aws

Guide du développeur

Amazon API Gateway



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

Amazon API Gateway: Guide du développeur

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

Les marques commerciales et la présentation commerciale d’Amazon ne peuvent pas être utilisées en relation avec un produit ou un service extérieur à Amazon, d’une manière susceptible d’entraîner une confusion chez les clients, ou d’une manière qui dénigre ou discrédite Amazon. Toutes les autres marques commerciales qui ne sont pas la propriété d'Amazon sont la propriété de leurs propriétaires respectifs, qui peuvent ou non être affiliés ou connectés à Amazon, ou sponsorisés par Amazon.

Table of Contents

Qu'est-ce qu'Amazon API Gateway ? ... 1
Architecture API Gateway ... 2
Fonctions d'API Gateway ... 3
Cas d'utilisation d'API Gateway ... 3
Utilisation d'API Gateway pour créer des API REST ... 4
Utilisation d'API Gateway pour créer des API HTTP ... 5
Utiliser API Gateway pour créer des WebSocket API ... 5
Qui utilise API Gateway ? ... 6
Accès à API Gateway ... 7
Fait partie de AWS infrastructure sans serveur ... 7
Comment démarrer avec Amazon API Gateway ... 8
Concepts API Gateway ... 8
Choix entre les API HTTP et les API REST ... 14
Type de point de terminaison ... 15
Sécurité ... 15
Autorisation ... 15
Gestion des API ... 16
Développement ... 17
Contrôle ... 17
Intégrations ... 18
Prise en main de la console d'API REST ... 19
Étape 1 : créer une fonction Lambda ... 20
Étape 2 : Créer une API REST ... 21
Étape 3 : créer une intégration de proxy Lambda ... 21
Étape 4 : déployer votre API ... 22
Étape 5 : Invoquer l'API ... 22
(Facultatif) Étape 6 : nettoyer ... 23
Conditions préalables ... 25
Inscrivez-vous pour un Compte AWS ... 25
Mise en route ... 26
Étape 1 : créer une fonction Lambda ... 27
Étape 2 : créer une API HTTP ... 27
Étape 3 : tester votre API ... 28
(Facultatif) Étape 4 : nettoyer ... 29

Étapes suivantes 31
Didacticiels et ateliers 32
Didacticiels sur les API REST 33
Choix d'un didacticiel d'intégration Lambda 33
Didacticiel : création d'une API REST par l'importation d'un exemple 58
Didacticiel sur la sélection d'une intégration HTTP 68
Didacticiel : création d'une API REST avec une intégration privée 82
Tutoriel : Création d'une API REST avec une AWS intégration 88
Didacticiel : création d'une API REST de calculatrice avec trois intégrations 94
Didacticiel : création d'une API REST en tant que proxy Amazon S3 122
Didacticiel : création d'une API REST en tant que proxy Amazon Kinesis 167
Tutoriel : Création d'une API REST à l'aide de AWS SDK ou AWS CLI 213
Didacticiel : création d'une API REST privée 247
Didacticiels sur l'API HTTP 254
Didacticiel : création d'une API HTTP CRUD avec Lambda et DynamoDB 254
Didacticiel : création d'une API HTTP avec une intégration privée à un service Amazon ECS 267
Didacticiels sur les API WebSocket 274
Tutoriel : Création d'une application de WebSocket chat 275
Tutoriel : Création d'une WebSocket API avec une AWS intégration 280
API REST API Gateway 295
Développement 295
Exemple de ressource pour une API REST 296
Autres fonctionnalités pour le développement des API REST 299
Types de points de terminaison API Gateway 301
Stratégies de sécurité 305
Types d'adresses IP pour les API REST dans API Gateway 327
Méthodes 329
Contrôle d'accès 350
Intégrations 441
Validation des requêtes 537
Transformations de données 567
Réponses de passerelle 652
CORS 665
Types de supports binaires 681
Invocation 714

OpenAPI ... 747
Publication ... 762
Déploiement d'une API REST ... 763
Noms de domaine personnalisés ... 811
Optimisez ... 887
Paramètres du cache ... 887
Encodage de contenu ... 901
Distribuer ... 907
Plans d'utilisation ... 908
Documentation sur les API ... 935
Génération de kits SDK ... 1002
Vendez votre produit APIs en tant que SaaS ... 1029
Protection ... 1034
Exiger des certificats clients ... 1034
Certificats clients principaux ... 1042
AWS WAF ... 1091
étranglement ... 1094
API REST privées ... 1097
Surveillance ... 1158
CloudWatch métriques ... 1159
CloudWatch journaux ... 1168
X-Ray ... 1200
Portails API Gateway ... 1215
Considérations ... 1215
Produit de portail ... 1216
Exemple de produit de portail d'adoption d'animaux de compagnie ... 1216
Étapes suivantes ... 1218
Création d'un produit de portail ... 1219
Création d'un point de terminaison REST du produit ... 1221
Création d'une page produit ... 1223
Mettre à jour un produit de portail ... 1225
Mettre à jour un terminal REST du produit ... 1226
Mettre à jour la page du produit ... 1227
Partagez les produits du portail ... 1227
Activez Essayez-le pour un point de terminaison REST du produit sur votre portail ... 1233
Supprimer un produit de portail ... 1234

Création d'un portail 1235
Choisissez un nom de domaine 1235
Création d'un portail 1236
Mettre à jour un portail 1239
Mettre à jour un portail 1239
Afficher un aperçu d'un portail 1239
Considérations 1240
Afficher un aperçu d'un portail 1240
Publier un portail 1240
Considérations 1240
Publier un portail 1241
Utiliser un portail 1241
Utiliser un portail 1241
Résoudre les problèmes liés à votre portail 1242
Désactiver un portail 1243
Désactiver un portail 1243
Supprimer un portail 1244
Supprimer un portail 1244
API HTTP API Gateway 1245
Développement 1245
Création d’une API HTTP 1246
Routes 1247
Types d’adresses IP pour les API HTTP dans API Gateway 1250
Contrôle d’accès 1252
Intégrations 1272
CORS 1291
Mappage de paramètres 1294
OpenAPI 1301
Publication 1311
Stages 1312
Politique de sécurité pour le protocole HTTP APIs dans API Gateway 1315
Noms de domaine personnalisés 1317
Protection 1327
étranglement 1327
Authentification TLS mutuelle 1329
Surveillance 1335

Métriques 1336
Logging 1339
Résolution des problèmes 1350
Intégrations Lambda 1350
Mécanismes d’autorisation JWT 1353
API WebSocket API Gateway 1355
Vue d'ensemble des WebSocket API 1355
Utiliser des routes pour une WebSocket API 1356
Envoi de données à des applications client connectées 1357
WebSocket Codes d'état de l'API 1357
Gestion des utilisateurs et des applications client connectées 1358
Invocation de votre intégration backend 1362
WebSocket expressions de sélection 1366
Développement 1377
Création et configuration 1378
Types d’adresses IP pour les API WebSocket dans API Gateway 1379
Routes 1381
Contrôle d’accès 1390
Intégrations 1399
Validation des requêtes 1408
Transformations de données 1412
Types de supports binaires 1425
Invocation 1425
Publication 1429
Stages 1429
Déployer une WebSocket API 1432
Politique de sécurité pour WebSocket APIs in API Gateway 1435
Noms de domaine personnalisés 1437
Protection 1445
Limitation au niveau du compte par région 1445
Limitation au niveau de l’acheminement 1446
Surveillance 1446
Métriques 1447
Logging 1449
API Gateway ARNs 1458
API HTTP et ressources WebSocket d'API 1458

Ressources API REST 1461
Ressources du nom de domaine 1465
execute-api(HTTP APIs WebSocket APIs et REST APIs) 1466
Extensions OpenAPI 1468
x-amazon-apigateway-any-method 1469
Exemples x-amazon-apigateway-any-method 1470
x-amazon-apigateway-cors 1471
Exemple x-amazon-apigateway-cors 1471
x-amazon-apigateway-api-key-source 1472
Exemple x-amazon-apigateway-api-key-source 1473
x-amazon-apigateway-auth 1474
Exemple x-amazon-apigateway-auth 1474
x-amazon-apigateway-authorizer 1475
Exemples x-amazon-apigateway-authorizer pour les API REST 1479
Exemples x-amazon-apigateway-authorizer pour les API HTTP 1483
x-amazon-apigateway-authtype 1484
Exemple de propriété x-amazon-apigateway-authtype 1485
Voir aussi 1487
x-amazon-apigateway-binary-media-type 1487
Exemple x-amazon-apigateway-binary-media-types 1487
x-amazon-apigateway-documentation 1487
Exemple x-amazon-apigateway-documentation 1488
mode d'accès à x-amazon-apigateway-endpoint-endpoint 1488
Exemple de mode d'accès x-amazon-apigateway-endpoint-point 1489
x-amazon-apigateway-endpoint-configuration 1489
Exemples x-amazon-apigateway-endpoint-configuration 1490
x-amazon-apigateway-gateway-responses 1491
Exemple x-amazon-apigateway-gateway-responses 1491
x-amazon-apigateway-gateway-responses.gatewayResponse 1492
Exemple x-amazon-apigateway-gateway-responses.gatewayResponse 1492
x-amazon-apigateway-gateway-responses.responseParameters 1493
Exemple x-amazon-apigateway-gateway-responses.responseParameters 1493
x-amazon-apigateway-gateway-responses.responseTemplates 1494
Exemple x-amazon-apigateway-gateway-responses.responseTemplates 1494
x-amazon-apigateway-importexport-version 1495
x-amazon-apigateway-importexport-version example 1495

x-amazon-apigateway-integration 1495
Exemples d'objet x-amazon-apigateway-integration 1503
x-amazon-apigateway-integrations 1505
Exemple x-amazon-apigateway-integrations 1505
x-amazon-apigateway-integration.requestTemplates 1507
Exemple d'objet x-amazon-apigateway-integration.requestTemplates 1507
x-amazon-apigateway-integration.requestParameters 1508
Exemple X-Amazon-APIGateway-Integration.RequestParameters 1509
x-amazon-apigateway-integration.responses 1510
Exemple x-amazon-apigateway-integration.responses 1511
x-amazon-apigateway-integration.response 1512
Exemple x-amazon-apigateway-integration.response 1513
x-amazon-apigateway-integration.responseTemplates 1514
Exemple d'objet x-amazon-apigateway-integration.responseTemplate 1514
x-amazon-apigateway-integration.responseParameters 1515
Exemple X-Amazon-APIGateway-Integration.ResponseParameters 1515
x-amazon-apigateway-integration.tlsConfig 1515
Exemples x-amazon-apigateway-integration.tlsConfig 1518
x-amazon-apigateway-minimum-compression-size 1519
x-amazon-apigateway-minimum-compression-size example 1519
x-amazon-apigateway-policy 1519
Exemple de politique x-amazon-apigateway-policy 1519
x-amazon-apigateway-request-validator 1520
Exemple de validateur x-amazon-apigateway-request-validator 1521
x-amazon-apigateway-request-validators 1521
Exemple de x-amazon-apigateway-request-validators 1522
x-amazon-apigateway-request-validators.requestValidator 1523
Exemple X-Amazon-APIGateway-Request-Validators.RequestValidator 1523
Politique de sécurité de x-amazon-apigateway- 1523
Exemple de politique de sécurité x-amazon-apigateway- 1524
x-amazon-apigateway-tag-value 1524
Exemple x-amazon-apigateway-tag-value 1524
Sécurité 1526
Protection des données 1527
Chiffrement des données 1528
Confidentialité du trafic inter-réseaux 1530

Identity and Access Management 1531
Public ciblé 1531
Authentification par des identités 1532
Gestion de l’accès à l’aide de politiques 1533
Fonctionnement d'Amazon API Gateway avec IAM 1535
Exemples de stratégies basées sur l'identité 1540
Exemples de stratégies basées sur les ressources 1553
Dépannage 1553
Utilisation des rôles liés à un service 1555
Journalisation et surveillance 1560
Travailler avec CloudTrail 1561
Travailler avec AWS Config 1565
Validation de conformité 1568
Résilience 1569
Sécurité de l'infrastructure 1570
Analyse de la configuration et des vulnérabilités 1570
Bonnes pratiques 1570
Balisage 1573
Ressources API Gateway pouvant être balisées 1574
Héritage de balises dans l’API Amazon API Gateway V1 1575
Restrictions liées aux balises et conventions d’utilisation 1576
Attribute-based contrôle d'accès 1577
Limiter les actions en fonction des balises de ressource 1577
Autoriser des actions en fonction des balises de ressource 1578
Refuser les opérations de balisage 1579
Autoriser les opérations de balisage 1580
Références d’API 1582
Points de terminaison de service API Gateway 1582
IPv4 points de terminaison 1582
Points de terminaison à double pile (IPv4 et IPv6) 1583
Spécification des points de terminaison 1583
Utilisation des IPv6 adresses dans les politiques IAM 1584
Quotas et remarques importantes 1586
Quotas au niveau du compte API Gateway, par région 1586
Quotas API Gateway concernant la création, le déploiement et la gestion d’une API 1587
Quotas des API REST 1591

Bonnes pratiques pour réduire vos quotas 1598
Quotas API HTTP 1599
Bonnes pratiques pour réduire vos quotas 1603
Quotas du portail 1603
WebSocket Quotas d'API 1604
Remarques importantes 1607
Remarques importantes concernant Amazon API Gateway pour le protocole HTTP APIs .. 1608
Remarques importantes concernant Amazon API Gateway pour HTTP et WebSocket APIs 1608
Remarques importantes concernant REST APIs et WebSocket APIs 1608
Remarques importantes pour WebSocket APIs 1609
Remarques importantes pour REST APIs 1609
Historique du document 1616
Mises à jour antérieures 1631
..... mdcxliv

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 plus d’informations, consultez the section called “Qui utilise API Gateway ?”.

API Gateway crée des API RESTful qui :

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

Pour plus d’informations sur les API REST et HTTP API Gateway, consultez the section called “Choix entre les API HTTP et les API REST ”, API HTTP API Gateway, 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 “Vue d'ensemble des WebSocket API”.

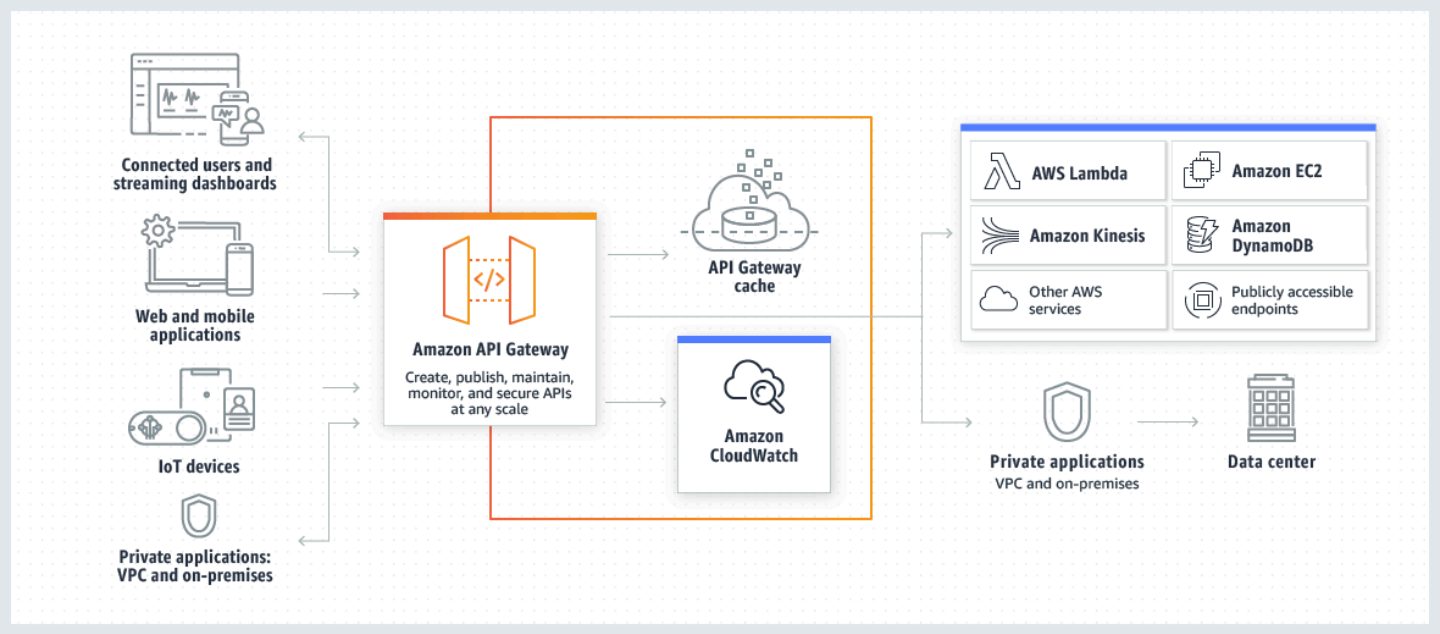
Rubriques

- Architecture API Gateway
- Fonctions d’API Gateway
- Cas d’utilisation d’API Gateway
- Accès à API Gateway

- Fait partie de AWS infrastructure sans serveur
- Comment démarrer avec Amazon API Gateway
- Concepts Amazon API Gateway
- Choix entre les API HTTP et les API REST
- Prise en main de 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 Gestion des identités et des accès AWS 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.
- Journalisation et surveillance de CloudTrail de l'utilisation des API et des modifications d'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 CloudFormation modèles pour permettre la création d'API. Pour plus d’informations, consultez 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, consultez Historique du document.

Cas d’utilisation d’API Gateway

La section des cas d’utilisation suivante présente un aperçu des différents types d’API d’API Gateway et de développeurs qui utilisent API Gateway. Pour plus d’informations sur les différences entre les API REST et les API HTTP, consultez the section called “Choix entre les API HTTP et les API REST ”.

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 politiques 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 avec 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
- Real-time des tableaux de bord tels que des indicateurs boursiers
- Real-time alertes et notifications

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 facile avec les HTTP/HTTPS terminaux

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) Extensions OpenAPI pour API Gateway.

Pour obtenir la liste des régions où API Gateway est disponible, ainsi que les points de terminaison du service de contrôle associés, consultez 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 Invocation d'une API REST dans API Gateway et Invoquer WebSocket APIs.

Accès à API Gateway

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

- Console de gestion AWS— Console de gestion AWS Fournit une interface Web pour créer et gérer des API. Après avoir suivi les étapes de Conditions préalables, 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 avec votre environnement de développement et permettent d'accéder facilement aux commandes API Gateway. Pour plus d'informations, consultez 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, consultez 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, consultez 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, consultez Configuration de AWS Tools for Windows PowerShell dans le Guide de l'utilisateur Outils AWS pour PowerShell .

Fait partie de 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 hautement disponible. 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

La section suivante décrit les concepts d'introduction à l'utilisation d'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, les AWS Lambda fonctions ou d'autres services HTTP principaux. AWS
- Création, déploiement et gestion d'une WebSocket API pour exposer des AWS Lambda fonctions ou d'autres AWS services.
- Invocation de méthodes d'API exposées via le 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 avec les points de terminaison HTTP du backend ou les 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 ”.

API WebSocket API Gateway

Ensemble d' WebSocket itinéraires et de clés de routage intégrés 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. 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

Instantané à un moment donné 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 qui possède un déploiement 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 :

- Edge-optimized Point de terminaison API
- 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 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

Une application dans laquelle les fournisseurs d'API peuvent partager leurs API et leur documentation d'API avec les consommateurs d'API. Les API sont regroupées en produits, qui sont un ensemble de points de terminaison d'API REST, de documentation d'API et de documentation produit supplémentaire.

Consultez la section Portails ??? API Gateway.

Edge-optimized Point de terminaison API

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 diversifiés.

Consultez l'entrée Points de terminaison d'API.

Demande d’intégration

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

Réponse d'intégration

Interface interne d'un itinéraire 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 simulée

Dans une intégration simulée, 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 simulée. Pour cela, vous configurez la demande d'intégration et la réponse d'intégration de la méthode pour associer une réponse à un code de statut 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.

Portal

Consultez le portail des développeurs.

Produit Portal

Un service ou une fonctionnalité que vous souhaitez partager. Le produit de votre portail est un ensemble de points de terminaison REST et de pages de produits. Les points de terminaison REST du produit sont les points d'accès au produit de votre portail. Ils comprennent le chemin et la méthode d'une API REST ainsi que l'étape à laquelle elle est déployée. Les pages produits sont de la documentation que vous fournissez pour expliquer comment les utilisateurs d'API peuvent utiliser les points de terminaison de votre produit. Vous pouvez partager des produits sur plusieurs AWS comptes pour les ajouter à des portails.

Consultez la section Création d'un produit.

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 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 sont dirigées directement vers l' Region-specific API API Gateway sans passer par aucune CloudFront distribution. Pour les demandes internes à la région, un point de terminaison régional permet d'éviter l'aller-retour inutile 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 requêtes pour accéder au backend via l'API.

Réponse de routage

Interface publique d'une WebSocket API qui définit les codes d'état, les en-têtes et les modèles de corps qu'un développeur d'applications doit attendre d'API Gateway.

Plan d’utilisation

Un plan d'utilisation permet à des clients d'API sélectionnés d'accéder à un ou plusieurs REST ou WebSocket API déployés. 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 d’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. Si nécessaire, des liens supplémentaires sont fournis pour naviguer entre les sections relatives aux API REST et aux API HTTP du Guide du développeur API Gateway.

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
Edge-optimized	Oui	Non
Régional	Oui	Oui
Privé	Oui	Non

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 "Protection" et the section called "Protection".

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

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	Oui	Oui
Politiques basées sur une ressource	Oui	Non
Amazon Cognito	Oui	Oui [1]
Autorisation personnalisée avec une AWS Lambda fonction	Oui	Oui
Jeton JWT (JSON Web Token) [2]	Non	Oui

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

[2] Vous pouvez utiliser un mécanisme 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 d'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".

Caractéristiques	API REST	API HTTP
Domaines personnalisés	Oui	Oui
Clés d'API	Oui	Non
Per-client limitation de débit	Oui	Non
Per-client limitation de l'utilisation	Oui	Non
Portail pour développeurs	Oui	Non

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".

Caractéristiques	API REST	API HTTP
Configuration CORS	Oui	Oui
Invocations de tests	Oui	Non
Mise en cache	Oui	Non
User-controlled déploiements	Oui	Oui
Déploiements automatiques	Non	Oui
Réponses de passerelle personnalisées	Oui	Non
Déploiement d'une version Canary	Oui	Non
Validation des demandes	Oui	Non
Transformation des paramètres de demande	Oui	Oui
Transformation du corps de la demande	Oui	Non

Contrôle

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 "Surveillance" et the section called "Surveillance".

Fonctionnalité	API REST	API HTTP
Statistiques CloudWatch Amazon	Oui	Oui
Accédez aux CloudWatch journaux	Oui	Oui
Accès aux journaux Amazon Data Firehose	Oui	Non
Journaux d'exécution	Oui	Non
AWS X-Ray traçage	Oui	Non

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	Oui	Oui
AWS services	Oui	Oui
AWS Lambda fonctions	Oui	Oui
Intégrations privées avec instances de Network Load Balancer	Oui	Oui
Intégrations privées avec instances d'Application Load Balancer	Oui	Oui

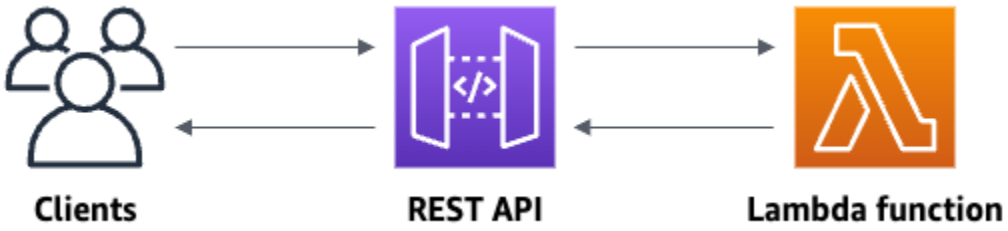
Fonctionnalité	API REST	API HTTP
Intégrations privées avec AWS Cloud Map	Non	Oui
Intégrations simulées	Oui	Non
Streaming des réponses	Oui	Non

Prise en main de 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. Le mode APIs Serverless vous permet de vous concentrer sur vos applications au lieu de perdre votre temps à approvisionner et à 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 avec 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 Gestion des identités et des accès AWS (IAM) ayant accès à la console. Pour de plus amples informations, veuillez consulter Configuration d'API Gateway avant son utilisation.

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 /lambda. https://console.aws.amazon.com
2. Sélectionnez Create function (Créer une fonction).
3. Sous Informations de base, pour Nom de la fonction, entrez **`my-function`**.
4. Pour toutes les autres options, utilisez le paramètre par défaut.
5. 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. Pour le type d'adresse IP, sélectionnez IPv4.
7. 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 avec 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 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 /lambda. https://console.aws.amazon.com
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 à l'aide de AWS CloudFormation or AWS Serverless Application Model (AWS SAM). Pour des exemples de CloudFormation modèles, consultez les exemples de modèles pour API Gateway dans le référentiel awsdocs GitHub .

Configuration d’API Gateway avant son utilisation

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

Inscrivez-vous pour un Compte AWS

Pour commencer AWS, vous avez besoin d'un Compte AWS. Il existe deux façons de s'inscrire à AWS. Si vous souhaitez contrôler totalement la configuration de votre compte et l'emplacement de stockage de votre contenu, ou si vous avez des charges de travail réglementées telles que HIPAA ou FedRAMP, utilisez Sign up for (advanced). AWS Si vous êtes un nouvel utilisateur et que vous souhaitez utiliser un identifiant que vous possédez déjà pour vous connecter AWS et créer un AWS environnement avec des paramètres par défaut préconfigurés dans Select Régions AWS, utilisez S'inscrire pour. AWS

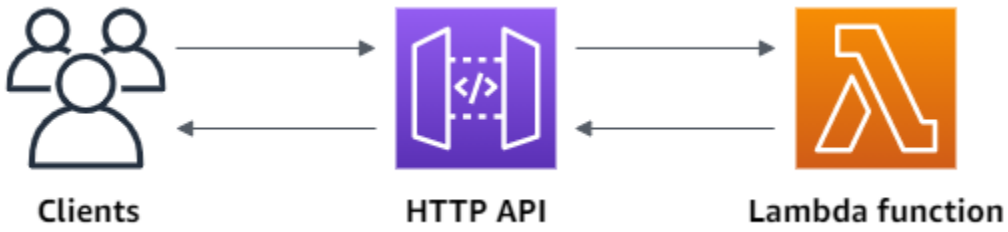
Mise en route avec API Gateway

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 AWSl'offre gratuite.

Vous commencez par créer une fonction Lambda à l'aide de la console AWS Lambda. 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 accéder à un didacticiel utilisant une API REST, consultez the section called "Prise en main de la console d'API REST".
> Pour plus d'informations sur les différences entre les API HTTP et les API REST, consultez the 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 compte AWS et d'un utilisateur Gestion des identités et des accès AWS avec accès à la console. Pour de plus amples informations, veuillez consulter Conditions préalables.

Rubriques

- Étape 1 : créer une fonction Lambda
- Étape 2 : créer une API HTTP
- Étape 3 : tester votre API

- (Facultatif) Étape 4 : nettoyer
- Étapes suivantes

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. Pour toutes les autres options, utilisez le paramètre par défaut.
5. 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 REST et les API WebSocket, 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 fonctions afin de pouvoir être proposées à un prix inférieur. Les API WebSocket maintiennent des connexions persistantes avec les clients pour une communication en duplex intégral, mais nous n'avons pas besoin de cette fonctionnalité 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.

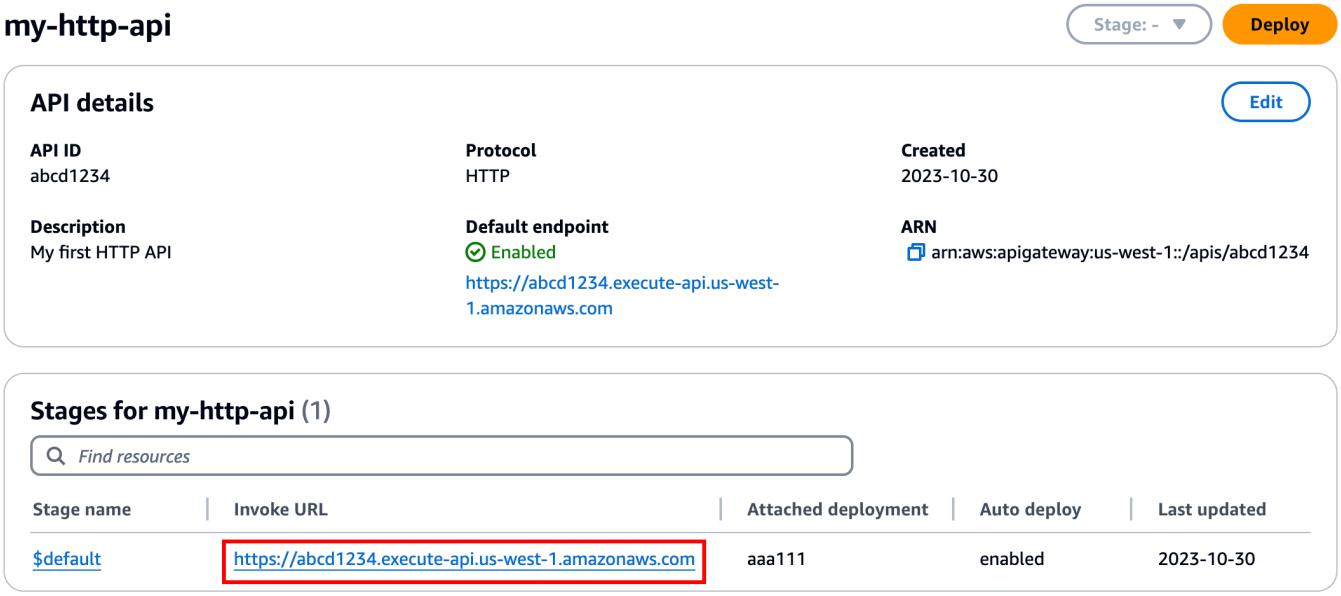
Vous avez maintenant créé une API HTTP avec une intégration Lambda, prête à recevoir des demandes de clients.

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 savoir, `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 console Amazon CloudWatch, ouvrez la page Groupes de journaux.
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 console Gestion des identités et des accès AWS, ouvrez la page Rôles.
2. Sélectionnez le rôle de la fonction, par exemple, `my-function-31exxmpl`.
3. Choisissez Supprimer le rôle.
4. Choisissez Oui, supprimer.

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

Étapes suivantes

Pour cet exemple, vous avez utilisé la Console de gestion AWS 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
  - Services AWS tels qu'Amazon Simple Queue Service, AWS Step Functions et Kinesis Data Streams
- 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, consultez le forum de discussion API Gateway. Lorsque vous entrez dans ce forum, AWS peut vous demander de vous connecter.

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

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. Pour commencer, nous vous recommandons de suivre le the section called “Didacticiel : création d’une API REST avec une intégration de proxy Lambda”.

Didacticiels sur les API REST

- Choisissez un didacticiel AWS Lambda d'intégration
- Didacticiel : création d’une API REST par l’importation d’un exemple
- Didacticiel sur la sélection d’une intégration HTTP
- Didacticiel : création d’une API REST avec une intégration privée
- Tutoriel : Création d'une API REST avec une AWS intégration
- Tutoriel : Création d'une API REST de calculatrice avec deux AWS intégrations de services et une intégration Lambda sans proxy
- Didacticiel : création d’une API REST en tant que proxy Amazon S3
- Didacticiel : création d’une API REST en tant que proxy Amazon Kinesis
- Tutoriel : Création d'une API REST à l'aide de AWS SDK ou AWS CLI
- Didacticiel : création d’une API REST privée

Didacticiels sur l'API HTTP

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

Didacticiels sur les API WebSocket

- Tutoriel : Création d'une application de WebSocket chat 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 les API REST Amazon API Gateway

Les didacticiels suivants proposent des exercices pratiques destinés à vous aider à découvrir API Gateway REST APIs.

Rubriques

- Choisissez un didacticiel AWS Lambda d'intégration
- Didacticiel : création d'une API REST par l'importation d'un exemple
- Didacticiel sur la sélection d'une intégration HTTP
- Didacticiel : création d'une API REST avec une intégration privée
- Tutoriel : Création d'une API REST avec une AWS intégration
- Tutoriel : Création d'une API REST de calculatrice avec deux AWS intégrations de services et une intégration Lambda sans proxy
- Didacticiel : création d'une API REST en tant que proxy Amazon S3
- Didacticiel : création d'une API REST en tant que proxy Amazon Kinesis
- Tutoriel : Création d'une API REST à l'aide de AWS SDK ou AWS CLI
- Didacticiel : 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 Lambda autre que de proxy.

Dans l'intégration de proxy Lambda, l'entrée de la fonction Lambda peut être exprimée sous la forme d'une combinaison d'en-têtes de demande, de variables de chemin d'accès, de paramètres de chaîne de requête, d'un corps et de données de configuration d'API. Vous n'avez qu'à 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 avoir à modifier les paramètres existants. Cette possibilité est due au fait que la fonction Lambda du backend analyse les données des demandes entrantes et répond au client.

Dans l'intégration Lambda autre que de proxy, 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 que le client a fournies comme paramètres de demande au corps de la demande d'intégration appropriée. 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'une intégration de 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

- Didacticiel : création d'une API REST avec une intégration de proxy Lambda
- Didacticiel : création d'une API REST avec une intégration Lambda sans proxy
- Didacticiel : création d'une API REST Gateway avec une intégration de proxy Lambda entre comptes

Didacticiel : création d'une API REST avec une 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, avec 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 allez effectuer les opérations suivantes :

- Créer une fonction Lambda « Hello, World! » pour être le backend de l'API.
- Créer et tester une API « Hello, World! » avec intégration de proxy Lambda.

Rubriques

- Création d'une fonction Lambda « Hello, World! »
- Création d'une API « Hello, World! »
- Déploiement et test de l'API

Création d'une fonction Lambda « Hello, World! »

Pour créer une fonction Lambda « Hello, World! » dans la console Lambda

1. Connectez-vous à la console Lambda à l'adresse /lambda. https://console.aws.amazon.com
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. Pour Architecture, conservez le paramètre par défaut.

   d. 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 politique AWS .

   e. Sous Nom du rôle, entrez **`GetStartedLambdaBasicExecutionRole`**.

   f. Laissez vide le champ Modèles de politique.

   g. Choisissez Créer une fonction.

7. Sous Code de fonction, dans l'éditeur de code en ligne, copy/paste le code suivant :

   Node.js

```
export const handler = async(event, context) => {
    console.log('Received event:', JSON.stringify(event, null, 2));
    var res ={
        "statusCode": 200,
        "headers": {
            "Content-Type": "*/*"
```

```
        }
    };
    var greeter = 'World';
    if (event.greeter && event.greeter!=="") {
        greeter =  event.greeter;
    } else if (event.body && event.body !== "") {
        var body = JSON.parse(event.body);
        if (body.greeter && body.greeter !== "") {
            greeter = body.greeter;
        }
    } else if (event.queryStringParameters &&
 event.queryStringParameters.greeter && event.queryStringParameters.greeter !==
 "") {
        greeter = event.queryStringParameters.greeter;
    } else if (event.multiValueHeaders && event.multiValueHeaders.greeter &&
 event.multiValueHeaders.greeter != "") {
        greeter = event.multiValueHeaders.greeter.join(" and ");
    } else if (event.headers && event.headers.greeter && event.headers.greeter !
= "") {
        greeter = event.headers.greeter;
    }
    res.body = "Hello, " + greeter + "!";
    return res
};
```

Python

```
import json


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

    greeter = 'World'

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

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

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

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

    res = {
        "statusCode": 200,
        "headers": {
            "Content-Type": "*/*"
        },
        "body": "Hello, " + greeter + "!"
    }

    return res
```

8. Choisissez Deploy (Déployer).

Création d’une API « Hello, World! »

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

Pour créer une API « Hello, World! »

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. Pour le type d'adresse IP, sélectionnez IPv4.
7. 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. 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 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 demandes `GET` en utilisant uniquement les paramètres de chaîne de requête, vous pouvez saisir l'URL de la ressource `helloworld` de l'API dans la barre d'adresse d'un navigateur.

Pour créer l'URL de la ressource `helloworld` de l'API, ajoutez la ressource `helloworld` et le paramètre de chaîne de requête `?greeter=John` à votre URL d'invocation. 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 didacticiel, 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!
```

Didacticiel : création d'une API REST avec une intégration Lambda sans proxy

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 Lambda autre que de proxy (également désignée sous le nom d'intégration personnalisée). Pour en savoir plus sur AWS Lambda et les fonctions Lambda, consultez 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, consultez Tutoriel : Création d'une API REST de calculatrice avec deux AWS intégrations 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 Lambda autre que de proxy
- Création d'une API avec l'intégration Lambda autre que de proxy
- Test de l'appel de la méthode d'API
- Déploiement de l'API
- Test de l'API dans une phase de déploiement
- Nettoyage

Création d'une fonction Lambda pour l'intégration Lambda autre que de proxy

> **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'];

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

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

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

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

Python

```
import json

days = {
    'Sunday',
    'Monday',
    'Tuesday',
    'Wednesday',
    'Thursday',
```

```
    'Friday',
    'Saturday'}
times = {'morning', 'afternoon', 'evening', 'night', 'day'}


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

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

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 politiques 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 politique et un rôle d'appel pour qu'API Gateway puisse appeler les fonctions Lambda. Pour plus d'informations sur la configuration des rôles d'appel et d'exécution Lambda pour une API Amazon API Gateway, consultez Contrôle de l'accès à une API REST 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. Pour Architecture, conservez le paramètre par défaut.
   d. 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 politique AWS .
   e. Pour Role name (Nom du rôle), indiquez le nom de votre rôle, par exemple **`GetStartedLambdaIntegrationRole`**.
   f. Pour Policy templates (Modèles de politique), choisissez Simple microservice permissions (Autorisations Microservice simples).
   g. 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 politiques 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.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "logs:CreateLogGroup",
            "Resource": "arn:aws:logs:us-east-1:111111111111:*"
        },
        {
```

```
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogStream",
                "logs:PutLogEvents"
            ],
            "Resource": [
                "arn:aws:logs:us-east-1:111111111111:log-group:/aws/lambda/
GetStartedLambdaIntegration:*"
            ]
        }
    ]
}
```

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 :

JSON

```
{
  "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 intégrée 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 Lambda autre que de proxy

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 Lambda autre que de proxy

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. Pour le type d'adresse IP, sélectionnez IPv4.
7. 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 Paramètres de requête de méthode.

