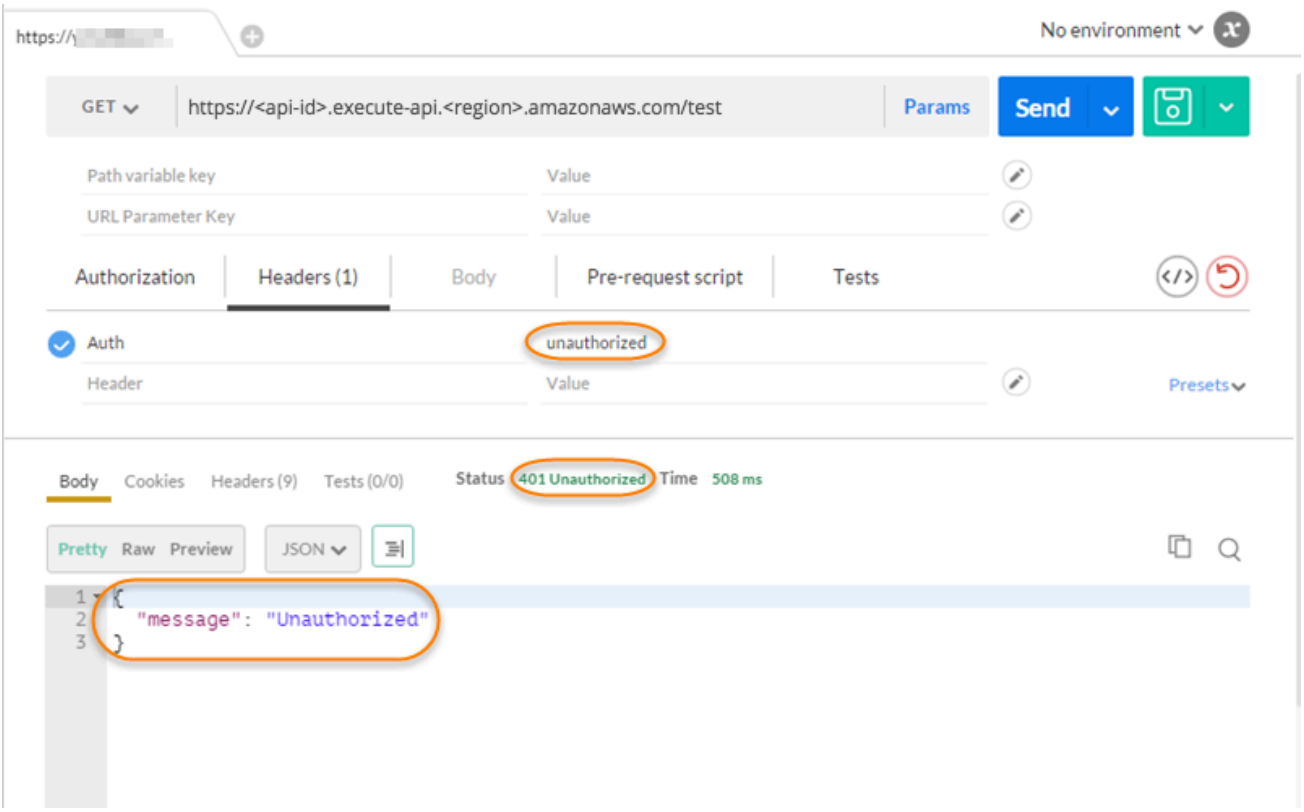
La réponse montre que le mécanisme d'autorisation Lambda API Gateway renvoie une réponse 200 OK et autorise l'appel à accéder au point de terminaison HTTP (http://httpbin.org/get) intégré à la méthode.

2. Toujours dans l'application Postman, remplacez la valeur de l'en-tête de jeton d'autorisation Lambda par `deny`. Sélectionnez Send (Envoyer).



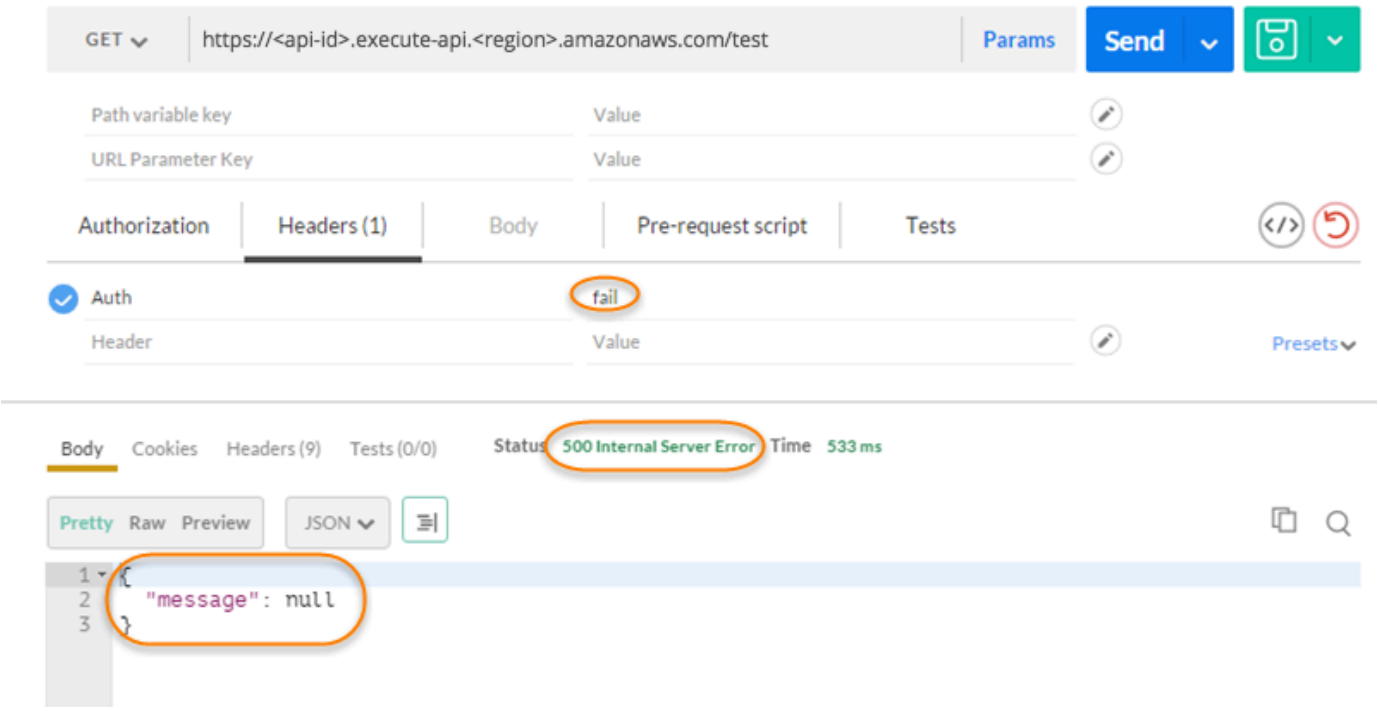
La réponse montre que le mécanisme d'autorisation Lambda API Gateway renvoie une réponse 403 Forbidden (403 Interdit) sans autoriser l'appel à accéder au point de terminaison HTTP.

3. Dans l'application Postman, remplacez la valeur de l'en-tête de jeton d'autorisation Lambda par `unauthorized`, puis sélectionnez Send (Envoyer).



La réponse montre qu'API Gateway renvoie une réponse 401 Unauthorized (401 Accès non autorisé) sans autoriser l'appel à accéder au point de terminaison HTTP.

4. Maintenant, remplacez la valeur de l'en-tête de jeton d'autorisation Lambda par `fail`. Sélectionnez Send (Envoyer).



La réponse montre qu'API Gateway renvoie une réponse 500 Internal Server Error (500 Erreur interne du serveur) sans autoriser l'appel à accéder au point de terminaison HTTP.

Configuration d'un mécanisme d'autorisation Lambda entre comptes

Vous pouvez désormais également utiliser une AWS Lambda fonction d'un autre AWS compte comme fonction d'autorisation d'API. Chaque compte peut se trouver dans n'importe quelle région où Amazon API Gateway est disponible. La fonction de mécanisme d'automatisation Lambda peut utiliser des stratégies d'authentification par jeton de porteur, par exemple OAuth ou SAML. Cela facilite la gestion centralisée et le partage de la fonction de mécanisme d'autorisation Lambda centrale entre plusieurs API API Gateway.

Dans cette section, nous montrons comment configurer une fonction d'autorisation Lambda entre comptes à l'aide de la console Amazon API Gateway.

Ces instructions supposent que vous disposez déjà d'une API API Gateway dans un AWS compte et d'une fonction d'autorisation Lambda dans un autre compte.

Configuration d'un mécanisme d'autorisation Lambda entre comptes à l'aide de la console API Gateway

Connectez-vous à la console Amazon API Gateway dans le compte qui contient votre API, puis procédez comme suit :

1. Choisissez votre API, puis dans le panneau de navigation principal, choisissez Mécanismes d’autorisation.
2. Choisissez Créer un mécanisme d'autorisation.
3. Pour Nom du mécanisme d’autorisation, entrez un nom pour le mécanisme d’autorisation.
4. Pour Type de mécanisme d'autorisation, sélectionnez Lambda.
5. Pour Fonction Lambda, entrez l'ARN complet de la fonction de mécanisme d'autorisation Lambda que vous avez dans votre deuxième compte.

   > **Note**
   >
   > Dans la console Lambda, vous trouverez l'ARN de votre fonction dans le coin supérieur droit de la fenêtre de la console.

6. Un avertissement accompagné d'une chaîne de commande `aws lambda add-permission` s'affichera. Cette politique accorde à API Gateway l'autorisation d'invoquer la fonction Lambda de mécanisme d'autorisation. Copiez la commande et enregistrez-la pour plus tard. Vous exécuterez la commande après avoir créé le mécanisme d'autorisation.
7. Laissez Rôle d'appel Lambda vide pour que la console API Gateway définisse une politique basée sur les ressources. Cette politique accorde à API Gateway l'autorisation d'invoquer la fonction Lambda de mécanisme d'autorisation. Vous pouvez également choisir d'entrer un rôle IAM pour autoriser API Gateway à invoquer la fonction Lambda de mécanisme d'autorisation. Pour voir un exemple de rôle, consultez Création d'un rôle IAM assumable.
8. Pour Charge utile d'événement Lambda, sélectionnez Jeton pour un mécanisme d'autorisation `TOKEN` ou Effectuer une demande pour un mécanisme d'autorisation `REQUEST`.
9. Selon le choix effectué à l'étape précédente, réalisez l'une des actions suivantes :

   a. Pour l'option Jeton, procédez comme suit :

   - Pour Source de jeton, entrez le nom de l'en-tête contenant le jeton d'autorisation. Le client d'API doit inclure un en-tête portant ce nom pour envoyer le jeton d'autorisation au mécanisme d'autorisation Lambda.

- Pour la validation du jeton, entrez éventuellement une RegEx instruction. API Gateway effectue la validation initiale du jeton d'entrée en fonction de cette expression et appelle le mécanisme d'autorisation quand la validation aboutit. Cela contribue à réduire le nombre d'appels à votre API.
- Pour mettre en cache la politique d'autorisation générée par le mécanisme d'autorisation, maintenez active l'option Mise en cache des autorisations. Quand la mise en cache de la stratégie est activée, vous pouvez choisir de remplacer la valeur TTL. Le fait de définir TTL sur zéro désactive la mise en cache de la politique. Si la mise en cache de la politique est activée, le nom de l'en-tête spécifié dans Source de jeton devient la clé de cache. Si plusieurs valeurs sont transmises à cet en-tête dans la demande, toutes les valeurs deviennent la clé de cache, l'ordre étant préservé.

> **Note**
>
> La valeur TTL par défaut est de 300 secondes. Actuellement, sa valeur maximale est de 3 600 secondes ; elle ne peut pas être augmentée.

b. Pour l'option Request, effectuez les opérations suivantes :

- Pour Type de source d'identité, sélectionnez un type de paramètre. Les types de paramètre pris en charge sont `Header`, `Query string`, `Stage variable` et `Context`. Pour ajouter des sources d'identité supplémentaires, choisissez Ajouter un paramètre.
- Pour mettre en cache la politique d'autorisation générée par le mécanisme d'autorisation, maintenez active l'option Mise en cache des autorisations. Quand la mise en cache de la stratégie est activée, vous pouvez choisir de remplacer la valeur TTL. Le fait de définir TTL sur zéro désactive la mise en cache de la politique.

  API Gateway utilise les sources d'identité spécifiées comme clé de mise en cache du mécanisme d'autorisation de la demande. Lorsque la mise en cache est activée, API Gateway appelle la fonction Lambda du mécanisme d'autorisation uniquement si toutes les sources d'identité spécifiées sont présentes lors de l'exécution. Si une source d'identité spécifiée est absente, nulle ou vide, API Gateway renvoie une réponse `401 Unauthorized` sans appeler la fonction Lambda du mécanisme d'autorisation.

  Lorsque plusieurs sources d'identité sont définies, elles sont toutes utilisées pour obtenir la clé du cache du mécanisme d'autorisation. Si une partie de la clé du cache est modifiée, le mécanisme d'autorisation ignore le document de la stratégie mis en cache et

en génère un nouveau. Si un en-tête à plusieurs valeurs est transmis dans la demande, toutes les valeurs font partie de la clé de cache, l'ordre étant préservé.

- Lorsque la mise en cache est désactivée, il n'est pas nécessaire de spécifier une source d'identité.

> **Note**
>
> Pour activer la mise en cache, votre mécanisme d'autorisation doit renvoyer une stratégie applicable à toutes les méthodes d'une API. Pour appliquer une politique spécifique à la méthode, vous pouvez désactiver Mise en cache des autorisations.

10. Choisissez Créer un mécanisme d'autorisation.
11. Collez la chaîne de `aws lambda add-permission` commande que vous avez copiée à l'étape précédente dans une AWS CLI fenêtre configurée pour votre deuxième compte. Remplacez `AUTHORIZER_ID` par l'ID de votre mécanisme d'autorisation. Cela permet d'accorder à votre premier compte l'accès à la fonction de mécanisme d'autorisation Lambda de votre deuxième compte.

Contrôle de l'accès à une API REST à l'aide de groupes d'utilisateurs Amazon Cognito en tant que mécanisme d'autorisation

Au lieu d'utiliser des rôles et stratégies IAM ou des mécanismes d'autorisation Lambda (anciennement appelés mécanismes d'autorisation personnalisée), vous pouvez utiliser un groupe d'utilisateurs Amazon Cognito pour contrôler les personnes pouvant accéder à votre API dans Amazon API Gateway.

Pour utiliser un groupe d'utilisateurs Amazon Cognito avec votre API, vous devez d'abord créer un mécanisme d'autorisation de type `COGNITO_USER_POOLS`, puis configurer une méthode d'API permettant d'utiliser ce mécanisme d'autorisation. Une fois l'API déployée, le client doit d'abord connecter l'utilisateur au groupe d'utilisateurs, obtenir une identité ou un jeton d'accès pour l'utilisateur, puis appeler la méthode API avec l'un des jetons, qui sont généralement définis sur l'en-tête `Authorization` de la demande. L'appel d'API réussit uniquement si le jeton requis est fourni et qu'il est valide. Dans le cas contraire, le client n'est pas autorisé à effectuer l'appel, il ne dispose pas des informations d'identification qui peuvent être autorisées.

Le jeton d'identité est utilisé pour autoriser les appels d'API en fonction des requêtes d'identité de l'utilisateur connecté. Le jeton d'accès est utilisé pour autoriser les appels d'API en fonction

des règles personnalisées des ressources protégées par un accès spécifié. Pour de plus amples informations, veuillez consulter Utilisation des jetons avec les groupes d'utilisateurs et Serveur de ressources et portées personnalisées.

Pour créer et configurer un groupe d'utilisateurs Amazon Cognito pour votre API, exécutez les tâches suivantes :

- Utilisez la console Amazon Cognito, le CLI/SDK ou l'API pour créer un groupe d'utilisateurs, ou utilisez-en un appartenant à un autre compte. AWS
- Utilisez la console API Gateway, CLI/kit SDK, ou encore l'API pour créer un mécanisme d'autorisation API Gateway avec le groupe d'utilisateurs choisi.
- Utilisez la console API Gateway, CLI/kit SDK, ou encore l'API pour activer le mécanisme d'autorisation sur les méthodes d'API sélectionnées.

Pour appeler des méthodes d'API avec un groupe d'utilisateurs activé, vos clients d'API effectuent les tâches suivantes :

- Utilisez CLI/SDK ou l'API Amazon Cognito pour ajouter un utilisateur dans le groupe d'utilisateurs choisi et obtenir un jeton d'identité ou un jeton d'accès. Pour en savoir plus sur l'utilisation des kits de développement logiciel, consultez les exemples de code pour Amazon Cognito AWS à l'aide de kits de développement logiciel.
- Utilisez un cadre spécifique au client pour appeler l'API API Gateway déployée et fournir le jeton approprié dans l'en-tête `Authorization`.

En tant que développeur d'API, vous devez fournir à vos développeurs clients l'ID du groupe d'utilisateurs, un ID client et éventuellement les secrets client associés, qui sont définis dans le cadre groupe d'utilisateurs.

> **Note**
>
> Pour permettre à un utilisateur de se connecter avec les informations d'identification Amazon Cognito et d'obtenir également des informations d'identification temporaires à utiliser avec les autorisations d'un rôle IAM, utilisez les identités fédérées Amazon Cognito. Pour chaque méthode HTTP du point de terminaison de ressources d'API, définissez le type d'autorisation, la catégorie `Method Execution`, sur `AWS_IAM`.

Dans cette section, nous allons expliquer comment créer un groupe d'utilisateurs, intégrer une API API Gateway au groupe d'utilisateurs et appeler une API intégrée au groupe d'utilisateurs.

Rubriques

- Obtenir les autorisations pour créer des mécanismes d'autorisation de groupe d'utilisateurs Amazon Cognito pour une API REST
- Création d'un groupe d'utilisateurs Amazon Cognito pour une API REST
- Intégration d'une API REST à un groupe d'utilisateurs Amazon Cognito
- Appel d'une API REST intégrée à un groupe d'utilisateurs Amazon Cognito
- Configuration d'un mécanisme d'autorisation Amazon Cognito entre comptes pour une API REST à l'aide de la console API Gateway
- Créez un autorisateur Amazon Cognito pour une API REST à l'aide de AWS CloudFormation

Obtenir les autorisations pour créer des mécanismes d'autorisation de groupe d'utilisateurs Amazon Cognito pour une API REST

Pour créer un mécanisme d'autorisation avec un groupe d'utilisateurs Amazon Cognito, vous devez disposer des autorisations `Allow` pour créer ou mettre à jour un mécanisme d'autorisation avec le groupe d'utilisateurs Amazon Cognito choisi. Le document de stratégie IAM suivant présente un exemple de ce type d'autorisation :

```
{
    "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_aD06NQmj0",
                        "arn:aws:cognito-idp:us-east-1:234567890123:userpool/us-
east-1_xJ1MQtPEN"
                    ]
                }
            }
        }
    ]
}
```

Assurez-vous que la stratégie est attachée à un groupe IAM auquel vous appartenez ou à un rôle IAM qui vous a été attribué.

Dans le document de stratégie précédent, l'action `apigateway:POST` est destinée à créer un nouveau mécanisme d'autorisation et l'action `apigateway:PATCH`, à mettre à jour un mécanisme d'autorisation existant. Vous pouvez restreindre la stratégie à une région spécifique ou à une API particulière en écrasant les deux premiers caractères génériques (*), respectivement, des valeurs `Resource`.

Les clauses `Condition` utilisées ici ont pour but de limiter les autorisations `Allowed` aux groupes d'utilisateurs spécifiés. Lorsqu'une clause `Condition` est présente, l'accès aux groupes d'utilisateurs ne correspondant pas aux conditions est refusé. Lorsqu'une autorisation n'est associée à aucune clause `Condition`, l'accès à tous les groupes d'utilisateurs est autorisé.

Vous disposez des options suivantes pour définir la clause `Condition` :

- Vous pouvez définir une expression conditionnelle `ArnLike` ou `ArnEquals` afin de permettre la création ou la mise à jour de mécanismes d'autorisation `COGNITO_USER_POOLS` uniquement avec les groupes d'utilisateurs spécifiés.

- Vous pouvez définir une expression conditionnelle `ArnNotLike` ou `ArnNotEquals` afin de permettre la création ou la mise à jour de mécanismes d'autorisation `COGNITO_USER_POOLS` avec tout groupe d'utilisateurs non spécifié dans l'expression.
- Vous pouvez omettre la clause `Condition` pour autoriser la création ou la mise à jour de mécanismes d'autorisations `COGNITO_USER_POOLS` avec n'importe quel groupe d'utilisateurs, de n'importe quel compte AWS et dans n'importe quelle région.

Pour de plus amples informations sur les expressions conditionnelles Amazon Resource Name (ARN), veuillez consulter Opérateurs de condition d'ARN (Amazon Resource Name). Comme le montre l'exemple, `apigateway:CognitoUserPoolProviderArn` est une liste d'ARN des groupes d'utilisateurs `COGNITO_USER_POOLS` qui peuvent ou ne peuvent pas être utilisés avec un mécanisme d'autorisation API Gateway de type `COGNITO_USER_POOLS`.

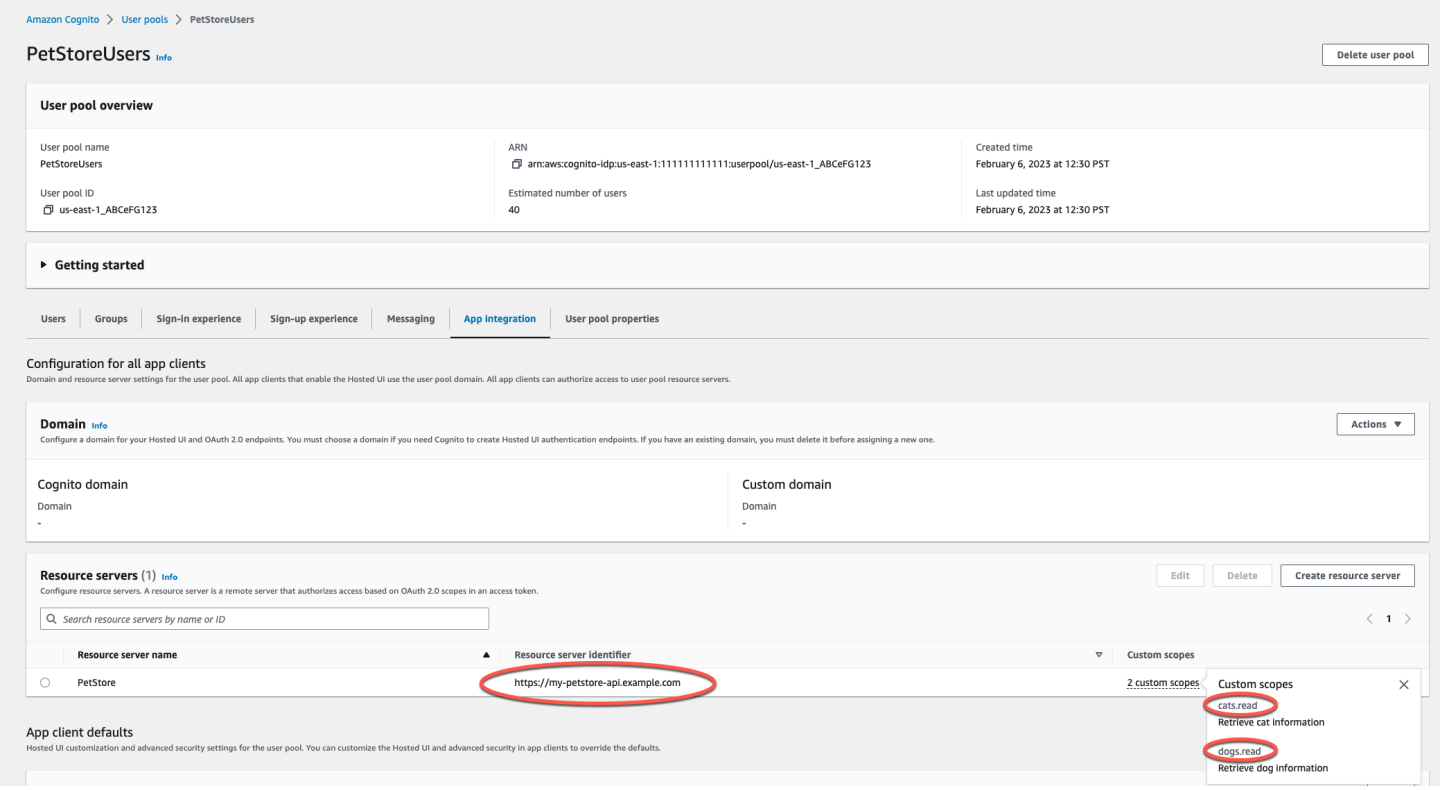
Création d'un groupe d'utilisateurs Amazon Cognito pour une API REST

Avant d'intégrer votre API à un groupe d'utilisateurs, vous devez créer ce groupe d'utilisateurs dans Amazon Cognito. La configuration de votre groupe d'utilisateurs doit respecter tous les quotas de ressources pour Amazon Cognito. Toutes les variables Amazon Cognito définies par l'utilisateur, telles que les groupes, les utilisateurs et les rôles doivent utiliser uniquement des caractères alphanumériques. Pour obtenir des instructions afin de créer un groupe d'utilisateurs, consultez Didacticiel : Création d'un groupe d'utilisateurs dans le Guide du développeur Amazon Cognito.

Notez l'ID du groupe d'utilisateurs, l'ID du client et les codes secrets du client. Le client doit fournir ces informations à Amazon Cognito pour permettre à l'utilisateur de s'inscrire dans le groupe d'utilisateurs, de se connecter au groupe d'utilisateurs et d'obtenir un jeton d'identité ou d'accès à inclure dans les demandes pour appeler les méthodes d'API configurées avec le groupe d'utilisateurs. Vous devez également spécifier le nom du groupe d'utilisateurs lorsque vous le configurez en tant que mécanisme d'autorisation dans API Gateway, comme décrit dans la section suivante.

Si vous utilisez des jetons d'accès pour autoriser des appels de méthode d'API, assurez-vous de configurer l'intégration de l'application avec le groupe d'utilisateurs pour configurer les règles personnalisées de votre choix sur un serveur de ressources donné. Pour de plus amples informations sur l'utilisation de jetons avec des groupes d'utilisateurs Amazon Cognito, veuillez consulter Utilisation des jetons avec les groupes d'utilisateurs. Pour de plus amples informations sur les serveurs de ressources, veuillez consulter Définition des serveurs de ressources pour votre groupe d'utilisateurs.

Notez les identificateurs du serveur de ressources configurés et les noms de règles personnalisées. Vous en aurez besoin pour construire les noms complets de règle d'accès pour OAuth Scopes (Règles OAuth) qui sont utilisés par le mécanisme d'autorisation `COGNITO_USER_POOLS`.



Intégration d'une API REST à un groupe d'utilisateurs Amazon Cognito

Après avoir créé un groupe d'utilisateurs Amazon Cognito, dans API Gateway, vous devez ensuite créer un mécanisme d'autorisation `COGNITO_USER_POOLS` qui utilise le groupe d'utilisateurs. La procédure suivante montre comment procéder à l'aide de la console API Gateway.

> **Note**
>
> Vous pouvez utiliser l'action `CreateAuthorizer` pour créer un mécanisme d'autorisation `COGNITO_USER_POOLS` qui utilise plusieurs groupes d'utilisateurs. Vous pouvez utiliser jusqu'à 1 000 groupes d'utilisateurs pour un mécanisme d'autorisation `COGNITO_USER_POOLS`. Cette limite ne peut pas être augmentée.

> **Important**
>
> Après avoir effectué l'une des procédures ci-dessous, vous devez déployer ou redéployer votre API pour propager les modifications. Pour plus d'informations sur le déploiement de votre API, consultez Déploiement d'une API REST dans Amazon API Gateway.

Pour créer un mécanisme d'autorisation **`COGNITO_USER_POOLS`** à l'aide de la console API Gateway

1. Créez une API ou sélectionnez une API existante dans API Gateway.
2. Dans le panneau de navigation principal, choisissez Mécanismes d'autorisation.
3. Choisissez Créer un mécanisme d'autorisation.
4. Pour configurer le nouveau mécanisme d'autorisation afin d'utiliser un groupe d'utilisateurs, procédez comme suit :
   a. Pour Nom du mécanisme d’autorisation, entrez un nom.
   b. Pour Type de mécanisme d'autorisation, sélectionnez Cognito.
   c. Pour le groupe d'utilisateurs Cognito, choisissez l' Région AWS endroit où vous avez créé votre Amazon Cognito et sélectionnez un groupe d'utilisateurs disponible.

      Vous pouvez utiliser une variable d'étape pour définir votre groupe d'utilisateurs. Utilisez le format suivant pour votre groupe d'utilisateurs :`arn:aws:cognito-idp:`*`us-east-2`*`:`*`111122223333`*`:userpool/${stageVariables.`*`MyUserPool`*`}`.
   d. Pour Source du jeton, entrez **`Authorization`** comme nom d'en-tête pour transmettre le jeton d'identité ou le jeton d'accès renvoyé par Amazon Cognito lorsqu'un utilisateur se connecte avec succès.
   e. (Facultatif) Entrez une expression régulière dans le champ Validation du jeton pour valider le champ `aud` (public) du jeton d'identité avant que la demande soit autorisée avec Amazon Cognito.. Notez que lors de l'utilisation d'un jeton d'accès, cette validation rejette la demande en raison du jeton d'accès ne contenant pas le champ `aud`.
   f. Choisissez Créer un mécanisme d'autorisation.
5. Après avoir créé le mécanisme d'autorisation `COGNITO_USER_POOLS`, vous pouvez éventuellement tester son appel en fournissant un jeton d'identité alloué à partir du groupe d'utilisateurs. Vous pouvez obtenir ce jeton d'identité en appelant le kit SDK d'identité Amazon Cognito afin de connecter l'utilisateur. Vous pouvez également utiliser l'action `InitiateAuth`.

Si vous ne configurez pas de Portées d'autorisation, API Gateway traite le jeton fourni comme un jeton d'identité.

La procédure précédente crée un mécanisme d'autorisation `COGNITO_USER_POOLS` qui utilise le groupe d'utilisateurs Amazon Cognito que vous venez de créer. En fonction de la façon dont vous activez le mécanisme d'autorisation sur une méthode d'API, vous pouvez utiliser un jeton d'identité ou un jeton d'accès qui est alloué à partir du groupe d'utilisateurs intégré.

Pour configurer un mécanisme d'autorisation **`COGNITO_USER_POOLS`** sur les méthodes

1. Sélectionnez Ressources. Choisissez une nouvelle méthode ou choisissez une méthode existante. Si nécessaire, créez une ressource.
2. Dans l'onglet Demande de méthode, sous Paramètres de demande de méthode, choisissez Modifier.
3. Pour Mécanisme d'autorisation, dans le menu déroulant, sélectionnez les mécanismes d'autorisation de groupe d'utilisateurs Amazon Cognito que vous venez de créer.
4. Pour utiliser un jeton d'identité, procédez comme suit :

   a. Laissez l'option Portées d'autorisation vide.

   b. Si nécessaire, dans Demande d'intégration, ajoutez les expressions `$context.authorizer.claims['property-name']` ou `$context.authorizer.claims.property-name` dans un modèle de mappage de corps pour transmettre la propriété des champs standard d'identité spécifiée du groupe d'utilisateurs au backend. Pour les noms de propriété simples, tels que `sub` ou `custom-sub`, les deux notations sont identiques. Pour les noms de propriété complexes, tels que `custom:role`, vous ne pouvez pas utiliser la notation par points. Par exemple, les expressions de mappage suivantes transmettent les champs standard `sub` et `email` de la requête au backend :

```
{
 "context" : {
  "sub" : "$context.authorizer.claims.sub",
  "email" : "$context.authorizer.claims.email"
 }
}
```

Si vous avez déclaré un champ de requête personnalisé lors de la configuration d'un groupe d'utilisateurs, vous pouvez respecter le même modèle pour accéder aux champs personnalisés. L'exemple suivant permet d'obtenir un champ `role` personnalisé d'une requête :

```
{
 "context" : {
  "role" : "$context.authorizer.claims.role"
    }
}
```

Si le champ de requête personnalisé est déclaré en tant que `custom:role`, utilisez l'exemple suivant pour obtenir la propriété de la requête :

```
{
 "context" : {
  "role" : "$context.authorizer.claims['custom:role']"
    }
}
```

5. Pour utiliser un jeton d'accès, procédez comme suit :

   a. Pour Portées d'autorisation, entrez un ou plusieurs noms complets d'une portée qui a été configurée lorsque le groupe d'utilisateurs Amazon Cognito a été créé. Par exemple, en suivant l'exemple fourni dans Création d'un groupe d'utilisateurs Amazon Cognito pour une API REST, l'une des portées est `https://my-petstore-api.example.com/cats.read`.

      Lors de l'exécution, l'appel de méthode réussit si une portée indiquée sur la méthode de cette étape correspond à une portée demandée dans le jeton entrant. Dans le cas contraire, l'appel échoue en renvoyant une réponse `401 Unauthorized`.

   b. Choisissez Enregistrer.

6. Répétez ces étapes pour les autres méthodes de votre choix.

Avec le mécanisme d'autorisation `COGNITO_USER_POOLS`, si l'option OAuth Scopes n'est pas spécifiée, API Gateway traite le jeton fourni comme un jeton d'identité et vérifie l'identité demandée par rapport à celle du groupe d'utilisateurs. Sinon, API Gateway traite le jeton fourni comme un jeton

d'accès et vérifie les portées d'accès qui sont demandées dans le jeton par rapport aux portées d'autorisation déclarées sur la méthode.

Au lieu d'utiliser la console API Gateway, vous pouvez également activer un groupe d'utilisateurs Amazon Cognito sur une méthode en spécifiant un fichier de définition OpenAPI et en important la définition d'API dans API Gateway.

Pour importer un mécanisme d'autorisation COGNITO_USER_POOLS avec un fichier de définition OpenAPI

1. Créez (ou exportez) un fichier de définition OpenAPI pour votre API.
2. Spécifiez la définition JSON du mécanisme d'autorisation `COGNITO_USER_POOLS` (`MyUserPool`) dans la section `securitySchemes` dans OpenAPI 3.0 ou dans la section `securityDefinitions` dans OpenAPI 2.0 comme suit :

   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. Pour utiliser le jeton d'identité pour l'autorisation de méthode, ajoutez `{ "MyUserPool": [] }` à la définition `security` de la méthode, comme indiqué dans la méthode GET suivante sur la ressource racine.

```
"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. Pour utiliser le jeton d'accès pour l'autorisation de méthode, modifiez la définition de sécurité ci-dessus pour `{ "MyUserPool": [resource-server/scope, ...] }`:

```
 "paths": {
   "/": {
     "get": {
       "consumes": [
         "application/json"
       ],
       "produces": [
         "text/html"
       ],
       "responses": {
         "200": {
           "description": "200 response",
           "headers": {
             "Content-Type": {
               "type": "string"
             }
           }
         }
       },
       "security": [
         {
           "MyUserPool": ["https://my-petstore-api.example.com/cats.read",
"http://my.resource.com/file.read"]
         }
       ],
       "x-amazon-apigateway-integration": {
         "type": "mock",
         "responses": {
           "default": {
             "statusCode": "200",
```

```
            "responseParameters": {
              "method.response.header.Content-Type": "'text/html'"
            },
          }
        },
        "requestTemplates": {
          "application/json": "{\"statusCode\": 200}"
        },
        "passthroughBehavior": "when_no_match"
      }
    },
    ...
}
```

5. Si nécessaire, vous pouvez définir d'autres paramètres de configuration d'API à l'aide des définitions ou des extensions OpenAPI appropriées. Pour plus d'informations, consultez Utilisation des extensions API Gateway vers OpenAPI.

Appel d'une API REST intégrée à un groupe d'utilisateurs Amazon Cognito

Pour appeler une méthode pour laquelle un mécanisme d'autorisation de groupe d'utilisateurs est configuré, le client doit effectuer les opérations suivantes :

- Autoriser l'utilisateur à s'inscrire dans le groupe d'utilisateurs.
- Autoriser l'utilisateur à se connecter au groupe d'utilisateurs.
- Obtenez un jeton d'identité ou d'accès de l'utilisateur connecté à partir du groupe d'utilisateurs.
- Inclure le jeton dans l'en-tête `Authorization` (ou tout autre en-tête vous avez spécifié lors de la création du mécanisme d'autorisation).

Vous pouvez utiliser AWS Amplify pour effectuer ces tâches. Pour plus d'informations, consultez Intégration d'Amazon Cognito aux applications Web et mobiles.

- Pour Android, consultez Mise en route avec Amplify pour Android.
- Pour utiliser iOS, consultez Mise en route avec Amplify pour iOS.
- Pour l'utiliser JavaScript, consultez Getting Started with Amplify pour Javascript.

Configuration d'un mécanisme d'autorisation Amazon Cognito entre comptes pour une API REST à l'aide de la console API Gateway

Vous pouvez désormais également utiliser un groupe d'utilisateurs Amazon Cognito d'un autre AWS compte comme autorisateur d'API. Le groupe d'utilisateurs Amazon Cognito peut utiliser des stratégies d'authentification par jeton de porteur, par exemple OAuth ou SAML. Cela facilite la gestion centralisée et le partage d'un mécanisme d'autorisation d'un groupe d'utilisateurs Amazon Cognito central entre plusieurs API API Gateway.

Dans cette section, nous montrons comment configurer un groupe d'utilisateurs Amazon Cognito entre comptes à l'aide de la console Amazon API Gateway.

Ces instructions supposent que vous disposez déjà d'une API API Gateway sur un AWS compte et d'un groupe d'utilisateurs Amazon Cognito sur un autre compte.

Configuration d'un mécanisme d'autorisation Amazon Cognito entre comptes à l'aide de la console API Gateway

Connectez-vous à la console Amazon API Gateway dans le compte qui contient votre API, puis procédez comme suit :

1. Créez une API ou sélectionnez une API existante dans API Gateway.
2. Dans le panneau de navigation principal, choisissez Mécanismes d'autorisation.
3. Choisissez Créer un mécanisme d'autorisation.
4. Pour configurer le nouveau mécanisme d'autorisation afin d'utiliser un groupe d'utilisateurs, procédez comme suit :

    a. Pour Nom du mécanisme d’autorisation, entrez un nom.

    b. Pour Type de mécanisme d'autorisation, sélectionnez Cognito.

    c. Pour Groupe d'utilisateurs Cognito, entrez l'ARN complet du groupe d'utilisateurs que vous avez dans votre deuxième compte.

    > **Note**
    >
    > Dans la console Amazon Cognito, vous pouvez trouver l'ARN de votre groupe d'utilisateurs dans le champ Pool ARN (ARN du groupe) du volet General Settings (Paramètres généraux).

d. Pour Source du jeton, entrez **`Authorization`** comme nom d'en-tête pour transmettre le jeton d'identité ou le jeton d'accès renvoyé par Amazon Cognito lorsqu'un utilisateur se connecte avec succès.

e. (Facultatif) Entrez une expression régulière dans le champ Validation du jeton pour valider le champ `aud` (public) du jeton d'identité avant que la demande soit autorisée avec Amazon Cognito.. Notez que lors de l'utilisation d'un jeton d'accès, cette validation rejette la demande en raison du jeton d'accès ne contenant pas le champ `aud`.

f. Choisissez Créer un mécanisme d'autorisation.

Créez un autorisateur Amazon Cognito pour une API REST à l'aide de AWS CloudFormation

Vous pouvez l'utiliser AWS CloudFormation pour créer un groupe d'utilisateurs Amazon Cognito et un autorisateur Amazon Cognito. L'exemple de AWS CloudFormation modèle effectue les opérations suivantes :

- Créez un groupe d’utilisateurs Amazon Cognito. Un client doit d’abord connecter l’utilisateur à un groupe d’utilisateurs et obtenir un jeton d’identité ou d’accès. Si vous utilisez des jetons d'accès pour autoriser des appels de méthode d'API, assurez-vous de configurer l'intégration de l'application avec le groupe d'utilisateurs pour configurer les règles personnalisées de votre choix sur un serveur de ressources donné.
- Il crée une API API Gateway avec une méthode `GET`.
- Crée un mécanisme d’autorisation Amazon Cognito qui utilise l’en-tête `Authorization` comme source du jeton.

```
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
          - /
```

Configuration des intégrations d'API REST

Après avoir configuré une méthode d'API, vous devez l'intégrer à un point de terminaison dans le serveur principal. Un point de terminaison principal est également appelé point de terminaison d'intégration et peut être une fonction Lambda, une page Web HTTP ou AWS une action de service.

Comme avec la méthode d'API, l'intégration de l'API possède une demande d'intégration et une réponse d'intégration. Une demande d'intégration encapsule une demande HTTP reçue par le backend. Elle peut ou non être différente de la demande de méthode soumise par le client. Une réponse d'intégration est une réponse HTTP encapsulant la sortie renvoyée par le backend.

La configuration d'une demande d'intégration comprend les opérations suivantes : la configuration de la transmission des demandes de méthode soumises par le client au backend ; la configuration de la transformation des données de la demande, le cas échéant, aux données de la demande d'intégration ; la définition de la fonction Lambda à appeler, la définition du serveur HTTP auquel envoyer la demande entrante, ou de l'action de service AWS à appeler.

La configuration d'une réponse d'intégration (applicable uniquement aux intégrations autres que de proxy) comprend les opérations suivantes : la configuration de la transmission des résultats renvoyés par le backend à une réponse de méthode d'un code de statut donné, la configuration de la transformation des paramètres de réponse d'intégration spécifiés aux paramètres de réponse de méthode préconfigurés, et la configuration du mappage du corps de la réponse d'intégration corps au corps de la réponse de la méthode en fonction des modèles de mappage de corps spécifiés.

Par programmation, une demande d'intégration est encapsulée par la ressource `Integration` et une réponse d'intégration par la ressource `IntegrationResponse` d'API Gateway.

Pour configurer une demande d'intégration, vous créez une ressource `Integration` et vous l'utilisez pour configurer l'URL du point de terminaison de l'intégration. Vous définissez ensuite les autorisations IAM d'accès au backend, et vous spécifiez les mappages pour transformer les données des demandes entrantes avant de les transmettre au backend. Pour configurer une réponse d'intégration pour l'intégration autre que d'un proxy, vous créez une ressource `IntegrationResponse` et vous l'utilisez pour définir sa réponse de méthode cible. Vous pouvez ensuite configurer le mappage de la sortie du backend à la réponse de la méthode.

Rubriques

- Configuration d'une demande d'intégration dans API Gateway
- Configuration d'une réponse d'intégration dans API Gateway
- Configuration d'intégrations Lambda dans API Gateway
- Configuration des intégrations HTTP dans API Gateway
- Configuration des intégrations privées API Gateway
- Configuration des intégrations fictives dans API Gateway

Configuration d'une demande d'intégration dans API Gateway

Pour configurer une demande d'intégration, vous effectuez les tâches facultatives et obligatoires suivantes :

1. Choisissez un type d'intégration qui détermine la façon dont la demande de méthode est transmise au backend.
2. Pour les intégrations non fictives, spécifiez une méthode HTTP et l'URI du point de terminaison de l'intégration ciblée, à l'exception de l'intégration `MOCK`.
3. Pour les intégrations avec les fonctions Lambda et les AWS autres actions de service, définissez un rôle IAM avec les autorisations requises pour qu'API Gateway appelle le backend en votre nom.
4. Pour les intégrations autres que de proxy, définissez les mappages de paramètres nécessaires pour mapper les paramètres de demande de méthode prédéfinis sur les paramètres de demande d'intégration correspondants.
5. Pour les intégrations autres que de proxy, définissez les mappages de corps nécessaires pour mapper le corps de la demande de méthode entrante d'un type de contenu donné en fonction du modèle de mappage spécifié.
6. Pour les intégrations autres que de proxy, spécifiez la condition selon laquelle les données de la demande de méthode entrante sont transmises au backend en l'état.
7. Vous pouvez également spécifier la manière de gérer la conversion de type pour une charge utile binaire.
8. Si vous le souhaitez, vous pouvez déclarer un nom d'espace de nom de cache et des paramètres clés de cache pour activer la mise en cache d'API.

Ces tâches impliquent la création d'une ressource d'intégration d'API Gateway et la configuration des valeurs de propriétés correspondantes. Vous pouvez le faire à l'aide de la console API Gateway, de AWS CLI commandes, d'un AWS SDK ou de l'API REST API Gateway.

Rubriques

- Tâches de base d'une demande d'intégration d'API
- Choisir un type d'intégration d’API API Gateway
- Configuration de l'intégration de proxy avec une ressource de proxy
- Configuration d'une demande d'intégration d'API à l'aide de la console API Gateway

Tâches de base d'une demande d'intégration d'API

Une demande d'intégration est une demande HTTP qu'API Gateway soumet au backend en la transmettant avec les données de demande soumises par le client et en les transformant, le cas échéant. La méthode HTTP (ou verbe) et l'URI de la demande d'intégration sont dictées par le

backend (à savoir, le point de terminaison d'intégration). Elles peuvent être identiques ou différentes de la méthode HTTP et de l'URI de la demande de méthode, respectivement.

Par exemple, lorsqu'une fonction Lambda renvoie un fichier récupéré dans Amazon S3, vous pouvez exposer cette opération intuitivement en tant que demande de méthode `GET` au client même si la demande d'intégration correspondante nécessite l'utilisation d'une demande `POST` pour appeler la fonction Lambda. Pour un point de terminaison HTTP, il est probable que la demande de méthode et la demande d'intégration correspondante utilisent toutes deux le même verbe HTTP. Toutefois, cela n'est pas obligatoire. Vous pouvez intégrer la demande de méthode suivante :

```
GET /{var}?query=value
Host: api.domain.net
```

Avec la demande d'intégration suivante :

```
POST /
Host: service.domain.com
Content-Type: application/json
Content-Length: ...

{
   path: "{var}'s value",
   type: "value"
}
```

En tant que développeur d'API, vous pouvez utiliser n'importe quel verbe HTTP et URI pour qu'une demande de méthode corresponde à vos besoins. Cependant, vous devez suivre les conditions du point de terminaison de l'intégration. Lorsque les données de la demande de méthode sont différentes des données de la demande d'intégration, vous pouvez rapprocher la différence en fournissant des mappages à partir des données de demande de la méthode aux données de la demande d'intégration.

Dans les exemples précédents, le mappage traduit les valeurs de variable de chemin d'accès (`{var}`) et de paramètre de demande (`query`) de la demande de méthode `GET` aux valeurs des propriétés `path` et `type` de charge utile de la demande d'intégration. Les autres données de demandes mappables comprennent les en-têtes et le corps des demandes. Celles-ci sont décrites dans Configuration des mappages de données de demande et de réponse à l'aide de la console API Gateway.

Lors de la configuration de la demande d'intégration HTTP ou de proxy HTTP, vous attribuez l'URL du point de terminaison HTTP backend en tant que valeur d'URI de la demande d'intégration. Par exemple, dans l' PetStoreAPI, la demande de méthode pour obtenir une page de pets possède l'URI de demande d'intégration suivant :

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

Lors de la configuration de l'intégration Lambda ou de proxy Lambda, vous attribuez l'Amazon Resource Name (ARN) pour appeler la fonction Lambda en tant que valeur d'URI de la demande d'intégration. Cet ARN présente le format suivant :

```
arn:aws:apigateway:api-region:lambda:path//2015-03-31/functions/arn:aws:lambda:lambda-region:account-id:function:lambda-function-name/invocations
```

La partie après `arn:aws:apigateway:`*`api-region`*`:lambda:path/`, à savoir `/2015-03-31/functions/arn:aws:lambda:`*`lambda-region`*`:`*`account-id`*`:function:`*`lambda-function-name`*`/invocations`, est le chemin d'URI de l'API REST de l'action Invoke Lambda. Si vous utilisez la console API Gateway pour configurer l'intégration Lambda, API Gateway crée l'ARN et l'attribue à l'URI d'intégration après vous avoir invité à choisir le *`lambda-function-name`* à partir d'une région.

Lorsque vous configurez la demande d'intégration avec une autre action de AWS service, l'URI de la demande d'intégration est également un ARN, similaire à l'intégration avec l'action Lambda`Invoke`. Par exemple, pour l'intégration avec l'GetBucketaction d'Amazon S3, l'URI de demande d'intégration est un ARN au format suivant :

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

L'URI de la demande d'intégration correspond à la convention de chemin pour spécifier l'action, où *`{bucket}`* est l'espace réservé d'un nom de compartiment. Une action de AWS service peut également être référencée par son nom. Grâce au nom de l'action, l'URI de demande d'intégration pour l'action `GetBucket` d'Amazon S3 devient la suivante :

```
arn:aws:apigateway:api-region:s3:action/GetBucket
```

Grâce à l'URI de la demande d'intégration basée sur l'action, le nom du compartiment (*`{bucket}`*) doit être spécifié dans le corps de la demande d'intégration (`{ Bucket: "`*`{bucket}`*`" }`), en respectant le format d'entrée de l'action `GetBucket`.

Pour les AWS intégrations, vous devez également configurer les informations d'identification pour permettre à API Gateway d'appeler les actions intégrées. Vous pouvez créer un rôle IAM ou choisir un rôle IAM existant pour qu'API Gateway appelle l'action, puis spécifier le rôle à l'aide de son ARN. Voici un exemple de cet ARN :

```
arn:aws:iam::account-id:role/iam-role-name
```

Ce rôle IAM doit contenir une stratégie permettant l'exécution de l'action. Il doit également avoir déclaré API Gateway (dans la relation d'approbation du rôle) en tant qu'entité de confiance pour assumer le rôle. Ces autorisations peuvent être accordées sur l'action même. Ce sont les autorisations basées sur les ressources. Pour l'intégration Lambda, vous pouvez appeler l'action addPermission de Lambda pour définir les autorisations basées sur les ressources, puis définissez `credentials` sur null dans la demande d'intégration API Gateway.

Nous avons vu la configuration de l'intégration de base. Les paramètres avancés comprennent les données de demande de méthode de mappage aux données de demande d'intégration. Après avoir présenté la configuration de base pour une réponse d'intégration, nous abordons les rubriques avancées dans Configuration des mappages de données de demande et de réponse à l'aide de la console API Gateway, où nous abordons également la transmission de la charge utile et le traitement des codages de contenus.

Choisir un type d'intégration d’API API Gateway

Vous choisissez un type d'intégration d'API en fonction des types de point de terminaison d'intégration que vous utilisez et de la façon dont vous souhaitez que vos données soient transmises de et vers le point de terminaison d'intégration. Pour une fonction Lambda, vous pouvez avoir l'intégration de proxy Lambda ou l'intégration personnalisée Lambda. Pour un point de terminaison HTTP, vous pouvez avoir l'intégration de proxy HTTP ou l'intégration personnalisée HTTP. Pour une action de AWS service, vous avez uniquement AWS intégré le type non proxy. API Gateway prend également en charge l'intégration fictive, où API Gateway sert de point de terminaison d'intégration pour répondre à une demande de méthode.

L'intégration personnalisée Lambda est un cas particulier d'intégration, où le point de terminaison AWS d'intégration correspond à l'action d'appel de fonction du service Lambda.

Par programmation, vous choisissez un type d'intégration en définissant la propriété `type` de la ressource `Integration`. Pour l'intégration de proxy Lambda, la valeur est `AWS_PROXY`.

Pour l'intégration personnalisée Lambda et toutes les autres intégrations AWS , c'est `AWS`. Pour l'intégration de proxy HTTP et l'intégration HTTP, la valeur est `HTTP_PROXY` et `HTTP`, respectivement. Pour l'intégration fictive, la valeur `type` est `MOCK`.

L'intégration de proxy Lambda prend en charge une configuration d'intégration simplifiée avec une seule fonction Lambda. Cette configuration est simple et peut évoluer avec le backend sans besoin de manipuler la configuration existante. Pour ces raisons, il est vivement recommandé d'effectuer l'intégration avec une fonction Lambda.

En revanche, l'intégration personnalisée Lambda permet de réutiliser les modèles de mappage configurés pour divers points de terminaison d'intégration qui ont des exigences similaires de formats de données d'entrée et de sortie. La configuration est plus complexe ; elle est recommandée pour des scénarios d'application plus avancés.

De même, l'intégration de proxy HTTP dispose d'une configuration d'intégration simplifiée et peut évoluer avec le backend sans besoin de manipuler la configuration existante. L'intégration personnalisée HTTP est plus complexe à configurer, mais elle permet de réutiliser les modèles de mappage configurés pour d'autres points de terminaison d'intégration.

La liste suivante récapitule les types d'intégration pris en charge :

- `AWS` : ce type d'intégration permet à une API d'exposer les actions de service AWS . Dans l'intégration `AWS`, vous devez configurer à la fois la demande d'intégration et la réponse d'intégration et configurer les mappages de données nécessaires de la demande de méthode à la demande d'intégration et de la réponse d'intégration à la réponse de méthode.
- `AWS_PROXY` : Ce type d'intégration permet d'intégrer une méthode d'API avec l'action d'appel de fonction Lambda grâce à une configuration d'intégration simplifiée, souple et polyvalente. Cette intégration s'appuie sur les interactions directes entre le client et la fonction Lambda intégrée.

  Avec ce type d'intégration, également appelée intégration de proxy Lambda, vous ne configurez pas la demande d'intégration ou la réponse d'intégration. API Gateway transmet la demande entrante du client comme entrée à la fonction Lambda de backend. La fonction Lambda intégrée prend l'entrée de ce format et analyse l'entrée de tous les sources disponibles, y compris les en-têtes de demande, les variables d'URL de chemin, les paramètres des chaînes de requête et le corps correspondants. La fonction renvoie le résultat respectant ce format de sortie.

  Il s'agit du type d'intégration préféré pour appeler une fonction Lambda via API Gateway. Il ne s'applique à aucune autre action de AWS service, y compris aux actions Lambda autres que l'action d'appel de fonction.

- `HTTP` : ce type d'intégration permet à une API d'exposer les points de terminaison HTTP du backend. Avec l'intégration `HTTP`, également appelée intégration personnalisée HTTP, vous devez configurez pas la demande d'intégration et la réponse d'intégration. Vous devez configurer les mappages de données nécessaires de la demande de méthode à la demande d'intégration et de la réponse d'intégration à la réponse de méthode.
- `HTTP_PROXY`: L'intégration de proxy HTTP permet à un client d'accéder aux points de terminaison HTTP du backend avec une configuration d'intégration rationalisée sur une seule méthode d'API. Vous ne définissez pas la demande d'intégration ou la réponse d'intégration. API Gateway transmet la demande entrante du client vers le point de terminaison HTTP et transmet la réponse sortante du point de terminaison HTTP au client.
- `MOCK` : ce type d'intégration permet à API Gateway de renvoyer une réponse sans envoyer la demande au backend. Cela est utile pour le test d'API, car le code peut être utilisé pour tester la configuration de l'intégration sans encourir des frais pour l'utilisation du backend et pour permettre le développement collaboratif d'une API.

  Dans le cadre du développement collaboratif, une équipe peut isoler ses efforts de développement en configurant des simulations de composants d'API détenus par d'autres équipes en utilisant les intégrations `MOCK`. Cela permet également de renvoyer les en-têtes CORS et de s'assurer que la méthode d'API autorise l'accès CORS. De fait, la console API Gateway intègre la méthode `OPTIONS` pour prendre en charge CORS avec une intégration fictive. Les réponses de passerelle sont d'autres exemples d'intégrations fictives.

Configuration de l'intégration de proxy avec une ressource de proxy

Pour configurer une intégration de proxy dans une API API Gateway avec une ressource de proxy, exécutez les tâches suivantes :

- Créer une ressource de proxy avec une variable de chemin gourmande `{proxy+}`.
- Définir la méthode `ANY` sur la ressource de proxy.
- Intégrer la ressource et la méthode à un backend à l'aide du type d'intégration HTTP ou Lambda.

> **Note**
>
> Les variables de chemin gourmandes en ressources, les méthodes `ANY` et les types d'intégration de proxy sont des fonctions indépendantes, même si elles sont couramment

utilisées ensemble. Vous pouvez configurer une méthode spécifique HTTP sur une ressource gourmande ou appliquer des types d'intégration différente de proxy à une ressource de proxy.

API Gateway adopte certaines restrictions et limitations lors de la gestion des méthodes avec une intégration de proxy Lambda ou une intégration de proxy HTTP. Pour plus de détails, consultez the section called “Remarques importantes”.

> **Note**
>
> Lorsque vous utilisez l'intégration de proxy avec un comportement relais, API Gateway renvoie l'en-tête `Content-Type:application/json` par défaut si le type de contenu d'une charge utile n'est pas spécifié.

Une ressource de proxy est plus puissante lorsqu'elle est intégrée à un backend à l'aide de l'intégration de proxy HTTP ou de l'intégration de proxy Lambda.

Intégration de proxy HTTP avec une ressource de proxy

L'intégration de proxy HTTP, désignée par `HTTP_PROXY` dans l'API REST API Gateway, est utilisée pour l'intégration d'une demande de méthode avec un point de terminaison HTTP du backend. Avec ce type d'intégration, API Gateway passe simplement l'intégralité de la demande et de la réponse entre le frontend et le backend, avec certaines restrictions et limitations.

> **Note**
>
> L'intégration de proxy HTTP prend en charge les en-têtes à valeurs multiples et les chaînes de requête.

Lors de l'application de l'intégration de proxy HTTP à une ressource du proxy, vous pouvez configurer votre API de façon à exposer tout ou partie de la hiérarchie des points de terminaison du backend HTTP avec une configuration d'intégration unique. Par exemple, supposons que le site Web de votre backend est organisé en plusieurs branches de nœuds d'arborescence éloignées du nœud racine (`/site`) comme : $/site/a_0/a_1/.../a_N$, $/site/b_0/b_1/.../b_M$, etc. Si vous intégrez la méthode `ANY` sur une ressource de proxy `/api/{proxy+}` aux points de terminaison du backend avec des chemins d'URL `/site/{proxy}`, une demande d'intégration unique peut prendre

en charge n'importe quelle opération HTTP (GET, POST, etc.) sur [a_0, a_1, ..., a_N, b_0, b_1, ...b_M, ...]. Si vous appliquez une intégration de proxy à une méthode HTTP spécifique, par exemple, GET, à la place, la demande d'intégration qui en résulte fonctionne avec les opérations (par exemple, GET) spécifiées sur n'importe lequel de ces nœuds du backend.

Intégration de proxy Lambda avec une ressource de proxy

L'intégration de proxy Lambda, désignée par AWS_PROXY dans l'API REST API Gateway, est utilisée pour l'intégration d'une demande de méthode avec une fonction Lambda sur le backend. Avec ce type d'intégration, API Gateway applique un modèle de mappage par défaut pour envoyer l'ensemble de la demande à la fonction Lambda et transforme la sortie de la fonction Lambda en réponses HTTP.

De même, vous pouvez appliquer l'intégration de proxy Lambda à une ressource de proxy /api/{proxy+} afin de configurer une intégration unique de façon pour qu'une fonction Lambda du backend réagisse individuellement aux modifications des ressources d'API sous /api.

Configuration d'une demande d'intégration d'API à l'aide de la console API Gateway

La configuration d'une méthode d'API définit la méthode et décrit ses comportements. Pour configurer une méthode, vous devez spécifier une ressource, y compris la racine (« / »), sur laquelle cette méthode est exposée, une méthode HTTP (GET, POST, etc.) et son mode d'intégration au backend ciblé. La demande et la réponse de méthode spécifient le contrat avec l'application appelante, en stipulant les paramètres que l'API peut recevoir et à quoi ressemble la réponse.

Les procédures suivantes décrivent comment utiliser la console API Gateway pour créer une demande d'intégration.

Rubriques

- Configurer une intégration Lambda
- Configuration d'une intégration HTTP
- Configuration d'une intégration AWS de services
- Configurer une intégration fictive

Configurer une intégration Lambda

Utilisez une intégration de fonction Lambda pour intégrer votre API à une fonction Lambda. Au niveau de l'API, il s'agit d'un type d'intégration AWS si vous créez une intégration sans proxy, ou d'un type d'intégration AWS_PROXY si vous créez une intégration proxy.

Pour configurer une intégration Lambda

1. Dans le volet Ressources, choisissez Créer une méthode.
2. Pour Type de méthode, sélectionnez une méthode HTTP.
3. Sous Integration type (Type d'intégration), choisissez Lambda function (Fonction Lambda).
4. Pour utiliser une intégration proxy Lambda, activez Intégration proxy Lambda. Pour en savoir plus sur les intégrations proxy Lambda, consultez the section called “ Présentation de l'intégration de proxy Lambda ”.
5. Pour Fonction Lambda, entrez le nom de la Fonction Lambda.

   Si vous utilisez une Fonction Lambda dans une région différente de celle de votre API, sélectionnez la région dans le menu déroulant et entrez le nom de la Fonction Lambda. Si vous utilisez une Fonction Lambda entre comptes, entrez l'ARN de la fonction.
6. Pour utiliser la valeur de délai d'expiration par défaut de 29 secondes, gardez Délai d'expiration activé. Pour définir un délai d'expiration personnalisé, choisissez Délai d'expiration et entrez une valeur de délai d'expiration comprise entre `50` et `29000` millisecondes.
7. (Facultatif) Vous pouvez configurer les paramètres de demande de méthode à l'aide des menus déroulants suivants. Choisissez les paramètres de demande de méthode et configurez votre demande de méthode. Pour plus d'informations, reportez-vous à l'étape 3 dethe section called “Modifier une demande de méthode dans la console”.

   Vous pouvez également configurer les paramètres de votre demande de méthode après avoir créé votre méthode.
8. Choisissez Créer une méthode.

Configuration d'une intégration HTTP

Utilisez une intégration HTTP pour intégrer votre API à un point de terminaison HTTP. Au niveau API, il s'agit du type d'intégration `HTTP`.

Pour configurer une intégration HTTP

1. Dans le volet Ressources, choisissez Créer une méthode.
2. Pour Type de méthode, sélectionnez une méthode HTTP.
3. Pour Type d'intégration, choisissez HTTP.

4. Pour utiliser une intégration proxy HTTP, activez Intégration proxy HTTP. Pour en savoir plus sur les intégrations proxy HTTP, consultez the section called “Configuration des intégrations de proxy HTTP dans API Gateway”.
5. Dans le champ HTTP method, sélectionnez le type de méthode HTTP qui correspond le mieux au service backend HTTP.
6. Pour URL de point de terminaison, entrez l'URL du service backend HTTP que cette méthode doit utiliser.
7. Pour Gestion du contenu, sélectionnez un comportement de gestion du contenu.
8. Pour utiliser la valeur de délai d'expiration par défaut de 29 secondes, gardez Délai d'expiration activé. Pour définir un délai d'expiration personnalisé, choisissez Délai d'expiration et entrez une valeur de délai d'expiration comprise entre `50` et `29000` millisecondes.
9. (Facultatif) Vous pouvez configurer les paramètres de demande de méthode à l'aide des menus déroulants suivants. Choisissez les paramètres de demande de méthode et configurez votre demande de méthode. Pour plus d'informations, reportez-vous à l'étape 3 dethe section called “Modifier une demande de méthode dans la console”.

   Vous pouvez également configurer les paramètres de votre demande de méthode après avoir créé votre méthode.
10. Choisissez Créer une méthode.

Configuration d'une intégration AWS de services

Utilisez une intégration AWS de service pour intégrer votre API directement à un AWS service. Au niveau API, il s'agit du type d'intégration `AWS`.

Pour configurer une API Gateway, effectuez l'une des opérations suivantes :

- Créez une fonction Lambda.
- Définissez une autorisation de ressource sur la Fonction Lambda.
- Effectuez toute autre action de service Lambda.

Vous devez choisir Service AWS .

Pour configurer une intégration AWS de services

1. Dans le volet Ressources, choisissez Créer une méthode.

2. Pour Type de méthode, sélectionnez une méthode HTTP.
3. Dans Type d'intégration, sélectionnez AWS service.
4. Pour AWS Région, choisissez la AWS région que vous souhaitez que cette méthode utilise pour appeler l'action.
5. Pour le AWS service, choisissez le AWS service que vous souhaitez appeler par cette méthode.
6. Pour AWS sous-domaine, entrez le sous-domaine utilisé par le AWS service. En règle générale, vous laisserez ce champ vide. Certains services AWS peuvent prendre en charge les sous-domaines comme partie des hôtes. Consultez la documentation du service pour savoir si cela possible et, le cas échéant, obtenir davantage d'informations.
7. Dans le champ HTTP method, sélectionnez le type de méthode HTTP qui correspond à l'action. Pour le type de méthode HTTP, consultez la documentation de référence de l'API pour le AWS service que vous avez choisi pour le AWS service.
8. Pour Type d'action, sélectionnez Utiliser un nom d'action pour utiliser une action d'API ou Utiliser un remplacement de chemin pour utiliser un chemin de ressource personnalisé. Pour connaître les actions disponibles et les chemins de ressources personnalisés, consultez la documentation de référence de l'API pour le AWS service que vous avez choisi pour le AWS service.
9. Entrez un Nom d'action ou un Remplacement de chemin.
10. Pour Rôle d'exécution, entrez l'ARN du rôle IAM que la méthode va utiliser pour appeler l'action.

    Pour créer un rôle IAM, vous pouvez adapter les instructions dans the section called “Étape 1 : créer le rôle d'exécution du proxy de AWS service”. Spécifiez une stratégie d'accès au format suivant, avec le nombre d'actions et de déclarations de ressources souhaité :

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

```
}
```

Pour connaître la syntaxe des instructions d'action et de ressource, consultez la documentation du AWS service que vous avez choisi pour le AWS service.

Pour la relation d'approbation du rôle IAM, spécifiez la déclaration suivante qui autorise API Gateway à effectuer des actions au nom de votre compte AWS :

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

11. Pour utiliser la valeur de délai d'expiration par défaut de 29 secondes, gardez Délai d'expiration activé. Pour définir un délai d'expiration personnalisé, choisissez Délai d'expiration et entrez une valeur de délai d'expiration comprise entre `50` et `29000` millisecondes.
12. (Facultatif) Vous pouvez configurer les paramètres de demande de méthode à l'aide des menus déroulants suivants. Choisissez les paramètres de demande de méthode et configurez votre demande de méthode. Pour plus d'informations, reportez-vous à l'étape 3 dethe section called “Modifier une demande de méthode dans la console”.

    Vous pouvez également configurer les paramètres de votre demande de méthode après avoir créé votre méthode.
13. Choisissez Créer une méthode.

Configurer une intégration fictive

Utilisez une intégration fictive si vous souhaitez qu'API Gateway fasse office de backend pour renvoyer des réponses statiques. Au niveau API, il s'agit du type d'intégration `MOCK`. Généralement, vous pouvez utiliser l'intégration `MOCK` si votre API n'est pas encore finalisée, mais que vous devez

générer des réponses d'API pour débloquer les équipes dépendantes et réaliser les tests. Pour la méthode `OPTION`, API Gateway définit l'intégration `MOCK` par défaut afin de renvoyer des en-têtes CORS pour la ressource d'API appliquée.

Pour configurer une intégration fictive

1. Dans le volet Ressources, choisissez Créer une méthode.
2. Pour Type de méthode, sélectionnez une méthode HTTP.
3. Pour le type d'intégration, choisissez Mock.
4. (Facultatif) Vous pouvez configurer les paramètres de demande de méthode à l'aide des menus déroulants suivants. Choisissez les paramètres de demande de méthode et configurez votre demande de méthode. Pour plus d'informations, reportez-vous à l'étape 3 dethe section called “Modifier une demande de méthode dans la console”.

   Vous pouvez également configurer les paramètres de votre demande de méthode après avoir créé votre méthode.
5. Choisissez Créer une méthode.

Configuration d'une réponse d'intégration dans API Gateway

Dans le cas d'une intégration autre que de proxy, vous devez configurer au moins une réponse d'intégration, et en faire la réponse par défaut, pour transmettre le résultat renvoyé par le backend au client. Vous pouvez choisir de transmettre le résultat en l'état ou de transformer les données de réponse d'intégration aux données de réponse de la méthode si les deux ont des formats différents.

Dans le cas d'une intégration de proxy, API Gateway transmet automatiquement la sortie du backend au client sous la forme d'une réponse HTTP. Vous ne configurez pas soit une réponse d'intégration soit une réponse de méthode.

Pour configurer une réponse d'intégration, effectuez les tâches facultatives et obligatoires suivantes :

1. Spécifiez un code de statut HTTP d'une méthode de réponse à laquelle les données de réponse d'intégration sont mappées. C'est obligatoire.
2. Définissez une expression régulière pour sélectionner la sortie du serveur principal à représenter par cette réponse d'intégration. Si vous laissez ce champ vide, la réponse est la réponse par défaut utilisée pour recueillir les réponses qui n'ont pas encore été configurées.

3. Le cas échéant, déclarez des mappages composés de paires clé/valeur pour mapper des paramètres de réponse d'intégration spécifiés à des paramètres de réponse de méthode donnés.
4. Le cas échéant, ajoutez des modèles de mappage de corps pour transformer les charges utiles de réponse d'intégration données en charges utiles de réponse de méthode spécifiées.
5. Le cas échéant, spécifiez la manière de gérer la conversion de type pour une charge utile binaire.

Une réponse d'intégration est une réponse HTTP encapsulant la réponse du backend. Pour un point de terminaison HTTP, la réponse du backend est une réponse HTTP. Le code de statut de réponse d'intégration peut prendre le code de statut renvoyé par le backend, et le corps de réponse d'intégration est la charge utile renvoyée par le backend. Pour un point de terminaison Lambda, la réponse du backend est la sortie renvoyée par la fonction Lambda. Avec l'intégration Lambda, la sortie de fonction Lambda est renvoyée en tant que réponse `200 OK`. La charge utile peut contenir le résultat sous la forme de données JSON, y compris une chaîne JSON ou un objet JSON, ou un message d'erreur sous la forme d'un objet JSON. Vous pouvez attribuer une expression régulière à la propriété selectionPattern pour mapper une réponse d'erreur à une réponse d'erreur HTTP appropriée. Pour de plus amples informations sur la réponse d'erreur de la fonction Lambda, veuillez consulter Gestion des erreurs Lambda dans API Gateway. Avec l'intégration de proxy Lambda, la fonction Lambda doit renvoyer une sortie au format suivant :

```
{
    statusCode: "...",            // a valid HTTP status code
    headers: {
        custom-header: "..."      // any API-specific custom header
    },
    body: "...",                  // a JSON string.
    isBase64Encoded:  true|false  // for binary support
}
```

Il n'y a pas besoin de mapper la réponse de la fonction Lambda à la réponse HTTP adéquate.

Pour renvoyer le résultat au client, configurez la réponse d'intégration pour transmettre la réponse du point de terminaison en l'état à la réponse de méthode correspondante. Vous pouvez également mapper les données de réponse du point de terminaison aux données de réponse de la méthode. Les données de réponse qui peuvent être mappées comprennent le code de statut de réponse, les paramètres d'en-tête de réponse et le corps de réponse. Si aucune réponse de méthode n'est définie pour le code de statut renvoyé, API Gateway renvoie une erreur 500. Pour plus d'informations, voir Utiliser un modèle de mappage pour substituer les codes de statut et paramètres des requêtes et réponses d'une API.

Configuration d'intégrations Lambda dans API Gateway

Vous pouvez intégrer une méthode API avec une fonction Lambda en utilisant l'intégration de proxy Lambda ou l'intégration autre que proxy (personnalisée) Lambda.

Avec l'intégration de proxy Lambda, la configuration requise est simple. Définissez la méthode HTTP de l'intégration sur POST, l'URI du point de terminaison de l'intégration sur l'ARN de l'action d'appel de la fonction Lambda d'une fonction Lambda spécifique, et autorisez API Gateway à appeler la fonction Lambda en votre nom.

Avec l'intégration autre que de proxy Lambda, outre la configuration de l'intégration de proxy, vous pouvez spécifier la façon dont les données des demandes entrantes sont mappées à la demande d'intégration et la façon dont les données de réponse d'intégration résultantes sont mappées à la réponse de méthode.

Rubriques

Configuration d'intégrations de proxy Lambda dans API Gateway

Rubriques

Présentation de l'intégration de proxy Lambda API Gateway

L'intégration de proxy Lambda Amazon API Gateway est un mécanisme simple, puissant et agile pour créer une API avec la configuration d'une seule méthode d'API. L'intégration de proxy

Lambda permet au client d'appeler une seule fonction Lambda du backend. La fonction accède à de nombreuses ressources ou fonctionnalités d'autres AWS services, notamment en appelant d'autres fonctions Lambda.

Dans l'intégration de proxy Lambda, quand un client envoie une demande d'API, API Gateway transmet à la fonction Lambda intégrée un objet d'événement, si ce n'est que l'ordre des paramètres de la demande n'est pas préservé. Les données de la demande incluent les en-têtes de demande, les paramètres de chaîne de requête, les variables du chemin de l'URL, la charge utile et les données de configuration de l'API. Les données de configuration peuvent inclure le nom de la phase de déploiement en cours, les variables de la phase, l'identité de l'utilisateur ou le contexte d'autorisation (le cas échéant). La fonction Lambda du backend analyse les données de la demande entrante pour déterminer la réponse qu'elle renvoie. Pour qu'API Gateway transmette la sortie Lambda comme réponse de l'API au client, la fonction Lambda doit renvoyer le résultat dans ce format.

Dans la mesure où API Gateway n'intervient pas beaucoup entre le client et la fonction Lambda du backend pour l'intégration de proxy Lambda, le client et la fonction Lambda intégrée peuvent s'adapter aux modifications de l'autre sans interrompre la configuration de l'intégration existante de l'API. Pour ce faire, le client doit suivre les protocoles d'application édictés par la fonction Lambda du backend.

Vous pouvez configurer une intégration de proxy Lambda pour n'importe quelle méthode d'API. Mais une intégration de proxy Lambda est plus puissante quand elle est configurée pour une méthode d'API impliquant une ressource de proxy générique. La ressource de proxy générique peut être symbolisée par la variable spéciale de chemin modélisé `{proxy+}`, l'espace réservé de la méthode fourre-tout `ANY`, ou les deux. Le client peut transmettre les entrées de la fonction Lambda du backend dans la demande entrante comme paramètres de la demande ou comme charge utile applicable. Les paramètres de la demande comprennent les en-têtes, les variables du chemin de l'URL, les paramètres de la chaîne de requête et la charge utile applicable. La fonction Lambda intégrée vérifie l'ensemble des sources d'entrée avant de traiter la demande et de répondre au client avec des messages d'erreur significatifs si l'une des entrées requises est manquante.

Lors de l'appel d'une méthode d'API intégrée avec la méthode HTTP générique `ANY` et la ressource générique `{proxy+}`, le client soumet une demande avec une méthode HTTP particulière à la place de `ANY`. Le client spécifie également un chemin d'URL particulier au lieu de `{proxy+}` et inclut tous les en-têtes requis, les paramètres de chaîne de requête ou une charge utile applicable.

La liste suivante résume les comportements d'exécution de différentes méthodes d'API avec l'intégration de proxy Lambda :

- `ANY /{proxy+}` : le client doit choisir une méthode HTTP particulière, doit définir une hiérarchie de chemins de ressources spécifiques et peut définir des en-têtes, des paramètres de chaîne de requête et une charge utile applicable pour transmettre les données en tant qu'entrées de la fonction Lambda intégrée.
- `ANY /res` : le client doit choisir une méthode HTTP particulière et peut définir des en-têtes, des paramètres de chaîne de requête et une charge utile applicable pour transmettre les données en tant qu'entrées de la fonction Lambda intégrée.
- `GET|POST|PUT|... /{proxy+}` : le client peut définir une hiérarchie de chemins de ressources spécifiques, des en-têtes, des paramètres de chaîne de requête et une charge utile applicable pour transmettre les données en tant qu'entrées de la fonction Lambda intégrée.
- `GET|POST|PUT|... /res/{path}/...` : le client doit choisir un segment de chemin particulier (pour la variable `{path}`) et peut définir des en-têtes de demande, des paramètres de chaîne de requête et une charge utile applicable pour transmettre les données en entrée à la fonction Lambda intégrée.
- `GET|POST|PUT|... /res` : le client peut choisir des en-têtes de demande, des paramètres de chaîne de requête et une charge utile applicable pour transmettre les données en entrée à la fonction Lambda intégrée.

Dans les deux cas, la ressource de proxy `{proxy+}` et la ressource personnalisée `{custom}` sont exprimées comme des variables de chemin modélisé. Cependant, `{proxy+}` peut faire référence à n'importe quelle ressource de la hiérarchie des chemins, tandis que `{custom}` ne fait référence qu'à un segment de chemin particulier. Par exemple, une épicerie peut organiser son stock de produits en ligne par noms de département, catégories de produit et types de produits. Le site web de l'épicerie peut alors représenter les produits disponibles par les variables de chemin modélisé suivantes des ressources personnalisée: `/{department}/{produce-category}/{product-type}`. Par exemple, les pommes sont représentées par `/produce/fruit/apple` et les carottes par `/produce/vegetables/carrot`. Le site peut également utiliser `/{proxy+}` pour représenter n'importe quel département, catégorie de produit ou type de produit qu'un client peut rechercher lors de ses achats dans la boutique en ligne. Par exemple, `/{proxy+}` peut faire référence à l'un des articles suivants :

- `/produce`
- `/produce/fruit`
- `/produce/vegetables/carrot`

Pour permettre aux clients de rechercher n'importe quel produit disponible, sa catégorie et le département du magasin associé, vous pouvez exposer une seule méthode `GET /{proxy+}` avec les autorisations en lecture seule. De même, pour autoriser un superviseur à mettre à jour le stock du département `produce`, vous pouvez configurer une autre méthode unique `PUT /produce/{proxy+}` avec les autorisations en lecture/écriture. Pour autoriser une caissière pour mettre à jour le total cumulé d'un légume, vous pouvez configurer une méthode `POST /produce/vegetables/{proxy+}` avec les autorisations en lecture/écriture. Pour permettre à un gestionnaire de stockage d'effectuer n'importe quelle action possible sur n'importe quel produit disponible, le développeur de la boutique en ligne peut exposer la méthode `ANY /{proxy+}` avec les autorisations en lecture/écriture. Dans tous les cas, au moment de l'exécution, le client ou l'employé doit sélectionner un produit spécifique d'un type donné dans un département choisi, une catégorie de produits spécifique d'un département donné ou un département spécifique.

Pour de plus amples informations sur la configuration des intégrations de proxy API Gateway, veuillez consulter Configuration de l'intégration de proxy avec une ressource de proxy.

L'intégration de proxy nécessite que le client ait une connaissance plus approfondie des exigences du backend. Par conséquent, pour garantir des performances d'application et une expérience de l'utilisateur optimales, le développeur du backend doit communiquer clairement au développeur du client les exigences du backend et fournir un mécanisme de retour d'erreur robuste lorsque les exigences ne sont pas satisfaites.

Prise en charge des en-têtes à valeurs multiples et des paramètres de chaîne de requête

API Gateway prend désormais en charge les en-têtes multiples et les paramètres de chaîne de requête portant le même nom. Les en-têtes à valeurs multiples et les en-têtes et paramètres à valeur unique peuvent être combinés dans les mêmes demandes et réponses. Pour plus d'informations, consultez Format d'entrée d'une fonction Lambda pour l'intégration proxy et Format de sortie d'une fonction Lambda pour l'intégration proxy.

Configuration d'une ressource de proxy avec l'intégration de proxy Lambda

Pour configurer une ressource de proxy avec le type d'intégration de proxy Lambda, créez une ressource d'API avec un paramètre de chemin gourmand (par exemple, `/parent/{proxy+}`) et intégrez cette ressource à un backend de fonction Lambda (par exemple, `arn:aws:lambda:us-west-2:123456789012:function:SimpleLambda4ProxyResource`) sur la méthode `ANY`. Le paramètre de chemin gourmand doit se trouver à la fin du chemin de ressource d'API. Comme avec une ressource autre que de proxy, vous pouvez configurer la ressource de proxy à l'aide de la

console API Gateway, en important un fichier de définitions OpenAPI ou en appelant l'API REST API Gateway directement.

Le fichier de définitions d'API OpenAPI suivant présente un exemple d'API avec une ressource de proxy intégrée à une fonction Lambda nommée `SimpleLambda4ProxyResource`.

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"
        }
      }
    }
  }
}
```

Avec l'intégration de proxy Lambda, au moment de l'exécution, API Gateway mappe une demande entrante dans le paramètre d'entrée `event` de la fonction Lambda. L'entrée inclut la méthode de demande, le chemin, les en-têtes, les paramètres de chaîne de demande, la charge utile, le contexte associé et les variables d'étape définies. Le format d'entrée est expliqué dans Format d'entrée d'une fonction Lambda pour l'intégration proxy. Pour qu'API Gateway mappe correctement la sortie Lambda aux réponses HTTP, la fonction Lambda doit sortir le résultat au format décrit dans Format de sortie d'une fonction Lambda pour l'intégration proxy.

Avec l'intégration de proxy Lambda d'une ressource de proxy via la méthode `ANY`, la fonction Lambda du backend unique sert de gestionnaire d'événements pour toutes les demandes via la ressource de proxy. Par exemple, pour consigner les modèles de trafic, vous pouvez demander à un appareil mobile d'envoyer ses informations de localisation, notamment l'état, la ville, la rue et le bâtiment en soumettant une demande avec `/state/city/street/house` dans le chemin d'URL de la ressource de proxy. La fonction Lambda du backend peut ensuite analyser le chemin d'URL et insérer les tuples d'emplacement dans une table DynamoDB.

Configurez l'intégration du proxy Lambda à l'aide du AWS CLI

Dans cette section, nous montrons comment AWS CLI configurer une API avec l'intégration du proxy Lambda.

> **Note**
>
> Pour obtenir des instructions détaillées sur l'utilisation de la console API Gateway afin de configurer une ressource de proxy avec l'intégration de proxy Lambda, veuillez consulter Tutoriel : Création d'une API REST Hello World avec l'intégration de proxy Lambda.

À titre d'exemple, nous utilisons l'exemple suivant de fonction Lambda comme backend de l'API :

```
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);
```

```
};
```

Si l'on compare ce code à la configuration de l'intégration personnalisée Lambda, l'entrée pour cette fonction Lambda peut être exprimée dans les paramètres et le corps de la demande. Vous avez plus de latitude pour permettre au client de transmettre les mêmes données d'entrée. Ici, le client peut transmettre le nom de l'hôte sous la forme d'une propriété de paramètre de chaîne de demande, d'en-tête ou de corps. La fonction peut également prendre en charge l'intégration personnalisée Lambda. La configuration de l'API est plus simple. Vous ne configurez pas du tout la réponse de la méthode ou la réponse de l'intégration.

Pour configurer une intégration de proxy Lambda à l'aide du AWS CLI

1. Appelez la commande `create-rest-api` pour créer une API :

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

   Notez la valeur d'`id` (`te6si5ach7`) de l'API résultante dans la réponse :

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

   Vous avez besoin de l'`id` d'API tout au long de cette section.
2. Appelez la commande `get-resources` pour obtenir l'`id` de la ressource racine :

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

   La réponse positive est affichée comme suit :

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

Notez la valeur d'`id` de la ressource racine (`krznpq9xpg`). Vous en aurez besoin à l'étape suivante et ultérieurement.

3. Appelez `create-resource` pour créer une ressource API Gateway `/greeting` :

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

La réponse positive est semblable à ce qui suit :

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

Notez la valeur d'`id` (`2jf6xt`) de la ressource `{proxy+}` résultante. Vous en avez besoin pour créer une méthode dans la ressource `/{proxy+}` à l'étape suivante.

4. Appelez `put-method` pour créer une demande de méthode `ANY` de `ANY /{proxy+}` :

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

La réponse positive est semblable à ce qui suit :

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

Cette méthode d'API permet au client de recevoir ou d'envoyer des salutations de la fonction Lambda au backend.

5. Appelez `put-integration` pour configurer l'intégration de la méthode `ANY /{proxy+}` avec une fonction Lambda nommée `HelloWorld`. Cette fonction répond à la demande par un message `"Hello, {name}!"` si le paramètre `greeter` est précisé, ou `"Hello, World!"` si le paramètre de chaîne de demande n'est pas défini.

```
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**
>
> Pour les intégrations Lambda, vous devez utiliser la méthode HTTP `POST` pour la demande d'intégration, conformément à la spécification de l'action de service Lambda pour les appels de fonction. Le rôle IAM `apigAwsProxyRole` doit avoir des stratégies permettant au service `apigateway` d'appeler des fonctions Lambda. Pour plus d'informations sur les autorisations IAM, consultez the section called “ Modèle d'autorisation API Gateway pour l'appel d'une API”.

La sortie positive est semblable à ce qui suit :

```
{
    "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"
}
```

Plutôt que de préciser un rôle IAM pour `credentials`, vous pouvez appeler la commande add-permission pour ajouter des autorisations basées sur les ressources. C'est ce que fait la console API Gateway.

6. Appelez `create-deployment` pour déployer l'API dans une étape de `test` :

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

7. Testez l'API à l'aide des commandes cURL suivantes dans un terminal.

   Appel de l'API avec le paramètre de chaîne de demande `?greeter=jane` :

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

   Appel de l'API avec un paramètre d'en-tête `greeter:jane` :

```
curl -X GET https://te6si5ach7.execute-api.us-west-2.amazonaws.com/test/hi \
  -H 'content-type: application/json' \
  -H 'greeter: jane'
```

   Appel de l'API avec un corps `{"greeter":"jane"}` :

```
curl -X POST https://te6si5ach7.execute-api.us-west-2.amazonaws.com/test/hi \
  -H 'content-type: application/json' \
  -d '{ "greeter": "jane" }'
```

   Dans tous les cas, la sortie est une réponse 200 avec le corps de réponse suivant :

```
Hello, jane!
```

Format d'entrée d'une fonction Lambda pour l'intégration proxy

Avec l'intégration de proxy Lambda, API Gateway mappe l'intégralité de la demande du client avec le paramètre `event` en entrée de la fonction Lambda du backend. L'exemple suivant montre la structure d'un événement qu'API Gateway envoie à une intégration de proxy Lambda.

```
{
```

```
  "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

Dans l'entrée :

- La clé `headers` ne peut contenir que des en-têtes à valeur unique.
- La clé `multiValueHeaders` peut contenir des en-têtes à valeurs multiples, ainsi que des en-têtes à valeur unique.

- Si vous spécifiez des valeurs pour `headers` et `multiValueHeaders`, API Gateway les fusionne en une seule liste. Si la même paire clé-valeur est spécifiée dans les deux, seules les valeurs de `multiValueHeaders` s'afficheront dans la liste fusionnée.

Dans l'entrée de la fonction Lambda du backend, l'objet `requestContext` est une association de paires clé/valeur. Dans chaque paire, la clé est le nom d'une propriété de la variable $context et la valeur est la valeur de cette propriété. API Gateway peut ajouter de nouvelles clés à la carte.

D'après les fonctions activées, le mappage `requestContext` peut varier d'une API à l'autre. Ainsi, dans l'exemple précédent, puisqu'aucun type d'autorisation n'est spécifié, il n'y a aucune propriété `$context.authorizer.*` ou `$context.identity.*` présente. Lorsqu'un type d'autorisation est spécifié, API Gateway transmet les informations sur l'utilisateur autorisé au point de terminaison d'intégration dans un objet `requestContext.identity`, comme suit :

- Lorsque le type d'autorisation est `AWS_IAM`, les informations sur l'utilisateur autorisé comprennent des propriétés `$context.identity.*`.
- Lorsque le type d'autorisation est `COGNITO_USER_POOLS` (mécanisme d'autorisation Amazon Cognito), les informations sur l'utilisateur autorisé comprennent les propriétés `$context.identity.cognito*` et `$context.authorizer.claims.*`.
- Lorsque le type d'autorisation est `CUSTOM` (mécanisme d'autorisation Lambda), les informations sur l'utilisateur autorisé comprennent les propriétés `$context.authorizer.principalId`, ainsi que d'autres propriétés `$context.authorizer.*` applicables.

Format de sortie d'une fonction Lambda pour l'intégration proxy

Avec l'intégration de proxy Lambda, API Gateway demande à la fonction Lambda du backend de renvoyer la sortie selon le format JSON suivant :

```
{
    "isBase64Encoded": true|false,
    "statusCode": httpStatusCode,
    "headers": { "headerName": "headerValue", ... },
    "multiValueHeaders": { "headerName": ["headerValue", "headerValue2", ...], ... },
    "body": "..."
}
```

Dans la sortie :

- Les clés `headers` et`multiValueHeaders` peuvent être non précisées si aucun en-tête de réponse supplémentaire ne doit être renvoyé.
- La clé `headers` ne peut contenir que des en-têtes à valeur unique.
- La clé `multiValueHeaders` peut contenir des en-têtes à valeurs multiples, ainsi que des en-têtes à valeur unique. Vous pouvez utiliser la clé `multiValueHeaders` pour spécifier l'ensemble de vos en-têtes supplémentaires, y compris n'importe quel en-tête à valeur unique.
- Si vous spécifiez des valeurs pour `headers` et `multiValueHeaders`, API Gateway les fusionne en une seule liste. Si la même paire clé-valeur est spécifiée dans les deux, seules les valeurs de `multiValueHeaders` s'afficheront dans la liste fusionnée.

Pour activer CORS pour l'intégration de proxy Lambda, vous devez ajouter `Access-Control-Allow-Origin:`*`domain-name`* à la sortie `headers`. *`domain-name`* peut être `*` pour n'importe quel nom de domaine. La sortie `body` est regroupée côté frontal en tant que charge utile de réponse de méthode. Si `body` est un blob binaire, vous pouvez l'encoder en tant que chaîne codée en Base64 en définissant `isBase64Encoded` sur `true` et en configurant `*/*` comme un type de fichier multimédia binaire. Sinon, vous pouvez le définir sur `false` ou le laisser non spécifié.

> **Note**
>
> Pour plus d'informations sur l'activation de la prise en charge des fichiers binaires, voir Activation de la prise en charge binaire à l'aide de la console API Gateway. Pour obtenir un exemple de fonction Lambda, veuillez consulter Renvoi d'un support binaire à partir d'une intégration de proxy Lambda.

Si le format de la sortie de la fonction est différent, API Gateway renvoie une réponse d'erreur `502 Bad Gateway`.

Pour renvoyer une réponse dans une fonction Lambda dans Node.js, vous pouvez utiliser des commandes telles que les suivantes :

- Pour renvoyer un résultat positif, appelez `callback(null, {"statusCode": 200, "body": "results"})`.
- Pour générer une exception, appelez `callback(new Error('internal server error'))`.

- Pour une erreur côté client (par exemple, si un paramètre obligatoire est manquant), vous pouvez appeler `callback(null, {"statusCode": 400, "body": "Missing parameters of ..."})` pour renvoyer l'erreur sans générer d'exception.

Dans une fonction `async` Lambda dans Node.js, la syntaxe équivalente serait la suivante :

- Pour renvoyer un résultat positif, appelez `return {"statusCode": 200, "body": "results"}`.
- Pour générer une exception, appelez `throw new Error("internal server error")`.
- Pour une erreur côté client (par exemple, si un paramètre obligatoire est manquant), vous pouvez appeler `return {"statusCode": 400, "body": "Missing parameters of ..."}` pour renvoyer l'erreur sans générer d'exception.

Configuration d'intégrations personnalisées Lambda dans API Gateway

Pour vous montrer comment configurer l'intégration personnalisée Lambda, nous créons une API API Gateway pour exposer la méthode `GET /greeting?greeter={name}` pour appeler une fonction Lambda. Utilisez l’un des exemples de fonctions Lambda suivants pour votre API.

Utilisez l’un des exemples de fonctions Lambda suivants :

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
```

La fonction répond avec un message `"Hello, {name}!"` si la valeur du paramètre `greeter` est une chaîne non vide. Elle renvoie un message `"Hello, World!"` si la valeur `greeter` est une chaîne vide. La fonction renvoie un message d'erreur `"Missing the required greeter parameter."` si le paramètre d'hôte n'est pas défini dans la demande entrante. Nous nommons la fonction `HelloWorld`.

Vous pouvez la créer dans la console Lambda ou à l'aide de la AWS CLI. Dans cette section, nous faisons référence à cette fonction avec l'ARN suivant :

```
arn:aws:lambda:us-east-1:123456789012:function:HelloWorld
```

Avec la fonction Lambda définie dans le backend, commencez à configurer l'API.

Pour configurer l'intégration personnalisée Lambda à l'aide du AWS CLI

1. Appelez la commande `create-rest-api` pour créer une API :

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

Notez la valeur d'`id` (`te6si5ach7`) de l'API résultante dans la réponse :

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

Vous avez besoin de l'`id` d'API tout au long de cette section.

2. Appelez la commande `get-resources` pour obtenir l'`id` de la ressource racine :

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

La réponse positive est la suivante :

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

Notez la valeur d'`id` de la ressource racine (`krznpq9xpg`). Vous en aurez besoin à l'étape suivante et ultérieurement.

3. Appelez `create-resource` pour créer une ressource API Gateway `/greeting` :

```
aws apigateway create-resource --rest-api-id te6si5ach7 \
      --region us-west-2 \
      --parent-id krznpq9xpg \
      --path-part greeting
```

La réponse positive est semblable à ce qui suit :

```
{
```

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

Notez la valeur d'`id` (`2jf6xt`) de la ressource `greeting` résultante. Vous en avez besoin pour créer une méthode dans la ressource `/greeting` à l'étape suivante.

4. Appelez `put-method` pour créer une demande de méthode d'API `GET /greeting?greeter={name}` :

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

La réponse positive est semblable à ce qui suit :

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

Cette méthode d'API permet au client de recevoir des salutations de la fonction Lambda au backend. Le paramètre `greeter` est facultatif, car le backend doit gérer soit un appelant anonyme, soit un appelant auto-identifié.

5. Appelez `put-method-response` pour configurer la réponse `200 OK` à la demande de méthode `GET /greeting?greeter={name}` :

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

```
        --status-code 200
```

6. Appelez `put-integration` pour configurer l'intégration de la méthode `GET /greeting?greeter={name}` avec une fonction Lambda nommée `HelloWorld`. La fonction répond à la demande par un message `"Hello, {name}!"` si le paramètre `greeter` est précisé, ou `"Hello, World!"` si le paramètre de chaîne de demande n'est pas défini.

```
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
```

Le modèle de mappage fourni ici convertit le paramètre de chaîne de demande `greeter` en propriété `greeter` de la charge utile JSON. Ceci est nécessaire parce que l'entrée d'une fonction Lambda doit être exprimée dans le corps.

> **Important**
>
> Pour les intégrations Lambda, vous devez utiliser la méthode HTTP `POST` pour la demande d'intégration, conformément à la spécification de l'action de service Lambda pour les appels de fonction. Le paramètre `uri` est l'ARN de l'action d'appel de fonction. La sortie est similaire à ce qui suit :

```
{
    "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"
}
```

Le rôle IAM `apigAwsProxyRole` doit avoir des stratégies qui permettent au service `apigateway` d'appeler des fonctions Lambda. Plutôt que de préciser un rôle IAM pour `credentials`, vous pouvez appeler la commande add-permission pour ajouter des autorisations basées sur les ressources. C'est ainsi que la console API Gateway ajoute ces autorisations.

7. Appelez `put-integration-response` pour configurer la réponse d'intégration afin de transmettre la sortie de la fonction Lambda au client comme réponse de méthode `200 OK`.

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

En définissant le modèle de sélection sur une chaîne vide, la réponse `200 OK` est la valeur par défaut.

La réponse positive doit être semblable à ce qui suit :

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

8. Appelez `create-deployment` pour déployer l'API dans une étape de `test` :

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

9. Testez l'API à l'aide de la commande cURL suivante dans un terminal :

```
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'
```

Configuration d'un appel asynchrone de la fonction Lambda du backend

Dans l'intégration autre que de proxy (personnalisée) Lambda, la fonction Lambda du backend est, par défaut, invoquée de façon synchrone. Il s'agit du comportement souhaité pour la plupart des opérations d'API REST. Dans certaines applications, toutefois, les appels doivent avoir lieu de façon asynchrone (sous la forme d'une opération par lots ou d'une opération à latence longue), généralement par un autre composant du backend. Dans ce cas, la fonction Lambda du backend est invoquée de façon asynchrone et la méthode d'API REST frontale ne renvoie aucun résultat.

Vous pouvez également configurer la fonction Lambda pour une intégration autre que de proxy Lambda de sorte qu'elle soit invoquée de façon asynchrone en spécifiant `'Event'` en tant que type d'appel. La procédure à suivre est expliquée ci-après :

Configuration de l'appel asynchrone Lambda dans la console API Gateway

Pour que tous les appels soient asynchrones :

- Dans Requête d’intégration, ajoutez un en-tête `X-Amz-Invocation-Type` avec une valeur statique de `'Event'`.

Pour que les clients décident si les appels sont asynchrones ou synchrones :

1. Dans Requête de méthode, ajoutez un en-tête `InvocationType`.
2. Dans Requête d’intégration, ajoutez un en-tête `X-Amz-Invocation-Type` avec une expression de mappage de `method.request.header.InvocationType`.
3. Les clients peuvent inclure l'en-tête `InvocationType: Event` dans les requêtes API pour les appels asynchrones, ou `InvocationType: RequestResponse` pour les appels synchrones.

Configuration de l'invocation asynchrone de Lambda à l'aide d'OpenAPI

Pour que tous les appels soient asynchrones :

- Ajoutez l'`X-Amz-Invocation-Type`en-tête à la x-amazon-apigateway-integrationsection.

```
"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"
        }
```

Pour que les clients décident si les appels sont asynchrones ou synchrones :

1. Ajoutez l'en-tête suivant sur tout objet d'élément de chemin OpenAPI.

```
"parameters" : [ {
"name" : "InvocationType",
"in" : "header",
"schema" : {
  "type" : "string"
}
} ]
```

2. Ajoutez l'`X-Amz-Invocation-Type`en-tête à la x-amazon-apigateway-integrationsection.

```
"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. Les clients peuvent inclure l'en-tête `InvocationType: Event` dans les requêtes API pour les appels asynchrones, ou `InvocationType: RequestResponse` pour les appels synchrones.

Configurer l'appel asynchrone Lambda à l'aide de AWS CloudFormation

Les AWS CloudFormation modèles suivants montrent comment configurer les `AWS::ApiGateway::Method` appels asynchrones.

Pour que tous les appels soient asynchrones :

```
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'
```

Pour que les clients décident si les appels sont asynchrones ou synchrones :

```
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'
```

Les clients peuvent inclure l'en-tête `InvocationType: Event` dans les requêtes API pour les appels asynchrones, ou `InvocationType: RequestResponse` pour les appels synchrones.

Gestion des erreurs Lambda dans API Gateway

Pour les intégrations personnalisées Lambda, vous devez mapper les erreurs renvoyées par Lambda dans la réponse d'intégration à des réponses d'erreurs HTTP standard pour vos clients. Sinon, les erreurs Lambda sont renvoyées en tant que réponses `200 OK` par défaut et le résultat n'est pas intuitif pour les utilisateurs de l'API.

Lambda peut renvoyer deux types d'erreurs : des erreurs standard et des erreurs personnalisées. Dans votre API, vous devez les gérer différemment.

Avec l'intégration de proxy Lambda, Lambda doit renvoyer une sortie au format suivant :

```
{
  "isBase64Encoded" : "boolean",
  "statusCode": "number",
```

```
  "headers": { ... },
  "body": "JSON string"
}
```

Dans cette sortie, `statusCode` est généralement `4XX` pour une erreur de client et `5XX` pour une erreur de serveur. API Gateway traite ces erreurs en mappant l'erreur Lambda à une réponse d'erreur HTTP, en fonction du code `statusCode` spécifié. Pour qu'API Gateway transmette le type d'erreur, (par exemple `InvalidParameterException`), dans le cadre de la réponse au client, la fonction Lambda doit inclure un en-tête (par exemple, `"X-Amzn-ErrorType":"InvalidParameterException"`) dans la propriété `headers`.

Rubriques

- Gestion des erreurs Lambda standard dans API Gateway
- Gestion des erreurs Lambda personnalisées dans API Gateway

Gestion des erreurs Lambda standard dans API Gateway

Une erreur AWS Lambda standard se présente sous le format suivant :

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

Ici, `errorMessage` est une expression de chaîne de l'erreur. `errorType` est un type d'erreur ou d'exception dépendant du langage. `stackTrace` est une liste d'expressions de chaîne montrant la trace de la pile conduisant à l'apparition de l'erreur.

Prenons l'exemple de la fonction Lambda JavaScript (Node.js) suivante.

```
export const handler = function(event, context, callback) {
    callback(new Error("Malformed input ..."));
};
```

Cette fonction renvoie l'erreur Lambda standard suivante, contenant `Malformed input ...` comme message d'erreur :

```
{
  "errorMessage": "Malformed input ...",
  "errorType": "Error",
  "stackTrace": [
    "export const handler (/var/task/index.js:3:14)"
  ]
}
```

De même, examinez la fonction Lambda Python suivante qui génère une `Exception` avec le même message d'erreur `Malformed input ...`.

```
def lambda_handler(event, context):
    raise Exception('Malformed input ...')
```

Cette fonction renvoie l'erreur Lambda standard suivante :

```
{
  "stackTrace": [
    [
      "/var/task/lambda_function.py",
      3,
      "lambda_handler",
      "raise Exception('Malformed input ...')"
    ]
  ],
  "errorType": "Exception",
  "errorMessage": "Malformed input ..."
}
```

Notez que les valeurs de propriété `errorType` et `stackTrace` sont dépendantes du langage. L'erreur standard s'applique également à n'importe quel objet d'erreur qui est une extension de l'objet `Error` ou une sous-classe de la classe `Exception`.

Pour mapper l'erreur Lambda standard à une réponse de méthode, vous devez d'abord décider d'un code de statut HTTP pour une erreur Lambda donnée. Vous devez ensuite définir un modèle d'expression régulière sur la propriété `selectionPattern` de la réponse d'intégration IntegrationResponse associée au code de statut HTTP donné. Dans la console API Gateway, ce

`selectionPattern` est désigné comme Expressions régulières de l'erreur Lambda dans la section Réponse d'intégration sous chaque réponse d'intégration.

> **Note**
>
> API Gateway utilise des expressions régulières de type modèle Java pour le mappage de réponse. Pour plus d'informations, consultez la section Modèles dans la documentation Oracle.

Par exemple, pour configurer une nouvelle expression `selectionPattern`, à l'aide de l'AWS CLI, appelez la commande put-integration-response suivante :

```
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
```

Veillez également à configurer le code d'erreur correspondant (`400`) dans la réponse de méthode. Sinon, API Gateway renvoie une réponse d'erreur de configuration non valide lors de l'exécution.

> **Note**
>
> Lors de l'exécution, API Gateway met en correspondance le message `errorMessage` de l'erreur Lambda avec le modèle de l'expression régulière sur la propriété `selectionPattern`. S'il existe une correspondance, API Gateway renvoie l'erreur Lambda en tant que réponse HTTP du code de statut HTTP correspondant. S'il n'existe pas de correspondance, API Gateway renvoie l'erreur sous la forme d'une réponse par défaut ou émet une exception de configuration non valide si aucune réponse par défaut n'est configurée.
>
> La définition de la valeur `selectionPattern` sur `.*` pour une réponse donnée revient à réinitialiser cette réponse comme la réponse par défaut. En effet, un tel modèle de sélection correspond à tous les messages d'erreur, y compris null, c'est-à-dire, tout message d'erreur non spécifié. Le mappage résultant remplace le mappage par défaut.

Pour mettre à jour une valeur `selectionPattern` existante à l'aide de l'AWS CLI appelez l'opération integrationresponse:update pour remplacer la valeur de chemin `/selectionPattern` par l'expression regex spécifiée du modèle `Malformed*`.

Pour définir l'expression `selectionPattern` à l'aide de la console API Gateway, saisissez l'expression dans la zone de texte Expressions régulières de l'erreur Lambda lors de la configuration ou de la mise à jour d'une réponse d'intégration d'un code de statut HTTP spécifié.

Gestion des erreurs Lambda personnalisées dans API Gateway

Au lieu de l'erreur standard décrite dans la section précédente, AWS Lambda vous permet de renvoyer un objet d'erreur personnalisé en tant que chaîne JSON. L'erreur peut être n'importe quel objet JSON valide. Par exemple, la fonction Lambda JavaScript (Node.js) renvoie une erreur personnalisée :

```
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));
};
```

Vous devez transformer l'objet `myErrorObj` en chaîne JSON avant d'appeler `callback` pour quitter la fonction. Sinon, `myErrorObj` est renvoyé sous la forme d'une chaîne `"[object Object]"`. Lorsqu'une méthode de votre API est intégrée à la fonction Lambda précédente, API Gateway reçoit une réponse d'intégration avec la charge utile suivante :

```
{
    "errorMessage": "{\"errorType\":\"InternalServerError\",\"httpStatus\":500,
\"requestId\":\"e5849002-39a0-11e7-a419-5bb5807c9fb2\",\"trace\":{\"function\":
\"abc()\",\"line\":123,\"file\":\"abc.js\"}}"
}
```

Comme avec toute réponse d'intégration, vous pouvez transmettre cette réponse d'erreur telle quelle à la réponse de méthode. Vous pouvez aussi avoir un modèle de mappage pour transformer la

charge utile en un autre format. Prenons l'exemple du modèle de mappage de corps suivant pour une réponse de méthode d'un code du statut `500` :

```
{
    errorMessage: $input.path('$.errorMessage');
}
```

Ce modèle convertit le corps de réponse d'intégration qui contient la chaîne JSON d'erreur personnalisée en le corps de réponse de méthode suivant. Ce corps de réponse de méthode contient l'objet JSON d'erreur personnalisée :

```
{
    "errorMessage" : {
        errorType : "InternalServerError",
        httpStatus : 500,
        requestId : context.awsRequestId,
        trace : {
            "function": "abc()",
            "line": 123,
            "file": "abc.js"
        }
    }
};
```

En fonction des exigences de votre API, il se peut que vous ayez besoin de transmettre tout ou partie des propriétés d'erreur personnalisée comme paramètres d'en-tête de réponse de méthode. Vous pouvez le faire en appliquant les mappages d'erreur personnalisée du corps de réponse d'intégration aux en-têtes de réponse de méthode.

Par exemple, l'extension OpenAPI suivante définit un mappage à partir des propriétés `errorMessage.errorType`, `errorMessage.httpStatus`, `errorMessage.trace.function` et `errorMessage.trace` aux en-têtes `error_type`, `error_status`, `error_trace_function` et `error_trace`, respectivement.

```
"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"
          },
          ...
        }
    }
}
```

Lors de l'exécution, API Gateway désérialise le paramètre `integration.response.body` lorsqu'il effectue les mappages d'en-tête. Cependant, cette désérialisation s'applique uniquement aux mappages corps à en-tête pour les réponses d'erreur personnalisée Lambda et ne s'applique pas aux mappages corps à corps à l'aide de `$input.body`. Avec ces mappages corps à en-tête d'erreur personnalisée, le client reçoit les en-têtes suivants dans la réponse de méthode, dans la mesure où les en-têtes `error_status`, `error_trace`, `error_trace_function` et `error_type` sont déclarés dans la demande de méthode.

```
"error_status":"500",
"error_trace":"{\"function\":\"abc()\",\"line\":123,\"file\":\"abc.js\"}",
"error_trace_function":"abc()",
"error_type":"InternalServerError"
```

La propriété `errorMessage.trace` du corps de réponse d'intégration est une propriété complexe. Elle est mappée à l'en-tête `error_trace` sous la forme d'une chaîne JSON.

Configuration des intégrations HTTP dans API Gateway

Vous pouvez intégrer une méthode API à un point de terminaison HTTP grâce à l'intégration de proxy HTTP ou à l'intégration personnalisée HTTP.

API Gateway prend en charge les ports de point de terminaison suivants : 80, 443 et 1024-65535.

Avec l'intégration de proxy, la configuration est simple. Il vous suffit de définir la méthode HTTP et l'URI du point de terminaison HTTP, en fonction des exigences du backend, si vous n'êtes pas concerné par le codage de contenu ou la mise en cache.

Avec l'intégration personnalisée, la configuration est plus complexe. Outre la configuration de l'intégration de proxy, vous devez spécifier la façon dont les données des demandes entrantes

sont mappées à la demande d'intégration et la façon dont les données de réponse d'intégration résultantes sont mappées à la réponse de méthode.

Rubriques

- Configuration des intégrations de proxy HTTP dans API Gateway
- Configuration des intégrations personnalisées HTTP dans API Gateway

Configuration des intégrations de proxy HTTP dans API Gateway

Pour configurer une ressource de proxy avec le type d'intégration de proxy HTTP, créez une ressource d'API avec un paramètre de chemin gourmand (par exemple, `/parent/{proxy+}`) et intégrez cette ressource à un point de terminaison de backend HTTP (par exemple, `https://petstore-demo-endpoint.execute-api.com/petstore/{proxy}`) sur la méthode `ANY`. Le paramètre de chemin gourmand doit se trouver à la fin du chemin de ressource.

Comme avec une ressource autre que de proxy, vous pouvez configurer une ressource de proxy avec l'intégration de proxy HTTP à l'aide de la console API Gateway, en important un fichier de définition OpenAPI ou en appelant directement l'API REST API Gateway. Pour obtenir des instructions détaillées sur l'utilisation de la console API Gateway afin de configurer une ressource de proxy avec l'intégration HTTP, veuillez consulter Tutoriel : Création d'une API REST avec une intégration de proxy HTTP.

Le fichier de définition OpenAPI suivant montre un exemple d'API avec une ressource proxy intégrée au PetStoresite Web.

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"
        }
      }
    }
  }
}
```

Dans cet exemple, une clé de cache est déclarée sur le paramètre de chemin `method.request.path.proxy` de la ressource de proxy. Il s'agit du paramètre par défaut lorsque vous créez l'API à l'aide de la console API Gateway. Le chemin de base de l'API (`/test`correspondant à une étape) est mappé à la PetStore page du site Web (`/petstore`). La demande d'intégration unique reflète l'ensemble du PetStore site Web en utilisant la variable de chemin gourmand de l'API et la méthode ANY fourre-tout. Les exemples suivants illustrent cette mise en miroir.

- Définir **`ANY`** en tant que **`GET`** et **`{proxy+}`** en tant que **`pets`**

  Demande de méthode initiée depuis le serveur frontal :

  ```
  GET https://4z9giyi2c1.execute-api.us-west-2.amazonaws.com/test/pets HTTP/1.1
  ```

  Demande d'intégration envoyée au backend :

  ```
  GET http://petstore-demo-endpoint.execute-api.com/petstore/pets HTTP/1.1
  ```

  Les instances d'exécution de la méthode ANY et de la ressource de proxy sont toutes les deux valides. L'appel renvoie une réponse `200 OK` avec la charge utile contenant le premier lot d'animaux domestiques, telle que retournée depuis le serveur principal.

- Définir **`ANY`** en tant que **`GET`** et **`{proxy+}`** en tant que **`pets?type=dog`**

  ```
  GET https://4z9giyi2c1.execute-api.us-west-2.amazonaws.com/test/pets?type=dog
   HTTP/1.1
  ```

  Demande d'intégration envoyée au backend :

  ```
  GET http://petstore-demo-endpoint.execute-api.com/petstore/pets?type=dog HTTP/1.1
  ```

Les instances d'exécution de la méthode ANY et de la ressource de proxy sont toutes les deux valides. L'appel renvoie une réponse `200 OK` avec la charge utile contenant le premier lot de chiens spécifiés, telle que retournée depuis le serveur principal.

- Définir **`ANY`** en tant que **`GET`** et **`{proxy+}`** en tant que **`pets/{petId}`**

  Demande de méthode initiée depuis le serveur frontal :

  ```
  GET https://4z9giyi2c1.execute-api.us-west-2.amazonaws.com/test/pets/1 HTTP/1.1
  ```

  Demande d'intégration envoyée au backend :

  ```
  GET http://petstore-demo-endpoint.execute-api.com/petstore/pets/1 HTTP/1.1
  ```

  Les instances d'exécution de la méthode ANY et de la ressource de proxy sont toutes les deux valides. L'appel renvoie une réponse `200 OK` avec la charge utile contenant l'animal spécifié, telle que retournée depuis le backend.

- Définir **`ANY`** en tant que **`POST`** et **`{proxy+}`** en tant que **`pets`**

  Demande de méthode initiée depuis le serveur frontal :

  ```
  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
  }
  ```

  Demande d'intégration envoyée au backend :

  ```
  POST http://petstore-demo-endpoint.execute-api.com/petstore/pets HTTP/1.1
  Content-Type: application/json
  Content-Length: ...

  {
    "type" : "dog",
    "price" : 1001.00
  ```

```
}
```

Les instances d'exécution de la méthode ANY et de la ressource de proxy sont toutes les deux valides. L'appel renvoie une réponse `200 OK` avec la charge utile contenant l'animal récemment créé, telle que retournée depuis le backend.

- Définir **`ANY`** en tant que **`GET`** et **`{proxy+}`** en tant que **`pets/cat`**

  Demande de méthode initiée depuis le serveur frontal :

  ```
  GET https://4z9giyi2c1.execute-api.us-west-2.amazonaws.com/test/pets/cat
  ```

  Demande d'intégration envoyée au backend :

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

  L'instance d'exécution du chemin de ressource de proxy ne correspond pas au point de terminaison du backend et la demande qui en résulte n'est pas valide. Par conséquent, une réponse `400 Bad Request` est renvoyée avec le message d'erreur suivant.

  ```
  {
    "errors": [
      {
        "key": "Pet2.type",
        "message": "Missing required field"
      },
      {
        "key": "Pet2.price",
        "message": "Missing required field"
      }
    ]
  }
  ```

- Définir **`ANY`** en tant que **`GET`** et **`{proxy+}`** en tant que **`null`**

  Demande de méthode initiée depuis le serveur frontal :

  ```
  GET https://4z9giyi2c1.execute-api.us-west-2.amazonaws.com/test
  ```

  Demande d'intégration envoyée au backend :

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

La ressource ciblée est le parent de la ressource de proxy, mais l'instance d'exécution de la méthode ANY n'est pas définie dans l'API sur cette ressource. Par conséquent, cette demande GET renvoie une réponse `403 Forbidden` avec le message d'erreur `Missing Authentication Token` comme celle renvoyée par API Gateway. Si l'API expose la méthode ANY ou GET sur la ressource parent (`/`), l'appel renvoie une réponse `404 Not Found` avec le message `Cannot GET /petstore` telle que retournée depuis le serveur principal.

Pour toute demande de client, si l'URL de point de terminaison ciblée n'est pas valide ou si le verbe HTTP est valide, mais non pris en charge, le backend renvoie une réponse `404 Not Found`. Pour une méthode HTTP non prise en charge, une réponse `403 Forbidden` est renvoyée.

Configuration des intégrations personnalisées HTTP dans API Gateway

Avec l'intégration personnalisée HTTP, vous avez davantage de contrôle sur les données à transmettre entre une méthode d'API et une intégration d'API, et sur la façon de les transmettre. Pour ce faire, vous devez utiliser les mappages de données.

Dans le cadre de la configuration de la demande de méthode, vous définissez la propriété requestParameters sur une ressource Method. Cette action permet de déclarer quels paramètres de la demande de méthode (qui sont mis en service à partir du client) doivent être mappés aux paramètres de demande d'intégration ou aux propriétés de corps applicables avant d'être envoyés vers le backend. Ensuite, dans le cadre de la configuration de la demande d'intégration, vous définissez la propriété RequestParameters sur la ressource d'intégration correspondante pour spécifier les mappages. parameter-to-parameter Vous définissez également la propriété requestTemplates afin de spécifier les modèles de mappage, un par type de contenu pris en charge. Les modèles de mappage mappent les paramètres de la demande de méthode, ou le corps, au corps de la demande d'intégration.

De même, dans le cadre de la configuration de la réponse de la méthode, vous définissez la propriété ResponseParameters sur la ressource. MethodResponse Cette action permet de déclarer quels paramètres de la réponse de méthode (qui sont envoyés au client) doivent être mappés depuis les paramètres de réponse d'intégration ou certaines propriétés de corps applicables renvoyés par le backend. Ensuite, dans le cadre de la configuration de la réponse d'intégration, vous définissez la propriété ResponseParameters sur la ressource IntegrationResponsecorrespondante pour spécifier les mappages. parameter-to-parameter Vous définissez également le mappage responseTemplates

afin de spécifier les modèles de mappage, un par type de contenu pris en charge. Les modèles de mappage mappent les paramètres de la réponse d'intégration, ou les propriétés du corps de la réponse d'intégration, au corps de la réponse de méthode.

Pour de plus amples informations sur la configuration des modèles de mappage, veuillez consulter Configuration des transformations de données pour les API REST.

Configuration des intégrations privées API Gateway

L'intégration privée API Gateway facilite l'exposition des ressources HTTP/HTTPS dans un Amazon VPC pour permettre aux clients en dehors du VPC d'y accéder. Pour étendre l'accès à vos ressources VPC privées au-delà des limites du VPC, vous pouvez créer une API avec l'intégration privée. Vous pouvez contrôler l'accès à votre API à l'aide de n'importe quelle méthode d'autorisation prise en charge par API Gateway.

Pour créer une intégration privée, vous devez d'abord créer un Network Load Balancer. Votre Network Load Balancer doit disposer d'un écouteur qui achemine les demandes vers les ressources de votre VPC. Afin d'améliorer la disponibilité de votre API, assurez-vous que votre Network Load Balancer achemine le trafic vers des ressources dans plusieurs zones de disponibilité dans le Région AWS. Ensuite, vous créez un lien VPC que vous utilisez pour connecter votre API et votre Network Load Balancer. Après avoir créé une liaison VPC, vous créez des intégrations privées pour acheminer le trafic de votre API vers les ressources de votre VPC via votre liaison VPC et votre Network Load Balancer.

> **Note**
>
> Le Network Load Balancer et l'API doivent appartenir au même AWS compte.

Avec l'intégration privée API Gateway, vous pouvez activer l'accès aux ressources HTTP/HTTPS au sein d'un VPC sans connaître dans le détail les configurations des réseaux privés ni les appareils spécifiques aux technologies.

Rubriques

- Configuration d'un équilibreur Network Load Balancer pour les intégrations privées API Gateway
- Octroi d'autorisations pour créer un lien VPC
- Configuration d'une API API Gateway avec des intégrations privées à l'aide de la console API Gateway

- Configurez une API API Gateway avec des intégrations privées à l'aide du AWS CLI
- Configuration d'une API avec des intégrations privées à l'aide d'OpenAPI
- Comptes API Gateway utilisés pour les intégrations privées

Configuration d'un équilibreur Network Load Balancer pour les intégrations privées API Gateway

La procédure suivante décrit les étapes de configuration d'un équilibreur Network Load Balancer (NLB) pour les intégrations privées API Gateway à l'aide de la console Amazon EC2, et fournit des références renvoyant à des instructions détaillées pour chaque étape.

Pour chaque VPC contenant des ressources, il vous suffit de configurer un NLB et un VPCLink. L'équilibreur NLB prend en charge plusieurs écouteurs et groupes cibles par équilibreur NLB. Vous pouvez configurer chaque service en tant qu'écouteur spécifique sur le NLB et utiliser un seul VPCLink pour vous connecter au NLB. Lors de la création de l'intégration privée dans API Gateway, vous définissez chaque service à l'aide du port spécifique qui a été attribué à chaque service. Pour plus d'informations, consultez the section called “Tutoriel : Création d'une API avec intégration privée”.

> **Note**
>
> Le Network Load Balancer et l'API doivent appartenir au même AWS compte.

Pour créer un Network Load Balancer pour une intégration privée à l'aide de la console API Gateway

1. Connectez-vous à la console Amazon EC2 AWS Management Console et ouvrez-la à l'adresse https://console.aws.amazon.com/ec2/.
2. Configurez un serveur web sur une instance Amazon EC2. Pour voir un exemple de configuration, consultez Installing a LAMP Web Server on Amazon Linux 2 (Installation d'un serveur web LAMP sur Amazon Linux 2).
3. Créez un équilibreur Network Load Balancer, enregistrez l'instance EC2 auprès d'un groupe cible et ajoutez le groupe cible à un écouteur de l'équilibreur Network Load Balancer. Pour de plus amples informations, veuillez suivre les instructions fournies dans la section Premiers pas avec les équilibreurs Network Load Balancer.
4. Une fois que l'équilibreur Network Load Balancer est créé, procédez de la façon suivante :

a. Notez l'ARN de l'équilibreur Network Load Balancer. Vous en aurez besoin pour créer un lien VPC dans API Gateway pour intégrer l'API aux ressources VPC derrière l'équilibreur Network Load Balancer.

b. Désactiver l'évaluation des groupes de sécurité pour PrivateLink.

- Pour désactiver l'évaluation des groupes de sécurité pour le PrivateLink trafic à l'aide de la console, vous pouvez choisir l'onglet Sécurité, puis Modifier. Dans les paramètres de sécurité, décochez Appliquer les règles de trafic entrant au PrivateLink trafic.
- Pour désactiver l'évaluation des groupes de sécurité pour le PrivateLink trafic à l'aide de AWS CLI, utilisez la commande suivante :

```
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**
>
> N'ajoutez pas de dépendances aux CIDR d'API Gateway car ils sont susceptibles de changer sans préavis.

Octroi d'autorisations pour créer un lien VPC

Pour que vous ou un utilisateur de votre compte puissiez créer et gérer un lien VPC, vous devez disposer des autorisations requises pour créer, supprimer et afficher les configurations de service du point de terminaison du VPC, modifier ses autorisations de service et examiner les équilibreurs de charge. Pour accorder ces autorisations, suivez la procédure ci-dessous.

Pour accorder des autorisations afin de créer, mettre à jour et supprimer un lien VPC

1. Créez une stratégie IAM similaire à celle-ci :

```
{
    "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. Créez ou choisissez un rôle IAM et attachez-y la stratégie précédente.
3. Attribuez le rôle IAM à vous-même ou à un utilisateur de votre compte qui crée des liens VPC.

Configuration d'une API API Gateway avec des intégrations privées à l'aide de la console API Gateway

Pour obtenir des instructions sur l'utilisation de la console API Gateway afin de configurer une API avec intégration privée, consultezTutoriel : Création d'une API REST avec l'intégration privée API Gateway.

Configurez une API API Gateway avec des intégrations privées à l'aide du AWS CLI

Avant de créer une API avec l'intégration privée, votre ressource VPC doit être configurée et un équilibreur Network Load Balancer doit être créé et configuré avec votre source de VPC en tant que cible. Si les exigences ne sont pas satisfaites, suivez Configuration d'un équilibreur Network Load Balancer pour les intégrations privées API Gateway pour installer la ressource du VPC, créer un équilibreur Network Load Balancer et définir la ressource du VPC en tant que cible de l'équilibreur Network Load Balancer.

> **Note**
>
> Le Network Load Balancer et l'API doivent appartenir au même AWS compte.

Pour que vous puissiez être en mesure de créer et de gérer un `VpcLink`, vous devez également disposer des autorisations appropriées configurés. Pour plus d'informations, consultez Octroi d'autorisations pour créer un lien VPC.

> **Note**
>
> Vous avez uniquement besoin des autorisations pour créer une `VpcLink` dans votre API. Vous n'avez pas besoin des autorisations permettant d'utiliser le composant `VpcLink`.

Une fois que l'équilibreur Network Load Balancer est créé, prenez note de son ARN. Vous en aurez besoin pour créer un lien VPC pour l'intégration privée.

Pour configurer une API avec l'intégration privée à l'aide de AWS CLI

1. Créez un `VpcLink` ciblant l'équilibreur Network Load Balancer spécifié.

   ```
   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
   ```

   La sortie de cette commande accuse réception de la requête et indique l'état `PENDING` de la création de `VpcLink`.

```
{
    "status": "PENDING",
    "targetArns": [
        "arn:aws:elasticloadbalancing:us-east-2:123456789012:loadbalancer/net/my-
vpclink-test-nlb/1234567890abcdef"
    ],
    "id": "gim7c3",
    "name": "my-test-vpc-link"
}
```

La création du `VpcLink` par API Gateway prend entre 2 et 4 minutes. Lorsque l'opération se termine correctement, l'attribut `status` indique `AVAILABLE`. Pour vérifier cette information, exécutez la commande CLI suivante :

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

Si l'opération échoue, vous obtenez le statut `FAILED` et un message d'erreur sous `statusMessage`. Par exemple, si vous tentez de créer un `VpcLink` avec un équilibreur Network Load Balancer qui est déjà associé au point de terminaison du VPC, vous obtenez le message suivant au niveau de la propriété `statusMessage` :

```
"NLB is already associated with another VPC Endpoint Service"
```

Une fois la création de `VpcLink` réussie, vous pouvez créer une API et l'intégrer à la ressource VPC par le biais de `VpcLink`.

Notez la valeur `id` du `VpcLink` que vous venez de créer (*`gim7c3`* dans la sortie précédente). Vous en aurez besoin pour configurer l'intégration privée.

2. Configurez une API en créant une ressource API Gateway `RestApi` :

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

Notez la valeur `RestApi` de `id` dans le résultat renvoyé. Vous avez besoin de cette valeur pour effectuer d'autres opérations sur l'API.

À des fins d'illustration, nous allons créer une API avec seulement une méthode `GET` sur la ressource racine (`/`) et intégrer la méthode au `VpcLink`.

3. Configurez la méthode `GET /`. Tout d'abord, obtenez l'identifiant de la ressource racine (`/`):

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

Dans la sortie, notez la valeur `id` du chemin `/`. Dans cet exemple, nous présumons qu'il s'agit de *skpp60rab7*.

Configurez la demande pour la méthode d'API de `GET /`:

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

Si vous n'utilisez pas l'intégration proxy au `VpcLink`, vous devez également configurer au moins une réponse pour la méthode avec le code de statut `200`. Dans le cas présent, nous allons utiliser l'intégration proxy.

4. Configurez l'intégration privée de type `HTTP_PROXY` et appelez la commande `put-integration` comme suit :

```
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
```

Pour une intégration privée, définissez `connection-type` sur `VPC_LINK` et `connection-id` sur l'identifiant de votre `VpcLink` ou sur une variable d'étape faisant référence à votre identifiant `VpcLink`. Le paramètre `uri` n'est pas utilisé pour l'acheminement des demandes au point de terminaison, mais pour définir l'en-tête `Host` et pour la validation de certificat.

La commande renvoie le résultat suivant :

```
{
```

```
    "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": []
}
```

Avec une variable d'étape, vous définissez la propriété `connectionId` lors de la création de l'intégration :

```
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}"
```

Assurez-vous de délimiter l'expression de la variable d'étape avec des guillemets droits (`${stageVariables.vpcLinkId}`) et d'y ajouter le caractère d'échappement `$`.

Vous pouvez également mettre à jour l'intégration pour réinitialiser la valeur `connectionId` avec une variable d'étape :

```
 aws apigateway update-integration \
    --rest-api-id abcdef123 \
    --resource-id skpp60rab7 \
    --http-method GET \
    --patch-operations '[{"op":"replace","path":"/
connectionId","value":"${stageVariables.vpcLinkId}"}]'
```

Assurez-vous d'utiliser une liste JSON obtenue à l'aide de stringify comme valeur du paramètre `patch-operations`.

Vous pouvez utiliser une variable d'étape pour intégrer votre API à un autre VPC ou à un Network Load Balancer (équilibreur de charge réseau) en réinitialisant la valeur de la variable d'étape de `VpcLink`.

Comme nous avons utilisé l'intégration proxy privée, l'API peut maintenant être déployée et testée. Avec l'intégration non proxy, vous devez également configurer la réponse de la méthode et la réponse de l'intégration, comme vous le feriez lorsque vous configurez une API avec des intégrations HTTP personnalisées.

5. Pour tester l'API, déployez-la. Ceci est nécessaire si vous avez utilisé la variable d'étape en tant qu'espace réservé de l'ID `VpcLink`. Pour déployer l'API avec une variable d'étape, appelez la commande `create-deployment` comme suit :

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

Pour mettre à jour la variable d'étape avec un autre ID `VpcLink` (par exemple, *asf9d7*), appelez la commande `update-stage` :

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

Utilisez la commande suivante pour appeler votre API :

```
curl -X GET https://abcdef123.execute-api.us-east-2.amazonaws.com/test
```

Vous pouvez également appeler l'URL d'appel de l'API dans un navigateur web pour afficher le résultat.

Lorsque vous codez en dur la propriété `connection-id` avec l'ID littéral `VpcLink`, vous pouvez également appeler `test-invoke-method` pour tester l'appel de l'API avant qu'elle ne soit déployée.

Configuration d'une API avec des intégrations privées à l'aide d'OpenAPI

Vous pouvez configurer une API avec l'intégration privée via l'importation de fichier d'API OpenAPI. Les paramètres sont similaires aux définitions OpenAPI d'une API avec des intégrations HTTP, à l'exception des points suivants :

- Vous devez explicitement définir `connectionType` sur `VPC_LINK`.
- Vous devez définir explicitement `connectionId` sur l'ID d'un `VpcLink` ou d'une variable d'étape renvoyant à l'ID d'un `VpcLink`.
- Le paramètre `uri` dans les intégrations privées renvoie vers un point de terminaison HTTP/HTTPS dans le VPC, mais est utilisé pour configurer l'en-tête `Host` de la demande d'intégration.
- Le paramètre `uri` de l'intégration privée avec un point de terminaison HTTPS dans le VPC est utilisé pour vérifier le nom de domaine par rapport à celui indiqué dans le certificat installé sur le point de terminaison du VPC.

Vous pouvez utiliser une variable d'étape pour référencer l'ID `VpcLink`. Vous pouvez également attribuer la valeur d'ID directement à `connectionId`.

Le fichier d'API OpenAPI au format JSON suivant montre un exemple de fichier de lien VPC référencé par une variable d'étape (`${stageVariables.vpcLinkId}`) :

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"
    }
  }
}
```

Comptes API Gateway utilisés pour les intégrations privées

Les ID de compte API Gateway spécifiques à la région suivants sont automatiquement ajoutés à votre service de point de terminaison de VPC comme `AllowedPrincipals` lorsque vous créez un lien `VpcLink`.

Région	ID de compte
us-east-1	392220576650
us-east-2	718770453195
us-west-1	968246515281
us-west-2	109351309407
ca-central-1	796887884028
eu-west-1	631144002099
eu-west-2	544388816663
eu-west-3	061510835048
eu-central-1	474240146802
eu-central-2	166639821150
eu-north-1	394634713161
eu-south-1	753362059629
eu-south-2	359345898052
ap-northeast-1	969236854626
ap-northeast-2	020402002396
ap-northeast-3	360671645888
ap-southeast-1	195145609632
ap-southeast-2	798376113853
ap-southeast-3	652364314486
ap-southeast-4	849137399833

Région	ID de compte
ap-south-1	507069717855
ap-south-2	644042651268
ap us-east-1	174803364771
sa-east-1	287228555773
me-south-1	855739686837
me-central-1	614065512851

Configuration des intégrations fictives dans API Gateway

Amazon API Gateway prend en charge les intégrations fictives pour les méthodes API. Cette fonction permet aux développeurs d'API de générer des réponses d'API directement à partir d'API Gateway, sans nécessiter de backend d'intégration. En tant que développeur d'API, vous pouvez utiliser cette fonction pour débloquer les équipes dépendantes qui doivent utiliser une API avant la fin du développement du projet. Vous pouvez également utiliser cette fonction pour mettre en service une page de destination pour votre API, qui peut fournir une présentation de votre API et un accès à cette dernière. Pour obtenir un exemple de page de destination de ce type, consultez la demande et la réponse d'intégration de la méthode GET sur la ressource racine de l'exemple d'API présentées dans Tutoriel : Création d'une API REST par l'importation d'un exemple.

En tant que développeur d'API, vous décidez de la façon dont API Gateway répond à une demande d'intégration fictive. Pour cela, vous configurez la demande d'intégration et la réponse d'intégration de la méthode pour associer une réponse à un code d'état donné. Pour une méthode avec l'intégration fictive permettant de renvoyer une réponse `200`, configurez le modèle de mappage du corps de la demande d'intégration pour qu'il renvoie ce qui suit.

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

Configurer une réponse d'intégration `200` pour que le modèle de mappage de corps suivant, par exemple :

```
{
```

```
    "statusCode": 200,
    "message": "Go ahead without me."
}
```

De même, pour que la méthode renvoie, par exemple, une réponse d'erreur `500`, configurez le modèle de mappage du corps de la demande d'intégration pour qu'il renvoie ce qui suit.

```
{"statusCode": 500}
```

Configurer une réponse d'intégration `500` avec, par exemple, le modèle de mappage de corps suivant :

```
{
    "statusCode": 500,
    "message": "The invoked method is not supported on the API resource."
}
```

Sinon, vous pouvez obtenir qu'une méthode d'intégration fictive renvoie la réponse d'intégration par défaut sans définir le modèle de mappage de demande d'intégration. La réponse d'intégration par défaut correspond à celle ayant un statut de HTTP status regex non défini. Assurez-vous que les comportements de transfert appropriés sont définis.

> **Note**
>
> les intégrations fictives ne sont pas destinés à prendre en charge des modèles de réponse de grande taille. Si vous en avez besoin pour votre cas d'utilisation, vous devriez plutôt envisager d'utiliser une intégration Lambda.

À l'aide d'un modèle de mappage de demande d'intégration, vous pouvez injecter une logique d'application permettant de choisir la réponse d'intégration fictive à renvoyer selon certaines conditions. Par exemple, vous pouvez utiliser un paramètre de requête `scope` sur les demandes entrantes pour déterminer s'il faut renvoyer une réponse positive ou une réponse d'erreur :

```
{
  #if( $input.params('scope') == "internal" )
    "statusCode": 200
  #else
    "statusCode": 500
```

```
  #end
}
```

Ainsi, la méthode de l'intégration fictive laisse passer les appels internes tout en rejetant d'autres types d'appels en renvoyant une réponse d'erreur.

Dans cette section, nous décrivons comment utiliser la console API Gateway pour activer l'intégration fictive d'une méthode d'API.

Rubriques

- Activation de l'intégration fictive à l'aide de la console API Gateway

Activation de l'intégration fictive à l'aide de la console API Gateway

La méthode doit être disponible dans API Gateway. Suivez les instructions de la section Tutoriel : Création d'une API REST avec une intégration HTTP autre que de proxy.

1. Choisissez une ressource d’API et choisissez Créer une méthode.

   Pour créer la méthode, procédez comme suit :

   a. Pour Type de méthode, sélectionnez une méthode.
   b. Pour Type d’intégration, sélectionnez HTTP.
   c. Choisissez Créer une méthode.
   d. Dans l’onglet Requête de méthode, pour Paramètres de requête de méthode, choisissez Modifier.
   e. Choisissez Paramètres de chaîne de requête d’URL. Choisissez Ajouter une chaîne de requête et pour Nom, saisissez**`scope`**. Ce paramètres de requête permet de déterminer si l’appelant est interne ou autre.
   f. Choisissez Enregistrer.
2. Dans l’onglet Méthode de réponse, choisissez Créer une réponse, puis procédez comme suit :

   a. Pour Statut HTTP, saisissez **`500`**.
   b. Choisissez Enregistrer.
3. Dans l’onglet Demande d’intégration, pour Paramètres de requête d’intégration, choisissez Modifier.

4. Choisissez Modèles de mappage, puis procédez comme suit :

   a. Sélectionnez Add mapping template.

   b. Pour Type de contenu, entrez **`application/json`**.

   c. Pour Corps du modèle, saisissez ce qui suit :

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

   d. Choisissez Enregistrer.

5. Dans l'onglet Réponse d'intégration, pour Par défaut - Réponse, choisissez Modifier.

6. Choisissez Modèles de mappage, puis procédez comme suit :

   a. Pour Type de contenu, entrez **`application/json`**.

   b. Pour Corps du modèle, saisissez ce qui suit :

   ```
   {
       "statusCode": 200,
       "message": "Go ahead without me"
   }
   ```

   c. Choisissez Enregistrer.

7. Choisissez Créer une réponse.

   Pour créer une réponse 500, procédez comme suit :

   a. Pour HTTP status regex (Regex statut HTTP), saisissez **`5\d{2}`**.

   b. Pour Statut de la réponse de méthode, sélectionnez **`500`**.

   c. Choisissez Enregistrer.

   d. Pour 5\d{2} - Réponse, choisissez Modifier.

   e. Choisissez Modèles de mappage, puis choisissez Ajouter un modèle de mappage.

   f. Pour Type de contenu, entrez **`application/json`**.

   g. Pour Corps du modèle, saisissez ce qui suit :

```
{
    "statusCode": 500,
    "message": "The invoked method is not supported on the API resource."
}
```

h. Choisissez Enregistrer.

8. Choisissez l'onglet Test. Vous devrez peut-être choisir la flèche droite pour afficher l'onglet. Pour tester votre intégration simulée, procédez comme suit :

a. Saisissez `scope=internal` sous Chaînes de requête. Sélectionnez Test. Le résultat de test indique :

```
Request: /?scope=internal
Status: 200
Latency: 26 ms
Response Body

{
  "statusCode": 200,
  "message": "Go ahead without me"
}

Response Headers

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

b. Saisissez `scope=public` sous `Query strings` ou laissez le champ vide. Sélectionnez Test. Le résultat de test indique :

```
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"}
```

Vous pouvez également renvoyer des en-têtes dans une réponse d'intégration fictive en ajoutant d'abord un en-tête à la réponse de méthode, puis en configurant un mappage d'en-tête d'intégration dans la réponse d'intégration. En fait, c'est ainsi que la console API Gateway permet à la prise en charge de CORS en renvoyant les en-têtes CORS requis.

Utilisation de la validation des demandes dans API Gateway

Vous pouvez configurer API Gateway afin d'exécuter la validation de base d'une demande d'API avant de procéder à la demande d'intégration. Lorsque la validation échoue, API Gateway échoue immédiatement à la demande, renvoie une réponse d'erreur 400 à l'appelant et publie les résultats de la validation dans CloudWatch Logs. Cela permet de réduire les appels non nécessaires au backend. Qui plus est, il vous permet de vous concentrer sur les efforts de validation spécifiques à votre application. Vous pouvez valider un corps de demande en vérifiant que les paramètres de demande obligatoires sont valides et autres que null ou en indiquant un schéma de modèle pour la validation de données plus compliquées.

Rubriques

- Présentation de la validation de demande de base dans API Gateway
- Comprendre les modèles de données
- Configuration de la validation de demande de base dans API Gateway
- Définitions OpenAPI d'un exemple d'API avec validation de demande de base
- AWS CloudFormation modèle d'un exemple d'API avec validation de demande de base

Présentation de la validation de demande de base dans API Gateway

API Gateway peut effectuer la validation de base des demandes, afin que vous puissiez vous concentrer sur la validation spécifique à l'application dans le backend. Pour la validation, API Gateway vérifie l'une ou l'ensemble des conditions suivantes :

- Les paramètres de demande obligatoires dans l'URI, la chaîne de requête et les en-têtes d'une demande entrante sont inclus et non vides.

- La charge utile de la demande adhère à la demande de schéma JSON configurée de la méthode.

Pour activer la validation, vous indiquez des règles de validation dans un validateur de demande, vous ajoutez le validateur au mappage des validateurs de demande de l'API, puis vous affectez le validateur à des méthodes d'API individuelles.

> **Note**
>
> La validation du corps de la demande et les Comportements de transfert direct sont deux sujets distincts. Quand la charge utile d'une demande n'a pas de schéma de modèle correspondant, vous pouvez choisir de transmettre ou de bloquer la charge utile d'origine. Pour plus d'informations, consultez Comportements de transfert direct.

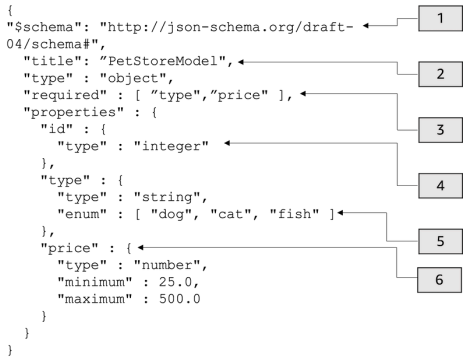
Comprendre les modèles de données

Dans API Gateway, un modèle définit la structure de données d'une charge utile. Dans API Gateway, les modèles sont définis à l'aide du schéma JSON version 4. L'objet JSON suivant est un exemple de données figurant dans l'exemple d'animalerie.

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

Les données contiennent les éléments `id`, `type` et `price` de l'animal de compagnie. Un modèle de ces données vous permet de :

- Utiliser la validation de base des demandes.
- Créer des modèles de mappage pour la transformation des données.
- Créer un type de données défini par l'utilisateur (UDT) lorsque vous générez un kit SDK.



Dans ce modèle :

1. L'objet `$schema` représente un identifiant de version de schéma JSON valide. Ce schéma est le brouillon de schéma JSON version 4.
2. L'objet `title` est un identifiant du modèle lisible à l'œil. Ce titre est `PetStoreModel`.
3. Le mot clé de validation `required` requiert `type` et `price` pour la validation de base des demandes.
4. Les propriétés (`properties`) du modèle sont `id`, `type` et `price`. Chaque objet possède des propriétés qui sont décrites dans le modèle.
5. L'objet `type` ne peut avoir que les valeurs `dog`, `cat` ou `fish`.
6. L'objet `price` est un nombre limité entre un `minimum` de 25 et un `maximum` de 500.

PetStore modèle

```
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 }
```

Dans ce modèle :

1. Ligne 2, l'objet `$schema` représente un identifiant de version de schéma JSON valide. Ce schéma est le brouillon de schéma JSON version 4.
2. Ligne 3, l'objet `title` est un identifiant lisible du modèle. Ce titre est `PetStoreModel`.
3. Ligne 5, le mot clé de validation `required` requiert `type` et `price` pour la validation de base des demandes.
4. Lignes 6 à 17, les propriétés (`properties`) du modèle sont `id`, `type` et `price`. Chaque objet possède des propriétés qui sont décrites dans le modèle.
5. Ligne 12, l'objet `type` ne peut avoir que les valeurs `dog`, `cat` ou `fish`.
6. Lignes 14 à 17, l'objet `price` est un nombre limité entre un `minimum` de 25 et un `maximum` de 500.

Création de modèles plus complexes

Vous pouvez utiliser la primitive `$ref` pour créer des définitions réutilisables pour des modèles plus longs. Par exemple, vous pouvez créer une définition appelée `Price` dans la section `definitions` décrivant l'objet `price`. La valeur de `$ref` est la définition `Price`.

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

Vous pouvez également référencer un autre schéma de modèle défini dans un fichier de modèle externe. Définissez la valeur de la propriété `$ref` en fonction de l'emplacement du modèle. Dans l'exemple suivant, le modèle `Price` est défini dans le modèle `PetStorePrice` dans l'API `a1234`.

```
{
  "$schema" : "http://json-schema.org/draft-04/schema#",
  "title" : "PetStorePrice",
  "type": "number",
  "minimum": 25,
  "maximum": 500
}
```

Le modèle le plus long peut référencer le modèle `PetStorePrice`.

```
{
  "$schema" : "http://json-schema.org/draft-04/schema#",
  "title" : "PetStoreModelReusableRefAPI",
  "required" : [ "price", "type" ],
  "type" : "object",
  "properties" : {
    "id" : {
      "type" : "integer"
    },
    "type" : {
      "type" : "string",
      "enum" : [ "dog", "cat", "fish" ]
    },
    "price" : {
        "$ref": "https://apigateway.amazonaws.com/restapis/a1234/models/PetStorePrice"
    }
```

```
  }
}
```

Utilisation de modèles de données de sortie

Si vous transformez vos données, vous pouvez définir un modèle de charge utile dans la réponse d'intégration. Un modèle de charge utile peut être utilisé lorsque vous générez un kit SDK. Pour les langages fortement typés, comme Java, Objective-C ou Swift, l'objet correspond à un type de données défini par l'utilisateur (UDT). API Gateway crée un type UDT si vous lui fournissez un modèle de données lorsque vous générez un kit SDK. Pour plus d'informations sur les transformations de données, consultez Présentation des modèles de mappage.

Données de sortie

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

Modèle de sortie

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

Avec ce modèle, vous pouvez appeler un kit SDK pour récupérer les valeurs des propriétés `description` et `askingPrice` en lisant les propriétés `PetStoreOutputModel[i].description` et `PetStoreOutputModel[i].askingPrice`. Si aucun modèle n'est fourni, API Gateway utilise le modèle vide pour créer un type UDT par défaut.

Étapes suivantes

- Cette section fournit des ressources que vous pouvez utiliser pour en apprendre davantage sur les concepts présentés dans cette rubrique.

  Vous pouvez suivre les didacticiels de validation des demandes :

  - Configuration la validation des demandes à l'aide de la console API Gateway
  - Configurez la validation de base des demandes à l'aide du AWS CLI
  - Configuration de la validation de base des demandes à l'aide d'une définition OpenAPI
- Vous pouvez obtenir plus d'informations sur la transformation des données et les modèles de mappage, Présentation des modèles de mappage.
- Vous pouvez également voir des modèles de données plus complexes. veuillez consulter Exemples de modèles de données et de modèles de mappage pour API Gateway.

Configuration de la validation de demande de base dans API Gateway

Cette section explique comment configurer la validation des demandes pour API Gateway à l'aide de la console et d'une définition OpenAPI. AWS CLI

Rubriques

- Configuration la validation des demandes à l'aide de la console API Gateway
- Configurez la validation de base des demandes à l'aide du AWS CLI

- Configuration de la validation de base des demandes à l'aide d'une définition OpenAPI

Configuration la validation des demandes à l'aide de la console API Gateway

Vous pouvez utiliser la console API Gateway pour valider une demande en sélectionnant l'un des trois validateurs suivants pour une demande d'API :

- Valider le corps.
- Valider les paramètres de chaîne de requête et les en-têtes.
- Valider le corps et les paramètres et en-têtes des chaînes de requête.

Lorsque vous appliquez l'un des validateurs à une méthode d'API, la console API Gateway ajoute le validateur à la carte de RequestValidatorsl'API.

Pour suivre ce didacticiel, vous allez utiliser un AWS CloudFormation modèle pour créer une API API Gateway incomplète. Cette API possède une ressource `/validator` avec les méthodes `GET` et `POST`. Ces deux méthodes sont intégrées au point de terminaison HTTP `http://petstore-demo-endpoint.execute-api.com/petstore/pets`. Vous allez configurer deux types de validation des demandes :

- Dans la méthode `GET`, vous allez configurer la validation des demandes pour les paramètres de chaîne de requête d'URL.
- Dans la méthode `POST`, vous allez configurer la validation des demandes pour le corps de la demande.

Cela permettra uniquement à certains appels d'API d'être transmis à l'API.

Téléchargez et décompressez le modèle de création d'application pour AWS CloudFormation. Vous allez utiliser ce modèle pour créer une API incomplète. Vous terminerez les étapes restantes dans la console API Gateway.

Pour créer une AWS CloudFormation pile

1. Ouvrez la AWS CloudFormation console à l'adresse https://console.aws.amazon.com/cloudformation.
2. Choisissez Créer une pile, puis choisissez Avec de nouvelles ressources (standard).
3. Dans Spécifier le modèle, choisissez Charger un modèle de fichier.

4. Sélectionnez le modèle que vous avez téléchargé.
5. Choisissez Suivant.
6. Pour Nom de la pile, saisissez **`request-validation-tutorial-console`**, puis choisissez Suivant.
7. Pour Configurer les options de pile, choisissez Suivant.
8. Pour les fonctionnalités, reconnaissez que AWS CloudFormation vous pouvez créer des ressources IAM dans votre compte.
9. Sélectionnez Envoyer.

AWS CloudFormation fournit les ressources spécifiées dans le modèle. La fin du provisionnement de vos ressources peut prendre quelques minutes. Lorsque le statut de votre AWS CloudFormation pile est CREATE_COMPLETE, vous êtes prêt à passer à l'étape suivante.

Pour sélectionner l'API que vous venez de créer

1. Sélectionnez la pile **`request-validation-tutorial-console`** nouvellement créée.
2. Sélectionnez Ressources.
3. Sous ID physique, choisissez votre API. Ce lien vous dirigera vers la console API Gateway.

Avant de modifier les méthodes `GET` et `POST`, vous devez créer un modèle.

Pour créer un modèle

1. Un modèle est requis pour utiliser la validation des demandes sur le corps d'une demande entrante. Pour créer un modèle, dans le volet de navigation principal, choisissez Modèles.
2. Sélectionnez Create model.
3. Pour Name (Nom), saisissez **`PetStoreModel`**.
4. Pour Type de contenu, entrez **`application/json`**. Si aucun type de contenu correspondant n'est trouvé, la validation de demande n'est pas effectuée. Pour utiliser le même modèle quel que soit le type de contenu, saisissez **`$default`**.
5. Pour Description, saisissez **`My PetStore Model`** comme description du modèle.
6. Pour Schéma du modèle, collez le modèle suivant dans l'éditeur de code, puis choisissez Créer.

```
{
  "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
    }
  }
}
```

Pour plus d'informations sur le modèle, consultez Comprendre les modèles de données.

Pour configurer la validation de requête pour une méthode **GET**

1. Dans le volet de navigation principal, choisissez Ressources, puis sélectionnez la méthode GET.
2. Dans l'onglet Demande de méthode, sous Paramètres de demande de méthode, choisissez Modifier.
3. Pour Validateur de requête, sélectionnez Valider les paramètres de chaîne de requête et les en-têtes.
4. Sous Paramètres de chaîne de requête d'URL, procédez comme suit :
   a. Sélectionnez Add query string (Ajouter une chaîne de requêtes).
   b. Pour Name (Nom), saisissez **petType**.
   c. Activez Obligatoire.
   d. Maintenez Mise en cache désactivée.
5. Choisissez Enregistrer.
6. Dans l'onglet Requête d'intégration, sous Paramètres de requête d'intégration, choisissez Modifier.

7. Sous Paramètres de chaîne de requête d'URL, procédez comme suit :

   a. Sélectionnez Add query string (Ajouter une chaîne de requêtes).

   b. Pour Name (Nom), saisissez **`petType`**.

   c. Pour Mappage à partir de, entrez **`method.request.querystring.petType`**. Cela mappe **`petType`** au type d'animal de compagnie.

      Pour plus d'informations sur le mappage des données, consultez le didacticiel sur le mappage des données.

   d. Maintenez Mise en cache désactivée.

8. Choisissez Enregistrer.

Pour tester la validation des requêtes pour la méthode **`GET`**

1. Choisissez l'onglet Test. Vous devrez peut-être choisir la flèche droite pour afficher l'onglet.
2. Pour Chaînes de requête, saisissez **`petType=dog`**, puis choisissez Tester.
3. Le test de méthode renverra `200 OK` et une liste de chiens.

   Pour obtenir des informations sur la façon de transformer ces données de sortie, consultez le didacticiel sur le mappage des données.

4. Supprimez **`petType=dog`** et choisissez Tester.
5. Le test de méthode renverra une erreur `400` avec le message d'erreur suivant :

```
{
  "message": "Missing required request parameters: [petType]"
}
```

Pour configurer la validation de requête pour la méthode **`POST`**

1. Dans le volet de navigation principal, sélectionnez Ressources, puis sélectionnez la méthode POST.
2. Dans l'onglet Demande de méthode, sous Paramètres de demande de méthode, choisissez Modifier.
3. Pour Validateur de requête, sélectionnez Valider le corps.
4. Sous Corps de la requête, choisissez Ajouter un modèle.

5. Pour Type de contenu, entrez**`application/json`**, puis pour Modèle, sélectionnez PetStoreModel.
6. Choisissez Enregistrer.

Pour tester la validation de requête pour une méthode **`POST`**

1. Choisissez l'onglet Test. Vous devrez peut-être choisir la flèche droite pour afficher l'onglet.
2. Sous Corps de la requête, collez ce qui suit dans l'éditeur de code :

```
{
  "id": 2,
  "name": "Bella",
  "type": "dog",
  "price": 400
}
```

   Sélectionnez Tester).
3. Le test de méthode renverra `200 OK` et un message de réussite.
4. Sous Corps de la requête, collez ce qui suit dans l'éditeur de code :

```
{
  "id": 2,
  "name": "Bella",
  "type": "dog",
  "price": 4000
}
```

   Sélectionnez Tester).
5. Le test de méthode renverra une erreur `400` avec le message d'erreur suivant :

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

   Au bas des journaux de test, la raison pour laquelle le corps de la requête est non valide est renvoyée. Dans ce cas, le prix de l'animal de compagnie était supérieur au maximum spécifié dans le modèle.

Pour supprimer une AWS CloudFormation pile

1. Ouvrez la AWS CloudFormation console à l'adresse https://console.aws.amazon.com/cloudformation.
2. Sélectionnez votre AWS CloudFormation pile.
3. Choisissez Supprimer, puis confirmez votre choix.

Étapes suivantes

- Pour obtenir des informations sur la façon de transformer les données de sortie et d'effectuer plus de mappage des données, consultez le didacticiel sur le mappage des données.
- Suivez le didacticiel Configuration d'une validation de base des demandes à l'aide de l'interface AWS CLI, pour effectuer des étapes similaires à l'aide de l'interface AWS CLI.

Configurez la validation de base des demandes à l'aide du AWS CLI

Vous pouvez créer un validateur pour configurer la validation des demandes à l'aide de l'interface AWS CLI. Pour suivre ce didacticiel, vous allez utiliser un AWS CloudFormation modèle pour créer une API API Gateway incomplète.

> **Note**
>
> Il ne s'agit pas du même AWS CloudFormation modèle que le didacticiel de la console.

À l'aide d'une ressource `/validator` pré-exposée, vous allez créer les méthodes `GET` et `POST`. Ces deux méthodes seront intégrées au point de terminaison HTTP `http://petstore-demo-endpoint.execute-api.com/petstore/pets`. Vous allez configurer les deux validations de demandes suivantes :

- Sur la méthode `GET`, vous allez créer un validateur `params-only` pour valider les paramètres de chaîne de requête d'URL.
- Sur la méthode `POST`, vous allez créer un validateur `body-only` pour valider le corps de la demande.

Cela permettra uniquement à certains appels d'API d'être transmis à l'API.

Pour créer une AWS CloudFormation pile

Téléchargez et décompressez le modèle de création d'application pour AWS CloudFormation.

Pour effectuer le didacticiel suivant, vous avez besoin de l'AWS Command Line Interface (AWS CLI) version 2.

Pour les commandes longues, un caractère d'échappement (\) est utilisé pour les fractionner en plusieurs lignes.

> **Note**
>
> Dans Windows, certaines commandes CLI Bash que vous utilisez couramment (par exemple `zip`) ne sont pas prises en charge par les terminaux intégrés du système d'exploitation. Installez le sous-système Windows pour Linux afin d'obtenir une version intégrée à Windows d'Ubuntu et Bash. Les exemples de commandes CLI de ce guide utilisent le formatage Linux. Les commandes qui incluent des documents JSON en ligne doivent être reformatées si vous utilisez la CLI Windows.

1. Utilisez la commande suivante pour créer la AWS CloudFormation pile.

```
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 fournit les ressources spécifiées dans le modèle. La fin du provisionnement de vos ressources peut prendre quelques minutes. Utilisez la commande suivante pour voir l'état de votre AWS CloudFormation pile.

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

3. Lorsque le statut de votre AWS CloudFormation pile est définiStackStatus: "CREATE_COMPLETE", utilisez la commande suivante pour récupérer les valeurs de sortie pertinentes pour les étapes futures.

```
aws cloudformation describe-stacks --stack-name request-validation-tutorial-cli
--query "Stacks[*].Outputs[*].{OutputKey: OutputKey, OutputValue: OutputValue,
Description: Description}"
```

Les valeurs de sortie sont les suivantes :

- ApiId, qui est l'ID de l'API. Pour ce didacticiel, l'ID d'API est `abc123`.
- ResourceId, qui est l'ID de la ressource de validation dans laquelle les `POST` méthodes `GET` et sont exposées. Pour ce didacticiel, l'ID de ressource est `efg456`.

Pour créer les validateurs de demandes et importer un modèle

1. Un validateur est requis pour utiliser la validation des demandes avec l'interface AWS CLI. Utilisez la commande suivante pour créer un validateur qui valide uniquement les paramètres de demande.

```
aws apigateway create-request-validator --rest-api-id abc123 \
      --no-validate-request-body \
      --validate-request-parameters \
      --name params-only
```

   Notez l'ID du validateur `params-only`.

2. Utilisez la commande suivante pour créer un validateur qui valide uniquement le corps de la demande.

```
aws apigateway create-request-validator --rest-api-id abc123 \
      --validate-request-body \
      --no-validate-request-parameters \
      --name body-only
```

   Notez l'ID du validateur `body-only`.

3. Un modèle est requis pour utiliser la validation des demandes sur le corps d'une demande entrante. Utilisez la commande suivante pour importer un modèle.

```
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}}}}'
```

Si aucun type de contenu correspondant n'est trouvé, la validation de demande n'est pas effectuée. Pour utiliser le même modèle quel que soit le type de contenu, spécifiez `$default` comme clé.

Pour créer les méthodes **`GET`** et **`POST`**

1. Utilisez la commande suivante pour ajouter la méthode HTTP `GET` sur la ressource `/validate`. Cette commande crée la méthode `GET`, ajoute le validateur `params-only` et définit la chaîne de requête `petType` selon les besoins.

```
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"
```

   Utilisez la commande suivante pour ajouter la méthode HTTP `POST` sur la ressource `/validate`. Cette commande crée la méthode `POST`, ajoute le validateur `body-only` et attache le modèle au validateur de corps uniquement.

```
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. Utilisez la commande suivante pour configurer la réponse `200 OK` de la méthode `GET /validate`.

```
aws apigateway put-method-response --rest-api-id abc123  \
            --resource-id efg456 \
            --http-method GET \
            --status-code 200
```

   Utilisez la commande suivante pour configurer la réponse `200 OK` de la méthode `POST /validate`.

```
aws apigateway put-method-response --rest-api-id abc123  \
            --resource-id efg456 \
            --http-method POST \
            --status-code 200
```

3. Utilisez la commande suivante pour configurer une `Integration` avec un point de terminaison HTTP spécifié pour la méthode `GET /validation`.

```
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'
```

   Utilisez la commande suivante pour configurer une `Integration` avec un point de terminaison HTTP spécifié pour la méthode `POST /validation`.

```
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. Utilisez la commande suivante pour configurer une réponse d'intégration pour la méthode `GET /validation`.

```
aws apigateway put-integration-response --rest-api-id abc123 \
              --resource-id efg456\
              --http-method GET \
              --status-code 200 \
              --selection-pattern ""
```

   Utilisez la commande suivante pour configurer une réponse d'intégration pour la méthode `POST /validation`.

```
aws apigateway put-integration-response --rest-api-id abc123 \
```

```
            --resource-id efg456 \
            --http-method POST \
            --status-code 200 \
            --selection-pattern ""
```

Pour tester l'API

1. Pour tester la méthode GET qui effectuera la validation des demandes pour les chaînes de requête, utilisez la commande suivante :

   ```
   aws apigateway test-invoke-method --rest-api-id abc123 \
               --resource-id efg456 \
               --http-method GET \
               --path-with-query-string '/validate?petType=dog'
   ```

   Le résultat sera `200 OK` et une liste de chiens.

2. Utilisez la commande suivante pour tester sans inclure la chaîne de requête `petType`.

   ```
   aws apigateway test-invoke-method --rest-api-id abc123 \
               --resource-id efg456 \
               --http-method GET
   ```

   Le résultat sera une erreur `400`.

3. Pour tester la méthode POST qui effectuera la validation des demandes pour le corps de la demande, utilisez la commande suivante :

   ```
   aws apigateway test-invoke-method --rest-api-id abc123 \
               --resource-id efg456 \
               --http-method POST \
               --body '{"id": 1, "name": "bella", "type": "dog", "price" : 400 }'
   ```

   Le résultat sera `200 OK` et un message de réussite.

4. Utilisez la commande suivante pour tester en utilisant un corps non valide.

   ```
   aws apigateway test-invoke-method --rest-api-id abc123 \
                --resource-id efg456 \
                --http-method POST \
                --body '{"id": 1, "name": "bella", "type": "dog", "price" : 1000 }'
   ```

Le résultat sera une erreur `400`, car le prix du chien est supérieur au prix maximum défini par le modèle.

Pour supprimer une AWS CloudFormation pile

- Utilisez la commande suivante pour supprimer vos AWS CloudFormation ressources.

```
aws cloudformation delete-stack  --stack-name request-validation-tutorial-cli
```

Configuration de la validation de base des demandes à l'aide d'une définition OpenAPI

Vous pouvez déclarer un validateur de demande au niveau de l'API en spécifiant un ensemble d'objets x-amazon-apigateway-requestobjet -Validators.RequestValidator dans le mappage d'x-amazon-apigateway-requestobjet -validators afin de sélectionner la partie de la demande qui sera validée. Dans l'exemple de définition OpenAPI, il existe deux validateurs :

- Le validateur `all` qui valide à la fois le corps, à l'aide du modèle de données `RequestBodyModel`, et les paramètres.
- Le validateur `param-only` qui valide uniquement les paramètres.

Pour activer un validateur de demande sur toutes les méthodes d'une API, spécifiez une propriété x-amazon-apigateway-requestpropriété -validator au niveau de l'API de la définition OpenAPI. Dans l'exemple de définition OpenAPI, le validateur `all` est utilisé sur toutes les méthodes d'API, sauf s'il est remplacé. Lorsque vous utilisez un modèle pour valider le corps et qu'aucun type de contenu correspondant n'est trouvé, la validation de demande n'est pas effectuée. Pour utiliser le même modèle quel que soit le type de contenu, spécifiez `$default` comme clé.

Pour activer un validateur de demande sur une méthode individuelle, spécifiez la propriété `x-amazon-apigateway-request-validator` au niveau de la méthode. Dans cet exemple de définition OpenAPI, le validateur `param-only` remplace le validateur `all` sur la méthode `GET`.

Pour importer l'exemple OpenAPI dans API Gateway, consultez les instructions suivantes pour réaliser l'Importer une API régionale dans API Gateway ou l'Importation d'une API optimisée pour les périphériques dans API Gateway.

OpenAPI 3.0

```
{
  "openapi" : "3.0.1",
  "info" : {
    "title" : "ReqValidators Sample",
    "version" : "1.0.0"
  },
  "servers" : [ {
    "url" : "/{basePath}",
    "variables" : {
      "basePath" : {
        "default" : "/v1"
      }
    }
  } ],
  "paths" : {
    "/validation" : {
      "get" : {
        "parameters" : [ {
          "name" : "q1",
          "in" : "query",
          "required" : true,
          "schema" : {
            "type" : "string"
          }
        } ],
        "responses" : {
          "200" : {
            "description" : "200 response",
            "headers" : {
              "test-method-response-header" : {
                "schema" : {
                  "type" : "string"
                }
              }
            },
            "content" : {
              "application/json" : {
                "schema" : {
                  "$ref" : "#/components/schemas/ArrayOfError"
                }
              }
            }
```

```
          }
        },
        "x-amazon-apigateway-request-validator" : "params-only",
        "x-amazon-apigateway-integration" : {
          "httpMethod" : "GET",
          "uri" : "http://petstore-demo-endpoint.execute-api.com/petstore/pets",
          "responses" : {
            "default" : {
              "statusCode" : "400",
              "responseParameters" : {
                "method.response.header.test-method-response-header" : "'static
value'"
              },
              "responseTemplates" : {
                "application/xml" : "xml 400 response template",
                "application/json" : "json 400 response template"
              }
            },
            "2\\d{2}" : {
              "statusCode" : "200"
            }
          },
          "requestParameters" : {
            "integration.request.querystring.type" : "method.request.querystring.q1"
          },
          "passthroughBehavior" : "when_no_match",
          "type" : "http"
        }
      },
      "post" : {
        "parameters" : [ {
          "name" : "h1",
          "in" : "header",
          "required" : true,
          "schema" : {
            "type" : "string"
          }
        } ],
        "requestBody" : {
          "content" : {
            "application/json" : {
              "schema" : {
                "$ref" : "#/components/schemas/RequestBodyModel"
              }
```

```
            }
          },
          "required" : true
        },
        "responses" : {
          "200" : {
            "description" : "200 response",
            "headers" : {
              "test-method-response-header" : {
                "schema" : {
                  "type" : "string"
                }
              }
            },
            "content" : {
              "application/json" : {
                "schema" : {
                  "$ref" : "#/components/schemas/ArrayOfError"
                }
              }
            }
          }
        },
        "x-amazon-apigateway-request-validator" : "all",
        "x-amazon-apigateway-integration" : {
          "httpMethod" : "POST",
          "uri" : "http://petstore-demo-endpoint.execute-api.com/petstore/pets",
          "responses" : {
            "default" : {
              "statusCode" : "400",
              "responseParameters" : {
                "method.response.header.test-method-response-header" : "'static
value'"
              },
              "responseTemplates" : {
                "application/xml" : "xml 400 response template",
                "application/json" : "json 400 response template"
              }
            },
            "2\\d{2}" : {
              "statusCode" : "200"
            }
          },
          "requestParameters" : {
```

```
            "integration.request.header.custom_h1" : "method.request.header.h1"
          },
          "passthroughBehavior" : "when_no_match",
          "type" : "http"
        }
      }
    }
  },
  "components" : {
    "schemas" : {
      "RequestBodyModel" : {
        "required" : [ "name", "price", "type" ],
        "type" : "object",
        "properties" : {
          "id" : {
            "type" : "integer"
          },
          "type" : {
            "type" : "string",
            "enum" : [ "dog", "cat", "fish" ]
          },
          "name" : {
            "type" : "string"
          },
          "price" : {
            "maximum" : 500.0,
            "minimum" : 25.0,
            "type" : "number"
          }
        }
      },
      "ArrayOfError" : {
        "type" : "array",
        "items" : {
          "$ref" : "#/components/schemas/Error"
        }
      },
      "Error" : {
        "type" : "object"
      }
    }
  },
  "x-amazon-apigateway-request-validators" : {
    "all" : {
```

```
      "validateRequestParameters" : true,
      "validateRequestBody" : true
    },
    "params-only" : {
      "validateRequestParameters" : true,
      "validateRequestBody" : false
    }
  }
}
```

OpenAPI 2.0

```
{
  "swagger" : "2.0",
  "info" : {
    "version" : "1.0.0",
    "title" : "ReqValidators Sample"
  },
  "basePath" : "/v1",
  "schemes" : [ "https" ],
  "paths" : {
    "/validation" : {
      "get" : {
        "produces" : [ "application/json", "application/xml" ],
        "parameters" : [ {
          "name" : "q1",
          "in" : "query",
          "required" : true,
          "type" : "string"
        } ],
        "responses" : {
          "200" : {
            "description" : "200 response",
            "schema" : {
              "$ref" : "#/definitions/ArrayOfError"
            },
            "headers" : {
              "test-method-response-header" : {
                "type" : "string"
              }
            }
          }
        },
```

```
        "x-amazon-apigateway-request-validator" : "params-only",
        "x-amazon-apigateway-integration" : {
          "httpMethod" : "GET",
          "uri" : "http://petstore-demo-endpoint.execute-api.com/petstore/pets",
          "responses" : {
            "default" : {
              "statusCode" : "400",
              "responseParameters" : {
                "method.response.header.test-method-response-header" : "'static
value'"
              },
              "responseTemplates" : {
                "application/xml" : "xml 400 response template",
                "application/json" : "json 400 response template"
              }
            },
            "2\\d{2}" : {
              "statusCode" : "200"
            }
          },
          "requestParameters" : {
            "integration.request.querystring.type" : "method.request.querystring.q1"
          },
          "passthroughBehavior" : "when_no_match",
          "type" : "http"
        }
      },
      "post" : {
        "consumes" : [ "application/json" ],
        "produces" : [ "application/json", "application/xml" ],
        "parameters" : [ {
          "name" : "h1",
          "in" : "header",
          "required" : true,
          "type" : "string"
        }, {
          "in" : "body",
          "name" : "RequestBodyModel",
          "required" : true,
          "schema" : {
            "$ref" : "#/definitions/RequestBodyModel"
          }
        } ],
        "responses" : {
```

```
          "200" : {
            "description" : "200 response",
            "schema" : {
              "$ref" : "#/definitions/ArrayOfError"
            },
            "headers" : {
              "test-method-response-header" : {
                "type" : "string"
              }
            }
          }
        },
        "x-amazon-apigateway-request-validator" : "all",
        "x-amazon-apigateway-integration" : {
          "httpMethod" : "POST",
          "uri" : "http://petstore-demo-endpoint.execute-api.com/petstore/pets",
          "responses" : {
            "default" : {
              "statusCode" : "400",
              "responseParameters" : {
                "method.response.header.test-method-response-header" : "'static
value'"
              },
              "responseTemplates" : {
                "application/xml" : "xml 400 response template",
                "application/json" : "json 400 response template"
              }
            },
            "2\\d{2}" : {
              "statusCode" : "200"
            }
          },
          "requestParameters" : {
            "integration.request.header.custom_h1" : "method.request.header.h1"
          },
          "passthroughBehavior" : "when_no_match",
          "type" : "http"
        }
      }
    }
  },
  "definitions" : {
    "RequestBodyModel" : {
      "type" : "object",
```

```
      "required" : [ "name", "price", "type" ],
      "properties" : {
        "id" : {
          "type" : "integer"
        },
        "type" : {
          "type" : "string",
          "enum" : [ "dog", "cat", "fish" ]
        },
        "name" : {
          "type" : "string"
        },
        "price" : {
          "type" : "number",
          "minimum" : 25.0,
          "maximum" : 500.0
        }
      }
    },
    "ArrayOfError" : {
      "type" : "array",
      "items" : {
        "$ref" : "#/definitions/Error"
      }
    },
    "Error" : {
      "type" : "object"
    }
  },
  "x-amazon-apigateway-request-validators" : {
    "all" : {
      "validateRequestParameters" : true,
      "validateRequestBody" : true
    },
    "params-only" : {
      "validateRequestParameters" : true,
      "validateRequestBody" : false
    }
  }
}
```

Définitions OpenAPI d'un exemple d'API avec validation de demande de base

Les définitions OpenAPI suivantes concernent un exemple d'API avec une validation de demande de base activée. L'API est un sous-ensemble de l'PetStoreAPI. Elle expose une méthode `POST` pour l'ajout d'un animal de compagnie à la collection `pets` et d'une méthode `GET` pour interroger les animaux de compagnie par un type spécifié.

Deux valideurs de demande sont déclarés dans la mappe `x-amazon-apigateway-request-validators` au niveau de l'API. Le valideur `params-only` est activé sur l'API et hérité par la méthode `GET`. Ce valideur autorise API Gateway à vérifier que le paramètre de requête obligatoire (`q1`) est inclus et non vide dans la demande entrante. Le valideur `all` est activé sur la méthode `POST`. Ce valideur vérifie que le paramètre d'en-tête obligatoire (`h1`) est défini et non vide. Il vérifie également que le format de charge utile est conforme au `RequestBodyModel` spécifié. Si aucun type de contenu correspondant n'est trouvé, la validation de demande n'est pas effectuée. Lorsque vous utilisez un modèle pour valider le corps et qu'aucun type de contenu correspondant n'est trouvé, la validation de demande n'est pas effectuée. Pour utiliser le même modèle quel que soit le type de contenu, spécifiez `$default` comme clé.

Ce modèle exige que l'objet JSON en entrée contienne les propriétés `name`, `type` et `price`. La propriété `name` peut être une chaîne, `type` doit être l'un de champs d'énumération spécifiés (`["dog", "cat", "fish"]`) et `price` doit être compris entre 25 et 500. Le paramètre `id` n'est pas obligatoire.

OpenAPI 2.0

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

```
      "validateRequestParameters" : true
    },
    "params-only" : {
      "validateRequestBody" : false,
      "validateRequestParameters" : true
    }
  },
  "x-amazon-apigateway-request-validator" : "params-only",
  "paths": {
    "/validation": {
      "post": {
        "x-amazon-apigateway-request-validator" : "all",
        "parameters": [
          {
            "in": "header",
            "name": "h1",
            "required": true
          },
          {
            "in": "body",
            "name": "RequestBodyModel",
            "required": true,
            "schema": {
              "$ref": "#/definitions/RequestBodyModel"
            }
          }
        ],
        "responses": {
          "200": {
            "schema": {
              "type": "array",
              "items": {
                "$ref": "#/definitions/Error"
              }
            },
            "headers" : {
              "test-method-response-header" : {
                "type" : "string"
              }
            }
          }
        },
        "security" : [{
          "api_key" : []
```

```
        }],
        "x-amazon-apigateway-auth" : {
          "type" : "none"
        },
        "x-amazon-apigateway-integration" : {
          "type" : "http",
          "uri" : "http://petstore-demo-endpoint.execute-api.com/petstore/pets",
          "httpMethod" : "POST",
          "requestParameters": {
            "integration.request.header.custom_h1": "method.request.header.h1"
          },
          "responses" : {
            "2\\d{2}" : {
              "statusCode" : "200"
            },
            "default" : {
              "statusCode" : "400",
              "responseParameters" : {
                "method.response.header.test-method-response-header" : "'static
value'"
              },
              "responseTemplates" : {
                "application/json" : "json 400 response template",
                "application/xml" : "xml 400 response template"
              }
            }
          }
        }
      },
      "get": {
        "parameters": [
          {
            "name": "q1",
            "in": "query",
            "required": true
          }
        ],
        "responses": {
          "200": {
            "schema": {
              "type": "array",
              "items": {
                "$ref": "#/definitions/Error"
              }
```

```
            },
            "headers" : {
              "test-method-response-header" : {
                "type" : "string"
              }
            }
          }
        },
        "security" : [{
          "api_key" : []
        }],
        "x-amazon-apigateway-auth" : {
          "type" : "none"
        },
        "x-amazon-apigateway-integration" : {
          "type" : "http",
          "uri" : "http://petstore-demo-endpoint.execute-api.com/petstore/pets",
          "httpMethod" : "GET",
          "requestParameters": {
            "integration.request.querystring.type": "method.request.querystring.q1"
          },
          "responses" : {
            "2\\d{2}" : {
              "statusCode" : "200"
            },
            "default" : {
              "statusCode" : "400",
              "responseParameters" : {
                "method.response.header.test-method-response-header" : "'static
value'"
              },
              "responseTemplates" : {
                "application/json" : "json 400 response template",
                "application/xml" : "xml 400 response template"
              }
            }
          }
        }
      }
    }
  },
  "definitions": {
    "RequestBodyModel": {
      "type": "object",
```

```
      "properties": {
        "id":    { "type": "integer" },
        "type":  { "type": "string", "enum": ["dog", "cat", "fish"] },
        "name":  { "type": "string" },
        "price": { "type": "number", "minimum": 25, "maximum": 500 }
      },
      "required": ["type", "name", "price"]
    },
    "Error": {
      "type": "object",
      "properties": {

      }
    }
  }
}
```

AWS CloudFormation modèle d'un exemple d'API avec validation de demande de base

L' AWS CloudFormation exemple de définition de modèle suivant définit un exemple d'API avec la validation des demandes activée. L'API est un sous-ensemble de l'PetStoreAPI. Elle expose une méthode `POST` pour l'ajout d'un animal de compagnie à la collection `pets` et d'une méthode `GET` pour interroger les animaux de compagnie par un type spécifié.

Deux valideurs de demandes sont déclarés :

`GETValidator`

Ce valideur est activé sur la méthode `GET`. Il autorise API Gateway à vérifier que le paramètre de requête obligatoire (`q1`) est inclus et non vide dans la demande entrante.

`POSTValidator`

Ce valideur est activé sur la méthode `POST`. Il autorise API Gateway à vérifier que le format de demande de charge utile est conforme au `RequestBodyModel` spécifié quand le type de contenu est `application/json`. Si aucun type de contenu correspondant n'est trouvé, la validation de demande n'est pas effectuée. Pour utiliser le même modèle quel que soit le type de contenu, spécifiez `$default`. `RequestBodyModel` contient un modèle supplémentaire, `RequestBodyModelId`, permettant de définir l'identifiant d'animal de compagnie.

```
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}'
```

Configuration des transformations de données pour les API REST

Dans API Gateway, la demande de méthode d'une API peut accepter une charge utile dans un format différent de celui de la charge utile de la demande d'intégration. De même, le backend peut renvoyer une charge utile de réponse d'intégration différente de la charge utile de réponse de méthode. Vous pouvez mapper les paramètres de chemin d'URL, les paramètres de chaîne de requête d'URL, les en-têtes HTTP et le corps de la demande sur API Gateway à l'aide de modèles de mappage.

Un modèle de mappage est un script exprimé en langage VTL (Velocity Template Language) et appliqué à la charge utile à l'aide d'expressions JSONPath.

La charge utile peut comporter un modèle de données en fonction du schéma JSON version 4. Pour en apprendre davantage sur les modèles, consultez Comprendre les modèles de données.

> **Note**
>
> Vous n'avez pas besoin de définir de modèle pour créer un modèle de mappage, mais vous devez définir un modèle pour disposer d'API Gateway afin de générer un kit SDK ou d'activer la validation du corps de la demande pour votre API.

Rubriques

Présentation des modèles de mappage

Dans API Gateway, la demande ou la réponse de méthode d'une API peut accepter une charge utile dans un format différent de celui de la demande ou réponse d'intégration.

Vous pouvez transformer vos données pour :

- Faire correspondre la charge utile à un format spécifié par l'API.
- Remplacer les codes de statut et les paramètres de demande et réponse d'une API.
- Renvoyer les en-têtes de réponse sélectionnés par le client.
- Associez des paramètres de chemin, des paramètres de chaîne de requête ou des paramètres d'en-tête dans la demande de méthode du proxy ou du Service AWS proxy HTTP.
- Sélectionnez les données à envoyer à l'aide de l'intégration Services AWS, telles que les fonctions Amazon DynamoDB ou Lambda, ou les points de terminaison HTTP.

Vous pouvez utiliser des modèles de mappage pour transformer vos données. Un modèle de mappage est un script exprimé en langage VTL (Velocity Template Language) et appliqué à la charge utile à l'aide de JSONPath.

L'exemple suivant montre des données d'entrée, un modèle de mappage et des données de sortie pour une transformation des PetStore données.

Donnée d'entrée

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

Modèle de mappag

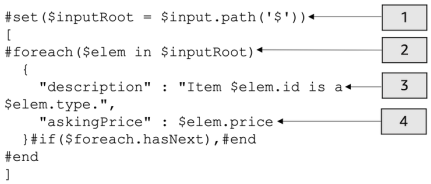
```
#set($inputRoot = $input.path('$'))
[
#foreach($elem in $inputRoot)
  {
    "description" : "Item $elem.id is a $elem.type.",
    "askingPrice" : $elem.price
  }#if($foreach.hasNext),#end

#end
]
```

Donnée de sortie

```
[
  {
    "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
  }
]
```

Le schéma suivant présente les détails de ce modèle de mappage.



1. La variable `$inputRoot` représente l'objet racine dans les données JSON d'origine de la section précédente. Les directives commencent par le symbole #.
2. Une boucle `foreach` parcourt chaque objet des données JSON d'origine.
3. La description est une concaténation des paramètres `id` et `type` de l'animal de compagnie, issus des données JSON d'origine.
4. `askingPrice` correspond au paramètre `price` de prix issu des données JSON d'origine.

PetStore modèle de mappage

```
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 ]
```

Dans ce modèle de mappage :

1. Ligne 1, la variable `$inputRoot` représente l'objet racine dans les données JSON d'origine de la section précédente. Les directives commencent par le symbole #.
2. Ligne 3, une boucle `foreach` parcourt chaque objet des données JSON d'origine.
3. Ligne 5, la `description` est une concaténation des paramètres `id` et `type` de l'animal de compagnie, issus des données JSON d'origine.
4. Ligne 6, `askingPrice` correspond au paramètre `price` de prix issu des données JSON d'origine.

Pour plus d'informations sur le langage VTL, consultez Apache Velocity - VTL Reference. Pour plus d'informations sur JSONPath, consultez JSONPath - XPath for JSON.

Le modèle de mappage suppose que les données sous-jacentes sont issues d'un objet JSON. Il n'a pas besoin qu'un modèle soit défini pour les données. Toutefois, un modèle pour les données de sortie permet de renvoyer les données précédentes sous la forme d'un objet spécifique à la langue. Pour plus d'informations, consultez Comprendre les modèles de données.

Modèles de mappage complexes

Vous pouvez aussi créer des modèles de mappage plus complexes. L'exemple suivant montre la concaténation de références et un seuil de 100 pour déterminer si un animal de compagnie est abordable.

Donnée d'entrée

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

Modèle de mappag

```
#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
]
```

Donnée de sortie

```
[
  {
    "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
  }
]
```

Vous pouvez également voir des modèles de données plus complexes. veuillez consulter Exemples de modèles de données et de modèles de mappage pour API Gateway.

Configuration de transformations de données dans API Gateway

Cette section explique comment configurer des modèles de mappage pour transformer les demandes et réponses d'intégration à l'aide de la console et de la AWS CLI.

Rubriques

Configuration d'une transformation de données à l'aide de la console API Gateway

Dans ce didacticiel, vous allez créer une API et une table DynamoDB incomplètes à l'aide du fichier .zip .zip suivant. data-transformation-tutorial-console Cette API incomplète possède une ressource `/pets` avec les méthodes `GET` et `POST`.

- La méthode `GET` obtiendra des données à partir du point de terminaison HTTP `http://petstore-demo-endpoint.execute-api.com/petstore/pets`. Les données de sortie seront transformées conformément au modèle de mappage figurant dans PetStore modèle de mappage.
- La méthode `POST` permettra à l'utilisateur de publier (`POST`) les informations sur les animaux dans une table Amazon DynamoDB à l'aide d'un modèle de mappage.

Téléchargez et décompressez le modèle de création d'application pour AWS CloudFormation. Vous allez utiliser ce modèle pour créer une table DynamoDB afin de publier les informations sur les animaux et une API incomplète. Vous terminerez les étapes restantes dans la console API Gateway.

Pour créer une AWS CloudFormation pile

1. Ouvrez la AWS CloudFormation console à l'adresse https://console.aws.amazon.com/cloudformation.
2. Choisissez Créer une pile, puis choisissez Avec de nouvelles ressources (standard).
3. Dans Spécifier le modèle, choisissez Charger un modèle de fichier.
4. Sélectionnez le modèle que vous avez téléchargé.
5. Choisissez Suivant.

6. Pour Nom de la pile, saisissez **`data-transformation-tutorial-console`**, puis choisissez Suivant.
7. Pour Configurer les options de pile, choisissez Suivant.
8. Pour les fonctionnalités, reconnaissez que AWS CloudFormation vous pouvez créer des ressources IAM dans votre compte.
9. Sélectionnez Envoyer.

AWS CloudFormation fournit les ressources spécifiées dans le modèle. La fin du provisionnement de vos ressources peut prendre quelques minutes. Lorsque le statut de votre AWS CloudFormation pile est CREATE_COMPLETE, vous êtes prêt à passer à l'étape suivante.

Pour tester la réponse d'intégration **`GET`**

1. Dans l'onglet Ressources de la AWS CloudFormation pile pour**`data-transformation-tutorial-console`**, sélectionnez l'ID physique de votre API.
2. Dans le volet de navigation principal, choisissez Ressources, puis sélectionnez la méthode GET.
3. Choisissez l’onglet Test. Vous devrez peut-être choisir la flèche droite pour afficher l'onglet.

   La sortie du test affichera les informations suivantes :

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

Vous allez transformer cette sortie conformément au modèle de mappage figurant dans PetStore modèle de mappage.

Pour transformer la réponse d'intégration **GET**

1. Choisissez l'onglet Réponse d'intégration.

   Aucun modèle de mappage n'est actuellement défini, de sorte que la réponse d'intégration ne sera pas transformée.
2. Pour Par défaut - Réponse, choisissez Modifier.
3. Choisissez Modèles de mappage, puis procédez comme suit :

   a. Sélectionnez Add mapping template.

   b. Pour Type de contenu, entrez **application/json**.

   c. Pour Corps du modèle, saisissez ce qui suit :

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

#end
]
```

   Choisissez Enregistrer.

Pour tester la réponse d'intégration **GET**

- Choisissez l'onglet Tester, puis choisissez Tester.

  La sortie du test affichera la réponse transformée.

```
[
  {
```

```
    "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
  }
]
```

Pour transformer les données d'entrée issues de la méthode **POST**

1. Choisissez la méthode POST.
2. Choisissez l’onglet Requête d’intégration, puis dans la section Paramètres de requête d’intégration, choisissez Modifier.

   Le AWS CloudFormation modèle a rempli certains champs de demande d'intégration.

   - Le type d'intégration est Service AWS.
   - Service AWS Il s'agit de DynamoDB.
   - La méthode HTTP est `POST`.
   - L'action est `PutItem`.
   - Le rôle d'exécution permettant à API Gateway de placer un élément dans la table DynamoDB est. `data-transformation-tutorial-console-APIGatewayRole` AWS CloudFormation a créé ce rôle pour permettre à API Gateway de disposer des autorisations minimales nécessaires pour interagir avec DynamoDB.

   Le nom de la table DynamoDB n'a pas été spécifié. Vous allez spécifier ce nom dans les étapes suivantes.
3. Pour Transmission du corps de requête, sélectionnez Jamais.

   Cela signifie que l'API rejettera les données avec des types de contenu qui ne disposent pas d'un modèle de mappage.
4. Choisissez Modèles de mappage.

5. Type de contenu est défini sur `application/json`. Cela signifie que tous les types de contenu autres qu'application/json seront rejetés par l'API. Pour plus d'informations sur les comportements de transmission d'intégration, consultez Comportements de transfert direct.
6. Saisissez le code suivant dans l'éditeur de texte.

```
{
    "TableName":"data-transformation-tutorial-console-ddb",
    "Item": {
        "id": {
            "N": $input.json("$.id")
        },
        "type": {
            "S": $input.json("$.type")
        },
        "price": {
            "N": $input.json("$.price")
        }
    }
}
```

   Ce modèle spécifie la table en tant que `data-transformation-tutorial-console-ddb` et définit les éléments en tant que `id`, `type` et `price`. Ces éléments proviennent du corps de la méthode `POST`. Vous pouvez également utiliser un modèle de données pour favoriser la création d'un modèle de mappage. Pour plus d'informations, consultez Utilisation de la validation des demandes dans API Gateway.
7. Choisissez Enregistrer pour enregistrer votre modèle de mappage.

Pour ajouter une réponse de méthode et d'intégration à partir de la méthode **`POST`**

Ils AWS CloudFormation ont créé une méthode vide et une réponse d'intégration. Vous allez modifier cette réponse pour fournir plus d'informations. Pour plus d'informations sur la façon de modifier des réponses, consultez Guide de référence du mappage des données de demande d'API et de réponse Amazon API Gateway.

1. Dans l'onglet Réponse d'intégration, pour Par défaut - Réponse, choisissez Modifier.
2. Choisissez Modèles de mappage, puis choisissez Ajouter un modèle de mappage.
3. Pour Type de contenu, saisissez **`application/json`**.

4. Dans l'éditeur de code, entrez le modèle de mappage de sortie suivant pour envoyer un message de sortie :

```
{ "message" : "Your response was recorded at $context.requestTime" }
```

   Pour plus d'informations sur les variables de contexte, consultez $contextVariables pour les modèles de données, les autorisateurs, les modèles de mappage et la journalisation des CloudWatch accès.

5. Choisissez Enregistrer pour enregistrer votre modèle de mappage.

Test de la méthode **POST**

Choisissez l'onglet Test. Vous devrez peut-être choisir la flèche droite pour afficher l'onglet.

1. Dans le corps de la demande, entrez l'exemple suivant.

```
{
          "id": "4",
          "type" : "dog",
          "price": "321"
}
```

2. Sélectionnez Tester).

   La sortie doit afficher votre message de réussite.

   Vous pouvez ouvrir la console DynamoDB à l'adresse https://console.aws.amazon.com/dynamodb/ pour vérifier que l'exemple d'élément figure dans votre table.

Pour supprimer une AWS CloudFormation pile

1. Ouvrez la AWS CloudFormation console à l'adresse https://console.aws.amazon.com/cloudformation.
2. Sélectionnez votre AWS CloudFormation pile.
3. Choisissez Supprimer, puis confirmez votre choix.

Configuration de la transformation des données à l'aide de la AWS CLI

Dans ce didacticiel, vous allez créer une API et une table DynamoDB incomplètes à l'aide du fichier .zip .zip suivant. data-transformation-tutorial-cli Cette API incomplète possède une ressource `/pets` avec une méthode `GET` intégrée au point de terminaison HTTP `http://petstore-demo-endpoint.execute-api.com/petstore/pets`. Vous allez créer une méthode `POST` pour vous connecter à une table DynamoDB et utiliser des modèles de mappage pour entrer des données dans une table DynamoDB.

- Vous allez transformer les données de sortie conformément au modèle de mappage figurant dans PetStore modèle de mappage.
- Vous allez créer une méthode `POST` pour permettre à l'utilisateur de publier (`POST`) les informations sur les animaux dans une table Amazon DynamoDB à l'aide d'un modèle de mappage.

Pour créer une AWS CloudFormation pile

Téléchargez et décompressez le modèle de création d'application pour AWS CloudFormation.

Pour effectuer le didacticiel suivant, vous avez besoin de l'AWS Command Line Interface (AWS CLI) version 2.

Pour les commandes longues, un caractère d'échappement (\) est utilisé pour les fractionner en plusieurs lignes.

> **Note**
>
> Dans Windows, certaines commandes CLI Bash que vous utilisez couramment (par exemple `zip`) ne sont pas prises en charge par les terminaux intégrés du système d'exploitation. Installez le sous-système Windows pour Linux afin d'obtenir une version intégrée à Windows d'Ubuntu et Bash. Les exemples de commandes CLI de ce guide utilisent le formatage Linux. Les commandes qui incluent des documents JSON en ligne doivent être reformatées si vous utilisez la CLI Windows.

1. Utilisez la commande suivante pour créer la AWS CloudFormation pile.

```
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 fournit les ressources spécifiées dans le modèle. La fin du provisionnement de vos ressources peut prendre quelques minutes. Utilisez la commande suivante pour voir l'état de votre AWS CloudFormation pile.

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

3. Lorsque le statut de votre AWS CloudFormation pile est définiStackStatus: "CREATE_COMPLETE", utilisez la commande suivante pour récupérer les valeurs de sortie pertinentes pour les étapes futures.

```
aws cloudformation describe-stacks --stack-name data-transformation-tutorial-cli
--query "Stacks[*].Outputs[*].{OutputKey: OutputKey, OutputValue: OutputValue,
Description: Description}"
```

   Les valeurs de sortie sont les suivantes :

   - ApiRole, qui est le nom du rôle qui permet à API Gateway de placer des éléments dans la table DynamoDB. Pour ce didacticiel, le nom du rôle est `data-transformation-tutorial-cli-APIGatewayRole-ABCDEFG`.
   - DDBTableName, qui est le nom de la table DynamoDB. Pour ce didacticiel, le nom de la table est `data-transformation-tutorial-cli-ddb`.
   - ResourceId, qui est l'identifiant de la ressource pour animaux de compagnie dans laquelle les `POST` méthodes `GET` et sont exposées. Pour ce didacticiel, l'ID de ressource est `efg456`.
   - ApiId, qui est l'ID de l'API. Pour ce didacticiel, l'ID d'API est `abc123`.

Pour tester la méthode **`GET`** avant la transformation des données

- Utilisez la commande suivante pour tester la méthode `GET`.

```
aws apigateway test-invoke-method --rest-api-id abc123 \
          --resource-id efg456 \
          --http-method GET
```

  La sortie du test affichera les informations suivantes.

```
[
  {
    "id": 1,
```

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

Vous allez transformer cette sortie conformément au modèle de mappage figurant dans PetStore modèle de mappage.

Pour transformer la réponse d'intégration **GET**

- Utilisez la commande suivante pour mettre à jour la réponse d'intégration de la méthode GET. Remplacez le *rest-api-id*et *resource-id* par vos valeurs.

  Utilisez la commande suivante pour créer la réponse d'intégration.

```
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]"}'
```

Pour tester la méthode **GET**

- Utilisez la commande suivante pour tester la méthode GET.

```
aws apigateway test-invoke-method --rest-api-id abc123 \
```

```
  --resource-id efg456 \
  --http-method GET \
```

La sortie du test affichera la réponse transformée.

```
[
  {
    "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
  }
]
```

Pour créer une méthode **POST**

1. Utilisez la commande suivante pour créer une nouvelle méthode sur la ressource `/pets`.

```
aws apigateway put-method --rest-api-id abc123 \
  --resource-id efg456 \
  --http-method POST \
  --authorization-type "NONE" \
```

Cette méthode vous permettra d'envoyer des informations relatives aux animaux de compagnie à la table DynamoDB que vous avez créée dans la pile. AWS CloudFormation

2. Utilisez la commande suivante pour créer une Service AWS intégration sur la `POST` méthode.

```
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. Utilisez la commande suivante pour créer une réponse de méthode pour un appel réussi de la méthode POST.

```
aws apigateway put-method-response --rest-api-id abc123 \
  --resource-id efg456 \
  --http-method POST \
  --status-code 200
```

4. Utilisez la commande suivante pour créer une réponse d'intégration pour l'appel réussi de la méthode POST.

```
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\"}"}'
```

Pour tester la méthode **POST**

- Utilisez la commande suivante pour tester la méthode POST.

```
aws apigateway test-invoke-method --rest-api-id abc123 \
  --resource-id efg456 \
  --http-method POST \
  --body '{\"id\": \"4\", \"type\": \"dog\", \"price\": \"321\"}'
```

  La sortie affichera le message de réussite.

Pour supprimer une AWS CloudFormation pile

- Utilisez la commande suivante pour supprimer vos AWS CloudFormation ressources.

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

AWS CloudFormation Modèle de transformation de données terminé

L'exemple suivant est un AWS CloudFormation modèle complet, qui crée une API et une table DynamoDB avec `/pets` une ressource `GET` avec des méthodes et. `POST`

- La méthode `GET` obtiendra des données à partir du point de terminaison HTTP `http://petstore-demo-endpoint.execute-api.com/petstore/pets`. Les données de sortie seront transformées conformément au modèle de mappage figurant dans PetStore modèle de mappage.
- La méthode `POST` permettra à l'utilisateur de publier (`POST`) les informations sur les animaux de compagnie dans une table DynamoDB à l'aide d'un modèle de mappage.

```
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
```

Étapes suivantes

Pour explorer des modèles de mappage plus complexes, consultez les exemples suivants :

- Découvrez des modèles et des modèles de mappage plus complexes avec l'exemple d'album photo Exemple d'album photo.
- Pour plus d'informations sur les modèles, consultez Comprendre les modèles de données.
- Pour obtenir des informations sur la façon de mapper différentes sorties de code de réponse, Configuration des mappages de données de demande et de réponse à l'aide de la console API Gateway.
- Pour obtenir des informations sur la façon de définir des mappages de données à partir des données de demande de méthode d'une API, Modèle de mappage API Gateway et référence à la variable de journalisation des accès.

Utiliser un modèle de mappage pour substituer les codes de statut et paramètres des requêtes et réponses d'une API

Les modèles de mappage de paramètres et de codes de réponse API Gateway standard vous permettent de mapper des paramètres one-to-one et de mapper une famille de codes d'état de réponse d'intégration (assortis d'une expression régulière) à un code d'état de réponse unique. Les remplacements de modèles de mappage vous permettent d'effectuer des mappages de many-to-one paramètres, de remplacer les paramètres une fois que les mappages API Gateway standard ont été appliqués, de mapper les paramètres de manière conditionnelle en fonction du contenu du corps ou d'autres valeurs de paramètres, de créer de nouveaux paramètres par programmation à la volée et de remplacer les codes d'état renvoyés par votre point de terminaison d'intégration. N'importe quel type de paramètre de requête, en-tête de réponse ou code de statut de réponse peut être remplacé.

Voici des exemples d'utilisation d'un remplacement par modèle de mappage :

- Pour créer un nouvel en-tête (ou remplacer un en-tête existant) sous la forme d'une concaténation de deux paramètres
- Pour remplacer le code de réponse à un code de réussite ou d'échec en fonction du contenu du corps
- Pour remapper un paramètre de façon conditionnelle, en fonction de son contenu ou du contenu d'un autre paramètre
- Pour itérer sur le contenu d'un corps json et remapper des paires clé-valeur avec les en-têtes ou chaînes d'interrogation

Pour créer un remplacement par modèle de mappage, utilisez une ou plusieurs des variables `$context` suivantes dans un modèle de mappage :

Modèle de mappage du corps d'une requête	Modèle de mappage du corps d'une réponse
`$context.requestOverride.header.` *`header_name`*	`$context.responseOverride.header.` *`header_name`*
`$context.requestOverride.path.` *`path_name`*	`$context.responseOverride.status`
`$context.requestOverride.querystring.` *`querystring_name`*	

> **Note**
>
> Les remplacements par modèle de mappage ne peuvent pas être utilisés avec les points de terminaison d'intégration de proxy, lesquels ne disposent pas de mappages de données. Pour plus d'informations sur les types d'intégration, consultez Choisir un type d'intégration d'API API Gateway.

> **Important**
>
> Les remplacement sont définitifs. Un remplacement ne peut être appliqué qu'une seule fois à chaque paramètre. Toute tentative de remplacement d'un même paramètre à plusieurs reprises se soldera par une réponse `5XX` d'Amazon API Gateway. Si vous devez remplacer le même paramètre plusieurs fois tout au long du modèle, nous vous recommandons de créer une variable et d'appliquer le remplacement à la fin du modèle. Notez que le modèle est appliqué uniquement après analyse de la totalité du modèle. veuillez consulter Tutoriel : Remplacer les paramètres de requête et les en-têtes d'une API en utilisant la console API Gateway.

Les tutoriels suivants montrent comment créer et tester un remplacement par modèle de mappage dans la console API Gateway. Ces didacticiels utilisent l'PetStore exemple d'API comme point de départ. Les deux didacticiels supposent que vous avez déjà créé l'PetStore exemple d'API.

Rubriques

- Tutoriel : Remplacer le code de statut de réponse d'une API via la console API Gateway
- Tutoriel : Remplacer les paramètres de requête et les en-têtes d'une API en utilisant la console API Gateway
- Exemples : Remplacer les paramètres et les en-têtes de requête d'une API en utilisant la CLI API Gateway
- Exemple : remplacez les paramètres de demande et les en-têtes d'une API à l'aide du SDK pour JavaScript

Tutoriel : Remplacer le code de statut de réponse d'une API via la console API Gateway

Pour récupérer un animal de compagnie à l'aide de l' PetStore exemple d'API, vous devez utiliser la méthode d'API request of`GET /pets/{petId}`, where `{petId}` est un paramètre de chemin qui peut prendre un certain nombre au moment de l'exécution.

Dans ce didacticiel, vous allez remplacer ce code de réponse de la méthode `GET` en créant un modèle de mappage qui mappe `$context.responseOverride.status` à `400` dès qu'une condition d'erreur est identifiée.

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.

2. Sous API, choisissez l' PetStore API, puis sélectionnez Ressources.
3. Dans l'arborescence Ressources, sous `GET`, choisissez la méthode sous `/{petId}`.
4. Choisissez l'onglet Test. Vous devrez peut-être choisir la flèche droite pour afficher l'onglet.
5. Pour petId, saisissez **`-1`**, puis choisissez Tester.

   Dans les résultats, vous verrez deux choses :

   Tout d'abord, le corps de la réponse indique une out-of-range erreur :

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

   Deuxièmement, la dernière ligne sous la case Journal se termine par : `Method completed with status: 200`.
6. Dans l'onglet Réponse d'intégration, pour Par défaut - Réponse, choisissez Modifier.
7. Choisissez Modèles de mappage.
8. Sélectionnez Add mapping template.
9. Pour Type de contenu, entrez **`application/json`**.
10. Pour Corps du modèle, saisissez ce qui suit :

    ```
    #set($inputRoot = $input.path('$'))
    $input.json("$")
    #if($inputRoot.toString().contains("error"))
    #set($context.responseOverride.status = 400)
    #end
    ```
11. Choisissez Enregistrer.
12. Choisissez l'onglet Test.
13. Pour petId, saisissez **`-1`**.
14. Dans les résultats, le corps de réponse indique une out-of-range erreur :

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

Cependant, la dernière ligne sous Logs (Journaux) se termine maintenant par : `Method completed with status: 400`.

Tutoriel : Remplacer les paramètres de requête et les en-têtes d'une API en utilisant la console API Gateway

Dans ce didacticiel, vous allez remplacer le code d'en-tête de la requête de méthode `GET` en créant un modèle de mappage qui mappe `$context.requestOverride.header.`*`header_name`* avec un nouvel en-tête qui combine deux autres en-têtes.

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Sous API, choisissez l' PetStore API.
3. Dans l’arborescence Ressources, sous `GET`, choisissez la méthode sous `/pet`.
4. Dans l’onglet Requête de méthode, pour Paramètres de requête de méthode, choisissez Modifier.
5. Choisissez En-têtes de demande HTTP, puis Ajouter un en-tête.
6. Pour Name (Nom), saisissez **`header1`**.
7. Choisissez Ajouter un en-tête, puis créez un second en-tête appelé **`header2`**.
8. Choisissez Enregistrer.
9. Dans l’onglet Demande d’intégration, pour Paramètres de requête d’intégration, choisissez Modifier.
10. Pour Transmission du corps de requête, sélectionnez Lorsqu’aucun modèle n’est défini (recommandé).
11. Choisissez Modèles de mappage, puis procédez comme suit :

a. Sélectionnez Add mapping template.

b. Pour Type de contenu, entrez **`application/json`**.

c. Pour Corps du modèle, saisissez ce qui suit :

```
#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. Choisissez Enregistrer.
13. Choisissez l'onglet Test.
14. Sous Headers (En-têtes) pour {pets}, copiez le code suivant :

```
header1:header1Val
header2:header2Val
```

15. Sélectionnez Test.

    Dans le journal, vous devriez voir une entrée qui inclut ce texte :

```
Endpoint request headers: {header3=header1Valheader2Val,
header2=header2Val, header1=foo, x-amzn-apigateway-api-id=<api-id>,
Accept=application/json, multivalueheader=foo,header1Valheader2Val}
```

Exemples : Remplacer les paramètres et les en-têtes de requête d'une API en utilisant la CLI API Gateway

L'exemple de CLI suivant montre comment utiliser la commande `put-integration` pour remplacer un code de réponse :

```
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>
```

où *<REQUEST_TEMPLATE_MAP>* est un mappage depuis le type de contenu vers une chaîne du modèle à appliquer. La structure de ce mappage se présente comme suit :

```
Content_type1=template_string,Content_type2=template_string
```

ou, dans la syntaxe JSON :

```
{"content_type1": "template_string"
...}
```

L'exemple suivant montre comment utiliser la commande `put-integration-response` pour remplacer le code de réponse d'une API :

```
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>
```

où *<RESPONSE_TEMPLATE_MAP>* a le même format que *<REQUEST_TEMPLATE_MAP>* ci-dessus.

Exemple : remplacez les paramètres de demande et les en-têtes d'une API à l'aide du SDK pour JavaScript

L'exemple suivant montre comment utiliser la commande `put-integration` pour remplacer un code de réponse :

Requête :

```
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
});
```

Réponse :

```
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
});
```

Configuration des mappages de données de demande et de réponse à l'aide de la console API Gateway

Pour utiliser la console API Gateway pour définir la demande/réponse d'intégration de l'API, suivez ces instructions.

> **Note**
>
> Ces instructions supposent que vous avez déjà réalisé les étapes de Configuration d'une demande d'intégration d'API à l'aide de la console API Gateway.

1. Dans le volet Ressources, choisissez votre méthode.
2. Dans l’onglet Requête d’intégration, sous Paramètres de requête d’intégration, choisissez Modifier.
3. Choisissez une option pour le transfert du corps de la requête afin de configurer la manière dont le corps de la demande de méthode d'un type de contenu non mappé sera transmis à travers la demande d'intégration sans transformation vers la fonction Lambda, le proxy HTTP ou le proxy de service. AWS Trois options sont disponibles :

   - Sélectionnez Quand aucun modèle ne correspond à l’en-tête de requête Content-Type si vous voulez que le corps de la requête de méthode soit transmis au backend via la requête d’intégration sans transformation lorsque le type de contenu de la requête de méthode ne

correspond à aucun type de contenu associé aux modèles de mappage, tel que défini à l'étape suivante.

> **Note**
>
> Lorsque vous appelez l'API API Gateway, vous sélectionnez cette option en affectant la valeur `WHEN_NO_MATCH` à la propriété `passthroughBehavior` sur la ressource d'intégration.

- Sélectionnez When there are no templates defined (recommended) si vous voulez que le corps de la demande de méthode soit transmis au backend via la demande d'intégration sans transformation lorsqu'aucun modèle de mappage n'est défini dans la demande d'intégration. Si un modèle est défini lorsque cette option est sélectionnée, la demande de méthode d'un type de contenu non mappé est rejetée en renvoyant une réponse HTTP 415 Type de support non pris en charge.

  > **Note**
  >
  > Lorsque vous appelez l'API API Gateway, vous sélectionnez cette option en affectant la valeur `WHEN_NO_TEMPLATE` à la propriété `passthroughBehavior` sur la ressource d'intégration.

- Sélectionnez Never si vous ne souhaitez pas que la demande de méthode soit transmise lorsque le type de contenu de la demande de méthode ne correspond à aucun type de contenu associé aux modèles de mappage définis dans la demande d'intégration ou lorsqu'aucun modèle de mappage n'est défini dans la demande d'intégration. La demande de méthode d'un type de contenu non mappé sera rejetée en renvoyant une réponse HTTP 415 Type de support non pris en charge.

  > **Note**
  >
  > Lorsque vous appelez l'API API Gateway, vous sélectionnez cette option en affectant la valeur `NEVER` à la propriété `passthroughBehavior` sur la ressource d'intégration.

Pour plus d'informations sur les comportements de transfert d'intégration, consultez Comportements de transfert direct.

4. Pour un proxy HTTP ou un proxy de AWS service, pour associer un paramètre de chemin, un paramètre de chaîne de requête ou un paramètre d'en-tête défini dans la demande d'intégration à un paramètre de chemin, un paramètre de chaîne de requête ou un paramètre d'en-tête correspondant dans la demande de méthode du proxy HTTP ou du proxy de AWS service, procédez comme suit :

   a. Choisissez respectivement Paramètres du chemin de l'URL, Paramètres de la chaîne de requêtes de l'URL ou En-têtes de requête HTTP, puis choisissez respectivement Ajouter un chemin, Ajouter une chaîne de requête ou Ajouter un en-tête.

   b. Pour Nom, tapez le nom du paramètre de chemin, du paramètre de chaîne de requête ou du paramètre d'en-tête dans le proxy HTTP ou le proxy AWS de service.

   c. Pour Mappage à partir de, saisissez la valeur de mappage du paramètre de chemin, du paramètre de chaîne de requête ou du paramètre d'en-tête. Utilisez l'un des formats suivants :

      - **`method.request.path.`*`parameter-name`*** pour un paramètre de chemin nommé *`parameter-name`*, tel que défini dans la page Requête de méthode.
      - **`method.request.querystring.`*`parameter-name`*** pour un paramètre de chaîne de requête nommé *`parameter-name`*, tel que défini dans la page Requête de méthode.
      - **`method.request.multivaluequerystring.`*`parameter-name`*** pour un paramètre de chaîne de requête multi-valeurs nommé *`parameter-name`*, tel que défini dans la page Requête de méthode.
      - **`method.request.header.`*`parameter-name`*** pour un paramètre d'en-tête nommé *`parameter-name`*, tel que défini dans la page Requête de méthode.

        Sinon, vous pouvez affecter une valeur de chaîne littérale (délimitée par une paire de guillemets simples) à un en-tête d'intégration.
      - **`method.request.multivalueheader.`*`parameter-name`*** pour un paramètre d'en-tête multi-valeurs nommé *`parameter-name`*, tel que défini dans la page Requête de méthode.

   d. Pour ajouter un autre paramètre, cliquez sur le bouton Ajouter.

5. Pour ajouter un modèle de mappage, choisissez Modèles de mappage.

6. Pour définir un modèle de mappage pour une requête entrante, choisissez Ajouter un modèle de mappage. Pour Type de contenu, saisissez un type de contenu (par exemple, **`application/json`**). Entrez ensuite le modèle de mappage. Pour plus d'informations, consultez Présentation des modèles de mappage.

7. Choisissez Enregistrer.
8. Vous pouvez mapper une réponse d'intégration du backend à une réponse de méthode de l'API renvoyée à l'application appelante. Cela implique de renvoyer au client les en-têtes de réponse sélectionnés à partir de ceux disponibles sur le backend, en convertissant le format de données de la charge utile de la réponse du backend dans un format spécifié par l'API. Vous pouvez définir le mappage en configurant les champs Réponse de méthode et Réponses d'intégration.

   Pour que la méthode reçoive un format de données de réponse personnalisé basé sur le code d'état HTTP renvoyé par la fonction Lambda, le proxy HTTP ou le proxy de AWS service, procédez comme suit :

   a. Choisissez Réponses d'intégration. Choisissez Modifier sur Par défaut - Réponse, pour spécifier les paramètres d'un code de réponse HTTP 200 renvoyé par la méthode, ou sélectionnez Créer une réponse pour spécifier les paramètres de tout autre code de statut de réponse HTTP renvoyé par la méthode.

   b. Pour l'expression régulière d'erreur Lambda (pour une fonction Lambda) ou l'expression régulière d'état HTTP (pour un proxy HTTP ou un proxy de AWS service), entrez une expression régulière pour spécifier quelles chaînes d'erreur de fonction Lambda (pour une fonction Lambda) ou quels codes d'état de réponse HTTP (pour un proxy HTTP ou un proxy de service) correspondent à ce mappage de sortie. AWS Par exemple, pour mapper tous les codes d'état de réponse HTTP 2xx d'un proxy HTTP à ce mappage de sortie, saisissez **`2\d{2}`** dans le champ HTTP status regex (Expressions régulières de l'erreur HTTP). Pour renvoyer un message d'erreur contenant « Requête non valide » à partir d'une fonction Lambda à une réponse `400 Bad Request`, saisissez « **`.*Invalid request.*`** » comme Expressions régulières de l'erreur Lambda. En revanche, pour renvoyer `400 Bad Request` pour tous les messages d'erreur non mappés à partir de Lambda, saisissez « **`(\n|.)+`** » dans le champ Expressions régulières de l'erreur Lambda. Cette dernière expression régulière peut être utilisée pour la réponse d'erreur par défaut d'une API.

   > **Note**
   >
   > API Gateway utilise des expressions régulières de type modèle Java pour le mappage de réponse. Pour plus d'informations, consultez la section Modèles dans la documentation Oracle.
   > Les modèles d'erreur sont mis en correspondance avec toute la chaîne de la propriété `errorMessage` dans la réponse Lambda, qui est renseignée par `callback(errorMessage)` dans Node.js ou par `throw new`

`MyException(errorMessage)` en Java. De plus, les caractères d'échappement sont sans séquence d'échappement avant l'application d'une expression régulière. Si vous utilisez « + » comme modèle de sélection pour filtrer les réponses, sachez qu'il est possible que ce modèle ne corresponde pas à une réponse contenant un caractère de nouvelle ligne.

c. Si l'option est activée, pour Statut de la réponse de méthode, sélectionnez le code de statut de réponse HTTP que vous avez défini dans la page Réponse de méthode.

d. Pour Mappages d'en-tête, pour chaque en-tête que vous avez défini pour le code de statut de réponse HTTP dans la page Réponse de méthode, spécifiez une valeur de mappage. Pour Mapping value (Valeur de mappage), utilisez l'un des formats suivants :

- **`integration.response.multivalueheaders.`*`header-name`***, où *`header-name`* est le nom d'un en-tête de réponse multi-valeurs du backend.

  Par exemple, pour renvoyer l'en-tête `Date` de la réponse du backend en tant qu'en-tête `Timestamp` de la réponse de méthode d'une API, la colonne Response header (En-tête de réponse) doit contenir une entrée Timestamp (Horodatage) et la valeur associée du champ Mapping value (Valeur de mappage) doit être integration.response.multivalueheaders.Date.

- **`integration.response.header.`*`header-name`***, où *`header-name`* est le nom d'un en-tête de réponse à valeur unique du backend.

  Par exemple, pour renvoyer l'en-tête `Date` de la réponse du backend en tant qu'en-tête `Timestamp` de la réponse de méthode d'une API, la colonne Response header (En-tête de réponse) doit contenir une entrée Timestamp (Horodatage) et la valeur associée du champ Mapping value (Valeur de mappage) doit être integration.response.header.Date.

e. Choisissez Modèles de mappage, puis choisissez Ajouter un modèle de mappage. Dans le champ Type de contenu, entrez le type de contenu des données qui seront transmises par la fonction Lambda, le proxy HTTP ou le proxy de AWS service à la méthode. Entrez ensuite le modèle de mappage. Pour plus d'informations, consultez Présentation des modèles de mappage.

f. Choisissez Enregistrer.

Exemples de modèles de données et de modèles de mappage pour API Gateway

Les sections suivantes fournissent des exemples de modèles et des modèles de mappage qui pourraient être utilisés comme point de départ pour vos propres API dans API Gateway. Pour plus d'informations sur les transformations de données, consultez Présentation des modèles de mappage. Pour plus d'informations sur les modèles de données, consultezthe section called “Comprendre les modèles de données”.

Rubriques

- Exemple d'album photo
- Exemple d'article de presse

Exemple d'album photo

L'exemple suivant montre une API d'album photo dans API Gateway. Nous fournissons un exemple de transformation des données, des modèles supplémentaires et des modèles de mappage.

Rubriques

- Exemple de transformation de données
- Modèle d'entrée pour les données photographiques
- Modèle de sortie pour les données photographiques
- Modèle de mappage d'entrée pour les données photographiques

Exemple de transformation de données

L'exemple suivant montre comment vous pouvez transformer les données d'entrée relatives à des photos à l'aide d'un modèle de mappage VTL (Velocity Template Language). Pour plus d'informations sur le langage VTL (Velocity Template Language), consultez Apache Velocity - Référence VTL (langue française non garantie).

Donnée d'entrée

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

Modèle de mappage de sortie

```
#set($inputRoot = $input.path('$'))
{
  "photos": [
#foreach($elem in $inputRoot.photos.photo)
    {
      "id": "$elem.id",
      "photographedBy": "$elem.photographer_first_name $elem.pho
tographer_last_name",
      "title": "$elem.title",
      "ispublic": $elem.ispublic,
      "isfriend": $elem.isfriend,
      "isfamily": $elem.isfamily
    }#if($foreach.hasNext),#end

#end
  ]
}
```

Donnée de sortie

```
{
  "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
    }
  ]
}
```

Modèle d'entrée pour les données photographiques

Vous pouvez définir un modèle pour vos données d'entrée. Ce modèle de saisie nécessite que vous téléchargiez une photo, et il spécifie un minimum de 10 photos pour chaque page. Vous pouvez utiliser ce modèle d'entrée pour générer un kit SDK ou pour activer la validation d'une demande pour votre API.

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

```
  }
}
```

Modèle de sortie pour les données photographiques

Vous pouvez définir un modèle pour vos données de sortie. Vous pouvez utiliser ce modèle comme modèle de réponse de méthode, ce qui s'avère nécessaire quand vous générez un kit SDK fortement typé pour l'API. Cela entraîne la conversion (cast) de la sortie en une classe appropriée en Java ou Objective-C.

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

Modèle de mappage d'entrée pour les données photographiques

Vous pouvez définir un modèle de mappage pour modifier les données d'entrée. Vous pouvez modifier les données d'entrée pour poursuivre l'intégration des fonctions ou les réponses d'intégration.

```
#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
    ]
  }
}
```

Exemple d'article de presse

L'exemple suivant montre l'API d'un article de presse dans API Gateway. Nous fournissons un exemple de transformation des données, des modèles supplémentaires et des modèles de mappage.

Rubriques

- Exemple de transformation de données
- Modèle d'entrée pour les données d'actualité
- Modèle de sortie pour les données d'actualité
- Modèle de mappage d'entrée pour les données d'actualité

Exemple de transformation de données

L'exemple suivant montre comment transformer les données d'entrée relatives à un article de presse à l'aide d'un modèle de mappage VTL (Velocity Template Language). Pour plus d'informations sur

le langage VTL (Velocity Template Language), consultez Apache Velocity - Référence VTL (langue française non garantie).

Donnée d'entrée

```
{
  "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
}
```

Modèle de mappage de sortie

```
#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
}
```

Donnée de sortie

```
{
  "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
}
```

Modèle d'entrée pour les données d'actualité

Vous pouvez définir un modèle pour vos données d'entrée. Ce modèle de saisie nécessite qu'un article de presse contienne une URL, un titre et un corps. Vous pouvez utiliser ce modèle d'entrée pour générer un kit SDK ou pour activer la validation d'une demande pour votre API.

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

Modèle de sortie pour les données d'actualité

Vous pouvez définir un modèle pour vos données de sortie. Vous pouvez utiliser ce modèle comme modèle de réponse de méthode, ce qui s'avère nécessaire quand vous générez un kit SDK fortement typé pour l'API. Cela entraîne la conversion (cast) de la sortie en une classe appropriée en Java ou Objective-C.

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

Modèle de mappage d'entrée pour les données d'actualité

Vous pouvez définir un modèle de mappage pour modifier les données d'entrée. Vous pouvez modifier les données d'entrée pour poursuivre l'intégration des fonctions ou les réponses d'intégration.

```
#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
}
```

Guide de référence du mappage des données de demande d'API et de réponse Amazon API Gateway

Cette section explique comment configurer les mappages de données de demande de méthode d'une API, y compris des autres données stockées dans les variables `context`, `stage` ou `util`, aux paramètres de demande d'intégration correspondants et les mappages de données de réponse d'intégration, y compris des autres données, aux paramètres de réponse de méthode. Les données de la demande de méthode incluent le corps et les paramètres de la demande (chemin, chaîne de requête, en-têtes). Les données de réponse d'intégration incluent les paramètres de réponse (en-têtes) et le corps. Pour plus d'informations sur l'utilisation des variables d'étape, consultez Référence des variables d'étape Amazon API Gateway.

Rubriques

- Mappage des données de demande de méthode aux paramètres de demande d'intégration
- Mappage des données de réponse d'intégration aux en-têtes de réponse de méthode
- Mappage des charges utiles de demande et de réponse entre méthode et intégration
- Comportements de transfert direct

Mappage des données de demande de méthode aux paramètres de demande d'intégration

Les paramètres de demande d'intégration, sous forme de variables de chemin, de chaînes de requête ou d'en-têtes, peuvent être mappés à partir de n'importe quels paramètres de demande de méthode définis et de la charge utile.

Dans le tableau suivant, *PARAM_NAME* est le nom d'un paramètre de demande de méthode du type de paramètre donné. Il doit respecter le modèle d'expression régulière `'^[a-zA-Z0-9._$-]+`

`$]'`. Il doit être défini avant de pouvoir être référencé. *`JSONPath_EXPRESSION`* est une expression JSONPath pour un champ JSON du corps d'une demande ou une réponse.

> **Note**
>
> Le préfixe "`$`" est omis dans cette syntaxe.

Expressions de mappage de données de demande d'intégration

Source de données mappée	Expression de mappage
Chemin de la demande de méthode	`method.request.path.` *`PARAM_NAME`*
Chaîne de requête de la demande de méthode	`method.request.querystring.` *`PARAM_NAME`*
Chaîne de requête de la demande de méthode à valeurs multiples	`method.request.multivaluequerystring.` *`PARAM_NAME`*
En-tête de la demande de méthode	`method.request.header.` *`PARAM_NAME`*
En-tête de demande de méthode à valeurs multiples	`method.request.multivalueheader.` *`PARAM_NAME`*
Corps de la demande de méthode	`method.request.body`
corps de la demande de méthode (JsonPath)	`method.request.body.` *`JSONPath_EXPRESSION`* .
Variables d'étape	`stageVariables.` *`VARIABLE_NAME`*
Variables de contexte	`context.`*`VARIABLE_NAME`* qui doit être l'une des variables de contexte prises en charge.
Valeur statique	*`'STATIC_VALUE'`* . La valeur *`STATIC_VALUE`* est un littéral de chaîne qui doit être délimité par des guillemets simples.

Example Mappages du paramètre de demande de méthode au format OpenAPI

L'exemple suivant montre un extrait de code OpenAPI qui mappe :

- l'en-tête de la demande de méthode, nommé `methodRequestHeaderParam`, au paramètre de chemin de la demande d'intégration, nommé `integrationPathParam` ;
- la chaîne de requête de la demande de méthode à valeurs multiples, nommée `methodRequestQueryParam`, à la chaîne de requête de la demande d'intégration, nommée `integrationQueryParam`.

```
...
"requestParameters" : {

    "integration.request.path.integrationPathParam" :
 "method.request.header.methodRequestHeaderParam",
    "integration.request.querystring.integrationQueryParam" :
 "method.request.multivaluequerystring.methodRequestQueryParam"

}
...
```

Les paramètres de demande d'intégration peuvent également être mappés à partir des champs du corps de la demande JSON à l'aide d'une expression JSONPath. Le tableau suivant présente les expressions de mappage pour le corps d'une demande de méthode et ses champs JSON.

Example Mappage du corps d'une demande de méthode au format OpenAPI

L'exemple suivant présente un extrait de code OpenAPI qui mappe 1) le corps d'une demande de méthode à l'en-tête de demande d'intégration, nommé `body-header`, et 2) un champ JSON du corps, spécifié par une expression JSON (`petstore.pets[0].name`, sans le préfixe `$.`).

```
...
"requestParameters" : {

    "integration.request.header.body-header" : "method.request.body",
    "integration.request.path.pet-name" : "method.request.body.petstore.pets[0].name",
```

```
}
...
```

Mappage des données de réponse d'intégration aux en-têtes de réponse de méthode

Les paramètres d'en-tête de réponse de méthode peuvent être mappés à partir de n'importe quel en-tête de réponse d'intégration ou corps de réponse d'intégration, variables, `$context`, ou valeurs statiques.

Expressions de mappage d'en-tête de réponse de méthode

Source de données mappée	Expression de mappage
En-tête de réponse d'intégration	`integration.response.header.`*`PARAM_NAME`*
En-tête de réponse d'intégration	`integration.response.multivalueheader.`*`PARAM_NAME`*
Corps de réponse intégration	`integration.response.body`
Organisme de réponse à l'intégration (JsonPath)	`integration.response.body.`*`JSONPath_EXPRESSION`*
Variable d'étape	`stageVariables.`*`VARIABLE_NAME`*
Variable de contexte	`context.`*`VARIABLE_NAME`* qui doit être l'une des variables de contexte prises en charge.
Valeur statique	*`'STATIC_VALUE'`* . La valeur *`STATIC_VALUE`* est un littéral de chaîne qui doit être délimité par des guillemets simples.

Example Mappage de données à partir d'une réponse d'intégration au format OpenAPI

L'exemple suivant présente un extrait de code OpenAPI qui mappe 1) la valeur `redirect.url` de la réponse d'intégration, le champ JSONPath dans l'en-tête `location` de la réponse de demande ; et 2) l'en-tête `x-app-id` de la réponse intégration à l'en-tête `id` de la réponse de méthode.

```
...
"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",

}
...
```

Mappage des charges utiles de demande et de réponse entre méthode et intégration

API Gateway utilise un moteur VTL (Velocity Template Language) pour traiter des modèles de mappage de corps pour la demande d'intégration et la réponse d'intégration. Les modèles de mappage convertissent les charges utiles de demande de méthode en les charges utiles de demande d'intégration correspondantes, et les corps de réponse d'intégration en corps de réponse de méthode.

Les modèles VTL utilisent des expressions JSONPath, d'autres paramètres comme des contextes d'appel et des variables d'étape, ainsi que des fonctions d'utilitaire pour traiter les données JSON.

Si un modèle est défini pour décrire la structure de données d'une charge utile, API Gateway peut utiliser le modèle pour générer un squelette de modèle de mappage pour une demande d'intégration ou une réponse d'intégration. Vous pouvez utiliser le squelette de modèle pour vous aider à personnaliser et étendre le script VTL de mappage. Cependant, vous pouvez créer un modèle de mappage à partir de zéro sans définir de modèle la structure de données de la charge utile.

Sélection d'un modèle de mappage VTL

API Gateway utilise la logique suivante pour sélectionner un modèle de mappage, dans Velocity Template Language (VTL), pour mapper la charge utile d'une demande de méthode à la demande d'intégration correspondante ou pour mapper la charge utile d'une réponse d'intégration à la réponse de méthode correspondante.

Pour une charge utile de demande, API Gateway utilise la valeur d'en-tête `Content-Type` de la demande en tant que clé pour sélectionner le modèle de mappage de la charge utile de demande. Pour une charge utile de réponse, API Gateway utilise la valeur d'en-tête `Accept` de la demande entrante en tant que clé pour sélectionner le modèle de mappage.

Lorsque l'en-tête `Content-Type` est absent dans la demande, API Gateway part du principe que sa valeur par défaut est `application/json`. Pour une telle demande, API Gateway utilise `application/json` comme clé par défaut pour sélectionner le modèle de mappage, s'il est défini. Lorsqu'aucun modèle ne correspond à cette clé, API Gateway transmet la charge utile via un contenu non mappé si la propriété passthroughBehavior est définie sur `WHEN_NO_MATCH` ou `WHEN_NO_TEMPLATES`.

Lorsque l'en-tête `Accept` n'est pas spécifié dans la demande, API Gateway part du principe que sa valeur par défaut est `application/json`. Dans ce cas, API Gateway sélectionne un modèle de mappage existant pour que `application/json` mappe la charge utile de réponse. Si aucun modèle n'est défini pour `application/json`, API Gateway sélectionne le premier modèle existant et l'utilise comme modèle par défaut pour mapper la charge utile de réponse. De même, API Gateway utilise le premier modèle existant lorsque la valeur d'en-tête `Accept` spécifiée ne correspond pas à une clé de modèle existante. Si aucun modèle n'est défini, API Gateway transmet simplement la charge utile de réponse via un contenu non mappé.

Supposons, par exemple, qu'une API a un modèle `application/json` défini pour une charge utile de demande et un modèle `application/xml` défini pour la charge utile de réponse. Si le client définit les en-têtes `"Content-Type : application/json"` et `"Accept : application/xml"` dans la demande, les charges utiles de demande et de réponse seront traitées avec les modèles de mappage correspondants. Si l'en-tête `Accept:application/xml` est absent, le modèle de mappage `application/xml` sera utilisé pour mapper la charge utile de réponse. Pour renvoyer la charge utile de réponse non mappé à la place, vous devez configurer un modèle vide pour `application/json`.

Seul le type MIME est utilisé dans les en-têtes `Accept` et `Content-Type` lors de la sélection d'un modèle de mappage. Par exemple, pour un en-tête `"Content-Type: application/json; charset=UTF-8"`, un modèle de demande avec la clé `application/json` sera sélectionné.

Comportements de transfert direct

Avec des intégrations différentes de proxy, lorsqu'une demande de méthode comporte une charge utile et que l'en-tête `Content-Type` ne correspond à aucun modèle de mappage spécifié ou qu'aucun modèle de mappage n'est défini, vous pouvez choisir de transmettre la charge utile de la demande fournie par le client via la demande d'intégration au backend sans transformation. Ce processus est appelé transfert direct d'intégration.

Pour les intégrations de proxy, API Gateway transmet la totalité de la demande à votre backend sans que vous n'ayez la possibilité de modifier les comportements de transfert.

Le comportement de transfert direct réel d'une demande entrante est déterminé par l'option que vous choisissez pour un modèle de mappage spécifique pendant la configuration de la demande d'intégration et par l'en-tête Content-Type qu'un client définit dans la demande entrante. Trois options sont disponibles :

Quand aucun modèle ne correspond à l'en-tête Content-Type de la demande

Sélectionnez cette option si vous voulez que le corps de la demande de méthode soit transmis au backend via la demande d'intégration sans transformation lorsque le type de contenu de la demande de méthode ne correspond à aucun type de contenu associé aux modèles de mappage.

Lorsque vous appelez l'API API Gateway, vous choisissez cette option en définissant `WHEN_NO_MATCH` en tant que valeur de la propriété `passthroughBehavior` sur Intégration.

Quand aucun modèle n'est défini (recommandé)

Choisissez cette option si vous voulez que le corps de la demande de méthode soit transmis au backend via la demande d'intégration sans transformation lorsqu'aucun modèle de mappage n'est défini dans la demande d'intégration. Si un modèle est défini lorsque cette option est sélectionnée, la demande de méthode d'un type de contenu non mappé est rejetée en renvoyant une réponse HTTP 415 Type de support non pris en charge.

Lorsque vous appelez l'API API Gateway, vous choisissez cette option en définissant `WHEN_NO_TEMPLATES` en tant que valeur de la propriété `passthroughBehavior` sur Intégration.

Jamais

Choisissez cette option si vous ne voulez pas que le corps de la demande de méthode soit transmis au backend via la demande d'intégration sans transformation lorsqu'aucun modèle de mappage n'est défini dans la demande d'intégration. Si un modèle est défini lorsque cette option est sélectionnée, la demande de méthode d'un type de contenu non mappé est rejetée en renvoyant une réponse HTTP 415 Type de support non pris en charge.

Lorsque vous appelez l'API API Gateway, vous choisissez cette option en définissant `NEVER` en tant que valeur de la propriété `passthroughBehavior` sur Intégration.

Les exemples suivants illustrent les comportements de transfert direct possibles.

Exemple 1 : un modèle de mappage est défini dans la demande d'intégration pour le type de contenu `application/json`.

En-tête Content-type \Option de transfert direct sélectionnée	**WHEN_NO_MATCH**	**WHEN_NO_TEMPLATES**	**NEVER**
Aucun (la valeur par défaut est `application/json`)	La charge utile de la demande est transformée à l'aide du modèle.	La charge utile de la demande est transformée à l'aide du modèle.	La charge utile de la demande est transformée à l'aide du modèle.
`application/json`	La charge utile de la demande est transformée à l'aide du modèle.	La charge utile de la demande est transformée à l'aide du modèle.	La charge utile de la demande est transformée à l'aide du modèle.
`application/xml`	La charge utile de la demande n'est pas transformée et est envoyée en l'état au backend.	La demande est rejetée avec une réponse HTTP `415 Unsupported Media Type`.	La demande est rejetée avec une réponse HTTP `415 Unsupported Media Type`.

Exemple 2 : un modèle de mappage est défini dans la demande d'intégration pour le type de contenu `application/xml`.

En-tête Content-type \Option de transfert direct sélectionnée	**WHEN_NO_MATCH**	**WHEN_NO_TEMPLATES**	**NEVER**
Aucun (la valeur par défaut est `application/json`)	La charge utile de la demande n'est pas transformée et est envoyée en l'état au backend.	La demande est rejetée avec une réponse HTTP `415 Unsupported Media Type`.	La demande est rejetée avec une réponse HTTP `415 Unsupported Media Type`.
`application/json`	La charge utile de la demande n'est pas transformée et est	La demande est rejetée avec une réponse HTTP `415`	La demande est rejetée avec une réponse HTTP `415`

En-tête Content-type \Option de transfert direct sélectionnée	**WHEN_NO_MATCH**	**WHEN_NO_TEMPLATES**	**NEVER**
	envoyée en l'état au backend.	`Unsupported Media Type.`	`Unsupported Media Type.`
`application/xml`	La charge utile de la demande est transformée à l'aide du modèle.	La charge utile de la demande est transformée à l'aide du modèle.	La charge utile de la demande est transformée à l'aide du modèle.

Modèle de mappage API Gateway et référence à la variable de journalisation des accès

Cette section fournit des informations de référence sur les variables et les fonctions définies par Amazon API Gateway pour une utilisation avec les modèles de données, les autorisateurs, les modèles de mappage et la journalisation des CloudWatch accès. Pour obtenir des informations détaillées sur l'utilisation de ces variables et fonctions, consultez Présentation des modèles de mappage. Pour plus d'informations sur le Velocity Template Language (VTL), consultez la Référence VTL.

Rubriques

- $contextVariables pour les modèles de données, les autorisateurs, les modèles de mappage et la journalisation des CloudWatch accès
- Exemple de modèle de variable $context
- Variables $context pour la journalisation des accès uniquement
- Variables $input
- Exemples de modèles de variable $input
- $stageVariables
- Variables $util

> **Note**
>
> Pour les variables `$method` et `$integration`, consultez the section called “Référence du mappage des données de demande et de réponse”.

`$context`Variables pour les modèles de données, les autorisateurs, les modèles de mappage et la journalisation des CloudWatch accès

Les `$context` variables suivantes peuvent être utilisées dans les modèles de données, les autorisateurs, les modèles de mappage et la journalisation des CloudWatch accès. API Gateway peut ajouter des variables de contexte supplémentaires.

Pour les `$context` variables qui ne peuvent être utilisées que dans la journalisation des CloudWatch accès, voirthe section called “Variables `$context` pour la journalisation des accès uniquement”.

Paramètre	Description
`$context.accountId`	ID de AWS compte du propriétaire de l'API.
`$context.apiId`	Identifiant qu'API Gateway attribue à votre API.
`$context.authorizer.claims.`*`property`*	Propriété des requêtes renvoyées depuis le groupe d'utilisateurs Amazon Cognito une fois que l'appelant de la méthode a été authentifié avec succès. Pour plus d’informations, consultez the section called “Utilisation d'un groupe d'utilisateurs Amazon Cognito en tant que mécanisme d'autorisation pour une API REST”. **Note** L'appel de `$context.authorizer.claims` renvoie la valeur null.
`$context.authorizer.principalId`	Identifiant utilisateur principal associé au jeton envoyé par le client et retourné par

Paramètre	Description
	un mécanisme d'autorisation Lambda API Gateway (anciennement appelé Custom Authorizer). Pour plus d'informations, consultez the section called "Utilisation des mécanismes d'autorisation Lambda".
`$context.authorizer.` *property*	Valeur obtenue à l'aide de stringify de la paire clé-valeur spécifiée du mappage `context` renvoyé par une fonction du mécanisme d'autorisation Lambda API Gateway. Par exemple, si le mécanisme d'autorisation retourne le mappage `context` suivant : `"context" : { "key": "value", "numKey": 1, "boolKey": true }` l'appel de `$context.authorizer.key` renvoie la chaîne `"value"`, l'appel de `$context.authorizer.numKey` renvoie la chaîne `"1"` et l'appel de `$context.authorizer.boolKey` renvoie la chaîne `"true"`. Pour plus d'informations, consultez the section called "Utilisation des mécanismes d'autorisation Lambda".
`$context.awsEndpointRequestId`	ID de demande du AWS point de terminaison.
`$context.deploymentId`	ID du déploiement de l'API.
`$context.domainName`	Nom de domaine complet utilisé pour invoquer l'API. Il doit être identique à l'en-tête `Host` entrant.

Paramètre	Description
`$context.domainPrefix`	Première étiquette de `$context.domainName` .
`$context.error.message`	Chaîne contenant un message d'erreur API Gateway. Cette variable ne peut être utilisée que pour une simple substitution de variables dans un modèle de GatewayResponsemappage corporel, qui n'est pas traité par le moteur Velocity Template Language, et dans la journalisation des accès. Pour plus d'informations, consultez the section called "Métriques" et the section called "Configuration de réponses de passerelle pour personnaliser des réponses d'erreur".
`$context.error.messageString`	La valeur entre guillemets de `$context.error.message` , à savoir `"$context.error.message"` .
`$context.error.responseType`	Un type de GatewayResponse. Cette variable ne peut être utilisée que pour une simple substitution de variables dans un modèle de GatewayResponsemappage corporel, qui n'est pas traité par le moteur Velocity Template Language, et dans la journalisation des accès. Pour plus d'informations, consultez the section called "Métriques" et the section called "Configuration de réponses de passerelle pour personnaliser des réponses d'erreur".
`$context.error.validationErrorString`	Chaîne contenant un message d'erreur de validation détaillé.

Paramètre	Description
`$context.extendedRequestId`	L'ID généré et attribué par API Gateway à la demande d'API. L'ID de requête étendu contient des informations utiles pour le débogage et le dépannage.
`$context.httpMethod`	La méthode HTTP utilisée. Les valeurs valides sont les suivantes : `DELETE`, `GET`, `HEAD`, `OPTIONS`, `PATCH`, `POST` et `PUT`.
`$context.identity.accountId`	L'ID de AWS compte associé à la demande.
`$context.identity.apiKey`	Pour les méthodes d'API qui nécessitent une clé d'API, cette variable est la clé d'API associée à la demande de méthode. Pour les méthodes qui ne nécessitent aucune clé d'API, cette variable est null. Pour plus d'informations, consultez the section called "Plans d'utilisation".
`$context.identity.apiKeyId`	ID de la clé d'API associé à une demande d'API qui nécessite une clé d'API.
`$context.identity.caller`	Identifiant principal de l'appelant qui a signé la demande. Pris en charge pour les ressources qui utilisent l'autorisation IAM.

Paramètre	Description
`$context.identity.cognitoAuthenticationProvider`	Liste séparée par des virgules des fournisseurs d'authentification Amazon Cognito utilisés par l'appelant à l'origine de la demande. Disponible uniquement si la demande a été signée avec les informations d'identification Amazon Cognito. Par exemple, pour une identité provenant d'un pool d'utilisateurs Amazon Cognito, `cognito-idp.` *`region`*`.amazonaws.com/` *`user_pool_id`* `,cognito-idp.` *`region`*`.amazonaws.com/` *`user_pool_id`* `:CognitoSignIn:` *`token subject claim`* Pour de plus amples informations, veuillez consulter Utilisation des identités fédérées dans le Manuel du développeur Amazon Cognito.
`$context.identity.cognitoAuthenticationType`	Type d'authentification Amazon Cognito de l'appelant effectuant la demande. Disponible uniquement si la demande a été signée avec les informations d'identification Amazon Cognito. Les valeurs possibles incluent `authenticated` pour les identités authentifiées et `unauthenticated` pour les identités non authentifiées.
`$context.identity.cognitoIdentityId`	ID d'identité Amazon Cognito de l'appelant effectuant la demande. Disponible uniquement si la demande a été signée avec les informations d'identification Amazon Cognito.

Paramètre	Description
`$context.identity.cognitoId entityPoolId`	ID de groupe d'identités Amazon Cognito de l'appelant effectuant la demande. Disponibl e uniquement si la demande a été signée avec les informations d'identification Amazon Cognito.
`$context.identity.principalOrgId`	ID d'organisation AWS.
`$context.identity.sourceIp`	Adresse IP source de la connexion TCP immédiate qui envoie la demande au point de terminaison API Gateway.
`$context.identity.clientCer t.clientCertPem`	Certificat client codé PEM présenté par le client lors de l'authentification TLS mutuelle. Présent lorsqu'un client accède à une API à l'aide d'un nom de domaine personnalisé pour lequel l'authentification TLS mutuelle est activée. Présent uniquement dans les journaux d'accès si l'authentification TLS mutuelle échoue.
`$context.identity.clientCer t.subjectDN`	Nom distinctif de l'objet du certificat présenté par un client. Présent lorsqu'un client accède à une API à l'aide d'un nom de domaine personnalisé pour lequel l'authentification TLS mutuelle est activée. Présent uniquement dans les journaux d'accès si l'authentification TLS mutuelle échoue.
`$context.identity.clientCer t.issuerDN`	Nom distinctif de l'émetteur du certificat présenté par un client. Présent lorsqu'un client accède à une API à l'aide d'un nom de domaine personnalisé pour lequel l'authentification TLS mutuelle est activée. Présent uniquement dans les journaux d'accès si l'authentification TLS mutuelle échoue.

Paramètre	Description
`$context.identity.clientCert.serialNumber`	Numéro de série du certificat. Présent lorsqu'un client accède à une API à l'aide d'un nom de domaine personnalisé pour lequel l'authentification TLS mutuelle est activée. Présent uniquement dans les journaux d'accès si l'authentification TLS mutuelle échoue.
`$context.identity.clientCert.validity.notBefore`	Date avant laquelle le certificat n'est pas valide. Présent lorsqu'un client accède à une API à l'aide d'un nom de domaine personnalisé pour lequel l'authentification TLS mutuelle est activée. Présent uniquement dans les journaux d'accès si l'authentification TLS mutuelle échoue.
`$context.identity.clientCert.validity.notAfter`	Date après laquelle le certificat n'est pas valide. Présent lorsqu'un client accède à une API à l'aide d'un nom de domaine personnalisé pour lequel l'authentification TLS mutuelle est activée. Présent uniquement dans les journaux d'accès si l'authentification TLS mutuelle échoue.
`$context.identity.vpcId`	L'ID VPC du VPC qui envoie la demande au point de terminaison API Gateway.
`$context.identity.vpceId`	ID du point de terminaison VPC du point de terminaison VPC qui envoie la demande au point de terminaison API Gateway. Présent uniquement lorsque vous disposez d'une API privée.
`$context.identity.user`	Identifiant principal de l'utilisateur qui sera autorisé à accéder aux ressources. Pris en charge pour les ressources qui utilisent l'autorisation IAM.

Paramètre	Description
`$context.identity.userAgent`	En-tête `User-Agent` de l'appelant d'API.
`$context.identity.userArn`	ARN (Amazon Resource Name) de l'utilisateur identifié après l'authentification. Pour plus d'informations, consultez https://docs.aws.amazon.com/IAM/latest/UserGuide/id_users.html.
`$context.isCanaryRequest`	Renvoie `true` si la demande était dirigée vers le canari et `false` si la demande n'était pas dirigée vers le canari. Présent uniquement lorsque vous avez activé un Canary.
`$context.path`	Chemin d'accès de la demande. Par exemple, pour une URL de demande autre que de proxy de `https://{rest-api-id}.execute-api.{region}.amazonaws.com/{stage}/root/child`, la valeur `$context.path` est `/{stage}/root/child`.
`$context.protocol`	Protocole de demande, par exemple, `HTTP/1.1`. **Note** Les API d'API Gateway peuvent accepter les requêtes HTTP/2, mais API Gateway envoie les requêtes aux intégrations backend en utilisant HTTP/1.1. Par conséquent, le protocole de requête est enregistré comme HTTP/1.1 même si un client envoie une requête qui utilise HTTP/2.

Paramètre	Description
`$context.requestId`	Un ID pour la demande. Les clients peuvent remplacer cet ID de demande. Utiliser `$context.extendedRequestId` pour un ID de demande unique généré par API Gateway.
`$context.requestOverride.header. header_name`	Remplacement de l'en-tête de la requête. Si ce paramètre est défini, il contient les en-têtes à utiliser à la place des HTTP Headers (En-têtes HTTP) qui sont définis dans le volet Integration Request (Demande d'intégration). Pour plus d'informations, consultez Utiliser un modèle de mappage pour substituer les codes de statut et paramètres des requêtes et réponses d'une API.
`$context.requestOverride.path. path_name`	Remplacement du chemin de la requête. Si ce paramètre est défini, il contient le chemin de requête à utiliser à la place des URL Path Parameters (Paramètres de chemin d'URL) qui sont définis dans le volet Integration Request (Demande d'intégration). Pour plus d'informations, consultez Utiliser un modèle de mappage pour substituer les codes de statut et paramètres des requêtes et réponses d'une API.

Paramètre	Description
`$context.requestOverride.querystring.` *querystring_name*	Remplacement de la chaîne d'interrogation de la requête. Si ce paramètre est défini, il contient les chaînes d'interrogation de la requête à utiliser à la place des URL Query String Parameters (Paramètres de chaîne de requête d'URL) qui sont définis dans le volet Integration Request (Demande d'intégration). Pour plus d'informations, consultez Utiliser un modèle de mappage pour substituer les codes de statut et paramètres des requêtes et réponses d'une API.
`$context.responseOverride.header.` *header_name*	Remplacement de l'en-tête de la réponse. Si ce paramètre est défini, il contient l'en-tête qui est renvoyé à la place du Response header (En-tête de réponse) qui est défini comme le Default mapping (Mappage par défaut) dans le volet Integration Response (Réponse d'intégration). Pour plus d'informations, consultez Utiliser un modèle de mappage pour substituer les codes de statut et paramètres des requêtes et réponses d'une API.
`$context.responseOverride.status`	Remplacement du code de statut de la réponse. Si ce paramètre est défini, il contient le code du statut qui est renvoyé à la place du Method response status (Statut de la réponse de méthode) qui est défini comme le Default mapping (Mappage par défaut) dans le volet Integration Response (Réponse d'intégration). Pour plus d'informations, consultez Utiliser un modèle de mappage pour substituer les codes de statut et paramètres des requêtes et réponses d'une API.

Paramètre	Description
`$context.requestTime`	Durée des demandes au format CLF (`dd/MMM/yyyy:HH:mm:ss +-hhmm`).
`$context.requestTimeEpoch`	Heure de la demande au format Epoch, en millisecondes.
`$context.resourceId`	Identifiant attribué par API Gateway à votre ressource.
`$context.resourcePath`	Chemin de votre ressource. Par exemple, pour l'URI de demande autre que de proxy de `https://{rest-api-id}.execute-api.{region}.amazonaws.com/{stage}/root/child` , la valeur `$context.resourcePath` est `/root/child` . Pour plus d'informations, consultez Tutoriel : Création d'une API REST avec une intégration HTTP autre que de proxy.
`$context.stage`	Étape de déploiement de la demande d'API (par exemple, `Beta` ou `Prod`).
`$context.wafResponseCode`	Réponse reçue de AWS WAF: `WAF_ALLOW` ou `WAF_BLOCK` . N'est pas défini si l'étape n'est pas associée à une liste ACL web. Pour plus d'informations, consultez the section called "AWS WAF".
`$context.webaclArn`	ARN complet de la liste ACL web utilisée pour déterminer s'il convient d'autoriser ou de bloquer la demande. N'est pas défini si l'étape n'est pas associée à une liste ACL web. Pour plus d'informations, consultez the section called "AWS WAF".

Exemple de modèle de variable `$context`

Vous pouvez utiliser des variables `$context` dans un modèle de mappage si votre méthode d'API transmet des données structurées à un backend qui nécessite que les données aient un format particulier.

L'exemple suivant illustre un modèle de mappage qui mappe les variables `$context` entrantes aux variables de backend avec des noms légèrement différents dans une charge utile de la demande d'intégration :

> **Note**
>
> L'une des variables est une clé d'API. Cet exemple suppose que la méthode nécessite une clé d'API.

```
{
    "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"
}
```

La sortie de ce modèle de mappage doit ressembler à ce qui suit :

```
{
  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'
}
```

Variables **`$context`** pour la journalisation des accès uniquement

Les variables `$context` ci-après sont disponibles uniquement pour la journalisation des accès Pour plus d'informations, consultez the section called "CloudWatch journaux". (Pour les WebSocket API, voirthe section called "Métriques".)

Paramètre	Description
`$context.authorize.error`	Message d'erreur d'autorisation.
`$context.authorize.latency`	Latence d'autorisation en millisecondes.
`$context.authorize.status`	Code d'état renvoyé à la suite d'une tentative d'autorisation.
`$context.authorizer.error`	Message d'erreur renvoyé par un mécanisme d'autorisation.
`$context.authorizer.integrationLatency`	Latence du mécanisme d'autorisation en millisecondes (ms).
`$context.authorizer.integrationStatus`	Code d'état renvoyé par un mécanisme d'autorisation Lambda.
`$context.authorizer.latency`	Latence du mécanisme d'autorisation en millisecondes (ms).
`$context.authorizer.requestId`	ID de demande du AWS point de terminaison.
`$context.authorizer.status`	Code d'état renvoyé par un mécanisme d'autorisation.

Paramètre	Description
`$context.authenticate.error`	Message d'erreur renvoyé à la suite d'une tentative d'authentification.
`$context.authenticate.latency`	Latence d'authentification en millisecondes.
`$context.authenticate.status`	Code d'état renvoyé à la suite d'une tentative d'authentification.
`$context.customDomain.basePathMatched`	Chemin d'accès d'un mappage d'API correspondant à une demande entrante. Applicable lorsqu'un client utilise un nom de domaine personnalisé pour accéder à une API. Par exemple, si un client envoie une demande à `https://api.example.com/v1/orders/1234` et que cette demande correspond au mappage d'API dont le chemin d'accès est `v1/orders`, la valeur est `v1/orders`. Pour en savoir plus, consultez la section the section called “Mappages d'API”.
`$context.endpointType`	Type de point de terminaison de l'API.
`$context.integration.error`	Message d'erreur renvoyé à partir d'une intégration.
`$context.integration.integrationStatus`	Pour l'intégration du proxy Lambda, le code d'état est renvoyé par le code de fonction Lambda principal AWS Lambda, et non par le code de fonction Lambda principal.
`$context.integration.latency`	Latence d'intégration en millisecondes (ms). Équivalent à `$context.integrationLatency`.

Paramètre	Description
`$context.integration.requestId`	ID de demande du AWS point de terminaison. Équivalent à `$context.awsEndpointRequestId` .
`$context.integration.status`	Code d'état renvoyé à partir d'une intégration. Pour les intégrations de proxy Lambda, code d'état que votre code de fonction Lambda renvoie.
`$context.integrationLatency`	Latence d'intégration en millisecondes (ms).
`$context.integrationStatus`	Pour l'intégration du proxy Lambda, ce paramètre représente le code d'état renvoyé par le code de fonction Lambda principal AWS Lambda, et non par le code de fonction Lambda principal.
`$context.responseLatency`	Latence de la réponse en millisecondes (ms).
`$context.responseLength`	La longueur de la charge utile de la réponse en octets.
`$context.status`	Statut de la réponse de la méthode.
`$context.waf.error`	Le message d'erreur renvoyé par AWS WAF.
`$context.waf.latency`	La AWS WAF latence en ms.
`$context.waf.status`	Le code d'état renvoyé par AWS WAF.
`$context.xrayTraceId`	ID du suivi X-Ray. Pour plus d'informations, consultez the section called “Con AWS X-Ray figuration”.

Variables **`$input`**

La variable `$input` représente la charge utile de la demande de méthode et les paramètres devant être traités par un modèle de mappage. Il fournit les fonctions suivantes :

Variable et fonction	Description
`$input.body`	Renvoie la charge utile de la demande brute sous forme de chaîne.
`$input.json(x)`	Cette fonction évalue une expression JSONPath et renvoie les résultats sous la forme une chaîne JSON. Par exemple, `$input.json('$.pets')` renvoie une chaîne JSON représentant la structure `pets`. Pour plus d'informations sur JSONPath, consultez JSONPath ou JSONPath pour Java.
`$input.params()`	Renvoie une carte de tous les paramètres de demande. Nous vous recommandons d'utiliser `$util.escapeJavaScript` pour désinfecter le résultat afin d'éviter une attaque potentielle par injection. Pour un contrôle total de la désinfection des requêtes, utilisez une intégration proxy sans modèle et gérez la désinfection des requêtes dans votre intégration.
`$input.params(x)`	Renvoie la valeur d'un paramètre de demande de méthode à partir du chemin, de la chaîne de requête ou d'une valeur d'en-tête (dans cet ordre) avec une chaîne de nom de paramètre `x`. Nous vous recommandons d'utiliser `$util.escapeJavaScript` pour désinfecter le paramètre afin d'éviter une attaque potentielle par injection. Pour un contrôle total de la désinfection des paramètre

Variable et fonction	Description
	s, utilisez une intégration proxy sans modèle et gérez la désinfection des requêtes dans votre intégration.
`$input.path(x)`	Prend une chaîne d'expression JSONPath (x) et renvoie une représentation d'objet JSON du résultat. Cela vous permet d'accéder aux éléments de la charge utile et de les manipuler en mode natif en langage VTL (Apache Velocity Template Language). Par exemple, si l'expression `$input.path('$.pets')` renvoie un objet comme suit : `[ { "id": 1, "type": "dog", "price": 249.99 }, { "id": 2, "type": "cat", "price": 124.99 }, { "id": 3, "type": "fish", "price": 0.99 } ]` `$input.path('$.pets').count()` renvoie `"3"`. Pour plus d'informations sur JSONPath, consultez JSONPath ou JSONPath pour Java.

Exemples de modèles de variable **`$input`**

Les exemples suivants montrent comment utiliser les `$input` variables dans les modèles de mappage. Vous pouvez utiliser une intégration fictive ou une intégration Lambda sans proxy qui renvoie l'événement d'entrée à API Gateway pour essayer ces exemples.

Exemple de modèle de mappage de paramètres

L'exemple suivant transmet tous les paramètres de demande, y compris `pathquerystring`, et`header`, au point de terminaison d'intégration via une charge utile JSON :

```
#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
  }
}
```

Pour une demande qui inclut les paramètres d'entrée suivants :

- Un paramètre de chemin nommé `myparam`
- Paramètres de chaîne de requête `querystring1=value1,value2&querystring2=value3`
- En-têtes `"header1" : "value1""header2" : "value2","header3" : "value3"`.

La sortie de ce modèle de mappage doit ressembler à ce qui suit :

```
{
  "params" : {
        "path" : {
            "path" : "myparam"
                }
    ,        "querystring" : {
```

```
            "querystring1" : "value1,value2"
      ,           "querystring2" : "value3"
               }
    ,         "header" : {
            "header3" : "value3"
      ,           "header2" : "value2"
      ,           "header1" : "value1"
               }
          }
}
```

Exemple de modèle de mappage JSON

Vous pouvez utiliser la variable `$input` pour obtenir les chaînes de requête et le corps de la demande avec ou sans l'aide de modèles. Vous souhaiterez peut-être également obtenir le paramètre et la charge utile, ou une sous-section de la charge utile. Les trois exemples suivants montrent comment procéder.

L'exemple suivant utilise un modèle de mappage pour obtenir une sous-section de la charge utile. Cet exemple permet d'obtenir le paramètre d'entrée, `name` puis l'intégralité du corps du POST :

```
{
    "name" : "$input.params('name')",
    "body" : $input.json('$')
}
```

Pour une demande qui inclut les paramètres de la chaîne de requête `name=Bella&type=dog` et le corps suivant :

```
{
    "Price" : "249.99",
    "Age": "6"
}
```

La sortie de ce modèle de mappage doit ressembler à ce qui suit :

```
{
    "name" : "Bella",
    "body" : {"Price":"249.99","Age":"6"}
}
```

Si l'entrée JSON contient des caractères non échappés qui ne peuvent pas être analysés JavaScript, API Gateway peut renvoyer une réponse 400. Appliquez `$util.escapeJavaScript($input.json('$'))` pour vous assurer que l'entrée JSON peut être analysée correctement.

L'exemple précédent avec `$util.escapeJavaScript($input.json('$'))` appliqué est le suivant :

```
{
    "name" : "$input.params('name')",
    "body" : $util.escapeJavaScript($input.json('$'))
}
```

Dans ce cas, le résultat de ce modèle de mappage doit ressembler à ce qui suit :

```
{
    "name" : "Bella",
    "body": {\"Price\":\"249.99\",\"Age\":\"6\"}
}
```

Exemple d'expression JSONPath

L'exemple suivant montre comment transmettre une expression JSONPath à la méthode `json()`. Vous pouvez également lire une sous-section de l'objet du corps de votre demande en utilisant un `.` point, pour spécifier une propriété :

```
{
    "name" : "$input.params('name')",
    "body" : $input.json('$.Age')
}
```

Pour une demande qui inclut les paramètres de la chaîne de requête `name=Bella&type=dog` et le corps suivant :

```
{
    "Price" : "249.99",
    "Age": "6"
}
```

La sortie de ce modèle de mappage doit ressembler à ce qui suit :

```
{
    "name" : "Bella",
    "body" : "6"
}
```

Si la charge utile d'une demande de méthode contient des caractères non échappés qui ne peuvent pas être analysés, API JavaScript Gateway peut renvoyer une réponse. `400` Appliquez `$util.escapeJavaScript()` pour vous assurer que l'entrée JSON peut être analysée correctement.

L'exemple précédent avec `$util.escapeJavaScript($input.json('$.Age'))` appliqué est le suivant :

```
{
    "name" : "$input.params('name')",
    "body" : "$util.escapeJavaScript($input.json('$.Age'))"
}
```

Dans ce cas, le résultat de ce modèle de mappage doit ressembler à ce qui suit :

```
{
    "name" : "Bella",
    "body": "\"6\""
}
```

Exemple de demande et de réponse

L'exemple suivant utilise `$input.params()``$input.path()`, et `$input.json()` pour une ressource dont le chemin est le suivant `/things/{id}` :

```
{
    "id" : "$input.params('id')",
    "count" : "$input.path('$.things').size()",
    "things" : $input.json('$.things')"
}
```

Pour une demande qui inclut le paramètre path `123` et le corps suivant :

```
{
      "things": {
            "1": {},
```

```
            "2": {},
            "3": {}
      }
}
```

La sortie de ce modèle de mappage doit ressembler à ce qui suit :

```
{"id":"123","count":"3","things":{"1":{},"2":{},"3":{}}}
```

Si la charge utile d'une demande de méthode contient des caractères non échappés qui ne peuvent pas être analysés, API JavaScript Gateway peut renvoyer une réponse. `400` Appliquez `$util.escapeJavaScript()` pour vous assurer que l'entrée JSON peut être analysée correctement.

L'exemple précédent avec `$util.escapeJavaScript($input.json('$.things'))` appliqué est le suivant :

```
{
     "id" : "$input.params('id')",
     "count" : "$input.path('$.things').size()",
     "things" : "$util.escapeJavaScript($input.json('$.things'))"
}
```

La sortie de ce modèle de mappage doit ressembler à ce qui suit :

```
{"id":"123","count":"3","things":"{\"1\":{},\"2\":{},\"3\":{}}"}
```

Pour plus d'exemples de mappage, consultez Présentation des modèles de mappage.

`$stageVariables`

Les variables d'étape peuvent être utilisées dans le mappage des paramètres et les modèles de mappage, et comme espaces réservés dans les ARN et les URL utilisés dans les intégrations de méthode. Pour plus d’informations, consultez the section called “Configurer des variables d'étape”.

Syntaxe	Description
`$stageVariables.` *`<variable_name>`* , `$stageVariables['` *`<variable`*	*`<variable_name>`* représente un nom de variable d'étape.

Syntaxe	Description
_name> '] ou ${stageVariables[' *<variable_name>* ']}	

Variables **$util**

La variable $util contient les fonctions utilitaires à utiliser dans les modèles de mappage.

> **Note**
>
> Sauf indication contraire, le jeu de caractères par défaut est UTF-8.

Fonction	Description
$util.escapeJavaScript()	Échape les caractères d'une chaîne en utilisant les règles relatives aux JavaScript chaînes. **Note** Cette fonction convertit tout guillemet simple (') en guillemet d'échappement (\'). Cependant, les guillemets simples d'échappement ne sont pas valides en JSON. Par conséquent, lorsque la sortie de cette fonction est utilisée dans une propriété JSON, vous devez reconvertir les guillemets simples d'échappement (\') en guillemets simples ('), comme illustré dans l'exemple suivant : `"input" : "$util.escapeJavaScript( data).replaceAll("\\'","'")"`

Fonction	Description

`$util.parseJson()`	Prend la chaîne JSON (obtenue à l'aide de stringify) et renvoie une représentation objet du résultat. Vous pouvez utiliser le résultat de cette fonction pour accéder aux éléments de la charge utile et les manipuler en mode natif en langage VTL (Apache Velocity Template Language). Par exemple, si vous avez la charge utile suivante :
	`{"errorMessage":"{\"key1\":\"var1\", \"key2\":{\"arr\":[1,2,3]}}"}`
	et utilisez le modèle de mappage suivant :
	`#set ($errorMessageObj = $util.parseJson($input.path('$.errorMessage')))`
	`{`
	`"errorMessageObjKey2ArrVal" : $errorMessageObj.key2.arr[0]`
	`}`
	vous obtenez la sortie suivante :
	`{`
	`"errorMessageObjKey2ArrVal" : 1`
	`}`

`$util.urlEncode()`	Convertit une chaîne au format « application/ x-www-form-urlencoded ».
`$util.urlDecode()`	Décode une chaîne « application/ x-www-form-urlencoded ».
`$util.base64Encode()`	Encode les données dans une chaîne encodée en base64.

Fonction	Description
`$util.base64Decode()`	Décode les données d'une chaîne encodée en base64.

Réponses de passerelle dans API Gateway

Un réponse de passerelle est identifiée par un type de réponse défini par API Gateway. La réponse se compose d'un code de statut HTTP, d'un ensemble d'en-têtes supplémentaires qui sont spécifiés par des mappages de paramètres et une charge utile qui est générée par un modèle de mappage non-VTL.

Dans l'API REST API Gateway, une réponse de passerelle est représentée par le GatewayResponse. Dans OpenAPI, une `GatewayResponse` instance est décrite par l'extension x-amazon-apigateway-gateway-Respones.gatewayResponse.

Pour activer une réponse de passerelle, vous configurez une réponse de passerelle pour un type de réponse pris en charge au niveau de l'API. Chaque fois qu'API Gateway renvoie une réponse de ce type, les mappages d'en-tête et les modèles de mappage de charge utile définis dans la réponse de passerelle sont appliqués pour renvoyer les résultats mappés à l'appelant de l'API.

Dans la section suivante, nous allons voir comment configurer des réponses de passerelle à l'aide de la console API Gateway et de l'API REST API Gateway.

Configuration de réponses de passerelle pour personnaliser des réponses d'erreur

Si API Gateway ne parvient pas à traiter une demande entrante, le service renvoie au client une réponse d'erreur sans transmettre la demande au backend d'intégration. Par défaut, la réponse d'erreur contient un court message d'erreur descriptif. Par exemple, si vous tentez d'appeler une opération sur une ressource d'API non définie, vous recevez une réponse d'erreur avec le message `{ "message": "Missing Authentication Token" }`. Si vous n'êtes pas encore familiarisé avec API Gateway, il peut être difficile de comprendre ce qui s'est réellement passé.

Pour certaines des réponses d'erreur, API Gateway autorise les développeurs d'API à effectuer une personnalisation pour renvoyer les réponses dans des formats différents. Pour l'exemple `Missing Authentication Token`, vous pouvez ajouter un conseil à la charge utile de réponse d'origine avec les causes possibles, comme dans cet exemple : `{"message":"Missing Authentication Token", "hint":"The HTTP method or resources may not be supported."}`.

Lorsque votre API assure la médiation entre un échange externe et le AWS Cloud, vous utilisez des modèles de mappage VTL pour les demandes d'intégration ou les réponses d'intégration afin de mapper la charge utile d'un format à un autre. Toutefois, les modèles de mappage VTL fonctionnent uniquement pour les demandes valides avec des réponses positives.

Pour les demandes non valides, API Gateway contourne complètement l'intégration et renvoie une réponse d'erreur. Vous devez utiliser la personnalisation pour rendre les réponses d'erreur dans un format conforme pour l'échange. Ici, la personnalisation est rendue dans un modèle de mappage non-VTL prenant en charge uniquement les substitutions de variables simples.

Pour désigner de manière générale les réponses d'erreur générées par API Gateway à des réponses générées par API Gateway, nous les appelons réponses de passerelle. Cela permet de distinguer les réponses générées par API Gateway des réponses d'intégration. Un modèle de mappage de réponse de passerelle peut accéder à des valeurs de variable `$context` et des valeurs de propriété `$stageVariables`, ainsi qu'à des paramètres de demande de méthode, sous la forme `method.request.`*`param-position`*`.`*`param-name`*.

Pour plus d'informations sur les variables `$context`, consultez $contextVariables pour les modèles de données, les autorisateurs, les modèles de mappage et la journalisation des CloudWatch accès. Pour plus d'informations sur `$stageVariables`, consultez `$stageVariables`. Pour plus d'informations sur les paramètres de demande de méthode, reportez-vous à la section the section called “Variables `$input`”.

Rubriques

- Configurer une réponse de passerelle pour une API REST à l'aide de la console API Gateway
- Configuration d'une réponse de passerelle à l'aide de l'API REST API Gateway
- Configuration d'une personnalisation de réponse de passerelle dans OpenAPI
- Types de réponses de passerelle

Configurer une réponse de passerelle pour une API REST à l'aide de la console API Gateway

Pour personnaliser une réponse de passerelle à l'aide de la console API Gateway

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Choisissez une API REST.

3. Dans le panneau de navigation principal, choisissez Réponses de la passerelle.
4. Choisissez un type de réponse, puis choisissez Modifier. Dans cette procédure, nous allons utiliser Jeton d'authentification manquant comme exemple.
5. Vous pouvez modifier le Code de statut généré par API Gateway pour renvoyer un autre code de statut qui répond aux exigences de votre API. Dans cet exemple, la personnalisation modifie le code de statut de la valeur par défaut (`403`) à `404`, car ce message d'erreur est émis lorsqu'un client appelle une ressource non prise en charge ou non valide qui peut être considérée comme non trouvée.
6. Pour renvoyer des en-têtes personnalisés, choisissez Ajouter un en-tête de réponse sous En-têtes de réponse. A titre d'illustration, nous allons ajouter les en-têtes personnalisés suivants :

```
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
```

   Dans les mappages d'en-tête précédents, un nom de domaine statique (`'a.b.c'`) est mappé à l'en-tête `Allow-Control-Allow-Origin` pour autoriser l'accès CORS à l'API, l'en-tête de demande d'entrée de `x-amzn-RequestId` est mappé à `request-id` dans la réponse, la variable de chemin `petId` de la demande entrante est mappée à l'en-tête `request-path` dans la réponse et le paramètre de requête q de la demande d'origine est mappé à l'en-tête `request-query` de la réponse.
7. Sous Modèles de réponse, conservez `application/json` pour le Type de contenu et saisissez le modèle de mappage de corps suivant dans l'éditeur Modèle de mappage de corps :

```
{
     "message":"$context.error.messageString",
     "type": "$context.error.responseType",
     "statusCode": "'404'",
     "stage": "$context.stage",
     "resourcePath": "$context.resourcePath",
     "stageVariables.a": "$stageVariables.a"
}
```

   Cet exemple montre comment mapper les propriétés `$context` et `$stageVariables` aux propriétés du corps de demande de passerelle.
8. Sélectionnez Enregistrer les modifications.

9. Déployez l'API dans une étape nouvelle ou existante.

Testez la réponse de votre passerelle en appelant la commande CURL suivante, en supposant que l'URL d'invocation de la méthode d'API correspondante est `https://o81lxisefl.execute-api.us-east-1.amazonaws.com/custErr/pets/{petId}` :

```
curl -v -H 'x-amzn-RequestId:123344566' https://o81lxisefl.execute-api.us-
east-1.amazonaws.com/custErr/pets/5/type?q=1
```

Comme le paramètre de chaîne de requête supplémentaire q=1 n'est pas compatible avec l'API, une erreur est renvoyée pour déclencher la réponse de passerelle spécifiée. Vous devriez obtenir une réponse de passerelle semblable à ce qui suit :

```
> 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
}
```

L'exemple précédent suppose que le serveur principal d'API est Pet Store et que l'API a une variable d'étape a définie.

Configuration d'une réponse de passerelle à l'aide de l'API REST API Gateway

Avant de personnaliser une réponse de passerelle à l'aide de l'API REST API Gateway, vous devez avoir déjà créé une API et obtenu son identifiant. Pour extraire l'identifiant de l'API, vous pouvez suivre la relation de lien restapi:gateway-responses et examiner le résultat.

Pour personnaliser une réponse de passerelle à l'aide de l'API REST API Gateway

1. Pour remplacer une GatewayResponseinstance entière, appelez l'action gatewayresponse:put. Spécifiez la valeur souhaitée pour responseType dans le paramètre de chemin d'URL et indiquez dans la charge utile de la requête les mappages statusCode, responseParameters et responseTemplates.
2. Pour mettre à jour une partie d'une instance `GatewayResponse`, appelez l'action GatewayResponse:update. Spécifiez une valeur `responseType` souhaitée dans le paramètre de chemin d'URL et indiquez dans la charge utile de la requête les propriétés `GatewayResponse` individuelles que vous souhaitez, par exemple, le mappage `responseParameters` ou `responseTemplates`.

Configuration d'une personnalisation de réponse de passerelle dans OpenAPI

Vous pouvez utiliser l'extension `x-amazon-apigateway-gateway-responses` au niveau racine de l'API pour personnaliser des réponses de passerelle dans OpenAPI. La définition OpenAPI suivante montre un exemple de personnalisation GatewayResponsedu type. `MISSING_AUTHENTICATION_TOKEN`

```
"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}"
      }
    }
```

Dans cet exemple, la personnalisation remplace la valeur par défaut du code du statut (`403`) par `404`. La personnalisation ajoute également à la réponse de passerelle quatre paramètres d'en-tête et un modèle de mappage de corps pour le type de média `application/json`.

Types de réponses de passerelle

API Gateway expose les réponses de passerelle suivantes pour personnalisation par les développeurs d'API.

Type de réponse de passerelle	Code de statut par défaut	Description
`ACCESS_DENIED`	`403`	Réponse de passerelle pour un échec d'autorisation, par exemple, lorsque l'accès est refusé par un mécanisme d'autorisation personnalisée ou Amazon Cognito. Si le type de réponse n'est pas spécifié, cette réponse est définie par défaut sur le type `DEFAULT_4XX` .
`API_CONFIGURATION_ERROR`	`500`	Réponse de passerelle pour une configuration d'API non valide, y compris une adresse de point de terminaison non valide soumise, l'échec du décodage base64 sur des données binaires lorsque la prise en charge binaire est activée, ou un mappage de réponse d'intégration qui ne

Type de réponse de passerelle	Code de statut par défaut	Description
		peut correspondre à aucun modèle alors qu'aucun modèle par défaut n'est configuré. Si le type de réponse n'est pas spécifié, cette réponse est définie par défaut sur le type `DEFAULT_5XX` .
`AUTHORIZER_CONFIGURATION_ERROR`	`500`	Réponse de passerelle pour un échec de connexion à un mécanisme d'autorisation personnalisée ou Amazon Cognito. Si le type de réponse n'est pas spécifié, cette réponse est définie par défaut sur le type `DEFAULT_5XX` .
`AUTHORIZER_FAILURE`	`500`	Réponse de passerelle quand un mécanisme d'autorisation personnalisée ou Amazon Cognito n'a pas pu authentifier l'appelant. Si le type de réponse n'est pas spécifié, cette réponse est définie par défaut sur le type `DEFAULT_5XX` .

Type de réponse de passerelle	Code de statut par défaut	Description
BAD_REQUEST_PARAMETERS	400	Réponse de passerelle quand le paramètre de demande ne peut pas être validé selon un valideur de demande activé. Si le type de réponse n'est pas spécifié, cette réponse est définie par défaut sur le type DEFAULT_4XX .
BAD_REQUEST_BODY	400	Réponse de passerelle quand le corps de demande ne peut pas être validé selon un valideur de demande activé. Si le type de réponse n'est pas spécifié, cette réponse est définie par défaut sur le type DEFAULT_4XX .

Type de réponse de passerelle	Code de statut par défaut	Description
DEFAULT_4XX	Null	Réponse de passerelle par défaut pour un type de réponse non spécifié avec le code de statut 4XX. La modification du code de statut de cette réponse de passerelle de rechange modifie les codes de statut de toutes les autres réponses 4XX sur une nouvelle valeur. La réinitialisation de ce code de statut à null rétablit les valeurs d'origine des codes de statut de toutes les autres réponses 4XX. **Note** AWS WAF les réponses personnalisées ont priorité sur les réponses de passerelle personnalisées.

Type de réponse de passerelle	Code de statut par défaut	Description
DEFAULT_5XX	Null	Réponse de passerelle par défaut pour un type de réponse non spécifié avec un code de statut 5XX. La modification du code de statut de cette réponse de passerelle de rechange modifie les codes de statut de toutes les autres réponses 5XX sur une nouvelle valeur. La réinitialisation de ce code de statut à null rétablit les valeurs d'origine des codes de statut de toutes les autres réponses 5XX.
EXPIRED_TOKEN	403	La réponse de la passerelle à une erreur d'expiration du jeton d' AWS authentification. Si le type de réponse n'est pas spécifié, cette réponse est définie par défaut sur le type DEFAULT_4XX .
INTEGRATION_FAILURE	504	Réponse de passerelle pour une erreur d'échec d'intégration. Si le type de réponse n'est pas spécifié, cette réponse est définie par défaut sur le type DEFAULT_5XX .

Type de réponse de passerelle	Code de statut par défaut	Description
INTEGRATION_TIMEOUT	504	Réponse de passerelle pour une erreur d'intégration expirée. Si le type de réponse n'est pas spécifié, cette réponse est définie par défaut sur le type DEFAULT_5XX .
INVALID_API_KEY	403	Réponse de passerelle pour une clé d'API non valide soumise pour une méthode nécessitant une clé API. Si le type de réponse n'est pas spécifié, cette réponse est définie par défaut sur le type DEFAULT_4XX .
INVALID_SIGNATURE	403	Réponse de la passerelle à une erreur de AWS signature non valide. Si le type de réponse n'est pas spécifié, cette réponse est définie par défaut sur le type DEFAULT_4XX .

Type de réponse de passerelle	Code de statut par défaut	Description
MISSING_AUTHENTICATION_TOKEN	403	Réponse de passerelle pour une erreur de jeton d'authentification manquant, y compris les cas où le client tente d'appeler une méthode ou une ressource d'API non prise en charge. Si le type de réponse n'est pas spécifié, cette réponse est définie par défaut sur le type DEFAULT_4XX .
QUOTA_EXCEEDED	429	Réponse de passerelle pour une erreur de dépassement de quota de plan d'utilisation. Si le type de réponse n'est pas spécifié, cette réponse est définie par défaut sur le type DEFAULT_4XX .
REQUEST_TOO_LARGE	413	Réponse de passerelle pour une erreur de demande trop volumineuse. Si le type de réponse n'est pas spécifié, la valeur par défaut de cette réponse est :. HTTP content length exceeded 10485760 bytes

Type de réponse de passerelle	Code de statut par défaut	Description
RESOURCE_NOT_FOUND	404	Réponse de passerelle quand API Gateway ne peut pas trouver la ressource spécifiée après qu'une demande d'API a transmis une authentification et une autorisation, sauf pour une authentification et une autorisation de clé API. Si le type de réponse n'est pas spécifié, cette réponse est définie par défaut sur le type DEFAULT_4XX .
THROTTLED	429	Réponse de passerelle quand des limitations d'un plan d'utilisation sont dépassées au niveau de la méthode, de l'étape ou du compte. Si le type de réponse n'est pas spécifié, cette réponse est définie par défaut sur le type DEFAULT_4XX .
UNAUTHORIZED	401	Réponse de passerelle quand le mécanisme d'autorisation personnalisée ou Amazon Cognito n'a pas pu authentifier l'appelant.

Type de réponse de passerelle	Code de statut par défaut	Description
`UNSUPPORTED_MEDIA_TYPE`	`415`	Réponse de passerelle quand une charge utile est d'un type de média non pris en charge, si le comportement de transmission strict est activé. Si le type de réponse n'est pas spécifié, cette réponse est définie par défaut sur le type `DEFAULT_4XX` .
`WAF_FILTERED`	`403`	La réponse de passerelle quand une demande est bloquée par AWS WAF. Si le type de réponse n'est pas spécifié, cette réponse est définie par défaut sur le type `DEFAULT_4XX` . **Note** AWS WAF les réponses personnalisées ont priorité sur les réponses de passerelle personnalisées.

Activation de CORS pour une ressource de l'API REST

Le partage des ressources cross-origin (CORS) est une fonctionnalité de sécurité des navigateurs qui restreint les demandes HTTP cross-origin lancées à partir de scripts s'exécutant dans le navigateur. Pour plus d'informations, voir Qu'est-ce que le CORS ? .

Déterminer s'il convient d'activer la prise en charge de CORS

Une demande HTTP cross-origin est une demande effectuée pour :

- Un autre domaine (par exemple, de `example.com` à `amazondomains.com`)
- Un autre sous-domaine (par exemple, de `example.com` à `petstore.example.com`)
- Un autre port (par exemple, de `example.com` à `example.com:10777`)
- Un autre protocole (par exemple, de `https://example.com` à `http://example.com`)

Si vous ne parvenez pas à accéder à votre API et que vous recevez un message d'erreur contenant `Cross-Origin Request Blocked`, vous devrez peut-être activer CORS.

Les demandes HTTP cross-origin peuvent être réparties en deux types : les demandes simples et les demandes non simples.

Activation de CORS pour une demande simple

Une demande HTTP est simple si toutes les conditions suivantes sont réunies :

- Elle est émise par rapport à une ressource d'API qui autorise uniquement les demandes `GET`, `HEAD` et `POST`.
- S'il s'agit d'une demande de la méthode `POST`, elle doit inclure un en-tête `Origin`.
- Le type de contenu de la charge utile de la demande est `text/plain`, `multipart/form-data` ou `application/x-www-form-urlencoded`.
- La demande ne contient pas d'en-têtes personnalisés.
- Les exigences supplémentaires répertoriées dans la documentation CORS Mozilla pour les demandes simples.

Pour les demandes de méthode `POST` d'origine croisée simples, la réponse de votre ressource doit inclure l'en-tête `Access-Control-Allow-Origin: '*'` ou `Access-Control-Allow-Origin:'origin'`.

Toutes les autres demandes HTTP cross-origin sont des demandes non simples.

Activation de CORS pour une demande non simple

Si les ressources de votre API reçoivent des demandes non simples, vous devez activer une prise en charge de CORS supplémentaire en fonction de votre type d'intégration.

Activation de CORS pour les intégrations non-proxy

Pour de telles intégrations, le protocole CORS exige du navigateur qu'il envoie une demande en amont au serveur et attende l'approbation (ou une demande d'informations d'identification) du serveur avant d'envoyer la demande réelle. Vous devez configurer votre API pour envoyer une réponse appropriée à la demande en amont.

Pour créer une réponse en amont :

1. Créez une méthode `OPTIONS` avec une intégration fictive.
2. Ajoutez les en-têtes de réponse suivants à la réponse de la méthode 200 :
   - `Access-Control-Allow-Headers`
   - `Access-Control-Allow-Methods`
   - `Access-Control-Allow-Origin`
3. Définissez le comportement de transfert de l'intégration sur`NEVER`. Dans ce cas, la demande de méthode d'un type de contenu non mappé sera rejetée avec une réponse HTTP 415 Unsupported Media Type. Pour plus d'informations, consultez Comportements de transfert direct.
4. Entrez des valeurs pour les en-têtes de réponse. Pour autoriser toutes les origines, toutes les méthodes et les en-têtes communs, utilisez les valeurs d'en-tête suivantes :
   - `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: '*'`

Après avoir créé la demande en amont, vous devez renvoyer l'en-tête `Access-Control-Allow-Origin: '*'` ou `Access-Control-Allow-Origin:'origin'` pour toutes les méthodes compatibles CORS pour toutes les 200 réponses, au moins.

Activation de CORS pour les intégrations sans proxy à l'aide du AWS Management Console

Vous pouvez utiliser le AWS Management Console pour activer CORS. API Gateway crée une méthode `OPTIONS` et ajoute l'en-tête `Access-Control-Allow-Origin` à vos réponses d'intégration de méthode existantes. Cela ne fonctionne pas toujours et vous devez parfois modifier manuellement la réponse d'intégration pour renvoyer l'en-tête `Access-Control-Allow-Origin` pour toutes les méthodes compatibles CORS pour toutes 200 réponses, au moins.

Activation de la prise en charge de CORS pour les intégrations de proxy

Pour une intégration de proxy Lambda ou de proxy HTTP, votre backend est chargé de renvoyer les en-têtes `Access-Control-Allow-Origin`, `Access-Control-Allow-Methods` et `Access-Control-Allow-Headers`, car une intégration de proxy ne renvoie pas de réponse d'intégration.

Les exemples de fonctions Lambda suivants renvoient les en-têtes CORS requis :

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!')
    }
```

Rubriques

- Activation de CORS sur une ressource à l'aide de la console API Gateway
- Activation de CORS sur une ressource à l'aide de l'API d'importation API Gateway

- Test du CORS

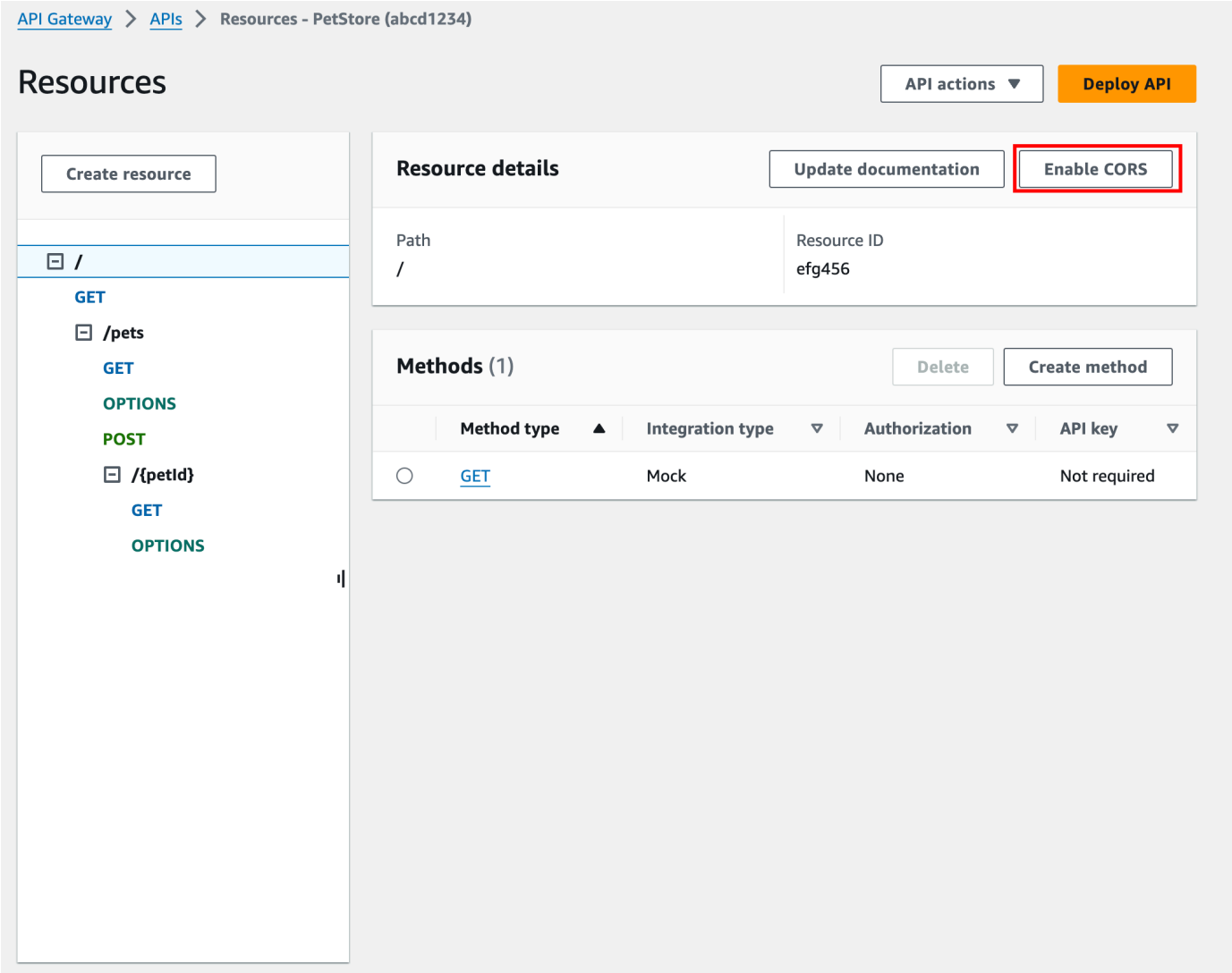
Activation de CORS sur une ressource à l'aide de la console API Gateway

Vous pouvez utiliser la console API Gateway afin d'activer la prise en charge de CORS pour une ou toutes les méthodes sur une ressource d'API REST que vous avez créée. Après avoir activé le support COR, définissez le comportement de transfert d'intégration sur`NEVER`. Dans ce cas, la demande de méthode d'un type de contenu non mappé sera rejetée avec une réponse HTTP 415 Unsupported Media Type. Pour plus d'informations, consultez Comportements de transfert direct.

> **Important**
>
> Les ressources peuvent contenir des ressources enfants. L'activation de la prise en charge de CORS pour une ressource et ses méthodes ne permet pas de l'activer de façon récursive pour les ressources enfants et leurs méthodes.

Pour activer la prise en charge du CORS sur une ressource d'API REST

1. Connectez-vous à la console API Gateway à l'adresse : https://console.aws.amazon.com/apigateway.
2. Choisissez une API.
3. Sélectionnez une ressource sous Resources.
4. Dans la section Détails de la ressource, choisissez Activer le CORS.



5. Dans la zone Activer le CORS, procédez comme suit :

   a. (Facultatif) Si vous avez créé une réponse de passerelle personnalisée et que vous souhaitez activer la prise en charge du CORS pour une réponse, sélectionnez une réponse de passerelle.

   b. Sélectionnez chaque méthode pour activer la prise en charge du CORS. Le CORS doit être activé sur la méthode `OPTION`.

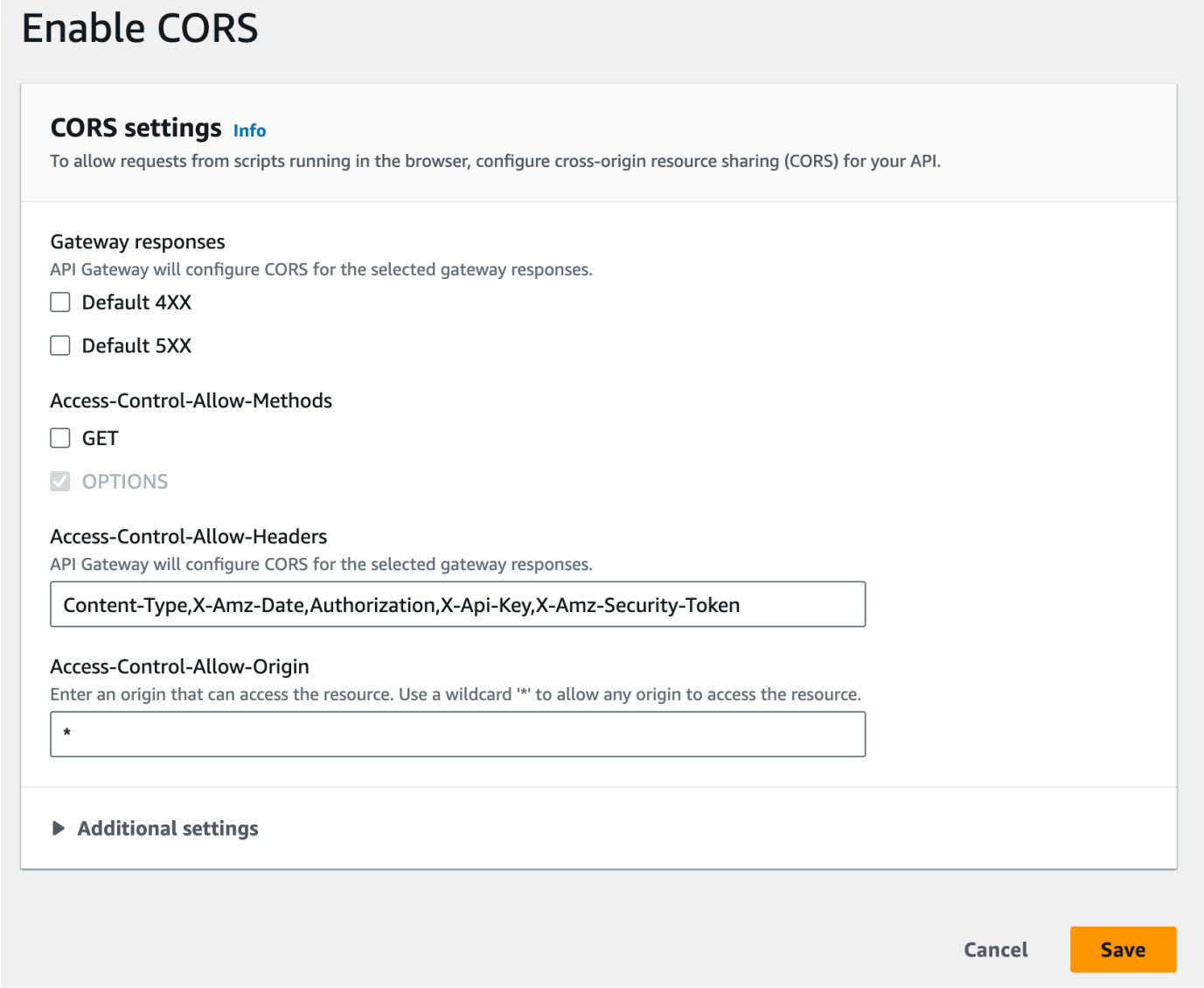
      Si vous activez la prise en charge du CORS pour une méthode `ANY`, le CORS est activé pour toutes les méthodes.

   c. Dans le champ de saisie Access-Control-Allow-Headers, entrez la chaîne statique d'une liste d'en-têtes séparés par des virgules que le client doit envoyer dans la demande réelle de la ressource. Utilisez la liste d'en-têtes `'Content-Type,X-Amz-Date,Authorization,X-`

`Api-Key,X-Amz-Security-Token'` fournie par la console ou spécifiez vos propres en-têtes.

d. Utilisez la valeur `'*'` fournie par la console en tant que valeur d'en-tête Access-Control-Allow-Origin afin d'autoriser les demandes d'accès depuis toutes les origines, ou spécifiez les origines autorisées à accéder à la ressource.

e. Choisissez Enregistrer.



> **Important**
>
> Lors de l'application des instructions ci-dessus à la méthode ANY dans une intégration de proxy, aucun en-tête CORS applicable n'est défini. Au lieu de cela, votre backend doit renvoyer les en-têtes CORS applicables, tels que `Access-Control-Allow-Origin`.

Une fois que CORS est activé sur la méthode `GET`, une méthode `OPTIONS` est ajoutée à la ressource, si elle n'existe pas déjà. La réponse `200` de la méthode `OPTIONS` est automatiquement configurée pour renvoyer les trois en-têtes `Access-Control-Allow-*` afin d'effectuer l'établissement de liaison en amont. En outre, la méthode réelle (`GET`) est également configurée par défaut pour renvoyer l'en-tête `Access-Control-Allow-Origin` dans sa réponse 200. En ce qui concerne les autres types de réponses, vous devrez les configurer manuellement pour renvoyer l'en-tête `Access-Control-Allow-Origin'` avec la valeur '*' ou les origines spécifiques, si vous ne souhaitez pas renvoyer l'erreur `Cross-origin access`.

Une fois la prise en charge de CORS activée sur votre ressource, vous devez déployer ou redéployer l'API pour que les nouveaux paramètres prennent effet. Pour plus d'informations, consultez the section called “Déployer une API REST (console)”.

> **Note**
>
> Si vous ne parvenez pas à activer le support CORS sur votre ressource après avoir suivi la procédure, nous vous recommandons de comparer votre configuration CORS à l'exemple de ressource API `/pets`. Pour savoir comment créer l'exemple d'API, consultezthe section called “Tutoriel : Création d'une API REST par l'importation d'un exemple”.

Activation de CORS sur une ressource à l'aide de l'API d'importation API Gateway

Si vous utilisez la fonction API Gateway Import API (API d'importation API Gateway), vous pouvez configurer la prise en charge de CORS à l'aide d'un fichier OpenAPI. Vous devez d'abord définir une méthode `OPTIONS` dans votre ressource, qui renvoie les en-têtes requis.

> **Note**
>
> Les navigateurs web s'attendent à ce que les en-têtes Access-Control-Allow-Headers et Access-Control-Allow-Origin soient configurés dans chaque méthode d'API qui accepte les

demandes CORS. En outre, certains navigateurs effectuent d'abord une demande HTTP auprès d'une méthode `OPTIONS` dans la même ressource, puis s'attendent à recevoir les mêmes en-têtes.

Exemple de méthode **`Options`**

L'exemple suivant crée une méthode `OPTIONS` méthode pour une intégration fictive.

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"
```

Une fois que vous avez configuré la méthode `OPTIONS` pour votre ressource, vous pouvez ajouter les en-têtes requis aux autres méthodes de la même ressource qui doivent accepter des demandes CORS.

1. Déclarez les en-têtes Access-Control-Allow-Origin et Access-Control-Allow-Headers pour les types de réponse.

   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. Dans la balise `x-amazon-apigateway-integration`, configurez le mappage de ces en-têtes à vos valeurs statiques :

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 : "'*'"
```

Exemple d’API

L’exemple suivant crée une API complète avec une méthode `OPTIONS` et une méthode `GET` avec une intégration `HTTP`.

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"
```

Test du CORS

Vous pouvez tester la configuration CORS de votre API en appelant votre API et en vérifiant les en-têtes CORS dans la réponse. La commande `curl` suivante envoie une demande OPTIONS à une API déployée.

```
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
```

Les en-têtes `Access-Control-Allow-Origin`, `Access-Control-Allow-Headers` et `Access-Control-Allow-Methods` dans la réponse montrent que l'API prend en charge CORS. Pour plus d'informations, voir Activation de CORS pour une ressource de l'API REST.

Utilisation des types de médias binaires pour les API REST

Dans API Gateway, la demande et la réponse d'API peuvent avoir une charge utile textuelle ou binaire. Une charge utile de texte est une chaîne JSON codée en `UTF-8`. Une charge utile binaire est tout élément autre qu'une charge utile textuelle. Par exemple, une charge utile binaire peut être un fichier JPEG, un fichier GZip ou un fichier XML. La configuration d'API requise pour prendre en charge les supports binaires varie selon que votre API utilise des intégrations proxy ou non proxy.

AWS Lambda intégrations de proxy

Pour gérer les charges utiles binaires pour les intégrations de AWS Lambda proxy, vous devez encoder la réponse de votre fonction en base64. Vous devez également configurer le binaryMediaTypespour votre API. La configuration `binaryMediaTypes` de votre API est une liste de types de contenu que votre API traite comme des données binaires. Par exemple, les types de média binaire incluent `image/png` ou `application/octet-stream`. Vous pouvez utiliser le caractère

générique (*) pour couvrir plusieurs types de support. Par exemple, `*/*` inclut tous les types de contenu.

Pour obtenir un exemple de code, consultez the section called “Renvoi d'un support binaire à partir d'une intégration de proxy Lambda”.

Intégrations non proxy

Pour gérer les charges utiles binaires pour les intégrations sans proxy, vous devez ajouter les types de média à la binaryMediaTypesliste de la ressource. `RestApi` La configuration `binaryMediaTypes` de votre API est une liste de types de contenu que votre API traite comme des données binaires. Vous pouvez également définir les propriétés ContentHandling sur l'intégration et les ressources. IntegrationResponse La valeur `contentHandling` peut être `CONVERT_TO_BINARY`, `CONVERT_TO_TEXT` ou indéfinie.

Selon la valeur de `contentHandling` et si l'en-tête `Content-Type` de la réponse ou l'en-tête `Accept` de la demande entrante correspond à une entrée de la liste `binaryMediaTypes`, API Gateway peut encoder les octets binaires bruts sous forme de chaîne codée en base64, décoder une chaîne codée en base64 pour obtenir les octets bruts ou passer le corps sans modification.

Vous devez configurer l'API comme suit pour prendre en charge les charges utiles binaires de votre API API Gateway :

- Ajoutez les types de supports binaires souhaités à la `binaryMediaTypes` liste de la RestApiressource. Si cette propriété et la propriété `contentHandling` ne sont pas définies, les charges utiles sont traitées en tant que chaînes JSON codées UTF-8.
- Adressez la propriété `contentHandling` de la ressource Integration.
  - Pour que la charge utile de la demande soit convertie d'une chaîne codée en Base64 en son blob binaire, définissez la propriété sur `CONVERT_TO_BINARY`.
  - Pour que la charge utile de la demande soit convertie d'un blob binaire en une chaîne codée en Base64, définissez la propriété sur `CONVERT_TO_TEXT`.
  - Pour transmettre la charge utile sans modification, laissez la propriété indéfinie. Pour transmettre une charge utile binaire sans modification, vous devez également vous assurer que `Content-Type` correspond à l'une des entrées `binaryMediaTypes` et que les comportements de transmission sont activés pour l'API.
- Définissez la `contentHandling` propriété de la IntegrationResponseressource. La propriété `contentHandling`, l'en-tête `Accept` dans les demandes client et les `binaryMediaTypes` de

vos API déterminent la façon dont API Gateway gère les conversions de type de contenu. Pour plus de détails, consultez the section called “Conversions du type de contenu dans API Gateway”.

> **Important**
>
> Lorsqu'une demande contient plusieurs types de support dans son en-tête `Accept`, API Gateway respecte uniquement le premier type de support `Accept`. Si vous ne pouvez pas contrôler l'ordre des types de support `Accept` et si le type de support de votre contenu binaire n'est pas le premier de la liste, ajoutez le premier type de support `Accept` dans la liste `binaryMediaTypes` de votre API. API Gateway gère tous les types de contenu de cette liste sous forme binaire.
>
> Par exemple, pour envoyer un fichier JPEG en utilisant un élément `<img>` dans un navigateur, ce dernier peut envoyer `Accept:image/webp,image/*,*/*;q=0.8` dans une demande. Si vous ajoutez `image/webp` à la liste `binaryMediaTypes`, le point de terminaison reçoit le fichier JPEG sous forme binaire.

Pour de plus amples informations sur la façon dont API Gateway gère les charges utiles textuelles et binaires, veuillez consulter Conversions du type de contenu dans API Gateway.

Conversions du type de contenu dans API Gateway

La combinaison des `binaryMediaTypes` de vos API, les en-têtes dans les demandes client et la propriété `contentHandling` d'intégration déterminent la manière dont API Gateway encode les charges utiles.

Le tableau suivant montre comment API Gateway convertit la charge utile de la demande pour des configurations spécifiques de l'`Content-Type`en-tête d'une demande, de la `binaryMediaTypes` liste d'une RestApiressource et de la valeur de `contentHandling` propriété de la ressource d'intégration.

Conversions du type de contenu des demandes d'API dans API Gateway

Charge utile de la demande de méthode	En-tête **`Content-Type`** de la demande	**`binaryMediaTypes`**	**`contentHandling`**	Charge utile de la demande d'intégration
Données de texte	Tout type de données	Non défini	Non défini	Chaîne encodée en UTF8
Données de texte	Tout type de données	Non défini	`CONVERT_TO_BINARY`	Blob binaire décodé en Base64
Données de texte	Tout type de données	Non défini	`CONVERT_TO_TEXT`	Chaîne encodée en UTF8
Données de texte	Un type de données texte	Défini avec les types de supports correspondants	Non défini	Données de texte
Données de texte	Un type de données texte	Défini avec les types de supports correspondants	`CONVERT_TO_BINARY`	Blob binaire décodé en Base64
Données de texte	Un type de données texte	Défini avec les types de supports correspondants	`CONVERT_TO_TEXT`	Données de texte
Données binaires	Un type de données binaires	Défini avec les types de supports correspondants	Non défini	Données binaires
Données binaires	Un type de données binaires	Défini avec les types	`CONVERT_TO_BINARY`	Données binaires

Charge utile de la demande de méthode	En-tête **`Content-Type`** de la demande	**`binaryMediaTypes`**	**`contentHandling`**	Charge utile de la demande d'intégration
		de supports correspondants		
Données binaires	Un type de données binaires	Défini avec les types de supports correspondants	`CONVERT_TO_TEXT`	Chaîne d'encodage en Base64

Le tableau suivant montre comment API Gateway convertit la charge utile de réponse pour des configurations spécifiques de l'`Accept`en-tête d'une demande, de la `binaryMediaTypes` liste d'une RestApiressource et de la valeur de `contentHandling` propriété de la IntegrationResponseressource.

> **Important**
>
> Lorsqu'une demande contient plusieurs types de support dans son en-tête `Accept`, API Gateway respecte uniquement le premier type de support `Accept`. Si vous ne pouvez pas contrôler l'ordre des types de support `Accept` et si le type de support de votre contenu binaire n'est pas le premier de la liste, ajoutez le premier type de support `Accept` dans la liste `binaryMediaTypes` de votre API. API Gateway gère tous les types de contenu de cette liste sous forme binaire.
>
> Par exemple, pour envoyer un fichier JPEG en utilisant un élément `<img>` dans un navigateur, ce dernier peut envoyer `Accept:image/webp,image/*,*/*;q=0.8` dans une demande. Si vous ajoutez `image/webp` à la liste `binaryMediaTypes`, le point de terminaison reçoit le fichier JPEG sous forme binaire.

Conversions de type de contenu des réponses API Gateway

Charge utile de la réponse d'intégration	En-tête **Accept** de la demande	**binaryMediaTypes**	**contentHandling**	Charge utile de la réponse de méthode
Données texte ou binaires	Un type de texte	Non défini	Non défini	Chaîne encodée en UTF8
Données texte ou binaires	Un type de texte	Non défini	`CONVERT_TO_BINARY`	Blob décodé en Base64
Données texte ou binaires	Un type de texte	Non défini	`CONVERT_TO_TEXT`	Chaîne encodée en UTF8
Données de texte	Un type de texte	Défini avec les types de supports correspondants	Non défini	Données de texte
Données de texte	Un type de texte	Défini avec les types de supports correspondants	`CONVERT_TO_BINARY`	Blob décodé en Base64
Données de texte	Un type de texte	Défini avec les types de supports correspondants	`CONVERT_TO_TEXT`	Chaîne encodée en UTF8
Données de texte	Un type binaire	Défini avec les types de supports correspondants	Non défini	Blob décodé en Base64
Données de texte	Un type binaire	Défini avec les types de supports correspondants	`CONVERT_TO_BINARY`	Blob décodé en Base64

Charge utile de la réponse d'intégration	En-tête **Accept** de la demande	**binaryMediaTypes**	**contentHandling**	Charge utile de la réponse de méthode
Données de texte	Un type binaire	Défini avec les types de supports correspondants	`CONVERT_TO_TEXT`	Chaîne encodée en UTF8
Données binaires	Un type de texte	Défini avec les types de supports correspondants	Non défini	Chaîne d'encodage en Base64
Données binaires	Un type de texte	Défini avec les types de supports correspondants	`CONVERT_TO_BINARY`	Données binaires
Données binaires	Un type de texte	Défini avec les types de supports correspondants	`CONVERT_TO_TEXT`	Chaîne d'encodage en Base64
Données binaires	Un type binaire	Défini avec les types de supports correspondants	Non défini	Données binaires
Données binaires	Un type binaire	Défini avec les types de supports correspondants	`CONVERT_TO_BINARY`	Données binaires
Données binaires	Un type binaire	Défini avec les types de supports correspondants	`CONVERT_TO_TEXT`	Chaîne d'encodage en Base64

Lors de la conversion d'une charge utile de texte en blob binaire, API Gateway suppose que les données de texte sont une chaîne encodée en base64 et génère les données binaires comme blob décodé en base64. Si la conversion échoue, elle retourne une réponse `500` indiquant une erreur de configuration de l'API. Vous ne fournissez pas de modèle de mappage pour ce type de conversion, même si vous devez activer les comportements relais sur l'API.

Lors de la conversion d'une charge utile binaire en une chaîne de texte, API Gateway applique toujours un encodage en base64 sur les données binaires. Vous pouvez définir un modèle de mappage pour une charge utile de ce type, mais vous pouvez uniquement accéder à la chaîne d'encodage base64 dans le modèle de mappage via `$input.body`, comme illustré dans l'extrait d'un exemple de modèle de mappage.

```
{
    "data": "$input.body"
}
```

Pour que la charge utile binaire soit passée sans modification, vous devez activer les comportements relais sur l'API.

Activation de la prise en charge binaire à l'aide de la console API Gateway

Cette section explique comment activer la prise en charge binaire à l'aide de la console API Gateway. À titre d'exemple, nous utilisons une API intégrée à Amazon S3. Nous nous concentrons sur les tâches visant à définir les types de supports pris en charge afin de spécifier la façon dont la charge utile doit être gérée. Pour obtenir des informations détaillées sur la création d'une API intégrée à Amazon S3, veuillez consulter Tutoriel : Création d'une API REST en tant que proxy Amazon S3 dans API Gateway.

Pour activer la prise en charge binaire à l'aide de la console API Gateway

1. Définissez les types de supports binaires pour l'API :

    a. Créez une API ou choisissez une API existante. Pour cet exemple, nous appelons l'API `FileMan`.
    b. Sous l'API sélectionnée dans le panneau de navigation principal, choisissez Paramètres de l'API.
    c. Dans le volet Paramètres de l'API, choisissez Gérer les types de médias dans la section Types de médias binaires.
    d. Choisissez Ajouter un type de médias binaires.

e. Saisissez un type de média requis, par exemple **`image/png`**, dans le champ de texte d'entrée. Si nécessaire, répétez cette étape pour ajouter d'autres types de médias. Pour prendre en charge tous les types de supports binaires, spécifiez `*/*`.

f. Sélectionnez Enregistrer les modifications.

2. Définissez la façon dont les charges utiles du message sont traitées pour la méthode de l'API :

a. Créez une nouvelle ressource ou choisissez-en une existante dans l'API. Dans le cadre de cet exemple, nous utilisons la ressource `/{folder}/{item}`.

b. Créez une méthode ou choisissez-en une existante sur la ressource. À titre d'exemple, nous utilisons la méthode `GET /{folder}/{item}` intégrée à l'action `Object GET` dans Amazon S3.

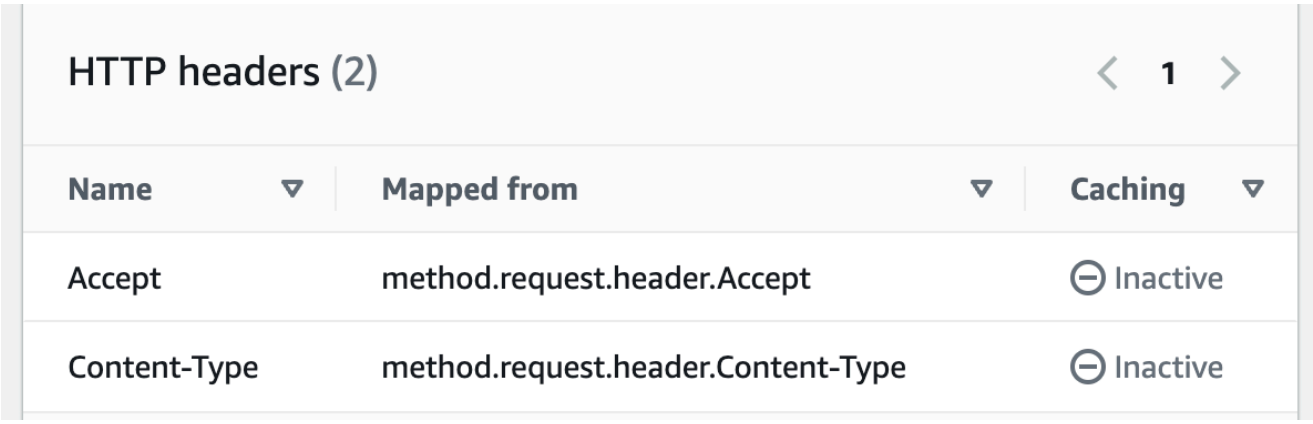
c. Dans Gestion de contenu, choisissez une option.

Choisissez Passthrough si vous ne souhaitez pas convertir le corps lorsque le client et le backend acceptent le même format binaire. Choisissez Convertir en texte pour convertir le corps binaire en chaîne encodée en base64 lorsque, par exemple, le backend a besoin qu'une charge utile de requête binaire soit passée en tant que propriété JSON. Choisissez Convertir en binaire lorsque le client envoie une chaîne encodée en base64 et si le backend nécessite le format binaire d'origine, ou lorsque le point de terminaison retourne une chaîne encodée en base64 et si le client accepte uniquement la sortie binaire.

d. Pour Transmission du corps de requête, choisissez Lorsqu'aucun modèle n'est défini (recommandé) pour activer le comportement de transmission pour le corps de la requête.

Vous pouvez également choisir Jamais. Cela signifie que l'API rejettera les données avec des types de contenu qui ne disposent pas d'un modèle de mappage.

e. Conservez l'en-tête `Accept` de la demande entrante dans la demande d'intégration. Procédez ainsi si vous avez défini `contentHandling` sur `passthrough` et que vous souhaitez remplacer ce paramètre au moment de l'exécution.



f. Pour la conversion en texte, définissez un modèle de mappage afin de convertir les données binaires encodées en base64 au format requis.

Voici un exemple de modèle de mappage à convertir en texte :

```
{
  "operation": "thumbnail",
  "base64Image": "$input.body"
}
```

Le format de ce modèle de mappage s'appuie sur les exigences de point de terminaison de l'entrée.

g. Choisissez Enregistrer.

Activation de la prise en charge binaire à l'aide de l'API REST API Gateway

Les tâches suivantes montrent comment activer la prise en charge binaire à l'aide des appels de l'API REST API Gateway.

Rubriques

- Ajout et mise à jour des types de supports binaires d'une API

- Configuration des conversions de charge utile de la demande
- Configuration des conversions de charge utile pour la réponse
- Conversion des données binaires en données de texte
- Conversion des données de texte en charge utile binaire
- Passage via une charge utile binaire

Ajout et mise à jour des types de supports binaires d'une API

Pour permettre à API Gateway de prendre en charge un nouveau type de support binaire, vous devez ajouter le type de support binaire à la liste `binaryMediaTypes` de la ressource `RestApi`. Par exemple, pour qu'API Gateway gère les images JPEG, envoyez une demande `PATCH` à la ressource `RestApi` :

```
PATCH /restapis/<restapi_id>

{
  "patchOperations" : [ {
    "op" : "add",
    "path" : "/binaryMediaTypes/image~1jpeg"
  }
 ]
}
```

La spécification de type MIME de `image/jpeg` qui fait partie de la valeur de propriété `path` est échappée comme `image~1jpeg`.

Pour mettre à jour les types de supports binaires pris en charge, remplacez ou supprimez le type de support de la liste `binaryMediaTypes` de la ressource `RestApi`. Par exemple, pour remplacer la prise en charge binaire des fichiers JPEG par celle des octets bruts, soumettez une demande `PATCH` à la ressource `RestApi`, comme suit :

```
PATCH /restapis/<restapi_id>

{
  "patchOperations" : [{
    "op" : "replace",
    "path" : "/binaryMediaTypes/image~1jpeg",
    "value" : "application/octet-stream"
  },
```

```
  {
    "op" : "remove",
    "path" : "/binaryMediaTypes/image~1jpeg"
  }]
}
```

Configuration des conversions de charge utile de la demande

Si le point de terminaison nécessite une entrée binaire, définissez la propriété `contentHandling` de la ressource `Integration` sur `CONVERT_TO_BINARY`. Pour ce faire, présentez une `PATCH` demande, comme suit :

```
PATCH /restapis/<restapi_id>/resources/<resource_id>/methods/<http_method>/integration

{
  "patchOperations" : [ {
    "op" : "replace",
    "path" : "/contentHandling",
    "value" : "CONVERT_TO_BINARY"
  }]
}
```

Configuration des conversions de charge utile pour la réponse

Si le client accepte le résultat comme blob binaire au lieu de la charge utile encodée en base64 renvoyée par le point de terminaison, définissez la propriété `contentHandling` de la ressource `IntegrationResponse` sur `CONVERT_TO_BINARY`. Pour ce faire, soumettez une demande `PATCH`, comme suit :

```
PATCH /restapis/<restapi_id>/resources/<resource_id>/methods/<http_method>/integration/
responses/<status_code>

{
  "patchOperations" : [ {
    "op" : "replace",
    "path" : "/contentHandling",
    "value" : "CONVERT_TO_BINARY"
  }]
}
```

Conversion des données binaires en données de texte

Pour envoyer des données binaires sous forme de propriété JSON de l'entrée à AWS Lambda ou à Kinesis via API Gateway, procédez comme suit :

1. Activez la prise en charge de la charge utile binaire de l'API en ajoutant le nouveau type de support binaire de `application/octet-stream` à la liste de l'API `binaryMediaTypes`.

```
PATCH /restapis/<restapi_id>

{
  "patchOperations" : [ {
    "op" : "add",
    "path" : "/binaryMediaTypes/application~1octet-stream"
  }
 ]
}
```

2. Définissez `CONVERT_TO_TEXT` sur la propriété `contentHandling` de la ressource `Integration` et fournissez fournir un modèle de mappage pour attribuer la chaîne encodée en base64 des données binaires à une propriété JSON. Dans l'exemple suivant, la propriété JSON est `body` et `$input.body` contient la chaîne encodée en base64.

```
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\"}"
    }
  ]
}
```

Conversion des données de texte en charge utile binaire

Supposons qu'une fonction Lambda renvoie un fichier d'image sous forme de chaîne encodée en base64. Pour transmettre cette sortie binaire au client via API Gateway, procédez comme suit :

1. Mettez à jour la liste `binaryMediaTypes` de l'API en ajoutant le type de support binaire de `application/octet-stream`, s'il n'est pas déjà dans la liste.

```
PATCH /restapis/<restapi_id>

{
  "patchOperations" : [ {
    "op" : "add",
    "path" : "/binaryMediaTypes/application~1octet-stream",
  }]
}
```

2. Définissez la propriété `contentHandling` de la ressource `Integration` sur `CONVERT_TO_BINARY`. Ne définissez pas de modèle de mappage. Si vous ne définissez pas de modèle de mappage, API Gateway appelle le modèle de relais pour renvoyer le blob binaire encodé en base64 sous forme de fichier image au client.

```
PATCH /restapis/<restapi_id>/resources/<resource_id>/methods/<http_method>/
integration/responses/<status_code>

{
  "patchOperations" : [
    {
      "op" : "replace",
      "path" : "/contentHandling",
      "value" : "CONVERT_TO_BINARY"
    }
  ]
}
```

Passage via une charge utile binaire

Pour stocker une image dans un compartiment Amazon S3 à l'aide d'API Gateway, procédez comme suit :

1. Mettez à jour la liste `binaryMediaTypes` de l'API en ajoutant le type de support binaire de `application/octet-stream`, s'il n'est pas déjà dans la liste.

```
PATCH /restapis/<restapi_id>

{
  "patchOperations" : [ {
    "op" : "add",
    "path" : "/binaryMediaTypes/application~1octet-stream"
  }
 ]
}
```

2. Sur la propriété `contentHandling` de la ressource `Integration`, définissez `CONVERT_TO_BINARY`. Définissez `WHEN_NO_MATCH` comme valeur de propriété `passthroughBehavior` sans définir de modèle de mappage. Cela permet à API Gateway d'appeler le modèle de relais.

```
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"
    }
  ]
}
```

Importation et exportation des encodages de contenu

Pour importer la `binaryMediaTypes` liste sur un RestApi, utilisez l'extension API Gateway suivante dans le fichier de définition OpenAPI de l'API. L'extension est également utilisée pour exporter les paramètres de l'API.

- x-amazon-apigateway-binarypropriété -media-types

Pour importer et exporter la valeur de propriété `contentHandling` sur une ressource `Integration` ou `IntegrationResponse`, utilisez les extensions API Gateway suivantes pour les définitions OpenAPI :

- x-amazon-apigateway-integration objet
- x-amazon-apigateway-integrationobjet .response

Renvoi d'un support binaire à partir d'une intégration de proxy Lambda

Pour renvoyer un média binaire à partir d'une intégration de proxy AWS Lambda, encodez en base64 la réponse de votre fonction Lambda. Vous devez également configurer les types de support binaire de votre API. La taille de la charge utile ne doit pas dépasser 10 Mo.

> **Note**
>
> Pour utiliser un navigateur web pour appeler une API avec cet exemple d'intégration, définissez les types de supports binaires de votre API sur `*/*`. API Gateway utilise le premier en-tête `Accept` des clients pour déterminer si une réponse doit renvoyer un support binaire. Pour renvoyer un support binaire lorsque vous ne pouvez pas contrôler l'ordre des valeurs d'en-tête `Accept`, telles que les demandes d'un navigateur, définissez les types de support binaire de votre API sur `*/*` (pour tous les types de contenu).

L'exemple de fonction Lambda suivant peut renvoyer aux clients une image binaire à partir d'Amazon S3 ou du texte. La réponse de la fonction inclut un en-tête `Content-Type` pour indiquer au client le type de données qu'elle renvoie. La fonction définit de manière conditionnelle la propriété `isBase64Encoded` dans sa réponse, en fonction du type de données qu'elle renvoie.

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>",
        }
```

Pour de plus amples informations sur les types de support binaire, veuillez consulter Utilisation des types de médias binaires pour les API REST.

Accès aux fichiers binaires dans Amazon S3 via une API API Gateway

Les exemples suivants illustrent le fichier OpenAPI utilisé pour accéder aux images dans Amazon S3, le téléchargement d'une image depuis Amazon S3 et le chargement d'une image vers Amazon S3.

Rubriques

- Fichier OpenAPI d'un exemple d'API pour accéder aux images dans Amazon S3
- Téléchargement d'une image depuis Amazon S3
- Chargement d'une image sur Amazon S3

Fichier OpenAPI d'un exemple d'API pour accéder aux images dans Amazon S3

Le fichier OpenAPI suivant présent un exemple d'API qui illustre le téléchargement d'un fichier image depuis Amazon S3 et le chargement d'un fichier image vers Amazon S3. Cette API expose les méthodes `GET /s3?key={file-name}` et `PUT /s3?key={file-name}` de téléchargement et de chargement d'un fichier image spécifié. La méthode `GET` renvoie le fichier image sous la forme d'une chaîne encodée en base64 dans le cadre d'une sortie JSON, selon le modèle de mappage fourni,

dans une réponse 200 OK. La méthode PUT prend un blob binaire brut comme entrée et renvoie une réponse 200 OK avec une charge utile vide.

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"
    }
  }
}
```

Téléchargement d'une image depuis Amazon S3

Pour télécharger un fichier image (`image.jpg`) comme blob binaire depuis Amazon S3 :

```
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
```

La réponse de réussite ressemble à ceci :

```
200 OK HTTP/1.1

[raw bytes]
```

Les octets bruts sont renvoyés car l'en-tête `Accept` est défini sur un type de support binaire de `application/octet-stream` et la prise en charge binaire est activée pour l'API.

Vous pouvez également télécharger un fichier image (`image.jpg`) en tant que chaîne encodée en base64, au format d'une propriété JSON, à partir d'Amazon S3, ajouter un modèle de réponse à la réponse d'intégration 200, comme illustré dans le bloc de définition OpenAPI en caractères gras suivant :

```
        "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}"
              }
            }
          },
```

La demande pour télécharger le fichier image ressemble à ce qui suit :

```
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
```

La réponse de réussite ressemble à ce qui suit :

```
200 OK HTTP/1.1

{
  "image": "W3JhdyBieXRlc10="
}
```

Chargement d'une image sur Amazon S3

Pour charger un fichier image (`image.jpg`) comme blob binaire vers Amazon S3 :

```
PUT /v1/s3?key=image.jpg HTTP/1.1
```

```
Host: abcdefghi.execute-api.us-east-1.amazonaws.com
Content-Type: application/octet-stream
Accept: application/json

[raw bytes]
```

La réponse de réussite ressemble à ce qui suit :

```
200 OK HTTP/1.1
```

Pour charger un fichier image (`image.jpg`) sous la forme d'une chaîne encodée en base64 vers Amazon S3 :

```
PUT /v1/s3?key=image.jpg HTTP/1.1
Host: abcdefghi.execute-api.us-east-1.amazonaws.com
Content-Type: application/json
Accept: application/json

W3JhdyBieXRlc10=
```

La charge utile d'entrée doit être une chaîne encodée en base64 car la valeur de l'en-tête `Content-Type` est définie sur `application/json`. La réponse de réussite ressemble à ce qui suit :

```
200 OK HTTP/1.1
```

Accès aux fichiers binaires dans Lambda à l'aide d'une API API Gateway

L'exemple OpenAPI suivant montre comment accéder à un fichier binaire AWS Lambda via une API API Gateway. Cette API expose les méthodes `GET /lambda?key={file-name}` et les `PUT /lambda?key={file-name}` méthodes de téléchargement et de chargement d'un fichier image spécifié. La méthode `GET` renvoie le fichier image sous la forme d'une chaîne encodée en base64 dans le cadre d'une sortie JSON, selon le modèle de mappage fourni, dans une réponse 200 OK. La méthode `PUT` prend un blob binaire brut comme entrée et renvoie une réponse 200 OK avec une charge utile vide.