   À présent, vous allez activer un valideur de requête 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 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 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 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 :

```
#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 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 vous APIs comptez, elle ne APIs fonctionnera 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 ne fonctionnera plus, et APIs toutes celles qui s'appuient sur cette fonction 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 Console de gestion AWS et ouvrez-la à l'adresse https://console.aws.amazon.com/lambda/.
2. Dans la liste des fonctions, choisissez GetStartedLambdaIntegration, Actions, puis Delete function (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, choisissez GetStartedLambdaIntegrationRole, Role Actions (Actions du rôle), puis Delete Role (Supprimer le rôle). Suivez les étapes de la console pour supprimer le rôle.

Didacticiel : création d'une API REST Gateway avec une 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 du backend Lambda entre plusieurs. APIs

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. Pour le type d'adresse IP, sélectionnez IPv4.
7. 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. Pour Architecture, conservez le paramètre par défaut.
    d. 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.
    e. 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 "Didacticiel : création d'une API REST avec une 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. 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` avec 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 Console de gestion AWS et ouvrez-la à l'adresse https://console.aws.amazon.com/lambda/.
2. Choisissez votre fonction Lambda.
3. Choisissez Permissions.

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

Didacticiel : 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.

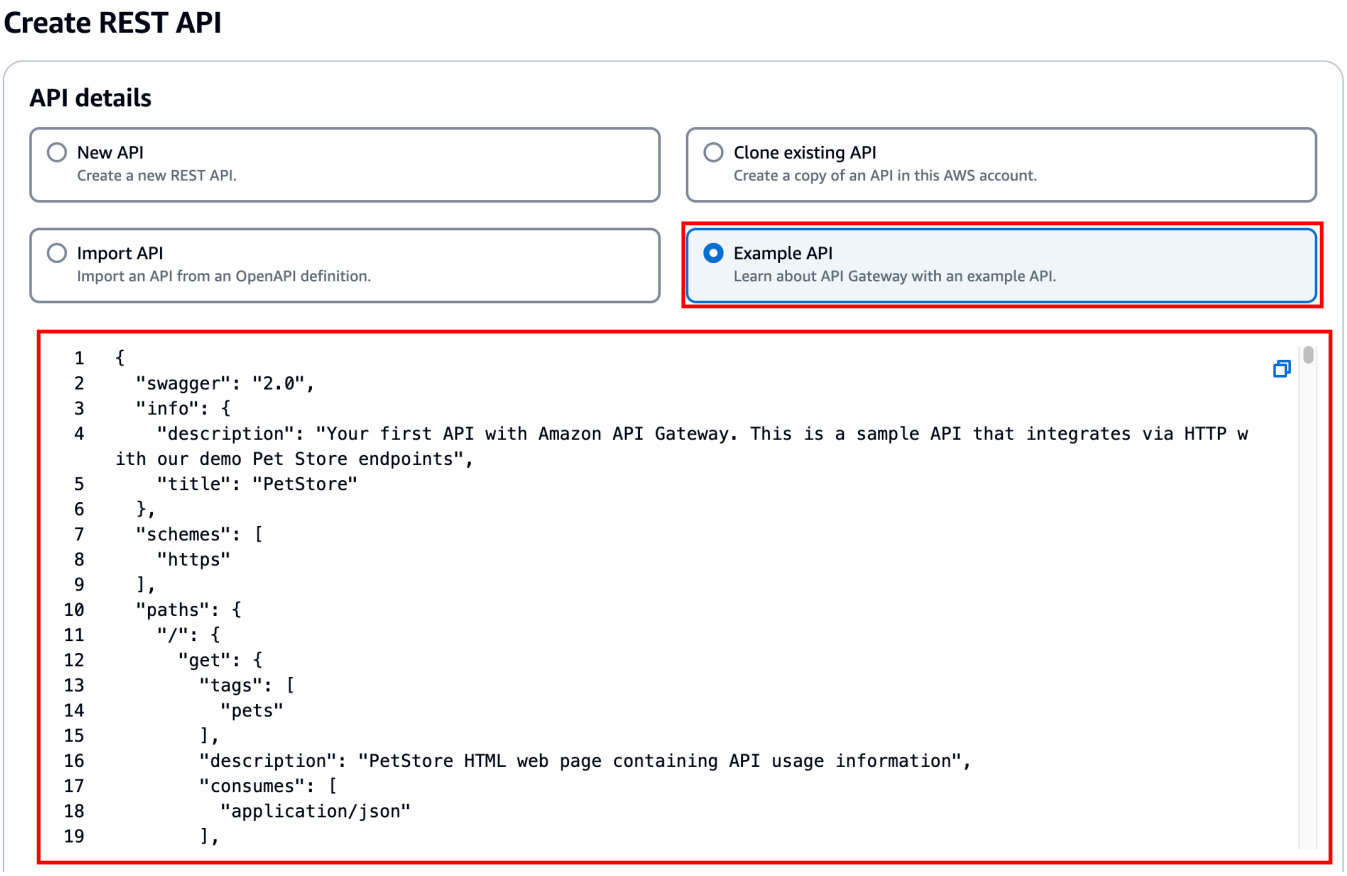
- `GET /` : pour l'accès en lecture de la ressource racine de l'API qui n'est pas intégrée avec un point de terminaison du serveur principal. API Gateway répond par un aperçu du PetStore site Web. Il s'agit d'un exemple du type d'intégration `MOCK`.
- `GET /pets` : pour l'accès en lecture de la ressource `/pets` de l'API intégrée avec 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 avec 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.

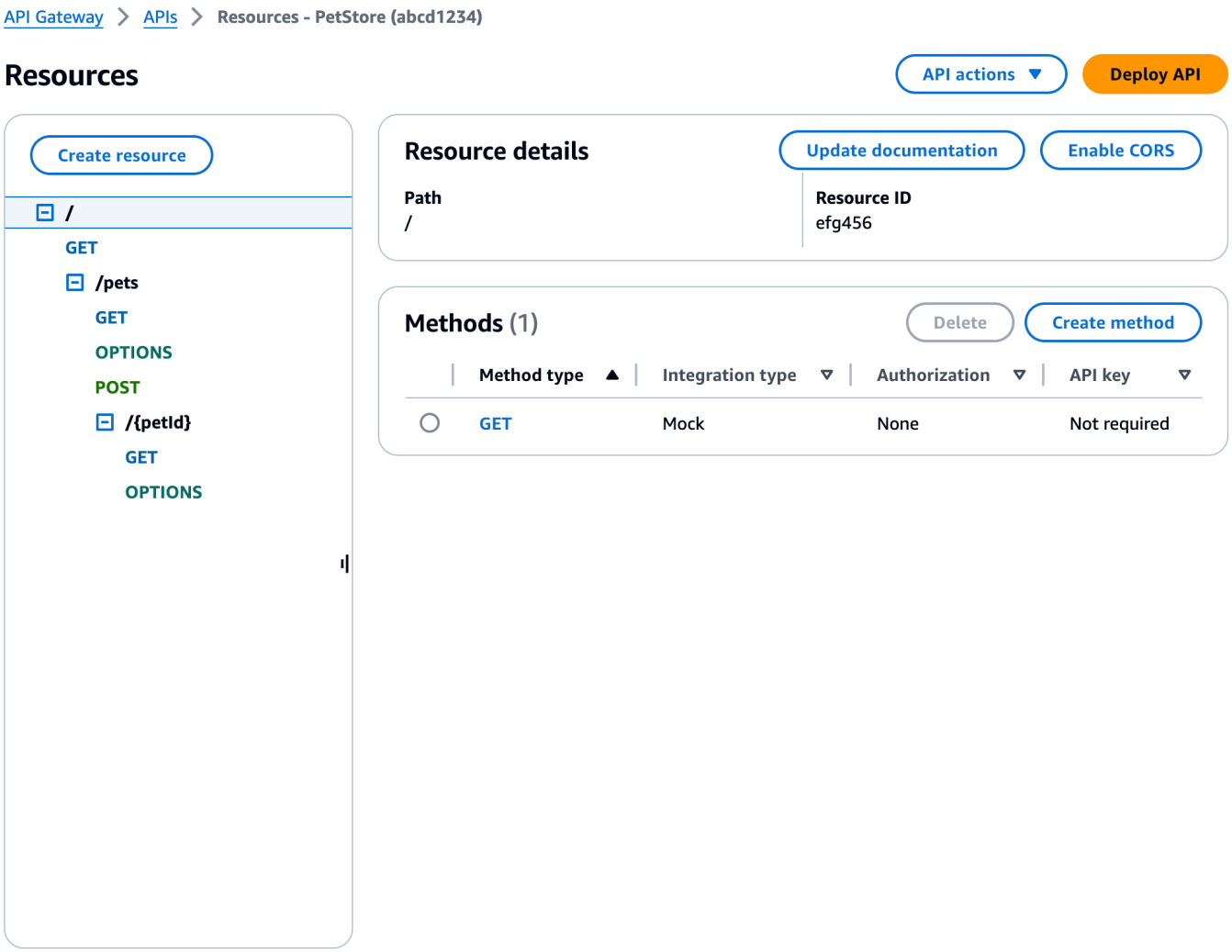
Pour importer, créer et tester l'exemple d'API

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.



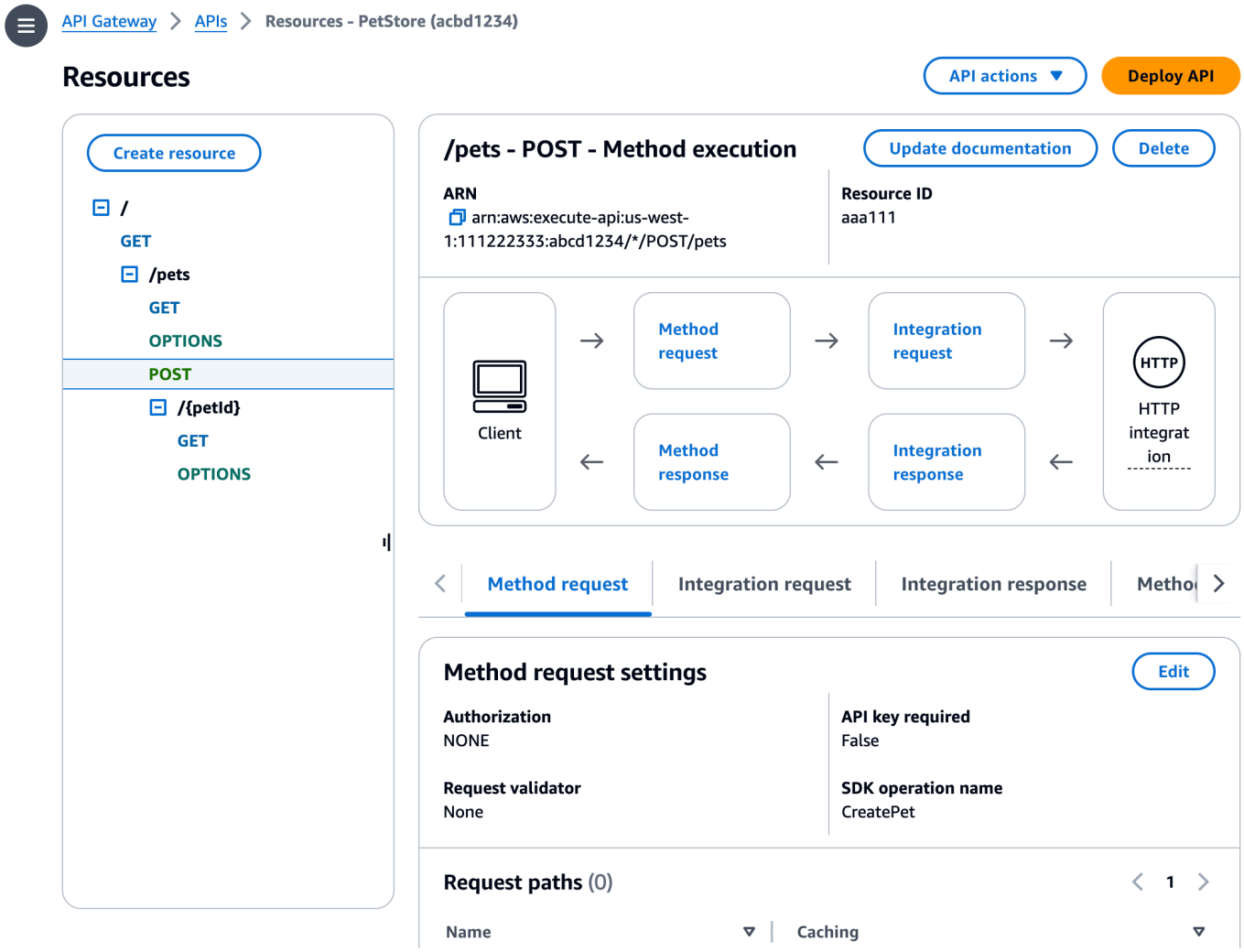
Vous pouvez faire défiler la définition OpenAPI vers le bas pour voir les détails de cet exemple d'API avant de choisir Créer une API.

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.

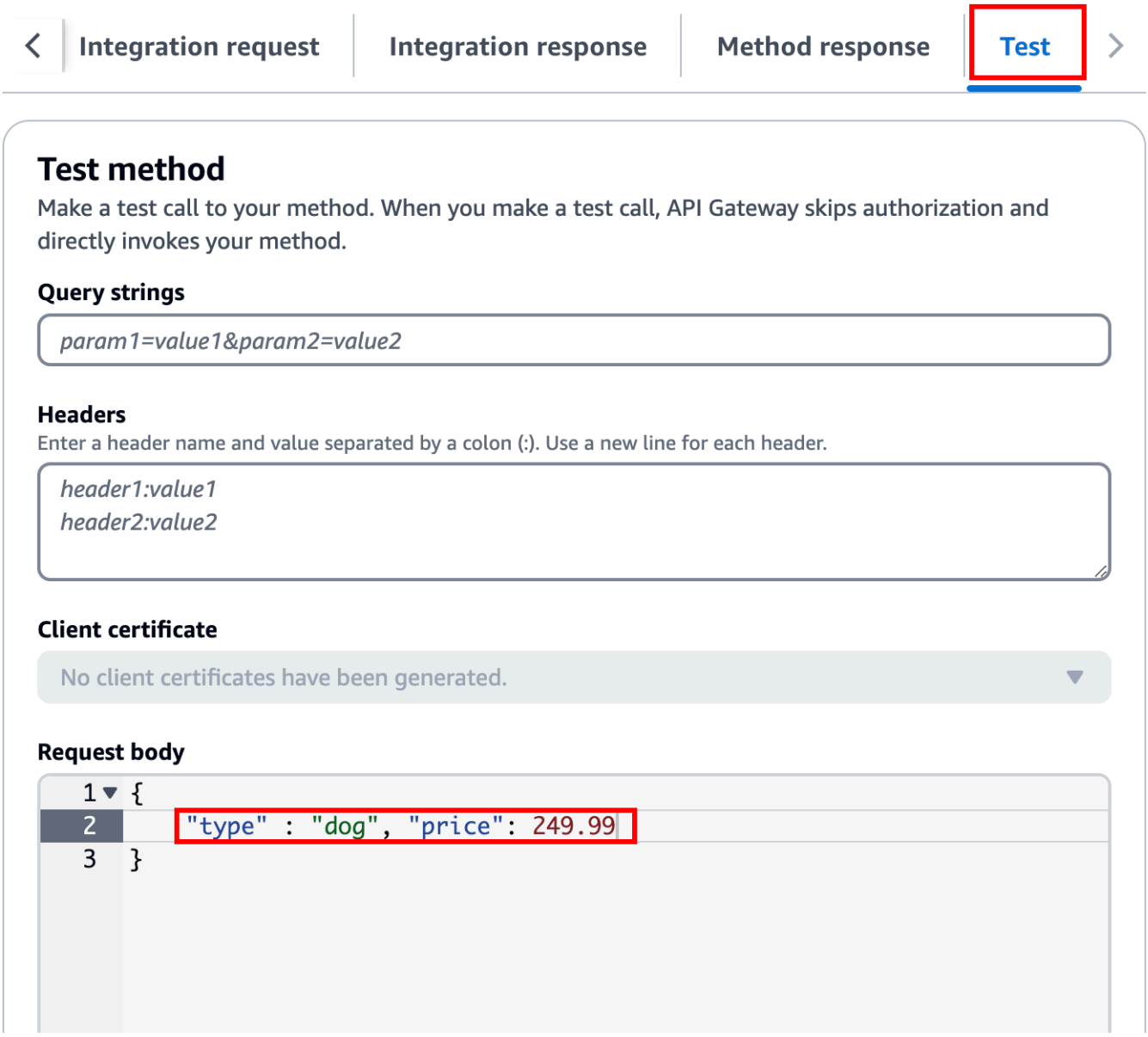
Un client utilise l'API pour accéder à une fonctionnalité backend via Demande de méthode. Si nécessaire, API Gateway convertit la demande du client au format acceptable pour le backend dans Demande d'intégration avant de la transférer vers ce backend. La demande transformée est connue en tant que demande 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 simulée 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 l'instant, nous nous concentrons sur le test de l'API pour fournir une expérience utilisateur de bout en bout.

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.

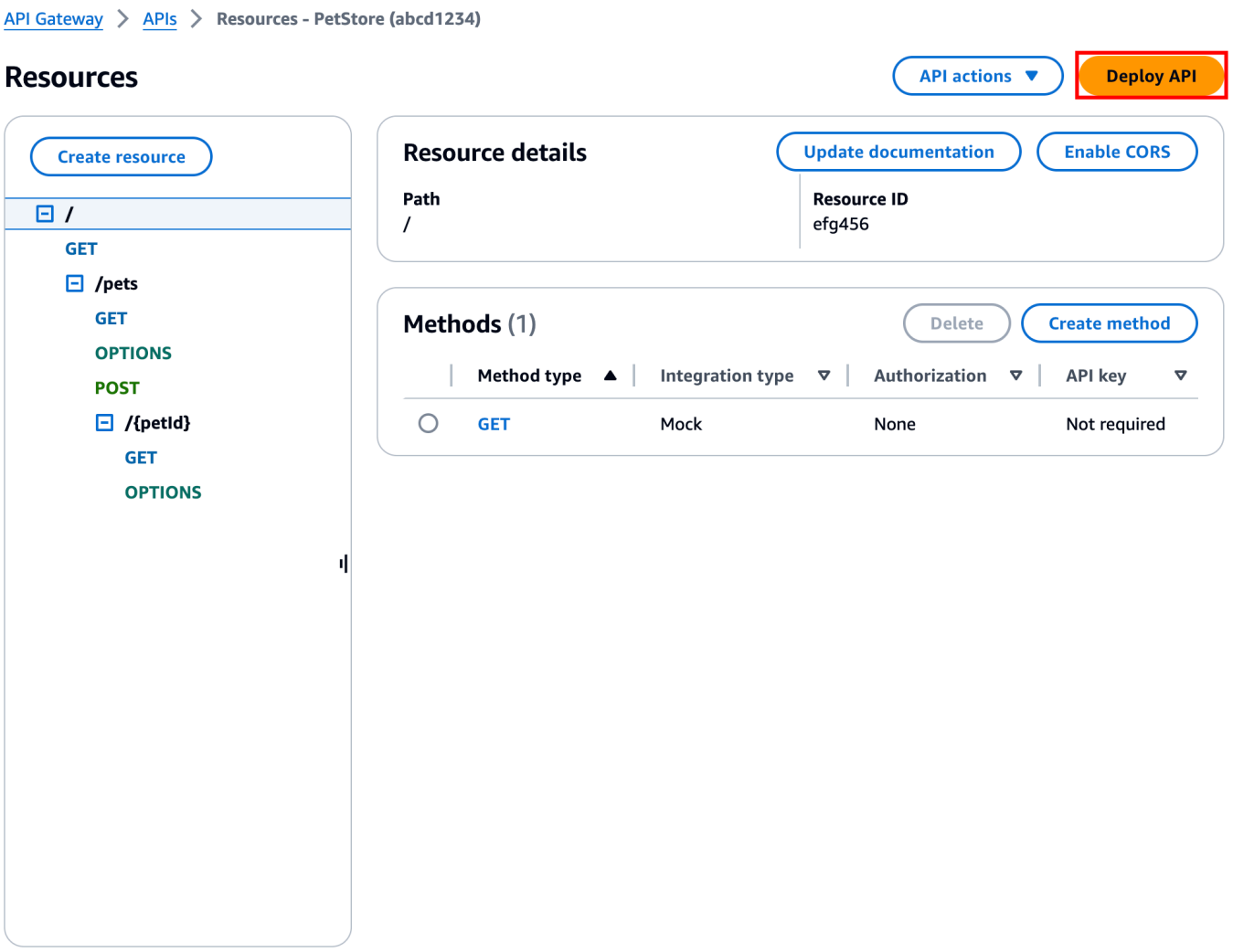
8. Les résultats s’affichent comme suit :



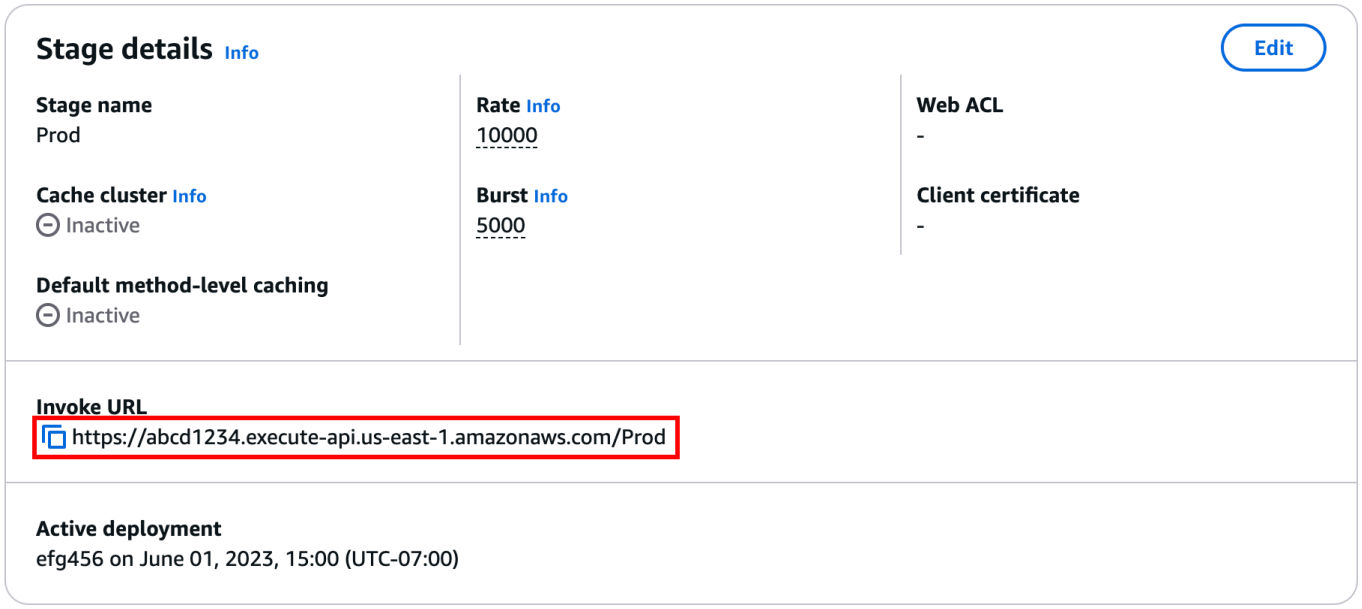
L'entrée Journal de la sortie indique les changements d'état entre la demande de la méthode et la demande d'intégration, et entre la réponse d'intégration et la réponse de la méthode. Cela peut être utile pour résoudre les éventuelles erreurs de mappage susceptibles d'entraîner l'échec de la demande. 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 autre client que la fonction de test de demande d'appel API Gateway, vous devez tout d'abord déployer l'API dans 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 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 signeriez la demande à l'aide du protocole Signature Version 4 (SigV4) ou Signature Version 4a (SigV4a). Pour voir un exemple de ce type de demande, consultez the section called "Didacticiel : création d'une API REST avec une intégration HTTP sans proxy".

Didacticiel sur la sélection d'une intégration HTTP

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

Dans une intégration de proxy HTTP, il vous suffit de définir la méthode HTTP et l'URI du point de terminaison HTTP selon les exigences du backend. Nous vous recommandons d'utiliser l'intégration de proxy HTTP, dans la mesure du possible, pour profiter de la configuration simplifiée de l'API.

Vous pouvez utiliser l'intégration HTTP personnalisée si vous avez besoin de transformer les données de la demande du client pour le backend, ou de transformer les données de la réponse du backend pour le client.

Rubriques

- Didacticiel : création d'une API REST avec une intégration de proxy HTTP
- Didacticiel : création d'une API REST avec une intégration HTTP sans proxy

Didacticiel : 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 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 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 Didacticiel : création d'une API REST avec une intégration HTTP sans 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

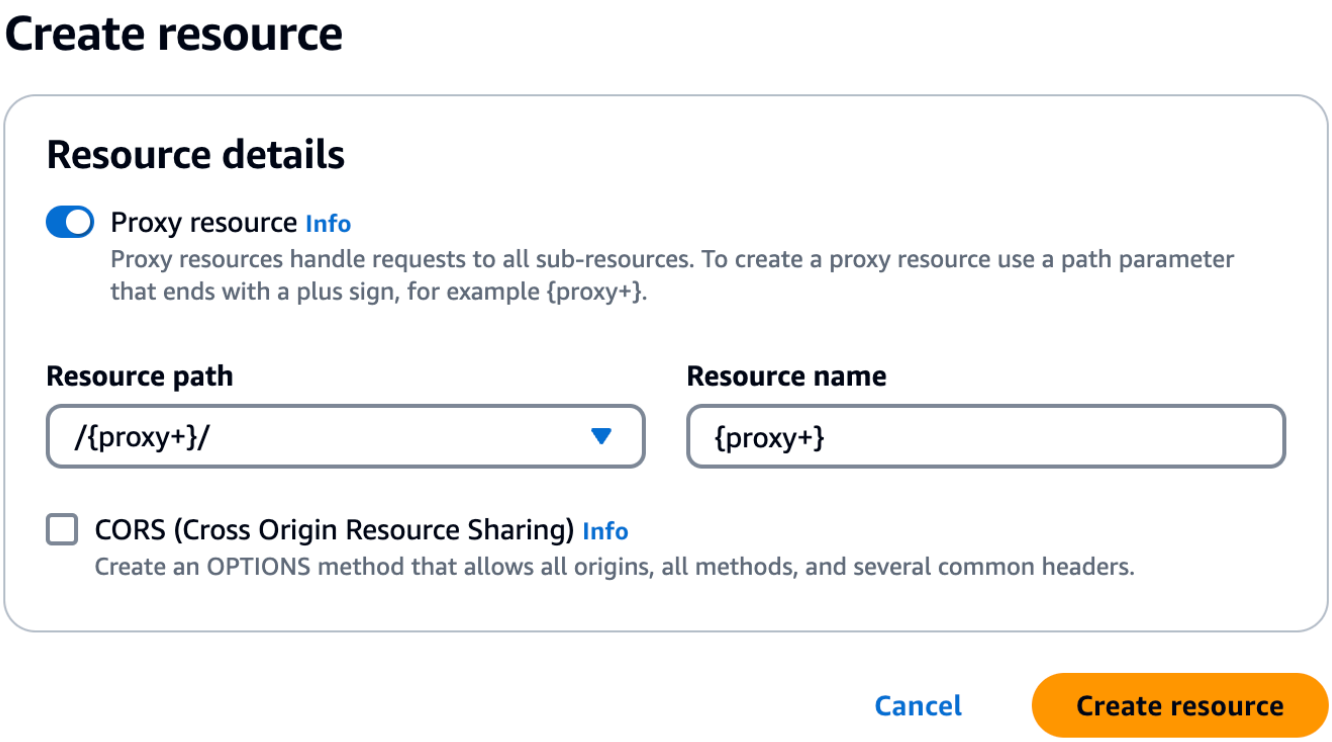
1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Si vous utilisez API Gateway pour la première fois, vous voyez une page qui vous présente les fonctions du service. Sous REST API (API REST), choisissez Build (Création). Lorsque la fenêtre contextuelle 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. Pour Type d'adresse IP, sélectionnez IPv4.
7. Sélectionnez Create API (Créer une API).

Dans cette étape, vous créez un chemin de ressource 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 un /{proxy+} ressource

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 avec 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.

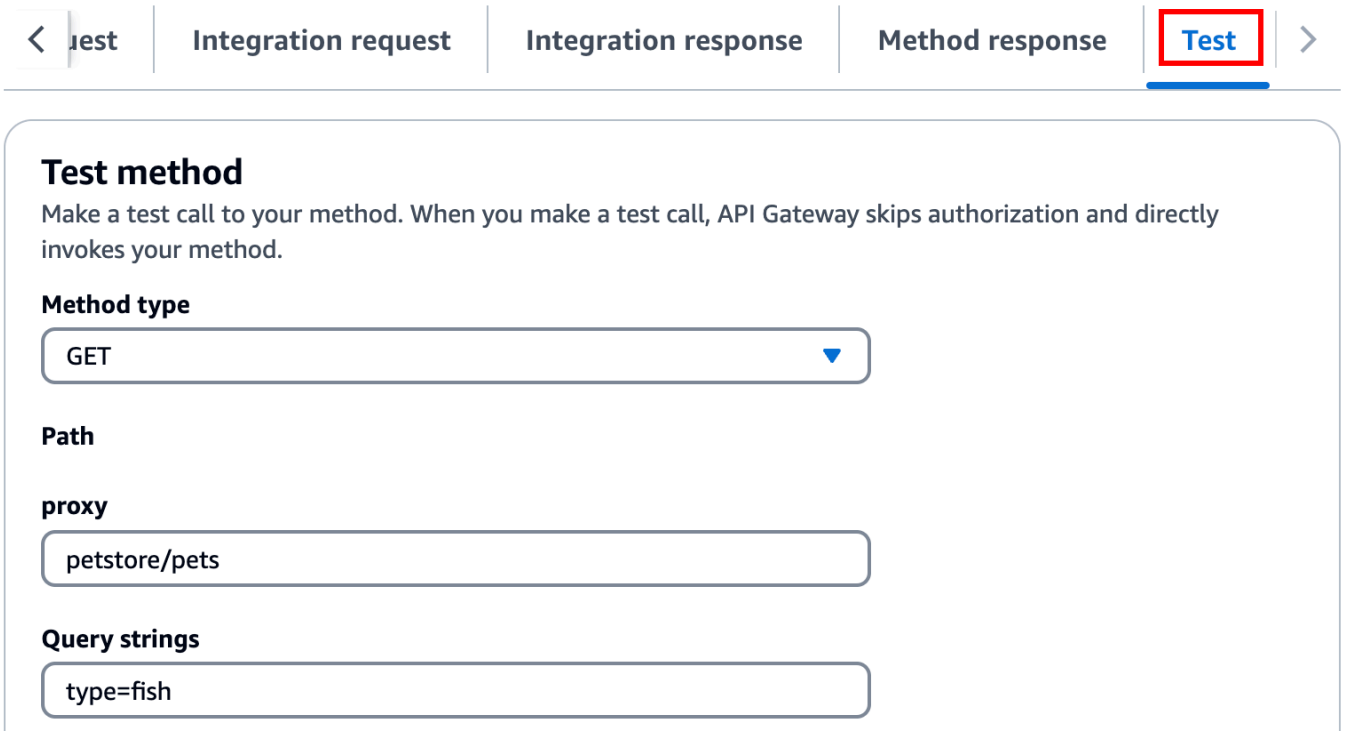
Par exemple, l' PetStore API utilisée ici n'expose pas la `/petstore` ressource. À ce titre, vous obtenez une réponse `404 Resource Not Found` contenant le message d'erreur `Cannot GET /petstore`.

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 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.

Didacticiel : création d'une API REST avec une intégration HTTP sans proxy

Dans ce didacticiel, 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 autre que de 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. Pour Type d'adresse IP, sélectionnez IPv4.
7. 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 pour qu'un client puisse récupérer une liste des articles Pets disponibles sur le PetStore site Web.

Pour créer une ressource /pets

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 **`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 avec le 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`** sur /animaux ressource

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 devez ajouter les paramètres de chaîne de requête à la demande de méthode et les mapper aux paramètres de chaîne de requête 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.

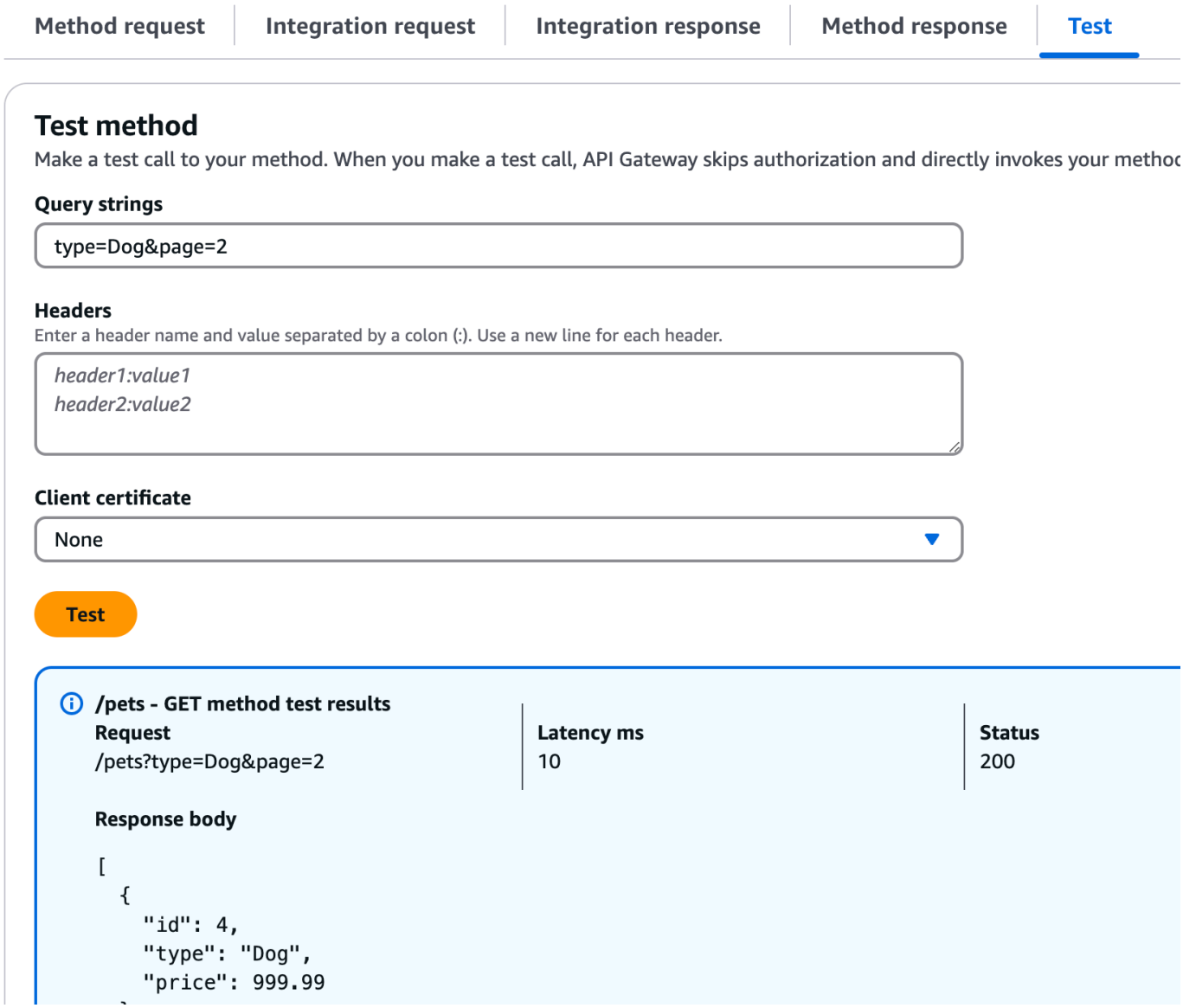
Pour mapper les paramètres d'entrée à la requête d'intégration

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 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 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 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, consultez le kit AWS SDK 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 de `{petId}` sur la demande de méthode de l'API pour spécifier un ID d'élément, mapper celui-ci au paramètre de chemin `{id}` dans l'URL de la demande d'intégration, et envoyer la demande 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}`que vous venez de créer au paramètre de chemin `{id}` dans l'URL du point de terminaison HTTP de la demande d'intégration. L'URL du point de terminaison HTTP était **`http://petstore-demo-endpoint.execute-api.com/petstore/pets/{id}`**.

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 demande d'intégration nommée petId, mais il n'est pas valide pour l'URL du point de terminaison HTTP que vous avez défini comme intégration backend. Le point de terminaison HTTP utilise `{id}` comme paramètre de chemin. Pour Nom, supprimez delete petId et saisissez **`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 : déployer 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 : tester 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 Tutoriel : Modifier la demande et la réponse d'intégration pour les intégrations vers AWS services.
- 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.

Didacticiel : création d'une API REST avec une intégration privée

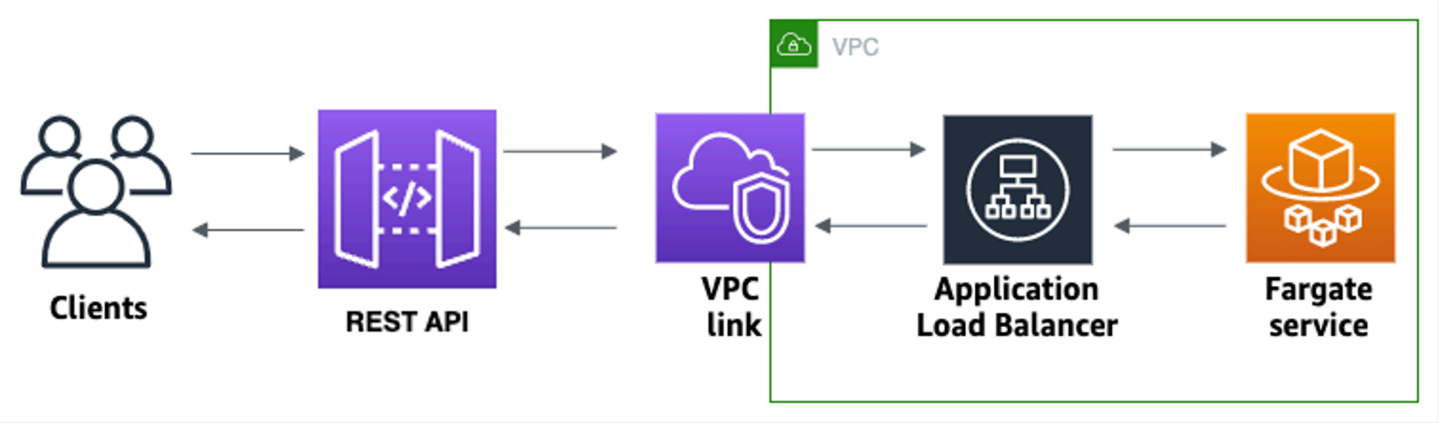
Dans ce didacticiel, vous allez créer une API REST qui se connecte à un service Amazon ECS qui s'exécute dans un Amazon VPC. Les clients en dehors de votre VPC Amazon peuvent utiliser l'API pour accéder à votre service Amazon ECS.

Ce didacticiel vous prendra environ 1 heure. Tout d'abord, vous utilisez un CloudFormation modèle pour créer un service Amazon VPC et Amazon ECS. Vous utilisez ensuite la console API Gateway

pour créer un lien VPC V2. Le lien VPC permet à API Gateway d'accéder à l'service Amazon ECS qui s'exécute dans votre VPC Amazon. Ensuite, vous créez une API REST qui utilise le lien VPC V2 pour se connecter à votre service Amazon ECS. Enfin, vous testez votre API.

Lorsque vous appelez votre API REST, API Gateway achemine la demande vers votre service Amazon ECS via votre lien VPC V2, puis renvoie la réponse du service.

> **Note**
>
> Ce didacticiel était auparavant pris en charge pour les API HTTP. Il est désormais pris en charge pour les API REST utilisant le lien VPC V2.



Pour suivre ce didacticiel, vous avez besoin d'un AWS compte et d'un Gestion des identités et des accès AWS utilisateur disposant d'un accès à la console. Pour de plus amples informations, veuillez consulter Conditions préalables.

Rubriques

- Étape 1 : créer un service Amazon ECS
- Étape 2 : créer un lien VPC
- Étape 3 : Création d'une API REST
- Étape 4 : Testez votre API
- Étape 5 : déploiement de votre API
- Étape 6 : appelez votre API
- Étape 7 : nettoyer

Étape 1 : créer un service Amazon ECS

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 didacticiel, vous exécuterez votre cluster sur une infrastructure sans serveur gérée par Amazon ECS.

Téléchargez et décompressez ce CloudFormation modèle, qui crée toutes les dépendances du service, y compris un Amazon VPC. Vous utilisez le modèle pour créer un service Amazon ECS qui utilise un Application Load Balancer.

Pour créer un CloudFormation pile

1. Ouvrez la 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 **`rest-api-private-integrations-tutorial`**, puis choisissez Suivant.
7. Pour Configurer les options de pile, choisissez Suivant.
8. Pour les fonctionnalités, reconnaissez que CloudFormation vous pouvez créer des ressources IAM dans votre compte.
9. Choisissez Suivant, puis Soumettre.

CloudFormation fournit le service ECS, ce qui peut prendre quelques minutes. Lorsque le statut de votre CloudFormation pile est CREATE_COMPLETE, vous êtes prêt à passer à l'étape suivante.

Étape 2 : créer 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 service Amazon ECS via votre API REST.

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 panneau de navigation principal, choisissez Liens VPC, puis Créer.

   Vous devrez peut-être sélectionner l'icône Menu pour ouvrir le volet de navigation principal.
3. Pour Choisir une version de liaison VPC, sélectionnez VPC link V2.
4. Pour Nom, saisissez **`private-integrations-tutorial`**.
5. Pour VPC, choisissez le VPC que vous avez créé à l'étape 1. Son nom doit commencer par RestApiStack.
6. Pour Sous-réseaux, sélectionnez les deux sous-réseaux privés de votre VPC. Leurs noms finissent par `PrivateSubnet`.
7. Pour Groupes de sécurité, sélectionnez l'ID de groupe commençant par `private-integrations-tutorial` et contenant la description `RestApiStack/RestApiTutorialService/Service/SecurityGroup`.
8. Choisissez Créer.

Après avoir créé votre lien VPC V2, API Gateway fournit des interfaces réseau élastiques pour accéder à votre VPC. Ce processus peut prendre quelques minutes. En attendant, vous pouvez créer votre API.

Étape 3 : Création d'une API REST

L'API REST fournit un point de terminaison HTTP pour votre service Amazon ECS.

Pour créer une API REST

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. Pour Nom, saisissez **`private-integration-api`**.
4. Pour Type d'adresse IP, sélectionnez IPv4.
5. Sélectionnez Create API (Créer une API).

   Après avoir créé votre API, vous créez une méthode.
6. 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 le lien VPC, choisissez le lien VPC V2 que vous avez créé à l'étape précédente.

f. Pour la cible d'intégration, entrez l'équilibreur de charge que vous avez créé avec le CloudFormation modèle à l'étape 1. Son nom doit commencer par rest-.

g. Pour URL du point de terminaison, saisissez `http://private-integrations-tutorial.com`.

   L'URL est utilisée pour définir l'`Host`en-tête de la demande d'intégration. Dans ce cas, l'en-tête de l'hôte est**`private-integrations-tutorial`**.

h. Choisissez Créer une méthode.

   Grâce à l'intégration du proxy, l'API est prête à être testée.

Étape 4 : Testez votre API

Ensuite, vous testez l'appel de la méthode API.

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. Choisissez l’onglet Test. Vous devrez peut-être choisir la flèche droite pour afficher l’onglet.
4. Choisissez le test

   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.

Étape 5 : déploiement de votre API

Ensuite, vous déployez 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 **`Prod`**.
4. (Facultatif) Sous Description, entrez une description.
5. Choisissez Déployer.

Étape 6 : appelez votre API

Une fois votre API déployée, vous pouvez l'appeler.

Pour appeler votre API

1. 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.
2. 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 service Amazon ECS qui s'exécute dans un Amazon VPC, et vous avez utilisé une API REST API Gateway avec un lien VPC V2 pour accéder au service Amazon ECS.

Étape 7 : nettoyer

Pour éviter des coûts inutiles, supprimez les ressources que vous avez créées dans le cadre de ce didacticiel. Les étapes suivantes suppriment votre lien VPC V2, votre CloudFormation stack et votre API REST.

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, 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 un CloudFormation pile

1. Ouvrez la CloudFormation console à l'adresse https://console.aws.amazon.com/cloudformation.
2. Sélectionnez votre CloudFormation pile.
3. Choisissez Supprimer, puis confirmez votre choix.

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

Les deux rubriques Didacticiel : création d'une API REST avec une intégration de proxy Lambda et Didacticiel : création d'une API REST avec une intégration Lambda sans proxy 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 de support sont dédiés APIs à 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 Didacticiel : création d'une API REST avec une intégration Lambda sans proxy. Si vous n'avez pas encore effectué cette procédure, nous vous conseillons de le faire en premier.

Rubriques

- Conditions préalables
- Étape 1 : créer le rôle d'exécution du proxy de AWS service
- Étape 2 : créer la ressource
- Étape 3 : créer la méthode GET
- Étape 4 : spécifier les paramètres et tester la méthode
- Étape 5 : déployer l'API
- Étape 6 : tester l'API
- Étape 7 : nettoyer

Conditions préalables

Avant de commencer cette procédure, exécutez les étapes suivantes :

1. Suivez les étapes de Configuration d'API Gateway avant son utilisation.
2. Créez une nouvelle API nommée `MyDemoAPI`. Pour de plus amples informations, veuillez consulter Didacticiel : création d'une API REST avec une intégration HTTP sans proxy.
3. Déployez l'API au moins une fois dans 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. Au cours de cette étape, vous allez créer un rôle IAM.

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

1. Connectez-vous à la console IAM Console de gestion AWS et ouvrez-la à https://console.aws.amazon.com/iam/l'adresse.
2. Sélectionnez Roles.
3. Choisissez 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 politique gérée pour plus de simplicité. Il est recommandé de créer votre propre politique 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 : spécifier les paramètres et tester 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 : déployer 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 : tester 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 tous APIs ceux 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 APIGatewayAWSProxyExecRole, puis Role Actions (Actions du rôle), et Delete Role (Supprimer le rôle). À l'invite, sélectionnez Yes, Delete.
4. Dans la zone Details, sélectionnez Policies.
5. Sélectionnez APIGatewayAWSProxyExecPolicy, puis choisissez Policy Actions (Actions de stratégie) et Delete (Supprimer). À 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 serviceDidacticiel : création d'une API REST en tant que proxy Amazon S3, consultezTutoriel : Création d'une API REST de calculatrice avec deux AWS intégrations de services et une intégration Lambda sans proxy, ouDidacticiel : création d'une API REST en tant que proxy Amazon Kinesis.

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

Le the section called "Didacticiel : création d'une API REST avec une intégration Lambda sans proxy" utilise exclusivement l'intégration `Lambda Function`. 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 invoquer 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 didacticiel, vous allez créer une fonction `Calc` Lambda qui implémente des opérations arithmétiques de base, en acceptant et en renvoyant des JSON-formatted entrées et des sorties. Ensuite, vous allez créer une API REST et l'intégrer avec 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 didacticiel, 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 Calc Lambda
- Testez la fonction Calc Lambda
- Création d'une API Calc
- Intégration 1 : créer une méthode GET avec des paramètres de requête pour appeler la fonction Lambda
- Intégration 2 : créer une méthode POST avec une charge utile JSON pour appeler la fonction Lambda
- Intégration 3 : créer une méthode GET avec des paramètres de chemin pour appeler la fonction Lambda
- Définitions OpenAPI de l'exemple d'API intégré avec 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 :

JSON

```
{
  "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 politique IAM suivante :

JSON

```
{
    "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 :

      JSON

```
{
    "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. Sélectionnez Next:Tags.

c. Sélectionnez Next:Review.

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

e. Choisissez 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 politique existante comme suit :

JSON

```
{
  "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 **`Calc Lambda`**

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 Runtime, choisissez soit le dernier runtime compatible, Node.jssoit le dernier runtime Python.
5. Pour toutes les autres options, utilisez le paramètre par défaut.
6. Choisissez Créer une fonction.
7. 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
```

8. Sous Execution role (Rôle d'exécution), choisissez Choose an existing role (Choisir un rôle existant).
9. Entrez l'ARN du rôle **`lambda_invoke_function_assume_apigw_role`** que vous avez créé précédemment.
10. 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.

Testez la fonction **`Calc Lambda`**

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

1. Choisissez l'onglet Test.
2. Entrez **`calc2plus5`** comme nom de l'événement de test.
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. Pour Type d'adresse IP, sélectionnez IPv4.
7. Sélectionnez Create API (Créer une API).

Intégration 1 : créer 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 un /calc 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 **`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 Paramètres de requête 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.

Maintenant, vous allez configurer les paramètres de chaîne de requête pour la méthode GET sur la ressource /calc de sorte qu'elle reçoive l'entrée au nom de la fonction Lambda du backend.

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

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

b. Pour 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 Demande 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')"
```

```
}
```

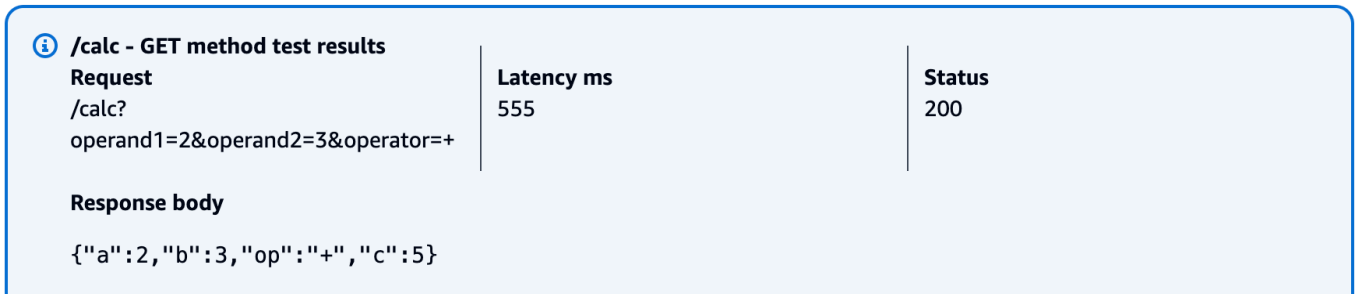
7. Choisissez Enregistrer.

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 :



Intégration 2 : créer 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 Créer un modèle.
3. Pour 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 Créer un modèle.

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 Créer un modèle.
2. Pour 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 Créer un modèle.

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 Créer un modèle.
2. Pour 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* nom est répertorié 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 Créer un modèle.

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

Pour activer la validation de la demande 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 requête 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 :

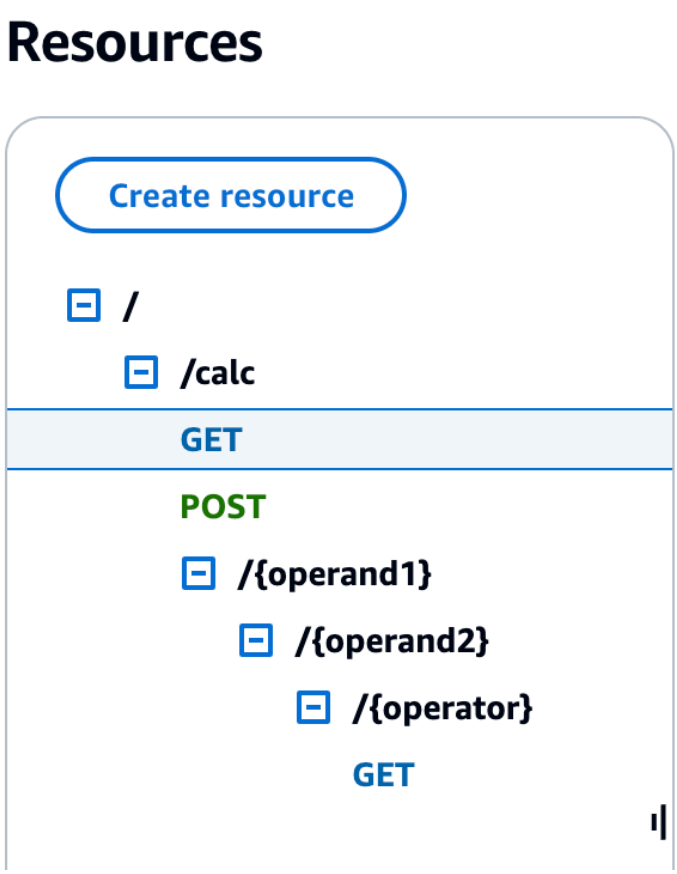
```
{
  "a": 1,
  "b": 2,
  "op": "+",
  "c": 3
}
```

Intégration 3 : créer 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 un /{opérande1}/{opérande2}/{opérateur} ressource

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. Comme les chemins d'URL doivent l'être URL-encoded, l'opérateur de division doit être spécifié comme étant `%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 :

```
{
```

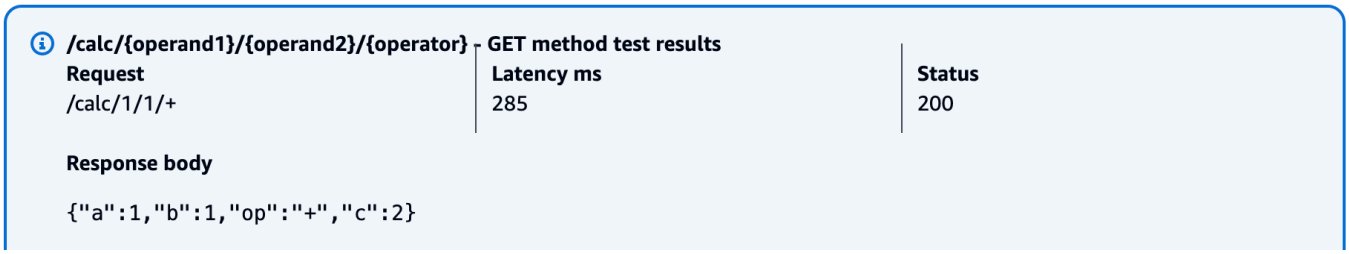
```
    "a": "$input.params('operand1')",
    "b": "$input.params('operand2')",
    "op":
  #if($input.params('operator')=='%2F')"/"#{else}"$input.params('operator')"#end
}
```

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. Toutefois, lorsque vous générez un SDK pour l'un des langages les plus courants, tel que Java ou C Objective-C, les utilisateurs de votre SDK recevront un objet vide en conséquence. 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 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 avec 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
    }
  }
}
```

Didacticiel : création d'une API REST en tant que proxy Amazon S3

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.
- Exposez GET sur une Folder/Item ressource pour afficher ou télécharger un objet depuis 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 plus d'informations sur l'importation d'une API à l'aide de la définition OpenAPI, consultez Développement d'API REST à l'aide d'OpenAPI dans API Gateway.

> **Note**
>
> Pour intégrer votre API Amazon API Gateway avec Amazon S3, vous devez choisir une région où les services API Gateway et Amazon S3 sont disponibles. Pour connaître la

disponibilité dans les régions, consultez 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. Au cours de cette étape, vous allez créer un rôle IAM.

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

1. Connectez-vous au Console de gestion AWS et ouvrez la console IAM à https://console.aws.amazon.com/iam/ l'adresse.
2. Sélectionnez Roles.
3. Choisissez Créer un rôle.
4. Choisissez le AWS service sous Sélectionner le type d'entité de confiance, puis sélectionnez API Gateway et sélectionnez Autoriser API Gateway à transmettre les journaux vers CloudWatch les journaux.
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 politique gérée pour plus de simplicité. Il est recommandé de créer votre propre politique 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 allez utiliser la ressource racine de l'API (`/`) comme conteneur des compartiments Amazon S3 d'un appelant authentifié. Vous allez également créer les ressources `Folder` et `Item` pour représenter respectivement un compartiment Amazon S3 spécifique et un objet Amazon S3 spécifique. 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.

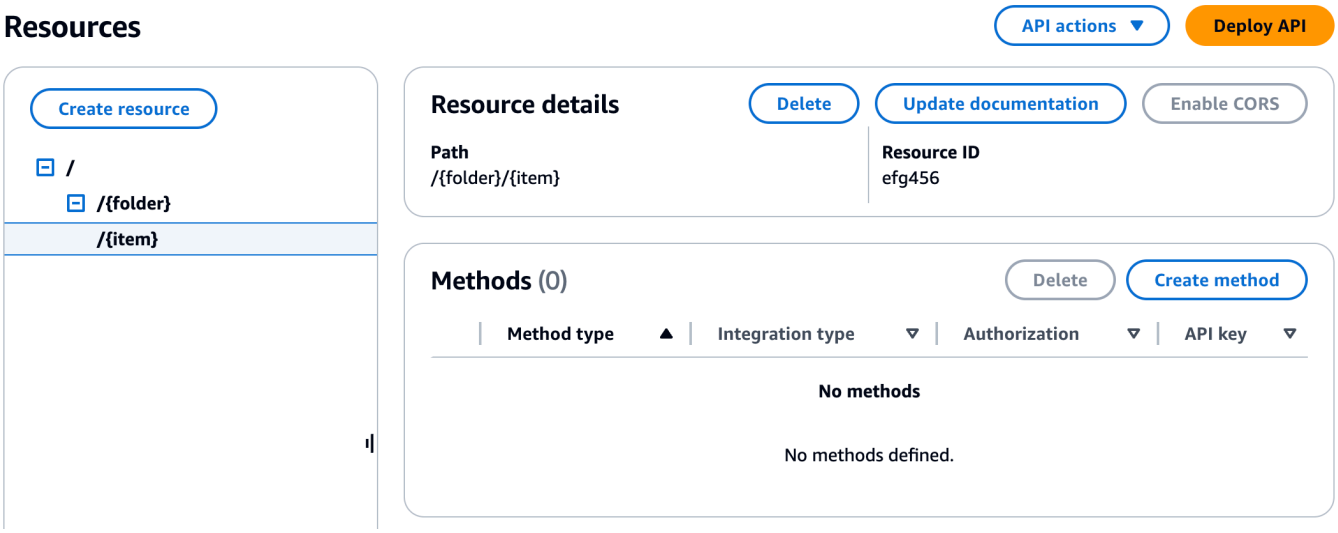
> **Note**
>
> Lorsque vous accédez à des objets dont la clé d'objet inclut `/` ou n'importe quel autre caractère spécial, ce caractère doit être encodé en URL. Par exemple, `test/test.txt` doit être codé en `test%2Ftest.txt`.

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

1. De la même manière 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. 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 avec Amazon S3, comme suit.

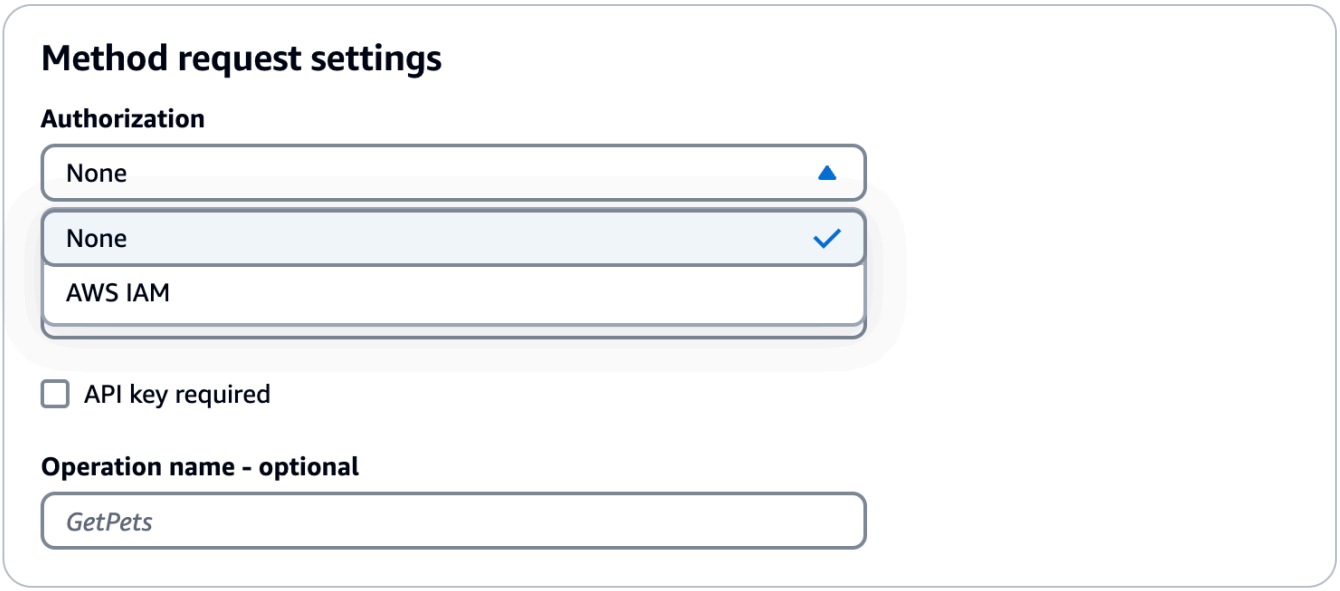
Pour créer et initialiser la méthode **`GET/`**de l'API

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 Paramètres de requête de méthode.

    Vous allez utiliser les paramètres de requête 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/` front-end avec la méthode `GET https://your-s3-host/` back-end.

Pour que votre API renvoie correctement à l'appelant les réponses positives et les exceptions, vous allez déclarer les réponses 200, 400 et 500 dans Réponse de méthode. Vous allez utiliser le mappage par défaut pour les réponses 200 afin que les réponses back-end du code de statut non déclarées ici soient renvoyées à l'appelant en tant que réponses 200.

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. Cliquez sur 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. Cliquez sur Enregistrer.
7. Répétez les étapes suivantes pour créer la réponse 500.

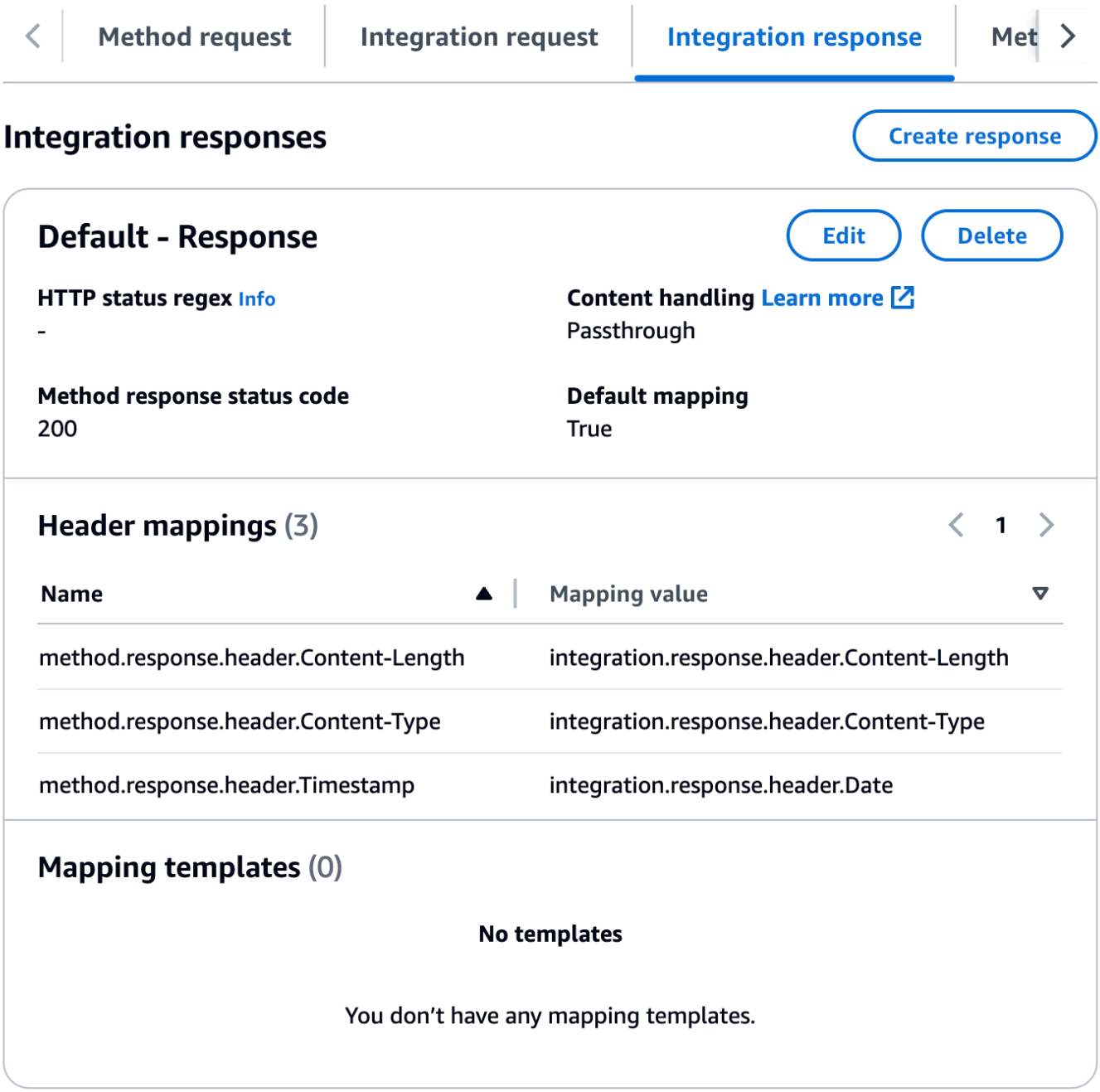
   Vous ne définissez aucun en-tête pour cette réponse.

Étant donné que la réponse d'intégration réussie d'Amazon S3 renvoie la liste des 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' Content-Type en-tête du backend à l'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 Date et Content-Length.

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'Content-Lengthen-tête, entrez **integration.response.header.Content-Length** la valeur de mappage.
3. Pour l'Content-Typeen-tête, entrez **integration.response.header.Content-Type** la valeur de mappage.
4. Pour l'en-tête Timestamp, saisissez **integration.response.header.Date** pour la valeur de mappage.

5. Cliquez sur Enregistrer. Le résultat devrait ressembler à ce qui suit :



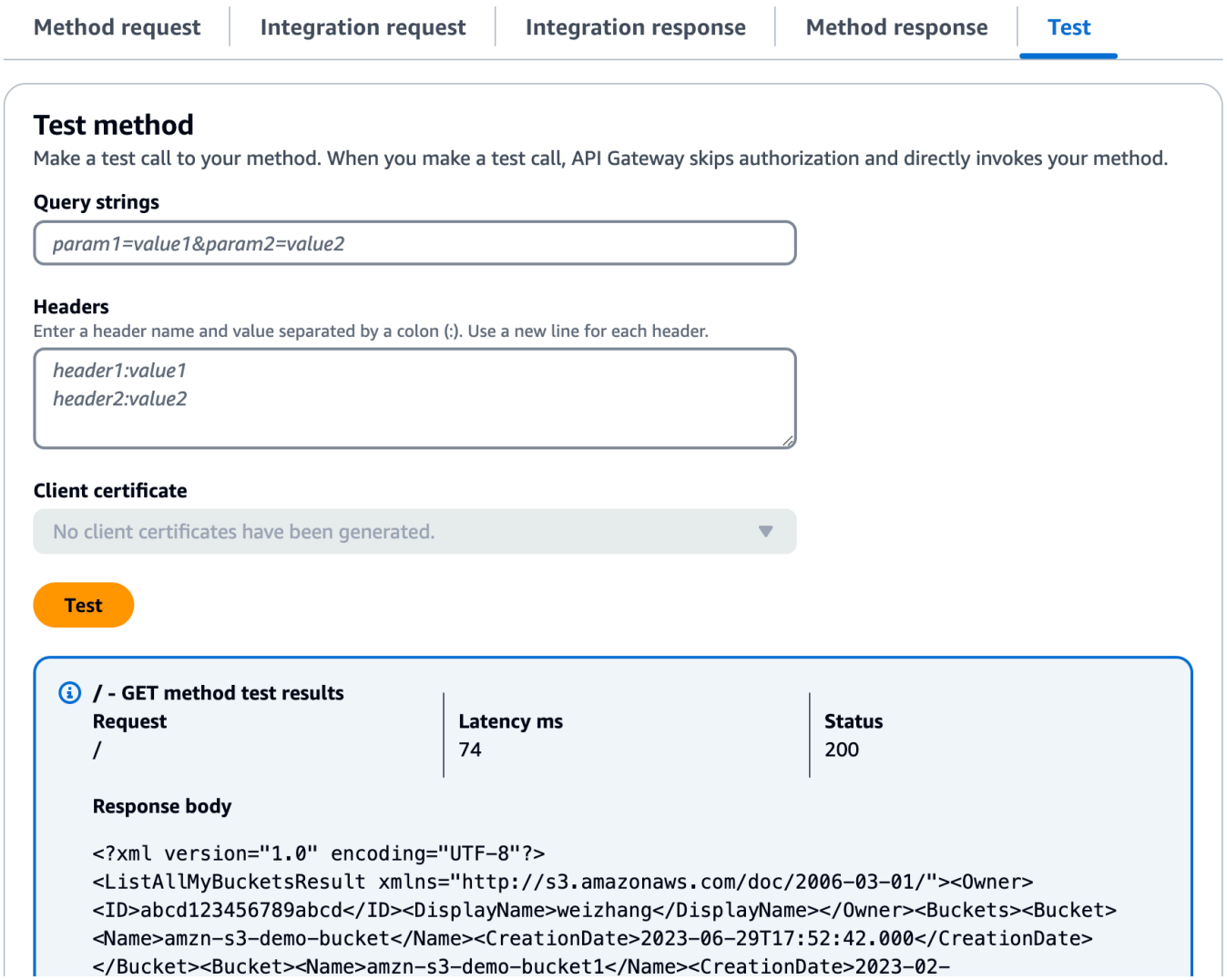
6. Dans l'onglet Réponse d'intégration, sous Réponses d'intégration, choisissez Créer une réponse.
7. Pour HTTP status regex (Regex statut HTTP), saisissez **`4\d{2}`**. Cela mappe tous les codes de statut de réponse HTTP 4xx à la réponse de la méthode.
8. Pour Code de statut de la réponse de méthode,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

À titre de 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 :



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

Pour utiliser un compartiment Amazon S3, vous allez exposer la méthode `GET` sur la ressource /{folder} afin de répertorier les objets dans 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 **`{dossier}`** à **`{bucket}`**

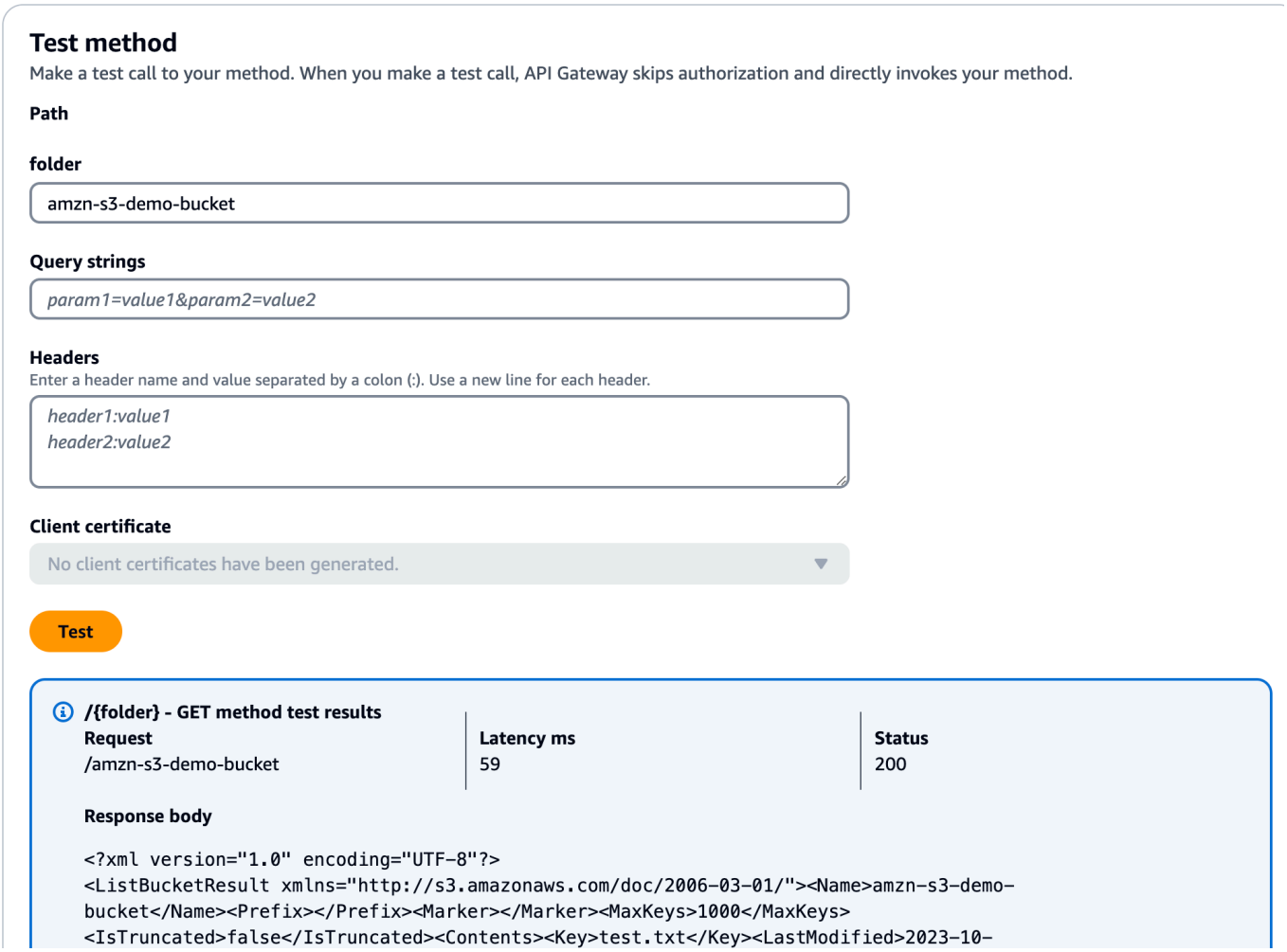
1. Dans l'onglet Requête d'intégration, sous Paramètres de requête d'intégration, choisissez Modifier.
2. Choisissez Paramètres de chemin d'URL, puis choisissez Ajouter un paramètre de chemin.
3. Pour Nom, saisissez **`bucket`**.
4. Pour Mappage à partir de, entrez **`method.request.path.folder`**.
5.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

Maintenant, vous testez votre API.

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

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 **`{dossier}`** à **`{bucket}`** et **`{item}`** à **`{object}`**

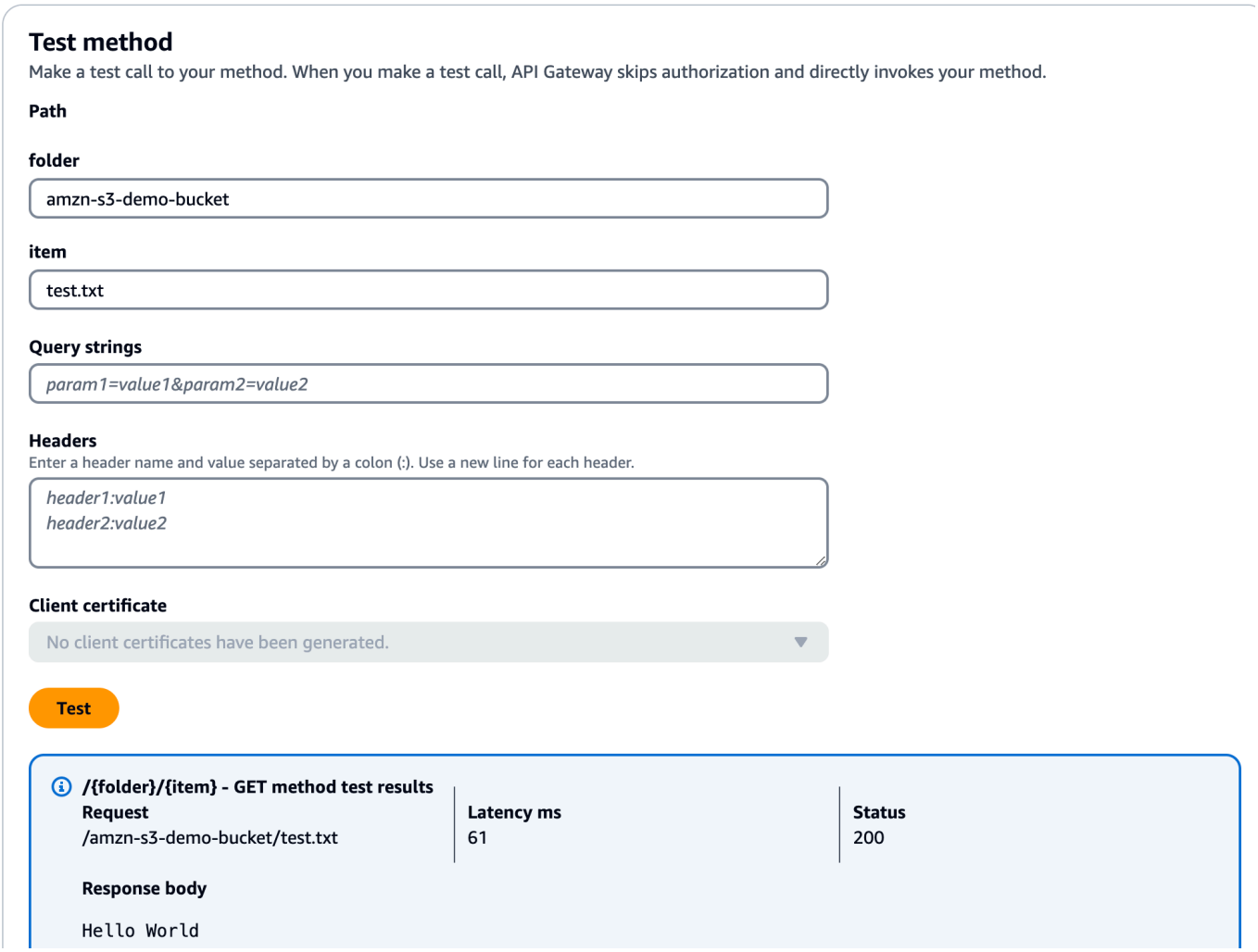
1. Dans l'onglet Requête d'intégration, sous Paramètres de requête d'intégration, choisissez Modifier.
2. Choisissez paramètres de chemin d'URL.