Vous créez la fonction Lambda que votre API appelle, et elle doit renvoyer une chaîne codée en base64 avec l'en-tête de. `Content-Type application/json`

Rubriques

- Fichier OpenAPI d'un exemple d'API pour accéder aux images dans Lambda
- Téléchargement d'une image depuis Lambda
- Chargement d'une image vers Lambda

Fichier OpenAPI d'un exemple d'API pour accéder aux images dans Lambda

Le fichier OpenAPI suivant présente un exemple d'API qui illustre le téléchargement d'un fichier image depuis Lambda et le chargement d'un fichier image vers Lambda.

OpenAPI 3.0

```
{
   "openapi": "3.0.0",
   "info": {
      "version": "2016-10-21T17:26:28Z",
      "title": "ApiName"
   },
   "paths": {
      "/lambda": {
         "get": {
            "parameters": [
               {
                  "name": "key",
                  "in": "query",
                  "required": false,
                  "schema": {
                     "type": "string"
                  }
               }
            ],
            "responses": {
               "200": {
                  "description": "200 response",
                  "content": {
                     "application/json": {
                        "schema": {
                           "$ref": "#/components/schemas/Empty"
                        }
                     }
                  }
               },
               "500": {
```

```
                "description": "500 response"
             }
          },
          "x-amazon-apigateway-integration": {
             "uri": "arn:aws:apigateway:us-east-1:lambda:path/2015-03-31/
functions/arn:aws:lambda:us-east-1:123456789012:function:image/invocations",
             "type": "AWS",
             "credentials": "arn:aws:iam::123456789012:role/Lambda",
             "httpMethod": "POST",
             "requestTemplates": {
                "application/json": "{\n   \"imageKey\":
 \"$input.params('key')\"\n}"
             },
             "responses": {
                "default": {
                   "statusCode": "500"
                },
                "2\\d{2}": {
                   "statusCode": "200",
                   "responseTemplates": {
                      "application/json": "{\n   \"image\": \"$input.body\"\n}"
                   }
                }
             }
          }
       },
       "put": {
          "parameters": [
             {
                "name": "key",
                "in": "query",
                "required": false,
                "schema": {
                   "type": "string"
                }
             }
          ],
          "responses": {
             "200": {
                "description": "200 response",
                "content": {
                   "application/json": {
                      "schema": {
                         "$ref": "#/components/schemas/Empty"
```

```
                        }
                     },
                     "application/octet-stream": {
                        "schema": {
                           "$ref": "#/components/schemas/Empty"
                        }
                     }
                  }
               },
               "500": {
                  "description": "500 response"
               }
            },
            "x-amazon-apigateway-integration": {
               "uri": "arn:aws:apigateway:us-east-1:lambda:path/2015-03-31/
functions/arn:aws:lambda:us-east-1:123456789012:function:image/invocations",
               "type": "AWS",
               "credentials": "arn:aws:iam::123456789012:role/Lambda",
               "httpMethod": "POST",
               "contentHandling": "CONVERT_TO_TEXT",
               "requestTemplates": {
                  "application/json": "{\n   \"imageKey\": \"$input.params('key')\",
 \"image\": \"$input.body\"\n}"
               },
               "responses": {
                  "default": {
                     "statusCode": "500"
                  },
                  "2\\d{2}": {
                     "statusCode": "200"
                  }
               }
            }
         }
      },
   "x-amazon-apigateway-binary-media-types": [
      "application/octet-stream",
      "image/jpeg"
   ],
   "servers": [
      {
         "url": "https://abcdefghi.execute-api.us-east-1.amazonaws.com/{basePath}",
         "variables": {
```

```
            "basePath": {
              "default": "/v1"
            }
         }
      }
   ],
   "components": {
      "schemas": {
         "Empty": {
            "type": "object",
            "title": "Empty Schema"
         }
      }
   }
}
```

OpenAPI 2.0

```
{
  "swagger": "2.0",
  "info": {
    "version": "2016-10-21T17:26:28Z",
    "title": "ApiName"
  },
  "host": "abcdefghi.execute-api.us-east-1.amazonaws.com",
  "basePath": "/v1",
  "schemes": [
    "https"
  ],
  "paths": {
    "/lambda": {
      "get": {
        "produces": [
          "application/json"
        ],
        "parameters": [
          {
            "name": "key",
            "in": "query",
            "required": false,
            "type": "string"
          }
        ],
```

```
        "responses": {
          "200": {
            "description": "200 response",
            "schema": {
              "$ref": "#/definitions/Empty"
            }
          },
          "500": {
            "description": "500 response"
          }
        },
        "x-amazon-apigateway-integration": {
          "uri": "arn:aws:apigateway:us-east-1:lambda:path/2015-03-31/functions/
arn:aws:lambda:us-east-1:123456789012:function:image/invocations",
          "type": "AWS",
          "credentials": "arn:aws:iam::123456789012:role/Lambda",
          "httpMethod": "POST",
          "requestTemplates": {
            "application/json": "{\n   \"imageKey\": \"$input.params('key')\"\n}"
          },
          "responses": {
            "default": {
              "statusCode": "500"
            },
            "2\\d{2}": {
              "statusCode": "200",
              "responseTemplates": {
                "application/json": "{\n   \"image\": \"$input.body\"\n}"
              }
            }
          }
        }
      },
      "put": {
        "produces": [
          "application/json", "application/octet-stream"
        ],
        "parameters": [
          {
            "name": "key",
            "in": "query",
            "required": false,
            "type": "string"
          }
```

```
        ],
        "responses": {
          "200": {
            "description": "200 response",
            "schema": {
              "$ref": "#/definitions/Empty"
            }
          },
          "500": {
            "description": "500 response"
          }
        },
        "x-amazon-apigateway-integration": {
          "uri": "arn:aws:apigateway:us-east-1:lambda:path/2015-03-31/functions/
arn:aws:lambda:us-east-1:123456789012:function:image/invocations",
          "type": "AWS",
          "credentials": "arn:aws:iam::123456789012:role/Lambda",
          "httpMethod": "POST",
          "contentHandling" : "CONVERT_TO_TEXT",
          "requestTemplates": {
            "application/json": "{\n   \"imageKey\": \"$input.params('key')\",
 \"image\": \"$input.body\"\n}"
          },
          "responses": {
            "default": {
              "statusCode": "500"
            },
            "2\\d{2}": {
              "statusCode": "200"
            }
          }
        }
      }
    },
    "x-amazon-apigateway-binary-media-types" : ["application/octet-stream", "image/
jpeg"],
    "definitions": {
      "Empty": {
        "type": "object",
        "title": "Empty Schema"
      }
    }
```

```
}
```

Téléchargement d'une image depuis Lambda

Pour télécharger un fichier image (`image.jpg`) comme blob binaire depuis Lambda :

```
GET /v1/lambda?key=image.jpg HTTP/1.1
Host: abcdefghi.execute-api.us-east-1.amazonaws.com
Content-Type: application/json
Accept: application/octet-stream
```

La réponse de réussite ressemble à ce qui suit :

```
200 OK HTTP/1.1

[raw bytes]
```

Pour télécharger un fichier image (`image.jpg`) sous la forme d'une chaîne encodée en base64, au format d'une propriété JSON, depuis Lambda :

```
GET /v1/lambda?key=image.jpg HTTP/1.1
Host: abcdefghi.execute-api.us-east-1.amazonaws.com
Content-Type: application/json
Accept: application/json
```

La réponse de réussite ressemble à ce qui suit :

```
200 OK HTTP/1.1

{
  "image": "W3JhdyBieXRlc10="
}
```

Chargement d'une image vers Lambda

Pour charger un fichier image (`image.jpg`) comme blob binaire vers Lambda :

```
PUT /v1/lambda?key=image.jpg HTTP/1.1
Host: abcdefghi.execute-api.us-east-1.amazonaws.com
Content-Type: application/octet-stream
```

```
Accept: application/json

[raw bytes]
```

La réponse de réussite ressemble à ce qui suit :

```
200 OK
```

Pour charger un fichier image (`image.jpg`) sous la forme d'une chaîne encodée en base64 vers Lambda :

```
PUT /v1/lambda?key=image.jpg HTTP/1.1
Host: abcdefghi.execute-api.us-east-1.amazonaws.com
Content-Type: application/json
Accept: application/json

W3JhdyBieXRlc10=
```

La réponse de réussite ressemble à ce qui suit :

```
200 OK
```

Appel d'une API REST dans Amazon API Gateway

Pour appeler une API déployée, les clients soumettent des demandes à l'URL du service des composants API Gateway pour l'exécution de l'API, connu sous le nom de `execute-api`.

L'URL de base pour les API REST se présente au format suivant :

```
https://restapi_id.execute-api.region.amazonaws.com/stage_name/
```

où *`restapi_id`* est l'identifiant de l'API, *`region est la AWS région`* et *`stage_name est le nom`* de l'étape du déploiement de l'API.

> **Important**
>
> Avant de pouvoir invoquer une API, vous devez la déployer dans API Gateway. Pour obtenir des instructions sur le déploiement d'une API, consultezDéploiement d'une API REST dans Amazon API Gateway.

Rubriques

Obtention de l'URL d'appel d'une API

Vous pouvez utiliser la console, le AWS CLI, ou une définition OpenAPI exportée pour obtenir l'URL d'appel d'une API.

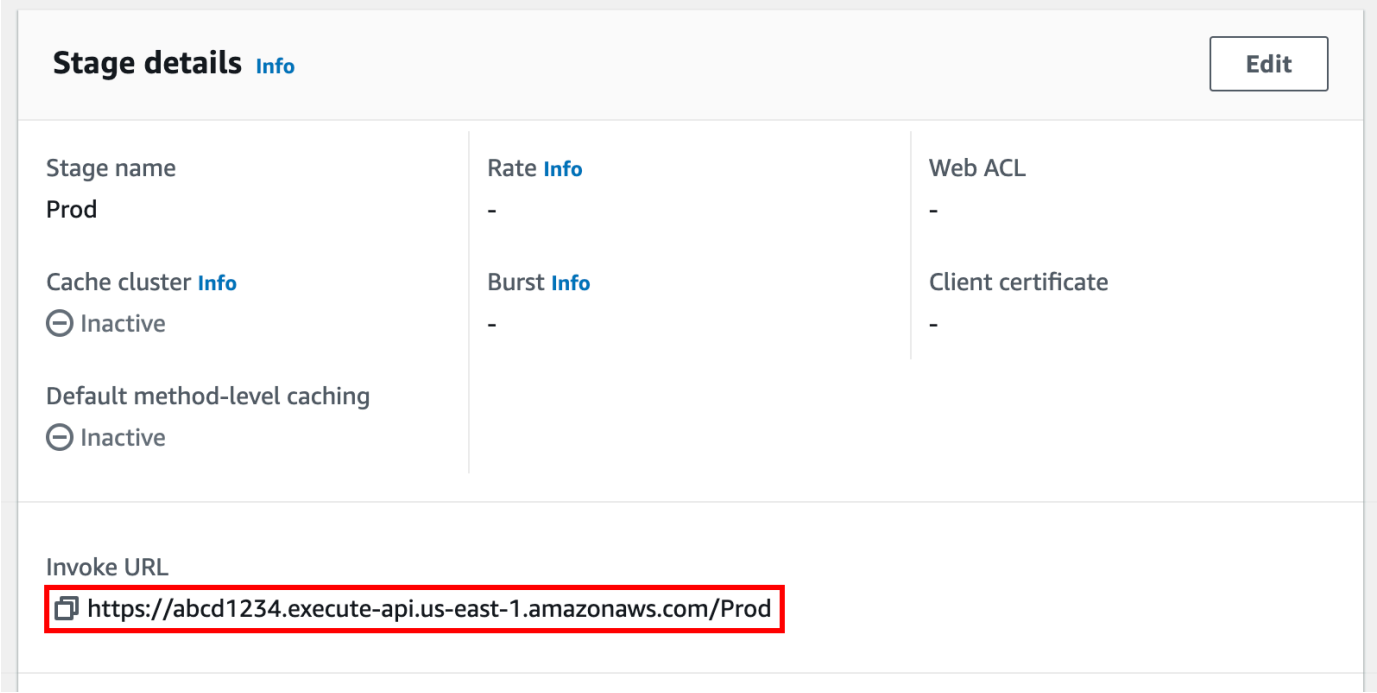
Obtention de l'URL d'appel d'une API à l'aide de la console

La procédure suivante montre comment obtenir l'URL d'appel d'une API dans la console de l'API REST.

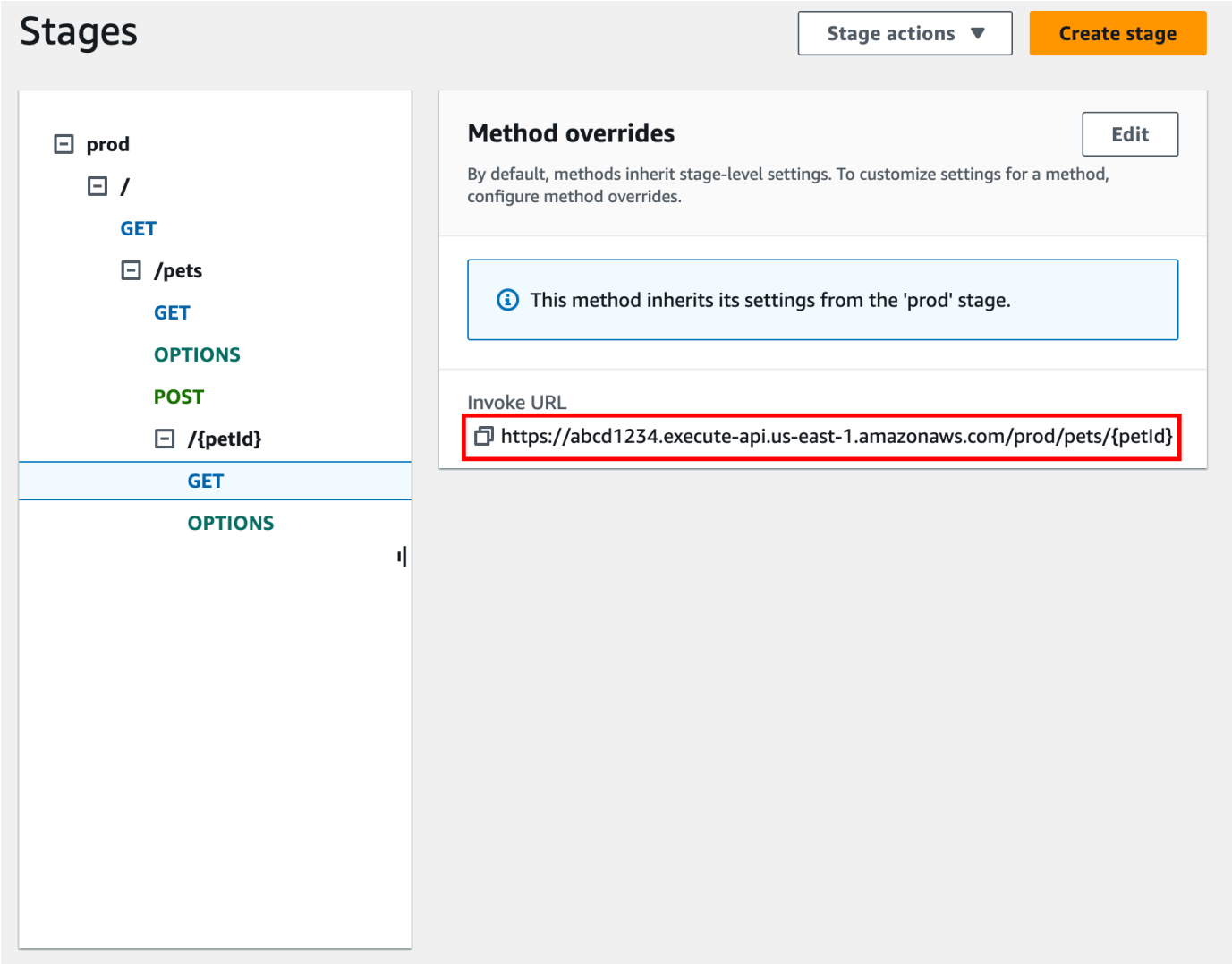
Pour obtenir l'URL d'appel d'une API à l'aide de la console d'API REST

1. Connectez-vous à la console API Gateway à l'adresse : https://console.aws.amazon.com/apigateway.
2. Choisissez une API déployée.
3. Dans le panneau de navigation principal, choisissez Étape.
4. Sous Détails de l'étape, choisissez l'icône de copie pour copier l'URL d'invocation de votre API.

   Cette URL est destinée à la ressource racine de votre API.



5. Pour obtenir l'URL d'appel d'une API pour une autre ressource de votre API, développez la scène sous le volet de navigation secondaire, puis choisissez une méthode.
6. Cliquez sur l'icône de copie pour copier l'URL d'appel au niveau des ressources de votre API.



Obtention de l'URL d'appel d'une API à l'aide du AWS CLI

La procédure suivante montre comment obtenir l'URL d'appel d'une API à l'aide du AWS CLI.

Pour obtenir l'URL d'appel d'une API à l'aide du AWS CLI

1. Utilisez la commande suivante pour obtenir le`rest-api-id`. Cette commande renvoie toutes les `rest-api-id` valeurs de votre région. Pour plus d'informations, consultez get-rest-apis.

```
aws apigateway get-rest-apis
```

2. Remplacez l'exemple `rest-api-id` par votre`rest-api-id`, remplacez l'exemple *`{stage-name}`* par votre *`{stage-name}`*, et remplacez *`{region} par votre région`*.

```
https://{restapi_id}.execute-api.{region}.amazonaws.com/{stage_name}/
```

Obtention de l'URL d'appel d'une API à l'aide du fichier de définition OpenAPI exporté de l'API

Vous pouvez également créer l'URL racine en combinant les `basePath` champs `host` et d'un fichier de définition OpenAPI exporté de l'API. Pour obtenir des instructions sur la façon d'exporter votre API, consultezthe section called “Exportation d'une API REST”.

Appel d'une API

Vous pouvez appeler votre API déployée à l'aide du navigateur, de curl ou d'autres applications, telles que Postman.

En outre, vous pouvez utiliser la console API Gateway pour tester un appel d'API. Le test utilise la `TestInvoke` fonctionnalité d'API Gateway, qui permet de tester l'API avant le déploiement de l'API. Pour plus d’informations, consultez the section called “Utilisez la console pour tester une méthode de l'API REST.”.

> **Note**
>
> Les valeurs de paramètres de chaîne de requêtes d'une URL d'invocation ne peuvent pas contenir `%%`.

Invoquer une API à l'aide d'un navigateur Web

Si votre API autorise un accès anonyme, vous pouvez utiliser n'importe quel navigateur Web pour appeler n'importe quelle `GET` méthode. Entrez l'URL d'appel complète dans la barre d'adresse du navigateur.

Pour les autres méthodes ou les appels nécessitant une authentification, vous devez spécifier une charge utile ou signer les demandes. Vous pouvez les gérer dans un script derrière une page HTML ou dans une application cliente à l'aide de l'un des AWS SDK.

Invoquer une API à l'aide de curl

Vous pouvez utiliser un outil tel que curl dans votre terminal pour appeler votre API. L'exemple de commande curl suivant invoque la méthode GET sur la `getUsers` ressource du `prod` stage d'une API.

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"
```

Utilisation de la console API Gateway pour tester une méthode API REST

Utilisez la console API Gateway pour tester une méthode API REST.

Rubriques

- Prérequis
- Test d'une méthode avec la console API Gateway

Prérequis

- Vous devez spécifier les paramètres des méthodes que vous souhaitez tester. Suivez les instructions de la section Méthodes pour les API REST dans API Gateway.

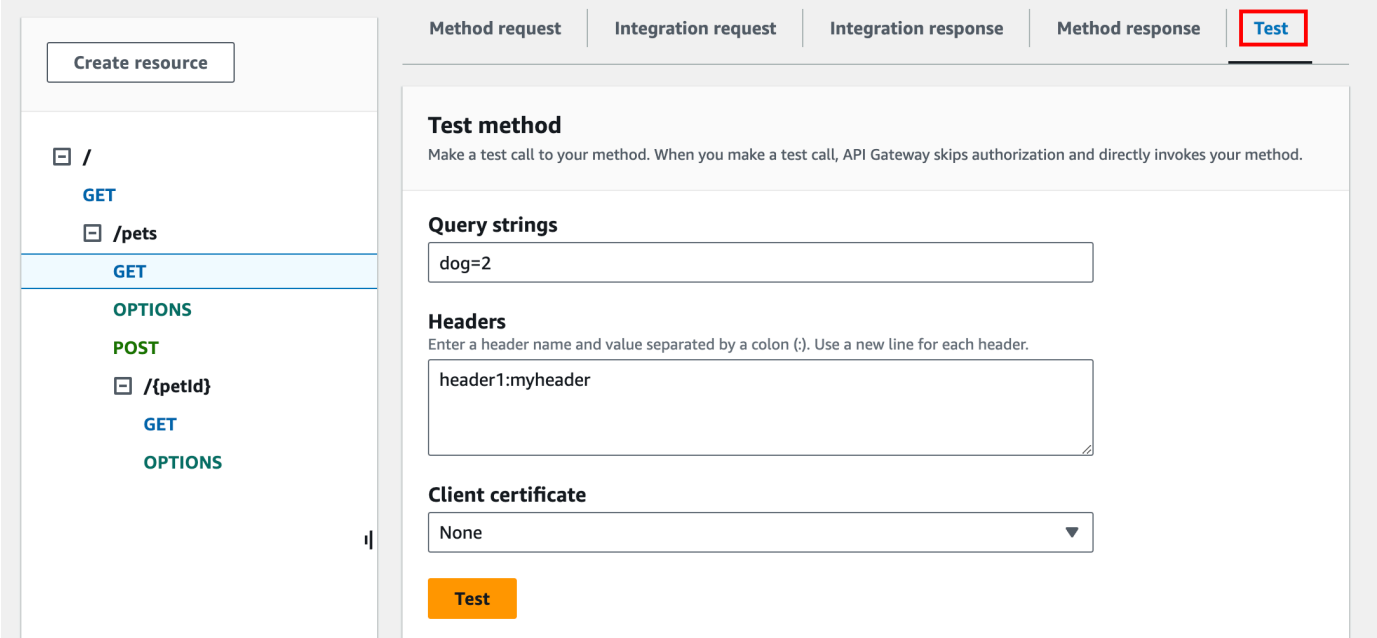
Test d'une méthode avec la console API Gateway

> **Important**
>
> Le test des méthodes avec la console API Gateway peut entraîner des modifications de ressources impossibles à annuler. Tester une méthode avec la console API Gateway revient à appeler la méthode en dehors de la console API Gateway. Par exemple, si vous utilisez la console API Gateway pour appeler une méthode qui supprime les ressources d'une API, si l'appel de la méthode aboutit, les ressources de l'API sont supprimées.

Pour tester une méthode

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.

2. Choisissez une API REST.
3. Dans le volet Resources, sélectionnez la méthode à tester.
4. Choisissez l'onglet Test. Vous devrez peut-être choisir la flèche droite pour afficher l'onglet.



Entrez des valeurs dans les zones affichées (par exemple, Chaînes de requête, En-têtes et Corps de la demande). La console inclut ces valeurs dans le formulaire application/json par défaut de la demande de méthode.

Pour obtenir des options supplémentaires, vous devrez peut-être spécifier ou contacter le propriétaire de l'API.

5. Sélectionnez Test. Les informations suivantes s'affichent alors :

   - Request est le chemin de la ressource qui a été appelée pour la méthode.
   - Status est le code d'état HTTP de la réponse.
   - La latence (ms) est le temps entre la réception de la demande de l'appelant et la réponse renvoyée.
   - Corps de la réponse correspond au corps de la réponse HTTP.
   - Le paramètre En-têtes de réponse correspond aux en-têtes de réponse HTTP.

> **Tip**
>
> Selon le mappage, le code d'état HTTP, le corps de la réponse et les en-têtes de réponse peuvent être différents de ceux envoyés par la fonction Lambda, le proxy HTTP AWS ou le proxy de service.

- Les journaux sont les entrées Amazon CloudWatch Logs simulées qui auraient été écrites si cette méthode avait été appelée en dehors de la console API Gateway.

> **Note**
>
> Bien que les entrées CloudWatch Logs soient simulées, les résultats de l'appel de méthode sont réels.

Outre l'utilisation de la console API Gateway, vous pouvez utiliser AWS CLI un AWS SDK pour API Gateway afin de tester l'appel d'une méthode. Pour ce faire en utilisant AWS CLI, voir test-invoke-method.

Utilisation d'un kit SDK Java généré par API Gateway pour une API REST

Dans cette section, nous présentons les étapes relatives à l'utilisation d'un kit SDK Java généré par API Gateway pour une API REST en utilisant l'API Simple Calculator comme exemple. Avant de poursuivre, vous devez suivre la procédure indiquée dans Génération d'un kit SDK pour une API REST dans API Gateway.

Pour installer et utiliser un kit SDK Java généré par API Gateway

1. Extrayez le contenu du fichier .zip généré par API Gateway que vous avez téléchargé plus tôt.
2. Téléchargez et installez Apache Maven (version 3.5 ou ultérieure).
3. Téléchargez et installez le kit JDK 8.
4. Définissez la variable d'environnement `JAVA_HOME`.
5. Accédez au dossier SDK décompressé où se trouve le fichier pom.xml. Ce dossier est `generated-code` par défaut. Exécutez la commande mvn install pour installer les fichiers d'artefact compilés dans votre référentiel Maven local. Cela crée un dossier `target` contenant la bibliothèque de kits SDK compilés.

6. Entrez la commande suivante dans un répertoire vide pour créer un stub de projet client et appeler l'API à l'aide de la bibliothèque de kit SDK installée.

```
mvn -B archetype:generate \
    -DarchetypeGroupdId=org.apache.maven.archetypes \
    -DgroupId=examples.aws.apig.simpleCalc.sdk.app \
    -DartifactId=SimpleCalc-sdkClient
```

> **Note**
>
> Le séparateur \ de la commande précédente est inclus pour faciliter la lecture. La commande entière doit figurer sur une seule ligne sans le séparateur.

Cette commande crée un stub d'application. Le stub de l'application contient un `pom.xml` fichier et un `src` dossier situés dans le répertoire racine du projet (*`SimpleCalc-SdkClient`* dans la commande précédente). A l'origine, il existe deux fichiers sources : `src/main/java/`*`{package-path}`*`/App.java` et `src/test/java/`*`{package-path}`*`/AppTest.java`. Dans cet exemple, le *`{chemin d'accès au package}`* est `examples/aws/apig/simpleCalc/sdk/app`. Ce chemin d'accès au package provient de la valeur `DarchetypeGroupdId`. Vous pouvez utiliser le fichier `App.java` comme modèle pour votre application cliente et vous pouvez en ajouter d'autres dans le même dossier si nécessaire. Vous pouvez utiliser le fichier `AppTest.java` comme modèle de test unitaire pour votre application et vous pouvez ajouter d'autres fichiers de code de test dans le même dossier de test si nécessaire.

7. Mettez à jour les dépendances de package dans le fichier `pom.xml` généré avec ce qui suit, en remplaçant les propriétés `groupId`, `artifactId`, `version` et `name` de votre projet si nécessaire :

```
<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**
>
> Lorsqu'une version plus récente de l'artéfact dépendant de `aws-java-sdk-core` n'est pas compatible avec la version indiquée ci-dessus (`1.11.94`), vous devez mettre à jour la balise `<version>` vers la nouvelle version.

8. Ensuite, nous montrons comment appeler l'API à l'aide du kit SDK en appelant les méthodes `getABOp(GetABOpRequest req)`, `getApiRoot(GetApiRootRequest req)` et `postApiRoot(PostApiRootRequest req)` du kit SDK. Ces méthodes correspondent aux méthodes `GET /{a}/{b}/{op}`, `GET /?a={x}&b={y}&op={operator}`, et `POST /`, avec, respectivement, la charge utile des demandes d'API `{"a": x, "b": y, "op": "operator"}`.

   Mettez à jour le fichier `App.java` comme suit :

```
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());


    }
}
```

Dans l'exemple précédent, les paramètres de configuration utilisés pour instancier le client du kit SDK le sont à titre d'illustration et ne sont pas nécessairement la bonne pratique recommandée. En outre, l'appel de `sdkClient.shutdown()`est facultatif, surtout si vous avez besoin d'un contrôle précis du moment où libérer des ressources.

Nous avons montré les modèles essentiels pour appeler une API à l'aide d'un kit SDK Java. Vous pouvez étendre les instructions à l'appel d'autres méthodes de l'API.

Utilisation d'un kit SDK Android généré par API Gateway pour une API REST

Dans cette section, nous décrivons les étapes pour utiliser un kit SDK Android généré par API Gateway pour une API REST. Avant de continuer, vous devez avoir déjà suivi la procédure de Génération d'un kit SDK pour une API REST dans API Gateway.

> **Note**
>
> Le kit SDK généré n'est pas compatible avec Android 4.4 et versions antérieures. Pour plus d'informations, consultez the section called “Remarques importantes”.

Pour installer et utiliser un kit SDK Android généré par API Gateway

1. Extrayez le contenu du fichier .zip généré par API Gateway que vous avez téléchargé plus tôt.

2. Téléchargez et installez Apache Maven (de préférence version 3.x).
3. Téléchargez et installez le kit JDK 8.
4. Définissez la variable d'environnement `JAVA_HOME`.
5. Exécutez la commande mvn install pour installer les fichiers d'artefact compilés dans votre référentiel Maven local. Cela crée un dossier `target` contenant la bibliothèque de kits SDK compilés.
6. Copiez le fichier du kit SDK (dont le nom est dérivé des valeurs que vous avez spécifiées dans les champs ID d'artifact et Version d'artifact lors de la génération du kit SDK, par exemple,`simple-calcsdk-1.0.0.jar`) du dossier `target`, ainsi que toutes les autres bibliothèques du dossier `target/lib`, dans le dossier `lib` de votre projet.

   Si vous utilisez Android Studio, créez un dossier `libs` sous le module de votre application client et copiez le fichier .jar requis dans ce dossier. Vérifiez que la section des dépendances dans le fichier gradle du module contient le code suivant.

   ```
       compile fileTree(include: ['*.jar'], dir: 'libs')
       compile fileTree(include: ['*.jar'], dir: 'app/libs')
   ```

   Assurez-vous qu'aucun fichier .jar dupliqué n'est déclaré.
7. Utilisez la classe `ApiClientFactory` pour initialiser le kit SDK généré par API Gateway. Par exemple :

   ```
   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. Pour utiliser un fournisseur d'informations d'identification Amazon Cognito afin d'autoriser les appels à votre API, utilisez la `ApiClientFactory` classe pour transmettre un ensemble d' AWS informations d'identification à l'aide du SDK généré par API Gateway, comme illustré dans l'exemple suivant.

```
// 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. Pour définir une clé API à l'aide du kit SDK généré par API Gateway, utilisez un code similaire au code suivant.

```
ApiClientFactory factory = new ApiClientFactory()
  .apiKey("YOUR_API_KEY");
```

Utiliser un JavaScript SDK généré par API Gateway pour une API REST

> **Note**
>
> Ces instructions supposent que vous avez déjà suivi les instructions de Génération d'un kit SDK pour une API REST dans API Gateway.

> **Important**
>
> Si seules des méthodes ANY sont définies pour votre API, le kit SDK généré ne contient aucun fichier `apigClient.js` et vous devez définir les méthodes ANY vous-même.

Pour installer, lancer et appeler un JavaScript SDK généré par API Gateway pour une API REST

1. Extrayez le contenu du fichier .zip généré par API Gateway que vous avez précédemment téléchargé.
2. Activez le partage des ressources cross-origin (CORS) pour toutes les méthodes que le kit SDK généré par API Gateway appelle. Pour obtenir des instructions, veuillez consulter Activation de CORS pour une ressource de l'API REST.
3. Dans votre page web, incluez des références aux scripts suivants.

```
<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. Dans votre code, initialisez le kit SDK généré par API Gateway à l'aide d'un code similaire à ce qui suit.

```
var apigClient = apigClientFactory.newClient();
```

Pour initialiser le SDK généré par API Gateway avec des AWS informations d'identification, utilisez un code similaire au suivant. Si vous utilisez des AWS informations d'identification, toutes les demandes adressées à l'API seront signées.

```
var apigClient = apigClientFactory.newClient({
  accessKey: 'ACCESS_KEY',
  secretKey: 'SECRET_KEY',
});
```

Pour utiliser une clé API avec le kit SDK généré par API Gateway, vous pouvez transmettre la clé API à l'objet `Factory` en tant que paramètre à l'aide d'un code similaire à ce qui suit. Si vous utilisez une clé API, elle est spécifiée dans l'en-tête `x-api-key` et toutes les demandes envoyées à l'API sont signées. Cela signifie que vous devez définir les en-têtes CORS Accept appropriés pour chaque demande.

```
var apigClient = apigClientFactory.newClient({
  apiKey: 'API_KEY'
});
```

5. Appelez les méthodes d'API dans API Gateway à l'aide d'un code similaire à ce qui suit. Chaque appel renvoie une promesse avec des rappels de réussite et d'échec.

```
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.
    });
```

Ici, le nom *methodName* est construit à partir du chemin d'accès des ressources de la demande de méthode et du verbe HTTP. Pour l' SimpleCalc API, les méthodes du SDK pour les méthodes d'API de

```
1. GET /?a=...&b=...&op=...
2. POST /

   { "a": ..., "b": ..., "op": ...}
```

```
3. GET /{a}/{b}/{op}
```

les méthodes de kit SDK correspondantes sont les suivantes :

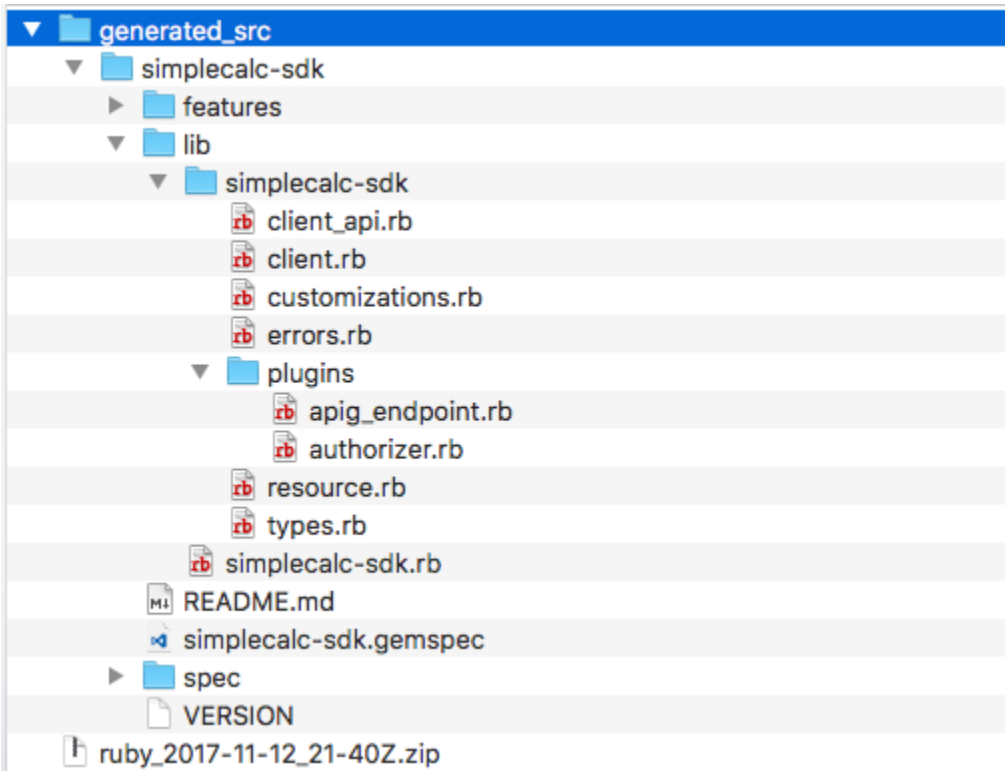
```
1. rootGet(params);      // where params={"a": ..., "b": ..., "op": ...} is
 resolved to the query parameters
2. rootPost(null, body); // where body={"a": ..., "b": ..., "op": ...}
3. aBOpGet(params);      // where params={"a": ..., "b": ..., "op": ...} is
 resolved to the path parameters
```

Utilisation d'un kit SDK Ruby généré par API Gateway pour une API REST

> **Note**
>
> Ces instructions supposent que vous avez déjà suivi les instructions de Génération d'un kit SDK pour une API REST dans API Gateway.

Pour installer, instancier et appeler un kit SDK Ruby généré par API Gateway pour une API REST

1. Décompressez le fichier du kit SDK Ruby téléchargé. La source du kit SDK généré est affichée comme suit.



2. Créez un gem Ruby à partir de la source du kit SDK généré, à l'aide des commandes shell suivantes dans une fenêtre de terminal :

```
# change to /simplecalc-sdk directory
cd simplecalc-sdk

# build the generated gem
gem build simplecalc-sdk.gemspec
```

   Ensuite, simplecalc-sdk-1.0.0.gem devient disponible.

3. Installation du gem :

```
gem install simplecalc-sdk-1.0.0.gem
```

4. Créez une application client. Instanciez et initialisez le client du kit SDK Ruby dans l'application :

```
require 'simplecalc-sdk'
client = SimpleCalc::Client.new(
    http_wire_trace: true,
    retry_limit: 5,
    http_read_timeout: 50
```

```
)
```

Si l'API dispose d'une autorisation du `AWS_IAM` type configuré, vous pouvez inclure les informations d' AWS identification de l'appelant en fournissant `accessKey` et `secretKey` lors de l'initialisation :

```
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. Effectuez des appels d'API via le kit SDK dans l'application.

> **Tip**
>
> Si vous n'êtes pas familiarisé avec les conventions d'appel de méthode de kit SDK, vous pouvez consulter le fichier `client.rb` dans le dossier `lib` du kit SDK généré. Le dossier contient la documentation de chaque appel de méthode d'API pris en charge.

Pour connaître les opérations prises en charge :

```
# to show supported operations:
puts client.operation_names
```

Cela génère l'affichage suivant, qui correspond aux méthodes d'API de `GET /?a={.}&b={.}&op={.}`, `GET /{a}/{b}/{op}` et `POST /`, ainsi qu'une charge utile au format `{a:"…", b:"…", op:"…"}`, respectivement :

```
[:get_api_root, :get_ab_op, :post_api_root]
```

Pour appeler la méthode d'API `GET /?a=1&b=2&op=+`, appelez la méthode de kit SDK Ruby suivante :

```
resp = client.get_api_root({a:"1", b:"2", op:"+"})
```

Pour appeler la méthode d'API `POST /` avec une charge utile de `{a: "1", b: "2", "op": "+"}`, appelez la méthode de kit SDK Ruby suivante :

```
resp = client.post_api_root(input: {a:"1", b:"2", op:"+"})
```

Pour appeler la méthode d'API `GET /1/2/+`, appelez la méthode de kit SDK Ruby suivante :

```
resp = client.get_ab_op({a:"1", b:"2", op:"+"})
```

Les appels de méthode de kit SDK réussies renvoient la réponse suivante :

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

Utilisation d'un kit SDK iOS généré par API Gateway pour une API REST dans Objective-C ou Swift

Dans ce tutoriel, nous vous expliquons comment utiliser un kit SDK iOS généré à l'aide d'API Gateway pour une API REST dans une application Objective-C ou Swift pour appeler l'API sous-jacente. Nous utiliserons l'SimpleCalc API comme exemple pour illustrer les sujets suivants :

- Comment installer les composants du SDK AWS mobile requis dans votre projet Xcode
- Création de l'objet de client d'API avant d'appeler les méthodes de l'API
- Appel des méthodes d'API via les méthodes de kit SDK correspondantes sur l'objet de client d'API

- Préparation d'une entrée de méthode et analyse de son résultat à l'aide des classes de modèle correspondantes du kit SDK

Rubriques

Utilisation d'un kit SDK iOS généré (Objective-C) pour appeler une API

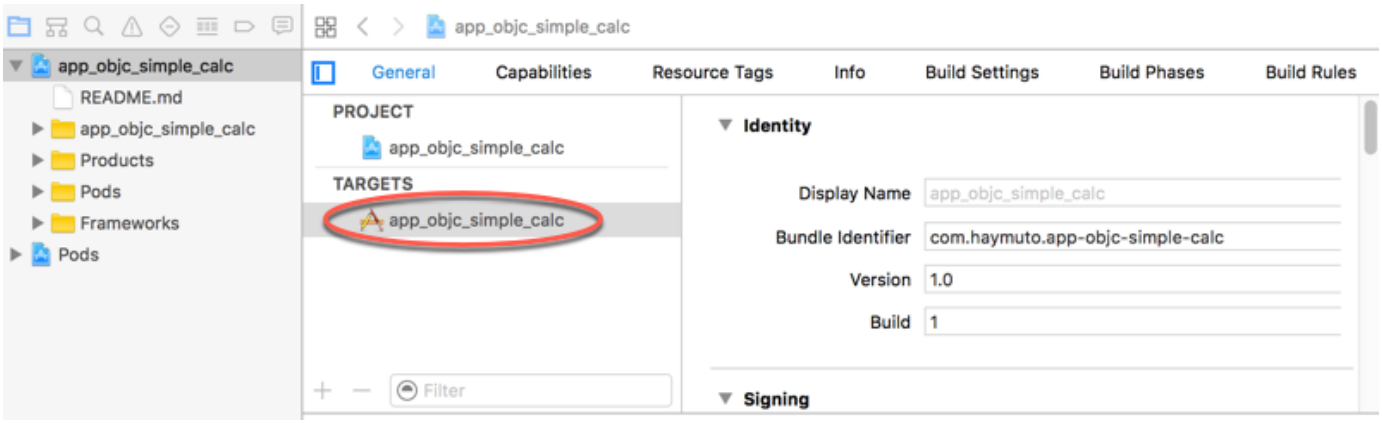
Avant de commencer la procédure suivante, vous devez suivre la procédure décrite dans Génération d'un kit SDK pour une API REST dans API Gateway pour iOS dans Objective-C et télécharger le fichier .zip du kit SDK généré.

Installer le SDK AWS mobile et un SDK iOS générés par API Gateway dans un projet Objective-C

La procédure suivante décrit comment installer le kit SDK.

Pour installer et utiliser un kit SDK iOS généré par API Gateway dans Objective-C

1. Extrayez le contenu du fichier .zip généré par API Gateway que vous avez précédemment téléchargé. À l'aide de l'SimpleCalc API, vous souhaiterez peut-être renommer le dossier du SDK décompressé en un nom similaire à. **`sdk_objc_simple_calc`** Dans ce dossier de kit SDK se trouvent un fichier `README.md` et un fichier `Podfile`. Le fichier `README.md` contient les instructions pour installer et utiliser le kit SDK. Ce didacticiel fournit les détails de ces instructions. L'installation s'appuie sur CocoaPodsl'importation des bibliothèques API Gateway requises et d'autres composants du SDK AWS mobile dépendants. Vous devez mettre à jour le fichier `Podfile` pour importer les kits SDK dans le projet Xcode de votre application. Le dossier de kit SDK décompressé contient également un dossier `generated-src` qui contient le code source du kit SDK généré de votre API.
2. Lancez Xcode et créez un projet iOS Objective-C. Notez la cible du projet. Vous en avez besoin pour la définir dans le fichier `Podfile`.



3. Pour l'importer AWS Mobile SDK for iOS dans le projet Xcode à l'aide de CocoaPods, procédez comme suit :

   a. Effectuez l'installation en CocoaPods exécutant la commande suivante dans une fenêtre de terminal :

   ```
   sudo gem install cocoapods
   pod setup
   ```

   b. Copiez le fichier `Podfile` à partir du dossier de kit SDK extrait dans le même répertoire contenant votre fichier de projet Xcode. Remplacez le bloc suivant :

   ```
   target '<YourXcodeTarget>' do
       pod 'AWSAPIGateway', '~> 2.4.7'
   end
   ```

   avec le nom cible de votre projet :

   ```
   target 'app_objc_simple_calc' do
       pod 'AWSAPIGateway', '~> 2.4.7'
   end
   ```

   Si votre projet Xcode contient déjà un fichier nommé `Podfile`, ajoutez-lui la ligne de code suivante :

   ```
   pod 'AWSAPIGateway', '~> 2.4.7'
   ```

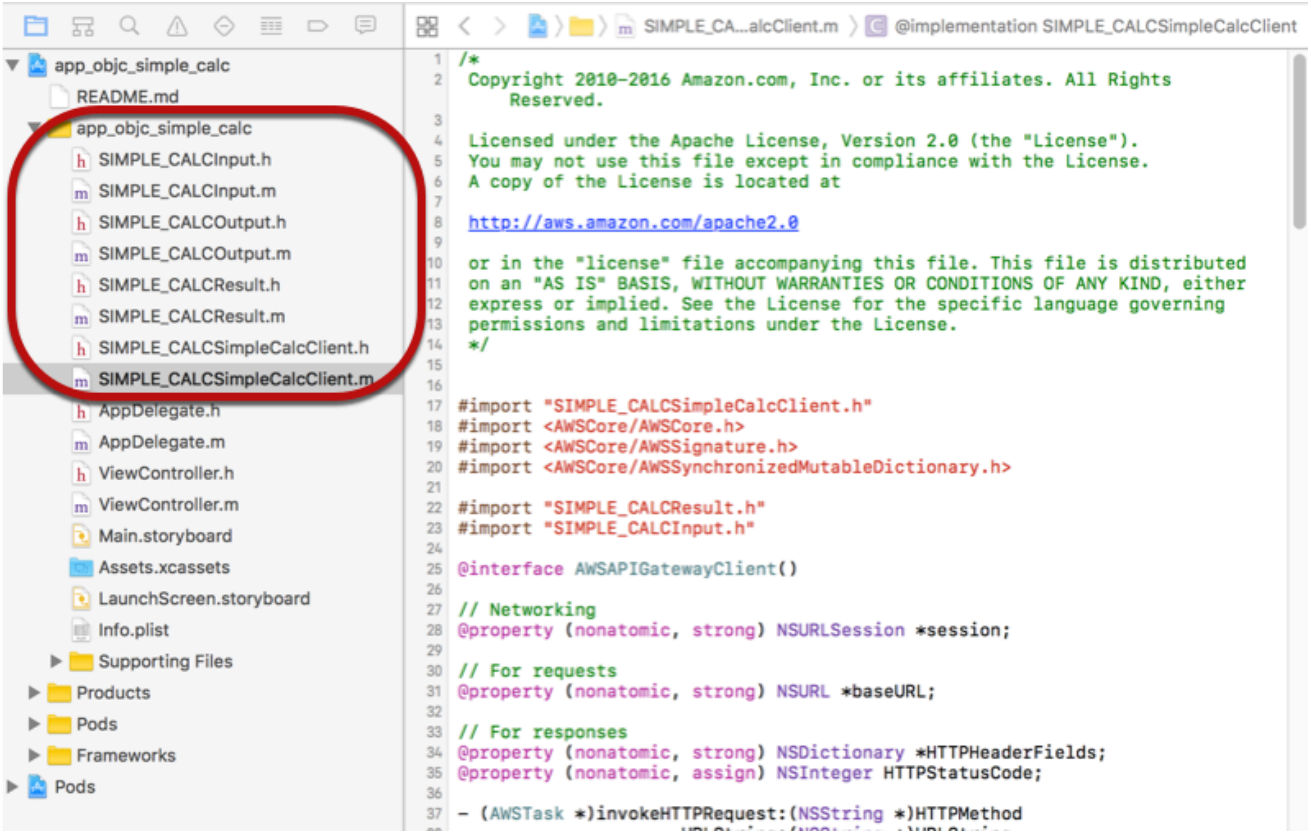
   c. Ouvrez une fenêtre de terminal et exécutez la commande suivante :

```
pod install
```

Cela installe le composant API Gateway et les autres composants du SDK AWS mobile dépendants.

d. Fermez le projet Xcode et ouvrez le fichier `.xcworkspace` pour relancer Xcode.

e. Ajoutez tous les fichiers `.h` et `.m` du répertoire `generated-src` du kit SDK extrait dans votre projet Xcode.



Pour importer l' AWS Mobile SDK for iOS Objective-C dans votre projet en téléchargeant explicitement le SDK AWS mobile ou en utilisant Carthage, suivez les instructions du fichier README.md. Veillez à n'utiliser qu'une seule de ces options pour importer le SDK AWS mobile.

Appel des méthodes d'API à l'aide du kit SDK iOS généré par API Gateway dans un projet Objective-C

Lorsque vous avez généré le SDK avec le préfixe `SIMPLE_CALC` for this SimpleCalc API avec deux modèles pour l'entrée (`Input`) et la sortie (`Result`) des méthodes, dans le SDK, la classe client d'API résultante devient `SIMPLE_CALCSimpleCalcClient` et, respectivement, `SIMPLE_CALCInput` et `SIMPLE_CALCResult` les classes de données correspondantes. Les demandes et réponses d'API sont mappées aux méthodes de kit SDK comme suit :

- La demande d'API

  ```
  GET /?a=...&b=...&op=...
  ```

  devient la méthode du kit SDK

  ```
  (AWSTask *)rootGet:(NSString *)op a:(NSString *)a b:(NSString *)b
  ```

  La propriété `AWSTask.result` est de type `SIMPLE_CALCResult` si le modèle `Result` a été ajouté à la réponse de méthode. Sinon, la propriété est de type `NSDictionary`.

- Cette demande d'API

  ```
  POST /

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

  devient la méthode du kit SDK

  ```
  (AWSTask *)rootPost:(SIMPLE_CALCInput *)body
  ```

- La demande d'API

  ```
  GET /{a}/{b}/{op}
  ```

  devient la méthode du kit SDK

```
(AWSTask *)aBOpGet:(NSString *)a b:(NSString *)b op:(NSString *)op
```

La procédure suivante décrit comment appeler les méthodes d'API dans le code source de l'application Objective-C, par exemple, comme faisant partie du délégué `viewDidLoad` dans un fichier `ViewController.m`.

Pour appeler l'API via le kit SDK iOS généré par API Gateway

1. Importez le fichier d'en-tête de la classe de client d'API pour que la classe de client d'API puisse être appelée dans l'application :

   ```
   #import "SIMPLE_CALCSimpleCalc.h"
   ```

   La déclaration `#import` importe également `SIMPLE_CALCInput.h` et `SIMPLE_CALCResult.h` pour les deux classes de modèle.

2. Instanciez la classe de client d'API :

   ```
   SIMPLE_CALCSimpleCalcClient *apiInstance = [SIMPLE_CALCSimpleCalcClient
    defaultClient];
   ```

   Pour utiliser Amazon Cognito avec l'API, définissez la propriété `defaultServiceConfiguration` sur l'objet `AWSServiceManager` par défaut, comme illustré ci-après, avant d'appeler la méthode `defaultClient` pour créer l'objet de client d'API (comme illustré dans l'exemple précédent) :

   ```
   AWSCognitoCredentialsProvider *creds = [[AWSCognitoCredentialsProvider alloc]
    initWithRegionType:AWSRegionUSEast1 identityPoolId:your_cognito_pool_id];
   AWSServiceConfiguration *configuration = [[AWSServiceConfiguration alloc]
    initWithRegion:AWSRegionUSEast1 credentialsProvider:creds];
   AWSServiceManager.defaultServiceManager.defaultServiceConfiguration =
    configuration;
   ```

3. Appelez la méthode `GET /?a=1&b=2&op=+` pour exécuter 1+2 :

   ```
   [[apiInstance rootGet: @"+" a:@"1" b:@"2"] continueWithBlock:^id _Nullable(AWSTask
    * _Nonnull task) {
       _textField1.text = [self handleApiResponse:task];
   ```

```
    return nil;
}];
```

où `handleApiResponse:task` de la fonction d'assistant formate le résultat sous forme de chaîne pour l'afficher dans un champ de texte (`_textField1`).

```
- (NSString *)handleApiResponse:(AWSTask *)task {
    if (task.error != nil) {
        return [NSString stringWithFormat: @"Error: %@", task.error.description];
    } else if (task.result != nil && [task.result isKindOfClass:[SIMPLE_CALCResult
 class]]) {
        return [NSString stringWithFormat:@"%@ %@ %@ = %@\n",task.result.input.a,
 task.result.input.op, task.result.input.b, task.result.output.c];
    }
    return nil;
}
```

Le résultat qui s'affiche est `1 + 2 = 3`.

4. Appelez la méthode `POST /` avec une charge utile pour exécuter `1-2` :

```
SIMPLE_CALCInput *input = [[SIMPLE_CALCInput alloc] init];
    input.a = [NSNumber numberWithInt:1];
    input.b = [NSNumber numberWithInt:2];
    input.op = @"-";
    [[apiInstance rootPost:input] continueWithBlock:^id _Nullable(AWSTask *
 _Nonnull task) {
        _textField2.text = [self handleApiResponse:task];
        return nil;
    }];
```

Le résultat qui s'affiche est `1 - 2 = -1`.

5. Appelez la méthode `GET /{a}/{b}/{op}` pour exécuter `1/2` :

```
[[apiInstance aBOpGet:@"1" b:@"2" op:@"div"] continueWithBlock:^id
 _Nullable(AWSTask * _Nonnull task) {
    _textField3.text = [self handleApiResponse:task];
    return nil;
}];
```

Le résultat qui s'affiche est `1 div 2 = 0.5`. Ici, `div` est utilisé à la place de `/`, car la fonction Lambda simple du backend ne gère pas les variables de chemin d'URL codée.

Utilisation d'un kit SDK iOS généré (Swift) pour appeler une API

Avant de commencer la procédure suivante, vous devez suivre les étapes décrites dans Génération d'un kit SDK pour une API REST dans API Gateway pour iOS dans Swift et téléchargez le fichier .zip du kit SDK généré.

Rubriques

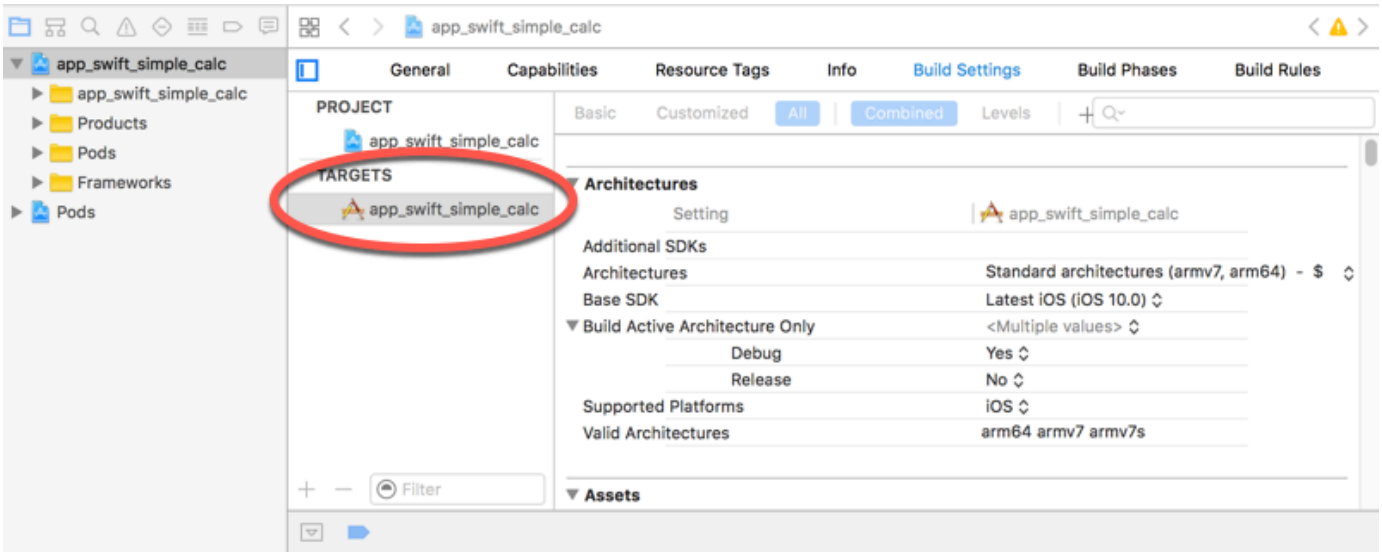
- Installation du SDK AWS mobile et du SDK généré par API Gateway dans un projet Swift
- Appel des méthodes d'API via le kit SDK iOS généré par API Gateway dans un projet Swift

Installation du SDK AWS mobile et du SDK généré par API Gateway dans un projet Swift

La procédure suivante décrit comment installer le kit SDK.

Pour installer et utiliser un kit SDK iOS généré par API Gateway dans Swift

1. Extrayez le contenu du fichier .zip généré par API Gateway que vous avez précédemment téléchargé. À l'aide de l'SimpleCalc API, vous souhaiterez peut-être renommer le dossier du SDK décompressé en un nom similaire à. **`sdk_swift_simple_calc`** Dans ce dossier de kit SDK se trouvent un fichier `README.md` et un fichier `Podfile`. Le fichier `README.md` contient les instructions pour installer et utiliser le kit SDK. Ce didacticiel fournit les détails de ces instructions. L'installation s'appuie sur l'importation CocoaPodsdes composants du SDK AWS mobile requis. Vous devez mettre à jour le fichier `Podfile` pour importer les kits SDK dans le projet Xcode de votre application Swift. Le dossier de kit SDK décompressé contient également un dossier `generated-src` qui contient le code source du kit SDK généré de votre API.
2. Lancez Xcode et créez un projet iOS Swift. Notez la cible du projet. Vous en avez besoin pour la définir dans le fichier `Podfile`.



3. Pour importer les composants du SDK AWS mobile requis dans le projet Xcode à l'aide de CocoaPods, procédez comme suit :

   a. S'il n'est pas installé, installez-le CocoaPods en exécutant la commande suivante dans une fenêtre de terminal :

   ```
   sudo gem install cocoapods
   pod setup
   ```

   b. Copiez le fichier `Podfile` à partir du dossier de kit SDK extrait dans le même répertoire contenant votre fichier de projet Xcode. Remplacez le bloc suivant :

   ```
   target '<YourXcodeTarget>' do
       pod 'AWSAPIGateway', '~> 2.4.7'
   end
   ```

   avec le nom cible de votre projet comme indiqué :

   ```
   target 'app_swift_simple_calc' do
       pod 'AWSAPIGateway', '~> 2.4.7'
   end
   ```

   Si votre fichier de projet Xcode contient un fichier nommé `Podfile` avec la cible correcte, vous pouvez simplement ajouter la ligne de code suivante à la boucle do `...` end :

   ```
   pod 'AWSAPIGateway', '~> 2.4.7'
   ```

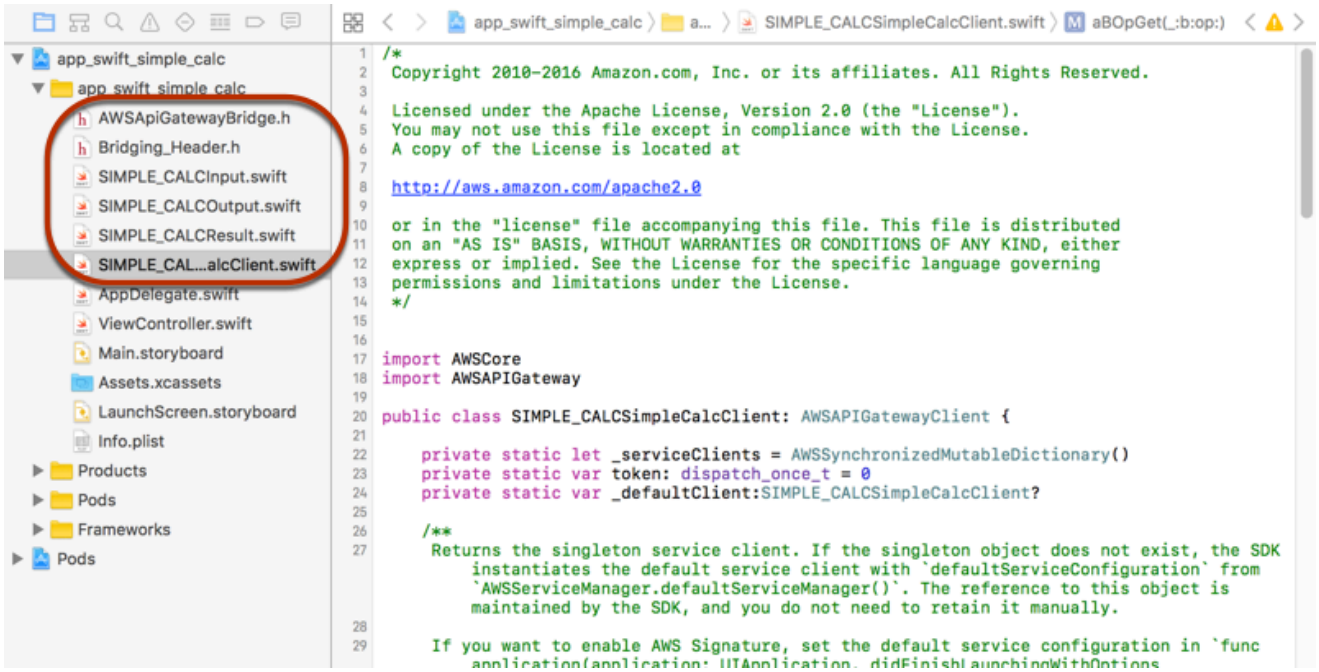
c. Ouvrez une fenêtre de terminal et exécutez la commande suivante dans le répertoire de l'application :

```
pod install
```

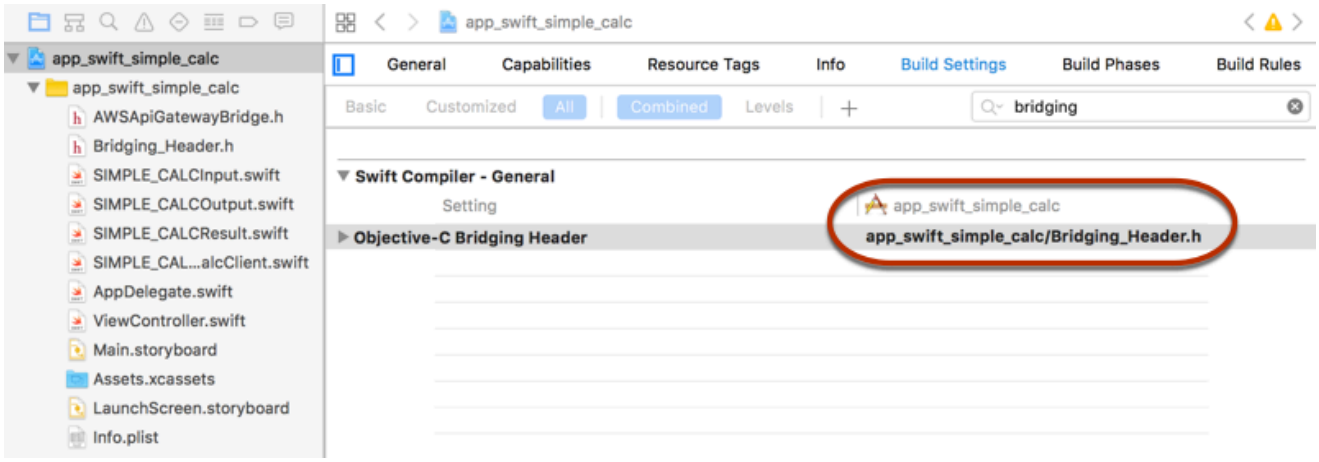
Cela installe le composant API Gateway et tous les composants du SDK AWS mobile dépendants dans le projet de l'application.

d. Fermez le projet Xcode et ouvrez le fichier `*.xcworkspace` pour relancer Xcode.

e. Ajoutez tous les fichiers d'en-tête du kit SDK (`.h`) et les fichiers de code source Swift (`.swift`) du répertoire `generated-src` extrait à votre projet Xcode.

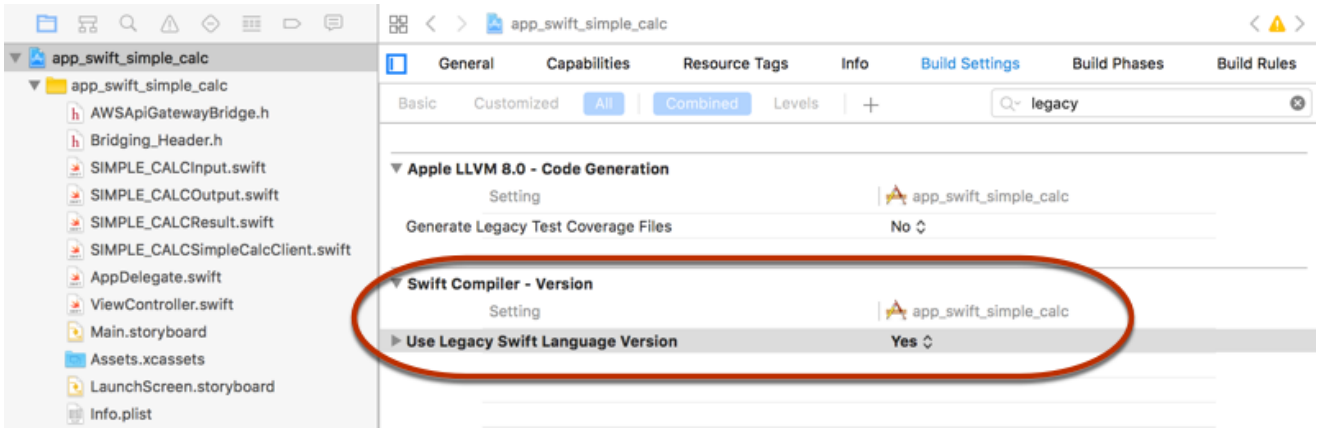


f. Pour permettre d'appeler les bibliothèques Objective-C du SDK AWS mobile à partir de votre projet de code Swift, définissez le chemin du `Bridging_Header.h` fichier sur la propriété Objective-C Bridging Header sous le compilateur Swift - Paramètre général de la configuration de votre projet Xcode :



> **Tip**
>
> Vous pouvez saisir **`bridging`** dans la zone de recherche de Xcode pour localiser la propriété Objective-C Bridging Header (En-tête de transition Objective-C).

g. Créez le projet Xcode pour vérifier qu'il est correctement configuré avant de continuer. Si votre Xcode utilise une version de Swift plus récente que celle prise en charge par le SDK AWS mobile, vous obtiendrez des erreurs de compilation Swift. Dans ce cas, définissez la propriété Use Legacy Swift Language Version sur Yes sous le paramètre Swift Compiler - Version :



Pour importer le SDK AWS mobile pour iOS dans Swift dans votre projet en téléchargeant explicitement AWS le SDK mobile ou en utilisant Carthage, suivez les instructions du fichier fourni avec `README.md` le package du SDK. Veillez à n'utiliser qu'une seule de ces options pour importer le SDK AWS mobile.

Appel des méthodes d'API via le kit SDK iOS généré par API Gateway dans un projet Swift

Lorsque vous avez généré le SDK avec le préfixe `SIMPLE_CALC` for this SimpleCalc API avec deux modèles pour décrire l'entrée (`Input`) et la sortie (`Result`) des demandes et réponses de l'API, dans le SDK, la classe client d'API résultante devient `SIMPLE_CALCSimpleCalcClient` et, respectivement, les classes de données correspondantes sont `SIMPLE_CALCInput` et`SIMPLE_CALCResult`. Les demandes et réponses d'API sont mappées aux méthodes de kit SDK comme suit :

- La demande d'API

  ```
  GET /?a=...&b=...&op=...
  ```

  devient la méthode du kit SDK

  ```
  public func rootGet(op: String?, a: String?, b: String?) -> AWSTask
  ```

  La propriété `AWSTask.result` est de type `SIMPLE_CALCResult` si le modèle `Result` a été ajouté à la réponse de méthode. Sinon, elle est de type `NSDictionary`.

- Cette demande d'API

  ```
  POST /

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

  devient la méthode du kit SDK

  ```
  public func rootPost(body: SIMPLE_CALCInput) -> AWSTask
  ```

- La demande d'API

  ```
  GET /{a}/{b}/{op}
  ```

  devient la méthode du kit SDK

```
public func aBOpGet(a: String, b: String, op: String) -> AWSTask
```

La procédure suivante décrit comment appeler les méthodes d'API dans le code source de l'application Swift, par exemple, comme faisant partie du délégué `viewDidLoad()` dans un fichier `ViewController.m`.

Pour appeler l'API via le kit SDK iOS généré par API Gateway

1. Instanciez la classe de client d'API :

   ```
   let client = SIMPLE_CALCSimpleCalcClient.default()
   ```

   Pour utiliser Amazon Cognito avec l'API, définissez une configuration de AWS service par défaut (illustrée ci-dessous) avant d'obtenir la `default` méthode (illustrée précédemment) :

   ```
   let credentialsProvider =
    AWSCognitoCredentialsProvider(regionType: AWSRegionType.USEast1, identityPoolId:
    "my_pool_id")
   let configuration = AWSServiceConfiguration(region: AWSRegionType.USEast1,
    credentialsProvider: credentialsProvider)
   AWSServiceManager.defaultServiceManager().defaultServiceConfiguration =
    configuration
   ```

2. Appelez la méthode `GET /?a=1&b=2&op=+` pour exécuter 1+2 :

   ```
   client.rootGet("+", a: "1", b:"2").continueWithBlock {(task: AWSTask) -> AnyObject?
    in
       self.showResult(task)
       return nil
   }
   ```

   où `self.showResult(task)` de la fonction d'assistant imprime le résultat ou l'erreur vers la console ; par exemple :

   ```
   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)")
        }
    }
}
```

Dans une application de production, vous pouvez afficher le résultat ou l'erreur dans un champ de texte. Le résultat qui s'affiche est `1 + 2 = 3`.

3. Appelez la méthode `POST /` avec une charge utile pour exécuter `1-2` :

```
let body = SIMPLE_CALCInput()
body.a=1
body.b=2
body.op="-"
client.rootPost(body).continueWithBlock {(task: AWSTask) -> AnyObject? in
    self.showResult(task)
    return nil
}
```

Le résultat qui s'affiche est `1 - 2 = -1`.

4. Appelez la méthode `GET /{a}/{b}/{op}` pour exécuter `1/2` :

```
client.aBOpGet("1", b:"2", op:"div").continueWithBlock {(task: AWSTask) ->
 AnyObject? in
    self.showResult(task)
    return nil
}
```

Le résultat qui s'affiche est `1 div 2 = 0.5`. Ici, `div` est utilisé à la place de `/`, car la fonction Lambda simple du backend ne gère pas les variables de chemin d'URL codée.

Configuration d'une API REST à l'aide d'OpenAPI

Vous pouvez utiliser API Gateway pour importer une API REST à partir d'un fichier de définition externe dans API Gateway. Actuellement, API Gateway prend en charge les fichiers de définition OpenAPI v2.0 et OpenAPI v3.0 avec les exceptions répertoriées dans Remarques importantes concernant Amazon API Gateway pour les API REST. Vous pouvez mettre à jour une API en la remplaçant par une nouvelle définition, ou vous pouvez fusionner une définition avec une API existante. Vous précisez les options grâce à un paramètre de requête `mode` dans l'URL de demande.

Pour consulter un tutoriel sur l'utilisation de la fonction d'importation d'API à partir de la console API Gateway, consultez Tutoriel : Création d'une API REST par l'importation d'un exemple.

Rubriques

- Importation d'une API optimisée pour les périphériques dans API Gateway
- Importer une API régionale dans API Gateway
- Importation d'un fichier OpenAPI pour mettre à jour une définition d'API existante
- Définition de la propriété basePath OpenAPI
- AWS variables pour l'importation d'OpenAPI
- Erreurs et avertissements pendant l'importation
- Exportation d'une API REST à partir d'API Gateway

Importation d'une API optimisée pour les périphériques dans API Gateway

Vous pouvez importer un fichier de définition d'API OpenAPI pour créer une API optimisée pour les périphériques en spécifiant le type de point de terminaison `EDGE` sous la forme d'une entrée supplémentaire, en plus du fichier OpenAPI, dans l'opération d'importation. Vous pouvez le faire à l'aide de la console API Gateway ou d'un AWS SDK. AWS CLI

Pour consulter un tutoriel sur l'utilisation de la fonction d'importation d'API à partir de la console API Gateway, consultez Tutoriel : Création d'une API REST par l'importation d'un exemple.

Rubriques

- Importation d'une API optimisée pour les périphériques à l'aide de la console API Gateway
- Importez une API optimisée pour les périphériques à l'aide du AWS CLI

Importation d'une API optimisée pour les périphériques à l'aide de la console API Gateway

Pour importer une API optimisée pour les périphériques à l'aide de la console API Gateway, procédez comme suit :

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Sélectionnez Create API (Créer une API).
3. Sous API REST, choisissez Importer.
4. Copiez une définition OpenAPI d'API et collez-la dans l'éditeur de code ou choisissez Choisir un fichier pour charger un fichier OpenAPI à partir d'un lecteur local.
5. Pour Type de point de terminaison d'API, sélectionnez Optimisé pour la périphérie.
6. Choisissez Créer une API pour importer les définitions OpenAPI.

Importez une API optimisée pour les périphériques à l'aide du AWS CLI

Pour importer une API depuis un fichier de définition OpenAPI afin de créer une nouvelle API optimisée pour les périphériques à l'aide de AWS CLI, utilisez la `import-rest-api` commande suivante :

```
aws apigateway import-rest-api \
    --fail-on-warnings \
    --body 'file://path/to/API_OpenAPI_template.json'
```

ou en spécifiant explicitement le paramètre de chaîne de requête `endpointConfigurationTypes` à `EDGE` :

```
aws apigateway import-rest-api \
    --parameters endpointConfigurationTypes=EDGE \
    --fail-on-warnings \
    --body 'file://path/to/API_OpenAPI_template.json'
```

Importer une API régionale dans API Gateway

Lors de l'importation d'une API, vous pouvez choisir la configuration de points de terminaison régionaux pour l'API. Vous pouvez utiliser la console API Gateway AWS CLI, le ou un AWS SDK.

Lorsque vous exportez une API, sa configuration de point de terminaison ne figure pas dans les définitions d'API exportées.

Pour consulter un tutoriel sur l'utilisation de la fonction d'importation d'API à partir de la console API Gateway, consultez Tutoriel : Création d'une API REST par l'importation d'un exemple.

Rubriques

- Importation d'une API régionale à l'aide de la console API Gateway
- Importation d'une API régionale à l'aide de l' AWS CLI

Importation d'une API régionale à l'aide de la console API Gateway

Pour importer une API d'un point de terminaison régional à l'aide de la console API Gateway, procédez comme suit :

1. Connectez-vous à la console API Gateway à l'adresse : https://console.aws.amazon.com/apigateway.
2. Sélectionnez Create API (Créer une API).
3. Sous API REST, choisissez Importer.
4. Copiez une définition OpenAPI d'API et collez-la dans l'éditeur de code ou choisissez Choisir un fichier pour charger un fichier OpenAPI à partir d'un lecteur local.
5. Pour Type de point de terminaison d'API, sélectionnez Régional.
6. Choisissez Créer une API pour importer les définitions OpenAPI.

Importation d'une API régionale à l'aide de l' AWS CLI

Pour importer une API depuis un fichier de définition OpenAPI à l'aide de AWS CLI, utilisez la `import-rest-api` commande :

```
aws apigateway import-rest-api \
    --parameters endpointConfigurationTypes=REGIONAL \
    --fail-on-warnings \
    --body 'file://path/to/API_OpenAPI_template.json'
```

Importation d'un fichier OpenAPI pour mettre à jour une définition d'API existante

Vous pouvez importer des définitions d'API uniquement pour mettre à jour une API existante, sans modifier la configuration de son point de terminaison, ainsi que ses étapes et variables d'étapes ou ses références à des clés d'API.

L' import-to-update opération peut se produire selon deux modes : fusion ou remplacement.

Lorsqu'une API (A) est fusionnée avec une autre (B), l'API résultante conserve les définitions des API A et B, si les deux API ne partagent aucune définition conflictuelle. Lorsque des conflits surviennent, les définitions de méthode de l'API fusionnante (A) remplacent les définitions de méthode correspondantes de l'API fusionnée (B). Par exemple, supposons que B déclare les méthodes suivantes pour renvoyer les réponses 200 et 206 :

```
GET /a
POST /a
```

et A déclare la méthode suivante pour renvoyer les réponses 200 et 400 :

```
GET /a
```

Lorsque A est fusionnée avec B, l'API résultante génère les méthodes suivantes :

```
GET /a
```

qui retourne les réponses 200 et 400, et

```
POST /a
```

qui retourne les réponses 200 et 206.

La fusion d'une API s'avère utile lorsque vous avez décomposé vos définitions d'API externes en plusieurs parties plus petites et souhaitez seulement appliquer les modifications à partir d'une de ces parties à la fois. Cela peut être le cas, par exemple, si plusieurs équipes sont responsables de différentes parties d'une API et que leurs modifications deviennent disponibles à des rythmes différents. Dans ce mode, les éléments de l'API existante qui ne sont pas spécifiquement définis dans la définition importée seront conservés.

Lorsqu'une API (A) remplace une autre (B), l'API résultante prend les définitions de l'API remplaçante (A). Le remplacement d'une API est utile lorsqu'une définition d'API externe contient la définition

complète d'une API. Dans ce mode, les éléments d'une API existante qui ne sont pas spécifiquement définis dans la définition importée sont supprimés.

Pour fusionner une API, envoyez une requête PUT à `https://apigateway.<region>.amazonaws.com/restapis/<restapi_id>?mode=merge`. La valeur du paramètre de chemin `restapi_id` spécifie l'API avec laquelle la définition d'API fournie va être fusionnée.

L'extrait de code suivant présente un exemple de requête PUT pour fusionner une définition d'API OpenAPI dans JSON en tant que charge utile, avec l'API spécifiée déjà présente dans API Gateway.

```
PUT /restapis/<restapi_id>?mode=merge
Host:apigateway.<region>.amazonaws.com
Content-Type: application/json
Content-Length: ...

An OpenAPI API definition in JSON
```

L'opération de mise à jour par fusion prend deux définitions d'API complètes et les fusionne ensemble. Pour une petite modification incrémentielle, vous pouvez utiliser l'opération de mise à jour des ressources.

Pour remplacer une API, envoyez une requête PUT à `https://apigateway.<region>.amazonaws.com/restapis/<restapi_id>?mode=overwrite`. Le paramètre de chemin `restapi_id` spécifie l'API qui sera remplacée par les définitions d'API fournies.

L'extrait de code suivant présente un exemple de requête de remplacement avec la charge utile d'une définition OpenAPI au format JSON :

```
PUT /restapis/<restapi_id>?mode=overwrite
Host:apigateway.<region>.amazonaws.com
Content-Type: application/json
Content-Length: ...

An OpenAPI API definition in JSON
```

Lorsque le paramètre `mode` de la requête n'est pas spécifié, le mode de fusion est sélectionné par défaut.

> **Note**
>
> Les opérations PUT sont idempotentes, mais pas atomiques. Cela signifie que si une erreur système se produit au cours du traitement, l'état final de l'API peut être incorrect. Cependant, la réussite de la répétition de l'opération fera passer l'API au même état final que si la première opération avait réussi.

Définition de la propriété **`basePath`** OpenAPI

Dans OpenAPI 2.0, vous pouvez utiliser la propriété `basePath` pour fournir une ou plusieurs parties de chemin précédant chaque chemin défini dans la propriété `paths`. API Gateway peut exprimer le chemin d'une ressource de plusieurs façons. Par conséquent, la fonction d'importation d'API propose les options ci-après pour interpréter la propriété `basePath` au cours d'une importation : ignorer, préfixer et fractionner.

Dans OpenAPI 3.0, `basePath` n'est plus une propriété de niveau supérieur. Au lieu de cela, API Gateway utilise une variable serveur en tant que convention. La fonction d'importation d'une API fournit les mêmes options pour interpréter le chemin de base lors de l'importation. Le chemin de base est identifié comme suit :

- Si l'API ne contient pas de variables `basePath`, la fonction d'importation d'API vérifie la chaîne `server.url` pour voir si elle contient un chemin au-delà de `"/"`. Si c'est le cas, ce chemin est utilisé comme chemin de base.
- Si l'API contient une seule variable `basePath`, la fonction d'importation d'API l'utilise comme chemin de base, même si elle n'est pas référencée dans la chaîne `server.url`.
- Si l'API contient plusieurs variables `basePath`, la fonction d'importation d'API utilise uniquement la première comme chemin de base.

Ignorer

Si le fichier OpenAPI comporte une valeur `basePath` `/a/b/c` et si la propriété `paths` contient `/e` et `/f`, la demande POST ou PUT suivante :

```
POST /restapis?mode=import&basepath=ignore
```

```
PUT /restapis/api_id?basepath=ignore
```

génère les ressources suivantes dans l'API :

- `/`
- `/e`
- `/f`

Cela revient à traiter la propriété `basePath` comme si elle n'était pas présente, et toutes les ressources API déclarées sont traitées par rapport à l'hôte. Vous pouvez utiliser cette option, par exemple, lorsque vous avez un nom de domaine personnalisé avec un mappage d'API qui n'inclut pas de propriété Base Path et de valeur Stage faisant référence à votre étape de production.

> **Note**
>
> API Gateway crée automatiquement une ressource racine, même si elle n'est pas explicitement déclarée dans votre fichier de définition.

Lorsqu'elle n'est pas spécifiée, la propriété `basePath` prend la valeur `ignore` par défaut.

Prepend

Si le fichier OpenAPI comporte une valeur `basePath` `/a/b/c` et si la propriété `paths` contient `/e` et `/f`, la demande `POST` ou `PUT` suivante :

```
POST /restapis?mode=import&basepath=prepend
```

```
PUT /restapis/api_id?basepath=prepend
```

génère les ressources suivantes dans l'API :

- `/`
- `/a`
- `/a/b`
- `/a/b/c`

- `/a/b/c/e`
- `/a/b/c/f`

Cela revient à traiter la propriété `basePath` comme des ressources supplémentaires de spécification (sans méthode) et à ajouter celles-ci à l'ensemble de ressources déclarées. Vous pouvez utiliser cette option, par exemple, lorsque différentes équipes sont responsables des différentes parties d'une API et que la propriété `basePath` peut faire référence à l'emplacement du chemin de la partie d'API de chaque équipe.

> **Note**
>
> API Gateway crée automatiquement les ressources intermédiaires, même si elles ne sont pas explicitement déclarées dans votre définition.

Split

Si le fichier OpenAPI comporte une valeur `basePath` `/a/b/c` et si la propriété `paths` contient `/e` et `/f`, la demande `POST` ou `PUT` suivante :

```
POST /restapis?mode=import&basepath=split
```

```
PUT /restapis/api_id?basepath=split
```

génère les ressources suivantes dans l'API :

- `/`
- `/b`
- `/b/c`
- `/b/c/e`
- `/b/c/f`

Cela revient à traiter la partie supérieure du chemin, `/a`, comme le début du chemin de chaque ressource et à créer des ressources supplémentaires (sans méthode) au sein de l'API proprement dite. Vous pouvez utiliser cette option, par exemple, lorsque a est un nom d'étape que vous souhaitez exposer dans votre API.

AWS variables pour l'importation d'OpenAPI

Vous pouvez utiliser les AWS variables suivantes dans les définitions d'OpenAPI. API Gateway résout les variables lorsque l'API est importée. Pour spécifier une variable, utilisez `${variable-name}`.

AWS variables

Nom de variable	Description
`AWS::AccountId`	L'ID de AWS compte qui importe l'API, par exemple 123456789012.
`AWS::Partition`	AWS Partition dans laquelle l'API est importée. Pour les AWS régions standard, la partition est `aws`.
`AWS::Region`	La AWS région dans laquelle l'API est importée, par exemple,. `us-east-2`

AWS exemple de variables

L'exemple suivant utilise des AWS variables pour spécifier une AWS Lambda fonction pour une intégration.

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
```

Erreurs et avertissements pendant l'importation

Erreurs lors de l'importation

Lors de l'importation, des erreurs peuvent être générées pour les principaux problèmes, comme un document OpenAPI non valide. Les erreurs sont renvoyées en tant qu'exceptions (par exemple, `BadRequestException`) dans une réponse infructueuse. Lorsqu'une erreur se produit, la nouvelle définition d'API est ignorée et aucune modification n'est apportée à l'API existante.

Avertissements lors de l'importation

Lors de l'importation, des avertissements peuvent être générés pour les problèmes mineurs, tels que l'absence d'une référence de modèle. En cas d'avertissement, l'opération continue si l'expression de requête `failonwarnings=false` est ajoutée à l'URL de la demande. Sinon, les mises à jour sont annulées. Par défaut, la propriété `failonwarnings` a la valeur `false`. Dans ce cas, les avertissements sont renvoyés sous forme de champ dans la RestApiressource résultante. Sinon, les avertissements sont renvoyés sous la forme d'un message dans l'exception.

Exportation d'une API REST à partir d'API Gateway

Une fois que vous avez créé et configuré une API REST dans API Gateway à l'aide de la console API Gateway ou par tout autre moyen, vous pouvez l'exporter dans un fichier OpenAPI à l'aide de la fonction d'exportation d'API API Gateway, qui fait partie du service de contrôle Amazon API Gateway. Pour utiliser l'API d'exportation d'API Gateway, vous devez signer vos demandes d'API. Pour plus d'informations sur les demandes de signature, consultez la section Signing AWS API du Guide de l'utilisateur IAM. Des options vous permettent d'inclure les extensions d'intégration API Gateway, ainsi que les extensions Postman dans le fichier de définition OpenAPI exporté.

> **Note**
>
> Lorsque vous exportez l'API à l'aide de AWS CLI, veillez à inclure le paramètre extensions comme indiqué dans l'exemple suivant, afin de garantir que l'`x-amazon-apigateway-request-validator`extension est incluse :
>
> ```
> aws apigateway get-export --parameters extensions='apigateway' --rest-api-id
>  abcdefg123 --stage-name dev --export-type swagger latestswagger2.json
> ```

Vous ne pouvez pas exporter une API si ses charges utiles ne sont pas de type `application/json`. Si vous essayez, vous obtenez un message d'erreur indiquant que les modèles de corps JSON sont introuvables.

Demande d'exportation d'une API REST

Avec l'API d'exportation, vous exportez une API REST existante en soumettant une requête GET, en spécifiant l' to-be-exported API dans le cadre des chemins d'URL. L'URL de la requête est au format suivant :

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
```

Vous pouvez ajouter la chaîne de requête `extensions` pour spécifier si vous souhaitez inclure les extensions API Gateway (avec la valeur `integration`) ou les extensions Postman (avec la valeur `postman`).

En outre, vous pouvez affecter à l'en-tête `Accept` la valeur `application/json` ou `application/yaml` pour recevoir la définition d'API au format JSON ou YAML, respectivement.

Pour plus d'informations sur la soumission de requêtes GET à l'aide de l'API API Gateway Export, consultez GetExport.

> **Note**
>
> Si vous définissez des modèles dans votre API, ils doivent être définis pour le type de contenu « application/json » pour qu'API Gateway les exporte. Sinon, API Gateway lance une exception avec le message d'erreur « Only found non-JSON body models for... » (Seuls des modèles de corps non-JSON ont été détectés pour...).
> Les modèles doivent contenir des propriétés ou être définis comme un type JSONSchema particulier.

Téléchargement d'une définition OpenAPI d'API REST au format JSON

Pour exporter et télécharger une API REST dans des définitions OpenAPI au format JSON :

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
```

Ici, *<region>* pourrait être `us-east-1`, par exemple. Pour toutes les régions où API Gateway est disponible, consultez Régions et points de terminaison.

Téléchargement d'une définition OpenAPI d'API REST au format YAML

Pour exporter et télécharger une API REST dans des définitions OpenAPI au format YAML :

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
```

Téléchargement d'une définition OpenAPI d'API REST avec les extensions Postman au format JSON

Pour exporter et télécharger une API REST dans des définitions OpenAPI avec Postman au format JSON :

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
```

Téléchargement d'une définition OpenAPI d'API REST avec intégration API Gateway au format YAML

Pour exporter et télécharger une API REST dans des définitions OpenAPI avec intégration API Gateway au format YAML :

OpenAPI 3.0

```
GET /restapis/<restapi_id>/stages/<stage_name>/exports/oas30?extensions=integrations
Host: apigateway.<region>.amazonaws.com
Accept: application/yaml
```

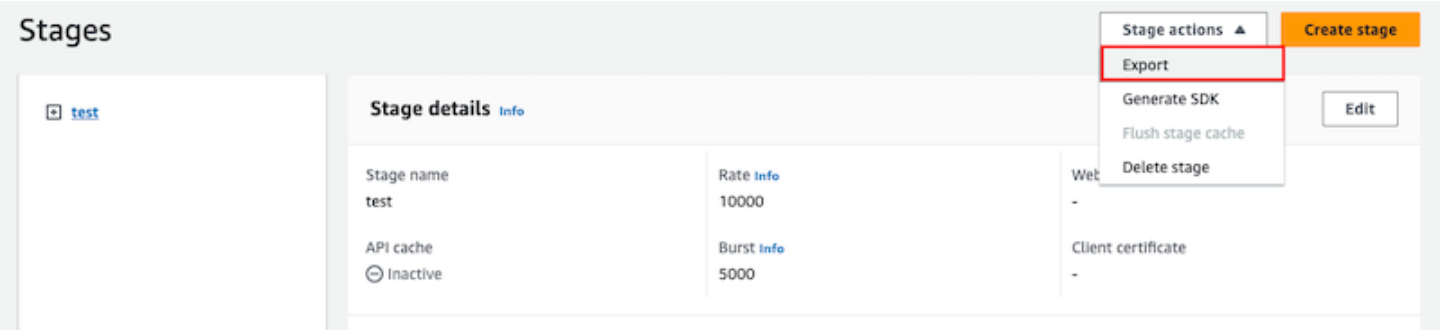
OpenAPI 2.0

```
GET /restapis/<restapi_id>/stages/<stage_name>/exports/swagger?
extensions=integrations
Host: apigateway.<region>.amazonaws.com
Accept: application/yaml
```

Exportation d'une API REST à l'aide de la console API Gateway

Après avoir déployé votre API REST jusqu'à un certain stade, vous pouvez passer à l'exportation de cette API à l'étape dans un fichier OpenAPI à l'aide de la console API Gateway.

Dans le volet Étapes de la console API Gateway, choisissez Actions d'étape, Exporter.



Spécifiez un type de spécification d'API, un format et des extensions pour télécharger la définition OpenAPI de votre API.

Publication d'API REST appelables par les clients

Le simple fait de créer et le développer une API API Gateway ne la rend pas automatiquement appelable par vos utilisateurs. Pour la rendre appelable, vous devez déployer votre API jusqu'à une étape. En outre, vous pouvez personnaliser l'URL que vos utilisateurs utiliseront pour accéder à votre API. Vous pouvez lui attribuer un domaine cohérent avec votre marque ou plus facilement mémorisable que l'URL par défaut de votre API.

Dans cette section, vous pouvez apprendre à déployer votre API et à personnaliser l'URL que vous fournissez aux utilisateurs pour y accéder.

> **Note**
>
> Pour renforcer la sécurité de vos API API Gateway, le domaine `execute-api.{region}.amazonaws.com` est enregistré dans la liste des suffixes publics (PSL). Pour

plus de sécurité, nous vous recommandons d'utiliser des cookies avec un préfixe `__Host-` si vous devez définir des cookies sensibles dans le nom de domaine par défaut de vos API API Gateway. Cette pratique vous aidera à protéger votre domaine contre les tentatives de falsification de requêtes intersites (CSRF). Pour plus d'informations, consultez la page Set-Cookie du Mozilla Developer Network.

Rubriques

- Déploiement d'une API REST dans Amazon API Gateway
- Configuration des noms de domaine personnalisés pour les API REST

Déploiement d'une API REST dans Amazon API Gateway

Après avoir créé votre API, vous devez la déployer pour que vos utilisateurs puissent l'appeler.

Pour déployer une API, vous devez créer un déploiement d'API et l'associer à une étape. Une étape est une référence logique à un état du cycle de vie de votre API (par exemple, `dev`, `prod`, `beta`, `v2`). Les étapes d'API sont identifiées par l'ID de l'API et un nom d'étape. Elles sont incluses dans l'URL que vous utilisez pour appeler l'API. Chaque étape est une référence nommée à un déploiement de l'API et elle est mise à la disposition des applications clientes à appeler.

> **Important**
>
> Chaque fois que vous mettez à jour une API, vous devez redéployer l'API vers une étape existante ou vers une nouvelle étape. La mise à jour d'une API inclut la modification des routes, des méthodes, des intégrations, des autorisateurs, des politiques de ressources et de tout autre élément autre que les paramètres d'étape.

À mesure que votre API évolue, vous pouvez continuer à la déployer dans différentes étapes sous forme de versions distinctes de l'API. Vous pouvez également déployer vos mises à jour d'API en tant que déploiement de version Canary. Cela permet à vos clients API d'accéder, au cours de la même étape, à la version de production via la version de production, et à la version mise à jour via la version Canary.

Pour appeler une API déployée, le client soumet une demande par rapport à l'URL d'une API. L'URL est déterminée par le protocole d'une API (HTTP(S) ou (WSS)), le nom d'hôte, le nom de l'étape et

(pour l'API REST) le chemin d'accès à la ressource. Le nom d'hôte et le nom de l'étape déterminent l'URL de base de l'API.

Avec le nom de domaine par défaut de l'API, l'URL de base (par exemple) d'une API REST à une étape donnée (*{stageName}*) a le format suivant :

```
https://{restapi-id}.execute-api.{region}.amazonaws.com/{stageName}
```

Pour rendre une URL de base par défaut de l'API plus conviviale, vous pouvez créer un nom de domaine personnalisé (par exemple, `api.example.com`) pour remplacer le nom d'hôte par défaut de l'API. Pour prendre en charge plusieurs API sous le nom de domaine personnalisé, vous devez mapper une étape d'API à un chemin de base.

Avec un nom de domaine personnalisé *{api.example.com}* et l'étape d'API mappée à un chemin de base (*{basePath}*) sous le nom de domaine personnalisé, l'URL de base d'une API REST est remplacée par :

```
https://{api.example.com}/{basePath}
```

Pour chaque étape, vous pouvez optimiser les performances de l'API en ajustant les limitations de demande au niveau du compte par défaut et en activant la mise en cache des API. Vous pouvez également activer la journalisation des appels d'API vers CloudTrail ou CloudWatch, et sélectionner un certificat client pour le backend afin d'authentifier les demandes d'API. En outre, vous pouvez remplacer des paramètres au niveau d'une étape pour des méthodes individuelles et définir des variables d'étape pour transmettre à l'intégration d'API des contextes d'environnement spécifiques à l'étape au moment de l'exécution.

Les étapes permettent un contrôle de version solide de votre API. Par exemple, vous pouvez déployer une API dans une étape `test` et une étape `prod`, puis utiliser l'étape `test` comme version de test et l'étape `prod` comme version stable. Une fois que les mises à jour passent le test, vous pouvez migrer l'étape `test` vers l'étape `prod`. Pour ce faire, redéployez l'API dans l'étape `prod` ou mettez à jour la valeur d'une variable d'étape en remplaçant le nom d'étape `test` par `prod`.

Dans cette section, nous expliquons comment déployer une API en utilisant la console API Gateway ou en appelant l' API REST API Gateway. Pour utiliser d'autres outils, reportez-vous à la documentation de la CLI AWS ou d'un kit SDK AWS.

Rubriques

- Déploiement d'une API REST dans API Gateway

- Configuration d'une étape pour une API REST
- Configuration d'un déploiement de la version canary API Gateway
- Mises à jour d'une API REST nécessitant un redéploiement

Déploiement d'une API REST dans API Gateway

Dans API Gateway, le déploiement d'une API REST est représenté par une ressource Deployment. Il est similaire à un exécutable d'une API qui est représentée par une ressource RestApi.

Pour que le client appelle l'API, vous devez créer un déploiement et y associer une étape. Une étape est représentée par une ressource Stage. Elle représente un instantané de l'API, y compris les méthodes, les intégrations, les modèles, les modèles de mappage et les mécanismes d'autorisation Lambda (anciennement appelés mécanismes d'autorisation personnalisés). Lorsque vous mettez à jour l'API, vous pouvez redéployer l'API en associant une nouvelle étape au déploiement existant. Nous aborderons la création d'une étape sous the section called “Configuration d'une étape”.

Rubriques

- Création d'un déploiement avec l AWS CLI
- Déploiement d'une API REST à partir de la console API Gateway

Création d'un déploiement avec l AWS CLI

Lorsque vous créez un déploiement, vous instanciez la ressource Deployment. Vous pouvez utiliser la console API Gateway, la AWS CLI, un kit SDK AWS ou l'API REST API Gateway pour créer un déploiement.

Pour utiliser la CLI afin de créer un déploiement, utilisez la commande create-deployment :

```
aws apigateway create-deployment --rest-api-id <rest-api-id> --region <region>
```

L'API ne peut pas être appelée tant que vous n'associez pas ce déploiement à une étape. Si l'étape existe déjà, remplacez la propriété `deploymentId` correspondante par l'ID du déploiement que vous venez de créer (`<deployment-id>`).

```
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>'
```

Lorsque vous déployez une API pour la première fois, vous pouvez combiner l'étape de création et la création du déploiement en même temps :

```
aws apigateway create-deployment --region <region> \
   --rest-api-id <rest-api-id> \
   --stage-name <stage-name>
```

C'est ce qui se produit en arrière-plan dans la console API Gateway, lorsque vous déployez une API pour la première fois ou lorsque vous redéployez l'API en l'intégrant à une nouvelle étape.

Déploiement d'une API REST à partir de la console API Gateway

Vous devez avoir créé une API REST avant de la déployer pour la première fois. Pour plus d'informations, consultez Développement d'une API REST dans API Gateway.

Rubriques

- Déploiement d'une API REST jusqu'à une étape
- Redéploiement d'une API REST jusqu'à une étape
- Mise à jour de la configuration d'étape d'un déploiement d'API REST
- Définition de variables d'étape pour le déploiement d'API REST
- Association d'une étape à un autre déploiement d'API REST

Déploiement d'une API REST jusqu'à une étape

La console API Gateway vous permet de déployer une API en créant un déploiement et en l'associant à une étape nouvelle ou existante.

> **Note**
>
> Pour associer une étape dans API Gateway à un autre déploiement, veuillez consulter plutôt Association d'une étape à un autre déploiement d'API REST.

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Dans le volet de navigation APIs, sélectionnez l'API que vous souhaitez déployer.

3. Dans le volet Resources, sélectionnez Deploy API.
4. Pour Étape, sélectionnez l'une des options suivantes :

   a. Pour créer une nouvelle étape, sélectionnez Nouvelle étape, puis entrez un nom dans Nom de l'étape. Vous pouvez éventuellement fournir une description du déploiement dans Description du déploiement.

   b. Pour choisir une étape existante, sélectionnez le nom de l'étape dans le menu déroulant. Vous pouvez également fournir une description du nouveau déploiement dans Description du déploiement.

   c. Pour créer un déploiement qui n'est pas associé à une étape, sélectionnez Aucune étape. Plus tard, vous pourrez associer ce déploiement à une étape.
5. Choisissez Deploy (Déployer).

Redéploiement d'une API REST jusqu'à une étape

Pour redéployer une API, exécutez les mêmes étapes que celles décrites dans the section called “Déploiement d'une API REST jusqu'à une étape”. Vous pouvez réutiliser la même étape autant de fois que vous le souhaitez.

Mise à jour de la configuration d'étape d'un déploiement d'API REST

Une fois qu'une API est déployée, vous pouvez modifier les paramètres d'étape pour activer ou désactiver le cache API, la journalisation ou les limitations de demande. Vous pouvez également choisir un certificat client pour que le backend authentifie API Gateway et définisse des variables d'étape afin de transmettre le contexte de déploiement à l'intégration d'API au moment de l'exécution. Pour de plus amples informations, veuillez consulter Mise à jour des paramètres d'étape.

> **Important**
>
> Après avoir modifié les paramètres d'étape, vous devez redéployer l'API pour que les modifications prennent effet.

> **Note**
>
> Si les paramètres mis à jour, par exemple, l'activation de la journalisation, nécessitent un nouveau rôle IAM, vous pouvez ajouter le rôle IAM requis sans redéployer l'API. Cependant,

cela peut prendre quelques minutes avant que le nouveau rôle IAM prenne effet. Avant cela, les traces de vos appels d'API ne sont pas consignées, même si vous avez activé l'option de journalisation.

Définition de variables d'étape pour le déploiement d'API REST

Pour un déploiement, vous pouvez définir ou modifier des variables d'étape pour transmettre des données spécifiques au déploiement à l'intégration d'API au moment de l'exécution. Vous pouvez effectuer cette opération dans l'onglet Stage Variables dans l'éditeur Stage Editor. Pour plus d'informations, consultez les instructions dans Configuration de variables d'étape pour le déploiement d'API REST.

Association d'une étape à un autre déploiement d'API REST

Etant donné qu'un déploiement représente un instantané d'API et qu'une étape définit un chemin vers un instantané, vous pouvez choisir différentes combinaisons d'étapes de déploiement pour contrôler la façon dont les utilisateurs appellent différentes versions de l'API. Cela s'avère utile, par exemple, lorsque vous voulez restaurer l'état de l'API à un déploiement antérieur ou fusionner une branche « privée » de l'API dans une branche publique.

La procédure suivante montre comment procéder à l'aide de l'éditeur Stage Editor (Éditeur d'étape) dans la console API Gateway. Nous supposons que vous devez avoir déployé une API plusieurs fois.

1. Si vous n'êtes pas déjà dans le volet Étapes, dans le panneau de navigation principal, choisissez Étapes.
2. Sélectionnez l'étape que vous souhaitez mettre à jour.
3. Sous l'onglet Historique de déploiement, sélectionnez le déploiement que vous voulez que l'étape utilise.
4. Choisissez Modifier le déploiement actif.
5. Confirmez que vous souhaitez changer le déploiement actif et choisissez Modifier le déploiement actif dans la boîte de dialogue Rendre le déploiement actif.

Configuration d'une étape pour une API REST

Une étape est une référence nommée à un déploiement qui est un instantané de l'API. Vous utilisez une Stage (Étape) pour gérer et optimiser un déploiement spécifique. Par exemple, vous pouvez configurer les paramètres d'une étape pour activer la mise en cache, personnaliser la limitation des

requêtes, configurer la journalisation, définir des variables d'étapes ou attacher une version Canary à des fins de test.

Rubriques

Configuration d'une étape à l'aide de la console API Gateway

Rubriques

Création d'une étape

Après le déploiement initial, vous pouvez ajouter d'autres étapes et les associer à des déploiements existants. Vous pouvez utiliser la console API Gateway pour créer une étape ou vous pouvez choisir une étape existante pendant le déploiement d'une API. En règle générale, vous pouvez ajouter une nouvelle étape à un déploiement d'API avant de redéployer l'API. Pour créer une étape à l'aide de la console API Gateway, procédez comme suit :

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Choisissez une API REST.
3. Dans le volet de navigation principal, choisissez Étapes sous une API.
4. Dans le volet de navigation Étapes, choisissez Créer une étape.
5. Pour Nom de l'étape, entrez un nom, par exemple **`prod`**.

> **Note**
>
> Les noms d'étape peuvent contenir uniquement des caractères alphanumériques, des tirets et des traits de soulignement. La longueur maximale est de 128 caractères.

6. (Facultatif) Pour Description, entrez une brève description.
7. Pour Déploiement, sélectionnez la date et l'heure du déploiement d'API existant que vous voulez associer à cette étape.
8. Sous Paramètres supplémentaires, vous pouvez définir des paramètres supplémentaires pour votre étape.
9. Choisissez Créer une étape.

Mise à jour des paramètres d'étape

Après un déploiement réussi d'une API, l'étape est remplie avec les paramètres par défaut. Vous pouvez utiliser la console ou l'API REST API Gateway pour modifier les paramètres de l'étape, y compris la mise en cache des API et la journalisation. Les étapes suivantes vous montrent comment effectuer cette opération dans l'Éditeur d'étape de la console API Gateway.

Mise à jour des paramètres de l'étape à l'aide de la console API Gateway

Cette procédure suppose que vous ayez déjà déployé l'API à une étape.

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Choisissez une API REST.
3. Dans le volet de navigation principal, choisissez Étapes sous une API.
4. Dans le volet Stages, choisissez le nom de l'étape.
5. Dans la section Détails de l'étape, choisissez Modifier.
6. (Facultatif) Pour Description de l'étape, modifiez la description.
7. Pour Paramètres supplémentaires, vous modifiez les paramètres suivants :

   Paramètres du cache

   Pour activer la mise en cache des API pour la phase, activez le cache de l'API Provisionner. Configurez ensuite la mise en cache au niveau de la méthode par défaut, la capacité du cache, le chiffrement des données du cache, le cache time-to-live (TTL), ainsi que les exigences relatives à l'invalidation du cache par clé.

La mise en cache n'est active que lorsque vous activez la mise en cache par défaut au niveau de la méthode ou que vous activez le cache au niveau de la méthode pour une méthode spécifique.

Pour plus d'informations sur les paramètres de cache, consultez Activation de la mise en cache des API pour améliorer la réactivité.

> **Note**
>
> Si vous activez la mise en cache d'API pour une étape d'API, la mise en cache d'API peut être facturée sur votre AWS compte. La mise en cache n'est pas éligible au niveau AWS gratuit.

Paramètres de limitation

Pour définir les cibles de limitation au niveau de l'étape pour toutes les méthodes associées à cette API, activez Limitation.

Pour Rate (Taux), entrez un taux cible. Il s'agit du taux, en requêtes par seconde, auquel les jetons sont ajoutés au compartiment de jetons. Ce taux au niveau de l'étape ne doit pas être supérieur au taux au niveau du compte, comme indiqué dans Quotas API Gateway pour la configuration et l'exécution d'une API REST.

Pour Rafale, entrez un taux de rafale cible. Le taux de rafale correspond à la capacité du compartiment de jetons. Cela permet de passer plus de demandes pendant une période de temps que le taux cible. Ce taux en rafale au niveau de l'étape doit pas être supérieur au taux en rafale au niveau du compte, comme indiqué dans Quotas API Gateway pour la configuration et l'exécution d'une API REST.

> **Note**
>
> Les limites des taux de limitation ne sont pas strictes et sont appliquées au mieux. Dans certains cas, les clients peuvent dépasser les quotas que vous avez définis. Ne comptez pas sur la limitation pour contrôler les coûts ou bloquer l'accès à une API. Pensez à utiliser AWS Budgets pour contrôler les coûts et AWS WAF pour gérer les demandes d'API.

Paramètres du pare-feu et des certificats

Pour associer une ACL AWS WAF Web à la scène, sélectionnez une ACL Web dans la liste déroulante Web ACL. Si vous le souhaitez, choisissez Block API Request if WebACL cannot be evaluated (Fail- Close) (Bloquer la demande d'API si la liste ACL web ne peut pas être évaluée (Échec - Fermeture)).

Pour sélectionner un certificat client pour votre étape, sélectionnez-le dans le menu déroulant Certificat de client.

8. Choisissez Enregistrer.
9. Pour activer Amazon CloudWatch Logs pour toutes les méthodes associées à cette étape de cette API API Gateway, dans la section Journaux et suivi, choisissez Modifier.

> **Note**
>
> Pour activer CloudWatch les journaux, vous devez également spécifier l'ARN d'un rôle IAM qui permet à API Gateway d'écrire des informations dans CloudWatch les journaux au nom de votre utilisateur. Pour ce faire, choisissez Settings dans le volet de navigation principal APIs. Ensuite, pour le rôle CloudWatch log, entrez l'ARN d'un rôle IAM.
> Pour les scénarios d'application courants, le rôle IAM pourrait attacher la stratégie gérée `AmazonAPIGatewayPushToCloudWatchLogs`, qui contient la déclaration de stratégie d'accès suivante :

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:DescribeLogGroups",
                "logs:DescribeLogStreams",
                "logs:PutLogEvents",
                "logs:GetLogEvents",
                "logs:FilterLogEvents"
            ],
            "Resource": "*"
```

```
        }
    ]
}
```

Le rôle IAM doit également contenir la déclaration de relation d'approbation suivante :

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

Pour plus d'informations CloudWatch, consultez le guide de CloudWatch l'utilisateur Amazon.

10. Sélectionnez un niveau de journalisation dans le menu déroulant CloudWatch Logs. Les niveaux de journalisation sont les suivants :

    - Désactivé — La journalisation n'est pas activée pour cette étape.
    - Erreurs uniquement : la journalisation n'est activée que pour les erreurs.
    - Journaux d'erreurs et d'informations : la journalisation est activée pour tous les événements.
    - Journaux complets des demandes et des réponses : la journalisation détaillée est activée pour tous les événements. Cela peut être utile pour dépanner les API, mais peut entraîner la consignation de données sensibles.

    > **Note**
    >
    > Nous vous recommandons de ne pas utiliser les Journaux complets des requêtes et des réponses pour les API de production.

11. Sélectionnez les métriques détaillées pour qu'API Gateway produise un rapport CloudWatch sur les métriques d'API de `API callsLatency`,`Integration latency`,`400 errors`, et`500 errors`. Pour plus d'informations CloudWatch, consultez les sections Surveillance de base et surveillance détaillée dans le guide de CloudWatch l'utilisateur Amazon.

    > **Important**
    >
    > Votre compte est débité pour accéder aux métriques au niveau de la méthode, mais pas aux CloudWatch métriques au niveau de l'API ou au niveau de l'étape.

12. Pour activer la journalisation des accès à une destination, activez Journalisation des accès personnalisée.
13. Pour l'ARN de destination du journal Access, entrez l'ARN d'un groupe de journaux ou d'un flux Firehose.

    Le format ARN pour Firehose est. `arn:aws:firehose:{region}:{account-id}:deliverystream/amazon-apigateway-{your-stream-name}` Le nom de votre stream Firehose doit être. `amazon-apigateway-{your-stream-name}`
14. Dans Format de journal, entrez un format de journal. Pour en savoir plus sur les exemples de formats de journal, consultez the section called “CloudWatch formats de journal pour API Gateway”.
15. Pour activer le suivi AWS X-Ray pour l'étape d'API, sélectionnez Suivi X-Ray. Pour plus d’informations, consultez Suivi des demandes utilisateur vers les API REST à l'aide de X-Ray.
16. Choisissez Enregistrer les modifications. Redéployez votre API pour que les nouveaux paramètres prennent effet.

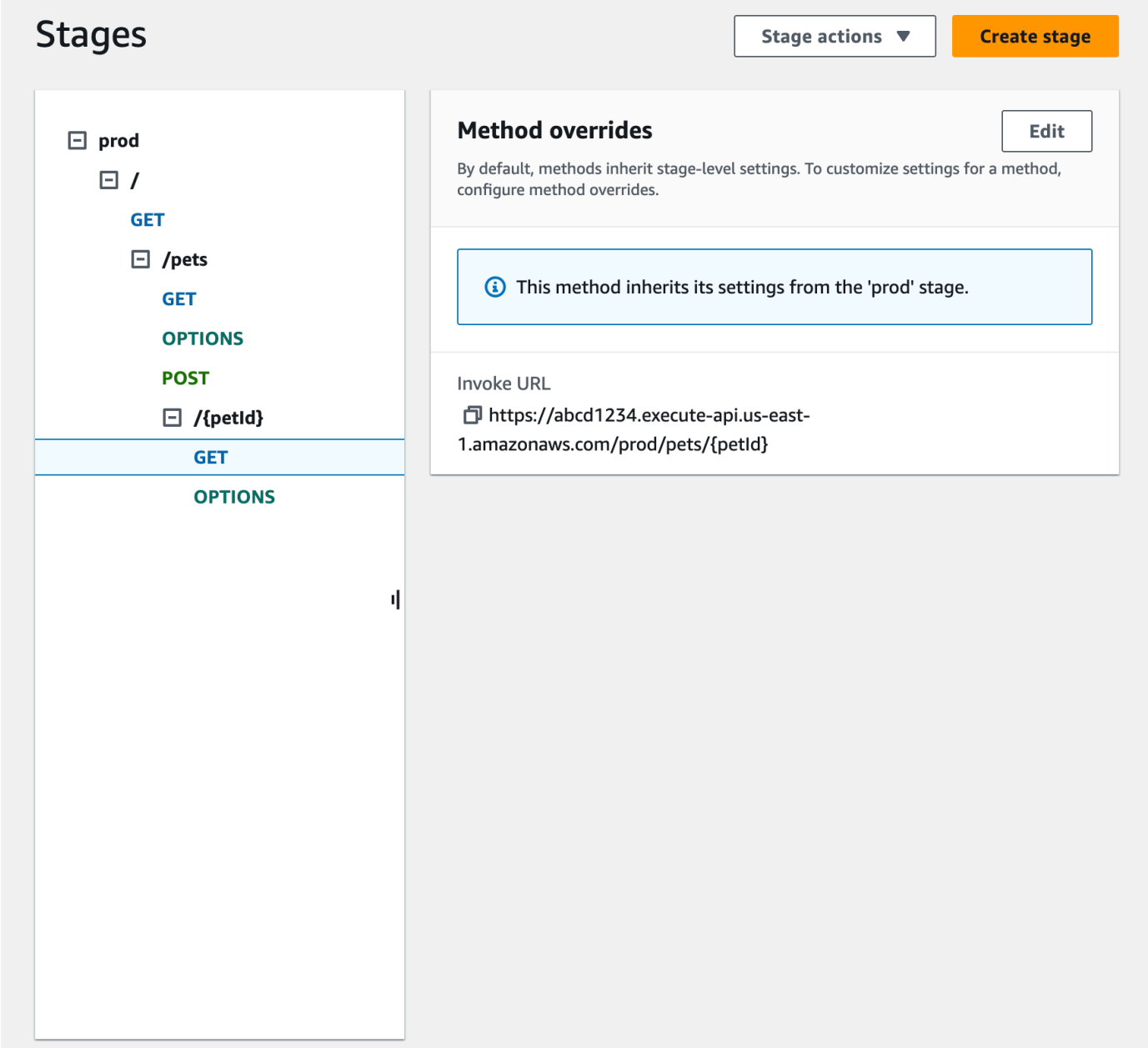
Remplacement des paramètres au niveau de l’étape

Vous pouvez remplacer les paramètres activés au niveau de l’étape. Certaines de ces options peuvent entraîner des frais supplémentaires pour votre Compte AWS.

Mise à jour des paramètres au niveau de l’étape à l’aide de la console API Gateway

Pour mettre à jour des paramètres au niveau de l’étape à l’aide de la console API Gateway

1. Pour configurer les remplacements de méthode, développez l’étape sous le volet de navigation secondaire, puis choisissez une méthode.



2. Ensuite, pour Remplacements de méthode, choisissez Modifier.
3. Pour activer les CloudWatch paramètres au niveau de la méthode, pour CloudWatch Logs, sélectionnez un niveau de journalisation.
4. Pour activer les métriques détaillées au niveau de la méthode, sélectionnez Métriques détaillées. Votre compte est débité pour accéder aux métriques au niveau de la méthode, mais pas aux CloudWatch métriques au niveau de l'API ou au niveau de l'étape.
5. Pour activer la limitation au niveau de la méthode, sélectionnez Limitation. Saisissez les options appropriées au niveau de la méthode. Pour en savoir plus sur la limitation, consultez the section called “Limitation”.

6. Pour configurer le cache au niveau de la méthode, sélectionnez Activer le cache de méthode. Si vous modifiez le paramètre de mise en cache par défaut au niveau de la méthode dans les détails de l'étape, cela n'affecte pas ce paramètre.
7. Choisissez Enregistrer.

Suppression d'une étape

Lorsque vous n'avez plus besoin d'une étape, vous pouvez le supprimer pour éviter de payer pour des ressources inutilisées. Les étapes suivantes vous montrent comment utiliser la console API Gateway pour supprimer une étape.

> ⚠ Warning
>
> Suite à la suppression d'une étape, il est possible que tout ou partie de l'API correspondante soit inutilisable par les appelants de l'API. La suppression d'une étape ne peut pas être annulée, mais vous pouvez recréer l'étape et l'associer au même déploiement.

Suppression d'une étape à l'aide de la console API Gateway

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Choisissez une API REST.
3. Dans le volet de navigation principal, choisissez Étapes.
4. Dans le volet Étapes, sélectionnez l'étape à supprimer, puis choisissez Actions d'étape et Supprimer une étape.
5. Lorsque vous y êtes invité, entrez **`confirm`**, puis choisissez Supprimer.

Configuration des balises pour une étape d'API dans API Gateway

Dans API Gateway, vous pouvez ajouter une balise à une étape d'API, supprimer la balise de l'étape ou afficher la balise. Pour ce faire, vous pouvez utiliser la console API Gateway, le AWS CLI/SDK ou l'API REST API Gateway.

Une étape peut également hériter des balises de son API REST parente. Pour de plus amples informations, veuillez consulter the section called “Héritage de balises dans l'API Amazon API Gateway V1”.

Pour de plus amples informations sur le balisage des ressources API Gateway, veuillez consulter Balisage.

Rubriques

- Configuration des balises pour une étape d'API à l'aide de la console API Gateway
- Configurez des balises pour une étape d'API à l'aide du AWS CLI
- Configuration des balises pour une étape d'API à l'aide de l'API REST API Gateway

Configuration des balises pour une étape d'API à l'aide de la console API Gateway

La procédure suivante explique comment configurer des balises pour une étape d'API.

Pour configurer les balises pour une étape d'API à l'aide de la console API Gateway

1. Connectez-vous à la console API Gateway.
2. Choisissez une API existante ou créez-en une qui comporte des ressources, des méthodes et les intégrations correspondantes.
3. Choisissez une étape ou déployez l'API dans une nouvelle étape.
4. Dans le volet de navigation principal, choisissez Étapes.
5. Sélectionnez l’onglet Tags (Identifications). Vous devrez peut-être choisir la flèche droite pour afficher l'onglet.
6. Choisissez Gérer les balises.
7. Dans Éditeur de balise, choisissez Ajouter une balise. Entrez une clé de balise (par exemple, `Department`) dans le champ Key (Clé), puis entrez une valeur de balise (par exemple, `Sales`) dans le champ colonne Value (Valeur). Choisissez Enregistrer pour enregistrer la balise.
8. Si nécessaire, répétez l'étape 5 pour ajouter des balises supplémentaires à l'étape d'API. Le nombre maximal de balises par étape est de 50.
9. Pour supprimer une balise existante de l'étape, choisissez Supprimer.
10. Si l'API a déjà été déployée dans la console API Gateway, vous devez la redéployer pour que les changements prennent effet.

Configurez des balises pour une étape d'API à l'aide du AWS CLI

Vous pouvez configurer des balises pour une étape d'API à l'aide de la AWS CLI commande create-stage ou de la commande tag-resource. Vous pouvez supprimer une ou plusieurs balises d'une étape d'API à l'aide de la commande untag-resource.

L'exemple suivant ajoute une balise lors de la création d'une `test` étape :

```
aws apigateway create-stage --rest-api-id abc1234 --stage-name test --description
 'Testing stage' --deployment-id efg456 --tag Department=Sales
```

L'exemple suivant ajoute une balise à une `prod` étape :

```
aws apigateway tag-resource --resource-arn arn:aws:apigateway:us-east-2::/
restapis/abc123/stages/prod --tags Department=Sales
```

L'exemple suivant supprime la `Department=Sales` balise de la `test` scène :

```
aws apigateway untag-resource --resource-arn arn:aws:apigateway:us-east-2::/
restapis/abc123/stages/test --tag-keys Department
```

Configuration des balises pour une étape d'API à l'aide de l'API REST API Gateway

Vous pouvez configurer des balises pour une étape d'API à l'aide de l'API REST API Gateway en effectuant l'une des actions suivantes :

- Appelez `tags:tag` pour baliser une étape d'API.
- Appelez `tags:untag` pour supprimer une ou plusieurs balises d'une étape d'API.
- Appelez `stage:create` pour ajouter une ou plusieurs balises à une étape d'API que vous êtes en train de créer.

Vous pouvez également appeler `tags:get` pour décrire les balises d'une étape d'API.

Balisage d'une étape d'API

Une fois que vous avez déployé une API (`m5zr3vnks7`) sur une étape (`test`), balisez cette dernière en appelant `tags:tag`. L'ARN (Amazon Resource Name) obligatoire de l'étape (`arn:aws:apigateway:us-east-1::/restapis/m5zr3vnks7/stages/test`) doit être encodé en URL (`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"
  }
}
```

Vous pouvez également utiliser la demande précédente pour mettre à jour une balise existante avec une nouvelle valeur.

Vous pouvez ajouter des balises à une étape lorsque vous appelez `stage:create` pour créer l'étape :

```
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"
  }
}
```

Suppression du balisage d'une étape d'API

Pour supprimer la balise `Department` de l'étape, appelez `tags:untag` :

```
DELETE /tags/arn%3Aaws%3Aapigateway%3Aus-east-1%3A%3A%2Frestapis%2Fm5zr3vnks7%2Fstages
%2Ftest?tagKeys=Department
Host: apigateway.us-east-1.amazonaws.com
Authorization: ...
```

Pour supprimer plusieurs balises, utilisez une liste de clés de balise séparées par des virgules dans l'expression de requête : par exemple, `?tagKeys=Department,Division,...`.

Description des balises d'une étape d'API

Pour décrire les balises existantes au niveau d'une étape donnée, appelez `tags:get`:

```
GET /tags/arn%3Aaws%3Aapigateway%3Aus-east-1%3A%3A%2Frestapis%2Fm5zr3vnks7%2Fstages
%2Ftags
Host: apigateway.us-east-1.amazonaws.com
Authorization: ...
```

La réponse positive est semblable à ce qui suit :

```
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"
    }
}
```

Configuration de variables d'étape pour le déploiement d'API REST

Les variables d'étape sont des paires nom-valeur que vous pouvez définir en tant qu'attributs de configuration associés à une étape de déploiement d'une API REST. Elles se comportent comme

les variables d'environnement et peuvent être utilisées dans vos modèles de mappage et de configuration d'API.

Par exemple, vous pouvez définir une variable d'étape dans une configuration d'étape, puis définir sa valeur en tant que chaîne d'URL d'une intégration HTTP pour une méthode de votre API REST. Par la suite, vous pouvez référencer la chaîne d'URL en utilisant le nom de variable d'étape associé à partir de la configuration de l'API. De cette façon, vous pouvez utiliser la même configuration d'API avec un point de terminaison différent à chaque étape en réinitialisant la variable d'étape pour lui affecter la valeur de l'URL correspondante.

Vous pouvez également accéder aux variables d'étape dans les modèles de mappage, ou transmettre les paramètres de configuration à votre backend HTTP ou AWS Lambda .

Pour plus d'informations sur les modèles de mappage, consultez Modèle de mappage API Gateway et référence à la variable de journalisation des accès.

> **Note**
>
> Les variables d'étape ne sont pas destinées à être utilisées pour des données sensibles, telles que les informations d'identification. Pour transmettre des données sensibles aux intégrations, utilisez un AWS Lambda autorisateur. Vous pouvez transmettre des données sensibles aux intégrations dans la sortie du mécanisme d'autorisation Lambda. Pour en savoir plus, consultez la section the section called “Résultat d'un autorisateur Lambda API Gateway”.

Cas d'utilisation

Grâce aux étapes de déploiement dans API Gateway, vous pouvez gérer plusieurs étapes de version pour chaque API, par exemple alpha, bêta et production. A l'aide des variables d'étape, vous pouvez configurer une étape de déploiement d'API pour interagir avec différents points de terminaison du serveur principal.

Par exemple, votre API peut transmettre une demande GET en tant que proxy HTTP à l'hôte Web backend (par exemple, `http://example.com`). Dans ce cas, l'hôte web backend est configuré dans une variable d'étape afin que lorsque les développeurs appellent votre point de terminaison de production, API Gateway appelle example.com. Lorsque vous appelez votre point de terminaison bêta, API Gateway utilise la valeur configurée dans la variable d'étape pour l'étape bêta et appelle un

autre hôte web (par exemple, `beta.example.com`). De même, les variables d'étape peuvent être utilisées pour spécifier un nom de AWS Lambda fonction différent pour chaque étape de votre API.

Vous pouvez également utiliser des variables d'étape pour transmettre des paramètres de configuration à une fonction Lambda via vos modèles de mappage. Par exemple, vous pouvez souhaiter réutiliser la même fonction Lambda pour plusieurs étapes de votre API, mais cette fonction doit lire les données d'une table Amazon DynamoDB différente selon l'étape à laquelle elle est appelée. Dans les modèles de mappage qui génèrent la demande de la fonction Lambda, vous pouvez utiliser des variables d'étape pour transmettre le nom de la table à Lambda.

Exemples

Pour utiliser une variable d'étape afin de personnaliser le point de terminaison de l'intégration HTTP, vous devez d'abord configurer une variable d'étape d'un nom spécifié, par exemple, **`url`**), puis lui attribuer une valeur (par exemple, **`example.com`**). Ensuite, à partir de la configuration de votre méthode, configurez une intégration de proxy HTTP. Au lieu d'entrer l'URL du point de terminaison, vous pouvez demander à API Gateway d'utiliser la valeur de la variable d'étape, **`http://${stageVariables.url}`**. Cette valeur dit à API Gateway de remplacer votre variable d'étape `${}` au moment de l'exécution, en fonction de l'étape où se trouve votre API.

Vous pouvez référencer les variables d'étape de la même manière pour spécifier un nom de fonction Lambda, un chemin de proxy de AWS service ou un AWS ARN de rôle dans le champ des informations d'identification.

Lorsque vous spécifiez un nom de fonction Lambda en tant que valeur de variable d'étape, vous devez configurer les autorisations sur cette fonction Lambda manuellement. Lorsque vous spécifiez une fonction Lambda dans la console API Gateway, une AWS CLI commande apparaît pour configurer les autorisations appropriées. Vous pouvez également utiliser le AWS Command Line Interface (AWS CLI) pour ce faire.

```
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
```

Définition de variables d'étape à l'aide de la console Amazon API Gateway

Dans ce tutoriel, vous allez apprendre à définir des variables d'étape pour deux étapes de déploiement d'un exemple d'API à l'aide de la console Amazon API Gateway. Avant de commencer, vérifiez que vous respectez les conditions requises suivantes :

- Vous devez disposer d'une API disponible dans API Gateway. Suivez les instructions de la section Développement d'une API REST dans API Gateway.
- Vous devez avoir déployé l'API au moins une fois. Suivez les instructions de la section Déploiement d'une API REST dans Amazon API Gateway.
- Vous devez avoir créé la première étape pour une API déployée. Suivez les instructions de la section Création d'une étape.

Pour déclarer des variables d'étape en utilisant la console API Gateway

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Créez une API, puis créez une méthode GET sur la ressource racine de l'API. Définissez le type d'intégration sur HTTP et définissez URL du point de terminaison sur **`http://${stageVariables.url}`**.
3. Déployez l'API vers une nouvelle étape nommée **`beta`**.
4. Dans le volet de navigation principal, choisissez Étapes, puis sélectionnez l'étape bêta.
5. Dans l'onglet Variables d'étape, choisissez Modifier.
6. Choisissez Ajouter une variable d'étape.
7. Pour Name (Nom), saisissez **`url`**. Pour Valeur, entrez **`httpbin.org/get`**.
8. Choisissez Ajouter une variable d'étape, puis procédez comme suit :

   Pour Name (Nom), saisissez **`stageName`**. Pour Valeur, entrez **`beta`**.
9. Choisissez Ajouter une variable d'étape, puis procédez comme suit :

   Pour Name (Nom), saisissez **`function`**. Pour Valeur, entrez **`HelloWorld`**.

   > **Note**
   >
   > Lorsque vous définissez une fonction Lambda en tant que valeur d'une variable d'étape, utilisez le nom local de la fonction, en incluant éventuellement son alias ou sa spécification de version, comme dans **`HelloWorld`**, **`HelloWorld:1`** ou **`HelloWorld:alpha`**. N'utilisez pas l'ARN de la fonction (par exemple, **`arn:aws:lambda:us-east-1:123456789012:function:HelloWorld`**). La console API Gateway considère la valeur de la variable d'étape d'une fonction Lambda

comme le nom de fonction non qualifié et développe la variable d'étape donnée en un ARN.

10. Choisissez Enregistrer.
11. Créez maintenant une deuxième étape. Dans le volet de navigation Étapes, choisissez Créer une étape. Sous Stage name (Nom de l'étape), entrez **`prod`**. Sélectionnez un déploiement récent dans Déploiement, puis choisissez Créer une étape.
12. Comme pour l'étape bêta, définissez les trois variables d'étape (url, stageName et function) sur des valeurs différentes (**`petstore-demo-endpoint.execute-api.com/petstore/pets`**, **`prod`** et **`HelloEveryone`**) respectivement.

    Pour découvrir comment utiliser les variables d'étape, consultez the section called “Utiliser les variables d'étape ”.

Utilisation de variables d'étape Amazon API Gateway

Vous pouvez utiliser les variables d'étape API Gateway pour accéder aux backends HTTP et Lambda pour différentes étapes de déploiement de l'API. Vous pouvez également utiliser des variables d'étape pour transmettre les métadonnées de configuration spécifiques à une étape dans un backend HTTP en tant que paramètre de requête et dans une fonction Lambda en tant que charge utile générée dans un modèle de mappage en entrée.

Prérequis

Vous devez créer deux étapes avec une variable d'étape url définie sur deux points de terminaison HTTP différents : une variable d'étape function affectée à deux fonctions Lambda différentes et une variable d'étape stageName qui contient les métadonnées spécifiques à l'étape.

Accès à un point de terminaison HTTP via une API avec une variable d'étape

1. Dans le volet de navigation Stages, sélectionnez beta. Sous Détails de l'étape, choisissez l'icône de copie pour copier l'URL d'invocation de votre API, puis saisissez l'URL d'invocation de votre API dans un navigateur web. Cela lance la demande `GET` de l'étape beta sur la ressource racine de l'API.

   > **Note**
   >
   > Le lien Invoke URL pointe sur la ressource racine de l'API à l'étape beta. L'entrée de l'URL dans un navigateur web appelle la méthode `GET` à l'étape bêta sur la ressource

racine. Si les méthodes sont définies sur les ressources enfants et non sur la ressource racine elle-même, l'entrée de l'URL dans un navigateur web renvoie une réponse d'erreur `{"message":"Missing Authentication Token"}`. Dans ce cas, vous devez ajouter le nom d'une ressource enfant spécifique au lien Invoke URL.

2. La réponse renvoyée par la demande `GET` à l'étape beta est présentée ci-dessous. Vous pouvez également vérifier le résultat en utilisant un navigateur pour accéder à l'adresse http://httpbin.org/get. Cette valeur a été affectée à la variable `url` à l'étape beta. Les deux réponses sont identiques.
3. Dans le volet de navigation Stages, sélectionnez l'étape prod. Sous Détails de l'étape, choisissez l'icône de copie pour copier l'URL d'invocation de votre API, puis saisissez l'URL d'invocation de votre API dans un navigateur web. Cela lance la demande `GET` de l'étape prod sur la ressource racine de l'API.
4. La réponse renvoyée par la demande `GET` à l'étape prod est présentée ci-dessous. Vous pouvez vérifier le résultat en utilisant un navigateur pour accéder à http ://petstore-demo-endpoint.execute-api.com/petstore/pets. Cette valeur a été affectée à la variable `url` à l'étape prod. Les deux réponses sont identiques.

Transmission des métadonnées spécifiques à l'étape à un backend HTTP via une variable d'étape dans une expression de paramètre de requête

Cette procédure explique comment utiliser une valeur de variable d'étape dans une expression de paramètre de requête pour transmettre les métadonnées spécifiques à l'étape à un backend HTTP. Nous allons utiliser la variable d'étape `stageName` déclarée dans la section Définition de variables d'étape à l'aide de la console Amazon API Gateway.

1. Dans le volet de navigation Resource, sélectionnez la méthode GET.

   Pour ajouter un paramètre de chaîne de requête à l'URL de la méthode, sélectionnez l'onglet Demande de méthode, puis dans la section Paramètres de la demande de méthode, choisissez Modifier.
2. Choisissez Paramètres de chaîne de requête d'URL et procédez comme suit :

   a. Sélectionnez Add query string (Ajouter une chaîne de requêtes).

   b. Pour Name (Nom), saisissez **`stageName`**.

   c. Gardez Obligatoire et Mise en cache désactivés.

3. Choisissez Enregistrer.
4. Choisissez l'onglet Demande d'intégration, puis dans la section Paramètres de la demande d'intégration, sélectionnez Modifier.
5. Pour URL du point de terminaison, ajoutez **`?stageName=${stageVariables.stageName}`** à la valeur d'URL définie précédemment, de sorte que l'URL du point de terminaison soit **`http://${stageVariables.url}?stageName=${stageVariables.stageName}`**.
6. Choisissez Déployer l'API et sélectionnez l'étape bêta.
7. Dans le volet de navigation principal, choisissez Étapes. Dans le volet de navigation Stages, sélectionnez beta. Sous Détails de l'étape, choisissez l'icône de copie pour copier l'URL d'invocation de votre API, puis saisissez l'URL d'invocation de votre API dans un navigateur web.

> **Note**
>
> Nous utilisons ici l'étape beta, car le point de terminaison HTTP spécifié par la variable `url`, « http://httpbin.org/get », accepte les expressions de paramètre de requête et les renvoie en tant qu'objet `args` dans sa réponse.

8. Vous recevez la réponse suivante. Notez que la valeur `beta`, affectée à la variable d'étape `stageName`, est transmise au backend en tant qu'argument `stageName`.

```
{
  "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"
}
```

Appel d'une fonction Lambda via une API avec une variable d'étape

Cette procédure explique comment utiliser une variable d'étape pour appeler une fonction Lambda en tant que backend de votre API. Nous allons utiliser la variable d'étape `function` déclarée précédemment. Pour plus d'informations, consultez Définition de variables d'étape à l'aide de la console Amazon API Gateway.

1. Créez une Fonction Lambda nommée **`HelloWorld`** à l'aide de l'exécution Node.js par défaut. Le code doit contenir ce qui suit :

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

   Pour plus d'informations sur la création d'une Fonction Lambda, consultez Bien démarrer avec la console d'API REST.

2. Dans le volet Ressources, sélectionnez Créer une ressource, puis procédez comme suit :

   a. Pour Chemin de ressources, sélectionnez /.

   b. Sous Resource Name (Nom de la ressource), entrez **`lambdav1`**.

   c. Choisissez Créer une ressource.

3. Choisissez la ressource /lambdav1, puis choisissez Create method.

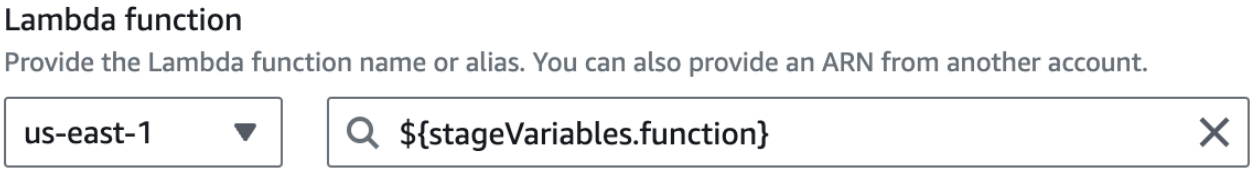
   Ensuite, procédez comme suit :

   a. Pour Type de méthode, sélectionnez GET.

   b. Pour Type d'intégration, sélectionnez Fonction Lambda.

   c. Maintenez Intégration proxy Lambda désactivée.

   d. Sous Lambda function (Fonction Lambda), saisissez `${stageVariables.function}`.

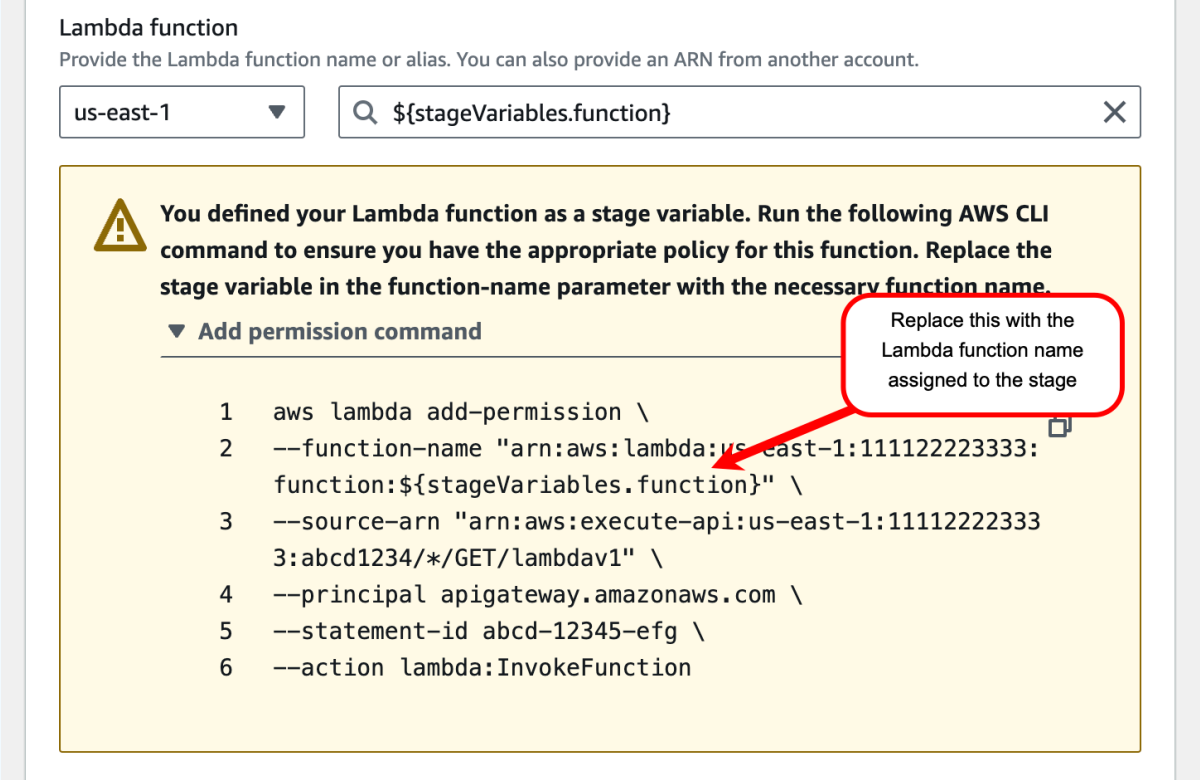


> **Tip**
>
> Lorsque vous y êtes invité avec la commande Ajouter une autorisation, copiez la commande de l'interface de ligne de commande AWS . Exécutez la commande sur chaque Fonction Lambda qui sera affectée à la variable d'étape `function`. Par

exemple, si la `$stageVariables.function` valeur est`HelloWorld`, exécutez la AWS CLI commande suivante :

```
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
```

Si vous ne le faites pas, l'appel de la méthode renverra une réponse `500 Internal Server Error`. Remplacez `${stageVariables.function}` par le nom de la Fonction Lambda qui est affectée à la variable d'étape.



e. Choisissez Créer une méthode.

4. Déployez l'API vers les deux étapes **`prod`** et **`beta`**.
5. Dans le volet de navigation principal, choisissez Étapes. Dans le volet de navigation Stages, sélectionnez beta. Sous Détails de l'étape, choisissez l'icône de copie pour copier l'URL d'invocation de votre API, puis saisissez l'URL d'invocation de votre API dans un navigateur web. Ajoutez **`/lambdav1`** à l'URL avant d'appuyer sur Entrée.

Vous recevez la réponse suivante.

```
"Hello, World! I'm not sure where I'm calling from..."
```

Transmission des métadonnées spécifiques à l'étape à une fonction Lambda via une variable d'étape

Cette procédure explique comment utiliser une variable d'étape pour transmettre les métadonnées de configuration spécifiques à l'étape à une fonction Lambda. Vous créez une méthode `POST` et un modèle de mappage d'entrée pour générer la charge utile à l'aide de la variable d'étape `stageName` déclarée précédemment.

1. Choisissez la ressource /lambdav1, puis choisissez Create method.

   Ensuite, procédez comme suit :

   a. Pour Type de méthode, sélectionnez POST.

   b. Pour Type d'intégration, sélectionnez Fonction Lambda.

   c. Maintenez Intégration proxy Lambda désactivée.

   d. Sous Lambda function (Fonction Lambda), saisissez `${stageVariables.function}`.

   e. Lorsque vous y êtes invité avec la commande Ajouter une autorisation, copiez la commande de l'interface de ligne de commande AWS . Exécutez la commande sur chaque Fonction Lambda qui sera affectée à la variable d'étape `function`.

   f. Choisissez Créer une méthode.
2. Choisissez l'onglet Demande d'intégration, puis dans la section Paramètres de la demande d'intégration, sélectionnez Modifier.
3. Choisissez Modèles de mappage, puis choisissez Ajouter un modèle de mappage.
4. Pour Type de contenu, entrez **`application/json`**.
5. Pour Corps du modèle, entrez le modèle suivant :

```
#set($inputRoot = $input.path('$'))
{
    "stageName" : "$stageVariables.stageName"
}
```

> **Note**
>
> Dans un modèle de mappage, une variable d'étape doit être référencée entre guillemets (comme dans `"$stageVariables.stageName"` ou `"${stageVariables.stageName}"`). Dans d'autres endroits, il doit être référencé sans guillemets (comme dans `${stageVariables.function}`).

6. Choisissez Enregistrer.
7. Déployez l'API vers les deux étapes **`beta`** et **`prod`**.
8. Pour utiliser un client d'API REST afin de transmettre des métadonnées spécifiques à l'étape, procédez comme suit :

   a. Dans le volet de navigation Stages, sélectionnez beta. Sous Détails de l'étape, choisissez l'icône de copie pour copier l'URL d'invocation de votre API, puis entrez l'URL d'invocation de votre API dans le champ d'entrée d'un client d'API REST. Ajoutez **`/lambdav1`** avant de soumettre votre demande.

   Vous recevez la réponse suivante.

   ```
   "Hello, World! I'm calling from the beta stage."
   ```

   b. Dans le volet de navigation Étapes, choisissez prod. Sous Détails de l'étape, choisissez l'icône de copie pour copier l'URL d'invocation de votre API, puis entrez l'URL d'invocation de votre API dans le champ d'entrée d'un client d'API REST. Ajoutez **`/lambdav1`** avant de soumettre votre demande.

   Vous recevez la réponse suivante.

   ```
   "Hello, World! I'm calling from the prod stage."
   ```

9. Pour utiliser la fonctionnalité Test afin de transmettre des métadonnées spécifiques à l'étape, procédez comme suit :

   a. Dans le volet de navigation Ressources, choisissez l'onglet Test. Vous devrez peut-être choisir la flèche droite pour afficher l'onglet.

   b. Pour function, entrez **`HelloWorld`**.

   c. Pour stageName, entrez **`beta`**.

d. Sélectionnez Tester). Vous n'avez pas besoin d'ajouter de corps à votre demande `POST`.

   Vous recevez la réponse suivante.

   ```
   "Hello, World! I'm calling from the beta stage."
   ```

e. Vous pouvez répéter les étapes précédentes pour tester l'étape Prod. Pour stageName, entrez **`Prod`**.

   Vous recevez la réponse suivante.

   ```
   "Hello, World! I'm calling from the prod stage."
   ```

Référence des variables d'étape Amazon API Gateway

Vous pouvez utiliser les variables d'étape API Gateway dans les cas suivants.

Expressions de mappage de paramètres

Une variable d'étape peut être utilisée dans une expression de mappage de paramètres pour un paramètre d'en-tête de demande ou de réponse d'une méthode d'API, sans aucune substitution partielle. Dans l'exemple suivant, la variable d'étape est référencée sans le caractère $ et sans être indiquée entre accolades `{...}`.

- `stageVariables.<variable_name>`

Modèles de mappage

Une variable d'étape peut être utilisée n'importe où dans un modèle de mappage, comme illustré dans les exemples suivants.

- `{ "name" : "$stageVariables.<variable_name>"}`
- `{ "name" : "${stageVariables.<variable_name>}"}`

URI d'intégration HTTP

Une variable d'étape peut être utilisée dans une URL d'intégration HTTP, comme illustré dans les exemples suivants.

- URI complet sans protocole – `http://${stageVariables.<variable_name>}`
- Domaine complet – `http://${stageVariables.<variable_name>}/resource/operation`
- Sous-domaine – `http://${stageVariables.<variable_name>}.example.com/resource/operation`
- Chemin – `http://example.com/${stageVariables.<variable_name>}/bar`
- Chaîne de requête – `http://example.com/foo?q=${stageVariables.<variable_name>}`

AWS URI d'intégration

Une variable d'étape peut être utilisée dans le cadre d'une action d' AWS URI ou de composants de chemin, comme illustré dans l'exemple suivant.

- `arn:aws:apigateway:<region>:<service>:${stageVariables.<variable_name>}`

AWS URI d'intégration (fonctions Lambda)

Une variable d'étape peut être utilisée à la place d'un nom de fonction Lambda, ou de sa version/son alias, comme illustré dans les exemples suivants.

- `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**
>
> Pour utiliser une variable d'étape pour une fonction Lambda, la fonction doit se trouver dans le même compte que l'API. Les variables d'étape ne prennent pas en charge les fonctions Lambda inter-comptes.

Groupe d'utilisateurs Amazon Cognito

Une variable d'étape peut être utilisée à la place d'un groupe d'utilisateurs Amazon Cognito pour un `COGNITO_USER_POOLS` autorisateur.

- `arn:aws:cognito-idp:<region>:<account_id>:userpool/${stageVariables.<variable_name>}`

AWS informations d'identification d'intégration

Une variable d'étape peut être utilisée dans le cadre de l'ARN des informations d'identification AWS utilisateur/rôle, comme indiqué dans l'exemple suivant.

- `arn:aws:iam::<account_id>:${stageVariables.<variable_name>}`

Configuration d'un déploiement de la version canary API Gateway

La version Canary est une stratégie de développement logiciel qui consiste à déployer une nouvelle version d'API (ou de tout autre logiciel) à des fins de test, tandis que la version de base reste déployée en tant que version de production pour l'exploitation normale dans la même étape. Dans le cadre de cette documentation, la version de base est désignée sous le nom de version de production. Même si cela est raisonnable, vous êtes libre d'appliquer une version Canary à une version hors production à des fins de test.

Dans le cadre du déploiement d'une version Canary, l'ensemble du trafic d'API est séparé de façon aléatoire entre la version de publication et la version Canary selon un rapport préconfiguré. En règle générale, la version Canary reçoit un petit pourcentage du trafic d'API et la version de production utilise le reste. Les fonctions d'API mises à jour ne sont visibles au niveau du trafic d'API qu'à travers la version Canary. Vous pouvez ajuster le pourcentage de trafic Canary pour optimiser la couverture ou les performances de test.

En maintenant le trafic Canary à un niveau limité et en conservant la sélection aléatoire, les utilisateurs ne sont pour la plupart jamais perturbés par les bogues éventuels de la nouvelle version, et aucun utilisateur n'est perturbé de façon permanente sur le plan individuel.

Dès lors que les métriques de test ont satisfait à vos exigences, vous pouvez promouvoir la version Canary en version de production et désactiver la Canary dans le déploiement. Les nouvelles fonctions deviennent alors disponibles dans l'étape de production.

Rubriques

Déploiement des versions Canary dans API Gateway

Dans API Gateway, le déploiement d'une version Canary s'appuie sur l'étape de déploiement de la version de production de la version de base d'une API, et attache à l'étape une version Canary pour les nouvelles versions, par rapport à la version de base, de l'API. L'étape est associée au déploiement initial et Canary aux déploiements suivants. Au début, l'étape et Canary pointent toutes les deux vers la même version d'API. Tout au long de cette section, nous parlons indifféremment d'étape et de version de production, mais aussi de Canary et de version Canary.

Pour déployer une API avec une version Canary, vous devez créer un déploiement de version Canary en ajoutant des paramètres Canary à l'étape d'un déploiement standard. Les paramètres Canary décrivent la version Canary sous-jacente, tandis que l'étape représente la version de production de l'API dans le cadre de ce déploiement. Pour ajouter des paramètres Canary, définissez `canarySettings` pour l'étape de déploiement et spécifiez les éléments suivants :

- Un ID de déploiement, au départ identique à l'ID du déploiement de la version de base défini au niveau de l'étape.
- Un pourcentage du trafic d'API, entre 0,0 et 100,0 (inclus), pour la version Canary.
- Variables d'étape de la version Canary qui peuvent remplacer les variables d'étape de version de production.
- L'utilisation du cache d'étape pour les demandes Canary, si useStageCache est défini et si la mise en cache d'API est activée au niveau de l'étape.

Du moment qu'une version Canary est activée, l'étape de déploiement ne peut pas être associée à un autre déploiement de version non Canary tant que la version Canary n'est pas désactivée et que les paramètres Canary ne sont pas supprimés de l'étape.

Lorsque vous activez la journalisation d'exécution d'API, les journaux et les métriques propres à la version Canary sont générés pour toutes les demandes Canary. Ils sont signalés à un groupe de

journaux CloudWatch Logs de l'étape de production ainsi qu'à un groupe de journaux CloudWatch Logs spécifique à la version Canary. Il en va de même pour la journalisation d'accès. Les journaux propres à Canary s'avèrent utiles pour valider les nouvelles modifications apportées à l'API et pour décider s'il convient de les accepter et de promouvoir la version Canary en étape de production, ou s'il est préférable d'abandonner les modifications et de supprimer la version Canary de l'étape de production.

Le groupe de journaux d'exécution s'intitule `API-Gateway-Execution-Logs/{rest-api-id}/{stage-name}` dans le cas de l'étape de production et `API-Gateway-Execution-Logs/{rest-api-id}/{stage-name}/Canary` dans le cas de la version Canary. Pour la journalisation d'accès, vous devez créer un groupe de journaux ou en choisir un existant. Le nom du groupe de journaux d'accès de la version Canary est assorti du suffixe `/Canary`, qui est ajouté au nom du groupe de journaux sélectionné.

Une version Canary peut exploiter le cache d'étape, s'il est activé, pour stocker les réponses et utiliser les entrées mises en cache pour renvoyer des résultats aux prochaines demandes Canary, au cours d'une période time-to-live (TTL) préconfigurée.

Dans le cadre du déploiement d'une version Canary, la version de production et la version Canary de l'API peuvent être associées à une même version ou à des versions différentes. Lorsqu'elles sont associés à des versions différentes, les réponses concernant les demandes de production et Canary sont mises en cache séparément et le cache d'étape renvoie les résultats correspondant aux demandes de production et Canary. Lorsque la version de production et la version Canary sont associées à un même déploiement, le cache d'étape utilise une seule clé de cache pour les deux types de demande et renvoie la même réponse aux mêmes demandes de la version de production et de la version Canary.

Création d'un déploiement de version Canary

La création d'un déploiement de version Canary consiste à déployer l'API en ajoutant en entrée les paramètres Canary à l'opération de création du déploiement.

Vous pouvez également créer un déploiement de version Canary à partir d'un déploiement non Canary existant en formulant une demande `stage:update` de façon à ajouter les paramètres Canary à l'étape.

Au moment de créer un déploiement de version non Canary, vous pouvez spécifier un nom d'étape qui n'existe pas. Dans ce cas, API Gateway crée l'étape. En revanche, nous ne pouvez pas spécifier le nom d'une étape qui n'existe pas au moment de créer un déploiement de version Canary. Vous obtenez dans ce cas une erreur et API Gateway ne crée pas le déploiement de la version Canary.

Vous pouvez créer un déploiement de version Canary dans API Gateway à l'aide de la console API Gateway, de la AWS CLI ou d'un kit SDK AWS.

Rubriques

Création d'un déploiement Canary à l'aide de la console API Gateway

Pour créer un déploiement de version Canary à l'aide de la console API Gateway, suivez les instructions ci-dessous :

Pour créer le déploiement initial d'une version Canary

1. Connectez-vous à la console API Gateway.
2. Choisissez une API REST existante ou créez-en une.
3. Dans le volet de navigation principal, choisissez Resources, puis Déployer l'API. Suivez les instructions à l'écran dans Deploy API pour déployer l'API dans une nouvelle étape.

   À ce stade, vous avez déployé l'API dans une étape de version de production. La prochaine étape va consister à configurer les paramètres Canary au niveau de l'étape et, si nécessaire, à activer la mise en cache, à définir des variables d'étape ou à configurer des journaux d'accès ou d'exécution d'API.
4. Pour activer la mise en cache de l'API ou associer une liste ACL web AWS WAF à l'étape, dans la section Détails de l'étape, choisissez Modifier. Pour plus d’informations, consultez the section called “Paramètres du cache” ou the section called “Pour associer une ACL AWS WAF Web à un stage d'API API Gateway à l'aide de la console API Gateway”.
5. Pour configurer la journalisation d'exécution ou d'accès, dans la section Journaux et suivi, choisissez Modifier et suivez les instructions à l'écran. Pour de plus amples informations, veuillez consulter Configuration de la CloudWatch journalisation pour une API REST dans API Gateway.
6. Pour définir des variables d'étape, choisissez l'onglet Variables d'étape, puis suivez les instructions à l'écran pour ajouter ou modifier des variables d'étape. Pour de plus amples informations, veuillez consulter the section called “Configurer des variables d'étape”.
7. Choisissez l'onglet Canary, puis choisissez Créer Canary. Vous devrez peut-être choisir la flèche droite pour afficher l'onglet Canary.

8. Sous Paramètres Canary, pour Canary, entrez le pourcentage de demandes à renvoyer vers le canary.
9. Si vous le souhaitez, sélectionnez Cache d'étape pour activer la mise en cache pour la version canary. Le cache n'est pas accessible à la version Canary tant que la mise en cache d'API n'est pas activée.
10. Pour remplacer les variables d'étape existantes, pour Remplacement canary, entrez une nouvelle valeur de variable d'étape.

Une fois la version Canary initialisée dans l'étape de déploiement, vous pouvez modifier l'API et tester les modifications. Vous pouvez redéployer l'API dans la même étape pour pouvoir accéder à la version mise à jour et à la version de base via la même étape. Les étapes suivantes expliquent comment procéder.

Pour déployer la dernière version d'API dans une version Canary

1. À chaque mise à jour de l'API, choisissez Déployer l'API.
2. Dans Déployer l'API, choisissez l'étape qui contient un canary dans la liste déroulante Étape de déploiement.
3. (Facultatif) Entrez une description pour Description du déploiement.
4. Choisissez Deploy pour transférer (en mode push) la dernière version d'API vers la version Canary.
5. Si vous le souhaitez, reconfigurez les paramètres d'étape, les journaux ou les paramètres Canary, comme indiqué dans Pour créer le déploiement initial d'une version Canary.

En conséquence, la version Canary pointe vers la dernière version, alors que la version de production continue de pointer vers la version initiale de l'API. La propriété canarySettings affiche désormais une nouvelle valeur deploymentId, alors que l'étape présente toujours la valeur deploymentId initiale. En arrière-plan, la console appelle stage:update.

Création d'un déploiement Canary à l'aide de l AWS CLI

Dans un premier temps, créez un déploiement de base avec deux variables d'étape, mais sans Canary :

```
aws apigateway create-deployment \
    --variables sv0=val0,sv1=val1 \
```

```
    --rest-api-id abcd1234 \
    --stage-name 'prod' \
```

La commande renvoie une représentation de la ressource `Deployment` obtenue, comme dans l'exemple suivant :

```
{
    "id": "du4ot1",
    "createdDate": 1511379050
}
```

L'`id` de déploiement obtenu identifie un instantané (ou une version) de l'API.

À présent, créez un déploiement Canary au niveau de l'étape `prod` :

```
aws apigateway create-deployment  --rest-api-id abcd1234 \
    --canary-settings \
    '{
        "percentTraffic":10.5,
        "useStageCache":false,
        "stageVariableOverrides":{
            "sv1":"val2",
            "sv2":"val3"
        }
    }' \
    --stage-name 'prod'
```

Si l'étape spécifiée (`prod`) n'existe pas, la commande précédente renvoie une erreur. Dans le cas contraire, elle renvoie une représentation de la nouvelle ressource deployment créée comparable à l'exemple suivant :

```
{
    "id": "a6rox0",
    "createdDate": 1511379433
}
```

L'`id` de déploiement obtenu identifie la version de test de l'API pour la version Canary. En conséquence, l'étape associée est activée pour Canary. Vous pouvez afficher cette représentation d'étape en appelant la commande `get-stage`, comme dans l'exemple suivant :

```
aws apigateway get-stage --rest-api-id acbd1234 --stage-name prod
```

L'exemple suivant montre une représentation de `Stage` comme sortie de la commande :

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

Dans cet exemple, la version de base de l'API utilise les variables d'étape `{"sv0":val0", "sv1":val1"}`, tandis que la version de test utilise les variables d'étape `{"sv1":val2", "sv2":val3"}`. La version de production et la version Canary utilisent toutes les deux la même variable d'étape `sv1`, mais avec des valeurs différentes : `val1` et `val2`, respectivement. La variable d'étape `sv0` est seulement utilisée dans la version de production, alors que la variable d'étape `sv2` est utilisée dans la version Canary uniquement.

Vous pouvez créer un déploiement de version Canary à partir d'un déploiement standard existant en mettant à jour l'étape de façon à activer une version Canary. En guise de démonstration, créez d'abord un déploiement standard :

```
aws apigateway create-deployment \
    --variables sv0=val0,sv1=val1 \
    --rest-api-id abcd1234 \
```

```
    --stage-name 'beta'
```

La commande renvoie une représentation du déploiement de la version de base :

```
{
    "id": "cifeiw",
    "createdDate": 1511380879
}
```

L'étape bêta associée ne présente pas de paramètres Canary :

```
{
    "stageName": "beta",
    "variables": {
        "sv0": "val0",
        "sv1": "val1"
    },
    "cacheClusterEnabled": false,
    "cacheClusterStatus": "NOT_AVAILABLE",
    "deploymentId": "cifeiw",
    "lastUpdatedDate": 1511380879,
    "createdDate": 1511380879,
    "methodSettings": {}
}
```

À présent, créez un déploiement de version Canary en attachant une version Canary au niveau de l'étape :

```
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"
        }]'
```

Voici une représentation de l'étape mise à jour :

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

Comme nous venons d'activer une version Canary dans une version existante de l'API, la version de production (`Stage`) et la version Canary (`canarySettings`) pointent toutes les deux vers le même déploiement, c'est-à-dire la même version (`deploymentId`) de l'API. Une fois que vous avez modifié l'API et que vous l'avez redéployée dans cette étape, la nouvelle version se trouve dans la version Canary, alors que la version de base reste dans la version de production. Cela se manifeste dans l'évolution de l'étape lorsque `deploymentId` est mis à jour dans la version Canary avec le nouvel `id` de déploiement et que `deploymentId` reste inchangé dans la version de production.

Mise à jour d'une version Canary

Après avoir déployé une version Canary, vous pouvez souhaiter ajuster le pourcentage du trafic Canary ou activer/désactiver l'utilisation d'un cache d'étape pour optimiser les performances de

test. Vous pouvez également modifier les variables d'étape utilisées dans la version Canary lors de la mise à jour du contexte d'exécution. Pour effectuer de telles mises à jour, appelez l'opération stage:update avec des nouvelles valeurs dans canarySettings.

Vous pouvez mettre à jour une version Canary à l'aide de la console API Gateway, de la commande update-stage de la AWS CLI ou d'un kit SDK AWS.

Rubriques

- Mise à jour d'une version Canary à l'aide de la console API Gateway
- Mise à jour d'une version Canary à l'aide de l AWS CLI

Mise à jour d'une version Canary à l'aide de la console API Gateway

Pour mettre à jour les paramètres Canary existants d'une étape à l'aide de la console API Gateway, procédez comme suit :

Pour mettre à jour les paramètres canary existants

1. Connectez-vous à la console API Gateway et choisissez l'API REST existante.
2. Dans le volet de navigation principal, choisissez Étapes, puis choisissez une étape existante.
3. Choisissez l'onglet Canary, puis Modifier. Vous devrez peut-être choisir la flèche droite pour afficher l'onglet Canary.
4. Mettez à jour Distribution de demande en augmentant ou en diminuant la valeur de pourcentage entre 0,0 et 100,0 (inclus).
5. Activez ou désactivez la case à cocher Cache d'étape.
6. Ajoutez, supprimez ou modifiez Variables d'étape Canary.
7. Choisissez Save (Enregistrer).

Mise à jour d'une version Canary à l'aide de l AWS CLI

Pour mettre à jour une version Canary à l'aide de l'AWS CLI, appelez la commande `update-stage`.

Pour activer ou désactiver l'utilisation d'un cache d'étape pour la version Canary, appelez la commande `update-stage` comme suit :

```
aws apigateway update-stage \
```

```
    --rest-api-id {rest-api-id} \
    --stage-name '{stage-name}'  \
    --patch-operations op=replace,path=/canarySettings/useStageCache,value=true
```

Pour ajuster le pourcentage de trafic Canary, appelez `update-stage` pour remplacer la valeur de `/canarySettings/percentTraffic` de l'étape.