3. Choisissez Ajouter un paramètre de chemin.
4. Pour Nom, saisissez **`bucket`**.
5. Pour Mappage à partir de, entrez **`method.request.path.folder`**.
6. Choisissez Ajouter un paramètre de chemin.
7. Pour Nom, saisissez **`object`**.
8. Pour Mappage à partir de, entrez **`method.request.path.item`**.
9. Cliquez sur Enregistrer.

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

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 (amzn-s3-demo-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 le binaire de l'APIMediaTypes. Vous pouvez réaliser cette opération dans la console :

   a. Choisissez Paramètres de l'API pour l'API.

   b. Sous Types de supports binaires, choisissez Gérer les types de supports.

c. Choisissez Ajouter un type de support binaire, puis saisissez le type de support requis, par exemple `image/png`.

d. Choisissez Save changes (Enregistrer les modifications) pour sauvegarder le paramètre.

2. Ajoutez l'en-tête `Content-Type` (for upload) and/or `Accept` (for download) à la demande de méthode pour demander au client de spécifier le type de support binaire requis et de le mapper 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 the section called "Transformations de données".

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

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 plus d'informations sur la prise en charge des fichiers binaires dans API Gateway, consultez 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.
- Exposez GET sur une Folder/Item ressource pour afficher ou télécharger un objet depuis un compartiment Amazon S3.

- Exposez PUT sur une Folder/Item ressource pour télécharger un objet vers un compartiment Amazon S3.
- Exposez HEAD sur une Folder/Item ressource pour obtenir les métadonnées de l'objet dans un compartiment Amazon S3.
- Exposez DELETE sur une Folder/Item ressource pour supprimer un objet d'un compartiment Amazon S3.

Pour plus d'informations sur l'importation d'une API à l'aide de la définition OpenAPI, consultez Développement d'API REST à l'aide d'OpenAPI dans API Gateway.

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

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 didacticiel complet, 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 d’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 AccessKey et de SecretKey saisie,

respectivement. Entrez l'adresse Région AWS vers laquelle votre API est déployée dans la AWS zone de texte Région. Saisissez `execute-api` dans le champ de saisie Nom du service.

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 `amzn-s3-demo-bucket` à votre compte Amazon S3 dans la région *`{region}`* :

   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 **`amzn-s3-demo-bucket`** 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/amzn-s3-demo-bucket/Readme.txt HTTP/1.1
   Host: 9gn28ca086.execute-api.{region}.amazonaws.com
   Content-Type: application/xml
   X-Amz-Date: 20161015T062647Z
   Authorization: AWS4-HMAC-SHA256 Credential=access-key-id/20161015/{region}/execute-
   api/aws4_request, SignedHeaders=content-length;content-type;host;x-amz-date,
    Signature=ccadb877bdb0d395ca38cc47e18a0d76bb5eaf17007d11e40bf6fb63d28c705b
   Cache-Control: no-cache
   Postman-Token: 6135d315-9cc4-8af8-1757-90871d00847e
   ```

```
Hello, World!
```

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 `amzn-s3-demo-bucket`, exécutez une demande GET comme suit :

```
GET /S3/amzn-s3-demo-bucket/Readme.txt HTTP/1.1
Host: 9gn28ca086.execute-api.{region}.amazonaws.com
Content-Type: application/xml
X-Amz-Date: 20161015T063759Z
Authorization: AWS4-HMAC-SHA256 Credential=access-key-id/20161015/{region}/
execute-api/aws4_request, SignedHeaders=content-type;host;x-amz-date,
 Signature=ba09b72b585acf0e578e6ad02555c00e24b420b59025bc7bb8d3f7aed1471339
Cache-Control: no-cache
Postman-Token: d60fcb59-d335-52f7-0025-5bd96928098a
```

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 `amzn-s3-demo-bucket`, envoyez la demande suivante :

```
GET /S3/amzn-s3-demo-bucket HTTP/1.1
Host: 9gn28ca086.execute-api.{region}.amazonaws.com
Content-Type: application/xml
X-Amz-Date: 20161015T064324Z
Authorization: AWS4-HMAC-SHA256 Credential=access-key-id/20161015/{region}/
execute-api/aws4_request, SignedHeaders=content-type;host;x-amz-date,
 Signature=4ac9bd4574a14e01568134fd16814534d9951649d3a22b3b0db9f1f5cd4dd0ac
Cache-Control: no-cache
Postman-Token: 9c43020a-966f-61e1-81af-4c49ad8d1392
```

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.

Didacticiel : création d'une API REST en tant que proxy Amazon Kinesis

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 avec Kinesis, vous devez choisir une région où les services API Gateway et Kinesis sont disponibles. Pour connaître la disponibilité dans les régions, consultez Points de terminaison de service et quotas.

À 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, consultez 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 Développement d'API REST à l'aide d'OpenAPI dans API Gateway.

Création d'un rôle et d'une politique 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. Au cours de cette étape, vous allez créer un rôle IAM.

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

1. Connectez-vous à la console IAM Console de gestion AWS et ouvrez-la à https://console.aws.amazon.com/iam/l'adresse.
2. Sélectionnez Roles.
3. Choisissez 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 politique gérée pour plus de simplicité. Il est recommandé de créer votre propre politique 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 AWS proxy 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. Pour Type d'adresse IP, sélectionnez IPv4.
7. 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 avec 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 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 Content-Type, 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 allons exposer les méthodes HTTP `GET`, `POST` et `Delete` sur une ressource `Stream` à nommer. Nous utilisons la variable de chemin `{stream-name}` comme espace réservé de la ressource de flux et intégrons respectivement ces méthodes d'API avec les actions `DescribeStream`, `CreateStream` et `DeleteStream` de Kinesis, 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. Sélectionnez la ressource /streams, puis choisissez Créer une ressource.
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. Sélectionnez la ressource /{stream-name}, 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 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": "68056473384187692692674921486353642290",
          "StartingHashKey": "0"
        },
        "SequenceNumberRange": {
          "StartingSequenceNumber":
 "49559266461454070523309915164834022007924120923395850242"
        },
        "ShardId": "shardId-000000000000"
      },
      ...
      {
        "HashKeyRange": {
          "EndingHashKey": "340282366920938463463374607431768211455",
          "StartingHashKey": "272225893536750770770699685945414569164"
        },
        "SequenceNumberRange": {
          "StartingSequenceNumber":
 "49559266461543273504104037657400164881014714369419771970"
        },
        "ShardId": "shardId-000000000004"
      }
    ],
    "StreamARN": "arn:aws:kinesis:us-east-1:12345678901:stream/myStream",
    "StreamName": "myStream",
    "StreamStatus": "ACTIVE"
  }
}
```

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. Sélectionnez la ressource /{stream-name}, 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 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. Sélectionnez la ressource /{stream-name}, puis choisissez Créer une méthode.
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

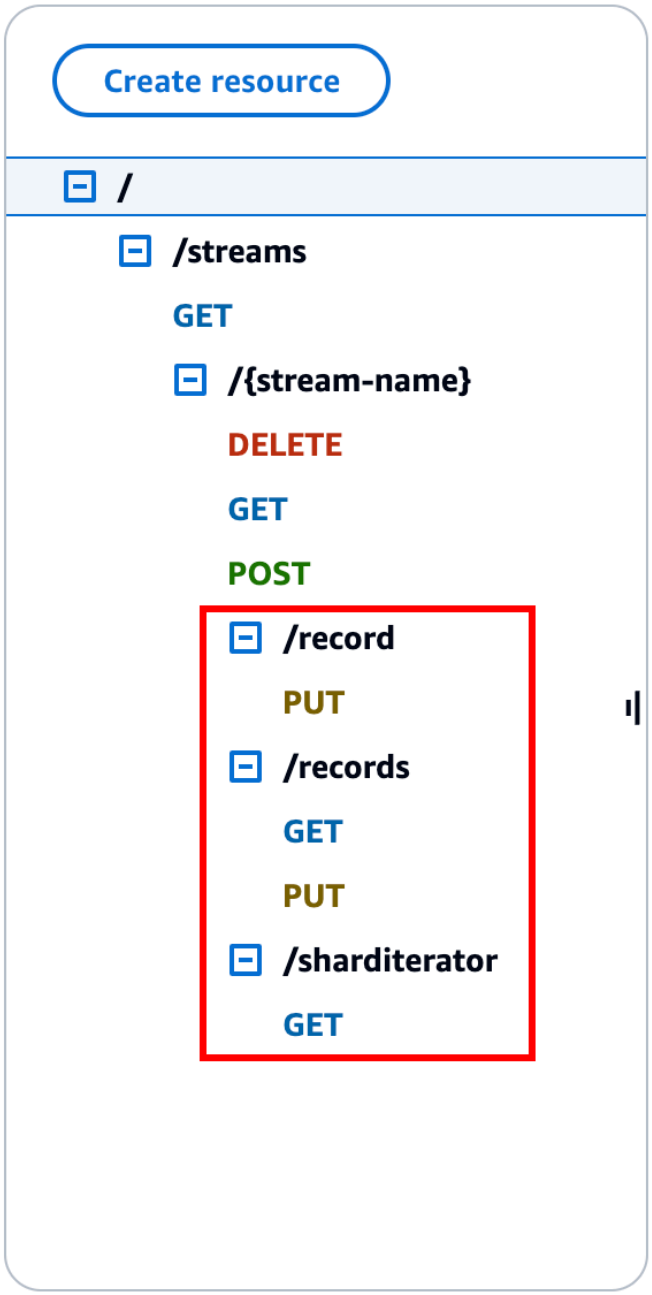
{
    "ShardId": "string",
    "ShardIteratorType": "string",
    "StartingSequenceNumber": "string",
    "StreamName": "string"
}
```

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`.

Étant 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. Sélectionnez la ressource /{stream-name}, puis choisissez Créer une ressource.
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 :



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`** à invoquer dans Kinesis : **`PutRecord`**

1. Sélectionnez /record, puis choisissez Créer une méthode.

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`** à invoquer dans Kinesis **`PutRecords`**

1. Sélectionnez /records, puis choisissez Créer une méthode.
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`**, invoquez Kinesis **`GetShardIterator`**

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. Sélectionnez la ressource /sharditerator, 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 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'action `GetShardIterator` nécessite une valeur ShardId en entrée. 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 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`** afin d'invoquer l'action dans Kinesis **`GetRecords`**

1. Sélectionnez /records, 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 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. Développez En-têtes de demande HTTP.

    L'action `GetRecords` nécessite une valeur `ShardIterator` en entrée. Pour transmettre une valeur `ShardIterator` fournie par le client, nous ajoutons un paramètre d'en-tête `Shard-Iterator` à la demande de méthode.
13. Sélectionnez Add header.
14. Pour 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 didacticiel.

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 REST à l'aide de AWS SDK ou AWS CLI

Utilisez le didacticiel suivant pour créer une PetStore API prenant en charge les `GET /pets/{petId}` méthodes `GET /pets` et. Ces méthodes sont intégrées avec un point de terminaison

HTTP. Vous pouvez suivre ce didacticiel à l'aide du AWS SDK pour JavaScript, le AWS SDK pour Python (Boto3), ou le AWS CLI. Vous pouvez utiliser les fonctions ou commandes suivantes pour configurer votre API :

JavaScript v3

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

Python

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

AWS CLI

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

Pour plus d'informations sur le AWS SDK pour la JavaScript version v3, voir À quoi sert le AWS SDK ? JavaScript . Pour plus d'informations sur le kit SDK pour Python (Boto3), consultez AWS SDK pour Python (Boto3). Pour plus d'informations sur le AWS CLI, voir Qu'est-ce que le AWS CLI ? .

Configurer une API optimisée pour les périphériques PetStore

Dans ce didacticiel, les exemples de commande utilisent des valeurs d'espace réservé pour les ID de valeur tels que l'ID d'API et l'ID de ressource. Lorsque vous réalisez ce didacticiel, remplacez ces valeurs par les vôtres.

Pour configurer une API optimisée pour les périphériques à l'aide PetStore de AWS Kits SDK

1. Utilisez l'exemple suivant pour créer une entité `RestApi` :

   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 l'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 l'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 l'ID d'API `abcd1234` et l'ID de ressource racine `efg567`. Vous allez utiliser ces valeurs pour configurer votre API.

2. Ensuite, vous allez ajouter une ressource enfant sous la racine et spécifier `RootResourceId` comme valeur de la propriété `parentId`. Utilisez l'exemple suivant pour créer une ressource `/pets` 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 dans une sortie similaire à la suivante :

```
{
    "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 dans une sortie similaire à la suivante :

```
{'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 ressource `/pets` que vous avez créée possède l'ID de ressource `aaa111`. Vous allez utiliser cette valeur pour configurer votre API.

3. Ensuite, vous allez ajouter une ressource enfant sous la ressource `/pets`. La ressource enfant `/{petId}` possède un paramètre de chemin pour `{petId}`. Pour qu'un chemin fasse partie d'un paramètre de chemin, vous devez l'entourer d'une paire d'accolades, `{ }`. Utilisez l'exemple suivant pour créer une ressource `/pets/{petId}` 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 dans une sortie similaire à la suivante :

```
{
    "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 dans une sortie similaire à la suivante :

```
{'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}'
```

Voici la sortie de cette commande :

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

La ressource `/pets/{petId}` que vous avez créée possède l'ID de ressource `bbb222`. Vous allez utiliser cette valeur pour configurer votre API.

4. Au cours des deux étapes suivantes, vous allez ajouter des méthodes HTTP à vos ressources. Dans ce didacticiel, vous allez définir les méthodes pour qu'elles soient en accès libre en définissant `authorization-type` sur `NONE`. Pour autoriser uniquement les utilisateurs authentifiés à appeler la méthode, vous pouvez utiliser des rôles et des politiques IAM, un mécanisme d'autorisation Lambda (anciennement appelé mécanisme d'autorisation personnalisée) ou un groupe d'utilisateurs Amazon Cognito. Pour plus d'informations, consultez the section called "Contrôle d'accès".

   Utilisez l'exemple suivant pour ajouter la méthode HTTP `GET` à la ressource `/pets` :

   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 une sortie similaire à la suivante :

```
{
    "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 une sortie similaire à la suivante :

```
{'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 la sortie de cette commande :

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

5. Utilisez l'exemple suivant pour ajouter la méthode HTTP `GET` à la ressource `/pets/{petId}` et définir la propriété `requestParameters` de manière à ce qu'elle transmettre la valeur `petId` 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 une sortie similaire à la suivante :

```
{
    "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 une sortie similaire à la suivante :

```
{'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 la sortie 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 à la méthode `GET /pets` :

   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 une sortie similaire à la suivante :

```
{
    "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 une sortie similaire à la suivante :

```
{'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 à la méthode `GET /pets/{petId}` :

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 une sortie similaire à la suivante :

```
{
    "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 une sortie similaire à la suivante :

```
{'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. Utilisez l'exemple suivant pour configurer une intégration pour la méthode `GET /pets` 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 une sortie similaire à la suivante :

```
{
    "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 une sortie similaire à la suivante :

```
{'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. Utilisez l'exemple suivant pour configurer une intégration pour la méthode `GET /pets/{petId}` avec un point de terminaison HTTP. Le point de terminaison HTTP est `http://petstore-demo-endpoint.execute-api.com/petstore/pets/{id}`. À cette étape, vous allez mapper le paramètre de chemin `petId` à celui 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 une sortie similaire à la suivante :

```
{
    "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 une sortie similaire à la suivante :

```
{'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. Utilisez l'exemple suivant pour ajouter la réponse d'intégration pour l'intégration `GET /pets` :

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 une sortie similaire à la suivante :

```
{
    "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 une sortie similaire à la suivante :

```
{'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. Utilisez l'exemple suivant pour ajouter la réponse d'intégration pour l'intégration `GET /pets/{petId}` :

    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 une sortie similaire à la suivante :

```
{
    "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 une sortie similaire à la suivante :

```
{'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 afficher un animal de compagnie individuel avec un identifiant spécifié. Avant que votre API puisse être appelée par vos clients, vous devez la déployer. Nous vous recommandons de la tester avant de la déployer.

12. Utilisez l'exemple suivant pour tester la méthode `GET /pets` :

    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. Utilisez l'exemple suivant pour tester la méthode `GET /pets/{petId}` 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 votre API, vous pouvez la déployer dans une étape.

14. Utilisez l'exemple suivant pour déployer votre API dans l'étape nommée `test`. Lorsque vous déployez votre API dans une étape, les appelants d'API 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"
}
```

Vos clients peuvent désormais appeler votre API. Vous pouvez tester cette API en saisissant l'URL `https://abcd1234.execute-api.us-west-2.amazonaws.com/test/pets` dans un navigateur et en remplaçant `abcd1234` par l'identifiant de votre API.

Pour plus d'exemples de création ou de mise à jour d'une API à l'aide de kits de AWS développement logiciel (SDK) AWS CLI, consultez la section Actions pour API Gateway à l'aide de kits de AWS développement logiciel (SDK).

Automatisation de la configuration de votre API

Au lieu de créer votre API étape par étape, vous pouvez automatiser la création et le nettoyage des AWS ressources en utilisant OpenAPI ou Terraform pour créer votre API. CloudFormation

Définition OpenAPI 3.0

Vous pouvez importer une définition 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 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
}

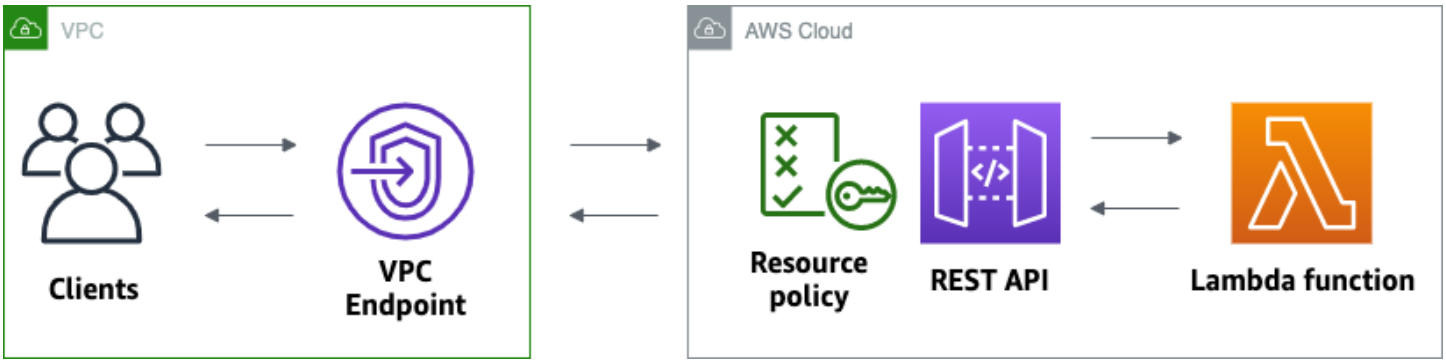

resource "aws_api_gateway_deployment" "Deployment" {
  rest_api_id = aws_api_gateway_rest_api.Api.id
  depends_on  =
 [aws_api_gateway_integration.petsIntegration,aws_api_gateway_integration.petIdIntegration ]
}
resource "aws_api_gateway_stage" "Stage" {
  stage_name    = "Prod"
  rest_api_id   = aws_api_gateway_rest_api.Api.id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

Didacticiel : création d’une API REST privée

Dans ce didacticiel, 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 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 Console de gestion AWS 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 Gestion des identités et des accès AWS utilisateur ayant accès à la console. Pour de plus amples informations, veuillez consulter Conditions préalables.

Dans ce didacticiel, vous utiliserez l' Console de gestion AWS. Pour un 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 politique 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 CloudFormation

Étape 1 : création de dépendances

Téléchargez et décompressez ce 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 un CloudFormation pile

1. Ouvrez la 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 CloudFormation vous pouvez créer des ressources IAM dans votre compte.
9. Choisissez Suivant, puis Soumettre.

CloudFormation fournit les dépendances de votre API, ce qui peut prendre quelques minutes. Lorsque le statut de votre 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 didacticiel.

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. CloudFormation
6. Pour Type d'adresse IP, choisissez Double pile.
7. 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. 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 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 politique de ressources

Vous attachez une politique 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 politique 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 politique 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 politique suivante. *`vpceID`*Remplacez-le par l'ID de point de terminaison de votre VPC figurant dans les sorties de votre CloudFormation stack.

JSON

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

5. 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-tutorial que vous avez créée avec le modèle CloudFormation 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 didacticiel. Les étapes suivantes suppriment votre API REST et votre 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 un CloudFormation pile

1. Ouvrez la CloudFormation console à l'adresse https://console.aws.amazon.com/cloudformation.
2. Sélectionnez votre CloudFormation pile.
3. Choisissez Supprimer, puis confirmez votre choix.

Prochaines étapes : Automatisez avec CloudFormation

Vous pouvez automatiser la création et le nettoyage de toutes les AWS ressources impliquées dans ce didacticiel. Pour un exemple complet de modèle 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

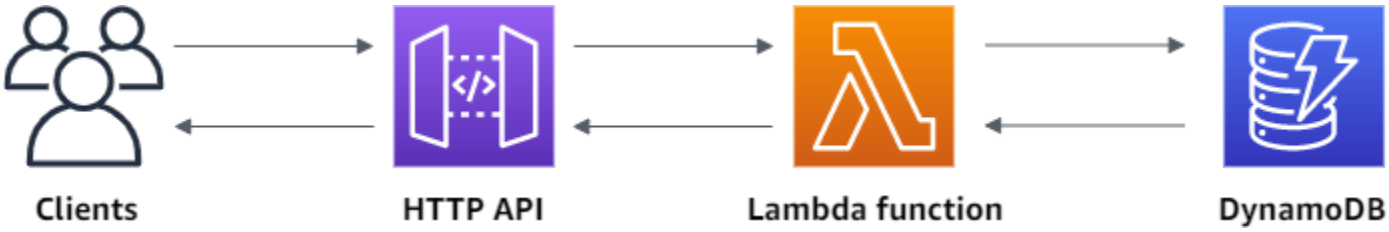
- Didacticiel : création d'une API HTTP CRUD avec Lambda et DynamoDB
- Didacticiel : création d'une API HTTP avec une intégration privée à un service Amazon ECS

Didacticiel : création d'une API HTTP CRUD avec Lambda et DynamoDB

Dans ce didacticiel, vous allez créer une API sans serveur 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 didacticiel 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 Gestion des identités et des accès AWS utilisateur disposant d'un accès à la console. Pour de plus amples informations, veuillez consulter Conditions préalables.

Dans ce didacticiel, vous utiliserez l' Console de gestion AWS. 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 2 : création d'une fonction Lambda
- Étape 3 : création d'une API HTTP
- Étape 4 : création de routes
- Étape 5 : création d'une intégration
- Étape 6 : attachement de votre intégration aux routes
- Étape 7 : test de votre API
- Étape 8 : Nettoyage
- Prochaines étapes : Automatisez avec AWS SAM or CloudFormation

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 plus d'informations, consultez Le monolithe Lambda (français non garanti).

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 Runtime, choisissez soit le dernier runtime compatible, Node.jssoit le dernier runtime Python.
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 politique), choisissez **`Simple microservice permissions`**. Cette politique accorde à la fonction Lambda l'autorisation d'interagir avec DynamoDB.

> **Note**
>
> Ce didacticiel utilise une politique gérée pour plus de simplicité. Il est recommandé de créer votre propre politique 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 routes 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 (Création).
3. Sous API name (Nom de l'API), saisissez **`http-crud-tutorial-api`**.

4. Pour Type d'adresse IP, sélectionnez IPv4.
5. Choisissez Suivant.
6. Sous Configure routes (Configurer des routes), choisissez Next (Suivant) afin d'ignorer la création de routes. Vous créerez des routes ultérieurement.
7. Consultez l'étape créée par API Gateway pour vous, puis choisissez Next (Suivant).
8. Choisissez Créer.

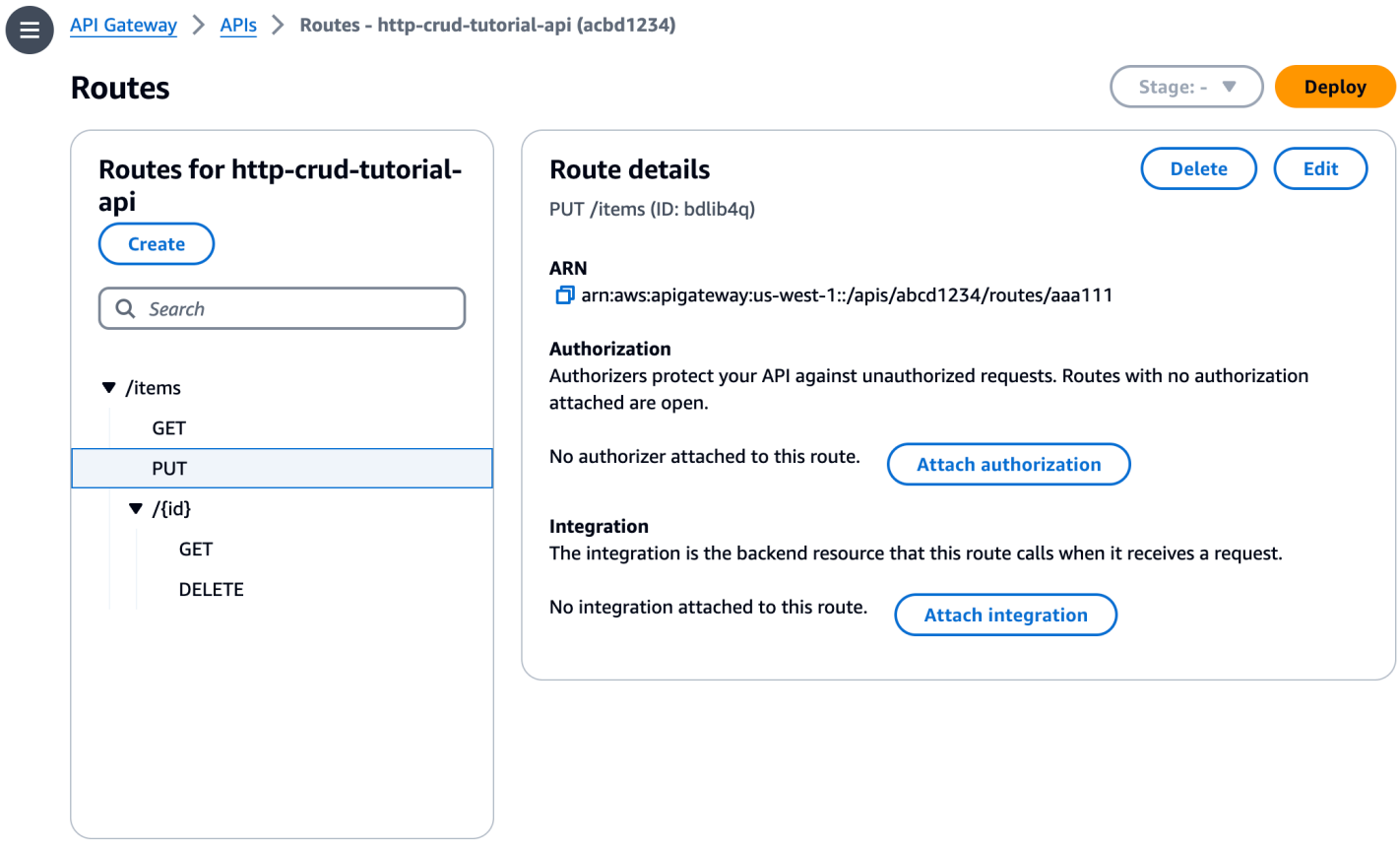
Étape 4 : création de routes

Les routes permettent d'envoyer les demandes d'API entrantes aux ressources backend. Les routes 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 routes :

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

Pour créer des routes

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Choisissez votre API.
3. Choisissez Routes.
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 une route aux ressources backend. Pour cet exemple d'API, vous créez une intégration Lambda que vous utilisez pour toutes les routes.

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 une route). 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.

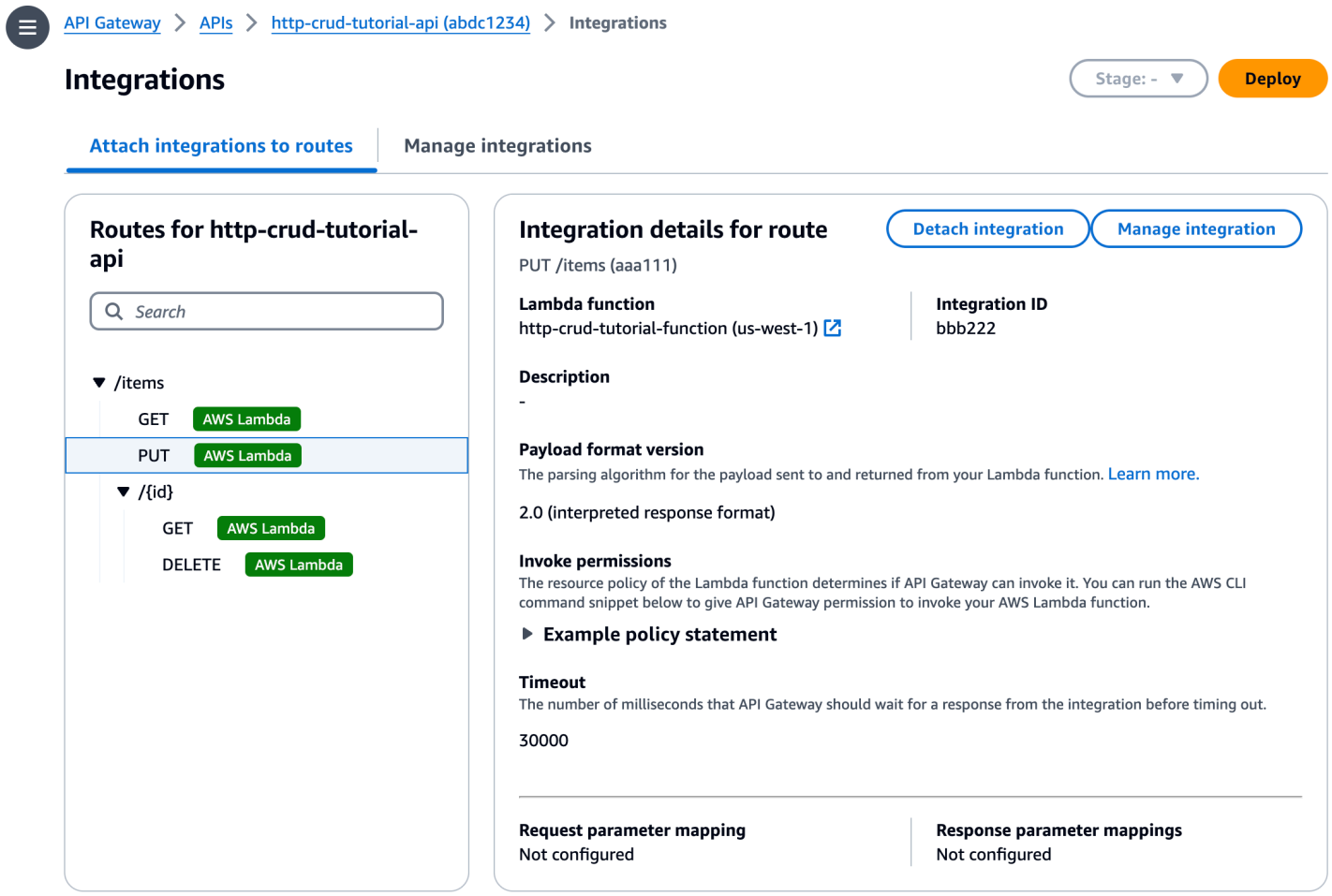
Étape 6 : attachement de votre intégration aux routes

Pour cet exemple d'API, vous utilisez la même intégration Lambda pour toutes les routes. Après que l'intégration a été attachée à l'ensemble des routes de l'API, votre fonction Lambda est appelée lorsqu'un client appelle l'une de vos routes.

Pour attacher des intégrations à des routes

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 une route.
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 toutes les routes.

Tous les itinéraires indiquent qu'une AWS Lambda intégration est attachée.



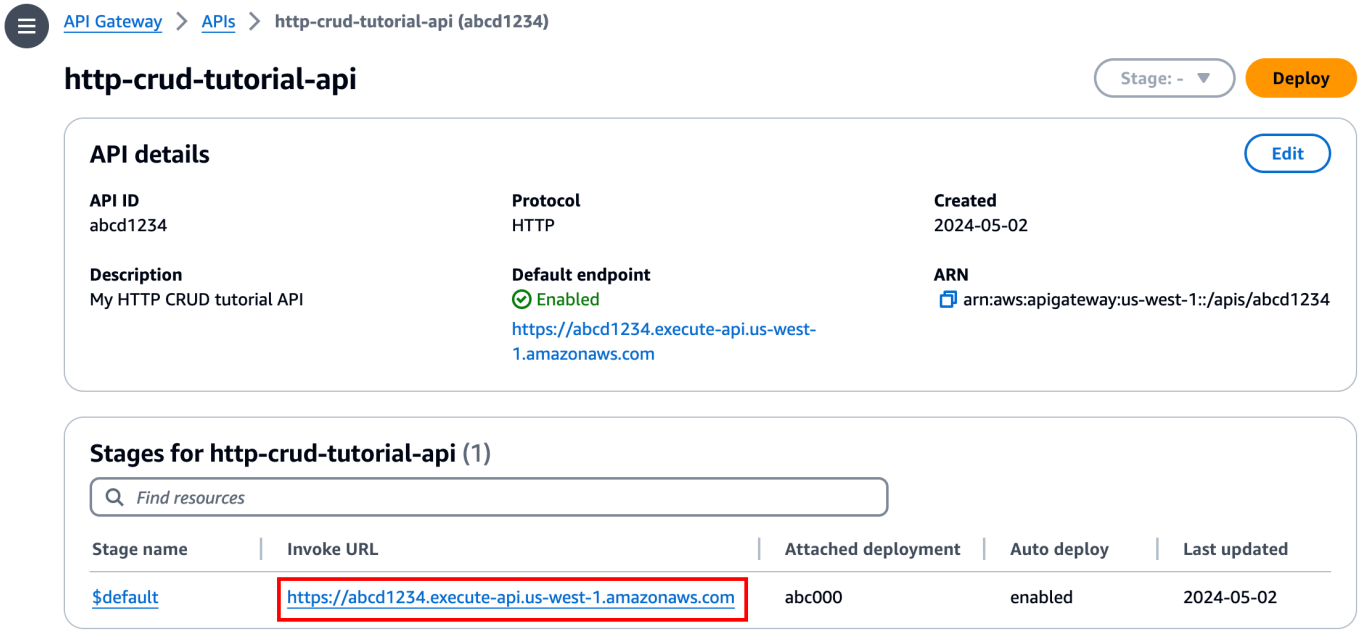
Maintenant que vous disposez d'une API HTTP avec des routes et des intégrations, vous pouvez tester votre API.

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 Gestion des identités et des accès AWS console, ouvrez la page Rôles.
2. Sélectionnez le rôle de la fonction, par exemple, `http-crud-tutorial-role`.
3. Choisissez Supprimer le rôle.
4. Choisissez Oui, supprimer.

Prochaines étapes : Automatisez avec AWS SAM or CloudFormation

Vous pouvez automatiser la création et le nettoyage des AWS ressources en utilisant CloudFormation ou AWS SAM. Afin d'obtenir un exemple de modèle AWS SAM pour ce didacticiel, consultez `template.yaml`.

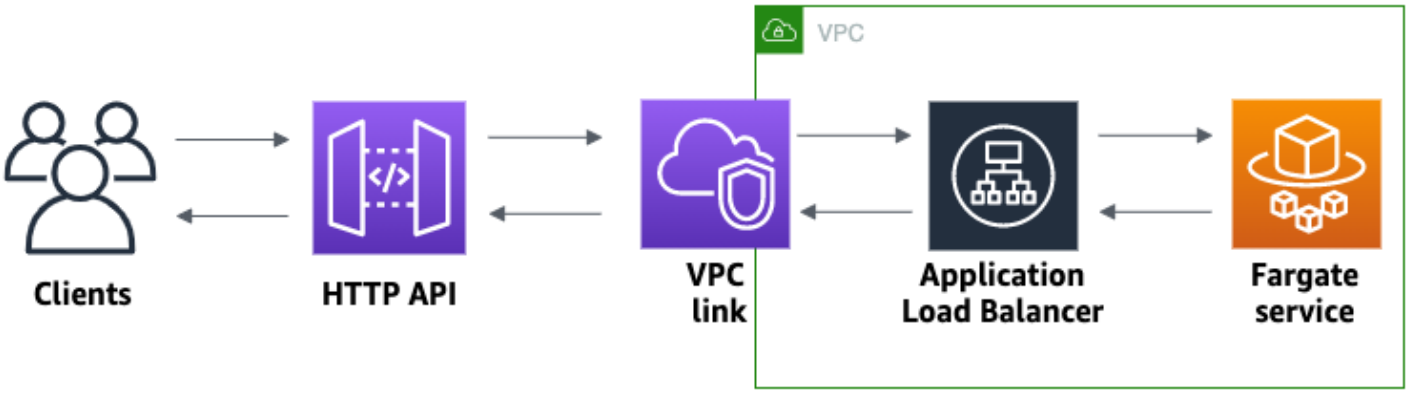
Pour des exemples CloudFormation de modèles, voir les exemples CloudFormation de modèles.

Didacticiel : création d'une API HTTP avec une intégration privée à un service Amazon ECS

Dans ce didacticiel, vous créerez une API sans serveur qui se connecte à un service Amazon ECS qui s'exécute dans un VPC Amazon. Les clients en dehors de votre VPC Amazon peuvent utiliser l'API pour accéder à votre service Amazon ECS.

Ce didacticiel vous prendra environ 1 heure. Tout d'abord, vous utilisez un 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'service Amazon ECS qui s'exécute dans votre VPC Amazon. Ensuite, vous créez une API HTTP qui utilise le lien VPC pour vous connecter à votre service Amazon ECS. Enfin, vous testez votre API.

Lorsque vous appelez votre API HTTP, API Gateway achemine la demande vers votre service Amazon ECS via votre lien VPC, puis renvoie la réponse du service.



Pour suivre ce didacticiel, vous avez besoin d'un AWS compte et d'un Gestion des identités et des accès AWS utilisateur disposant d'un accès à la console. Pour de plus amples informations, veuillez consulter Conditions préalables.

Dans ce didacticiel, vous utiliserez l' Console de gestion AWS. Pour un CloudFormation modèle qui crée cette API et toutes les ressources associées, consultez template.yaml.

Rubriques

- Étape 1 : créer un service Amazon ECS
- Étape 2 : créer un lien VPC
- Étape 3 : créer une API HTTP
- Étape 4 : créer une route
- Étape 5 : créer une intégration
- Étape 6 : tester votre API
- Étape 7 : nettoyer
- Prochaines étapes : Automatisez avec CloudFormation

Étape 1 : créer un service Amazon ECS

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 didacticiel, vous exécuterez votre cluster sur une infrastructure sans serveur gérée par Amazon ECS.

Téléchargez et décompressez ce CloudFormation modèle, qui crée toutes les dépendances du service, y compris un Amazon VPC. Vous utilisez le modèle pour créer un service Amazon ECS qui utilise un Application Load Balancer.

Pour créer un CloudFormation pile

1. Ouvrez la 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 CloudFormation vous pouvez créer des ressources IAM dans votre compte.
9. Choisissez Suivant, puis Soumettre.

CloudFormation fournit le service ECS, ce qui peut prendre quelques minutes. Lorsque le statut de votre CloudFormation pile est CREATE_COMPLETE, vous êtes prêt à passer à l'étape suivante.

Étape 2 : créer 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 service Amazon ECS 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 panneau de navigation principal, choisissez Liens VPC, puis Créer.

   Vous devrez peut-être sélectionner l'icône 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. Son nom doit commencer par PrivateIntegrationsStack.
6. Pour Sous-réseaux, sélectionnez les deux sous-réseaux privés de votre VPC. Leurs noms finissent par `PrivateSubnet`.
7. Pour Groupes de sécurité, sélectionnez l'ID de groupe commençant par `private-integrations-tutorial` et contenant la description `PrivateIntegrationsStack/PrivateIntegrationsTutorialService/Service/SecurityGroup`.
8. 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 : créer une API HTTP

L'API HTTP fournit un point de terminaison HTTP pour votre service Amazon ECS. 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 service Amazon ECS.

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 (Création).
3. Sous API name (Nom de l'API), saisissez **`http-private-integrations-tutorial`**.
4. Pour Type d'adresse IP, sélectionnez IPv4.
5. Choisissez Suivant.
6. Sous Configure routes (Configurer des routes), choisissez Next (Suivant) afin d'ignorer la création de routes. Vous créerez des routes ultérieurement.

7. 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 didacticiel. Choisissez Suivant.
8. Choisissez Créer.

Étape 4 : créer une route

Les routes permettent d'envoyer les demandes d'API entrantes aux ressources backend. Les routes 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 Routes.
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 : créer une intégration

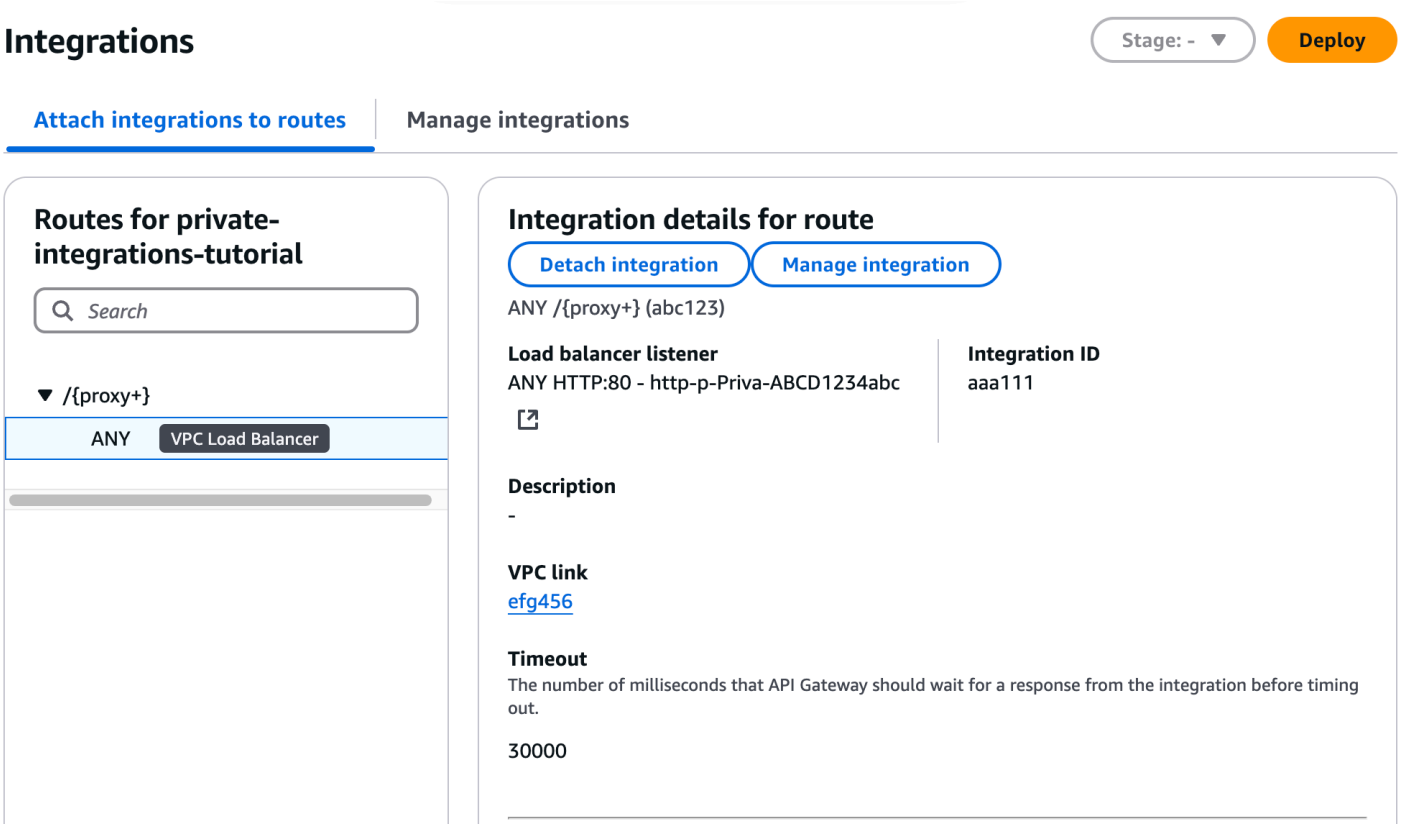
Vous créez une intégration afin de connecter une route aux ressources backend.

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 le service Target, choisissez ALB/NLB.
9. Pour Équilibreur de charge, choisissez l'équilibreur de charge que vous avez créé avec le modèle 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.



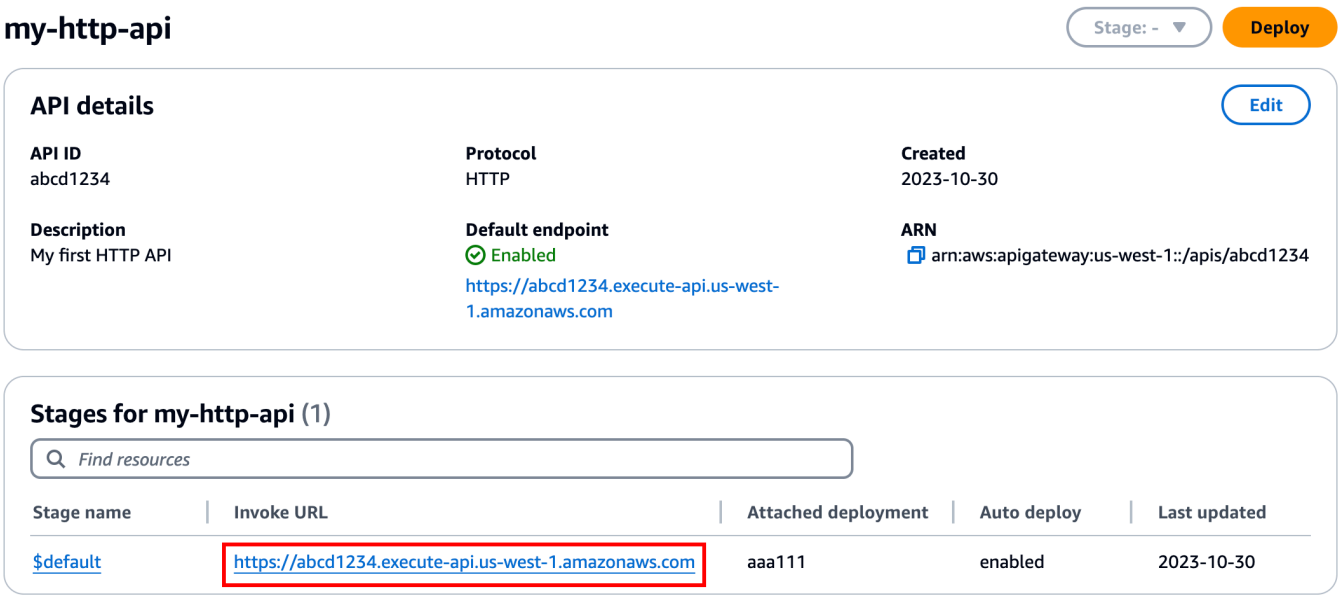
Vous êtes désormais prêt à tester votre API.

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 service Amazon ECS qui s'exécute dans un VPC Amazon et vous avez utilisé une API HTTP API Gateway avec un lien VPC pour accéder au service Amazon ECS.

Étape 7 : nettoyer

Pour éviter des coûts inutiles, supprimez les ressources que vous avez créées dans le cadre de ce didacticiel. Les étapes suivantes suppriment votre lien VPC, votre 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 un CloudFormation pile

1. Ouvrez la CloudFormation console à l'adresse https://console.aws.amazon.com/cloudformation.
2. Sélectionnez votre CloudFormation pile.
3. Choisissez Supprimer, puis confirmez votre choix.

Prochaines étapes : Automatisez avec CloudFormation

Vous pouvez automatiser la création et le nettoyage de toutes les AWS ressources impliquées dans ce didacticiel. Pour un exemple complet de modèle CloudFormation , veuillez consulter template.yaml.

Didacticiels sur les API WebSocket Amazon API Gateway

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

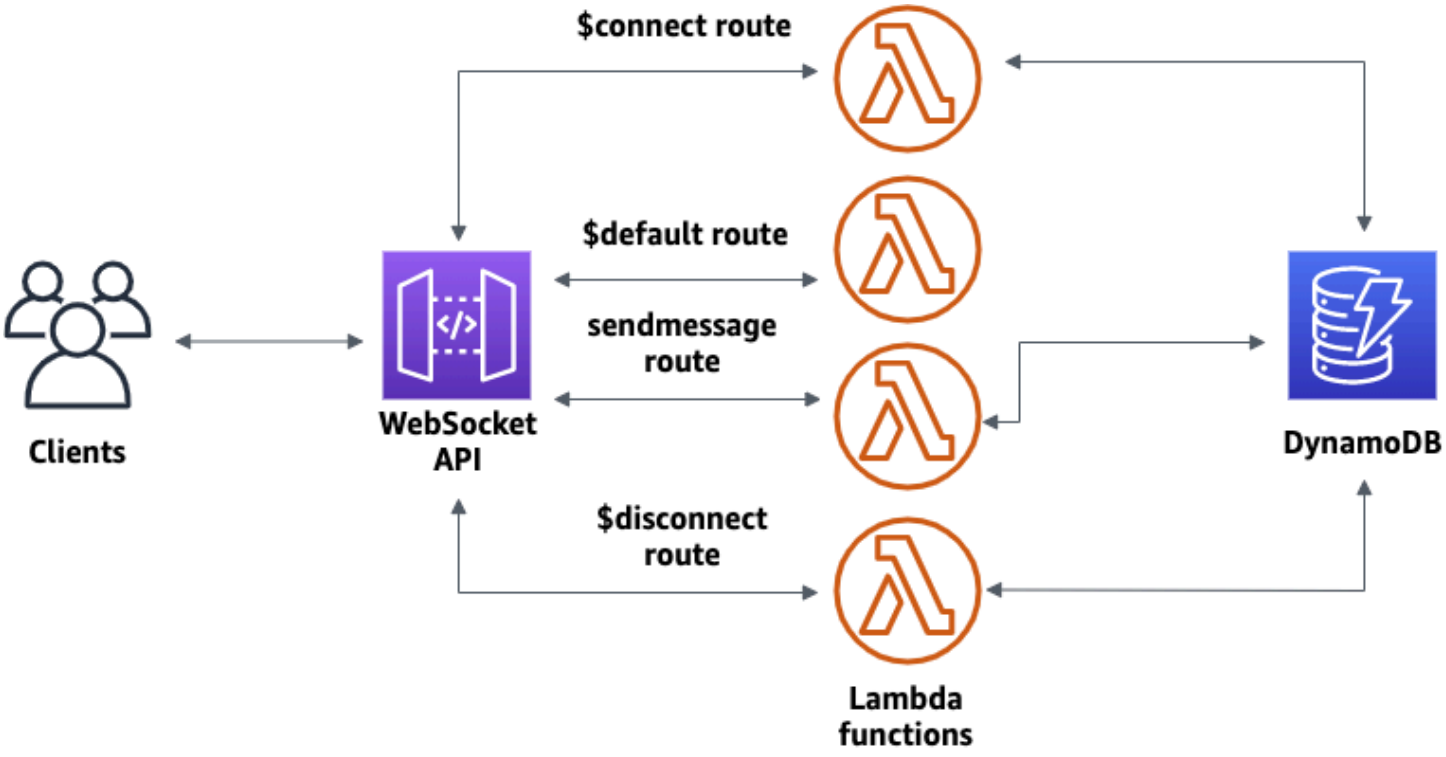
Rubriques

- Tutoriel : Création d'une application de WebSocket chat avec une WebSocket API, Lambda et DynamoDB
- Tutoriel : Création d'une WebSocket API avec un AWS intégration

Tutoriel : Création d'une application de WebSocket chat 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 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 Gestion des identités et des accès AWS utilisateur disposant d'un accès à la console. Pour de plus amples informations, veuillez consulter Conditions préalables.

Vous avez également besoin de `wscat` pour vous connecter à votre API. Pour de plus amples informations, veuillez consulter the section called “`wscat`À utiliser pour se connecter à une WebSocket API et y envoyer des messages”.

Rubriques

- Étape 1 : créer des fonctions Lambda et une table DynamoDB
- Étape 2 : Création d'une WebSocket API
- Étape 3 : tester votre API
- Étape 4 : nettoyer
- Prochaines étapes : Automatisez avec CloudFormation

Étape 1 : créer des fonctions Lambda et une table DynamoDB

Téléchargez et décompressez le modèle de création d'application pour 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 un CloudFormation pile

1. Ouvrez la 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 CloudFormation vous pouvez créer des ressources IAM dans votre compte.

9. Choisissez Suivant, puis Soumettre.

CloudFormation fournit les ressources spécifiées dans le modèle. La fin du provisionnement de vos ressources peut prendre quelques minutes. Lorsque le statut de votre 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 Type d'adresse IP, sélectionnez IPv4.
5. Pour Route selection expression (Expression de sélection de route), entrez **`request.body.action`**. Expression de sélection de route pour déterminer la route à appeler lorsqu'un client envoie un message.
6. Choisissez Suivant.
7. Pour Predefined routes (Routes prédéfinies), choisissez Add $connect (Ajouter $connect), Add $disconnect (Ajouter $disconnect) et Add $default (Ajouter $default). Les routes $connect et $disconnect sont des itinéraires spéciaux qu'API Gateway appelle automatiquement lorsqu'un client se connecte à une API ou se déconnecte de celle-ci. API Gateway invoque la route `$default` lorsqu'aucune autre route ne correspond à une demande.
8. 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.
9. Choisissez Suivant.
10. 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. 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`**.

11. 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.
12. 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 la route appelée 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 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 4 : nettoyer

Pour éviter des coûts inutiles, supprimez les ressources que vous avez créées dans le cadre de ce didacticiel. Les étapes suivantes suppriment votre 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 un CloudFormation pile

1. Ouvrez la CloudFormation console à l'adresse https://console.aws.amazon.com/cloudformation.
2. Sélectionnez votre CloudFormation pile.
3. Choisissez Supprimer, puis confirmez votre choix.

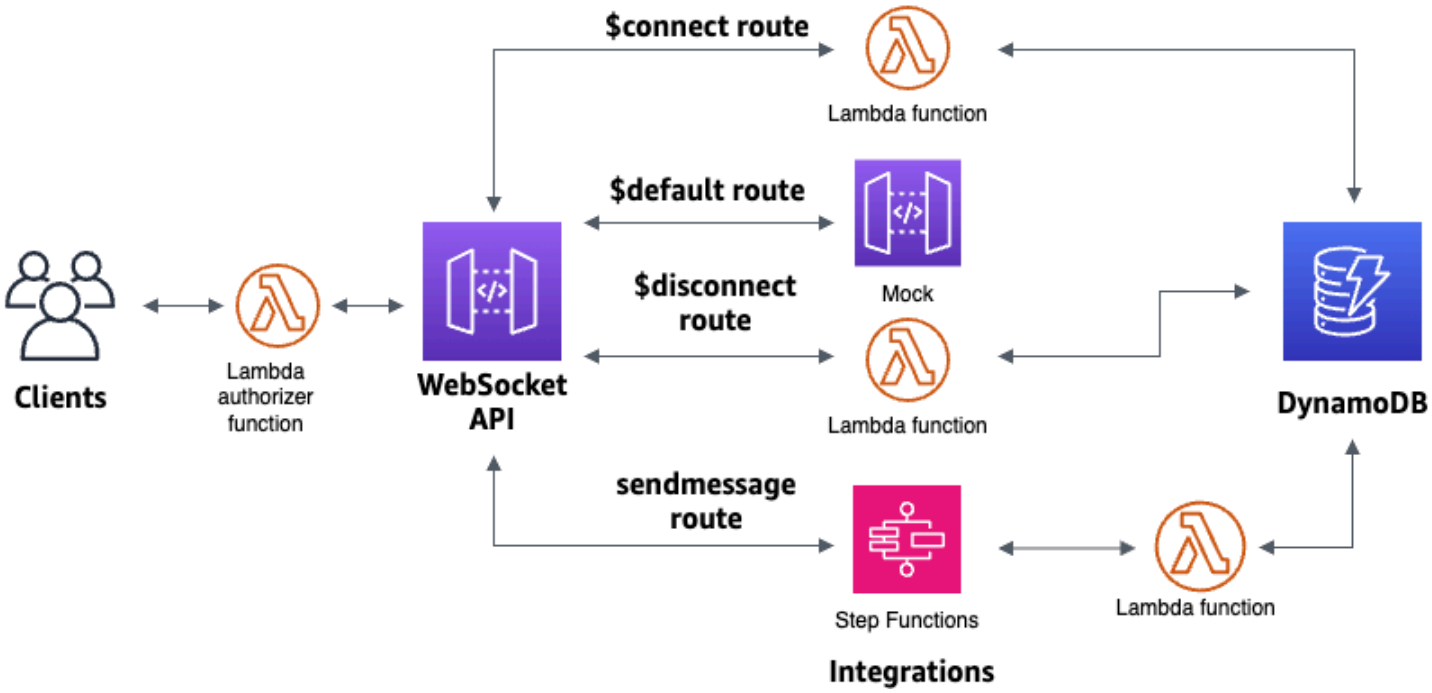
Prochaines étapes : Automatisez avec CloudFormation

Vous pouvez automatiser la création et le nettoyage de toutes les AWS ressources impliquées dans ce didacticiel. Pour un CloudFormation modèle qui crée cette API et toutes les ressources associées, consultez ws-chat-app.yaml.

Tutoriel : Création d'une WebSocket API avec un AWS 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 de mécanisme d’autorisation Lambda, une intégration simulée et une intégration sans proxy avec Step Functions.



Après avoir créé vos ressources à l'aide d'un 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 enverra un message à tous les clients connectés.

Après avoir créé votre API, vous devez tester votre connexion à votre API et vérifier que les messages sont bien envoyés et reçus. Ce didacticiel vous prendra environ 45 minutes.

Rubriques

- Conditions préalables
- Étape 1 : créer des ressources
- Étape 2 : Création d'une WebSocket API
- Étape 3 : créer un mécanisme d'autorisation Lambda
- Étape 4 : créer une intégration bidirectionnelle simulée
- Étape 5 : créer une intégration sans proxy avec Step Functions
- Étape 6 : tester votre API
- Étape 7 : nettoyer
- Étapes suivantes

Conditions préalables

Vous avez besoin des prérequis suivants :

- Un AWS compte et un Gestion des identités et des accès AWS utilisateur ayant accès à la console. Pour de plus amples informations, veuillez consulter Conditions préalables.
- `wscat` pour vous connecter à votre API. Pour de plus amples informations, veuillez consulter 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 “Tutoriel : Création d'une application de WebSocket chat”.

Étape 1 : créer des ressources

Téléchargez et décompressez le modèle de création d'application pour 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.
- Table DynamoDB pour stocker les ID des clients et l’identifiant de l’utilisateur principal renvoyés par le mécanisme d’autorisation Lambda.
- Machine d’état Step Functions pour envoyer des messages aux clients connectés.

Pour créer un CloudFormation pile

1. Ouvrez la 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 CloudFormation vous pouvez créer des ressources IAM dans votre compte.

9. Choisissez Suivant, puis Soumettre.

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 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 Type d'adresse IP, sélectionnez IPv4.
5. Pour Route selection expression (Expression de sélection de route), entrez **`request.body.action`**.

   Expression de sélection de route pour déterminer la route à appeler lorsqu'un client envoie un message.
6. Choisissez Suivant.
7. Pour Routages prédéfinis, choisissez Ajouter $connect, Ajouter $disconnect et 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 devez créer une route personnalisée pour vous connecter à Step Functions après avoir créé votre API.
8. Choisissez Suivant.
9. Pour 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 CloudFormation à l'étape 1. Le nom de la fonction Lambda doit commencer par **`websocket-step`**.

10. Pour 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 CloudFormation à l'étape 1. Le nom de la fonction Lambda doit commencer par **`websocket-step`**.

11. Pour Intégration de $default, choisissez Mock.

    Dans une intégration simulée, API Gateway gère la réponse de routage sans backend d'intégration.

12. Choisissez Suivant.
13. 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 dans cette étape. Choisissez Suivant.
14. Choisissez Create and deploy (Créer et déployer).

Étape 3 : créer un mécanisme d'autorisation Lambda

Pour contrôler l'accès à votre WebSocket API, vous devez créer un autorisateur Lambda. Le CloudFormation modèle a créé la fonction d'autorisation Lambda pour vous. Vous pouvez voir la fonction Lambda dans la console Lambda. Son 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 variable `$context.authorizer.principalId` à votre API, qui est ensuite utilisée dans la table DynamoDB pour identifier les appelants d'API.

Au cours de cette étape, vous allez configurer la route $connect pour utiliser le mécanisme d'autorisation Lambda.

Pour créer un mécanisme d'autorisation 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 mécanisme d'autorisation.
4. Pour Nom du mécanisme d'autorisation, saisissez **`LambdaAuthorizer`**.
5. Pour Authorizer ARN, entrez le nom de l'autorisateur créé par le CloudFormation modèle. Son nom doit commencer par **`websocket-step-functions-tutorial-AuthorizerHandler`**.

> **Note**
>
> Nous vous recommandons de ne pas utiliser cet exemple de mécanisme d'autorisation pour vos API de production.

6. Pour Type de source d'identité, choisissez En-tête. Pour Key (Clé), saisissez **`Authorization`**.
7. Choisissez Créer un mécanisme d'autorisation.

Après avoir créé votre mécanisme d'autorisation, vous devez l'attacher à la route $connect de votre API.

Pour attacher un mécanisme d'autorisation à la route $connect

1. Dans le volet de navigation principal, sélectionnez Routes.
2. Choisissez la route $connect.
3. Dans la section Paramètres de requête de routage, choisissez Modifier.
4. Pour Autorisation, sélectionnez le menu déroulant, puis votre mécanisme d'autorisation de requête.
5. Sélectionnez Enregistrer les modifications.

Étape 4 : créer une intégration bidirectionnelle simulée

Vous allez ensuite créer l'intégration bidirectionnelle simulée pour la route $default. Une intégration simulée 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 devez configurer la route $default pour indiquer aux clients d'utiliser la route sendmessage.

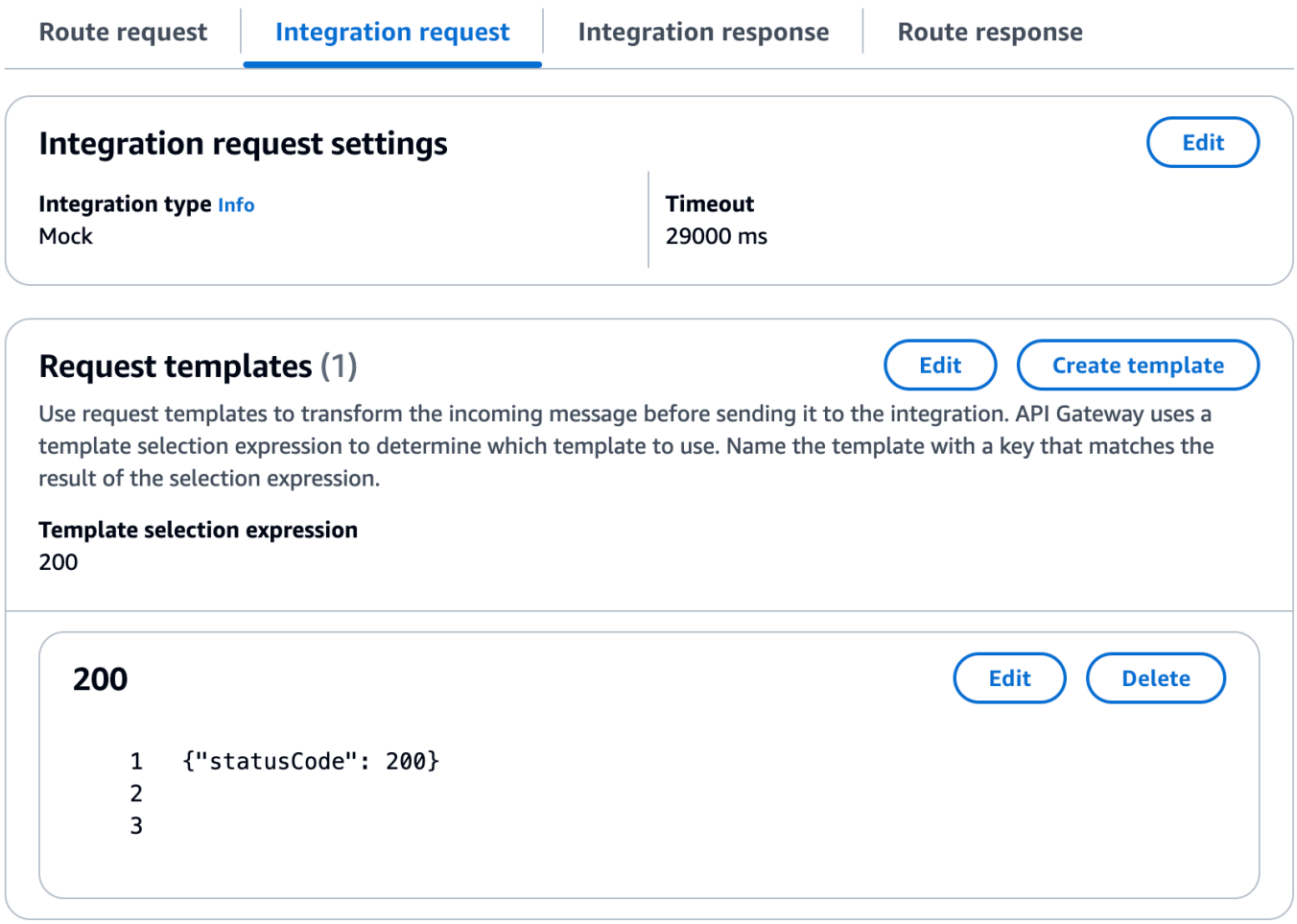
Pour créer une intégration simulée

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Choisissez la route $default, puis l'onglet Requête d'intégration.
3. Pour Modèles de requête, choisissez Modifier.
4. Pour Expression de sélection de modèle, saisissez **`200`**, puis choisissez Modifier.
5. Dans l'onglet Requête d'intégration, pour Modèles de requête, choisissez Créer un modèle.
6. Pour Clé de modèle, saisissez **`200`**.
7. Pour Générer un modèle, saisissez le modèle de mappage suivant :

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

   Sélectionnez Create template (Créer un modèle).

   Le résultat doit avoir l'aspect suivant :

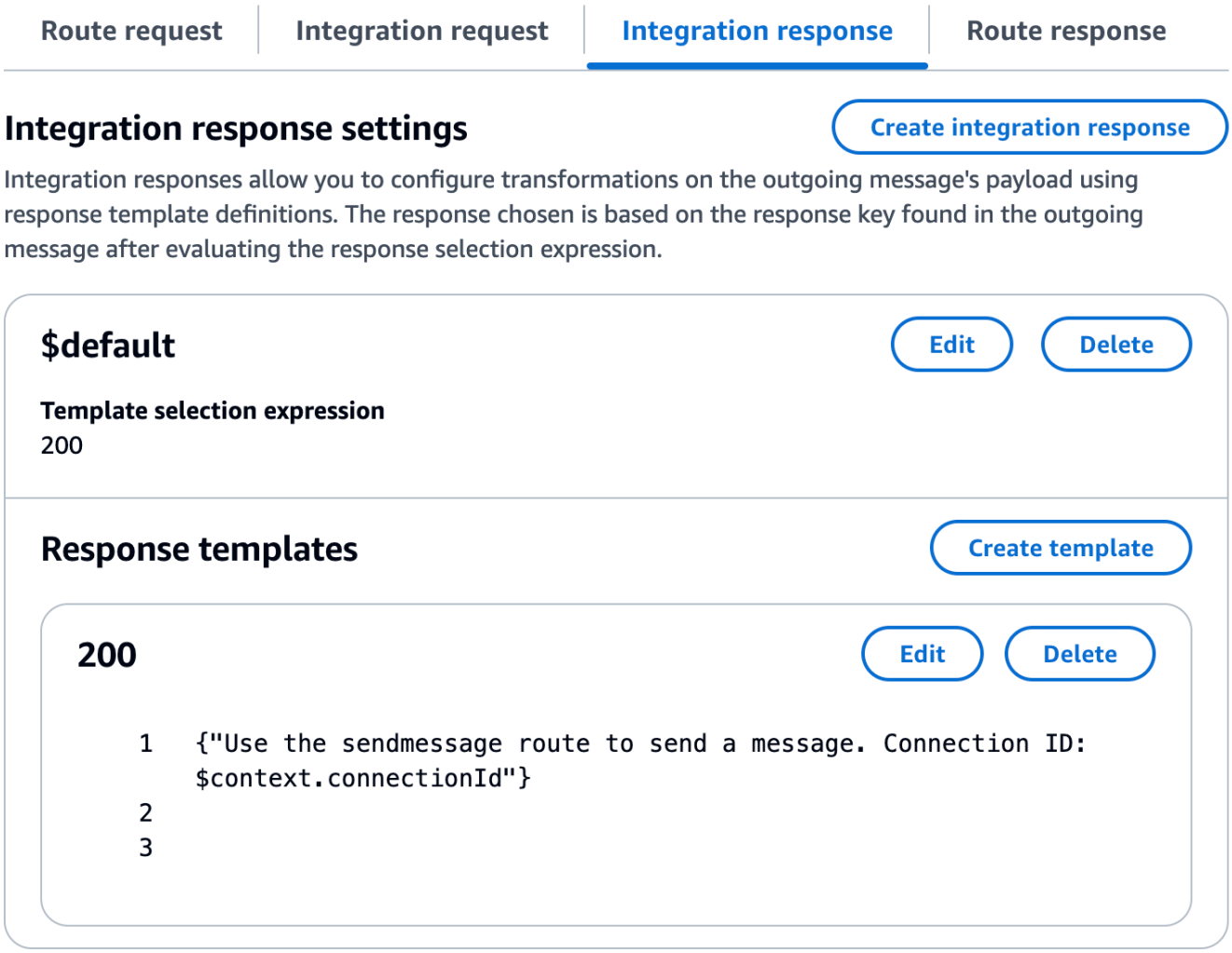


8. Dans le volet Route $default, choisissez Activer la communication bidirectionnelle.
9. Choisissez l'onglet Réponse d'intégration, puis sélectionnez Créer une réponse d'intégration.
10. Pour Clé de réponse, saisissez **`$default`**.
11. Pour Expression de sélection de modèle, saisissez **`200`**.
12. Choisissez Créer une réponse.
13. Sous Modèles de réponse, choisissez Créer un modèle.
14. Pour Clé de modèle, saisissez **`200`**.
15. Pour Modèle de réponse, saisissez 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 : créer une intégration sans proxy avec Step Functions

Vous allez ensuite créer une route sendmessage. 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 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. Pour Type d’intégration, choisissez Service AWS .
5. Pour AWS Région, entrez la région dans laquelle vous avez déployé votre CloudFormation modèle.
6. Pour Service AWS , 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 CloudFormation modèle. Le nom doit être WebsocketTutorialApiRole.
10. Choisissez Create Route (Créer un itinéraire).

Vous allez ensuite créer un modèle de mappage pour envoyer les paramètres de requête à la machine d’état Step Functions.

Pour créer un modèle de mappage

1. Choisissez la route sendmessage, puis l’onglet Requête d’intégration.
2. Dans la section Modèles de requête, choisissez Modifier.
3. Pour Expression de sélection de modèle, saisissez **`\$default`**.
4. Choisissez Modifier.
5. Dans la section Modèles de requête, choisissez Créer un modèle.
6. Pour Clé de modèle, saisissez **`\$default`**.
7. Pour Générer un modèle, saisissez 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"
}
```

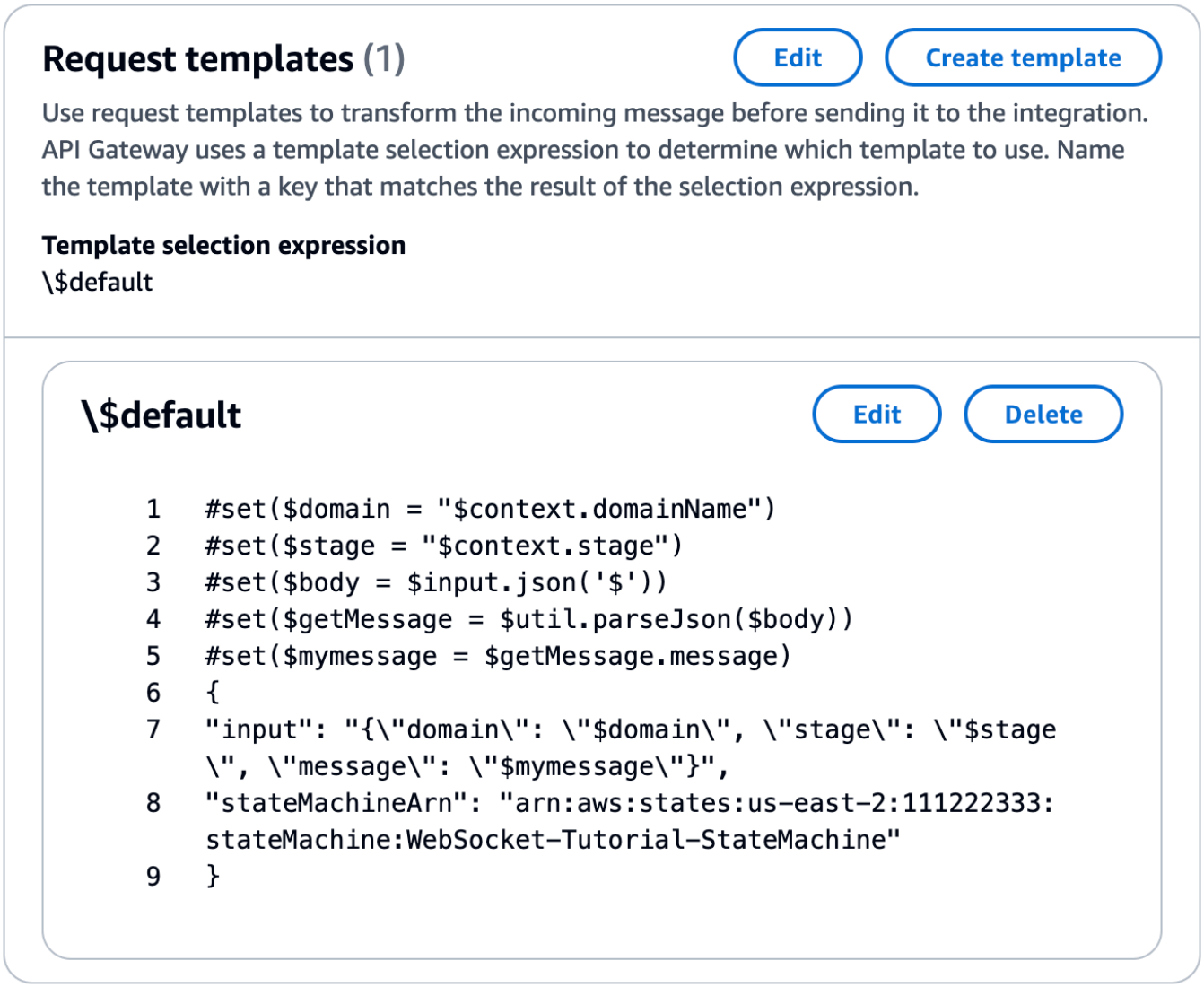
Remplacez le *stateMachineArn* par l'ARN de la machine à états créée par CloudFormation.

Le modèle de mappage :

- Crée la variable `$domain` à l'aide de la variable contextuelle `domainName`.
- Crée la variable `$stage` à l'aide de la variable contextuelle `stage`.

  Les variables `$domain` et `$stage` sont nécessaires pour créer une URL de rappel.
- Prend le message JSON `sendmessage` entrant et en extrait la propriété `message`.
- Crée l'entrée pour la machine d'état. 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 la route $connect ou $disconnect, pour ajouter ou supprimer directement un ID de connexion de la table DynamoDB, sans invoquer de fonction Lambda.