```
aws apigateway update-stage \
    --rest-api-id {rest-api-id} \
    --stage-name '{stage-name}' \
    --patch-operations op=replace,path=/canarySettings/percentTraffic,value=25.0
```

Pour mettre à jour des variables d'étape Canary, notamment en ajoutant, remplaçant ou supprimant une variable d'étape Canary :

```
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"
      }]'
```

Vous pouvez mettre à jour tout ce qui précède en combinant les opérations dans une même valeur `patch-operations` :

```
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"
      }]'
```

Promotion d'une version Canary

La promotion d'une version Canary a pour effet de rendre disponible la version d'API testée dans l'étape de production. L'opération implique les tâches suivantes :

- Réinitialisez l'ID de déploiement de l'étape avec les paramètres d'ID de déploiement de la version Canary. Cette tâche à pour effet de mettre à jour l'instantané d'API de l'étape par rapport à l'instantané de la version Canary, ce qui fait de la version de test aussi la version de production.
- Mise à jour des variables d'étape par rapport aux variables d'étape Canary, le cas échéant. Cette tâche a pour effet de mettre à jour le contexte d'exécution d'API de l'étape par rapport à celui de la version Canary. Sans cette mise à jour, la nouvelle version d'API peut produire des résultats inattendus si la version de test utilise des variables d'étape différentes ou des valeurs différentes pour les variables d'étape existantes.
- Définition du pourcentage de trafic Canary sur 0.0%.

La promotion d'une version Canary n'a pas pour effet de la désactiver au niveau de l'étape. Pour désactiver une version Canary, vous devez supprimer les paramètres Canary de l'étape.

Rubriques

- Promotion d'une version Canary à l'aide de la console API Gateway
- Promotion d'une version Canary à l'aide de l AWS CLI

Promotion d'une version Canary à l'aide de la console API Gateway

Pour promouvoir le déploiement d'une version Canary à l'aide de la console API Gateway, procédez comme suit :

Pour promouvoir un déploiement de version canary

1. Connectez-vous à la console API Gateway et choisissez l'API existante dans le panneau de navigation principal.
2. Dans le volet de navigation principal, choisissez Étapes, puis choisissez une étape existante.
3. Choisissez l'onglet Canary.
4. Choisissez Promouvoir Canary.
5. Confirmez les modifications à apporter, puis choisissez Promouvoir Canary.

À l'issue de la promotion, la version de production fait référence à la même version d'API (deploymentId) que la version Canary. Vous pouvez le vérifier à l'aide de l AWS CLI. Pour obtenir un exemple, consultez the section called “Promotion d'une version Canary à l'aide de l AWS CLI”.

Promotion d'une version Canary à l'aide de l AWS CLI

Pour promouvoir une version Canary en version de production à l'aide des commandes de l'AWS CLI, appelez la commande `update-stage` pour copier le `deploymentId` associé à la version Canary dans le `deploymentId` associé à l'étape, pour réinitialiser le pourcentage de trafic Canary à zéro (`0.0`) et pour copier les variables d'étape liées à la version Canary dans celles liées à l'étape.

Supposons que nous ayons un déploiement de version canary, décrit par une phase semblable à la suivante :

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

Nous appelons la demande `update-stage` ci-dessous pour en assurer la promotion :

```
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"
      }]'
```

Après fois la promotion, l'étape se présente comme suit :

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

Comme vous pouvez le voir, la promotion d'une version canary à l'étape ne désactive pas canary et le déploiement demeure un déploiement de version canary. Pour rendre régulier un déploiement de la version de production, vous devez désactiver la configuration canary. Pour plus d'informations sur la façon de désactiver un déploiement de version canary, consultez the section called “Désactivation d'une version canary”.

Désactivation d'une version canary

Pour désactiver le déploiement d'une version canary, vous devez attribuer à `canarySettings` la valeur null pour la supprimer de l'étape.

Vous pouvez désactiver un déploiement de version Canary à l'aide de la console API Gateway, de la AWS CLI ou d'un kit SDK AWS.

Rubriques

- Désactivation d'une version canary à l'aide de la console API Gateway
- Désactivation d'une version canary à l'aide de l'AWS CLI

Désactivation d'une version canary à l'aide de la console API Gateway

Pour désactiver le déploiement d'une version canary à l'aide de la console API Gateway, suivez les étapes suivantes :

Désactivation d'un déploiement de version canary

1. Connectez-vous à la console API Gateway et choisissez l'API existante dans le volet de navigation principal.
2. Dans le volet de navigation principal, choisissez Étapes, puis choisissez une étape existante.
3. Choisissez l'onglet Canary.
4. Choisissez Supprimer.
5. Confirmez votre souhait de supprimer la version Canary en choisissant Delete.

Résultat : la propriété `canarySettings` prend la valeur `null` et est supprimée de l'étape de déploiement. Vous pouvez le vérifier à l'aide de l AWS CLI. Pour obtenir un exemple, consultez the section called “Désactivation d'une version canary à l'aide de l'AWS CLI”.

Désactivation d'une version canary à l'aide de l'AWS CLI

Pour désactiver le déploiement d'une version canary à l'aide de l'AWS CLI, appelez la commande `update-stage` comme suit :

```
aws apigateway update-stage \
    --rest-api-id abcd1234 \
    --stage-name canary \
    --patch-operations '[{"op":"remove", "path":"/canarySettings"}]'
```

Une réponse positive renvoie une charge utile comparable à la suivante :

```
{
    "stageName": "prod",
    "accessLogSettings": {
        ...
    },
    "cacheClusterEnabled": false,
```

```
    "cacheClusterStatus": "NOT_AVAILABLE",
    "deploymentId": "nfcn0x",
    "lastUpdatedDate": 1511309280,
    "createdDate": 1511152939,
    "methodSettings": {
        ...
    }
}
```

Comme illustré dans la sortie, la propriété `canarySettings` n'est plus présente dans l'étape d'un déploiement canary désactivé.

Mises à jour d'une API REST nécessitant un redéploiement

La gestion d'une API revient à afficher, mettre à jour et supprimer les configurations de l'API existantes. Vous pouvez gérer une API à l'aide de la console API Gateway AWS CLI, d'un SDK ou de l'API REST API Gateway. La mise à jour d'une API implique de modifier certaines propriétés de ressource ou certains paramètres de configuration de l'API. Les mises à jour de ressource nécessitent le redéploiement de l'API, contrairement aux mises à jour de configuration.

Les ressources d'API qui peuvent être mises à jour sont répertoriées dans le tableau suivant.

Mises à jour de ressource d'API nécessitant le redéploiement de l'API

Ressource	Remarques
ApiKey	Pour les propriétés applicables et les opérations prises en charge, veuillez consulter apikey:update. La mise à jour nécessite le redéploiement de l'API.
Mécanisme d'autorisation	Pour les propriétés applicables et les opérations prises en charge, veuillez consulter authorizer:update. La mise à jour nécessite le redéploiement de l'API.
DocumentationPart	Pour les propriétés applicables et les opérations prises en charge, veuillez consulter documentationpart:update. La mise à jour nécessite le redéploiement de l'API.
DocumentationVersion	Pour les propriétés applicables et les opérations prises en charge, veuillez consulter documentationversion:update. La mise à jour nécessite le redéploiement de l'API.

Ressource	Remarques
GatewayResponse	Pour les propriétés applicables et les opérations prises en charge, veuillez consulter gatewayresponse:update. La mise à jour nécessite le redéploiement de l'API.
Integration	Pour les propriétés applicables et les opérations prises en charge, veuillez consulter integration:update. La mise à jour nécessite le redéploiement de l'API.
IntegrationResponse	Pour les propriétés applicables et les opérations prises en charge, veuillez consulter integrationresponse:update. La mise à jour nécessite le redéploiement de l'API.
Method	Pour les propriétés applicables et les opérations prises en charge, veuillez consulter method:update. La mise à jour nécessite le redéploiement de l'API.
MethodResponse	Pour les propriétés applicables et les opérations prises en charge, veuillez consulter methodresponse:update. La mise à jour nécessite le redéploiement de l'API.
Modèle	Pour les propriétés applicables et les opérations prises en charge, veuillez consulter model:update. La mise à jour nécessite le redéploiement de l'API.
RequestValidator	Pour les propriétés applicables et les opérations prises en charge, veuillez consulter requestvalidator:update. La mise à jour nécessite le redéploiement de l'API.
Ressource	Pour les propriétés applicables et les opérations prises en charge, veuillez consulter resource:update. La mise à jour nécessite le redéploiement de l'API.
RestApi	Pour les propriétés applicables et les opérations prises en charge, veuillez consulter restapi:update. La mise à jour nécessite le redéploiement de l'API.
VpcLink	Pour les propriétés applicables et les opérations prises en charge, veuillez consulter vpclink:update. La mise à jour nécessite le redéploiement de l'API.

Les configurations d'API qui peuvent être mises à jour sont répertoriées dans le tableau suivant.

Mises à jour de configuration d'API ne nécessitant pas le redéploiement de l'API

Configuration	Remarques
Compte	Pour les propriétés applicables et les opérations prises en charge, veuillez consulter account:update. La mise à jour ne nécessite pas le redéploiement de l'API.
Déploiement	Pour les propriétés applicables et les opérations prises en charge, veuillez consulter deployment:update.
DomainName	Pour les propriétés applicables et les opérations prises en charge, veuillez consulter domainname:update. La mise à jour ne nécessite pas le redéploie ment de l'API.
BasePathM apping	Pour les propriétés applicables et les opérations prises en charge, veuillez consulter basepathmapping:update. La mise à jour ne nécessite pas le redéploiement de l'API.
Étape	Pour les propriétés applicables et les opérations prises en charge, veuillez consulter stage:update. La mise à jour ne nécessite pas le redéploiement de l'API.
Utilisation	Pour les propriétés applicables et les opérations prises en charge, veuillez consulter usage:update. La mise à jour ne nécessite pas le redéploiement de l'API.
UsagePlan	Pour les propriétés applicables et les opérations prises en charge, veuillez consulter usageplan:update. La mise à jour ne nécessite pas le redéploiement de l'API.

Configuration des noms de domaine personnalisés pour les API REST

Les noms de domaine personnalisés sont des URL plus simples et plus intuitives que vous pouvez fournir à vos utilisateurs d'API.

Après avoir déployé votre API, vous (et vos clients) pouvez appeler cette API à l'aide de l'URL de base par défaut au format suivant :

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

où `api-id` est généré par API Gateway, `region` (AWS Region) est spécifié par vous lors de la création de l'API, et `stage` est spécifié par vous lors du déploiement de l'API.

La partie nom d'hôte de l'URL (c'est-à-dire, *`api-id`*`.execute-api.`*`region`*`.amazonaws.com`) fait référence à un point de terminaison de l'API. Le point de terminaison d'API par défaut peut être difficile à retenir et non convivial.

Avec des noms de domaine personnalisés, vous pouvez configurer le nom d'hôte de votre API et choisir un chemin de base (par exemple, `myservice`) pour mapper l'URL alternative à votre API. Par exemple, une URL de base de l'API plus conviviale peut devenir :

```
https://api.example.com/myservice
```

> **Note**
>
> Un domaine personnalisé régional peut être associé aux API REST et aux API HTTP. Vous pouvez utiliser les API API Gateway version 2 pour créer et gérer des noms de domaine régionaux personnalisés pour les API REST.
> Les noms de domaine personnalisés ne sont pas pris en charge pour les API privées.
> Vous pouvez choisir une version TLS minimale prise en charge par votre compte API REST. Pour les API REST, vous pouvez choisir TLS 1.2 ou TLS 1.0.

Enregistrement d'un nom de domaine

Pour pouvoir configurer des noms de domaine personnalisés pour vos API, vous devez avoir enregistré un nom de domaine Internet. Votre nom de domaine doit respecter la spécification RFC 1035 et peut comporter un maximum de 63 octets par étiquette et 255 octets au total. Si nécessaire, vous pouvez enregistrer un domaine Internet à l'aide de Amazon Route 53 ou utiliser le bureau d'enregistrement de domaine tiers de votre choix. Le nom de domaine personnalisé d'une API peut être le nom d'un sous-domaine ou du domaine racine (également nommé « zone apex ») d'un domaine Internet enregistré.

Après avoir créé un nom de domaine personnalisé dans API Gateway, vous devez créer ou mettre à jour l'enregistrement de ressource de votre fournisseur DNS pour mapper à votre point de

terminaison API. Sans ce mappage, les demandes d'API destinées au nom de domaine personnalisé ne peuvent pas atteindre API Gateway.

> **Note**
>
> Un nom de domaine personnalisé doit être unique au sein d'une région pour tous les AWS comptes.
> Pour déplacer un nom de domaine personnalisé optimisé pour les périphériques entre les régions ou les AWS comptes, vous devez supprimer la CloudFront distribution existante et en créer une nouvelle. Le processus peut prendre environ 30 minutes avant que le nouveau nom de domaine personnalisé ne soit disponible. Pour plus d'informations, consultez la section Mise à jour des distributions CloudFront .

Noms de domaine personnalisés optimisés pour la périphérie

Lorsque vous déployez une API optimisée pour les périphériques, API Gateway configure une CloudFront distribution Amazon et un enregistrement DNS pour mapper le nom de domaine de l'API au nom de domaine de CloudFront distribution. Les demandes relatives à l'API sont ensuite acheminées vers API Gateway via la distribution mappée CloudFront .

Lorsque vous créez un nom de domaine personnalisé pour une API optimisée pour les périphériques, API Gateway configure une CloudFront distribution. Mais vous devez configurer un enregistrement DNS pour mapper le nom de domaine personnalisé au nom de domaine de CloudFront distribution. Ce mappage concerne les demandes d'API liées au nom de domaine personnalisé à acheminer vers API Gateway via la distribution mappée CloudFront . Vous devez également fournir un certificat pour le nom de domaine personnalisé.

> **Note**
>
> La CloudFront distribution créée par API Gateway appartient à un compte spécifique à la région affilié à API Gateway. Lors du suivi des opérations visant à créer et à mettre à jour une telle CloudFront distribution dans CloudWatch Logs, vous devez utiliser cet ID de compte API Gateway. Pour plus d’informations, consultez Enregistrez la création d'un nom de domaine personnalisé CloudTrail.

Pour configurer un nom de domaine personnalisé optimisé pour les périphériques ou pour mettre à jour son certificat, vous devez être autorisé à mettre à jour CloudFront les distributions.

Pour activer l'accès, ajoutez des autorisations à vos utilisateurs, groupes ou rôles :

- Utilisateurs et groupes dans AWS IAM Identity Center :

  Créez un jeu d'autorisations. Suivez les instructions de la rubrique Création d'un jeu d'autorisations du Guide de l'utilisateur AWS IAM Identity Center .

- Utilisateurs gérés dans IAM par un fournisseur d'identité :

  Créez un rôle pour la fédération d'identité. Pour plus d'informations, voir la rubrique Création d'un rôle pour un fournisseur d'identité tiers (fédération) du Guide de l'utilisateur IAM.

- Utilisateurs IAM :
  - Créez un rôle que votre utilisateur peut assumer. Suivez les instructions de la rubrique Création d'un rôle pour un utilisateur IAM du Guide de l'utilisateur IAM.
  - (Non recommandé) Attachez une politique directement à un utilisateur ou ajoutez un utilisateur à un groupe d'utilisateurs. Suivez les instructions de la rubrique Ajout d'autorisations à un utilisateur (console) du Guide de l'utilisateur IAM.

Les autorisations suivantes sont requises pour mettre à jour CloudFront les distributions.

```
{
    "Version": "2012-10-17",
    "Statement": [
         {
            "Sid": "AllowCloudFrontUpdateDistribution",
            "Effect": "Allow",
            "Action": [
                "cloudfront:updateDistribution"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

API Gateway prend en charge les noms de domaine personnalisés optimisés pour les périphériques en tirant parti de l'indication du nom du serveur (SNI) sur la distribution. CloudFront Pour plus d'informations sur l'utilisation de noms de domaine personnalisés sur une CloudFront distribution, notamment sur le format de certificat requis et la taille maximale de la longueur d'une clé de certificat, consultez la section Utilisation de noms de domaine alternatifs et HTTPS dans le manuel Amazon CloudFront Developer Guide.

Pour configurer un nom de domaine personnalisé en tant que nom d'hôte de l'API, vous devez, en tant que propriétaire de celle-ci, fournir un certificat SSL/TLS pour ce nom de domaine.

Afin de fournir un certificat pour un nom de domaine personnalisé optimisé pour la périphérie, vous pouvez demander à AWS Certificate Manager (ACM) de générer un nouveau certificat dans ACM ou d'importer dans ACM un certificat émis par une autorité de certification tierce dans la Région `us-east-1` (USA Est (Virginie du Nord)).

Noms de domaine personnalisés régionaux

Lorsque vous créez un nom de domaine personnalisé pour une API régionale, API Gateway crée un nom de domaine régional pour l'API. Vous devez configurer un enregistrement DNS pour mapper le nom de domaine personnalisé vers le nom de domaine régional. Vous devez également fournir un certificat pour le nom de domaine personnalisé.

Noms de domaine personnalisés génériques

Avec les noms de domaine personnalisés génériques, vous pouvez prendre en charge un nombre presque infini de noms de domaine sans dépasser le quota par défaut. Par exemple, vous pouvez donner à chacun de vos clients son propre nom de domaine, *`customername`*`.api.example.com`.

Pour créer un nom de domaine personnalisé générique, vous pouvez spécifier un caractère générique (*) comme premier sous-domaine d'un domaine personnalisé qui représente tous les sous-domaines possibles d'un domaine racine.

Par exemple, le nom de domaine personnalisé générique `*.example.com` se traduit par des sous-domaines tels que `a.example.com`, `b.example.com` et `c.example.com`, qui effectuent tous un routage vers le même domaine.

Les noms de domaine personnalisés génériques prennent en charge des configurations distinctes des noms de domaine personnalisés standard d'API Gateway. Par exemple, dans un seul AWS compte, vous pouvez configurer `*.example.com` et `a.example.com` vous comporter différemment.

Vous pouvez utiliser les variables de contexte `$context.domainName` et `$context.domainPrefix` pour déterminer le nom de domaine utilisé par un client pour appeler votre API. Pour en savoir plus sur les variables de contexte, veuillez consulter Modèle de mappage API Gateway et référence à la variable de journalisation des accès.

Pour créer un nom de domaine personnalisé générique, vous devez fournir un certificat émis par ACM qui a été validé à l'aide du DNS ou de la méthode de validation par e-mail.

> **Note**
>
> Vous ne pouvez pas créer un nom de domaine personnalisé avec caractère générique si un autre AWS compte a créé un nom de domaine personnalisé en conflit avec le nom de domaine personnalisé avec caractère générique. Par exemple, si le compte A a créé `a.example.com`, le compte B ne peut pas créer le nom de domaine personnalisé générique `*.example.com`.
> Si les comptes A et B ont le même propriétaire, vous pouvez contacter le AWS Centre de support pour demander une exception.

Certificats pour les noms de domaine personnalisés

> **Important**
>
> Vous spécifiez le certificat de votre nom de domaine personnalisé. Si votre application utilise l'épinglage de certificat, parfois appelé épinglage SSL, pour épingler un certificat ACM, il est possible que l'application ne puisse pas se connecter à votre domaine après le AWS renouvellement du certificat. Pour plus d'informations, consultez la section Problèmes d'épinglage de certificat dans le Guide de l'utilisateur AWS Certificate Manager .

Pour fournir un certificat pour un nom de domaine personnalisé dans une région où ACM est pris en charge, vous devez demander un certificat à ACM. Pour fournir un certificat pour un nom de domaine personnalisé régional dans une région où ACM n'est pas pris en charge, vous devez importer un certificat dans API Gateway dans cette région.

Pour importer un certificat SSL/TLS, vous devez fournir le corps du certificat SSL/TLS au format PEM, sa clé privée, ainsi que la chaîne de certificats du nom de domaine personnalisé. Chaque

certificat stocké dans ACM est identifié par son ARN. Pour utiliser un certificat AWS géré pour un nom de domaine, il suffit de référencer son ARN.

ACM permet de configurer et d'utiliser un nom de domaine personnalisé pour une API. Vous créez un certificat pour le nom de domaine donné (ou vous importez un certificat), configurez le nom de domaine dans API Gateway avec l'ARN du certificat fourni par ACM, et vous mappez un chemin de base sous le nom de domaine personnalisé à une étape déployée de l'API. Avec les certificats émis par ACM, vous n'avez pas à vous inquiéter d'une éventuelle exposition des informations sensibles du certificat, par exemple sa clé privée.

Rubriques

Préparation des certificats dans AWS Certificate Manager

Avant de configurer un nom de domaine personnalisé pour une API, vous devez préparer un certificat SSL/TLS dans AWS Certificate Manager. Les étapes suivantes expliquent comment procéder. Pour de plus amples informations, veuillez consulter le Guide de l'utilisateur AWS Certificate Manager.

> **Note**
>
> Pour utiliser un certificat ACM avec un nom de domaine personnalisé optimisé pour la périphérie API Gateway, vous devez demander ou importer le certificat dans la région `us-east-1` (USA Est (Virginie du Nord)). Pour un nom de domaine personnalisé régional API Gateway, vous devez demander ou importer le certificat dans la même région que votre API. Le certificat doit être signé par une autorité de certification approuvée publiquement et couvrir le nom de domaine personnalisé.

Commencez par enregistrer votre domaine Internet, par exemple, *example*.com. Vous pouvez utiliser Amazon Route 53 ou un bureau d'enregistrement accrédité tiers. Pour une liste de ces bureaux, consultez la page Accredited Registrar Directory sur le site Web de l'ICANN.

Pour créer ou importer dans ACM un certificat SSL/TLS pour un nom de domaine, exécutez l'une des actions suivantes :

Pour demander un certificat fourni par ACM pour un nom de domaine

1. Connectez-vous à la console AWS Certificate Manager.
2. Choisissez Request a certificate.
3. Dans Domain name (Nom de domaine), entrez un nom de domaine personnalisé pour votre API, par exemple, `api.example.com`.
4. Le cas échéant, choisissez Add another name to this certificate.
5. Choisissez Review and request.
6. Choisissez Confirm and request.
7. Pour que la demande soit valide et quACM puisse émettre le certificat, le propriétaire inscrit du domaine Internet doit préalablement approuver cette demande.

Importation d'un certificat dans ACM pour un nom de domaine

1. Obtenez un certificat SSL/TLS codé en PEM pour votre nom de domaine personnalisé auprès d'une autorité de certification. Pour obtenir une liste partielle de ces autorités de certification, consultez la Liste de certificats CA inclus dans Mozilla

   a. Générez une clé privée pour le certificat et enregistrez la sortie dans un fichier en utilisant la boîte à outils OpenSSL sur le site Web OpenSSL :

   ```
   openssl genrsa -out private-key-file 2048
   ```

   > **Note**
   >
   > Amazon API Gateway s'appuie sur Amazon CloudFront pour prendre en charge les certificats pour les noms de domaine personnalisés. Ainsi, les exigences et les contraintes d'un certificat SSL/TLS de nom de domaine personnalisé sont dictées par. CloudFront Par exemple, la taille maximale de la clé publique est 2 048 bits et la taille de la clé privée peut être 1 024, 2 048 ou 4 096 bits. La taille de la

clé publique est déterminée par l'autorité de certification (CA) que vous utilisez. Demandez à votre autorité de certification de renvoyer des clés d'une taille différente de la longueur par défaut. Pour de plus amples informations, veuillez consulter Sécurisation de l'accès à vos objets et Création d'URL signées et de cookies signés.

b. Générez une demande de signature de certificat (CSR) avec la clé privée générée précédemment, à l'aide d'OpenSSL :

```
openssl req -new -sha256 -key private-key-file -out CSR-file
```

c. Envoyez cette CSR à l'autorité de certification et enregistrez le certificat obtenu.

d. Téléchargez la chaîne de certificats de l'autorité de certification.

Note

Si vous obtenez la clé privée d'une autre manière et que cette clé est chiffrée, vous pouvez utiliser la commande suivante pour déchiffrer la clé avant de l'envoyer à API Gateway pour configurer un nom de domaine personnalisé.

```
openssl pkcs8 -topk8 -inform pem -in MyEncryptedKey.pem -outform pem -
nocrypt -out MyDecryptedKey.pem
```

2. Téléchargez le certificat sur AWS Certificate Manager :

a. Connectez-vous à la console AWS Certificate Manager.

b. Choisissez Import a certificate.

c. Dans le champ Certificate body (Corps du certificat), saisissez ou collez le corps du certificat de serveur au format PEM de votre autorité de certification. Voici un exemple de certificat abrégé :

```
-----BEGIN CERTIFICATE-----
EXAMPLECA+KgAwIBAgIQJ1XxJ8Pl++gOfQtj0IBoqDANBgkqhkiG9w0BAQUFADBB
...
az8Cg1aicxLBQ7EaWIhhgEXAMPLE
-----END CERTIFICATE-----
```

d. Dans le champ Certificate private key (Clé privée de certificat), saisissez ou collez la clé privée de votre certificat au format PEM. Voici un exemple de clé abrégé :

```
-----BEGIN RSA PRIVATE KEY-----
EXAMPLEBAAKCAQEA2Qb3LDHD7StY7Wj6U2/opV6Xu37qUCCkeDWhwpZMYJ9/nETO
...
1qGvJ3u04vdnzaYN5WoyN5LFckrlA71+CszD1CGSqbVDWEXAMPLE
-----END RSA PRIVATE KEY-----
```

e. Dans le champ Certificate chain (Chaîne de certificats), saisissez ou collez les certificats intermédiaires au format PEM et, le cas échéant, le certificat racine, l'un après l'autre sans ligne vide. Si vous incluez le certificat racine, votre chaîne de certificats doit commencer par les certificats intermédiaires et se terminer par le certificat racine. Utilisez les certificats intermédiaires fournis par l'autorité de certification. N'incluez aucun certificat intermédiaire non approuvé. Voici un exemple abrégé :

```
-----BEGIN CERTIFICATE-----
EXAMPLECA4ugAwIBAgIQWrYdrB5NogYUx1U9Pamy3DANBgkqhkiG9w0BAQUFADCB
...
8/ifBlIK3se2e4/hEfcEejX/arxbx1BJCHBvlEPNnsdw8EXAMPLE
-----END CERTIFICATE-----
```

Voici un autre exemple :

```
-----BEGIN CERTIFICATE-----
Intermediate certificate 2
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
Intermediate certificate 1
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
Optional: Root certificate
-----END CERTIFICATE-----
```

f. Choisissez Review and import.

Une fois que le certificat a été créé ou importé, notez son ARN. Vous en aurez besoin pour configurer le nom de domaine personnalisé.

Choix d'une politique de sécurité pour votre domaine personnalisé dans API Gateway

Pour renforcer la sécurité de votre domaine personnalisé Amazon API Gateway, vous pouvez choisir une politique de sécurité dans la console API Gateway, dans le AWS CLI ou dans un AWS SDK.

Une politique de sécurité est une combinaison prédéfinie de version minimale de TLS et de suites de chiffrement proposées par API Gateway. Vous pouvez choisir une stratégie de sécurité TLS version 1.2 ou TLS version 1.0. Le protocole TLS résout les problèmes de sécurité de réseau tels que la falsification et le risque d'écoute illicite entre un client et un serveur. Lorsque vos clients établissent une liaison TLS vers votre API via le domaine personnalisé, la stratégie de sécurité applique les options de version TLS et de suite de chiffrement que vos clients peuvent choisir d'utiliser.

Dans les paramètres de domaine personnalisé, une stratégie de sécurité détermine deux paramètres :

- Version minimale du protocole TLS utilisée par API Gateway pour communiquer avec des clients d'API
- Le chiffrement utilisé par API Gateway pour chiffrer le contenu renvoyé aux clients d'API

Si vous choisissez une politique de sécurité TLS 1.0, celle-ci accepte le trafic TLS 1.0, TLS 1.2 et TLS 1.3. Si vous choisissez une politique de sécurité TLS 1.2, celle-ci accepte le trafic TLS 1.2 et TLS 1.3 et rejette le trafic TLS 1.0.

> **Note**
>
> Vous ne pouvez définir une politique de sécurité que pour un domaine personnalisé. Pour une API utilisant un point de terminaison par défaut, API Gateway applique la politique de sécurité suivante :
>
> - Pour les API optimisées pour les périphériques : `TLS-1-0`
> - Pour les API régionales : `TLS-1-0`
> - Pour les API privées : `TLS-1-2`

Rubriques

- Comment définir une politique de sécurité pour les domaines personnalisés
- Politiques de sécurité, versions du protocole TLS et chiffrements pris en charge pour les domaines personnalisés optimisés pour les périphériques
- Politiques de sécurité, versions du protocole TLS et chiffrements pris en charge pour les domaines personnalisés régionaux
- Versions de protocole TLS et chiffrements pris en charge pour les API privées

- Noms de chiffrement OpenSSL et RFC
- Informations sur les API HTTP et les WebSocket API

Comment définir une politique de sécurité pour les domaines personnalisés

Lorsque vous créez un nom de domaine personnalisé, vous spécifiez sa politique de sécurité. Pour savoir comment créer un domaine personnalisé, consultez the section called “Création d'un nom de domaine personnalisé optimisé pour la périphérie” outhe section called “Configuration d'un nom de domaine personnalisé régional”.

Pour modifier la politique de sécurité de votre nom de domaine personnalisé, mettez à jour les paramètres de domaine personnalisés. Vous pouvez mettre à jour vos paramètres de nom de domaine personnalisés à l'aide du AWS Management Console AWS CLI, du ou d'un AWS SDK.

Lorsque vous utilisez l'API REST API Gateway ou AWS CLI que vous spécifiez la nouvelle version de TLS, `TLS_1_0` ou `TLS_1_2` dans le `securityPolicy` paramètre. Pour plus d'informations, consultez domainname:update dans le manuel Amazon API Gateway REST API Gateway ou update-domain-namedans le document de référence.AWS CLI

L'opération de mise à jour peut prendre quelques minutes.

Politiques de sécurité, versions du protocole TLS et chiffrements pris en charge pour les domaines personnalisés optimisés pour les périphériques

Le tableau suivant décrit les politiques de sécurité qui peuvent être spécifiées pour les noms de domaine personnalisés optimisés pour les périphériques.

Politique de sécurité	TLS_1_0	TLS_1_2
Protocoles TLS		
TLSv1.3	♦	♦
TLSv1.2	♦	♦
TLSv1.1	♦	
TLSv1	♦	
Chiffrements TLS		

Politique de sécurité	TLS_1_0	TLS_1_2
TLS_AES_128_GCM_SHA256	♦	♦
TLS_AES_256_GCM_SHA384	♦	♦
TLS_CHACHA20_POLY1305_SHA256	♦	♦
ECDHE-ECDSA-AES128-GCM-SHA256	♦	♦
ECDHE-ECDSA-AES128-SHA256	♦	♦
ECDHE-ECDSA-AES128-SHA	♦	
ECDHE-ECDSA-AES256-GCM-SHA384	♦	♦
ECDHE-ECDSA-CHACHA20-POLY1305	♦	♦
ECDHE-ECDSA-AES256-SHA384	♦	♦
ECDHE-ECDSA-AES256-SHA	♦	
ECDHE-RSA-AES128-GCM-SHA256	♦	♦
ECDHE-RSA-AES128-SHA256	♦	♦
ECDHE-RSA-AES128-SHA	♦	
ECDHE-RSA-AES256-GCM-SHA384	♦	♦
ECDHE-RSA-CHACHA20-POLY1305	♦	♦

Politique de sécurité	TLS_1_0	TLS_1_2
ECDHE-RSA-AES256-SHA384	♦	♦
ECDHE-RSA-AES256-SHA	♦	
AES128-GCM-SHA256	♦	
AES256-GCM-SHA384	♦	♦
AES128-SHA256	♦	♦
AES256-SHA	♦	
AES128-SHA	♦	
DES-CBC3-SHA	♦	

Politiques de sécurité, versions du protocole TLS et chiffrements pris en charge pour les domaines personnalisés régionaux

Le tableau suivant décrit les politiques de sécurité qui peuvent être spécifiées pour les noms de domaine personnalisés régionaux.

Politique de sécurité	TLS_1_0	TLS_1_2
Protocoles TLS		
TLSv1.3	♦	♦
TLSv1.2	♦	♦
TLSv1.1	♦	
TLSv1	♦	
Chiffrements TLS		
TLS_AES_128_GCM_SHA256	♦	♦

Politique de sécurité	TLS_1_0	TLS_1_2
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	♦	
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	♦	♦

Politique de sécurité	TLS_1_0	TLS_1_2
AES128-SHA256	♦	♦
AES128-SHA	♦	
AES256-GCM-SHA384	♦	♦
AES256-SHA256	♦	♦
AES256-SHA	♦	

Versions de protocole TLS et chiffrements pris en charge pour les API privées

Le tableau suivant décrit le protocole TLS pris en charge et les chiffrements pour les API privées. La spécification d'une politique de sécurité pour les API privées n'est pas prise en charge.

Politique de sécurité	TLS_1_2
Protocoles TLS	
TLSv1.2	♦
Chiffrements TLS	
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	♦

Politique de sécurité	TLS_1_2
AES128-GCM-SHA256	♦
AES128-SHA256	♦
AES256-GCM-SHA384	♦
AES256-SHA256	♦

Noms de chiffrement OpenSSL et RFC

OpenSSL et IETF RFC 5246 utilisent des noms différents pour les mêmes chiffrements. Le tableau suivant met en correspondance le nom OpenSSL et le nom RFC pour chaque chiffrement.

Nom de chiffrement OpenSSL	Nom de chiffrement RFC
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

Nom de chiffrement OpenSSL	Nom de chiffrement RFC
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
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

Informations sur les API HTTP et les WebSocket API

Pour plus d'informations sur les API HTTP et WebSocket les API, consultez the section called “Politique de sécurité pour les API HTTP” etthe section called “Politique de sécurité pour les WebSocket API”.

Création d'un nom de domaine personnalisé optimisé pour la périphérie

Rubriques

- Configuration d'un nom de domaine personnalisé optimisé pour la périphérie pour une API API Gateway
- Enregistrez la création d'un nom de domaine personnalisé CloudTrail
- Configuration du mappage de chemin de base d'une API à l'aide d'un nom de domaine personnalisé servant de nom d'hôte
- Rotation d'un certificat importé dans ACM
- Appel de l'API avec des noms de domaine personnalisés

Configuration d'un nom de domaine personnalisé optimisé pour la périphérie pour une API API Gateway

La procédure suivante explique comment créer un nom de domaine personnalisé pour une API à l'aide de la console API Gateway.

Pour créer un nom de domaine personnalisé à l'aide de la console API Gateway

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Sélectionnez Noms de domaine personnalisés dans le volet de navigation principal.
3. Choisissez Créer.
4. Pour Nom de domaine, entrez un nom de domaine.
5. Sous Configuration, choisissez Optimisé pour la périphérie.
6. Choisissez une version TLS minimale.
7. Suppression d'un certificat ACM

   > **Note**
   >
   > Pour utiliser un certificat ACM avec un nom de domaine personnalisé optimisé pour la périphérie API Gateway, vous devez demander ou importer le certificat dans la région `us-east-1` (USA Est (Virginie du Nord)).

8. Choisissez Créer un nom de domaine.
9. Une fois le nom de domaine personnalisé créé, la console affiche le nom de domaine de CloudFront distribution associé, sous la forme de*`distribution-id`*`.cloudfront.net`, ainsi que l'ARN du certificat. Notez le nom CloudFront de domaine de distribution indiqué dans le

résultat. Vous en aurez besoin à l'étape suivante pour définir la valeur CNAME ou la cible de l'alias d'enregistrement A du domaine personnalisé dans votre DNS.

> **Note**
>
> Il faut environ 40 minutes pour que le nouveau nom de domaine personnalisé soit prêt. En attendant, vous pouvez configurer l'alias d'enregistrement DNS pour mapper le nom de domaine personnalisé au nom de domaine de CloudFront distribution associé et pour configurer le mappage du chemin de base pour le nom de domaine personnalisé pendant l'initialisation du nom de domaine personnalisé.

10. Ensuite, vous configurez les enregistrements DNS auprès de votre fournisseur DNS pour mapper le nom de domaine personnalisé à la CloudFront distribution associée. Pour obtenir des instructions sur Amazon Route 53, consultez Acheminement du trafic vers Amazon API Gateway à l'aide de votre nom de domaine dans le Manuel du développeur Amazon Route 53.

    Pour la plupart des fournisseurs DNS, un nom de domaine personnalisé est ajouté à la zone hébergée en tant que jeu d'enregistrements de ressources CNAME. Le nom d'enregistrement CNAME spécifie le nom de domaine personnalisé que vous avez entré précédemment dans le champ Nom de domaine (par exemple, `api.example.com`). La valeur d'enregistrement CNAME indique le nom de domaine de la CloudFront distribution. Toutefois, l'utilisation d'un enregistrement CNAME ne fonctionne pas si votre domaine personnalisé est une zone apex (par exemple, `example.com` au lieu d'`api.example.com`). Une zone apex est généralement appelée le domaine racine de votre organisation. Pour une zone apex, vous devez utiliser un alias d'enregistrement A, à condition qu'il soit pris en charge par votre fournisseur DNS.

    Avec Route 53, vous pouvez créer un alias d'enregistrement A pour votre nom de domaine personnalisé et spécifier le nom de domaine de CloudFront distribution comme cible de l'alias. Cela signifie que Route 53 peut acheminer votre nom de domaine personnalisé, même s'il s'agit d'une zone apex. Pour de plus amples informations, veuillez consulter Choix entre des jeux d'enregistrements de ressources avec ou sans alias dans le guide du développeur Amazon Route 53.

    L'utilisation d'alias d'enregistrement A élimine également l'exposition du nom de domaine de CloudFront distribution sous-jacent, car le mappage du nom de domaine s'effectue uniquement au sein de Route 53. Par conséquent, nous vous recommandons d'utiliser des alias d'enregistrement A Route 53 dans la mesure du possible.

Outre la console API Gateway, vous pouvez utiliser l'API REST API Gateway, la AWS CLI ou l'un des AWS SDK pour configurer le nom de domaine personnalisé pour vos API. A titre d'illustration, la procédure suivante présente les étapes à suivre pour effectuer cette opération à l'aide des appels d'API REST.

Pour configurer un nom de domaine personnalisé à l'aide de l'API REST API Gateway

1. Appelez domainname:create, en spécifiant le nom de domaine personnalisé et l'ARN d'un certificat stocké dans AWS Certificate Manager.

   L'appel d'API réussi renvoie une `201 Created` réponse contenant l'ARN du certificat ainsi que le nom de CloudFront distribution associé dans sa charge utile.
2. Notez le nom CloudFront de domaine de distribution indiqué dans le résultat. Vous en aurez besoin à l'étape suivante pour définir la valeur CNAME ou la cible de l'alias d'enregistrement A du domaine personnalisé dans votre DNS.
3. Suivez la procédure précédente pour configurer un alias d'enregistrement A afin de mapper le nom de domaine personnalisé à son nom CloudFront de distribution.

Pour des exemples de code de cet appel d'API REST, veuillez consulter domainname:create.

Enregistrez la création d'un nom de domaine personnalisé CloudTrail

Lorsque cette option CloudTrail est activée pour enregistrer les appels API Gateway effectués par votre compte, API Gateway enregistre les mises à jour de CloudFront distribution associées lorsqu'un nom de domaine personnalisé est créé ou mis à jour pour une API. Ces CloudFront distributions étant détenues par API Gateway, chacune des CloudFront distributions signalées est identifiée par l'un des identifiants de compte API Gateway spécifiques à la région suivants, au lieu de l'identifiant de compte du propriétaire de l'API.

Région	ID de compte
us-east-1	392220576650
us-east-2	718770453195
us-west-1	968246515281
us-west-2	109351309407

Région	ID de compte
ca-central-1	796887884028
eu-west-1	631144002099
eu-west-2	544388816663
eu-west-3	061510835048
eu-central-1	474240146802
eu-central-2	166639821150
eu-north-1	394634713161
eu-south-1	753362059629
eu-south-2	359345898052
ap-northeast-1	969236854626
ap-northeast-2	020402002396
ap-northeast-3	360671645888
ap-southeast-1	195145609632
ap-southeast-2	798376113853
ap-southeast-3	652364314486
ap-southeast-4	849137399833
ap-south-1	507069717855
ap-south-2	644042651268
ap us-east-1	174803364771
sa-east-1	287228555773

Région	ID de compte
me-south-1	855739686837
me-central-1	614065512851

Configuration du mappage de chemin de base d'une API à l'aide d'un nom de domaine personnalisé servant de nom d'hôte

Vous pouvez utiliser un nom de domaine personnalisé unique en tant que nom d'hôte de plusieurs API. Pour cela, vous devez configurer le mappage des chemins de base sur le nom de domaine personnalisé. Le mappage des chemins de base permet de rendre accessible une API sous le domaine personnalisé, en associant le nom de domaine personnalisé et le chemin de base correspondant.

Par exemple, si vous avez créé une API nommée `PetStore` et une autre nommée `PetShop` et configuré un nom de domaine personnalisé `api.example.com` dans API Gateway, vous pouvez affecter à l'URL de l'API `PetStore` la valeur `https://api.example.com` ou `https://api.example.com/myPetStore`. L'API `PetStore` est associée au chemin de base d'une chaîne vide ou de `myPetStore` sous le nom de domaine personnalisé `api.example.com`. De la même façon, vous pouvez affecter un chemin de base `yourPetShop` pour l'API `PetShop`. L'URL de `https://api.example.com/yourPetShop` est alors l'URL racine de l'API `PetShop`.

Effectuez les étapes de avant de définir le chemin de base d'une API Configuration d'un nom de domaine personnalisé optimisé pour la périphérie pour une API API Gateway.

La procédure suivante configure les mappages d'API pour mapper les chemins de votre nom de domaine personnalisé à vos étapes d'API.

Pour créer des mappages d'API à l'aide de la console API Gateway

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Choisissez un nom de domaine personnalisé.
3. Choisissez Configurer les mappages d'API.
4. Choisissez Ajouter un nouveau mappage.
5. Spécifiez l' API, l'étape, et le chemin (facultatif) pour le mappage.
6. Choisissez Enregistrer.

En outre, vous pouvez appeler l'API REST API Gateway, la AWS CLI ou l'un des AWS SDK pour configurer le mappage de chemin de base d'une API avec un nom de domaine personnalisé comme nom d'hôte. A titre d'illustration, la procédure suivante présente les étapes à suivre pour effectuer cette opération à l'aide des appels d'API REST.

Pour définir le mappage de chemin de base d'une API à l'aide de l'API REST API Gateway

- Appelez basepathmapping:create sur un nom de domaine personnalisé donné, en spécifiant le `basePath`, `restApiId` et une propriété `stage` de déploiement dans la charge utile de la demande.

  Si l'appel d'API aboutit, il renvoie le code de réponse `201 Created`.

Pour des exemples de code de l'appel d'API REST, veuillez consulter basepathmapping:create.

Rotation d'un certificat importé dans ACM

ACM gère automatiquement le renouvellement des certificats qu'il émet. Il n'est pas nécessaire de faire alterner les certificats émis par ACM pour vos noms de domaine personnalisés. CloudFront s'en charge en votre nom.

Toutefois, si vous importez un certificat dans ACM et que vous l'utilisez pour un nom de domaine personnalisé, vous devez effectuer la rotation de ce certificat avant qu'il n'arrive à expiration. Cela implique d'importer un nouveau certificat tiers pour le nom de domaine et d'effectuer la rotation du certificat existant vers le nouveau. Vous devez répéter ce processus lorsque le certificat nouvellement importé arrive à expiration. Sinon, vous pouvez demander à ACM d'émettre un nouveau certificat pour le nom de domaine et d'effectuer la rotation du certificat existant vers le nouveau certificat émis par ACM. Ensuite, vous pouvez quitter ACM et CloudFront gérer automatiquement la rotation des certificats. Pour créer ou importer un nouveau certificat ACM, suivez les étapes pour demander ou importer un nouveau certificat ACM pour le nom de domaine spécifié.

Pour faire pivoter un certificat pour un nom de domaine, vous pouvez utiliser la console API Gateway, l'API REST API Gateway, la AWS CLI ou l'un des AWS SDK.

Pour soumettre à rotation un certificat arrivant à expiration qui a été importé dans ACM à l'aide de la console API Gateway

1. Demandez ou importez un certificat dans ACM.
2. Revenez à la console API Gateway.

3. Sélectionnez Custom Domain Names dans le panneau de navigation principal de la console API Gateway.
4. Choisissez un nom de domaine personnalisé.
5. Choisissez Modifier.
6. Choisissez un certificat dans la liste déroulante Certificat ACM.
7. Sélectionnez Save pour commencer la rotation du certificat pour le nom de domaine personnalisé.

> **Note**
>
> Cette opération dure environ 40 minutes. Une fois la rotation effectuée, vous pouvez choisir l'icône représentant une flèche à deux pointes en regard de ACM Certificate pour restaurer le certificat d'origine.

Pour illustrer comment effectuer la rotation par programmation d'un certificat importé pour un nom de domaine personnalisé, nous présentons les étapes à suivre à l'aide de l'API REST API Gateway.

Rotation d'un certificat importé à l'aide de l'API REST API Gateway

- Appelez l'action domainname:update, en spécifiant l'ARN du nouveau certificat ACM pour le nom de domaine personnalisé spécifié.

Appel de l'API avec des noms de domaine personnalisés

L'appel d'une API avec un nom de domaine personnalisé est identique à l'appel de l'API avec son nom de domaine par défaut, sous réserve que l'URL correcte soit utilisée.

Les exemples suivants comparent et opposent un ensemble d'URL par défaut aux URL personnalisées correspondantes de deux API (`udxjef` et `qf3duz`) dans une région (`us-east-1`) et pour un nom de domaine personnalisé (`api.example.com`) donnés.

URL racines des API avec noms de domaine par défaut et personnalisé

ID d'API	Étape	URL par défaut	Chemin de base	URL personnalisé
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 prend en charge les noms de domaine personnalisés pour une API à l'aide de Server Name Indication (SNI). Vous pouvez appeler l'API avec un nom de domaine personnalisé en utilisant un navigateur ou une bibliothèque client prenant en charge SNI.

API Gateway applique le SNI à la CloudFront distribution. Pour plus d'informations sur l' CloudFront utilisation des noms de domaine personnalisés, consultez Amazon CloudFront Custom SSL.

Configuration d'un nom de domaine personnalisé régional dans API Gateway

Vous pouvez créer un nom de domaine personnalisé pour un point de terminaison d'API régional (pour une AWS région). Pour créer un nom de domaine personnalisé, vous devez fournir un certificat ACM spécifique à la région. Pour plus d'informations sur la création ou le chargement d'un certificat de nom de domaine personnalisé, consultez Préparation des certificats dans AWS Certificate Manager.

> **Important**
>
> Pour un nom de domaine personnalisé régional API Gateway, vous devez demander ou importer le certificat dans la même région que votre API.

Lorsque vous créez un nom de domaine personnalisé régional (ou que vous en migrez un) avec un certificat ACM, API Gateway crée un rôle lié à un service dans votre compte, si ce rôle n'existe pas déjà. Le rôle lié à un service est requis pour attacher votre certificat ACM à votre point de terminaison régional. Le rôle est nommé AWSServiceRoleForAPIGatewayet sera associé à la politique GatewayServiceRolePolicy gérée par l'API. Pour de plus amples informations sur l'utilisation du rôle lié à un service, veuillez consulter Utilisation des rôles liés à un service.

> **Important**
>
> Vous devez créer un enregistrement DNS pour pointer le nom de domaine personnalisé vers le nom de domaine régional. Cela permet au trafic lié au nom de domaine personnalisé d'être acheminé vers le nom d'hôte régional de l'API. L'enregistrement DNS peut être de type CNAME ou « A ».

Rubriques

- Configuration d'un nom de domaine personnalisé régional avec certificat ACM à l'aide de la console API Gateway
- Configurez un nom de domaine personnalisé régional avec un certificat ACM à l'aide de AWS CLI

Configuration d'un nom de domaine personnalisé régional avec certificat ACM à l'aide de la console API Gateway

Pour utiliser la console API Gateway afin de configurer un nom de domaine personnalisé régional, procédez comme suit.

Pour configurer un nom de domaine personnalisé régional à l'aide de la console API Gateway

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Sélectionnez Noms de domaine personnalisés dans le volet de navigation principal.
3. Choisissez Créer.
4. Pour Nom de domaine, entrez un nom de domaine.
5. Sous Configuration, choisissez Régional.
6. Choisissez une version TLS minimale.
7. Suppression d'un certificat ACM Le certificat doit se trouver dans la même région que l'API.
8. Choisissez Créer.
9. Suivez la documentation Route 53 sur la configuration de Route 53 pour acheminer le trafic vers API Gateway.

La procédure suivante configure les mappages d'API pour mapper les chemins de votre nom de domaine personnalisé à vos étapes d'API.

Pour créer des mappages d'API à l'aide de la console API Gateway

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Choisissez un nom de domaine personnalisé.
3. Choisissez Configurer les mappages d'API.
4. Choisissez Ajouter un nouveau mappage.
5. Spécifiez l'API, l'Étape et le Chemin du mappage.
6. Choisissez Enregistrer.

Pour de plus amples informations sur la définition des mappages de chemin de base pour le domaine personnalisé, veuillez consulter Configuration du mappage de chemin de base d'une API à l'aide d'un nom de domaine personnalisé servant de nom d'hôte.

Configurez un nom de domaine personnalisé régional avec un certificat ACM à l'aide de AWS CLI

Pour utiliser le AWS CLI afin de configurer un nom de domaine personnalisé pour une API régionale, suivez la procédure suivante.

1. Appelez `create-domain-name`, en spécifiant un nom de domaine personnalisé et l'ARN d'un certificat régional.

```
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
```

   Notez que le certificat spécifié provient de la région `us-west-2` et que pour cet exemple, nous supposons que l'API sous-jacente est de la même région.

   S'il aboutit, cet appel renvoie un résultat similaire à ce qui suit :

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

   La valeur de la propriété `DomainNameConfigurations` renvoie le nom d'hôte de l'API régionale. Vous devez créer un enregistrement DNS pour pointer votre nom de domaine personnalisé vers ce nom de domaine régional. Cela permet au trafic lié au nom de domaine personnalisé d'être acheminé vers le nom d'hôte de cette API régionale.

2. Créez un enregistrement DNS pour associer le nom de domaine personnalisé et le nom de domaine régional. Cela permet aux demandes liées au nom de domaine personnalisé d'être acheminées vers le nom d'hôte régional de l'API.
3. Ajoutez un mappage du chemin de base afin d'exposer les API spécifiées (par exemple, `0qzs2sy7bh`) dans une étape de déploiement (par exemple, `test`) sous le nom de domaine personnalisé spécifié (par exemple, `regional.example.com`).

```
aws apigatewayv2 create-api-mapping \
    --domain-name 'regional.example.com' \
    --api-mapping-key 'myApi' \
    --api-id 0qzs2sy7bh \
    --stage 'test'
```

   En conséquence, l'URL de base utilisant le nom de domaine personnalisé pour l'API déployée pendant l'étape devient `https://regional.example.com/myAPI`.
4. Configurez vos enregistrements DNS pour mapper le nom de domaine personnalisé régional à son nom d'hôte de l'ID de la zone hébergée donné. Commencez par créer un fichier JSON qui contient la configuration permettant de paramétrer un enregistrement DNS pour le nom de domaine régional. L'exemple suivant décrit comment créer un enregistrement `A` DNS afin de mapper un nom de domaine personnalisé régional (`regional.example.com`) à son nom d'hôte régional (`d-numh1z56v6.execute-api.us-west-2.amazonaws.com`) mis en service lors de la création du nom de domaine personnalisé. Les propriétés `DNSName` et `HostedZoneId` d'`AliasTarget` peuvent prendre respectivement les valeurs `regionalDomainName` et `regionalHostedZoneId` du nom de domaine personnalisé. Vous pouvez également obtenir les ID de la zone hébergée Route 53 régionale dans Points de terminaison et quotas Amazon API Gateway.

```
{
  "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. Exécutez la commande CLI suivante :

```
aws route53 change-resource-record-sets \
    --hosted-zone-id {your-hosted-zone-id} \
    --change-batch file://path/to/your/setup-dns-record.json
```

où *{your-hosted-zone-id}* est l'ID de zone hébergée Route 53 de l'enregistrement DNS défini dans votre compte. La valeur du `change-batch` paramètre pointe vers un fichier JSON (*setup-dns-record.json*) dans un dossier (*path/to/your*).

Migration d'un nom de domaine personnalisé à un point de terminaison d'API différent

Vous pouvez migrer votre nom de domaine personnalisé entre des points de terminaison optimisés pour les périphériques et régionaux. Vous ajoutez d'abord le nouveau type de configuration de point de terminaison à la liste `endpointConfiguration.types` existante pour le nom de domaine personnalisé. Ensuite, vous configurez un enregistrement DNS pour pointer le nom de domaine personnalisé vers le point de terminaison récemment configuré. Une dernière étape facultative consiste à supprimer les données de configuration du nom de domaine personnalisé obsolète.

Lorsque vous planifiez la migration, n'oubliez pas que pour un nom de domaine personnalisé de l'API optimisée pour la périphérie, le certificat requis fourni par ACM doit provenir de la région USA Est (Virginie du Nord) (`us-east-1`). Ce certificat est distribué à tous les emplacements géographiques. Toutefois, pour une API régionale, le certificat ACM pour le nom de domaine régional doit provenir de la même région hébergeant l'API. Vous pouvez migrer un nom de domaine personnalisé optimisé pour la périphérie qui ne se trouve pas dans la région `us-east-1` vers un nom de domaine personnalisé régional en demandant d'abord un nouveau certificat ACM de la région locale vers l'API.

Il peut falloir jusqu'à 60 secondes pour effectuer une migration entre un nom de domaine personnalisé optimisé pour la périphérie et un nom de domaine personnalisé régional dans API Gateway. Pour que le point de terminaison que vous venez de créer accepte le trafic, le délai de migration dépend également du moment où vous mettez à jour vos enregistrements DNS.

Rubriques

- Migrez des noms de domaine personnalisés à l'aide du AWS CLI

Migrez des noms de domaine personnalisés à l'aide du AWS CLI

Pour utiliser le AWS CLI pour migrer un nom de domaine personnalisé d'un point de terminaison optimisé pour les périphériques vers un point de terminaison régional ou vice versa, appelez la `update-domain-name`commande pour ajouter le nouveau type de point de terminaison et, éventuellement, appelez la `update-domain-name`commande pour supprimer l'ancien type de point de terminaison.

Rubriques

- Migration d'un nom de domaine personnalisé optimisé pour les périphériques en régional
- Migration d'un nom de domaine personnalisé régional en optimisé pour les périphériques

Migration d'un nom de domaine personnalisé optimisé pour les périphériques en régional

Pour migrer un nom de domaine personnalisé optimisé pour les périphériques vers un nom de domaine personnalisé régional, appelez la commande CLI `update-domain-name`, comme suit :

```
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' } \
      ]
```

Le certificat régional doit se situer dans la même région que l'API régionale.

La réponse positive renvoie un code de statut `200 OK` et un corps similaire à ce qui suit :

```
{
    "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"
}
```

Pour le nom de domaine personnalisé régional migré, la propriété `regionalDomainName` résultante renvoie le nom d'hôte de l'API régionale. Vous devez configurer un enregistrement DNS pour pointer le nom de domaine personnalisé régional vers ce nom d'hôte régional. Cela permet au trafic lié au nom de domaine personnalisé d'être acheminé vers le nom d'hôte régional.

Une fois l'enregistrement DNS défini, vous pouvez supprimer le nom de domaine personnalisé optimisé pour les périphériques en appelant la `update-domain-name` commande suivante : AWS CLI

```
aws apigateway update-domain-name \
    --domain-name api.example.com \
    --patch-operations [ \
            {op:'remove', path:'/endpointConfiguration/types', value:'EDGE'}, \
            {op:'remove', path:'certificateName'}, \
            {op:'remove', path:'certificateArn'} \
        ]
```

Migration d'un nom de domaine personnalisé régional en optimisé pour les périphériques

Pour migrer un nom de domaine personnalisé régional vers un nom de domaine personnalisé optimisé pour les périphériques, appelez la `update-domain-name` AWS CLI commande suivante :

```
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' } \
        ]
```

Le certificat de domaine optimisé pour les périphériques doit être créé dans la région `us-east-1`.

La réponse positive renvoie un code de statut `200 OK` et un corps similaire à ce qui suit :

```
{
    "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"
}
```

Pour le nom de domaine personnalisé spécifié, API Gateway renvoie le nom d'hôte de l'API optimisée pour la périphérie en tant que valeur de propriété `distributionDomainName`. Vous devez définir un enregistrement DNS pour pointer le nom de domaine personnalisé optimisé pour les périphériques vers ce nom de domaine de distribution. Cela permet au trafic lié au nom de domaine personnalisé optimisé pour les périphériques d'être acheminé vers le nom d'hôte d'API optimisé pour les périphériques.

Une fois l'enregistrement DNS défini, vous pouvez supprimer le type de point de terminaison `REGION` du nom de domaine personnalisé :

```
aws apigateway update-domain-name \
    --domain-name api.example.com \
    --patch-operations [ \
        {op:'remove', path:'/endpointConfiguration/types', value:'REGIONAL'}, \
        {op:'remove', path:'regionalCertificateArn'} \
      ]
```

Le résultat de cette commande est similaire à la sortie suivante, seulement avec les données de configuration de nom de domaine optimisé pour la périphérie :

```
{
```

```
    "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"
    }
}
```

Utilisation des mappages d'API pour les API REST

Les mappages d'API vous permettent de connecter des étapes d'API à un nom de domaine personnalisé. Après avoir créé un nom de domaine et configuré les enregistrements DNS, vous pouvez utiliser les mappages d'API pour envoyer le trafic vers vos API via votre nom de domaine personnalisé.

Un mappage d'API spécifie une API, une étape et éventuellement un chemin à utiliser pour le mappage. Par exemple, vous pouvez mapper l'étape `production` d'une API à `https://api.example.com/orders`.

Vous pouvez mapper les étapes d'API HTTP et REST au même nom de domaine personnalisé.

Avant de créer un mappage d'API, vous devez disposer d'une API, d'une étape et d'un nom de domaine personnalisé. Pour plus d'informations sur la création d'un nom de domaine personnalisé, consultez the section called “Configuration d'un nom de domaine personnalisé régional”.

Routage des demandes d'API

Vous pouvez configurer des mappages d'API à plusieurs niveaux, par exemple `orders/v1/items` et `orders/v2/items`.

> **Note**
>
> Pour configurer des mappages d'API à plusieurs niveaux, votre nom de domaine personnalisé doit être régional et utiliser la stratégie de sécurité TLS 1.2.

Pour les mappages d'API à plusieurs niveaux, API Gateway achemine les demandes vers le mappage d'API dont le chemin d'accès est le plus long. API Gateway prend uniquement en compte

les chemins configurés pour les mappages d'API, et non les routes d'API, pour sélectionner l'API à appeler. Si aucun chemin ne correspond à la demande, API Gateway envoie celle-ci à l'API que vous avez mappée au chemin vide (`none`).

Pour les noms de domaine personnalisés qui utilisent les mappages d'API à plusieurs niveaux, API Gateway achemine les demandes vers le mappage d'API doté du préfixe correspondant le plus long.

Par exemple, imaginons un nom de domaine personnalisé `https://api.example.com` doté des mappages d'API suivants :

1. (`none`) mappé à l'API 1.
2. `orders` mappé à l'API 2.
3. `orders/v1/items` mappé à l'API 3.
4. `orders/v2/items` mappé à l'API 4.
5. `orders/v2/items/categories` mappé à l'API 5.

Requête	API sélectionnée	Explication
`https://api.example.com/orders`	`API 2`	La demande correspond exactement à ce mappage d'API.
`https://api.example.com/orders/v1/items`	`API 3`	La demande correspond exactement à ce mappage d'API.
`https://api.example.com/orders/v2/items`	`API 4`	La demande correspond exactement à ce mappage d'API.
`https://api.example.com/orders/v1/items/123`	`API 3`	API Gateway choisit le mappage d'API dont le chemin d'accès est le plus long. La présence de `123` à la fin de la demande n'affecte pas la sélection.

Requête	API sélectionnée	Explication
`https://api.example.com/orders/v2/items/categories/5`	`API 5`	API Gateway choisit le mappage d'API dont le chemin d'accès est le plus long.
`https://api.example.com/customers`	`API 1`	API Gateway utilise le mappage vide comme fourre-tout.
`https://api.example.com/ordersandmore`	`API 2`	API Gateway choisit le mappage d'API doté du préfixe correspondant le plus long. Pour un nom de domaine personnalisé configuré avec des mappages à un seul niveau, tels que `https://api.example.com/orders` et `https://api.example.com/` uniquement, API Gateway choisirait `API 1`, car il n'y a pas de chemin correspondant avec `ordersandmore`.

Restrictions

- Dans un mappage d'API, le nom de domaine personnalisé et les API mappées doivent se trouver dans le même AWS compte.
- Les mappages d'API ne doivent contenir que des lettres, des chiffres et les caractères suivants : `$-_.+!*'()/`.
- La longueur maximale du chemin d'un mappage d'API est de 300 caractères.
- Vous pouvez disposer de 200 mappages d'API à plusieurs niveaux pour chaque nom de domaine.
- Vous ne pouvez mapper les API HTTP à un nom de domaine personnalisé régional qu'à l'aide de la stratégie de sécurité TLS 1.2.

- Vous ne pouvez pas associer WebSocket des API au même nom de domaine personnalisé qu'une API HTTP ou une API REST.

Créer un mappage d'API

Pour créer un mappage d'API, vous devez d'abord créer un nom de domaine personnalisé, une API et une étape. Pour plus d'informations sur la création d'un nom de domaine personnalisé, consultez the section called “Configuration d'un nom de domaine personnalisé régional”.

Pour des exemples AWS Serverless Application Model de modèles qui créent toutes les ressources, voir Sessions avec SAM activé GitHub.

AWS Management Console

Pour créer un mappage d'API

1. Connectez-vous à la console API Gateway à l'adresse : https://console.aws.amazon.com/apigateway.
2. Choisissez Noms de domaine personnalisés.
3. Sélectionnez un nom de domaine personnalisé que vous avez déjà créé.
4. Choisissez Mappages d'API.
5. Choisissez Configurer les mappages d'API.
6. Choisissez Ajouter un nouveau mappage.
7. Entrez une API, une Étape et, éventuellement, un Chemin d'accès.
8. Choisissez Enregistrer.

AWS CLI

La AWS CLI commande suivante crée un mappage d'API. Dans cet exemple, API Gateway envoie des demandes `api.example.com/v1/orders` à l'API et à l'étape spécifiés.

> **Note**
>
> Pour créer des mappages d'API à plusieurs niveaux, vous devez utiliser `apigatewayv2`.

```
aws apigatewayv2 create-api-mapping \
```

```
    --domain-name api.example.com \
    --api-mapping-key v1/orders \
    --api-id a1b2c3d4 \
    --stage test
```

AWS CloudFormation

L' AWS CloudFormation exemple suivant crée un mappage d'API.

> **Note**
>
> Pour créer des mappages d'API à plusieurs niveaux, vous devez utiliser `AWS::ApiGatewayV2`.

```
MyApiMapping:
  Type: 'AWS::ApiGatewayV2::ApiMapping'
  Properties:
    DomainName: api.example.com
    ApiMappingKey: 'orders/v2/items'
    ApiId: !Ref MyApi
    Stage: !Ref MyStage
```

Désactivation du point de terminaison par défaut pour une API REST

Par défaut, les clients peuvent appeler votre API en utilisant le point de terminaison `execute-api` généré par API Gateway pour votre API. Pour vous assurer que les clients peuvent accéder à votre API en utilisant uniquement un nom de domaine personnalisé, désactivez le point de terminaison par défaut `execute-api`. Les clients peuvent toujours se connecter à votre point de terminaison par défaut, mais ils recevront un code d'état `403 Forbidden`.

> **Note**
>
> Lorsque vous désactivez le point de terminaison par défaut, toutes les étapes d'une API sont affectées.

La AWS CLI commande suivante désactive le point de terminaison par défaut pour une API REST.

```
aws apigateway update-rest-api \
    --rest-api-id abcdef123 \
    --patch-operations op=replace,path=/disableExecuteApiEndpoint,value='True'
```

Après avoir désactivé le point de terminaison par défaut, vous devez déployer votre API pour que la modification prenne effet.

La AWS CLI commande suivante crée un déploiement.

```
aws apigateway create-deployment \
    --rest-api-id abcdef123 \
    --stage-name dev
```

Configurer des surveillances de l'état personnalisées pour le basculement DNS.

Vous pouvez utiliser les contrôles de santé d'Amazon Route 53 pour contrôler le basculement du DNS entre une API API Gateway située dans une région principale Région AWS et une API située dans une région secondaire. Cela peut aider à atténuer les impacts en cas de problème régional. Si vous utilisez un domaine personnalisé, vous pouvez effectuer un basculement sans demander aux clients de changer de point de terminaison d'API.

Lorsque vous choisissez Evaluate Target Health (Évaluer l'intégrité de la cible) pour un enregistrement d'alias, ces enregistrements échouent uniquement lorsque le service API Gateway est indisponible dans la région. Dans certains cas, vos propres API d'API Gateway peuvent subir une interruption avant cela. Pour contrôler directement le basculement DNS, configurez des surveillances de l'état Route 53 personnalisées pour vos API d'API Gateway. Dans cet exemple, vous utilisez une CloudWatch alarme qui aide les opérateurs à contrôler le basculement du DNS. Pour plus d'exemples et d'autres considérations lors de la configuration du basculement, consultez les sections Création de mécanismes de reprise après sinistre à l'aide de Route 53 et Réalisation de contrôles de santé de Route 53 sur des ressources privées dans un VPC AWS Lambda avec et. CloudWatch

Rubriques

- Prérequis
- Étape 1 : Configurer les ressources
- Étape 2 : Initier le basculement vers la région secondaire
- Étape 3 : Test du basculement
- Étape 4 : Retour à la région principale

- Prochaines étapes : personnalisez et testez régulièrement

Prérequis

Pour effectuer cette procédure, vous devez créer et configurer les ressources suivantes :

- Nom de domaine qui vous appartient.
- Un certificat ACM pour ce nom de domaine en deux Régions AWS. Pour plus d'informations, consultez the section called “Préparation des certificats dans AWS Certificate Manager”.
- Zone hébergée Route 53 pour votre nom de domaine. Pour obtenir plus d'informations, consultez Working with hosted zones (Utiliser des zones hébergées) dans le Guide du développeur Amazon Route 53.

Pour plus d'informations sur la manière de créer des enregistrements DNS de basculement Route 53 pour les noms de domaine, consultez Choix d'une politique de routage dans le Guide du développeur Amazon Route 53. Pour plus d'informations sur la surveillance d'une CloudWatch alarme, consultez la section Surveillance CloudWatch d'une alarme dans le manuel Amazon Route 53 Developer Guide.

Étape 1 : Configurer les ressources

Dans cet exemple, vous créez les ressources suivantes pour configurer le basculement DNS pour votre nom de domaine :

- API Gateway (API) en deux Régions AWS
- Noms de domaine personnalisés API Gateway portant le même nom sur deux Régions AWS
- Mappages d'API d'API Gateway qui connectent vos API d'API Gateway aux noms de domaine personnalisés
- Enregistrements DNS de basculement de Route 53 pour les noms de domaine
- Une CloudWatch alarme dans la région secondaire
- Un bilan de santé de la Route 53 basé sur l' CloudWatch alarme de la région secondaire

Tout d'abord, vérifiez que vous disposez de toutes les ressources nécessaires dans les régions principale et secondaire. La région secondaire doit contenir l'alarme et la surveillance de l'état. De cette façon, vous ne dépendez pas de la région principale pour effectuer le basculement. Pour des

exemples AWS CloudFormation de modèles qui créent ces ressources, voir primary.yamlet secondary.yaml.

> **Important**
>
> Avant le basculement vers la région secondaire, vérifiiez que toutes les ressources nécessaires sont disponibles. Sinon, votre API ne sera pas prête pour le trafic dans la région secondaire.

Étape 2 : Initier le basculement vers la région secondaire

Dans l'exemple suivant, la région de secours reçoit une CloudWatch métrique et initie le basculement. Nous utilisons une métrique personnalisée qui nécessite l'intervention de l'opérateur pour déclencher le basculement.

```
aws cloudwatch put-metric-data \
    --metric-name Failover \
    --namespace HealthCheck \
    --unit Count \
    --value 1 \
    --region us-west-1
```

Remplacez les données métriques par les données correspondantes pour l' CloudWatch alarme que vous avez configurée.

Étape 3 : Test du basculement

Appelez votre API et vérifiez que vous obtenez une réponse de la région secondaire. Si vous avez utilisé les modèles d'exemple à l'étape 1, la réponse passe de `{"message": "Hello from the primary Region!"}` à `{"message": "Hello from the secondary Region!"}` après le basculement.

```
curl https://my-api.example.com

{"message": "Hello from the secondary Region!"}
```

Étape 4 : Retour à la région principale

Pour revenir à la région principale, envoyez une CloudWatch métrique qui permet de valider le bilan de santé.

```
aws cloudwatch put-metric-data \
    --metric-name Failover \
    --namespace HealthCheck \
    --unit Count \
    --value 0 \
    --region us-west-1
```

Remplacez les données métriques par les données correspondantes pour l' CloudWatch alarme que vous avez configurée.

Appelez votre API et vérifiez que vous obtenez une réponse de la région principale. Si vous avez utilisé les modèles d'exemple à l'étape 1, la réponse passe de `{"message": "Hello from the secondary Region!"}` à `{"message": "Hello from the primary Region!"}`.

```
curl https://my-api.example.com

{"message": "Hello from the primary Region!"}
```

Prochaines étapes : personnalisez et testez régulièrement

Cet exemple montre une façon de configurer le basculement DNS. Vous pouvez utiliser une variété de CloudWatch métriques ou de points de terminaison HTTP pour les contrôles de santé qui gèrent le basculement. Testez régulièrement vos mécanismes de basculement pour vous assurer qu'ils fonctionnent comme prévu et que les opérateurs sont familiarisés avec vos procédures de basculement.

Optimisation des performances des API REST

Une fois que vous avez rendu votre API disponible en appel, vous pouvez constater qu'elle doit être optimisée pour améliorer sa réactivité. API Gateway fournit quelques stratégies d'optimisation de votre API, comme la mise en cache de réponse et la compression de la charge utile. Dans cette section, vous pouvez apprendre à activer ces fonctionnalités.

Rubriques

- Activation de la mise en cache des API pour améliorer la réactivité
- Activation de la compression de la charge utile pour une API

Activation de la mise en cache des API pour améliorer la réactivité

Vous pouvez activer la mise en cache des API dans Amazon API Gateway pour mettre en cache les réponses de votre point de terminaison. Avec la mise en cache, vous pouvez réduire le nombre d'appels envoyés à votre point de terminaison et également améliorer la latence des demandes adressées à votre API.

Lorsque vous activez la mise en cache pour une étape, API Gateway met en cache les réponses de votre point de terminaison pendant une période spécifiée time-to-live (TTL), en secondes. API Gateway répond ensuite à la demande en recherchant la réponse du point de terminaison dans le cache au lieu d'envoyer une demande à votre point de terminaison. La valeur TTL par défaut pour la mise en cache des API est 300 secondes. La valeur TTL maximale est 3 600 secondes. La valeur TTL = 0 signifie que la mise en cache est désactivée.

> **Note**
>
> La mise en cache est la meilleure solution. Vous pouvez utiliser les `CacheMissCount` métriques `CacheHitCount` et d'Amazon CloudWatch pour surveiller les demandes traitées par API Gateway à partir du cache d'API.

La taille maximale d'une réponse qui peut être mise en cache est de 1 048 576 octets. Le chiffrement des données de cache peut augmenter la taille de la réponse lorsqu'elle est mise en cache.

Il s'agit d'un service admissible en vertu de la loi HIPAA. Pour plus d'informations sur AWS le Health Insurance Portability and Accountability Act des États-Unis de 1996 (HIPAA) et sur l'utilisation de AWS services pour traiter, stocker et transmettre des informations de santé protégées (PHI), consultez la section Présentation de la HIPAA.

> **Important**
>
> Lorsque vous activez la mise en cache pour une étape, elle est activée seulement pour les méthodes `GET` par défaut. Cela permet d'assurer la sécurité et la disponibilité de votre API. Vous pouvez activer la mise en cache pour d'autres méthodes en remplaçant les paramètres de méthode.

> **Important**
>
> La mise en cache est facturée à l'heure en fonction de la taille du cache que vous avez choisie. La mise en cache n'est pas éligible au niveau AWS gratuit. Pour plus d'informations, consultez API Gateway Pricing (Tarification d'API Gateway).

Activation de la mise en cache Amazon API Gateway

Dans API Gateway, vous pouvez activer la mise en cache pour une étape spécifique.

Lorsque vous activez la mise en cache, vous devez choisir une capacité de cache. En général, une plus grande capacité offre de meilleures performances, mais coûte également plus cher. Pour connaître les tailles de cache prises cacheClusterSizeen charge, consultez le guide API Gateway API Reference.

API Gateway active la mise en cache en créant une instance de cache dédiée. Ce processus peut prendre jusqu'à 4 minutes.

API Gateway modifie la capacité de mise en cache en supprimant l'instance de cache existante et en en créant une avec une nouvelle capacité. Toutes les données mises en cache existantes sont supprimées.

> **Note**
>
> La capacité du cache affecte le processeur, la mémoire et la bande passante réseau de l'instance de cache. Par conséquent, la capacité du cache peut affecter les performances de ce dernier.
> API Gateway vous recommande d'exécuter un test de charge de 10 minutes pour vérifier que votre capacité de cache est appropriée à votre charge de travail. Assurez-vous que le trafic pendant le test de charge reflète le trafic de production. Par exemple, incluez la montée en puissance, le trafic constant et les pics de trafic. Le test de charge doit inclure des réponses pouvant être fournies à partir du cache, ainsi que des réponses uniques qui ajoutent des éléments au cache. Surveillez les mesures de latence, 4xx, 5xx, d'accès au cache et d'échec d'accès au cache pendant le test de charge. Ajustez votre capacité de cache selon vos besoins en fonction de ces mesures. Pour obtenir plus d'informations sur les tests de charge, consultez l'article How do I select the best API Gateway cache capacity to avoid hitting a rate

limit? (Comment sélectionner la meilleure capacité de cache d'API Gateway pour éviter de se heurter à une limite de débit ?)

Dans la console API Gateway, vous configurez la mise en cache sur la page Stages. Vous provisionnez le cache de stage et spécifiez un paramètre de cache par défaut au niveau de la méthode. Si vous activez le cache par défaut au niveau de la méthode, la mise en cache au niveau de la méthode est activée pour toutes les `GET` méthodes de votre scène, sauf si cette méthode comporte un remplacement de méthode. Toutes `GET` les méthodes supplémentaires que vous déployez sur votre scène disposeront d'un cache au niveau de la méthode. Pour configurer les paramètres de mise en cache au niveau de la méthode pour des méthodes spécifiques de votre étape, vous pouvez utiliser des remplacements de méthodes. Pour plus d'informations sur les remplacements de méthode, consultezthe section called “Remplacement de la mise en cache au niveau de l'étape pour la mise en cache de méthode”.

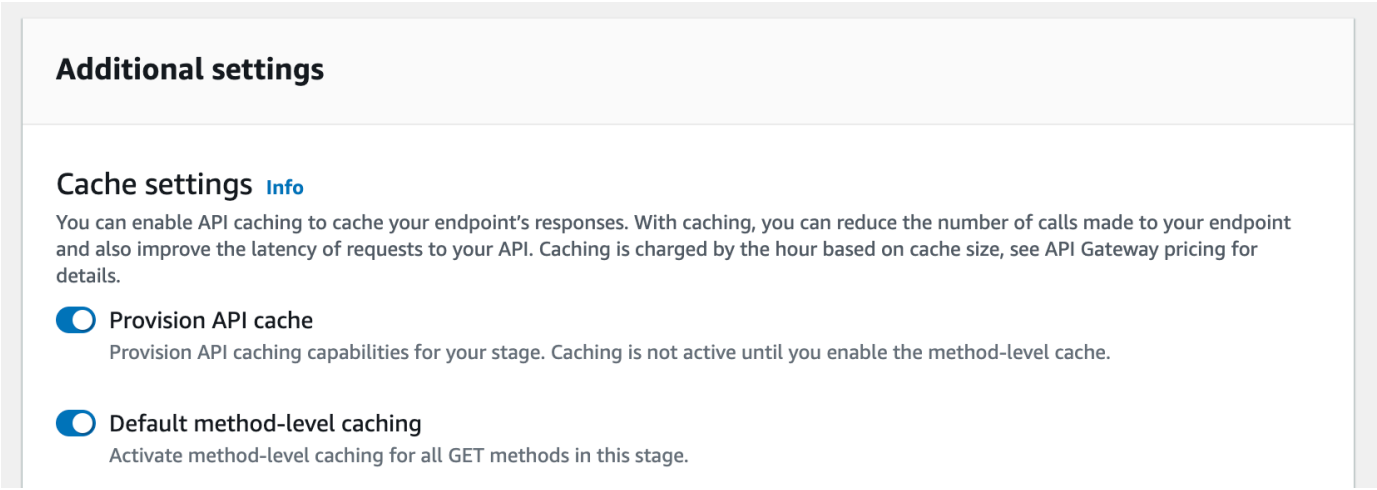
Pour configurer la mise en cache des API pour une étape donnée :

1. Connectez-vous à la console API Gateway à l'adresse : https://console.aws.amazon.com/apigateway.
2. Choisissez Stages (Étapes).
3. Dans la liste Stages (Étapes) de l'API, choisissez l'étape.
4. Dans la section Détails de l'étape, choisissez Modifier.
5. Sous Paramètres supplémentaires, pour les paramètres du cache, activez le cache de l'API Provisionner.

   Cela fournit un cluster de cache pour votre stage.
6. Pour activer la mise en cache pour votre étape, activez la mise en cache au niveau de la méthode par défaut.

   Cela active la mise en cache au niveau de la méthode pour toutes les `GET` méthodes de votre scène. Toutes `GET` les méthodes supplémentaires que vous déployez à ce stade disposeront d'un cache au niveau de la méthode.

> **Note**
>
> Si vous avez un paramètre existant pour un cache au niveau de la méthode, la modification du paramètre de mise en cache au niveau de la méthode par défaut n'affecte pas ce paramètre existant.



7. Sélectionnez Enregistrer les modifications.

> **Note**
>
> Le processus de création ou de suppression d'un cache prend environ 4 minutes dans API Gateway.
> Lorsqu'un cache est créé, la valeur du cluster de cache passe de `Create in progress` à`Active`. Lorsque la suppression du cache est terminée, la valeur du cluster de cache passe de `Delete in progress` à`Inactive`.
> Lorsque vous activez la mise en cache au niveau de la méthode pour toutes les méthodes de votre scène, la valeur de mise en cache au niveau de la méthode par défaut devient. `Active` Si vous désactivez la mise en cache au niveau de la méthode pour toutes les méthodes de votre scène, la valeur de mise en cache au niveau de la méthode par défaut devient. `Inactive` Si vous avez un paramètre existant pour un cache au niveau de la méthode, la modification de l'état du cache n'affecte pas ce paramètre.

Lorsque vous activez la mise en cache dans la section Paramètres de cache d'une étape, seules les méthodes GET sont mises en cache. Pour assurer la sécurité et la disponibilité de votre API, nous vous recommandons de ne pas modifier ce paramètre. Vous pouvez cependant activer la mise en cache pour d'autres méthodes en remplaçant les paramètres de méthode.

Si vous souhaitez vérifier si la mise en cache fonctionne comme prévu, vous avez deux options générales :

- Inspectez les CloudWatch métriques de CacheHitCountet CacheMissCountpour votre API et votre stage.
- Placez un horodatage dans la réponse.

> **Note**
>
> Vous ne devez pas utiliser l'`X-Cache`en-tête de la CloudFront réponse pour déterminer si votre API est servie à partir de votre instance de cache d'API Gateway.

Remplacer la mise en cache au niveau de l'étape par API Gateway par la mise en cache au niveau de la méthode

Vous pouvez remplacer les paramètres de cache au niveau de l'étape en activant ou en désactivant la mise en cache pour une méthode spécifique. Vous pouvez également modifier la période TTL ou activer ou désactiver le chiffrement pour les réponses mises en cache.

Si vous modifiez le paramètre de mise en cache par défaut au niveau de la méthode dans les détails de l'étape, cela n'affecte pas les paramètres de cache au niveau de la méthode qui ont des remplacements.

Si vous prévoyez qu'une méthode que vous mettez en cache va recevoir des données sensibles dans ses réponses, dans Cache Settings (Paramètres de cache), sélectionnez Encrypt cache data (Chiffrer les données de cache).

Pour configurer la mise en cache des API pour les méthodes individuelles à l'aide de la console :

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Choisissez l'API.

3. Choisissez Stages (Étapes).
4. Dans la liste Stages (Étapes) de l'API, développez l'étape et choisissez une méthode dans l'API.
5. Dans la section Dérogations de méthode, choisissez Modifier.
6. Dans la section Paramètres de méthode, activez ou désactivez Activer le cache de la méthode ou personnalisez les autres options souhaitées.

   > **Note**
   >
   > La mise en cache n'est pas active tant que vous n'avez pas provisionné un cluster de cache pour votre stage.

7. Choisissez Enregistrer.

Utilisation de paramètres de méthode ou d'intégration en tant que clés de cache pour indexer les réponses mises en cache

Lorsqu'une méthode ou une intégration mise en cache comporte des paramètres, qui peuvent prendre la forme d'en-têtes personnalisés, de chemins d'URL ou de chaînes de requête, vous pouvez utiliser certains ou la totalité de ces paramètres pour former des clés de cache. API Gateway peut mettre en cache les réponses de la méthode, en fonction des valeurs de paramètre utilisées.

> **Note**
>
> Les clés de cache sont obligatoires lorsque vous configurez la mise en cache d'une ressource.

Par exemple, supposons que vous avez une demande au format suivant :

```
GET /users?type=... HTTP/1.1
host: example.com
...
```

Dans cette demande, `type` peut prendre la valeur `admin` ou `regular`. Si vous incluez le paramètre `type` dans la clé de cache, les réponses de `GET /users?type=admin` sont mises en cache séparément de celles de `GET /users?type=regular`.

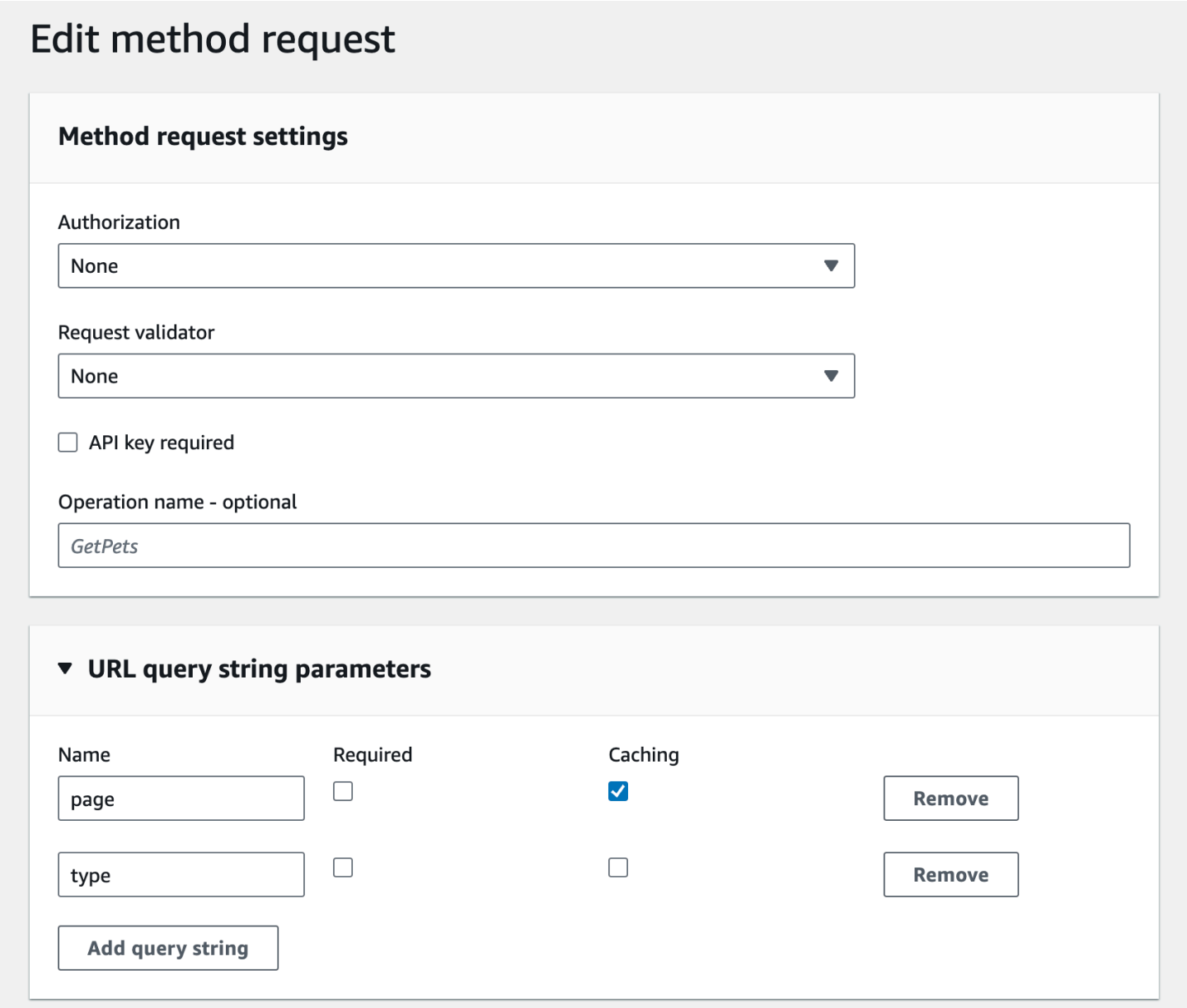
Lorsqu'une demande de méthode ou d'intégration utilise plus d'un paramètre, vous pouvez choisir d'inclure certains ou la totalité des paramètres pour créer la clé de cache. Par exemple, vous pouvez inclure uniquement le paramètre `type` dans la clé de cache pour la demande suivante, effectuée dans l'ordre indiqué pendant une période TTL :

```
GET /users?type=admin&department=A HTTP/1.1
host: example.com
...
```

La réponse de cette demande est mise en cache et elle est utilisée pour servir la demande suivante :

```
GET /users?type=admin&department=B HTTP/1.1
host: example.com
...
```

Pour inclure un paramètre de demande de méthode ou d'intégration dans une clé de cache dans la console API Gateway, sélectionnez Caching (Mise en cache) après avoir ajouté le paramètre.



Vidage du cache d'étape d'API dans API Gateway

Lorsque la mise en cache des API est activée, vous pouvez vider le cache au niveau d'une étape d'API afin de garantir que les clients de l'API obtiennent les réponses les plus récentes de vos points de terminaison d'intégration.

Pour vider le cache d'étape de l'API, choisissez le menu Actions de l'étape, puis sélectionnez Vider le cache de l'étape.

> **Note**
>
> Une fois que le cache est vidé, les réponses sont traitées à partir du point de terminaison d'intégration jusqu'à ce que le cache soit recréé. Durant cette période, le nombre de demandes envoyées au point de terminaison d'intégration peut augmenter. Cela peut temporairement augmenter la latence globale de votre API.

Invalidation d'une entrée de cache API Gateway

Un client de votre API peut invalider une entrée de cache existante et la recharger à partir du point de terminaison d'intégration pour les différentes demandes. Le client doit envoyer une demande contenant l'en-tête `Cache-Control: max-age=0`. Le client reçoit la réponse directement du point de terminaison d'intégration et non du cache, sous réserve qu'il dispose de l'autorisation nécessaire. L'entrée de cache existante est alors remplacée par la nouvelle réponse, qui est extraite du point de terminaison d'intégration.

Pour accorder cette autorisation à un client, attachez une stratégie au format suivant à un rôle d'exécution IAM pour l'utilisateur.

> **Note**
>
> L'invalidation du cache entre comptes n'est pas prise en charge.

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

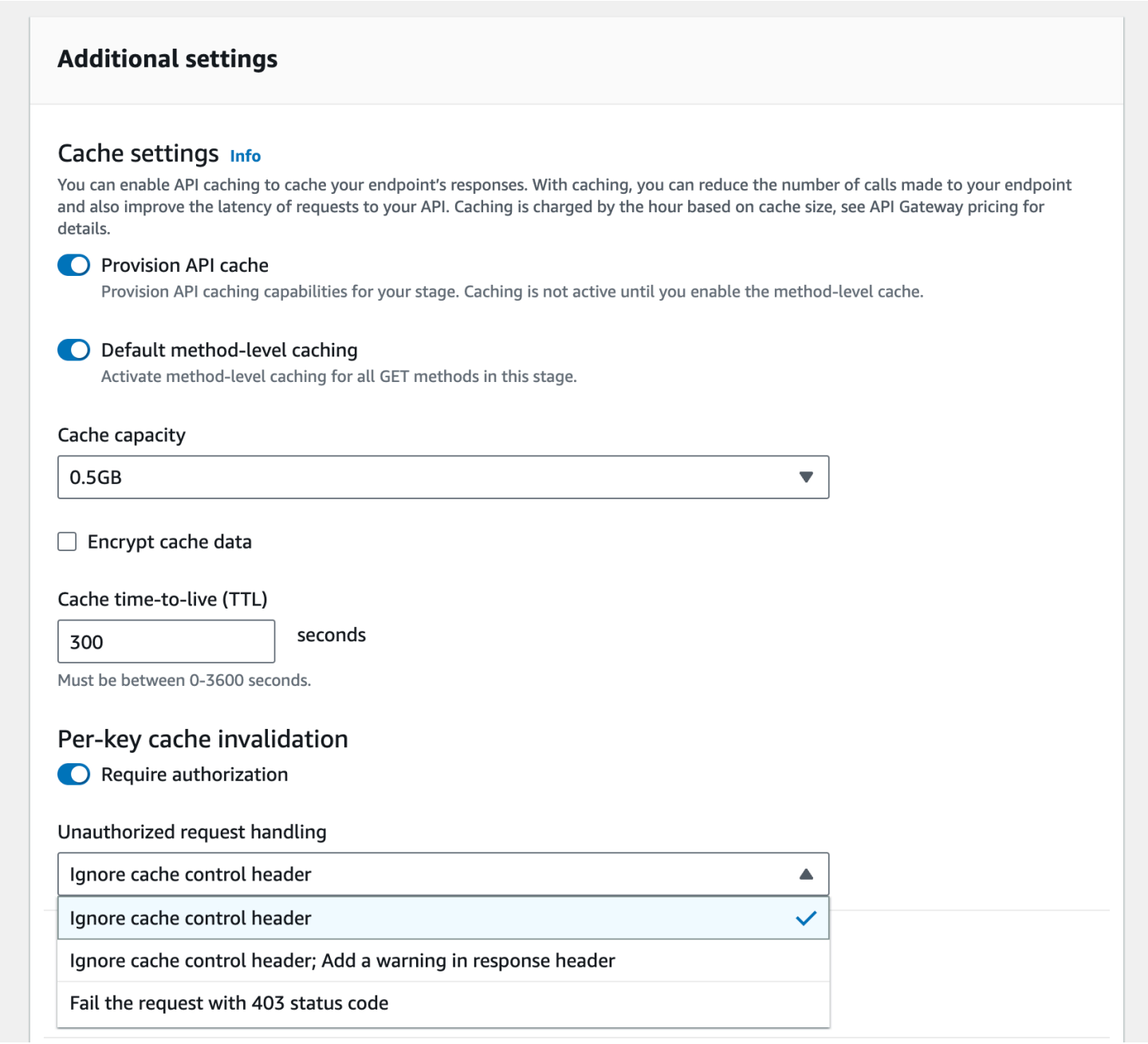
```
}
```

Cette stratégie autorise le service d'exécution API Gateway à invalider le cache pour les demandes sur les ressources spécifiées. Pour spécifier un groupe de ressources ciblées, utilisez un caractère générique (*) pour `account-id`, `api-id` et les autres entrées de la valeur d'ARN de `Resource`. Pour de plus amples informations sur la définition d'autorisations pour le service d'exécution API Gateway, veuillez consulter Contrôle de l'accès à une API avec des autorisations IAM.

Si vous n'imposez aucune stratégie `InvalidateCache` (ou si vous activez la case à cocher Require authorization (Exiger une autorisation) dans la console), n'importe quel client peut invalider le cache API. Si tous les clients ou la plupart d'entre eux invalident le cache API, cela peut augmenter la latence de votre API de façon significative.

Lorsque la stratégie est en place, la mise en cache est activée et l'autorisation est requise.

Vous pouvez contrôler la manière dont les demandes non autorisées sont traitées en choisissant une option dans Gestion des demandes non autorisées dans la console API Gateway.



Les trois options disponibles se traduisent par les comportements suivants :

- Fail the request with 403 status code (Échec de la demande avec le code de statut 403) : renvoie une réponse 403 Accès non autorisé.

  Pour définir cette option à l'aide de l'API, utilisez `FAIL_WITH_403`.
- Ignore cache control header ; Add a warning in response header (Ignorer l'en-tête de contrôle du cache ; ajouter un avertissement dans l'en-tête de réponse) : traite la demande et ajoute un en-tête d'avertissement dans la réponse.

Pour définir cette option à l'aide de l'API, utilisez `SUCCEED_WITH_RESPONSE_HEADER`.

- Ignore cache control header (Ignorer l'en-tête de contrôle du cache) : traite la demande sans ajouter aucun en-tête d'avertissement dans la réponse.

  Pour définir cette option à l'aide de l'API, utilisez `SUCCEED_WITHOUT_RESPONSE_HEADER`.

Activation de la compression de la charge utile pour une API

API Gateway permet au client d'appeler votre API avec des charges utiles compressées à l'aide de l'un des codages de contenu pris en charge. Par défaut, API Gateway prend en charge la décompression de la charge utile de la demande de méthode. Cependant, vous devez configurer votre API pour activer la compression de la charge utile de la réponse de la méthode.

Pour activer la compression sur une `API`, définissez la propriété `minimumCompressionsSize` sur un nombre entier positif compris entre 0 et 10485760 (10 Mo) lorsque vous créez l'API ou une fois que vous l'avez créée. Pour désactiver la compression sur l'API, définissez l'attribut `minimumCompressionSize` sur null ou supprimez-le complètement. Vous pouvez activer ou désactiver la compression pour une API à l'aide de la console API Gateway, de l' AWS CLI API REST API Gateway ou de l'API REST API Gateway.

Si vous souhaitez que la compression soit appliquée sur une charge utile de n'importe quelle taille, définissez la valeur `minimumCompressionSize` sur zéro. Toutefois, la compression des données de petite taille peut augmenter la taille des données finales. En outre, la compression dans API Gateway et la décompression dans le client peuvent accroître la latence globale et nécessitent plus de temps de calcul. Vous devez exécuter des tests par rapport à vos API afin de déterminer une valeur optimale.

Le client peut envoyer une demande d'API avec une charge utile compressée et un en-tête `Content-Encoding` approprié pour qu'API Gateway décompresse et applique les modèles de mappage appropriés avant de transmettre la demande au point de terminaison d'intégration. Une fois que la compression est activée et que l'API est déployée, le client peut recevoir une réponse d'API avec une charge utile compressée s'il spécifie un en-tête `Accept-Encoding` approprié dans la demande de méthode.

Lorsque le point de terminaison d'intégration attend et renvoie des charges utiles JSON non compressées, les modèles de mappage qui sont configurés pour une charge utile JSON non compressée sont applicables à la charge utile compressée. Pour la charge utile d'une demande de méthode compressée, API Gateway décompresse la charge utile, applique le modèle de mappage

et transmet la demande mappée au point de terminaison d'intégration. Pour la charge utile d'une réponse d'intégration non compressée, API Gateway applique le modèle de mappage, compresse la charge utile mappée et renvoie la charge utile compressée au client.

Rubriques

- Activation de la compression de la charge utile pour une API
- Appel d'une méthode d'API avec une charge utile compressée
- Réception d'une réponse d'API avec une charge utile compressée

Activation de la compression de la charge utile pour une API

Vous pouvez activer la compression pour une API à l'aide de la console API Gateway AWS CLI, du ou d'un AWS SDK.

Pour une API existante, vous devez déployer l'API après avoir activé la compression pour que la modification entre en vigueur. Pour une nouvelle API, vous pouvez déployer l'API une fois sa configuration terminée.

> **Note**
>
> La priorité d'encodage de contenu la plus élevée doit être prise en charge par API Gateway. Si ce n'est pas le cas, la compression n'est pas appliquée à la charge utile de la réponse.

Rubriques

- Activation de la compression de charge utile pour une API à l'aide de la console API Gateway
- Activez la compression de charge utile pour une API à l'aide de AWS CLI
- Codages de contenu pris en charge par API Gateway

Activation de la compression de charge utile pour une API à l'aide de la console API Gateway

La procédure suivante décrit comment activer la compression de la charge utile pour une API.

Pour activer la compression de la charge utile à l'aide de la console API Gateway

1. Connectez-vous à la console API Gateway à l'adresse : https://console.aws.amazon.com/apigateway.

2. Choisissez une API existante ou créez-en une.
3. Dans le panneau de navigation principal, choisissez Paramètres de l'API.
4. Dans la section Détails de l'API, choisissez Modifier.
5. Activez Encodage de contenu pour activer la compression de la charge utile. Pour Taille minimale du corps, entrez un nombre pour la taille de compression minimale (en octets). Pour désactiver la compression, désactivez l'option Encodage de contenu.
6. Sélectionnez Enregistrer les modifications.

Activez la compression de charge utile pour une API à l'aide de AWS CLI

Pour utiliser le AWS CLI pour créer une nouvelle API et activer la compression, appelez la `create-rest-api`commande comme suit :

```
aws apigateway create-rest-api \
    --name "My test API" \
    --minimum-compression-size 0
```

AWS CLI Pour activer la compression sur une API existante, appelez la `update-rest-api`commande comme suit :

```
aws apigateway update-rest-api \
    --rest-api-id 1234567890 \
    --patch-operations op=replace,path=/minimumCompressionSize,value=0
```

La propriété `minimumCompressionSize` possède un nombre entier non négatif compris entre 0 et 10485760 (10 Mo). Elle mesure le seuil de compression. Si la taille de charge est inférieure à cette valeur, la compression ou décompression ne sont pas appliquées sur la charge utile. La définition de l'option sur zéro permet la compression de n'importe quelle taille de charge utile.

AWS CLI Pour désactiver la compression, appelez la `update-rest-api`commande comme suit :

```
aws apigateway update-rest-api \
    --rest-api-id 1234567890 \
    --patch-operations op=replace,path=/minimumCompressionSize,value=
```

Vous pouvez également définir `value` sur une chaîne vide `""` ou omettez la propriété `value` complètement dans l'appel précédent.

Codages de contenu pris en charge par API Gateway

API Gateway prend en charge les codages de contenu suivants :

- `deflate`
- `gzip`
- `identity`

API Gateway prend également en charge le format d'en-tête `Accept-Encoding` suivant, conformément à la spécification RFC 7231 :

- `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`

Appel d'une méthode d'API avec une charge utile compressée

Pour effectuer une demande d'API avec une charge utile compressée, le client doit définir l'en-tête `Content-Encoding` avec l'un des codages de contenu pris en charge.

Supposons que vous soyez un client d'API et que vous souhaitiez appeler la méthode PetStore API (`POST /pets`). N'appelez pas cette méthode avec la sortie JSON suivante :

```
POST /pets
Host: {petstore-api-id}.execute-api.{region}.amazonaws.com
Content-Length: ...

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

Au lieu de cela, vous pouvez appeler la méthode avec la même charge utile compressée en utilisant le codage GZIP :

```
POST /pets
```

```
Host: {petstore-api-id}.execute-api.{region}.amazonaws.com
Content-Encoding:gzip
Content-Length: ...

[binary gzip data]
```

Lorsqu'API Gateway reçoit la demande, il vérifie si le codage de contenu spécifié est pris en charge. Ensuite, il tente de décompresser la charge utile avec le codage de contenu spécifié. Si la décompression est réussie, il envoie la demande au point de terminaison d'intégration. Si le codage spécifié n'est pas pris en charge ou si la charge utile fournie n'est pas compressée avec le codage spécifié, API Gateway renvoie la réponse d'erreur `415 Unsupported Media Type`. L'erreur n'est pas enregistrée dans CloudWatch Logs si elle survient au début de la phase de décompression avant que votre API et votre stage ne soient identifiés.

Réception d'une réponse d'API avec une charge utile compressée

Lorsque vous effectuez une demande pour une API dont la compression est activée, le client peut choisir de recevoir une charge utile de réponse compressée dans un format spécifique en spécifiant un en-tête `Accept-Encoding` avec un codage de contenu pris en charge.

API Gateway ne compresse la charge utile de la réponse que lorsque les conditions suivantes sont satisfaites :

- La demande entrante a l'en-tête `Accept-Encoding` avec un format et un codage de contenu pris en charge.

  > **Note**
  >
  > Si l'en-tête n'est pas défini, la valeur par défaut est * telle que définie dans RFC 7231. Dans ce cas, API Gateway ne compresse pas la charge utile. Certains navigateurs ou clients peuvent ajouter `Accept-Encoding` (par exemple, `Accept-Encoding:gzip, deflate, br`) automatiquement aux requêtes dont la compression est activée. Cela peut déclencher la compression de la charge utile dans API Gateway. Sans une spécification explicite des valeurs d'en-tête `Accept-Encoding` prises en charge, API Gateway ne compresse pas la charge utile.

- La valeur `minimumCompressionSize` est définie sur l'API pour activer la compression.
- La réponse d'intégration ne dispose pas d'en-tête `Content-Encoding`.

- La taille de la charge utile d'une réponse d'intégration corps après l'application du modèle de mappage approprié est supérieure ou égale à la valeur `minimumCompressionSize` spécifiée.

API Gateway applique n'importe quel modèle de mappage configuré pour la réponse d'intégration avant de compresser la charge utile. Si la réponse d'intégration contient un en-tête `Content-Encoding`, API Gateway suppose que la charge utile de la réponse d'intégration est déjà compressée et omet le traitement de compression.

L'exemple d' PetStore API et la requête suivante en sont un exemple :

```
GET /pets
Host: {petstore-api-id}.execute-api.{region}.amazonaws.com
Accept: application/json
```

Le back-end répond à la demande avec une charge utile JSON non compressée qui est semblable à ce qui suit :

```
200 OK

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

Pour recevoir cette sortie en tant que charge utile compressée, votre client d'API peut soumettre une demande comme suit :

```
GET /pets
```

```
Host: {petstore-api-id}.execute-api.{region}.amazonaws.com
Accept-Encoding:gzip
```

Le client reçoit la réponse avec un en-tête `Content-Encoding` et une charge utile encodée GZIP similaires à ce qui suit :

```
200 OK
Content-Encoding:gzip
...

���RP�

J�)JV
�:P^IeA*������+(�L ��X�YZ�ku0L0B7!9��C#�&����Y��a���^�X
```

Lorsque la charge utile de la réponse est compressée, seule la taille des données compressées est facturée pour le transfert de données.

Distribution de votre API REST aux clients

Cette section fournit des détails sur la distribution de vos API API Gateway à vos clients. Pour distribuer votre API, vous devez notamment générer des kits SDK pour permettre à vos clients de la télécharger et de l'intégrer à leurs applications clientes, documenter votre API de sorte que les clients sachent comment l'appeler à partir de leurs applications clientes, et mettre à disposition votre API dans le cadre des offres de produits.

Rubriques

- Création et utilisation de plans d'utilisation avec des clés d'API
- Documentation sur les API REST
- Génération d'un kit SDK pour une API REST dans API Gateway
- Vendez vos API API Gateway via AWS Marketplace

Création et utilisation de plans d'utilisation avec des clés d'API

Après avoir créé, testé et déployé vos API, vous pouvez proposer les plans d'utilisation API Gateway en tant qu'offres de produits à vos clients. Vous pouvez configurer des plans d'utilisation et clés d'API

pour permettre aux clients d'accéder à des API sélectionnées et commencer à limiter les requêtes vers ces API en fonction de limites et de quotas définis. Ils peuvent être définis au niveau de l'API ou de la méthode API.

Que sont les plans d'utilisation et les clés d'API ?

Un plan d'utilisation spécifie qui peut accéder à une ou plusieurs méthodes et étapes d'API déployées et définit le taux de requête cible pour commencer à limiter les requêtes. Le plan utilise des clés d'API afin d'identifier les clients et mesure les accès aux étapes d'API associées pour chaque clé.

Les clés d'API sont des valeurs de chaînes alphanumériques que vous distribuez aux clients des développeurs d'applications pour leur accorder l'accès à l'API. Vous pouvez utiliser des clés d'API avec des mécanismes d'autorisation Lambda, des Rôles IAM ou Amazon Cognito pour contrôler l'accès à vos API. API Gateway peut générer des clés d'API pour vous ou vous pouvez les importer à partir d'un fichier CSV. Vous pouvez générer une clé d'API dans API Gateway ou l'importer dans API Gateway à partir d'une source externe. Pour de plus amples informations, veuillez consulter the section called “Configuration des clés d'API à l'aide de la console API Gateway”.

Chaque clé d'API a un nom et une valeur. (Les termes « clé d'API » et « valeur de clé d'API » sont souvent utilisés de manière interchangeable.) Le nom ne peut pas dépasser 1 024 caractères. La valeur est une chaîne alphanumérique comportant entre 20 et 128 caractères, par exemple, `apikey1234abcdefghij0123456789`.

> **Important**
>
> Les valeurs de clés d'API doivent être uniques. Si vous tentez de créer deux clés d'API nommées différemment mais avec la même valeur, API Gateway les considère comme une seule et même clé d'API.
>
> Une clé d'API peut être associée à plusieurs plans d'utilisation. Un plan d'utilisation peut être associé à plusieurs étapes. Toutefois, une clé d'API donnée ne peut être associée qu'à un seul plan d'utilisation pour chaque étape de votre API.

Une limitation définit le point cible auquel la limitation des requêtes doit commencer. Ils peuvent être définis au niveau de l'API ou de la méthode d’API.

Une limite de quota détermine le nombre maximal de requêtes autorisées pour une clé d'API donnée sur un intervalle de temps spécifié. Vous pouvez configurer les méthodes d'API pour exiger une autorisation de clé d’API selon la configuration du plan d'utilisation.

Les limites de quota et de restriction s'appliquent aux diverses demandes de clés d'API qui sont regroupées pour toutes les étapes d'API au sein d'un plan d'utilisation.

> **Note**
>
> La limitation des plans d'utilisation et les quotas ne sont pas des limites strictes et sont appliquées au mieux. Dans certains cas, les clients peuvent dépasser les quotas que vous avez définis. Ne comptez pas sur les quotas de plan d'utilisation ni sur la limitation pour contrôler les coûts ou bloquer l'accès à une API. Pensez à utiliser AWS Budgets pour contrôler les coûts et AWS WAF pour gérer les demandes d'API.

Bonnes pratiques concernant les clés d'API et les plans d'utilisation

Voici quelques suggestions de bonnes pratiques à suivre lors de l'utilisation de clés d'API et de plans d'utilisation.

> **Important**
>
> - N'utilisez pas de clés d'API pour l'authentification ou l'autorisation pour contrôler l'accès à vos API. En effet, si vous utilisez plusieurs API dans un plan d'utilisation, un utilisateur doté d'une clé valide pour une API dans ce plan d'utilisation peut accéder à toutes les API dudit plan d'utilisation. Pour contrôler l'accès à votre API, utilisez plutôt un rôle IAM, un Mécanisme d'autorisation Lambda ou un Groupe d'utilisateurs Amazon Cognito.
> - Utilisez les clés d'API générées par API Gateway. Les clés d'API ne doivent pas inclure d'informations confidentielles ; les clients les transmettent généralement dans des en-têtes qui peuvent être enregistrés.

- Si vous utilisez un portail de développement pour publier vos API, notez que les clients peuvent souscrire à toutes les API d'un plan d'utilisation donné, même si vous ne les avez pas mises à la disposition de vos clients.
- Dans certains cas, les clients peuvent dépasser les quotas que vous avez définis. Ne vous fiez pas aux plans d'utilisation pour contrôler les coûts. Pensez à utiliser AWS Budgets pour contrôler les coûts et AWS WAF pour gérer les demandes d'API.
- Après avoir ajouté une clé d'API à un plan d'utilisation, l'opération de mise à jour peut prendre quelques minutes.

Procédure de configuration d'un plan d'utilisation

Les étapes suivantes vous expliquent comment vous, en tant que propriétaire de l'API, pouvez créer et configurer un plan d'utilisation pour vos clients.

Pour configurer un plan d'utilisation

1. Créez une ou plusieurs API, configurez les méthodes pour exiger une clé d'API, puis déployez les API aux différentes étapes.
2. Générez ou importez des clés d'API afin de les distribuer aux développeurs d'applications (vos clients) qui utiliseront votre API.
3. Créez le plan d'utilisation avec les limites de quota et de restriction souhaitées.
4. Associez les étapes d'API et les clés d'API au plan d'utilisation.

Les appelants de l'API doivent fournir la clé API qui leur a été attribuée dans l'en-tête `x-api-key` des demandes d'API.

> **Note**
>
> Pour pouvoir inclure des méthodes d'API dans un plan d'utilisation, vous devez configurer des méthodes d'API individuelles pour demander une clé d'API. Pour connaître les bonnes pratiques à prendre en compte, consultez the section called “Bonnes pratiques concernant les clés d'API et les plans d'utilisation”.

Choisir une source de clé d'API

Lorsque vous associez un plan d'utilisation à une API et activez des clés d'API sur les méthodes d'API, chaque demande entrante vers l'API doit inclure une clé d'API. API Gateway lit la clé et la compare à celles du plan d'utilisation. S'il y a correspondance, API Gateway restreint les demandes en fonction des limitations et quotas définis dans le plan. Sinon, il renvoie une exception `InvalidKeyParameter`. En conséquence, l'appelant reçoit une réponse `403 Forbidden`.

Votre API API Gateway peut recevoir des clés d'API à partir de l'une de ces deux sources :

`HEADER`

Vous distribuez les clés d'API à vos clients et leur demandez de les transmettre comme en-tête `X-API-Key` de chaque demande entrante.

`AUTHORIZER`

Un mécanisme d'autorisation Lambda renvoie la clé d'API dans le cadre de la réponse d'autorisation. Pour plus d'informations sur la réponse d'autorisation, consultez the section called “Résultat d'un autorisateur Lambda API Gateway”.

> **Note**
>
> Pour connaître les bonnes pratiques à prendre en compte, consultez the section called “Bonnes pratiques concernant les clés d'API et les plans d'utilisation”.

Pour choisir une source de clé d'API pour une API à l'aide de la console API Gateway

1. Connectez-vous à la console API Gateway.
2. Choisissez une API existante ou créez-en une.
3. Dans le panneau de navigation principal, choisissez Paramètres de l'API.
4. Dans la section Détails de l'API, choisissez Modifier.
5. Sous Source de clé d'API, sélectionnez `Header` ou `Authorizer` dans la liste déroulante.
6. Sélectionnez Enregistrer les modifications.

Pour choisir une source de clé d'API pour une API à l'aide de AWS CLI, appelez la `update-rest-api`commande comme suit :

```
aws apigateway update-rest-api --rest-api-id 1234123412 --patch-operations
 op=replace,path=/apiKeySource,value=AUTHORIZER
```

Pour que le client soumette une clé API, définissez la `value` sur HEADER dans la commande CLI précédente.

Pour choisir une source de clé d'API pour une API à l'aide de l'API REST API Gateway, appelez `restapi:update` comme suit :

```
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"
    }
  ]
}
```

Pour avoir qu'un mécanisme d'autorisation renvoie une clé API, définissez la `value` sur `AUTHORIZER` dans l'entrée `patchOperations` précédente.

En fonction du type de source de la clé d'API que vous choisissez, appliquez l'une des procédures suivantes pour utiliser une clé d'API issue de l'en-tête ou une clé d'API renvoyée par un mécanisme d'autorisation dans l'appel de méthode :

Pour utiliser une clé d'API issue d'un en-tête :

1. Créez une API avec les méthodes d'API souhaitées, puis déployez l'API dans une étape.
2. Créez un nouveau plan d'utilisation ou choisissez un plan d'utilisation existant. Ajoutez l'étape d'API déployée au plan d'utilisation. Attachez une clé API au plan d'utilisation ou choisissez une clé API existante dans le plan. Notez la valeur de clé API choisie.
3. Configurez des méthodes d'API pour exiger une clé API.
4. Redéployez l'API à la même étape. Si vous déployez l'API pour une nouvelle étape, assurez-vous de mettre à jour le plan d'utilisation pour attacher la nouvelle étape d'API.

Le client peut désormais appeler les méthodes d'API tout en fournissant l'en-tête `x-api-key` avec la clé API choisie comme valeur d'en-tête.

Pour utiliser une clé d'API issue d'un mécanisme d'autorisation :

1. Créez une API avec les méthodes d'API souhaitées, puis déployez l'API dans une étape.

2. Créez un nouveau plan d'utilisation ou choisissez un plan d'utilisation existant. Ajoutez l'étape d'API déployée au plan d'utilisation. Attachez une clé API au plan d'utilisation ou choisissez une clé API existante dans le plan. Notez la valeur de clé API choisie.
3. Créez un mécanisme d'autorisation Lambda par jeton. Incluez, `usageIdentifierKey:{api-key}` en tant que propriété de niveau racine de la réponse d'autorisation. Pour obtenir des instructions sur la création d'un système d'autorisation basé sur des jetons, consultez. the section called “Exemple de fonction TOKEN Lambda d'autorisation”
4. Configurez des méthodes d'API pour exiger une clé d'API et activer le mécanisme d'autorisation Lambda sur les méthodes.
5. Redéployez l'API à la même étape. Si vous déployez l'API pour une nouvelle étape, assurez-vous de mettre à jour le plan d'utilisation pour attacher la nouvelle étape d'API.

Le client peut désormais appeler les méthodes API key-required sans fournir explicitement une clé API. La clé API retournant le mécanisme d'autorisation est utilisée automatiquement.

Configuration des clés d'API à l'aide de la console API Gateway

Pour configurer les clés API, procédez comme suit :

- Configurez des méthodes API pour exiger une clé API.
- Créez ou importez une clé API pour l'API dans une région.

Avant de configurer des clés API, vous devez avoir créé une API et l'avoir déployée jusqu'à une étape. Une fois que vous avez créé une valeur de clé d'API, elle ne peut pas être modifiée.

Pour obtenir les instructions de création et de déploiement d'une API à l'aide de la console API Gateway, veuillez consulter Développement d'une API REST dans API Gateway et Déploiement d'une API REST dans Amazon API Gateway, respectivement.

Après avoir créé une clé d'API, vous devez l'associer à un plan d'utilisation. Pour plus d'informations, consultez Création, configuration et test des plans d'utilisation avec la console API Gateway.

> **Note**
>
> Pour connaître les bonnes pratiques à prendre en compte, consultez the section called “Bonnes pratiques concernant les clés d'API et les plans d'utilisation”.

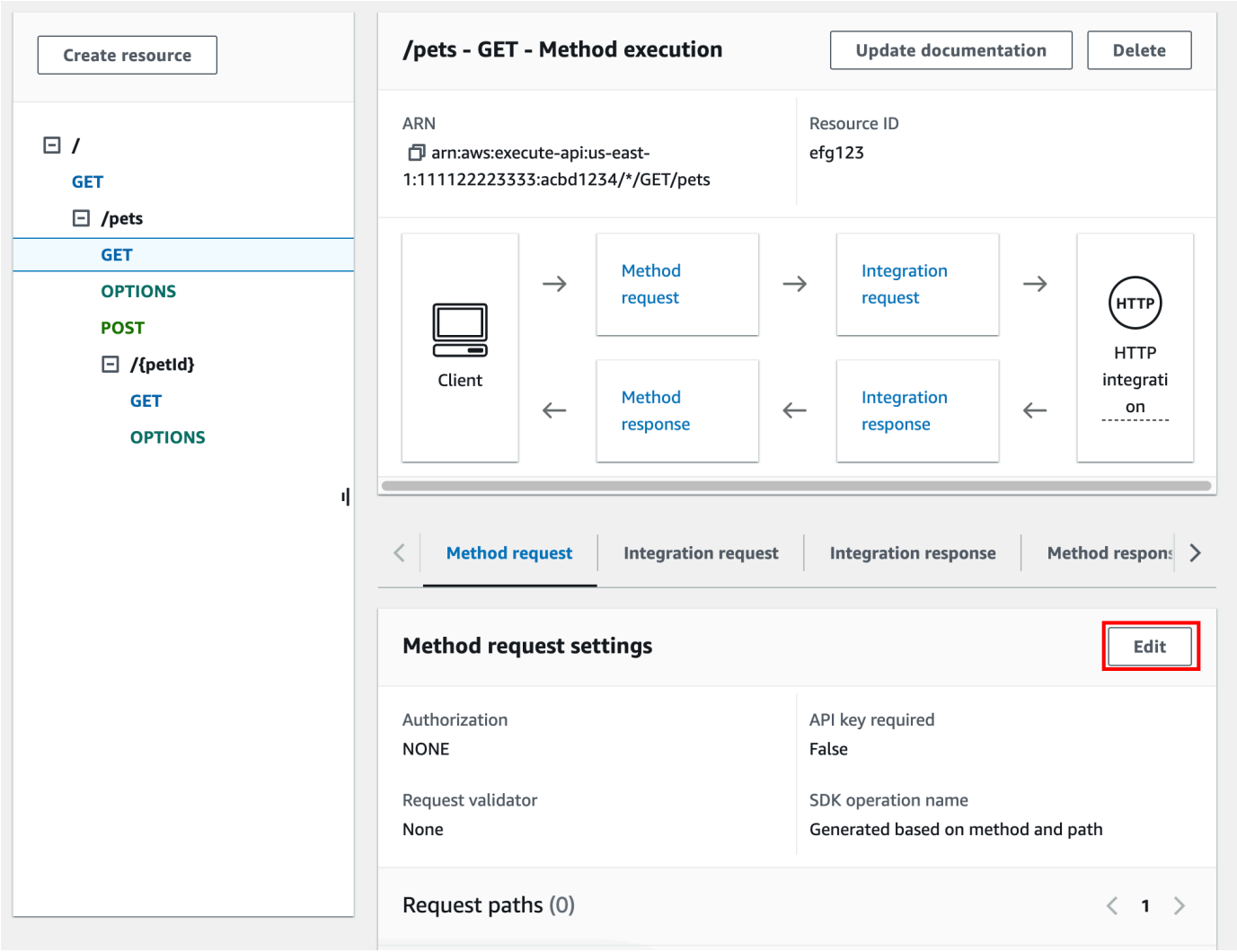
Rubriques

Exigence d'une clé API sur une méthode

La procédure suivante explique comment configurer une méthode d'API pour exiger une clé API.

Pour configurer une méthode d'API pour exiger une clé API

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Choisissez une API REST.
3. Dans le panneau de navigation principal d'API Gateway, choisissez Resources (Ressources).
4. Sous Resources, créez une méthode ou choisissez une méthode existante.
5. Dans l'onglet Demande de méthode, sous Paramètres de demande de méthode, choisissez Modifier.



6. Sélectionnez Clé API obligatoire.
7. Choisissez Enregistrer.
8. Déployez ou redéployez l'API pour que l'exigence prenne effet.

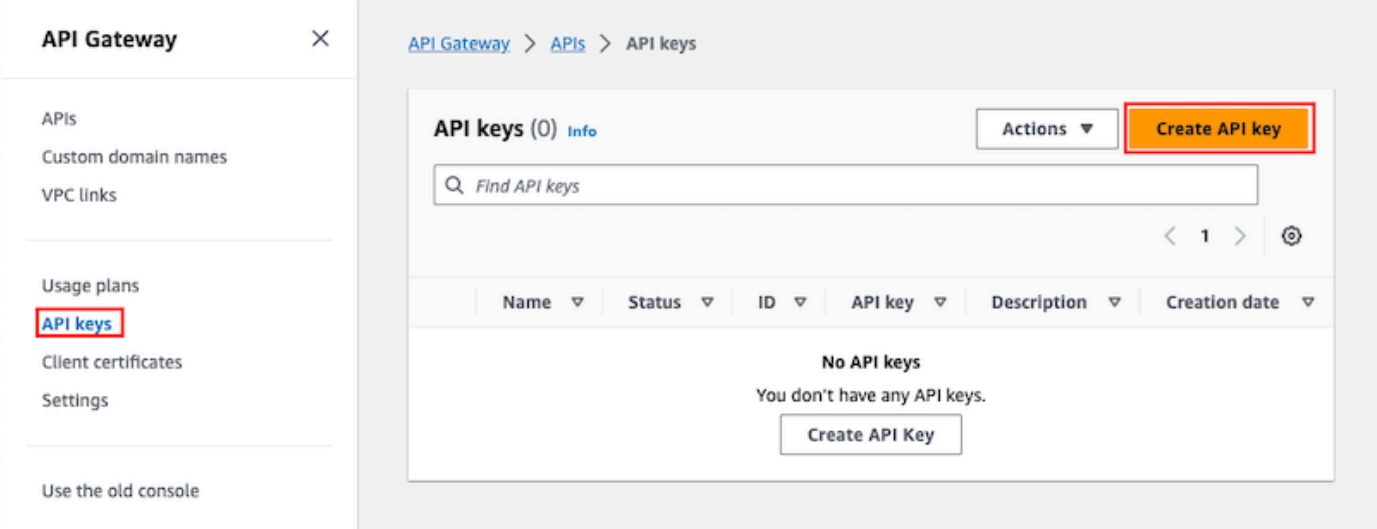
Si l'option Clé API obligatoire est définie sur `false` et que vous n'exécutez pas les étapes précédentes, aucune clé d'API associée à une étape d'API n'est utilisée pour la méthode.

Créez une clé API :

Si vous avez déjà créé ou importé des clés API à utiliser avec les plans d'utilisation, vous pouvez ignorer cette procédure et la suivante.

Pour créer une clé API

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Choisissez une API REST.
3. Dans le panneau de navigation principal d'API Gateway, choisissez Clés d'API.
4. Choisissez Créer une clé d'API.



5. Pour Nom, entrez un nom.
6. (Facultatif) Sous Description, entrez une description.
7. Pour Clé d'API, choisissez Autogénérer pour qu'API Gateway génère la valeur de la clé, ou choisissez Personnalisé pour créer votre propre valeur de clé.
8. Choisissez Enregistrer.

Importation de clés API

La procédure suivante explique comment importer des clés API à utiliser avec les plans d'utilisation.

Pour importer des clés API

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Choisissez une API REST.
3. Dans le panneau de navigation principal, choisissez Clés d'API.
4. Choisissez le menu déroulant Actions, puis choisissez Importer des clés API.

5. Pour charger un fichier de clés séparées par des virgules, choisissez Choisissez un fichier. Vous pouvez également entrer les clés dans l'éditeur de texte. Pour plus d'informations sur le format de fichier, consultez the section called “Format de fichier de clé d'API API Gateway”.
6. Choisissez Échouer avec les avertissements pour arrêter l'importation en cas d'erreur, ou Ignorer les avertissements pour continuer à importer les entrées de clé valides en cas d'avertissement.
7. Choisissez Importer pour importer vos clés d'API.

Création, configuration et test des plans d'utilisation avec la console API Gateway

Avant de créer un plan d'utilisation, assurez-vous d'avoir configuré les clés API souhaitées. Pour de plus amples informations, veuillez consulter Configuration des clés d'API à l'aide de la console API Gateway.

Cette section décrit comment créer et utiliser un plan d'utilisation à l'aide de la console API Gateway.

Rubriques

- Migration de votre API vers les plans d'utilisation par défaut (si nécessaire)
- Création d'un plan d'utilisation
- Test d'un plan d'utilisation
- Gérer un plan d'utilisation

Migration de votre API vers les plans d'utilisation par défaut (si nécessaire)

Si vous avez commencé à utiliser API Gateway après le déploiement de la fonction des plans d'utilisation le 11 août 2016, les plans d'utilisation seront automatiquement activés dans toutes les régions prises en charge.

Si vous avez commencé à utiliser API Gateway avant cette date, vous devrez peut-être migrer vers les plans d'utilisation par défaut. Vous serez invité à définir l'option Enable Usage Plans (Activer les plans d'utilisation) avant d'utiliser des plans d'utilisation pour la première fois dans la région sélectionnée. Lorsque vous activez cette option, des plans d'utilisation par défaut sont créés pour chaque étape d'API unique associée aux clés API existantes. Dans le plan d'utilisation par défaut, aucune limitation ou limite de quota n'est définie initialement, et les associations entre les clés d'API et les étapes d'API sont copiées dans les plans d'utilisation. L'API se comporte de la même manière qu'auparavant. Toutefois, vous devez utiliser la `UsagePlan`apiStagespropriété pour associer les valeurs d'étape d'API spécifiées (`apiIdetstage`) aux clés d'API incluses (via `UsagePlanKey`), au lieu d'utiliser la ApiKey`stageKeys`propriété.

Pour vérifier si vous avez déjà migré vers les plans d'utilisation par défaut, utilisez la commande CLI `get-account`. Dans la sortie de la commande, la liste `features` inclut une entrée de `"UsagePlans"` lorsque les plans d'utilisation sont activés.

Vous pouvez également migrer vos API vers des plans d'utilisation par défaut en utilisant AWS CLI ce qui suit :

Pour passer aux plans d'utilisation par défaut à l'aide du AWS CLI

1. Appelez cette commande CLI : `update-account`.
2. Pour le paramètre `cli-input-json`, utilisez le code JSON suivant :

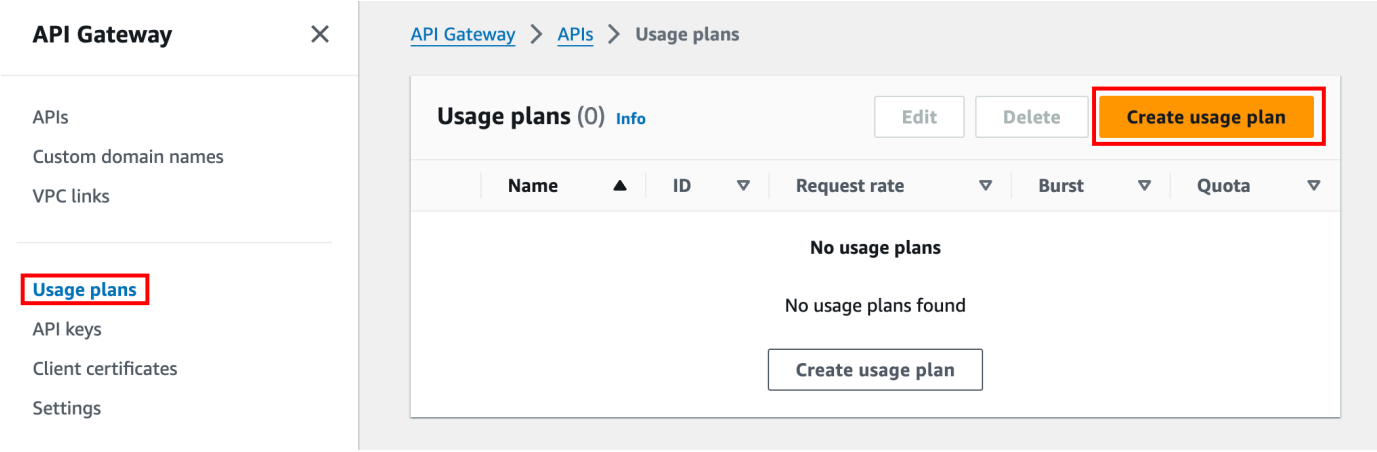
```
[
    {
        "op": "add",
        "path": "/features",
        "value": "UsagePlans"
    }
]
```

Création d'un plan d'utilisation

La procédure suivante explique comment créer un plan d'utilisation.

Pour créer un plan d'utilisation

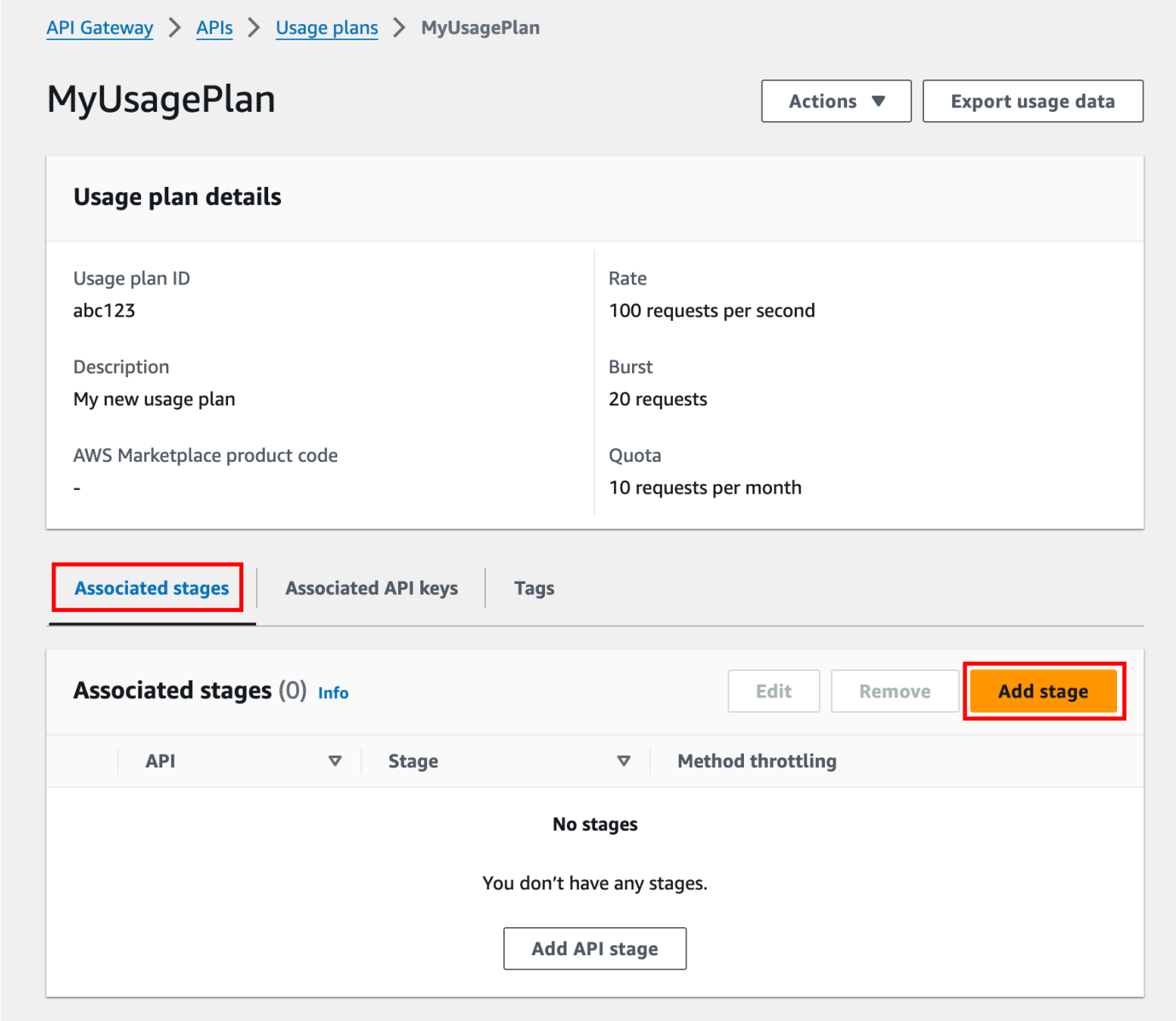
1. Connectez-vous à la console API Gateway à l'adresse : https://console.aws.amazon.com/apigateway.
2. Dans le panneau de navigation principal d'Amazon API Gateway, choisissez Plans d'utilisation, puis Créer un plan d’utilisation.



3. Pour Nom, entrez un nom.
4. (Facultatif) Sous Description, entrez une description.
5. Par défaut, les plans d'utilisation permettent une limitation. Entrez un Taux et un Débit pour votre plan d'utilisation. Choisissez Limitation pour désactiver la limitation.
6. Par défaut, les plans d'utilisation activent un quota pour une période donnée. Pour Demandes, entrez le nombre total de demandes qu'un utilisateur peut effectuer au cours de la période de votre plan d'utilisation. Choisissez Quota pour désactiver le quota.
7. Choisissez Créer un plan d'utilisation.

Pour ajouter une étape au plan d'utilisation

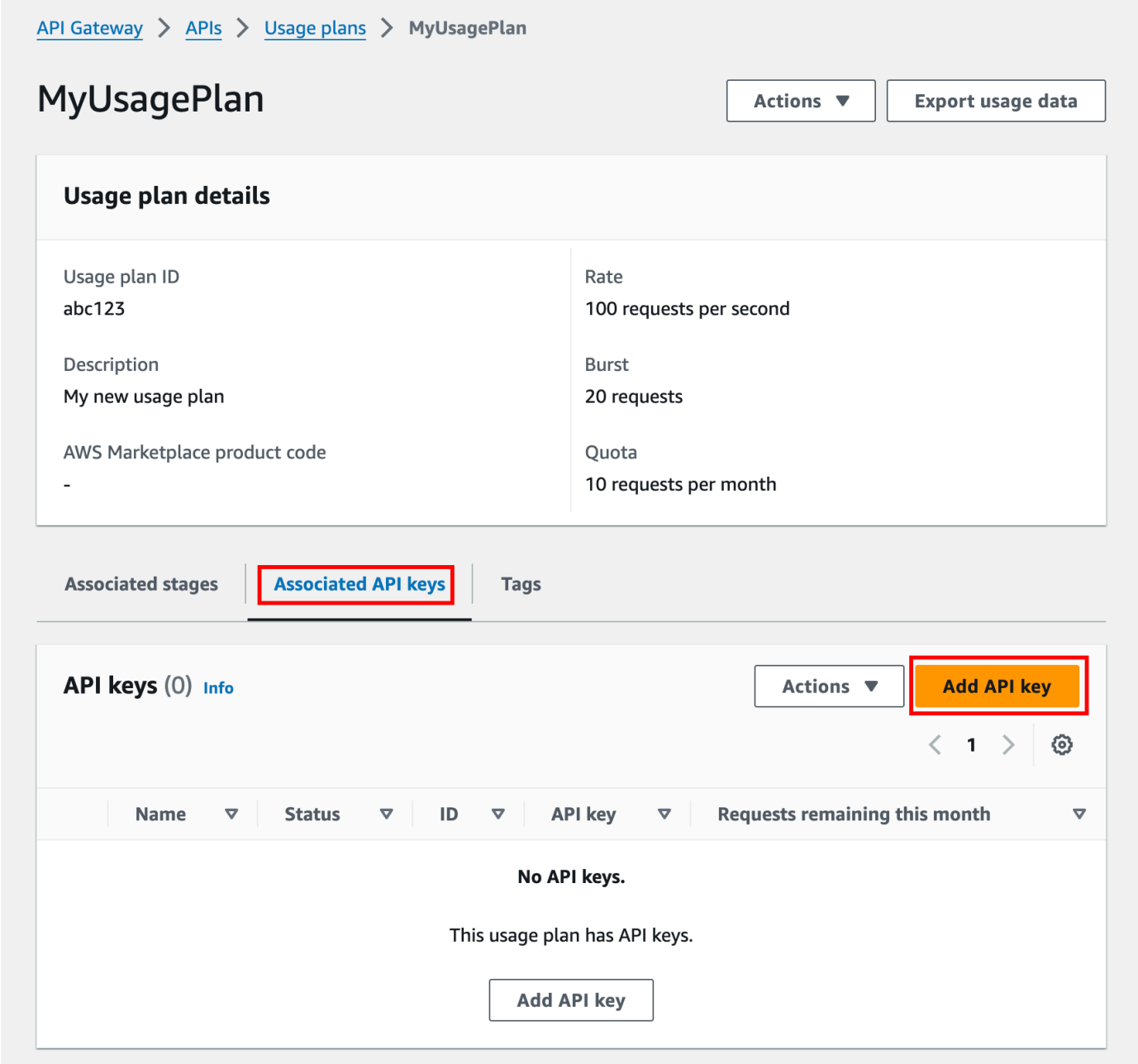
1. Sélectionnez votre plan d'utilisation.
2. Sous l'onglet Étapes associées, choisissez Ajouter une étape.



3. Pour API, sélectionnez une API.
4. Pour Étape, sélectionnez une étape.
5. (Facultatif) Pour activer la limitation au niveau de la méthode, procédez comme suit :
   a. Choisissez Limitation au niveau de la méthode, puis choisissez Ajouter une méthode.
   b. Pour Ressource, sélectionnez une ressource dans votre API.
   c. Pour Méthode, sélectionnez une méthode dans votre API.
   d. Entrez un Taux et un Débit pour votre plan d'utilisation.
6. Choisissez Ajouter au plan d'utilisation.

Pour ajouter une clé au plan d'utilisation

1. Sous l'onglet Clés d'API associées, choisissez Ajouter une clé d'API.



2. a. Pour associer une clé existante à votre plan d'utilisation, sélectionnez Ajouter une clé existante, puis sélectionnez votre clé existante dans le menu déroulant.

   b. Pour créer une nouvelle clé d'API, sélectionnez Créer et ajouter une clé, puis créez une nouvelle clé. Pour plus d'informations sur la création d'une nouvelle clé, consultez Créez une clé API :.

3. Choisissez Ajouter une clé d'API.

Test d'un plan d'utilisation

Pour tester le plan d'utilisation, vous pouvez utiliser un AWS SDK ou un client d'API REST tel que Postman. AWS CLI Pour accéder à un exemple d'utilisation de Postman afin de tester le plan d'utilisation, veuillez consulter Test des plans d'utilisation.

Gérer un plan d'utilisation

La gestion d'un plan d'utilisation implique la surveillance des quotas utilisés et restants sur une période donnée et, si besoin, l'extension des quotas restants selon une valeur spécifiée. Les procédures suivantes expliquent comment surveiller les quotas.

Pour surveiller les quotas utilisés et restants

1. Connectez-vous à la console API Gateway à l'adresse : https://console.aws.amazon.com/apigateway.
2. Dans le panneau de navigation principal d'API Gateway, choisissez Plans d'utilisation.
3. Sélectionnez un plan d'utilisation.
4. Choisissez l'onglet Clés d'API associées pour voir le nombre de demandes restantes pendant la période de temps pour chaque clé.
5. (Facultatif) Choisissez Exporter les données d'utilisation, puis choisissez une date de début De et une date de fin À. Choisissez ensuite JSON ou CSV pour le format des données exportées, puis choisissez Exporter.

   L'exemple suivant montre un fichier exporté.

```
{
  "thisPeriod": {
    "px1KW6...qBazOJH": [
      [
        0,
        5000
      ],
      [
        0,
        5000
      ],
      [
        0,
        10
      ]
```

```
    ]
  },
  "startDate": "2016-08-01",
  "endDate": "2016-08-03"
}
```

Les données d'utilisation de cet exemple présentent les données d'utilisation quotidiennes d'un client d'API, identifié par la clé API (`px1KW6...qBaz0JH`), entre le 1er août 2016 et le 3 août 2016. Les données d'utilisation de chaque jour montrent les quotas utilisés et restants. Dans cet exemple, l'abonné n'a pas encore commencé à utiliser ses quotas alloués et le propriétaire ou l'administrateur de l'API a réduit le quota restant de 5 000 à 10 le troisième jour.

Les procédures suivantes expliquent comment modifier les quotas.

Pour étendre les quotas restants

1. Connectez-vous à la console API Gateway à l'adresse : https://console.aws.amazon.com/apigateway.
2. Dans le panneau de navigation principal d'API Gateway, choisissez Plans d'utilisation.
3. Sélectionnez un plan d'utilisation.
4. Choisissez l'onglet Clés d'API associées pour voir le nombre de demandes restantes pendant la période de temps pour chaque clé.
5. Sélectionnez une clé d'API, puis choisissez Accorder une extension d'utilisation.
6. Entrez un nombre pour définir le quota Demandes restantes. Vous pouvez augmenter ou réduire le nombre de demandes restantes pendant la durée de votre plan d'utilisation.
7. Choisissez Mettre à jour le quota.

Configuration des clés d'API à l'aide de l'API REST API Gateway

Pour configurer les clés API, procédez comme suit :

- Configurez des méthodes API pour exiger une clé API.
- Créez ou importez une clé API pour l'API dans une région.

Avant de configurer des clés API, vous devez avoir créé une API et l'avoir déployée jusqu'à une étape. Une fois que vous avez créé une valeur de clé d'API, elle ne peut pas être modifiée.

Pour que les appels d'API REST créent et déploient une API, veuillez consulter les sections `restapi:create` et `deployment:create`, respectivement.

> **Note**
>
> Pour connaître les meilleures pratiques à prendre en compte, voirthe section called “Bonnes pratiques concernant les clés d'API et les plans d'utilisation”.

Rubriques

- Exigence d'une clé API sur une méthode
- Création ou importation de clés API

Exigence d'une clé API sur une méthode

Pour exiger une clé API sur une méthode, exécutez l'une des actions suivantes :

- Appelez `method:put` pour créer une méthode. Définissez `apiKeyRequired` sur `true` dans la charge utile de la demande.
- Appelez `method:update` pour définir `apiKeyRequired` sur `true`.

Création ou importation de clés API

Pour créer ou importer une clé API, exécutez l'une des actions suivantes :

- Appelez `apikey:create` pour créer une clé d'API.
- Appelez `apikey:import` pour importer une clé d'API à partir d'un fichier. Pour le format de fichier, consultez Format de fichier de clé d'API API Gateway.

Vous ne pouvez pas modifier la valeur de la nouvelle clé d'API. Pour découvrir comment configurer un plan d'utilisation, consultez Création, configuration et test des plans d'utilisation avec l'interface de ligne de commande API Gateway et l'API REST.

Création, configuration et test des plans d'utilisation avec l'interface de ligne de commande API Gateway et l'API REST

Avant de configurer un plan d'utilisation, vous devez avoir au préalable : configuré des méthodes d'une API sélectionnée pour demander des clés API ; déployé ou redéployé l'API jusqu'à une étape ; et avoir créé ou importé une ou plusieurs clés API. Pour de plus amples informations, veuillez consulter Configuration des clés d'API à l'aide de l'API REST API Gateway.

Pour configurer un plan d'utilisation à l'aide de l'API REST API Gateway, suivez les instructions ci-dessous, en supposant que vous avez déjà créé les API à ajouter au plan d'utilisation.

Rubriques

- Migration vers les plans d'utilisation par défaut
- Création d'un plan d'utilisation
- Gérer un plan d'utilisation à l'aide de la AWS CLI
- Test des plans d'utilisation

Migration vers les plans d'utilisation par défaut

Lorsque vous créez un plan d'utilisation pour la première fois, vous pouvez migrer les étapes d'API existantes associées aux clés d'API sélectionnées vers un plan d'utilisation en appelant `account:update` avec le corps suivant :

```
{
  "patchOperations" : [ {
    "op" : "add",
    "path" : "/features",
    "value" : "UsagePlans"
  } ]
}
```

Pour de plus amples informations sur la migration des étapes d'API associées à des clés d'API, veuillez consulter Migration vers les plans d'utilisation par défaut dans la console API Gateway.

Création d'un plan d'utilisation

La procédure suivante explique comment créer un plan d'utilisation.

Pour créer un plan d'utilisation à l'aide de l'API REST

1. Appelez `usageplan:create` pour créer un plan d'utilisation. Dans la charge utile, spécifiez le nom et la description du plan, les étapes d'API associées, les limites de taux et les quotas.

   Notez l'identifiant de plan d'utilisation obtenu. Vous en avez besoin à l'étape suivante.

2. Effectuez l'une des actions suivantes :

   a. Appelez `usageplankey:create` pour ajouter une clé d'API au plan d'utilisation. Spécifiez `keyId` et `keyType` dans la charge utile.

      Pour ajouter d'autres clés API au plan d'utilisation, répétez l'appel précédent, pour une clé API à la fois.

   b. Appelez `apikey:import` pour ajouter directement une ou plusieurs clés d'API au plan d'utilisation spécifié. La charge utile de la requête doit contenir les valeurs de clé API, l'identifiant du plan d'utilisation associé, les indicateurs booléens permettant d'indiquer que les clés sont activées pour le plan d'utilisation et, éventuellement, les noms et descriptions des clés API.

      L'exemple suivant de requête `apikey:import` ajoute trois clés API (identifiées par `key`, `name` et `description`) à un plan d'utilisation (identifié par `usageplanIds`) :

      ```
      POST /apikeys?mode=import&format=csv&failonwarnings=fase HTTP/1.1
      Host: apigateway.us-east-1.amazonaws.com
      Content-Type: text/csv
      Authorization: ...

      key,name, description, enabled, usageplanIds
      abcdef1234ghijklmnop8901234567, importedKey_1, firstone,  tRuE, n371pt
      abcdef1234ghijklmnop0123456789, importedKey_2, secondone, TRUE, n371pt
      abcdef1234ghijklmnop9012345678, importedKey_3,          , true, n371pt
      ```

      En conséquence, trois ressources `UsagePlanKey` sont créées et ajoutées au plan d'utilisation `UsagePlan`.

      Vous pouvez également ajouter des clés API à plusieurs plans d'utilisation de cette façon. Pour ce faire, remplacez la valeur de chaque colonne `usageplanIds` par une chaîne indiquée entre guillemets contenant les identifiants des plans d'utilisation sélectionnés, séparés par des virgules (`"n371pt,m282qs"` ou `'n371pt,m282qs'`).

> **Note**
>
> Une clé d'API peut être associée à plusieurs plans d'utilisation. Un plan d'utilisation peut être associé à plusieurs étapes. Toutefois, une clé d'API donnée ne peut être associée qu'à un seul plan d'utilisation pour chaque étape de votre API.

Gérer un plan d'utilisation à l'aide de la AWS CLI

Les exemples de code suivants montrent comment ajouter, supprimer ou modifier les paramètres de limitation au niveau méthode dans le cadre d'un plan d'utilisation en appelant la commande `update-usage-plan`.

> **Note**
>
> Assurez-vous de remplacer `us-east-1` par la région appropriée pour votre API.

Pour ajouter ou remplacer une limitation de taux afin de restreindre une ressource et une méthode en particulier :

```
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"
```

Pour ajouter ou remplacer une limitation du mode rafale afin de restreindre une ressource et une méthode en particulier :

```
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"
```

Pour supprimer les paramètres de limitation au niveau méthode pour une ressource et une méthode en particulier :

```
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=""
```

Pour supprimer tous les paramètres de limitation au niveau de la méthode pour une API :

```
aws apigateway --region us-east-1 update-usage-plan --usage-plan-id <planId> --patch-
operations op="remove",path="/apiStages/<apiId>:<stage>/throttle ",value=""
```

Voici un extrait utilisant l'exemple d'API Pet Store :

```
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}}"'
```

Test des plans d'utilisation

À titre d'exemple, utilisons l' PetStore API, qui a été créée dansTutoriel : Création d'une API REST par l'importation d'un exemple. Supposons que l'API est configurée pour utiliser la clé API `Hiorr45VR...c4GJc`. Les étapes suivantes expliquent comment tester un plan d'utilisation.

Pour tester votre plan d'utilisation

- Envoyez une demande `GET` sur la ressource Pets (`/pets`), avec les paramètres de requête ? `type=...&page=...` de l'API (par exemple, `xbvxlpijch`) dans un plan d'utilisation :

  ```
  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**
>
> Vous devez envoyer cette demande au composant `execute-api` d'API Gateway et fournir la clé d'API requise (par exemple, `Hiorr45VR...c4GJc`) dans l'en-tête `x-api-key` requis.

La réponse positive renvoie un code d'état `200 OK` et une charge utile qui contient les résultats demandés obtenus du backend. Si vous oubliez de définir l'en-tête `x-api-key` ou si vous le définissez avec une clé incorrecte, vous obtenez une réponse `403 Forbidden`. Toutefois, si vous n'avez pas configuré la méthode pour exiger une clé API, vous obtiendrez probablement une réponse `200 OK` que vous définissiez l'en-tête `x-api-key` correctement ou non, et les limites de quota et de restriction du plan d'utilisation seront ignorées.

Parfois, lorsqu'une erreur interne se produit qui empêche API Gateway d'appliquer les limitations de plan d'utilisation ou les quotas pour la demande, API Gateway traite la demande sans appliquer les limitations ou les quotas comme indiqué dans le plan d'utilisation. Mais, il enregistre un message d'erreur `Usage Plan check failed due to an internal error` d'entrée CloudWatch. Vous pouvez ignorer ces erreurs occasionnelles.

Créez et configurez des clés d'API et des plans d'utilisation avec AWS CloudFormation

Vous pouvez l'utiliser AWS CloudFormation pour exiger des clés d'API sur les méthodes d'API et créer un plan d'utilisation pour une API. L'exemple de AWS CloudFormation modèle effectue les opérations suivantes :

- Il crée une API dans l'API Gateway avec les méthodes `GET` et `POST`.
- Il nécessite une clé d'API pour les méthodes `GET` et `POST`. Cette API reçoit des clés à partir de l'en-tête `X-API-KEY` de chaque demande entrante.
- Il crée une clé d'API.
- Il crée un plan d'utilisation pour spécifier un quota mensuel de 1 000 demandes par mois, une limite de taux de limitation de 100 demandes par seconde et une limitation de débit en rafale de 200 demandes par seconde.
- Il spécifie une limite de fréquence de limitation au niveau de la méthode de 50 demandes par seconde et une limite de fréquence de limitation au niveau de la méthode de 100 demandes par seconde pour la méthode `GET`.

- Il associe l'étape d'API et la clé d'API au plan d'utilisation.

```
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}'
```

Configuration d'une méthode pour utiliser des clés d'API avec une définition OpenAPI

Vous pouvez utiliser une définition OpenAPI pour exiger des clés d'API sur une méthode.

Pour chaque méthode, créez un objet d'exigence de sécurité nécessitant une clé d'API pour appeler cette méthode. Définissez ensuite `api_key` dans la définition de sécurité. Après avoir créé votre API, ajoutez la nouvelle étape d'API à votre plan d'utilisation.

L'exemple suivant crée une API et nécessite une clé d'API pour les `GET` méthodes `POST` et :

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"
      }
    }
  }
}
```

Format de fichier de clé d'API API Gateway

API Gateway peut importer les clés d'API à partir de fichiers externes au format CSV (valeurs séparées par des virgules), puis associer les clés importées à un ou plusieurs plans d'utilisation. Le fichier importé doit comporter les colonnes `Name` et `Key`. Les noms d'en-tête de colonne ne sont pas sensibles à la casse et les colonnes peuvent être dans n'importe quel ordre, comme illustré dans l'exemple suivant :

```
Key,name
apikey1234abcdefghij0123456789,MyFirstApiKey
```

Une `Key` valeur doit être comprise entre 20 et 128 caractères. Une valeur `Name` ne peut pas dépasser 1 024 caractères.

Un fichier de clé API peut également contenir la colonne `Description`, `Enabled` ou `UsagePlanIds`, comme illustré dans l'exemple suivant :

```
Name,key,description,Enabled,usageplanIds
MyFirstApiKey,apikey1234abcdefghij0123456789,An imported key,TRUE,c7y23b
```

Lorsqu'une clé est associée à plusieurs plans d'utilisation, la valeur `UsagePlanIds` est une chaîne d'ID de plan d'utilisation séparés par des virgules, indiquée entre guillemets simples ou doubles, comme illustré dans l'exemple suivant :

```
Enabled,Name,key,UsageplanIds
true,MyFirstApiKey,apikey1234abcdefghij0123456789,"c7y23b,glvrsr"
```

Les colonnes non reconnues sont autorisées, mais sont ignorées. La valeur par défaut est une chaîne vide ou une valeur booléenne `true`.

Il est possible d'importer la même clé API plusieurs fois, auquel cas la version la plus récente remplace la précédente. Deux clés d'API sont identiques si elles ont la même valeur `key`.

> **Note**
>
> Pour connaître les bonnes pratiques à prendre en compte, consultez the section called "Bonnes pratiques concernant les clés d'API et les plans d'utilisation".

Documentation sur les API REST

Pour aider les clients à comprendre et à utiliser votre API, vous devez documenter cette dernière. Pour vous aider à documenter votre API, API Gateway vous permet d'ajouter et de mettre à jour le contenu de chaque entité de l'API dans le cadre du processus de développement de l'API. API Gateway stocke le contenu source et vous permet d'archiver différentes versions de la documentation. Vous pouvez associer une version de la documentation à une étape de l'API, exporter un instantané de la documentation propre à une étape donnée dans un fichier OpenAPI externe et distribuer le fichier sous forme de publication de la documentation.

Pour documenter votre API, vous pouvez appeler l'API REST API Gateway, utiliser l'un des AWS SDK, utiliser AWS CLIl'API Gateway ou utiliser la console API Gateway. En outre, vous pouvez importer ou exporter les parties de la documentation définies dans un fichier OpenAPI externe.

Pour partager la documentation de l'API avec les développeurs, vous pouvez utiliser un portail pour développeurs. Par exemple, consultez Integrating ReadMe with API Gateway to keep your Developer Hub à jour sur le blog AWS Partner Network (APN).

Rubriques

- Représentation de la documentation de l'API dans API Gateway
- Documentation d'une API avec la console API Gateway
- Publication de la documentation de l'API à l'aide de la console API Gateway
- Documentation d'une API en utilisant l'API REST API Gateway
- Publication de la documentation de l'API à l'aide de l'API REST
- Importation de la documentation d'une API
- Contrôle de l'accès à la documentation de l'API

Représentation de la documentation de l'API dans API Gateway

La documentation de l'API API Gateway est composée de plusieurs parties de la documentation associées aux entités API spécifiques qui incluent l'API, la ressource, la méthode, la demande, la réponse, les paramètres de message (par exemple, le chemin d'accès, la requête, l'en-tête), ainsi que les mécanismes d'autorisation et les modèles.

Dans API Gateway, une partie de la documentation est représentée par une DocumentationPartressource. La documentation de l'API dans son ensemble est représentée par la DocumentationPartscollection.

La documentation d'une API consiste à créer des instances `DocumentationPart`, à les ajouter à la collection `DocumentationParts` et à gérer les versions des parties de la documentation au fur et à mesure de l'évolution de l'API.

Rubriques

- Parties de la documentation
- Versions de la documentation

Parties de la documentation

Une DocumentationPartressource est un objet JSON qui stocke le contenu de la documentation applicable à une entité d'API individuelle. Son champ `properties` contient la documentation sous forme de mappage de paires clé-valeur. Sa propriété `location` identifie l'entité d'API associée.

La forme d'un mappage de contenu est déterminée par vous-même en tant que développeur de l'API. La valeur d'une paire clé-valeur peut être une chaîne, un nombre, une valeur booléenne, un objet ou un tableau. La forme de l'objet `location` varie selon le type d'entité ciblé.

La ressource `DocumentationPart` prend en charge l'héritage du contenu : le contenu de la documentation d'une entité d'API s'applique aux enfants de cette entité d'API. Pour plus d'informations sur la définition des entités enfants et sur l'héritage de contenu, consultez Héritage du contenu d'une entité d'API générale.

Emplacement de la partie d'une documentation

La propriété de localisation d'une DocumentationPartinstance identifie une entité d'API à laquelle s'applique le contenu associé. L'entité API peut être une ressource d'API REST API Gateway, telle que Resource RestApi, Method MethodResponse, Authorizer ou Model. L'entité peut également être un paramètre de message, par exemple un paramètre de chemin d'accès d'URL, un paramètre de chaîne de requête, un paramètre de demande ou d'en-tête de réponse, un corps de demande ou de réponse, ou un code de statut de réponse.

Pour spécifier une entité d'API, définissez l'attribut type de l'objet `location` comme `API`, `AUTHORIZER`, `MODEL`, `RESOURCE`, `METHOD`, `PATH_PARAMETER`, `QUERY_PARAMETER`, `REQUEST_HEADER`, `REQUEST_BODY`, `RESPONSE`, `RESPONSE_HEADER` ou `RESPONSE_BODY`.

Selon le `type` d'une entité d'API, vous pouvez spécifier d'autres attributs `location`, dont method, name, path et statusCode. Certains attributs ne sont pas valides pour une entité d'API donnée. Par exemple, `type`, `path`, `name` et `statusCode` sont des attributs valides pour l'entité `RESPONSE` ; mais seuls `type` et `path` sont des attributs d'emplacement valides pour l'entité `RESOURCE`. C'est une erreur d'inclure un champ non valide dans l'attribut `location` d'une `DocumentationPart` pour une entité d'API donnée.

Tous les champs `location` valides ne sont pas obligatoires. Par exemple, `type` est le champ valide et obligatoire `location` de toutes les entités d'API. Toutefois, `method`, `path` et `statusCode` sont des attributs valides mais facultatifs pour l'entité `RESPONSE`. Lorsqu'il n'est pas spécifié explicitement, un champ `location` valide se voit attribuer sa valeur par défaut. La valeur par défaut de `path` est

/, c'est-à-dire la ressource racine d'une API. La valeur par défaut de `method` ou `statusCode` est *, c'est-à-dire n'importe quelle méthode ou valeur de code de statut, respectivement.

Contenu d'une partie de la documentation

La valeur `properties` est codée sous forme de chaîne JSON. Le valeur `properties` contient les informations que vous choisissez afin de répondre à vos besoins en documentation. Par exemple, ce qui suit est un mappage de contenu valide :

```
{
  "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."
}
```

Même si API Gateway accepte une chaîne JSON valide en tant que mappage de contenu, les attributs de contenu sont traités selon deux catégories : ceux qui peuvent être reconnus par OpenAPI et ceux qui ne le peuvent pas. Dans l'exemple précédent, `info`, `description` et `x-custom-info` sont reconnus par OpenAPI comme objet, propriété ou extension OpenAPI standard. En revanche, `my-info` n'est pas conforme à la spécification OpenAPI. API Gateway diffuse les attributs de contenu conformes à OpenAPI dans les définitions d'entité API à partir des instances `DocumentationPart` associées. API Gateway ne diffuse pas les attributs de contenu non conformes à la définition d'entité d'API.

Voici un autre exemple, `DocumentationPart` ciblé pour une entité `Resource` :

```
{
    "location" : {
        "type" : "RESOURCE",
        "path": "/pets"
    },
    "properties" : {
        "summary" : "The /pets resource represents a collection of pets in PetStore.",
        "description": "... a child resource under the root...",
    }
}
```

Ici, `type` et `path` sont des champs valides pour identifier la cible du type `RESOURCE`. Pour la ressource racine (/), vous pouvez ignorer le champ `path`.

```
{
    "location" : {
        "type" : "RESOURCE"
    },
    "properties" : {
        "description" : "The root resource with the default path specification."
    }
}
```

C'est identique à l'instance `DocumentationPart` suivante :

```
{
    "location" : {
        "type" : "RESOURCE",
        "path": "/"
    },
    "properties" : {
        "description" : "The root resource with an explicit path specification"
    }
}
```

Héritage du contenu de spécifications plus générales d'une entité d'API

La valeur par défaut d'un champ `location` facultatif fournit la description de modèle d'une entité d'API. À l'aide de la valeur par défaut de l'objet `location`, vous pouvez ajouter une description générale dans le mappage `properties` à une instance `DocumentationPart` avec ce type de modèle `location`. API Gateway extrait les attributs de documentation applicables à partir du champ `DocumentationPart` de l'entité API générique et les insère dans une entité API spécifique avec les champs `location` correspondant au modèle général `location` ou correspondant à la valeur exacte, à moins que l'entité spécifique n'ait déjà une instance `DocumentationPart` associée. Ce comportement est également connu sous le nom d'héritage de contenu de spécifications plus générales à partir d'une entité d'API.

L'héritage de contenu ne s'applique pas à certains types d'entités d'API. Consultez le tableau ci-dessous pour plus d'informations.

Lorsqu'une entité d'API correspond à plusieurs modèles d'emplacement de `DocumentationPart`, l'entité hérite de la partie de la documentation avec les champs d'emplacement ayant la priorité et les spécificités les plus élevés. L'ordre de priorité est `path` > `statusCode`. Pour que cela corresponde

au champ `path`, API Gateway choisit l'entité ayant la valeur de chemin d'accès la plus spécifique. Le tableau suivant présente quelques exemples.

Cas	**path**	**statusCode**	**name**	Remar
1	`/pets`	`*`	`id`	La documentation associée à ce modèle d'emplacement sera héritée par les entités correspondant au modèle d'emplacement.
2	`/pets`	`200`	`id`	La documentation associée à ce modèle d'emplacement sera héritée par

Cas	**path**	**statusCode**	**name**	Remarques
				les entités correspondant au modèle d'emplacement lorsque les cas 1 et 2 font l'objet d'une correspondance car le cas 2 est plus spécifique que le cas 1.

Cas	**path**	**statusCode**	**name**	Remarques
3	`/pets/petId`	`*`	`id`	La documentation associée à ce modèle d'emplacement est héritée par les entités correspondant au modèle d'emplacement lorsque les cas 1, 2 et 3 font l'objet d'une correspondance, car le cas 3 a une

Cas	path	statusCode	name	Remarque
				priorité plus élevée que le cas 2 et il est plus spécifique que le cas 1.

Voici un autre exemple dont le but est de comparer une instance `DocumentationPart` plus générique à une instance plus spécifique. Le message d'erreur générale `"Invalid request error"` est ajouté aux définitions OpenAPI des réponses d'erreur `400`, à moins qu'il ne soit remplacé.

```
{
    "location" : {
        "type" : "RESPONSE",
        "statusCode": "400"
    },
    "properties" : {
        "description" : "Invalid request error."
    }"
}
```

Avec le remplacement suivant, les réponses `400` à toutes les méthodes sur la ressource `/pets` ont une description `"Invalid petId specified"` à la place.

```
{
    "location" : {
```

```
        "type" : "RESPONSE",
        "path": "/pets",
        "statusCode": "400"
    },
    "properties" : "{
        "description" : "Invalid petId specified."
    }"
}
```

Champs d'emplacement valides **`DocumentationPart`**

Le tableau suivant indique les champs valides et obligatoires ainsi que les valeurs par défaut applicables d'une DocumentationPartressource associée à un type donné d'entités API.

Entité de l'API	Champs d'emplacement valides	Champs d'emplacement obligatoires	Valeurs de champ par défaut	Contenu pouvant être hérité
API	`{ "location": { "type": "API" }, ... }`	`type`	S/O	Non
Ressource	`{ "location": { "type": "RESOURCE", "path": "resource_path " }, ... }`	`type`	La valeur par défaut de `path` est `/`.	Non
Method	`{ "location": { "type": "METHOD",`	`type`	Les valeurs par défaut de `path` et `method` sont	Oui, correspondance de

Entité de l'API	Champs d'emplacement valides	Champs d'emplacement obligatoires	Valeurs de champ par défaut	Contenu pouvant être hérité
	<pre>"path": "*resource_path* ", "method": "*http_verb* " }, ... }</pre>		/ et *, respectivement.	path par préfixe et correspondance de method quelle que soit la valeur.
Paramètre de la demande	<pre>{ "location": { "type": "QUERY_PARAMETER", "path": "*resource_path* ", "method": "*HTTP_verb* ", "name": "*query_parameter_name* " }, ... }</pre>	type	Les valeurs par défaut de path et method sont / et *, respectivement.	Oui, correspondance de path par préfixe et correspondance de method pour des valeurs exactes.

Entité de l'API	Champs d'emplacement valides	Champs d'emplacement obligatoires	Valeurs de champ par défaut	Contenu pouvant être hérité
Corps de la demande	`{ "location": { "type": "REQUEST_BODY", "path": "resource_path ", "method": "http_verb " }, ... }`	`type`	Les valeurs par défaut de `path` et `method` sont `/` et `*`, respectivement.	Oui, correspondance de `path` par préfixe et correspondance de `method` pour des valeurs exactes.
Paramètre d'en-tête de la demande	`{ "location": { "type": "REQUEST_HEADER", "path": "resource_path ", "method": "HTTP_verb ", "name": "header_name " }, ... }`	`type`, `name`	Les valeurs par défaut de `path` et `method` sont `/` et `*`, respectivement.	Oui, correspondance de `path` par préfixe et correspondance de `method` pour des valeurs exactes.

Entité de l'API	Champs d'emplacement valides	Champs d'emplacement obligatoires	Valeurs de champ par défaut	Contenu pouvant être hérité
Paramètre de chemin d'accès de la demande	`{ "location": { "type": "PATH_PARAMETER", "path": "resource/{path_parameter_name }", "method": "HTTP_verb ", "name": "path_parameter_name " }, ... }`	`type`, `name`	Les valeurs par défaut de `path` et `method` sont `/` et `*`, respectivement.	Oui, correspondance de `path` par préfixe et correspondance de `method` pour des valeurs exactes.
Réponse	`{ "location": { "type": "RESPONSE", "path": "resource_path ", "method": "http_verb ", "statusCode": "status_code " }, ... }`	`type`	Les valeurs par défaut de `path`, `method` et `statusCode` sont `/`, `*` et `*`, respectivement.	Oui, correspondance de `path` par préfixe et correspondance de `method` et `statusCode` pour des valeurs exactes.

Entité de l'API	Champs d'emplacement valides	Champs d'emplacement obligatoires	Valeurs de champ par défaut	Contenu pouvant être hérité
En-tête de réponse	{ "location": { "type": "RESPONSE_HEADER", "path": "*resource_path* ", "method": "*http_verb* ", "statusCode": "*status_code* ", "name": "*header_name* " }, ... }	`type`, `name`	Les valeurs par défaut de `path`, `method` et `statusCode` sont `/`, * et *, respectivement.	Oui, correspondance de `path` par préfixe et correspondance de `method` et `statusCode` pour des valeurs exactes.
Corps de la réponse	{ "location": { "type": "RESPONSE_BODY", "path": "*resource_path* ", "method": "*http_verb* ", "statusCode": "*status_code* " }, ... }	`type`	Les valeurs par défaut de `path`, `method` et `statusCode` sont `/`, * et *, respectivement.	Oui, correspondance de `path` par préfixe et correspondance de `method` et `statusCode` pour des valeurs exactes.

Entité de l'API	Champs d'emplacement valides	Champs d'emplacement obligatoires	Valeurs de champ par défaut	Contenu pouvant être hérité
Mécanisme d'autorisation	`{ "location": { "type": "AUTHORIZER", "name": "authorizer_name " }, ... }`	`type`	S/O	Non
Modèle	`{ "location": { "type": "MODEL", "name": "model_name " }, ... }`	`type`	S/O	Non

Versions de la documentation

Une version de documentation est un instantané de la DocumentationPartscollection d'une API et est étiquetée avec un identifiant de version. La publication de la documentation d'une API implique la création d'une version de documentation, son association à une étape de l'API et l'exportation cette version spécifique à l'étape de la documentation de l'API dans un fichier OpenAPI externe. Dans API Gateway, un instantané de documentation est représenté sous forme de DocumentationVersionressource.

Lorsque vous mettez à jour une API, vous créez de nouvelles versions de cette dernière. Dans API Gateway, vous gérez toutes les versions de documentation à l'aide de la DocumentationVersionscollection.

Documentation d'une API avec la console API Gateway

Dans cette section, nous décrivons la création et la gestion des parties de la documentation d'une API à l'aide de la console API Gateway.

Pour créer et de modifier la documentation d'une API, vous devez avoir déjà créé l'API. Dans cette section, nous utilisons l'PetStoreAPI comme exemple. Pour créer une API à l'aide de la console API Gateway, suivez les instructions de Tutoriel : Création d'une API REST par l'importation d'un exemple.

Rubriques

Documentation de l'entité **`API`**

Pour ajouter une nouvelle partie de la documentation pour l'entité `API`, procédez comme suit :

1. Dans le volet de navigation principal, choisissez Documentation, puis choisissez Créer une partie de la documentation.
2. Pour Type de documentation, sélectionnez API.

   Si une partie de la documentation n'a pas été créée pour l'`API`, vous obtenez l'éditeur de mappage `properties` de la partie de la documentation. Entrez la carte `properties` suivante dans l'éditeur de texte.

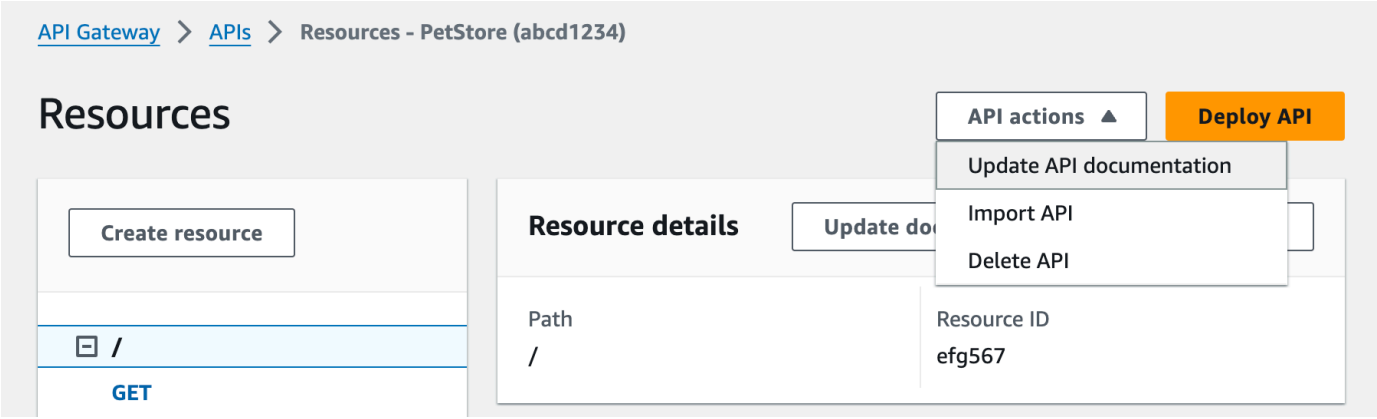
```
{
  "info": {
    "description": "Your first API Gateway API.",
    "contact": {
        "name": "John Doe",
        "email": "john.doe@api.com"
    }
  }
}
```

> **Note**
>
> Vous n'avez pas besoin d'encoder le mappage `properties` en une chaîne JSON. La console API Gateway convertit automatiquement l'objet JSON en chaîne.

3. Choisissez Créer une partie de la documentation.

Pour ajouter une nouvelle partie de la documentation pour l'entité `API` dans le volet Ressources, procédez comme suit :

1. Dans le volet de navigation principal, choisissez Ressources.
2. Choisissez le menu Actions API, puis Mettre à jour la documentation de l'API.



Pour modifier une partie de la documentation existante, procédez comme suit :

1. Dans le volet Documentation, choisissez l'onglet Ressources et méthodes.
2. Sélectionnez le nom de votre API, puis sur la carte de l'API, choisissez Modifier.

Documentation d'une entité **RESOURCE**

Pour ajouter une nouvelle partie de la documentation pour une entité RESOURCE, procédez comme suit :

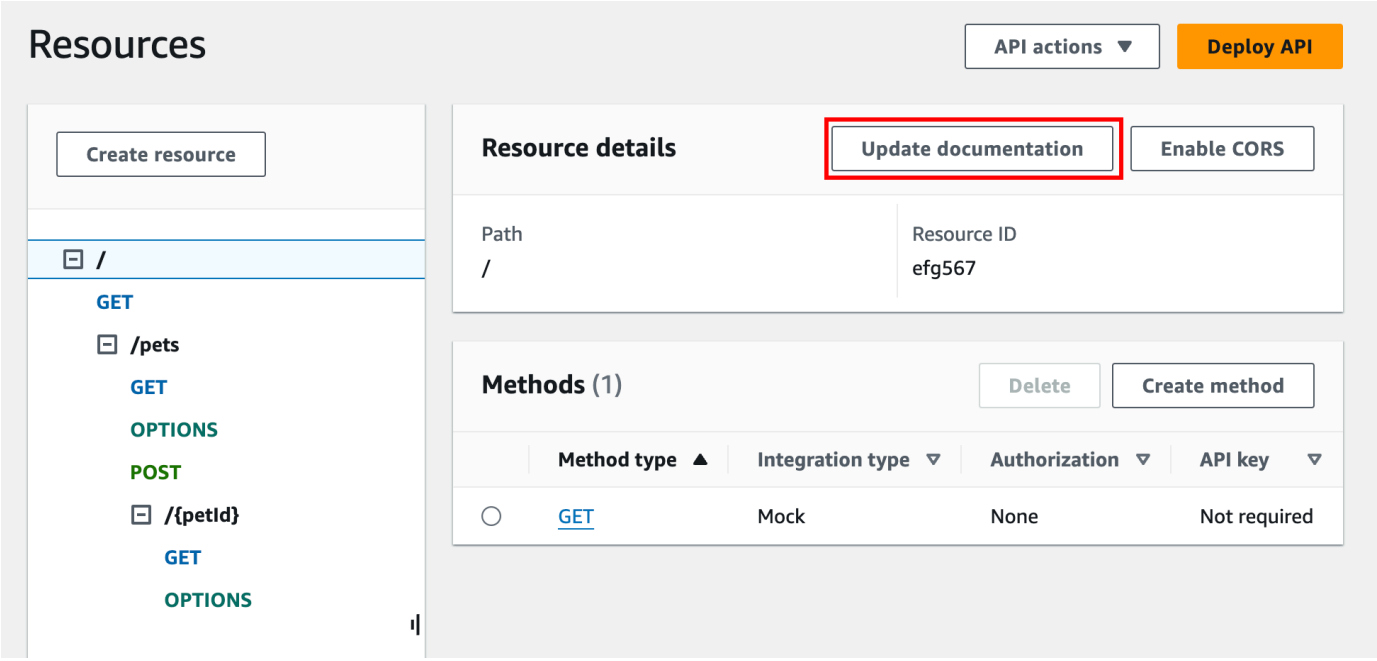
1. Dans le volet de navigation principal, choisissez Documentation, puis Créer une partie de la documentation.
2. Pour Type de documentation, sélectionnez Ressource.
3. Pour Chemin, entrez un chemin.
4. Entrez une description dans l'éditeur de texte, par exemple :

```
{
    "description": "The PetStore's root resource."
}
```

5. Choisissez Créer une partie de la documentation. Vous pouvez créer de la documentation pour une ressource non répertoriée.
6. Si nécessaire, répétez ces étapes pour ajouter ou modifier une autre partie de la documentation.

Pour ajouter une nouvelle partie de la documentation pour une entité RESOURCE dans le volet Ressources, procédez comme suit :

1. Dans le volet de navigation principal, choisissez Ressources.
2. Choisissez la ressource, puis Mettre à jour la documentation.



Pour modifier une partie de la documentation existante, procédez comme suit :

1. Dans le volet Documentation, choisissez l'onglet Ressources et méthodes.
2. Sélectionnez la ressource contenant votre partie de la documentation, puis choisissez Modifier.

Documentation d'une entité **`METHOD`**

Pour ajouter une nouvelle partie de la documentation pour une entité `METHOD`, procédez comme suit :

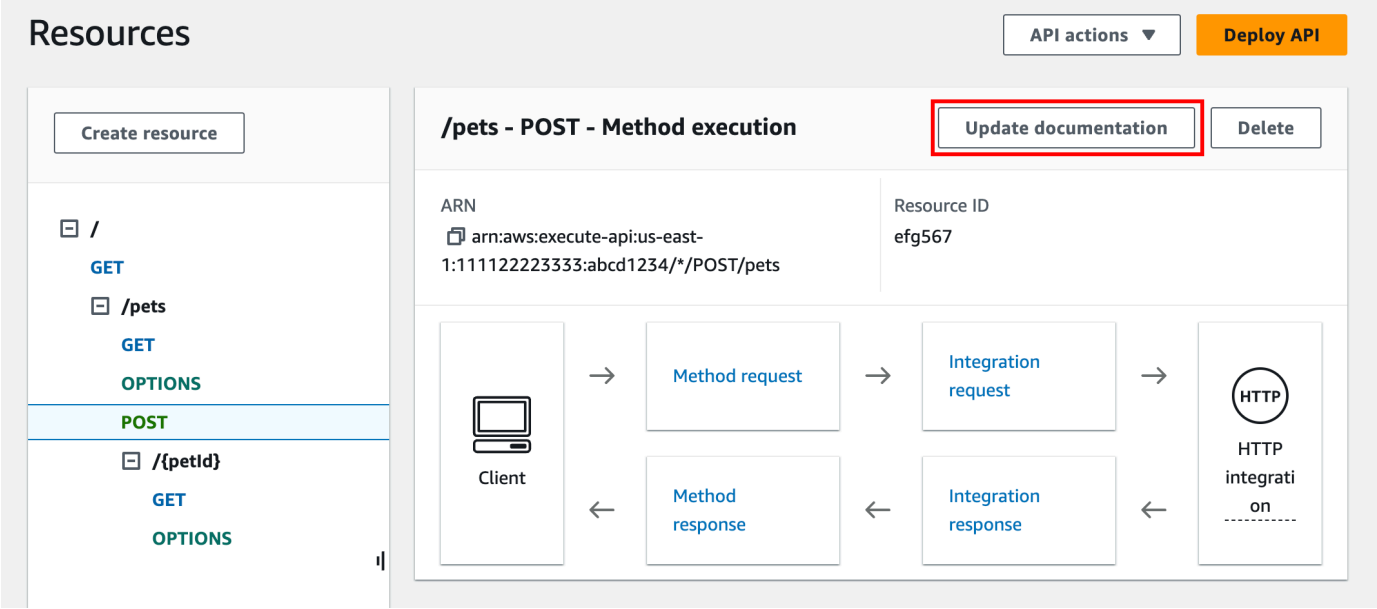
1. Dans le volet de navigation principal, choisissez Documentation, puis Créer une partie de la documentation.
2. Pour Type de documentation, sélectionnez Méthode.
3. Pour Chemin, entrez un chemin.
4. Pour Méthode, sélectionnez un verbe HTTP.
5. Entrez une description dans l'éditeur de texte, par exemple :

```
{
  "tags" : [ "pets" ],
  "summary" : "List all pets"
}
```

6. Choisissez Créer une partie de la documentation. Vous pouvez créer de la documentation pour une méthode non répertoriée.
7. Si nécessaire, répétez ces étapes pour ajouter ou modifier une autre partie de la documentation.

Pour ajouter une nouvelle partie de la documentation pour une entité `METHOD` dans le volet Ressources, procédez comme suit :

1. Dans le volet de navigation principal, choisissez Ressources.
2. Choisissez la méthode, puis Mettre à jour la documentation.



Pour modifier une partie de la documentation existante, procédez comme suit :

1. Dans le volet Documentation, choisissez l'onglet Ressources et méthodes.
2. Vous pouvez sélectionner la méthode ou la ressource contenant la méthode, puis utiliser la barre de recherche pour rechercher et sélectionner la partie de votre documentation.
3. Choisissez Modifier.

Documentation d'une entité `QUERY_PARAMETER`

Pour ajouter une nouvelle partie de la documentation pour une entité `QUERY_PARAMETER`, procédez comme suit :

1. Dans le volet de navigation principal, choisissez Documentation, puis Créer une partie de la documentation.
2. Pour Type de documentation, sélectionnez Paramètre de requête.
3. Pour Chemin, entrez un chemin.
4. Pour Méthode, sélectionnez un verbe HTTP.
5. Pour Nom, entrez un nom.
6. Entrez une description dans l'éditeur de texte.
7. Choisissez Créer une partie de la documentation. Vous pouvez créer de la documentation pour un paramètre de requête non répertorié.
8. Si nécessaire, répétez ces étapes pour ajouter ou modifier une autre partie de la documentation.

Pour modifier une partie de la documentation existante, procédez comme suit :

1. Dans le volet Documentation, choisissez l'onglet Ressources et méthodes.
2. Vous pouvez sélectionner le paramètre de requête ou la ressource contenant le paramètre de requête, puis utiliser la barre de recherche pour rechercher et sélectionner la partie de votre documentation.
3. Choisissez Modifier.

Documentation d'une entité **`PATH_PARAMETER`**

Pour ajouter une nouvelle partie de la documentation pour une entité `PATH_PARAMETER`, procédez comme suit :

1. Dans le volet de navigation principal, choisissez Documentation, puis Créer une partie de la documentation.
2. Pour Type de documentation, sélectionnez Paramètre de chemin.
3. Pour Chemin, entrez un chemin.
4. Pour Méthode, sélectionnez un verbe HTTP.
5. Pour Nom, entrez un nom.
6. Entrez une description dans l'éditeur de texte.
7. Choisissez Créer une partie de la documentation. Vous pouvez créer de la documentation pour un paramètre de chemin non répertorié.
8. Si nécessaire, répétez ces étapes pour ajouter ou modifier une autre partie de la documentation.

Pour modifier une partie de la documentation existante, procédez comme suit :

1. Dans le volet Documentation, choisissez l'onglet Ressources et méthodes.
2. Vous pouvez sélectionner le paramètre de chemin ou la ressource contenant le paramètre de chemin, puis utiliser la barre de recherche pour rechercher et sélectionner la partie de votre documentation.
3. Choisissez Modifier.

Documentation d'une entité **REQUEST_HEADER**

Pour ajouter une nouvelle partie de la documentation pour une entité `REQUEST_HEADER`, procédez comme suit :

1. Dans le volet de navigation principal, choisissez Documentation, puis Créer une partie de la documentation.
2. Pour Type de documentation, sélectionnez En-tête de demande.
3. Pour Chemin, entrez un chemin pour l'en-tête de demande.
4. Pour Méthode, sélectionnez un verbe HTTP.
5. Pour Nom, entrez un nom.
6. Entrez une description dans l'éditeur de texte.
7. Choisissez Créer une partie de la documentation. Vous pouvez créer de la documentation pour un en-tête de demande non répertorié.
8. Si nécessaire, répétez ces étapes pour ajouter ou modifier une autre partie de la documentation.

Pour modifier une partie de la documentation existante, procédez comme suit :

1. Dans le volet Documentation, choisissez l'onglet Ressources et méthodes.
2. Vous pouvez sélectionner l'en-tête de demande ou la ressource contenant l'en-tête de demande, puis utiliser la barre de recherche pour rechercher et sélectionner la partie de votre documentation.
3. Choisissez Modifier.

Documentation d'une entité **REQUEST_BODY**

Pour ajouter une nouvelle partie de la documentation pour une entité `REQUEST_BODY`, procédez comme suit :

1. Dans le volet de navigation principal, choisissez Documentation, puis Créer une partie de la documentation.
2. Pour Type de documentation, sélectionnez Corps de la demande.
3. Pour Chemin, entrez un chemin pour le corps de la demande.
4. Pour Méthode, sélectionnez un verbe HTTP.
5. Entrez une description dans l'éditeur de texte.
6. Choisissez Créer une partie de la documentation. Vous pouvez créer de la documentation pour un corps de la demande non répertorié.
7. Si nécessaire, répétez ces étapes pour ajouter ou modifier une autre partie de la documentation.

Pour modifier une partie de la documentation existante, procédez comme suit :

1. Dans le volet Documentation, choisissez l'onglet Ressources et méthodes.
2. Vous pouvez sélectionner le corps de la demande ou la ressource contenant le corps de la demande, puis utiliser la barre de recherche pour rechercher et sélectionner la partie de votre documentation.
3. Choisissez Modifier.

Documentation d'une entité **RESPONSE**

Pour ajouter une nouvelle partie de la documentation pour une entité `RESPONSE`, procédez comme suit :

1. Dans le volet de navigation principal, choisissez Documentation, puis Créer une partie de la documentation.
2. Pour Type de documentation, sélectionnez Réponse (code de statut).
3. Pour Chemin, entrez un chemin pour la réponse.
4. Pour Méthode, sélectionnez un verbe HTTP.
5. Pour Code de statut, entrez un code de statut HTTP.
6. Entrez une description dans l'éditeur de texte.

7. Choisissez Créer une partie de la documentation. Vous pouvez créer de la documentation pour un code de statut de réponse non répertorié.
8. Si nécessaire, répétez ces étapes pour ajouter ou modifier une autre partie de la documentation.

Pour modifier une partie de la documentation existante, procédez comme suit :

1. Dans le volet Documentation, choisissez l'onglet Ressources et méthodes.
2. Vous pouvez sélectionner le code de statut de réponse ou la ressource contenant le code de statut de réponse, puis utiliser la barre de recherche pour rechercher et sélectionner la partie de votre documentation.
3. Choisissez Modifier.

Documentation d'une entité **`RESPONSE_HEADER`**

Pour ajouter une nouvelle partie de la documentation pour une entité `RESPONSE_HEADER`, procédez comme suit :

1. Dans le volet de navigation principal, choisissez Documentation, puis Créer une partie de la documentation.
2. Pour Type de documentation, sélectionnez En-tête de réponse.
3. Pour Chemin, entrez un chemin pour l'en-tête de réponse.
4. Pour Méthode, sélectionnez un verbe HTTP.
5. Pour Code de statut, entrez un code de statut HTTP.
6. Entrez une description dans l'éditeur de texte.
7. Choisissez Créer une partie de la documentation. Vous pouvez créer de la documentation pour un en-tête de réponse non répertorié.
8. Si nécessaire, répétez ces étapes pour ajouter ou modifier une autre partie de la documentation.

Pour modifier une partie de la documentation existante, procédez comme suit :

1. Dans le volet Documentation, choisissez l'onglet Ressources et méthodes.
2. Vous pouvez sélectionner l'en-tête de réponse ou la ressource contenant l'en-tête de réponse, puis utiliser la barre de recherche pour rechercher et sélectionner la partie de votre documentation.
3. Choisissez Modifier.

Documentation d'une entité **RESPONSE_BODY**

Pour ajouter une nouvelle partie de la documentation pour une entité `RESPONSE_BODY`, procédez comme suit :

1. Dans le volet de navigation principal, choisissez Documentation, puis Créer une partie de la documentation.
2. Pour Type de documentation, sélectionnez Corps de la réponse.
3. Pour Chemin, entrez un chemin pour le corps de la réponse.
4. Pour Méthode, sélectionnez un verbe HTTP.
5. Pour Code de statut, entrez un code de statut HTTP.
6. Entrez une description dans l'éditeur de texte.
7. Choisissez Créer une partie de la documentation. Vous pouvez créer de la documentation pour un corps de réponse non répertorié.
8. Si nécessaire, répétez ces étapes pour ajouter ou modifier une autre partie de la documentation.

Pour modifier une partie de la documentation existante, procédez comme suit :

1. Dans le volet Documentation, choisissez l'onglet Ressources et méthodes.
2. Vous pouvez sélectionner le corps de la réponse ou la ressource contenant le corps de la réponse, puis utiliser la barre de recherche pour rechercher et sélectionner la partie de votre documentation.
3. Choisissez Modifier.

Documentation d'une entité **MODEL**

La documentation d'une entité `MODEL` implique la création et la gestion d'instances `DocumentPart` pour le modèle et chacune des `properties` du modèle. Par exemple, le modèle `Error` qui est associé à chaque API par défaut a la définition de schéma suivante,

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

```
}
```

et exige deux instances `DocumentationPart`, une pour le `Model` et l'autre pour sa propriété `message` :

```
{
  "location": {
    "type": "MODEL",
    "name": "Error"
  },
  "properties": {
    "title": "Error Schema",
    "description": "A description of the Error model"
  }
}
```

et

```
{
  "location": {
    "type": "MODEL",
    "name": "Error.message"
  },
  "properties": {
    "description": "An error message."
  }
}
```

Lorsque l'API est exportée, les propriétés de `DocumentationPart` remplacent les valeurs dans le schéma d'origine.

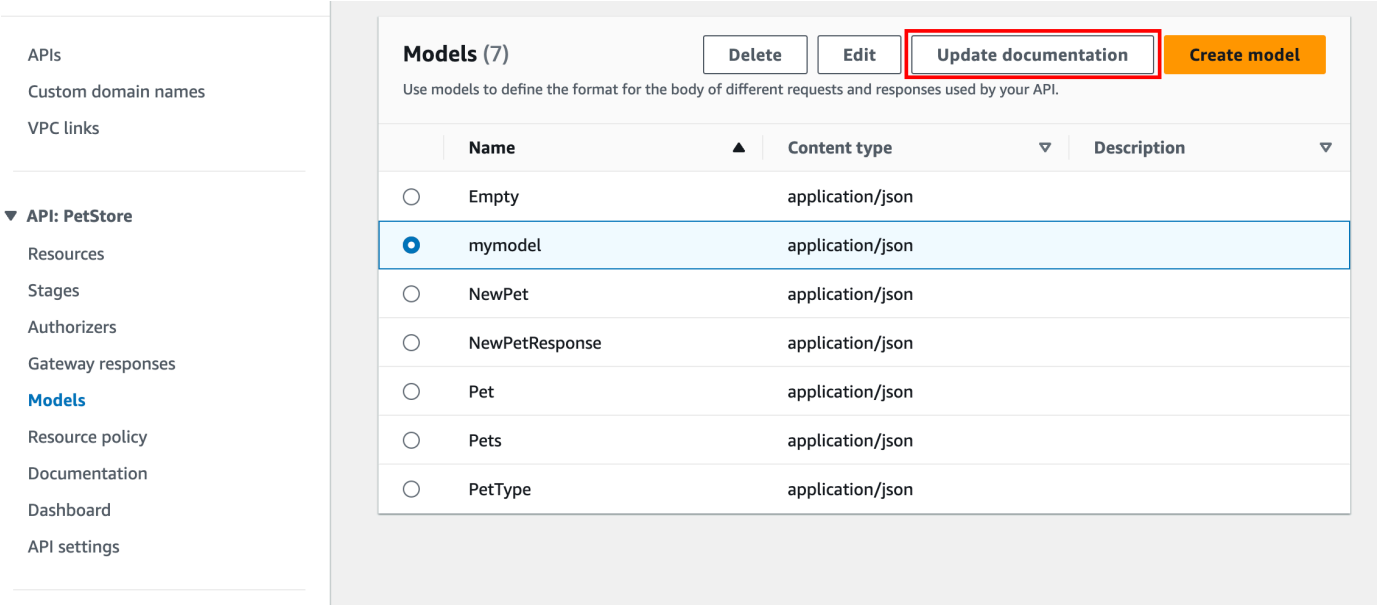
Pour ajouter une nouvelle partie de la documentation pour une entité `MODEL`, procédez comme suit :

1. Dans le volet de navigation principal, choisissez Documentation, puis Créer une partie de la documentation.
2. Pour Type de documentation, sélectionnez Modèle.
3. Pour Nom, entrez un nom pour le modèle.
4. Entrez une description dans l'éditeur de texte.
5. Choisissez Créer une partie de la documentation. Vous pouvez créer de la documentation pour des modèles non répertoriés.

6. Si nécessaire, répétez ces étapes pour ajouter ou modifier une partie de la documentation à d'autres modèles.

Pour ajouter une nouvelle partie de la documentation pour une entité `MODEL` dans le volet Modèles, procédez comme suit :

1. Dans le volet de navigation principal, choisissez Modèles.
2. Choisissez le modèle, puis Mettre à jour la documentation.



Pour modifier une partie de la documentation existante, procédez comme suit :

1. Dans le volet Documentation, choisissez l'onglet Modèles.
2. Utilisez la barre de recherche ou sélectionnez le modèle, puis choisissez Modifier.

Documentation d'une entité `AUTHORIZER`

Pour ajouter une nouvelle partie de la documentation pour une entité `AUTHORIZER`, procédez comme suit :

1. Dans le volet de navigation principal, choisissez Documentation, puis Créer une partie de la documentation.
2. Pour Type de documentation, sélectionnez Mécanisme d'autorisation.

3. Pour Nom, entrez le nom de votre mécanisme d'autorisation.
4. Entrez une description dans l'éditeur de texte. Spécifiez une valeur pour le champ `location` valide pour le mécanisme d'autorisation.
5. Choisissez Créer une partie de la documentation. Vous pouvez créer de la documentation pour des mécanismes d'autorisation non répertoriés.
6. Si nécessaire, répétez ces étapes pour ajouter ou modifier une partie de la documentation à d'autres mécanismes d'autorisation.

Pour modifier une partie de la documentation existante, procédez comme suit :

1. Dans le volet Documentation, choisissez l'onglet Mécanismes d'autorisation.
2. Utilisez la barre de recherche ou sélectionnez le mécanisme d'autorisation, puis choisissez Modifier.

Publication de la documentation de l'API à l'aide de la console API Gateway

La procédure suivante décrit la publication d'une version de la documentation.

Pour publier une version de la documentation à l'aide de la console API Gateway

1. Dans le volet de navigation principal, choisissez Documentation.
2. Choisissez Publier de la documentation.
3. Configurez la publication :
    a. Pour Étape, sélectionnez une étape.
    b. Pour Version, entrez un identifiant de version, par exemple `1.0.0`.
    c. (Facultatif) Sous Description, entrez une description.
4. Choisissez Publish.

Vous pouvez maintenant procéder au téléchargement de la documentation publiée en exportant la documentation dans un fichier OpenAPI externe. Pour en savoir plus, consultez la section the section called “Exportation d'une API REST”.

Documentation d'une API en utilisant l'API REST API Gateway

Dans cette section, nous décrivons la création et la gestion des parties de la documentation d'une API à l'aide de l'API REST API Gateway.

Avant de créer et de modifier la documentation d'une API, commencez par créer l'API. Dans cette section, nous utilisons l'PetStoreAPI comme exemple. Pour créer une API à l'aide de la console API Gateway, suivez les instructions de Tutoriel : Création d'une API REST par l'importation d'un exemple.

Rubriques

Documentation de l'entité **API**

Pour ajouter de la documentation pour une API, ajoutez une DocumentationPartressource pour l'entité API :

```
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}"
}
```

Si l'opération réussit, elle renvoie une réponse `201 Created` contenant l'instance `DocumentationPart` nouvellement créée dans la charge utile. Exemples :

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

Si la partie de la documentation a déjà été ajoutée, une réponse `409 Conflict` est renvoyée, avec le message d'erreur `Documentation part already exists for the specified location: type 'API'.`". Dans ce cas, vous devez appeler l'opération documentationpart:update.

```
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}"
  } ]
}
```

La réponse positive renvoie un code de statut `200 OK` avec la charge utile contenant l'instance `DocumentationPart` mise à jour dans la charge utile.

Documentation d'une entité **`RESOURCE`**

Pour ajouter de la documentation pour la ressource racine d'une API, ajoutez une DocumentationPartressource ciblée pour la ressource ressource correspondante :

```
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}"
}
```

Si l'opération réussit, elle renvoie une réponse `201 Created` contenant l'instance `DocumentationPart` nouvellement créée dans la charge utile. Exemples :

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

Lorsque le chemin d'accès à la ressource n'est pas spécifié, la ressource est supposée être la ressource racine. Vous pouvez ajouter `"path": "/"` à `properties` afin de rendre la spécification explicite.

Pour créer de la documentation pour une ressource enfant d'une API, ajoutez une DocumentationPartressource ciblée pour la ressource ressource correspondante :

```
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}"
```

```
}
```

Si l'opération réussit, elle renvoie une réponse `201 Created` contenant l'instance `DocumentationPart` nouvellement créée dans la charge utile. Par exemple :

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

Pour ajouter de la documentation pour une ressource enfant spécifiée par un paramètre de chemin, ajoutez une DocumentationPartressource ciblée pour la ressource ressource :

```
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}"
}
```

Si l'opération réussit, elle renvoie une réponse `201 Created` contenant l'instance `DocumentationPart` nouvellement créée dans la charge utile. Par exemple :

```
{
  "_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**
>
> L'DocumentationPartinstance d'une RESOURCE entité ne peut être héritée par aucune de ses ressources enfants.

Documentation d'une entité **METHOD**

Pour ajouter de la documentation pour une méthode d'une API, ajoutez une DocumentationPartressource ciblée pour la ressource de méthode correspondante :

```
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}"
}
```

Si l'opération réussit, elle renvoie une réponse `201 Created` contenant l'instance `DocumentationPart` nouvellement créée dans la charge utile. Exemples :

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

Si l'opération réussit, elle renvoie une réponse `201 Created` contenant l'instance `DocumentationPart` nouvellement créée dans la charge utile. Exemples :

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

Si le champ `location.method` n'est pas précisé dans la demande précédente, c'est la méthode ANY qui est utilisée et représentée par le caractère générique `*`.

Pour mettre à jour le contenu de la documentation d'une entité `METHOD`, appelez l'opération documentationpart:update en fournissant un nouveau mappage `properties` :

```
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}"
  } ]
}
```

La réponse positive renvoie un code de statut `200 OK` avec la charge utile contenant l'instance `DocumentationPart` mise à jour dans la charge utile. Exemples :

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

Documentation d'une entité **QUERY_PARAMETER**

Pour ajouter de la documentation pour un paramètre de requête, ajoutez une DocumentationPartressource ciblée pour le QUERY_PARAMETER type, avec les champs valides de path etname.

```
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}"
}
```

Si l'opération réussit, elle renvoie une réponse `201 Created` contenant l'instance `DocumentationPart` nouvellement créée dans la charge utile. Exemples :

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

Le mappage `properties` d'une partie de la documentation d'une entité `QUERY_PARAMETER` peut être hérité par l'une de ses entités enfants `QUERY_PARAMETER`. Par exemple, si vous ajoutez une ressource `treats` après `/pets/{petId}`, activez la méthode `GET` sur `/pets/{petId}/treats` et exposez le paramètre de demande `page`. Ce paramètre de demande enfant hérite pour `DocumentationPart` du mappage `properties` à partir du paramètre de demande du même nom

de la méthode `GET /pets`, sauf si vous ajoutez explicitement une ressource `DocumentationPart` au paramètre de demande `page` de la méthode `GET /pets/{petId}/treats`.

Documentation d'une entité **`PATH_PARAMETER`**

Pour ajouter de la documentation pour un paramètre de chemin, ajoutez une DocumentationPartressource pour l'PATH_PARAMETERentité.

```
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}"
}
```

Si l'opération réussit, elle renvoie une réponse `201 Created` contenant l'instance `DocumentationPart` nouvellement créée dans la charge utile. Exemples :

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

Documentation d'une entité **`REQUEST_BODY`**

Pour ajouter de la documentation pour le corps d'une demande, ajoutez une DocumentationPartressource pour le corps de la demande.

```
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}"
}
```

Si l'opération réussit, elle renvoie une réponse `201 Created` contenant l'instance `DocumentationPart` nouvellement créée dans la charge utile. Exemples :

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

Documentation d'une entité **REQUEST_HEADER**

Pour ajouter de la documentation pour un en-tête de demande, ajoutez une DocumentationPartressource pour l'en-tête de demande.

```
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}"
}
```

Si l'opération réussit, elle renvoie une réponse `201 Created` contenant l'instance `DocumentationPart` nouvellement créée dans la charge utile. Exemples :

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

Documentation d'une entité **RESPONSE**

Pour ajouter de la documentation pour une réponse à un code de statut, ajoutez une DocumentationPartressource ciblée pour la MethodResponseressource correspondante.

```
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}"
}
```

Si l'opération réussit, elle renvoie une réponse `201 Created` contenant l'instance `DocumentationPart` nouvellement créée dans la charge utile. Exemples :

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

Documentation d'une entité **`RESPONSE_HEADER`**

Pour ajouter de la documentation pour un en-tête de réponse, ajoutez une DocumentationPartressource pour l'en-tête de réponse.

```
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}"
```

Si l'opération réussit, elle renvoie une réponse `201 Created` contenant l'instance `DocumentationPart` nouvellement créée dans la charge utile. Exemples :

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

La documentation de cet en-tête de réponse `Content-Type` est la documentation par défaut des en-têtes `Content-Type` de toutes les réponses de l'API.

Documentation d'une entité **`AUTHORIZER`**

Pour ajouter de la documentation pour un autorisateur d'API, ajoutez une DocumentationPartressource ciblée pour l'autorisateur spécifié.

```
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}"
}
```

Si l'opération réussit, elle renvoie une réponse `201 Created` contenant l'instance `DocumentationPart` nouvellement créée dans la charge utile. Par exemple :


```
{
  "_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**
>
> L'DocumentationPartinstance d'une `AUTHORIZER` entité ne peut être héritée par aucune de ses ressources enfants.

Documentation d'une entité **`MODEL`**

La documentation d'une entité `MODEL` implique la création et la gestion d'instances `DocumentPart` pour le modèle et chacune des `properties` du modèle. Par exemple, le modèle `Error` qui est associé à chaque API par défaut a la définition de schéma suivante,

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

et exige deux instances `DocumentationPart`, une pour le `Model` et l'autre pour sa propriété `message` :

```
{
  "location": {
    "type": "MODEL",
    "name": "Error"
  },
  "properties": {
    "title": "Error Schema",
    "description": "A description of the Error model"
  }
}
```

et

```
{
  "location": {
    "type": "MODEL",
    "name": "Error.message"
  },
  "properties": {
    "description": "An error message."
  }
}
```

Lorsque l'API est exportée, les propriétés de `DocumentationPart` remplacent les valeurs dans le schéma d'origine.

Pour ajouter de la documentation pour un modèle d'API, ajoutez une DocumentationPartressource ciblée pour le modèle spécifié.

```
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}"
}
```

Si l'opération réussit, elle renvoie une réponse `201 Created` contenant l'instance `DocumentationPart` nouvellement créée dans la charge utile. Par exemple :

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

Répétez la même étape pour créer une DocumentationPart instance pour l'une des propriétés du modèle.

> **Note**
>
> L'DocumentationPartinstance d'une `MODEL` entité ne peut être héritée par aucune de ses ressources enfants.

Mise à jour de certaines parties de la documentation

Pour mettre à jour les parties de documentation de tout type d'entité API, soumettez une requête PATCH sur une DocumentationPartinstance d'un identifiant de pièce spécifié afin de remplacer la `properties` carte existante par une nouvelle.

```
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"
  } ]
}
```

La réponse positive renvoie un code de statut `200 OK` avec la charge utile contenant l'instance `DocumentationPart` mise à jour dans la charge utile.

Vous pouvez mettre à jour plusieurs parties de la documentation dans une seule demande `PATCH`.

Liste des parties de la documentation

Pour répertorier les parties de documentation de tout type d'entité d'API, soumettez une requête GET sur une DocumentationPartscollection.

```
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
```

La réponse positive renvoie un code de statut `200 OK` avec la charge utile contenant les instances `DocumentationPart` disponibles dans la charge utile.

Publication de la documentation de l'API à l'aide de l'API REST

Pour publier la documentation d'une API, créez, mettez à jour ou obtenez l'instantané de la documentation, puis associez-le à une étape de l'API. Lorsque vous créez un instantané de documentation, vous pouvez également l'associer à une étape de l'API en même temps.

Rubriques

- Création d'un instantané de la documentation et association à une étape de l'API
- Création d'un instantané de documentation
- Mise à jour d'un instantané de documentation
- Obtention d'un instantané de documentation
- Association d'un instantané de documentation avec une étape d'API
- Téléchargement d'un instantané de documentation associé à une étape

Création d'un instantané de la documentation et association à une étape de l'API

Pour créer un instantané de certaines parties de la documentation de l'API et l'associer à une étape de l'API en même temps, soumettez la demande `POST` suivante :

```
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"
}
```

Si l'opération réussit, elle renvoie une réponse `200 OK`, contenant l'instance `DocumentationVersion` nouvellement créée en tant que charge utile.

Sinon, vous pouvez créer un instantané de la documentation sans l'associer à une étape de l'API en premier lieu, puis appeler restapi:update pour associer cet instantané à une étape de l'API. Vous pouvez également mettre à jour ou interroger un instantané de documentation existant, puis mettre à jour son association à une étape donnée. Nous présentons les étapes correspondantes au cours des quatre sections suivantes.

Création d'un instantané de documentation

Pour créer un instantané des parties de la documentation d'une API, créez une nouvelle DocumentationVersionressource et ajoutez-la à la DocumentationVersionscollection de l'API :

```
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"
}
```

Si l'opération réussit, elle renvoie une réponse `200 OK`, contenant l'instance `DocumentationVersion` nouvellement créée en tant que charge utile.

Mise à jour d'un instantané de documentation

Vous ne pouvez mettre à jour un instantané de documentation qu'en modifiant la `description` propriété de la DocumentationVersionressource correspondante. L'exemple suivant montre comment mettre à jour la description de l'instantané de la documentation tel qu'identifié par son identifiant de version, *`version`*, par exemple, `1.0.0`.

```
PATCH /restapis/restapi_id/documentation/versions/version HTTP/1.1
Host: apigateway.region.amazonaws.com
Content-Type: application/json
X-Amz-Date: YYYYMMDDTttttttZ
Authorization: AWS4-HMAC-SHA256 Credential=access_key_id/YYYYMMDD/region/
apigateway/aws4_request, SignedHeaders=content-length;content-type;host;x-amz-date,
 Signature=sigv4_secret

{
    "patchOperations": [{
        "op": "replace",
        "path": "/description",
        "value": "My API for testing purposes."
    }]
}
```

Si l'opération réussit, elle renvoie une réponse `200 OK`, contenant l'instance `DocumentationVersion` mise à jour en tant que charge utile.

Obtention d'un instantané de documentation

Pour obtenir un instantané de la documentation, soumettez une `GET` demande pour la DocumentationVersionressource spécifiée. L'exemple suivant montre comment obtenir un instantané de documentation d'un identifiant de version donné, 1.0.0.

```
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
```

Association d'un instantané de documentation avec une étape d'API

Pour publier la documentation d'une API, associez un instantané de documentation à une étape de l'API. Vous devez déjà avoir déjà une étape de l'API avant d'associer la version de la documentation à l'étape.

Pour associer un instantané de documentation à une étape d'API à l'aide de l'API REST API Gateway, appelez l'opération stage:update afin de définir la version souhaitée de la documentation sur la propriété `stage.documentationVersion` :

```
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"
    }]
}
```

Téléchargement d'un instantané de documentation associé à une étape

Une fois qu'une version des parties de la documentation est associée à une étape, vous pouvez exporter les parties de la documentation avec les définitions d'entités API vers un fichier externe à l'aide de la console API Gateway, de l'API REST API Gateway, de l'un de ses kits SDK ou de la AWS CLI pour API Gateway. Le processus est le même que pour l'exportation de l'API. Le format de fichier exporté peut être JSON ou YAML.

À l'aide de l'API REST API Gateway, vous pouvez aussi définir explicitement le paramètre de requête `extension=documentation,integrations,authorizers` afin d'inclure les parties de la documentation de l'API, les intégrations de l'API et les mécanismes d'autorisation dans le cadre d'une exportation de l'API. Par défaut, les parties de la documentation sont incluses, mais les intégrations et les mécanismes d'autorisation sont exclus, lorsque vous exportez une API. La sortie par défaut d'une exportation d'API convient à la distribution de la documentation.

Pour exporter la documentation de l'API dans un fichier OpenAPI JSON externe en utilisant l'API REST API Gateway, envoyez la demande GET suivante :

```
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
```

Ici, l'objet `x-amazon-apigateway-documentation` contient les parties de la documentation et les définitions d'entités d'API contiennent les propriétés de la documentation prises en charge par OpenAPI. Le résultat n'inclut pas les détails de l'intégration ou des mécanismes d'autorisation Lambda (anciennement appelés mécanismes d'autorisation personnalisés). Pour inclure les deux détails, définissez `extensions=integrations,authorizers,documentation`. Pour inclure les détails des intégrations, mais pas les mécanismes d'autorisation, définissez `extensions=integrations,documentation`.

Vous devez définir l'en-tête `Accept:application/json` dans la demande pour faire sortir le résultat dans un fichier JSON. Pour produire la sortie YAML, modifiez l'en-tête de la requête sur `Accept:application/yaml`.

A titre d'exemple, nous allons étudier une API qui expose une méthode `GET` simple sur la ressource racine (`/`). Cette API a quatre entités d'API définies dans un fichier de définition OpenAPI, une pour chaque type `API`, `MODEL`, `METHOD` et `RESPONSE`. Une partie de la documentation a été ajoutée à chaque entité `API`, `METHOD` et `RESPONSE`. En appelant la commande d'exportation de documentation précédente, nous obtenons la sortie suivante, avec les parties de la documentation répertoriées dans l'objet `x-amazon-apigateway-documentation` sous forme d'extension d'un fichier OpenAPI standard.

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"
}
```

Pour un attribut compatible à OpenAPI défini dans le mappage `properties` d'une partie de la documentation, API Gateway insère l'attribut dans la définition d'entité API associée. Un attribut `x-`*`something`* est une extension OpenAPI standard. Cette extension est appliquée à la définition de l'entité d'API. Par exemple, consultez l'attribut `x-example` pour la méthode `GET`. Un attribut tel que `foo` ne fait pas partie de la spécification OpenAPI et n'est pas injecté dans les définitions de l'entité d'API associées.

Si un outil de rendu de documentation (par exemple, OpenAPI UI) analyse les définitions de l'entité d'API afin d'extraire les attributs de la documentation, tout attribut `properties` non conforme à OpenAPI d'une instance `DocumentationPart` n'est pas disponible pour l'outil. Toutefois, si un outil de rendu de documentation analyse l'objet `x-amazon-apigateway-documentation` pour obtenir le contenu ou si l'outil appelle restapi:documentation-parts et documentationpart:by-id pour récupérer des parties de la documentation API Gateway, tous les attributs de documentation sont disponibles et peuvent être affichés par l'outil.

Pour exporter la documentation avec les définitions d'entité d'API contenant les détails de l'intégration dans un fichier JSON OpenAPI, envoyez la requête `GET` suivante :

```
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
```

Pour exporter la documentation avec les définitions d'entité d'API contenant les détails d'intégration et les mécanismes d'autorisation dans un fichier OpenAPI YAML, envoyez la requête GET suivante :

```
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
```

Pour utiliser la console API Gateway afin d'exporter et de télécharger la documentation publiée d'une API, suivez les instructions de Exportation d'une API REST à l'aide de la console API Gateway.

Importation de la documentation d'une API

Comme avec l'importation des définitions d'entités d'API, vous pouvez importer les parties de la documentation d'un fichier OpenAPI externe vers une API dans API Gateway. Vous spécifiez les parties de la to-be-imported documentation au sein de l'x-amazon-apigateway-documentation objetextension dans un fichier de définition OpenAPI valide. L'importation de la documentation ne modifie pas les définitions d'entités d'API existantes.

Vous avez la possibilité de fusionner les parties de la documentation nouvellement spécifiées avec les parties de la documentation existantes dans API Gateway ou de remplacer les parties existantes de la documentation. En mode `MERGE`, une nouvelle partie de la documentation définie dans le fichier OpenAPI est ajoutée à la collection `DocumentationParts` de l'API. Si une `DocumentationPart` importée existe déjà, un attribut importé remplace celui qui existe déjà si les deux sont différents. Les autres attributs de documentation existants ne sont pas modifiés. En mode `OVERWRITE`, toute la collection `DocumentationParts` est remplacée selon le fichier de définition OpenAPI importé.

Importation de parties de la documentation à l'aide de l'API REST API Gateway

Pour importer la documentation de l'API en utilisant l'API REST API Gateway, appelez l'opération documentationpart:import. L'exemple suivant montre comment remplacer les parties de

documentation existantes d'une API par une seule méthode `GET /` , en renvoyant une réponse `200 OK` en cas de réussite.

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"
        }
      }
    ]
  }
}
```

En cas de réussite, cette demande renvoie une réponse 200 OK contenant la `DocumentationPartId` importée dans la charge utile.

```
{
  "ids": [
    "kg3mth",
    "796rtf",
    "zhek4p",
    "5ukm9s"
  ]
}
```

En outre, vous pouvez également appeler restapi:import ou restapi:put en fournissant les parties de la documentation dans l'objet `x-amazon-apigateway-documentation`, dans le cadre du fichier d'entrée OpenAPI de la définition d'API. Pour exclure les parties de la documentation de l'importation de l'API, définissez `ignore=documentation` dans les paramètres de la demande.

Importation de parties de la documentation avec la console API Gateway

Les instructions suivantes expliquent comment importer des parties de la documentation.

Pour utiliser la console afin d'importer des parties de la documentation d'une API à partir d'un fichier externe

1. Dans le volet de navigation principal, choisissez Documentation.
2. Choisissez Import (Importer).
3. Si vous avez de la documentation existante, sélectionnez Remplacer ou Fusionner votre nouvelle documentation.
4. Choisissez Choisir un fichier pour charger un fichier à partir d'un lecteur ou entrez le contenu d'un fichier dans l'affichage de fichier. Pour obtenir un exemple, consultez la charge utile de l'exemple de demande dans Importation de parties de la documentation à l'aide de l'API REST API Gateway.
5. Choisissez comment gérer les avertissements lors de l'importation. Sélectionnez Échouer avec les avertissements ou Ignorer les avertissements. Pour plus d'informations, consultez the section called “Erreurs et avertissements pendant l'importation”.
6. Choisissez Importer.

Contrôle de l'accès à la documentation de l'API

Si vous avez une équipe de documentation dédiée pour écrire et modifier la documentation de l'API, vous pouvez configurer des autorisations d'accès distinctes pour les développeurs (pour le développement de l'API) et pour les rédacteurs ou éditeurs (pour le développement du contenu). Cela s'applique notamment lorsqu'un fournisseur tiers est impliqué dans la création de la documentation pour vous.

Pour autoriser votre équipe de documentation à créer, mettre à jour et publier la documentation de votre API, vous pouvez attribuer à l'équipe de documentation un rôle IAM avec la politique IAM suivante, où *`account_id est l'ID`* de AWS compte de votre équipe de documentation.

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

Pour plus d'informations sur la définition des autorisations d'accès aux ressources API Gateway, consultez the section called “Fonctionnement d'Amazon API Gateway avec IAM”.

Génération d'un kit SDK pour une API REST dans API Gateway

Pour appeler votre API REST de façon spécifique à la plateforme ou au langage, vous devez générer un kit SDK spécifique à la plateforme et au langage pour l'API. Vous générez votre SDK après avoir créé, testé et déployé votre API sur une étape. Actuellement, API Gateway prend en charge la génération d'un SDK pour une API en Java JavaScript, Java pour Android, Objective-C ou Swift pour iOS et Ruby.

Cette section explique comment générer un SDK d'une API API Gateway. Il explique également comment utiliser le SDK généré dans une application Java, une application Java pour Android, des applications Objective-C et Swift pour iOS et une application. JavaScript

Pour faciliter la discussion, nous utilisons cette API API Gateway , qui expose cette fonction Lambda Simple Calculator.

Avant de continuer, créez ou importez l'API, puis déployez-la au moins une fois dans API Gateway. Pour obtenir des instructions, consultez Déploiement d'une API REST dans Amazon API Gateway.

Rubriques

- Fonction Lambda de calculatrice simple
- API de calculatrice simple dans API Gateway
- Définition OpenAPI d'API de calculatrice simple
- Génération du kit SDK Java d'une API
- Génération du kit SDK Android d'une API
- Génération du kit SDK iOS d'une API

- Génération du JavaScript SDK d'une API REST
- Génération du kit SDK Ruby d'une API
- Génération de SDK pour une API à l'aide AWS CLI de commandes

Fonction Lambda de calculatrice simple

A titre d'illustration, nous utilisons une fonction Lambda dans Node.js qui effectue des opérations binaires d'addition, de soustraction, de multiplication et de division.

Rubriques

- Format d'entrée de la fonction Lambda de calculatrice simple
- Format de sortie de la fonction Lambda de calculatrice simple
- Implémentation de la fonction Lambda de calculatrice simple

Format d'entrée de la fonction Lambda de calculatrice simple

La fonction utilise une entrée au format suivant :

```
{ "a": "Number", "b": "Number", "op": "string"}
```

où op peut avoir l'une des valeurs suivantes : `(+, -, *, /, add, sub, mul, div)`.

Format de sortie de la fonction Lambda de calculatrice simple

Lorsqu'une opération aboutit, elle renvoie le résultat au format suivant :

```
{ "a": "Number", "b": "Number", "op": "string", "c": "Number"}
```

où c contient le résultat du calcul.

Implémentation de la fonction Lambda de calculatrice simple

L'implémentation de la fonction Lambda est la suivante :

```
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 de calculatrice simple dans API Gateway

Notre API de calculatrice simple expose trois méthodes (GET, POST, GET) pour appeler la fonction the section called “Fonction Lambda de calculatrice simple”. Une représentation graphique de cette API se présente comme suit :


Resources
Create resource
/
GET
POST
/{a}
ANY
/{b}
ANY
/{op}
GET

Ces trois méthodes décrivent les différentes façons de fournir l'entrée à la fonction Lambda backend pour qu'elle effectue la même opération :

- La méthode `GET /?a=...&b=...&op=...` utilise les paramètres de requête pour spécifier l'entrée.
- La méthode `POST /` utilise une charge utile JSON `{"a":"Number", "b":"Number", "op":"string"}` pour spécifier l'entrée.
- La méthode `GET /{a}/{b}/{op}` utilise les paramètres de chemin pour spécifier l'entrée.

Si celle-ci n'est pas définie, API Gateway génère le nom de méthode du kit SDK correspondant en combinant les parties de la méthode et du chemin d'accès HTTP. La partie du chemin d'accès racine (`/`) est appelé `Api Root`. Par exemple, le nom de la méthode du kit SDK Java par défaut pour la méthode d'API de `GET /?a=...&b=...&op=...` est `getAB0p`, le nom de la méthode du kit SDK par défaut de `POST /` est `postApiRoot`, et le nom de la méthode du kit SDK par défaut de `GET /{a}/{b}/{op}` est `getAB0p`. Des kits SDK individuels peuvent personnaliser la convention. Consultez la documentation dans la source du kit SDK généré pour connaître les noms de méthode spécifiques aux kits SDK.

Vous pouvez, et devez, remplacer les noms de méthode du kit SDK par défaut en définissant la propriété operationName sur chaque méthode de l'API. Vous pouvez le faire lors de la création de la méthode d'API ou de la mise à jour de la méthode d'API à l'aide de l'API REST API Gateway. Dans la définition Swagger d'API, vous pouvez définir `operationId` pour obtenir le même résultat.

Avant de montrer comment appeler ces méthodes à l'aide d'un kit SDK généré par API Gateway pour cette API, rappelons rapidement comment les configurer. Pour obtenir des instructions complètes, veuillez consulter Développement d'une API REST dans API Gateway. Si vous débutez avec API Gateway, veuillez d'abord consulter Choisissez un didacticiel AWS Lambda d'intégration.

Création des modèles d'entrée et de sortie

Pour spécifier une entrée fortement typée dans le kit SDK, nous créons un modèle `Input` pour l'API : Pour décrire le type de données du corps de la réponse, nous créons un modèle `Output` et un modèle `Result`.

Pour créer des modèles pour l'entrée, la sortie et le résultat

1. Dans le volet de navigation principal, choisissez Modèles.
2. Sélectionnez Create model.

3. Pour Name (Nom), saisissez **`input`**.
4. Pour Type de contenu, entrez **`application/json`**.

   Si aucun type de contenu correspondant n'est trouvé, la validation de demande n'est pas effectuée. Pour utiliser le même modèle quel que soit le type de contenu, saisissez **`$default`**.
5. Pour Schéma du modèle, saisissez le modèle qui suit :

```
{
    "$schema" : "$schema": "http://json-schema.org/draft-04/schema#",
    "type":"object",
    "properties":{
        "a":{"type":"number"},
        "b":{"type":"number"},
        "op":{"type":"string"}
    },
    "title":"Input"
}
```

6. Sélectionnez Create model.
7. Répétez les étapes suivantes pour créer un modèle `Output` et un modèle `Result`.

   Pour le modèle `Output`, saisissez ce qui suit pour Schéma du modèle :

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "type": "object",
    "properties": {
        "c": {"type":"number"}
    },
    "title": "Output"
}
```

   Pour le modèle `Result`, saisissez ce qui suit pour Schéma du modèle : Remplacez l'ID d'API `abc123` par votre ID d'API.

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "type":"object",
    "properties":{
        "input":{
            "$ref":"https://apigateway.amazonaws.com/restapis/abc123/models/Input"
```

```
        },
        "output":{
            "$ref":"https://apigateway.amazonaws.com/restapis/abc123/models/Output"
        }
    },
    "title":"Result"
}
```

Configuration des paramètres de requête de la méthode GET /

Pour la méthode `GET /?a=..&b=..&op=..`, les paramètres de requête sont déclarés dans Method Request :

Pour configurer les paramètres de chaîne de requête GET / URL

1. Dans la section Requête de méthode pour la méthode `GET` sur la ressource racine (`/`), choisissez Modifier.
2. Choisissez Paramètres de chaîne de requête d'URL et procédez comme suit :

   a. Sélectionnez Add query string (Ajouter une chaîne de requêtes).
   b. Pour Name (Nom), saisissez **a**.
   c. Gardez Obligatoire et Mise en cache désactivés.
   d. Maintenez Mise en cache désactivée.

   Répétez les mêmes étapes et créez une chaîne de requête nommée **b** et une chaîne de requête nommée **op**.
3. Choisissez Enregistrer.

Configuration du modèle de données pour la charge utile en tant qu'entrée du backend

Pour la méthode `POST /`, nous créons le modèle `Input` et l'ajoutons à la méthode de demande pour définir la forme des données d'entrée.

Pour configurer le modèle de données pour la charge utile en tant qu'entrée du backend

1. Dans la section Requête de méthode pour la méthode `POST` sur la ressource racine (`/`), choisissez Modifier.
2. Choisissez Corps de la requête.

3. Choisissez Add model.
4. Pour Type de contenu, entrez **`application/json`**.
5. Pour Modèle, sélectionnez Entrée.
6. Choisissez Enregistrer.

Avec ce modèle, vos clients d'API peuvent appeler le kit SDK pour spécifier l'entrée en instanciant un objet `Input`. Sans ce modèle, vos clients devront créer un objet de dictionnaire pour représenter l'entrée JSON à la fonction Lambda.

Configuration du modèle de données pour la sortie Résultat depuis le backend

Pour les trois méthodes, nous créons le modèle `Result` et l'ajoutons à la `Method Response` de la méthode pour définir la forme de sortie renvoyée par la fonction Lambda.

Pour configurer le modèle de données pour la sortie Résultat depuis le backend

1. Sélectionnez la ressource/{a}/{b}/{op}, puis choisissez la méthode GET.
2. Dans l'onglet Méthode de réponse, sous Réponse 200, choisissez Modifier.
3. Sous Corps de la réponse, choisissez Ajouter un modèle.
4. Pour Type de contenu, entrez **`application/json`**.
5. Pour Modèle, sélectionnez Résultat.
6. Choisissez Enregistrer.

Avec ce modèle, vos clients d'API peuvent analyser une sortie positive par la lecture des propriétés d'un objet `Result`. Sans ce modèle, les clients devront créer un objet de dictionnaire pour représenter la sortie JSON.

Définition OpenAPI d'API de calculatrice simple

Voici une définition OpenAPI d'API de calculatrice simple. Vous pouvez l'importer dans votre compte. Toutefois, vous devez réinitialiser les autorisations basées sur les ressources sur la fonction Lambda après l'importation. Pour ce faire, sélectionnez une nouvelle fois la fonction Lambda que vous avez créée dans votre compte depuis Integration Request (Demande d'intégration) dans la console API Gateway. La console API Gateway est alors contrainte de réinitialiser les autorisations requises. Vous pouvez également utiliser AWS Command Line Interface pour la commande Lambda add-permission.

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"
          }
        }
      }
    }
  }
}
```

Génération du kit SDK Java d'une API

Pour générer le kit SDK Java d'une API dans API Gateway

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Choisissez une API REST.
3. Choisissez Stages (Étapes).
4. Dans le volet Étapes, sélectionnez le nom de l'étape.
5. Ouvrez le menu Actions d'étape, puis choisissez Générer un kit SDK.
6. Pour Plateforme, choisissez la plateforme Java et procédez comme suit :

    a. Pour Nom du service, précisez le nom de votre kit de développement logiciel. Par exemple, **`SimpleCalcSdk`**. Ce nom devient celui de la classe client de votre kit SDK. Le nom

correspond à la balise <name> sous <project> dans le fichier pom.xml, qui se trouve dans le dossier de projet du kit SDK. N'incluez pas les traits d'union.

b. Pour Nom du package Java, spécifiez un nom de package pour votre kit SDK. Par exemple, **examples.aws.apig.simpleCalc.sdk**. Le nom de package est utilisé comme espace de noms de votre bibliothèque SDK. N'incluez pas les traits d'union.

c. Pour Système de génération Java, entrez **maven** ou **gradle** afin de spécifier le système de génération.

d. Pour Id de groupe Java, entrez un identificateur de groupe pour votre projet de kit SDK. Par exemple, entrez **my-apig-api-examples**. Cet identifiant correspond à la balise <groupId> sous <project> dans le fichier pom.xml, qui se trouve dans le dossier projet du SDK.

e. Pour Id d'artefact Java, entrez un identificateur d'artefact pour votre projet de kit SDK. Par exemple, entrez **simple-calc-sdk**. Cet identifiant correspond à la balise <artifactId> sous <project> dans le fichier pom.xml, qui se trouve dans le dossier projet du SDK.

f. Pour Version d'artefact Java, entrez une chaîne d'identification de version. Par exemple, **1.0.0**. Cet identifiant de version correspond à la balise <version> sous <project> dans le fichier pom.xml, qui se trouve dans le dossier projet du kit SDK.

g. Pour Texte de licence du code source, entrez le texte de la licence de votre code source, le cas échéant.

7. Choisissez Generate SDK (Générer un kit SDK), puis suivez les instructions à l'écran pour télécharger le kit SDK généré par API Gateway.

Pour utiliser le kit SDK généré, suivez les instructions de Utilisation d'un kit SDK Java généré par API Gateway pour une API REST.

Chaque fois que vous mettez à jour une API, vous devez la redéployer et générer à nouveau le kit SDK pour que les mises à jour soient incluses.

Génération du kit SDK Android d'une API

Pour générer le kit SDK Android d'une API dans API Gateway

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Choisissez une API REST.
3. Choisissez Stages (Étapes).

4. Dans le volet Étapes, sélectionnez le nom de l'étape.
5. Ouvrez le menu Actions d'étape, puis choisissez Générer un kit SDK.
6. Pour Plateforme, choisissez la plateforme Android et procédez comme suit :

    a. Dans le champ Group ID (ID de groupe), saisissez l'identifiant unique du projet correspondant. Cette valeur est utilisée dans le fichier `pom.xml` (par exemple, **`com.mycompany`**).

    b. Dans le champ Invoker package (Package du mécanisme d'appel), entrez l'espace de noms des classes de client générées (par exemple, **`com.mycompany.clientsdk`**).

    c. Dans le champ Artifact ID (ID d'artefact), saisissez le nom du fichier .jar compilé sans la version. Cette valeur est utilisée dans le fichier `pom.xml` (par exemple, **`aws-apigateway-api-sdk`**).

    d. Dans le champ Artifact version (Version de l'artefact), entrez le numéro de version d'artefact du client généré. Cette valeur est utilisée dans le fichier `pom.xml` et doit suivre le modèle *`major.minor.patch`* (par exemple, **`1.0.0`**).
7. Choisissez Generate SDK (Générer un kit SDK), puis suivez les instructions à l'écran pour télécharger le kit SDK généré par API Gateway.

Pour utiliser le kit SDK généré, suivez les instructions de Utilisation d'un kit SDK Android généré par API Gateway pour une API REST.

Chaque fois que vous mettez à jour une API, vous devez la redéployer et générer à nouveau le kit SDK pour que les mises à jour soient incluses.

Génération du kit SDK iOS d'une API

Pour générer le kit SDK iOS d'une API dans API Gateway

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Choisissez une API REST.
3. Choisissez Stages (Étapes).
4. Dans le volet Étapes, sélectionnez le nom de l'étape.
5. Ouvrez le menu Actions d'étape, puis choisissez Générer un kit SDK.
6. Pour Plateforme, choisissez la plateforme iOS (Objective-C) ou iOS (Swift) et procédez comme suit :

- Dans la zone Prefix, tapez un préfixe unique.

  L'effet du préfixe est le suivant : si vous attribuez, par exemple, **`SIMPLE_CALC`** comme préfixe au SDK de l'SimpleCalcAPI avec`input`, et des `result` modèles`output`, le SDK généré contiendra la `SIMPLE_CALCSimpleCalcClient` classe qui encapsule l'API, y compris les demandes/réponses de méthode. En outre, le kit SDK généré contient les classes `SIMPLE_CALCinput`, `SIMPLE_CALCoutput` et `SIMPLE_CALCresult` pour représenter l'entrée, la sortie et les résultats, respectivement, pour représenter l'entrée de demande et la sortie de réponse. Pour de plus amples informations, veuillez consulter Utilisation d'un kit SDK iOS généré par API Gateway pour une API REST dans Objective-C ou Swift.

7. Choisissez Generate SDK (Générer un kit SDK), puis suivez les instructions à l'écran pour télécharger le kit SDK généré par API Gateway.

Pour utiliser le kit SDK généré, suivez les instructions de Utilisation d'un kit SDK iOS généré par API Gateway pour une API REST dans Objective-C ou Swift.

Chaque fois que vous mettez à jour une API, vous devez la redéployer et générer à nouveau le kit SDK pour que les mises à jour soient incluses.

Génération du JavaScript SDK d'une API REST

Pour générer le JavaScript SDK d'une API dans API Gateway

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Choisissez une API REST.
3. Choisissez Stages (Étapes).
4. Dans le volet Étapes, sélectionnez le nom de l'étape.
5. Ouvrez le menu Actions d'étape, puis choisissez Générer un kit SDK.
6. Pour Plateforme, choisissez la JavaScriptplateforme.
7. Choisissez Generate SDK (Générer un kit SDK), puis suivez les instructions à l'écran pour télécharger le kit SDK généré par API Gateway.

Pour utiliser le kit SDK généré, suivez les instructions de Utiliser un JavaScript SDK généré par API Gateway pour une API REST.

Chaque fois que vous mettez à jour une API, vous devez la redéployer et générer à nouveau le kit SDK pour que les mises à jour soient incluses.

Génération du kit SDK Ruby d'une API

Pour générer le kit SDK Ruby d'une API dans API Gateway

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Choisissez une API REST.
3. Choisissez Stages (Étapes).
4. Dans le volet Étapes, sélectionnez le nom de l'étape.
5. Ouvrez le menu Actions d'étape, puis choisissez Générer un kit SDK.
6. Pour Plateforme, choisissez la plateforme Ruby et procédez comme suit :
    a. Pour Nom du service, précisez le nom de votre kit de développement logiciel. Par exemple, **`SimpleCalc`**. Celui-ci sera utilisé pour générer l'espace de noms du gem Ruby de votre API. Ce nom doit être uniquement constitué de lettres, (`a-zA-Z`), sans aucun caractère spécial ni chiffre.
    b. Pour Ruby Gem Name, indiquez le nom du gem Ruby qui contiendra le code source du kit SDK généré pour votre API. Par défaut, il s'agit du nom de service, en minuscules, puis du suffixe `-sdk` (par exemple, **`simplecalc-sdk`**).
    c. Pour Ruby Gem Version, précisez un numéro de version pour le gem Ruby généré. Par défaut, l'attribut est défini sur `1.0.0`.
7. Choisissez Generate SDK (Générer un kit SDK), puis suivez les instructions à l'écran pour télécharger le kit SDK généré par API Gateway.

Pour utiliser le kit SDK généré, suivez les instructions de Utilisation d'un kit SDK Ruby généré par API Gateway pour une API REST.

Chaque fois que vous mettez à jour une API, vous devez la redéployer et générer à nouveau le kit SDK pour que les mises à jour soient incluses.

Génération de SDK pour une API à l'aide AWS CLI de commandes

Vous pouvez l'utiliser AWS CLI pour générer et télécharger le SDK d'une API pour une plate-forme prise en charge en appelant la commande get-sdk. Vous trouverez ci-dessous la procédure à suivre pour certaines de ces plateformes.

Rubriques

- Générez et téléchargez le SDK Java pour Android à l'aide du AWS CLI
- Générez et téléchargez le JavaScript SDK à l'aide du AWS CLI
- Générez et téléchargez le SDK Ruby à l'aide du AWS CLI

Générez et téléchargez le SDK Java pour Android à l'aide du AWS CLI

Pour générer et télécharger un kit SDK Java pour Android généré par API Gateway d'une API (`udpuvvzbkc`) à une étape donnée (`test`), appelez la commande comme suit :

```
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
```

La dernière entrée `~/apps/myApi/myApi-android-sdk.zip` correspond au chemin d'accès du fichier SDK téléchargé et dénommé `myApi-android-sdk.zip`.

Générez et téléchargez le JavaScript SDK à l'aide du AWS CLI

Pour générer et télécharger un JavaScript SDK généré par API Gateway d'une API (`udpuvvzbkc`) à un stade donné (`test`), appelez la commande comme suit :

```
aws apigateway get-sdk \
            --rest-api-id udpuvvzbkc \
            --stage-name test \
            --sdk-type javascript \
            ~/apps/myApi/myApi-js-sdk.zip
```

La dernière entrée `~/apps/myApi/myApi-js-sdk.zip` correspond au chemin d'accès du fichier SDK téléchargé et dénommé `myApi-js-sdk.zip`.

Générez et téléchargez le SDK Ruby à l'aide du AWS CLI

Pour générer et télécharger un kit SDK Ruby pour une API (`udpuvvzbkc`) à une étape donnée (`test`), appelez la commande comme suit :

```
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
```

La dernière entrée `~/apps/myApi/myApi-ruby-sdk.zip` correspond au chemin d'accès du fichier SDK téléchargé et dénommé `myApi-ruby-sdk.zip`.

Nous montrons ensuite comment utiliser le kit SDK généré pour appeler l'API sous-jacente. Pour plus d'informations, voir Appel d'une API REST dans Amazon API Gateway.

Vendez vos API API Gateway via AWS Marketplace

Après avoir créé, testé et déployé vos API, vous pouvez les intégrer dans un plan d'utilisation d'API Gateway et vendre le plan en tant que produit SaaS (Software as a Service) par le biais de ce plan AWS Marketplace. Les acheteurs d'API qui s'abonnent à votre offre de produits sont facturés en AWS Marketplace fonction du nombre de demandes effectuées dans le cadre du plan d'utilisation.

Pour vendre vos API AWS Marketplace, vous devez configurer le canal de vente afin de l'intégrer AWS Marketplace à API Gateway. D'une manière générale, cela implique de répertorier votre produit AWS Marketplace, de configurer un rôle IAM avec des politiques appropriées pour permettre à API Gateway d'envoyer des métriques d'utilisation AWS Marketplace, d'associer un AWS Marketplace produit à un plan d'utilisation d'API Gateway et d'associer un AWS Marketplace acheteur à une clé d'API API Gateway. Les détails sont présentés dans les sections suivantes.

Pour plus d'informations sur la vente de votre API en tant que produit SaaS sur AWS Marketplace, consultez le guide de AWS Marketplace l'utilisateur.

Rubriques

- Initialisation de l'intégration de AWS Marketplace avec API Gateway

- Gestion de l'abonnement des clients aux plans d'utilisation

Initialisation de l'intégration de AWS Marketplace avec API Gateway

Les tâches suivantes concernent l'initialisation unique de AWS Marketplace l'intégration avec API Gateway, qui vous permet de vendre vos API en tant que produit SaaS.

Répertorier un produit sur AWS Marketplace

Pour publier votre plan d'utilisation en tant que produit SaaS, soumettez un formulaire de chargement de produit via AWS Marketplace. Le produit doit comporter une dimension nommée `apigateway` du type `requests`. Cette dimension définit price-per-request et est utilisée par API Gateway pour mesurer les demandes adressées à vos API.

Création du rôle de mesure

Créez un rôle IAM nommé `ApiGatewayMarketplaceMeteringRole` avec la stratégie d'exécution et la stratégie d'approbation suivantes. Ce rôle permet à API Gateway d'envoyer des métriques d'utilisation AWS Marketplace en votre nom.

Stratégie d'exécution du rôle de mesure

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "aws-marketplace:BatchMeterUsage",
        "aws-marketplace:ResolveCustomer"
      ],
      "Resource": "*",
      "Effect": "Allow"
    }
  ]
}
```

Stratégie de la relation d'approbation du rôle de mesure

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

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

Associer le plan d'utilisation au AWS Marketplace produit

Lorsque vous mettez en vente un produit sur AWS Marketplace, vous recevez un code AWS Marketplace produit. Pour intégrer API Gateway AWS Marketplace, associez votre plan d'utilisation au code AWS Marketplace du produit. Vous activez l'association en définissant le `productCode`champ `UsagePlan` de l'API Gateway sur votre code AWS Marketplace produit, à l'aide de la console API Gateway, de l'API REST API Gateway, de l'API AWS CLI for API Gateway ou du AWS SDK pour API Gateway. L'exemple de code suivant utilise l'API REST API Gateway :

```
PATCH /usageplans/USAGE_PLAN_ID
Host: apigateway.region.amazonaws.com
Authorization: ...

{
    "patchOperations" : [{
        "path" : "/productCode",
        "value" : "MARKETPLACE_PRODUCT_CODE",
        "op" : "replace"
    }]
}
```

Gestion de l'abonnement des clients aux plans d'utilisation

Les tâches suivantes sont gérées par l'application portail développeur.

Lorsqu'un client s'abonne à votre produit via AWS Marketplace, AWS Marketplace transmet une `POST` demande à l'URL d'abonnement SaaS que vous avez enregistrée lors de la mise en vente de votre produit. AWS Marketplace La demande `POST` est associée à un paramètre `x-amzn-marketplace-token` contenant les informations de l'acheteur. Suivez les instructions fournies dans SaaS customer onboarding pour gérer cette redirection dans votre application de portail pour développeurs.

En réponse à la demande d'abonnement d'un client, AWS Marketplace envoie une `subscribe-success` notification à une rubrique Amazon SNS à laquelle vous pouvez vous abonner. (Veuillez consulter SaaS customer onboarding). Pour accepter la demande d'abonnement du client, vous gérez la `subscribe-success` notification en créant ou en récupérant une clé d'API API Gateway pour le client, en associant la clé fournie par le client AWS Marketplace`customerId` aux clés d'API, puis en ajoutant la clé d'API à votre plan d'utilisation.

Lorsque la demande d'abonnement du client se termine, l'application portail développeur doit présenter au client la clé d'API associée et informer celui-ci que la clé d'API doit être incluse dans l'en-tête `x-api-key` des demandes auprès des API.

Lorsqu'un client annule un abonnement à un plan d'utilisation, il AWS Marketplace envoie une `unsubscribe-success` notification à la rubrique SNS. Pour désinscrire le client, vous gérez la notification `unsubscribe-success` en supprimant les clés d'API du client du plan d'utilisation.

Autorisation d'accès à un plan d'utilisation

Pour autoriser un client spécifique à accéder à votre plan d'utilisation, utilisez l'API API Gateway afin d'extraire ou de créer une clé d'API pour le client et ajoutez celle-ci au plan d'utilisation.

L'exemple suivant montre comment appeler l'API REST API Gateway pour créer une nouvelle clé d'API avec une AWS Marketplace `customerId` valeur spécifique (*`MARKETPLACE_CUSTOMER_ID`*).

```
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"
}
```

L'exemple suivant montre comment obtenir une clé d'API avec une AWS Marketplace `customerId` valeur spécifique (*`MARKETPLACE_CUSTOMER_ID`*).

```
GET apikeys?customerId=MARKETPLACE_CUSTOMER_ID HTTP/1.1
Host: apigateway.region.amazonaws.com
Authorization: ...
```

Pour ajouter une clé d'API à un plan d'utilisation, créez une UsagePlanKey avec la clé d'API correspondant au plan d'utilisation approprié. L'exemple suivant montre comment effectuer cette opération à l'aide de l'API REST API Gateway, où `n371pt` correspond à l'ID du plan d'utilisation et `q5ugs7qjjh` un exemple de `keyId` d'API retourné des exemples précédents.

```
POST /usageplans/n371pt/keys HTTP/1.1
Host: apigateway.region.amazonaws.com
Authorization: ...

{
    "keyId": "q5ugs7qjjh",
    "keyType": "API_KEY"
}
```

Association d'un client à une clé d'API

Vous devez mettre à jour ApiKeyle `customerId` champ « s » avec le numéro AWS Marketplace client du client. Cette opération associe la clé d'API au client AWS Marketplace , ce qui permet d'effectuer les mesures et la facturation pour l'acheteur. L'exemple de code suivant appelle l'API REST API Gateway à cet effet.

```
PATCH /apikeys/q5ugs7qjjh
Host: apigateway.region.amazonaws.com
Authorization: ...

{
    "patchOperations" : [{
        "path" : "/customerId",
        "value" : "MARKETPLACE_CUSTOMER_ID",
        "op" : "replace"
    }]
}
```

Protection de votre API REST

API Gateway fournit un certain nombre de façons de protéger votre API contre certaines menaces, comme les utilisateurs malveillants ou les pics de trafic. Vous pouvez protéger votre API à l'aide de stratégies telles que la génération de certificats SSL, la configuration d'un pare-feu d'application web, la définition de cibles pour les limitations et l'accès à votre API uniquement à partir d'un Virtual Private Cloud (VPC). Dans cette section, vous pouvez apprendre à activer ces fonctionnalités à l'aide d'API Gateway.

Rubriques

Configuration de l'authentification TLS mutuelle pour une API REST

L'authentification TLS mutuelle nécessite une authentification bidirectionnelle entre le client et le serveur. Avec l'authentification TLS mutuelle, les clients doivent présenter des certificats X.509 pour vérifier leur identité afin d'accéder à votre API. Le protocole TLS mutuel est une exigence courante pour l'Internet des objets (IoT) et business-to-business les applications.

Vous pouvez utiliser l'authentification TLS mutuelle avec d'autres opérations d'autorisation et d'authentification prises en charge par API Gateway. API Gateway transmet les certificats que les clients fournissent aux mécanismes d'autorisation Lambda et aux intégrations de backend.

> **Important**
>
> Par défaut, les clients peuvent appeler votre API en utilisant le point de terminaison `execute-api` généré par API Gateway pour votre API. Pour vous assurer que les clients peuvent accéder à votre API en utilisant uniquement un nom de domaine personnalisé avec authentification TLS mutuelle, désactivez le point de terminaison par défaut `execute-api`. Pour en savoir plus, consultez la section the section called “Désactiver le point de terminaison par défaut”.

Rubriques

Conditions préalables pour l'authentification TLS mutuelle

Pour configurer des authentifications TLS mutuelles, vous avez besoin des éléments suivants :

- Un nom de domaine personnalisé
- Au moins un certificat configuré AWS Certificate Manager pour votre nom de domaine personnalisé
- Un magasin de confiance configuré et chargé vers Amazon S3

Noms de domaine personnalisés

Pour activer l'authentification TLS mutuelle pour une API REST, vous devez configurer un nom de domaine personnalisé pour votre API. Vous pouvez activer l'authentification TLS mutuelle pour un nom de domaine personnalisé, puis fournir le nom de domaine personnalisé aux clients. Pour accéder à une API à l'aide d'un nom de domaine personnalisé sur lequel l'authentification TLS mutuelle est activée, les clients doivent présenter des certificats approuvés dans les demandes d'API. Vous trouverez plus d'informations dans the section called “Noms de domaine personnalisés”.

Utilisation de certificats AWS Certificate Manager émis

Vous pouvez demander un certificat approuvé publiquement directement à partir d'ACM ou importer des certificats publics ou auto-signés. Pour configurer un certificat dans ACM, accédez à ACM. Si vous souhaitez importer un certificat, poursuivez votre lecture dans la section suivante.

Utilisation d'un AWS Private Certificate Authority certificat ou d'un produit importé

Pour utiliser un certificat importé dans ACM ou un certificat provenant AWS Private Certificate Authority d'un protocole TLS mutuel, API Gateway a besoin d'un certificat `ownershipVerificationCertificate` émis par ACM. Ce certificat de propriété est utilisé uniquement pour vérifier que vous disposez des autorisations nécessaires pour utiliser le nom de domaine. Il n'est pas utilisé pour la poignée de main TLS. Si vous n'avez pas encore de `ownershipVerificationCertificate`, accédez àhttps://console.aws.amazon.com/acm/ pour en configurer un.

Vous devrez conserver ce certificat valide pendant toute la durée de vie de votre nom de domaine. Si un certificat expire et que le renouvellement automatique échoue, toutes les mises à jour du nom de domaine seront verrouillées. Vous devrez mettre à jour le `ownershipVerificationCertificateArn`avec un `ownershipVerificationCertificate` valide avant de pouvoir effectuer d'autres modifications. Le `ownershipVerificationCertificate` ne peut pas être utilisé comme certificat de serveur pour un autre domaine TLS mutuel dans API Gateway. Si un certificat est directement réimporté dans ACM, le diffuseur doit rester le même.

Configuration de votre magasin de confiance

Les magasins de confiance sont des fichiers texte avec une `.pem` extension de fichier. Il s'agit d'une liste de certificats approuvés provenant des autorités de certification. Pour utiliser l'authentification TLS mutuelle, créez un magasin de confiance de certificats X.509 auxquels vous faites confiance pour accéder à votre API.

Vous devez inclure la chaîne de confiance complète à partir du certificat de l'autorité de certification émettrice jusqu'au certificat de l'autorité de certification racine, dans votre magasin de confiance. API Gateway accepte les certificats clients émis par toute autorité de certification présente dans la chaîne de confiance. Les certificats peuvent être délivrés par des autorités de certification publiques ou privées. Les certificats peuvent avoir une longueur de chaîne maximale de quatre. Vous pouvez également fournir des certificats auto-signés. Les algorithmes suivants sont pris en charge dans le truststore :

- SHA-256 ou plus
- RSA-2048 ou plus
- ECDSA-256 ou ECDSA-384

API Gateway valide un certain nombre de propriétés de certificat. Vous pouvez utiliser des mécanismes d'autorisation Lambda pour effectuer des vérifications supplémentaires lorsqu'un client appelle une API, notamment pour vérifier si un certificat a été révoqué. API Gateway valide les propriétés suivantes :

Validation	Description
Syntaxe X.509	Le certificat doit répondre aux exigences de syntaxe X.509.
Intégrité	Le contenu du certificat ne doit pas avoir été modifié par rapport à celui signé par l'autorité de certification à partir du magasin de confiance.
Validité	La période de validité du certificat doit en cours.
Chaînage de noms / chaînage de clés	Les noms et les objets des certificats doivent former une chaîne ininterrompue. Les certificats peuvent avoir une longueur de chaîne maximale de quatre.

Chargez le magasin de confiance dans un fichier unique d'un compartiment Amazon S3.

Un fichier .pem devrait ressembler à l'exemple suivant.

Example certificates.pem

```
-----BEGIN CERTIFICATE-----
<Certificate contents>
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
<Certificate contents>
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
<Certificate contents>
-----END CERTIFICATE-----
...
```

La AWS CLI commande suivante est chargée dans `certificates.pem` votre compartiment Amazon S3.

```
aws s3 cp certificates.pem s3://bucket-name
```

Votre compartiment Amazon S3 doit disposer d'une autorisation de lecture pour API Gateway afin de permettre à API Gateway d'accéder à votre truststore.

Configuration de l'authentification TLS mutuelle pour un nom de domaine personnalisé

Pour configurer le protocole TLS mutuel pour une API REST, vous devez utiliser un nom de domaine personnalisé régional pour votre API, avec une politique `TLS_1_2` de sécurité. Pour plus d'informations sur le choix d'une politique de sécurité, consultezthe section called “Choix d'une politique de sécurité”.

> **Note**
>
> L'authentification TLS mutuelle n'est pas prise en charge pour les API privées.

Une fois que vous avez chargé votre magasin de confiance dans Amazon S3, vous pouvez configurer votre nom de domaine personnalisé pour utiliser l'authentification TLS mutuelle. Collez le texte suivant (barres obliques incluses) dans un terminal :

```
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
```

Après avoir créé le nom de domaine, vous devrez configurer des enregistrements DNS et des mappages de chemin de base pour les opérations API. Pour en savoir plus, consultez la section Configuration d'un nom de domaine personnalisé régional dans API Gateway.

Appeler une API à l'aide d'un nom de domaine personnalisé qui nécessite une authentification TLS mutuelle

Pour appeler une API avec l'authentification TLS mutuelle activée, les clients doivent présenter un certificat approuvé dans la demande d'API. Lorsqu'un client tente d'appeler votre API, API Gateway recherche le diffuseur du certificat client dans votre magasin de confiance. Pour que l'API Gateway puisse poursuivre la demande, le diffuseur du certificat et la chaîne de confiance complète jusqu'au certificat d'autorité de certification racine doivent se trouver dans votre magasin de confiance.

L'exemple de commande `curl` suivant envoie à `api.example.com`, une demande qui inclut `my-cert.pem` dans la demande. `my-key.key` est la clé privée du certificat.

```
curl -v --key ./my-key.key --cert ./my-cert.pem api.example.com
```

Votre API est appelée uniquement si votre magasin de confiance approuve le certificat. Les conditions suivantes provoqueront l'échec de la poignée de main de l'API Gateway avec TLS et le refus de la demande d'un `403` code d'état. Si votre certificat :

- n'est pas approuvé
- a expiré
- n'utilise pas d'algorithme pris en charge

> **Note**
>
> L'API Gateway ne vérifie pas si un certificat a été révoqué.

Mise à jour de votre magasin de confiance

Pour mettre à jour les certificats dans votre magasin de confiance, chargez un nouveau lot de certificats dans Amazon S3. Vous pourrez ensuite mettre à jour votre nom de domaine personnalisé de manière à utiliser le certificat mis à jour.

Utilisez la gestion des versions Amazon S3 pour gérer plusieurs versions de votre magasin de confiance. Lorsque vous mettez à jour votre nom de domaine personnalisé de manière à utiliser une nouvelle version du magasin de confiance, API Gateway renvoie des avertissements si les certificats ne sont pas valides.

API Gateway ne génère des avertissements de certificat que lorsque vous mettez à jour votre nom de domaine. API Gateway ne vous avertit pas si un certificat précédemment chargé arrive à expiration.

La AWS CLI commande suivante met à jour un nom de domaine personnalisé afin d'utiliser une nouvelle version de Truststore.

```
aws apigateway update-domain-name \
    --domain-name api.example.com \
    --patch-operations op='replace',path='/mutualTlsAuthentication/
truststoreVersion',value='abcdef123'
```

Désactiver l'authentification TLS mutuelle

Pour désactiver l'authentification TLS mutuelle pour un nom de domaine personnalisé, supprimez le magasin de confiance de votre nom de domaine personnalisé, comme indiqué dans la commande suivante.

```
aws apigateway update-domain-name \
    --domain-name api.example.com \
    --patch-operations op='replace',path='/mutualTlsAuthentication/
truststoreUri',value=''
```

Résolution des problèmes liés aux avertissements de certificat

Lors de la création d'un nom de domaine personnalisé avec authentification TLS mutuelle, API Gateway renvoie des avertissements si les certificats du magasin de confiance ne sont pas valides. Cela peut également se produire lors de la mise à jour d'un nom de domaine personnalisé de manière à utiliser un nouveau magasin de confiance. Les avertissements indiquent le problème lié au certificat et l'objet du certificat ayant produit l'avertissement. L'authentification TLS mutuelle reste activée pour votre API, mais certains clients risquent de ne pas être en mesure d'accéder à votre API.

Vous devrez décoder les certificats de votre magasin de confiance pour identifier le certificat ayant émis l'avertissement. Vous pouvez utiliser des outils tels que `openssl` pour décoder les certificats et identifier leur objet.

La commande suivante affiche le contenu d'un certificat, y compris son objet.

```
openssl x509 -in certificate.crt -text -noout
```

Mettez à jour ou supprimez les certificats ayant produit des avertissements, puis chargez un nouveau magasin de confiance dans Amazon S3. Après avoir chargé le nouveau magasin de confiance, mettez à jour votre nom de domaine personnalisé de manière à utiliser le nouveau magasin de confiance.

Résolution des conflits de noms de domaine

L'erreur `"The certificate subject <certSubject> conflicts with an existing certificate from a different issuer."` signifie que plusieurs autorités de certification ont émis un certificat pour ce domaine. Pour chaque sujet du certificat, il ne peut y avoir qu'un seul diffuseur dans API Gateway pour les domaines TLS mutuels. Vous devrez obtenir tous vos certificats pour ce sujet par l'intermédiaire d'un seul diffuseur. Si le problème est dû à un certificat dont vous n'avez pas le contrôle, mais pour lequel vous pouvez prouver la propriété du nom de domaine, contactez AWS Support pour ouvrir un ticket.

Dépannage des messages d'état des noms de domaine

`PENDING_CERTIFICATE_REIMPORT` : cela signifie que vous avez réimporté un certificat dans ACM qui a provoqué l'échec de la validation, car le nouveau certificat a un SAN (nom alternatif de l'objet) qui n'est pas couvert par le `ownershipVerificationCertificate` ou par l'objet ou que les SAN du certificat ne couvrent pas le nom de domaine. Quelque chose peut être configuré de manière incorrecte ou un certificat non valide a été importé. Vous devez réimporter un certificat valide dans ACM. Pour en savoir plus sur la validation, consultez Validation de la propriété de domaine.

`PENDING_OWNERSHIP_VERIFICATION` : cela signifie que votre certificat précédemment vérifié a expiré et qu'ACM n'a pas pu le renouveler automatiquement. Vous devrez renouveler le certificat ou demander un nouveau certificat. Pour en savoir plus sur le renouvellement du certificat, vous trouverez des informations supplémentaires dans le guide : Résolution des problèmes liés au renouvellement des certificats gérés par ACM.

Génération et configuration d'un certificat SSL pour l'authentification backend

Vous pouvez utiliser API Gateway pour générer un certificat SSL et utiliser sa clé publique sur le backend afin de vérifier que les requêtes HTTP adressées à votre système backend proviennent d'API Gateway. Votre backend HTTP peut ainsi contrôler et accepter uniquement les demandes provenant d'Amazon API Gateway, même si le backend est accessible publiquement.

> **Note**
>
> Certains backends peuvent ne pas prendre en charge l'authentification de client SSL comme le fait API Gateway et peuvent renvoyer une erreur de certificat SSL. Pour obtenir la liste des serveurs backend non compatibles, consultez the section called “Remarques importantes”.

Les certificats SSL générés par API Gateway sont auto-signés et seule la clé publique d'un certificat est visible dans la console API Gateway ou via les API.

Rubriques

- Génération d'un certificat de client à l'aide de la console API Gateway
- Configuration d'une API pour utiliser les certificats SSL
- Appel de test pour vérifier la configuration de certificat client
- Configuration du serveur HTTPS backend pour vérifier le certificat client
- Rotation d'un certificat client arrivant à expiration
- Autorités de certification prises en charge par API Gateway pour HTTP et les intégrations de proxy HTTP

Génération d'un certificat de client à l'aide de la console API Gateway

1. Ouvrez la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Choisissez une API REST.
3. Dans le volet de navigation principal, choisissez Certificats de clients.
4. Sur la page Certificats de clients, choisissez Générer un certificat.
5. (Facultatif) Sous Description, entrez une description.
6. Choisissez Générer un certificat pour générer le certificat. API Gateway génère un nouveau certificat et renvoie le GUID du nouveau certificat, ainsi que la clé publique codée en PEM.

Vous êtes maintenant prêt à configurer une API pour utiliser le certificat.

Configuration d'une API pour utiliser les certificats SSL

Ces instructions supposent que vous avez déjà terminé Génération d'un certificat de client à l'aide de la console API Gateway.

1. Dans la console API Gateway, créez ou ouvrez une API pour laquelle vous souhaitez utiliser le certificat de client. Vérifiez que l'API a été déployée à une étape.
2. Dans le volet de navigation principal, choisissez Étapes.
3. Dans la section Détails de l'étape, choisissez Modifier.
4. Pour Certificat de client, sélectionnez un certificat.
5. Sélectionnez Enregistrer les modifications.

   Si l'API a déjà été déployée dans la console API Gateway, vous devez la redéployer pour que les changements prennent effet. Pour de plus amples informations, veuillez consulter the section called “Redéploiement d'une API REST jusqu'à une étape”.

Une fois qu'un certificat a été sélectionné pour l'API et enregistré, API Gateway l'utilise pour tous les appels aux intégrations HTTP de votre API.

Appel de test pour vérifier la configuration de certificat client

1. Sélectionnez une méthode d'API. Choisissez l’onglet Test. Vous devrez peut-être choisir la flèche droite pour afficher l'onglet Test.
2. Pour Certificat de client, sélectionnez un certificat.
3. Sélectionnez Tester).

API Gateway présente le certificat SSL choisi pour permettre au backend HTTP d'authentifier l'API.

Configuration du serveur HTTPS backend pour vérifier le certificat client

Ces instructions supposent que vous avez déjà terminé Génération d'un certificat de client à l'aide de la console API Gateway et téléchargé une copie du certificat client. Vous pouvez télécharger un certificat client en appelant la commande `clientcertificate:by-id` de l'API REST API Gateway ou la commande `get-client-certificate` de la AWS CLI.

Avant de configurer un serveur HTTPS backend pour vérifier le certificat SSL client d'API Gateway, vous devez avoir obtenu la clé privée codée PEM et un certificat côté serveur qui est fourni par une autorité de certification reconnue.

Si le nom de domaine du serveur est `myserver.mydomain.com`, la valeur CNAME du certificat du serveur doit être `myserver.mydomain.com` ou `*.mydomain.com`.

Les autorités de certification prises en charge comprennent Let's Encrypt ou l'une des the section called “Autorités de certification prises en charge pour HTTP et l'intégration de proxy HTTP”.

Par exemple, supposons que le fichier de certificat client est `apig-cert.pem` et que les fichiers de clé privée et de certificat du serveur sont `server-key.pem` et `server-cert.pem`, respectivement. Pour un serveur Node.js dans le backend, vous pouvez configurer le serveur comme suit :

```
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);
```

Pour une application Node Express, vous pouvez utiliser les client-certificate-authmodules pour authentifier les demandes des clients à l'aide de certificats codés PEM.

Pour les autres serveurs HTTPS, consultez la documentation du serveur.

Rotation d'un certificat client arrivant à expiration

Le certificat client généré par API Gateway est valide pendant 365 jours. Vous devez effectuer la rotation de ce certificat avant qu'un certificat client sur une étape d'API arrive à expiration afin d'éviter les temps d'arrêt pour l'API. Vous pouvez vérifier la date d'expiration du certificat en appelant ClientCertificate:by-ID de l'API REST API Gateway ou en AWS CLI commandant et en get-client-certificateinspectant la propriété ExpirationDate renvoyée.

Pour faire pivoter un certificat client, procédez comme suit :

1. Générez un nouveau certificat client en appelant clientcertificate:generate de l'API REST API Gateway ou en utilisant la commande de. AWS CLI generate-client-certificate Dans ce didacticiel, nous supposons que l'ID du nouveau certificat client est `ndiqef`.

2. Mettez à jour le serveur backend pour inclure le nouveau certificat client. Ne supprimez pas le certificat client existant pour l'instant.

   Certains serveurs peuvent nécessiter un redémarrage pour terminer la mise à jour. Consultez la documentation du serveur pour voir si vous devez redémarrer le serveur pendant la mise à jour.

3. Mettez à jour l'étape d'API pour utiliser le nouveau certificat client en appelant la commande stage:update de l'API REST API Gateway, avec l'ID du nouveau certificat client (`ndiqef`) :

   ```
   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"
       }
     ]
   }
   ```

   ou en appelant la commande CLI update-stage.

4. Mettez à jour le serveur backend pour supprimer l'ancien certificat.

5. Supprimez l'ancien certificat d'API Gateway en appelant le clientcertificate:delete de l'API REST API Gateway, en spécifiant le clientCertificateId (`a1b2c3`) de l'ancien certificat :

   ```
   DELETE /clientcertificates/a1b2c3
   ```

   ou en appelant la commande CLI de delete-client-certificate:

   ```
   aws apigateway delete-client-certificate --client-certificate-id a1b2c3
   ```

Pour effectuer la rotation d'un certificat client dans la console pour une API déployée précédemment, procédez comme suit :

1. Dans le volet de navigation principal, choisissez Certificats de clients.
2. Dans le volet Certificats de clients, choisissez Générer un certificat.
3. Ouvrez l'API pour laquelle vous souhaitez utiliser le certificat client.
4. Sélectionnez Stages sous l'API sélectionnée, puis choisissez une étape.
5. Dans la section Détails de l'étape, choisissez Modifier.
6. Pour Certificat de client, sélectionnez le nouveau certificat.
7. Pour enregistrer les paramètres, choisissez Enregistrer les modifications.

   Vous devez redéployer l'API pour que les modifications prennent effet. Pour de plus amples informations, veuillez consulter the section called “Redéploiement d'une API REST jusqu'à une étape”.

Autorités de certification prises en charge par API Gateway pour HTTP et les intégrations de proxy HTTP

La liste suivante présente les autorités de certification prises en charge par API Gateway pour HTTP, HTTP proxy, et les intégrations privées.

```
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
```

Utilisation AWS WAF pour protéger vos API

AWS WAF est un pare-feu d'applications Web qui aide à protéger les applications Web et les API contre les attaques. Il vous permet de configurer un ensemble de règles appelé liste de contrôle d'accès web (ACL web) qui autorisent, bloquent ou comptent les requêtes web en fonction des règles et conditions de sécurité web personnalisables que vous définissez. Pour plus d'informations, consultez la section AWS WAF Fonctionnement.

Vous pouvez l'utiliser AWS WAF pour protéger votre API REST API Gateway contre les exploits Web courants, tels que l'injection SQL et les attaques par script intersite (XSS). Celles-ci peuvent

affecter la disponibilité et les performances des API, compromettre la sécurité ou consommer des ressources excessives. Par exemple, vous pouvez créer des règles pour autoriser ou bloquer les demandes provenant de plages d'adresses IP spécifiées, les demandes provenant de blocs CIDR, les demandes provenant d'un pays ou d'une région spécifique, les demandes contenant du code SQL malveillant ou les demandes contenant des scripts malveillants.

Vous pouvez également créer des règles qui correspondent à une chaîne spécifiée ou à un modèle d'expression régulière dans les en-têtes HTTP, la méthode, la chaîne de requête, l'URI et le corps de la demande (limité aux 64 premiers Ko). De plus, vous pouvez créer des règles pour bloquer les attaques émanant d'agents utilisateurs spécifiques, de robots malveillants et d'extracteurs de contenu. Par exemple, vous pouvez utiliser des règles basées sur le débit pour spécifier le nombre de requêtes web que chaque adresse IP du client est autorisée à envoyer au cours d'une période de 5 minutes mise à jour en continu.

> **Important**
>
> AWS WAF est votre première ligne de défense contre les exploits Web. Lorsqu'elle AWS WAF est activée sur une API, AWS WAF les règles sont évaluées avant les autres fonctionnalités de contrôle d'accès, telles que les politiques de ressources, les politiques IAM, les autorisateurs Lambda et les autorisateurs Amazon Cognito. Par exemple, si AWS WAF l'accès à partir d'un bloc CIDR autorisé par une politique de ressources est AWS WAF prioritaire et que la stratégie de ressources n'est pas évaluée.

AWS WAF Pour activer votre API, vous devez effectuer les opérations suivantes :

1. Utilisez la AWS WAF console, le AWS SDK ou la CLI pour créer une ACL Web contenant la combinaison souhaitée de règles AWS WAF gérées et de vos propres règles personnalisées. Pour plus d'informations, consultez les sections Mise en route AWS WAF et listes de contrôle d'accès Web (ACL Web).

   > **Important**
   >
   > API Gateway nécessite une ACL AWS WAFV2 Web pour une application régionale ou une ACL AWS WAF Classic Regional Web.

2. Associez l'ACL AWS WAF Web à une étape d'API. Vous pouvez le faire à l'aide de la AWS WAF console, du AWS SDK, de la CLI ou de la console API Gateway.

Pour associer une ACL AWS WAF Web à un stage d'API API Gateway à l'aide de la console API Gateway

Pour utiliser la console API Gateway afin d'associer une ACL AWS WAF Web à un stage d'API API Gateway existant, procédez comme suit :

1. Connectez-vous à la console API Gateway à l'adresse : https://console.aws.amazon.com/apigateway.
2. Choisissez une API existante ou créez-en une.
3. Dans le volet de navigation principal, choisissez Étapes, puis sélectionnez une étape.
4. Dans la section Détails de l'étape, choisissez Modifier.
5. Sous Pare-feu d'application Web (AWS WAF), sélectionnez votre ACL Web.

   Si vous utilisez AWS WAFV2, sélectionnez une ACL AWS WAFV2 Web pour une application régionale. L'ACL Web et toutes les autres AWS WAFV2 ressources qu'il utilise doivent être situées dans la même région que votre API.

   Si vous utilisez AWS WAF Classic Regional, sélectionnez une ACL Web régionale.
6. Sélectionnez Enregistrer les modifications.

Associez une ACL AWS WAF Web à un stage d'API API Gateway à l'aide du AWS CLI

AWS CLI Pour associer une ACL AWS WAFV2 Web pour une application régionale à un stage d'API API Gateway existant, appelez la associate-web-aclcommande, comme dans l'exemple suivant :

```
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
```

Pour associer une ACL AWS WAF Classic Regional Web AWS CLI à un stage d'API API Gateway existant, appelez la associate-web-aclcommande, comme dans l'exemple suivant :

```
aws waf-regional associate-web-acl \
--web-acl-id 'aabc123a-fb4f-4fc6-becb-2b00831cadcf' \
--resource-arn 'arn:aws:apigateway:{region}::/restapis/4wk1k4onj3/stages/prod'
```

Associer une ACL AWS WAF Web à une étape d'API à l'aide de l' AWS WAF API REST

Pour utiliser l' AWS WAFV2 API REST afin d'associer une ACL AWS WAFV2 Web pour une application régionale à un stage d'API API Gateway existant, utilisez la commande AssociateWebACL, comme dans l'exemple suivant :

```
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'
)
```

Pour utiliser l' AWS WAF API REST afin d'associer une ACL AWS WAF Classic Regional Web à un stage d'API API Gateway existant, utilisez la commande AssociateWebACL, comme dans l'exemple suivant :

```
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'
)
```

Limiter les demandes d'API pour améliorer le débit

Vous pouvez configurer des limitations et des quotas pour vos API afin d'éviter un trop grand nombre de demandes. Les limitations et les quotas sont appliqués dans la mesure du possible et doivent être considérés comme des cibles plutôt que des plafonds de demandes garantis.

API Gateway régule les demandes de limitations soumises à votre API à l'aide de l'algorithme de compartiment de jetons, où un jeton compte pour une demande. En particulier, API Gateway examine le taux de soumissions de demandes à un débit régulier et en mode rafale pour toutes les API de votre compte, par région. Dans l'algorithme de compartiment de jetons, une rafale peut permettre

un dépassement prédéfini de ces limites, mais d'autres facteurs peuvent également entraîner le dépassement des limites dans certains cas.

Lorsque les soumissions de demandes dépassent les limites de taux régulier et en mode rafale des demandes, API Gateway commence à limiter les demandes. Les clients peuvent recevoir des réponses aux erreurs `429 Too Many Requests` à ce stade. Lors de la capture de ces exceptions, le client peut renvoyer les demandes en échec de façon à limiter le débit tout en respectant les limitations.

En tant que développeur d'API, vous pouvez définir les limites pour les différentes étapes ou méthodes d'API afin d'améliorer les performances globales de toutes les API dans votre compte. Sinon, vous pouvez activer des plans d'utilisation pour restreindre les envois de demande client à des taux et des quotas de demande spécifiés.

Rubriques

Comment les paramètres de limitation sont appliqués dans API Gateway

Avant de configurer des paramètres de limitation et de quota pour votre API, il est utile de comprendre comment ils sont appliqués par Amazon API Gateway.

Amazon API Gateway fournit deux types élémentaires de paramètres relatifs à la limitation :

- AWS des limites de limitation sont appliquées à tous les comptes et clients d'une région. Ces paramètres de limitation empêchent votre API, ainsi que votre compte, d'être submergés par un trop grand nombre de requêtes. Ces limites sont définies par le client AWS et ne peuvent pas être modifiées par celui-ci.
- Les limites par compte sont appliquées à toutes les API d'un compte dans une région spécifiée. La limite du taux au niveau du compte peut être augmentée à la demande. Des limitations plus élevées sont possibles avec des API qui ont des délais d'attente plus courts et des charges utiles plus petites. Pour demander une augmentation des limitations au niveau du compte par

Région, contactez le Centre de support AWS. Pour plus d'informations, consultez Quotas et remarques importantes. Notez que ces limites ne peuvent pas être supérieures aux limites d' AWS étranglement.

- Les limitations par API et par étape sont appliquées au niveau de la méthode API pour une étape. Vous pouvez configurer les mêmes paramètres pour toutes les méthodes ou configurer différents paramètres d'accélération pour chaque méthode. Notez que ces limites ne peuvent pas être supérieures aux limites d' AWS étranglement.
- Les limitations par client sont appliquées aux clients qui utilisent les clés d'API associées à votre stratégie d'utilisation comme identifiant de client. Notez que ces limites ne peuvent pas être supérieures aux limites par compte.

Les paramètres relatifs à la limitation API Gateway sont appliqués dans l'ordre suivant :

1. Limitations par méthode et par client que vous avez définies pour une étape d'API dans un plan d'utilisation
2. Limitations par méthode que vous définissez pour une étape d'API
3. Limitations au niveau du compte par région
4. AWS Régulation régionale

Limitation au niveau du compte par région

API Gateway limite par défaut les demandes régulières par seconde (rps) pour toutes les API d'un compte AWS , par Région. Il limite également la rafale (c'est-à-dire, la taille maximale de compartiment) sur toutes les API au sein d'un compte AWS , par région. Dans API Gateway, la limite en mode rafale correspond au nombre maximal d'envois de demandes simultanés qu'API Gateway peut traiter à tout moment sans renvoyer de réponses d'erreur `429 Too Many Requests`. Pour plus d'informations sur les quotas de limitation, consultez Quotas et remarques importantes.

Configuration des cibles de limitation au niveau de l'API et de l'étape dans un plan d'utilisation

Dans un plan d'utilisation, vous pouvez définir une cible de limitation par méthode pour toutes les méthodes de l'API ou au niveau d'une étape. Vous pouvez spécifier un taux de limitation, c'est-à-dire le taux, en demandes par seconde, auquel les jetons sont ajoutés au compartiment de jetons. Vous pouvez également spécifier une rafale de limitation, qui correspond à la capacité du compartiment de jetons.

Vous pouvez utiliser la AWS CLI, les SDK et le AWS Management Console pour créer un plan d'utilisation. Pour plus d'informations sur la création d'un plan d'utilisation, consultez???.

Configuration de cibles de limitation au niveau de l’étape

Vous pouvez utiliser la AWS CLI, les SDK et le AWS Management Console pour créer des cibles de régulation au niveau de l'étape.

Pour plus d'informations sur la façon d'utiliser le pour AWS Management Console créer des cibles de régulation au niveau de l'étape, consultez. ??? Pour plus d'informations sur l'utilisation de la AWS CLI pour créer des cibles de régulation au niveau de l'étape, voir create-stage.

Configuration des cibles de limitations au niveau de la méthode dans un plan d'utilisation

Vous pouvez définir des cibles de limitations supplémentaires au niveau de la méthode dans les Usage Plans (Plans d'utilisation), comme indiqué dans Création d'un plan d'utilisation. Dans la console API Gateway, elles sont définies en spécifiant `Resource=<resource>`, `Method=<method>` dans le paramètre Configure Method Throttling (Configurer une limitation précise). Par PetStoreexemple, vous pouvez spécifier`Resource=/pets`,`Method=GET`.

API REST privées dans Amazon API Gateway

Une API privée est une API REST qui ne peut être appelée que depuis un Amazon VPC. Vous pouvez accéder à votre API à l'aide d'un point de terminaison VPC d'interface, qui est une interface réseau de point de terminaison que vous créez dans votre VPC. Les points de terminaison d'interface sont AWS PrivateLink alimentés par une technologie qui vous permet d'accéder à AWS des services de manière privée en utilisant des adresses IP privées.

Vous pouvez également l'utiliser AWS Direct Connect pour établir une connexion entre un réseau local et Amazon VPC, puis accéder à votre API privée via cette connexion. Dans tous les cas, le trafic vers votre API privée utilise des connexions sécurisées et est isolé de l'Internet public. Le trafic ne quitte pas le réseau Amazon.

Bonnes pratiques pour les API privées

Nous vous recommandons d'appliquer les meilleures pratiques suivantes lorsque vous créez votre API privée :

- Utilisez un point de terminaison VPC unique pour accéder à plusieurs API privées. Cela réduit le nombre de points de terminaison VPC dont vous pourriez avoir besoin.
- Associez votre point de terminaison VPC à votre API. Cela crée un enregistrement DNS alias Route 53 et simplifie l'appel de votre API privée.
- Activez le DNS privé pour votre VPC. De cette façon, vous pouvez appeler votre API au sein d'un VPC sans avoir à transmettre l'hôte ou `x-apigw-api-id` l'en-tête. Si vous choisissez de ne pas activer le DNS privé, vous ne pouvez accéder à votre API que via le DNS public.
- Limitez l'accès à votre API privée à des VPC ou points de terminaison VPC spécifiques. Ajoutez `aws:SourceVpc` des `aws:SourceVpce` conditions à la politique de ressources de votre API pour restreindre l'accès.
- Pour le périmètre de données le plus sécurisé, vous pouvez créer une politique de point de terminaison VPC. Cela contrôle l'accès aux points de terminaison VPC qui peuvent appeler votre API privée.

Considérations relatives aux API privées

Les considérations suivantes peuvent avoir un impact sur votre utilisation des API privées :

- Seules les API REST sont prises en charge.
- Les noms de domaine personnalisés ne sont pas pris en charge pour les API privées.
- Vous ne pouvez pas convertir une API privée en une API optimisée pour les périphériques.
- Les API privées ne prennent en charge que le protocole TLS 1.2. Les versions antérieures de TLS ne sont pas prises en charge.
- Les points de terminaison d'un VPC pour les API privées sont soumis aux mêmes limitations que les autres points de terminaison d'un VPC d'interface. Pour plus d'informations, consultez la section Accès à un AWS service à l'aide d'un point de terminaison VPC d'interface dans le AWS PrivateLink Guide. Pour en savoir plus sur l'utilisation d'API Gateway avec des VPC et des sous-réseaux partagés, consultez Sous-réseaux partagés dans le Guide AWS PrivateLink .

Prochaines étapes pour les API privées

Pour savoir comment créer une API privée et associer un point de terminaison VPC, consultez. the section called “Création d'une API privée” Pour suivre un didacticiel dans lequel vous créez des dépendances AWS CloudFormation et une API privée dans le AWS Management Console, voirthe section called “Tutoriel : Création d'une API REST privée”.

Création d'une API privée

Avant de créer une API privée, vous devez d'abord créer un point de terminaison VPC pour API Gateway. Ensuite, vous créez votre API privée et vous y associez une politique de ressources. Vous pouvez éventuellement associer votre point de terminaison VPC à votre API privée pour simplifier la façon dont vous appelez votre API. Enfin, vous déployez votre API.

Les procédures suivantes décrivent comment procéder. Vous pouvez créer une API REST privée à l'aide du AWS Management Console AWS CLI ou d'un AWS SDK.

Prérequis

Pour suivre ces étapes, vous devez disposer d'un VPC entièrement configuré. Pour savoir comment créer un VPC, consultez la section Créer un VPC uniquement dans le guide de l'utilisateur Amazon VPC. Pour suivre toutes les étapes recommandées lors de la création de votre VPC, activez le DNS privé. De cette façon, vous pouvez appeler votre API au sein d'un VPC sans avoir à transmettre l'hôte ou `x-apigw-api-id` l'en-tête.

Pour activer le DNS privé, les `enableDnsHostnames` attributs `enableDnsSupport` et de votre VPC doivent être définis sur. `true` Pour plus d'informations, consultez Support DNS dans votre VPC et Mise à jour du support DNS pour votre VPC.

Étape 1 : créer un point de terminaison VPC pour API Gateway dans votre VPC

La procédure suivante montre comment créer un point de terminaison VPC pour API Gateway. Pour créer un point de terminaison VPC pour API Gateway, vous devez spécifier le `execute-api` domaine dans Région AWS lequel vous créez votre API privée. Le `execute-api` domaine est le service du composant API Gateway pour l'exécution de l'API.

Lorsque vous créez votre point de terminaison VPC pour API Gateway, vous spécifiez les paramètres DNS. Si vous désactivez le DNS privé, vous ne pouvez accéder à votre API que via le DNS public. Pour plus d'informations, consultez the section called “Problème : je ne parviens pas à me connecter à mon API publique depuis un point de terminaison VPC API Gateway”.

AWS Management Console

Pour créer un point de terminaison VPC d'interface pour API Gateway

1. Connectez-vous à la console Amazon VPC AWS Management Console et ouvrez-la à l'adresse https://console.aws.amazon.com/vpc/.

2. Dans le panneau de navigation, sous Cloud privé virtuel, choisissez Points de terminaison.
3. Choisissez Créer un point de terminaison.
4. (Facultatif) Dans le champ Name tag, entrez un nom pour identifier le point de terminaison de votre VPC.
5. Pour Service category (Catégorie de service), choisissez Services AWS .
6. Sous Services, dans la barre de recherche, entrez**`execute-api`**. Choisissez ensuite le point de terminaison du service API Gateway dans Région AWS lequel vous allez créer votre API. Le nom du service doit ressembler à Interface `com.amazonaws.us-east-1.execute-api` et le type doit être Interface.
7. Pour VPC, choisissez le VPC dans lequel vous souhaitez créer le point de terminaison.
8. (Facultatif) Pour désactiver l'option Activer le nom DNS privé, sélectionnez Paramètres supplémentaires, puis désactivez Activer le nom DNS privé.
9. Pour les sous-réseaux, choisissez les zones de disponibilité dans lesquelles vous avez créé les interfaces réseau des terminaux. Pour améliorer la disponibilité de votre API, choisissez plusieurs sous-réseaux.
10. Pour Security group (Groupe de sécurité), sélectionnez le groupe de sécurité à associer aux interfaces réseau du point de terminaison d'un VPC.

    Le groupe de sécurité que vous choisissez doit être configuré pour autoriser le trafic HTTPS entrant du port TCP 443 à partir d'une plage IP de votre VPC ou d'un autre groupe de sécurité de votre VPC.
11. Pour Policy, effectuez l'une des opérations suivantes :
    - Si vous n'avez pas créé votre API privée ou si vous ne souhaitez pas configurer de politique de point de terminaison VPC personnalisée, choisissez Accès complet.
    - Si vous avez déjà créé une API privée et que vous souhaitez configurer une politique de point de terminaison VPC personnalisée, vous pouvez entrer une politique de point de terminaison VPC personnalisée. Pour plus d’informations, consultez the section called “Utilisation de stratégies de point de terminaison de VPC pour des API privées”.

    Vous pouvez mettre à jour la politique de point de terminaison du VPC après avoir créé le point de terminaison du VPC. Pour plus d'informations, consultez Mettre à jour une politique de point de terminaison VPC.
12. Choisissez Créer un point de terminaison.
13. Copiez l'ID de point de terminaison VPC obtenu, tel que vous pourriez l'utiliser dans les étapes futures.

AWS CLI

La create-vpc-endpointcommande suivante peut être utilisée pour créer un point de terminaison VPC :

```
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
```

Copiez l'ID de point de terminaison VPC obtenu, tel que vous pourriez l'utiliser dans les étapes futures.

Étape 2 : Créer une API privée

Après avoir créé votre point de terminaison VPC, vous créez une API REST privée. La procédure suivante montre comment créer une API privée.

AWS Management Console

Pour créer une API privée

1. Connectez-vous à la console API Gateway à l'adresse : https://console.aws.amazon.com/apigateway.
2. Sélectionnez Create API (Créer une API).
3. Sous REST API (API REST), choisissez Build (Création).
4. Pour Nom, entrez un nom.
5. (Facultatif) Sous Description, entrez une description.
6. Pour Type de point de terminaison d'API, sélectionnez Privé.
7. (Facultatif) Pour les ID de point de terminaison VPC, entrez un ID de point de terminaison VPC.

   Si vous associez un ID de point de terminaison VPC à votre API privée, vous pouvez appeler votre API depuis votre VPC sans remplacer un `Host` en-tête ni transmettre un. `x-apigw-api-id header` Pour plus d'informations, consultez. the section called “(Facultatif) Associer ou dissocier un point de terminaison VPC à une API privée”

8. Sélectionnez Create API (Créer une API).

Après avoir effectué les étapes précédentes, vous pouvez suivre les instructions the section called “Bien démarrer avec la console d'API REST” pour configurer les méthodes et les intégrations pour cette API, mais vous ne pouvez pas déployer votre API. Pour déployer votre API, suivez l'étape 3 et associez une politique de ressources à votre API.

AWS CLI

La update-rest-apicommande suivante montre comment créer une API privée :

```
aws apigateway create-rest-api \
        --name 'Simple PetStore (AWS CLI, Private)' \
        --description 'Simple private PetStore API' \
        --region us-west-2 \
        --endpoint-configuration '{ "types": ["PRIVATE"] }'
```

Un appel fructueux renvoie une sortie similaire à ce qui suit :

```
{
    "createdDate": "2017-10-13T18:41:39Z",
    "description": "Simple private PetStore API",
    "endpointConfiguration": {
        "types": "PRIVATE"
    },
    "id": "0qzs2sy7bh",
    "name": "Simple PetStore (AWS CLI, Private)"
}
```

Après avoir effectué les étapes précédentes, vous pouvez suivre les instructions the section called “Tutoriel : Création d'une API optimisée pour les périphériques à l'aide AWS de SDK ou AWS CLI” pour configurer les méthodes et les intégrations pour cette API, mais vous ne pouvez pas déployer votre API. Pour déployer votre API, suivez l'étape 3 et associez une politique de ressources à votre API.

SDK JavaScript v3

L'exemple suivant montre comment créer une API privée à l'aide du AWS SDK pour la JavaScript version 3 :

```
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)
 }
};
```

Un appel fructueux renvoie une sortie similaire à ce qui suit :

```
{
  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'
}
```

Après avoir effectué les étapes précédentes, vous pouvez suivre les instructions the section called “Tutoriel : Création d'une API optimisée pour les périphériques à l'aide AWS de SDK ou AWS CLI” pour configurer les méthodes et les intégrations pour cette API, mais vous ne pouvez pas déployer votre API. Pour déployer votre API, suivez l'étape 3 et associez une politique de ressources à votre API.

Python SDK

L'exemple suivant montre comment créer une API privée à l'aide du AWS SDK pour Python :

```
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
```

Un appel fructueux renvoie une sortie similaire à ce qui suit :

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

Après avoir effectué les étapes précédentes, vous pouvez suivre les instructions the section called “Tutoriel : Création d'une API optimisée pour les périphériques à l'aide AWS de SDK ou AWS CLI” pour configurer les méthodes et les intégrations pour cette API, mais vous ne pouvez pas déployer votre API. Pour déployer votre API, suivez l'étape 3 et associez une politique de ressources à votre API.

Étape 3 : configurer une politique de ressources pour une API privée

Votre API privée actuelle est inaccessible à tous les VPC. Utilisez une politique de ressources pour accorder à vos VPC et points de terminaison VPC l'accès à vos API privées. Vous pouvez accorder l'accès à un point de terminaison VPC dans n'importe quel AWS compte.

Votre politique en matière de ressources doit contenir `aws:SourceVpc` des `aws:SourceVpce` conditions visant à restreindre l'accès. Nous vous recommandons d'identifier des VPC et des points de terminaison VPC spécifiques et de ne pas créer de politique de ressources autorisant l'accès à tous les VPC et points de terminaison VPC.

La procédure suivante montre comment associer une politique de ressources à votre API.

AWS Management Console

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Choisissez une API REST.
3. Dans le panneau de navigation principal, choisissez Stratégie de ressources.
4. Choisissez Créer une politique.
5. Choisissez Sélectionnez un modèle, puis choisissez Source VPC.
6. `{{vpceID}}`Remplacez-le (y compris les bretelles) par votre identifiant de point de terminaison VPC.
7. Sélectionnez Enregistrer les modifications.

AWS CLI

La update-rest-apicommande suivante montre comment associer une politique de ressources à une API existante :

```
aws apigateway update-rest-api \
    --rest-api-id a1b2c3 \
    --patch-operations op=replace,path=/
policy,value='"{\"jsonEscapedPolicyDocument\"}"'
```

Vous souhaiterez peut-être également contrôler les ressources qui ont accès à votre point de terminaison VPC. Pour contrôler quelles ressources ont accès à votre point de terminaison VPC,

associez une politique de point de terminaison à votre point de terminaison VPC. Pour plus d'informations, consultez the section called "Utilisation de stratégies de point de terminaison de VPC pour des API privées".

(Facultatif) Associer ou dissocier un point de terminaison VPC à une API privée

Lorsque vous associez un point de terminaison VPC à votre API privée, API Gateway génère un nouvel enregistrement DNS d'alias Route 53. Vous pouvez utiliser cet enregistrement pour appeler vos API privées comme vous le faites pour vos API publiques sans remplacer un `Host` en-tête ni en transmettre un`x-apigw-api-id`.

Format de l'URL de base générée :

```
https://{rest-api-id}-{vpce-id}.execute-api.{region}.amazonaws.com/{stage}
```

Associate a VPC endpoint (AWS Management Console)

Vous pouvez associer un point de terminaison VPC à votre API privée lors de sa création ou après sa création. La procédure suivante montre comment associer un point de terminaison VPC à une API créée précédemment.

Pour associer un point de terminaison VPC à une API privée

1. Connectez-vous à la console API Gateway à l'adresse : https://console.aws.amazon.com/apigateway.
2. Choisissez votre API privée.
3. Dans le panneau de navigation principal, choisissez Stratégie de ressources.
4. Modifiez votre politique de ressources pour autoriser les appels depuis votre point de terminaison d'un VPC supplémentaire.
5. Dans le panneau de navigation principal, choisissez Paramètres de l'API.
6. Dans la section Détails de l'API, choisissez Modifier.
7. Pour ID de points de terminaison d'un VPC sélectionnez les ID de point de terminaison d'un VPC supplémentaires.
8. Choisissez Enregistrer.
9. Redéployez l'API pour que les modifications prennent effet.

Dissociate a VPC endpoint (AWS Management Console)

Pour dissocier un point de terminaison VPC d'une API REST privée

1. Connectez-vous à la console API Gateway à l'adresse : https://console.aws.amazon.com/apigateway.
2. Choisissez votre API privée.
3. Dans le panneau de navigation principal, choisissez Stratégie de ressources.
4. Modifiez votre stratégie de ressources pour supprimer les mentions du point de terminaison d'un VPC que vous souhaitez dissocier de votre API privée.
5. Dans le panneau de navigation principal, choisissez Paramètres de l'API.
6. Dans la section Détails de l'API, choisissez Modifier.
7. Pour les ID de points de terminaison d'un VPC, choisissez le X pour dissocier le point de terminaison d'un VPC.
8. Choisissez Enregistrer.
9. Redéployez l'API pour que les modifications prennent effet.

Associate a VPC endpoint (AWS CLI)

La create-rest-apicommande suivante montre comment associer des points de terminaison VPC au moment de la création de l'API :

```
aws apigateway create-rest-api \
    --name Petstore \
    --endpoint-configuration '{ "types": ["PRIVATE"], "vpcEndpointIds" :
 ["vpce-0212a4ababd5b8c3e", "vpce-0393a628149c867ee"] }' \
    --region us-west-2
```

Le résultat se présente comme suit :

```
{
    "apiKeySource": "HEADER",
    "endpointConfiguration": {
        "types": [
            "PRIVATE"
        ],
        "vpcEndpointIds": [
            "vpce-0212a4ababd5b8c3e",
```

```
            "vpce-0393a628149c867ee"
        ]
    },
    "id": "u67n3ov968",
    "createdDate": 1565718256,
    "name": "Petstore"
}
```

La update-rest-apicommande suivante montre comment associer des points de terminaison VPC à une API que vous avez déjà créée :

```
aws apigateway update-rest-api \
    --rest-api-id u67n3ov968 \
    --patch-operations "op='add',path='/endpointConfiguration/
vpcEndpointIds',value='vpce-01d622316a7df47f9'" \
    --region us-west-2
```

Le résultat se présente comme suit :

```
{
    "name": "Petstore",
    "apiKeySource": "1565718256",
    "tags": {},
    "createdDate": 1565718256,
    "endpointConfiguration": {
        "vpcEndpointIds": [
            "vpce-0212a4ababd5b8c3e",
            "vpce-0393a628149c867ee",
            "vpce-01d622316a7df47f9"
        ],
        "types": [
            "PRIVATE"
        ]
    },
    "id": "u67n3ov968"
}
```

Redéployez l'API pour que les modifications prennent effet.

Disassociate a VPC endpoint (AWS CLI)

La update-rest-apicommande suivante montre comment dissocier un point de terminaison VPC d'une API privée :

```
aws apigateway update-rest-api \
    --rest-api-id u67n3ov968 \
    --patch-operations "op='remove',path='/endpointConfiguration/
vpcEndpointIds',value='vpce-0393a628149c867ee'" \
    --region us-west-2
```

Le résultat se présente comme suit :

```
{
    "name": "Petstore",
    "apiKeySource": "1565718256",
    "tags": {},
    "createdDate": 1565718256,
    "endpointConfiguration": {
        "vpcEndpointIds": [
            "vpce-0212a4ababd5b8c3e",
            "vpce-01d622316a7df47f9"
        ],
        "types": [
            "PRIVATE"
        ]
    },
    "id": "u67n3ov968"
}
```

Redéployez l’API pour que les modifications prennent effet.

Étape 4 : Déployer une API privée

Pour déployer votre API, vous devez créer un déploiement d'API et l'associer à une étape. La procédure suivante indique comment déployer votre API privée.

AWS Management Console

Pour déployer une API privée

1. Choisissez votre API.

2. Sélectionnez Deploy API (Déployer une API).
3. Pour Étape, sélectionnez Nouvelle étape.
4. Pour Nom de l'étape, entrez un nom d'étape.
5. (Facultatif) Sous Description, entrez une description.
6. Choisissez Deploy (Déployer).

AWS CLI

La commande create-deployment suivante montre comment déployer une API privée :

```
aws apigateway create-deployment --rest-api-id a1b2c3 \
  --stage-name test \
  --stage-description 'Private API test stage' \
  --description 'First deployment'
```

Résoudre les problèmes liés à votre API privée

Vous trouverez ci-dessous des conseils de résolution des erreurs et des problèmes que vous pourriez rencontrer lors de la création d'une API privée.

Problème : je ne parviens pas à me connecter à mon API publique depuis un point de terminaison VPC API Gateway

Lorsque vous créez votre VPC, vous pouvez configurer les paramètres DNS. Nous vous recommandons d'activer le DNS privé pour votre VPC. Si vous choisissez de désactiver le DNS privé, vous ne pouvez accéder à votre API que via le DNS public.

Si vous activez le DNS privé, vous ne pouvez pas accéder au point de terminaison par défaut d'une API API Gateway publique depuis votre point de terminaison VPC. Vous pouvez accéder à une API avec un nom de domaine personnalisé.

Si vous créez un nom de domaine personnalisé régional, utilisez un enregistrement d'alias de type A, si vous créez un nom de domaine personnalisé optimisé pour les périphériques, votre type d'enregistrement n'est soumis à aucune restriction. Vous pouvez accéder à ces API publiques avec le DNS privé activé. Pour plus d'informations, consultez Problème : je me connecte à mon API publique depuis un point de terminaison VPC API Gateway.

Problème : Mon API renvoie **{"Message":"User: anonymous is not authorized to perform: execute-api:Invoke on resource: arn:aws:execute-api:us-east-1:********/****/****/"}**

Dans votre politique de ressources, si vous attribuez au principal un AWS principal, tel que ce qui suit :

```
{
    "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:/*"
            ]
        },
        ...
        }
```

Vous devez utiliser `AWS_IAM` l'autorisation pour chaque méthode de votre API, sinon votre API renvoie le message d'erreur précédent. Pour plus d'instructions sur la façon d'activer `AWS_IAM` l'autorisation pour une méthode, consultezthe section called “Méthodes”.

Problème : je ne peux pas dire si mon point de terminaison VPC est associé à mon API

Si vous associez ou dissociez un point de terminaison VPC à votre API privée, vous devez redéployer votre API. L'opération de mise à jour peut prendre quelques minutes en raison de la propagation du DNS. Pendant ce temps, votre API est disponible, mais la propagation DNS pour les URL DNS nouvellement générées peut encore être en cours. Si, au bout de quelques minutes, vos nouvelles URL ne sont pas résolues dans le DNS, nous vous recommandons de redéployer votre API.

Appelez une API privée

Vous ne pouvez appeler une API privée que depuis un VPC. Votre API privée doit disposer d'une politique de ressources qui autorise des VPC et des points de terminaison VPC spécifiques à appeler votre API.

Vous pouvez appeler votre API privée de la manière suivante :

- Appelez votre API à l'aide d'un alias Route53. Ceci n'est disponible que si vous avez associé votre point de terminaison VPC à votre API. Pour plus d'informations, consultez the section called "(Facultatif) Associer ou dissocier un point de terminaison VPC à une API privée".
- Appelez votre API à l'aide d'un DNS privé. Ceci n'est disponible que si vous avez activé le DNS privé pour votre VPC.
- Appelez votre API en utilisant AWS Direct Connect.
- Appelez votre API à l'aide de noms d'hôtes DNS publics spécifiques au point de terminaison.

Pour appeler votre API privée à l'aide d'un nom DNS, vous devez identifier les noms DNS de votre API. La procédure suivante indique comment trouver vos noms DNS.

AWS Management Console

Pour trouver les noms DNS

1. Connectez-vous à la console Amazon VPC AWS Management Console et ouvrez-la à l'adresse https://console.aws.amazon.com/vpc/.
2. Dans le volet de navigation principal, choisissez Endpoints, puis choisissez le point de terminaison VPC de votre interface pour API Gateway.
3. Dans le volet Détails, vous verrez cinq valeurs dans le champ des noms DNS. Les trois premiers sont les noms DNS publics de votre API. Les deux autres sont les noms DNS privés correspondants.

AWS CLI

Utilisez la describe-vpc-endpointscommande suivante pour répertorier vos valeurs DNS.

```
aws ec2 describe-vpc-endpoints --filters vpc-endpoint-id=vpce-01234567abcdef012
```

Les trois premiers sont les noms DNS publics de votre API. Les deux autres sont les noms DNS privés correspondants.

Invoquer une API privée à l'aide d'un alias Route53

Vous pouvez associer ou dissocier un point de terminaison VPC à votre API privée. Pour plus d'informations, consultez the section called “(Facultatif) Associer ou dissocier un point de terminaison VPC à une API privée”.

Après avoir associé vos points de terminaison VPC à votre API privée, vous pouvez utiliser l'URL de base suivante pour appeler l'API :

```
https://{rest-api-id}-{vpce-id}.execute-api.{region}.amazonaws.com/{stage}
```

Par exemple, si vous avez configuré la `GET /pets` méthode pour l'`test`étape et que votre ID d'API REST était`01234567ab`, que votre identifiant de point de terminaison VPC était et que votre région l'était `vpce-01234567abcdef012us-west-2`, vous pouvez appeler votre API sous la forme suivante :

```
curl -v https://01234567ab-vpce-01234567abcdef012.execute-api.us-west-2.amazonaws.com/
test/pets
```

Invoquer une API privée à l'aide de noms DNS privés

Si vous avez activé le DNS privé, vous pouvez accéder à votre API privée en utilisant le nom de DNS privé suivant :

```
{restapi-id}.execute-api.{region}.amazonaws.com
```

L'URL de base pour appeler l'API est au format suivant :

```
https://{restapi-id}.execute-api.{region}.amazonaws.com/{stage}
```

Par exemple, si vous avez configuré la `GET /pets` méthode pour l'`test`étape et que votre ID d'API REST était `01234567ab` et que votre région l'était`us-west-2`, vous pouvez appeler votre API privée en saisissant l'URL suivante dans un navigateur :

```
https://01234567ab.execute-api.us-west-2.amazonaws.com/test/pets
```

Vous pouvez également utiliser la commande cURL suivante pour appeler votre API privée :

```
curl -X GET https://01234567ab.execute-api.us-west-2.amazonaws.com/test/pets
```

> **Warning**
>
> Si vous activez le DNS privé pour votre point de terminaison VPC, vous pouvez accéder au point de terminaison par défaut pour les API publiques. Pour de plus amples informations, veuillez consulter Pourquoi ne puis-je pas me connecter à mon API publique à partir d'un point de terminaison d'un VPC API Gateway ?.

Invoquez une API privée en utilisant AWS Direct Connect

Vous pouvez l'utiliser AWS Direct Connect pour établir une connexion privée dédiée entre un réseau local et Amazon VPC et accéder à votre point de terminaison d'API privé via cette connexion en utilisant des noms DNS publics.

Vous pouvez également utiliser des noms DNS privés pour accéder à votre API privée depuis un réseau local en configurant un point de terminaison Amazon Route 53 Resolver entrant et en lui transmettant toutes les requêtes DNS du DNS privé depuis votre réseau distant. Pour plus d'informations, consultez Consignation des requêtes DNS vers vos VPC dans le Guide du développeur Amazon Route 53.

Invoquer une API privée à l'aide de noms d'hôtes DNS publics spécifiques au point de terminaison

Vous pouvez accéder à votre API privée en utilisant des noms d'hôte DNS spécifiques au point de terminaison. Il s'agit des noms d'hôte DNS publics contenant l'ID du point de terminaison d'un VPC ou l'ID d'API de votre API privée.

Format de l'URL de base générée :

```
https://{public-dns-hostname}.execute-api.{region}.vpce.amazonaws.com/{stage}
```

Par exemple, si vous avez configuré la `GET /pets` méthode pour l'`test`étape et que votre ID d'API REST était`abc1234`, son nom d'hôte DNS public et votre région `vpce-def-01234567us-west-2`, vous pourriez appeler votre API privée à l'aide de son identifiant vPCE en utilisant l'`Host`en-tête d'une commande cURL :

```
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'
```

Vous pouvez également appeler votre API privée via son ID d'API en utilisant l'`x-apigw-api-id`en-tête d'une commande cURL au format suivant :

```
curl -v https://{public-dns-hostname}.execute-api.{region}.vpce.amazonaws.com/{stage} -
H 'x-apigw-api-id:{api-id}'
```

Surveillance des API REST

Dans cette section, vous pouvez apprendre à surveiller votre API à l'aide des CloudWatch métriques, des CloudWatch journaux, de Firehose et. AWS X-Ray En combinant les journaux CloudWatch d'exécution et CloudWatch les métriques, vous pouvez enregistrer les erreurs et les traces d'exécution, et surveiller les performances de votre API. Vous souhaiterez peut-être également enregistrer les appels d'API à Firehose. Vous pouvez également l'utiliser AWS X-Ray pour suivre les appels via les services en aval qui constituent votre API.

> **Note**
>
> API Gateway peut ne pas générer de journaux et de métriques dans les cas suivants :
>
> - Nombre d'erreurs de demande d'entité 413 trop important
> - Nombre d'erreurs de demande 429 trop important
> - Erreurs de la série 400 provenant de demandes envoyées à un domaine personnalisé qui n'a pas de mappage d'API
> - Erreurs de la série 500 causées par des défaillances internes
>
> API Gateway ne génère pas de journaux et de métriques lors du test d'une méthode d'API REST. Les CloudWatch entrées sont simulées. Pour plus d'informations, voir the section called “Utilisez la console pour tester une méthode de l'API REST.”.

Rubriques

- Journalisation des appels d'API vers Amazon Data Firehose
- Suivi des demandes utilisateur vers les API REST à l'aide de X-Ray

Surveillance de l'exécution de l'API REST avec CloudWatch les métriques Amazon

Vous pouvez surveiller l'exécution des API en utilisant CloudWatch, qui collecte et traite les données brutes d'API Gateway en near-real-time métriques lisibles. Ces statistiques sont enregistrées pour une durée de 15 mois et, par conséquent, vous pouvez accéder aux informations historiques et acquérir un meilleur point de vue de la façon dont votre service ou application web s'exécute. Par défaut, les données métriques d'API Gateway sont automatiquement envoyées par CloudWatch intervalles d'une minute. Pour plus d'informations, consultez Qu'est-ce qu'Amazon CloudWatch ? dans le guide de CloudWatch l'utilisateur Amazon.

Les métriques présentées par API Gateway fournissent des informations permettant différents types d'analyse. La liste suivante présente certaines utilisations courantes des métriques qui sont des suggestions pour vous aider à démarrer :

- Surveillez les IntegrationLatencymétriques pour mesurer la réactivité du backend.
- Surveillez les métriques Latency pour mesurer la réactivité globale de vos appels d'API.
- Surveillez les CacheMissCountindicateurs CacheHitCountet pour optimiser les capacités du cache afin d'atteindre les performances souhaitées.

Rubriques

- Dimensions et métriques Amazon API Gateway
- Afficher CloudWatch les métriques à l'aide du tableau de bord de l'API dans API Gateway
- Afficher les métriques d'API Gateway dans la CloudWatch console
- Afficher les événements du journal d'API Gateway dans la CloudWatch console
- Outils de surveillance dans AWS

Dimensions et métriques Amazon API Gateway

Les métriques et les dimensions qu'API Gateway envoie à Amazon CloudWatch sont répertoriées ci-dessous. Pour de plus amples informations, veuillez consulter Surveillance de l'exécution de l'API REST avec CloudWatch les métriques Amazon.

Métriques API Gateway

Amazon API Gateway envoie des données métriques à CloudWatch chaque minute.

L'espace de noms `AWS/ApiGateway` inclut les métriques suivantes.

Métrique	Description
4XXError	Nombre d'erreurs côté client capturées dans une période donnée. API Gateway considère les codes d'état de réponse de la passerelle modifiés comme des erreurs 4XXError. La statistique `Sum` représente cette métrique, à savoir le nombre total d'erreurs 4XXError sur la période nommée. La statistique `Average` représente le taux d'erreurs 4XXError, à savoir le nombre total d'erreurs 4XXError divisé par le nombre total de demandes pendant la période. Le dénominateur correspond à la métrique Count (ci-dessous). Unit: Count
5XXError	Nombre d'erreurs côté serveur capturées dans une période donnée. La statistique `Sum` représente cette métrique, à savoir le nombre total d'erreurs 5XXError sur la période nommée. La statistique `Average` représente le taux d'erreurs 5XXError, à savoir le nombre total d'erreurs 5XXError divisé par le nombre total de demandes pendant la période. Le dénominateur correspond à la métrique Count (ci-dessous). Unit: Count
CacheHitCount	Le nombre de demandes servies depuis le cache de l'API sur une période donnée.

Métrique	Description
	La statistique `Sum` représente cette métrique, à savoir le nombre total d'accès au cache sur la période donnée. La statistique `Average` représente le taux d'accès au cache, à savoir le nombre total d'accès au cache divisé par le nombre total de demandes pendant la période. Le dénominateur correspond à la métrique Count (ci-dessous). Unit: Count
CacheMissCount	Nombre de demandes traitées à partir du backend sur une période donnée lorsque la mise en cache des API est activée. La statistique `Sum` représente cette métrique, à savoir le nombre total d'échecs d'accès au cache sur la période nommée. La statistique `Average` représent e le taux d'échecs d'accès au cache, à savoir le nombre total d'accès au cache divisé par le nombre total de demandes pendant la période. Le dénominat eur correspond à la métrique Count (ci-dessous). Unit: Count
Count	Nombre total de demandes d'API sur une période donnée. La statistique `SampleCount` représente cette métrique. Unit: Count
IntegrationLatency	Délai entre le moment de la transmission de la demande au backend par API Gateway et celui de la réception de la réponse du backend. Unit: Millisecond

Métrique	Description
Latency	Délai entre le moment de la réception par API Gateway d'une demande d'un client et celui du renvoi de la réponse au client. La latence prend en compte la latence d'intégration et autres surcharges d'API Gateway. Unit: Millisecond

Dimensions pour les métriques

Vous pouvez utiliser les dimensions du tableau suivant pour filtrer les métriques API Gateway.

> **Note**
>
> API Gateway supprime les caractères non ASCII de la ApiName dimension avant d'envoyer des métriques à. CloudWatch Si l'APIName ne contient aucun caractère ASCII, l'API ID est utilisé comme ApiName.

Dimension	Description
ApiName	Filtre les métriques API Gateway à la recherche de l'API REST avec le nom d'API spécifié.
ApiName, Method, Resource, Stage	Filtre les métriques API Gateway à la recherche de la méthode d'API avec le nom d'API, l'étape, la ressource et la méthode spécifiés. API Gateway n'enverra pas ces métriques à moins que vous n'ayez explicitement activé CloudWatch les métriques détaillées. Dans la console, choisisse z une étape, puis pour Journaux et suivi, sélection nez Modifier. Sélectionnez Métriques détaillées, puis cliquez sur Enregistrer les modifications. Vous pouvez également appeler la AWS CLI comma

Dimension	Description
	nde update-stage pour mettre à jour la `metricsEnabled` propriété en. `true` L'activation de ces métriques implique des frais supplémentaires pour votre compte. Pour plus d'informations sur les tarifs, consultez Amazon CloudWatch Pricing.
ApiName, Stage	Filtre les métriques API Gateway pour trouver la ressource d'étape d'API avec le nom d'API et l'étape spécifiés.

Afficher CloudWatch les métriques à l'aide du tableau de bord de l'API dans API Gateway

Vous pouvez utiliser le tableau de bord de l'API dans la console API Gateway pour afficher les CloudWatch métriques de votre API déployée dans API Gateway. Celles-ci sont présentées sous forme de récapitulatif de l'activité de l'API au fil du temps.

Rubriques

- Prérequis
- Examen des activités de l'API dans le tableau de bord

Prérequis

1. Vous devez avoir créé une API dans API Gateway. Suivez les instructions de la section Développement d'une API REST dans API Gateway.
2. Vous devez avoir déployé l'API au moins une fois. Suivez les instructions de la section Déploiement d'une API REST dans Amazon API Gateway.

Examen des activités de l'API dans le tableau de bord

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.

2. Choisissez une API.
3. Dans le panneau de navigation principal, choisissez Tableau de bord.
4. Pour Étape, choisissez l'étape souhaitée.
5. Choisissez Plage de dates pour spécifier une plage de dates.
6. Actualisez la page, si nécessaire, et affichez les diverses métriques présentées dans des graphiques distincts intitulés Appels d'API, Latence, Latence d'intégration, Latence, Erreur 4xx et Erreur 5xx.

> **Tip**
>
> Pour examiner les CloudWatch métriques au niveau de la méthode, assurez-vous que vous avez activé les CloudWatch journaux au niveau de la méthode. Pour de plus amples informations sur la configuration de la journalisation au niveau de la méthode, veuillez consulter Mise à jour des paramètres de l'étape à l'aide de la console API Gateway.

Afficher les métriques d'API Gateway dans la CloudWatch console

Les métriques sont d'abord regroupées par espace de noms de service, puis par les différentes combinaisons de dimension au sein de chaque espace de noms. Pour afficher les métriques au niveau métrique de votre API, activez les métriques détaillées. Pour plus d'informations, consultez the section called “Mise à jour des paramètres d'étape”.

Pour afficher les métriques d'API Gateway à l'aide de la CloudWatch console

1. Ouvrez la CloudWatch console à l'adresse https://console.aws.amazon.com/cloudwatch/.
2. Si nécessaire, changez la Région AWS. Dans la barre de navigation, sélectionnez la région dans laquelle se trouvent vos AWS ressources.
3. Dans le panneau de navigation, sélectionnez Métriques.
4. Sous l'onglet Toutes les métriques, choisissez API Gateway.
5. Pour voir les métriques par étape, choisissez le volet By Stage. Sélectionnez ensuite vos API et les noms de vos métriques.
6. Pour voir les métriques par API spécifique, choisissez le volet By Api Name. Sélectionnez ensuite vos API et les noms de vos métriques.

Pour afficher les métriques à l'aide de la AWS CLI

1. À partir d'une invite de commande, utilisez la commande suivante pour répertorier les métriques :

   ```
   aws cloudwatch list-metrics --namespace "AWS/ApiGateway"
   ```

   Après avoir créé une métrique, attendez jusqu'à 15 minutes pour qu'elle apparaisse. Pour consulter les statistiques métriques plus rapidement, utilisez get-metric-dataou get-metric-statistics.
2. Pour afficher des statistiques spécifiques (par exemple, `Average`) sur des intervalles de 5 minutes, appelez la commande suivante :

   ```
   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
   ```

Afficher les événements du journal d'API Gateway dans la CloudWatch console

Prérequis

1. Vous devez avoir créé une API dans API Gateway. Suivez les instructions de la section Développement d'une API REST dans API Gateway.
2. Vous devez avoir déployé et invoqué l'API au moins une fois. Suivez les instructions de Déploiement d'une API REST dans Amazon API Gateway et Appel d'une API REST dans Amazon API Gateway.
3. Les CloudWatch journaux doivent être activés pour une étape. Suivez les instructions de la section Configuration de la CloudWatch journalisation pour une API REST dans API Gateway.

Pour afficher les demandes et réponses d'API enregistrées à l'aide de la CloudWatch console

1. Ouvrez la CloudWatch console à l'adresse https://console.aws.amazon.com/cloudwatch/.
2. Si nécessaire, changez la Région AWS. Dans la barre de navigation, sélectionnez la région dans laquelle se trouvent vos AWS ressources. Pour plus d'informations, consultez Régions et points de terminaison.

3. Dans le panneau de navigation de gauche, choisissez Logs (Journaux), Log groups (Groupes de journaux).
4. Dans le tableau Log Groups, choisissez un groupe de journaux portant le nom API-Gateway-Execution-Logs_ {}/{stage-name}. rest-api-id
5. Sous la table Flux de journaux, choisissez un flux de journaux. Vous pouvez utiliser l'horodatage pour aider à localiser le flux de journal qui vous intéresse.
6. Choisissez Texte pour afficher du texte brut ou choisissez Row pour afficher l'événement ligne par ligne.

> **Important**
>
> CloudWatch vous permet de supprimer des groupes de journaux ou des flux. Ne supprimez pas manuellement les groupes ou flux de journaux API Gateway ; laissez API Gateway gérer ces ressources. La suppression manuelle de groupes ou de flux de journaux peut entraîner la non journalisation des demandes et réponses d'API. Si tel est le cas, vous pouvez supprimer l'intégralité du groupe de journaux pour l'API et redéployer l'API. La raison en est que l'API Gateway crée des groupes ou flux de journaux pour une étape d'API au moment de son déploiement.

Outils de surveillance dans AWS

AWS fournit différents outils que vous pouvez utiliser pour surveiller API Gateway. Vous pouvez configurer certains de ces outils pour qu'ils effectuent la surveillance automatiquement, tandis que d'autres nécessitent une intervention manuelle. Nous vous recommandons d'automatiser le plus possible les tâches de supervision.

Outils de surveillance automatisés dans AWS

Vous pouvez utiliser les outils de surveillance automatique pour surveiller API Gateway et signaler un problème éventuel :

- Amazon CloudWatch Alarms : surveillez une seule métrique sur une période que vous spécifiez et effectuez une ou plusieurs actions en fonction de la valeur de la métrique par rapport à un seuil donné sur un certain nombre de périodes. L'action est une notification envoyée à une rubrique Amazon Simple Notification Service (Amazon SNS) ou à une politique Amazon EC2 Auto Scaling. CloudWatch les alarmes n'appellent pas d'actions simplement parce qu'elles sont dans un état

particulier ; l'état doit avoir changé et être maintenu pendant un certain nombre de périodes. Pour plus d'informations, consultez Surveillance de l'exécution de l'API REST avec CloudWatch les métriques Amazon.

- Amazon CloudWatch Logs — Surveillez, stockez et accédez à vos fichiers journaux depuis AWS CloudTrail ou d'autres sources. Pour plus d'informations, voir Qu'est-ce que CloudWatch les journaux ? dans le guide de CloudWatch l'utilisateur Amazon.
- Amazon EventBridge (anciennement CloudWatch Events) : associez les événements et acheminez-les vers une ou plusieurs fonctions ou flux cibles afin d'apporter des modifications, de capturer des informations d'état et de prendre des mesures correctives. Pour plus d'informations, consultez Qu'est-ce qu'Amazon EventBridge ? dans le guide de EventBridge l'utilisateur.
- AWS CloudTrail Surveillance des journaux : partagez les fichiers journaux entre les comptes, surveillez les fichiers CloudTrail CloudWatch journaux en temps réel en les envoyant à Logs, écrivez des applications de traitement des journaux en Java et vérifiez que vos fichiers journaux n'ont pas changé après leur livraison par CloudTrail. Pour plus d'informations, consultez la section Utilisation des fichiers CloudTrail journaux dans le guide de AWS CloudTrail l'utilisateur.

Outils de surveillance manuelle

Un autre élément important de la surveillance d'API Gateway consiste à surveiller manuellement les éléments non couverts par les CloudWatch alarmes. L'API Gateway et CloudWatch les autres tableaux de bord de AWS console fournissent une at-a-glance vue de l'état de votre AWS environnement. Nous recommandons de consulter également les fichiers journaux sur l'exécution des API.

- Le tableau de bord API Gateway présente les statistiques suivantes pour une étape d'API donnée sur une durée définie :
  - Appels d'API
  - Accès au cache, uniquement lorsque la mise en cache des API est activée.
  - Échec de cache, uniquement lorsque la mise en cache des API est activée.
  - Latence
  - Latence d'intégration
  - Erreur 4XX
  - Erreur 5XX
- La page d' CloudWatch accueil indique :
  - Alarmes et statuts en cours

- Graphiques des alarmes et des ressources
- Statut d'intégrité du service

En outre, vous pouvez CloudWatch effectuer les opérations suivantes :

- Créer des tableaux de bord personnalisés pour surveiller les services de votre choix
- Représenter graphiquement les données de métriques pour résoudre les problèmes et découvrir les tendances
- Recherchez et parcourez tous les indicateurs de vos AWS ressources
- Créer et modifier des alarmes pour être informé des problèmes

Création d' CloudWatch alarmes pour surveiller API Gateway

Vous pouvez créer une CloudWatch alarme qui envoie un message Amazon SNS lorsque l'alarme change d'état. Une alarme surveille une seule métrique pendant une durée que vous définissez et exécute une ou plusieurs actions en fonction de la valeur de la métrique par rapport à un seuil donné pendant un certain nombre de périodes. L'action est une notification envoyée à une rubrique Amazon SNS ou à une politique Auto Scaling. Les alarmes déclenchent des actions uniquement pour les changements d'état prolongés. CloudWatch les alarmes n'appellent pas d'actions simplement parce qu'elles sont dans un état particulier ; l'état doit avoir changé et être maintenu pendant un certain nombre de périodes.

Configuration de la CloudWatch journalisation pour une API REST dans API Gateway

Pour résoudre les problèmes liés à l'exécution des demandes ou à l'accès des clients à votre API, vous pouvez activer Amazon CloudWatch Logs pour enregistrer les appels d'API. Pour plus d'informations sur CloudWatch, voirthe section called “CloudWatch métriques”.

CloudWatch formats de journal pour API Gateway

Il existe deux types de connexion à l'API CloudWatch : la journalisation de l'exécution et la journalisation des accès. Lors de la journalisation de l'exécution, API Gateway gère les CloudWatch journaux. Le processus comprend la création de groupes de journaux et de flux de journaux, et la génération de rapports dans les flux de journaux sur toutes les demandes et réponses des utilisateurs.

Les données enregistrées incluent les erreurs ou les traces d'exécution (telles que les valeurs des paramètres de demande ou de réponse ou les charges utiles), les données utilisées par les autorisateurs Lambda (anciennement appelés autorisateurs personnalisés), si les clés d'API sont requises, si les plans d'utilisation sont activés, et d'autres informations. API Gateway supprime les en-têtes d'autorisation, les valeurs des clés d'API et les paramètres de demande sensibles similaires à partir des données enregistrées.

Lorsque vous déployez une API, API Gateway crée un groupe de journaux et, sous celui-ci, les flux de journaux. Le groupe de journaux est nommé selon le format `API-Gateway-Execution-Logs_{rest-api-id}/{stage_name}`. Dans chaque groupe de journaux, les journaux sont ensuite divisés en flux de journaux, qui sont ordonnés par Heure du dernier événement à mesure que les données sont rapportées.

Dans la journalisation des accès, vous, en tant que développeur d'API, souhaitez enregistrer qui a accédé à votre API et comment l'appelant à eu accès à l'API. Vous pouvez créer votre propre groupe de journaux ou en choisir un existant, qui peut être géré par API Gateway. Pour spécifier les détails d'accès, vous sélectionnez `$context`des variables, un format de journal et une destination de groupe de journaux.

Le format du journal d'accès doit inclure au moins `$context.requestId` ou `$context.extendedRequestId`. La meilleure pratique consiste à inclure `$context.requestId` et `$context.extendedRequestId` dans le format de votre journal.

`$context.requestId`

Cela enregistre la valeur dans l'`x-amzn-RequestId`en-tête. Les clients peuvent écraser la valeur de l'en-tête `x-amzn-RequestId` avec une valeur au format d'un identifiant unique universel (UUID). API Gateway renvoie cet ID de demande dans l'en-tête de réponse `x-amzn-RequestId`. API Gateway remplace les identifiants de demande remplacés qui ne sont pas au format UUID par ceux de vos journaux d'*`UUID`*`_REPLACED_INVALID_REQUEST_ID`accès.

`$context.extendedRequestId`

L'ExtendedRequestID est un identifiant unique généré par API Gateway. API Gateway renvoie cet ID de demande dans l'en-tête de réponse `x-amz-apigw-id`. Un appelant API ne peut pas fournir ni remplacer cet ID de demande. Vous devrez peut-être fournir cette valeur au AWS Support pour résoudre les problèmes liés à votre API. Pour plus d'informations, consultez the section called "`$context`Variables pour les modèles de données, les autorisateurs, les modèles de mappage et la journalisation des CloudWatch accès".

> **Note**
>
> Seules `$context` les variables sont prises en charge.

Choisissez un format de journal également adopté par votre backend analytique, comme Common Log Format (CLF), JSON, XML ou CSV. Vous pouvez ensuite y renseigner les journaux d'accès directement pour que vos mesures soient calculées et renvoyées. Pour définir le format du journal, définissez l'ARN du groupe de journaux sur la propriété accessLogSettings/DestinationArn de la scène. Vous pouvez obtenir un ARN de groupe de journaux dans la CloudWatch console. Pour définir le format du journal d'accès, définissez le format choisi sur la propriété accessLogSetting/format de la scène.

Quelques exemples de certains formats de journaux d'accès utilisés couramment sont affichés dans la console API Gateway et répertoriés ci-dessous.

- `CLF` (Format de journal commun):

```
$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 (valeurs séparées par des virgules) :

```
$context.identity.sourceIp,$context.identity.caller,$context.identity.user,
$context.requestTime,$context.httpMethod,$context.resourcePath,$context.protocol,
$context.status,$context.responseLength,$context.requestId,$context.extendedRequestId
```

Autorisations pour la CloudWatch journalisation

Pour activer CloudWatch les journaux, vous devez autoriser API Gateway à lire et à écrire les journaux CloudWatch de votre compte. La stratégie gérée `AmazonAPIGatewayPushToCloudWatchLogs` (avec un ARN `arn:aws:iam::aws:policy/service-role/AmazonAPIGatewayPushToCloudWatchLogs`) possède toutes les autorisations nécessaires :

```
{
    "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 appelle AWS Security Token Service pour assumer le rôle IAM. Assurez-vous donc que celui-ci AWS STS est activé pour la région. Pour plus d'informations, consultez la section Gestion du AWS STS dans une AWS région.