Étape 6 : tester votre API

Vous allez ensuite déployer et tester votre API pour vérifier son bon fonctionnement. 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, sélectionnez Routes.
3. Sélectionnez Deploy API (Déployer une API).
4. Pour Étape, sélectionnez production.
5. (Facultatif) Pour Description du déploiement, saisissez une description.
6. Choisissez Déployer.

Une fois que vous avez déployé votre API, vous pouvez l'invoquer. Utilisez l'URL d'invocation pour appeler votre API.

Pour obtenir l'URL d'invocation 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 allez tester la connexion en invoquant le chemin `/ping`.

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

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

2. Saisissez la commande suivante pour effectuer un ping du cadre de contrôle. Vous pouvez utiliser un cadre de contrôle pour le keepalive côté client.

```
/ping
```

Le résultat doit avoir l'aspect suivant :

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

Maintenant que vous avez testé la connexion, vous pouvez tester le bon fonctionnement de votre API. 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 requête de routage 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 invoque 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 didacticiel. Les étapes suivantes suppriment votre 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 API, sélectionnez votre websocket-api.
3. Choisissez Actions, choisissez Supprimer, puis confirmez votre choix.

Pour supprimer un CloudFormation pile

1. Ouvrez la CloudFormation console à l'adresse https://console.aws.amazon.com/cloudformation.
2. Sélectionnez votre 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 CloudFormation modèle qui automatise ces actions pour ce didacticiel, consultez le fichier ws-sfn.zip.

API REST API Gateway

Une API REST dans API Gateway est un ensemble de ressources et de méthodes qui sont intégrées avec les points de terminaison HTTP backend, les fonctions Lambda ou 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'API REST dans API Gateway
- Publiez REST APIs pour que les clients puissent l'invoquer
- Optimisez les performances de REST APIs
- Distribuez votre REST APIs aux clients dans API Gateway
- Protection de vos API REST dans API Gateway
- Surveillance des API REST dans API Gateway

Développement d'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 entité `Resource` peut disposer d'une ou de plusieurs ressources Method. Une `Method` est une demande entrante soumise par le client, qui peut contenir les paramètres de demande suivants : un paramètre de chemin, un en-tête ou un paramètre de chaîne de requête. En outre, selon la méthode HTTP, la demande peut contenir un corps. Votre méthode définit la manière dont le client accède à la `Resource` exposée. Pour intégrer la `Method` avec un point de terminaison de backend, également appelé point de terminaison d'intégration, vous allez créer une ressource Integration. Cette opération 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 afin de répondre aux exigences du backend, ou vous pouvez créer une intégration de proxy, dans laquelle API

Gateway envoie la demande complète dans un format normalisé à l'URI du point de terminaison d'intégration, puis envoie directement la réponse au client.

Pour les réponses, vous pouvez créer une MethodResponseressource pour représenter une réponse reçue par le client et vous créer une IntegrationResponseressource pour représenter la réponse renvoyée par le backend. Utilisez une 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.

Exemple de ressource pour une API REST

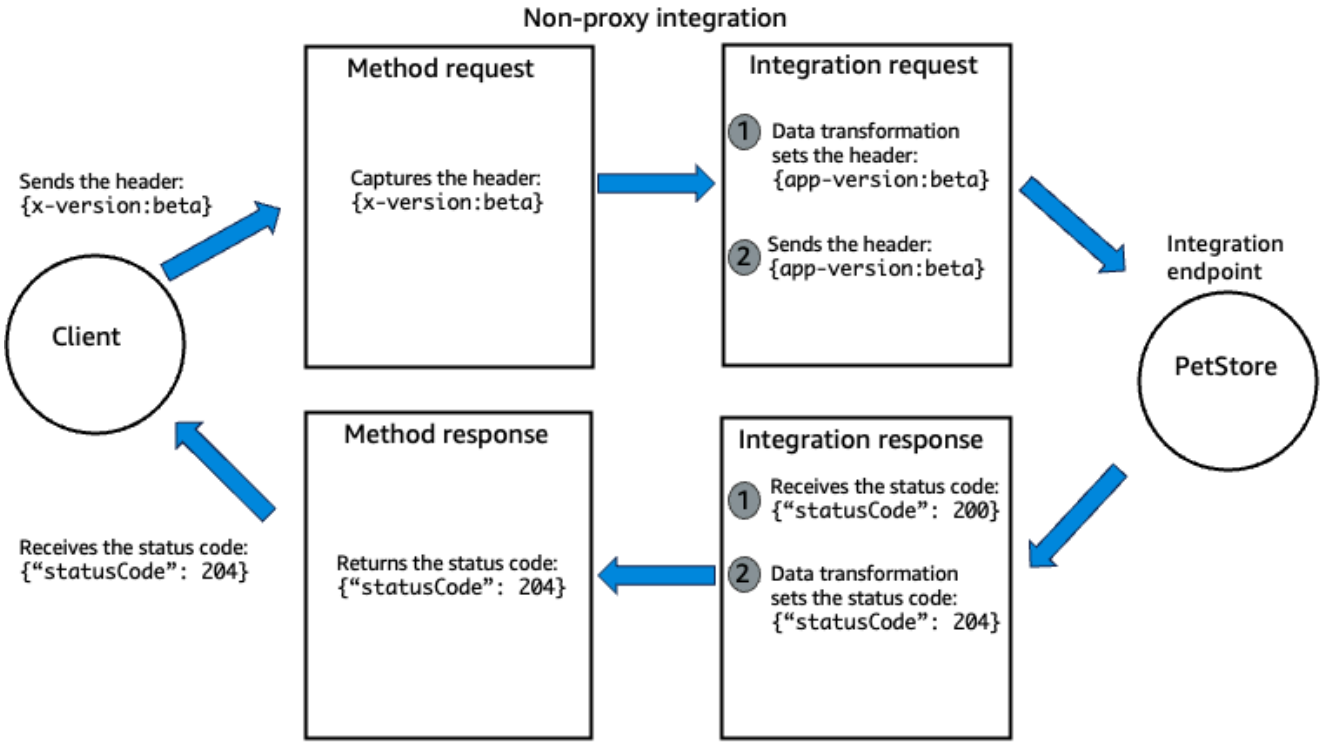
Le schéma suivant montre comment API Gateway implémente ce request/response modèle pour un proxy HTTP et une intégration HTTP sans proxy pour la `GET /pets` ressource. Le client envoie l'en-tête `x-version:beta` à API Gateway, qui envoie le code d'état `204` au client.

Dans une intégration sans proxy, API Gateway transforme les données pour répondre aux exigences du backend en modifiant la demande d'intégration et la réponse d'intégration. Dans une intégration sans proxy, vous accédez au corps dans la demande de méthode, mais vous le transformez dans la demande d'intégration. Lorsque le point de terminaison d'intégration renvoie une réponse avec un corps, vous y accédez et vous le transformez dans la réponse d'intégration. Vous ne pouvez pas modifier le corps dans la réponse de méthode.

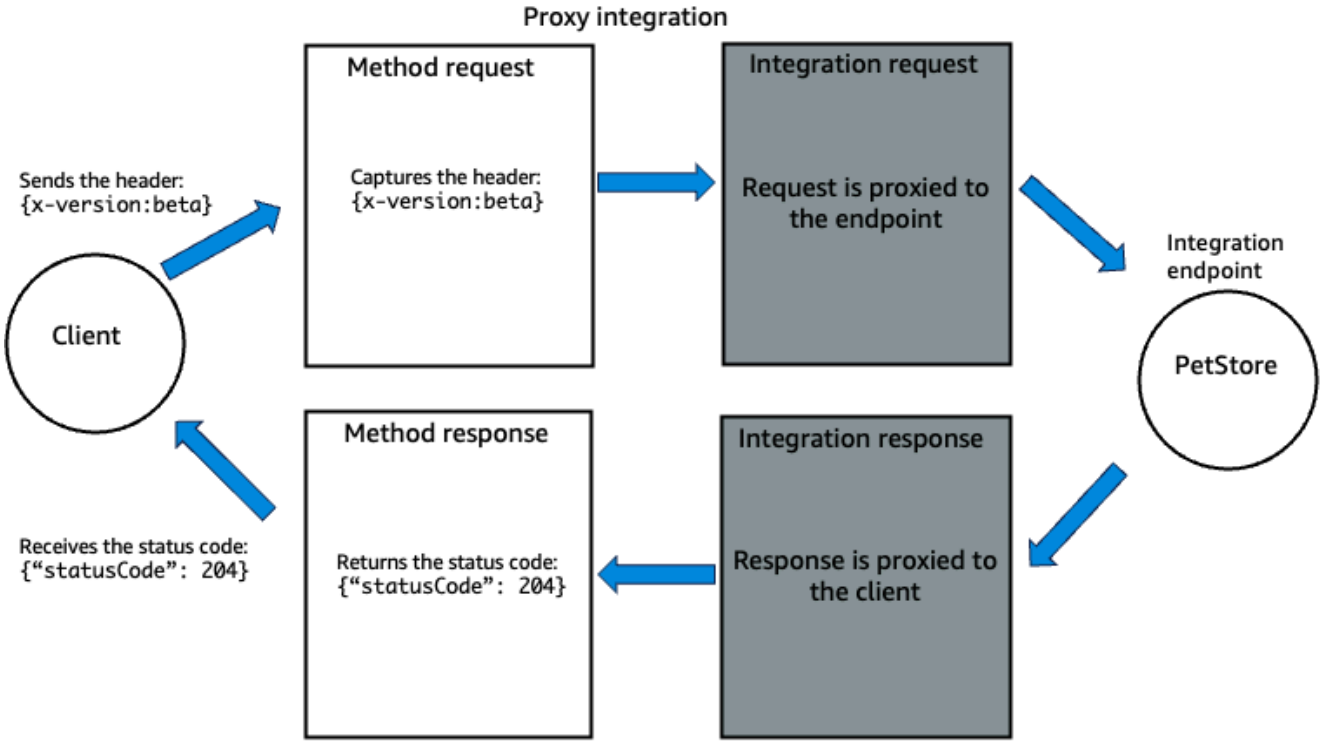
Dans une intégration de proxy, le point de terminaison d'intégration modifie la demande et la réponse. API Gateway ne modifie ni la demande d'intégration ni la réponse d'intégration, et envoie la demande entrante au backend telle quelle.

Quel que soit le type d'intégration, le client envoie une demande à API Gateway, qui y répond de manière synchrone.

Non-proxy integration



Proxy integration



Les exemples de journaux d'exécution suivants montrent ce qu'API Gateway enregistrerait dans l'exemple précédent. Pour plus de clarté, certaines valeurs et certains journaux initiaux ont été supprimés :

Non-proxy integration

```
Wed Feb 12 23:56:44 UTC 2025 : Starting execution for request: abcd-1234-5678
Wed Feb 12 23:56:44 UTC 2025 : HTTP Method: GET, Resource Path: /pets
Wed Feb 12 23:56:44 UTC 2025 : Method request path: {}
Wed Feb 12 23:56:44 UTC 2025 : Method request query string: {}
Wed Feb 12 23:56:44 UTC 2025 : Method request headers: {x-version=beta}
Wed Feb 12 23:56:44 UTC 2025 : Method request body before transformations:
Wed Feb 12 23:56:44 UTC 2025 : Endpoint request URI: http://petstore-demo-
endpoint.execute-api.com/petstore/pets
Wed Feb 12 23:56:44 UTC 2025 : Endpoint request headers: {app-version=beta}
Wed Feb 12 23:56:44 UTC 2025 : Endpoint request body after transformations:
Wed Feb 12 23:56:44 UTC 2025 : Sending request to http://petstore-demo-
endpoint.execute-api.com/petstore/pets
Wed Feb 12 23:56:45 UTC 2025 : Received response. Status: 200, Integration latency:
 123 ms
Wed Feb 12 23:56:45 UTC 2025 : Endpoint response headers: {Date=Wed, 12 Feb 2025
 23:56:45 GMT}
Wed Feb 12 23:56:45 UTC 2025 : Endpoint response body before transformations:
Wed Feb 12 23:56:45 UTC 2025 : Method response body after transformations: (null)
Wed Feb 12 23:56:45 UTC 2025 : Method response headers: {X-Amzn-Trace-Id=Root=1-
abcd-12345}
Wed Feb 12 23:56:45 UTC 2025 : Successfully completed execution
Wed Feb 12 23:56:45 UTC 2025 : Method completed with status: 204
```

Proxy integration

```
Wed Feb 12 23:59:42 UTC 2025 : Starting execution for request: abcd-1234-5678
Wed Feb 12 23:59:42 UTC 2025 : HTTP Method: GET, Resource Path: /pets
Wed Feb 12 23:59:42 UTC 2025 : Method request path: {}
Wed Feb 12 23:59:42 UTC 2025 : Method request query string: {}
Wed Feb 12 23:59:42 UTC 2025 : Method request headers: {x-version=beta}
Wed Feb 12 23:59:42 UTC 2025 : Method request body before transformations:
Wed Feb 12 23:59:42 UTC 2025 : Endpoint request URI: http://petstore-demo-
endpoint.execute-api.com/petstore/pets
Wed Feb 12 23:59:42 UTC 2025 : Endpoint request headers: { x-version=beta}
Wed Feb 12 23:59:42 UTC 2025 : Endpoint request body after transformations:
Wed Feb 12 23:59:42 UTC 2025 : Sending request to http://petstore-demo-
endpoint.execute-api.com/petstore/pets
```

```
Wed Feb 12 23:59:43 UTC 2025 : Received response. Status: 204, Integration latency:
 123 ms
Wed Feb 12 23:59:43 UTC 2025 : Endpoint response headers: {Date=Wed, 12 Feb 2025
 23:59:43 GMT}
Wed Feb 12 23:59:43 UTC 2025 : Endpoint response body before transformations:
Wed Feb 12 23:59:43 UTC 2025 : Method response body after transformations:
Wed Feb 12 23:59:43 UTC 2025 : Method response headers: {Date=Wed, 12 Feb 2025
 23:59:43 GMT}
Wed Feb 12 23:59:43 UTC 2025 : Successfully completed execution
Wed Feb 12 23:59:43 UTC 2025 : Method completed with status: 204
```

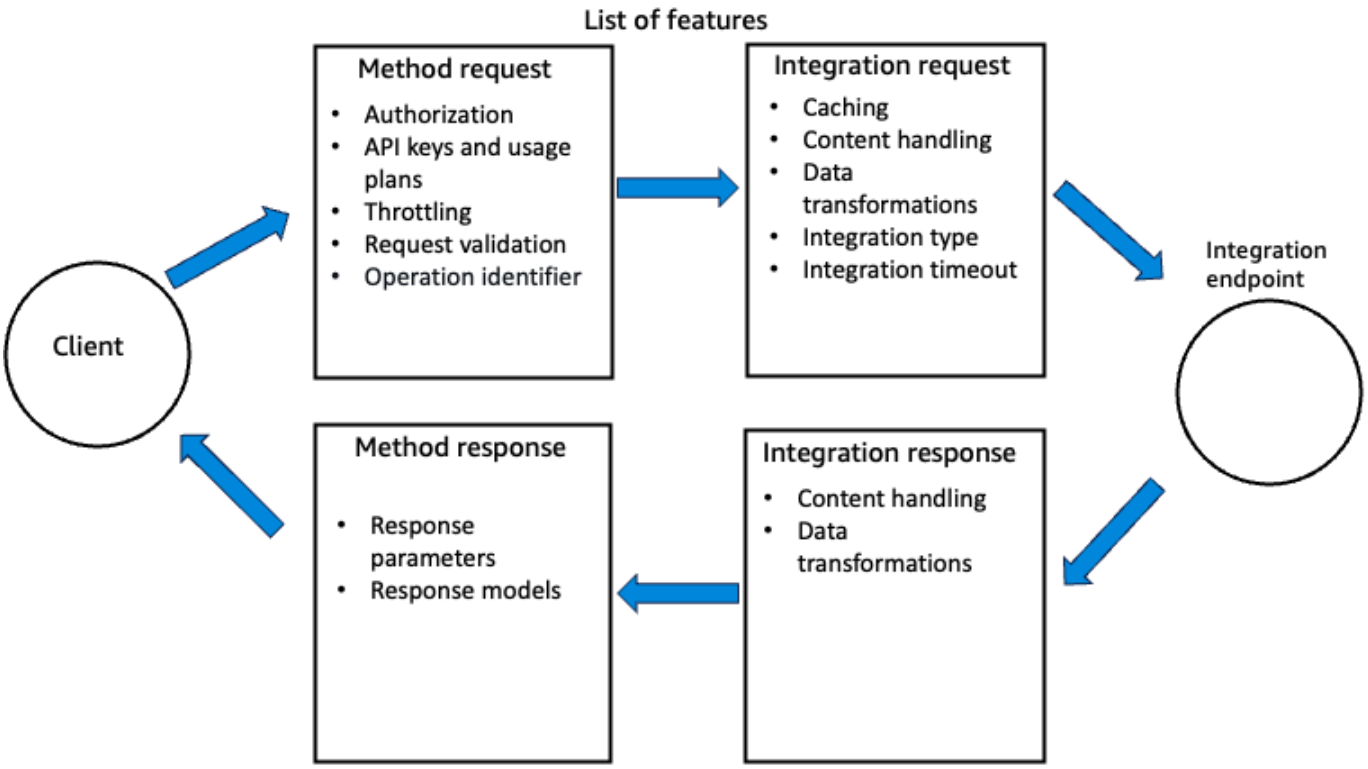
Pour importer une API similaire et la tester dans le Console de gestion AWS, consultez l'exemple d'API.

Autres fonctionnalités pour le développement des API REST

API Gateway prend en charge d'autres fonctionnalités pour le développement de votre API REST. Par exemple, pour aider vos clients à comprendre votre API, vous pouvez fournir une documentation sur l'API. Pour ce faire, ajoutez une ressource DocumentationPart 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 des plans d'utilisation pour limiter les demandes d'API. Vous pouvez les activer lors de la création ou de la mise à jour de l'API.

Le schéma suivant montre les fonctionnalités disponibles pour le développement d'API REST et indique où ces fonctionnalités sont configurées dans le request/response modèle.



Pour une introduction à la création d'une API, consultez the section called "Didacticiel : création d'une API REST avec une 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 pour REST APIs dans API Gateway
- Politiques de sécurité pour les API REST dans API Gateway
- Types d'adresses IP pour les API REST dans API Gateway
- Méthodes pour REST APIs dans API Gateway
- Contrôler et gérer l'accès à REST APIs dans API Gateway
- Intégrations pour REST APIs dans API Gateway
- Validation des demandes pour les API REST dans API Gateway
- Transformations de données pour les API REST dans API Gateway
- Réponses de passerelle pour REST APIs dans API Gateway

- Partage des ressources entre origines multiples pour les API REST dans API Gateway
- Types de supports binaires pour les API REST dans API Gateway
- Invocation d'une API REST dans API Gateway
- Développement d'API REST à l'aide d'OpenAPI dans API Gateway

Types de points de terminaison d'API pour REST APIs dans 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 la périphérie, régional ou privé, selon la provenance de la majorité du trafic de votre API.

Points de terminaison d'API optimisés pour la périphérie

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 API Gateway REST APIs.

Optimisé pour les bords, mettez en APIs majuscule 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 la périphérie s'applique à toutes les régions.

Points de terminaison d'API régional

Un point de terminaison d'API régional est destiné aux clients situés dans la même région. Lorsqu'un client s'exécutant sur une instance EC2 appelle une API de la même région, ou lorsqu'une API est destinée à servir un petit nombre de clients aux demandes très exigeantes, une API régionale réduit la charge de connexion.

Pour une API régionale, le nom de domaine personnalisé que vous utilisez est spécifique à la région de déploiement de l'API. Si vous déployez une API régionale dans plusieurs régions, elle peut 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 “Configuration d’un nom de domaine personnalisé optimisé pour la périphérie dans API Gateway”.

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 « optimisés pour la périphérie » à « régionaux » ou « privés ».
- De « régionaux » à « optimisés pour la périphérie » ou « privés ».
- De « privés » à « régionaux »

Vous ne pouvez pas modifier une API privée en une API optimisée pour la périphérie.

Si vous modifiez une API publique « optimisée pour la périphérie » en API publique « régionale » et inversement, notez qu'une API optimisée pour la périphérie peut présenter des comportements différents de ceux d'une API régionale. Par exemple, une API optimisée pour la périphérie supprime l'en-tête `Content-MD5`. Toute valeur de MD5 hachage transmise au backend peut être exprimée dans un paramètre de chaîne de requête ou dans une propriété de corps. Toutefois, l'API régionale transmet cet en-tête, même si cela peut remplacer le nom de l'en-tête par un autre nom. Comprendre les différences vous aide à décider comment mettre à jour une API optimisée pour la périphérie en API régionale ou inversement.

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 un 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 politique de ressources de votre API afin de supprimer toute mention des points de terminaison VPC afin que les appels d'API provenant de l'extérieur VPCs ou 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 politique de ressources de votre API.
9. Redéployez votre API afin que les modifications prennent effet.

   Comme vous migrez le type de point de terminaison de privé vers régional, API Gateway remplace le type d'adresse IP par. IPv4 Pour de plus amples informations, veuillez consulter the section called "Types d'adresses IP pour les API REST dans API Gateway".

Pour convertir un point de terminaison régional en un 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 votre point de terminaison de VPC. Pour de plus amples informations, veuillez consulter ???.
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 le point de terminaison VPC IDs, sélectionnez le point de terminaison VPC IDs que vous souhaitez associer à votre API privée.
8. Sélectionnez Enregistrer les modifications.
9. Redéployez votre API afin que les modifications prennent effet.

   Étant donné que vous migrez le type de point de terminaison d'une API régionale vers une API privée, API Gateway remplace le type d'adresse IP par Double pile. Pour de plus amples informations, veuillez consulter the section called "Types d'adresses IP pour les API REST dans API Gateway".

Utilisez le AWS CLI pour modifier le type de point de terminaison d'une API

La update-rest-apicommande suivante met à jour une API optimisée pour les périphériques vers une API régionale :

```
aws apigateway update-rest-api \
    --rest-api-id a1b2c3 \
    --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": "a1b2c3",
    "name": "PetStore imported as edge-optimized"
}
```

La update-rest-apicommande suivante met à jour une API régionale vers une API optimisée pour les périphériques :

```
aws apigateway update-rest-api \
    --rest-api-id a1b2c3 \
    --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.

Politiques de sécurité pour les API REST dans API Gateway

Une politique de sécurité est une combinaison prédéfinie d’une version minimale du protocole TLS et de suites de chiffrement offertes par API Gateway. Lorsque vos clients établissent une liaison TLS vers votre API ou nom de domaine personnalisé, la politique de sécurité applique la version TLS et la suite de chiffrement acceptées par API Gateway. Les politiques de sécurité protègent vos API et vos noms de domaine personnalisés contre les problèmes de sécurité réseau tels que la falsification et l’écoute clandestine entre un client et un serveur.

API Gateway prend en charge les politiques de sécurité existantes et les politiques de sécurité améliorées. `TLS_1_0`et `TLS_1_2` sont des politiques de sécurité héritées du passé. Utilisez ces politiques de sécurité pour des raisons de rétrocompatibilité. Toute politique commençant par `SecurityPolicy_` est une politique de sécurité renforcée. Utilisez-les pour les charges de travail régulées, la gouvernance avancée ou pour utiliser la cryptographie post-quantique. Lorsque vous utilisez une politique de sécurité améliorée, vous devez également définir le mode d'accès aux points de terminaison pour une meilleure gouvernance. Pour plus d'informations, consultez the section called "Mode d'accès au point de terminaison".

Comment API Gateway applique les politiques de sécurité

L'exemple suivant montre comment API Gateway applique les politiques de sécurité en utilisant la politique de `SecurityPolicy_TLS13_1_3_2025_09` sécurité comme exemple.

La politique `SecurityPolicy_TLS13_1_3_2025_09` de sécurité accepte le trafic TLS 1.3 et rejette le trafic TLS 1.2 et TLS 1.0. Pour le trafic TLS 1.3, la politique de sécurité accepte les suites de chiffrement suivantes :

- `TLS_AES_128_GCM_SHA256`
- `TLS_AES_256_GCM_SHA384`
- `TLS_CHACHA20_POLY1305_SHA256`

API Gateway n'accepte aucune autre suite de chiffrement. Par exemple, la politique de sécurité rejetterait tout trafic TLS 1.3 utilisant la suite de `AES128-SHA` chiffrement. Pour plus d'informations sur les versions et chiffrements TLS pris en charge, consultez. the section called "Politiques de sécurité prises en charge"

Pour contrôler le protocole TLS et les chiffrements que les clients ont utilisés pour accéder à votre API Gateway, vous pouvez utiliser les variables de `$context.cipherSuite` contexte `$context.tlsVersion` et de vos journaux d'accès. Pour plus d'informations, consultez the section called "Surveillance".

Mode d'accès au point de terminaison

Le mode d'accès au point de terminaison est un paramètre supplémentaire que vous devez spécifier pour toute API REST ou tout nom de domaine personnalisé utilisant une politique de sécurité améliorée commençant par `SecurityPolicy_`. Vous le faites lorsque vous créez votre ressource ou si vous modifiez la politique de sécurité d'une politique héritée à une politique améliorée.

Lorsque le mode d'accès aux terminaux est défini surSTRICT, toutes les demandes adressées à votre API REST ou à votre nom de domaine personnalisé doivent passer les contrôles suivants :

- La demande doit provenir du même type de point de terminaison API Gateway que votre ressource. Il peut s'agir d'un point de terminaison régional, d'un point de terminaison optimisé pour la périphérie ou d'un point de terminaison privé.
- Si vous utilisez un point de terminaison régional ou privé, API Gateway utilise la correspondance d'hôtes SNI. Si vous utilisez un point de terminaison optimisé pour la périphérie, API Gateway est conforme à la protection frontale CloudFront de domaine de la plateforme. Pour plus d'informations, consultez la section Domain Fronting.

Si l'une de ces conditions n'est pas remplie, API Gateway rejette la demande. Nous vous recommandons d'utiliser le mode d'accès aux STRICT terminaux lorsque cela est possible.

Pour migrer une API ou un nom de domaine existant afin d'utiliser le mode d'accès strict aux terminaux, mettez d'abord à jour votre politique de sécurité pour adopter une politique de sécurité renforcée et maintenez le mode d'accès aux terminaux défini surBASIC. Après avoir validé vos journaux de trafic et d'accès, définissez le mode d'accès des terminaux surSTRICT. Lorsque vous migrez le mode d'accès au terminal de STRICT àBASIC, votre terminal sera indisponible pendant environ 15 minutes à mesure que les modifications se propagent.

Vous ne devez pas définir le mode d'accès aux terminaux STRICT pour certaines architectures d'applications, mais plutôt définir le mode d'accès aux terminaux surBASIC. Le tableau suivant présente certaines architectures d'applications et propose une recommandation pour que votre API REST ou votre nom de domaine personnalisé puisse utiliser le mode d'accès aux STRICT terminaux.

Architecture	Migration suggérée
Utilisation d'un point de terminaison VPC pour accéder à un nom de domaine personnalisé public.	Cette architecture utilise un trafic de type inter-terminaux. Nous vous recommandons de migrer versthe section called "Noms de domaine personnalisés pour les API privées".
Utiliser n'importe quelle méthode pour invoquer une API privée qui n'utilise pas de nom de domaine personnalisé ni de noms DNS privés.	Cette architecture crée une incompatibilité entre l'en-tête de l'hôte et le SNI utilisé lors de la prise de contact TLS et ne passe pas les restrictions de front CloudFront de domaine.

Architecture	Migration suggérée
	Nous vous recommandons de migrer votre VPC pour utiliser un DNS privé.
Utiliser le partage de domaine pour distribuer du contenu sur plusieurs domaines ou sous-domaines.	Cette architecture crée une incompatibilité entre l'en-tête de l'hôte et le SNI utilisé lors de la prise de contact TLS et ne passe pas les restrictions de front CloudFront de domaine. Nous vous recommandons d'utiliser cet anti-modèle `HTTP/2` et de vous en éloigner.

Les considérations suivantes concernent l'utilisation du mode d'accès aux terminaux :

- Si le mode d'accès au point de terminaison d'une API ou d'un nom de domaine l'est`STRICT`, vous ne pouvez pas modifier le type de point de terminaison. Pour modifier le type de point de terminaison, modifiez d'abord le mode d'accès au point de terminaison sur`BASIC`.
- Une fois que vous avez modifié le mode d'accès aux terminaux de `BASIC` à`STRICT`, API Gateway applique un délai de 15 minutes pour appliquer le mode d'accès strict aux terminaux.
- Lorsque vous modifiez une politique de sécurité, passant d'une politique commençant par `SecurityPolicy_` à une politique héritée, vous devez désactiver le mode d'accès aux terminaux sur`""`.

Considérations

Les considérations suivantes concernent les politiques de sécurité pour les API REST dans API Gateway :

- Vous pouvez importer la politique de sécurité dans un fichier de définition OpenAPI. Pour plus d’informations, consultez Politique de sécurité de x-amazon-apigateway-.
- Votre API peut être mappée à un nom de domaine personnalisé avec une politique de sécurité différente de celle de votre API. Lorsque vous invoquez ce nom de domaine personnalisé, API Gateway utilise la politique de sécurité du domaine personnalisé pour négocier la prise de contact TLS. La désactivation du point de terminaison de votre API par défaut peut avoir une incidence sur la manière dont les appelants peuvent invoquer votre API.

- Si vous modifiez votre politique de sécurité, la mise à jour prend environ 15 minutes. Vous pouvez surveiller l'état `apiStatus` de votre API. Au fur et à mesure que votre API sera mise à jour, elle le sera `UPDATING` et quand elle sera terminée`AVAILABLE`. `apiStatus` Lorsque le statut de votre API est `UPDATING` défini, vous pouvez toujours l'invoquer.
- API Gateway prend en charge les politiques de sécurité sur toutes les API. Toutefois, vous ne pouvez choisir une politique de sécurité que pour les API REST. API Gateway prend uniquement en charge la politique de `TLS_1_2` sécurité pour HTTP ou WebSocket les API.
- Vous ne pouvez pas mettre à jour la politique de sécurité d'une API de `TLS_1_0` à`TLS_1_2`.
- Certaines politiques de sécurité prennent en charge les suites de chiffrement ECDSA et RSA. Si vous utilisez ce type de politique avec un nom de domaine personnalisé, les suites de chiffrement correspondent au type de clé de certificat fourni par le client, RSA ou ECDSA. Si vous utilisez ce type de politique avec une API REST, les suites de chiffrement correspondent aux suites de chiffrement compatibles avec les types de certificats RSA.

Politiques de sécurité prises en charge

Les tableaux suivants décrivent les politiques de sécurité qui peuvent être spécifiées pour chaque type de point de terminaison de l'API REST et chaque type de nom de domaine personnalisé. Ces politiques vous permettent de contrôler les connexions entrantes. API Gateway prend uniquement en charge le protocole TLS 1.2 en sortie. Vous pouvez mettre à jour la politique de sécurité de votre API ou de votre nom de domaine personnalisé à tout moment.

Les politiques contenues `FIPS` dans le titre sont compatibles avec la norme FIPS (Federal Information Processing Standard), une norme gouvernementale américaine et canadienne qui spécifie les exigences de sécurité pour les modules cryptographiques qui protègent les informations sensibles. Pour en savoir plus, consultez la norme FIPS (Federal Information Processing Standard) 140 sur la page Conformité à la sécurité du AWS cloud.

Toutes les politiques FIPS s'appuient sur le module cryptographique validé AWS-LC FIPS. Pour en savoir plus, consultez la page du module AWS-LC cryptographique sur le site du programme de validation des modules cryptographiques du NIST.

Les politiques contenues `PQ` dans le titre utilisent la Post-Quantum cryptographie (PQC) pour implémenter des algorithmes d'échange de clés hybrides pour TLS afin de garantir la confidentialité du trafic contre les futures menaces informatiques quantiques.

Les politiques contenues `PFS` dans le titre utilisent Perfect Forward Secrecy (PFS) pour s'assurer que les clés de session ne sont pas compromises.

Les politiques qui contiennent les deux `FIPS` et `PQ` dans leur titre prennent en charge ces deux fonctionnalités.

Politiques de sécurité par défaut

Lorsque vous créez une nouvelle API REST ou un domaine personnalisé, une politique de sécurité par défaut est affectée à la ressource. Le tableau suivant présente la politique de sécurité par défaut pour ces ressources.

Ressource	Nom de la politique de sécurité par défaut		
API régionales	TLS_1_0		
Edge-optimized API	TLS_1_0		
API privées	TLS_1_2		
Domaine régional	TLS_1_2		
Edge-optimized domaine	TLS_1_2		
Domaine privé	TLS_1_2		

Politiques de sécurité prises en charge pour les API régionales et privées et les noms de domaine personnalisés

Le tableau suivant décrit les politiques de sécurité qui peuvent être spécifiées pour les API régionales et privées et les noms de domaine personnalisés :

Politique de sécurité	Versions TLS prises en charge	Chiffrements pris en charge
SecurityPolicy_TLS13_1_3_2025_09	TLS1.3	TLS1.3 • TLS_AES_128_GCM_SHA256 • TLS_AES_256_GCM_SHA384

Politique de sécurité	Versions TLS prises en charge	Chiffrements pris en charge
		• TLS_CHACHA20_POLY1305_SHA256
SecurityPolicy_TLS13_1_3_FIPS_2025_09	TLS1.3	TLS1.3 • TLS_AES_128_GCM_SHA256 • TLS_AES_256_GCM_SHA384
SecurityPolicy_TLS13_1_2_FIPS_PFS_PQ_2025_09	TLS1.3 TLS1.2	TLS1.3 • TLS_AES_128_GCM_SHA256 • TLS_AES_256_GCM_SHA384 TLS1.2 • ECDHE-ECDSA-AES128-GCM-SHA256 • ECDHE-RSA-AES128-GCM-SHA256 • ECDHE-ECDSA-AES256-GCM-SHA384 • ECDHE-RSA-AES256-GCM-SHA384

Politique de sécurité	Versions TLS prises en charge	Chiffrements pris en charge
SecurityPolicy_TLS13_1_2_PFS_PQ_2025_09	TLS1.3 TLS1.2	TLS1.3 • TLS_AES_128_GCM_SHA256 • TLS_AES_256_GCM_SHA384 • TLS_CHACHA20_POLY1305_SHA256 TLS1.2 • ECDHE-ECDSA-AES128-GCM-SHA256 • ECDHE-RSA-AES128-GCM-SHA256 • ECDHE-ECDSA-AES256-GCM-SHA384 • ECDHE-RSA-AES256-GCM-SHA384

Politique de sécurité	Versions TLS prises en charge	Chiffrements pris en charge
SecurityPolicy_TLS 13_1_2_PQ_2025_09	TLS1.3 TLS1.2	TLS1.3 • TLS_AES_128_GCM_SHA256 • TLS_AES_256_GCM_SHA384 • TLS_CHACHA20_POLY1305_SHA256 TLS1.2 • 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

Politique de sécurité	Versions TLS prises en charge	Chiffrements pris en charge
SecurityPolicy_TLS13_1_2_2021_06	TLS1.3 TLS1.2	TLS1.3 • TLS_AES_128_GCM_SHA256 • TLS_AES_256_GCM_SHA384 • TLS_CHACHA20_POLY1305_SHA256 TLS1.2 • 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

Politique de sécurité	Versions TLS prises en charge	Chiffrements pris en charge
TLS_1_2	TLS1.3 TLS1.2	TLS1.3 • TLS_AES_128_GCM_SHA256 • TLS_AES_256_GCM_SHA384 • TLS_CHACHA20_POLY1305_SHA256 TLS1.2 • 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

Politique de sécurité	Versions TLS prises en charge	Chiffrements pris en charge
TLS_1_0	TLS1.3 TLS1.2 TLS1.1 TLS1.0	TLS1.3 • TLS_AES_128_GCM_SHA256 • TLS_AES_256_GCM_SHA384 • TLS_CHACHA20_POLY1305_SHA256 TLS1.0-TLS1.2 • 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-ECDSA-AES256-SHA

Politique de sécurité	Versions TLS prises en charge	Chiffrements pris en charge
		• ECDHE-RSA-AES256-SHA • AES128-GCM-SHA256 • AES128-SHA256 • AES128-SHA • AES256-GCM-SHA384 • AES256-SHA256 • AES256-SHA

Politiques de sécurité prises en charge pour les API optimisées pour les périphériques et les noms de domaine personnalisés

Le tableau suivant décrit les politiques de sécurité qui peuvent être spécifiées pour les API optimisées pour la périphérie et les noms de domaine personnalisés optimisés pour la périphérie :

Nom de la politique de sécurité	Versions TLS prises en charge	Chiffrements pris en charge
SecurityPolicy_TLS13_2025_EDGE	TLS1.3	TLS1.3 • TLS_AES_128_GCM_SHA256 • TLS_AES_256_GCM_SHA384 • TLS_CHACHA20_POLY1305_SHA256
SecurityPolicy_TLS12_PFS_2025_EDGE	TLS1.3 TLS1.2	TLS1.3 • TLS_AES_128_GCM_SHA256 • TLS_AES_256_GCM_SHA384 • TLS_CHACHA20_POLY1305_SHA256

Nom de la politique de sécurité	Versions TLS prises en charge	Chiffrements pris en charge
		TLS1.2 • ECDHE-ECDSA-AES128-GCM-SHA256 • ECDHE-ECDSA-AES256-GCM-SHA384 • ECDHE-ECDSA-CHACHA20-POLY1305 • ECDHE-RSA-AES128-GCM-SHA256 • ECDHE-RSA-AES256-GCM-SHA384 • ECDHE-RSA-CHACHA20-POLY1305

Nom de la politique de sécurité	Versions TLS prises en charge	Chiffrements pris en charge
SecurityPolicy_TLS12_2018_EDGE	TLS1.3 TLS1.2	TLS1.3 • TLS_AES_128_GCM_SHA256 • TLS_AES_256_GCM_SHA384 • TLS_CHACHA20_POLY1305_SHA256 TLS1.2 • ECDHE-ECDSA-AES128-GCM-SHA256 • ECDHE-ECDSA-AES128-SHA256 • ECDHE-ECDSA-AES256-GCM-SHA384 • ECDHE-ECDSA-CHACHA20-POLY1305 • ECDHE-ECDSA-AES256-SHA384 • ECDHE-RSA-AES128-GCM-SHA256 • ECDHE-RSA-AES128-SHA256 • ECDHE-RSA-AES256-GCM-SHA384 • ECDHE-RSA-CHACHA20-POLY1305 • ECDHE-RSA-AES256-SHA384

Nom de la politique de sécurité	Versions TLS prises en charge	Chiffrements pris en charge
TLS_1_2 (noms de domaine personnalisés optimisés pour les bords uniquement)	TLS1.3 TLS1.2	TLS1.3 • TLS_AES_128_GCM_SHA256 • TLS_AES_256_GCM_SHA384 • TLS_CHACHA20_POLY1305_SHA256 TLS1.2 • ECDHE-ECDSA-AES128-GCM-SHA256 • ECDHE-ECDSA-AES128-SHA256 • ECDHE-ECDSA-AES256-GCM-SHA384 • ECDHE-ECDSA-CHACHA20-POLY1305 • ECDHE-ECDSA-AES256-SHA384 • ECDHE-RSA-AES128-GCM-SHA256 • ECDHE-RSA-AES128-SHA256 • ECDHE-RSA-AES256-GCM-SHA384 • ECDHE-RSA-CHACHA20-POLY1305 • ECDHE-RSA-AES256-SHA384

Nom de la politique de sécurité	Versions TLS prises en charge	Chiffrements pris en charge
		• AES128-GCM-SHA256 • AES256-GCM-SHA384 • AES128-SHA256

Nom de la politique de sécurité	Versions TLS prises en charge	Chiffrements pris en charge
TLS_1_0	TLS1.3 TLS1.2 TLS1.1 TLS1.0	TLS1.3 • TLS_AES_128_GCM_SHA256 • TLS_AES_256_GCM_SHA384 • TLS_CHACHA20_POLY1305_SHA256 TLS1.0-TLS1.2 • 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

Nom de la politique de sécurité	Versions TLS prises en charge	Chiffrements pris en charge
		• ECDHE-ECDSA-AES256-SHA • ECDHE-RSA-AES256-SHA • AES256-SHA256 • AES256-SHA

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. Pour plus d'informations, consultez ciphers dans la documentation OpenSSL.

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

Comment modifier une politique de sécurité

Vous pouvez modifier la politique de sécurité de votre API. Si vous envoyez du trafic vers vos API via votre nom de domaine personnalisé, il n'est pas nécessaire que l'API et le nom de domaine personnalisé aient la même politique de sécurité. Lorsque vous invoquez ce nom de domaine personnalisé, API Gateway utilise la politique de sécurité du domaine personnalisé pour négocier la prise de contact TLS. Toutefois, pour des raisons de cohérence, nous vous recommandons d'utiliser la même politique de sécurité pour votre nom de domaine personnalisé et votre API.

Si vous modifiez votre politique de sécurité, la mise à jour prend environ 15 minutes. Vous pouvez surveiller l'état `apiStatus` de votre API. Au fur et à mesure que votre API sera mise à jour, elle le sera `UPDATING` et quand elle sera terminée`AVAILABLE`. `apiStatus` Lorsque votre API est en cours de mise à jour, vous pouvez toujours l'invoquer.

Console de gestion AWS

Pour modifier la politique de sécurité d'une API

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, puis Modifier.
4. Pour Politique de sécurité, sélectionnez une nouvelle politique commençant par`SecurityPolicy_`.
5. Pour le mode d'accès Endpoint, choisissez Strict.
6. Cliquez sur Enregistrer les modifications.

   Redéployez l'API pour que les modifications prennent effet. Comme vous avez modifié le mode d'accès aux terminaux en mode strict, il faudra environ 15 minutes pour que les modifications se propagent complètement.

AWS CLI

La commande update-rest-api suivante met à jour une API pour utiliser la politique de sécurité : `SecurityPolicy_TLS13_1_3_2025_09`