Pour accorder ces autorisations à votre compte, créez un rôle IAM en `apigateway.amazonaws.com` tant qu'entité de confiance, associez la politique précédente au rôle IAM et définissez l'ARN du rôle IAM sur la cloudWatchRole propriété Arn de votre compte. Vous devez définir la cloudWatchRolepropriété Arn séparément pour chaque AWS région dans laquelle vous souhaitez activer CloudWatch les journaux.

Si vous recevez un message d'erreur lors de la définition de l'ARN du rôle IAM, vérifiez les paramètres de votre AWS Security Token Service compte pour vous assurer qu' AWS STS il est activé dans la région que vous utilisez. Pour plus d'informations sur l'activation AWS STS, consultez la section Gestion du AWS STS dans une AWS région dans le guide de l'utilisateur IAM.

Configuration de la journalisation des CloudWatch API à l'aide de la console API Gateway

Pour configurer la journalisation de CloudWatch l'API, vous devez avoir déployé l'API sur une étape. Vous devez également avoir configuré un ARN de rôle CloudWatch Logs approprié pour votre compte.

1. Connectez-vous à la console API Gateway à l'adresse : https://console.aws.amazon.com/apigateway.
2. Dans le volet de navigation principal, choisissez Paramètres, puis sous Journalisation, choisissez Modifier.
3. Pour l'ARN du rôle de CloudWatch journal, entrez l'ARN d'un rôle IAM avec les autorisations appropriées. Vous devez le faire une fois pour chaque utilisateur Compte AWS qui crée des API à l'aide d'API Gateway.
4. Dans le volet de navigation principal, choisissez API, puis effectuez l'une des opérations suivantes :
   a. Choisissez une API existante, puis une étape.
   b. Créez une API et déployez-la dans une étape.

5. Dans le volet de navigation principal, choisissez Étapes.
6. Dans la section Journaux et suivi, choisissez Modifier.
7. Pour activer la journalisation de l'exécution :

   a. Sélectionnez un niveau de journalisation dans le menu déroulant CloudWatch Logs. Les niveaux de journalisation sont les suivants :

      - Désactivé — La journalisation n'est pas activée pour cette étape.
      - Erreurs uniquement : la journalisation n'est activée que pour les erreurs.
      - Journaux d'erreurs et d'informations : la journalisation est activée pour tous les événements.
      - Journaux complets des demandes et des réponses : la journalisation détaillée est activée pour tous les événements. Cela peut être utile pour dépanner les API, mais peut entraîner la consignation de données sensibles.

      > **Note**
      >
      > Nous vous recommandons de ne pas utiliser les Journaux complets des requêtes et des réponses pour les API de production.

   b. Si vous le souhaitez, sélectionnez Mesures détaillées pour activer les CloudWatch mesures détaillées.

   Pour plus d'informations sur CloudWatch les métriques, consultezthe section called “CloudWatch métriques”.
8. Pour activer la journalisation des accès :

   a. Activez Journalisation des accès personnalisée.

   b. Pour ARN de destination des journaux d'accès, entrez l'ARN d'un groupe de journaux. Le format ARN est le suivant : `arn:aws:logs:{region}:{account-id}:log-group:log-group-name`.

   c. Dans Format des journaux, entrez un format de journal. Vous pouvez choisir CLF, JSON, XML ou CSV. Pour en savoir plus sur les exemples de formats de journal, consultez the section called “CloudWatch formats de journal pour API Gateway”.
9. Sélectionnez Enregistrer les modifications.

> **Note**
>
> Vous pouvez activer la journalisation de l'exécution et la journalisation de l'accès indépendamment l'une de l'autre.

API Gateway est maintenant prêt à enregistrer les demandes adressées à votre API. Vous n'avez pas besoin de redéployer l'API lorsque vous mettez à jour les paramètres de l'étape, les journaux ou variables de l'étape.

Configurer la journalisation des CloudWatch API à l'aide de AWS CloudFormation

Utilisez l'exemple de AWS CloudFormation modèle suivant pour créer un groupe de CloudWatch journaux Amazon Logs et configurer la journalisation de l'exécution et des accès pour une étape. Pour activer CloudWatch les journaux, vous devez autoriser API Gateway à lire et à écrire les journaux CloudWatch de votre compte. Pour en savoir plus, consultez Associer le compte au rôle IAM dans leGuide de l'utilisateur AWS CloudFormation .

```
 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
```

Journalisation des appels d'API vers Amazon Data Firehose

Pour résoudre les problèmes liés à l'accès des clients à votre API, vous pouvez enregistrer les appels d'API à Amazon Data Firehose. Pour plus d'informations sur Firehose, consultez Qu'est-ce qu'Amazon Data Firehose ? .

Pour la journalisation des accès, vous pouvez uniquement activer CloudWatch Firehose, mais vous ne pouvez pas activer les deux. Cependant, vous pouvez activer la journalisation CloudWatch des exécutions et Firehose pour la journalisation des accès.

Rubriques

- Formats de journal Firehose pour API Gateway
- Autorisations pour la journalisation de Firehose
- Configurer la journalisation des accès Firehose à l'aide de la console API Gateway

Formats de journal Firehose pour API Gateway

La journalisation Firehose utilise le même format que CloudWatch la journalisation.

Autorisations pour la journalisation de Firehose

Lorsque la journalisation des accès Firehose est activée sur une scène, API Gateway crée un rôle lié à un service dans votre compte si ce rôle n'existe pas déjà. Le rôle est nommé `AWSServiceRoleForAPIGateway` et la stratégie gérée `APIGatewayServiceRolePolicy` lui est attachée. Pour de plus amples informations sur les rôles liés à un service, veuillez consulter Utilisation des rôles liés à un service.

> **Note**
>
> Le nom de votre stream Firehose doit être. `amazon-apigateway-{your-stream-name}`

Configurer la journalisation des accès Firehose à l'aide de la console API Gateway

Pour configurer la journalisation d'API, vous devez avoir déployé l'API à une étape. Vous devez également avoir créé un stream Firehose.

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Effectuez l'une des actions suivantes :
   a. Choisissez une API existante, puis une étape.
   b. Créez une API et déployez-la à une étape.
3. Dans le volet de navigation principal, choisissez Étapes.
4. Dans la section Journaux et suivi, choisissez Modifier.
5. Pour activer la journalisation des accès à un stream Firehose :
   a. Activez Journalisation des accès personnalisée.
   b. Pour l'ARN de destination du journal d'accès, entrez l'ARN d'un flux Firehose. Le format ARN est le suivant : `arn:aws:firehose:{region}:{account-id}:deliverystream/amazon-apigateway-{your-stream-name}`.

      > **Note**
      >
      > Le nom de votre stream Firehose doit être. `amazon-apigateway-{your-stream-name}`

   c. Pour Format des journaux, entrez un format de journal. Vous pouvez choisir CLF, JSON, XML ou CSV. Pour en savoir plus sur les exemples de formats de journal, consultez the section called "CloudWatch formats de journal pour API Gateway".
6. Sélectionnez Enregistrer les modifications.

API Gateway est désormais prêt à enregistrer les requêtes adressées à votre API dans Firehose. Vous n'avez pas besoin de redéployer l'API lorsque vous mettez à jour les paramètres de l'étape, les journaux ou les variables de l'étape.

Suivi des demandes utilisateur vers les API REST à l'aide de X-Ray

Vous pouvez utiliser AWS X-Ray pour suivre et analyser les demandes des utilisateurs lorsqu'ils passent par vos API REST Amazon API Gateway vers les services sous-jacents. API Gateway prend en charge le suivi X-Ray pour tous les types de point de terminaison API REST Gateway : régional, optimisé pour la périphérie et privé. Vous pouvez utiliser X-Ray avec Amazon API Gateway dans toutes les AWS régions où X-Ray est disponible.

Comme X-Ray vous donne une end-to-end vue d'ensemble d'une demande, vous pouvez analyser les latences de vos API et de leurs services principaux. Vous pouvez utiliser une carte des services X-Ray pour afficher la latence de l'ensemble d'une demande et celle des services en aval intégrés à X-Ray. Vous pouvez également configurer des règles d'échantillonnage pour indiquer à X-Ray les demandes à enregistrer et les taux d'échantillonnage, selon les critères que vous spécifiez.

Si vous appelez une API Gateway à partir d'un service déjà en cours de suivi, API Gateway transmet le suivi, même si le suivi X-Ray n'est pas activé sur l'API.

Vous pouvez activer X-Ray pour une étape d'API à l'aide de la console API Gateway, ou à l'aide de l'API API Gateway ou de l'interface de ligne de commande.

Rubriques

Configuration AWS X-Ray avec les API REST API Gateway

Dans cette section, vous trouverez des informations détaillées sur la façon de configurer AWS X-Ray avec les API REST API Gateway.

Rubriques

Modes de suivi X-Ray pour API Gateway

Le chemin d'une demande via votre application est suivie avec un ID de suivi. Une trace collecte tous les segments générés par une seule requête, généralement une requête HTTP `GET` ou `POST`.

Il existe deux modes de suivi pour une API API Gateway :

- Suivi passif : il s'agit du paramètre par défaut si vous n'avez pas activé le suivi X-Ray sur une étape de l'API. Avec cette approche, l'API API Gateway est suivie uniquement si X-Ray a été activé sur un service en amont.
- Suivi actif : lorsque ce paramètre est appliqué à une étape d'API API Gateway, API Gateway crée automatiquement des exemples de demandes d'appels d'API, en fonction de l'algorithme d'échantillonnage spécifié par X-Ray.

  Lorsque le suivi actif est activé sur une étape, API Gateway crée un rôle lié à un service dans votre compte, si ce rôle n'existe pas déjà. Le rôle est nommé `AWSServiceRoleForAPIGateway`la stratégie gérée `APIGatewayServiceRolePolicy` est attachée au rôle. Pour de plus amples informations sur les rôles liés à un service, veuillez consulter Utilisation des rôles liés à un service.

  > **Note**
  >
  > X-Ray applique un algorithme d'échantillonnage pour s'assurer que le suivi est efficace, tout en fournissant un échantillon représentatif des demandes reçues par votre API. L'algorithme d'échantillonnage par défaut est de 1 demande par seconde, avec 5 % des demandes échantillonnées au-delà de cette limite.

Vous pouvez modifier le mode de suivi de votre API à l'aide de la console de gestion API Gateway, de la CLI API Gateway ou d'un AWS SDK.

Autorisations pour le suivi X-Ray

Lorsque vous activez le suivi X-Ray sur une étape, API Gateway crée un rôle lié à un service dans votre compte, si ce rôle n'existe pas déjà. Le rôle est nommé `AWSServiceRoleForAPIGateway`la stratégie gérée `APIGatewayServiceRolePolicy` est attachée au rôle. Pour de plus amples informations sur les rôles liés à un service, veuillez consulter Utilisation des rôles liés à un service.

Activation du suivi X-Ray dans la console API Gateway

Vous pouvez utiliser la console Amazon API Gateway pour activer le suivi actif sur une étape d'API.

Cette procédure suppose que vous ayez déjà déployé l'API à une étape.

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Choisissez votre API, puis dans le volet de navigation principal, choisissez Étapes.

3. Dans le volet Étapes, choisissez une étape.
4. Dans la section Journaux et suivi, choisissez Modifier.
5. Pour activer le suivi X-Ray, sélectionnez Suivi X-Ray pour activer le suivi X-Ray.
6. Sélectionnez Enregistrer les modifications.

Une fois que vous avez activé X-Ray pour l'étape d'API, vous pouvez utiliser la console de gestion X-Ray pour afficher les suivis et les cartes de service.

Activation du AWS X-Ray suivi à l'aide de l'API Gateway CLI

AWS CLI Pour activer le suivi X-Ray actif pour une étape d'API lorsque vous créez cette étape, appelez la `create-stage`commande comme dans l'exemple suivant :

```
aws apigateway create-stage \
    --rest-api-id {rest-api-id} \
    --stage-name {stage-name} \
    --deployment-id {deployment-id} \
    --region {region} \
    --tracing-enabled=true
```

Voici un exemple de sortie d'un appel réussi :

```
{
    "tracingEnabled": true,
    "stageName": {stage-name},
    "cacheClusterEnabled": false,
    "cacheClusterStatus": "NOT_AVAILABLE",
    "deploymentId": {deployment-id},
    "lastUpdatedDate": 1533849811,
    "createdDate": 1533849811,
    "methodSettings": {}
}
```

AWS CLI Pour désactiver le suivi X-Ray actif pour une étape d'API lorsque vous créez la phase, appelez la `create-stage`commande comme dans l'exemple suivant :

```
aws apigateway create-stage \
    --rest-api-id {rest-api-id} \
    --stage-name {stage-name} \
```

```
    --deployment-id {deployment-id} \
    --region {region} \
    --tracing-enabled=false
```

Voici un exemple de sortie d'un appel réussi :

```
{
    "tracingEnabled": false,
    "stageName": {stage-name},
    "cacheClusterEnabled": false,
    "cacheClusterStatus": "NOT_AVAILABLE",
    "deploymentId": {deployment-id},
    "lastUpdatedDate": 1533849811,
    "createdDate": 1533849811,
    "methodSettings": {}
}
```

AWS CLI Pour activer le suivi X-Ray actif pour une API déjà déployée, appelez la update-stagecommande comme suit :

```
aws apigateway update-stage \
    --rest-api-id {rest-api-id} \
    --stage-name {stage-name} \
    --patch-operations op=replace,path=/tracingEnabled,value=true
```

AWS CLI Pour désactiver le suivi X-Ray actif pour une API déjà déployée, appelez la update-stagecommande comme dans l'exemple suivant :

```
aws apigateway update-stage \
    --rest-api-id {rest-api-id} \
    --stage-name {stage-name} \
    --region {region} \
    --patch-operations op=replace,path=/tracingEnabled,value=false
```

Voici un exemple de sortie d'un appel réussi :

```
{
    "tracingEnabled": false,
    "stageName": {stage-name},
    "cacheClusterEnabled": false,
```

```
    "cacheClusterStatus": "NOT_AVAILABLE",
    "deploymentId": {deployment-id},
    "lastUpdatedDate": 1533850033,
    "createdDate": 1533849811,
    "methodSettings": {}
}
```

Une fois que vous avez activé X-Ray pour l'étape d'API, utilisez l'interface de ligne de commande X-Ray pour récupérer les informations de suivi. Pour plus d'informations, consultez la section Utilisation de l' AWS X-Ray API avec la AWS CLI.

Utilisation AWS X-Ray de cartes de service et de vues de trace avec API Gateway

Dans cette section, vous trouverez des informations détaillées sur l'utilisation des cartes de service et des vues de suivi AWS X-Ray avec API Gateway.

Pour plus d'informations sur la cartographie des services et les vues de suivi et leur interprétation, consultez la section AWS X-Ray Console.

Rubriques

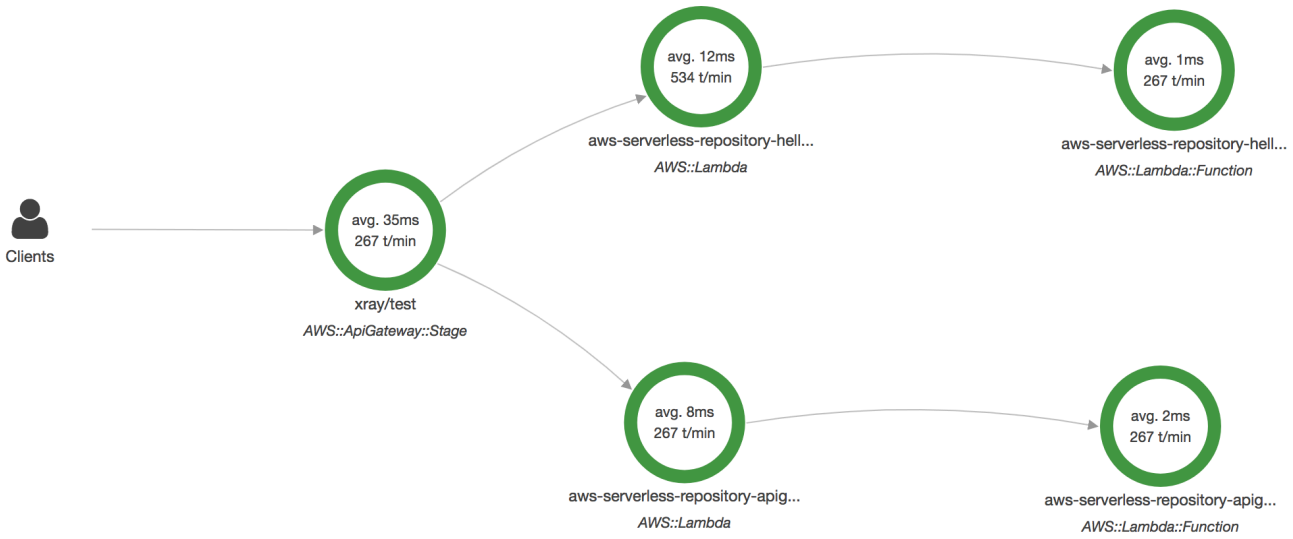
- Exemple de carte de service X-Ray
- Exemple de vue de suivi X-Ray

Exemple de carte de service X-Ray

AWS X-Ray les cartes de service affichent des informations sur votre API et tous ses services en aval. Quand X-Ray est activé pour une étape d'API dans API Gateway, un nœud contenant des informations sur le temps total passé dans le service API Gateway s'affiche dans la carte de service. Vous pouvez obtenir des informations détaillées sur le statut de réponse et un histogramme du temps de réponse de l'API pour la période sélectionnée. Pour les API intégrées à AWS des services tels qu' AWS Lambda Amazon DynamoDB, vous verrez davantage de nœuds fournissant des indicateurs de performance liés à ces services. Une cartographie des services est disponible pour chaque étape de l'API.

L'exemple suivant montre une cartographie des services pour l'étape `test` d'une API appelée `xray`. Cette API a une intégration Lambda avec une fonction d'autorisation Lambda et une fonction de backend Lambda. Les nœuds représentent le service API Gateway, le service Lambda et les deux fonctions Lambda.

Pour une explication détaillée de la structure de la carte de service, veuillez consulter Affichage de la carte de service.



Depuis la cartographie des services, vous pouvez zoomer afin de consulter un suivi de votre étape d'API. Le suivi affiche des informations détaillées concernant votre API, sous la forme de segments et de sous-segments. Par exemple, le suivi de la carte de service présentée ci-dessus inclut des segments pour le service Lambda et la fonction Lambda. Pour plus d'informations, reportez-vous AWS Lambda aux sections et AWS X-Ray.

Si vous sélectionnez un nœud ou un arc sur une carte de service X-Ray, la console X-Ray affiche un histogramme de distribution des latences. Vous pouvez utiliser un histogramme de latence pour voir de combien de temps a besoin un service pour traiter ses requêtes. Voici un histogramme de l'étape d'API Gateway nommée `xray/test` dans la précédente carte de service. Pour obtenir une explication détaillée des histogrammes de distribution de la latence, consultez la section Utilisation d'histogrammes des latences dans la console AWS X-Ray.

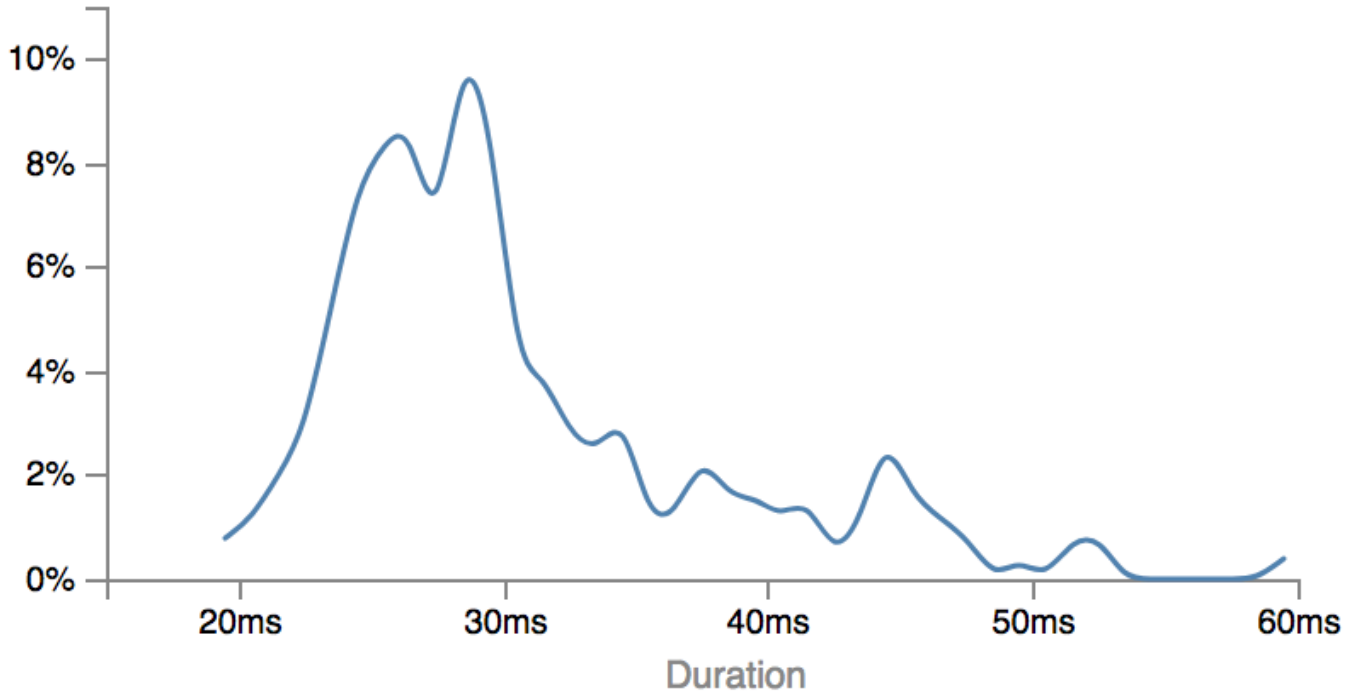
Service details

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.

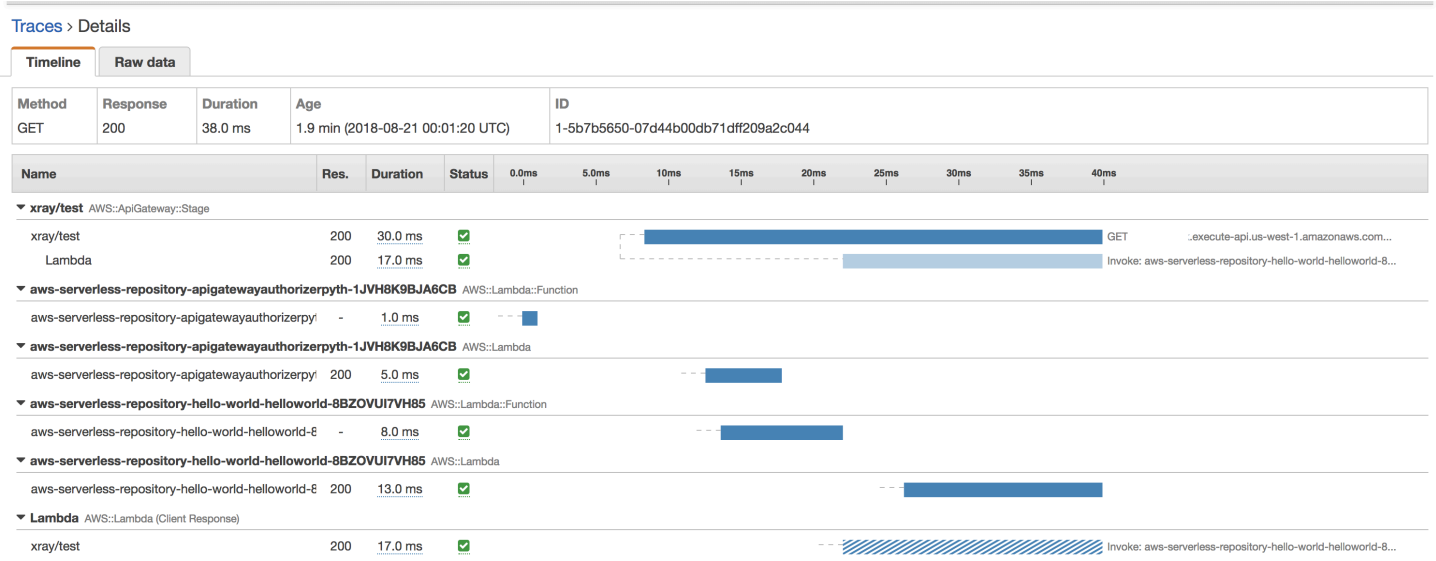
OK: 100% Error: 0%

Fault: 0% Throttle: 0%

Exemple de vue de suivi X-Ray

Le schéma suivant présente une vue de suivi générée pour l'exemple d'API décrit ci-dessus, avec une fonction de backend Lambda et une fonction de mécanisme d'autorisation Lambda. Une demande de méthode d'API réussie s'affiche avec le code de réponse 200.

Pour obtenir une explication détaillée des vues de suivi, veuillez consulter Affichage des suivis.



Configuration des règles AWS X-Ray d'échantillonnage pour les API API Gateway

Vous pouvez utiliser AWS X-Ray la console ou le SDK pour configurer les règles d'échantillonnage pour votre API Amazon API Gateway. Une règle d'échantillonnage spécifie quelles demandes X-Ray doit enregistrer pour votre API. En personnalisant les règles d'échantillonnage, vous pouvez contrôler la quantité de données que vous enregistrez, et modifier le comportement d'échantillonnage à la volée, sans modifier ni redéployer votre code.

Avant de spécifier vos règles d'échantillonnage X-Ray, veuillez consulter les rubriques suivantes dans le Manuel du développeur X-Ray :

- Configuration des règles d'échantillonnage dans la AWS X-Ray console
- Utilisation de règles d'échantillonnage avec l'API X-Ray

Rubriques

- Valeurs des options de règle d'échantillonnage X-Ray pour les API API Gateway

- Exemples de règles d'échantillonnage X-Ray

Valeurs des options de règle d'échantillonnage X-Ray pour les API API Gateway

Les options d'échantillonnage X-Ray suivantes sont pertinentes pour API Gateway. Les valeurs de chaîne peuvent comporter des caractères génériques pour correspondre à un seul caractère (?) ou à zéro caractère ou plus (*). Pour plus de détails, y compris une explication détaillée de la façon dont les paramètres de réservoir et de débit sont utilisés, voir Configuration des règles d'échantillonnage dans la AWS X-Ray console.

- Rule name (Nom de la règle) (chaîne) : nom unique de la règle.
- Priority (Priorité) (nombre entier compris entre 1 et 9999) : priorité de la règle d'échantillonnage. Les services évaluent les règles dans l'ordre croissant de priorité, et prennent une décision d'échantillonnage avec la première règle correspondante.
- Reservoir (Réservoir) (entier non négatif) : nombre fixe de demandes correspondantes à instrumenter par seconde, avant d'appliquer la fréquence fixe. Le réservoir n'est pas utilisé directement par les services, mais s'applique à tous les services qui utilisent la règle collectivement.
- Rate (Fréquence) (nombre compris entre 0 et 100) : pourcentage de demandes correspondantes à instrumenter une fois que le réservoir est épuisé.
- Service name (Nom du service) (chaîne) : nom de l'étape d'API, sous la forme ***{api-name}*/*{stage-name}***. Par exemple, si vous deviez déployer l'PetStoreexemple d'API sur une étape nomméetest, la valeur du nom de service à spécifier dans votre règle d'échantillonnage serait**pets/test**.
- Service type (Type de service) (chaîne) : pour une API API Gateway, vous pouvez spécifier **AWS::ApiGateway::Stage** ou **AWS::ApiGateway::***.
- Host (Hôte) (chaîne) : nom d'hôte de l'en-tête d'hôte HTTP. Définissez cette valeur sur ***** pour l'associer à tous les noms d'hôte. Vous pouvez également spécifier tout ou partie d'un nom d'hôte pour le faire correspondre (par exemple, **api.example.com** ou ***.example.com**).
- Resource ARN (ARN de la ressource) (chaîne) : ARN de l'étape de l'API, au format ; par exemple, **arn:aws:apigateway:*region*::/restapis/*api-id*/stages/*stage-name***.

  Le nom de l'étape peut être obtenu à partir de la console ou de l'interface de ligne de commande ou de l'API API Gateway. Pour plus d'informations sur les formats ARN, consultez Référence générale d'Amazon Web Services.
- HTTP method (Méthode HTTP) (chaîne) : la méthode à échantillonner (par exemple, **GET**).

- URL path (Chemin URL) (chaîne) : chemin URL de la demande.
- (facultatif) Attributes (Attributs) (clé et valeur) : en-têtes de la demande HTTP d'origine (par exemple, **`Connection`**, **`Content-Length`** ou **`Content-Type`**). Chaque attribut peut contenir jusqu'à 32 caractères.

Exemples de règles d'échantillonnage X-Ray

Exemple de règle d'échantillonnage n°1

Cette règle échantillonne toutes les requêtes `GET` pour l'API `testxray` à l'étape `test`.

- Nom de la règle — **`test-sampling`**
- Priorité — **`17`**
- Taille du réservoir — **`10`**
- Fréquence fixe — **`10`**
- Nom du service — **`testxray/test`**
- Type de service — **`AWS::ApiGateway::Stage`**
- Méthode HTTP — **`GET`**
- ARN de la ressource — **`*`**
- Hôte — **`*`**

Exemple de règle d'échantillonnage n°2

Cette règle échantillonne toutes les requêtes pour l'API `testxray` à l'étape `prod`.

- Nom de la règle — **`prod-sampling`**
- Priorité — **`478`**
- Taille du réservoir — **`1`**
- Fréquence fixe — **`60`**
- Nom du service — **`testxray/prod`**
- Type de service — **`AWS::ApiGateway::Stage`**
- Méthode HTTP — **`*`**
- ARN de la ressource — **`*`**
- Hôte — **`*`**

- Attributs — **{}**

Comprendre AWS X-Ray les traces pour les API Amazon API Gateway

Cette section décrit les segments de AWS X-Ray trace, les sous-segments et les autres champs de trace pour les API Amazon API Gateway.

Avant de lire cette section, veuillez consulter les rubriques suivantes dans le Manuel du développeur X-Ray :

- AWS X-Ray Console
- AWS X-Ray Documents segmentés
- Concepts X-Ray

Rubriques

- Exemples d'objets de suivi pour une API API Gateway
- Présentation des suivis

Exemples d'objets de suivi pour une API API Gateway

Cette section décrit quelques-uns des objets que vous pouvez voir dans le suivi d'une API API Gateway.

Annotations

Des annotations peuvent s'afficher dans les segments et les sous-segments. Elles sont utilisées comme expressions de filtrage dans les règles d'échantillonnage pour filtrer les suivis. Pour plus d'informations, consultez la section Configuration des règles d'échantillonnage dans la console AWS X-Ray.

Voici un exemple d'objet `annotations`, dans lequel une étape d'API est identifiée par l'ID d'API et le nom de l'étape d'API :

```
"annotations": {
    "aws:api_id": "a1b2c3d4e5",
    "aws:api_stage": "dev"
}
```

AWS données sur les ressources

L'objet `aws` s'affiche uniquement dans des segments. Voici un exemple d'objet qui `aws` correspondant à la règle d'échantillonnage par défaut. Pour une explication approfondie des règles d'échantillonnage, consultez la section Configuration de règles d'échantillonnage dans la console AWS X-Ray.

```
"aws": {
    "xray": {
        "sampling_rule_name": "Default"
    },
    "api_gateway": {
        "account_id": "123412341234",
        "rest_api_id": "a1b2c3d4e5",
        "stage": "dev",
        "request_id": "a1b2c3d4-a1b2-a1b2-a1b2-a1b2c3d4e5f6"
    }
}
```

Présentation des suivis

Voici un segment de suivi pour une étape d'API Gateway. Pour une explication détaillée des champs qui constituent le segment de trace, consultez la section Documents AWS X-Ray du segment dans le guide du AWS X-Ray développeur.

```
        {
            "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"
    }
```

Utilisation des API HTTP

Les API REST et les API HTTP sont toutes des produits d'API RESTful. Les API REST prennent en charge plus de fonctionnalités que les API HTTP, tandis que les API HTTP sont conçues avec un minimum de fonctionnalités afin de pouvoir être proposées à un prix inférieur. Pour plus d'informations, consultez the section called “Choix entre les API HTTP et les API REST ”.

Vous pouvez utiliser les API HTTP pour envoyer des demandes à des AWS Lambda fonctions ou à n'importe quel point de terminaison HTTP routable. Par exemple, vous pouvez créer une API HTTP qui s'intègre à une fonction Lambda sur le backend. Lorsqu'un client appelle votre API, API Gateway envoie la demande à la fonction Lambda et renvoie la réponse de la fonction au client.

Les API HTTP prennent en charge OpenID Connect et l'autorisation OAuth 2.0. Ils sont fournis avec la prise en charge intégrée du partage de ressources d'origine croisée (CORS) et des déploiements automatiques.

Vous pouvez créer des API HTTP à l'aide de la console de AWS gestion AWS CLI AWS CloudFormation, des API ou des SDK.

Rubriques

Développement d'une API HTTP dans API Gateway

Cette section fournit des détails sur les fonctionnalités d'API Gateway dont vous avez besoin pendant le développement de vos API API Gateway.

Au fur et à mesure que vous développez votre API API Gateway, vous décidez d'un certain nombre de caractéristiques de votre API. Ces caractéristiques dépendent du cas d'utilisation de votre API. Par exemple, vous pourriez vouloir autoriser uniquement certains clients à appeler votre API ou qu'elle soit disponible pour tout le monde. Vous pouvez souhaiter utiliser un appel d'API pour exécuter une fonction Lambda, créer une requête de base de données ou appeler une application.

Rubriques

Création d'une API HTTP

Pour créer une API fonctionnelle, vous devez disposer d'au moins un itinéraire, une intégration, une étape et un déploiement.

Les exemples suivants montrent comment créer une API avec une intégration AWS Lambda ou HTTP, une route et une étape par défaut configurée pour déployer automatiquement les modifications.

Ce guide suppose que vous connaissez déjà API Gateway et Lambda. Pour obtenir un guide plus détaillé, veuillez consulter Mise en route.

Rubriques

Créez une API HTTP à l'aide du AWS Management Console

1. Ouvrez la console API Gateway.
2. Sélectionnez Create API (Créer une API).
3. Sous API HTTP, choisissez Créer.
4. Choisissez Ajouter une intégration, puis choisissez une fonction AWS Lambda ou entrez un point de terminaison HTTP.
5. Dans Name (Nom), entrez le nom de votre API.

6. Choisissez Review and create.
7. Choisissez Créer.

Maintenant, votre API est prête à être appelée. Vous pouvez tester votre API en entrant son URL d'appel dans un navigateur, ou en utilisant Curl.

```
curl https://api-id.execute-api.us-east-2.amazonaws.com
```

Création d'une API HTTP à l'aide de la AWS CLI

Vous pouvez utiliser la création rapide pour créer une API avec une intégration Lambda ou HTTP, une route fourre-tout par défaut et une étape par défaut configurée pour déployer automatiquement les modifications. La commande suivante utilise la création rapide pour créer une API qui s'intègre à une fonction Lambda sur le backend.

> **Note**
>
> Pour appeler une intégration Lambda, API Gateway doit disposer des autorisations requises. Vous pouvez utiliser une stratégie basée sur les ressources ou un rôle IAM pour accorder des autorisations API Gateway permettant d'appeler une fonction Lambda. Pour en savoir plus, consultez la section AWS Lambda Permissions dans le guide du AWS Lambda développeur.

Example

```
aws apigatewayv2 create-api --name my-api --protocol-type HTTP --target
 arn:aws:lambda:us-east-2:123456789012:function:function-name
```

Maintenant, votre API est prête à être appelée. Vous pouvez tester votre API en entrant son URL d'appel dans un navigateur, ou en utilisant Curl.

```
curl https://api-id.execute-api.us-east-2.amazonaws.com
```

Utilisation des itinéraires pour les API HTTP

Route les demandes d'API entrantes directes vers les ressources dorsales. Les itinéraires se composent de deux parties : une méthode HTTP et un chemin de ressource (, par exempl, `GET /`

`pets`. Vous pouvez définir des méthodes HTTP spécifiques pour votre itinéraire. Vous pouvez également utiliser la méthode `ANY` pour faire correspondre toutes les méthodes que vous n'avez pas définies pour une ressource. Vous pouvez créer un itinéraire `$default` qui agit comme un fourre-tout pour les demandes qui ne correspondent à aucun autre itinéraire.

> **Note**
>
> API Gateway décode les paramètres de demande encodés par URL avant de les transmettre à votre intégration backend.

Utilisation des variables de chemin

Vous pouvez utiliser des variables de chemin dans les itinéraires des API HTTP.

Par exemple, l'itinéraire `GET /pets/{petID}` attrape une demande `GET` qu'un client soumet à `https://api-id.execute-api.us-east-2.amazonaws.com/pets/6`.

Une variable de chemin gourmandes capture toutes les ressources enfants d'un itinéraire. Pour créer une variable de chemin gourmand, ajoutez + au nom de la variable (`{proxy+}`, par exemple). La variable de chemin gourmande doit se trouver à la fin du chemin de ressource.

Utilisation des paramètres de chaîne de requête

Par défaut, API Gateway envoie les paramètres de chaîne de requête à votre intégration backend s'ils sont inclus dans une demande adressée à une API HTTP.

Par exemple, lorsqu'un client envoie une demande à `https://api-id.execute-api.us-east-2.amazonaws.com/pets?id=4&type=dog`, les paramètres de chaîne de requête `?id=4&type=dog` sont envoyés à votre intégration.

Utilisation de l'itinéraire `$default`

L'itinéraire `$default` attrape les demandes qui ne correspondent pas explicitement aux autres itinéraires de votre API.

Lorsque l'itinéraire `$default` reçoit une demande, API Gateway envoie le chemin de demande complet à l'intégration. Par exemple, vous pouvez créer une API avec seulement un itinéraire `$default` et l'intégrer à la méthode `ANY` avec le point de terminaison HTTP `https://petstore-`

demo-endpoint.execute-api.com. Lorsque vous envoyez une demande à https://*api-id*.execute-api.*us-east-2*.amazonaws.com/store/checkout, API Gateway envoie une demande à https://petstore-demo-endpoint.execute-api.com/store/checkout.

Pour en savoir plus sur les intégrations HTTP, consultez Utilisation des intégrations de proxy HTTP pour les API HTTP.

Routage des demandes d'API

Lorsqu'un client envoie une demande d'API, API Gateway détermine d'abord à quelle étape doit être acheminée la demande. Si la demande correspond explicitement à une étape, API Gateway envoie la demande à cette dernière. Si aucune étape ne correspond entièrement à la demande, API Gateway envoie la demande à l'étape $default. S'il n'y a pas d'$defaultétape, l'API revient {"message":"Not Found"} et ne génère pas de CloudWatch journaux.

Après avoir sélectionné une étape, API Gateway sélectionne un itinéraire. API Gateway sélectionne l'itinéraire avec la correspondance la plus spécifique, en utilisant les priorités suivantes :

1. Correspondance complète pour un itinéraire et une méthode.
2. Correspondance pour une route et une méthode avec une variable de chemin gourmande ({proxy+}).
3. L'itinéraire $default

Si aucun itinéraire ne correspond à une demande, API Gateway retourne {"message":"Not Found"} au client.

Par exemple, examinez une API avec une étape $default et les exemples d'itinéraires suivants :

1. GET /pets/dog/1
2. GET /pets/dog/{id}
3. GET /pets/{proxy+}
4. ANY /{proxy+}
5. $default

Le tableau suivant résume la façon dont API Gateway achemine les demandes vers les itinéraires de l'exemple.

Requête	Itinéraire sélectionné	Explication
`GET https://api-id.execute-api. region.amazonaws.com/pets/dog/1`	`GET /pets/dog/1`	La demande correspond entièrement à cet itinéraire statique.
`GET https://api-id.execute-api. region.amazonaws.com/pets/dog/2`	`GET /pets/dog/{id}`	La demande correspond entièrement à cet itinéraire.
`GET https://api-id.execute-api. region.amazonaws.com/pets/cat/1`	`GET /pets/{proxy+}`	La demande ne correspond pas entièrement à un itinéraire. L'itinéraire avec une méthode `GET` et une variable de chemin gourmande attrape cette requête.
`POST https://api-id.execute-api. region.amazonaws.com/test/5`	`ANY /{proxy+}`	La méthode `ANY` correspond à toutes les méthodes que vous n'avez pas définies pour un itinéraire. Les itinéraires avec des variables de chemin gourmandes ont une priorité plus élevée que l'itinéraire `$default`.

Contrôle et gestion de l'accès à une API HTTP dans API Gateway

API Gateway prend en charge plusieurs mécanismes pour contrôler et gérer l'accès à votre API :

- Les autorisateurs Lambda utilisent les fonctions Lambda pour contrôler l'accès aux API. Pour de plus amples informations, veuillez consulter Travailler avec les AWS Lambda autorisateurs pour les API HTTP.

- Les mécanismes d'autorisation JWT utilisent des jetons Web JSON pour contrôler l'accès aux API. Pour plus d'informations, consultez Contrôle de l'accès aux API HTTP avec les mécanismes d'autorisation JWT.
- Les rôles et politiques AWS IAM standard offrent des contrôles d'accès flexibles et robustes. Vous pouvez utiliser les rôles et les stratégies IAM afin de contrôler qui peut créer et gérer vos API, mais aussi qui peut les appeler. Pour plus d'informations, voir Utilisation de l'autorisation IAM.

Travailler avec les AWS Lambda autorisateurs pour les API HTTP

Vous utilisez un mécanisme d'autorisation Lambda pour contrôler l'accès à votre API HTTP à l'aide d'une fonction Lambda. Ensuite, lorsqu'un client appelle votre API, API Gateway appelle votre fonction Lambda. API Gateway utilise la réponse de votre fonction Lambda pour déterminer si le client peut accéder à votre API.

Version du format de charge utile

La version du format de charge utile de l'autorisation spécifie le format des données qu'API Gateway envoie à un mécanisme d'autorisation Lambda, et la façon dont API Gateway interprète la réponse de Lambda. Si vous ne spécifiez pas de version de format de charge utile, la dernière version est AWS Management Console utilisée par défaut. Si vous créez un autorisateur Lambda à l'aide du AWS CLI, ou d'un SDK AWS CloudFormation, vous devez spécifier un. `authorizerPayloadFormatVersion` Les valeurs prises en charge sont `1.0` et `2.0`.

Si vous avez besoin de compatibilité avec les API REST, utilisez la version `1.0`.

Les exemples suivants montrent la structure de chaque version de format de charge utile.

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"
  }
}
```

Format de réponse du mécanisme d'autorisation Lambda

La version du format de charge utile détermine également la structure de la réponse que vous devez renvoyer à partir de votre fonction Lambda.

Réponse de la fonction Lambda pour le format 1.0

Si vous choisissez la version de format `1.0`, les mécanismes d'autorisation Lambda doivent renvoyer une stratégie IAM qui autorise ou refuse l'accès à votre itinéraire d'API. Vous pouvez utiliser la syntaxe de stratégie standard IAM dans la stratégie. Pour obtenir des exemples de stratégies IAM, consultez the section called “ Contrôler l'accès pour l'appel d'une API”. Vous pouvez transmettre des propriétés de contexte aux intégrations Lambda ou accéder aux journaux avec `$context.authorizer.`*`property`*. L'objet `context` est facultatif et `claims` est un espace réservé qui ne peut être utilisé comme objet contextuel. Pour en savoir plus, consultez la section the section called “Variables de journalisation”.

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"
  }
}
```

Réponse de la fonction Lambda pour le format 2.0

Si vous choisissez la version de format `2.0`, vous pouvez renvoyer une valeur booléenne ou une stratégie IAM qui utilise la syntaxe de stratégie IAM standard de votre fonction Lambda. Pour renvoyer une valeur booléenne, activez des réponses simples pour le mécanisme d'autorisation. Les exemples suivants montrent le format que vous devez coder pour retourner votre fonction Lambda. L'objet `context` est facultatif. Vous pouvez transmettre des propriétés de contexte aux intégrations Lambda ou accéder aux journaux avec `$context.authorizer.`*`property`*. Pour en savoir plus, consultez la section the section called “Variables de journalisation”.

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"
  }
```

```
}
```

Exemple de fonctions d'autorisation Lambda

Les exemples de fonction Lambda Node.js suivants illustrent les formats de réponse requis que vous devez renvoyer à partir de votre fonction Lambda pour la version du format de charge utile `2.0`.

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
```

Sources d'identité

Si vous le souhaitez, vous pouvez spécifier des sources d'identité pour un mécanisme d'autorisation Lambda. Les sources d'identité spécifient l'emplacement des données requises pour autoriser une demande. Par exemple, vous pouvez spécifier des valeurs d'en-tête ou de chaîne de requête en tant que sources d'identité. Si vous spécifiez des sources d'identité, les clients doivent les inclure dans la demande. Si la demande du client n'inclut pas les sources d'identité, API Gateway n'appelle pas votre mécanisme d'autorisation Lambda et le client reçoit une erreur `401`. Les sources d'identité suivantes sont prises en charge :

Expressions de sélection

Type	Exemple	Remarques
Valeur d'en-tête	$request.header.*nom*	Les noms d'en-tête ne sont pas sensibles à la casse.
Valeur de chaîne de requête	$request.querystring.*nom*	Les noms de chaîne de requête sont sensibles à la casse.
Variable de contexte	$context.*Nom_variable*	Valeur d'une variable de contexte prise en charge.
Variable d'étape	$stageVariables.*Nom_variable*	Valeur d'une variable d'étape.

Mise en cache des réponses du mécanisme d'autorisation

Vous pouvez activer la mise en cache pour un autorisateur Lambda en spécifiant un. authorizerResultTtlInSeconds Lorsque la mise en cache est activée pour un mécanisme d'autorisation, API Gateway utilise les sources d'identité de ce mécanisme comme clé de cache. Si un client spécifie les mêmes paramètres dans les sources d'identité au sein de la TTL configurée, API Gateway utilise le résultat du mécanisme d'autorisation mis en cache, plutôt que d'appeler votre fonction Lambda.

Pour activer la mise en cache, votre mécanisme d'autorisation doit avoir au moins une source d'identité.

Si vous activez des réponses simples pour un mécanisme d'autorisation, la réponse de ce mécanisme autorise ou refuse totalement toutes les demandes d'API qui correspondent aux valeurs de source d'identité mises en cache. Pour obtenir des autorisations plus détaillées, désactivez les réponses simples et renvoyez une stratégie IAM.

Par défaut, API Gateway utilise la réponse du mécanisme d'autorisation mise en cache pour tous les itinéraires d'une API utilisant le mécanisme d'autorisation. Pour mettre en cache les réponses par route, ajoutez `$context.routeKey` aux sources d'identité de votre mécanisme d'autorisation.

Création d'un mécanisme d'autorisation Lambda

Lorsque vous créez un mécanisme d'autorisation Lambda, vous spécifiez la fonction Lambda qu'API Gateway doit utiliser. Vous devez accorder l'autorisation à API Gateway d'appeler la fonction Lambda à l'aide de la stratégie de ressource de la fonction ou d'un rôle IAM. Dans cet exemple, nous mettons à jour la stratégie de ressource pour la fonction afin qu'elle accorde à API Gateway l'autorisation d'appeler notre fonction Lambda.

```
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
```

La commande suivante accorde l'autorisation à API Gateway d'appeler votre fonction Lambda. Si API Gateway n'a pas l'autorisation d'appeler votre fonction, les clients reçoivent un `500 Internal Server Error`.

```
aws lambda add-permission \
    --function-name my-authorizer-function \
    --statement-id apigateway-invoke-permissions-abc123 \
    --action lambda:InvokeFunction \
    --principal apigateway.amazonaws.com \
    --source-arn "arn:aws:execute-api:us-west-2:123456789012:api-
id/authorizers/authorizer-id"
```

Une fois que vous avez créé un mécanisme d'autorisation et que vous avez accordé à API Gateway l'autorisation de l'appeler, mettez à jour votre itinéraire de manière à l'utiliser.

```
aws apigatewayv2 update-route \
    --api-id abcdef123 \
    --route-id acd123 \
    --authorization-type CUSTOM \
    --authorizer-id def123
```

Dépannage des autorisations Lambda

Si API Gateway ne peut pas appeler votre autorisation Lambda, ou si votre mécanisme d'autorisation Lambda renvoie une réponse dans un format non valide, les clients reçoivent un `500 Internal Server Error`.

Pour résoudre les erreurs, activez la journalisation des accès pour votre étape d'API. Incluez la variable de journalisation `$context.authorizer.error` dans votre format de journal.

Si les journaux indiquent qu'API Gateway n'a pas l'autorisation d'appeler votre fonction, mettez à jour la stratégie de ressources de votre fonction ou fournissez un rôle IAM pour accorder à API Gateway l'autorisation d'appeler votre mécanisme d'autorisation.

Si les journaux indiquent que votre fonction Lambda renvoie une réponse non valide, vérifiez qu'elle renvoie une réponse dans le format requis.

Contrôle de l'accès aux API HTTP avec les mécanismes d'autorisation JWT

Vous pouvez utiliser JSON Web Tokens (JWT) dans le cadre des frameworks OpenID Connect (OIDC) et OAuth 2.0 pour restreindre l'accès client à vos API.

Si vous configurez un mécanisme d'autorisation JWT pour un itinéraire de votre API, API Gateway valide les JWT que les clients soumettent avec des demandes d'API. API Gateway autorise ou refuse les demandes basées sur la validation de jeton, et éventuellement, les étendues dans le jeton. Si vous configurez des étendues pour un routage, le jeton doit inclure au moins une des étendues du routage.

Vous pouvez configurer des autorisations distinctes pour chaque itinéraire d'une API ou utiliser le même mécanisme d'autorisation pour plusieurs itinéraires.

> **Note**
>
> Il n'y a pas de mécanisme standard pour différencier les jetons d'accès JWT des autres types de JWT tels que les jetons OpenID Connect ID. À moins que vous n'ayez besoin de jetons d'ID pour l'autorisation de l'API, nous vous recommandons de configurer vos itinéraires pour qu'ils requièrent des étendues d'autorisation. Vous pouvez également configurer vos mécanismes d'autorisation JWT pour exiger des émetteurs ou des audiences que votre fournisseur d'identité utilise uniquement lors de l'émission de jetons d'accès JWT.

Autoriser les demandes d'API avec un autorisateur JWT

API Gateway utilise le workflow général suivant pour autoriser les demandes d'itinéraires configurés pour utiliser un mécanisme d'autorisation JWT.

1. Vérifiez `identitySource` pour un jeton. `identitySource` peut inclure uniquement le jeton, ou le jeton préfixé avec `Bearer`.
2. Décodez le jeton.
3. Vérifiez l'algorithme et la signature du jeton en utilisant la clé publique récupérée auprès du de l'émetteu `jwks_uri`. Actuellement, seuls les algorithmes basés sur RSA sont pris en charge. API Gateway peut mettre en cache la clé publique pendant deux heures. Lorsque vous effectuez la rotation des clés, une bonne pratique consiste à prévoir un délai de grâce pendant lequel les anciennes et les nouvelles clés sont valides.
4. Validez les demandes. API Gateway évalue les demandes de jeton suivantes :
   - `kid` : le jeton doit avoir une demande d'en-tête qui correspond à la clé du `jwks_uri` ayant signé le jeton.
   - `iss` : doit correspondre à l'`issuer` configuré pour le mécanisme d'autorisation.

- `aud` ou `client_id` : doit correspondre à l'une des entrées `audience` configurées pour le mécanisme d'autorisation. API Gateway `client_id` ne valide que s'il n'`aud`est pas présent. Lorsque `aud` les deux `client_id` sont présents, API Gateway évalue. `aud`
- `exp` – doit être postérieure à l'heure actuelle (UTC).
- `nbf` – doit être antérieure à l'heure actuelle(UTC).
- `iat` – doit être antérieure à l'heure actuelle(UTC).
- `scope` ou `scp` : le jeton doit inclure au moins une des portées dans les `authorizationScopes` de l'itinéraire.

Si l'une de ces étapes échoue, API Gateway refuse la demande d'API.

Après avoir validé le JWT, API Gateway transmet les revendications du jeton à l'intégration de l'itinéraire d'API. Les ressources backend, telles que les fonctions Lambda, peuvent accéder aux revendications JWT. Par exemple, si le JWT inclut une revendication d'identité `emailID`, elle est disponible pour une intégration Lambda dans `$event.requestContext.authorizer.jwt.claims.emailID`. Pour plus d'informations sur la charge utile qu'API Gateway envoie aux intégrations Lambda, veuillez consulter the section called "AWS Lambda intégrations".

Créer un mécanisme d'autorisation JWT

Avant de créer un mécanisme d'autorisation JWT, vous devez enregistrer une application cliente auprès d'un fournisseur d'identité. Vous devez également avoir créé une API HTTP. Pour obtenir des exemples de création d'une API HTTP, veuillez consulter Création d'une API HTTP.

Créez un autorisateur JWT à l'aide de la console

Les étapes suivantes montrent comment créer un autorisateur JWT à l'aide de la console.

Pour créer un autorisateur JWT à l'aide de la console

1. Connectez-vous à la console API Gateway à l'adresse : https://console.aws.amazon.com/apigateway.
2. Choisissez une API HTTP.
3. Dans le volet de navigation principal, choisissez Authorization.
4. Choisissez l'onglet Gérer les autorisateurs.
5. Choisissez Créer.

6. Pour le type d'autorisateur, choisissez JWT.
7. Configurez votre autorisateur JWT et spécifiez une source d'identité qui définit la source du jeton.
8. Choisissez Créer.

Créez un autorisateur JWT à l'aide du AWS CLI

La AWS CLI commande suivante crée un autorisateur JWT. Pour `jwt-configuration`, spécifiez les paramètres `Audience` et `Issuer` pour votre fournisseur d'identité. Si vous utilisez Amazon Cognito comme fournisseur d'identité, c'est le `IssuerUrl` cas. `https://cognito-idp.`*`us-east-2`*`.amazonaws.com/`*`userPoolID`*

```
aws apigatewayv2 create-authorizer \
    --name authorizer-name \
    --api-id api-id \
    --authorizer-type JWT \
    --identity-source '$request.header.Authorization' \
    --jwt-configuration Audience=audience,Issuer=IssuerUrl
```

Créez un autorisateur JWT en utilisant AWS CloudFormation

Le AWS CloudFormation modèle suivant crée une API HTTP avec un autorisateur JWT qui utilise Amazon Cognito comme fournisseur d'identité.

La sortie du AWS CloudFormation modèle est une URL pour une interface utilisateur hébergée par Amazon Cognito où les clients peuvent s'inscrire et se connecter pour recevoir un JWT. Une fois qu'un client s'est connecté, il est redirigé vers votre API HTTP avec un jeton d'accès dans l'URL. Pour appeler l'API avec le jeton d'accès, remplacez # l'URL par a ? pour utiliser le jeton comme paramètre de chaîne de requête.

Exemple de AWS CloudFormation modèle

```
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
```

Mettre à jour un itinéraire pour utiliser un autorisateur JWT

Vous pouvez utiliser la console AWS CLI, le ou un AWS SDK pour mettre à jour un itinéraire afin d'utiliser un autorisateur JWT.

Mettre à jour un itinéraire pour utiliser un autorisateur JWT à l'aide de la console

Les étapes suivantes montrent comment mettre à jour un itinéraire pour utiliser l'autorisateur JWT à l'aide de la console.

Pour créer un autorisateur JWT à l'aide de la console

1. Connectez-vous à la console API Gateway à l'adresse : https://console.aws.amazon.com/apigateway.
2. Choisissez une API HTTP.

3. Dans le volet de navigation principal, choisissez Authorization.
4. Choisissez une méthode, puis sélectionnez votre autorisateur dans le menu déroulant, puis choisissez Joindre l'autorisateur.

Mettez à jour un itinéraire pour utiliser un autorisateur JWT à l'aide du AWS CLI

La commande suivante met à jour une route pour utiliser un autorisateur JWT à l'aide du. AWS CLI

```
aws apigatewayv2 update-route \
   --api-id api-id  \
   --route-id route-id  \
   --authorization-type JWT \
   --authorizer-id authorizer-id \
   --authorization-scopes user.email
```

Utilisation de l'autorisation IAM

Vous pouvez activer l'autorisation IAM pour les itinéraires des API HTTP. Lorsque l'autorisation IAM est activée, les clients doivent utiliser Signature Version 4 (SigV4) pour signer leurs demandes avec AWS des informations d'identification. API Gateway n'appelle votre route API que si le client dispose de l'autorisation `execute-api` pour l'itinéraire.

L'autorisation IAM pour les API HTTP est similaire à celle pour les API REST.

> **Note**
>
> Les stratégies de ressources ne sont actuellement pas prises en charge pour les API HTTP.

Pour obtenir des exemples de stratégie IAM qui accordent aux clients l'autorisation d'appeler des API, veuillez consulter the section called “ Contrôler l'accès pour l'appel d'une API”.

Activation d'une autorisation IAM pour un itinéraire

La AWS CLI commande suivante active l'autorisation IAM pour une route d'API HTTP.

```
aws apigatewayv2 update-route \
    --api-id abc123 \
    --route-id abcdef \
    --authorization-type AWS_IAM
```

Configuration des intégrations pour les API HTTP

Les intégrations connectent un itinéraire aux ressources backend. Les API HTTP prennent en charge les intégrations de proxy, de AWS service et de proxy HTTP Lambda. Par exemple, vous pouvez configurer une demande `POST` sur l'itinéraire `/signup` de votre API pour l'intégrer à une fonction Lambda qui gère l'inscription des clients.

Rubriques

- Utilisation des intégrations de AWS Lambda proxy pour les API HTTP
- Utilisation des intégrations de proxy HTTP pour les API HTTP
- Utilisation des intégrations AWS de services pour les API HTTP
- Utilisation des intégrations privées pour les API HTTP

Utilisation des intégrations de AWS Lambda proxy pour les API HTTP

Une intégration de proxy Lambda vous permet d'intégrer un itinéraire API à une fonction Lambda. Lorsqu'un client appelle votre API, API Gateway envoie la demande à la fonction Lambda et renvoie la réponse de la fonction au client. Pour obtenir des exemples de création d'une API HTTP, veuillez consulter Création d'une API HTTP.

Version du format de charge utile

La version du format de charge utile spécifie le format de l'événement qu'API Gateway envoie à une intégration Lambda, ainsi que la manière dont API Gateway interprète la réponse de Lambda. Si vous ne spécifiez pas de version de format de charge utile, la dernière version est AWS Management Console utilisée par défaut. Si vous créez une intégration Lambda à l'aide du AWS CLI AWS CloudFormation, ou d'un SDK, vous devez spécifier un. `payloadFormatVersion` Les valeurs prises en charge sont `1.0` et `2.0`.

Pour plus d'informations sur la façon de définir le`payloadFormatVersion`, consultez la section create-integration. Pour plus d'informations sur la façon de déterminer la valeur `payloadFormatVersion` d'une intégration existante, consultez get-integration

Différences de format de charge utile

La liste suivante indique les différences entre les versions du format `1.0` et du format `2.0` de charge utile :

- Le format `2.0` n'a pas de champs `multiValueHeaders` ou `multiValueQueryStringParameters`. Les en-têtes dupliqués sont combinés avec des virgules et inclus dans le champ `headers`. Les chaînes de requête en double sont combinées avec des virgules et incluses dans le champ `queryStringParameters`.
- Le format `2.0` `rawPath` a. Si vous utilisez un mappage d'API pour connecter votre stage à un nom de domaine personnalisé, vous `rawPath` ne fournirez pas la valeur de mappage d'API. Utilisez le format `1.0` et accédez `path` au mappage d'API pour votre nom de domaine personnalisé.
- Le format `2.0` inclut un nouveau champ `cookies`. Tous les en-têtes de cookie dans la demande sont combinés avec des virgules et ajoutés au champ `cookies`. Dans la réponse au client, chaque cookie devient un en-tête `set-cookie`.

Structure du format de charge utile

Les exemples suivants montrent la structure de chaque version de format de charge utile. Tous les noms d'en-tête sont en minuscules.

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
}
```

Format de réponse de fonction Lambda

La version du format de charge utile détermine la structure de la réponse que votre fonction Lambda doit renvoyer.

Réponse de la fonction Lambda pour le format 1.0

Avec la version `1.0` au format, les intégrations Lambda doivent renvoyer une réponse au format JSON suivant :

Example

```
{
    "isBase64Encoded": true|false,
    "statusCode": httpStatusCode,
    "headers": { "headername": "headervalue", ... },
    "multiValueHeaders": { "headername": ["headervalue", "headervalue2", ...], ... },
    "body": "..."
}
```

Réponse de la fonction Lambda pour le format 2.0

Avec la version de format `2.0`, API Gateway peut déduire le format de réponse pour vous. API Gateway fait les hypothèses suivantes si votre fonction Lambda renvoie JSON valide et ne renvoie pas un `statusCode`:

- `isBase64Encoded` est `false`.
- `statusCode` est `200`.
- `content-type` est `application/json`.
- `body` est la réponse de la fonction.

Les exemples suivants montrent la sortie d'une fonction Lambda et l'interprétation d'API Gateway.

Sortie de la fonction Lambda	Interprétation d'API Gateway
`"Hello from Lambda!"`	`{ "isBase64Encoded": false, "statusCode": 200, "body": "Hello from Lambda!", "headers": {`

Sortie de la fonction Lambda	Interprétation d'API Gateway

	<pre>"content-type": "application/
	json"
	}
	}</pre>

<pre>{ "message": "Hello from Lambda!" }</pre>	<pre>{
	"isBase64Encoded": false,
	"statusCode": 200,
	"body": "{ \"message\": \"Hello from
	Lambda!\" }",
	"headers": {
	"content-type": "application/
	json"
	}
	}</pre>

Pour personnaliser la réponse, votre fonction Lambda doit renvoyer une réponse au format suivant.

```
{
    "cookies" : ["cookie1", "cookie2"],
    "isBase64Encoded": true|false,
    "statusCode": httpStatusCode,
    "headers": { "headername": "headervalue", ... },
    "body": "Hello from Lambda!"
}
```

Utilisation des intégrations de proxy HTTP pour les API HTTP

Une intégration de proxy HTTP vous permet de connecter une route d'API à un point de terminaison HTTP routable publiquement. Avec ce type d'intégration, API Gateway passe l'intégralité de la demande et de la réponse entre le frontend et le backend.

Pour créer une intégration de proxy HTTP, fournissez l'URL d'un point de terminaison HTTP routable publiquement.

Intégration de proxy HTTP avec les variables de chemin

Vous pouvez utiliser des variables de chemin dans les itinéraires des API HTTP.

Par exemple, la route `/pets/{petID}` attrape les demandes faites à `/pets/6`. Vous pouvez référencer des variables de chemin dans l'URI d'intégration pour envoyer le contenu de la variable à une intégration. Par exemple : `/pets/extendedpath/{petID}`.

Vous pouvez utiliser des variables de chemin gourmandes pour attraper toutes les ressources enfants d'une route. Pour créer une variable de chemin gourmand, ajoutez + au nom de la variable (`{proxy+}`, par exemple).

Pour configurer une route avec une intégration de proxy HTTP qui attrape toutes les demandes, créez une route d'API avec une variable de chemin gourmande (par exemple, `/parent/{proxy+}`). Intégrez la route à un point de terminaison HTTP (par exemple, `https://petstore-demo-endpoint.execute-api.com/petstore/{proxy}`) sur la méthode `ANY`. La variable de chemin gourmande doit se trouver à la fin du chemin de ressource.

Utilisation des intégrations AWS de services pour les API HTTP

Vous pouvez intégrer votre API HTTP à des AWS services en utilisant des intégrations de premier ordre. Une intégration de première classe connecte une route API HTTP à une API de service AWS . Lorsqu'un client invoque une route soutenue par une intégration de premier ordre, API Gateway invoque une API de AWS service pour vous. Par exemple, vous pouvez utiliser des intégrations de premier ordre pour envoyer un message à une file d'attente Amazon Simple Queue Service ou pour démarrer une machine à AWS Step Functions états. Pour connaître les actions de service prises en charge, veuillez consulter the section called “AWS référence sur les intégrations de services”.

Mappage des paramètres de demande

Les intégrations de première classe ont des paramètres requis et des paramètres facultatifs. Vous devez configurer tous les paramètres requis pour créer une intégration. Vous pouvez utiliser des valeurs statiques ou des paramètres de mappage évalués de manière dynamique lors de l'exécution. Pour obtenir la liste complète des intégrations et des paramètres pris en charge, veuillez consulter the section called “AWS référence sur les intégrations de services”.

Mappage de paramètres

Type	Exemple	Remarques
Valeur d'en-tête	$request.header.*nom*	Les noms d'en-tête ne sont pas sensibles à la casse. API Gateway combine plusieurs valeurs d'en-

Type	Exemple	Remarques
		tête avec des virgules, par exemple `"header1": "value1,value2"` .
Valeur de chaîne de requête	$request.querystring.*nom*	Les noms de chaîne de requête sont sensibles à la casse. API Gateway combine plusieurs valeurs avec des virgules, par exemple `"querystring1": "Value1,Value2"` .
Paramètre de chemin	$request.path.*nom*	Valeur d'un paramètre de chemin dans la demande. Par exemple, si la route est `/pets/{petId}` , vous pouvez mapper le paramètre `petId` de la demande avec *`$request.path.petid`* .
Transmission de corps de demande	$request.body	API Gateway transmet l'ensemble du corps de la demande.

Type	Exemple	Remarques
Corps de la demande	$ request.body.*nom*	Expression de chemin JSON. La descente récursive (`$request.body..` *`name`*) et les expressions de filtre (`?(`*`expression`*` )`) ne sont pas prises en charge. **Note** Lorsque vous spécifiez un chemin JSON, API Gateway tronque le corps de la requête à 100 Ko, puis applique l'expression de sélection. Pour envoyer des charges utiles supérieures à 100 Ko, spécifiez `$request.body` .
Variable de contexte	$context.*`Nom_variable`*	Valeur d'une variable de contexte prise en charge.
Variable d'étape	$stageVariables.*`Nom_variable`*	Valeur d'une variable d'étape.
Valeur statique	*`string`*	Valeur constante.

Créer une intégration de première classe

Avant de créer une intégration de premier ordre, vous devez créer un rôle IAM qui accorde à API Gateway les autorisations nécessaires pour appeler l'action de AWS service à laquelle vous effectuez l'intégration. Pour en savoir plus, veuillez consulter Création d'un rôle pour un service AWS.

Pour créer une intégration de premier ordre, choisissez une action de AWS service prise en charge, par exemple`SQS-SendMessage`, configurez les paramètres de demande et fournissez un rôle qui accorde à API Gateway l'autorisation d'appeler l'API du AWS service intégré. Selon le sous-type d'intégration, différents paramètres de demande sont requis. Pour en savoir plus, consultez la section the section called “AWS référence sur les intégrations de services”.

La AWS CLI commande suivante crée une intégration qui envoie un message Amazon SQS.

```
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"}'
```

Créez une intégration de premier ordre à l'aide de AWS CloudFormation

L'exemple suivant montre un AWS CloudFormation extrait qui crée un `/{source}/{detailType}` itinéraire avec une intégration de premier ordre avec Amazon. EventBridge

Le paramètre `Source` est mappé au paramètre de chemin `{source}`, le paramètre `DetailType` au paramètre de chemin `{DetailType}` et le paramètre `Detail` au corps de la demande.

L’extrait n’affiche pas le bus d’événements ni le rôle IAM qui accorde à API Gateway les autorisations nécessaires pour invoquer l’action `PutEvents`.

```
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"
```

Référence de sous-type d'intégration

Les sous-types d'intégration suivants sont pris en charge pour les API HTTP.

Sous-types d'intégration

- EventBridge-PutEvents
- SQS- SendMessage
- SQS- ReceiveMessage
- SQS- DeleteMessage
- SQS- PurgeQueue
- AppConfig-GetConfiguration
- Kinésis- PutRecord
- StepFunctions-StartExecution
- StepFunctions-StartSyncExecution
- StepFunctions-StopExecution

EventBridge-PutEvents

Envoie des événements personnalisés à Amazon EventBridge afin qu'ils puissent être mis en correspondance avec les règles.

EventBridge- PutEvents 1,0

Paramètre	Obligatoire
Detail	Vrai

Paramètre	Obligatoire
DetailType	True
Source	Vrai
Heure	Faux
EventBusName	False
Ressources	Faux
Région	False
TraceHeader	False

Pour en savoir plus, consultez PutEventsle Amazon EventBridge API Reference.

SQS- SendMessage

Remet un message dans la file d'attente spécifiée.

SQS-1,0 SendMessage

Paramètre	Obligatoire
QueueUrl	True
MessageBody	True
DelaySeconds	False
MessageAttributes	False
MessageDeduplicationId	False
MessageGroupId	False
MessageSystemAttributes	False
Région	False

Pour en savoir plus, consultez SendMessagele manuel Amazon Simple Queue Service API Reference.

SQS- ReceiveMessage

Extrait un ou plusieurs messages (jusqu'à 10) de la file d'attente spécifiée.

SQS-1,0 ReceiveMessage

Paramètre	Obligatoire
QueueUrl	True
AttributeNames	False
MaxNumberOfMessages	False
MessageAttributeNames	False
ReceiveRequestAttemptId	False
VisibilityTimeout	False
WaitTimeSeconds	False
Région	False

Pour en savoir plus, consultez ReceiveMessagele manuel Amazon Simple Queue Service API Reference.

SQS- DeleteMessage

Supprime le message spécifié de la file d'attente désignée.

SQS-1,0 DeleteMessage

Paramètre	Obligatoire
ReceiptHandle	True
QueueUrl	True

Paramètre	Obligatoire
Région	False

Pour en savoir plus, consultez DeleteMessagele manuel Amazon Simple Queue Service API Reference.

SQS- PurgeQueue

Supprime tous les messages de la file d'attente spécifiée.

SQS-1,0 PurgeQueue

Paramètre	Obligatoire
QueueUrl	True
Région	False

Pour en savoir plus, consultez PurgeQueuele manuel Amazon Simple Queue Service API Reference.

AppConfig-GetConfiguration

Reçoit des informations sur une configuration.

AppConfig- GetConfiguration 1,0

Paramètre	Obligatoire
Application	Vrai
Environnement	Vrai
Configuration	True
ClientId	True
ClientConfigurationVersion	False
Région	False

Pour en savoir plus, consultez GetConfigurationle Guide de référence des AWS AppConfig API.

Kinésis- PutRecord

Écrit un enregistrement de données unique dans Amazon Kinesis Data Stream.

Kinesis- 1.0 PutRecord

Paramètre	Obligatoire
StreamName	True
non structurées	True
PartitionKey	True
SequenceNumberForOrdering	False
ExplicitHashKey	False
Région	False

Pour en savoir plus, consultez PutRecordle manuel Amazon Kinesis Data Streams API Reference.

StepFunctions-StartExecution

Démarre l'exécution d'une machine d'état.

StepFunctions- StartExecution 1,0

Paramètre	Obligatoire
StateMachineArn	True
Nom	Faux
Entrée	Faux
Région	False

Pour en savoir plus, consultez StartExecutionle Guide de référence des AWS Step Functions API.

StepFunctions-StartSyncExecution

Démarre une exécution de machine d'état synchrone.

StepFunctions- StartSyncExecution 1,0

Paramètre	Obligatoire
StateMachineArn	True
Nom	Faux
Entrée	Faux
Région	False
TraceHeader	False

Pour en savoir plus, consultez StartSyncExecutionle Guide de référence des AWS Step Functions API.

StepFunctions-StopExecution

Arrête une exécution.

StepFunctions- StopExecution 1,0

Paramètre	Obligatoire
ExecutionArn	True
Cause	Faux
Erreur	Faux
Région	False

Pour en savoir plus, consultez StopExecutionle Guide de référence des AWS Step Functions API.

Utilisation des intégrations privées pour les API HTTP

Les intégrations privées vous permettent de créer des intégrations d'API avec des ressources privées dans un VPC, telles que des équilibreurs Application Load Balancer ou des applications basées sur un conteneur Amazon ECS.

Vous pouvez exposer vos ressources dans un VPC pour que des clients puissent y accéder en dehors du VPC à l'aide d'intégrations privées. Vous pouvez contrôler l'accès à votre API à l'aide de n'importe quelle méthode d'autorisation prise en charge par API Gateway.

Pour créer une intégration privée, vous devez d'abord créer un lien VPC. Pour de plus amples informations sur les liens VPC, veuillez consulter Utilisation des liens VPC pour les API HTTP.

Après avoir créé un lien VPC, vous pouvez configurer des intégrations privées qui se connectent à un Application Load Balancer, à un Network Load Balancer ou à des ressources enregistrées auprès d'un service. AWS Cloud Map

Pour créer une intégration privée, toutes les ressources doivent appartenir au même AWS compte (y compris l'équilibreur de charge ou le AWS Cloud Map service, le lien VPC et l'API HTTP).

Par défaut, le trafic d'intégration privée utilise le protocole HTTP. Vous pouvez spécifier un `tlsConfig` si vous avez besoin d'un trafic d'intégration privée pour utiliser HTTPS.

> **Note**
>
> Pour les intégrations privées, API Gateway inclut la partie de l'étape du point de terminaison d'API dans la demande adressée à vos ressources backend. Par exemple, une demande à l'étape `test` d'une API inclut `test/route-path` dans la demande à votre intégration privée. Pour supprimer le nom de l'étape de la demande pour vos ressources backend, utilisez le mappage de paramètres pour écraser le chemin d'accès à la requête vers `$request.path`.

Créer une intégration privée à l'aide de Application Load Balancer ou de Network Load Balancer

Avant de créer une intégration privée, vous devez créer un lien VPC. Pour de plus amples informations sur les liens VPC, veuillez consulter Utilisation des liens VPC pour les API HTTP.

Pour créer une intégration privée avec un équilibreur Application Load Balancer ou un équilibreur Network Load Balancer, créez une intégration de proxy HTTP, spécifiez le lien VPC à utiliser et fournissez l'ARN de l'écouteur de l'équilibreur de charge.

Utilisez la commande suivante pour créer une intégration privée qui se connecte à un équilibreur de charge à l'aide d'un lien VPC.

```
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
```

Création d'une intégration privée à l'aide de la découverte AWS Cloud Map de services

Avant de créer une intégration privée, vous devez créer un lien VPC. Pour de plus amples informations sur les liens VPC, veuillez consulter Utilisation des liens VPC pour les API HTTP.

Pour les intégrations avec AWS Cloud Map, API Gateway les utilise `DiscoverInstances` pour identifier les ressources. Vous pouvez utiliser des paramètres de requête pour cibler des ressources spécifiques. Les attributs des ressources enregistrées doivent inclure les adresses IP et les ports. API Gateway distribue les demandes entre les ressources saines qui sont renvoyées à partir de `DiscoverInstances`. Pour en savoir plus, consultez DiscoverInstancesle Guide de référence des AWS Cloud Map API.

> **Note**
>
> Si vous utilisez Amazon ECS pour renseigner les entrées AWS Cloud Map, vous devez configurer votre tâche Amazon ECS pour utiliser les enregistrements SRV avec Amazon ECS Service Discovery ou activer Amazon ECS Service Connect. Pour en savoir plus, consultez Interconnexion des services dans le Guide du développeur Amazon Elastic Container Service.

Pour créer une intégration privée avec AWS Cloud Map, créez une intégration de proxy HTTP, spécifiez le lien VPC à utiliser et fournissez l'ARN du AWS Cloud Map service.

Utilisez la commande suivante pour créer une intégration privée qui utilise la découverte AWS Cloud Map de services pour identifier les ressources.

```
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
```

Utilisation des liens VPC pour les API HTTP

Les liens VPC vous permettent de créer des intégrations privées qui connectent vos itinéraires d'API HTTP à des ressources privées dans un VPC, comme des équilibreurs Application Load Balancer ou des applications basées sur les conteneurs Amazon ECS. Pour de plus amples informations sur la création d'intégrations privées, veuillez consulter Utilisation des intégrations privées pour les API HTTP.

Une intégration privée utilise un lien VPC pour encapsuler les connexions entre API Gateway et les ressources VPC ciblées. Vous pouvez réutiliser les liens VPC avec différentes routes et API.

Lorsque vous créez un lien VPC, API Gateway crée et gère des interfaces réseau Elastic pour le lien VPC de votre compte. Ce processus peut prendre quelques minutes. Lorsqu'un lien VPC est prêt à être utilisé, son état passe de `PENDING` à `AVAILABLE`.

> **Note**
>
> Si aucun trafic n'est envoyé sur le lien VPC pendant 60 jours, il devient `INACTIVE`. Lorsqu'un lien VPC atteint l'état `INACTIVE`, API Gateway supprime toutes les interfaces réseau du lien VPC. Cela provoque l'échec des demandes d'API qui dépendent du lien VPC. Si les demandes d'API reprennent, API Gateway provisionne à nouveau les interfaces réseau. La création des interfaces réseau et la réactivation du lien VPC peut prendre quelques minutes. Vous pouvez utiliser l'état du lien VPC pour surveiller l'état de votre lien VPC.

Créez un lien VPC à l'aide du AWS CLI

Utilisez la commande suivante pour créer un lien VPC. Pour créer un lien VPC, toutes les ressources impliquées doivent appartenir au même AWS compte.

```
aws apigatewayv2 create-vpc-link --name MyVpcLink \
    --subnet-ids subnet-aaaa subnet-bbbb \
    --security-group-ids sg1234 sg5678
```

> **Note**
>
> Les liens VPC sont immuables. Après avoir créé un lien VPC, vous ne pouvez pas modifier ses sous-réseaux ou groupes de sécurité.

Supprimez un lien VPC à l'aide du AWS CLI

Utilisez la commande suivante pour supprimer un lien VPC.

```
aws apigatewayv2 delete-vpc-link --vpc-link-id abcd123
```

Disponibilité par région

Les liens VPC pour les API HTTP sont pris en charge dans les régions et zones de disponibilité suivantes :

Nom de la région	Région	Zones de disponibilité prises en charge
USA Est (Ohio)	us-east-2	use2-az1, use2-az2, use2-az3
US East (Virginie du Nord)	us-east-1	use1-az1, use1-az2, use1-az4, use1-az5, use1-az6
USA Ouest (Californie du Nord)	us-west-1	usw1-az1, usw1-az3
USA Ouest (Oregon)	us-west-2	usw2-az1, usw2-az2, usw2-az3, usw2-az4
Asie-Pacifique (Hong Kong)	ap-east-1	ape1-az2, ape1-az3

Nom de la région	Région	Zones de disponibilité prises en charge
Asie-Pacifique (Mumbai)	ap-south-1	aps1-az1, aps1-az2, aps1-az3
Asie-Pacifique (Séoul)	ap-northeast-2	apne2-az1, apne2-az2, apne2-az3
Asie-Pacifique (Singapour)	ap-southeast-1	apse1-az1, apse1-az2, apse1-az3
Asie-Pacifique (Sydney)	ap-southeast-2	apse2-az1, apse2-az2, apse2-az3
Asie-Pacifique (Tokyo)	ap-northeast-1	apne1-az1, apne1-az2, apne1-az4
Canada (Centre)	ca-central-1	cac1-az1, cac1-az2
Europe (Francfort)	eu-central-1	euc1-az1, euc1-az2, euc1-az3
Europe (Irlande)	eu-west-1	euw1-az1, euw1-az2, euw1-az3
Europe (Londres)	eu-west-2	euw2-az1, euw2-az2, euw2-az3
Europe (Paris)	eu-west-3	euw3-az1, euw3-az3
Europe (Stockholm)	eu-north-1	eun1-az1, eun1-az2, eun1-az3

Nom de la région	Région	Zones de disponibilité prises en charge
Moyen-Orient (Bahreïn)	me-south-1	mes1-az1, mes1-az2, mes1-az3
Amérique du Sud (São Paulo)	sa-east-1	sae1-az1, sae1-az2, sae1-az3
AWS GovCloud (US-Ouest)	us-gov-west-1	usgw1-az1, usgw1-az2, usgw1-az3

Configuration de CORS pour une API HTTP

Le partage des ressources cross-origin (CORS) est une fonctionnalité de sécurité des navigateurs qui restreint les demandes HTTP lancées à partir de scripts s'exécutant dans le navigateur. Si vous ne parvenez pas à accéder à votre API et que vous recevez un message d'erreur contenant `Cross-Origin Request Blocked`, vous devrez peut-être activer CORS. Pour plus d'informations, voir Qu'est-ce que le CORS ? .

CORS est généralement requis pour créer des applications Web qui accèdent aux API hébergées sur un domaine ou une origine différent. Vous pouvez activer CORS pour autoriser les demandes à votre API à partir d'une application Web hébergée sur un autre domaine. Par exemple, si votre API est hébergée sur `https://{api_id}.execute-api.{region}.amazonaws.com/` et que vous souhaitez appeler votre API à partir d'une application Web hébergée sur `example.com`, votre API doit prendre en charge CORS.

Si vous configurez CORS pour une API, API Gateway envoie automatiquement une réponse aux demandes OPTIONS en amont, même s'il n'y a pas d'itinéraire OPTIONS configuré pour votre API. Pour une demande CORS, API Gateway ajoute les en-têtes CORS configurés à la réponse d'une intégration.

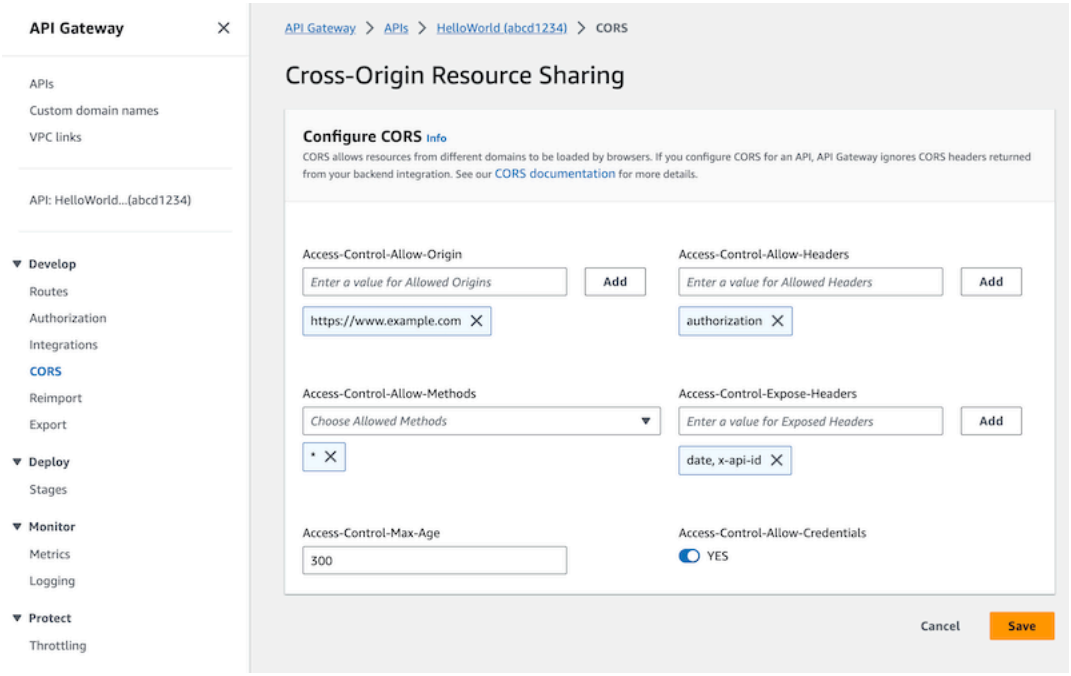
> **Note**
>
> Si vous configurez CORS pour une API, API Gateway ignore les en-têtes CORS renvoyés par votre intégration backend.

Vous pouvez spécifier les paramètres suivants dans une configuration CORS. Pour ajouter ces paramètres à l'aide de la console API HTTP API Gateway, choisissez Ajouter après avoir entré votre valeur.

En-têtes CORS	Propriété de configuration CORS	Exemples de valeur
Access-Control-Allow-Origin	allowOrigins	• `https://www.example.com` • `*` (autoriser toutes les origines) • `https://*` (autoriser toute origine commençant par https://) • `http://*` (autoriser toute origine commençant par http://)
Access-Control-Allow-Credentials	allowCredentials	true
Access-Control-Expose-Headers	exposeHeaders	Date x-api-id, *
Access-Control-Max-Age	maxAge	300
Access-Control-Allow-Methods	allowMethods	GET, POST, DELETE , *
Access-Control-Allow-Headers	allowHeaders	Authorization, *

Pour renvoyer des en-têtes CORS, votre demande doit contenir un en-tête `origin`.

Votre configuration CORS peut s'apparenter à ce qui suit :



Configuration de CORS pour une API HTTP avec une **`$default`** route et un autorisateur

Vous pouvez activer CORS et configurer l'autorisation pour n'importe quel itinéraire d'une API HTTP. Lorsque vous activez CORS et l'autorisation pour l'itinéraire `$default`, certaines considérations particulières sont à prendre en compte. L'itinéraire `$default` attrape les demandes pour toutes les méthodes et itinéraires que vous n'avez pas explicitement définies, y compris les demandes `OPTIONS`. Pour prendre en charge les demandes `OPTIONS` non autorisées, ajoutez un itinéraire `OPTIONS /{proxy+}` à votre API qui ne nécessite pas d'autorisation et attachez une intégration à l'itinéraire. La priorité de l'itinéraire `OPTIONS /{proxy+}` est supérieure à celle de l'itinéraire `$default`. Par conséquent, il permet aux clients de soumettre des demandes `OPTIONS` à votre API sans autorisation. Pour plus d'informations sur les priorités de routage, consultez Routage des demandes d'API.

Configuration de CORS pour une API HTTP à l'aide de la CLI AWS

Vous pouvez utiliser la commande update-api suivante pour activer les requêtes CORS depuis. `https://www.example.com`

Example

```
aws apigatewayv2 update-api --api-id api-id --cors-configuration AllowOrigins="https://
www.example.com"
```

Pour de plus amples informations, consultez CORS dans le document Référence des API Amazon API Gateway Version 2.

Transformer les demandes et les réponses d'API

Vous pouvez modifier les demandes d'API des clients avant qu'elles n'atteignent vos intégrations backend. Vous pouvez également modifier la réponse des intégrations avant qu'API Gateway ne la renvoie aux clients. Le mappage de paramètres vous permet de modifier les demandes et les réponses d'API pour les API HTTP. Pour utiliser le mappage de paramètres, spécifiez les paramètres de demande ou de réponse d'API à modifier, et indiquez comment modifier ces paramètres.

Transformation des demandes d'API

Les paramètres de demande vous permettent de modifier les demandes avant qu'elles n'atteignent vos intégrations backend. Vous pouvez modifier les en-têtes, les chaînes de la demande ou le chemin de la demande.

Les paramètres de demande sont représentés par un mappage clé-valeur. La clé identifie l'emplacement du paramètre de demande à modifier, ainsi que la façon de le modifier. La valeur spécifie les nouvelles données pour le paramètre.

Le tableau suivant présente les clés prises en charge.

Clés de mappage de paramètres

Type	Syntaxe
En-tête	`append|overwrite|remove:header.` *`headername`*
Chaîne de requête	`append|overwrite|remove:querystring.` *`querystring-name`*
Chemin	`overwrite:path`

Le tableau suivant présente les valeurs prises en charge que vous pouvez mapper aux paramètres.

Valeurs de mappage des paramètres de demande

Type	Syntaxe	Remarques
Valeur d'en-tête	$request.header.*name* ou ${request.header.*name*}	Les noms d'en-tête ne sont pas sensibles à la casse. API Gateway combine plusieurs valeurs d'en-tête avec des virgules, par exemple `"header1" : "value1,value2"` . Certains en-têtes sont réservés. Pour en savoir plus, consultez la section the section called “En-têtes réservés”.
Valeur de chaîne de requête	$request.querystring.*name* ou ${request.querystring.*name*}	Les noms de chaîne de requête sont sensibles à la casse. API Gateway combine plusieurs valeurs avec des virgules, par exemple `"querystring1" "Value1,Value2"` .
Corps de la demande	$request.body.*name* ou ${request.body.*name*}	Expression de chemin JSON. La descente récursive (`$request.body..name`) et les expressions de filtre (`?(expression)`) ne sont pas prises en charge. **Note** Lorsque vous spécifiez un chemin JSON, API Gateway tronque le

Type	Syntaxe	Remarques
		corps de la requête à 100 Ko, puis applique l'expression de sélection. Pour envoyer des charges utiles supérieures à 100 Ko, spécifiez `$request.body` .
Chemin de la demande.	$request.path ou ${request .path}	Chemin de la demande, sans le nom de l'étape.
Paramètre de chemin	$request.path.*name* ou ${request.path.*name*}	Valeur d'un paramètre de chemin dans la demande. Par exemple, si la route est `/pets/{petId}` , vous pouvez mapper le paramètre `petId` de la demande avec `$request.path.petId` .
Variable de contexte	$context.*variableName* ou ${context.*variableName* }	Valeur d'une variable de contexte. **Note** Seuls les caractères spéciaux `.` et `_` sont pris en charge.
Variable d'étape	$stageVariables.*variableName* ou ${stageVariables.*variableName* }	Valeur d'une variable d'étape.
Valeur statique	*string*	Valeur constante.

> **Note**
>
> Pour utiliser plusieurs variables dans une expression de sélection, placez la variable entre crochets. Par exemple, `${request.path.name} ${request.path.id}`.

Transformation des réponses d'API

Les paramètres de réponse vous permettent de transformer la réponse HTTP à partir d'une intégration backend avant de retourner la réponse aux clients. Vous pouvez modifier les en-têtes ou le code d'état d'une réponse avant qu'API Gateway ne renvoie la réponse aux clients.

Vous configurez les paramètres de réponse pour chaque code d'état renvoyé par votre intégration. Les paramètres de réponse sont représentés par un mappage clé-valeur. La clé identifie l'emplacement du paramètre de demande à modifier, ainsi que la façon de le modifier. La valeur spécifie les nouvelles données pour le paramètre.

Le tableau suivant présente les clés prises en charge.

Clés de mappage des paramètres de réponse

Type	Syntaxe
En-tête	`append|overwrite|remove:header.` *`headername`*
Code d'état	`overwrite:statuscode`

Le tableau suivant présente les valeurs prises en charge que vous pouvez mapper aux paramètres.

Valeurs de mappage des paramètres de réponse

Type	Syntaxe	Remarques
Valeur d'en-tête	$response.header.*nom* ou ${response.header.*nom*}	Les noms d'en-tête ne sont pas sensibles à la casse. API Gateway combine plusieurs valeurs d'en-tête avec des virgules,

Type	Syntaxe	Remarques
		par exemple `"header1" : "value1,value2"` . Certains en-têtes sont réservés. Pour en savoir plus, consultez la section the section called “En-têtes réservés”.
Corps de la réponse	$response.body.*name* ou ${response.body.*name*}	Expression de chemin JSON. La descente récursive (`$response.body..name`) et les expressions de filtre (`?(expression)`) ne sont pas prises en charge. **Note** Lorsque vous spécifiez un chemin JSON, API Gateway tronque le corps de la réponse à 100 Ko, puis applique l'expression de sélection. Pour envoyer des charges utiles supérieures à 100 Ko, spécifiez `$response.body` .
Variable de contexte	$context.*variableName* ou ${context.*variableName* }	Valeur d'une variable de contexte prise en charge.
Variable d'étape	$stageVariables.*variableName* ou ${stageVariables.*variableName* }	Valeur d'une variable d'étape.

Type	Syntaxe	Remarques
Valeur statique	*string*	Valeur constante.

> **Note**
>
> Pour utiliser plusieurs variables dans une expression de sélection, placez la variable entre crochets. Par exemple, `${request.path.name} ${request.path.id}`.

En-têtes réservés

Les en-têtes suivants sont réservés. Vous ne pouvez pas configurer les mappages de demande ou de réponse pour ces en-têtes.

- access-control-*
- apigw-*
- Autorisation
- Connection
- Encodage-Contenu
- 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

Exemples

Les AWS CLI exemples suivants configurent les mappages de paramètres. Pour des exemples AWS CloudFormation de modèles, voir GitHub.

Ajouter un en-tête à une demande d'API

L'exemple suivant ajoute un en-tête nommé `header1` à une demande d'API avant qu'elle n'atteigne votre intégration backend. API Gateway remplit l'en-tête avec l'ID de demande.

```
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" }'
```

Renommer un en-tête de demande

L'exemple suivant renomme un en-tête de demande `header1` en `header2`.

```
aws apigatewayv2 create-integration \
    --api-id abcdef123 \
    --integration-type HTTP_PROXY \
    --payload-format-version 1.0 \
    --integration-uri 'https://api.example.com' \
    --integration-method ANY \
    --request-parameters '{ "append:header.header2": "$request.header.header1",
 "remove:header.header1": "''"}'
```

Modifier la réponse d'une intégration

L'exemple suivant configure les paramètres de réponse pour une intégration. Lorsque les intégrations renvoient un code d'état 500, API Gateway modifie le code d'état en 403 et ajoute `header11` à la

réponse. Lorsque l'intégration renvoie un code d'état 404, API Gateway ajoute un en-tête `error` à la réponse.

```
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"}  }'
```

Supprimer les mappages de paramètres configurés

L'exemple de commande suivant supprime les paramètres de demande précédemment configurés pour `append:header.header1`. Il supprime également les paramètres de réponse précédemment configurés pour un code d'état 200.

```
aws apigatewayv2 update-integration \
    --api-id abcdef123 \
    --integration-id hijk456 \
    --request-parameters '{"append:header.header1" : ""}' \
    --response-parameters '{"200" : {}}'
```

Utilisation des définitions OpenAPI pour les API HTTP

Vous pouvez définir votre API HTTP à l'aide d'un fichier de définition OpenAPI 3.0. Ensuite, vous pouvez importer la définition dans API Gateway pour créer une API. Pour en savoir plus sur les extensions API Gateway vers OpenAPI, veuillez consulter Extensions OpenAPI.

Importation d'une API HTTP

Vous pouvez créer une API HTTP en important un fichier de définition OpenAPI 3.0.

Pour migrer d'une API REST vers une API HTTP, vous pouvez exporter votre API REST en tant que fichier de définition OpenAPI 3.0. Ensuite, importez la définition de l'API en tant qu'API HTTP. Pour en savoir plus sur l'exportation d'une API REST, veuillez consulter Exportation d'une API REST à partir d'API Gateway.

> **Note**
>
> Les API HTTP prennent en charge les mêmes AWS variables que les API REST. Pour en savoir plus, consultez la section AWS variables pour l'importation d'OpenAPI.

Importer des informations de validation

Lorsque vous importez une API, API Gateway fournit trois catégories d'informations de validation.

Infos

Une propriété est valide selon la spécification OpenAPI, mais cette propriété n'est pas prise en charge pour les API HTTP.

Par exemple, l'extrait OpenAPI 3.0 suivant produit des informations sur l'importation, car les API HTTP ne prennent pas en charge la validation de la demande. API Gateway ignore les champs `requestBody` et `schema`.

```
"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"
        }
      }
    }
    ...
  }
  ...
}
```

Avertissement

Une propriété ou une structure n'est pas valide selon la spécification OpenAPI, mais elle ne bloque pas la création d'API. Vous pouvez spécifier si API Gateway doit ignorer ces avertissements et continuer à créer l'API, ou arrêter de créer l'API sur les avertissements.

Le document OpenAPI 3.0 suivant produit des avertissements lors de l'importation, car les API HTTP ne prennent en charge que les intégrations proxy Lambda et proxy HTTP.

```
"x-amazon-apigateway-integration": {
  "type": "AWS",
  "httpMethod": "POST",
  "uri": "arn:aws:lambda:us-east-2:123456789012:function:HelloWorld",
  "payloadFormatVersion": "1.0"
}
```

Erreur

La spécification OpenAPI n'est pas valide ou mal formée. API Gateway ne peut pas créer de ressources à partir du document mal formé. Vous devez corriger les erreurs, puis réessayer.

La définition d'API suivante génère des erreurs lors de l'importation, car les API HTTP ne prennent en charge que la spécification OpenAPI 3.0.

```
{
  "swagger": "2.0.0",
  "info": {
    "title": "My API",
    "description": "An Example OpenAPI definition for Errors/Warnings/ImportInfo",
```

```
    "version": "1.0"
  }
  ...
}
```

Autre exemple, alors qu'OpenAPI permet aux utilisateurs de définir une API avec plusieurs exigences en matière de sécurité liées à une opération particulière, API Gateway ne prend pas en charge cette fonctionnalité. Chaque opération ne peut avoir qu'un seul des éléments suivants : autorisation IAM, mécanisme d'autorisation Lambda ou mécanisme d'autorisation JWT. Tenter de modéliser plusieurs exigences de sécurité entraîne une erreur.

Importez une API à l'aide du AWS CLI

La commande suivante importe le fichier de définition OpenAPI 3.0 `api-definition.json` en tant qu'API HTTP.

Example

```
aws apigatewayv2 import-api --body file://api-definition.json
```

Example

Vous pouvez importer l'exemple suivant de définition OpenAPI 3.0 pour créer une API HTTP.

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

Exportation d'une API HTTP depuis API Gateway

Après avoir créé une API HTTP, vous pouvez exporter une définition OpenAPI 3.0 de votre API à partir d'API Gateway. Vous pouvez choisir une étape à exporter ou exporter la dernière configuration de votre API. Vous pouvez également importer une définition d'API exportée dans API Gateway pour créer une autre API identique. Pour de plus amples informations sur l'importation de définitions d'API, veuillez consulter Importation d'une API HTTP.

Exporter une définition OpenAPI 3.0 d'une étape à l'aide de la CLI AWS

La commande suivante exporte une définition OpenAPI d'une étape d'API nommée `prod` dans un fichier YAML intitulé `stage-definition.yaml`. Le fichier de définition exporté inclut les extensions API Gateway par défaut.

```
aws apigatewayv2 export-api \
    --api-id api-id  \
    --output-type YAML  \
    --specification OAS30 \
    --stage-name prod \
    stage-definition.yaml
```

Exportez une définition OpenAPI 3.0 des dernières modifications de votre API à l'aide de la CLI AWS

La commande suivante exporte une définition OpenAPI d'une API HTTP vers un fichier JSON intitulé `latest-api-definition.json`. Étant donné que la commande ne spécifie pas d'étape, API Gateway exporte la dernière configuration de votre API, qu'elle ait été déployée ou non sur une étape. Le fichier de définition exporté n'inclut pas les extensions API Gateway.

```
aws apigatewayv2 export-api \
    --api-id api-id  \
    --output-type JSON  \
    --specification OAS30 \
    --no-include-extensions \
    latest-api-definition.json
```

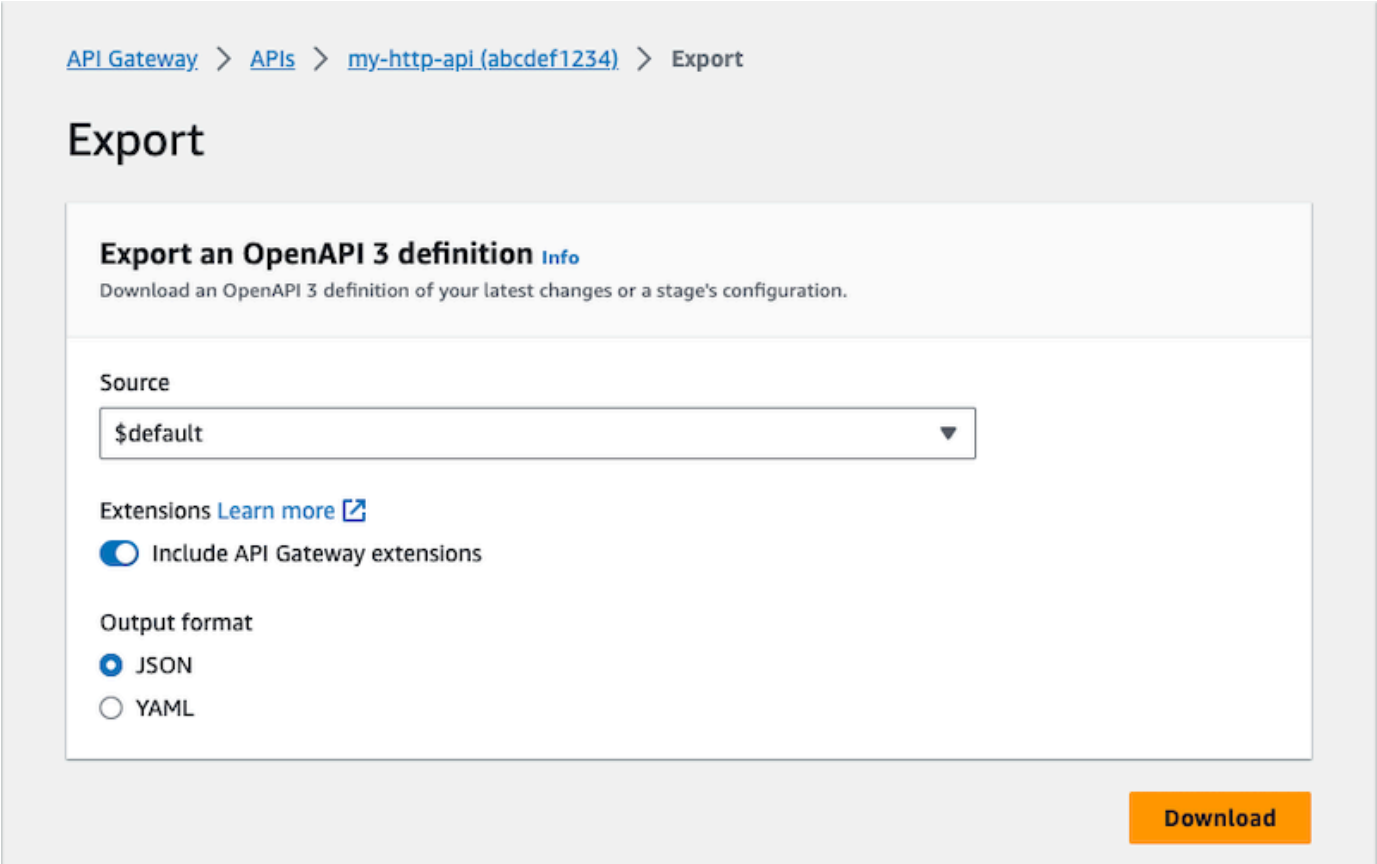
Pour de plus amples informations, veuillez consulter ExportAPI dans le document Référence des API Amazon API Gateway Version 2.

Exportation d'une définition OpenAPI 3.0 à l'aide de la console API Gateway

La procédure suivante illustre comment exporter une définition OpenAPI d'une API HTTP.

Pour exporter une définition OpenAPI 3.0 à l'aide de la console API Gateway

1. Connectez-vous à la console API Gateway à l'adresse : https://console.aws.amazon.com/apigateway.
2. Choisissez une API HTTP.
3. Dans le panneau de navigation principal, sous Développer, choisissez Exporter.
4. Sélectionnez l'une des options suivantes pour exporter votre API :



a. Pour Source, sélectionnez une source pour la définition OpenAPI 3.0. Vous pouvez choisir une étape à exporter ou exporter la dernière configuration de votre API.

b. Activez l'option Inclure les extensions API Gateway pour inclure les extensions API Gateway.

c. Pour Format de sortie, sélectionnez un format de sortie.

5. Choisissez Téléchargement.

Publication d'API HTTP à appeler par les clients

Vous pouvez utiliser des étapes et des noms de domaine personnalisés pour publier l'API que les clients peuvent appeler.

Une étape d'API est une référence logique à un état du cycle de vie de votre API (par exemple, `dev`, `prod`, `beta` ou `v2`). Chaque étape est une référence nommée à un déploiement de l'API et elle est mise à la disposition des applications clientes à appeler. Vous pouvez configurer différentes intégrations et divers paramètres pour chaque étape d'une API.

Vous pouvez utiliser des noms de domaine personnalisés pour fournir aux clients une URL plus simple et plus intuitive pour appeler votre API que l'URL par défaut, `https://api-id.execute-api.region.amazonaws.com/stage`.

> **Note**
>
> Pour renforcer la sécurité de vos API API Gateway, le domaine `execute-api.{region}.amazonaws.com` est enregistré dans la liste des suffixes publics (PSL). Pour plus de sécurité, nous vous recommandons d'utiliser des cookies avec un préfixe `__Host-` si vous devez définir des cookies sensibles dans le nom de domaine par défaut de vos API API Gateway. Cette pratique vous aidera à protéger votre domaine contre les tentatives de falsification de requêtes intersites (CSRF). Pour plus d'informations, consultez la page Set-Cookie du Mozilla Developer Network.

Rubriques

Utilisation des étapes pour les API HTTP

Une étape d'API est une référence logique à un état du cycle de vie de votre API (par exemple, `dev`, `prod`, `beta` ou `v2`). Les étapes API sont identifiées par leur ID d'API et leur nom d'étape, et elles sont incluses dans l'URL que vous utilisez pour appeler l'API. Chaque étape est une référence nommée à un déploiement de l'API et elle est mise à la disposition des applications clientes à appeler.

Vous pouvez créer une étape `$default` qui est servie à partir de la base de l'URL de votre API, par exemple `https://{api_id}.execute-api.{region}.amazonaws.com/`. Vous utilisez cette URL pour appeler une étape d'API.

Un déploiement est un instantané de la configuration de votre API. Après que vous avez déployé une API sur une étape, les clients peuvent l'appeler. Vous devez déployer une API pour que les modifications prennent effet. Si vous activez les déploiements automatiques, les modifications apportées à une API sont automatiquement libérées pour vous.

Variables d'étape

Les variables d'étape sont des paires clé-valeur que vous pouvez définir pour une étape d'une API HTTP. Elles se comportent comme les variables d'environnement et peuvent être utilisées dans votre configuration d'API.

Par exemple, vous pouvez définir une variable d'étape, puis définir sa valeur en tant que point de terminaison HTTP pour une intégration de proxy HTTP. Par la suite, vous pouvez référencer le point de terminaison à l'aide du nom de la variable d'étape associée. Ce faisant, vous pouvez utiliser la même configuration d'API avec un point de terminaison différent à chaque étape. De même, vous pouvez utiliser des variables d'étape pour spécifier une intégration de AWS Lambda fonction différente pour chaque étape de votre API.

> **Note**
>
> Les variables d'étape ne sont pas destinées à être utilisées pour des données sensibles, telles que les informations d'identification. Pour transmettre des données sensibles aux intégrations, utilisez un AWS Lambda autorisateur. Vous pouvez transmettre des données sensibles aux intégrations dans la sortie du mécanisme d'autorisation Lambda. Pour en savoir plus, consultez la section the section called “Format de réponse du mécanisme d'autorisation Lambda”.

Exemples

Pour utiliser une variable d'étape afin de personnaliser le point de terminaison d'intégration HTTP, vous devez d'abord définir le nom et la valeur de la variable stage (par exemple, `url`) avec la valeur `example.com`. Ensuite, configurez une intégration de proxy HTTP. Au lieu d'entrer l'URL du point de terminaison, vous pouvez demander à API Gateway d'utiliser la valeur de la variable d'étape, **`http://${stageVariables.url}`**. Cette valeur demande à API Gateway de remplacer votre variable d'étape `${}` au moment de l'exécution, en fonction de l'étape à laquelle se trouve votre API.

Vous pouvez référencer les variables d'étape de la même manière pour spécifier un nom de fonction Lambda ou un AWS ARN de rôle.

Lorsque vous spécifiez un nom de fonction Lambda en tant que valeur de variable d'étape, vous devez configurer les autorisations sur cette fonction Lambda manuellement. Pour ce faire, vous pouvez utiliser AWS Command Line Interface (AWS CLI).

```
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
```

Référence des variables d'étape API Gateway

URI d'intégration HTTP

Une variable d'étape peut être utilisée dans une URI d'intégration HTTP, comme illustré dans les exemples suivants.

- URI complet sans protocole – `http://${stageVariables.<variable_name>}`
- Domaine complet – `http://${stageVariables.<variable_name>}/resource/operation`
- Sous-domaine – `http://${stageVariables.<variable_name>}.example.com/resource/operation`
- Chemin – `http://example.com/${stageVariables.<variable_name>}/bar`
- Chaîne de requête – `http://example.com/foo?q=${stageVariables.<variable_name>}`

Fonctions Lambda

Vous pouvez utiliser une variable d'étape à la place d'un nom d'intégration ou d'un alias de fonction Lambda, comme illustré dans les exemples suivants.

- `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**
>
> Pour utiliser une variable d'étape pour une fonction Lambda, la fonction doit se trouver dans le même compte que l'API. Les variables d'étape ne prennent pas en charge les fonctions Lambda inter-comptes.

AWS informations d'identification d'intégration

Vous pouvez utiliser une variable d'étape dans le cadre de l'ARN d'identification AWS d'un utilisateur ou d'un rôle, comme illustré dans l'exemple suivant.

- `arn:aws:iam::<account_id>:${stageVariables.<variable_name>}`

Politique de sécurité pour les API HTTP

API Gateway applique une politique de sécurité `TLS_1_2` pour tous les points de terminaison de l'API HTTP.

Une politique de sécurité est une combinaison prédéfinie de version minimale de TLS et de suites de chiffrement proposées par Amazon API Gateway. Le protocole TLS résout les problèmes de sécurité de réseau tels que la falsification et le risque d'écoute illicite entre un client et un serveur. Lorsque vos clients établissent une liaison TLS vers votre API via le domaine personnalisé, la stratégie de sécurité applique les options de version TLS et de suite de chiffrement que vos clients peuvent choisir d'utiliser. Cette politique de sécurité accepte le trafic TLS 1.2 et TLS 1.3 et rejette le trafic TLS 1.0.

Protocoles et chiffrements TLS pris en charge pour les API HTTP

Le tableau suivant décrit les protocoles et chiffrements TLS pris en charge pour les API HTTP.

Politique de sécurité	TLS_1_2
Protocoles TLS	
TLSv1.3	♦
TLSv1.2	♦
Chiffrements TLS	

Politique de sécurité	TLS_1_2
TLS-AES-128-GCM-SHA256	♦
TLS-AES-256-GCM-SHA384	♦
TLS-CHACHA20-POLY1305-SHA256	♦
ECDHE-ECDSA-AES128-GCM-SHA256	♦
ECDHE-RSA-AES128-GCM-SHA256	♦
ECDHE-ECDSA-AES128-SHA256	♦
ECDHE-RSA-AES128-SHA256	♦
ECDHE-ECDSA-AES256-GCM-SHA384	♦
ECDHE-RSA-AES256-GCM-SHA384	♦
ECDHE-ECDSA-AES256-SHA384	♦
ECDHE-RSA-AES256-SHA384	♦
AES128-GCM-SHA256	♦
AES128-SHA256	♦
AES256-GCM-SHA384	♦
AES256-SHA256	♦

Noms de chiffrement OpenSSL et RFC

OpenSSL et IETF RFC 5246 utilisent des noms différents pour les mêmes chiffrements. Pour obtenir la liste des noms chiffrés, voirthe section called “Noms de chiffrement OpenSSL et RFC”.

Informations sur les API et WebSocket API REST

Pour plus d'informations sur les API et WebSocket API REST, consultez the section called “Choix d'une politique de sécurité” etthe section called “Politique de sécurité pour les WebSocket API”.

Configuration des noms de domaine personnalisés pour les API HTTP

Les noms de domaine personnalisés sont des URL plus simples et plus intuitives que vous pouvez fournir à vos utilisateurs d'API.

Après avoir déployé votre API, vous (et vos clients) pouvez appeler cette API à l'aide de l'URL de base par défaut au format suivant :

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

où `api-id` est généré par API Gateway, `region` (AWS Region) est spécifié par vous lors de la création de l'API, et `stage` est spécifié par vous lors du déploiement de l'API.

La partie nom d'hôte de l'URL (c'est-à-dire, *`api-id`*`.execute-api.`*`region`*`.amazonaws.com`) fait référence à un point de terminaison de l'API. Le point de terminaison d'API par défaut peut être difficile à retenir et non convivial.

Avec des noms de domaine personnalisés, vous pouvez configurer le nom d'hôte de votre API et choisir un chemin de base (par exemple, `myservice`) pour mapper l'URL alternative à votre API. Par exemple, une URL de base de l'API plus conviviale peut devenir :

```
https://api.example.com/myservice
```

> **Note**
>
> Un domaine personnalisé peut être associé aux API REST et aux API HTTP. Vous pouvez utiliser API Gateway Version 2 API pour créer et gérer des noms de domaine régionaux personnalisés pour les API REST et les API HTTP.
> Pour les API HTTP, TLS 1.2 est la seule version TLS prise en charge.

Enregistrement d'un nom de domaine

Pour pouvoir configurer des noms de domaine personnalisés pour vos API, vous devez avoir enregistré un nom de domaine Internet. Votre nom de domaine doit respecter la spécification RFC 1035 et peut comporter un maximum de 63 octets par étiquette et 255 octets au total. Si nécessaire, vous pouvez enregistrer un domaine Internet à l'aide de Amazon Route 53 ou utiliser le bureau d'enregistrement de domaine tiers de votre choix. Le nom de domaine personnalisé d'une API peut

être le nom d'un sous-domaine ou du domaine racine (également nommé « zone apex ») d'un domaine Internet enregistré.

Après avoir créé un nom de domaine personnalisé dans API Gateway, vous devez créer ou mettre à jour l'enregistrement de ressource de votre fournisseur DNS pour mapper à votre point de terminaison API. Sans ce mappage, les demandes d'API destinées au nom de domaine personnalisé ne peuvent pas atteindre API Gateway.

Noms de domaine personnalisés régionaux

Lorsque vous créez un nom de domaine personnalisé pour une API régionale, API Gateway crée un nom de domaine régional pour l'API. Vous devez configurer un enregistrement DNS pour mapper le nom de domaine personnalisé vers le nom de domaine régional. Vous devez également fournir un certificat pour le nom de domaine personnalisé.

Noms de domaine personnalisés génériques

Avec les noms de domaine personnalisés génériques, vous pouvez prendre en charge un nombre presque infini de noms de domaine sans dépasser le quota par défaut. Par exemple, vous pouvez donner à chacun de vos clients son propre nom de domaine, *`customername`*`.api.example.com`.

Pour créer un nom de domaine personnalisé générique, vous pouvez spécifier un caractère générique (*) comme premier sous-domaine d'un domaine personnalisé qui représente tous les sous-domaines possibles d'un domaine racine.

Par exemple, le nom de domaine personnalisé générique `*.example.com` se traduit par des sous-domaines tels que `a.example.com`, `b.example.com` et `c.example.com`, qui effectuent tous un routage vers le même domaine.

Les noms de domaine personnalisés génériques prennent en charge des configurations distinctes des noms de domaine personnalisés standard d'API Gateway. Par exemple, dans un seul AWS compte, vous pouvez configurer `*.example.com` et `a.example.com` vous comporter différemment.

Pour créer un nom de domaine personnalisé générique, vous devez fournir un certificat émis par ACM qui a été validé à l'aide du DNS ou de la méthode de validation par e-mail.

> **Note**
>
> Vous ne pouvez pas créer de nom de domaine personnalisé générique si un autre AWS compte a créé un nom de domaine personnalisé en conflit avec le nom de domaine

personnalisé générique. Par exemple, si le compte A a créé `a.example.com`, le compte B ne peut pas créer le nom de domaine personnalisé générique `*.example.com`.
Si les comptes A et B ont le même propriétaire, vous pouvez contacter le AWS Centre de support pour demander une exception.

Certificats pour les noms de domaine personnalisés

> **Important**
>
> Vous spécifiez le certificat de votre nom de domaine personnalisé. Si votre application utilise l'épinglage de certificat, parfois appelé épinglage SSL, pour épingler un certificat ACM, il est possible que l'application ne puisse pas se connecter à votre domaine après le AWS renouvellement du certificat. Pour plus d'informations, consultez la section Problèmes d'épinglage de certificat dans le Guide de l'utilisateur AWS Certificate Manager .

Pour fournir un certificat pour un nom de domaine personnalisé dans une région où ACM est pris en charge, vous devez demander un certificat à ACM. Pour fournir un certificat pour un nom de domaine personnalisé régional dans une région où ACM n'est pas pris en charge, vous devez importer un certificat dans API Gateway dans cette région.

Pour importer un certificat SSL/TLS, vous devez fournir le corps du certificat SSL/TLS au format PEM, sa clé privée, ainsi que la chaîne de certificats du nom de domaine personnalisé. Chaque certificat stocké dans ACM est identifié par son ARN. Pour utiliser un certificat AWS géré pour un nom de domaine, il suffit de référencer son ARN.

ACM permet de configurer et d'utiliser un nom de domaine personnalisé pour une API. Vous créez un certificat pour le nom de domaine donné (ou vous importez un certificat), configurez le nom de domaine dans API Gateway avec l'ARN du certificat fourni par ACM, et vous mappez un chemin de base sous le nom de domaine personnalisé à une étape déployée de l'API. Avec les certificats émis par ACM, vous n'avez pas à vous inquiéter d'une éventuelle exposition des informations sensibles du certificat, par exemple sa clé privée.

Pour de plus amples informations sur la configuration d'un nom de domaine personnalisé, veuillez consulter Préparation des certificats dans AWS Certificate Manager et Configuration d'un nom de domaine personnalisé régional dans API Gateway.

Utilisation des mappages d'API pour les API HTTP

Les mappages d'API vous permettent de connecter des étapes d'API à un nom de domaine personnalisé. Après avoir créé un nom de domaine et configuré les enregistrements DNS, vous pouvez utiliser les mappages d'API pour envoyer le trafic vers vos API via votre nom de domaine personnalisé.

Un mappage d'API spécifie une API, une étape et éventuellement un chemin à utiliser pour le mappage. Par exemple, vous pouvez mapper l'étape `production` d'une API à `https://api.example.com/orders`.

Vous pouvez mapper les étapes d'API HTTP et REST au même nom de domaine personnalisé.

Avant de créer un mappage d'API, vous devez disposer d'une API, d'une étape et d'un nom de domaine personnalisé. Pour plus d'informations sur la création d'un nom de domaine personnalisé, consultez the section called “Configuration d'un nom de domaine personnalisé régional”.

Routage des demandes d'API

Vous pouvez configurer des mappages d'API à plusieurs niveaux, par exemple `orders/v1/items` et `orders/v2/items`.

Pour les mappages d'API à plusieurs niveaux, API Gateway achemine les demandes vers le mappage d'API dont le chemin d'accès est le plus long. API Gateway prend uniquement en compte les chemins configurés pour les mappages d'API, et non les routes d'API, pour sélectionner l'API à appeler. Si aucun chemin ne correspond à la demande, API Gateway envoie celle-ci à l'API que vous avez mappée au chemin vide `(none)`.

Pour les noms de domaine personnalisés qui utilisent les mappages d'API à plusieurs niveaux, API Gateway achemine les demandes vers le mappage d'API doté du préfixe correspondant le plus long.

Par exemple, imaginons un nom de domaine personnalisé `https://api.example.com` doté des mappages d'API suivants :

1. `(none)` mappé à l'API 1.
2. `orders` mappé à l'API 2.
3. `orders/v1/items` mappé à l'API 3.
4. `orders/v2/items` mappé à l'API 4.
5. `orders/v2/items/categories` mappé à l'API 5.

Requête	API sélectionnée	Explication
`https://api.example.com/orders`	`API 2`	La demande correspond exactement à ce mappage d'API.
`https://api.example.com/orders/v1/items`	`API 3`	La demande correspond exactement à ce mappage d'API.
`https://api.example.com/orders/v2/items`	`API 4`	La demande correspond exactement à ce mappage d'API.
`https://api.example.com/orders/v1/items/123`	`API 3`	API Gateway choisit le mappage d'API dont le chemin d'accès est le plus long. La présence de `123` à la fin de la demande n'affecte pas la sélection.
`https://api.example.com/orders/v2/items/categories/5`	`API 5`	API Gateway choisit le mappage d'API dont le chemin d'accès est le plus long.
`https://api.example.com/customers`	`API 1`	API Gateway utilise le mappage vide comme fourre-tout.
`https://api.example.com/ordersandmore`	`API 2`	API Gateway choisit le mappage d'API doté du préfixe correspondant le plus long. Pour un nom de domaine personnalisé configuré avec des mappages à un seul niveau, tels que `https://api.example.com/orders` et `https://`

Requête	API sélectionnée	Explication
		`api.example.com/` uniquement, API Gateway choisirait `API 1`, car il n'y a pas de chemin correspondant avec `ordersandmore` .

Restrictions

- Dans un mappage d'API, le nom de domaine personnalisé et les API mappées doivent se trouver dans le même AWS compte.
- Les mappages d'API ne doivent contenir que des lettres, des chiffres et les caractères suivants : `$-_.+!*'()/`.
- La longueur maximale du chemin d'un mappage d'API est de 300 caractères.
- Vous pouvez disposer de 200 mappages d'API à plusieurs niveaux pour chaque nom de domaine.
- Vous ne pouvez mapper les API HTTP à un nom de domaine personnalisé régional qu'à l'aide de la stratégie de sécurité TLS 1.2.
- Vous ne pouvez pas associer WebSocket des API au même nom de domaine personnalisé qu'une API HTTP ou une API REST.

Créer un mappage d'API

Pour créer un mappage d'API, vous devez d'abord créer un nom de domaine personnalisé, une API et une étape. Pour plus d'informations sur la création d'un nom de domaine personnalisé, consultez the section called “Configuration d'un nom de domaine personnalisé régional”.

Pour des exemples AWS Serverless Application Model de modèles qui créent toutes les ressources, voir Sessions avec SAM activé GitHub.

AWS Management Console

Pour créer un mappage d'API

1. Connectez-vous à la console API Gateway à l'adresse : https://console.aws.amazon.com/apigateway.

2. Choisissez Noms de domaine personnalisés.
3. Sélectionnez un nom de domaine personnalisé que vous avez déjà créé.
4. Choisissez Mappages d'API.
5. Choisissez Configurer les mappages d'API.
6. Choisissez Ajouter un nouveau mappage.
7. Entrez une API, une Étape et, éventuellement, un Chemin d'accès.
8. Choisissez Enregistrer.

AWS CLI

La AWS CLI commande suivante crée un mappage d'API. Dans cet exemple, API Gateway envoie des demandes `api.example.com/v1/orders` à l'API et à l'étape spécifiés.

```
aws apigatewayv2 create-api-mapping \
    --domain-name api.example.com \
    --api-mapping-key v1/orders \
    --api-id a1b2c3d4 \
    --stage test
```

AWS CloudFormation

L' AWS CloudFormation exemple suivant crée un mappage d'API.

```
MyApiMapping:
  Type: 'AWS::ApiGatewayV2::ApiMapping'
  Properties:
    DomainName: api.example.com
    ApiMappingKey: 'orders/v2/items'
    ApiId: !Ref MyApi
    Stage: !Ref MyStage
```

Désactivation du point de terminaison par défaut pour une API HTTP

Par défaut, les clients peuvent appeler votre API en utilisant le point de terminaison `execute-api` généré par API Gateway pour votre API. Pour vous assurer que les clients peuvent accéder à votre API en utilisant uniquement un nom de domaine personnalisé, désactivez le point de terminaison par défaut `execute-api`.

> **Note**
>
> Lorsque vous désactivez le point de terminaison par défaut, toutes les étapes d'une API sont affectées.

La AWS CLI commande suivante désactive le point de terminaison par défaut pour une API HTTP.

```
aws apigatewayv2 update-api \
    --api-id abcdef123 \
    --disable-execute-api-endpoint
```

Après avoir désactivé le point de terminaison par défaut, vous devez déployer votre API pour que la modification prenne effet, sauf si les déploiements automatiques sont activés.

La AWS CLI commande suivante crée un déploiement.

```
aws apigatewayv2 create-deployment \
    --api-id abcdef123 \
    --stage-name dev
```

Protection de votre API HTTP

API Gateway fournit un certain nombre de façons de protéger votre API contre certaines menaces, comme les utilisateurs malveillants ou les pics de trafic. Vous pouvez protéger votre API à l'aide de stratégies telles que la définition de limitations et l'activation de l'authentification TLS mutuelle. Dans cette section, vous pouvez apprendre à activer ces fonctionnalités à l'aide d'API Gateway.

Rubriques

- Limitation des demandes envoyées à votre API HTTP
- Configuration de l'authentification TLS mutuelle pour une API HTTP

Limitation des demandes envoyées à votre API HTTP

Vous pouvez configurer la limitation pour vos API pour éviter qu'elles soient submergées de demandes. Les limitations sont appliquées dans la mesure du possible et doivent être considérées comme des cibles plutôt que des plafonds de demandes garantis.

API Gateway régule les demandes de limitations soumises à votre API à l'aide de l'algorithme de compartiment de jetons, où un jeton compte pour une demande. En particulier, API Gateway examine le taux de soumissions de demandes à un débit régulier et en mode rafale pour toutes les API de votre compte, par région. Dans l'algorithme de compartiment de jetons, une rafale peut permettre un dépassement prédéfini de ces limites, mais d'autres facteurs peuvent également entraîner le dépassement des limites dans certains cas.

Lorsque les soumissions de demandes dépassent les limites de taux régulier et en mode rafale des demandes, API Gateway commence à limiter les demandes. Les clients peuvent recevoir des réponses aux erreurs `429 Too Many Requests` à ce stade. Lors de la capture de ces exceptions, le client peut renvoyer les demandes en échec de façon à limiter le débit tout en respectant les limitations.

En tant que développeur d'API, vous pouvez définir les limites pour les étapes ou acheminements d'API individuels afin d'améliorer les performances globales de toutes les API de votre compte.

Limitation au niveau du compte par région

API Gateway limite par défaut les demandes régulières par seconde (rps) pour toutes les API d'un compte AWS , par Région. Il limite également la rafale (c'est-à-dire, la taille maximale de compartiment) sur toutes les API au sein d'un compte AWS , par région. Dans API Gateway, la limite en mode rafale correspond au nombre maximal d'envois de demandes simultanés qu'API Gateway peut traiter à tout moment sans renvoyer de réponses d'erreur `429 Too Many Requests`. Pour plus d'informations sur les quotas de limitation, consultez Quotas et remarques importantes.

Les limites par compte sont appliquées à toutes les API d'un compte dans une région spécifiée. La limite du taux au niveau du compte peut être augmentée à la demande. Des limitations plus élevées sont possibles avec des API qui ont des délais d'attente plus courts et des charges utiles plus petites. Pour demander une augmentation des limitations au niveau du compte par Région, contactez le Centre de support AWS. Pour plus d’informations, consultez Quotas et remarques importantes. Notez que ces limites ne peuvent pas être supérieures aux limites d' AWS étranglement.

Limitation au niveau de l’acheminement

Vous pouvez définir une limitation au niveau des acheminements, afin de remplacer les limitations de requêtes au niveau du compte pour une étape spécifique ou pour des acheminements particuliers de votre API. Les limites de limitation d'acheminement par défaut ne peuvent pas dépasser les limites de débit au niveau du compte.

Vous pouvez configurer la limitation au niveau des acheminements à l'aide de AWS CLI. La commande suivante configure la limitation personnalisée pour l'étape et l'acheminement spécifiés d'une API.

```
aws apigatewayv2 update-stage \
    --api-id a1b2c3d4 \
    --stage-name dev \
    --route-settings '{"GET /pets":
{"ThrottlingBurstLimit":100,"ThrottlingRateLimit":2000}}'
```

Configuration de l'authentification TLS mutuelle pour une API HTTP

L'authentification TLS mutuelle nécessite une authentification bidirectionnelle entre le client et le serveur. Avec l'authentification TLS mutuelle, les clients doivent présenter des certificats X.509 pour vérifier leur identité afin d'accéder à votre API. Le protocole TLS mutuel est une exigence courante pour l'Internet des objets (IoT) et business-to-business les applications.

Vous pouvez utiliser l'authentification TLS mutuelle avec d'autres opérations d'autorisation et d'authentification prises en charge par API Gateway. API Gateway transmet les certificats que les clients fournissent aux mécanismes d'autorisation Lambda et aux intégrations de backend.

> **Important**
>
> Par défaut, les clients peuvent appeler votre API en utilisant le point de terminaison `execute-api` généré par API Gateway pour votre API. Pour vous assurer que les clients peuvent accéder à votre API en utilisant uniquement un nom de domaine personnalisé avec authentification TLS mutuelle, désactivez le point de terminaison par défaut `execute-api`. Pour en savoir plus, consultez la section the section called “Désactiver le point de terminaison par défaut”.

Conditions préalables pour l'authentification TLS mutuelle

Pour configurer des authentifications TLS mutuelles, vous avez besoin des éléments suivants :

- Un nom de domaine personnalisé
- Au moins un certificat configuré AWS Certificate Manager pour votre nom de domaine personnalisé
- Un magasin de confiance configuré et chargé vers Amazon S3

Noms de domaine personnalisés

Pour activer l'authentification TLS mutuelle pour une API HTTP, vous devez configurer un nom de domaine personnalisé pour votre API. Vous pouvez activer l'authentification TLS mutuelle pour un nom de domaine personnalisé, puis fournir le nom de domaine personnalisé aux clients. Pour accéder à une API à l'aide d'un nom de domaine personnalisé sur lequel l'authentification TLS mutuelle est activée, les clients doivent présenter des certificats approuvés dans les demandes d'API. Vous trouverez plus d'informations dans the section called “Noms de domaine personnalisés”.

Utilisation de certificats AWS Certificate Manager émis

Vous pouvez demander un certificat approuvé publiquement directement à partir d'ACM ou importer des certificats publics ou auto-signés. Pour configurer un certificat dans ACM, accédez à ACM. Si vous souhaitez importer un certificat, poursuivez votre lecture dans la section suivante.

Utilisation d'un AWS Private Certificate Authority certificat ou d'un produit importé

Pour utiliser un certificat importé dans ACM ou un certificat provenant AWS Private Certificate Authority d'un protocole TLS mutuel, API Gateway a besoin d'un certificat `ownershipVerificationCertificate` émis par ACM. Ce certificat de propriété est utilisé uniquement pour vérifier que vous disposez des autorisations nécessaires pour utiliser le nom de domaine. Il n'est pas utilisé pour la poignée de main TLS. Si vous n'avez pas encore de `ownershipVerificationCertificate`, accédez àhttps://console.aws.amazon.com/acm/ pour en configurer un.

Vous devrez conserver ce certificat valide pendant toute la durée de vie de votre nom de domaine. Si un certificat expire et que le renouvellement automatique échoue, toutes les mises à jour du nom de domaine seront verrouillées. Vous devrez mettre à jour le `ownershipVerificationCertificateArn`avec un `ownershipVerificationCertificate` valide avant de pouvoir effectuer d'autres modifications. Le `ownershipVerificationCertificate` ne peut pas être utilisé comme certificat de serveur pour un autre domaine TLS mutuel dans API Gateway. Si un certificat est directement réimporté dans ACM, le diffuseur doit rester le même.

Configuration de votre magasin de confiance

Les magasins de confiance sont des fichiers texte avec une `.pem` extension de fichier. Il s'agit d'une liste de certificats approuvés provenant des autorités de certification. Pour utiliser l'authentification TLS mutuelle, créez un magasin de confiance de certificats X.509 auxquels vous faites confiance pour accéder à votre API.

Vous devez inclure la chaîne de confiance complète à partir du certificat de l'autorité de certification émettrice jusqu'au certificat de l'autorité de certification racine, dans votre magasin de confiance. API Gateway accepte les certificats clients émis par toute autorité de certification présente dans la chaîne de confiance. Les certificats peuvent être délivrés par des autorités de certification publiques ou privées. Les certificats peuvent avoir une longueur de chaîne maximale de quatre. Vous pouvez également fournir des certificats auto-signés. Les algorithmes de hachage suivants sont pris en charge dans le magasin de confiance :

- SHA-256 ou plus
- RSA-2048 ou plus
- ECDSA-256 ou plus

API Gateway valide un certain nombre de propriétés de certificat. Vous pouvez utiliser des mécanismes d'autorisation Lambda pour effectuer des vérifications supplémentaires lorsqu'un client appelle une API, notamment pour vérifier si un certificat a été révoqué. API Gateway valide les propriétés suivantes :

Validation	Description
Syntaxe X.509	Le certificat doit répondre aux exigences de syntaxe X.509.
Intégrité	Le contenu du certificat ne doit pas avoir été modifié par rapport à celui signé par l'autorité de certification à partir du magasin de confiance.
Validité	La période de validité du certificat doit en cours.
Chaînage de noms / chaînage de clés	Les noms et les objets des certificats doivent former une chaîne ininterrompue. Les certificats peuvent avoir une longueur de chaîne maximale de quatre.

Chargez le magasin de confiance dans un fichier unique d'un compartiment Amazon S3.

Example certificates.pem

```
-----BEGIN CERTIFICATE-----
<Certificate contents>
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
<Certificate contents>
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
<Certificate contents>
-----END CERTIFICATE-----
...
```

La AWS CLI commande suivante est chargée dans `certificates.pem` votre compartiment Amazon S3.

```
aws s3 cp certificates.pem s3://bucket-name
```

Configuration de l'authentification TLS mutuelle pour un nom de domaine personnalisé

Pour configurer l'authentification TLS mutuelle pour une API HTTP, vous devez utiliser un nom de domaine personnalisé régional pour votre API, avec une version minimale TLS 1.2. Pour de plus amples informations sur la création et la configuration d'un nom de domaine personnalisé, veuillez consulter the section called “Configuration d'un nom de domaine personnalisé régional”.

> **Note**
>
> L'authentification TLS mutuelle n'est pas prise en charge pour les API privées.

Une fois que vous avez chargé votre magasin de confiance dans Amazon S3, vous pouvez configurer votre nom de domaine personnalisé pour utiliser l'authentification TLS mutuelle. Collez le texte suivant (barres obliques incluses) dans un terminal :

```
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
```

Après avoir créé le nom de domaine, vous devrez configurer des enregistrements DNS et des mappages de chemin de base pour les opérations API. Pour en savoir plus, consultez la section Configuration d'un nom de domaine personnalisé régional dans API Gateway.

Appeler une API à l'aide d'un nom de domaine personnalisé qui nécessite une authentification TLS mutuelle

Pour appeler une API avec l'authentification TLS mutuelle activée, les clients doivent présenter un certificat approuvé dans la demande d'API. Lorsqu'un client tente d'appeler votre API, API Gateway recherche le diffuseur du certificat client dans votre magasin de confiance. Pour que l'API Gateway puisse poursuivre la demande, le diffuseur du certificat et la chaîne de confiance complète jusqu'au certificat d'autorité de certification racine doivent se trouver dans votre magasin de confiance.

L'exemple de commande `curl` suivant envoie à `api.example.com`, une demande qui inclut `my-cert.pem` dans la demande. `my-key.key` est la clé privée du certificat.

```
curl -v --key ./my-key.key --cert ./my-cert.pem api.example.com
```

Votre API est appelée uniquement si votre magasin de confiance approuve le certificat. Les conditions suivantes provoqueront l'échec de la poignée de main de l'API Gateway avec TLS et le refus de la demande d'un `403` code d'état. Si votre certificat :

- n'est pas approuvé
- a expiré
- n'utilise pas d'algorithme pris en charge

> **Note**
>
> L'API Gateway ne vérifie pas si un certificat a été révoqué.

Mise à jour de votre magasin de confiance

Pour mettre à jour les certificats dans votre magasin de confiance, chargez un nouveau lot de certificats dans Amazon S3. Vous pourrez ensuite mettre à jour votre nom de domaine personnalisé de manière à utiliser le certificat mis à jour.

Utilisez la gestion des versions Amazon S3 pour gérer plusieurs versions de votre magasin de confiance. Lorsque vous mettez à jour votre nom de domaine personnalisé de manière à utiliser une nouvelle version du magasin de confiance, API Gateway renvoie des avertissements si les certificats ne sont pas valides.

API Gateway ne génère des avertissements de certificat que lorsque vous mettez à jour votre nom de domaine. API Gateway ne vous avertit pas si un certificat précédemment chargé arrive à expiration.

La AWS CLI commande suivante met à jour un nom de domaine personnalisé afin d'utiliser une nouvelle version de Truststore.

```
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'
```

Désactiver l'authentification TLS mutuelle

Pour désactiver l'authentification TLS mutuelle pour un nom de domaine personnalisé, supprimez le magasin de confiance de votre nom de domaine personnalisé, comme indiqué dans la commande suivante.

```
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=''
```

Résolution des problèmes liés aux avertissements de certificat

Lors de la création d'un nom de domaine personnalisé avec authentification TLS mutuelle, API Gateway renvoie des avertissements si les certificats du magasin de confiance ne sont pas valides. Cela peut également se produire lors de la mise à jour d'un nom de domaine personnalisé de manière à utiliser un nouveau magasin de confiance. Les avertissements indiquent le problème lié au certificat et l'objet du certificat ayant produit l'avertissement. L'authentification TLS mutuelle reste activée pour votre API, mais certains clients risquent de ne pas être en mesure d'accéder à votre API.

Vous devrez décoder les certificats de votre magasin de confiance pour identifier le certificat ayant émis l'avertissement. Vous pouvez utiliser des outils tels que `openssl` pour décoder les certificats et identifier leur objet.

La commande suivante affiche le contenu d'un certificat, y compris son objet.

```
openssl x509 -in certificate.crt -text -noout
```

Mettez à jour ou supprimez les certificats ayant produit des avertissements, puis chargez un nouveau magasin de confiance dans Amazon S3. Après avoir chargé le nouveau magasin de confiance, mettez à jour votre nom de domaine personnalisé de manière à utiliser le nouveau magasin de confiance.

Résolution des conflits de noms de domaine

L'erreur `"The certificate subject <certSubject> conflicts with an existing certificate from a different issuer."` signifie que plusieurs autorités de certification ont émis un certificat pour ce domaine. Pour chaque sujet du certificat, il ne peut y avoir qu'un seul diffuseur dans API Gateway pour les domaines TLS mutuels. Vous devrez obtenir tous vos certificats pour ce sujet par l'intermédiaire d'un seul diffuseur. Si le problème est dû à un certificat dont vous n'avez pas le contrôle, mais pour lequel vous pouvez prouver la propriété du nom de domaine, contactez AWS Support pour ouvrir un ticket.

Dépannage des messages d'état des noms de domaine

`PENDING_CERTIFICATE_REIMPORT` : cela signifie que vous avez réimporté un certificat dans ACM qui a provoqué l'échec de la validation, car le nouveau certificat a un SAN (nom alternatif de l'objet) qui n'est pas couvert par le `ownershipVerificationCertificate` ou par l'objet ou que les SAN du certificat ne couvrent pas le nom de domaine. Quelque chose peut être configuré de manière incorrecte ou un certificat non valide a été importé. Vous devez réimporter un certificat valide dans ACM. Pour en savoir plus sur la validation, consultez Validation de la propriété de domaine.

`PENDING_OWNERSHIP_VERIFICATION` : cela signifie que votre certificat précédemment vérifié a expiré et qu'ACM n'a pas pu le renouveler automatiquement. Vous devrez renouveler le certificat ou demander un nouveau certificat. Pour en savoir plus sur le renouvellement du certificat, vous trouverez des informations supplémentaires dans le guide : Résolution des problèmes liés au renouvellement des certificats gérés par ACM.

Surveillance de votre API HTTP

Vous pouvez utiliser CloudWatch les métriques et CloudWatch les journaux pour surveiller les API HTTP. En combinant les journaux et les métriques, vous pouvez enregistrer les erreurs et surveiller les performances de votre API.

> **Note**
>
> API Gateway peut ne pas générer de journaux et de métriques dans les cas suivants :
>
> - Nombre d'erreurs de demande d'entité 413 trop important
> - Nombre d'erreurs de demande 429 trop important
> - Erreurs de la série 400 provenant de demandes envoyées à un domaine personnalisé qui n'a pas de mappage d'API
> - Erreurs de la série 500 causées par des défaillances internes

Rubriques

Utilisation des métriques pour les API HTTP

Vous pouvez surveiller l'exécution des API en utilisant CloudWatch, qui collecte et traite les données brutes d'API Gateway en near-real-time métriques lisibles. Ces statistiques sont enregistrées pour une durée de 15 mois et, par conséquent, vous pouvez accéder aux informations historiques et acquérir un meilleur point de vue de la façon dont votre service ou application web s'exécute. Par défaut, les données métriques d'API Gateway sont automatiquement envoyées par CloudWatch intervalles d'une minute. Pour surveiller vos indicateurs, créez un CloudWatch tableau de bord pour votre API. Pour plus d'informations sur la création d'un CloudWatch tableau de bord, consultez la section Création d'un CloudWatch tableau de bord dans le guide de CloudWatch l'utilisateur Amazon. Pour plus d'informations, consultez Qu'est-ce qu'Amazon CloudWatch ? dans le guide de CloudWatch l'utilisateur Amazon.

Les métriques suivantes sont prises en charge pour les API HTTP. Vous pouvez également activer les métriques détaillées pour écrire des métriques au niveau de l'itinéraire à Amazon. CloudWatch

Métrique	Description
4xx	Nombre d'erreurs côté client capturées dans une période donnée.
5xx	Nombre d'erreurs côté serveur capturées dans une période donnée.
Nombre	Nombre total de demandes d'API sur une période donnée.
IntegrationLatency	Délai entre le moment de la transmission de la demande au backend par API Gateway et celui de la réception de la réponse du backend.
Latence	Délai entre le moment de la réception par API Gateway d'une demande d'un client et celui du renvoi de la réponse au client. La latence prend en compte la latence d'intégra tion et autres surcharges d'API Gateway.
DataProcessed	Quantité de données traitées en octets.

Vous pouvez utiliser les dimensions du tableau suivant pour filtrer les métriques API Gateway.

Dimension	Description
ApiId	Filtre les métriques API Gateway pour une API avec l'ID d'API spécifié.
ApiId, Scène	Filtre les métriques API Gateway pour trouver une étape d'API portant l'ID d'API et l'ID d'étape spécifiés.
ApiId, Méthode, ressource, étape	Filtre les métriques d'API Gateway pour une méthode d'API avec l'ID d'API, l'ID d'étape, le chemin de ressource et l'ID de route spécifiés. API Gateway n'enverra pas ces métriques à moins que vous n'ayez explicite ment activé CloudWatch les métriques détaillées. Vous pouvez le faire en appelant l'UpdateStageaction de l'API REST API Gateway V2 pour mettre à jour la `detailedM etricsEnabled` propriété sur`true`. Vous pouvez également appeler la AWS CLI commande update-st age pour mettre à jour la `DetailedMetricsEna bled` propriété en. `true` L'activation de ces métriques implique des frais supplémen taires pour votre compte.

Dimension	Description
	Pour plus d'informations sur les tarifs, consultez Amazon CloudWatch Pricing.

Configuration de la journalisation pour une API HTTP

Vous pouvez activer la journalisation pour écrire des CloudWatch journaux dans Logs. Vous pouvez utiliser des variables de journalisation pour personnaliser le contenu de vos journaux.

Pour activer la journalisation pour une API HTTP, vous devez effectuer les opérations suivantes.

1. Vérifiez que votre utilisateur dispose des autorisations nécessaires pour activer la journalisation.
2. Créez un groupe de CloudWatch journaux Logs.
3. Indiquez l'ARN du groupe de CloudWatch journaux Logs pour une étape de votre API.

Autorisations pour activer la journalisation

Pour activer la journalisation pour une API, votre utilisateur doit disposer des autorisations suivantes.

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": "*"
        }
    ]
}
```

Création d'un groupe de journaux et activation de la journalisation pour les API HTTP

Vous pouvez créer un groupe de journaux et activer la journalisation des accès à l'aide du AWS Management Console ou du AWS CLI.

AWS Management Console

1. Créez un groupe de journaux .

   Pour savoir comment créer un groupe de journaux à l'aide de la console, consultez Créer un groupe de CloudWatch journaux dans le guide de l'utilisateur d'Amazon Logs.
2. Connectez-vous à la console API Gateway à l'adresse : https://console.aws.amazon.com/apigateway.
3. Choisissez une API HTTP.
4. Sous l'onglet Monitor (Surveiller) du panneau de navigation principal, choisissez Logging (Journalisation).
5. Sélectionnez une étape pour activer la journalisation, puis choisissez Select (Sélectionner).
6. Choisissez Edit (Modifier) pour activer la journalisation des accès.
7. Activez la journalisation des accès, entrez un CloudWatch journal et sélectionnez un format de journal.
8. Choisissez Enregistrer.

AWS CLI

La AWS CLI commande suivante crée un groupe de journaux.

```
aws logs create-log-group --log-group-name my-log-group
```

Vous avez besoin de l'Amazon Resource Name (ARN) de votre groupe de journaux pour activer la journalisation. *Le format ARN est arn:aws:logs : region : account-id:log-group :. log-group-name*

La AWS CLI commande suivante active la journalisation pour le `$default` stage d'une API HTTP.

```
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"}'
```

Exemples de format de journal

Des exemples de format de journal utilisé couramment sont disponibles dans la console API Gateway et répertoriés ci-dessous.

- `CLF` (Format de journal commun):

```
$context.identity.sourceIp - - [$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 (valeurs séparées par des virgules) :

```
$context.identity.sourceIp,$context.requestTime,$context.httpMethod,
$context.routeKey,$context.protocol,$context.status,$context.responseLength,
$context.requestId,$context.extendedRequestId
```

Personnalisation des journaux d'accès à l'API HTTP

Vous pouvez utiliser les variables suivantes pour personnaliser les journaux d'accès des API HTTP. Pour de plus amples informations sur les journaux d'accès pour les API HTTP, veuillez consulter Configuration de la journalisation pour une API HTTP.

Paramètre	Description
`$context.accountId`	ID de AWS compte du propriétaire de l'API.
`$context.apiId`	Identifiant qu'API Gateway attribue à votre API.
`$context.authorizer.claims.`*`property`*	Une propriété des demandes renvoyées par le jeton Web JSON (JWT) une fois que l'appelant de la méthode a été authentifié avec succès, telle que. `$context.authorizer.claims.username` Pour plus d'informations, consultez Contrôle de l'accès aux API HTTP avec les mécanismes d'autorisation JWT. **Note** L'appel de `$context.authorizer.claims` renvoie la valeur null.

Paramètre	Description
`$context.authorizer.error`	Message d'erreur renvoyé par un mécanisme d'autorisation.
`$context.authorizer.principalId`	Identification de l'utilisateur principal renvoyée par un mécanisme d'autorisation Lambda.

`$context.authorizer.` *property*	Valeur de la paire clé-valeur spécifiée du mappage `context` renvoyé par une fonction API Gateway du mécanisme d'autorisation Lambda. Par exemple, si le mécanisme d'autorisation retourne le mappage `context` suivant :
	`"context" : {`
	`"key": "value",`
	`"numKey": 1,`
	`"boolKey": true`
	`}`
	l'appel `$context.authorizer.key` renvoie la chaîne `"value"`, l'appel `$context.authorizer.numKey` renvoie 1 et l'appel `$context.authorizer.boolKey` renvoie `true`.

`$context.awsEndpointRequestId`	L'ID de demande du AWS point de terminaison indiqué dans l'`x-amzn-requestId` en-tête `x-amz-request-id` ou.
`$context.awsEndpointRequestId2`	L'ID de demande du AWS point de terminaison indiqué dans l'`x-amz-id-2` en-tête.

Paramètre	Description
`$context.customDomain.basePathMatched`	Chemin d'accès d'un mappage d'API correspondant à une demande entrante. Applicable lorsqu'un client utilise un nom de domaine personnalisé pour accéder à une API. Par exemple, si un client envoie une demande à `https://api.example.com/v1/orders/1234` et que cette demande correspond au mappage d'API dont le chemin d'accès est `v1/orders`, la valeur est `v1/orders`. Pour en savoir plus, consultez la section the section called “Mappages d'API”.
`$context.dataProcessed`	Quantité de données traitées en octets.
`$context.domainName`	Nom de domaine complet utilisé pour invoquer l'API. Il doit être identique à l'en-tête `Host` entrant.
`$context.domainPrefix`	Première étiquette de `$context.domainName`.
`$context.error.message`	Chaîne contenant un message d'erreur API Gateway.
`$context.error.messageString`	La valeur entre guillemets de `$context.error.message`, à savoir `"$context.error.message"`.
`$context.error.responseType`	Type de `GatewayResponse`. Pour de plus amples informations, veuillez consulter the section called “Métriques” et the section called “Configuration de réponses de passerelle pour personnaliser des réponses d'erreur”.
`$context.extendedRequestId`	Équivalent à `$context.requestId`.

Paramètre	Description
`$context.httpMethod`	La méthode HTTP utilisée. Les valeurs valides sont les suivantes : `DELETE`, `GET`, `HEAD`, `OPTIONS`, `PATCH`, `POST` et `PUT`.
`$context.identity.accountId`	L'ID de AWS compte associé à la demande. Pris en charge pour les routes qui utilisent l'autorisation IAM.
`$context.identity.caller`	Identifiant principal de l'appelant qui a signé la demande. Pris en charge pour les routes qui utilisent l'autorisation IAM.
`$context.identity.cognitoAuthenticationProvider`	Liste séparée par des virgules des fournisseurs d'authentification Amazon Cognito utilisés par l'appelant à l'origine de la demande. Disponible uniquement si la demande a été signée avec les informations d'identification Amazon Cognito. Par exemple, pour une identité provenant d'un pool d'utilisateurs Amazon Cognito, `cognito-idp.`*`region`*`.amazonaws.com/`*`user_pool_id`*`,cognito-idp.`*`region`*`.amazonaws.com/`*`user_pool_id`*`:CognitoSignIn:`*`token subject claim`* Pour de plus amples informations, veuillez consulter Utilisation des identités fédérées dans le Manuel du développeur Amazon Cognito.

Paramètre	Description
`$context.identity.cognitoAuthenticationType`	Type d'authentification Amazon Cognito de l'appelant effectuant la demande. Disponible uniquement si la demande a été signée avec les informations d'identification Amazon Cognito. Les valeurs possibles incluent `authenticated` pour les identités authentifiées et `unauthenticated` pour les identités non authentifiées.
`$context.identity.cognitoIdentityId`	ID d'identité Amazon Cognito de l'appelant effectuant la demande. Disponible uniquement si la demande a été signée avec les informations d'identification Amazon Cognito.
`$context.identity.cognitoIdentityPoolId`	ID de groupe d'identités Amazon Cognito de l'appelant effectuant la demande. Disponible uniquement si la demande a été signée avec les informations d'identification Amazon Cognito.
`$context.identity.principalOrgId`	ID d'organisation AWS. Pris en charge pour les routes qui utilisent l'autorisation IAM.
`$context.identity.clientCert.clientCertPem`	Certificat client codé PEM présenté par le client lors de l'authentification TLS mutuelle. Présent lorsqu'un client accède à une API à l'aide d'un nom de domaine personnalisé pour lequel l'authentification TLS mutuelle est activée.
`$context.identity.clientCert.subjectDN`	Nom distinctif de l'objet du certificat présenté par un client. Présent lorsqu'un client accède à une API à l'aide d'un nom de domaine personnalisé pour lequel l'authentification TLS mutuelle est activée.

Paramètre	Description
`$context.identity.clientCert.issuerDN`	Nom distinctif de l'émetteur du certificat présenté par un client. Présent lorsqu'un client accède à une API à l'aide d'un nom de domaine personnalisé pour lequel l'authentification TLS mutuelle est activée.
`$context.identity.clientCert.serialNumber`	Numéro de série du certificat. Présent lorsqu'un client accède à une API à l'aide d'un nom de domaine personnalisé pour lequel l'authentification TLS mutuelle est activée.
`$context.identity.clientCert.validity.notBefore`	Date avant laquelle le certificat n'est pas valide. Présent lorsqu'un client accède à une API à l'aide d'un nom de domaine personnalisé pour lequel l'authentification TLS mutuelle est activée.
`$context.identity.clientCert.validity.notAfter`	Date après laquelle le certificat n'est pas valide. Présent lorsqu'un client accède à une API à l'aide d'un nom de domaine personnalisé pour lequel l'authentification TLS mutuelle est activée.
`$context.identity.sourceIp`	L'adresse IP source de la connexion TCP envoyant la demande au point de terminaison de l'API Gateway.
`$context.identity.user`	Identifiant principal de l'utilisateur qui sera autorisé à accéder aux ressources. Pris en charge pour les routes qui utilisent l'autorisation IAM.
`$context.identity.userAgent`	En-tête `User-Agent` de l'appelant d'API.

Paramètre	Description
`$context.identity.userArn`	ARN (Amazon Resource Name) de l'utilisateur identifié après l'authentification. Pris en charge pour les routes qui utilisent l'autorisation IAM. Pour de plus amples informations, veuillez consulter https://docs.aws.amazon.com/IAM/latest/UserGuide/id_users.html.
`$context.integration.error`	Message d'erreur renvoyé à partir d'une intégration. Équivalent à `$context.integrationErrorMessage` .
`$context.integration.integrationStatus`	Pour l'intégration du proxy Lambda, le code d'état est renvoyé par le code de fonction Lambda principal AWS Lambda, et non par le code de fonction Lambda principal.
`$context.integration.latency`	Latence d'intégration en millisecondes (ms). Équivalent à `$context.integrationLatency` .
`$context.integration.requestId`	ID de demande du AWS point de terminaison. Équivalent à `$context.awsEndpointRequestId` .
`$context.integration.status`	Code d'état renvoyé à partir d'une intégration. Pour les intégrations de proxy Lambda, code d'état que votre code de fonction Lambda renvoie.
`$context.integrationErrorMessage`	Chaîne contenant un message d'erreur d'intégration.
`$context.integrationLatency`	Latence d'intégration en millisecondes (ms).

Paramètre	Description
`$context.integrationStatus`	Pour l'intégration du proxy Lambda, ce paramètre représente le code d'état renvoyé par la fonction Lambda principale AWS Lambda, et non par celle-ci.
`$context.path`	Chemin d'accès de la demande. Par exemple, `/{stage}/root/child` .
`$context.protocol`	Protocole de demande, par exempl, `HTTP/1.1`. **Note** Les API d'API Gateway peuvent accepter les requêtes HTTP/2, mais API Gateway envoie les requêtes aux intégrations backend en utilisant HTTP/1.1. Par conséquent, le protocole de requête est enregistré comme HTTP/1.1 même si un client envoie une requête qui utilise HTTP/2.
`$context.requestId`	ID attribué par API Gateway à la demande d'API.
`$context.requestTime`	Durée des demandes au format CLF (`dd/MMM/yyyy:HH:mm:ss +-hhmm`).
`$context.requestTimeEpoch`	Durée des demandes au format Epoch.
`$context.responseLatency`	Latence de la réponse en millisecondes (ms).
`$context.responseLength`	La longueur de la charge utile de la réponse en octets.
`$context.routeKey`	La clé de routage de la demande d'API, par exemple `/pets`.

Paramètre	Description
`$context.stage`	Étape de déploiement de la demande d'API (par exemple, `beta` ou `prod`).
`$context.status`	Statut de la réponse de la méthode.

Résolution des problèmes liés aux API HTTP

Les rubriques suivantes fournissent des conseils de dépannage pour les erreurs et problèmes que vous pouvez rencontrer en utilisant des API HTTP.

Rubriques

- Résolution des problèmes liés aux intégrations Lambda d'API HTTP
- Résolution des problèmes liés aux mécanismes d'autorisation JWT pour les API HTTP

Résolution des problèmes liés aux intégrations Lambda d'API HTTP

La section suivante fournit des conseils de dépannage pour les erreurs et problèmes que vous pouvez rencontrer en utilisant les AWS Lambda intégrations avec les API HTTP.

Problème : Mon API avec une intégration Lambda renvoie **`{"message":"Internal Server Error"}`**

Pour résoudre l'erreur interne du serveur, ajoutez la variable de journalisation `$context.integrationErrorMessage` à votre format de journal et affichez vos journaux d'API HTTP. Pour ce faire, procédez comme suit :

Pour créer un groupe de journaux à l'aide du AWS Management Console

1. Ouvrez la CloudWatch console à l'adresse https://console.aws.amazon.com/cloudwatch/.
2. Choisissez Groupes de journaux.
3. Sélectionnez Créer un groupe de journaux.
4. Saisissez un nom de groupe de journaux, puis choisissez Créer.
5. Notez l'Amazon Resource Name (ARN) de votre groupe de journaux. *`Le format ARN est arn:aws:logs : region : account-id:log-group :. log-group-name`* Vous avez

besoin de l'ARN du groupe de journaux pour activer la journalisation de l'accès pour votre API HTTP.

Pour ajouter la variable de journalisation **`$context.integrationErrorMessage`**

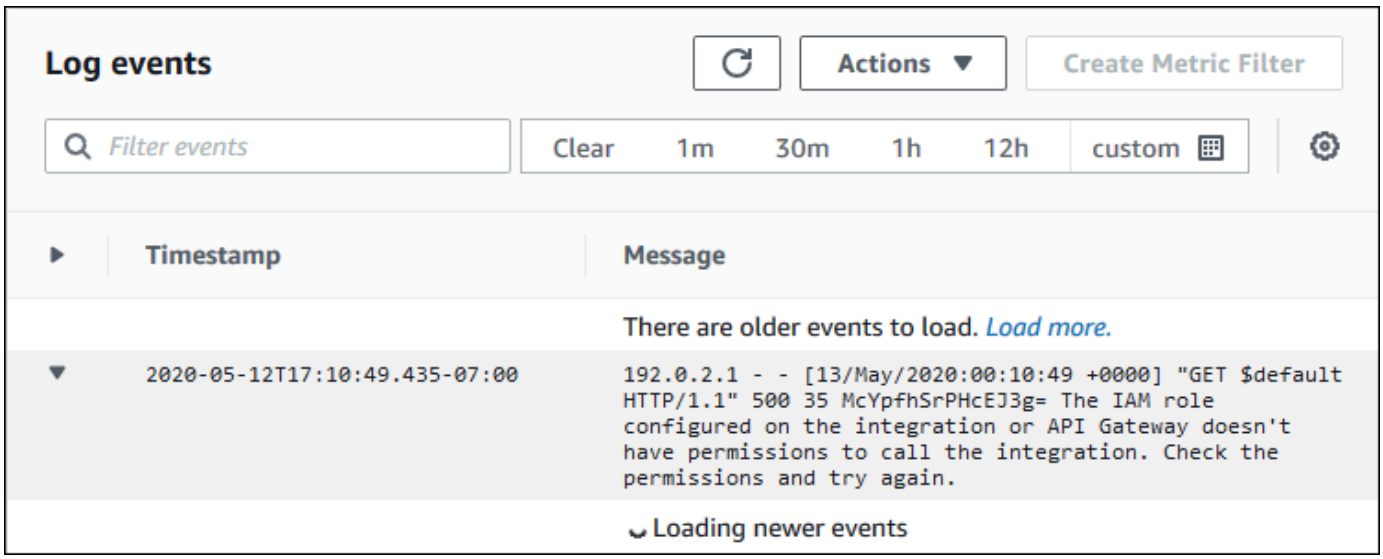
1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Choisissez votre API HTTP.
3. Sous Moniteur, choisissez Journalisation.
4. Sélectionnez une étape de votre API.
5. Choisissez Modifier, puis activez la journalisation des accès.
6. Pour Destination du journal, saisissez l'ARN du groupe de journaux que vous avez créé à l'étape précédente.
7. Pour Format de journal, choisissez CLF. API Gateway crée un exemple de format de journal.
8. Ajouter `$context.integrationErrorMessage` à la fin du format de journal.
9. Choisissez Enregistrer.

Pour afficher les journaux de votre API

1. Génération de journaux Utilisez un navigateur ou `curl` pour appeler votre API.

```
$curl https://api-id.execute-api.us-west-2.amazonaws.com/route
```

2. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
3. Choisissez votre API HTTP.
4. Sous Moniteur, choisissez Journalisation.
5. Sélectionnez l’étape de votre API pour laquelle vous avez activé la journalisation.
6. Choisissez Afficher les connexions CloudWatch.
7. Choisissez le dernier flux de journaux pour afficher vos journaux d'API HTTP.
8. Votre entrée de journal doit ressembler à la suivante :



Comme nous avons ajouté `$context.integrationErrorMessage` au format de journal, nous voyons un message d'erreur dans nos journaux qui résume le problème.

Vos journaux peuvent inclure un autre message d'erreur indiquant l'existence d'un problème avec votre code de fonction Lambda. Dans ce cas, vérifiez votre code de fonction Lambda et assurez-vous que votre fonction Lambda renvoie une réponse au format requis. Si vos journaux n'incluent pas de message d'erreur, ajoutez `$context.error.message` et `$context.error.responseType` à votre format de journal pour plus d'informations pour vous aider à résoudre les problèmes.

Si c'est le cas, les journaux montrent qu'API Gateway ne disposait pas des autorisations requises pour appeler la fonction Lambda.

Lorsque vous créez une intégration Lambda dans la console API Gateway, API Gateway configure automatiquement les autorisations pour appeler la fonction Lambda. Lorsque vous créez une intégration Lambda à l'aide du AWS CLI AWS CloudFormation, ou d'un SDK, vous devez autoriser API Gateway à appeler la fonction. Les exemples de AWS CLI commandes suivants autorisent différentes routes d'API HTTP à invoquer une fonction Lambda.

Example Exemple — Pour l'**`$default`**étape et l'**`$default`**itinéraire d'une API HTTP

```
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 Exemple — Pour l'**prod**étape et l'**test**itinéraire d'une API HTTP

```
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"
```

Vérifiez la stratégie de fonction dans l'onglet Permissions (Autorisations) de la console Lambda.

Essayez à nouveau d'appeler votre API. Vous devriez voir la réponse de votre fonction Lambda.

Résolution des problèmes liés aux mécanismes d'autorisation JWT pour les API HTTP

La section suivante fournit des conseils de dépannage pour les erreurs et problèmes que vous pouvez rencontrer en utilisant des mécanismes d'autorisation JSON Web Token (JWT) avec des API HTTP.

Problème : Mon API renvoie **`401 {"message":"Unauthorized"}`**

Vérifiez l'en-tête `www-authenticate` dans la réponse de l'API.

La commande suivante utilise `curl` pour envoyer une demande à une API avec un mécanisme d'autorisation JWT qui utilise `$request.header.Authorization` comme source d'identité.

```
$curl -v -H "Authorization: token" https://api-id.execute-api.us-
west-2.amazonaws.com/route
```

La réponse de l'API inclut un en-tête `www-authenticate`.

```
...
< 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"}}
```

Dans ce cas, l'en-tête `www-authenticate` indique que le jeton n'a pas été émis pour un public ciblé approprié. Pour qu'API Gateway autorise une demande, la demande `aud` ou `client_id` de JWT doit correspondre à l'une des entrées de public ciblé configurées pour le mécanisme d'autorisation. API Gateway `client_id` ne valide que s'il n'`aud`est pas présent. Lorsque `aud` les deux `client_id` sont présents, API Gateway évalue. `aud`

Vous pouvez également décoder un JWT et vérifier qu'il correspond à l'émetteur, au public ciblé et aux portées dont votre API a besoin. Le site jwt.io peut déboguer des JWT dans le navigateur. L'OpenID Foundation gère également une liste de bibliothèques pour travailler avec des JWT.

Pour de plus amples informations sur les mécanismes d'autorisation JWT, veuillez consulter Contrôle de l'accès aux API HTTP avec les mécanismes d'autorisation JWT.

Utilisation des WebSocket API

Dans WebSocket API Gateway, une API est un ensemble de WebSocket routes intégrées aux points de terminaison HTTP du backend, aux fonctions Lambda ou à d'autres services. AWS Vous pouvez utiliser les fonctions API Gateway pour vous aider dans tous les aspects du cycle de vie de l'API, de la création à la surveillance de vos API de production.

Les API WebSocket API Gateway sont bidirectionnelles. Un client peut envoyer des messages à un service, et les services peuvent indépendamment envoyer des messages aux clients. Ce comportement bidirectionnel permet d'enrichir les interactions entre le client et le service, car les services peuvent transmettre des données aux clients sans que les clients aient à faire une demande explicite. WebSocket Les API sont souvent utilisées dans des applications en temps réel telles que les applications de chat, les plateformes de collaboration, les jeux multijoueurs et les plateformes de trading financier.

Pour obtenir un exemple d'application avec lequel commencer, consultez Tutoriel : Création d'une application de chat sans serveur avec une WebSocket API, Lambda et DynamoDB.

Dans cette section, vous découvrirez comment développer, publier, protéger et surveiller vos WebSocket API à l'aide d'API Gateway.

Rubriques

À propos WebSocket des API dans API Gateway

Dans API Gateway, vous pouvez créer une WebSocket API en tant que frontend dynamique pour un AWS service (tel que Lambda ou DynamoDB) ou pour un point de terminaison HTTP. L' WebSocket API appelle votre backend en fonction du contenu des messages qu'il reçoit des applications clientes.

Contrairement à une API REST, qui reçoit et répond aux demandes, une WebSocket API prend en charge la communication bidirectionnelle entre les applications clientes et votre backend. Le serveur principal peut envoyer des messages de rappel aux clients connectés.

Dans votre WebSocket API, les messages JSON entrants sont dirigés vers les intégrations du backend en fonction des routes que vous configurez. (Les messages non JSON sont dirigés vers une route `$default` que vous configurez.)

Une route comprend une clé de routage, qui correspond à la valeur attendue une fois qu'une expression de sélection de la route est évaluée. L'attribut `routeSelectionExpression` est défini au niveau de l'API. Il spécifie une propriété JSON attendue dans la charge utile du message. Pour plus d'informations sur les expressions de sélection de la route, consultez la section the section called “”.

Par exemple, si vos messages JSON contiennent une propriété `action` et que vous souhaitez effectuer différentes actions en fonction de cette propriété, votre expression de sélection de la route peut être `${request.body.action}`. Votre table de routage spécifie l'action à exécuter en mettant en correspondance la valeur de la propriété `action` avec les valeurs des clés de routage personnalisées que vous avez définies dans la table.

Trois routes prédéfinies peuvent être utilisées : `$connect`, `$disconnect` et `$default`. De plus, vous pouvez créer des routes personnalisées.

- API Gateway appelle la `$connect` route lorsqu'une connexion persistante entre le client et une WebSocket API est initiée.
- API Gateway appelle la route `$disconnect` lorsque le client ou le serveur se déconnecte de l'API.
- API Gateway appelle une route personnalisée après évaluation de l'expression de sélection de la route par rapport au message si une route correspondante est trouvée ; la correspondance détermine l'intégration appelée.
- API Gateway appelle la route `$default` si l'expression de sélection de la route ne peut pas être évaluée par rapport au message ou si aucune route correspondante n'est trouvée.

Pour plus d'informations sur les routes `$connect` et `$disconnect`, consultez la section the section called “Gestion des utilisateurs et des applications client connectés”.

Pour plus d'informations sur la route `$default` et les routes personnalisées, consultez la section the section called “Appel de votre intégration backend”.

Les services backend peuvent envoyer des données vers des applications client connectées. Pour de plus amples informations, veuillez consulter the section called “Envoi de données depuis des services backend à des clients connectés”.

Gestion des utilisateurs et des applications client connectés : routes `$connect` et `$disconnect`

Rubriques

Route `$connect`

Les applications clientes se connectent à votre WebSocket API en envoyant une demande de WebSocket mise à niveau. Si la demande aboutit, la route `$connect` est exécutée pendant l'établissement de la connexion.

Comme il s'agit d'une connexion dynamique, vous pouvez configurer l'autorisation uniquement sur l'`$connect`itinéraire. WebSocket `AuthN`/ne `AuthZ` sera effectué qu'au moment de la connexion.

Tant que l'exécution de l'intégration associée à la route `$connect` n'est pas terminée, la demande est en attente de mise à niveau et la connexion n'est pas établie. Si la demande `$connect` échoue (par exemple, en raison de l'échec de `AuthN`/`AuthZ` ou de l'intégration), la connexion ne sera pas établie.

> **Note**
>
> Si l'autorisation échoue sur `$connect`, la connexion ne sera pas établie, et le client recevra une réponse `401` ou `403`.

La configuration d'une intégration pour `$connect` est facultative. Vous devez envisager la configuration d'une intégration `$connect` dans les cas suivants :

- Vous souhaitez permettre aux clients de spécifier des sous-protocoles à l'aide du champ `Sec-WebSocket-Protocol`. Pour obtenir un exemple de code, veuillez consulter Configuration d'un `$connect` itinéraire qui nécessite un WebSocket sous-protocole.
- Vous souhaitez être averti lorsque des clients se connectent.
- Vous souhaitez limiter les connexions ou contrôler qui se connecte.

- Vous voulez que votre serveur principal renvoie des messages aux clients à l'aide d'une URL de rappel.
- Vous souhaitez stocker chaque ID de connexion et d'autres informations dans une base de données (par exemple, Amazon DynamoDB).

Transmission des informations de connexion depuis l'itinéraire **`$connect`**

Vous pouvez utiliser des intégrations proxy et non proxy pour transmettre les informations de la route `$connect` vers une base de données ou un autre Service AWS.

Pour transmettre les informations de connexion en utilisant une intégration de proxy

Vous pouvez accéder aux informations de connexion d'une intégration proxy Lambda dans l'événement. Utilisez une autre AWS Lambda fonction Service AWS ou une autre pour publier sur la connexion.

La fonction Lambda suivante montre comment utiliser l'objet `requestContext` pour enregistrer l'ID de connexion, le nom de domaine, le nom d'étape et les chaînes de requête dans un journal.

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}
```

Pour transmettre les informations de connexion en utilisant une intégration sans proxy

- Vous pouvez accéder aux informations de connexion avec une intégration non proxy. Configurez la demande d'intégration et fournissez un modèle de demande d' WebSocket API. Le modèle de mappage VTL (Velocity Template Language) suivant fournit une demande d'intégration. Cette demande envoie les informations suivantes à une intégration non proxy :

  - ID de connexion
  - Nom de domaine
  - Nom de l'environnement
  - Chemin
  - En-têtes
  - Chaînes de requête

  Cette demande envoie l'ID de connexion, le nom de domaine, le nom d'étape, les chemins, les en-têtes et les chaînes de requête à une intégration non proxy.

```
{
    "connectionId": "$context.connectionId",
    "domain": "$context.domainName",
    "stage": "$context.stage",
    "params": "$input.params()"
}
```

  Pour obtenir plus d'informations sur la configuration des transformations de données, consultez the section called “Transformations de données”.

Pour terminer la demande d'intégration, définissez `StatusCode: 200` pour la réponse d'intégration. Pour en savoir plus sur la configuration d'une réponse d'intégration, consultez Configuration d'une réponse d'intégration à l'aide de la console API Gateway.

Route **`$disconnect`**

La route `$disconnect` est exécutée une fois que la connexion est fermée.

La connexion peut être fermée par le serveur ou par le client. Comme la connexion est déjà fermée lorsqu'elle est exécutée, `$disconnect` est un événement exécuté au mieux de ce qui est possible. API Gateway fera de son mieux pour fournir l'événement `$disconnect` à votre intégration, mais il ne peut pas garantir sa fourniture.

Le serveur principal peut lancer la déconnexion à l'aide de l'API `@connections`. Pour de plus amples informations, veuillez consulter the section called “Utilisation des commandes `@connections` dans votre service backend”.

Appel de votre intégration backend : route **`$default`** et routes personnalisées

Rubriques

- Utilisation de routes pour traiter les messages
- Route $default
- Routes personnalisées
- Utilisation des intégrations WebSocket d'API API Gateway pour vous connecter à votre logique métier
- Différences importantes entre les WebSocket API et les API REST

Utilisation de routes pour traiter les messages

Dans les API WebSocket API Gateway, les messages peuvent être envoyés du client à votre service principal et vice versa. Contrairement au modèle de demande/réponse de HTTP, WebSocket le backend peut envoyer des messages au client sans que celui-ci n'entreprenne aucune action.

Les messages peuvent être de type JSON ou autre que JSON. Toutefois, seuls les messages JSON peuvent être acheminés vers des intégrations spécifiques en fonction du contenu du message. Les messages autres que JSON sont transmis au serveur principal par la route `$default`.

> **Note**
>
> API Gateway prend en charge des charges utiles de messages jusqu'à 128 Ko, avec une taille de trame maximale de 32 Ko. Si un message dépasse 32 Ko, vous devez le diviser en plusieurs trames, chacune de 32 Ko ou moins. Si un message (ou trame) plus grand est reçu, la connexion se ferme avec le code 1009.
> À l'heure actuelle, les charges utiles binaires ne sont pas prises en charge. Si une trame binaire est reçue, la connexion se ferme avec le code 1003. Il est toutefois possible de convertir les charges utiles binaires en texte. Voir the section called “Types de médias binaires”.

Avec WebSocket les API d'API Gateway, les messages JSON peuvent être routés pour exécuter un service principal spécifique en fonction du contenu du message. Lorsqu'un client envoie un message via sa WebSocket connexion, cela entraîne une demande de route vers l' WebSocket API. La demande est mise en correspondance avec la route à l'aide de la clé de routage correspondante dans API Gateway. Vous pouvez configurer une demande de route pour une WebSocket API dans la console API Gateway, à l'aide du AWS CLI ou à l'aide d'un AWS SDK.

> **Note**
>
> Dans les AWS SDK AWS CLI et, vous pouvez créer des itinéraires avant ou après avoir créé des intégrations. À l'heure actuelle, la console ne prend pas en charge la réutilisation d'intégrations. Par conséquent, vous devez d'abord créer la route, puis l'intégration pour cette route.

Vous pouvez configurer API Gateway afin qu'il exécute la validation d'une demande de routage avant de continuer avec la demande d'intégration. Si la validation échoue, API Gateway échoue à la demande sans appeler votre backend, envoie une réponse de `"Bad request body"` passerelle similaire à la suivante au client et publie les résultats de la validation dans CloudWatch Logs :

```
{"message" : "Bad request body", "connectionId": "{connectionId}", "messageId":
 "{messageId}"}
```

Cela réduit les appels inutiles à votre serveur principal et vous permet de vous concentrer sur les autres exigences de votre API.

Vous pouvez également définir une réponse de routage pour les routes de votre API afin d'activer la communication bidirectionnelle. Une réponse de routage indique quelles données seront envoyées à votre client à la fin de l'intégration d'une route particulière. Il n'est pas nécessaire de définir une réponse pour une route si, par exemple, vous souhaitez qu'un client envoie des messages à votre serveur principal sans recevoir de réponse (communication unidirectionnelle). Toutefois, si vous ne fournissez pas de réponse de routage, API Gateway ne renverra aucune information sur le résultat de votre intégration à vos clients.

Route `$default`

Chaque API WebSocket API Gateway peut avoir un `$default` itinéraire. Il s'agit d'une valeur de routage spéciale qui peut être utilisée de différentes manières :

- Vous pouvez l'utiliser conjointement avec les clés de routage définies, pour spécifier une route « de secours » (par exemple, une intégration fictive générique qui renvoie un message d'erreur spécifique) pour des messages entrants qui ne correspondent à aucune des clés de routage définies.
- Vous pouvez l'utiliser sans clés de routage définies pour spécifier un modèle de proxy qui délègue le routage à un composant backend.
- Vous pouvez l'utiliser pour spécifier une route pour des charges utiles autres que JSON.

Routes personnalisées

Si vous souhaitez invoquer une intégration spécifique sur la base du contenu du message, vous pouvez le faire en créant une route personnalisée.

Une route personnalisée utilise une clé de routage et l'intégration que vous spécifiez. Lorsqu'un message entrant contient une propriété JSON et que cette propriété équivaut à une valeur correspondant à celle de la clé de routage, API Gateway appelle l'intégration. (Pour plus d'informations, consultez the section called “À propos WebSocket des API”.)

Par exemple, supposons que vous vouliez créer une application de conversation. Vous pouvez commencer par créer une WebSocket API dont l'expression de sélection d'itinéraire est`$request.body.action`. Vous pouvez ensuite définir deux routes : `joinroom` et `sendmessage`. Une application client peut invoquer la route `joinroom` en envoyant un message similaire au suivant :

```
{"action":"joinroom","roomname":"developers"}
```

Elle peut également invoquer la route `sendmessage` en envoyant un message similaire au suivant :

```
{"action":"sendmessage","message":"Hello everyone"}
```

Utilisation des intégrations WebSocket d'API API Gateway pour vous connecter à votre logique métier

Après avoir configuré un itinéraire pour une API WebSocket API Gateway, vous devez spécifier l'intégration que vous souhaitez utiliser. Comme pour une route, qui peut faire l'objet d'une demande et d'une réponse de routage, une intégration peut être associée à une demande d'intégration et à une réponse d'intégration. Une demande d'intégration contient les informations attendues par votre serveur principal pour traiter la demande provenant de votre client. Une réponse d'intégration contient les données que votre backend renvoie à API Gateway, et qui peuvent être utilisées pour construire un message à envoyer au client (si une réponse de routage est définie).

Pour plus d'informations sur la configuration d'intégrations, consultez la section the section called “Intégrations”.

Différences importantes entre les WebSocket API et les API REST

Les intégrations pour les WebSocket API sont similaires aux intégrations pour les API REST, à l'exception des différences suivantes :

- Actuellement, dans la console API Gateway, vous devez d'abord créer une route, puis une intégration en tant que cible de cette route. En revanche, dans l'API et l'interface de ligne de commande, vous pouvez créer des routes et des intégrations en toute indépendance et dans n'importe quel ordre.
- Vous pouvez utiliser une même intégration pour plusieurs routes. Par exemple, si vous avez un ensemble d'actions étroitement liées entre elles, vous souhaiterez peut-être intégrer toutes ces routes dans une Fonction Lambda unique. Au lieu de définir les détails de l'intégration plusieurs fois, vous pouvez spécifier celle-ci une seule fois et l'affecter à chacune des routes connexes.

> **Note**
>
> À l'heure actuelle, la console ne prend pas en charge la réutilisation d'intégrations. Par conséquent, vous devez d'abord créer la route, puis l'intégration pour cette route.

Dans les AWS SDK AWS CLI et, vous pouvez réutiliser une intégration en définissant la cible de l'itinéraire sur une valeur de"`integrations/{integration-id}`", où *`{integration-id}`*" est l'identifiant unique de l'intégration à associer à l'itinéraire.

- API Gateway propose plusieurs expressions de sélection, que vous pouvez utiliser dans vos routes et intégrations. Vous n'avez pas besoin de vous appuyer sur le type de contenu pour sélectionner un modèle d'entrée ou un mappage de sortie. De même que pour les expressions de sélection de la route, vous pouvez définir une expression de sélection qui sera évaluée par API Gateway afin de choisir l'élément adéquat. si aucun modèle correspondant n'est trouvé, le modèle `$default` est utilisé.
  - Dans les demandes d'intégration, l'expression de sélection du modèle prend en charge `$request.body.`*`<json_path_expression>`* et les valeurs statiques.
  - Dans les réponses d'intégration, l'expression de sélection du modèle prend en charge `$request.body.`*`<json_path_expression>`*, `$integration.response.statuscode`, `$integration.response.header.`*`<headerName>`* et les valeurs statiques.

Dans le protocole HTTP, dans lequel les demandes et les réponses sont envoyées de manière synchrone, la communication est essentiellement unidirectionnelle. Dans le WebSocket protocole, la communication est bidirectionnelle. Les réponses sont asynchrones et ne sont pas nécessairement reçues par le client dans l'ordre où les messages du client ont été envoyés. De plus, le serveur principal peut envoyer des messages au client.

> **Note**
>
> Pour une route qui est configurée pour utiliser l'intégration `AWS_PROXY` ou `LAMBDA_PROXY`, la communication est unidirectionnelle et API Gateway ne transmet pas automatiquement la réponse du backend par le biais de la réponse de routage. Par exemple, dans le cas de l'intégration `LAMBDA_PROXY`, le corps de message renvoyé par la fonction Lambda n'est pas renvoyé au client. Si vous voulez que le client reçoive les réponses de l'intégration, vous devez définir une réponse de routage pour permettre la communication bidirectionnelle.

Envoi de données depuis des services backend à des clients connectés

Les API WebSocket API Gateway proposent les méthodes suivantes pour envoyer des données depuis les services principaux aux clients connectés :

- Une intégration peut envoyer une réponse, qui est renvoyée au client par une réponse de routage que vous avez définie.
- Vous pouvez utiliser l'API `@connections` pour envoyer une demande POST. Pour plus d'informations, voir the section called “Utilisation des commandes `@connections` dans votre service backend”.

WebSocket expressions de sélection dans API Gateway

Rubriques

- Expressions de sélection de la réponse de routage
- Expressions de sélection de la clé d'API
- Expressions de sélection du mappage d'API
- WebSocketrésumé de l'expression de sélection

API Gateway utilise des expressions de sélection pour évaluer le contexte de demande et de réponse, et générer une clé. La clé est ensuite utilisée pour effectuer une sélection parmi un ensemble de valeurs possibles, généralement fournies par vous, en tant que développeur de l'API. L'ensemble exact des variables prises en charge varie en fonction de l'expression en question. Les différentes expressions sont présentées plus en détail ci-après.

Pour toutes les expressions, le langage suit le même ensemble de règles :

- Une variable est précédée du préfixe `"$"`.
- Des accolades peuvent être utilisées pour définir explicitement les limites des variables (par exemple., `"${request.body.version}-beta"`.
- Plusieurs variables sont prises en charge, mais l'évaluation n'intervient qu'une seule fois (pas d'évaluation récursive).
- L'échappement du symbole du dollar ($) est effectué avec `"\"`. C'est particulièrement utile lorsque vous définissez une expression qui est mappée à la clé `$default` réservée (par exemple, `"\$default"`).
- Dans certains cas, un format de modèle est requis. Dans ce cas, l'expression doit être encapsulée avec des barres obliques (`"/"`) (par exemple, `"/2\d\d/"`) pour correspondre aux codes de statut 2*XX*.

Expressions de sélection de la réponse de routage

Une réponse de routage est utilisée pour modéliser une réponse du serveur principal au client. Pour les WebSocket API, une réponse d'itinéraire est facultative. Une fois défini, il indique à API Gateway qu'il doit renvoyer une réponse à un client à la réception d'un WebSocket message.

L'évaluation de l'expression de sélection de la réponse de routage génère une clé de réponse de routage. Au final, cette clé est utilisée pour choisir l'un des éléments `RouteResponses` associés à l'API. Cependant, à l'heure actuelle, seule la clé `$default` est prise en charge.

Expressions de sélection de la clé d'API

Cette expression est évaluée lorsque le service détermine que la demande en question doit continuer uniquement si le client fournit une clé d'API.

À l'heure actuelle, les seules deux valeurs prises en charge sont `$request.header.x-api-key` et `$context.authorizer.usageIdentifierKey`.

Expressions de sélection du mappage d'API

Cette expression est évaluée pour déterminer quelle étape d'API est sélectionné lorsqu'une demande est effectuée à l'aide d'un domaine personnalisé.

À l'heure actuelle, la seule valeur prise en charge est `$request.basepath`.

WebSocketrésumé de l'expression de sélection

Le tableau suivant récapitule les cas d'utilisation des expressions de sélection dans les WebSocket API :

Expression de sélection	Correspond à la clé pour	Remarqu	Exemple de cas d'utilisation
`Api.RouteSelectionExpression`	`Route.RouteKey`	`$defaul` est pris en charge en tant	WebSocket Acheminez les messages en

Expression de sélection	Correspond à la clé pour	Remarqu	Exemple de cas d'utilisation
		que route fourre-tout.	fonction du contexte de la demande d'un client.

Expression de sélection	Correspond à la clé pour	Remarqu	Exemple de cas d'utilisation
`Route.ModelSelectionExpression`	Clé pour `Route.RequestModels`	Facultatif. Si elle est fournie pour une intégration autre que de proxy, la validation du modèle a lieu. `$defaul` est pris en charge en tant que fourre-tout.	Effectuer une validation de demande de manière dynamique au sein de la même route.

Expression de sélection	Correspond à la clé pour	Remarqu	Exemple de cas d'utilisation
`Integration.TemplateSelectionExpression`	Clé pour `Integration.RequestTemplates`	Facultatif. Peut être fournie pour une intégration autre que de proxy pour manipuler des charges utiles entrantes. `${reque.body.jnPath}` et les valeurs statiques sont pris en charge.	Manipuler la demande de l'appelant en fonction des propriétés dynamiques de la demande.

Expression de sélection	Correspond à la clé pour	Remarqu	Exemple de cas d'utilisation
		`$defaul` est pris en charge en tant que fourre-tout.	

Expression de sélection	Correspond à la clé pour	Remarqu	Exemple de cas d'utilisation
`Integration.IntegrationResponseSelectionExpression`	`IntegrationResponse.IntegrationResponseKey`	Facultatif. Peut être fournie pour une intégration autre que de proxy. Agit en tant que modèle de correspondance pour des message d'erreur (à partir de Lambda) ou des codes d'état (à partir	Manipuler la réponse du serveur principal. Choisir l'action à exécuter en fonction de la réponse dynamique du serveur principal (par exemple, traitement distinctif de certaines erreurs).

Expression de sélection	Correspond à la clé pour	Remarqu	Exemple de cas d'utilisation
		d'intégrations HTTP). `$defaul` est obligatoire pour des intégrations autres que de proxy pour agir en tant que fourre-tout pour les réponses fructueuses.	

Expression de sélection	Correspond à la clé pour	Remarqu	Exemple de cas d'utilisation
`IntegrationResponse.TemplateSelectionExpression`	Clé pour `IntegrationResponse.ResponseTemplates`	Facultatif. Peut être fournie pour une intégration autre que de proxy. $default est pris en charge.	Dans certains cas, une propriété dynamique de la réponse peut imposer différentes transformations au sein de la même route et de l'intégration associée. `${request.body.jsonPath}` , `${integration.response.statuscode}` , `${integra`

Expression de sélection	Correspond à la clé pour	Remarqu	Exemple de cas d'utilisation
			`tion.response.header.headerName}` , `${integration.response.multivalueheader.headerName}` , et les valeurs statiques sont pris en charge. `$default` est pris en charge en tant que fourre-tout.

Expression de sélection	Correspond à la clé pour	Remarqu	Exemple de cas d'utilisation
`Route.RouteResponseSelectionExpression`	`RouteResponse.RouteResponseKey`	Doit être fourni pour initier une communication bidirectionnelle pour un WebSocl itinéraire. À l'heure actuelle, cette valeur est limitée à `$defaul` uniquement.	

Expression de sélection	Correspond à la clé pour	Remarqu	Exemple de cas d'utilisation
`RouteResponse.ModelSelectionExpression`	Clé pour `RouteResponse.RequestModels`	Non prise en charge actuellement.	

Développement d'une WebSocket API dans API Gateway

Cette section fournit des détails sur les fonctionnalités d'API Gateway dont vous avez besoin pendant le développement de vos API API Gateway.

Au fur et à mesure que vous développez votre API API Gateway, vous décidez d'un certain nombre de caractéristiques de votre API. Ces caractéristiques dépendent du cas d'utilisation de votre API. Par exemple, vous pourriez vouloir autoriser uniquement certains clients à appeler votre API ou qu'elle soit disponible pour tout le monde. Vous pouvez souhaiter utiliser un appel d'API pour exécuter une fonction Lambda, créer une requête de base de données ou appeler une application.

Rubriques

- Création d'une WebSocket API dans API Gateway
- Utilisation de routes pour les WebSocket API
- Contrôle et gestion de l'accès à une WebSocket API dans API Gateway
- Configuration des intégrations WebSocket d'API
- Validation des demandes
- Configuration des transformations de données pour les WebSocket API
- Utilisation de types de supports binaires pour les WebSocket API
- Invoquer une API WebSocket

Création d'une WebSocket API dans API Gateway

Vous pouvez créer une WebSocket API dans la console API Gateway, à l'aide de la commande AWS CLI create-api ou à l'aide de la `CreateApi` commande d'un AWS SDK. Les procédures suivantes montrent comment créer une nouvelle WebSocket API.

> **Note**
>
> WebSocket Les API ne prennent en charge que le protocole TLS 1.2. Les versions antérieures de TLS ne sont pas prises en charge.

Création d'une WebSocket API à l'aide de AWS CLI commandes

La création d'une WebSocket API à l'aide de la commande create-api AWS CLI nécessite l'appel de la commande create-api, comme indiqué dans l'exemple suivant, qui crée une API avec l'expression de sélection d'`$request.body.action`itinéraire :

```
aws apigatewayv2 --region us-east-1 create-api --name "myWebSocketApi3" --protocol-type
 WEBSOCKET --route-selection-expression '$request.body.action'
```

Exemple de sortie :

```
{
    "ApiKeySelectionExpression": "$request.header.x-api-key",
    "Name": "myWebSocketApi3",
    "CreatedDate": "2018-11-15T06:23:51Z",
    "ProtocolType": "WEBSOCKET",
    "RouteSelectionExpression": "'$request.body.action'",
    "ApiId": "aabbccddee"
}
```

Création d'une WebSocket API à l'aide de la console API Gateway

Vous pouvez créer une WebSocket API dans la console en choisissant le WebSocket protocole et en lui attribuant un nom.

> **Important**
>
> Une fois l'API créée, vous ne pouvez pas modifier le protocole que vous avez choisi. Il n'existe aucun moyen de convertir une WebSocket API en API REST ou vice versa.

Pour créer une WebSocket API à l'aide de la console API Gateway

1. Connectez-vous à la console API Gateway et choisissez Create API (Créer une API).
2. Sous WebSocket API, choisissez Build. Seuls les points de terminaison régionaux sont pris en charge.
3. Pour Nom de l'API, saisissez le nom de votre API.
4. Pour Expression de sélection de routage, saisissez une valeur. Par exemple, `$request.body.action`.

   Pour plus d'informations sur les expressions de sélection de la route, consultez la section the section called "".
5. Effectuez l'une des actions suivantes :
   - Choisissez Créer une API vide pour créer une API sans routes.
   - Choisissez Suivant pour joindre des routes à votre API.

   Vous pouvez joindre des routes après avoir créé votre API.

Utilisation de routes pour les WebSocket API

Dans votre WebSocket API, les messages JSON entrants sont dirigés vers les intégrations du backend en fonction des routes que vous configurez. (Les messages non JSON sont dirigés vers une route `$default` que vous configurez.)

Une route comprend une clé de routage, qui correspond à la valeur attendue une fois qu'une expression de sélection de la route est évaluée. L'attribut `routeSelectionExpression` est défini au niveau de l'API. Il spécifie une propriété JSON attendue dans la charge utile du message. Pour plus d'informations sur les expressions de sélection de la route, consultez la section the section called "".

Par exemple, si vos messages JSON contiennent une propriété `action` et que vous souhaitez effectuer différentes actions en fonction de cette propriété, votre expression de sélection de la

route peut être `${request.body.action}`. Votre table de routage spécifie l'action à exécuter en mettant en correspondance la valeur de la propriété `action` avec les valeurs des clés de routage personnalisées que vous avez définies dans la table.

Trois routes prédéfinies peuvent être utilisées : `$connect`, `$disconnect` et `$default`. De plus, vous pouvez créer des routes personnalisées.

- API Gateway appelle la `$connect` route lorsqu'une connexion persistante entre le client et une WebSocket API est initiée.
- API Gateway appelle la route `$disconnect` lorsque le client ou le serveur se déconnecte de l'API.
- API Gateway appelle une route personnalisée après évaluation de l'expression de sélection de la route par rapport au message si une route correspondante est trouvée ; la correspondance détermine l'intégration appelée.
- API Gateway appelle la route `$default` si l'expression de sélection de la route ne peut pas être évaluée par rapport au message ou si aucune route correspondante n'est trouvée.

Expressions de sélection de la route

Une expression de sélection de la route est évaluée lorsque le service sélectionne la route que doit suivre un message entrant. Le service utilise la route dont la clé `routeKey` correspond exactement à la valeur évaluée. Si aucune clé ne correspond et qu'une route avec la clé `$default` existe, celle-ci est sélectionnée. Si aucune route ne correspond à la valeur évaluée et qu'il n'y a pas de route `$default`, le service renvoie une erreur. Pour les API WebSocket basées, l'expression doit être de la forme`$request.body.`*`{path_to_body_element}`*.

Par exemple, supposons que vous envoyiez le message JSON suivant :

```
{
    "service" : "chat",
    "action" : "join",
    "data" : {
        "room" : "room1234"
   }
}
```

Vous pouvez sélectionner le comportement de votre API en fonction de la propriété `action`. Dans ce cas, vous pouvez définir l'expression de sélection de la route suivante :

```
$request.body.action
```

Dans cet exemple, `request.body` fait référence à la charge utile JSON du message, et `.action` est une expression JSONPath. Vous pouvez utiliser n'importe quelle expression de chemin JSON après `request.body`, mais n'oubliez pas que le résultat sera obtenu à l'aide de stringify. Par exemple, si votre expression JSONPath renvoie un ensemble de deux éléments, ceux-ci seront présentés sous forme de chaîne `"[item1, item2]"`. C'est la raison pour laquelle il est recommandé d'utiliser une expression qui correspond à une valeur plutôt qu'à un tableau ou un objet.

Vous pouvez utiliser simplement une valeur statique ou plusieurs variables. Le tableau suivant montre des exemples et leurs résultats évalués par rapport à la charge utile précédente.

Expression	Résultat évalué	Description
`$request.body.action`	`join`	Variable désencapsulée
`${request.body.action}`	`join`	Variable encapsulée
`${request.body.service}/${request.body.action}`	`chat/join`	Plusieurs variables avec des valeurs statiques
`${request.body.action}-${request.body.invalidPath}`	`join-`	Si JSONPath n'est pas trouvé, la variable est

Expression	Résultat évalué	Description
		résolue en tant que « ».
`action`	`action`	Valeur statique
`\$default`	`$default`	Valeur statique

Le résultat évalué est utilisé pour trouver une route. S'il existe une route avec une clé de routage correspondante, elle est sélectionnée pour traiter le message. Si aucune route correspondante n'existe, API Gateway tente de trouver la route `$default`, si elle est disponible. Si la route `$default` n'est pas définie, API Gateway renvoie une erreur.

Configuration de routes pour une WebSocket API dans API Gateway

Lorsque vous créez une nouvelle WebSocket API pour la première fois, il existe trois itinéraires prédéfinis : `$connect``$disconnect`, et`$default`. Vous pouvez les créer à l'aide de la console, de l'API ou AWS CLI. Si vous le souhaitez, vous pouvez créer des routes personnalisées. Pour de plus amples informations, veuillez consulter the section called “À propos WebSocket des API”.

> **Note**
>
> Dans l'interface de ligne de commande, vous pouvez créer des routes avant ou après avoir créé des intégrations, et réutiliser la même intégration pour plusieurs routes.

Création d'une route à l'aide de la console API Gateway

Pour créer une route à l'aide de la console API Gateway

1. Connectez-vous à la console API Gateway et choisissez l'API, puis choisissez Routes.
2. Choisissez Créer une route.

3. Pour Clé de route, entrez le nom de la clé de route. Vous pouvez créer les routes prédéfinies (`$connect`, `$disconnect` et `$default`) ou une route personnalisée.

   > **Note**
   >
   > Lorsque vous créez une route personnalisée, n'utilisez pas le préfixe $ dans le nom de la clé de routage. Ce préfixe est réservé aux routes prédéfinies.

4. Sélectionnez et configurez le type d'intégration pour la route. Pour plus d'informations, consultez the section called “Configuration d'une demande d'intégration d' WebSocket API à l'aide de la console API Gateway”.

Créez un itinéraire à l'aide du AWS CLI

Pour créer un itinéraire à l'aide du AWS CLI, appelez `create-route`comme indiqué dans l'exemple suivant :

```
aws apigatewayv2 --region us-east-1 create-route --api-id aabbccddee --route-key
 $default
```

Exemple de sortie :

```
{
    "ApiKeyRequired": false,
    "AuthorizationType": "NONE",
    "RouteKey": "$default",
    "RouteId": "1122334"
}
```

Spécification des paramètres de la demande de route pour **`$connect`**

Lorsque vous configurez la route `$connect` pour votre API, les paramètres facultatifs suivants sont disponibles pour activer les autorisations pour votre API. Pour de plus amples informations, veuillez consulter the section called “Route `$connect`”.

- Authorization (Autorisation) : Si aucune autorisation n'est nécessaire, vous pouvez spécifier `NONE`. Sinon, vous pouvez spécifier :
  - `AWS_IAM`pour utiliser des politiques AWS IAM standard afin de contrôler l'accès à votre API.

- `CUSTOM` pour implémenter l'autorisation pour une API en spécifiant une fonction de mécanisme d'autorisation Lambda que vous avez créée précédemment. L'autorisateur peut résider dans votre propre AWS compte ou sur un autre AWS compte. Pour de plus amples informations sur les mécanismes d'autorisation Lambda, veuillez consulter Utilisation des mécanismes d'autorisation Lambda API Gateway.

  > **Note**
  >
  > Dans la console API Gateway, le paramètre `CUSTOM` est visible uniquement une fois que vous avez configuré une fonction de mécanisme d'autorisation, comme décrit dans the section called “Configuration d'un autorisateur Lambda (console)”.

  > **Important**
  >
  > Le paramètre Authorization (Autorisation) est appliqué à l'ensemble de l'API, et pas uniquement à la route `$connect`. La route `$connect` protège les autres routes, car elle est appelée sur chaque connexion.

- API Key Required (Clé API requise) : Si vous le souhaitez, vous pouvez exiger une clé API pour une route `$connect` de l'API. Vous pouvez utiliser des clés API avec des plans d'utilisation pour contrôler et suivre l'accès à vos API. Pour de plus amples informations, veuillez consulter the section called “Plans d'utilisation”.

Configuration de la demande de route **`$connect`** à l'aide de la console API Gateway

Pour configurer la demande de `$connect` route pour une WebSocket API à l'aide de la console API Gateway :

1. Connectez-vous à la console API Gateway et choisissez l'API, puis choisissez Routes.
2. Sous Routes, choisissez `$connect` ou créez une route `$connect` en suivant the section called “Création d'une route à l'aide de la console API Gateway”.
3. Dans la section Paramètres de la demande de route, choisissez Modifier.
4. Pour Autorisation, sélectionnez un type d'autorisation.
5. Pour demander une API pour la route `$connect`, sélectionnez Exiger une clé d'API.
6. Sélectionnez Enregistrer les modifications.

Configuration des réponses d'itinéraire pour une WebSocket API dans API Gateway

WebSocket les itinéraires peuvent être configurés pour une communication bidirectionnelle ou unidirectionnelle. API Gateway ne transmet pas la réponse du backend à la réponse de routage, sauf si vous avez configuré une réponse de routage.

> **Note**
>
> Vous pouvez uniquement définir la réponse de l'`$default`itinéraire pour les WebSocket API. Vous pouvez utiliser une réponse d'intégration pour manipuler la réponse d'un service backend. Pour plus d’informations, consultez the section called “Présentation des réponses d'intégration”.

Vous pouvez configurer les réponses d'itinéraire et les expressions de sélection des réponses à l'aide de la console API Gateway AWS CLI ou d'un AWS SDK.

Pour plus d'informations sur les expressions de sélection de la réponse de routage, consultez la section the section called “”.

Rubriques

- Configuration d'une réponse de routage à l'aide de la console API Gateway
- Configurez une réponse d'itinéraire à l'aide du AWS CLI

Configuration d'une réponse de routage à l'aide de la console API Gateway

Après avoir créé une WebSocket API et attaché une fonction Lambda proxy à la route par défaut, vous pouvez configurer la réponse de route à l'aide de la console API Gateway :

1. Connectez-vous à la console API Gateway, choisissez une WebSocket API intégrant une fonction Lambda proxy sur la `$default` route.
2. Sous Routes (Routes), choisissez la route `$default`.
3. Choisissez Activer la communication bidirectionnelle.
4. Sélectionnez Deploy API (Déployer une API).
5. Déployez votre API vers une étape.

Utilisez la commande wscat suivante pour vous connecter à votre API. Pour plus d'informations sur `wscat`, consultez the section called "`wscat`À utiliser pour se connecter à une WebSocket API et y envoyer des messages".

```
wscat -c wss://api-id.execute-api.us-east-2.amazonaws.com/test
```

Appuyez sur la touche Entrée pour appeler la route par défaut. Le corps de votre fonction Lambda vous est renvoyé.

Configurez une réponse d'itinéraire à l'aide du AWS CLI

Pour configurer une réponse de route pour une WebSocket API à l'aide de AWS CLI, appelez la `create-route-response`commande comme indiqué dans l'exemple suivant. Vous pouvez identifier l'ID d'API et l'ID de routage en appelant `get-apis` et `get-routes`.

```
aws apigatewayv2 create-route-response \
    --api-id aabbccddee \
    --route-id 1122334  \
    --route-response-key '$default'
```

Exemple de sortie :

```
{
    "RouteResponseId": "abcdef",
    "RouteResponseKey": "$default"
}
```

Configuration d'un **`$connect`** itinéraire qui nécessite un WebSocket sous-protocole

Les clients peuvent utiliser `Sec-WebSocket-Protocol` ce champ pour demander un WebSocket sous-protocole lors de la connexion à votre WebSocket API. Vous pouvez configurer une intégration pour la route `$connect` afin d'autoriser les connexions uniquement si un client demande un sous-protocole pris en charge par votre API.

L'exemple de fonction Lambda suivant renvoie l'en-tête `Sec-WebSocket-Protocol` aux clients. La fonction établit une connexion à votre API uniquement si le client spécifie le sous-protocole `myprotocol`.

Pour un AWS CloudFormation modèle qui crée cet exemple d'intégration d'API et de proxy Lambda, consultez. `ws-subprotocol.yaml`

```
export const handler = async (event) => {
    if (event.headers != undefined) {
        const headers = toLowerCaseProperties(event.headers);

        if (headers['sec-websocket-protocol'] != undefined) {
            const subprotocolHeader = headers['sec-websocket-protocol'];
            const subprotocols = subprotocolHeader.split(',');

            if (subprotocols.indexOf('myprotocol') >= 0) {
                const response = {
                    statusCode: 200,
                    headers: {
                        "Sec-WebSocket-Protocol" : "myprotocol"
                    }
                };
                return response;
            }
        }
    }

    const response = {
        statusCode: 400
    };

    return response;
};

function toLowerCaseProperties(obj) {
    var wrapper = {};
    for (var key in obj) {
        wrapper[key.toLowerCase()] = obj[key];
    }
    return wrapper;
}
```

Vous pouvez utiliser `wscat` pour vérifier que votre API autorise les connexions uniquement si un client demande un sous-protocole pris en charge par votre API. Les commandes suivantes utilisent l'indicateur `-s` pour spécifier des sous-protocoles pendant la connexion.

La commande suivante tente une connexion avec un sous-protocole non pris en charge. Étant donné que le client a spécifié le sous-protocole `chat1`, l'intégration Lambda renvoie une erreur 400 et la connexion échoue.

```
wscat -c wss://api-id.execute-api.region.amazonaws.com/beta -s chat1
error: Unexpected server response: 400
```

La commande suivante inclut un sous-protocole pris en charge dans la demande de connexion. L'intégration Lambda permet la connexion.

```
wscat -c wss://api-id.execute-api.region.amazonaws.com/beta -s chat1,myprotocol
connected (press CTRL+C to quit)
```

Pour en savoir plus sur l'appel d' WebSocket API, consultezInvoquer une API WebSocket .

Contrôle et gestion de l'accès à une WebSocket API dans API Gateway

API Gateway prend en charge plusieurs mécanismes de contrôle et de gestion de l'accès à votre WebSocket API.

Les mécanismes suivants peuvent être utilisés pour l'authentification et l'autorisation :

- Les rôles et politiques AWS IAM standard offrent des contrôles d'accès flexibles et robustes. Vous pouvez utiliser les rôles et les stratégies IAM pour contrôler qui peut créer et gérer vos API, ainsi que qui peut les appeler. Pour de plus amples informations, veuillez consulter Utilisation de l'autorisation IAM.
- Les balises IAM peuvent être utilisées avec des stratégies IAM pour contrôler l'accès. Pour de plus amples informations, veuillez consulter Utilisation de balises pour contrôler l'accès aux ressources API REST API Gateway.
- Les mécanismes d'autorisation Lambda sont des fonctions Lambda qui contrôlent l'accès aux API. Pour de plus amples informations, veuillez consulter Création d'une fonction de mécanisme d'autorisation Lambda `REQUEST`.

Rubriques

- Utilisation de l'autorisation IAM
- Création d'une fonction de mécanisme d'autorisation Lambda REQUEST

Utilisation de l'autorisation IAM

L'autorisation IAM dans WebSocket les API est similaire à celle des API REST, avec les exceptions suivantes :

- L'action `execute-api` prend en charge `ManageConnections` en plus des actions existantes (`Invoke`, `InvalidateCache`). `ManageConnections` contrôle l'accès à l'API @connections.
- WebSocket les routes utilisent un format d'ARN différent :

```
arn:aws:execute-api:region:account-id:api-id/stage-name/route-key
```

- L'API `@connections` utilise le même format ARN que les API REST :

```
arn:aws:execute-api:region:account-id:api-id/stage-name/POST/@connections
```

> **Important**
>
> Lorsque vous utilisez Autorisation IAM, vous devez signer les requêtes avec Signature Version 4 (SigV4).

Par exemple, vous pouvez configurer la stratégie suivante pour le client. Cet exemple autorise tout le monde à envoyer un message (`Invoke`) pour toutes les routes, sauf pour une route secrète de l'étape `prod`, et empêche tout le monde de renvoyer un message aux clients connectés (`ManageConnections`) pour toutes les étapes.

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

Création d'une fonction de mécanisme d'autorisation Lambda **REQUEST**

Une fonction d'autorisation Lambda dans les WebSocket API est similaire à celle des API REST, avec les exceptions suivantes :

- Vous pouvez uniquement utiliser une fonction de mécanisme d'autorisation Lambda pour la route `$connect`.
- Vous ne pouvez pas utiliser de variables de chemin (`event.pathParameters`), car le chemin d'accès est fixe.
- `event.methodArn` est différent de son équivalent de l'API REST, car il ne possède pas de méthode HTTP. Dans le cas de `$connect`, `methodArn` se termine par `"$connect"` :

  ```
  arn:aws:execute-api:region:account-id:api-id/stage-name/$connect
  ```

- Les variables de contexte dans `event.requestContext` sont différentes de celles des API REST.

L'exemple suivant montre une entrée dans un `REQUEST` autorisateur pour une WebSocket API :

```
{
```

```
    "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"
    }
}
```

L'exemple de fonction d'autorisation Lambda suivant est une WebSocket version de la fonction d'autorisation Lambda pour les API REST dans : the section called “Exemples supplémentaires de fonctions d'autorisation Lambda”

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)
```

Pour configurer la fonction Lambda précédente en tant que fonction d'`REQUEST`autorisation pour une WebSocket API, suivez la même procédure que pour les API REST.

Pour configurer la route `$connect` afin qu'elle utilise ce mécanisme d'autorisation Lambda dans la console, sélectionnez ou créez la route `$connect`. Dans la section Paramètres de la demande de route, choisissez Modifier. Sélectionnez votre mécanisme d'autorisation dans le menu déroulant Autorisation, puis choisissez Enregistrer les modifications.

Pour tester le mécanisme d'autorisation, vous devez créer une nouvelle connexion. La modification du mécanisme d'autorisation dans `$connect` n'affecte pas le client déjà connecté. Lorsque vous vous connectez à votre WebSocket API, vous devez fournir des valeurs pour toutes les sources d'identité configurées. Par exemple, vous pouvez vous connecter en envoyant une chaîne de requête et un en-tête en utilisant `wscat` comme dans l'exemple suivant :

```
wscat -c 'wss://myapi.execute-api.us-east-1.amazonaws.com/beta?
QueryString1=queryValue1' -H HeaderAuth1:headerValue1
```

Si vous tentez de vous connecter sans valeur d'identité valide, vous recevrez une réponse `401` :

```
wscat -c wss://myapi.execute-api.us-east-1.amazonaws.com/beta
error: Unexpected server response: 401
```

Configuration des intégrations WebSocket d'API

Après avoir configuré une route d'API, vous devez l'intégrer à un point de terminaison dans le serveur principal. Un point de terminaison principal est également appelé point de terminaison d'intégration et peut être une fonction Lambda, un point de terminaison HTTP ou AWS une action de service. L'intégration de l'API a une demande d'intégration et une réponse d'intégration.

Dans cette section, vous découvrirez comment configurer les demandes d'intégration et les réponses d'intégration pour votre WebSocket API.

Rubriques

- Configuration d'une demande d'intégration d' WebSocket API dans API Gateway
- Configuration d'une réponse d'intégration d' WebSocket API dans API Gateway

Configuration d'une demande d'intégration d' WebSocket API dans API Gateway

La configuration d'une demande d'intégration comprend les opérations suivantes :

- La sélection d'une clé de routage à intégrer au serveur principal
- Spécifier le point de terminaison du backend à appeler. WebSocket Les API prennent en charge les types d'intégration suivants :
  - `AWS_PROXY`
  - `AWS`
  - `HTTP_PROXY`
  - `HTTP`
  - `MOCK`

  Pour plus d'informations sur les types d'intégration, consultez IntegrationTypel'API REST API Gateway V2.
- La configuration de la transformation des données de la demande de routage, si nécessaire, en données de demande d'intégration en spécifiant un ou plusieurs modèles de demande

Configuration d'une demande d'intégration d' WebSocket API à l'aide de la console API Gateway

Pour ajouter une demande d'intégration à un itinéraire dans une WebSocket API à l'aide de la console API Gateway

1. Connectez-vous à la console API Gateway et choisissez l'API, puis choisissez Routes.
2. Sous Routes, choisissez la route.
3. Choisissez l'onglet Demande d'intégration, puis dans la section Paramètres de la demande d'intégration, choisissez Modifier.
4. Pour Type d'intégration, sélectionnez l'une des valeurs suivantes :

   - Choisissez la fonction Lambda uniquement si votre API doit être intégrée à une AWS Lambda fonction que vous avez déjà créée dans ce compte ou dans un autre compte.

     Pour créer une nouvelle fonction Lambda dans AWS Lambda, pour définir une autorisation de ressource sur la fonction Lambda ou pour effectuer toute autre action de service Lambda, choisissez plutôt Service.AWS
   - Sélectionnez HTTP si votre API doit être intégrée à un point de terminaison HTTP existant. Pour de plus amples informations, veuillez consulter Configuration des intégrations HTTP dans API Gateway.
   - Choisissez Mock si vous souhaitez générer des réponses d'API directement depuis API Gateway, sans recourir à un backend d'intégration. Pour plus d’informations, consultez Configuration des intégrations fictives dans API Gateway.
   - Choisissez AWS un service si votre API doit être intégrée à un AWS service.
   - Choisissez Lien VPC si votre API doit utiliser un `VpcLink` en tant que point de terminaison d'intégration privé. Pour de plus amples informations, veuillez consulter Configuration des intégrations privées API Gateway.
5. Si vous avez sélectionné Fonction Lambda, procédez comme suit :

   a. Pour Utiliser une intégration proxy Lambda, cochez la case si vous avez l'intention d'utiliser Intégration proxy Lambda ou Intégration proxy Lambda entre comptes.

   b. Pour Fonction Lambda, spécifiez la fonction de l'une des manières suivantes :

      - Si votre Fonction Lambda se trouve dans le même compte, entrez le nom de la fonction, puis sélectionnez la fonction dans la liste déroulante.

> **Note**
>
> Le nom de la fonction peut éventuellement inclure son alias ou sa spécification de version, comme dans `HelloWorld`, `HelloWorld:1` ou `HelloWorld:alpha`.

- Si la fonction se trouve dans un autre compte, tapez l'ARN de la fonction.

c. Pour utiliser la valeur de délai d'expiration par défaut de 29 secondes, gardez Délai d'expiration activé. Pour définir un délai d'expiration personnalisé, choisissez Délai d'expiration et entrez une valeur de délai d'expiration comprise entre `50` et `29000` millisecondes.

6. Si vous avez sélectionné HTTP, suivez les instructions de l'étape 4 de la section the section called “ Configuration d'une demande d'intégration à l'aide de la console”.
7. Si vous avez sélectionné Mock (Fictif), passez à l'étape Request Templates (Modèles de demande).
8. Si vous avez choisi Service AWS , suivez les instructions de l'étape 6 de the section called “ Configuration d'une demande d'intégration à l'aide de la console”.
9. Si vous avez sélectionné Lien VPC, procédez comme suit :

   a. Pour Intégration proxy au VPC, sélectionnez la case à cocher si vous souhaitez que vos demandes soient traitées par proxy vers le point de terminaison de votre `VPCLink`.

   b. Dans le champ HTTP method, sélectionnez le type de méthode HTTP qui correspond le mieux au service backend HTTP.

   c. Dans la liste déroulante Lien VPC, sélectionnez un lien VPC. Vous pouvez sélectionner `[Use Stage Variables]` et entrer **`${stageVariables.vpcLinkId}`** dans la zone de texte en dessous de la liste.

      Vous pouvez définir la variable d'étape `vpcLinkId` après le déploiement de l'API à une étape et définir sa valeur sur l'ID du `VpcLink`.

   d. Pour URL du point de terminaison, entrez l'URL du back-end HTTP que cette intégration doit utiliser.

   e. Pour utiliser la valeur de délai d'expiration par défaut de 29 secondes, gardez Délai d'expiration activé. Pour définir un délai d'expiration personnalisé, choisissez Délai d'expiration et entrez une valeur de délai d'expiration comprise entre `50` et `29000` millisecondes.

10. Sélectionnez Enregistrer les modifications.

11. Sous Modèles de demande, procédez comme suit :

    a. Pour entrer une Expression de sélection de modèle, sous Modèles de demande, choisissez Modifier.

    b. Entrez une Expression de sélection de modèle. Utilisez une expression qu'API Gateway recherche dans la charge utile du message. S'il la trouve, il l'évalue, et le résultat est une valeur de clé de modèle qui est utilisée pour sélectionner le modèle de mappage de données à appliquer aux données dans la charge utile du message. Vous créez le modèle de mappage de données à la prochaine étape. Choisissez Modifier pour enregistrer vos modifications.

    c. Choisissez Créer un modèle pour créer le modèle de mappage de données. Pour Clé de modèle, entrez une valeur de clé de modèle qui est utilisée pour sélectionner le modèle de mappage de données à appliquer aux données dans la charge utile du message. Entrez ensuite un modèle de mappage. Sélectionnez Create template (Créer un modèle).

       Pour plus d'informations sur les expressions de sélection du modèle, consultez la section the section called “Expressions de sélection du modèle”.

Configurez une demande d'intégration à l'aide du AWS CLI

Vous pouvez configurer une demande d'intégration pour un itinéraire dans une WebSocket API en utilisant l' AWS CLI exemple suivant, qui crée une intégration fictive :

1. Créez un fichier nommé `integration-params.json`, avec le contenu suivant :

```
{"PassthroughBehavior": "WHEN_NO_MATCH", "TimeoutInMillis": 29000,
 "ConnectionType": "INTERNET", "RequestTemplates": {"application/json":
 "{\"statusCode\":200}"}, "IntegrationType": "MOCK"}
```

2. Exécutez la commande create-integration comme indiqué dans l'exemple suivant :

```
aws apigatewayv2 --region us-east-1 create-integration --api-id aabbccddee --cli-
input-json file://integration-params.json
```

Cet exemple renvoie l'exemple de sortie suivant :

```
{
    "PassthroughBehavior": "WHEN_NO_MATCH",
```

```
    "TimeoutInMillis": 29000,
    "ConnectionType": "INTERNET",
    "IntegrationResponseSelectionExpression": "${response.statuscode}",
    "RequestTemplates": {
        "application/json": "{\"statusCode\":200}"
    },
    "IntegrationId": "0abcdef",
    "IntegrationType": "MOCK"
}
```

Vous pouvez également configurer une demande d'intégration pour une intégration par proxy AWS CLI en utilisant l'exemple suivant :

1. Créez une fonction Lambda dans la console Lambda et attribuez-lui un rôle d'exécution Lambda de base.
2. Exécutez la commande create-integration comme dans l'exemple suivant :

```
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
```

Cet exemple renvoie l'exemple de sortie suivant :

```
{
    "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"
}
```

Format d'entrée d'une fonction Lambda pour l'intégration de proxy pour les API WebSocket

Avec l'intégration de proxy Lambda, API Gateway mappe l'intégralité de la demande du client avec le paramètre `event` en entrée de la fonction Lambda du backend. L'exemple suivant montre

la structure de l'événement d'entrée de la `$connect` route et de l'événement d'entrée de la `$disconnect` route qu'API Gateway envoie à une intégration de proxy Lambda.

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
  }
```

Configuration d'une réponse d'intégration d' WebSocket API dans API Gateway

Rubriques

- Présentation des réponses d'intégration

Présentation des réponses d'intégration

La réponse d'intégration d'API Gateway permet de modéliser et de manipuler la réponse reçue d'un service backend. Il existe certaines différences entre la configuration d'une API REST et une réponse d'intégration d' WebSocket API, mais le comportement est conceptuellement le même.

WebSocket les itinéraires peuvent être configurés pour une communication bidirectionnelle ou unidirectionnelle.

- Lorsqu'une route est configurée pour la communication bidirectionnelle, une réponse d'intégration vous permet de configurer des transformations sur la charge utile du message retourné, similaire aux réponses d'intégration des API REST.
- Si un itinéraire est configuré pour une communication unidirectionnelle, quelle que soit la configuration de réponse d'intégration, aucune réponse ne sera renvoyée sur le WebSocket canal une fois le message traité.

API Gateway ne transmet pas la réponse du backend à la réponse de routage, sauf si vous avez configuré une réponse de routage. Pour en savoir plus sur la configuration d'une réponse de routage, consultez the section called “Configurer les réponses WebSocket d'itinéraire de l'API”.

Réponses d'intégration pour une communication bidirectionnelle

Les intégrations peuvent être divisées en intégrations de proxy et autres que de proxy.

> **Important**
>
> Pour les intégrations de proxy, API Gateway transmet automatiquement la sortie du backend à l'appelant en tant que charge utile complète. Il n'y a pas de réponse d'intégration.

Pour les intégrations autres que de proxy, vous devez configurer au moins une réponse d'intégration :

- Idéalement, une de vos réponses d'intégration doit servir de fourre-tout lorsqu'aucun choix explicite ne peut être effectuée. Ce cas par défaut est représenté en définissant une clé de réponse d'intégration `$default`.
- Dans tous les autres cas, la clé de réponse d'intégration fonctionne en tant qu'expression régulière. Elle doit suivre le format `"/expression/"`.

Pour les intégrations HTTP autres que de proxy :

- API Gateway tente de mettre en correspondance le code d'état HTTP de la réponse du backend. Dans ce cas, la clé de réponse d'intégration fonctionne en tant qu'expression régulière. Si aucune correspondance n'est trouvée, `$default` est choisi en tant que réponse d'intégration.
- L'expression de sélection du modèle décrite ci-dessus fonctionne de manière identique. Exemples :
  - `/2\d\d/` : reçoit et transforme les réponses fructueuses
  - `/4\d\d/` : reçoit et transforme les erreurs de demande incorrecte
  - `$default` : reçoit et transforme toutes les réponses inattendues

Pour en savoir plus sur les expressions de sélection de modèle, consultez the section called “Expressions de sélection du modèle”.

Configuration d'une réponse d'intégration à l'aide de la console API Gateway

Pour configurer une réponse d'intégration d'itinéraires pour une WebSocket API à l'aide de la console API Gateway :

1. Connectez-vous à la console API Gateway à l'adresse : https://console.aws.amazon.com/apigateway.
2. Choisissez votre WebSocket API et votre itinéraire.
3. Choisissez l'onglet Demande d'intégration, puis dans la section Paramètres de la réponse d'intégration, choisissez Créer une réponse d'intégration.
4. Pour Clé de réponse, saisissez une valeur qui sera trouvée dans la clé de réponse trouvée dans le message sortant après avoir évalué l’expression de sélection de réponse. Par exemple, vous pouvez entrer **`/4\d\d/`** pour recevoir et transformer les erreurs de mauvaise demande ou entrer **`$default`** pour recevoir et transformer toutes les réponses qui correspondent à l'expression de sélection du modèle.
5. Pour Expression de sélection du modèle, entrez une expression de sélection pour évaluer le message sortant.

6. Choisissez Créer une réponse.
7. Vous pouvez également définir un modèle de mappage pour configurer les transformations de votre charge utile de message renvoyé. Sélectionnez Create template (Créer un modèle).
8. Saisissez un nom de clé. Si vous choisissez l'expression de sélection de modèle par défaut, saisissez **`\$default`**.
9. Pour Modèle de réponse, entrez votre modèle de mappage dans l'éditeur de code.
10. Sélectionnez Create template (Créer un modèle).
11. Choisissez Déployer l'API pour déployer votre API.

Utilisez la commande wscat suivante pour vous connecter à votre API. Pour plus d'informations sur `wscat`, consultez the section called "`wscat`À utiliser pour se connecter à une WebSocket API et y envoyer des messages".

```
wscat -c wss://api-id.execute-api.us-east-2.amazonaws.com/test
```

Lorsque vous appelez votre route, la charge utile de message renvoyé est renvoyée.

Configurez une réponse d'intégration à l'aide du AWS CLI

Pour configurer une réponse d'intégration pour une WebSocket API à l'aide de la commande, AWS CLI appelez la create-integration-responsecommande. La commande d'interface de ligne de commande suivante montre un exemple de création d'une réponse d'intégration `$default` :

```
aws apigatewayv2 create-integration-response \
    --api-id vaz7da96z6 \
    --integration-id a1b2c3 \
    --integration-response-key '$default'
```

Validation des demandes

Vous pouvez configurer API Gateway afin qu'il exécute la validation d'une demande de routage avant de continuer avec la demande d'intégration. Si la validation échoue, API Gateway échoue à la demande sans appeler votre backend, envoie une réponse de passerelle « Bad request body » au client et publie les résultats de la validation dans les CloudWatch journaux. L'utilisation de la validation de cette façon réduit les appels inutiles vers votre back-end d'API.

Expressions de sélection du modèle

Vous pouvez utiliser une expression de sélection de modèle pour valider dynamiquement les demandes au sein d'une même route. La validation du modèle se produit si vous fournissez une expression de sélection de modèle pour les intégrations proxy ou non proxy. Vous devrez peut-être définir le modèle `$default` comme solution de secours si aucun modèle correspondant n'est trouvé. S'il n'y a pas de modèle correspondant et que `$default` n'est pas défini, la validation échoue. L'expression de sélection ressemble à `Route.ModelSelectionExpression` et évalue à la clé pour `Route.RequestModels`.

Lorsque vous définissez un itinéraire pour une WebSocket API, vous pouvez éventuellement spécifier une expression de sélection de modèle. Cette expression est évaluée pour sélectionner le modèle à utiliser pour la validation du corps lors de la réception d'une demande. L'expression correspond à l'une des entrées de l'élément d'une route `requestmodels`.

Un modèle est exprimé sous forme de schéma JSON et décrit la structure des données du corps de la demande. La nature de ces expressions de sélection vous permet de choisir de manière dynamique le modèle à utiliser pour la validation lors de l'exécution pour une route particulière. Pour plus d'informations sur la création d'un modèle, consultez la section the section called “Comprendre les modèles de données”.

Configuration la validation des demandes à l'aide de la console API Gateway

L'exemple suivant montre comment configurer la validation des demandes sur un itinéraire.

Tout d'abord, vous créez un modèle, puis vous créez un itinéraire. Ensuite, vous configurez la validation des demandes sur l'itinéraire que vous venez de créer. Enfin, vous déployez et testez votre API. Pour terminer ce didacticiel, vous avez besoin d'une WebSocket API `$request.body.action` servant d'expression de sélection d'itinéraire et d'un point de terminaison d'intégration pour votre nouvel itinéraire.

Vous avez également besoin de `wscat` pour vous connecter à votre API. Pour plus d’informations, consultez the section called “`wscat`À utiliser pour se connecter à une WebSocket API et y envoyer des messages”.

Pour créer un modèle

1. Connectez-vous à la console API Gateway à l'adresse : https://console.aws.amazon.com/apigateway.
2. Choisissez une WebSocket API.

3. Dans le volet de navigation principal, choisissez Modèles.
4. Sélectionnez Create model.
5. Pour Name (Nom), saisissez **`emailModel`**.
6. Pour Type de contenu, entrez **`application/json`**.
7. Pour Schéma du modèle, saisissez le modèle qui suit :

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "type" : "object",
    "required" : [ "address"],
    "properties" : {
        "address": {
            "type": "string"
        }
    }
}
```

   Ce modèle nécessite que la demande contienne une adresse e-mail.
8. Choisissez Enregistrer.

Au cours de cette étape, vous allez créer un itinéraire pour votre WebSocket API.

Pour créer une route

1. Dans le volet de navigation principal, choisissez Routes.
2. Choisissez Create Route (Créer un itinéraire).
3. Pour Route key (Clé de route), entrez **`sendMessage`**.
4. Choisissez un type d'intégration et spécifiez un point de terminaison d'intégration. Pour plus d’informations, consultez the section called “Intégrations”.
5. Choisissez Create Route (Créer un itinéraire).

Au cours de cette étape, vous configurez la validation des demandes pour l'`sendMessage`itinéraire.

Pour configurer la validation des demandes

1. Dans l'onglet Demande d'itinéraire, sous Paramètres de demande d'itinéraire, choisissez Modifier.

2. Pour Expression de sélection du modèle, entrez**`${request.body.messageType}`**.

   API Gateway utilise cette `messageType` propriété pour valider la demande entrante.
3. Choisissez Ajouter un modèle de demande.
4. Pour Clé du modèle, entrez**`email`**.
5. Pour Modèle, choisissez EmailModel.

   API Gateway valide les messages entrants avec la `messageType` propriété définie sur par `email` rapport à ce modèle.

   > **Note**
   >
   > Si API Gateway ne parvient pas à faire correspondre l'expression de sélection du modèle à une clé de modèle, elle sélectionne le `$default` modèle. S'il n'existe aucun `$default` modèle, la validation échoue. Pour les API de production, nous vous recommandons de créer un `$default` modèle.
6. Sélectionnez Enregistrer les modifications.

Au cours de cette étape, vous déployez et testez votre API.

Pour déployer et tester votre API

1. Sélectionnez Deploy API (Déployer une API).
2. Choisissez l'étape souhaitée dans la liste déroulante ou saisissez le nom d'une nouvelle étape.
3. Choisissez Deploy (Déployer).
4. Dans le volet de navigation principal, choisissez Étapes.
5. Copiez l' WebSocket URL de votre API. L'URL doit ressembler à `wss://abcdef123.execute-api.us-east-2.amazonaws.com/production`.
6. Ouvrez un nouveau terminal et exécutez la wscat commande avec les paramètres suivants.

   ```
   wscat -c wss://abcdef123.execute-api.us-west-2.amazonaws.com/production
   ```

   ```
   Connected (press CTRL+C to quit)
   ```
7. Utilisez la commande suivante pour tester votre API.

```
{"action": "sendMessage", "messageType": "email"}
```

```
{"message": "Invalid request body", "connectionId":"ABCD1=234",
 "requestId":"EFGH="}
```

API Gateway échouera à la demande.

Utilisez la commande suivante pour envoyer une demande valide à votre API.

```
{"action": "sendMessage", "messageType": "email", "address":
 "mary_major@example.com"}
```

Configuration des transformations de données pour les WebSocket API

Dans API Gateway, la demande de méthode d'une WebSocket API peut prendre une charge utile dans un format différent de la charge utile de la demande d'intégration correspondante, comme l'exige le backend. De même, le backend peut renvoyer une charge utile de réponse d'intégration différente de la charge utile de réponse de méthode attendue par le serveur frontal.

API Gateway vous permet d'utiliser des modèles de mappage pour mapper la charge utile d'une demande de méthode à la demande d'intégration correspondante, et d'une réponse d'intégration à la réponse de méthode correspondante. Vous spécifiez une expression de sélection de modèle pour déterminer le modèle à utiliser afin d'effectuer les transformations de données nécessaires.

Vous pouvez utiliser des mappages de données pour mapper des données d'une demande de routage à une intégration backend. Pour en savoir plus, consultez la section the section called “Mappage de données”.

Modèles et modèles de mappage

Un modèle de mappage est un script exprimé en langage VTL (Velocity Template Language) et appliqué à la charge utile à l'aide d'expressions JSONPath. Pour de plus amples informations sur les modèles de mappage API Gateway, veuillez consulter Présentation des modèles de mappage.

La charge utile peut comporter un modèle de données en fonction du schéma JSON version 4. Vous n'avez pas besoin de définir un modèle pour créer un modèle de mappage. Toutefois, un modèle peut vous aider à créer un modèle de mappage, car API Gateway génère un plan de modèle en

fonction du modèle fourni. Pour de plus amples informations sur les modèles API Gateway, veuillez consulter Comprendre les modèles de données.

Expressions de sélection du modèle

Pour transformer une charge utile à l'aide d'un modèle de mappage, vous devez spécifier une expression de sélection de modèle d' WebSocket API dans une demande d'intégration ou une réponse d'intégration. Cette expression est évaluée pour déterminer le modèle d'entrée ou de sortie (le cas échéant) à utiliser pour transformer le corps de la demande en corps de demande d'intégration (via un modèle d'entrée) ou le corps de la réponse au corps de la réponse de routage (via un modèle de sortie).

`Integration.TemplateSelectionExpression` prend en charge `${request.body.jsonPath}` et les valeurs statiques.

`IntegrationResponse.TemplateSelectionExpression` prend en charge `${request.body.jsonPath}`, `${integration.response.statuscode}`, `${integration.response.header.headerName}`, `${integration.response.multivalueheader.headerName}` et des valeurs statiques.

Expressions de sélection de la réponse d'intégration

Lorsque vous configurez une réponse d'intégration pour une WebSocket API, vous pouvez éventuellement spécifier une expression de sélection de réponse d'intégration. Cette expression détermine quelle `IntegrationResponse` doit être sélectionnée lorsqu'une intégration est renvoyée. La valeur de cette expression est actuellement restreinte par API Gateway, comme défini ci-dessous. Gardez à l'esprit que cette expression est uniquement pertinente pour des intégrations autres que de proxy. Une intégration de proxy transmet simplement la charge utile de la réponse à l'appelant sans modélisation ni modification.

Contrairement aux autres expressions de sélection précédentes, cette expression prend actuellement en charge un format de correspondance de modèle. L'expression doit être encapsulée avec des barres obliques.

Actuellement, la valeur est fixée en fonction de l'élément `integrationType`:

- Pour les intégrations basées sur Lambda, la valeur est `$integration.response.body.errorMessage`.
- Pour les intégrations `HTTP` et `MOCK`, la valeur est `$integration.response.statuscode`.

- Pour `HTTP_PROXY` et `AWS_PROXY`, l'expression n'est pas utilisée, car vous demandez que la charge utile soit transmise à l'appelant.

Configuration du mappage des données pour les WebSocket API

Le mappage de données vous permet de mapper des données d'une demande de routage vers une intégration backend.

> **Note**
>
> Le mappage des données pour WebSocket les API n'est pas pris en charge dans le AWS Management Console. Vous devez utiliser le AWS CLI AWS CloudFormation, ou un SDK pour configurer le mappage des données.

Rubriques

- Mappage des données de demande de routage à des paramètres de demande d'intégration
- Exemples

Mappage des données de demande de routage à des paramètres de demande d'intégration

Les paramètres de demande d'intégration peuvent être mappés à partir de n'importe quels paramètres de demande de routage défini, du corps de la demande, des variables `context` ou `stage`, ainsi que des valeurs statiques.

Dans le tableau suivant, *`PARAM_NAME`* est le nom d'un paramètre de demande de routage du type de paramètre donné. Il doit respecter le modèle d'expression régulière `'^[a-zA-Z0-9._$-]+$]'`. *`JSONPath_EXPRESSION`* est une expression JSONPath pour un champ JSON du corps de la demande.

Expressions de mappage de données de demande d'intégration

Source de données mappée	Expression de mappage
Chaîne de requête de demande (prise en charge uniquement pour le routage `$connect`)	`route.request.querystring.` *`PARAM_NAME`*

Source de données mappée	Expression de mappage
En-tête de demande (prise en charge uniquement pour le routage `$connect`)	`route.request.header.` *`PARAM_NAME`*
Chaîne de demande à plusieurs valeurs (prise en charge uniquement pour le routage `$connect`)	`route.request.multivaluequerystring.` *`PARAM_NAME`*
En-tête de demande à plusieurs valeurs (prise en charge uniquement pour le routage `$connect`)	`route.request.multivalueheader.` *`PARAM_NAME`*
Corps de la demande	`route.request.body.` *`JSONPath_EXPRESSION`*
Variables d'étape	`stageVariables.` *`VARIABLE_NAME`*
Variables de contexte	`context.`*`VARIABLE_NAME`* qui doit être l'une des variables de contexte prises en charge.
Valeur statique	*`'STATIC_VALUE'`* . *`STATIC_VALUE`* est une chaîne littérale qui doit être placée entre guillemets simples.

Exemples

Les AWS CLI exemples suivants configurent les mappages de données. Pour un exemple AWS CloudFormation de modèle, voir `websocket-data-mapping.yaml`.

Mapper ConnectionID d'un client à un en-tête dans une demande d'intégration

L'exemple de commande suivant mappe `connectionId` d'un client à un en-tête `connectionId` dans la demande à une intégration backend.

```
aws apigatewayv2 update-integration \
    --integration-id abc123 \
    --api-id a1b2c3d4 \
```

```
    --request-parameters
 'integration.request.header.connectionId'='context.connectionId'
```

Mapper un paramètre de chaîne de demande à un en-tête dans une demande d'intégration

Les exemples de commande suivants mappent un paramètre de chaîne de demande `authToken` à un en-tête `authToken` dans la demande d'intégration.

Tout d'abord, ajoutez le paramètre de chaîne de demande `authToken` aux paramètres de demande du routage.

```
aws apigatewayv2 update-route --route-id 0abcdef \
    --api-id a1b2c3d4 \
    --request-parameters '{"route.request.querystring.authToken": {"Required": false}}'
```

Ensuite, mappez le paramètre de chaîne de demande à l'en-tête `authToken` dans la demande à l'intégration backend.

```
aws apigatewayv2 update-integration \
    --integration-id abc123 \
    --api-id a1b2c3d4 \
    --request-parameters
 'integration.request.header.authToken'='route.request.querystring.authToken'
```

Si nécessaire, supprimez le paramètre de chaîne de requête `authToken` dans les paramètres de demande de l'itinéraire.

```
aws apigatewayv2 delete-route-request-parameter \
    --route-id 0abcdef \
    --api-id a1b2c3d4 \
    --request-parameter-key 'route.request.querystring.authToken'
```

Référence du modèle de mappage d' WebSocket API API Gateway

Cette section résume l'ensemble de variables actuellement prises en charge pour les WebSocket API dans API Gateway.

Paramètre	Description
`$context.connectionId`	ID unique pour la connexion qui peut être utilisé pour effectuer un rappel au client.
`$context.connectedAt`	Temps de connexion au format Epoch.
`$context.domainName`	Nom de domaine pour l' WebSocket API. Ce nom peut être utilisé pour effectuer un rappel au client (au lieu d'une valeur codée en dur).
`$context.eventType`	Type d'événement : `CONNECT`, `MESSAGE` ou `DISCONNECT`
`$context.messageId`	ID côté serveur unique pour un message. Uniquement disponible lorsque `$context.eventType` est défini sur `MESSAGE`.
`$context.routeKey`	Clé de routage sélectionnée.
`$context.requestId`	Identique à `$context.extendedRequestId` .
`$context.extendedRequestId`	Un ID généré automatiquement pour l'appel d'API, qui contient d'autres informations utiles pour le débogage et le dépannage.
`$context.apiId`	Identifiant qu'API Gateway attribue à votre API.
`$context.authorizer.principalId`	Identification de l'utilisateur principal associée au jeton envoyé par le client et retourné par une fonction Lambda du mécanisme d'autorisation Lambda API Gateway (anciennement appelé Custom Authorizer (mécanisme d'autorisation personnalisé)).
`$context.authorizer.` *`property`*	Valeur obtenue à l'aide de stringify de la paire clé-valeur spécifiée du mappage `context` renvoyé par une fonction du mécanisme

Paramètre	Description
	d'autorisation Lambda API Gateway. Par exemple, si le mécanisme d'autorisation retourne le mappage `context` suivant : `"context" : { "key": "value", "numKey": 1, "boolKey": true }` l'appel de `$context.authorizer.key` renvoie la chaîne `"value"`, l'appel de `$context.authorizer.numKey` renvoie la chaîne `"1"` et l'appel de `$context.authorizer.boolKey` renvoie la chaîne `"true"`.
`$context.error.messageString`	La valeur entre guillemets de `$context.error.message`, à savoir `"$context.error.message"`.
`$context.error.validationErrorString`	Chaîne contenant un message d'erreur de validation détaillé.
`$context.identity.accountId`	L'ID de AWS compte associé à la demande.
`$context.identity.apiKey`	Clé du propriétaire d'API associée à la demande d'API activée par clé.
`$context.identity.apiKeyId`	ID de clé du propriétaire d'API associée à la demande d'API activée par clé
`$context.identity.caller`	Identifiant principal de l'appelant effectuant la demande.

Paramètre	Description
`$context.identity.cognitoAuthenticationProvider`	Liste séparée par des virgules des fournisseurs d'authentification Amazon Cognito utilisés par l'appelant à l'origine de la demande. Disponible uniquement si la demande a été signée avec les informations d'identification Amazon Cognito. Par exemple, pour une identité provenant d'un pool d'utilisateurs Amazon Cognito, `cognito-idp.` *`region`*`.amazonaws.com/` *`user_pool_id`* `,cognito-idp.` *`region`*`.amazonaws.com/` *`user_pool_id`* `:CognitoSignIn:` *`token subject claim`* Pour de plus amples informations, veuillez consulter Utilisation des identités fédérées dans le Manuel du développeur Amazon Cognito.
`$context.identity.cognitoAuthenticationType`	Type d'authentification Amazon Cognito de l'appelant effectuant la demande. Disponible uniquement si la demande a été signée avec les informations d'identification Amazon Cognito. Les valeurs possibles incluent `authenticated` pour les identités authentifiées et `unauthenticated` pour les identités non authentifiées.
`$context.identity.cognitoIdentityId`	ID d'identité Amazon Cognito de l'appelant effectuant la demande. Disponible uniquement si la demande a été signée avec les informations d'identification Amazon Cognito.

Paramètre	Description
`$context.identity.cognitoId entityPoolId`	ID de groupe d'identités Amazon Cognito de l'appelant effectuant la demande. Disponibl e uniquement si la demande a été signée avec les informations d'identification Amazon Cognito.
`$context.identity.sourceIp`	L'adresse IP source de la connexion TCP envoyant la demande au point de terminaison de l'API Gateway.
`$context.identity.user`	Identifiant principal de l'utilisateur effectuant la demande.
`$context.identity.userAgent`	Agent utilisateur de l'appelant de l'API.
`$context.identity.userArn`	ARN (Amazon Resource Name) de l'utilisateur identifié après l'authentification.
`$context.requestTime`	Durée des demandes au format CLF (`dd/MMM/ yyyy:HH:mm:ss +-hhmm`).
`$context.requestTimeEpoch`	Heure de la demande au format Epoch, en millisecondes.
`$context.stage`	Étape de déploiement de l'appel d'API (par exemple, bêta ou production).
`$context.status`	Statut de la réponse.
`$input.body`	Renvoie la charge utile brute sous forme de chaîne.

Paramètre	Description
`$input.json(x)`	Cette fonction évalue une expression JSONPath et renvoie les résultats sous la forme une chaîne JSON. Par exemple, `$input.json('$.pets')` renvoie une chaîne JSON représentant la structure pets. Pour plus d'informations sur JSONPath, consultez JSONPath ou JSONPath pour Java.

Paramètre	Description
`$input.path(x)`	Prend une chaîne d'expression JSONPath (x) et renvoie une représentation d'objet JSON du résultat. Cela vous permet d'accéder aux éléments de la charge utile et de les manipuler en mode natif en langage VTL (Apache Velocity Template Language). Par exemple, si l'expression `$input.path('$.pets')` renvoie un objet comme suit : `[ { "id": 1, "type": "dog", "price": 249.99 }, { "id": 2, "type": "cat", "price": 124.99 }, { "id": 3, "type": "fish", "price": 0.99 } ]` `$input.path('$.pets').count()` renvoie `"3"`. Pour plus d'informations sur JSONPath, consultez JSONPath ou JSONPath pour Java.
`$stageVariables.` *`<variable_name>`*	*`<variable_name>`* représente un nom de variable d'étape.

Paramètre	Description
`$stageVariables['` *`<variable_name>`* `']`	*`<variable_name>`* représente n'importe quel nom de variable d'étape.
`${stageVariables['` *`<variable_name>`* `']}`	*`<variable_name>`* représente n'importe quel nom de variable d'étape.
`$util.escapeJavaScript()`	Échape les caractères d'une chaîne en utilisant les règles relatives aux JavaScript chaînes. **Note** Cette fonction convertit tout guillemet simple (`'`) en guillemet d'échappement (`\'`). Cependant, les guillemets simples d'échappement ne sont pas valides en JSON. Par conséquent, lorsque la sortie de cette fonction est utilisée dans une propriété JSON, vous devez reconvertir les guillemets simples d'échappement (`\'`) en guillemets simples (`'`), comme illustré dans l'exemple suivant : `$util.escapeJavaScript( data).replaceAll("\\'","'")`

Paramètre	Description

`$util.parseJson()`	Prend la chaîne JSON (obtenue à l'aide de stringify) et renvoie une représentation objet du résultat. Vous pouvez utiliser le résultat de cette fonction pour accéder aux éléments de la charge utile et les manipuler en mode natif en langage VTL (Apache Velocity Template Language). Par exemple, si vous avez la charge utile suivante :
	`{"errorMessage":"{\"key1\":\"var1\", \"key2\":{\"arr\":[1,2,3]}}"}`
	et utilisez le modèle de mappage suivant :
	`#set ($errorMessageObj = $util.parseJson($input.path('$.errorMessage')))`
	`{`
	`"errorMessageObjKey2ArrVal" : $errorMessageObj.key2.arr[0]`
	`}`
	vous obtenez la sortie suivante :
	`{`
	`"errorMessageObjKey2ArrVal" : 1`
	`}`

`$util.urlEncode()`	Convertit une chaîne au format « application/ x-www-form-urlencoded ».
`$util.urlDecode()`	Décode une chaîne « application/ x-www-form-urlencoded ».
`$util.base64Encode()`	Encode les données dans une chaîne encodée en base64.

Paramètre	Description
`$util.base64Decode()`	Décode les données d'une chaîne encodée en base64.

Utilisation de types de supports binaires pour les WebSocket API

Les API WebSocket API Gateway ne prennent actuellement pas en charge les trames binaires dans les charges utiles des messages entrants. Si une application client envoie une trame binaire, API Gateway la rejette et déconnecte le client avec le code 1003.

Il existe une solution pour contourner ce comportement. Si le client envoie des données binaires encodées au format texte (par exemple, base64) en tant que trame de texte, vous pouvez définir la propriété `contentHandlingStrategy` de l'intégration sur `CONVERT_TO_BINARY` pour convertir la charge utile de la chaîne encodée en base64 au format binaire.

Pour renvoyer une réponse de routage pour une charge utile binaire dans des intégrations autres que proxy, vous pouvez définir la propriété `contentHandlingStrategy` de la réponse d'intégration sur `CONVERT_TO_TEXT` pour convertir la charge utile du format binaire en chaîne encodée en base64.

Invoquer une API WebSocket

Une fois que vous avez déployé votre WebSocket API, les applications clientes peuvent s'y connecter et lui envoyer des messages, et votre service principal peut envoyer des messages aux applications clientes connectées :

- Vous pouvez l'utiliser `wscat` pour vous connecter à votre WebSocket API et lui envoyer des messages afin de simuler le comportement du client. Voir the section called “`wscat`À utiliser pour se connecter à une WebSocket API et y envoyer des messages”.
- Vous pouvez utiliser l'API @connections à partir de votre service backend pour envoyer un message de rappel à un client connecté, obtenir des informations de connexion ou déconnecter le client. Voir the section called “Utilisation des commandes `@connections` dans votre service backend”.
- Une application cliente peut utiliser sa propre WebSocket bibliothèque pour appeler votre WebSocket API.

`wscat`À utiliser pour se connecter à une WebSocket API et y envoyer des messages

Cet `wscat` utilitaire est un outil pratique pour tester une WebSocket API que vous avez créée et déployée dans API Gateway. Vous pouvez installer et utiliser `wscat` comme suit :

1. Téléchargez `wscat` depuis le site https://www.npmjs.com/package/wscat.
2. Installez `wscat` en exécutant les commandes suivantes.

   ```
   npm install -g wscat
   ```

3. Pour vous connecter à votre API, exécutez la commande `wscat` comme illustré dans l'exemple suivant. Notez que cet exemple part du principe que le paramètre `Authorization` est défini sur `NONE`.

   ```
   wscat -c wss://aabbccddee.execute-api.us-east-1.amazonaws.com/test/
   ```

   Vous devez remplacer *`aabbccddee`* par l'ID réel de l'API qui est affiché dans la console API Gateway ou renvoyé par la commande `create-api` de la AWS CLI .

   En outre, si votre API se trouve dans une région autre que `us-east-1`, vous devez remplacer la région par la région correcte.
4. Pour tester votre API, entrez un message tel que le suivant une fois connecté :

   ```
   {"{jsonpath-expression}":"{route-key}"}
   ```

   où *`{jsonpath d'expression}`* est une expression JSONPath et *`{route-key}`* est une clé de routage pour l'API. Exemples :

   ```
   {"action":"action1"}
   {"message":"test response body"}
   ```

   Pour plus d'informations sur JSONPath, consultez JSONPath ou JSONPath pour Java.
5. Pour vous déconnecter de votre API, entrez `ctrl-C`.

Utilisation des commandes **`@connections`** dans votre service backend

Votre service principal peut utiliser les requêtes HTTP de WebSocket connexion suivantes pour envoyer un message de rappel à un client connecté, obtenir des informations de connexion ou déconnecter le client.

> **Important**
>
> Ces demandes utilisent l'autorisation IAM. Vous devez donc les signer avec Signature Version 4 (SigV4). Pour ce faire, vous pouvez utiliser l'API de gestion API Gateway. Pour plus d'informations, consultez ApiGatewayManagementApi.

Dans la commande suivante, vous devez le *{api-id}* remplacer par l'identifiant d'API réel, qui est affiché dans la console API Gateway ou renvoyé par la commande AWS CLI create-api. Vous devez établir la connexion avant d'utiliser cette commande.

Pour envoyer un message de rappel au client, utilisez :

```
POST https://{api-id}.execute-api.us-east-1.amazonaws.com/{stage}/
@connections/{connection_id}
```

Vous pouvez tester cette demande en utilisant Postman ou en appelant awscurl comme dans l'exemple suivant :

```
awscurl --service execute-api -X POST -d "hello world" https://{prefix}.execute-api.us-
east-1.amazonaws.com/{stage}/@connections/{connection_id}
```

Vous avez besoin de coder la commande en URL comme dans l'exemple suivant :

```
awscurl --service execute-api -X POST -d "hello world" https://aabbccddee.execute-
api.us-east-1.amazonaws.com/prod/%40connections/R0oXAdfD0kwCH6w%3D
```

Pour obtenir le dernier état de connexion du client, utilisez :

```
GET https://{api-id}.execute-api.us-east-1.amazonaws.com/{stage}/
@connections/{connection_id}
```

Pour déconnecter le client, utilisez :

```
DELETE https://{api-id}.execute-api.us-east-1.amazonaws.com/{stage}/
@connections/{connection_id}
```

Vous pouvez générer dynamiquement une URL de rappel en utilisant les variables `$context` dans votre intégration. Par exemple, si vous utilisez l'intégration de proxy Lambda avec une fonction Lambda `Node.js`, vous pouvez générer l'URL et envoyer un message à un client connecté comme suit :

```
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,
  };
};
```

Lorsque vous envoyez un message de rappel, votre fonction Lambda doit être autorisée à appeler l’API de gestion d’API Gateway. Vous pouvez recevoir un message d’erreur contenant

`GoneException` si vous publiez un message avant l'établissement de la connexion ou après la déconnexion du client.

Publication d' WebSocket API que les clients peuvent invoquer

Le simple fait de créer et le développer une API API Gateway ne la rend pas automatiquement appelable par vos utilisateurs. Pour la rendre appelable, vous devez déployer votre API jusqu'à une étape. En outre, vous pouvez personnaliser l'URL que vos utilisateurs utiliseront pour accéder à votre API. Vous pouvez lui attribuer un domaine cohérent avec votre marque ou plus facilement mémorisable que l'URL par défaut de votre API.

Dans cette section, vous pouvez apprendre à déployer votre API et à personnaliser l'URL que vous fournissez aux utilisateurs pour y accéder.

> **Note**
>
> Pour renforcer la sécurité de vos API API Gateway, le domaine `execute-api.{region}.amazonaws.com` est enregistré dans la liste des suffixes publics (PSL). Pour plus de sécurité, nous vous recommandons d'utiliser des cookies avec un préfixe `__Host-` si vous devez définir des cookies sensibles dans le nom de domaine par défaut de vos API API Gateway. Cette pratique vous aidera à protéger votre domaine contre les tentatives de falsification de requêtes intersites (CSRF). Pour plus d'informations, consultez la page Set-Cookie du Mozilla Developer Network.

Rubriques

- Utilisation des étapes pour les WebSocket API
- Déployer une WebSocket API dans API Gateway
- Politique de sécurité pour les WebSocket API
- Configuration de noms de domaine personnalisés pour les WebSocket API

Utilisation des étapes pour les WebSocket API

Une étape d'API est une référence logique à un état du cycle de vie de votre API (par exemple, `dev`, `prod`, `beta` ou `v2`). Les étapes API sont identifiées par leur ID d'API et leur nom d'étape, et elles sont incluses dans l'URL que vous utilisez pour appeler l'API. Chaque étape est une référence

nommée à un déploiement de l'API et elle est mise à la disposition des applications clientes à appeler.

Un déploiement est un instantané de la configuration de votre API. Après que vous avez déployé une API sur une étape, les clients peuvent l'appeler. Vous devez déployer une API pour que les modifications prennent effet.

Variables d'étape

Les variables d'étape sont des paires clé-valeur que vous pouvez définir pour une étape d'une WebSocket API. Elles se comportent comme les variables d'environnement et peuvent être utilisées dans votre configuration d'API.

Par exemple, vous pouvez définir une variable d'étape, puis définir sa valeur en tant que point de terminaison HTTP pour une intégration de proxy HTTP. Par la suite, vous pouvez référencer le point de terminaison à l'aide du nom de la variable d'étape associée. Ce faisant, vous pouvez utiliser la même configuration d'API avec un point de terminaison différent à chaque étape. De même, vous pouvez utiliser des variables d'étape pour spécifier une intégration de AWS Lambda fonction différente pour chaque étape de votre API.

> **Note**
>
> Les variables d'étape ne sont pas destinées à être utilisées pour des données sensibles, telles que les informations d'identification. Pour transmettre des données sensibles aux intégrations, utilisez un AWS Lambda autorisateur. Vous pouvez transmettre des données sensibles aux intégrations dans la sortie du mécanisme d'autorisation Lambda. Pour en savoir plus, consultez la section the section called “Format de réponse du mécanisme d'autorisation Lambda”.

Exemples

Pour utiliser une variable d'étape afin de personnaliser le point de terminaison d'intégration HTTP, vous devez d'abord définir le nom et la valeur de la variable stage (par exemple, `url`) avec la valeur `example.com`. Ensuite, configurez une intégration de proxy HTTP. Au lieu d'entrer l'URL du point de terminaison, vous pouvez demander à API Gateway d'utiliser la valeur de la variable d'étape, **`http://${stageVariables.url}`**. Cette valeur demande à API Gateway de remplacer votre variable d'étape `${}` au moment de l'exécution, en fonction de l'étape à laquelle se trouve votre API.

Vous pouvez référencer les variables d'étape de la même manière pour spécifier un nom de fonction Lambda ou un AWS ARN de rôle.

Lorsque vous spécifiez un nom de fonction Lambda en tant que valeur de variable d'étape, vous devez configurer les autorisations sur cette fonction Lambda manuellement. Pour ce faire, vous pouvez utiliser AWS Command Line Interface (AWS CLI).

```
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
```

Référence des variables d'étape API Gateway

URI d'intégration HTTP

Une variable d'étape peut être utilisée dans une URI d'intégration HTTP, comme illustré dans les exemples suivants.

- URI complet sans protocole – `http://${stageVariables.<variable_name>}`
- Domaine complet – `http://${stageVariables.<variable_name>}/resource/operation`
- Sous-domaine – `http://${stageVariables.<variable_name>}.example.com/resource/operation`
- Chemin – `http://example.com/${stageVariables.<variable_name>}/bar`
- Chaîne de requête – `http://example.com/foo?q=${stageVariables.<variable_name>}`

Fonctions Lambda

Vous pouvez utiliser une variable d'étape à la place d'un nom de fonction Lambda ou d'un alias, comme illustré dans les exemples suivants.

- `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**
>
> Pour utiliser une variable d'étape pour une fonction Lambda, la fonction doit se trouver dans le même compte que l'API. Les variables d'étape ne prennent pas en charge les fonctions Lambda inter-comptes.

AWS informations d'identification d'intégration

Vous pouvez utiliser une variable d'étape dans le cadre de l'ARN d'identification AWS d'un utilisateur ou d'un rôle, comme illustré dans l'exemple suivant.

- `arn:aws:iam::<account_id>:${stageVariables.<variable_name>}`

Déployer une WebSocket API dans API Gateway

Après avoir créé votre WebSocket API, vous devez la déployer pour que vos utilisateurs puissent l'invoquer.

Pour déployer une API, vous devez créer un déploiement d'API et l'associer à une étape. Chaque étape est un instantané de l'API et peut être appelée par les applications client.

> **Important**
>
> Chaque fois que vous mettez à jour une API, vous devez la redéployer. Les modifications apportées à tout autre élément que les paramètres d'étape nécessitent un redéploiement, notamment des modifications des ressources suivantes :
>
> - Acheminements
> - Intégrations
> - Mécanismes d'autorisation

Par défaut, vous êtes limité à 10 étapes pour chaque API. Nous recommandons de réutiliser les étapes pour vos déploiements.

Pour appeler une WebSocket API déployée, le client envoie un message à l'URL de l'API. L'URL est déterminée par le nom d'hôte et le nom de l'étape de l'API.

> **Note**
>
> API Gateway prend en charge des charges utiles jusqu'à 128 Ko, avec une taille de trame maximale de 32 Ko. Si un message dépasse 32 Ko, il devra être scindé en plusieurs trames, chacune de 32 Ko ou moins.

En utilisant le nom de domaine par défaut de l'API, l'URL (par exemple) d'une WebSocket API dans une étape donnée (*`{stageName}`*) est au format suivant :

```
wss://{api-id}.execute-api.{region}.amazonaws.com/{stageName}
```

Pour rendre l'URL de l' WebSocket API plus conviviale, vous pouvez créer un nom de domaine personnalisé (par exemple,`api.example.com`) pour remplacer le nom d'hôte par défaut de l'API. Le processus de configuration est le même que pour les API REST. Pour de plus amples informations, veuillez consulter the section called “Noms de domaine personnalisés”.

Les étapes permettent un contrôle de version solide de votre API. Par exemple, vous pouvez déployer une API dans une étape `test` et une étape `prod`, puis utiliser l'étape `test` comme version de test et l'étape `prod` comme version stable. Une fois que les mises à jour passent le test, vous pouvez migrer l'étape `test` vers l'étape `prod`. La promotion peut être effectuée en redéployant l'API à l'étape `prod`. Pour de plus amples informations sur les étapes, veuillez consulter the section called “Configuration d'une étape”.

Rubriques

- Créez un déploiement d' WebSocketAPI à l'aide du AWS CLI
- Création d'un déploiement d' WebSocket API à l'aide de la console API Gateway

Créez un déploiement d' WebSocketAPI à l'aide du AWS CLI

AWS CLI Pour créer un déploiement, utilisez la commande create-deployment comme indiqué dans l'exemple suivant :

```
aws apigatewayv2 --region us-east-1 create-deployment --api-id aabbccddee
```

Exemple de sortie :

```
{
    "DeploymentId": "fedcba",
    "DeploymentStatus": "DEPLOYED",
    "CreatedDate": "2018-11-15T06:49:09Z"
}
```

L'API déployée ne peut pas être appelée tant que vous n'associez pas le déploiement à une étape. Vous pouvez créer une nouvelle étape ou réutiliser une étape créée précédemment.

Pour créer une nouvelle étape et l'associer au déploiement, utilisez la commande create-stage comme indiqué dans l'exemple suivant :

```
aws apigatewayv2 --region us-east-1 create-stage --api-id aabbccddee --deployment-id
 fedcba --stage-name test
```

Exemple de sortie :

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

Pour réutiliser un stage existant, mettez à jour la `deploymentId` propriété du stage avec le nouvel ID de déploiement (*{deployment-id}*) à l'aide de la commande update-stage.

```
aws apigatewayv2 update-stage --region {region} \
   --api-id {api-id} \
   --stage-name {stage-name} \
   --deployment-id {deployment-id}
```

Création d'un déploiement d' WebSocket API à l'aide de la console API Gateway

Pour utiliser la console API Gateway afin de créer un déploiement pour une WebSocket API :

1. Connectez-vous à la console API Gateway et choisissez l'API.
2. Sélectionnez Deploy API (Déployer une API).
3. Choisissez l'étape souhaitée dans la liste déroulante ou saisissez le nom d'une nouvelle étape.

Politique de sécurité pour les WebSocket API

API Gateway applique une politique de sécurité `TLS_1_2` pour tous les points de terminaison WebSocket d'API.

Une politique de sécurité est une combinaison prédéfinie de version minimale de TLS et de suites de chiffrement proposées par Amazon API Gateway. Le protocole TLS résout les problèmes de sécurité de réseau tels que la falsification et le risque d'écoute illicite entre un client et un serveur. Lorsque vos clients établissent une liaison TLS vers votre API via le domaine personnalisé, la stratégie de sécurité applique les options de version TLS et de suite de chiffrement que vos clients peuvent choisir d'utiliser. Cette politique de sécurité accepte le trafic TLS 1.2 et TLS 1.3 et rejette le trafic TLS 1.0.

Protocoles TLS et chiffrements pris en charge pour les API WebSocket

Le tableau suivant décrit les protocoles TLS et les chiffrements pris en charge pour les API. WebSocket

Politique de sécurité	TLS_1_2
Protocoles TLS	
TLSv1.3	♦

Politique de sécurité	TLS_1_2
TLSv1.2	♦
Chiffrements TLS	
TLS_AES_128_GCM_SHA256	♦
TLS_AES_256_GCM_SHA384	♦
TLS_CHACHA20_POLY1305_SHA256	♦
ECDHE-ECDSA-AES128-GCM-SHA256	♦
ECDHE-RSA-AES128-GCM-SHA256	♦
ECDHE-ECDSA-AES128-SHA256	♦
ECDHE-RSA-AES128-SHA256	♦
ECDHE-ECDSA-AES256-GCM-SHA384	♦
ECDHE-RSA-AES256-GCM-SHA384	♦
ECDHE-ECDSA-AES256-SHA384	♦
ECDHE-RSA-AES256-SHA384	♦
AES128-GCM-SHA256	♦
AES128-SHA256	♦
AES256-GCM-SHA384	♦
AES256-SHA256	♦

Noms de chiffrement OpenSSL et RFC

OpenSSL et IETF RFC 5246 utilisent des noms différents pour les mêmes chiffrements. Pour une liste des noms de chiffrement, voirthe section called “Noms de chiffrement OpenSSL et RFC”.

Informations sur les API REST et les API HTTP

Pour plus d'informations sur les API REST et les API HTTP, consultez the section called “Choix d'une politique de sécurité” etthe section called “Politique de sécurité pour les API HTTP”.

Configuration de noms de domaine personnalisés pour les WebSocket API

Les noms de domaine personnalisés sont des URL plus simples et plus intuitives que vous pouvez fournir à vos utilisateurs d'API.

Après avoir déployé votre API, vous (et vos clients) pouvez appeler cette API à l'aide de l'URL de base par défaut au format suivant :

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

où `api-id` est généré par API Gateway, `region` (AWS Region) est spécifié par vous lors de la création de l'API, et `stage` est spécifié par vous lors du déploiement de l'API.

La partie nom d'hôte de l'URL (c'est-à-dire, *`api-id`*`.execute-api.`*`region`*`.amazonaws.com`) fait référence à un point de terminaison de l’API. Le point de terminaison d'API par défaut peut être difficile à retenir et non convivial.

Avec des noms de domaine personnalisés, vous pouvez configurer le nom d'hôte de votre API et choisir un chemin de base (par exemple, `myservice`) pour mapper l'URL alternative à votre API. Par exemple, une URL de base de l’API plus conviviale peut devenir :

```
https://api.example.com/myservice
```

> **Note**
>
> Un nom de domaine personnalisé pour une WebSocket API ne peut pas être mappé à des API REST ou à des API HTTP.
> Pour les WebSocket API, les noms de domaine personnalisés régionaux sont pris en charge.
> Pour les WebSocket API, TLS 1.2 est la seule version TLS prise en charge.

Enregistrement d'un nom de domaine

Pour pouvoir configurer des noms de domaine personnalisés pour vos API, vous devez avoir enregistré un nom de domaine Internet. Votre nom de domaine doit respecter la spécification RFC

1035 et peut comporter un maximum de 63 octets par étiquette et 255 octets au total. Si nécessaire, vous pouvez enregistrer un domaine Internet à l'aide de Amazon Route 53 ou utiliser le bureau d'enregistrement de domaine tiers de votre choix. Le nom de domaine personnalisé d'une API peut être le nom d'un sous-domaine ou du domaine racine (également nommé « zone apex ») d'un domaine Internet enregistré.

Après avoir créé un nom de domaine personnalisé dans API Gateway, vous devez créer ou mettre à jour l'enregistrement de ressource de votre fournisseur DNS pour mapper à votre point de terminaison API. Sans ce mappage, les demandes d'API destinées au nom de domaine personnalisé ne peuvent pas atteindre API Gateway.

Noms de domaine personnalisés régionaux

Lorsque vous créez un nom de domaine personnalisé pour une API régionale, API Gateway crée un nom de domaine régional pour l'API. Vous devez configurer un enregistrement DNS pour mapper le nom de domaine personnalisé vers le nom de domaine régional. Vous devez également fournir un certificat pour le nom de domaine personnalisé.

Noms de domaine personnalisés génériques

Avec les noms de domaine personnalisés génériques, vous pouvez prendre en charge un nombre presque infini de noms de domaine sans dépasser le quota par défaut. Par exemple, vous pouvez donner à chacun de vos clients son propre nom de domaine, `customername.api.example.com`.

Pour créer un nom de domaine personnalisé générique, vous pouvez spécifier un caractère générique (*) comme premier sous-domaine d'un domaine personnalisé qui représente tous les sous-domaines possibles d'un domaine racine.

Par exemple, le nom de domaine personnalisé générique `*.example.com` se traduit par des sous-domaines tels que `a.example.com`, `b.example.com` et `c.example.com`, qui effectuent tous un routage vers le même domaine.

Les noms de domaine personnalisés génériques prennent en charge des configurations distinctes des noms de domaine personnalisés standard d'API Gateway. Par exemple, dans un seul AWS compte, vous pouvez configurer `*.example.com` et `a.example.com` vous comporter différemment.

Vous pouvez utiliser les variables de contexte `$context.domainName` et `$context.domainPrefix` pour déterminer le nom de domaine utilisé par un client pour appeler votre API. Pour en savoir plus sur les variables de contexte, veuillez consulter Modèle de mappage API Gateway et référence à la variable de journalisation des accès.

Pour créer un nom de domaine personnalisé générique, vous devez fournir un certificat émis par ACM qui a été validé à l'aide du DNS ou de la méthode de validation par e-mail.

> **Note**
>
> Vous ne pouvez pas créer un nom de domaine personnalisé avec caractère générique si un autre AWS compte a créé un nom de domaine personnalisé en conflit avec le nom de domaine personnalisé avec caractère générique. Par exemple, si le compte A a créé `a.example.com`, le compte B ne peut pas créer le nom de domaine personnalisé générique `*.example.com`.
> Si les comptes A et B ont le même propriétaire, vous pouvez contacter le AWS Centre de support pour demander une exception.

Certificats pour les noms de domaine personnalisés

> **Important**
>
> Vous spécifiez le certificat de votre nom de domaine personnalisé. Si votre application utilise l'épinglage de certificat, parfois appelé épinglage SSL, pour épingler un certificat ACM, il est possible que l'application ne puisse pas se connecter à votre domaine après le AWS renouvellement du certificat. Pour plus d'informations, consultez la section Problèmes d'épinglage de certificat dans le Guide de l'utilisateur AWS Certificate Manager .

Pour fournir un certificat pour un nom de domaine personnalisé dans une région où ACM est pris en charge, vous devez demander un certificat à ACM. Pour fournir un certificat pour un nom de domaine personnalisé régional dans une région où ACM n'est pas pris en charge, vous devez importer un certificat dans API Gateway dans cette région.

Pour importer un certificat SSL/TLS, vous devez fournir le corps du certificat SSL/TLS au format PEM, sa clé privée, ainsi que la chaîne de certificats du nom de domaine personnalisé. Chaque certificat stocké dans ACM est identifié par son ARN. Pour utiliser un certificat AWS géré pour un nom de domaine, il suffit de référencer son ARN.

ACM permet de configurer et d'utiliser un nom de domaine personnalisé pour une API. Vous créez un certificat pour le nom de domaine donné (ou vous importez un certificat), configurez le nom de domaine dans API Gateway avec l'ARN du certificat fourni par ACM, et vous mappez un chemin de

base sous le nom de domaine personnalisé à une étape déployée de l'API. Avec les certificats émis par ACM, vous n'avez pas à vous inquiéter d'une éventuelle exposition des informations sensibles du certificat, par exemple sa clé privée.

Configuration d'un nom de domaine personnalisé

Pour de plus amples informations sur la configuration d'un nom de domaine personnalisé, veuillez consulter Préparation des certificats dans AWS Certificate Manager et Configuration d'un nom de domaine personnalisé régional dans API Gateway.

Utilisation des mappages d'API pour WebSocket les API

Les mappages d'API vous permettent de connecter des étapes d'API à un nom de domaine personnalisé. Après avoir créé un nom de domaine et configuré les enregistrements DNS, vous pouvez utiliser les mappages d'API pour envoyer le trafic vers vos API via votre nom de domaine personnalisé.

Un mappage d'API spécifie une API, une étape et éventuellement un chemin à utiliser pour le mappage. Par exemple, vous pouvez mapper l'étape `production` d'une API à `wss://api.example.com/orders`.

Avant de créer un mappage d'API, vous devez disposer d'une API, d'une étape et d'un nom de domaine personnalisé. Pour plus d'informations sur la création d'un nom de domaine personnalisé, consultez the section called “Configuration d'un nom de domaine personnalisé régional”.

Restrictions

- Dans un mappage d'API, le nom de domaine personnalisé et les API mappées doivent se trouver dans le même AWS compte.
- Les mappages d'API ne doivent contenir que des lettres, des chiffres et les caractères suivants : `$-_.+!*'()`.
- La longueur maximale du chemin d'un mappage d'API est de 300 caractères.
- Vous ne pouvez pas associer WebSocket des API au même nom de domaine personnalisé qu'une API HTTP ou une API REST.

Créer un mappage d'API

Pour créer un mappage d'API, vous devez d'abord créer un nom de domaine personnalisé, une API et une étape. Pour plus d'informations sur la création d'un nom de domaine personnalisé, consultez the section called “Configuration d'un nom de domaine personnalisé régional”.

AWS Management Console

Pour créer un mappage d'API

1. Connectez-vous à la console API Gateway à l'adresse : https://console.aws.amazon.com/apigateway.
2. Choisissez Noms de domaine personnalisés.
3. Sélectionnez un nom de domaine personnalisé que vous avez déjà créé.
4. Choisissez Mappages d'API.
5. Choisissez Configurer les mappages d'API.
6. Choisissez Ajouter un nouveau mappage.
7. Entrez une API, une Étape et, éventuellement, un Chemin d'accès.
8. Choisissez Enregistrer.

AWS CLI

La AWS CLI commande suivante crée un mappage d'API. Dans cet exemple, API Gateway envoie des demandes `api.example.com/v1` à l'API et à l'étape spécifiés.

```
aws apigatewayv2 create-api-mapping \
    --domain-name api.example.com \
    --api-mapping-key v1 \
    --api-id a1b2c3d4 \
    --stage test
```

AWS CloudFormation

L' AWS CloudFormation exemple suivant crée un mappage d'API.

```
MyApiMapping:
  Type: 'AWS::ApiGatewayV2::ApiMapping'
  Properties:
    DomainName: api.example.com
```

```
    ApiMappingKey: 'v1'
    ApiId: !Ref MyApi
    Stage: !Ref MyStage
```

Désactivation du point de terminaison par défaut pour une API WebSocket

Par défaut, les clients peuvent appeler votre API en utilisant le point de terminaison `execute-api` généré par API Gateway pour votre API. Pour vous assurer que les clients peuvent accéder à votre API en utilisant uniquement un nom de domaine personnalisé, désactivez le point de terminaison par défaut `execute-api`.

> **Note**
>
> Lorsque vous désactivez le point de terminaison par défaut, toutes les étapes d'une API sont affectées.

La AWS CLI commande suivante désactive le point de terminaison par défaut pour une WebSocket API.

```
aws apigatewayv2 update-api \
    --api-id abcdef123 \
    --disable-execute-api-endpoint
```

Après avoir désactivé le point de terminaison par défaut, vous devez déployer votre API pour que la modification prenne effet.

La AWS CLI commande suivante crée un déploiement.

```
aws apigatewayv2 create-deployment \
    --api-id abcdef123 \
    --stage-name dev
```

Protection de votre WebSocket API

Vous pouvez configurer la limitation pour vos API pour éviter qu'elles soient submergées de demandes. Les limitations sont appliquées dans la mesure du possible et doivent être considérées comme des cibles plutôt que des plafonds de demandes garantis.

API Gateway régule les demandes de limitations soumises à votre API à l'aide de l'algorithme de compartiment de jetons, où un jeton compte pour une demande. En particulier, API Gateway examine le taux de soumissions de demandes à un débit régulier et en mode rafale pour toutes les API de votre compte, par région. Dans l'algorithme de compartiment de jetons, une rafale peut permettre un dépassement prédéfini de ces limites, mais d'autres facteurs peuvent également entraîner le dépassement des limites dans certains cas.

Lorsque les soumissions de demandes dépassent les limites de taux régulier et en mode rafale des demandes, API Gateway commence à limiter les demandes. Les clients peuvent recevoir des réponses aux erreurs `429 Too Many Requests` à ce stade. Lors de la capture de ces exceptions, le client peut renvoyer les demandes en échec de façon à limiter le débit tout en respectant les limitations.

En tant que développeur d'API, vous pouvez définir les limites pour les étapes ou acheminements d'API individuels afin d'améliorer les performances globales de toutes les API de votre compte.

Limitation au niveau du compte par région

API Gateway limite par défaut les demandes régulières par seconde (rps) pour toutes les API d'un compte AWS , par Région. Il limite également la rafale (c'est-à-dire, la taille maximale de compartiment) sur toutes les API au sein d'un compte AWS , par région. Dans API Gateway, la limite en mode rafale correspond au nombre maximal d'envois de demandes simultanés qu'API Gateway peut traiter à tout moment sans renvoyer de réponses d'erreur `429 Too Many Requests`. Pour plus d'informations sur les quotas de limitation, consultez Quotas et remarques importantes.

Les limites par compte sont appliquées à toutes les API d'un compte dans une région spécifiée. La limite du taux au niveau du compte peut être augmentée à la demande. Des limitations plus élevées sont possibles avec des API qui ont des délais d'attente plus courts et des charges utiles plus petites. Pour demander une augmentation des limitations au niveau du compte par Région, contactez le Centre de support AWS. Pour plus d’informations, consultez Quotas et remarques importantes. Notez que ces limites ne peuvent pas être supérieures aux limites d' AWS étranglement.

Limitation au niveau de l’acheminement

Vous pouvez définir une limitation au niveau des acheminements, afin de remplacer les limitations de requêtes au niveau du compte pour une étape spécifique ou pour des acheminements particuliers de votre API. Les limites de limitation d'acheminement par défaut ne peuvent pas dépasser les limites de débit au niveau du compte.

Vous pouvez configurer la limitation au niveau des acheminements à l'aide de AWS CLI. La commande suivante configure la limitation personnalisée pour l'étape et l'acheminement spécifiés d'une API.

```
aws apigatewayv2 update-stage \
    --api-id a1b2c3d4 \
    --stage-name dev \
    --route-settings '{"messages":
{"ThrottlingBurstLimit":100,"ThrottlingRateLimit":2000}}'
```

WebSocket API de surveillance

Vous pouvez utiliser CloudWatch les métriques et CloudWatch les journaux pour surveiller WebSocket les API. En combinant les journaux et les métriques, vous pouvez enregistrer les erreurs et surveiller les performances de votre API.

> **Note**
>
> API Gateway peut ne pas générer de journaux et de métriques dans les cas suivants :
>
> - Nombre d'erreurs de demande d'entité 413 trop important
> - Nombre d'erreurs de demande 429 trop important
> - Erreurs de la série 400 provenant de demandes envoyées à un domaine personnalisé qui n'a pas de mappage d'API
> - Erreurs de la série 500 causées par des défaillances internes

Rubriques

- Surveillance de WebSocket l'exécution des API à l'aide de CloudWatch métriques
- Configuration de la journalisation pour une WebSocket API

Surveillance de WebSocket l'exécution des API à l'aide de CloudWatch métriques

Vous pouvez utiliser CloudWatch les métriques Amazon pour surveiller WebSocket les API. La configuration est similaire à celle utilisée pour les API REST. Pour plus d'informations, consultez Surveillance de l'exécution de l'API REST avec CloudWatch les métriques Amazon.

Les métriques suivantes sont prises en charge pour les WebSocket API :

Métrique	Description
ConnectCount	Nombre de messages envoyés à l'intégration de route `$connect`.
MessageCount	Le nombre de messages envoyés à l' WebSocket API, depuis ou vers le client.
IntegrationError	Nombre de demandes qui renvoient une réponse 4XX/5XX depuis l'intégration.
ClientError	Nombre de demandes pour lesquelles API Gateway renvoie une réponse 4XX avant l'invocation de l'intégration.
ExecutionError	Erreurs survenues lors de l'appel de l'intégration.
IntegrationLatency	Différence de temps entre l'envoi de la demande à l'intégration par API Gateway et la réception de la réponse de l'intégration par API Gateway. Supprimé pour les rappels et les intégrations fictives.

Vous pouvez utiliser les dimensions du tableau suivant pour filtrer les métriques API Gateway.

Dimension	Description
ApiId	Filtre les métriques API Gateway pour une API avec l'ID d'API spécifié.
ApiId, Scène	Filtre les métriques API Gateway pour trouver une étape d'API portant l'ID d'API et l'ID d'étape spécifiés.
ApiId, Méthode, ressource, étape	Filtre les métriques d'API Gateway pour une méthode d'API avec l'ID d'API, l'ID d'étape, le chemin de ressource et l'ID de route spécifiés. API Gateway n'enverra pas ces métriques à moins que vous n'ayez explicite ment activé CloudWatch les métriques détaillées. Vous pouvez le faire en appelant l'UpdateStageaction de l'API REST API Gateway V2 pour mettre à jour la `detailedM etricsEnabled` propriété sur`true`. Vous pouvez également appeler la AWS CLI commande update-st age pour mettre à jour la `DetailedMetricsEna bled` propriété en. `true` L'activation de ces métriques implique des frais supplémen taires pour votre compte.

Dimension	Description
	Pour plus d'informations sur les tarifs, consultez Amazon CloudWatch Pricing.

Configuration de la journalisation pour une WebSocket API

Vous pouvez activer la journalisation pour écrire des CloudWatch journaux dans Logs. Il existe deux types de connexion à l'API CloudWatch : la journalisation de l'exécution et la journalisation des accès. Lors de la journalisation de l'exécution, API Gateway gère les CloudWatch journaux. Le processus comprend la création de groupes de journaux et de flux de journaux, et la génération de rapports dans les flux de journaux sur toutes les demandes et réponses des utilisateurs.

Dans la journalisation des accès, vous, en tant que développeur d'API, souhaitez enregistrer qui a accédé à votre API et comment l'appelant à eu accès à l'API. Vous pouvez créer votre propre groupe de journaux ou en choisir un existant, qui peut être géré par API Gateway. Pour spécifier les détails d'accès, vous sélectionnez des variables `$context` (exprimées au format de votre choix) et vous choisissez un groupe de journaux comme destination.

Pour obtenir des instructions sur la configuration de la CloudWatch journalisation, consultezthe section called “Configuration de la journalisation des CloudWatch API à l'aide de la console API Gateway”.

Lorsque vous spécifiez Log Format (Format de journal), vous pouvez choisir les variables de contexte à journaliser. Les variables suivantes sont prises en charge.

Paramètre	Description
`$context.apiId`	Identifiant qu'API Gateway attribue à votre API.
`$context.authorize.error`	Message d'erreur d'autorisation.
`$context.authorize.latency`	Latence d'autorisation en millisecondes.
`$context.authorize.status`	Code d'état renvoyé à la suite d'une tentative d'autorisation.

Paramètre	Description
`$context.authorizer.error`	Message d'erreur renvoyé par un mécanisme d'autorisation.
`$context.authorizer.integrationLatency`	Latence de l'autorisation Lambda en ms.
`$context.authorizer.integrationStatus`	Code d'état renvoyé par un mécanisme d'autorisation Lambda.
`$context.authorizer.latency`	Latence du mécanisme d'autorisation en millisecondes (ms).
`$context.authorizer.requestId`	ID de demande du AWS point de terminaison.
`$context.authorizer.status`	Code d'état renvoyé par un mécanisme d'autorisation.
`$context.authorizer.principalId`	L'identifiant utilisateur principal qui est associé au jeton envoyé par le client et retourné par une fonction Lambda du mécanisme d'autorisation API Gateway Lambda. (Un mécanisme d'autorisation Lambda était auparavant connu sous le nom de mécanisme d'autorisation personnalisé.)

Paramètre	Description
`$context.authorizer.` *`property`*	Valeur obtenue à l'aide de stringify de la paire clé-valeur spécifiée du mappage `context` renvoyé par une fonction du mécanisme d'autorisation Lambda API Gateway. Par exemple, si le mécanisme d'autorisation retourne le mappage `context` suivant : `"context" : { "key": "value", "numKey": 1, "boolKey": true }` l'appel de `$context.authorizer.key` renvoie la chaîne `"value"`, l'appel de `$context.authorizer.numKey` renvoie la chaîne `"1"` et l'appel de `$context.authorizer.boolKey` renvoie la chaîne `"true"`.
`$context.authenticate.error`	Message d'erreur renvoyé à la suite d'une tentative d'authentification.
`$context.authenticate.latency`	Latence d'authentification en millisecondes.
`$context.authenticate.status`	Code d'état renvoyé à la suite d'une tentative d'authentification.
`$context.connectedAt`	Temps de connexion au format Epoch.
`$context.connectionId`	ID unique pour la connexion qui peut être utilisé pour effectuer un rappel au client.

Paramètre	Description
`$context.domainName`	Nom de domaine pour l' WebSocket API. Ce nom peut être utilisé pour effectuer un rappel au client (au lieu d'une valeur codée en dur).
`$context.error.message`	Chaîne contenant un message d'erreur API Gateway.
`$context.error.messageString`	La valeur entre guillemets de `$context.error.message`, à savoir `"$context.error.message"`.
`$context.error.responseType`	Type de réponse d'erreur.
`$context.error.validationErrorString`	Chaîne contenant un message d'erreur de validation détaillé.
`$context.eventType`	Type d'événement : `CONNECT`, `MESSAGE` ou `DISCONNECT`
`$context.extendedRequestId`	Équivalent à `$context.requestId`.
`$context.identity.accountId`	L'ID de AWS compte associé à la demande.
`$context.identity.apiKey`	Clé du propriétaire d'API associée à la demande d'API activée par clé.
`$context.identity.apiKeyId`	ID de clé du propriétaire d'API associée à la demande d'API activée par clé
`$context.identity.caller`	Identifiant principal de l'appelant qui a signé la demande. Pris en charge pour les routes qui utilisent l'autorisation IAM.

Paramètre	Description
`$context.identity.cognitoAuthenticationProvider`	Liste séparée par des virgules des fournisseurs d'authentification Amazon Cognito utilisés par l'appelant à l'origine de la demande. Disponible uniquement si la demande a été signée avec les informations d'identification Amazon Cognito. Par exemple, pour une identité provenant d'un pool d'utilisateurs Amazon Cognito, `cognito-idp.`*`region`*`.amazonaws.com/`*`user_pool_id`*`,cognito-idp.`*`region`*`.amazonaws.com/`*`user_pool_id`*`:CognitoSignIn:`*`token subject claim`* Pour de plus amples informations, veuillez consulter Utilisation des identités fédérées dans le Manuel du développeur Amazon Cognito.
`$context.identity.cognitoAuthenticationType`	Type d'authentification Amazon Cognito de l'appelant effectuant la demande. Disponible uniquement si la demande a été signée avec les informations d'identification Amazon Cognito. Les valeurs possibles incluent `authenticated` pour les identités authentifiées et `unauthenticated` pour les identités non authentifiées.
`$context.identity.cognitoIdentityId`	ID d'identité Amazon Cognito de l'appelant effectuant la demande. Disponible uniquement si la demande a été signée avec les informations d'identification Amazon Cognito.

Paramètre	Description
`$context.identity.cognitoId entityPoolId`	ID de groupe d'identités Amazon Cognito de l'appelant effectuant la demande. Disponibl e uniquement si la demande a été signée avec les informations d'identification Amazon Cognito.
`$context.identity.principalOrgId`	ID d'organisation AWS. Pris en charge pour les routes qui utilisent l'autorisation IAM.
`$context.identity.sourceIp`	Adresse IP source de la connexion TCP envoyant la demande à API Gateway.
`$context.identity.user`	Identifiant principal de l'utilisateur qui sera autorisé à accéder aux ressources. Pris en charge pour les routes qui utilisent l'autorisation IAM.
`$context.identity.userAgent`	Agent utilisateur de l'appelant de l'API.
`$context.identity.userArn`	ARN (Amazon Resource Name) de l'utilisateur identifié après l'authentification.
`$context.integration.error`	Message d'erreur renvoyé à partir d'une intégration.
`$context.integration.integr ationStatus`	Pour l'intégration du proxy Lambda, le code d'état est renvoyé par le code de fonction Lambda principal AWS Lambda, et non par le code de fonction Lambda principal.
`$context.integration.latency`	Latence d'intégration en millisecondes (ms). Équivalent à `$context.integrati onLatency` .
`$context.integration.requestId`	ID de demande du AWS point de terminais on. Équivalent à `$context.awsEndpoi ntRequestId` .

Paramètre	Description
`$context.integration.status`	Code d'état renvoyé à partir d'une intégration. Pour les intégrations de proxy Lambda, code d'état que votre code de fonction Lambda renvoie. Équivalent à `$context.integrationStatus` .
`$context.integrationLatency`	Latence d'intégration en ms, disponible pour la journalisation des accès uniquement.
`$context.messageId`	ID côté serveur unique pour un message. Uniquement disponible lorsque `$context.eventType` est défini sur `MESSAGE`.
`$context.requestId`	Identique à `$context.extendedRequestId` .
`$context.requestTime`	Durée des demandes au format CLF (`dd/MMM/yyyy:HH:mm:ss +-hhmm`).
`$context.requestTimeEpoch`	Heure de la demande au format Epoch, en millisecondes.
`$context.routeKey`	Clé de routage sélectionnée.
`$context.stage`	Étape de déploiement de l'appel d'API (par exemple, bêta ou production).
`$context.status`	Statut de la réponse.
`$context.waf.error`	Le message d'erreur renvoyé par AWS WAF.
`$context.waf.latency`	La AWS WAF latence en ms.
`$context.waf.status`	Le code d'état renvoyé par AWS WAF.

Quelques exemples de certains formats de journaux d'accès utilisés couramment sont affichés dans la console API Gateway et répertoriés ci-dessous.

- `CLF` (Format de journal commun):

```
$context.identity.sourceIp $context.identity.caller \
$context.identity.user [$context.requestTime] "$context.eventType $context.routeKey
 $context.connectionId" \
$context.status $context.requestId
```

  Les caractères de continuation (\) sont destinés à être une aide visuelle. Le format du journal doit être une seule ligne. Vous pouvez ajouter un caractère de nouvelle ligne (\n) à la fin du format du journal pour inclure une nouvelle ligne à la fin de chaque entrée du journal.

- `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"
}
```

  Les caractères de continuation (\) sont destinés à être une aide visuelle. Le format du journal doit être une seule ligne. Vous pouvez ajouter un caractère de nouvelle ligne (\n) à la fin du format du journal pour inclure une nouvelle ligne à la fin de chaque entrée du journal.

- `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>
```

Les caractères de continuation (\) sont destinés à être une aide visuelle. Le format du journal doit être une seule ligne. Vous pouvez ajouter un caractère de nouvelle ligne (\n) à la fin du format du journal pour inclure une nouvelle ligne à la fin de chaque entrée du journal.

- CSV (valeurs séparées par des virgules) :

```
$context.identity.sourceIp,$context.identity.caller, \
$context.identity.user,$context.requestTime,$context.eventType, \
$context.routeKey,$context.connectionId,$context.status, \
$context.requestId
```

Les caractères de continuation (\) sont destinés à être une aide visuelle. Le format du journal doit être une seule ligne. Vous pouvez ajouter un caractère de nouvelle ligne (\n) à la fin du format du journal pour inclure une nouvelle ligne à la fin de chaque entrée du journal.

Référence Amazon Resource Name (ARN) API Gateway

Les tableaux suivants répertorient les noms Amazon Resource Names (ARN) pour les ressources API Gateway. Pour en savoir plus sur l'utilisation des ARN dans AWS Identity and Access Management les politiques, consultez Fonctionnement d'Amazon API Gateway avec IAM etContrôle de l'accès à une API avec des autorisations IAM.

API HTTP et ressources WebSocket d'API

Ressource	ARN
AccessLogSettings	`arn:partition :apigatew ay: region::/apis/api-id/ stages/stage-name /accesslo gsettings`
Api	`arn:partition :apigatew ay: region::/apis/api-id`
Apis	`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`
Mécanisme d'autorisation	`arn:partition :apigatew ay: region::/apis/api-id/authoriz ers/ id`
Mécanismes d'autorisation	`arn:partition :apigatew ay: region::/apis/api-id/authoriz ers`

Ressource	ARN
Cors	arn:*partition* :apigatew ay: *region*::/apis/*api-id*/cors
Déploiement	arn:*partition* :apigatew ay: *region*::/apis/*api-id*/deployme nts/ *id*
Déploiements	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*
Intégrations	arn:*partition* :apigatew ay: *region*::/apis/*api-id*/integrat ions
IntegrationResponse	arn:*partition* :apigatew ay: *region*::/apis/*api-id*/integrat ionresponses/ *integration-respon se*

Ressource	ARN
IntegrationResponses	arn:*partition* :apigatew ay: *region*::/apis/*api-id*/integrat ionresponses
Modèle	arn:*partition* :apigatew ay: *region*::/apis/*api-id*/models/*id*
Modèles	arn:*partition* :apigatew ay: *region*::/apis/*api-id*/models
ModelTemplate	arn:*partition* :apigatew ay: *region*::/apis/*api-id*/models/*id*/ template
Acheminement	arn:*partition* :apigatew ay: *region*::/apis/*api-id*/routes/*id*
Acheminements	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*

Ressource	ARN
Étape	arn:*partition* :apigatew ay: *region*::/apis/*api-id*/ stages/*stage-name*
Étapes	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

Ressources API REST

Ressource	ARN
Compte	arn:*partition* :apigatew ay: *region*::/account
ApiKey	arn:*partition* :apigatew ay: *region*::/apikeys/ *id*
ApiKeys	arn:*partition* :apigatew ay: *region*::/apikeys
Mécanisme d'autorisation	arn:*partition* :apigatew ay: *region*::/restapis/ *api-id*/ authorizers/ *id*
Mécanismes d'autorisation	arn:*partition* :apigatew ay: *region*::/restapis/ *api-id*/ authorizers

Ressource	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
Déploiement	arn:*partition* :apigatew ay: *region*::/restapis/ *api-id*/ deployments/ *id*
Déploiements	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*

Ressource	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*
Méthode	arn:*partition* :apigatew ay: *region*::/restapis/ *api-id*/ resources/ *resource-id* /methods/ *http-method*

Ressource	ARN
MethodResponse	arn:*partition* :apigateway: *region*::/restapis/ *api-id*/resources/ *resource-id* /methods/ *http-method* /responses/ *status-code*
Modèle	arn:*partition* :apigateway: *region*::/restapis/ *api-id*/models/*model-name*
Modèles	arn:*partition* :apigateway: *region*::/restapis/ *api-id*/models
RequestValidator	arn:*partition* :apigateway: *region*::/restapis/ *api-id*/requestvalidators/ *id*
RequestValidators	arn:*partition* :apigateway: *region*::/restapis/ *api-id*/requestvalidators
Ressource	arn:*partition* :apigateway: *region*::/restapis/ *api-id*/resources/ *id*
Ressources	arn:*partition* :apigateway: *region*::/restapis/ *api-id*/resources
RestApi	arn:*partition* :apigateway: *region*::/restapis/ *api-id*
RestApis	arn:*partition* :apigateway: *region*::/restapis

Ressource	ARN
Étape	arn:*partition* :apigatew ay: *region*::/restapis/ *api-id*/ stages/*stage-name*
Étapes	arn:*partition* :apigatew ay: *region*::/restapis/ *api-id*/ stages
Balises	arn:*partition* :apigatew ay: *region*::/tags/*url-encoded-resource-arn*
Modèle	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`(API HTTP, WebSocket API et API REST)

Ressource	ARN
WebSocket Point de terminaison API	`arn:`*`partition`*` :execute-api: `*`region`*`:`*`account-id`*` :`*`api-id`*`/`*`stage`*`/`*`route-key`*
Point de terminaison API HTTP et API REST *	`arn:`*`partition`*` :execute-api: `*`region`*`:`*`account-id`*` :`*`api-id`*`/`*`stage`*`/`*`http-method`*` /`*`resource-path`*
Mécanisme d'autorisation Lambda **	`arn:`*`partition`*` :execute-api: `*`region`*`:`*`account-id`*` :`*`api-id`*`/authorizers/ `*`authorizer-id`*

* L'ARN du point de terminaison de route `$default` pour les API HTTP est `arn:`*`partition`*`:execute-api:`*`region`*`:`*`account-id`*`:`*`api-id`*`/*/$default`.

** Cet ARN s'applique uniquement lors de la définition de la condition `SourceArn` dans la stratégie de ressource pour une fonction de mécanisme d'autorisation Lambda. Pour obtenir un exemple, consultez the section called "Création d'un mécanisme d'autorisation Lambda".

Utilisation des extensions API Gateway vers OpenAPI

Les extensions API Gateway prennent en charge l'autorisation AWS spécifique et les intégrations d'API spécifiques à API Gateway pour les API REST et les API HTTP. Dans cette section, nous allons décrire les extensions API Gateway de la spécification OpenAPI.

> **Tip**
>
> Pour comprendre comment les extensions API Gateway sont utilisées dans une application, vous pouvez utiliser la console API Gateway pour créer une API REST ou une API HTTP et l'exporter vers un fichier de définition OpenAPI. Pour de plus amples informations sur la procédure d'exportation d'une API, veuillez consulter Exportation d'une API REST à partir d'API Gateway et Exportation d'une API HTTP depuis API Gateway.

Rubriques

- x-amazon-apigateway-integrationObjet .RequestTemplates
- x-amazon-apigateway-integrationObjet .RequestParameters
- x-amazon-apigateway-integrationobjet .responses
- x-amazon-apigateway-integrationobjet .response
- x-amazon-apigateway-integrationObjet .ResponseTemplates
- x-amazon-apigateway-integrationObjet .ResponseParameters
- x-amazon-apigateway-integrationObjet .TLSConfig
- x-amazon-apigateway-minimum-taille de compression
- x-amazon-apigateway-policy
- x-amazon-apigateway-requestpropriété -validator
- x-amazon-apigateway-requestobjet -validators
- x-amazon-apigateway-requestobjet -Validators.RequestValidator
- x-amazon-apigateway-tagpropriété -value

x-amazon-apigateway-anyobjet -method

Spécifie l'objet Opération OpenAPI de la méthode fourre-tout `ANY` d'API Gateway dans un objet Élément de chemin OpenAPI. Cet objet peut coexister avec d'autres objets Opération et attrape toute méthode HTTP non déclarée explicitement.

Le tableau suivant répertorie les propriétés étendues par API Gateway. Pour les autres propriétés Opération OpenAPI, consultez la spécification OpenAPI.

Propriétés

Nom de la propriété	Type	Description
`isDefaultRoute`	`Boolean`	Spécifie si un itinéraire est l'itinéraire `$default`. Prise en charge uniquement pour les API HTTP. Pour en savoir plus, consultez la section Utilisation des itinéraires pour les API HTTP.

Nom de la propriété	Type	Description
`x-amazon-apigateway-integration`	x-amazon-apigateway-integration objet	Spécifie l'intégration de la méthode au backend. Cette est une propriété étendue de l'objet opération OpenAPI. L'intégration peut être de type `AWS`, `AWS_PROXY` , `HTTP`, `HTTP_PROXY` ou `MOCK`.

x-amazon-apigateway-any-exemples de méthodes

L'exemple suivant intègre la méthode `ANY` sur une ressource de proxy, `{proxy+}`, avec une fonction Lambda, `TestSimpleProxy`.

```
    "/{proxy+}": {
      "x-amazon-apigateway-any-method": {
        "produces": [
          "application/json"
        ],
        "parameters": [
          {
            "name": "proxy",
            "in": "path",
            "required": true,
            "type": "string"
          }
        ],
        "responses": {},
        "x-amazon-apigateway-integration": {
          "uri": "arn:aws:apigateway:us-east-1:lambda:path/2015-03-31/functions/
arn:aws:lambda:us-east-1:123456789012:function:TestSimpleProxy/invocations",
          "httpMethod": "POST",
          "type": "aws_proxy"
        }
```

L'exemple suivant crée une route `$default` pour une API HTTP qui s'intègre à une fonction Lambda, `HelloWorld`.

```
"/$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 objet

Spécifie la configuration de partage de ressources d'origine croisée (CORS) pour une API HTTP. L'extension s'applique à la structure OpenAPI au niveau racine. Pour en savoir plus, consultez la section Configuration de CORS pour une API HTTP.

Propriétés

Nom de la propriété	Type	Description
`allowOrigins`	`Array`	Spécifie les origines autorisées.
`allowCredentials`	`Boolean`	Spécifie si les informations d'identification sont incluses dans la demande CORS.
`exposeHeaders`	`Array`	Spécifie les en-têtes qui sont exposés.
`maxAge`	`Integer`	Spécifie le nombre de secondes pendant lesquelles le navigateur doit mettre en cache les résultats de la demande de contrôle en amont.

Nom de la propriété	Type	Description
`allowMethods`	`Array`	Spécifie les méthodes HTTP autorisées.
`allowHeaders`	`Array`	Spécifie les en-têtes autorisés.

x-amazon-apigateway-cors exemple

Voici un exemple de configuration CORS pour une API HTTP.

```
"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-apipropriété -key-source

Spécifiez la source qui recevra une clé API pour limiter les méthodes d'API qui requièrent une clé. Cette propriété d'API est de type `String`. Pour plus d'informations sur la configuration d'une méthode nécessitant une clé d'API, consultezthe section called “Configuration d'une méthode pour utiliser des clés d'API avec une définition OpenAPI”.

Spécifiez la source de la clé API pour les demandes. Les valeurs valides sont :

- `HEADER` pour recevoir la clé API à partir de l'en-tête `X-API-Key` d'une demande.
- `AUTHORIZER` pour la réception de la clé d'API à partir de l'élément `UsageIdentifierKey` à partir d'un mécanisme d'autorisation Lambda (anciennement appelé Custom Authorizer (mécanisme d'autorisation personnalisé)).

x-amazon-apigateway-apiexemple de -key-source

L'exemple suivant définit l'en-tête `X-API-Key` en tant que source de clé API.

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 objet

Définit un type d'autorisation à appliquer pour l'autorisation des appels de méthode dans API Gateway.

Propriétés

Nom de la propriété	Type	Description
`type`	`string`	Spécifie le type d'autorisation. Spécifiez `"NONE"` pour un accès ouvert. Spécifiez `"AWS_IAM"` pour utiliser des autorisations IAM. Les valeurs ne sont pas sensibles à la casse.

x-amazon-apigateway-auth exemple

L'exemple suivant définit le type d'autorisation pour une méthode d'API.

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 objet

Définit un mécanisme d'autorisation Lambda, un groupe d'utilisateurs Amazon Cognito ou un mécanisme d'autorisation JWT à appliquer pour l'autorisation des appels de méthode dans API Gateway. Cette extension s'applique à la définition de sécurité dans OpenAPI 2 et OpenAPI 3.

Propriétés

Nom de la propriété	Type	Description
`type`	`string`	Type du mécanisme d'autorisation. Cette propriété est requise. Pour les API REST, spécifiez `token` pour un mécanisme d'autorisation avec l'identité de l'appelant incorporée dans un jeton d'autorisation. Spécifiez `request` pour un mécanisme d'autorisation avec l'identité de l'appelant contenue dans les paramètres de la demande. Spécifiez `cognito_user_pools` pour un mécanisme d'autorisation qui utilise un groupe d'utilisateurs Amazon Cognito pour contrôler l'accès à votre API.

Nom de la propriété	Type	Description
		Pour les API HTTP, spécifiez `request` pour un mécanisme d'autorisation Lambda avec l'identité de l'appelant contenue dans les paramètres de la demande. Spécifiez `jwt` pour un mécanisme d'autorisation JWT.
`authorizerUri`	`string`	Identificateur de ressource uniforme (URI) de la fonction Lambda du mécanisme d'autorisation. La syntaxe est la suivante : `"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`	Informations d'identification requises pour appeler le mécanisme d'autorisation, le cas échéant, sous la forme de l'ARN d'un rôle d'exécution IAM. Par exemple, « arn:aws:iam::*account-id* :*IAM_role* ».

Nom de la propriété	Type	Description
`authorizerPayloadFormatVersion`	`string`	Pour les API HTTP, spécifie le format des données envoyées par API Gateway à un mécanisme d'autorisation Lambda et comment API Gateway interprète la réponse de Lambda. Pour en savoir plus, consultez la section the section called “Version du format de charge utile”.
`enableSimpleResponses`	`Boolean`	Pour les API HTTP, spécifie si un mécanisme d'autorisation `request` renvoie une valeur booléenne ou une stratégie IAM. Prise en charge uniquement pour les mécanismes d'autorisation avec un `authorizerPayloadFormatVersion` de `2.0`. Si cette option est activée, la fonction du mécanisme d'autorisation Lambda renvoie une valeur booléenne. Pour en savoir plus, consultez la section the section called “Réponse de la fonction Lambda pour le format 2.0”.

Nom de la propriété	Type	Description
`identitySource`	`string`	Liste (séparée par des virgules) des expressions de mappage des paramètres de la demande en tant que source d'identité. Applicable pour le mécanisme d'autorisation de types `request` et `jwt` uniquement.
`jwtConfiguration`	`Object`	Spécifie l'émetteur et les audiences d'un mécanisme d'autorisation JWT. Pour de plus amples informations, veuillez consulter JWTConfiguration dans la Référence d'API API Gateway version 2. Prise en charge uniquement pour les API HTTP.
`identityValidationExpression`	`string`	Expression régulière permettant de valider le jeton en tant qu'identité entrante. Par exemple, « ^x-[a-z]+ ». Pris en charge uniquement pour les `TOKEN` autorisateurs d'API REST.
`authorizerResultTtlInSeconds`	`string`	Nombre de secondes pendant lesquelles le résultat du mécanisme d'autorisation est mis en cache.

Nom de la propriété	Type	Description
`providerARNs`	Tableau d'éléments `string`	Liste des ARN des groupes d'utilisateurs Amazon Cognito pour `COGNITO_USER_POOLS` .

x-amazon-apigateway-authorizer exemples d'API REST

L'exemple de définitions de sécurité OpenAPI suivant spécifie un mécanisme d'autorisation Lambda de type « token » nommé `test-authorizer`.

```
 "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
     }
   }
 }
```

L'extrait d'objet Opération OpenAPI suivant configure la méthode `GET /http` afin d'utiliser le mécanisme d'autorisation Lambda précédent.

```
   "/http" : {
      "get" : {
        "responses" : { },
        "security" : [ {
          "test-authorizer" : [ ]
        } ],
        "x-amazon-apigateway-integration" : {
          "type" : "http",
          "responses" : {
            "default" : {
              "statusCode" : "200"
            }
          },
          "httpMethod" : "GET",
          "uri" : "http://api.example.com"
        }
      }
    }
```

L'exemple de définitions de sécurité OpenAPI suivant spécifie un mécanisme d'autorisation Lambda de type « request », avec un seul paramètre d'en-tête (`auth`) comme source d'identité. L'exemple `securityDefinitions` est nommé `request_authorizer_single_header`.

```
"securityDefinitions": {
    "request_authorizer_single_header" : {
      "type" : "apiKey",
      "name" : "auth",              // The name of a single header or query parameter
 as the identity source.
      "in" : "header",              // The location of the single identity source
 request parameter. The valid value is "header" or "query"
      "x-amazon-apigateway-authtype" : "custom",
      "x-amazon-apigateway-authorizer" : {
        "type" : "request",
        "identitySource" : "method.request.header.auth",   // Request parameter mapping
 expression of the identity source. In this example, it is the 'auth' header.
        "authorizerCredentials" : "arn:aws:iam::123456789012:role/AWSepIntegTest-CS-
LambdaRole",
        "authorizerUri" : "arn:aws:apigateway:us-east-1:lambda:path/2015-03-31/
functions/arn:aws:lambda:us-east-1:123456789012:function:APIGateway-Request-
Authorizer:vtwo/invocations",
        "authorizerResultTtlInSeconds" : 300
      }
    }
```

```
}
```

L'exemple de définitions de sécurité OpenAPI suivant spécifie un mécanisme d'autorisation Lambda de type « request », avec un seul en-tête (`HeaderAuth1`) et un seul paramètre de chaîne de requête `QueryString1` comme sources d'identité.

```
"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
      }
    }
}
```

L'exemple de définitions de sécurité OpenAPI suivant spécifie un mécanisme d'autorisation Lambda API Gateway de type « request », avec une seule variable d'étape (`stage`) comme source d'identité.

```
"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
      }
    }
}
```

L'exemple de définition de sécurité OpenAPI suivant spécifie un groupe d'utilisateurs Amazon Cognito en tant que mécanisme d'autorisation.

```
 "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"
        ]
      }
    }
```

L'extrait d'objet d'opération OpenAPI suivant définit `GET /http` pour utiliser le groupe d'utilisateurs Amazon Cognito précédent en tant que mécanisme d'autorisation, sans portées personnalisées.

```
  "/http" : {
     "get" : {
       "responses" : { },
       "security" : [ {
         "cognito-pool" : [ ]
       } ],
       "x-amazon-apigateway-integration" : {
         "type" : "http",
         "responses" : {
```

```
            "default" : {
              "statusCode" : "200"
            }
          },
          "httpMethod" : "GET",
          "uri" : "http://api.example.com"
        }
      }
    }
```

x-amazon-apigateway-authorizer exemples d'API HTTP

L'exemple OpenAPI 3.0 suivant crée un mécanisme d'autorisation JWT pour une API HTTP qui utilise Amazon Cognito comme fournisseur d'identité, avec l'en-tête `Authorization` comme source d'identité.

```
"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"
    }
  }
}
```

L'exemple OpenAPI 3.0 suivant produit le même mécanisme d'autorisation JWT que l'exemple précédent. Toutefois, cet exemple utilise la propriété `openIdConnectUrl` d'OpenAPI pour détecter automatiquement l'émetteur. Le `openIdConnectUrl` doit être entièrement formé.

```
"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"
    }
  }
}
```

L'exemple suivant crée un mécanisme d'autorisation Lambda pour une API HTTP. Cet exemple de mécanisme d'autorisation utilise l'en-tête `Authorization` comme source d'identité. Le mécanisme d'autorisation utilise la version de format de charge utile `2.0` et renvoie une valeur booléenne, car `enableSimpleResponses` est défini sur `true`.

```
"securitySchemes" : {
  "lambda-authorizer" : {
    "type" : "apiKey",
    "name" : "Authorization",
    "in" : "header",
    "x-amazon-apigateway-authorizer" : {
      "type" : "request",
      "identitySource" : "$request.header.Authorization",
      "authorizerUri" : "arn:aws:apigateway:us-west-2:lambda:path/2015-03-31/functions/
arn:aws:lambda:us-west-2:123456789012:function:function-name/invocations",
      "authorizerPayloadFormatVersion" : "2.0",
      "authorizerResultTtlInSeconds" : 300,
      "enableSimpleResponses" : true
    }
  }
}
```

x-amazon-apigateway-authtype propriété

Pour les API REST, cette extension peut être utilisée pour définir un type personnalisé de mécanisme d'autorisation Lambda. Dans ce cas, la valeur est de forme libre. Par exemple, une API peut avoir plusieurs mécanismes d'autorisation Lambda qui utilisent différents schémas internes. Cette extension vous permet d'identifier le schéma interne d'un mécanisme d'autorisation Lambda.

Plus souvent, dans les API HTTP et les API REST, il peut également être utilisé pour définir l'autorisation IAM sur plusieurs opérations qui partagent le même schéma de sécurité. Dans ce cas, le terme `awsSigv4` est un terme réservé, ainsi que tout terme doté du préfixe`aws`.

Cette extension s'applique au schéma de sécurité de type `apiKey` dans OpenAPI 2 et OpenAPI 3.

x-amazon-apigateway-authtype exemple

L'exemple OpenAPI 3 suivant définit l'autorisation IAM sur plusieurs ressources dans une API REST ou une API HTTP :

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

L'exemple OpenAPI 3 suivant définit un mécanisme d'autorisation Lambda avec un schéma personnalisé pour une API REST :

```
{
  "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"
}
```

Voir aussi

authorizer.authType

x-amazon-apigateway-binarypropriété -media-types

Spécifie la liste des types de médias binaires pris en charge par API Gateway, par exemple `application/octet-stream` et `image/jpeg`. Cette extension est un tableau JSON. Elle doit être incluse en tant qu'extension fournisseur de premier niveau dans le document OpenAPI.

x-amazon-apigateway-binaryexemple de -media-types

L'exemple suivant montre l'ordre de recherche d'encodage d'une API.

```
"x-amazon-apigateway-binary-media-types": [ "application/octet", "image/jpeg" ]
```

x-amazon-apigateway-documentation objet

Définit les parties de la documentation à importer dans API Gateway. Cet objet est un objet JSON qui contient un tableau d'instances `DocumentationPart`.

Propriétés

Nom de la propriété	Type	Description
`documentationParts`	`Array`	Un tableau des instances `DocumentationPart` exportées ou importées.
`version`	`String`	L'identifiant de version de l'instantané des parties de la documentation exportées.

x-amazon-apigateway-documentation exemple

L'exemple suivant de l'extension API Gateway vers OpenAPI définit les instances `DocumentationParts` à importer ou à exporter depuis une API dans API Gateway.

```
{ ...
  "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-objet de configuration

Spécifie les détails de la configuration du point de terminaison pour une API. Cette extension est une propriété étendue de l'objet opération OpenAPI. Cet objet doit être présent dans les extensions fournisseur de premier niveau pour Swagger 2.0. Pour OpenAPI 3.0, il doit être présent sous les extensions fournisseur de l'objet serveur.

Propriétés

Nom de la propriété	Type	Description
`disableExecuteApiEndpoint`	Booléen	Spécifie si les clients peuvent appeler votre API à l'aide du point de terminaison `execute-api` par défaut. Par défaut, les clients peuvent appeler votre API avec le point de terminaison `https://{api_id}.execute-api.{region}.amazonaws.com` par défaut. Pour obliger les clients à utiliser un nom de domaine personnalisé pour appeler votre API, spécifiez `true`.
`vpcEndpointIds`	Tableau d'éléments `String`	Liste d' VpcEndpoint identifiants permettant de créer des enregistrements d'alias Route 53 pour une API REST. Ce paramètre est uniquement pris en charge pour le type de point de terminaison `PRIVATE` des API REST.

x-amazon-apigateway-endpoint-exemples de configuration

L'exemple suivant associe les points de terminaison de VPC spécifiés à l'API REST.

```
"x-amazon-apigateway-endpoint-configuration": {
    "vpcEndpointIds": ["vpce-0212a4ababd5b8c3e", "vpce-01d622316a7df47f9"]
}
```

L'exemple suivant désactive le point de terminaison par défaut d'une API.

```
"x-amazon-apigateway-endpoint-configuration": {
    "disableExecuteApiEndpoint": true
}
```

x-amazon-apigateway-gatewayobjet -responses

Définit les réponses de passerelle pour une API sous la forme d'une chaîne à mappage de paires GatewayResponseclé-valeur. L'extension s'applique à la structure OpenAPI au niveau racine.

Propriétés

Nom de la propriété	Type	Description
responseType	x-amazon-apigateway-gateway- Réponses. Réponse de la passerelle	Une réponse `GatewayResponse` pour le *responseType* spécifié.

x-amazon-apigateway-gatewayexemple de -réponses

L'exemple d'extension API Gateway pour OpenAPI suivant définit une GatewayResponsescarte contenant deux GatewayResponseinstances, l'une pour le `DEFAULT_4XX` type et l'autre pour le type. `INVALID_API_KEY`

```
{
  "x-amazon-apigateway-gateway-responses": {
    "DEFAULT_4XX": {
      "responseParameters": {
```

```
        "gatewayresponse.header.Access-Control-Allow-Origin": "'domain.com'"
      },
      "responseTemplates": {
        "application/json": "{\"message\": test 4xx b }"
      }
    },
    "INVALID_API_KEY": {
      "statusCode": "429",
      "responseTemplates": {
        "application/json": "{\"message\": test forbidden }"
      }
    }
  }
}
```

x-amazon-apigateway-gateway-Responses.GatewayResponse objet

Définit une réponse de passerelle d'un type de réponse donné, y compris le code de statut, les paramètres de réponse applicables, ou les modèles de réponse.

Propriétés

Nom de la propriété	Type	Description
responseParameters	x-amazon-apigateway-gateway-Responses.ResponseParameters	Spécifie les GatewayResponseparamètres, à savoir les paramètres d'en-tête. Les valeurs des paramètres peuvent prendre n'importe quelle valeur de paramètre de demande entrante ou une valeur personnalisée statique.
responseTemplates	x-amazon-apigateway-gateway-Responses.ResponseModèles	Spécifie les modèles de mappage de la réponse de passerelle. Les modèles ne sont pas traités par le moteur VTL.

Nom de la propriété	Type	Description
statusCode	string	Code de statut HTTP pour la réponse de passerelle.

x-amazon-apigateway-gateway-Responses.gatewayResponse exemple

L'exemple suivant de l'extension API Gateway pour OpenAPI définit un GatewayResponsepour personnaliser la INVALID_API_KEY réponse afin de renvoyer le code d'état456, la valeur d'api-keyen-tête de la demande entrante et un "Bad api-key" message.

```
"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-gatewayObjet - Respones.ResponseParameters

Définit une string-to-string carte de paires clé-valeur pour générer les paramètres de réponse de la passerelle à partir des paramètres de demande entrante ou à l'aide de chaînes littérales. Prise en charge uniquement pour les API REST.

Propriétés

Nom de la propriété	Type	Description
gatewayresponse. *param-position* .*param-name*	string	*param-position* peut être header, path ou querystring . Pour plus d'informations, consultez Mappage des données de demande de

Nom de la propriété	Type	Description
		méthode aux paramètres de demande d'intégration.

x-amazon-apigateway-gatewayExemple de -Respones.ResponseParameters

L'exemple d'extensions OpenAPI suivant montre une expression de mappage de paramètres de GatewayResponseréponse pour activer le support CORS pour les ressources des domaines. `*.example.domain`

```
"responseParameters": {
  "gatewayresponse.header.Access-Control-Allow-Origin": '*.example.domain',
  "gatewayresponse.header.from-request-header" : method.request.header.Accept,
  "gatewayresponse.header.from-request-path" : method.request.path.petId,
  "gatewayresponse.header.from-request-query" : method.request.querystring.qname
}
```

x-amazon-apigateway-gateway-ResponseTemplates, objet ResponseTemplates

Définit les modèles de GatewayResponsemappage, sous forme de string-to-string carte de paires clé-valeur, pour une réponse de passerelle donnée. Pour chaque paire clé-valeur, la clé est le type de contenu. Par exemple, « application/json » et la valeur est un modèle de mappage obtenu à l'aide de stringify pour les substitutions de variables simples. Un modèle de mappage `GatewayResponse` n'est pas traité par le moteur VTL (Velocity Template Language).

Propriétés

Nom de la propriété	Type	Description
`content-type`	`string`	Corps de modèle de mappage `GatewayResponse` prenant en charge uniquement les substitutions de variables

Nom de la propriété	Type	Description
		simples pour personnaliser un corps de réponse de passerelle.

x-amazon-apigateway-gatewayExemple de -ResponseResponseTemplates

L'exemple d'extensions OpenAPI suivant montre un modèle de GatewayResponsemappage permettant de personnaliser une réponse d'erreur générée par API Gateway dans un format spécifique à l'application.

```
      "responseTemplates": {
        "application/json": "{ \"message\": $context.error.messageString, \"type\":
$context.error.responseType, \"statusCode\": '488' }"
      }
```

L'exemple d'extensions OpenAPI suivant montre un modèle de GatewayResponsemappage permettant de remplacer une réponse d'erreur générée par API Gateway par un message d'erreur statique.

```
      "responseTemplates": {
        "application/json": "{ \"message\": 'API-specific errors' }"
      }
```

x-amazon-apigateway-importexport-version

Spécifie la version de l'algorithme d'importation et d'exportation API Gateway pour les API HTTP. À l'heure actuelle, la seule valeur prise en charge est `1.0`. Pour de plus amples informations, veuillez consulter exportVersion dans la Référence d'API API Gateway Version 2.

x-amazon-apigateway-importexport-exemple de version

L'exemple suivant définit la version d'importation et d'exportation sur `1.0`.

```
{
    "openapi": "3.0.1",
```

```
    "x-amazon-apigateway-importexport-version": "1.0",
    "info": { ...
```

x-amazon-apigateway-integration objet

Spécifie les détails de l'intégration au backend utilisée pour cette méthode. Cette extension est une propriété étendue de l'objet opération OpenAPI. Le résultat est un objet Intégration API Gateway.

Propriétés

Nom de la propriété	Type	Description
`cacheKeyParameters`	Tableau d'éléments `string`	Liste de paramètres de demande dont les valeurs doivent être mises en cache.
`cacheNamespace`	`string`	Groupe de balises spécifiques à l'API des paramètres associés mis en cache.
`connectionId`	`string`	L'ID d'un VpcLinkpour l'intégration privée.
`connectionType`	`string`	Type de connexion de l'intégration. La valeur valide est `"VPC_LINK"` pour l'intégration privée ou `"INTERNET"`, dans le cas contraire.
`credentials`	`string`	Pour les informations d'identification basées sur les rôles AWS IAM, spécifiez l'ARN d'un rôle IAM approprié. Si cette propriété n'est pas spécifiée, les informations d'identification utilisent par défaut les autorisations basées sur la ressource, qui doivent être

Nom de la propriété	Type	Description
		ajoutées manuellement pour permettre à l'API d'accéder à la ressource. Pour de plus amples informations, veuillez consulter Accord d'autorisations à l'aide d'une stratégie de ressources. Remarque : pour des performances optimales, lorsque vous utilisez les informations d'identification IAM, assurez-vous que les points de terminaison régionaux STS AWS sont activés pour la région où cette API est déployée.

Nom de la propriété	Type	Description
`contentHandling`	`string`	Types de conversion de l'encodage des charges utiles de la demande. Les valeurs valides sont 1) `CONVERT_TO_TEXT` , pour la conversion d'une charge utile binaire en chaîne encodée en base64 ou pour la conversion d'une charge utile de texte en chaîne encodée en `utf-8` ou pour le passage de la charge utile de texte en mode natif sans modification, et 2) `CONVERT_TO_BINARY` , pour la conversion d'une charge utile de texte en bloc encodé en base64 ou le passage par une charge utile binaire en mode natif sans modification.
`httpMethod`	`string`	Méthode HTTP utilisée dans la demande d'intégration. Pour les appels de fonction Lambda, la valeur doit être `POST`.

Nom de la propriété	Type	Description
`integrationSubtype`	`string`	Spécifie le sous-type d'intégration pour une intégration AWS de services. Prise en charge uniquement pour les API HTTP. Pour connaître les sous-types d'intégration pris en charge, veuillez consulter the section called “AWS référence sur les intégrations de services”.
`passthroughBehavior`	`string`	Spécifie comment une charge utile de demande d'un type de contenu non mappé est transmise par le biais de la demande d'intégration sans modification. Les valeurs prises en charge sont `when_no_templates` , `when_no_match` et `never`. Pour de plus amples informations, veuillez consulter Integration.passthroughBehavior.

Nom de la propriété	Type	Description
`payloadFormatVersion`	`string`	Spécifie le format de la charge utile envoyée à une intégration. Obligatoire pour les API HTTP. Pour les API HTTP, les valeurs prises en charge pour les intégrations de proxy Lambda sont `1.0` et `2.0`. Pour toutes les autres intégrations, `1.0` est la seule valeur prise en charge. Pour en savoir plus, consultez the section called “AWS Lambda intégrations” et the section called “AWS référence sur les intégrations de services”.

Nom de la propriété	Type	Description
`requestParameters`	x-amazon-apigateway-integrationObjet .RequestParameters	Pour les API REST, spécifie les mappages des paramètres de demande de méthode aux paramètres de demande d'intégration. Les paramètres de demande pris en charge sont `querystring` , `path`, `header` et `body`. Pour les API HTTP, les paramètres de demande sont une carte clé-valeur spécifiant les paramètres qui sont transmis aux intégrations `AWS_PROXY` avec un paramètre spécifié `integrationSubtype` . Vous pouvez fournir des valeurs statiques ou des données de demande de carte, des variables intermédiaires ou des variables de contexte évaluées au moment de l'exécution. Pour en savoir plus, consultez la section the section called “AWS intégrations de services”.
`requestTemplates`	x-amazon-apigateway-integrationObjet .RequestTemplates	Modèles de mappage pour la charge utile des demandes des types MIME spécifiés.

Nom de la propriété	Type	Description
`responses`	x-amazon-apigateway-integrationobjet .responses	Définit les réponses de la méthode et spécifie les mappages de paramètres souhaités ou les mappages de charge utile des réponses d'intégration aux réponses de méthode.
`timeoutInMillis`	`integer`	Délais d'expiration de l'intégration compris entre 50 ms et 29 000 ms.

Nom de la propriété	Type	Description
`type`	`string`	Type d'intégration au backend spécifié. Les valeurs valides sont : • `http` ou `http_proxy` : pour l'intégration avec un backend HTTP. • `aws_proxy` , pour l'intégration avec les fonctions AWS Lambda. • `aws`, pour l'intégration aux fonctions AWS Lambda ou à d'autres AWS services, tels qu'Amazon DynamoDB, Amazon Simple Notification Service ou Amazon Simple Queue Service. • `mock` : pour l'intégration avec API Gateway sans appel de backend. Pour de plus amples informations sur les types d'intégrations, veuillez consulter integration:type.
`tlsConfig`	the section called “x-amazon-apigateway-integration.TLSConfig”	Spécifie la configuration TLS pour une intégration.