```
aws apigateway update-rest-api \
    --rest-api-id abcd1234 \
    --patch-operations '[
        {
            "op": "replace",
            "path": "/securityPolicy",
            "value": "SecurityPolicy_TLS13_1_3_2025_09"
        },
        {
            "op": "replace",
            "path": "/endpointAccessMode",
            "value": "STRICT"
```

```
        }
    ]'
```

Le résultat se présente comme suit :

```
{
    "id": "abcd1234",
    "name": "MyAPI",
    "description": "My API with a new security policy",
    "createdDate": "2025-02-04T11:47:06-08:00",
    "apiKeySource": "HEADER",
    "endpointConfiguration": {
        "types": [
            "REGIONAL"
        ],
        "ipAddressType": "dualstack"
    },
    "tags": {},
    "disableExecuteApiEndpoint": false,
    "securityPolicy": "SecurityPolicy_TLS13_1_3_2025_09",
    "endpointAccessMode": "STRICT"
    "rootResourceId": "efg456"
}
```

La commande update-rest-api suivante met à jour une API qui utilisait une politique de sécurité renforcée pour utiliser la politique de sécurité. `TLS_1_0`

```
aws apigateway update-rest-api \
    --rest-api-id abcd1234 \
    --patch-operations '[
        {
            "op": "replace",
            "path": "/securityPolicy",
            "value": "TLS_1_0"
        },
        {
            "op": "replace",
            "path": "/endpointAccessMode",
            "value": ""
        }
    ]'
```

Le résultat se présente comme suit :

```
{
    "id": "abcd1234",
    "name": "MyAPI",
    "description": "My API with a new security policy",
    "createdDate": "2025-02-04T11:47:06-08:00",
    "apiKeySource": "HEADER",
    "endpointConfiguration": {
        "types": [
            "REGIONAL"
        ],
        "ipAddressType": "dualstack"
    },
    "tags": {},
    "disableExecuteApiEndpoint": false,
    "securityPolicy": "TLS_1_0",
    "rootResourceId": "efg456"
}
```

Types d’adresses IP pour les API REST dans API Gateway

Lorsque vous créez une API, vous spécifiez le type d’adresses IP qui peut invoquer votre API. Vous avez le choix entre IPv4 (pour autoriser les adresses IPv4 à invoquer votre API) et Dualstack (pour autoriser les adresses IPv4 et IPv6 à invoquer votre API). Nous vous recommandons de définir le type d’adresse IP sur Dualstack pour éviter l’épuisement de l’espace IP ou renforcer votre niveau de sécurité. Pour plus d’informations sur les avantages d’un type d’adresse IP à double pile, consultez IPv6 sur AWS.

Pour limiter votre API au trafic IPv6 uniquement, vous pouvez créer une stratégie de ressources qui limite les adresses IP sources aux plages IPv6 uniquement. Vous pouvez modifier le type d’adresse IP en mettant à jour la configuration de l’API. Cette modification prend effet immédiatement. Vous n’avez pas besoin de redéployer votre API. Pour plus d’informations, consultez the section called “Exemple : refuser le trafic API en fonction de l’adresse IP source ou de la plage”.

Considérations relatives aux types d’adresses IP

Les considérations suivantes peuvent avoir une incidence sur votre utilisation des types d’adresses IP :

- Le type d'adresse IP par défaut pour toutes les API régionales et optimisées pour la périphérie est IPv4.
- Les API privées ne peuvent avoir qu'un type d'adresse IP à double pile.
- Si vous remplacez le type d'adresse IP IPv4 d'une API existante par un type d'adresse IP à double pile, vérifiez que toutes les stratégies d'accès à vos API ont été mises à jour pour prendre en compte les appels IPv6. La modification du type d'adresse IP prend effet immédiatement.
- Si vous migrez le type de point de terminaison d'une API régionale ou optimisée pour la périphérie vers une API privée, API Gateway remplace le type d'adresse IP par une adresse à double pile. Pour plus d'informations, consultez the section called "Modifier un type de point de terminaison d'API publique ou privée".
- Si vous migrez le type de point de terminaison d'une API privée vers une API régionale, vous devez définir le type d'adresse IP sur Dualstack. Une fois la migration du point de terminaison terminée, vous pouvez remplacer le type d'adresse IP par IPv4. Pour plus d'informations, consultez the section called "Modifier un type de point de terminaison d'API publique ou privée".
- Votre API peut être mappée à un nom de domaine personnalisé présentant un type d'adresse IP différent de celui de votre API. La désactivation du point de terminaison de votre API par défaut peut avoir une incidence sur la manière dont les appelants peuvent invoquer votre API.
- Vous ne pouvez pas utiliser un fichier de définition externe pour configurer le type d'adresse IP de votre API.

Modification du type d'adresse IP d'une API REST

Vous pouvez modifier le type d'adresse IP en mettant à jour la configuration de l'API. Vous pouvez mettre à jour la configuration de l'API à l'aide de la Console de gestion AWS, de l'AWS CLI, d'CloudFormation ou d'un kit SDK AWS. Si vous modifiez le type d'adresse IP de l'API, vous n'avez pas besoin de redéployer votre API pour que les modifications prennent effet. Avant de modifier le type d'adresse IP, vérifiez que toutes les stratégies d'accès à vos API ont été mises à jour pour prendre en compte les appels IPv6.

Console de gestion AWS

Pour modifier le type d'adresse IP d'une API REST

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, puis Modifier.
4. Pour Type d'adresse IP, choisissez IPv4 ou Dualstack.
5. Sélectionnez Enregistrer les modifications.

   La modification de la configuration de votre API prend effet immédiatement.

AWS CLI

La commande update-rest-api suivante met à jour une API pour qu'elle possède le type d'adresse IP Dualstack :

```
aws apigateway update-rest-api \
    --rest-api-id abcd1234 \
    --patch-operations "op='replace',path='/endpointConfiguration/
ipAddressType',value='dualstack'"
```

Le résultat se présente comme suit :

```
{
    "id": "abcd1234",
    "name": "MyAPI",
    "description": "My API with a dualstack IP address type",
    "createdDate": "2025-02-04T11:47:06-08:00",
    "apiKeySource": "HEADER",
    "endpointConfiguration": {
        "types": [
            "REGIONAL"
        ],
        "ipAddressType": "dualstack"
    },
    "tags": {},
    "disableExecuteApiEndpoint": false,
    "rootResourceId": "efg456"
}
```

Méthodes pour REST APIs 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 renvoyées 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.

Conditions préalables

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 Didacticiel : création d'une API REST avec une intégration HTTP sans 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 Lambda autre que de proxy 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 les intégrations HTTP et les intégrations de proxy HTTP.

Rubriques

- Configuration d’une demande de méthode dans API Gateway
- Configuration d’une réponse 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 d’un 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, le *`{api-id}`* représente l'identifiant d'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.

La commande get-resources suivante récupère toutes les ressources d'une API :

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

Pour l' PetStore exemple d'API, le résultat est le suivant :

```
{
    "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, utilisez la commande suivante pour ajouter une ressource enfant :

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

Par exemple, pour ajouter de la nourriture pour animaux à vendre sur le PetStore site Web, utilisez la commande suivante :

```
aws apigateway create-resource --rest-api-id a1b2c3 \
    --parent-id svzr2028x8 \
```

```
    --path-part food
```

Le résultat se présente comme suit :

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

Utilisation d'une ressource 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 proxy API Gateway pour exposer un jeu de ressources d'API simultanément.

API Gateway définit une ressource proxy comme espace réservé afin qu'une ressource soit spécifiée lorsque la demande est envoyée. Une ressource 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/*`. Vous pouvez utiliser n'importe quelle chaîne pour le nom du paramètre Greedy Path.

La commande create-resource suivante crée une ressource proxy sous la racine (`/{proxy+}`) :

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

Le résultat se présente comme 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 proxy. Par exemple, les ressources proxy suivantes sont autorisées au sein d'une API, en supposant que `/parent/{proxy+}` n'est pas le même parent que `/parent/{child}/{proxy+}`.

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

Lorsqu'une ressource proxy dispose de sœurs autres que de proxy, les ressources sœurs sont exclues de la représentation de la ressource 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.

Le tableau suivant montre comment API Gateway achemine les demandes vers les ressources suivantes pour l'étape `prod` d'une API.

```
ANY /{proxy+}
GET /pets/{proxy+}
GET /pets/dog
```

Demande	Route sélectionnée	Explication
`GET https://`*`api-id`*`.execute-`	`GET /pets/dog`	La demande correspond entièrement à cette ressource.

Demande	Route sélectionnée	Explication
`api. region.amazonaws.com/prod/pets/dog`		
`GET https://api-id.execute-api. region.amazonaws.com/prod/pets/cats`	`GET /pets/{proxy+}`	La variable greedy path `/pets/{proxy+}` intercepte cette demande.
`GET https://api-id.execute-api. region.amazonaws.com/prod/animals`	`GET /{proxy+}`	La variable greedy path `/{proxy+}` intercepte cette demande.

Une ressource proxy ne peut pas disposer de ressource enfant. Toute ressource d'API située après `{proxy+}` est redondante et ambiguë. Les ressources 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 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 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 commande put-method suivante crée une demande de méthode pour le verbe `ANY` 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
```

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. La commande put-method suivante crée une méthode avec un paramètre de chemin obligatoire :

```
aws apigateway put-method --rest-api-id vaz7da96z6 \
    --resource-id rjkmth \
    --http-method GET \
    --authorization-type "NONE" \
    --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 requête `type` facultatif et un paramètre d'en-tête `age` facultatif, vous pouvez les déclarer à l'aide de la commande put-method suivante :

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

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.age":false}'
```

Grâce à cette configuration, le client peut rechercher les animaux par type :

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

De plus, le client peut rechercher les chiens qui sont des chiots comme suit :

```
GET /pets?type=dog
age:puppy
```

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 d'un 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 utiliser la commande put-method suivante :

```
aws apigateway put-method \
    --rest-api-id vaz7da96z6 \
    --resource-id 6sxz2j \
    --http-method POST \
    --authorization-type "NONE" \
    --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 politiques 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 :

JSON

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

Pour de plus amples informations, veuillez consulter 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 plus d'informations sur les autorisations entre comptes, consultez 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 plus d'informations sur la création d'un mécanisme d'autorisation Lambda, consultez 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 plus d'informations sur la création d'un mécanisme d'autorisation de groupe d'utilisateurs Amazon Cognito, consultez 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. La create-request-validatorcommande suivante crée un validateur de demande uniquement pour le corps.

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

Le résultat se présente comme suit :

```
{
    "validateRequestParameters": false,
    "validateRequestBody": true,
    "id": "jgpyy6",
    "name": "bodyOnlyValidator"
}
```

Vous pouvez utiliser ce validateur de demande pour utiliser la validation de demande dans le cadre de la configuration de la demande de méthode. La commande put-method suivante crée une demande de méthode qui exige que le corps de la demande entrante corresponde à `PetModel` et comprenne deux paramètres de demande facultatifs :

```
aws apigateway put-method \
    --rest-api-id 7zw9uyk9kl \
    --resource-id xdsvhp \
    --http-method PUT \
```

```
    --authorization-type "NONE" \
    --request-parameters '{"method.request.querystring.type": false,
"method.request.querystring.page":false}' \
    --request-models '{"application/json":"petModel"}' \
    --request-validator-id jgpyy6
```

Pour inclure un paramètre de demande dans la validation de la demande, vous devez définir `validateRequestParameters` sur `true` pour le validateur de demande, et le paramètre de demande spécifique sur `true` dans la commande `put-method`.

Configuration d'une réponse 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 put-method-responsecommande suivante crée une réponse de `200` méthode.

```
aws apigateway put-method-response \
    --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 put-method-responsecommande suivante définit l'`my-header`en-tête.

```
aws apigateway put-method-response \
        --rest-api-id vaz7da96z6 \
        --resource-id 6sxz2j \
        --http-method GET \
        --status-code 200  \
        --response-parameters method.response.header.my-header=false
```

Configuration des modèles de réponse de méthode

Un modèle de réponse de méthode définit un format pour le corps de la réponse de méthode. 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.

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`. La commande create-model suivante crée un modèle `PetStorePet` pour décrire le corps de la réponse à la demande de méthode `GET /pets/{petId}`.

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

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 put-method-responsecommande suivante crée une méthode de réponse qui utilise un modèle de réponse pour définir le format de charge utile :

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

Configuration d'une méthode à l'aide de la console API Gateway

Lorsque vous créez une méthode à l'aide de la console d'API REST, vous configurez à la fois la demande d'intégration et la demande de méthode. Par défaut, API Gateway crée la réponse de méthode `200` de votre méthode.

Les instructions suivantes expliquent comment modifier les paramètres de requête de méthode et comment créer des réponses de méthode supplémentaires pour votre méthode.

Rubriques

- Modification d'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

Modification d'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, consultez the section called " Configuration d'une demande d'intégration à l'aide de la console".

1. Dans le volet Ressources, choisissez votre méthode, puis l'onglet Demande de méthode.
2. Dans la section Paramètres de requête 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 politique 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 :

    JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "execute-api:Invoke"
      ],
      "Resource": [
        "arn:aws:execute-api:us-east-1:111111111111:aaabbb/*/GET/"
      ]
    }
```

```
  ]
}
```

Dans cette stratégie d'accès, *arn:aws:execute-api:us-east-1:111111111111:aaabbb/*/GET/* 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 REST avec des autorisations IAM.

Pour créer le rôle IAM, vous pouvez adapter les instructions du didacticiel ???.

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 plus d'informations sur la création d'un mécanisme d'autorisation Lambda, consultez 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 plus d'informations sur la manière de créer un mécanisme d'autorisation de groupe d'utilisateurs Amazon Cognito, consultez Contrôle de l'accès aux 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 Validation des demandes pour les API REST dans API Gateway.

5. Sélectionnez Clé d'API obligatoire pour exiger une clé d'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 Validation des demandes pour les API REST 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 Validation des demandes pour les API REST 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 Modèles de données pour les API REST.

> **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 l'onglet Réponse de 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.

Contrôler et gérer l'accès à REST APIs 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 politiques de ressources vous permettent de créer des politiques basées sur les ressources pour autoriser ou refuser l'accès à vos méthodes APIs et à partir d'adresses IP sources ou de points de terminaison VPC spécifiés. Pour de plus amples informations, veuillez consulter the section called "Utilisation des politiques 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 politiques IAM peuvent être utilisés pour contrôler qui peut les créer et les gérer APIs, ainsi que les personnes habilitées à les invoquer. Pour de plus amples informations, veuillez consulter the section called "Utilisation des autorisations IAM".
- Les balises IAM peuvent être utilisées avec des politiques IAM pour contrôler l'accès. Pour de plus amples informations, veuillez consulter the section called "Attribute-based contrôle d'accès".
- Les politiques de point de terminaison pour les points de terminaison VPC d'interface vous permettent d'associer des politiques de ressources IAM aux points de terminaison VPC d'interface afin d'améliorer la sécurité de vos données privées. APIs Pour de plus amples informations, veuillez consulter the section called "Utilisation de politiques 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 de plus amples informations, veuillez consulter 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 votre REST. APIs Les groupes d'utilisateurs Amazon Cognito sont utilisés pour contrôler qui peut appeler les méthodes d'API REST. 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".

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 de plus amples informations, veuillez consulter 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 de plus amples informations, veuillez consulter the section called " Gateway-generated Certificats API".
- AWS WAF peut être utilisé pour protéger votre API API Gateway contre les menaces web courantes. Pour de plus amples informations, veuillez consulter 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 de plus amples informations, veuillez consulter the section called "Plans d'utilisation".

Contrôle d'accès à une API REST avec des politiques de ressources API Gateway

Les politiques 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 politiques 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 ;
- Clouds privés virtuels (VPCs) ou points de terminaison VPC spécifiés (quel que soit le compte).

Vous pouvez joindre une politique de ressources pour n'importe quel type de point de terminaison d'API dans API Gateway à l' Console de gestion AWS aide de la AWS CLI ou AWS SDKs. Pour le mode privé APIs, vous pouvez utiliser des politiques de ressources ainsi que des politiques de point de terminaison VPC pour contrôler quels principaux ont accès à quelles ressources et actions. Pour de plus amples informations, veuillez consulter the section called "Utilisation de politiques de point de terminaison de VPC pour des API privées".

Les politiques de ressources API Gateway sont différentes des politiques basées sur l'identité IAM. Les politiques 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 politiques de ressources API Gateway sont attachées à des ressources. Vous pouvez utiliser des politiques de ressources API Gateway avec les politiques IAM. Pour plus d'informations, consultez Politiques basées sur l'identité et politiques basées sur une ressource.

Rubriques

- Présentation du langage de la politique d'accès pour Amazon API Gateway
- Comment les politiques de ressources API Gateway affectent le flux de travail d'autorisation
- Exemples de politiques de ressources API Gateway
- Création et attachement d'une politique de ressources API Gateway à une API
- AWS clés de condition pouvant être utilisées dans les politiques de ressources d'API Gateway

Présentation du langage de la politique d'accès pour Amazon API Gateway

Cette page décrit les éléments de base utilisés dans les politiques de ressources Amazon API Gateway.

Les politiques de ressources sont spécifiées en utilisant la même syntaxe que dans les politiquess IAM. Pour obtenir des informations complètes sur le langage de politique d'accès, consultez

Présentation des politiques IAM et Référence des politiques Gestion des identités et des accès AWS 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 politique de ressources contient à la base les éléments suivants :

- Ressources : APIs il s'agit des ressources Amazon API Gateway pour lesquelles vous pouvez autoriser ou refuser des autorisations. Dans une politique, 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 politique de ressources. Pour en savoir plus, veuillez consulter la section Exemples de politiques de ressources API Gateway.

  Pour connaître le format de l'élément `Resource` complet, consultez 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 politique 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 plus d'informations, consultez Différence entre refus par défaut et refus explicite.

- Principal – Compte ou utilisateur autorisé à accéder aux actions ou ressources dans l'instruction. Dans une politique de ressources, le principal est l'utilisateur ou le compte qui reçoit cette autorisation.

L'exemple de politique de ressources suivant montre les éléments de politique courants précédemment évoqués. La politique accorde l'accès à l'API selon les termes spécifiés *account-id region* à tout utilisateur dont l'adresse IP source se trouve dans le bloc d'adresses*123.4.5.6/24*. La politique refuse tout accès à l'API si l'adresse IP source de l'utilisateur n'est pas comprise dans cette plage.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": "*",
            "Action": "execute-api:Invoke",
            "Resource": "arn:aws:execute-api:us-east-1:111111111111:*"
        },
        {
            "Effect": "Deny",
            "Principal": "*",
            "Action": "execute-api:Invoke",
            "Resource": "arn:aws:execute-api:us-east-1:111111111111:*",
            "Condition": {
                "NotIpAddress": {
                    "aws:SourceIp": "123.4.5.6/24"
                }
            }
        }
    ]
}
```

Comment les politiques de ressources API Gateway affectent le flux de travail d'autorisation

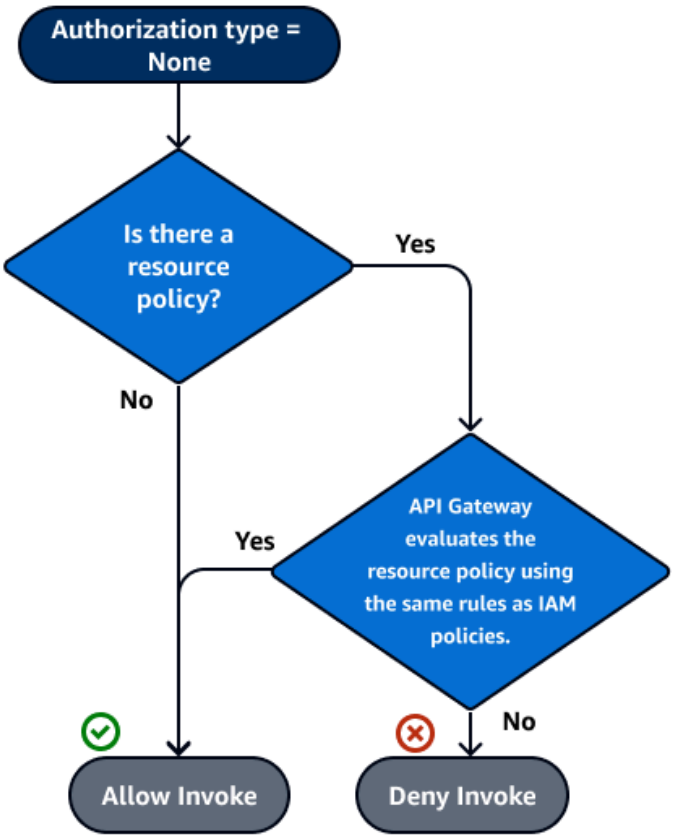
Lorsqu'API Gateway évalue la politique de ressources attachée à votre API, le résultat est affecté par le type d'authentification que vous avez défini pour l'API, comme illustré dans les diagrammes présentés dans les sections suivantes.

Rubriques

- Politique de ressources API Gateway uniquement
- Mécanisme d'autorisation Lambda et politique de ressources
- Politique de ressources et authentification IAM
- Politique de ressources et d'authentification Amazon Cognito
- Tableaux de résultat des évaluations de politiques

Politique de ressources API Gateway uniquement

Dans ce flux de travail, une politique de ressources API Gateway est attachée à l'API, mais aucun type d'authentification n'est défini pour l'API. L'évaluation de la politique 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 de politique de ressources.

JSON

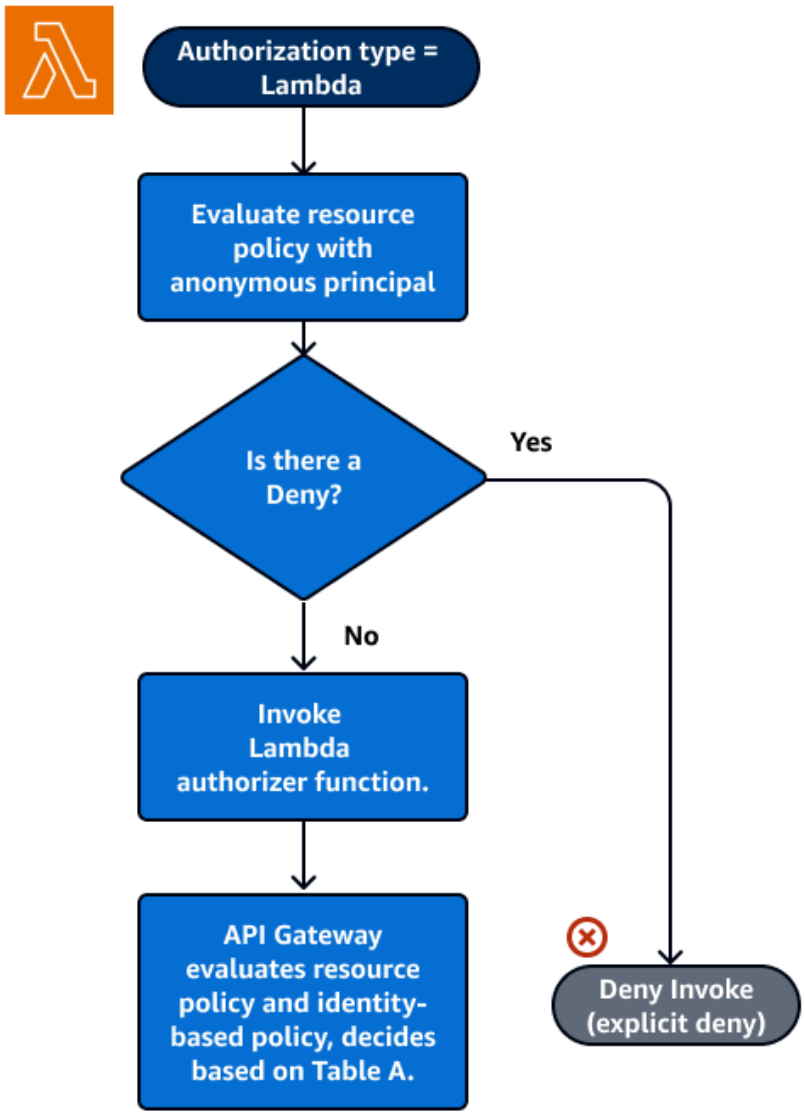
```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": "*",
            "Action": "execute-api:Invoke",
            "Resource": "arn:aws:execute-api:us-east-1:111111111111:api-id/",
            "Condition": {
                "IpAddress": {
                    "aws:SourceIp": ["192.0.2.0/24", "198.51.100.0/24" ]
                }
            }
        }
    ]
```

```
}
```

Mécanisme d'autorisation Lambda et politique de ressources

Dans ce flux de travail, un mécanisme d'autorisation Lambda est configuré pour l'API en plus d'une politique de ressources. La politique de ressources est évaluée en deux phases. Avant d'appeler le mécanisme d'autorisation Lambda, API Gateway évalue la politique 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 politique évalué avec la politique de ressources. Si votre mécanisme d'autorisation utilise la mise en cache, API Gateway peut renvoyer le document de politique mis en cache. Le résultat est déterminé sur la base du tableau A.

Dans l'exemple de politique 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. Ce flux de travail d'autorisation est le même si vous utilisez un nom de domaine personnalisé pour une API privée.



JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Principal": "*",
            "Action": "execute-api:Invoke",
            "Resource": [
                "arn:aws:execute-api:us-east-1:111111111111:api-id/"
            ],
            "Condition" : {
```

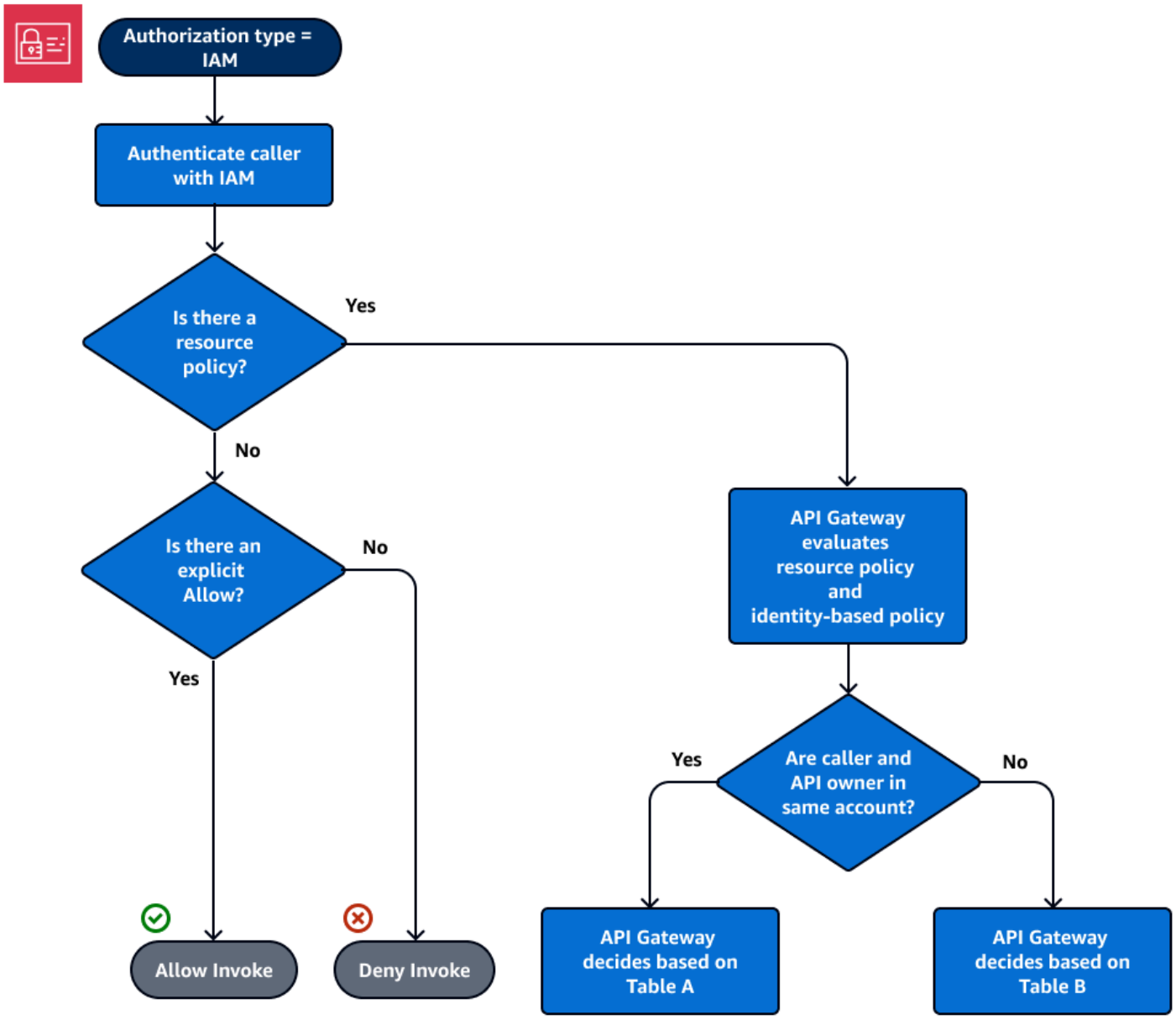
```
                "StringNotEquals": {
                    "aws:SourceVpce": "vpce-1a2b3c4d"
                }
            }
        }
    ]
}
```

Politique de ressources et authentification IAM

Dans ce flux de travail, vous configurez l'authentification IAM pour l'API en plus d'une politique de ressources. Une fois que vous avez authentifié l'utilisateur auprès du service IAM, l'API évalue à la fois les politiques attachées à l'utilisateur et la politique 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 politique de ressources autorisent toutes deux explicitement l'appelant à continuer. Pour plus d'informations, consultez Tableau B.

Cependant, si l'appelant et le propriétaire de l'API relèvent du même Compte AWS, les politiques utilisateur IAM ou la politique de ressources doivent explicitement autoriser l'appelant à continuer. Pour plus d'informations, consultez Tableau A.



Voici un exemple de politique de ressources entre comptes. À supposer que la politique IAM contient un effet Allow (Autoriser), cette politique de ressources autorise uniquement les appels du VPC dont l'ID est *vpc-2f09a348*. Pour plus d'informations, consultez Tableau B.

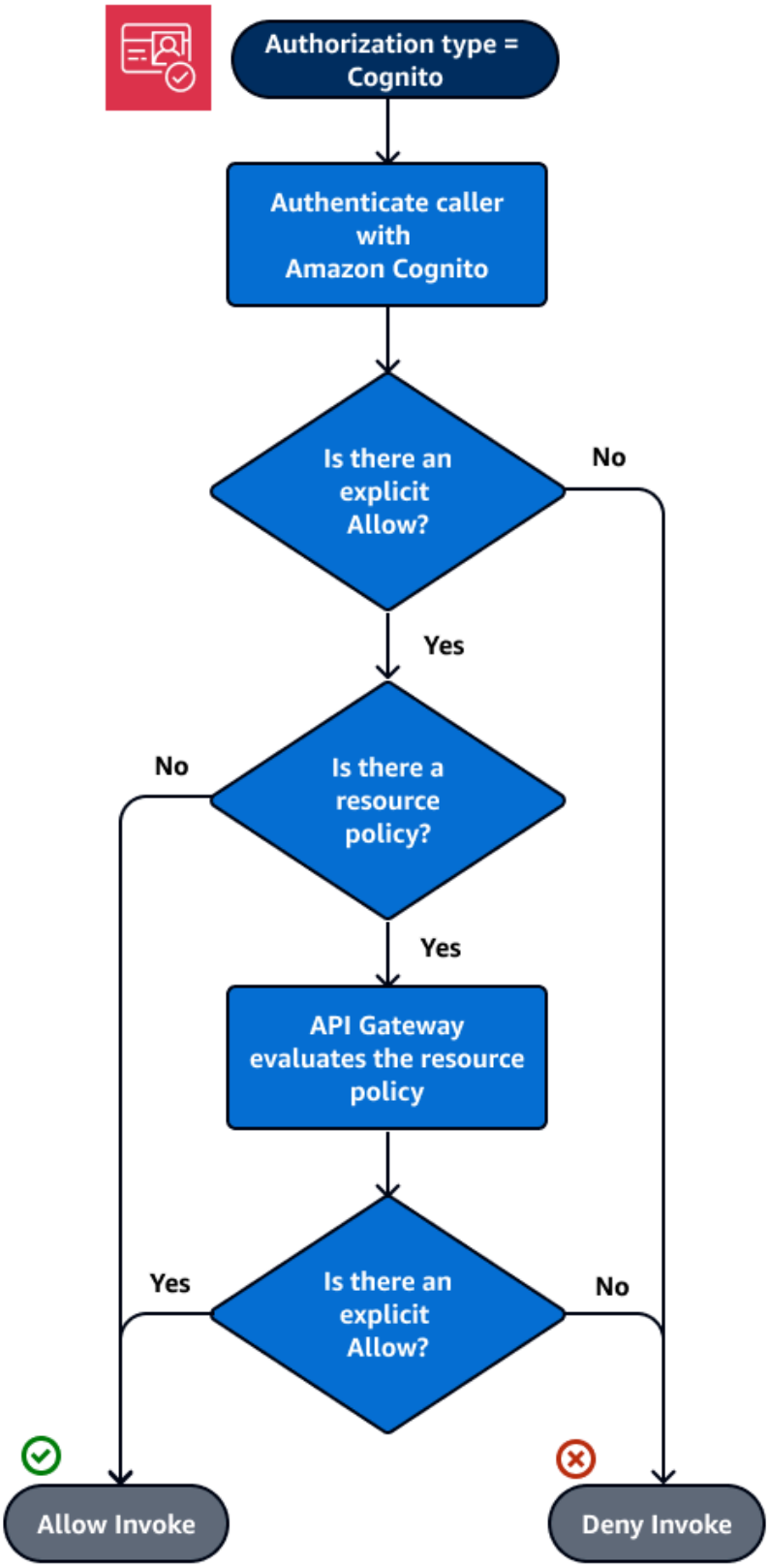
JSON

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

```
            "Effect": "Allow",
            "Principal": "*",
            "Action": "execute-api:Invoke",
            "Resource": [
                "arn:aws:execute-api:us-east-1:111111111111:api-id/"
            ],
            "Condition" : {
                "StringEquals": {
                    "aws:SourceVpc": "vpc-2f09a348"
                    }
            }
        }
    ]
}
```

Politique 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 politique 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 politique 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 politique 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 politique 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 Tableau B.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": "*",
            "Action": "execute-api:Invoke",
            "Resource": "arn:aws:execute-api:us-east-1:111111111111:api-id/",
            "Condition": {
                "IpAddress": {
                    "aws:SourceIp": ["192.0.2.0/24", "198.51.100.0/24" ]
                }
            }
        }
    ]
}
```

Tableaux de résultat des évaluations de politiques

Le tableau A répertorie le comportement obtenu lorsque l'accès à une API API Gateway est contrôlé par une politique IAM ou un mécanisme d'autorisation Lambda et une politique de ressources API Gateway, qui se trouvent toutes les deux dans le même Compte AWS.

Politique IAM (ou mécanisme d'autorisation Lambda)	Politique 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

Politique IAM (ou mécanisme d’autorisation Lambda)	Politique de ressources API Gateway	Comportement obtenu
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
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 intercompte nécessite que la politique de ressources et la politique IAM ou le mécanisme d’autorisation des groupes d’utilisateurs Amazon Cognito accordent explicitement l’accès.

Politique IAM (ou mécanisme d’autorisation des groupes d’utilisateurs Amazon Cognito)	Politique 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 mécanisme d'autorisation des groupes d'utilisateurs Amazon Cognito)	Politique de ressources API Gateway	Comportement obtenu
Refuser	Refuser	Refus explicite

Exemples de politiques de ressources API Gateway

Cette page présente quelques exemples de cas d'utilisation standard des politiques de ressources API Gateway.

Les exemples de politique 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 politique. Par exemple, vous pouvez spécifier la ressource `execute-api:/stage-name/GET/pets` dans une politique de ressources. API Gateway convertit la ressource `arn:aws:execute-api:us-east-2:123456789012:aabbccddee/stage-name/GET/pets` lorsque vous enregistrez la politique 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 de la politique d'accès, consultez Présentation du langage de la politique d'accès pour Amazon API Gateway.

Rubriques

- Exemple : autoriser les rôles d'un autre AWS compte à utiliser une API
- Exemple : refuser le trafic API en fonction de l'adresse IP source ou de la plage
- Exemple : Refuser le trafic API en fonction de l'adresse IP source ou de la plage lors de l'utilisation d'une API privée
- Exemple : autoriser le trafic de l'API privée en fonction du point de terminaison d'un VPC ou d'un VPC source

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) ou Signature

Version 4a (SigV4a). 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`

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": [
                    "arn:aws:iam::111122223333:role/developer",
                    "arn:aws:iam::111122223333:role/Admin"
                ]
            },
            "Action": "execute-api:Invoke",
            "Resource": [
                "execute-api:/stage/GET/pets"
            ]
        }
    ]
}
```

Exemple : refuser le trafic API en fonction de l’adresse IP source ou de la plage

L’exemple de politique de ressources suivant refuse (bloque) le trafic entrant vers une API à partir de deux blocs d’adresses IP source spécifiés.