Nom de la propriété	Type	Description
`uri`	`string`	URI du point de terminaison du backend. Pour les intégrations de type `aws`, il s'agit de la valeur d'ARN. Pour l'intégration HTTP, il s'agit de l'URL du point de terminaison HTTP, y compris le schéma `https` ou `http`.

x-amazon-apigateway-integration exemples

Pour les API HTTP, vous pouvez définir des intégrations dans la section composants de votre définition OpenAPI. Pour en savoir plus, consultez la section x-amazon-apigateway-integrations objet.

```
"x-amazon-apigateway-integration": {
    "$ref": "#/components/x-amazon-apigateway-integrations/integration1"
}
```

L'exemple suivant crée une intégration avec une fonction Lambda. A titre de démonstration, les exemples de modèles de mappage présentés dans les sections `requestTemplates` et `responseTemplates` des exemples ci-dessous sont supposés s'appliquer à la charge utile au format JSON suivante : `{ "name":"value_1", "key":"value_2", "redirect": {"url" :"..."} }` pour générer une sortie JSON `{ "stage":"value_1", "user-id":"value_2" }` ou une sortie XML `<stage>value_1</stage>`.

```
"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'"
            }
        }
    }
}
```

Notez que les guillemets (") pour la chaîne JSON doivent être mis en séquence d'échappement (\") dans les modèles de mappage.

x-amazon-apigateway-integrations objet

Définit une collection d'intégrations. Vous pouvez définir des intégrations dans la section Composants de votre définition OpenAPI et réutiliser les intégrations pour plusieurs routes. Prise en charge uniquement pour les API HTTP.

Propriétés

Nom de la propriété	Type	Description
intégration	x-amazon-apigateway-integration objet	Collection d'objets d'intégration.

x-amazon-apigateway-integrations exemple

L'exemple suivant crée une API HTTP qui définit deux intégrations et fait référence aux intégrations à l'aide de `$ref": "#/components/x-amazon-apigateway-integrations/`*integration-name*.

```
{
  "openapi": "3.0.1",
  "info":
    {
      "title": "Integrations",
      "description": "An API that reuses integrations",
      "version": "1.0"
    },
  "servers": [
  {
    "url": "https://example.com/{basePath}",
    "description": "The production API server",
    "variables":
      {
        "basePath":
          {
            "default": "example/path"
          }
      }
  }],
  "paths":
    {
```

```
      "/":
        {
          "get":
            {
              "x-amazon-apigateway-integration":
                {
                  "$ref": "#/components/x-amazon-apigateway-integrations/integration1"

                }
            }
        },
       "/pets":
        {
          "get":
            {
              "x-amazon-apigateway-integration":
                {
                  "$ref": "#/components/x-amazon-apigateway-integrations/integration1"

                }
            }
        },
       "/checkout":
        {
          "get":
            {
              "x-amazon-apigateway-integration":
                {
                  "$ref": "#/components/x-amazon-apigateway-integrations/integration2"
                }
            }
        }
    },
    "components": {
      "x-amazon-apigateway-integrations":
        {
          "integration1":
            {
              "type": "aws_proxy",
              "httpMethod": "POST",
              "uri": "arn:aws:apigateway:us-east-2:lambda:path/2015-03-31/functions/
arn:aws:lambda:us-east-2:123456789012:function:my-function/invocations",
              "passthroughBehavior": "when_no_templates",
              "payloadFormatVersion": "1.0"
```

```
            },
          "integration2":
            {
              "type": "aws_proxy",
              "httpMethod": "POST",
              "uri": "arn:aws:apigateway:us-east-2:lambda:path/2015-03-31/functions/
arn:aws:lambda:us-east-2:123456789012:function:example-function/invocations",
              "passthroughBehavior": "when_no_templates",
              "payloadFormatVersion" : "1.0"
            }
        }
    }
}
```

x-amazon-apigateway-integrationObjet .RequestTemplates

Spécifie les modèles de mappage pour la charge utile des demandes des types MIME spécifiés.

Propriétés

Nom de la propriété	Type	Description
MIME type	`string`	est un exemple de type MIME `application/json` . Pour plus d'informations sur la création d'un modèle de mappage, consultez PetStore modèle de mappage.

x-amazon-apigateway-integrationExemple de fichier .requestTemplates

L'exemple suivant définit les modèles de mappage pour la charge utile des demandes de types MIME `application/json` et `application/xml`.

```
"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-integrationObjet .RequestParameters

Pour les API REST, spécifie les mappages des paramètres de demande de méthode nommés aux paramètres de demande d'intégration. Les paramètres de demande de méthode doivent être définis avant d'être référencés.

Pour les API HTTP, spécifie les paramètres qui sont transmis aux intégrations `AWS_PROXY` avec un paramètre `integrationSubtype` spécifié.

Propriétés

Nom de la propriété	Type	Description
`integration.request. <param-type> .<param-name>`	`string`	Pour les API REST, la valeur est généralement un paramètre de demande de méthode prédéfini au format `method.request. <param-type> .<param-name>` , où `<param-type>` peut avoir la valeur `querystring` , `path`, `header` ou `body`. Toutefois , `$context. VARIABLE_NAME` , `$stageVariables. VARIABLE_NAME` et `STATIC_VALUE` sont également valides. Pour le paramètre `body`, la valeur `<param-name>` est une expression de chemin JSON sans le préfixe `$.`.

Nom de la propriété	Type	Description
parameter	string	Pour les API HTTP, les paramètres de demande sont une carte clé-valeur spécifiant les paramètres qui sont transmis aux intégrations AWS_PROXY avec un paramètre spécifié integrationSubtype . Vous pouvez fournir des valeurs statiques ou des données de demande de carte, des variables intermédiaires ou des variables de contexte évaluées au moment de l'exécution. Pour en savoir plus, consultez la section the section called “AWS intégrations de services”.

x-amazon-apigateway-integration.requestParametersExemple

L'exemple de mappage de paramètres de demande suivant convertit respectivement les paramètres de requête (version), d'en-tête (x-user-id) et de chemin (service) d'une demande de méthode en paramètres de requête (stage), d'en-tête (x-userid) et de chemin (op) de la demande d'intégration.

> **Note**
>
> Si vous créez des ressources via OpenAPI AWS CloudFormation, les valeurs statiques doivent être placées entre guillemets simples.
> Pour ajouter cette valeur à partir de la console, saisissez application/json dans la zone, sans guillemets.

```
"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-integrationobjet .responses

Définit les réponses de la méthode et spécifie les mappages de paramètres ou les mappages de charge utile des réponses d'intégration aux réponses de méthode.

Propriétés

Nom de la propriété	Type	Description
`Modèle d'état de réponse`	x-amazon-apigateway-integrationobjet .response	Expression régulière utilisée pour faire correspondre la réponse d'intégration à la réponse de méthode, ou `default` pour capturer toute réponse que vous n'avez pas configurée. Pour les intégrations HTTP, l'expression régulière s'applique au code d'état de la réponse d'intégration. Pour les invocations Lambda, l'expression régulière s'applique au `errorMessage` champ de l'objet d'information d'erreur renvoyé par en AWS Lambda tant que corps de réponse en cas de défaillance lorsque l'exécution de la fonction Lambda génère une exception.

Nom de la propriété	Type	Description
		Note Le nom de propriété *Modèle d'état de réponse* fait référence à un code d'état de réponse ou une expression régulière qui décrit un groupe de codes d'état de réponse. Il ne correspond à aucun identifiant de IntegrationResponseressource dans l'API REST API Gateway.

`x-amazon-apigateway-integration.responses`Exemple

L'exemple suivant montre une liste de réponses obtenues à partir des réponses `2xx` et `302`. Pour la réponse `2xx`, la réponse de méthode est mappée à partir de la charge utile de la réponse d'intégration de type MIME `application/json` ou `application/xml`. Cette réponse utilise les modèles de mappage fournis. Pour la réponse `302`, la réponse de méthode renvoie un en-tête `Location` dont la valeur est dérivée de la propriété `redirect.url` de la charge utile de la réponse d'intégration.

```
"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-integrationobjet .response

Définit une réponse et spécifie les mappages de paramètres ou les mappages de charge utile de la réponse d'intégration à la réponse de méthode.

Propriétés

Nom de la propriété	Type	Description
`statusCode`	`string`	Code d'état HTTP de la réponse de méthode, par exemple, `"200"`. Cette valeur doit correspondre à une réponse correspondante dans le champ `responses` de l'objet opération OpenAPI.
`responseTemplates`	x-amazon-apigateway-integrationObjet .Response Templates	Spécifie les modèles de mappage spécifiques au type MIME pour la charge utile de la réponse.
`responseParameters`	x-amazon-apigateway-integrationObjet .Response Parameters	Spécifie les mappages de paramètres pour la réponse. Seuls les paramètres `header` et `body` de la réponse d'intégration peuvent être

Nom de la propriété	Type	Description
		mappés aux paramètres `header` de la méthode.
`contentHandling`	`string`	Types de conversion de l'encodage des charges utiles de réponse. Les valeurs valides sont 1) `CONVERT_TO_TEXT` , pour la conversion d'une charge utile binaire en chaîne encodée en base64 ou pour la conversion d'une charge utile de texte en chaîne encodée en `utf-8` ou pour le passage de la charge utile de texte en mode natif sans modification, et 2) `CONVERT_TO_BINARY` , pour la conversion d'une charge utile de texte en bloc encodé en base64 ou le passage par une charge utile binaire en mode natif sans modification.

x-amazon-apigateway-integration.response Exemple

L'exemple suivant définit une réponse `302` pour la méthode qui dérive une charge utile du type MIME `application/json` ou `application/xml` du backend. La réponse utilise les modèles de mappage fournis et renvoie l'URL de redirection à partir de la réponse d'intégration dans l'en-tête `Location` de la méthode.

```
{
    "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-integrationObjet .ResponseTemplates

Spécifie les modèles de mappage pour la charge utile des réponses des types MIME spécifiés.

Propriétés

Nom de la propriété	Type	Description
MIME type	`string`	Spécifie un modèle de mappage pour transformer le corps de réponse d'intégration en corps de réponse de méthode pour un type MIME donné. Pour plus d'informations sur la création d'un modèle de mappage, consultez PetStore modèle de mappage. `application/json` est un exemple de *type MIME*.

x-amazon-apigateway-integrationExemple de fichier .responseTemplate

L'exemple suivant définit les modèles de mappage pour la charge utile des demandes de types MIME `application/json` et `application/xml`.

```
"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-integrationObjet .ResponseParameters

Spécifie les mappages des paramètres de réponse d'intégration aux paramètres de réponse de méthode. Vous pouvez cartographier `header`, `body`, ou des valeurs statiques au type `header` de la réponse de la méthode. Prise en charge uniquement pour les API REST.

Propriétés

Nom de la propriété	Type	Description
`method.response.header. <param-name>`	`string`	La valeur du paramètre nommé peut être dérivée de types d'intégration `header` et `body` uniquement.

`x-amazon-apigateway-integration.responseParameters` exemple

L'exemple suivant mappe les paramètres `body` et `header` de la réponse d'intégration aux deux paramètres `header` de la réponse de méthode.

```
"responseParameters" : {
    "method.response.header.Location" : "integration.response.body.redirect.url",
    "method.response.header.x-user-id" : "integration.response.header.x-userid"
}
```

x-amazon-apigateway-integrationObjet .TLSConfig

Spécifie la configuration TLS pour une intégration.

Propriétés

Nom de la propriété	Type	Description
`insecureSkipVerification`	`Boolean`	Prise en charge uniquement pour les API REST. Spécifie si API Gateway vérifie ou non si le certificat d'un point de terminaison d'intégration est émis par une autorité de certification prise en charge. Ceci n'est pas recommandé, mais il vous permet d'utiliser des certificats signés par des autorités de certification privées ou des certificats auto-signés. Si cette option est activée, API Gateway effectue toujours la validation de base du certificat, ce qui inclut la vérification de la date d'expiration du certificat, du nom d'hôte et de la présence d'une autorité de certification racine. Le certificat racine appartenant à l'autorité privée doit satisfaire les contraintes suivantes : • L'extension x509 `keyUsage` doit avoir `keyCertSign` . • L'extension x509 `basicConstraints` doit avoir `CA:TRUE`.

Nom de la propriété	Type	Description
		Prise en charge uniquement pour les intégrations `HTTP` et `HTTP_PROXY` . **Warning** L'activation de `insecureSkipVerification` n'est pas recommandée, en particulier pour les intégrations avec des points de terminaison HTTPS publics. Si vous l'activez`insecureSkipVerification` , vous augmentez le risque d' man-in-the-middle attaques.

Nom de la propriété	Type	Description
`serverNameToVerify`	`string`	Prise en charge uniquement pour les intégrations privées de l'API HTTP. Si vous spécifiez un nom de serveur, API Gateway l'utilise pour vérifier le nom d'hôte sur le certificat de l'intégration. Le nom du serveur est également inclus dans la poignée de main TLS pour prendre en charge l'indication de nom de serveur (SNI) ou l'héberge ment virtuel.

x-amazon-apigateway-integrationExemples de fichiers .tlsConfig

L'exemple OpenAPI 3.0 suivant active `insecureSkipVerification` pour une intégration de proxy HTTP d'API REST.

```
"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",
}
```

L'exemple OpenAPI 3.0 suivant spécifie un élément `serverNameToVerify` pour une intégration privée d'API HTTP.

```
"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-taille de compression

Spécifie la taille de compression minimale pour une API REST. Pour activer la compression, spécifiez un entier compris entre 0 et 10 485 760. Pour en savoir plus, consultez la section Activation de la compression de la charge utile pour une API.

x-amazon-apigateway-minimum-exemple de taille de compression

L'exemple suivant spécifie une taille de compression minimale de `5242880` octets pour une API REST.

```
"x-amazon-apigateway-minimum-compression-size": 5242880
```

x-amazon-apigateway-policy

Spécifie une stratégie de ressources pour une API REST. Pour de plus amples informations sur les stratégies de ressources, veuillez consulter Contrôle d'accès à une API avec des stratégies de ressources API Gateway. Pour obtenir des exemples de stratégie de ressources, veuillez consulter Exemples de stratégies de ressources API Gateway.

`x-amazon-apigateway-policy`Exemple

L'exemple suivant spécifie une stratégie de ressources pour une API REST. La stratégie de ressources refuse (bloque) le trafic entrant vers une API en provenance d'un bloc d'adresses IP source spécifié. Lors de l'importation, `"execute-api:/*"` est converti en`arn:aws:execute-`

`api:region:account-id:api-id/*`, en utilisant la région actuelle, votre ID de AWS compte et l'ID d'API REST actuel.

```
"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-requestpropriété -validator

Indique un valideur de demande, en référençant un *request_validator_name* de la mappe x-amazon-apigateway-requestobjet -validators, pour permettre la validation de demande sur l'API contenant ou une méthode. La valeur de cette extension est une chaîne JSON.

Cette extension peut être spécifiée au niveau de l'API ou au niveau de la méthode. Le valideur au niveau de l'API s'applique à l'ensemble des méthodes sauf s'il est remplacé par le valideur au niveau de la méthode.

x-amazon-apigateway-request-validator Exemple

L'exemple suivant applique le valideur de demande `basic` au niveau de l'API tout en appliquant le valideur de demande `parameter-only` sur la demande `POST /validation`.

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-requestobjet -validators

Définir les valideurs de demande pris en charge pour l'API contenant en tant que mappe entre un nom de valideur et les règles de validation de demande associées. Cette extension s'applique à une API REST.

Propriétés

Nom de la propriété	Type	Description
request_validator_name	x-amazon-apigateway-request objet -Validators.RequestValidator	Spécifie les règles de validation composées du valideur nommé. Exemples : `"basic" : { "validateRequestBody" : true, "validateRequestParameters" : true },` Pour appliquer ce valideur à une méthode spécifique, référencez le nom de valideur (`basic`) comme valeur de la propriété x-amazon-apigateway-requestpropriété -validator.

`x-amazon-apigateway-request-validators`Exemple

L'exemple ci-après illustre un ensemble de valideurs de demande pris en charge pour une API contenant en tant que mappe entre un nom de valideur et les règles de validation de demande associées.

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-requestobjet -Validators.RequestValidator

Spécifie les règles de validation d'un valideur de demande dans le cadre de la définition de mappe [x-amazon-apigateway-requestobjet -validators](#).

Propriétés

Nom de la propriété	Type	Description
`validateRequestBody`	`Boolean`	Spécifie s'il faut oui (`true`) ou non (`false`) valider le corps de la demande.
`validateRequestParameters`	`Boolean`	Spécifie s'il faut oui (`true`) ou non (`false`) valider les paramètres de demande requis.

x-amazon-apigateway-request-validators.requestValidator Exemple

L'exemple suivant illustre un valideur de demande parameter-only :

```
"params-only": {
    "validateRequestBody" : false,
    "validateRequestParameters" : true
}
```

x-amazon-apigateway-tagpropriété -value

Spécifie la valeur d'une AWS balise pour une API HTTP. Vous pouvez utiliser cette `x-amazon-apigateway-tag-value` propriété dans le cadre de l'objet de balise OpenAPI au niveau de la racine pour spécifier les AWS balises d'une API HTTP. Si vous spécifiez un nom de balise sans la propriété `x-amazon-apigateway-tag-value`, API Gateway crée une balise avec une chaîne vide pour une valeur.

Pour de plus amples informations sur le balisage, veuillez consulter Ajout de balises à vos ressources API Gateway.

Propriétés

Nom de la propriété	Type	Description
`name`	`String`	Spécifie la clé de balise.
`x-amazon-apigateway-tag-value`	`String`	Spécifie la valeur de la balise.

`x-amazon-apigateway-tag-value`Exemple

L'exemple suivant spécifie deux balises pour une API HTTP :

- « Propriétaire » : « Admin »
- « Production » : ""

```
"tags": [
    {
      "name": "Owner",
      "x-amazon-apigateway-tag-value": "Admin"
    },
    {
      "name": "Prod"
    }
]
```

Sécurité dans Amazon API Gateway

La sécurité du cloud AWS est la priorité absolue. En tant que AWS client, vous bénéficiez d'un centre de données et d'une architecture réseau conçus pour répondre aux exigences des entreprises les plus sensibles en matière de sécurité.

La sécurité est une responsabilité partagée entre vous AWS et vous. Le modèle de responsabilité partagée décrit ce concept comme la sécurité du cloud et la sécurité dans le cloud :

- Sécurité du cloud : AWS est chargée de protéger l'infrastructure qui exécute les AWS services dans le AWS cloud. AWS vous fournit également des services que vous pouvez utiliser en toute sécurité. Des auditeurs tiers testent et vérifient régulièrement l'efficacité de notre sécurité dans le cadre des programmes de AWS conformité Programmes de de conformité. Pour en savoir plus sur les programmes de conformité qui s'appliquent à Amazon API Gateway, consultez la section AWS services concernés par programme de conformité et .
- Sécurité dans le cloud — Votre responsabilité est déterminée par le AWS service que vous utilisez. Vous êtes également responsable d'autres facteurs, y compris de la sensibilité de vos données, des exigences de votre entreprise, ainsi que de la législation et de la réglementation applicables.

Cette documentation vous aide à comprendre comment appliquer le modèle de responsabilité partagée lorsque vous utilisez API Gateway. Les rubriques suivantes vous montrent comment configurer API Gateway pour répondre à vos objectifs de sécurité et de conformité. Vous apprendrez également à utiliser d'autres AWS services qui vous aident à surveiller et à sécuriser vos ressources API Gateway.

Pour plus d'informations, veuillez consulter Présentation de la sécurité dans Amazon API Gateway.

Rubriques

- Bonnes pratiques de sécurité dans Amazon API Gateway

Protection des données dans Amazon API Gateway

Le modèle de responsabilité partagée AWS s'applique à la protection des données dans Amazon API Gateway. Comme décrit dans ce modèle, AWS est responsable de la protection de l'infrastructure globale sur laquelle l'ensemble de AWS Cloud s'exécute. La gestion du contrôle de votre contenu hébergé sur cette infrastructure relève de votre responsabilité. Vous êtes également responsable des tâches de configuration et de gestion de la sécurité pour les Services AWS que vous utilisez. Pour en savoir plus sur la confidentialité des données, consultez Questions fréquentes (FAQ) sur la confidentialité des données. Pour en savoir plus sur la protection des données en Europe, consultez le billet de blog Modèle de responsabilité partagée AWSet RGPD (Règlement général sur la protection des données) sur le AWSBlog de sécurité.

À des fins de protection des données, nous vous recommandons de protéger les informations d'identification Compte AWS et de configurer les comptes utilisateur individuels avec AWS IAM Identity Center ou AWS Identity and Access Management (IAM). Ainsi, chaque utilisateur se voit attribuer uniquement les autorisations nécessaires pour exécuter ses tâches. Nous vous recommandons également de sécuriser vos données comme indiqué ci-dessous :

- Utilisez l'authentification multifactorielle (MFA) avec chaque compte.
- Utilisez les certificats SSL/TLS pour communiquer avec les ressources AWS. Nous exigeons TLS 1.2 et recommandons TLS 1.3.
- Configurez une API (Interface de programmation) et le journal de l'activité des utilisateurs avec AWS CloudTrail.
- Utilisez des solutions de chiffrement AWS, ainsi que tous les contrôles de sécurité par défaut au sein des Services AWS.
- Utilisez des services de sécurité gérés avancés tels qu'Amazon Macie, qui contribuent à la découverte et à la sécurisation des données sensibles stockées dans Amazon S3.
- Si vous avez besoin de modules cryptographiques validés FIPS (Federal Information Processing Standard) 140-2 lorsque vous accédez à AWS via une CLI (Interface de ligne de commande) ou une API (Interface de programmation), utilisez un point de terminaison FIPS (Federal Information Processing Standard). Pour en savoir plus sur les points de terminaison FIPS (Federal Information Processing Standard) disponibles, consultez Federal Information Processing Standard (FIPS) 140-2 (Normes de traitement de l'information fédérale).

Nous vous recommandons fortement de ne jamais placer d'informations confidentielles ou sensibles, telles que les adresses e-mail de vos clients, dans des balises ou des champs de texte libre tels que le champ Name (Nom). Cela s'applique lorsque vous utilisez API Gateway ou d'autres Services AWS avec la console, l'API, l'AWS CLI ou les kits SDK AWS. Toutes les données que vous saisissez dans des balises ou des champs de texte de forme libre utilisés pour les noms peuvent être utilisées à des fins de facturation ou dans les journaux de diagnostic. Si vous fournissez une adresse URL à un serveur externe, nous vous recommandons fortement de ne pas inclure d'informations d'identification dans l'adresse URL permettant de valider votre demande adressée à ce serveur.

Chiffrement des données dans Amazon API Gateway

La protection des données fait référence au fait de protéger les données pendant leur transit (lorsqu'elles sont transmises en direction ou en provenance d'API Gateway) et au repos (lorsqu'elles sont stockées dans AWS).

Chiffrement des données au repos dans Amazon API Gateway

Si vous choisissez d'activer la mise en cache pour une API REST, vous pouvez activer le chiffrement du cache. Pour en savoir plus, consultez la section Activation de la mise en cache des API pour améliorer la réactivité.

Pour en savoir plus sur la protection des données, consultez le billet de blog Modèle de responsabilité partagée AWS et RGPD sur le Blog sur la sécurité d’AWS.

Chiffrement des données en transit dans Amazon API Gateway

Les API créées avec Amazon API Gateway présentent uniquement des points de terminaison HTTPS. API Gateway ne prend pas en charge les points de terminaison non chiffrés (HTTP).

API Gateway gère les certificats pour les points de terminaison `execute-api` par défaut. Si vous configurez un nom de domaine personnalisé, vous spécifiez le certificat correspondant. La meilleure pratique consiste à ne pas épingler les certificats.

Pour plus de sécurité, vous pouvez choisir une version de protocole TLS (Transport Layer Security) minimale à appliquer pour votre domaine personnalisé API Gateway. Les API WebSocket et les API HTTP ne prennent en charge que TLS 1.2. Pour en savoir plus, consultez la section Choix d'une politique de sécurité pour votre domaine personnalisé dans API Gateway.

Vous pouvez également configurer une distribution Amazon CloudFront avec un certificat SSL personnalisé dans votre compte et l'utiliser avec des API régionales. Vous pouvez ensuite configurer

la stratégie de sécurité pour la distribution CloudFront avec TLS 1.1 ou supérieur, en fonction de vos exigences de sécurité et de conformité.

Pour de plus amples informations sur la protection des données, veuillez consulter Protection de votre API REST et la publication de blog Responsabilité partagée AWS et RGPD sur le Blog de sécurité AWS.

Confidentialité du trafic inter-réseau

À l'aide d'Amazon API Gateway, vous pouvez créer des API REST privées accessibles uniquement à partir de votre Amazon Virtual Private Cloud (VPC). Le VPC utilise un point de terminaison VPC d'interface, qui est une interface réseau de terminaison que vous créez dans votre VPC. À l'aide des stratégies de ressources, vous pouvez accorder ou refuser l'accès à votre API depuis des VPC et points de terminaison de VPC sélectionnés, y compris entre plusieurs comptes AWS. Chaque point de terminaison peut être utilisé pour accéder à plusieurs API privées. Vous pouvez également utiliser AWS Direct Connect pour établir une connexion à partir d'un réseau sur site à Amazon VPC et pour accéder à votre API privée avec cette connexion. Dans tous les cas, le trafic vers votre API privée utilise des connexions sécurisées et ne quitte pas le réseau Amazon : il est isolé de l'Internet public. Pour en savoir plus, consultez la section the section called “API REST privées”.

Identity and Access Management pour Amazon API Gateway

AWS Identity and Access Management (IAM) est un Service AWS qui aide un administrateur à contrôler en toute sécurité l'accès aux ressources AWS. Des administrateurs IAM contrôlent les personnes qui s'authentifient (sont connectées) et sont autorisées (disposent d'autorisations) à utiliser des ressources API Gateway. IAM est un Service AWS que vous pouvez utiliser sans frais supplémentaires.

Rubriques

- Utilisation de rôles liés au service pour API Gateway

Public ciblé

Votre utilisation d'AWS Identity and Access Management (IAM) évolue selon la tâche que vous réalisez dans API Gateway.

Utilisateur du service - Si vous utilisez le service API Gateway pour effectuer votre tâche, votre administrateur vous fournit les informations d'identification et les autorisations dont vous avez besoin. Plus vous utiliserez de fonctions API Gateway pour effectuer votre tâche, plus vous pourrez avoir besoin d'autorisations supplémentaires. Comprendre la gestion des accès peut vous aider à demander à votre administrateur les autorisations appropriées. Si vous ne pouvez pas accéder à une fonction dans API Gateway, veuillez consulter Résolution des problèmes d'identité et d'accès à Amazon API Gateway.

Administrateur du service - Si vous êtes le responsable des ressources API Gateway de votre entreprise, vous bénéficiez probablement d'un accès total à API Gateway. C'est à vous de déterminer les fonctions et les ressources API Gateway auxquelles vos services pourront accéder. Vous devez ensuite soumettre les demandes à votre administrateur IAM pour modifier les autorisations des utilisateurs de votre service. Consultez les informations sur cette page pour comprendre les concepts de base d'IAM. Pour en savoir plus sur la façon dont votre entreprise peut utiliser IAM avec API Gateway, veuillez consulter Fonctionnement d'Amazon API Gateway avec IAM.

Administrateur IAM - Si vous êtes un administrateur IAM, vous souhaiterez peut-être obtenir des détails sur la façon dont vous pouvez écrire des stratégies pour gérer l'accès à API Gateway. Pour voir des exemples de stratégies API Gateway basées sur l'identité que vous pouvez utiliser dans IAM, veuillez consulter Exemples de stratégie basée sur l'identité d'Amazon API Gateway.

Authentification avec des identités

L'authentification correspond au processus par lequel vous vous connectez à AWS avec vos informations d'identification. Vous devez vous authentifier (être connecté à AWS) en tant qu'Utilisateur racine d'un compte AWS, en tant qu'utilisateur IAM ou en endossant un rôle IAM.

Vous pouvez vous connecter à AWS en tant qu'identité fédérée à l'aide des informations d'identification fournies par le biais d'une source d'identité. AWS IAM Identity Center Les utilisateurs (IAM Identity Center), l'authentification de connexion unique de votre entreprise et vos informations d'identification Google ou Facebook sont des exemples d'identités fédérées. Lorsque vous vous connectez avec une identité fédérée, votre administrateur aura précédemment configuré une

fédération d'identités avec des rôles IAM. Lorsque vous accédez à AWS en utilisant la fédération, vous endossez indirectement un rôle.

Selon le type d'utilisateur que vous êtes, vous pouvez vous connecter à la AWS Management Console ou au portail d'accès AWS. Pour plus d'informations sur la connexion à AWS, consultez Connexion à votre Compte AWSdans le Guide de l'utilisateurConnexion à AWS.

Si vous accédez à AWS par programmation, AWS fournit un kit de développement logiciel (SDK) et une interface de ligne de commande (CLI) pour signer cryptographiquement vos demandes en utilisant vos informations d'identification. Si vous n'utilisez pas les outils AWS, vous devez signer les requêtes vous-même. Pour plus d'informations sur l'utilisation de la méthode recommandée pour signer des demandes vous-même, consultez Signature des demandes d'API AWS dans le Guide de l'utilisateur IAM.

Quelle que soit la méthode d'authentification que vous utilisez, vous devrez peut-être fournir des informations de sécurité supplémentaires. Par exemple, AWS vous recommande d'utiliser l'authentification multifactorielle (MFA) pour améliorer la sécurité de votre compte. Pour en savoir plus, veuillez consulter Multi-factor authentication (Authentification multifactorielle) dans le Guide de l'utilisateur AWS IAM Identity Center et Utilisation de l'authentification multifactorielle (MFA) dans l'interface AWS dans le Guide de l'utilisateur IAM.

Utilisateur root Compte AWS

Lorsque vous créez un Compte AWS, vous commencez avec une seule identité de connexion disposant d'un accès complet à tous les Services AWS et ressources du compte. Cette identité est appelée utilisateur root du Compte AWS. Vous pouvez y accéder en vous connectant à l'aide de l'adresse électronique et du mot de passe que vous avez utilisés pour créer le compte. Il est vivement recommandé de ne pas utiliser l'utilisateur root pour vos tâches quotidiennes. Protégez vos informations d'identification d'utilisateur root et utilisez-les pour effectuer les tâches que seul l'utilisateur root peut effectuer. Pour obtenir la liste complète des tâches qui vous imposent de vous connecter en tant qu'utilisateur root, consultez Tâches nécessitant des informations d'identification d'utilisateur root dans le Guide de l'utilisateur IAM.

Utilisateurs et groupes IAM

Un utilisateur IAM est une identité dans votre Compte AWS qui dispose d'autorisations spécifiques pour une seule personne ou application. Dans la mesure du possible, nous vous recommandons de vous appuyer sur des informations d'identification temporaires plutôt que de créer des utilisateurs IAM ayant des informations d'identification à long terme tels que les clés d'accès. Toutefois, si

certains cas d'utilisation spécifiques nécessitent des informations d'identification à long terme avec les utilisateurs IAM, nous vous recommandons de faire pivoter les clés d'accès. Pour plus d'informations, consultez Rotation régulière des clés d'accès pour les cas d'utilisation nécessitant des informations d'identification dans le Guide de l'utilisateur IAM.

Un groupe IAM est une identité qui concerne un ensemble d'utilisateurs IAM. Vous ne pouvez pas vous connecter en tant que groupe. Vous pouvez utiliser les groupes pour spécifier des autorisations pour plusieurs utilisateurs à la fois. Les groupes permettent de gérer plus facilement les autorisations pour de grands ensembles d'utilisateurs. Par exemple, vous pouvez avoir un groupe nommé IAMAdmins et accorder à ce groupe les autorisations d'administrer des ressources IAM.

Les utilisateurs sont différents des rôles. Un utilisateur est associé de manière unique à une personne ou une application, alors qu'un rôle est conçu pour être endossé par tout utilisateur qui en a besoin. Les utilisateurs disposent d'informations d'identification permanentes, mais les rôles fournissent des informations d'identification temporaires. Pour en savoir plus, consultez Quand créer un utilisateur IAM (au lieu d'un rôle) dans le Guide de l'utilisateur IAM.

Rôles IAM

Un rôle IAM est une entité au sein de votre Compte AWS qui dispose d'autorisations spécifiques. Le concept ressemble à celui d'utilisateur IAM, mais le rôle IAM n'est pas associé à une personne en particulier. Vous pouvez temporairement endosser un rôle IAM dans la AWS Management Console en changeant de rôle. Vous pouvez obtenir un rôle en appelant une opération d'API AWS CLI ou AWS à l'aide d'une URL personnalisée. Pour plus d'informations sur les méthodes d'utilisation des rôles, consultez Utilisation de rôles IAM dans le Guide de l'utilisateur IAM.

Les rôles IAM avec des informations d'identification temporaires sont utiles dans les cas suivants :

- Accès utilisateur fédéré : pour attribuer des autorisations à une identité fédérée, vous créez un rôle et définissez des autorisations pour le rôle. Quand une identité externe s'authentifie, l'identité est associée au rôle et reçoit les autorisations qui sont définies par celui-ci. Pour obtenir des informations sur les rôles pour la fédération, consultez Création d'un rôle pour un fournisseur d'identité tiers (fédération) dans le Guide de l'utilisateur IAM. Si vous utilisez IAM Identity Center, vous configurez un jeu d'autorisations. IAM Identity Center met en corrélation le jeu d'autorisations avec un rôle dans IAM afin de contrôler à quoi vos identités peuvent accéder après leur authentification. Pour plus d'informations sur les jeux d'autorisations, veuillez consulter la rubrique Jeux d'autorisations dans le Guide de l'utilisateur AWS IAM Identity Center.
- Autorisations d'utilisateur IAM temporaires : un rôle ou un utilisateur IAM peut endosser un rôle IAM pour profiter temporairement d'autorisations différentes pour une tâche spécifique.

- Accès intercompte : vous pouvez utiliser un rôle IAM pour permettre à un utilisateur (principal de confiance) d'un compte différent d'accéder aux ressources de votre compte. Les rôles constituent le principal moyen d'accorder l'accès intercompte. Toutefois, certains Services AWS vous permettent d'attacher une politique directement à une ressource (au lieu d'utiliser un rôle en tant que proxy). Pour en savoir plus sur la différence entre les rôles et les politiques basées sur les ressources pour l'accès intercompte, consultez Différence entre les rôles IAM et les politiques basées sur les ressources dans le Guide de l'utilisateur IAM.
- Accès interservices : certains Services AWS utilisent des fonctions dans d'autres Services AWS. Par exemple, lorsque vous effectuez un appel dans un service, il est courant que ce service exécute des applications dans Amazon EC2 ou stocke des objets dans Amazon S3. Un service peut le faire en utilisant les autorisations d'appel du principal, une fonction de service ou un rôle lié au service.
  - Sessions de transmission d'accès (FAS) : lorsque vous utilisez un utilisateur ou un rôle IAM pour effectuer des actions dans AWS, vous êtes considéré comme un principal. Lorsque vous utilisez certains services, vous pouvez effectuer une action qui initie une autre action dans un autre service. FAS utilise les autorisations du principal qui appelle un Service AWS, combinées au Service AWS qui demande pour effectuer des demandes aux services en aval. Les demandes FAS ne sont faites que lorsqu'un service reçoit une demande dont l'exécution nécessite des interactions avec d'autres Services AWS ou ressources. Dans ce cas, vous devez disposer d'autorisations nécessaires pour effectuer les deux actions. Pour plus de détails sur les politiques relatives à l'envoi de demandes FAS, consultez Transférer les sessions d'accès.
  - Fonction du service : il s'agit d'un rôle IAM attribué à un service afin de réaliser des actions en votre nom. Un administrateur IAM peut créer, modifier et supprimer une fonction du service à partir d'IAM. Pour plus d'informations, consultez Création d'un rôle pour la délégation d'autorisations à un Service AWS dans le Guide de l'utilisateur IAM.
  - Rôle lié au service : un rôle lié au service est un type de fonction du service lié à un Service AWS. Le service peut endosser le rôle afin d'effectuer une action en votre nom. Les rôles liés à un service s'affichent dans votre Compte AWS et sont détenus par le service. Un administrateur IAM peut consulter, mais ne peut pas modifier, les autorisations concernant les rôles liés à un service.
- Applications s'exécutant sur Amazon EC2 : vous pouvez utiliser un rôle IAM pour gérer des informations d'identification temporaires pour les applications s'exécutant sur une instance EC2 et effectuant des demandes d'API AWS CLI ou AWS. Cette solution est préférable au stockage des clés d'accès au sein de l'instance EC2. Pour attribuer un rôle AWS à une instance EC2 et le rendre disponible à toutes les applications associées, vous pouvez créer un profil d'instance

attaché à l'instance. Un profil d'instance contient le rôle et permet aux programmes qui s'exécutent sur l'instance EC2 d'obtenir des informations d'identification temporaires. Pour plus d'informations, consultez Utilisation d'un rôle IAM pour accorder des autorisations à des applications s'exécutant sur des instances Amazon EC2 dans le Guide de l'utilisateur IAM.

Pour savoir dans quel cas utiliser des rôles ou des utilisateurs IAM, consultez Quand créer un rôle IAM (au lieu d'un utilisateur) dans le Guide de l'utilisateur IAM.

Gestion des accès à l'aide de politiques

Vous contrôlez les accès dans AWS en créant des politiques et en les attachant à des identités AWS ou à des ressources. Une politique est un objet dans AWS qui, lorsqu'il est associé à une identité ou à une ressource, définit les autorisations de ces dernières. AWS évalue ces politiques lorsqu'un principal (utilisateur, utilisateur racine ou séance de rôle) envoie une demande. Les autorisations dans les politiques déterminent si la demande est autorisée ou refusée. La plupart des politiques sont stockées dans AWS en tant que documents JSON. Pour plus d'informations sur la structure et le contenu des documents de politique JSON, consultez Présentation des politiques JSON dans le Guide de l'utilisateur IAM.

Les administrateurs peuvent utiliser les politiques JSON AWS pour spécifier qui a accès à quoi. C'est-à-dire, quel principal peut effectuer des actions sur quelles ressources et dans quelles conditions.

Par défaut, les utilisateurs et les rôles ne disposent d'aucune autorisation. Pour octroyer aux utilisateurs des autorisations d'effectuer des actions sur les ressources dont ils ont besoin, un administrateur IAM peut créer des politiques IAM. L'administrateur peut ensuite ajouter les politiques IAM aux rôles et les utilisateurs peuvent assumer les rôles.

Les politiques IAM définissent les autorisations d'une action, quelle que soit la méthode que vous utilisez pour exécuter l'opération. Par exemple, supposons que vous disposiez d'une politique qui autorise l'action `iam:GetRole`. Un utilisateur avec cette politique peut obtenir des informations utilisateur à partir de la AWS Management Console, de la AWS CLI ou de l'API AWS.

Politiques basées sur l'identité

Les politiques basées sur l'identité sont des documents de politique d'autorisations JSON que vous pouvez attacher à une identité telle qu'un utilisateur, un groupe d'utilisateurs ou un rôle IAM. Ces politiques contrôlent quel type d'actions des utilisateurs et des rôles peuvent exécuter, sur quelles

ressources et dans quelles conditions. Pour découvrir comment créer une politique basée sur l'identité, consultez Création de politiques IAM dans le Guide de l'utilisateur IAM.

Les politiques basées sur l'identité peuvent être classées comme des politiques en ligne ou des politiques gérées. Les politiques en ligne sont intégrées directement à un utilisateur, groupe ou rôle. Les politiques gérées sont des politiques autonomes que vous pouvez attacher à plusieurs utilisateurs, groupes et rôles dans votre Compte AWS. Les politiques gérées incluent les politiques gérées par AWS et les politiques gérées par le client. Pour découvrir comment choisir entre une politique gérée et une politique en ligne, consultez Choix entre les politiques gérées et les politiques en ligne dans le Guide de l'utilisateur IAM.

politiques basées sur les ressources

Les politiques basées sur les ressources sont des documents de politique JSON que vous attachez à une ressource. Des politiques basées sur les ressources sont, par exemple, les politiques de confiance de rôle IAM et des politiques de compartiment Amazon S3. Dans les services qui sont compatibles avec les politiques basées sur les ressources, les administrateurs de service peuvent les utiliser pour contrôler l'accès à une ressource spécifique. Pour la ressource dans laquelle se trouve la politique, cette dernière définit quel type d'actions un principal spécifié peut effectuer sur cette ressource et dans quelles conditions. Vous devez spécifier un principal dans une politique basée sur les ressources. Les principaux peuvent inclure des comptes, des utilisateurs, des rôles, des utilisateurs fédérés ou des Services AWS.

Les politiques basées sur les ressources sont des politiques en ligne situées dans ce service. Vous ne pouvez pas utiliser les politiques gérées AWS depuis IAM dans une politique basée sur une ressource.

Listes de contrôle d'accès (ACL)

Les listes de contrôle d'accès (ACL) vérifie quels principaux (membres de compte, utilisateurs ou rôles) ont l'autorisation d'accéder à une ressource. Les listes de contrôle d'accès sont similaires aux politiques basées sur les ressources, bien qu'elles n'utilisent pas le format de document de politique JSON.

Amazon S3, AWS WAF et Amazon VPC sont des exemples de services prenant en charge les ACL. Pour en savoir plus sur les listes de contrôle d'accès, consultez Présentation des listes de contrôle d'accès (ACL) dans le Guide du développeur Amazon Simple Storage Service.

Autres types de politique

AWS prend en charge d'autres types de politiques moins courantes. Ces types de politiques peuvent définir le nombre maximum d'autorisations qui vous sont accordées par des types de politiques plus courants.

- Limite d'autorisations : une limite d'autorisations est une fonction avancée dans laquelle vous définissez le nombre maximal d'autorisations qu'une politique basée sur l'identité peut accorder à une entité IAM (utilisateur ou rôle IAM). Vous pouvez définir une limite d'autorisations pour une entité. Les autorisations qui en résultent représentent la combinaison des politiques basées sur l'identité d'une entité et de ses limites d'autorisation. Les politiques basées sur les ressources qui spécifient l'utilisateur ou le rôle dans le champ `Principal` ne sont pas limitées par les limites d'autorisations. Un refus explicite dans l'une de ces politiques remplace l'autorisation. Pour plus d'informations sur les limites d'autorisations, consultez Limites d'autorisations pour des entités IAM dans le Guide de l'utilisateur IAM.
- Politiques de contrôle des services (SCP) - les SCP sont des politiques JSON qui spécifient le nombre maximal d'autorisations pour une organisation ou une unité d'organisation (OU) dans AWS Organizations. AWS Organizations est un service qui vous permet de regrouper et de gérer de façon centralisée plusieurs Comptes AWS détenus par votre entreprise. Si vous activez toutes les fonctions d'une organisation, vous pouvez appliquer les politiques de contrôle de service (SCP) à l'un ou à l'ensemble de vos comptes. La politique de contrôle des services limite les autorisations pour les entités dans les comptes membres, y compris dans chaque Utilisateur racine d'un compte AWS. Pour plus d'informations sur les organisations et les SCP, consultez Fonctionnement des SCP dans le Guide de l'utilisateur AWS Organizations.
- Politiques de séance : les politiques de séance sont des politiques avancées que vous passez en tant que paramètre lorsque vous programmez afin de créer une séance temporaire pour un rôle ou un utilisateur fédéré. Les autorisations de la séance obtenue sont une combinaison des politiques basées sur l'identité de l'utilisateur ou du rôle et des politiques de séance. Les autorisations peuvent également provenir d'une politique basée sur les ressources. Un refus explicite dans l'une de ces politiques remplace l'autorisation. Pour plus d'informations, consultez Politiques de séance dans le Guide de l'utilisateur IAM.

Plusieurs types de politique

Lorsque plusieurs types de politiques s'appliquent à la requête, les autorisations obtenues sont plus compliquées à comprendre. Pour découvrir la façon dont AWS détermine s'il convient d'autoriser une

demande en présence de plusieurs types de stratégies, veuillez consulter Logique d'évaluation de stratégies dans le Guide de l'utilisateur IAM.

Fonctionnement d'Amazon API Gateway avec IAM

Avant d'utiliser IAM pour gérer l'accès à API Gateway, vous devez comprendre les fonctions IAM disponibles avec API Gateway. Pour obtenir davantage d'informations sur la manière dont API Gateway et d'autres services AWS fonctionnent avec IAM, veuillez consulter Services AWS qui fonctionnent avec IAM dans le Guide de l'utilisateur IAM.

Rubriques

- Stratégies basées sur l'identité d'API Gateway
- Stratégies basées sur les ressources API Gateway
- Autorisation basée sur les balises API Gateway
- Rôles IAM API Gateway

Stratégies basées sur l'identité d'API Gateway

Avec les stratégies IAM basées sur l'identité, vous pouvez spécifier quelles actions et ressources sont autorisées ou refusées, ainsi que les conditions dans lesquelles elles le sont. API Gateway prend en charge des actions, ressources et clés de condition spécifiques. Pour plus d'informations sur les actions, ressources et clés de condition spécifiques à API Gateway, consultezActions, ressources et clés de condition pour Amazon API Gateway ManagementetActions, ressources et clés de condition pour Amazon API Gateway Management V2. Pour plus d'informations sur tous les éléments que vous utilisez dans une stratégie JSON, consultez Références des éléments de stratégie JSON IAM dans le Guide de l'utilisateur IAM.

L'exemple suivant illustre une stratégie basée sur l'identité qui permet à un utilisateur de créer ou de mettre à jour des API REST privées uniquement. Pour obtenir plus d'exemples, consultez the section called “Exemples de stratégies basées sur l'identité”.

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

Actions

L'élément `Action` d'une stratégie JSON décrit les actions que vous pouvez utiliser pour autoriser ou refuser l'accès à une stratégie.

Les actions de stratégie dans API Gateway utilisent le préfixe suivant avant l'action : `apigateway:`. Les déclarations de stratégie doivent inclure un élément `Action` ou `NotAction`. API Gateway définit son propre ensemble d'actions qui décrivent les tâches que vous pouvez effectuer avec ce service.

L'expression `Action` de gestion des API est au format `apigateway:`*`action`*, où *`action`* est l'une des actions API Gateway suivantes : GET, POST, PUT, DELETE, PATCH (pour mettre à jour les ressources), ou *, qui correspond à l'ensemble des actions précédentes.

Voici quelques exemples d'expression `Action` :

- **`apigateway:*`** pour toutes les actions API Gateway.
- **`apigateway:GET`** pour l'action GET uniquement dans API Gateway.

Pour spécifier plusieurs actions dans une seule déclaration, séparez-les par des virgules comme suit :

```
"Action": [
      "apigateway:action1",
      "apigateway:action2"
```

Pour plus d'informations sur les verbes HTTP à utiliser pour des opérations API Gateway spécifiques, consultez Référence de l'API Amazon API Gateway Version 1 (API REST) et Référence de l'API Amazon API Gateway Version 2 (API WebSocket et HTTP).

Pour de plus amples informations, veuillez consulter the section called “Exemples de stratégies basées sur l'identité”.

Ressources

Les administrateurs peuvent utiliser les politiques JSON AWS pour spécifier qui a accès à quoi. C'est-à-dire, quel principal peut effectuer des actions sur quelles ressources et dans quelles conditions.

L'élément de politique JSON `Resource` indique le ou les objets pour lesquels l'action s'applique. Les instructions doivent inclure un élément `Resource` ou `NotResource`. Il est recommandé de définir une ressource à l'aide de son Amazon Resource Name (ARN). Vous pouvez le faire pour des actions qui prennent en charge un type de ressource spécifique, connu sous la dénomination autorisations de niveau ressource.

Pour les actions qui ne sont pas compatibles avec les autorisations de niveau ressource, telles que les opérations de liste, utilisez un caractère générique (*) afin d’indiquer que l'instruction s'applique à toutes les ressources.

```
"Resource": "*"
```

Les ressources API Gateway sont au format ARN suivant :

```
arn:aws:apigateway:region::resource-path-specifier
```

Par exemple, pour spécifier une API REST avec l'ID *api-id* et ses sous-ressources, telles que les autorisations dans votre instruction, utilisez l'ARN suivant :

```
"Resource": "arn:aws:apigateway:us-east-2::/restapis/api-id/*"
```

Pour spécifier toutes les API REST et les sous-ressources qui appartiennent à un compte spécifique, utilisez le caractère générique (*).

```
"Resource": "arn:aws:apigateway:us-east-2::/restapis/*"
```

Pour accéder à la liste des types de ressources API Gateway et de leurs ARN, consultez Référence Amazon Resource Name (ARN) API Gateway.

Clés de condition

Les administrateurs peuvent utiliser les politiques JSON AWS pour spécifier qui a accès à quoi. C'est-à-dire, quel principal peut effectuer des actions sur quelles ressources et dans quelles conditions.

L'élément `Condition` (ou le bloc `Condition`) vous permet de spécifier des conditions lorsqu'une instruction est appliquée. L'élément `Condition` est facultatif. Vous pouvez créer des expressions conditionnelles qui utilisent des opérateurs de condition, tels que les signes égal ou inférieur à, pour faire correspondre la condition de la politique aux valeurs de la demande.

Si vous spécifiez plusieurs éléments `Condition` dans une instruction, ou plusieurs clés dans un seul élément `Condition`, AWS les évalue à l'aide d'une opération `AND` logique. Si vous spécifiez plusieurs valeurs pour une seule clé de condition, AWS évalue la condition à l'aide d'une opération `OR` logique. Toutes les conditions doivent être remplies avant que les autorisations associées à l'instruction ne soient accordées.

Vous pouvez aussi utiliser des variables d'espace réservé quand vous spécifiez des conditions. Par exemple, vous pouvez accorder à un utilisateur IAM l'autorisation d'accéder à une ressource uniquement si elle est balisée avec son nom d'utilisateur IAM. Pour plus d'informations, consultez Éléments d'une politique IAM : variables et identifications dans le Guide de l'utilisateur IAM.

AWS prend en charge les clés de condition globales et les clés de condition spécifiques à un service. Pour afficher toutes les clés de condition globales AWS, consultez Clés de contexte de condition globale AWS dans le Guide de l'utilisateur IAM.

API Gateway définit son propre ensemble de clés de condition et prend également en charge l'utilisation des clés de condition globales. Pour accéder à la liste des clés de condition API Gateway, consultez Clés de condition pour Amazon API Gateway dans le Guide de l'utilisateur IAM. Pour savoir avec quelles actions et ressources vous pouvez utiliser une clé de condition, consultez Actions définies par Amazon API Gateway.

Pour plus d'informations sur l'étiquette, y compris le contrôle d'accès basé sur les attributs, consultez Balisage.

Exemples

Pour accéder à des exemples de stratégies API Gateway basées sur l'identité, consultez Exemples de stratégie basée sur l'identité d'Amazon API Gateway.

Stratégies basées sur les ressources API Gateway

Les stratégies basées sur les ressources sont des documents de stratégie JSON précisant les actions qu'un principal indiqué peut effectuer sur la ressource API Gateway et dans quelles conditions. API Gateway prend en charge les stratégies d'autorisations basées sur les ressources pour les API REST. Vous pouvez utiliser des stratégies de ressources pour déterminer qui peut appeler une API REST. Pour de plus amples informations, veuillez consulter the section called "Utilisation des stratégies de ressources API Gateway".

Exemples

Pour accéder à des exemples de stratégies API Gateway basées sur les ressources, consultez Exemples de stratégies de ressources API Gateway.

Autorisation basée sur les balises API Gateway

Vous pouvez attacher des balises aux ressources API Gateway ou transmettre des balises dans une requête à API Gateway. Pour contrôler l'accès basé sur des balises, vous devez fournir les informations des balises dans l'élément de condition d'une stratégie en utilisant les clés de condition `apigateway:ResourceTag/`*`key-name`*, `aws:RequestTag/`*`key-name`* ou `aws:TagKeys`. Pour de plus amples informations sur le balisage des ressources API Gateway, veuillez consulter the section called "Contrôle d'accès basé sur les attributs".

Pour accéder à des exemples de stratégies basées sur l'identité permettant de limiter l'accès à une ressource en fonction des balises de cette ressource, consultez Utilisation de balises pour contrôler l'accès aux ressources API REST API Gateway.

Rôles IAM API Gateway

Un rôle IAM est une entité au sein de votre compte AWS qui dispose d'autorisations spécifiques.

Utilisation des informations d'identification temporaires avec API Gateway

Vous pouvez utiliser des informations d'identification temporaires pour vous connecter à l'aide de la fédération, endosser un rôle IAM ou encore pour endosser un rôle entre comptes. Vous obtenez des informations d'identification de sécurité temporaires en appelant des opérations d'API AWS STS comme AssumeRole ou GetFederationToken.

API Gateway prend en charge l'utilisation des informations d'identification temporaires.

Rôles liés à un service

Les rôles liés à un service permettent aux services AWS d'accéder à des ressources dans d'autres services pour effectuer une action en votre nom. Les rôles liés à un service s'affichent dans votre compte IAM et sont la propriété du service. Un administrateur IAM peut consulter, mais ne peut pas modifier, les autorisations concernant les rôles liés à un service.

API Gateway prend en charge les rôles liés au service. Pour plus d'informations sur la création ou la gestion des rôles liés à un service API Gateway, consultez Utilisation de rôles liés au service pour API Gateway.

Rôles de service

Un service peut endosser un rôle de service en votre nom. Un rôle de service autorise le service à accéder à des ressources d'autres services pour effectuer une action en votre nom. Sachant que les fonctions du service apparaissent dans votre compte IAM et qu'elles sont la propriété du compte, un administrateur peut modifier les autorisations associées à ce rôle. Toutefois, une telle action peut perturber le bon fonctionnement du service.

API Gateway prend en charge les rôles de service.

Exemples de stratégie basée sur l'identité d'Amazon API Gateway

Par défaut, les utilisateurs et rôles IAM ne sont pas autorisés à créer ou modifier les ressources API Gateway. Ils ne peuvent pas non plus effectuer de tâches à l'aide de la AWS Management Console, de la AWS CLI ou des kits SDK AWS. Un administrateur IAM doit créer des stratégies IAM autorisant les utilisateurs et les rôles à exécuter des opérations d'API spécifiques sur les ressources spécifiées

dont ils ont besoin. Il doit ensuite attacher ces stratégies aux utilisateurs ou aux groupes IAM ayant besoin de ces autorisations.

Pour plus d'informations sur la création de stratégies IAM, consultez la section Création de stratégies sous l'onglet JSON du Guide de l'utilisateur IAM. Pour plus d'informations sur les actions, les ressources et les conditions spécifiques à API Gateway, voir Actions, ressources et clés de condition pour Amazon API Gateway Management et Actions, ressources et clés de condition pour Amazon API Gateway Management V2.

Rubriques

Bonnes pratiques en matière de politiques

Les stratégies basées sur l'identité déterminent si une personne peut créer, consulter ou supprimer des ressources API Gateway dans votre compte. Ces actions peuvent entraîner des frais pour votre Compte AWS. Lorsque vous créez ou modifiez des politiques basées sur l'identité, suivez ces instructions et recommandations :

- Démarrer avec AWS gérées et évoluez vers les autorisations de moindre privilège - Pour commencer à accorder des autorisations à vos utilisateurs et charges de travail, utilisez les politiques gérées AWS qui accordent des autorisations dans de nombreux cas d'utilisation courants. Elles sont disponibles dans votre Compte AWS. Nous vous recommandons de réduire encore les autorisations en définissant des Politiques gérées par le client AWS qui sont spécifiques à vos cas d'utilisation. Pour de plus amples informations, consultez Politiques gérées AWS ou Politiques gérées AWS pour les activités professionnelles dans le Guide de l'utilisateur IAM.
- Accorder les autorisations de moindre privilège - Lorsque vous définissez des autorisations avec des politiques IAM, accordez uniquement les autorisations nécessaires à l'exécution d'une

seule tâche. Pour ce faire, vous définissez les actions qui peuvent être entreprises sur des ressources spécifiques dans des conditions spécifiques, également appelées autorisations de moindre privilège. Pour plus d'informations sur l'utilisation de IAM pour appliquer des autorisations, consultez Politiques et autorisations dans IAM dans le Guide de l'utilisateur IAM.

- Utiliser des conditions dans les politiques IAM pour restreindre davantage l'accès - Vous pouvez ajouter une condition à vos politiques afin de limiter l'accès aux actions et aux ressources. Par exemple, vous pouvez écrire une condition de politique pour spécifier que toutes les demandes doivent être envoyées via SSL. Vous pouvez également utiliser des conditions pour accorder l'accès aux actions de service si elles sont utilisées via un Service AWS spécifique, comme AWS CloudFormation. Pour plus d'informations, consultez Conditions pour éléments de politique JSON IAM dans le Guide de l'utilisateur IAM.
- Utilisez IAM Access Analyzer pour valider vos politiques IAM afin de garantir des autorisations sécurisées et fonctionnelles : IAM Access Analyzer valide les politiques nouvelles et existantes de manière à ce que les politiques IAM respectent le langage de politique IAM (JSON) et les bonnes pratiques IAM. IAM Access Analyzer fournit plus de 100 vérifications de politiques et des recommandations exploitables pour vous aider à créer des politiques sécurisées et fonctionnelles. Pour de plus amples informations, consultez Validation de politique IAM Access Analyzer dans le Guide de l'utilisateur IAM.
- Authentification multifactorielle (MFA) nécessaire : si vous avez un scénario qui nécessite des utilisateurs IAM ou un utilisateur root dans votre Compte AWS, activez l'authentification multifactorielle pour une sécurité renforcée. Pour exiger le MFA lorsque des opérations d'API sont appelées, ajoutez des conditions MFA à vos politiques. Pour de plus amples informations, consultez Configuration de l'accès aux API protégé par MFA dans le Guide de l'utilisateur IAM.

Pour plus d'informations sur les bonnes pratiques dans IAM, consultez Bonnes pratiques de sécurité dans IAM dans le Guide de l'utilisateur IAM.

Autorisation accordée aux utilisateurs pour afficher leurs propres autorisations

Cet exemple montre comment créer une politique qui permet aux utilisateurs IAM d'afficher les politiques en ligne et gérées attachées à leur identité d'utilisateur. Cette stratégie inclut les autorisations nécessaires pour réaliser cette action sur la console ou par programmation à l'aide de l'AWS CLI ou de l'API AWS.

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

Autorisations de lecture simple

Cet exemple de stratégie donne à un utilisateur l'autorisation d'obtenir des informations sur toutes les ressources d'une API HTTP ou WebSocket avec l'identifiant `a123456789` dans la Région AWS us-east-1. La ressource `arn:aws:apigateway:`*`us-east-1`*`::/apis/a123456789/*` inclut toutes les sous-ressources de l'API, comme les mécanismes d'autorisation et les déploiements.

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

```
        "apigateway:GET"
      ],
      "Resource": [
        "arn:aws:apigateway:us-east-1::/apis/a123456789/*"
      ]
    }
  ]
}
```

Créer uniquement des mécanismes d'autorisation REQUEST ou JWT

Cet exemple de stratégie permet à un utilisateur de ne créer des API qu'avec les mécanismes d'autorisation `REQUEST` ou `JWT`, y compris par le biais d'une importation. Dans la section `Resource` de la stratégie, `arn:aws:apigateway:us-east-1::/apis/??????????` exige que les ressources comportent un maximum de 10 caractères, ce qui exclut les sous-ressources d'une API. Cet exemple utilise `ForAllValues` dans la section `Condition` car les utilisateurs peuvent créer plusieurs mécanismes d'autorisation à la fois en important une API.

```
{
  "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"
          ]
        }
```

```
      }
    }
  ]
}
```

Exiger que le point de terminaison **`execute-api`** par défaut soit désactivé

Cet exemple de stratégie permet aux utilisateurs de créer, de mettre à jour ou d'importer une API, avec l'exigence que `DisableExecuteApiEndpoint` soit `true`. Lorsque `DisableExecuteApiEndpoint` est `true`, les clients ne peuvent pas utiliser le point de terminaison `execute-api` par défaut pour appeler une API.

Nous utilisons la condition `BoolIfExists` pour gérer un appel visant à mettre à jour une API pour laquelle la clé de condition `DisableExecuteApiEndpoint` n'est pas renseignée. Lorsqu'un utilisateur tente de créer ou d'importer une API, la clé de condition `DisableExecuteApiEndpoint` est toujours renseignée.

Étant donné que la ressource `apis/*` capture également des sous-ressources telles que des mécanismes d'autorisation ou des méthodes, nous l'étendons explicitement aux seules API qui disposent d'une instruction `Deny`.

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

Autoriser les utilisateurs à ne créer ou mettre à jour que des API REST privées

Cet exemple de stratégie utilise des clés de condition pour exiger qu'un utilisateur ne crée que des API `PRIVATE` et pour empêcher les mises à jour susceptibles de remplacer une API de type `PRIVATE` par un API d'autre type, par exemple `REGIONAL`.

Nous utilisons `ForAllValues` pour exiger que chaque `EndpointType` ajouté à une API soit `PRIVATE`. Nous utilisons une clé de condition de ressource pour autoriser toute mise à jour d'une API à condition qu'elle soit de type `PRIVATE`. `ForAllValues` s'applique uniquement si une clé de condition est présente.

Nous utilisons ? pour établir une correspondance explicite avec les identifiants d'API afin d'éviter d'autoriser des ressources qui ne sont pas des API, telles que des mécanismes d'autorisation.

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

Exiger que les routes d'API disposent d'une autorisation

Cette stratégie entraîne l'échec des tentatives de création ou de mise à jour d'une route (y compris par le biais d'une importation) si celle-ci ne dispose d'aucune autorisation. `ForAnyValue` prend la valeur false en l'absence de la clé, par exemple lorsqu'une route n'est pas créée ou mise à jour.

Nous utilisons `ForAnyValue` car plusieurs acheminements peuvent être créées par le biais d'une importation.

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

Empêcher un utilisateur de créer ou de mettre à jour un lien VPC

Cette politique empêche un utilisateur de créer ou de mettre à jour un lien VPC Un lien VPC vous permet d'exposer des ressources dans un VPC Amazon à des clients en dehors du VPC.

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

Exemples de politique basée sur les ressources d'Amazon API Gateway

Pour accéder à des exemples de stratégies basées sur les ressources, veuillez consulter the section called “Exemples de stratégies de ressources API Gateway”.

Résolution des problèmes d'identité et d'accès à Amazon API Gateway

Utilisez les informations suivantes pour identifier et résoudre les problèmes courants que vous pouvez rencontrer lorsque vous travaillez avec API Gateway et IAM.

Rubriques

- Je ne suis pas autorisé à effectuer une action dans API Gateway
- Je ne suis pas autorisé à exécuter : iam:PassRole
- Je souhaite autoriser des personnes extérieures à mon compte AWS à accéder à mes ressources API Gateway

Je ne suis pas autorisé à effectuer une action dans API Gateway

Si vous recevez une erreur qui indique que vous n'êtes pas autorisé à effectuer une action, vos politiques doivent être mises à jour afin de vous permettre d'effectuer l'action.

L'exemple d'erreur suivant se produit quand l'utilisateur IAM `mateojackson` tente d'utiliser la console pour afficher des informations détaillées sur une ressource *`my-example-widget`* fictive, mais ne dispose pas des autorisations `apigateway:`*`GetWidget`* fictives.

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

Dans ce cas, la politique qui s'applique à l'utilisateur `mateojackson` doit être mise à jour pour autoriser l'accès à la ressource *`my-example-widget`* à l'aide de l'action `apigateway:`*`GetWidget`*.

Si vous avez encore besoin d'aide, contactez votre administrateur AWS. Votre administrateur vous a fourni vos informations de connexion.

Je ne suis pas autorisé à exécuter : iam:PassRole

Si vous recevez un message d'erreur selon lequel vous n'êtes pas autorisé à effectuer l'action `iam:PassRole`, vos stratégies doivent être mises à jour pour vous permettre de transmettre un rôle à API Gateway.

Certains Services AWS vous permettent de transmettre un rôle existant à ce service au lieu de créer un nouveau rôle de service ou rôle lié à un service. Pour ce faire, un utilisateur doit disposer des autorisations nécessaires pour transmettre le rôle au service.

L'exemple d'erreur suivant se produit lorsqu'un utilisateur IAM nommé `marymajor` essaie d'utiliser la console pour exécuter une action dans API Gateway. Toutefois, l'action nécessite que le service ait des autorisations accordées par une fonction du service. Mary ne dispose pas des autorisations nécessaires pour transférer le rôle au service.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

Dans ce cas, les stratégies de Mary doivent être mises à jour pour lui permettre d'exécuter l'action `iam:PassRole`.

Si vous avez encore besoin d'aide, contactez votre administrateur AWS. Votre administrateur vous a fourni vos informations de connexion.

Je souhaite autoriser des personnes extérieures à mon compte AWS à accéder à mes ressources API Gateway

Vous pouvez créer un rôle que les utilisateurs provenant d'autres comptes ou les personnes extérieures à votre organisation peuvent utiliser pour accéder à vos ressources. Vous pouvez spécifier qui est autorisé à assumer le rôle. Pour les services qui prennent en charge les politiques basées sur les ressources ou les listes de contrôle d'accès (ACL), vous pouvez utiliser ces politiques pour donner l'accès à vos ressources.

Pour en savoir plus, consultez les éléments suivants :

- Pour savoir si API Gateway prend en charge ces fonctions, veuillez consulter Fonctionnement d'Amazon API Gateway avec IAM.
- Pour savoir comment octroyer l'accès à vos ressources à des Comptes AWS dont vous êtes propriétaire, veuillez consulter la section Fournir l'accès à un utilisateur IAM dans un autre Compte AWS que vous possédez dans le Guide de l’utilisateur IAM.
- Pour savoir comment octroyer l'accès à vos ressources à des Comptes AWS tiers, consultez Fournir l'accès aux Comptes AWS appartenant à des tiers dans le Guide de l'utilisateur IAM.
- Pour savoir comment fournir un accès par le biais de la fédération d'identité, consultez Fournir un accès à des utilisateurs authentifiés en externe (fédération d'identité) dans le Guide de l'utilisateur IAM.
- Pour découvrir quelle est la différence entre l'utilisation des rôles et l'utilisation des politiques basées sur les ressources pour l'accès entre comptes, consultez Différence entre les rôles IAM et les politiques basées sur les ressources dans le Guide de l'utilisateur IAM.

Utilisation de rôles liés au service pour API Gateway

Amazon API Gateway utilise des rôles AWS Identity and Access Management liés à un service (IAM). Un rôle lié à un service est un type unique de rôle IAM lié directement à API Gateway. Les rôles liés à un service sont prédéfinis par API Gateway et incluent toutes les autorisations dont le service a besoin pour appeler d'autres AWS services en votre nom.

Un rôle lié à un service simplifie la configuration d'API Gateway, car vous n'avez pas besoin d'ajouter manuellement les autorisations requises. API Gateway définit les autorisations de ses rôles liés à un service et, sauf définition contraire, seul API Gateway peut endosser ses rôles. Les autorisations définies comprennent la stratégie d'approbation et la stratégie d'autorisation. De plus, cette stratégie d'autorisation ne peut pas être attachée à une autre entité IAM.

Vous pouvez supprimer un rôle lié à un service uniquement après la suppression préalable des ressources connexes. Vos ressources API Gateway sont ainsi protégées, car vous ne pouvez pas involontairement supprimer l'autorisation d'accès aux ressources.

Pour plus d'informations sur les autres services qui prennent en charge les rôles liés à un service, consultez Services AWS qui fonctionnent avec IAM et recherchez les services avec un Oui dans la colonne Rôle lié au service. Choisissez un Oui ayant un lien permettant de consulter la documentation du rôle lié à un service, pour ce service.

Autorisations des rôles liés à un service pour API Gateway

API Gateway utilise le rôle lié au service nommé AWSServiceRoleForAPIGateway— Permet à API Gateway d'accéder à Elastic Load Balancing, Amazon Data Firehose et à d'autres ressources de service en votre nom.

Le rôle AWSServiceRoleForAPIGateway lié à un service fait confiance aux services suivants pour assumer le rôle :

- `ops.apigateway.amazonaws.com`

La stratégie d'autorisations liée au rôle permet à API Gateway de réaliser les actions suivantes sur les ressources spécifiées :

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

Vous devez configurer les autorisations de manière à permettre à une entité IAM (comme un utilisateur, unn groupe ou un rôle) de créer, modifier ou supprimer un rôle lié à un service. Pour plus d'informations, consultez Service-Linked Role Permissions (autorisations du rôle lié à un service) dans le IAM User Guide (guide de l'utilisateur IAM).

Création d'un rôle lié à un service pour API Gateway

Vous n’avez pas besoin de créer manuellement un rôle lié à un service. Lorsque vous créez une API, un nom de domaine personnalisé ou un lien VPC dans l' AWS Management Console AWS API AWS CLI, API Gateway crée le rôle lié au service pour vous.

Si vous supprimez ce rôle lié à un service et que vous avez ensuite besoin de le recréer, vous pouvez utiliser la même procédure pour recréer le rôle dans votre compte. Lorsque vous créez une API, un nom de domaine personnalisé ou un lien VPC, API Gateway crée à nouveau le rôle lié au service pour vous.

Modification d'un rôle lié à un service pour API Gateway

API Gateway ne vous permet pas de modifier le rôle AWSServiceRoleForAPIGateway lié au service. Une fois que vous avez créé un rôle lié à un service, vous ne pouvez pas changer le nom du rôle, car plusieurs entités peuvent faire référence au rôle. Néanmoins, vous pouvez modifier la description du rôle à l'aide d'IAM. Pour plus d'informations, consultez Editing a Service-Linked Role (Modification d'un rôle lié à un service) dans le Guide de l'utilisateur IAM.

Suppression d'un rôle lié à un service pour API Gateway

Si vous n'avez plus besoin d'utiliser une fonction ou un service qui nécessite un rôle lié à un service, nous vous recommandons de supprimer ce rôle. De cette façon, vous n'avez aucune entité inutilisée qui n'est pas surveillée ou gérée activement. Cependant, vous devez nettoyer les ressources de votre rôle lié à un service avant de pouvoir les supprimer manuellement.

> **Note**
>
> Si le service API Gateway utilise le rôle lorsque vous essayez de supprimer les ressources, la suppression peut échouer. Si cela se produit, patientez quelques minutes et réessayez.

Pour supprimer les ressources API Gateway utilisées par AWSServiceRoleForAPIGateway

1. Ouvrez la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Accédez à l'API, au nom de domaine personnalisé ou au lien VPC qui utilise le rôle lié au service.
3. Utilisez la console pour supprimer la ressource.
4. Répétez la procédure pour supprimer toutes les API, les noms de domaine personnalisés ou les liens VPC qui utilisent le rôle lié au service.

Pour supprimer manuellement le rôle lié à un service à l'aide d'IAM

Utilisez la console IAM, le AWS CLI, ou l' AWS API pour supprimer le rôle lié au AWSServiceRoleForAPIGateway service. Pour de plus amples informations, veuillez consulter Suppression d'un rôle lié à un service dans le Guide de l'utilisateur IAM.

Régions prises en charge pour les rôles liés au service API Gateway

API Gateway prend en charge l'utilisation des rôles liés à un service dans toutes les régions où le service est disponible. Pour plus d'informations, consultez Points de terminaison du service AWS.

Mises à jour des politiques AWS gérées par API Gateway

Consultez les détails des mises à jour apportées aux politiques AWS gérées pour API Gateway depuis que ce service a commencé à suivre ces modifications. Pour obtenir des alertes automatiques concernant les modifications apportées à cette page, abonnez-vous au flux RSS de la page d'historique API Gateway Document.

Modification	Description	Date
Ajout de la prise en charge de `acm:GetCertificate` à la stratégie `AWSServiceRoleForAPIGateway` .	La stratégie `AWSServiceRoleForAPIGateway` inclut désormais l'autorisation d'appeler l'action d'API `GetCertificate` ACM.	12 Juillet 2021
API Gateway a commencé à suivre les modifications	API Gateway a commencé à suivre les modifications apportées AWS à ses politiques gérées.	12 Juillet 2021

Journalisation et surveillance dans Amazon API Gateway

La surveillance joue un rôle important dans le maintien de la fiabilité, de la disponibilité et des performances d'API Gateway et de vos AWS solutions. Vous devez collecter des données de surveillance provenant de toutes les parties de votre AWS solution afin de pouvoir corriger plus facilement une défaillance multipoint, le cas échéant. AWS fournit plusieurs outils pour surveiller les ressources de votre API Gateway et répondre aux incidents potentiels :

Amazon CloudWatch Logs

Pour résoudre les problèmes liés à l'exécution des demandes ou à l'accès des clients à votre API, vous pouvez activer CloudWatch Logs pour enregistrer les appels d'API. Pour plus d'informations, consultez the section called "CloudWatch journaux".

CloudWatch Alarmes Amazon

À l'aide d' CloudWatch alarmes, vous observez une seule métrique sur une période que vous spécifiez. Si la métrique dépasse un seuil donné, une notification est envoyée à un sujet ou à une AWS Auto Scaling politique Amazon Simple Notification Service. CloudWatch les alarmes n'appellent aucune action lorsqu'une métrique est dans un état particulier. L'état doit avoir changé et avoir été conservé pendant un nombre de périodes spécifié. Pour plus d'informations, consultez the section called "CloudWatch métriques".

Journalisation des accès à Firehose

Pour résoudre les problèmes liés à l'accès des clients à votre API, vous pouvez permettre à Firehose de consigner les appels d'API. Pour plus d'informations, consultez the section called "Firehose".

AWS CloudTrail

CloudTrail fournit un enregistrement des actions entreprises par un utilisateur, un rôle ou un AWS service dans API Gateway. À l'aide des informations collectées par CloudTrail, vous pouvez déterminer la demande envoyée à API Gateway, l'adresse IP à partir de laquelle la demande a été faite, l'auteur de la demande, la date à laquelle elle a été faite et des informations supplémentaires. Pour plus d'informations, consultez the section called "Travailler avec CloudTrail".

AWS X-Ray

X-Ray est un AWS service qui collecte des données sur les demandes traitées par votre application et les utilise pour créer une carte des services que vous pouvez utiliser pour identifier les problèmes liés à votre application et les opportunités d'optimisation. Pour plus d'informations, consultez the section called "Con AWS X-Ray figuration".

AWS Config

AWS Config fournit une vue détaillée de la configuration des AWS ressources de votre compte. Vous pouvez voir comment les ressources sont liées, obtenir un historique des changements de configuration, et voir comment les configurations et les relations changent au fil du temps. Vous

pouvez l'utiliser AWS Config pour définir des règles qui évaluent les configurations des ressources en termes de conformité des données. AWS Config les règles représentent les paramètres de configuration idéaux pour vos ressources API Gateway. Si une ressource enfreint une règle et est signalée comme non conforme, vous AWS Config pouvez être alertée via une rubrique Amazon Simple Notification Service (Amazon SNS). Pour plus d'informations, consultez the section called “Utilisation de AWS Config”.

Journalisation des appels d'API Amazon API Gateway à l'aide de AWS CloudTrail

Amazon API Gateway est intégré à AWS CloudTrailun service qui fournit un enregistrement des actions entreprises par un utilisateur, un rôle ou un Service AWS. CloudTrail capture tous les appels d'API REST pour le service API Gateway sous forme d'événements. Les appels capturés incluent des appels provenant de la console API Gateway et des appels de code vers les API du service API Gateway. À l'aide des informations collectées par CloudTrail, vous pouvez déterminer la demande qui a été envoyée à API Gateway, l'adresse IP à partir de laquelle la demande a été faite, la date à laquelle elle a été faite et des informations supplémentaires.

> **Note**
>
> TestInvokeAuthorizeret ne TestInvokeMethodsont pas connectés CloudTrail.

Chaque événement ou entrée de journal contient des informations sur la personne ayant initié la demande. Les informations relatives à l’identité permettent de déterminer :

- Si la demande a été effectuée avec des informations d’identification d’utilisateur root ou d’utilisateur root.
- Si la demande a été faite au nom d'un utilisateur de l'IAM Identity Center.
- Si la demande a été effectuée avec les informations d’identification de sécurité temporaires d’un rôle ou d’un utilisateur fédéré.
- Si la requête a été effectuée par un autre Service AWS.

CloudTrail est actif dans votre compte Compte AWS lorsque vous créez le compte et vous avez automatiquement accès à l'historique des CloudTrail événements. L'historique des CloudTrail événements fournit un enregistrement consultable, consultable, téléchargeable et immuable

des 90 derniers jours des événements de gestion enregistrés dans un. Région AWS Pour plus d'informations, consultez la section Utilisation de l'historique des CloudTrail événements dans le guide de AWS CloudTrail l'utilisateur. La consultation de CloudTrail l'historique des événements est gratuite.

Pour un enregistrement continu des événements de vos 90 Compte AWS derniers jours, créez un magasin de données sur les événements de Trail ou CloudTrailLake.

CloudTrail sentiers

Un suivi permet CloudTrail de fournir des fichiers journaux à un compartiment Amazon S3. Tous les sentiers créés à l'aide du AWS Management Console sont multirégionaux. Vous pouvez créer un parcours à région unique ou multirégionale à l'aide du. AWS CLI Il est recommandé de créer un parcours multirégional, car vous capturez l'activité dans l'ensemble Régions AWS de votre compte. Si vous créez un parcours à région unique, vous ne pouvez voir que les événements enregistrés dans le parcours. Région AWS Pour plus d'informations sur les sentiers, consultez les sections Création d'un sentier pour votre organisation Compte AWS et Création d'un sentier pour une organisation dans le guide de AWS CloudTrail l'utilisateur.

Vous pouvez envoyer une copie de vos événements de gestion en cours dans votre compartiment Amazon S3 gratuitement CloudTrail en créant un journal. Toutefois, des frais de stockage Amazon S3 sont facturés. Pour plus d'informations sur la CloudTrail tarification, consultez la section AWS CloudTrail Tarification. Pour obtenir des informations sur la tarification Amazon S3, consultez Tarification Amazon S3.

CloudTrail Stockages de données sur les événements du lac

CloudTrail Lake vous permet d'exécuter des requêtes SQL sur vos événements. CloudTrail Lake convertit les événements existants au format JSON basé sur les lignes au format Apache ORC. ORC est un format de stockage en colonnes qui est optimisé pour une récupération rapide des données. Les événements sont agrégés dans des magasins de données d'événement. Ceux-ci constituent des collections immuables d'événements basées sur des critères que vous sélectionnez en appliquant des sélecteurs d'événements avancés. Les sélecteurs que vous appliquez à un magasin de données d'événement contrôlent les événements qui persistent et que vous pouvez interroger. Pour plus d'informations sur CloudTrail Lake, consultez la section Travailler avec AWS CloudTrail Lake dans le guide de AWS CloudTrail l'utilisateur.

CloudTrail Les stockages et requêtes de données sur les événements de Lake entraînent des coûts. Lorsque vous créez un magasin de données d'événement, vous choisissez l'option de tarification que vous voulez utiliser pour le magasin de données d'événement. L'option de

tarification détermine le coût d'ingestion et de stockage des événements, ainsi que les périodes de conservation par défaut et maximale pour le magasin de données d'événement. Pour plus d'informations sur la CloudTrail tarification, consultez la section AWS CloudTrail Tarification.

Événements de gestion d'API Gateway dans CloudTrail

Les événements de gestion fournissent des informations sur les opérations de gestion effectuées sur les ressources de votre Compte AWS. Ils sont également connus sous le nom opérations de plan de contrôle. Par défaut, CloudTrail enregistre les événements de gestion.

Amazon API Gateway enregistre toutes les actions d'API Gateway en tant qu'événements de gestion, à l'exception de TestInvokeAuthorizeret TestInvokeMethod. Pour obtenir la liste des actions Amazon API Gateway auxquelles API Gateway se connecte CloudTrail, consultez le manuel Amazon API Gateway API Reference.

Exemple d'événement API Gateway

Un événement représente une demande unique provenant de n'importe quelle source et inclut des informations sur l'opération d'API demandée, la date et l'heure de l'opération, les paramètres de la demande, etc. CloudTrail les fichiers journaux ne constituent pas une trace ordonnée des appels d'API publics. Les événements n'apparaissent donc pas dans un ordre spécifique.

L'exemple suivant montre un CloudTrail événement illustrant l'`GetResource`action API Gateway :

```
{
    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 ...
    ]
}
```

Pour plus d'informations sur le contenu des CloudTrail enregistrements, voir le contenu des CloudTrail enregistrements dans le Guide de AWS CloudTrail l'utilisateur.

Surveillance de la configuration de l'API API Gateway avec AWS Config

Vous pouvez utiliser AWS Config afin d'enregistrer les modifications de configuration apportées à vos ressources de l'API d'API Gateway et envoyer des notifications en fonction de ces modifications. Il peut être utile de conserver un historique des modifications de configuration apportées aux ressources API Gateway pour un dépannage et un audit opérationnel ainsi que pour des cas d'utilisation de conformité.

AWS Config peut effectuer un suivi des modifications apportées à :

- Configuration des étapes d'API, notamment :
  - paramètres de cluster de cache ;
  - paramètres de restriction ;
  - paramètres des journaux d'accès ;
  - déploiement actif défini sur l'étape.

- Configuration d'API, notamment :
  - configuration du point de terminaison ;
  - version ;
  - protocole ;
  - balises.

En outre, la fonction AWS Config Rules permet de définir les règles de configuration, détecte et suit automatiquement les violations à ces règles et envoie des alertes à ce propos. En suivant les modifications apportées à ces propriétés de configuration des ressources, vous pouvez également créer des règles AWS Config s'exécutant en cas de modification pour vos ressources API Gateway et tester vos configurations de ressources par rapport aux bonnes pratiques.

Vous pouvez activer AWS Config dans votre compte à l'aide de la console AWS Config ou de l'AWS CLI. Sélectionnez les types de ressources pour lesquels vous souhaitez suivre les modifications. Si vous avez déjà configuré AWS Config pour enregistrer tous les types de ressources, alors ces ressources API Gateway seront automatiquement enregistrées dans votre compte. La prise en charge d'Amazon API Gateway dans AWS Config est disponible dans toutes les régions publiques AWS et AWS GovCloud (US). Pour obtenir la liste complète des Régions prises en charge, veuillez consulter les points de terminaison et quotas Amazon API Gateway dans le document Références générales AWS.

Rubriques

Types de ressources pris en charge

Les types de ressources API Gateway suivants sont intégrés avec AWS Config et sont documentés dans Types de ressources et relations entre ressources AWS pris en charge par AWS Config :

- `AWS::ApiGatewayV2::Api` (WebSocket et API HTTP)
- `AWS::ApiGateway::RestApi` (API REST)

- `AWS::ApiGatewayV2::Stage` (étape d'API WebSocket et HTTP)
- `AWS::ApiGateway::Stage` (étape de l'API REST)

Pour plus d'informations sur AWS Config, consultez le Manuel du développeur AWS Config. Pour obtenir des informations sur la tarification, consultez la page des informations de tarification d'AWS Config.

> **Important**
>
> Si vous modifiez l'une des propriétés de l'API suivantes après le déploiement de l'API, vous devez redéployer l'API pour propager les modifications. Sinon, les modifications d'attribut sont visibles dans la console AWS Config, mais les paramètres de propriété précédents restent en vigueur et le comportement d'exécution de l'API est inchangé.
>
> - **`AWS::ApiGateway::RestApi`** – `binaryMediaTypes`, `minimumCompressionSize`, `apiKeySource`
> - **`AWS::ApiGatewayV2::Api`** – `apiKeySelectionExpression`

Configuration de AWS Config

Pour configurer initialement AWS Config, veuillez consulter les rubriques suivantes dans le Guide du développeur AWS Config.

- Configuration de AWS Config avec la Console
- Configuration de AWS Config avec le AWS CLI

Configuration de AWS Config pour enregistrer les ressources API Gateway

Par défaut, AWS Config enregistre les modifications de configuration pour tous les types de ressources régionales pris en charge qu'il détecte dans la région dans laquelle votre environnement s'exécute. Vous pouvez personnaliser AWS Config pour enregistrer uniquement les modifications de types de ressources spécifiques ou les modifications apportées aux ressources globales.

Pour en savoir plus sur les ressources régionales et globales, ainsi que sur la procédure de personnalisation complète de la configuration d'AWS Config, consultez Sélection des ressources enregistrées par AWS Config.

Affichage des détails de configuration API Gateway dans la console AWS Config

Vous pouvez utiliser la console AWS Config pour rechercher des ressources API Gateway et obtenir des détails actuels et historiques sur leurs configurations. La procédure ci-après illustre comment rechercher des informations sur une API Amazon API Gateway.

Pour trouver une ressource API Gateway dans la console config AWS

1. Ouvrez la console AWS Config.
2. Choisissez Ressources.
3. Sur la page Resource Inventory (Inventaire des ressources), choisissez Resources (Ressources).
4. Ouvrez le menu Resource type (Type de ressource), faites-le défiler jusqu'à APIGateway ou APIGatewayV2, puis choisissez un ou plusieurs types de ressources API Gateway.
5. Choisissez Recherche.
6. Choisissez un ID de ressource dans la liste des ressources affichée par AWS Config. AWS Config affiche les détails de la configuration ainsi que d'autres informations sur la ressource que vous avez sélectionnée.
7. Pour voir tous les détails de la configuration enregistrée, choisissez View Details (Afficher les détails).

Pour en savoir plus sur d'autres manières de trouver une ressource et d'afficher des informations sur cette page, veuillez consulter Affichage des configurations et de l'historique des ressources AWS dans le Guide du développeur AWS Config.

Évaluation des ressources API Gateway à l'aide des règles AWS Config

Vous pouvez créer des règles AWS Config qui représentent les paramètres de configuration idéaux pour vos ressources API Gateway. Vous pouvez utiliser des règles gérées AWS Config ou définir des règles personnalisées. AWS Config surveille en permanence les modifications apportées à la configuration de vos ressources afin de déterminer si elles ne vont pas à l'encontre de l'une des conditions de vos règles. La console AWS Config affiche l'état de conformité de vos règles et ressources.

Si une ressource enfreint une règle et est signalée comme non conforme, AWS Config peut vous alerter à l'aide d'une rubrique du Guide du développeur Amazon Simple Notification Service (Amazon

SNS). Pour programmer la consommation des données contenues dans ces alertes AWS Config, utilisez une file d'attente Amazon Simple Queue Service (Amazon SQS) comme point de terminaison de notification pour la rubrique Amazon SNS.

Pour en savoir plus sur la configuration et l'utilisation de règles, veuillez consulter Évaluation des ressources à l'aide de règles dans le Guide du développeurAWS Config.

Validation de conformité pour Amazon API Gateway

Pour savoir si un programme Services AWS de conformité Service AWS s'inscrit dans le champ d'application de programmes de conformité spécifiques, consultez Services AWS la section de conformité et sélectionnez le programme de conformité qui vous intéresse. Pour des informations générales, voir Programmes de AWS conformité Programmes AWS de .

Vous pouvez télécharger des rapports d'audit tiers à l'aide de AWS Artifact. Pour plus d'informations, voir Téléchargement de rapports dans AWS Artifact .

Votre responsabilité en matière de conformité lors de l'utilisation Services AWS est déterminée par la sensibilité de vos données, les objectifs de conformité de votre entreprise et les lois et réglementations applicables. AWS fournit les ressources suivantes pour faciliter la mise en conformité :

- Guides de démarrage rapide sur la sécurité et la conformité : ces guides de déploiement abordent les considérations architecturales et indiquent les étapes à suivre pour déployer des environnements de base axés sur AWS la sécurité et la conformité.
- Architecture axée sur la sécurité et la conformité HIPAA sur Amazon Web Services : ce livre blanc décrit comment les entreprises peuvent créer des applications AWS conformes à la loi HIPAA.

  > **Note**
  >
  > Tous ne Services AWS sont pas éligibles à la loi HIPAA. Pour plus d'informations, consultez le HIPAA Eligible Services Reference.

- AWS Ressources de https://aws.amazon.com/compliance/resources/ de conformité — Cette collection de classeurs et de guides peut s'appliquer à votre secteur d'activité et à votre région.
- AWS Guides de conformité destinés aux clients — Comprenez le modèle de responsabilité partagée sous l'angle de la conformité. Les guides résument les meilleures pratiques en matière de sécurisation Services AWS et décrivent les directives relatives aux contrôles de sécurité dans

de nombreux cadres (notamment le National Institute of Standards and Technology (NIST), le Payment Card Industry Security Standards Council (PCI) et l'Organisation internationale de normalisation (ISO)).

- Évaluation des ressources à l'aide des règles du guide du AWS Config développeur : le AWS Config service évalue dans quelle mesure les configurations de vos ressources sont conformes aux pratiques internes, aux directives du secteur et aux réglementations.
- AWS Security Hub— Cela Service AWS fournit une vue complète de votre état de sécurité interne AWS. Security Hub utilise des contrôles de sécurité pour évaluer vos ressources AWS et vérifier votre conformité par rapport aux normes et aux bonnes pratiques du secteur de la sécurité. Pour obtenir la liste des services et des contrôles pris en charge, consultez Référence des contrôles Security Hub.
- Amazon GuardDuty — Cela Service AWS détecte les menaces potentielles qui pèsent sur vos charges de travail Comptes AWS, vos conteneurs et vos données en surveillant votre environnement pour détecter toute activité suspecte et malveillante. GuardDuty peut vous aider à répondre à diverses exigences de conformité, telles que la norme PCI DSS, en répondant aux exigences de détection des intrusions imposées par certains cadres de conformité.
- AWS Audit Manager— Cela vous Service AWS permet d'auditer en permanence votre AWS utilisation afin de simplifier la gestion des risques et la conformité aux réglementations et aux normes du secteur.

Résilience dans Amazon API Gateway

L'infrastructure mondiale AWS s'articule autours de régions et de zones de disponibilité AWS. AWS Les Régions fournissent plusieurs zones de disponibilité physiquement séparées et isolées, reliées par un réseau à latence faible, à haut débit et hautement redondant. Avec les zones de disponibilité, vous pouvez concevoir et exploiter des applications et des bases de données qui basculent automatiquement d'une zone à l'autre sans interruption. Les zones de disponibilité sont plus hautement disponibles, tolérantes aux pannes et évolutives que les infrastructures traditionnelles à un ou plusieurs centres de données.

Pour plus d'informations sur les régions et les zones de disponibilité AWS, consultez AWS Infrastructure mondiale.

Pour éviter que vos API soient submergées par un trop grand nombre de demandes, API Gateway régule le flux des demandes soumises à vos API. En particulier, API Gateway définit une limite sur les soumissions de demandes à un débit régulier et en mode rafale pour toutes les API dans

votre compte. Vous pouvez configurer la limitation personnalisée pour vos API. Pour en savoir plus, veuillez consulter la section Limiter les demandes d'API pour améliorer le débit.

Vous pouvez utiliser les surveillances de l'état de Route 53 pour contrôler le basculement DNS d'une API de passerelle API dans une région principale vers une API d'API Gateway dans une région secondaire. Pour obtenir un exemple, veuillez consulter the section called “basculement DNS”.

Sécurité de l'infrastructure dans Amazon API Gateway

En tant que service géré, Amazon API Gateway est protégé par la sécurité du réseau mondial AWS. Pour plus d'informations sur les services de sécurité AWS et la manière dont AWS protège l'infrastructure, consultez la section Sécurité du cloud AWS. Pour concevoir votre environnement AWS en utilisant les meilleures pratiques en matière de sécurité de l'infrastructure, consultez la section Protection de l'infrastructure dans le Security Pillar AWS Well-Architected Framework (Pilier de sécurité de l'infrastructure Well-Architected Framework).

Vous pouvez utiliser les appels d'API publiés par AWS pour accéder à API Gateway via le réseau. Les clients doivent prendre en charge les éléments suivants :

- Protocole TLS (Transport Layer Security). Nous exigeons TLS 1.2 et nous recommandons TLS 1.3.
- Ses suites de chiffrement PFS (Perfect Forward Secrecy) comme DHE (Ephemeral Diffie-Hellman) ou ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). La plupart des systèmes modernes tels que Java 7 et les versions ultérieures prennent en charge ces modes.

En outre, les demandes doivent être signées à l'aide d'un ID de clé d'accès et d'une clé d'accès secrète associée à un principal IAM. Vous pouvez également utiliser AWS Security Token Service (AWS STS) pour générer des informations d'identification de sécurité temporaires et signer les demandes.

Vous pouvez appeler ces opérations d'API à partir de n'importe quel emplacement sur le réseau, mais API Gateway prend en charge les stratégies d'accès basées sur les ressources, ce qui peut inclure des restrictions en fonction de l'adresse IP source. Vous pouvez également utiliser des stratégies basées sur les ressources pour contrôler l'accès à partir de points de terminaison Amazon Virtual Private Cloud (Amazon VPC) ou de VPC spécifiques. Effectivement, cela permet d'isoler l'accès réseau à une ressource API Gateway donnée uniquement depuis le VPC spécifique au sein du réseau AWS.

Analyse des vulnérabilités dans Amazon API Gateway

La configuration et les contrôles informatiques sont une responsabilité partagée entre AWS et vous, notre client. Pour en savoir plus, veuillez consulter Modèle de responsabilité partagée AWS.

Bonnes pratiques de sécurité dans Amazon API Gateway

API Gateway fournit différentes fonctions de sécurité à prendre en compte lorsque vous développez et implémentez vos propres stratégies de sécurité. Les bonnes pratiques suivantes doivent être considérées comme des instructions générales et ne représentent pas une solution de sécurité complète. Étant donné que ces bonnes pratiques peuvent ne pas être appropriées ou suffisantes pour votre environnement, considérez-les comme des remarques utiles plutôt que comme des recommandations.

Implémentation d'un accès sur la base du moindre privilège

Utilisez des stratégies IAM pour implémenter l'accès au moindre privilège pour la création, la lecture, la mise à jour ou la suppression d'API Amazon API Gateway. Pour en savoir plus, consultez la section Identity and Access Management pour Amazon API Gateway. API Gateway offre plusieurs options pour contrôler l'accès aux API que vous créez. Pour en savoir plus, consultez Contrôle et gestion de l'accès à une API REST dans API Gateway, Contrôle et gestion de l'accès à une WebSocket API dans API Gateway et Contrôle de l'accès aux API HTTP avec les mécanismes d'autorisation JWT.

Mise en œuvre de la journalisation

Utilisez CloudWatch Logs ou Amazon Data Firehose pour enregistrer les requêtes adressées à vos API. Pour en savoir plus, consultez Surveillance des API REST, Configuration de la journalisation pour une WebSocket API et Configuration de la journalisation pour une API HTTP.

Implémenter les CloudWatch alarmes Amazon

À l'aide d' CloudWatch alarmes, vous observez une seule métrique sur une période que vous spécifiez. Si la métrique dépasse un seuil donné, une notification est envoyée à un sujet ou à une AWS Auto Scaling politique Amazon Simple Notification Service. CloudWatch les alarmes n'appellent aucune action lorsqu'une métrique est dans un état particulier. L'état doit avoir changé et avoir été conservé pendant un nombre de périodes spécifié. Pour de plus amples informations, veuillez consulter the section called “CloudWatch métriques”.

Activer AWS CloudTrail

CloudTrail fournit un enregistrement des actions entreprises par un utilisateur, un rôle ou un AWS service dans API Gateway. À l'aide des informations collectées par CloudTrail, vous pouvez déterminer la demande qui a été faite à API Gateway, l'adresse IP à partir de laquelle la demande a été faite, l'auteur de la demande, la date à laquelle elle a été faite et des informations supplémentaires. Pour de plus amples informations, veuillez consulter the section called “Travailler avec CloudTrail”.

Activer AWS Config

AWS Config fournit une vue détaillée de la configuration des AWS ressources de votre compte. Vous pouvez voir comment les ressources sont liées, obtenir un historique des changements de configuration, et voir comment les configurations et les relations changent au fil du temps. Vous pouvez l'utiliser AWS Config pour définir des règles qui évaluent les configurations des ressources en termes de conformité des données. AWS Config les règles représentent les paramètres de configuration idéaux pour vos ressources API Gateway. Si une ressource enfreint une règle et est signalée comme non conforme, vous AWS Config pouvez être alertée via une rubrique Amazon Simple Notification Service (Amazon SNS). Pour plus de détails, consultez the section called “Utilisation de AWS Config”.

Utiliser AWS Security Hub

Surveillez votre utilisation d'API Gateway, car elle est liée aux bonnes pratiques de sécurité, en utilisant AWS Security Hub. Security Hub utilise des contrôles de sécurité pour évaluer les configurations des ressources et les normes de sécurité afin de vous aider à respecter divers cadres de conformité. Pour plus d'informations sur l'utilisation de Security Hub pour évaluer les ressources API Gateway, consultez Contrôles d'Amazon API Gateway dans le Guide de l'utilisateur AWS Security Hub .

Ajout de balises à vos ressources API Gateway

Une balise est une étiquette de métadonnées que vous attribuez ou que vous AWS attribuez à une AWS ressource. Chaque balise se compose de deux parties :

- Une clé de balise (par exemple, `CostCenter`, `Environment` ou `Project`). Les clés de balises sont sensibles à la casse.
- Un champ facultatif appelé valeur de balise (par exemple, `111122223333` ou `Production`). Si la valeur de balise est identique à l'utilisation d'une chaîne vide. Les valeurs de balises sont sensibles à la casse, tout comme les clés de balises.

Les balises vous permettent d'effectuer les actions suivantes :

- Contrôler l'accès à vos ressources en fonction des balises qui leur sont affectées. Vous pouvez contrôlez l'accès en spécifiant des clés et des valeurs de balise dans les conditions d'une stratégie AWS Identity and Access Management (IAM). Pour plus d'informations sur le contrôle d'accès basé sur des balises, consultez Contrôle de l'accès à l'aide de balises dans le Guide de l'utilisateur IAM.
- Suivez vos AWS coûts. Vous activez ces balises sur le AWS Billing and Cost Management tableau de bord. AWS utilise les balises pour classer vos coûts et vous fournir un rapport mensuel de répartition des coûts. Pour de plus amples informations, veuillez consulter Utilisation des balises d'allocation des coûts dans le Guide de l'utilisateur AWS Billing.
- Identifiez et organisez vos AWS ressources. De nombreux AWS services prennent en charge le balisage. Vous pouvez donc attribuer le même tag aux ressources de différents services pour indiquer que les ressources sont liées. Par exemple, vous pouvez attribuer la même balise à un stage d'API Gateway que celle que vous attribuez à une règle CloudWatch Events.

Pour obtenir des conseils sur l'utilisation des balises, consultez le livre blanc Stratégies de AWS balisage.

Les sections suivantes fournissent de plus amples informations sur les balises pour Amazon API Gateway.

Rubriques

- Ressources API Gateway pouvant être balisées
- Utilisation de balises pour contrôler l'accès aux ressources API REST API Gateway

Ressources API Gateway pouvant être balisées

Les balises peuvent être définies sur l'API HTTP ou les ressources WebSocket d'API suivantes dans l'API Amazon API Gateway V2 :

- `Api`
- `DomainName`
- `Stage`
- `VpcLink`

En outre, les balises peuvent être définies sur les ressources d'API REST suivantes dans l'API Amazon API Gateway V1 :

- `ApiKey`
- `ClientCertificate`
- `DomainName`
- `RestApi`
- `Stage`
- `UsagePlan`
- `VpcLink`

Les balises ne peuvent pas être définies directement sur d'autres ressources. Cependant, dans l'API Amazon API Gateway V1, les ressources enfants héritent des balises qui sont définies sur les ressources parents. Par exemple :

- Si une identification est définie sur une ressource `RestApi`, cette identification est héritée par les ressources enfants de cet élément `RestApi` dans le contrôle d’accès basé sur les attributs :
  - `Authorizer`
  - `Deployment`
  - `Documentation`
  - `GatewayResponse`
  - `Integration`
  - `Method`

- `Model`
- `Resource`
- `ResourcePolicy`
- `Setting`
- `Stage`

- Si une balise est définie sur une ressource `DomainName`, cette balise est héritée par les ressources `BasePathMapping` sous celle-ci.
- Si une balise est définie sur une ressource `UsagePlan`, cette balise est héritée par les ressources `UsagePlanKey` sous celle-ci.

> **Note**
>
> L'héritage d'identifications s'applique uniquement au contrôle d'accès basé sur les attributs. Par exemple, vous ne pouvez pas utiliser de balises héritées pour surveiller les coûts dans AWS Cost Explorer. API Gateway ne renvoie pas les balises héritées lorsque vous appelez GetTagsune ressource.

Héritage de balises dans l'API Amazon API Gateway V1

Auparavant, il était uniquement possible de configurer des balises sur des étapes. Maintenant que vous pouvez également les configurer sur d'autres ressources, une ressource `Stage` peut recevoir une balise de deux manières :

- La balise peut être définie directement sur la ressource `Stage`.
- L'étape peuvent hériter de la balise de sa ressource paren `RestApi`.

Si une étape reçoit une balise de ces deux manières, la balise qui a été définie directement sur l'étape est prioritaire. Par exemple, supposons qu'une étape hérite des balises suivantes de l'API REST de son parent :

```
{
 'foo': 'bar',
 'x':'y'
}
```

Supposons qu'il possède également les balises suivantes définies directement sur celle-ci :

```
{
 'foo': 'bar2',
 'hello': 'world'
}
```

Cela aurait pour effet que l'étape comporte les balises suivantes, avec les valeurs suivantes :

```
{
 'foo': 'bar2',
 'hello': 'world'
 'x':'y'
}
```

Restrictions liées aux balises et conventions d'utilisation

Les restrictions et les conventions d'utilisation suivantes s'appliquent à l'utilisation des balises avec les ressources API Gateway :

- Chaque ressource peut avoir un maximum de 50 balises.
- Pour chaque ressource, chaque clé de balise doit être unique, et chaque clé de balise peut avoir une seule valeur.
- La longueur maximale des clés de balise est de 128 caractères Unicode en UTF-8.
- La longueur maximale des valeurs de balise est de 256 caractères Unicode en UTF-8.
- Les caractères autorisés pour les clés et les valeurs sont les lettres, les espaces et les chiffres représentables en UTF-8, ainsi que les caractères spéciaux suivants : . : + = @ _ / - (tiret). Les ressources Amazon EC2 autorisent tous les caractères.
- Les clés et valeurs de balise sont sensibles à la casse. La bonne pratique consiste à choisir une stratégie pour mettre des balises en majuscule et mettre en œuvre cette stratégie de manière cohérente sur tous les types de ressources. Par exemple, décidez si vous souhaitez utiliser `Costcenter`, `costcenter` ou `CostCenter`, et utilisez la même convention pour toutes les balises. Évitez d'utiliser des balises avec une incohérence de traitement de cas similaires.
- Le préfixe `aws:` est interdit pour les balises ; il est réservé à l'utilisation d' AWS . Vous ne pouvez pas modifier ni supprimer des clés ou valeurs de balise ayant ce préfixe. Les étiquettes avec ce préfixe ne sont pas comptabilisées comme vos étiquettes pour la limite de ressources.

Utilisation de balises pour contrôler l'accès aux ressources API REST API Gateway

Les conditions figurant dans les AWS Identity and Access Management politiques font partie de la syntaxe que vous utilisez pour spécifier les autorisations d'accès aux ressources API Gateway. Pour de plus amples informations sur la spécification de stratégies IAM, veuillez consulter the section called “Utilisation des autorisations IAM”. Dans API Gateway, les ressources et certaines actions peuvent comporter des balises. Lorsque vous créez une stratégie IAM, vous pouvez utiliser des clés de condition de balise pour contrôler :

- quels utilisateurs peuvent effectuer des actions sur une ressource API Gateway, en fonction des balises que la ressource possède déjà ;
- quelles balises peuvent être transmises dans une demande d'action ;
- si des clés de balise spécifiques peuvent être utilisées dans une demande.

L’utilisation d’identifications pour le contrôle d'accès basé sur les attributs peut permettre un contrôle plus précis que le contrôle au niveau de l'API, et un contrôle plus dynamique que le contrôle d'accès basé sur les ressources. Il est possible de créer des stratégies IAM qui autorisent ou interdisent une opération en fonction des balises fournies dans la demande (balises de demande) ou des balises sur la ressource à laquelle les stratégies s’appliquent (balises de ressource). En général, les balises de ressource sont destinées aux ressources qui existent déjà. Les balises de demande conviennent lorsque vous créez des ressources.

Pour connaître la syntaxe complète et la sémantique des clés de condition de balise, consultez Contrôle de l'accès à l'aide de balises dans le Guide de l'utilisateur IAM .

Les exemples suivants montrent comment spécifier des conditions de balises dans les stratégies pour les utilisateurs API Gateway.

Limiter les actions en fonction des balises de ressource

L'exemple de politique suivant autorise les utilisateurs à effectuer toutes les actions sur toutes les ressources, à condition que ces ressources n'aient pas de balise `Environment` avec la valeur `prod`.

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

```
    {
      "Effect": "Allow",
      "Action": "apigateway:*",
      "Resource": "*"
    },
    {
      "Effect": "Deny",
      "Action": [
        "apigateway:*"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/Environment": "prod"
        }
      }
    }
  ]
}
```

Autoriser des actions en fonction des balises de ressource

L'exemple de politique suivant autorise les utilisateurs à effectuer toutes les actions sur les ressources API Gateway, à condition que ces ressources aient la balise `Environment` avec la valeur `Development`. L'instruction `Deny` empêche l'utilisateur de modifier la valeur de la balise `Environment`.

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

Refuser les opérations de balisage

L'exemple de politique suivant permet à un utilisateur d'exécuter toutes les actions API Gateway, sauf la modification de balises.

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

```
      "Resource": [
        "*"
      ],
    },
    {
      "Effect": "Deny",
      "Action": [
        "apigateway:*"
      ],
      "Resource": "arn:aws:apigateway:*::/tags*",
    }
  ]
}
```

Autoriser les opérations de balisage

L'exemple de politique suivant permet à un utilisateur d'obtenir toutes les ressources API Gateway et de modifier les balises de ces ressources. Pour obtenir les balises d'une ressource, l'utilisateur doit disposer des autorisations `GET` nécessaires pour cette ressource. Pour mettre à jour les balises d'une ressource, l'utilisateur doit disposer des autorisations `PATCH` nécessaires pour cette ressource.

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

Références d'API

Amazon API Gateway fournit des API pour créer et déployer votre propre protocole HTTP et vos propres WebSocket API. En outre, les API API Gateway sont disponibles dans les AWS SDK standard.

Si vous utilisez un langage pour lequel un AWS SDK existe, vous préférerez peut-être utiliser le SDK plutôt que d'utiliser directement les API REST d'API Gateway. Les kits SDK simplifient l'authentification, s'intègrent facilement à votre environnement de développement et permettent d'accéder facilement aux commandes API Gateway.

Voici où se trouvent les AWS SDK et la documentation de référence de l'API REST API Gateway :

- Outils pour Amazon Web Services
- Référence d'API REST Amazon API Gateway
- Amazon API Gateway WebSocket et référence d'API HTTP

Quotas Amazon API Gateway et remarques importantes

Rubriques

Sauf indication contraire, les quotas peuvent être augmentés sur simple demande. Pour demander une augmentation de quota, vous pouvez utiliser Service Quotas ou contacter le Centre de support AWS.

Lorsque l'autorisation est activée pour une méthode, la longueur maximale de l'ARN de cette méthode (par exemple, `arn:aws:execute-api:{region-id}:{account-id}:{api-id}/{stage-id}/{method}/{resource}/{path}`) est de 1 600 octets. Les valeurs des paramètres de chemin (dont la taille est déterminée au moment de l'exécution) peuvent entraîner un dépassement de la limite pour la longueur de l'ARN. Lorsque cela se produit, le client API reçoit une réponse `414 Request URI too long`.

> **Note**
>
> Cela limite la longueur de l'URI lorsque les stratégies de ressources sont utilisées. Dans le cas d'API privées où une stratégie de ressource est requise, la longueur de l'URI de toutes les API privées est limitée.

Quotas au niveau du compte API Gateway, par région

Les quotas suivants s'appliquent par compte et par région dans Amazon API Gateway.

Ressource ou opération	Quota par défaut	Peut être augmenté
Limitez le quota par compte et par région entre les API HTTP, les API REST, les WebSocket API et les API de WebSocket rappel	10 000 demandes par seconde (RPS) avec une capacité de transmission en mode rafale supplémentaire fournie par l'algorithme de compartiment à jeton, en utilisant une capacité de compartiment maximale de 5 000 demandes. * **Note** Le quota en mode rafale est déterminé par l'équipe de service API Gateway en fonction de l'ensemble des quotas RPS pour le compte dans la région. Ce n'est pas un quota qu'un client peut contrôler ou pour lequel il peut demander des modifications.	Oui
API régionales	600	Non
API optimisées pour les périphériques	120	Non

* Pour les régions suivantes, le quota d'accélération par défaut est de 2500 RPS et le quota de rafale par défaut est de 1250 RPS : Afrique (Le Cap), Europe (Milan), Asie-Pacifique (Jakarta), Moyen-Orient (Émirats arabes unis), Asie-Pacifique (Hyderabad), Asie-Pacifique (Melbourne), Europe (Espagne), Europe (Zurich), Israël (Tel Aviv) et Canada Ouest (Calgary).

Quotas API HTTP

Les quotas suivants s'appliquent à la configuration et à l'exécution d'une API HTTP dans API Gateway.

Ressource ou opération	Quota par défaut	Peut être augmenté
Routes par API	300	Oui
Intégrations par API	300	Non
Délai d'intégration maximal	30 secondes	Non
Étapes par API	10	Oui
Mappages d'API à plusieurs niveaux par domaine	200	Non
Balises par étape	50	Non
Taille totale combinée des valeurs de ligne de demande et d'en-tête	10 240 octets	Non
Taille de la charge utile	10 Mo	Non
Domaines personnalisés par compte et par région	120	Oui
Taille du modèle de journal d'accès	3 KO	Non
Entrée dans le CloudWatch	1 Mo	Non

Ressource ou opération	Quota par défaut	Peut être augmenté
journal Amazon Logs		
Mécanismes d'autorisation par API	10	Oui
Audiences par mécanisme d'autorisation	50	Non
Portées par itinéraire	10	Non
Délai d'expiration pour le point de terminaison JSON Web Key Set	1500 ms	Non
Taille de la réponse du point de terminaison JSON Web Key Set	150 000 octets	Non
Délai d'expiration du point de terminaison de découverte OpenID Connect	1500 ms	Non

Ressource ou opération	Quota par défaut	Peut être augmenté
Délai d'expiration pour la réponse du mécanisme d'autorisation Lambda	10 000 ms	Non
Liens VPC par compte et par région	10	Oui
Lien Sous-réseaux par VPC	10	Oui
Variables d'étape par étape	100	Non
Longueur, en caractères, de la clé d'une variable d'étape	64	Non
Longueur, en caractères, de la valeur d'une variable d'étape	512	Non

Quotas d'API Gateway pour configurer et exécuter une WebSocket API

Les quotas suivants s'appliquent à la configuration et à l'exécution d'une WebSocket API dans Amazon API Gateway.

Ressource ou opération	Quota par défaut	Peut être augmenté
Nouvelles connexions par seconde et par compte (toutes les WebSocket API) par région	500	Oui
Connexions simultanées	Ne s'applique pas *	Ne s’applique pas
AWS Lambda autorisateurs par API	10	Oui
AWS Lambda taille du résultat de l'autorisateur	8 Ko	Non
Routes par API	300	Oui
Intégrations par API	300	Oui
Délai d'intégration	50 millisecondes - 29 secondes pour tous les types d'intégration, y compris Lambda, le proxy Lambda, le HTTP, le proxy HTTP et les intégrations. AWS	Non
Étapes par API	10	Oui
WebSocket taille du cadre	32 Ko	Non
Taille de la charge utile des messages	128 Ko **	Non

Ressource ou opération	Quota par défaut	Peut être augmenté
Durée de connexion pour WebSocket l'API	2 heures	Non
Délai d'expiration de connexion inactive	10 minutes	Non
Longueur, en caractères, de l'URL d'une WebSocket API	4096	Non

* API Gateway n'applique pas de quota sur les connexions simultanées. Le nombre maximal de connexions simultanées est déterminé par le taux de nouvelles connexions par seconde et une durée maximale de connexion de deux heures. Par exemple, avec un quota par défaut de 500 nouvelles connexions par seconde, si les clients se connectent au taux maximal sur deux heures, API Gateway peut servir jusqu'à 3 600 000 connexions simultanées.

** En raison du quota de WebSocket taille de trame de 32 Ko, un message supérieur à 32 Ko doit être divisé en plusieurs trames, chacune inférieure ou égale à 32 Ko. Cela s'applique aux commandes `@connections`. Si un message plus grand (ou une taille de trame supérieure) est reçu, la connexion se ferme avec le code 1009.

Quotas API Gateway pour la configuration et l'exécution d'une API REST

Les quotas suivants s'appliquent à la configuration et à l'exécution d'une API REST dans Amazon API Gateway. Pour restapi:import ou restapi:put, la taille maximum du fichier de définition d'API est de 6 Mo.

Toutes les limites par API ne peuvent être augmentées que sur des API spécifiques.

Ressource ou opération	Quota par défaut	Peut être augmenté
Noms de domaine personnalisés par compte et par région	120	Oui
Mappages d'API à plusieurs niveaux par domaine	200	Non
Longueur, en caractères, de l'URL pour une API optimisée pour les périphériques	8192	Non
Longueur, en caractères, de l'URL pour une API régionale	10240	Non
API privées par compte et par région.	600	Non
Longueur, en caractères, de la stratégie de ressources API Gateway	8192	Oui
Clés d'API par compte et par région	10 000	Non

Ressource ou opération	Quota par défaut	Peut être augmenté
Certificats de client par compte et par région	60	Oui
Autorisateurs par API (AWS Lambda et Amazon Cognito)	10	Oui
Parties de la documentation par API	2000	Oui
Ressources par API	300	Oui
Étapes par API	10	Oui
Variables d'étape par étape	100	Non
Longueur, en caractères, de la clé d'une variable d'étape	64	Non
Longueur, en caractères, de la valeur d'une variable d'étape	512	Non
Plans d'utilisation par compte et par région	300	Oui

Ressource ou opération	Quota par défaut	Peut être augmenté
Plans d'utilisation par clé API	10	Oui
Liens VPC par compte et par région	20	Oui
Durée de vie de mise en cache des API	300 secondes par défaut et configurable entre 0 et 3 600 par un propriétaire d'API.	Pas la limite supérieure (3 600)
Taille de la réponse mise en cache	1 048 576 octets. Le chiffrement des données de cache peut augmenter la taille de l'élément mis en cache.	Non
Délai d'intégration	50 millisecondes - 29 secondes pour tous les types d'intégration, y compris Lambda, le proxy Lambda, le HTTP, le proxy HTTP et les intégrations. AWS	Oui*
Taille combinée totale de toutes les valeurs d'en-tête	10 240 octets	Non
Taille combinée totale de toutes les valeurs d'en-tête pour une API privée	8 000 octets	Non
Taille de la charge utile	10 Mo	Non
Balises par étape	50	Non

Ressource ou opération	Quota par défaut	Peut être augmenté
Nombre d'itérations dans une boucle `#foreach ... #end` dans les modèles de mappage	1 000	Non
Longueur de l'ARN d'une méthode avec autorisation	1 600 octets	Non
Paramètres de limitation au niveau de la méthode pour une étape dans un plan d'utilisation	20	Oui
Taille de modèle par API	400 Ko	Non
Nombre de certificats dans un truststore	1 000 certificats jusqu'à une taille totale d'objet de 1 Mo.	Non

* Vous ne pouvez pas définir le délai d'intégration à moins de 50 millisecondes. Vous pouvez augmenter le délai d'intégration à plus de 29 secondes pour les API régionales et les API privées, mais cela peut nécessiter une réduction du quota d'accélération au niveau de votre compte.

Quotas API Gateway concernant la création, le déploiement et la gestion d'une API

Les quotas fixes suivants s'appliquent à la création, au déploiement et à la gestion d'une API dans API Gateway, à l' AWS CLI aide de la console API Gateway ou de l'API REST API Gateway et de ses SDK. Ces quotas ne peuvent pas être augmentés.

Action	Quota par défaut	Peut être augmenté
CreateApiKey	5 demandes toutes les secondes par compte	Non
CreateDeployment	1 demande toutes les 5 secondes par compte	Non
CreateDocumentationVersion	1 demande toutes les 20 secondes par compte	Non
CreateDomainName	1 requête toutes les 30 secondes par compte	Non
CreateResource	5 demandes toutes les secondes par compte	Non
CreateRestApi	API régionale ou privée • 1 requête toutes les 3 secondes par compte API optimisée pour les périphériques • 1 requête toutes les 30 secondes par compte	Non
CreateVpcLink(V2)	1 demande toutes les 15 secondes par compte	Non

Action	Quota par défaut	Peut être augmenté
DeleteApiKey	5 demandes toutes les secondes par compte	Non
DeleteDomainName	1 requête toutes les 30 secondes par compte	Non
DeleteResource	5 demandes toutes les secondes par compte	Non
DeleteRestApi	1 requête toutes les 30 secondes par compte	Non
GetResources	5 requêtes toutes les 2 secondes par compte	Non
DeleteVpcLink(V2)	1 requête toutes les 30 secondes par compte	Non
ImportDocumentationParts	1 requête toutes les 30 secondes par compte	Non
ImportRestApi	API régionale ou privée • 1 requête toutes les 3 secondes par compte API optimisée pour les périphériques • 1 requête toutes les 30 secondes par compte	Non
PutRestApi	1 requête toutes les secondes par compte	Non
UpdateAccount	1 demande toutes les 20 secondes par compte	Non

Action	Quota par défaut	Peut être augmenté
UpdateDomainName	1 requête toutes les 30 secondes par compte	Non
UpdateUsagePlan	1 demande toutes les 20 secondes par compte	Non
Autres opérations	Pas de quota jusqu'au quota total du compte.	Non
Nombre total d'opérations	10 requêtes toutes les secondes avec un quota en mode rafale de 40 requêtes par seconde.	Non

Remarques importantes concernant Amazon API Gateway

Rubriques

- Remarques importantes d'Amazon API Gateway concernant les API REST, les API HTTP et les WebSocket API
- Remarques importantes relatives à Amazon API Gateway concernant REST et les WebSocket API
- Remarques importantes concernant Amazon API Gateway concernant les WebSocket API
- Remarques importantes concernant Amazon API Gateway pour les API REST

Remarques importantes d'Amazon API Gateway concernant les API REST, les API HTTP et les WebSocket API

- Signature Version 4A n'est pas officiellement prise en charge par Amazon API Gateway.

Remarques importantes relatives à Amazon API Gateway concernant REST et les WebSocket API

- API Gateway ne prend pas en charge le partage d'un nom de domaine personnalisé entre REST et WebSocket les API.
- Les noms d'étape peuvent contenir uniquement des caractères alphanumériques, des tirets et des traits de soulignement. La longueur maximale est de 128 caractères.
- Les chemins d'accès `/ping` et `/sping` sont réservés à la vérification de l'état du service. L'utilisation de ces chemins pour les ressources racine de l'API avec des domaines personnalisés ne permet pas d'obtenir le résultat attendu.
- API Gateway limite actuellement les évènements de journaux à 1 024 octets. Les événements de journal de plus de 1 024 octets, tels que les corps de demande et de réponse, seront tronqués par API Gateway avant d'être soumis à CloudWatch Logs.
- CloudWatch Metrics limite actuellement les noms et les valeurs des dimensions à 255 caractères XML valides. (Pour plus d'informations, consultez le guide de CloudWatch l'utilisateur.) Les valeurs de dimension sont une fonction des noms définis par l'utilisateur, y compris le nom de l'API, le nom de l'étiquette (étape) et le nom de la ressource. Lorsque vous choisissez ces noms, veillez à ne pas dépasser CloudWatch les limites des métriques.
- La taille maximale d'un modèle de mappage est de 300 Ko.

Remarques importantes concernant Amazon API Gateway concernant les WebSocket API

- API Gateway prend en charge des charges utiles de messages jusqu'à 128 Ko, avec une taille de trame maximale de 32 Ko. Si un message dépasse 32 Ko, vous devez le diviser en plusieurs trames, chacune de 32 Ko ou moins. Si un message plus grand est reçu, la connexion se ferme avec le code 1009.

Remarques importantes concernant Amazon API Gateway pour les API REST

- Le caractère pipe en texte brut (`|`) n'est pas pris en charge pour les chaînes de requête de l'URL de demande et doit être encodé en URL.

- Le caractère point-virgule (`;`) n'est pas pris en charge pour les chaînes de requête d'URL et entraîne le fractionnement des données.
- Les API REST décodent les paramètres de requête codés en URL avant de les transmettre aux intégrations du backend. Pour les paramètres de requête UTF-8, les API REST décodent les paramètres, puis les transmettent au format Unicode aux intégrations du backend.
- Lorsque vous utilisez la console API Gateway pour tester une API, vous pouvez recevoir une réponse de type « erreurs de point de terminaison inconnu » si un certificat auto-signé est présenté au backend, si le certificat intermédiaire est absent de la chaîne de certificats ou si toute autre exception liée à un certificat non reconnaissable est déclenchée par le backend.
- Pour les entités d'API `Resource` ou `Method` avec une intégration privée, vous devez les supprimer après avoir supprimé toute référence codée en dur de `VpcLink`. Dans le cas contraire, vous disposez d'une intégration instable et recevez une erreur indiquant que le lien du VPC est toujours actif même lorsque l'entité `Resource` ou `Method` est supprimée. Ce comportement ne s'applique pas lorsque l'intégration privée renvoie à `VpcLink` par le biais d'une variable d'étape.
- Les backends suivants peuvent ne pas prendre en charge l'authentification de client SSL d'une manière compatible avec API Gateway :
  - NGINX
  - Heroku
- API Gateway prend en charge la majeur partie de la spécification OpenAPI 2.0 et de la spécification OpenAPI 3.0, avec les exceptions suivantes :
  - Les segments de chemin peuvent contenir uniquement des caractères alphanumériques, des tirets, des points, des virgules, des doubles points et des accolades. Les paramètres de chemin doivent être des segments de chemin distincts. Par exemple, « resource/{path_parameter_name} » est valide ; « resource{path_parameter_name} » ne l'est pas.
  - Les noms de modèle ne peuvent contenir que des caractères alphanumériques.
  - Pour les paramètres d'entrée, seuls les attributs suivants sont pris en charge: `name`, `in`, `required`, `type`, `description`. Les autres attributs sont ignorés.
  - S'il est utilisé, le type `securitySchemes` doit être `apiKey`. Toutefois, l'authentification OAuth 2 et l'authentification de base HTTP sont prises en charge par l'intermédiaire des mécanismes d'autorisation Lambda ; les extensions fournisseur permettent de réaliser la configuration OpenAPI.
  - Le champ `deprecated` n'est pas pris en charge et est supprimé dans les API exportées.
  - Les modèles API Gateway sont définis à l'aide du schéma JSON version 4, plutôt que de celui utilisé par OpenAPI.

- Le paramètre `discriminator` n'est pris en charge dans aucun objet de schéma.
- La balise `example` n'est pas prise en charge.
- `exclusiveMinimum` n'est pas pris en charge par API Gateway.
- Les balises `maxItems` et `minItems` ne sont pas incluses dans une validation de demande simple. Pour contourner ce problème, mettez à jour le modèle après l'importation avant d'effectuer la validation.
- `oneOf` n'est pas pris en charge pour la génération d'OpenAPI 2.0 ou du kit SDK.
- Le champ `readOnly` n'est pas pris en charge.
- `$ref` ne peut pas être utilisé pour référencer d'autres fichiers.
- Les définitions de réponse sous la forme `"500": {"$ref": "#/responses/UnexpectedError"}` ne sont pas prises en charge dans la racine du document OpenAPI. Pour contourner ce problème, remplacez la référence par le schéma en ligne.
- Les nombres de type `Int32` ou `Int64` ne sont pas pris en charge. Voici un exemple :

```
"elementId": {
    "description": "Working Element Id",
    "format": "int32",
    "type": "number"
}
```

- Le type de format décimal (`"format": "decimal"`) n'est pas pris en charge dans une définition de schéma.
- Dans les réponses de méthode, la définition de schéma doit être de type objet et ne peut pas avoir l'un des types primitifs. Par exemple, `"schema": { "type": "string"}` n'est pas pris en charge. Cependant, vous pouvez contourner ce problème à l'aide du type d'objet suivant :

```
"schema": {
     "$ref": "#/definitions/StringResponse"
           }

 "definitions": {
    "StringResponse": {
      "type": "string"
    }
  }
```

  - API Gateway n'utilise pas la sécurité de niveau racine définie dans la spécification OpenAPI. Par conséquent, la sécurité doit être définie au niveau de l'opération pour être appliquée correctement.
  - Le mot-clé `default` n'est pas pris en charge.
- API Gateway adopte les restrictions et limitations suivantes lors de la gestion des méthodes avec intégration Lambda ou HTTP.
  - Les noms d'en-tête et les paramètres de requête sont traités de manière sensible à la casse.
  - Le tableau suivant répertorie les en-têtes qui peuvent être abandonnés, remappés ou modifiés autrement lorsqu'ils sont envoyés au point de terminaison d'intégration ou renvoyés par le point de terminaison d'intégration. Dans ce tableau :
    - `Remapped` signifie que le nom d'en-tête *`<string>`* est remplacé par `X-Amzn-Remapped-`*`<string>`*.

      `Remapped Overwritten` signifie que le nom d'en-tête *`<string>`* est remplacé par `X-Amzn-Remapped-`*`<string>`* et que la valeur est écrasée.

Nom de l'en-tête	Requête (**`http`**/**`http_proxy`** /**`lambda`**)	Réponse (**`http`**/**`http_proxy`** /**`lambda`**)
`Age`	Transmettre	Transmettre
`Accept`	Transmettre	Abandonné / Transmettre/ Transmettre
`Accept-Charset`	Transmettre	Transmettre
`Accept-Encoding`	Transmettre	Transmettre

Nom de l'en-tête	Requête (**http**/**http_proxy** /**lambda**)	Réponse (**http**/**http_proxy** /**lambda**)
`Authorization`	Transmettre *	Remappé
`Connection`	Transmettre/Transmettre/Abandonné	Remappé
`Content-Encoding`	Transmettre/Abandonné/Transmettre	Transmettre
`Content-Length`	Transmette (généré sur la base du corps)	Transmettre
`Content-MD5`	A abandonné	Remappé
`Content-Type`	Transmettre	Transmettre
`Date`	Transmettre	Remappé écrasé
`Expect`	A abandonné	A abandonné
`Host`	Remplacé au point de terminaison d'intégration	A abandonné
`Max-Forwards`	A abandonné	Remappé
`Pragma`	Transmettre	Transmettre
`Proxy-Authenticate`	A abandonné	A abandonné
`Range`	Transmettre	Transmettre

Nom de l'en-tête	Requête (**http**/**http_proxy** /**lambda**)	Réponse (**http**/**http_proxy** /**lambda**)
`Referer`	Transmettre	Transmettre
`Server`	A abandonné	Remappé écrasé
`TE`	A abandonné	A abandonné
`Transfer-Encoding`	Abandonné/Abandonné/Exception	A abandonné
`Trailer`	A abandonné	A abandonné
`Upgrade`	A abandonné	A abandonné
`User-Agent`	Transmettre	Remappé
`Via`	Abandonné/Abandonné/Transmettre	Transmettre/ Abandonné/ Abandonné
`Warn`	Transmettre	Transmettre
`WWW-Authenticate`	A abandonné	Remappé

  * L'en-tête `Authorization` est supprimé s'il contient une signature Signature Version 4 ou si une autorisation `AWS_IAM` est utilisée.

- Le kit SDK Android d'une API générée par API Gateway utilise la classe `java.net.HttpURLConnection`. Cette classe lève une exception non prise en charge, sur les appareils Android 4.4 et version antérieures, dans le cas d'une réponse 401 résultant du remappage de l'en-tête `WWW-Authenticate` en `X-Amzn-Remapped-WWW-Authenticate`.
- Contrairement aux SDK Java, Android et iOS d'une API générés par API Gateway, le JavaScript SDK d'une API générée par API Gateway ne prend pas en charge les nouvelles tentatives pour des erreurs de niveau 500.
- Le test d'appel d'une méthode utilise le type de contenu `application/json` par défaut et ignore les spécifications de tous les autres types de contenu.
- Lorsque vous envoyez des demandes à une API en transmettant l'en-tête `X-HTTP-Method-Override`, API Gateway remplace la méthode. Par conséquent, pour que l'en-tête soit transmise au backend, elle doit être ajoutée à la demande d'intégration.
- Lorsqu'une demande contient plusieurs types de supports dans son en-tête `Accept`, API Gateway respecte uniquement le premier type de support `Accept`. Lorsque vous ne pouvez pas contrôler l'ordre des types de supports `Accept` et que le type de support de votre contenu binaire n'est pas le premier de la liste, vous pouvez ajouter le premier type de support `Accept` dans la liste `binaryMediaTypes` de votre API. API Gateway retourne alors votre contenu sous forme binaire. Par exemple, pour envoyer un fichier JPEG en utilisant un élément `<img>` dans un navigateur, ce dernier peut envoyer `Accept:image/webp,image/*,*/*;q=0.8` dans une demande. Si vous ajoutez `image/webp` à la liste `binaryMediaTypes`, le point de terminaison reçoit le fichier JPEG sous forme binaire.
- La personnalisation de la réponse de passerelle par défaut pour `413 REQUEST_TOO_LARGE` n'est actuellement pas prise en charge.
- API Gateway inclut un en-tête `Content-Type` pour toutes les réponses d'intégration. Par défaut, le type de contenu est `application/json`.

Historique du document

Le tableau suivant décrit les modifications importantes apportées à la documentation depuis la dernière version d'Amazon API Gateway. Pour recevoir une notification sur les mises à jour apportées à cette documentation, vous pouvez vous abonner à un flux RSS en cliquant sur le bouton RSS dans le panneau du menu supérieur.

- Dernière mise à jour de la documentation : 15 février 2024

Modification	Description	Date
Ajout du support pour TLS 1.3	API Gateway prend désormais en charge le protocole TLS 1.3 sur les API REST régionales, les API HTTP et les WebSocket API. Pour plus d'informations, consultez Choisir une politique de sécurité pour votre domaine personnalisé dans API Gateway.	15 février 2024
Mises à jour de l' WebSocket API REST et de la console	Informations de console mises à jour pour les API REST et WebSocket les API.	10 décembre 2023
Mise à jour de la documentation	Mise à jour des informations conceptuelles et création de nouveaux didacticiels pour les transformations de données et la validation de demande pour les API REST API Gateway. Pour en savoir plus, consultez Utilisation de la validation de demande dans API Gateway et Configura	22 juin 2023

	tion de transformations de données pour les API REST.	
Configurer le basculement DNS pour une architecture API Gateway multi-région.	Ajout de la prise en charge de l'utilisation des contrôles de santé d'Amazon Route 53 pour contrôler le basculement du DNS entre une API REST API Gateway dans une région principale Région AWS et une API dans une région secondaire. Pour en savoir plus, consultez Configuration de surveillances de l'état personnalisées pour le basculement DNS.	31 octobre 2022
Mise à jour de la documentation	Mise à jour des résumés des fonctionnalités principales pour les API REST et les API HTTP. Pour en savoir plus, consultez Choix entre les API REST et les API HTTP.	31 mai 2022
Mise à jour de la stratégie gérée	Ajout de la prise en charge de `acm:GetCertificate` pour la stratégie `AWSServiceRoleForAPIGateway` . Pour plus d'informations, consultez Utilisation de rôles liés au service pour API Gateway.	12 Juillet 2021

Mappage de paramètres pour les API HTTP	Ajout de la prise en charge du mappage de paramètres pour les API HTTP. Pour plus d'informations, reportez-vous à Transformer les demandes et les réponses d'API.	7 janvier 2021
Désactivation du point de terminaison par défaut pour une API REST	Ajout de la prise en charge de la désactivation du point de terminaison par défaut pour les API REST. Pour de plus amples informations, veuillez consulter Désactivation du point de terminaison par défaut pour une API REST.	29 octobre 2020
Authentification TLS mutuelle	Ajout de la prise en charge de l'authentification TLS mutuelle pour les API REST et les API HTTP. Pour de plus amples informations, veuillez consulter Configuration de l'authentification TLS mutuelle pour une API REST et Configuration de l'authentification TLS mutuelle pour une API HTTP.	17 septembre 2020
AWS Lambda Autorisateurs d'API HTTP	Ajout de la prise en charge AWS Lambda des autorisateurs pour les API HTTP. Pour plus d'informations, consultez la section Utilisation des AWS Lambda autorisateurs pour les API HTTP.	9 septembre 2020

Intégrations des AWS services d'API HTTP	Ajout de la prise en charge des intégrations de AWS services pour les API HTTP. Pour plus d'informations, consultez la section Utilisation des intégrations de AWS services pour les API HTTP.	20 août 2020
Domaines personnalisés contenant des caractères génériques des API HTTP	Ajout de la prise en charge des noms de domaine personnalisé contenant des caractères génériques pour les API HTTP. Pour de plus amples informations, veuillez consulter Noms de domaine personnalisé contenant des caractères génériques.	10 août 2020
Améliorations du portail pour développeurs sans serveur	Ajout de la gestion des utilisateurs au panneau d'administration et prise en charge de l'exportation des définitions d'API. Pour de plus amples informations, veuillez consulter Utiliser le portail pour développeurs sans serveur pour cataloguer vos API API Gateway.	25 juin 2020
WebSocket `Sec-WebSocket-Protocol` Support de l'API	Ajout de la prise en charge du champ `Sec-WebSocket-Protocol` . Pour plus d'informations, consultez Configuration d'une route $connect qui nécessite un WebSocket sous-protocole.	16 juin 2020

Exportation d'API HTTP	Ajout du support pour l'exportation des définitions OpenAPI 3.0 des API HTTP. Pour de plus amples informations, veuillez consulter Exportation d'une API HTTP à partir d'API Gateway.	20 avril 2020
Documentation sur la sécurité	Ajout d'une documentation sur la sécurité Pour de plus amples informations, veuillez consulter Sécurité dans Amazon API Gateway.	31 mars 2020
Documentation réorganisée	Réorganisation du guide du développeur.	12 mars 2020
Disponibilité générale de l'API HTTP	API HTTP publiées en disponibilité générale. Pour de plus amples informations, veuillez consulter Utilisation des API HTTP.	12 mars 2020
Journalisation de l'API HTTP	Ajout de la prise en charge de `$context.integrationErrorMessage` pour les journaux de l'API HTTP. Pour de plus amples informations, veuillez consulter Variables de journalisation de l'API HTTP.	26 février 2020
AWS variables pour l'importation d'OpenAPI	Ajout de la prise en charge AWS des variables dans les définitions d'OpenAPI. Pour plus d'informations, voir Variables AWS pour l'importation OpenAPI.	17 février 2020

API HTTP	API HTTP publiées en version bêta. Pour de plus amples informations, veuillez consulter API HTTP.	4 décembre 2019
Noms de domaine personnalisés génériques	Ajout de la prise en charge des noms de domaine personnalisés génériques. Pour de plus amples informations, veuillez consulter Noms de domaine personnalisé contenant des caractères génériques.	21 octobre 2019
Journalisation d'Amazon Data Firehose	Ajout de la prise en charge d'Amazon Data Firehose en tant que destination pour les données de journalisation des accès. Pour plus d'informations, consultez Utiliser Amazon Data Firehose comme destination pour la journalisation des accès à API Gateway.	15 octobre 2019
Alias Route53 pour l'appel d'API privées	Ajout de la prise en charge d'enregistrements DNS d'alias Route53 supplémentaires pour l'appel d'API privées. Pour de plus amples informations, veuillez consulter Accès à votre API privée à l'aide de l'alias Route53.	18 septembre 2019

Contrôle d'accès basé sur des balises pour les API WebSocket	Ajout de la prise en charge du contrôle d'accès basé sur des balises pour les WebSocket API. Pour de plus amples informations, veuillez consulter Ressources API Gateway pouvant être balisées.	27 juin 2019
Sélection de la version TLS pour des domaines personnalisés	Ajout de la prise en charge de la sélection de la version TLS (Transport Layer Security) pour les API déployées sur des domaines personnalisés. Consultez la note dans les informations sur le choix d'une version TLS minimum pour un domaine personnalisé dans API Gateway.	20 juin 2019
Stratégies de point de terminaison de VPC pour des API privées	Ajout de la prise en charge pour l'amélioration de la sécurité des API privées par attachement de stratégies de point de terminaison à des points de terminaison de VPC. Pour de plus amples informations, veuillez consulter Utilisation de stratégies de point de terminaison de VPC pour des API privées dans API Gateway.	4 juin 2019
Documentation mise à jour	Réécriture de Mise en route avec Amazon API Gateway. Déplacement des tutoriels dans Tutoriels Amazon API Gateway.	29 mai 2019

Contrôle d'accès basé sur les balises pour les API REST	Ajout de la prise en charge du contrôle d'accès basé sur les balises pour les API REST. Pour de plus amples informations, veuillez consulter Utilisation des balises avec des stratégies IAM pour contrôler l'accès aux ressources API Gateway.	23 mai 2019
Documentation mise à jour	Réécriture de six rubriques : Qu'est-ce qu'Amazon API Gateway ?, Tutoriel : Création d'une API avec l'intégration de proxy HTTP, Tutoriel : Création d'une API REST Calc avec trois intégrations autres que de proxy, Modèle de mappage API Gateway et référence à la variable de journalisation des accès, Utilisation des mécanismes d'autorisation Lambda API Gateway et Activation de CORS pour une ressource d'API REST API Gateway.	5 avril 2019

Améliorations du portail pour développeurs sans serveur	Ajout du panneau d'administration et d'autres améliorations afin de faciliter la publication des API dans le portail pour développeurs Amazon API Gateway. Pour de plus amples informations, veuillez consulter Utilisation d'un portail pour développeurs pour cataloguer vos API.	28 mars 2019
Support pour AWS Config	Ajout du support pour AWS Config. Pour plus d'informations, consultez la section Surveillance de la configuration de l'API API Gateway avec AWS Config.	20 mars 2019
Support pour AWS CloudFormation	L'API API Gateway V2 a été ajoutée à la référence du AWS CloudFormation modèle. Pour de plus amples informations, veuillez consulter Référence des types de ressources Amazon API Gateway V2.	7 février 2019
Support pour les WebSocket API	Ajout du support pour les WebSocket API. Pour plus d'informations, consultez Création d'une WebSocket API dans Amazon API Gateway.	18 décembre 2018

Portail de développement sans serveur disponible via AWS Serverless Application Repository	L'application sans serveur du portail des développeurs Amazon API Gateway est désormais disponible auprès du AWS Serverless Application Repository(en plus de GitHub). Pour de plus amples informations, veuillez consulter la section Utilisation d'un portail pour développeurs pour cataloguer vos API Amazon API Gateway.	16 novembre 2018
Support pour AWS WAF	Ajout de la prise en charge de AWS WAF (pare-feu d'applications web). Pour plus d'informations, consultez Contrôler l'accès à une API à l'aide de AWS WAF.	5 novembre 2018
Portail pour développeurs sans serveur	Amazon API Gateway fournit désormais un portail pour développeurs entièrement personnalisable sous la forme d'une application sans serveur que vous pouvez déployer pour la publication de vos API Amazon API Gateway. Pour de plus amples informations, veuillez consulter la section Utilisation d'un portail pour développeurs pour cataloguer vos API Amazon API Gateway.	29 octobre 2018

Prise en charge des en-têtes à valeurs multiples et des paramètres de chaîne de requête	Amazon API Gateway prend désormais en charge les en-têtes multiples et les paramètres de chaîne de requête portant le même nom. Pour de plus amples informations, veuillez consulter Prise en charge des en-têtes à valeurs multiples et des paramètres de chaîne de requête.	4 octobre 2018
Prise en charge d'OpenAPI	Amazon API Gateway prend désormais en charge OpenAPI 3.0 ainsi que OpenAPI (Swagger) 2.0.	27 septembre 2018
Documentation mise à jour	Ajout d'une nouvelle rubrique : Comment les stratégies de ressources Amazon API Gateway affectent le flux de travail d'autorisation.	le 27 septembre 2018
AWS X-Ray Intégration active	Vous pouvez désormais l'utiliser AWS X-Ray pour suivre et analyser les latences des demandes des utilisateurs lorsqu'elles transitent par vos API vers les services sous-jacents. Pour plus d'informations, consultez Trace API Gateway API Execution with AWS X-Ray.	6 septembre 2018

Améliorations de la mise en cache	Lorsque vous activez la mise en cache pour une étape d'API, elle est activée seulement pour les méthodes `GET` par défaut. Cela permet d'assurer la sécurité de votre API. Vous pouvez activer la mise en cache pour d'autres méthodes en remplaçant les paramètres de méthode. Pour de plus amples informations, veuillez consulter Activation de la mise en cache des API pour améliorer la réactivité.	20 août 2018
Révision des limites de service	Plusieurs limites ont été révisées : augmentation du nombre d'API par compte. Augmentation des limites de taux pour les API Create/Import/Deploy (API de création, importation, déploiement). Certains taux à la minute sont désormais par seconde. Pour de plus amples informations, veuillez consulter Limites.	13 juillet 2018

Remplacement des en-têtes et des paramètres de réponse et de requête d'API	Ajout de la prise en charge du remplacement des en-têtes de requête, des chaînes d'interrogation et des chemins d'accès, ainsi que des en-têtes et codes de statut des réponses. Pour de plus amples informations, veuillez consulter Utiliser un modèle de mappage pour substituer les en-têtes et paramètres des requêtes et réponses d'une API.	12 juillet 2018
Limitation au niveau méthode pour les plans d'utilisation	Ajout de la prise en charge de la définition de limitations de méthode par défaut et de limitations concernant des méthodes d'API spécifiques dans les paramètres d'un plan d'utilisation. Ces paramètres viennent en plus des limitations existantes au niveau du compte et des limitations par défaut au niveau méthode que vous pouvez définir dans les paramètres d'étape. Pour de plus amples informations, veuillez consulter Limitation des demandes d'API pour améliorer le débit.	11 juillet 2018

Notifications de mise à jour du Manuel du développeur API Gateway désormais disponibles via RSS	La version HTML du Manuel du développeur API Gateway prend désormais en charge un flux RSS des mises à jour qui sont documentées sur cette page Historique du document. Le flux RSS inclut les mises à jour effectuées à partir du 27 juin 2018. Les mises à jour annoncées précédemment sont toujours disponibles sur cette page. Utilisez le bouton RSS dans le panneau du menu supérieur pour vous abonner au flux.	27 juin 2018

Mises à jour antérieures

Le tableau ci-après décrit les modifications importantes apportées dans chaque version du Manuel du développeur API Gateway avant le 27 juin 2018.

Modification	Description	Date de modification
API privées	Ajout de la prise en charge des API privées, que vous exposez via les points de terminaison d'un VPC d'interface. Le trafic vers votre API privée ne quitte pas le réseau Amazon ; il est isolé de l'Internet public.	14 juin 2018
Autorisations et intégrations Lambda entre comptes et autorisations de groupe d'utilisateurs Amazon Cognito	Utilisez une AWS Lambda fonction d'un autre AWS compte comme fonction d'autorisation Lambda ou comme backend d'intégration d'API. Ou utilisez un groupe d'utilisateurs Amazon Cognito en tant que mécanisme d'autorisation. L'autre compte peut se trouver dans n'importe quelle région où Amazon API Gateway est disponible. Pour plus d'informations, consultez the section called “Configuration	2 avril 2018

Modification	Description	Date de modification
	d'un mécanisme d'autorisation Lambda entre comptes”, the section called “Tutoriel : Création d'une API avec une intégration de proxy Lambda entre comptes” et the section called “Configuration d'un mécanisme d'autorisation Amazon Cognito entre comptes pour une API REST”.	
Stratégies de ressources pour les API	Utilisez les stratégies de ressource API Gateway pour permettre aux utilisateurs d'un autre compte AWS d'acc éder en toute sécurité à votre API ou de permettre de faire appel à l'API uniquement à partir de plages d'adresses IP source ou de blocs CIDR spécifiés. Pour de plus amples informations, veuillez consulter the section called “Utilisat ion des stratégies de ressources API Gateway”.	2 avril 2018
Balisage pour les ressources API Gateway	Balisez une étape d'API avec jusqu'à 50 balises pour la répartition des coûts des demandes d'API et la mise en cache dans API Gateway. Pour de plus amples informati ons, veuillez consulter the section called “Configurer des balises”.	19 décembre 2017
Compression et décompression de la charge utile	Activez l'appel de votre API avec des charges utiles compressées à l'aide de l'un des codages de contenu pris en charge. Les charges utiles compressées sont soumises au mappage si un modèle de mappage de corps est spécifié. Pour de plus amples informations, veuillez consulter the section called “Encodage de contenu”.	19 décembre 2017
clé API provenant d'un Custom authorizer	Renvoyez une clé API personnalisée d'un Custom authorizer à API Gateway pour appliquer un plan d'utilisa tion pour les méthodes d'API qui nécessitent la clé. Pour de plus amples informations, veuillez consulter the section called “Choisir une source de clé d'API”.	19 décembre 2017

Modification	Description	Date de modification
Autorisation avec des règles OAuth 2	Activez l'autorisation d'appel de méthode à l'aide des règles OAuth 2 et du mécanisme d'autorisation `COGNITO_USER_POOLS` . Pour de plus amples informations, veuillez consulter the section called “Utilisation d'un groupe d'utilisateurs Amazon Cognito en tant que mécanisme d'autorisation pour une API REST”.	14 décembre 2017
Intégration privée et lien VPC	Créez une API avec l'intégration privée API Gateway pour permettre aux clients d'accéder aux ressources HTTP/HTTPS d'un Amazon VPC depuis l'extérieur du VPC via une ressource. VpcLink Pour de plus amples informations, veuillez consulter the section called “Tutoriel : Création d'une API avec intégration privée” et the section called “Intégration privée”.	30 novembre 2017
Déployez une version Canary pour le test d'API	Ajoutez une version Canary à un déploiement d'API existant afin de tester une version plus récente de l'API tout en conservant la version actuelle en fonctionnement à la même étape. Vous pouvez définir un pourcentage du trafic d'étape pour la version de Canary et activer l'exécution et l'accès spécifiques à Canary dans des CloudWatch journaux journaux séparés. Pour plus d’informations, consultez the section called “Configuration du déploiement d'une version Canary”.	28 novembre 2017
Consignation des accès	Enregistrez l'accès client à votre API avec des données provenant de variables de $contexte dans un format de votre choix. Pour de plus amples informations, veuillez consulter the section called “CloudWatch journaux”.	21 novembre 2017

Modification	Description	Date de modification
Kit SDK Ruby d'une API	Générez un kit SDK Ruby pour votre API et utilisez-le pour appeler vos méthodes d'API. Pour de plus amples informations, veuillez consulter the section called “Génération du kit SDK Ruby d'une API” et the section called “Utilisation d'un kit SDK Ruby généré par API Gateway pour une API REST”.	20 novembre 2017
Point de terminaison d'API régional	Spécifiez un point de terminaison d'API régionale pour créer une API pour les clients non mobiles. Un client non mobile, comme une instance EC2, s'exécute dans la Région AWS où l'API est déployée. Comme pour une API optimisée pour les périphériques, vous pouvez créer un nom de domaine personnalisé pour une API régionale. Pour de plus amples informations, veuillez consulter the section called “Types de points de terminaison API Gateway” et the section called “Configuration d'un nom de domaine personnalisé régional”.	2 novembre 2017
Mécanisme d'autorisation personnalisée des demandes	Utilisez le mécanisme d'autorisation personnalisée des demandes pour transmettre les informations d'authentification de l'utilisateur dans les paramètres de la demande afin d'autoriser les appels de méthode d'API. Les paramètres de la demande comprennent les en-têtes et les chaînes d'interrogation, ainsi que les variables d'étape et de contexte. Pour de plus amples informations, veuillez consulter Utilisation des mécanismes d'autorisation Lambda API Gateway.	15 septembre 2017

Modification	Description	Date de modification
Personnalisation de réponses de passerelle	Personnalisez les réponses de passerelle générées par API Gateway aux demandes d'API qui n'ont pas pu atteindre le backend d'intégration. Un message de passerelle personnalisé peut fournir à l'appelant des messages d'erreur personnalisés propres à l'API, avec notamment le renvoi des en-têtes CORS nécessaires, ou peut transformer les données de la passerelle de réponse en un format d'échange externe. Pour de plus amples informations, veuillez consulter Configuration de réponses de passerelle pour personnaliser des réponses d'erreur.	6 juin 2017
Mappage des propriétés d'erreur personnalisée Lambda à des en-têtes de réponse de méthode	Mappez les différentes propriétés d'erreur personnalisée renvoyées par Lambda à des paramètres d'en-tête de réponse de méthode à l'aide du paramètre `integration.response.body` , en vous appuyant sur API Gateway pour désérialiser l'objet d'erreur personnalisée obtenu à l'aide de stringify au moment de l'exécution. Pour de plus amples informations, veuillez consulter Gestion des erreurs Lambda personnalisées dans API Gateway.	6 juin 2017
Augmentation des limitations	Augmentez la limite de taux de demandes à un taux régulier au niveau du compte à 10 000 demandes par seconde (rps) et la limite de transmission en rafale à 5 000 demandes simultanées. Pour de plus amples informations, veuillez consulter Limiter les demandes d'API pour améliorer le débit.	6 juin 2017

Modification	Description	Date de modification
Validation des demandes de méthode	Configurez des valideurs de demande de base aux niveaux d'API ou de méthode de sorte qu'API Gateway puisse valider les demandes entrantes. API Gateway vérifie que les paramètres obligatoires sont définis et non à blanc, puis il vérifie que le format des charges utiles applicables est conforme au modèle configuré. Pour de plus amples informations, veuillez consulter Utilisation de la validation des demandes dans API Gateway.	11 avril 2017
Intégration à ACM	Utilisez des certificats ACM pour les noms de domaine personnalisés de votre API. Vous pouvez créer un certifica t AWS Certificate Manager ou importer un certificat existant au format PEM dans ACM. Vous pouvez ensuite faire référence à l'ARN du certificat lors de la définition d'un nom de domaine personnalisé pour vos API. Pour de plus amples informations, veuillez consulter Configuration des noms de domaine personnalisés pour les API REST.	9 mars 2017
Génération et appel d'un SDK Java d'une API	Permettez à API Gateway de générer le SDK Java pour votre API et utilisez ce SDK pour appeler l'API dans votre client Java. Pour de plus amples informations, veuillez consulter Utilisation d'un kit SDK Java généré par API Gateway pour une API REST.	13 janvier 2017
Intégration avec AWS Marketplace	Vendez votre API dans le cadre d'un plan d'utilisation en tant que produit SaaS via AWS Marketplace. Utilisez AWS Marketplace pour étendre la portée de votre API. Fiez-vous AWS Marketplace à la facturation des clients en votre nom. Laissez API Gateway gérer l'autorisation utilisateur et la mesure de l'utilisation. Pour de plus amples informations, veuillez consulter Vendre vos API en tant que SaaS.	1 décembre 2016

Modification	Description	Date de modification
Activation de la documentation pour votre API	Ajoutez de la documentation pour les entités d'API dans les DocumentationPartressources d'API Gateway. Associez un instantané des `DocumentationPart` instances de collection à une étape d'API pour créer un DocumentationVersion. Publiez une documentation de l'API en exportant une version de la documentation dans un fichier externe, par exemple un fichier Swagger. Pour de plus amples informations, veuillez consulter Documentation sur les API REST.	1 décembre 2016
Mécanisme d'autorisation personnalisée mis à jour	Une fonction Lambda du mécanisme d'autorisation personnalisée retourne désormais l'identifiant principal de l'appelant. La fonction peut également retourner d'autres informations sous forme de paires clé-valeur du mappage `context` et de stratégie IAM. Pour de plus amples informations, veuillez consulter Résultat d'un autorisateur Lambda API Gateway.	1 décembre 2016
Prise en charge des charges utiles binaires	Définissez binaryMediaTypesvotre API pour prendre en charge les charges utiles binaires d'une demande ou d'une réponse. Définissez la `contentHandling` propriété sur une intégration ou spécifiez IntegrationResponses'il faut gérer une charge utile binaire en tant que blob binaire natif, en tant que chaîne encodée en Base64 ou en tant que transmission sans modification. Pour plus d'informations, consultez Utilisation des types de médias binaires pour les API REST.	17 novembre 2016

Modification	Description	Date de modification
Activation de l'intégration de proxy à un backend HTTP ou Lambda via une ressource de proxy d'une API.	Créez une ressource proxy avec un paramètre de chemin gourmand au format `{proxy+}` et la méthode `ANY` fourre-tout. La ressource proxy est intégrée à un backend HTTP ou Lambda en utilisant l'intégration proxy HTTP ou Lambda, respectivement. Pour de plus amples informations, veuillez consulter Configuration de l'intégration de proxy avec une ressource de proxy.	20 septembre 2016
Extension des API sélectionnées dans API Gateway en tant qu'offres de produits pour vos clients en fournissant un ou plusieurs plans d'utilisation.	Créez un plan d'utilisation dans API Gateway pour permettre aux clients des API sélectionnées d'accéder aux étapes d'API spécifiées selon les quotas et les taux de demandes convenus. Pour de plus amples informations, veuillez consulter Création et utilisation de plans d'utilisation avec des clés d'API.	11 août 2016
Activation de l'autorisation au niveau de la méthode avec un groupe d'utilisateurs dans Amazon Cognito	Créez un groupe d'utilisateurs dans Amazon Cognito et utilisez-le comme votre propre fournisseur d'identités. Vous pouvez configurer le groupe d'utilisateurs en tant que mécanisme d'autorisation au niveau de la méthode pour accorder l'accès aux utilisateurs qui sont inscrits dans ce groupe d'utilisateurs. Pour de plus amples informations, veuillez consulter Contrôle de l'accès à une API REST à l'aide de groupes d'utilisateurs Amazon Cognito en tant que mécanisme d'autorisation.	28 juillet 2016
Activation CloudWatch des métriques et des dimensions Amazon sous l'espace de `AWS/ApiGateway` noms.	Les métriques d'API Gateway sont désormais standardisées sous l' CloudWatch espace de noms de`AWS/ApiGateway` . Vous pouvez les consulter à la fois dans la console API Gateway et dans la CloudWatch console Amazon. Pour plus d'informations, consultez Dimensions et métriques Amazon API Gateway.	28 juillet 2016

Modification	Description	Date de modification
Activation de la rotation de certificat pour un nom de domaine personnalisé.	La rotation de certificat vous permet de charger et de renouveler un certificat arrivant à expiration pour un nom de domaine personnalisé. Pour de plus amples informations, veuillez consulter Rotation d'un certificat importé dans ACM.	27 avril 2016
Documentation des modifications de la console Amazon API Gateway mise à jour.	Découvrez comment créer et configurer une API à l'aide de la console API Gateway mise à jour. Pour de plus amples informations, veuillez consulter Tutoriel : Création d'une API REST par l'importation d'un exemple et Tutoriel : Création d'une API REST avec une intégration HTTP autre que de proxy.	5 avril 2016
Activation de la fonction d'importation d'API pour créer ou mettre à jour une API à partir de définitions d'API externes.	La fonction d'importation d'API vous permet de créer une API ou de mettre à jour une API existante en chargeant une définition d'API externe au format Swagger 2.0 avec les extensions API Gateway. Pour plus d'informations sur la fonction d'importation d'API, consultez Configuration d'une API REST à l'aide d'OpenAPI.	5 avril 2016

Modification	Description	Date de modification
Exposition de la variable `$input.body` pour accéder à la charge utile brute sous forme de chaîne et à la fonction `$util.parseJson()` pour convertir une chaîne JSON en objet JSON dans un modèle de mappage.	Pour plus d'informations sur `$input.body` et `$util.parseJson()`, consultez Modèle de mappage API Gateway et référence à la variable de journalisation des accès.	5 avril 2016
Activation des demandes de client avec invalidation du cache au niveau de la méthode et amélioration de la gestion des limitations de demandes.	Videz le cache au niveau d'une étape d'API et invalidez des entrées de cache individuelles. Pour de plus amples informations, veuillez consulter Vidage du cache d'étape d'API dans API Gateway et Invalidation d'une entrée de cache API Gateway. Améliorez l'expérience de la console en matière de gestion des limitations de demandes d'API. Pour de plus amples informations, veuillez consulter Limiter les demandes d'API pour améliorer le débit.	25 mars 2016
Activation et appel de l'API Amazon API Gateway à l'aide de l'autorisation personnalisée.	Créez et configurez une AWS Lambda fonction pour implémenter une autorisation personnalisée. Cette fonction renvoie un document de stratégie IAM qui accorde les autorisations Allow ou Deny aux demandes de client d'une API Amazon API Gateway. Pour de plus amples informations, veuillez consulter Utilisation des mécanismes d'autorisation Lambda API Gateway.	11 février 2016

Modification	Description	Date de modification
Importation et exportation d'API Amazon API Gateway à l'aide d'un fichier de définition Swagger et des extensions API Gateway.	Créez et mettez à jour votre API Amazon API Gateway à l'aide de la spécification Swagger avec les extensions API Gateway. Importez les définitions Swagger à l'aide d'API Gateway Importer. Exportez une API Amazon API Gateway vers un fichier de définition Swagger à l'aide de la console API Gateway ou de l'API Amazon API Export. Pour de plus amples informations, veuillez consulter Configuration d'une API REST à l'aide d'OpenAPI et Exportation d'une API REST à partir d'API Gateway.	18 décembre 2015
Mappage du corps de demande ou de réponse, ou des champs JSON du corps, sur les paramètres de demande ou de réponse.	Mappez le corps de la demande de méthode ou ses champs JSON dans le chemin de la demande d'intégration, la chaîne de requête ou les en-têtes. Mappez le corps de la réponse d'intégration ou ses champs JSON dans les en-têtes de la réponse à la demande. Pour de plus amples informations, veuillez consulter Guide de référence du mappage des données de demande d'API et de réponse Amazon API Gateway.	18 décembre 2015
Utilisation de variables de stage dans Amazon API Gateway	Découvrez comment associer des attributs de configuration à une étape de déploiement d'une API dans Amazon API Gateway. Pour de plus amples informations, veuillez consulter Configuration de variables d'étape pour le déploiement d'API REST.	5 novembre 2015
Procédure : Activation de CORS pour une méthode	Il est maintenant plus facile d'activer le partage des ressources cross-origin (CORS) pour les méthodes dans Amazon API Gateway. Pour de plus amples informations, veuillez consulter Activation de CORS pour une ressource de l'API REST.	3 novembre 2015

Modification	Description	Date de modification
Procédure : Utilisation de l'authentification SSL côté client	Utilisez Amazon API Gateway pour générer des certificats SSL que vous pouvez utiliser pour authentifier les appels envoyés à votre backend HTTP. Pour de plus amples informations, veuillez consulter Génération et configuration d'un certificat SSL pour l'authentification backend.	22 septembre 2015
Intégration fictive de méthodes	Découvrez comment intégrer une API avec Amazon API Gateway. Cette fonction permet aux développeurs de générer des réponses d'API directement depuis API Gateway sans nécessiter au préalable aucun backend d'intégration finale.	1er septembre 2015
Prise en charge d'Amazon Cognito Identity	Amazon API Gateway a étendu la portée de la variable `$context` de sorte qu'elle renvoie désormais des informations sur Amazon Cognito Identity lorsque les demandes sont signées avec des informations d'identification Amazon Cognito. De plus, nous avons ajouté une `$util` variable permettant d'échapper aux caractères et de coder JavaScript les URL et les chaînes. Pour plus d'informations, consultez Modèle de mappage API Gateway et référence à la variable de journalisation des accès.	28 août 2015
Intégration de Swagger	Utilisez l'outil d'importation Swagger pour importer les GitHub définitions d'API Swagger dans Amazon API Gateway. Consultez Utilisation des extensions API Gateway vers OpenAPI pour créer et déployer des API et des méthodes à l'aide de l'outil d'importation. L'outil d'importation Swagger vous permet également de mettre à jour des API existantes.	21 juillet 2015
Référence de modèle de mappage	Découvrez le paramètre `$input` et ses fonctions dans Modèle de mappage API Gateway et référence à la variable de journalisation des accès.	18 juillet 2015

Modification	Description	Date de modification
Première version publique	Il s'agit de la première version publique du Manuel du développeur API Gateway.	9 juillet 2015

Glossaire AWS

Pour connaître la terminologie la plus récente d'AWS, consultez le Glossaire AWS dans la Référence Glossaire AWS.

Les traductions sont fournies par des outils de traduction automatique. En cas de conflit entre le contenu d'une traduction et celui de la version originale en anglais, la version anglaise prévaudra.