JSON

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

```
                "execute-api:/*"
            ]
        },
        {
            "Effect": "Deny",
            "Principal": "*",
            "Action": "execute-api:Invoke",
            "Resource": [
               "execute-api:/*"
            ],
            "Condition" : {
                "IpAddress": {
                    "aws:SourceIp": ["192.0.2.0/24", "198.51.100.0/24" ]
                }
            }
        }
    ]
}
```

Si vous utilisez des politiques utilisateur IAM ou des politiques de ressources API Gateway pour contrôler l'accès à API Gateway ou à toute autre API Gateway APIs, vérifiez que vos politiques sont mises à jour pour inclure les plages d' IPv6 adresses. Les politiques qui ne sont pas mises à jour pour gérer IPv6 les adresses peuvent avoir un impact sur l'accès du client à API Gateway lorsqu'il commence à utiliser le point de terminaison dualstack. Pour de plus amples informations, veuillez consulter the section called “Utilisation des IPv6 adresses dans les politiques IAM”.

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 politique de ressources suivant refuse (bloque) le trafic entrant vers une API privée à partir de deux blocs d’adresses IP source spécifiés. Lorsque vous utilisez le mode privé APIs, 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.

JSON

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

```
            "Effect": "Allow",
            "Principal": "*",
            "Action": "execute-api:Invoke",
            "Resource": [
                "execute-api:/*"
            ]
        },
        {
            "Effect": "Deny",
            "Principal": "*",
            "Action": "execute-api:Invoke",
            "Resource": [
               "execute-api:/*"
            ],
            "Condition" : {
                "IpAddress": {
                    "aws:VpcSourceIp": ["192.0.2.0/24", "198.51.100.0/24"]
                }
            }
        }
    ]
}
```

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 politiques 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 politique de ressources spécifie un VPC source :

JSON

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

```
            ]
        },
        {
            "Effect": "Deny",
            "Principal": "*",
            "Action": "execute-api:Invoke",
            "Resource": [
                "execute-api:/*"
            ],
            "Condition" : {
                "StringNotEquals": {
                   "aws:SourceVpc": "vpc-1a2b3c4d"
                }
            }
        }
    ]
}
```

Cet exemple de politique de ressources spécifie un point de terminaison d'un VPC source :

JSON

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

```
                    "aws:SourceVpce": "vpce-1a2b3c4d"
                }
            }
        }
    ]
}
```

Création et attachement d'une politique de ressources API Gateway à une API

Pour permettre à un utilisateur d'accéder à votre API en appelant le service d'exécution des API, vous devez créer une politique de ressources API Gateway et l'attacher à l'API. Lorsque vous attachez une politique votre API, celle-ci applique les autorisations de la politique aux méthodes de l'API. En cas de mise à jour de la politique de ressources, vous devrez déployer l'API.

Rubriques

- Conditions préalables
- Attachement d'une politique de ressources à une API API Gateway
- Résolution des problèmes liés à votre politique de ressources

Conditions préalables

Pour mettre à jour une politique de ressources API Gateway, vous devez disposer des autorisations `apigateway:UpdateRestApiPolicy` et `apigateway:PATCH`.

Pour une API optimisée pour la périphérie ou régionale, vous pouvez attacher 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 de plus amples informations, veuillez consulter the section called "API REST privées".

Attachement d'une politique de ressources à une API API Gateway

La procédure suivante vous montre comment attacher une politique de ressources à une API API Gateway.

Console de gestion AWS

Pour attacher une politique 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 Create Policy (Créer une stratégie).
5. (Facultatif) Choisissez Sélectionnez un modèle pour générer un exemple de politique.

   Dans les exemples de politiques, 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 politique 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 politique de ressources prenne effet.

AWS CLI

AWS CLI Pour créer une nouvelle API et y associer une politique de ressources, utilisez la create-rest-apicommande suivante :

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

AWS CLI Pour associer une politique de ressources à une API existante, utilisez la update-rest-apicommande suivante :

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

Vous pouvez également joindre votre politique de ressources dans un `policy.json` fichier distinct et l'inclure dans votre create-rest-apicommande. La create-rest-apicommande suivante crée une nouvelle API avec une politique de ressources :

```
aws apigateway create-rest-api \
    --name "api-name" \
    --policy file://policy.json
```

`policy.json` est une politique de ressources API Gateway, comme the section called “Exemple : refuser le trafic API en fonction de l’adresse IP source ou de la plage”.

AWS CloudFormation

Vous pouvez l'utiliser 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
      RequestTemplates:
        application/json: '{"statusCode": 200}'
      IntegrationResponses:
        - StatusCode: 200
          ResponseTemplates:
            application/json: '{}'
    MethodResponses:
      - StatusCode: 200
        ResponseModels:
          application/json: 'Empty'
ApiDeployment:
  Type: 'AWS::ApiGateway::Deployment'
  DependsOn:
    - MethodGet
  Properties:
    RestApiId: !Ref Api
    StageName: test
```

Résolution des problèmes liés à votre politique de ressources

Les conseils suivants peuvent vous aider à résoudre les problèmes liés à votre politique de ressources.

Mon API renvoie {"Message":"L'utilisateur: anonyme 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 principal un AWS principal, tel que ce qui suit :

JSON

```
{
```

```
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": [
                    "arn:aws:iam::111111111111:role/developer",
                    "arn:aws:iam::111111111111:role/Admin"
                ]
            },
            "Action": "execute-api:Invoke",
            "Resource": [
                "execute-api:/stage/GET/pets"
            ]
        }
    ]
}
```

Vous devez utiliser l'autorisation `AWS_IAM` pour chaque méthode de votre API, sinon votre API renvoie le message d'erreur précédent. Pour plus d'instructions sur l'activation de l'autorisation `AWS_IAM` pour une méthode, consultez the section called "Méthodes".

Ma politique de ressources n'est pas mise à jour

Si vous mettez à jour la politique de ressources une fois l'API créée, vous devez déployer l'API pour propager les changements une fois le rattachement de la politique mise à jour effectué. La simple mise à jour ou le simple enregistrement de la politique 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".

Ma politique de ressources renvoie l'erreur suivante : document de politique non valide. Vérifiez la syntaxe de la politique et assurez-vous que les principaux sont valides.

Pour corriger cette erreur, nous vous recommandons de vérifier d'abord la syntaxe de la politique. Pour de plus amples informations, veuillez consulter the section called "Présentation du langage de la politique d'accès pour Amazon API Gateway". Nous vous recommandons également de vérifier que tous les principaux spécifiés sont valides et n'ont pas été supprimés.

En outre, si votre API se trouve dans une région d'acceptation, vérifiez que la région est activée pour tous les comptes de la politique de ressources.

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 d'API Gateway pour chaque type d'autorisation. APIs

Pour plus d'informations sur les clés de AWS condition, consultez la section Clés contextuelles de condition AWS globales.

Clés de condition	Critères	Nécessite **AuthN** ?	Type d’autorisation
`aws:CurrentTime`	Aucun	Non	Tous
`aws:EpochTime`	Aucun	Non	Tous
`aws:TokenIssueTime`	Cette clé est uniquement présente dans les demandes signées à l'aide d'informations d'identification de sécurité temporaires.	Oui	IAM
`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

Clés de condition	Critères	Nécessite **AuthN** ?	Type d'autorisation
`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
`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

Clés de condition	Critères	Nécessite **AuthN** ?	Type d'autorisation
`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
`aws:SourceVpc`	Cette clé ne peut être utilisée qu'à des fins privées APIs.	Non	Tous
`aws:SourceVpce`	Cette clé ne peut être utilisée qu'à des fins privées APIs.	Non	Tous
`aws:VpcSourceIp`	Cette clé ne peut être utilisée qu'à des fins privées APIs.	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 REST 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 politiques 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. Suivez les instructions de la rubrique Création d’un rôle pour un fournisseur d’identité tiers (fédération) dans le 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 dans le 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. API Gateway prend en charge Signature Version 4a (SigV4a) et Signature Version 4 (SigV4) pour authentifier les informations d'identification de l'utilisateur. Pour plus d'informations, consultez AWS Signature Version 4. Vous pouvez ensuite attacher la politique à un utilisateur, un groupe ou un rôle.

Dans cette déclaration de politique 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) pour le compte 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 politique d'approbation IAM suivante qui déclare API Gateway en tant qu'entité de confiance autorisée à assumer le rôle :

JSON

```
{
  "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 en appelant 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'autorisations IAM appropriées pour appeler les AWS services intégrés. Par exemple, pour appeler une fonction Lambda, vous devez inclure la politique d'autorisations IAM suivante dans le rôle IAM :

JSON

```
{
    "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 appliquer le contrôle d'accès à un AWS service, vous pouvez utiliser soit le modèle d'autorisations basé sur l'appelant, dans lequel une politique d'autorisations est directement associée à l'utilisateur ou au groupe de l'appelant, soit le modèle d'autorisation basé sur les rôles, dans lequel une politique d'autorisations est attachée à un rôle IAM qu'API Gateway peut assumer. Les politiques d'autorisation peuvent varier dans les deux modèles. Par exemple, la politique basée sur l'appelant bloque l'accès tandis que la politique basée sur les rôles l'autorise. Vous pouvez en tirer parti 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 découvrir le modèle d'autorisation permettant de contrôler l'accès à votre API avec des autorisations IAM. Lorsque l'autorisation IAM est activée, les clients doivent utiliser la version de signature 4a (SigV4a) et la version de signature 4 (Sigv4) pour signer leurs demandes à l'aide d'informations d'identification. AWS Pour plus d'informations, consultez AWS Signature Version 4.

Dans cette section, nous allons vous présenter un modèle de déclaration de politique IAM et la référence de cette déclaration. La référence de la déclaration de politique inclut les formats des champs `Action` et `Resource` liés au service d'exécution des API. Utilisez ces références pour créer votre déclaration de politique IAM. Lorsque vous créez votre déclaration de politique IAM, tenez compte de la façon dont les politiques de ressources API Gateway affectent le flux d'autorisation. Pour de plus amples informations, veuillez consulter the section called "Comment les politiques de ressources affectent le flux de travail d'autorisation".

Pour les API privées, vous devez utiliser une combinaison constituée d'une politique 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 politiques de ressources API Gateway"
- the section called "Utilisation de politiques de point de terminaison de VPC pour des API privées"

Contrôle des personnes habilitées à appeler une méthode d'API API Gateway avec les politiques IAM

Pour contrôler qui peut ou non appeler une API déployée avec les autorisations IAM, créez un document de politique IAM avec les autorisations requises. Un modèle pour un tel document de politique est affiché comme suit.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Permission",
      "Action": [
        "execute-api:Execution-operation"
      ],
      "Resource": [
        "arn:aws:execute-api:region:123456789012:api-
id/stage/METHOD_HTTP_VERB/Resource-path"
      ]
    }
  ]
}
```

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 de plus amples informations, veuillez consulter Référence de déclaration de politique IAM pour l'exécution des API dans API Gateway.

> **Note**
>
> Pour que les politiques 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 politique IAM :

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "execute-api:Invoke"
      ],
      "Resource": [
        "arn:aws:execute-api:us-east-1:111111111111:api-id/*/GET/pets"
      ]
    },
    {
      "Effect": "Deny",
      "Action": [
        "execute-api:Invoke"
      ],
      "Resource": [
        "arn:aws:execute-api:us-east-1:111111111111:api-id/*/POST/pets"
      ]
    }
  ]
}
```

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 politique IAM :

JSON

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

```
            "Effect": "Allow",
            "Action": [
                "execute-api:Invoke"
            ],
            "Resource": [
                "arn:aws:execute-api:us-east-1:111122223333:api-id/*/
GET/pets/a1b2"
            ]
        }
    ]
}
```

Référence de déclaration de politique IAM pour l'exécution des API dans API Gateway

Les informations suivantes décrivent le format des expressions Action et Resource des déclarations de politique 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

Le format général de l' API-executing `Action`expression est le suivant :

```
execute-api:action
```

où *`action`* est une API-executing action disponible :

- *, qui représente l'ensemble des actions suivantes.
- Invoke, utilisée pour appeler une API sur demande d'un client.
- InvalidateCache, utilisée pour invalider le cache API sur demande d'un client.

Format de l'expression Resource des autorisations d'exécution d'API dans API Gateway

Le format général de l' API-executing `Resource`expression est le suivant :

```
arn:aws:execute-api:region:account-id:api-id/stage-name/HTTP-VERB/resource-path-
specifier
```

où :

- *`region`*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 de 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 n'importe quel 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 n'importe quel chemin de ressource à n'importe quelle étape, pour l'API dont l'identifiant est *`api-id`* dans la AWS région us-east-1.
- **`arn:aws:execute-api:us-east-1:*:`*`api-id`*`/test/*`**pour tout chemin de ressource au stade de`test`, pour l'API avec l'identifiant de *`api-id`* dans la AWS région us-east-1.

Pour en savoir plus, veuillez consulter la section Référence Amazon Resource Name (ARN) API Gateway.

Exemples de politiques 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 politique 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 :

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "execute-api:Invoke"
      ],
      "Resource": [
        "arn:aws:execute-api:us-east-1:*:a123456789/test/POST/my-demo-resource-
path/*"
      ]
    }
  ]
}
```

L'exemple de déclaration de politique 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 :

JSON

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

Utilisation de politiques 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 de VPC. La politique de point de terminaison de VPC est une politique de ressources IAM qui est jointe au 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 pouvez créer une politique de point de terminaison de VPC pour exécuter les tâches suivantes.

- Autoriser uniquement certaines organisations ou ressources à accéder à votre point de terminaison de VPC et à invoquer votre API.
- Utiliser une seule politique et éviter les politiques basées sur les sessions ou les rôles pour contrôler le trafic à destination de votre API.
- Renforcez 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 de VPC

Tenez compte des points suivants pour votre politique de point de terminaison de VPC :

- L'identité de l'invocateur est évaluée en fonction de la valeur de l'en-tête `Authorization`. La politique du point de terminaison du VPC est d'abord évaluée, puis API Gateway évalue la demande en fonction du type d'autorisation configuré dans la demande de méthode. Le tableau suivant montre comment la politique de point de terminaison du VPC est évaluée en fonction du contenu de la valeur d'en-tête `Authorization`.

Contenu de la valeur d'en-tête **`Authorization`**	Méthode d'évaluation de la politique de point de terminaison du VPC
Pas de contenu	L'appelant est évalué en tant qu'utilisateur anonyme
Signature SigV4 ou SigV4a valide	L'appelant est évalué en tant qu'identité IAM authentifiée à partir de la signature
Signature SigV4 ou SigV4a non valide	API Gateway refuse l'accès

Contenu de la valeur d'en-tête **`Authorization`**	Méthode d'évaluation de la politique de point de terminaison du VPC
Non-SigV4 des informations d'autorisation, telles qu'un jeton au porteur	L'appelant est évalué en tant qu'utilisateur anonyme

- Si votre contrôle d'accès repose sur l'utilisation d'un jeton porteur, tel qu'un mécanisme d'autorisation Lambda ou Amazon Cognito, vous pouvez contrôler votre périmètre de sécurité à l'aide des propriétés de la ressource.
- Si votre contrôle d'autorisation utilise l'autorisation IAM, vous pouvez contrôler votre périmètre de sécurité en utilisant les propriétés de la ressource et les propriétés du principal.
- Les politiques de point de terminaison de VPC peuvent être utilisées conjointement avec des politiques de ressources API Gateway. La politique de ressources API Gateway spécifie les principaux pouvant accéder à l'API. La politique de point de terminaison spécifie 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 il n'est pas nécessaire de créer une politique de point de terminaison de VPC personnalisée.

Exemples de politique de point de terminaison de VPC

Vous pouvez créer des politiques pour les points de terminaison Amazon Virtual Private Cloud pour Amazon API Gateway dans lesquelles vous pouvez spécifier :

- Le principal qui peut exécuter des actions.
- Les actions qui peuvent être effectuées.
- Les ressources qui peuvent avoir des actions exécutées sur elles.

Cela dépend du contenu de votre en-tête d'autorisation. Pour de plus amples informations, veuillez consulter the section called "Considérations d'une politique de point de terminaison de VPC". Pour d'autres exemples de politiques, voir Exemples de politiques en matière de périmètre de données sur le GitHub site Web.

Pour attacher la politique 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 : politique de point de terminaison de VPC accordant l'accès à deux API

L'exemple de politique suivant accorde l'accès à seulement deux API spécifiques via le point de terminaison de VPC auquel la politique est attachée.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Principal": "*",
            "Action": [
                "execute-api:Invoke"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:execute-api:us-east-1:123412341234:a1b2c3d4e5/*",
                "arn:aws:execute-api:us-east-1:123412341234:aaaaa11111/*"
            ]
        }
    ]
}
```

Exemple 2 : politique de point de terminaison de VPC accordant l'accès aux méthodes GET

L'exemple de politique suivant accorde aux utilisateurs l'accès aux méthodes `GET` pour une API spécifique via le point de terminaison de VPC auquel la politique est attachée.

JSON

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

```
            "Resource": [
                "arn:aws:execute-api:us-east-1:123412341234:a1b2c3d4e5/stageName/
GET/*"
            ]
        }
    ]
}
```

Exemple 3 : politique de point de terminaison de VPC accordant à un utilisateur spécifique l'accès à une API spécifique

L'exemple de politique suivant accorde à un utilisateur spécifique l'accès à une API spécifique via le point de terminaison de VPC auquel la politique est attachée.

Dans ce cas, étant donné que la politique restreint l'accès à des principaux IAM spécifiques, vous devez définir `authorizationType` sur `AWS_IAM` ou `NONE`.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Principal": {
                "AWS": [
                    "arn:aws:iam::123412341234:user/MyUser"
                ]
            },
            "Action": [
                "execute-api:Invoke"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:execute-api:us-east-1:123412341234:a1b2c3d4e5/*"
            ]
        }
    ]
}
```

Exemple 4 : politique de point de terminaison de VPC autorisant les utilisateurs à accéder à un nom de domaine personnalisé spécifique et à chaque API mappée à ce domaine

L'exemple de politique suivant autorise les utilisateurs à accéder à un nom de domaine personnalisé spécifique pour des API privées via le point de terminaison de VPC auquel la politique est attachée. Avec cette politique, tant que l'utilisateur a créé une association d'accès au nom de domaine entre le point de terminaison du VPC et le nom de domaine personnalisé et qu'il est autorisé à invoquer le nom de domaine personnalisé et toutes les API privées mappées à celui-ci, il peut invoquer toutes les API mappées à ce nom de domaine personnalisé. Pour de plus amples informations, veuillez consulter the section called "Noms de domaine personnalisés pour les API privées".

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "*"
      },
      "Action": "execute-api:Invoke",
      "Resource": [
        "*"
      ],
       "Condition": {
        "ArnEquals": {
          "execute-api:viaDomainArn": "arn:aws:execute-api:us-
west-2:111122223333:/domainnames/private.test.com+f4g5h6"
        }
      }
    }
  ]
}
```

Exemple 5 : politique de point de terminaison de VPC accordant ou refusant l'accès à des API et à des ressources de domaine spécifiques

L'exemple de politique suivant autorise les utilisateurs à accéder à des API et à des ressources de domaine spécifiques. Avec cette politique, tant que l'utilisateur a créé une association d'accès au

nom de domaine entre le point de terminaison du VPC et le nom de domaine personnalisé et qu'il est autorisé à invoquer le nom de domaine personnalisé et toutes les API privées mappées à celui-ci, il peut invoquer toutes les API privées et ressources autorisées du domaine. Pour de plus amples informations, veuillez consulter the section called "Noms de domaine personnalisés pour les API privées".

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "*"
      },
      "Action": "execute-api:Invoke",
      "Resource": [
        "arn:aws:execute-api:us-west-2:111122223333:/domainnames/private.test.com
+f4g5h6",
        "arn:aws:execute-api:us-west-2:111122223333:a1b2c3d4e5/*"
      ]
    },
    {
      "Effect": "Deny",
      "Principal": {
        "AWS": "*"
      },
      "Action": "execute-api:Invoke",
      "Resource": [
        "arn:aws:execute-api:us-west-2:111122223333:a1b2c3d4e5/admin/*",
        "arn:aws:execute-api:us-west-2:111122223333:bcd123455/*"
      ]
    }
  ]
}
```

Exemple 6 : politique de point de terminaison de VPC accordant ou refusant l'accès aux principaux et aux ressources appartenant à une organisation

L'exemple de politique suivant autorise l'accès aux principaux et aux ressources appartenant à une organisation.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Condition": {
                "StringEquals": {
                    "aws:ResourceOrgID": "o-abcd1234",
                    "aws:PrincipalOrgID": "o-abcd1234"
                }
            },
            "Action": "*",
            "Resource": "*",
            "Effect": "Allow",
            "Principal": {
                "AWS": "*"
            },
            "Sid": "AllowRequestsByOrgsIdentitiesToOrgsResources"
        }
    ]
}
```

Utiliser des balises pour contrôler l'accès à REST APIs dans API Gateway

L'autorisation d'accès à REST APIs peut être affinée à l'aide du contrôle d'accès basé sur les attributs dans les politiques IAM.

Pour de plus amples informations, veuillez consulter the section called "Attribute-based contrôle d'accès".

Utilisation des mécanismes d'autorisation Lambda API Gateway

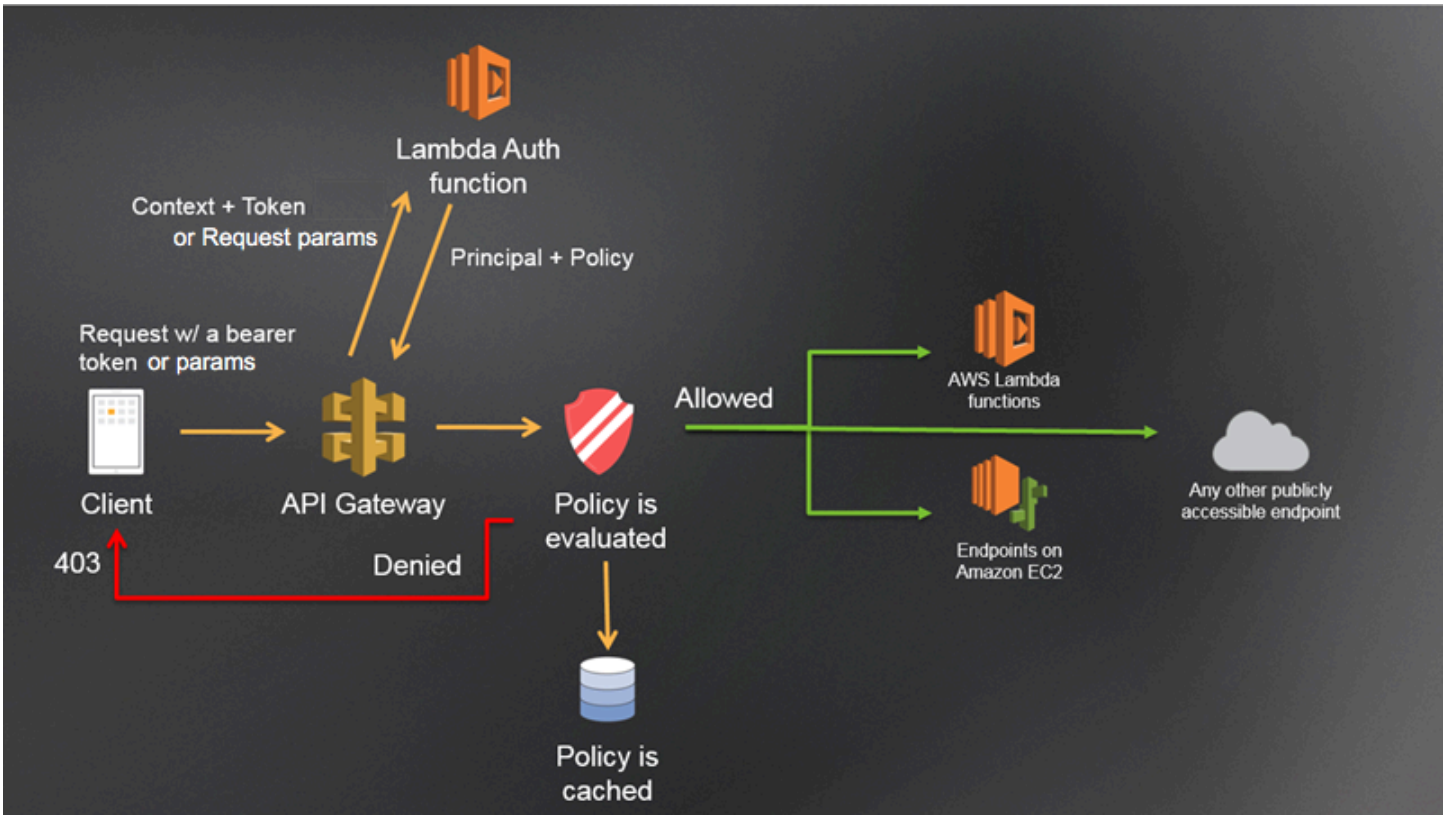
Utilisez un mécanisme d'autorisation Lambda (anciennement appelé mécanisme d'autorisation personnalisée) pour contrôler l'accès à votre API. Lorsqu'un client soumet une méthode sur votre

API, API Gateway appelle votre mécanisme d'autorisation Lambda. Ce dernier prend l'identité de l'appelant en entrée et renvoie une politique IAM en sortie.

Utilisez un mécanisme d'autorisation Lambda pour implémenter un schéma d'autorisation personnalisée. Votre schéma peut utiliser les paramètres de demande pour déterminer l'identité de l'appelant ou utiliser une politique d'authentification par jeton de porteur, telle qu'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 d'autorisation du mécanisme d'autorisation Lambda

Le schéma suivant illustre le flux d'autorisation d'un mécanisme d'autorisation Lambda.



Flux de travail d'autorisation Lambda API Gateway

1. Le client appelle une méthode sur une API API Gateway, en transmettant un jeton de porteur ou des paramètres de demande.
2. API Gateway vérifie si la demande de méthode est configurée avec un mécanisme d'autorisation Lambda. Si c'est le cas, API Gateway appelle la fonction Lambda.
3. La fonction Lambda authentifie l'appelant. La fonction peut procéder à l'authentification 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 la demande.
- En extrayant 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 approprié, par exemple `403 ACCESS_DENIED`.
   - Si l'accès est autorisé, API Gateway invoque la méthode.

     Si vous activez la mise en cache des autorisations, API Gateway met en cache la politique afin que la fonction du mécanisme d'autorisation Lambda ne soit pas invoquée à nouveau. Assurez-vous que votre politique s'applique à toutes les ressources et méthodes de votre API.

Vous pouvez personnaliser la réponse de passerelle `403 ACCESS_DENIED` ou `401 UNAUTHORIZED`. Pour en savoir plus, veuillez consulter la section the section called "Réponses de passerelle".

Choix d'un type de mécanisme d'autorisation Lambda

Il existe deux types de mécanismes d'autorisation Lambda :

Mécanisme d'autorisation Lambda basé sur les paramètres de la demande (mécanisme d'autorisation **`REQUEST`**)

Un mécanisme d'autorisation `REQUEST` reçoit l'identité de l'appelant dans une combinaison d'en-têtes, de paramètres de chaîne de requête, et de variables `stageVariables` et `$context`. Vous pouvez utiliser un mécanisme d'autorisation `REQUEST` pour créer des politiques précises basées sur les informations de plusieurs sources d'identité, telles que les variables contextuelles `$context.path` et `$context.httpMethod`.

Si vous activez la mise en cache des autorisations pour un mécanisme d'autorisation `REQUEST`, API Gateway vérifie que toutes les sources d'identité spécifiées sont présentes dans la demande. Si une source d'identité spécifiée est absente, nulle ou vide, API Gateway renvoie une réponse HTTP `401 Unauthorized` sans appeler la fonction du mécanisme d'autorisation Lambda.

Lorsque plusieurs sources d'identité sont définies, elles sont toutes utilisées pour obtenir la clé de cache du mécanisme d'autorisation, en respectant l'ordre établi. Vous pouvez définir une clé de cache précise en utilisant plusieurs sources d'identité.

Si vous modifiez une partie de la clé de cache et redéployez votre API, le mécanisme d'autorisation ignore le document de politique mis en cache et en génère un nouveau.

Si vous désactivez la mise en cache des autorisations pour un mécanisme d'autorisation `REQUEST`, API Gateway transmet directement la demande à la fonction Lambda.

Token-based Autorisateur Lambda (autorisateur) **`TOKEN`**

Un mécanisme d'autorisation `TOKEN` reçoit l'identité de l'appelant dans un jeton de porteur, par exemple un jeton JWT (JSON Web Token) ou un jeton OAuth.

Si vous activez la mise en cache des autorisations pour un mécanisme d'autorisation `TOKEN`, le nom de l'en-tête spécifié dans la source du jeton devient la clé de 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 en fonction de cette expression et invoque la fonction du mécanisme d'autorisation Lambda quand la validation aboutit. Cela contribue à réduire le nombre d'appels à votre API.

La propriété `IdentityValidationExpression` est prise en charge pour les mécanismes d'autorisation `TOKEN` uniquement. Pour de plus amples informations, veuillez consulter the section called "`x-amazon-apigateway-authorizer`".

> **Note**
>
> Nous vous recommandons d'utiliser un mécanisme d'autorisation `REQUEST` pour contrôler l'accès à votre API. Vous pouvez contrôler l'accès à votre API sur la base de plusieurs sources d'identité lorsque vous utilisez un mécanisme d'autorisation `REQUEST`, alors qu'une seule source d'identité suffit lorsque vous utilisez un mécanisme d'autorisation `TOKEN`. En outre, vous pouvez séparer les clés de cache en utilisant plusieurs sources d'identité pour un mécanisme d'autorisation `REQUEST`.

Exemple de fonction **Lambda** de l'autorisateur REQUEST

L'exemple de code suivant crée une fonction de mécanisme d'autorisation Lambda qui autorise une demande si l'en-tête `HeaderAuth1` fourni par le client, le paramètre de requête `QueryString1` et la variable d'étape `StageVar1` correspondent aux valeurs `headerValue1`, `queryValue1` et `stageValue1` spécifiées, respectivement.

Node.js

```
// A simple request-based authorizer example to demonstrate how to use request
// parameters to allow or deny a request. In this example, a request is
// authorized if the client-supplied HeaderAuth1 header, QueryString1
// query parameter, and stage variable of StageVar1 all match
// specified values of 'headerValue1', 'queryValue1', and 'stageValue1',
// respectively.

export const handler = function(event, context, callback) {
    console.log('Received event:', JSON.stringify(event, null, 2));

    // Retrieve request parameters from the Lambda function input:
    var headers = event.headers;
    var queryStringParameters = event.queryStringParameters;
    var pathParameters = event.pathParameters;
    var stageVariables = event.stageVariables;

    // Parse the input for the parameter values
    var tmp = event.methodArn.split(':');
    var apiGatewayArnTmp = tmp[5].split('/');
    var awsAccountId = tmp[4];
    var region = tmp[3];
    var restApiId = apiGatewayArnTmp[0];
    var stage = apiGatewayArnTmp[1];
    var method = apiGatewayArnTmp[2];
    var resource = '/'; // root resource
    if (apiGatewayArnTmp[3]) {
        resource += apiGatewayArnTmp[3];
    }

    // Perform authorization to return the Allow policy for correct parameters and
    // the 'Unauthorized' error, otherwise.


    if (headers.HeaderAuth1 === "headerValue1"
```

```
        && queryStringParameters.QueryString1 === "queryValue1"
        && stageVariables.StageVar1 === "stageValue1") {
        callback(null, generateAllow('me', event.methodArn));
    }  else {
        callback(null, generateDeny('me', event.methodArn));
    }
}

// Help function to generate an IAM policy
var generatePolicy = function(principalId, effect, resource) {
    // Required output:
    var authResponse = {};
    authResponse.principalId = principalId;
    if (effect && resource) {
        var policyDocument = {};
        policyDocument.Version = '2012-10-17'; // default version
        policyDocument.Statement = [];
        var statementOne = {};
        statementOne.Action = 'execute-api:Invoke'; // default action
        statementOne.Effect = effect;
        statementOne.Resource = resource;
        policyDocument.Statement[0] = statementOne;
        authResponse.policyDocument = policyDocument;
    }
    // Optional output with custom properties of the String, Number or Boolean type.
    authResponse.context = {
        "stringKey": "stringval",
        "numberKey": 123,
        "booleanKey": true
    };
    return authResponse;
}

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

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

Python

```
# A simple request-based authorizer example to demonstrate how to use request
# parameters to allow or deny a request. In this example, a request is
# authorized if the client-supplied headerauth1 header, QueryString1
# query parameter, and stage variable of StageVar1 all match
# specified values of 'headerValue1', 'queryValue1', and 'stageValue1',
# respectively.

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

    # Retrieve request parameters from the Lambda function input:
    headers = event['headers']
    queryStringParameters = event['queryStringParameters']
    pathParameters = event['pathParameters']
    stageVariables = event['stageVariables']

    # Parse the input for the parameter values
    tmp = event['methodArn'].split(':')
    apiGatewayArnTmp = tmp[5].split('/')
    awsAccountId = tmp[4]
    region = tmp[3]
    restApiId = apiGatewayArnTmp[0]
    stage = apiGatewayArnTmp[1]
    method = apiGatewayArnTmp[2]
    resource = '/'

    if (apiGatewayArnTmp[3]):
        resource += apiGatewayArnTmp[3]

    # Perform authorization to return the Allow policy for correct parameters
    # and the 'Unauthorized' error, otherwise.

    if (headers['HeaderAuth1'] == "headerValue1" and
 queryStringParameters['QueryString1'] == "queryValue1" and
 stageVariables['StageVar1'] == "stageValue1"):
        response = generateAllow('me', event['methodArn'])
        print('authorized')
        return response
    else:
        print('unauthorized')
        response = generateDeny('me', event['methodArn'])
        return response
```

```
    # Help function to generate IAM policy


def generatePolicy(principalId, effect, resource):
    authResponse = {}
    authResponse['principalId'] = principalId
    if (effect and resource):
        policyDocument = {}
        policyDocument['Version'] = '2012-10-17'
        policyDocument['Statement'] = []
        statementOne = {}
        statementOne['Action'] = 'execute-api:Invoke'
        statementOne['Effect'] = effect
        statementOne['Resource'] = resource
        policyDocument['Statement'] = [statementOne]
        authResponse['policyDocument'] = policyDocument

    authResponse['context'] = {
        "stringKey": "stringval",
        "numberKey": 123,
        "booleanKey": True
    }

    return authResponse


def generateAllow(principalId, resource):
    return generatePolicy(principalId, 'Allow', resource)


def generateDeny(principalId, resource):
    return generatePolicy(principalId, 'Deny', resource)
```

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 politique IAM qui ressemble à ce qui suit, et la demande de la méthode aboutit :

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Action": "execute-api:Invoke",
      "Effect": "Allow",
      "Resource": "arn:aws:execute-api:us-east-1:123456789012:ivdtdhp7b5/
ESTestInvoke-stage/GET/"
    }
  ]
}
```

- Sinon, la fonction du mécanisme d'autorisation renvoie une réponse HTTP `401 Unauthorized` 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. Elle peut éventuellement renvoyer un objet `context` qui inclut des informations supplémentaires transmissibles au backend d'intégration. Pour de plus amples informations, veuillez consulter Sortie d'un mécanisme d'autorisation Lambda API Gateway.

Dans le code de production, vous devrez peut-être 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 Lambda de l'autorisateur **`TOKEN`**

L'exemple de code suivant crée une fonction de mécanisme d'autorisation Lambda`TOKEN` 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 du mécanisme d'autorisation renvoie une réponse `401 UNAUTHORIZED`.

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 politique IAM qui ressemble à ce qui suit, et la demande de méthode aboutit :

  JSON

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

```
      "Resource": "arn:aws:execute-api:us-east-1:123456789012:ivdtdhp7b5/
ESTestInvoke-stage/GET/"
    }
  ]
}
```

- Si la valeur du jeton est `deny`, la fonction du mécanisme d'autorisation renvoie une réponse HTTP `200 OK` et une politique IAM `Deny` qui ressemble à ce qui suit, et la demande de méthode échoue :

  JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Action": "execute-api:Invoke",
      "Effect": "Deny",
      "Resource": "arn:aws:execute-api:us-east-1:123456789012:ivdtdhp7b5/
ESTestInvoke-stage/GET/"
    }
  ]
}
```

> **Note**
>
> En dehors de l'environnement de test, API Gateway renvoie une réponse HTTP `403 Forbidden` 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. Elle peut éventuellement renvoyer un objet `context` qui inclut des informations supplémentaires transmissibles au backend d'intégration. Pour de plus amples informations, veuillez consulter Sortie d'un mécanisme d'autorisation Lambda API Gateway.

Dans le code de production, vous devrez peut-être 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 de mécanisme d'autorisation Lambda

La liste suivante présente d'autres exemples de fonctions de mécanisme d'autorisation Lambda. Vous pouvez créer une fonction Lambda dans le même compte, ou dans un autre compte, que celui dans lequel 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 de mécanisme d'autorisation Lambda

- Pour un exemple d'application, voir Open Banking Brésil - Exemples GitHub d'autorisation sur.
- Pour d'autres exemples de fonctions Lambda, consultez aws-apigateway-lambda-authorizer-blueprints on. GitHub
- Vous pouvez créer un mécanisme d'autorisation Lambda qui authentifie les utilisateurs à l'aide de groupes d'utilisateurs Amazon Cognito, et autorise les appelants sur la base d'un entrepôt de politiques à l'aide de Verified Permissions. Pour de plus amples informations, veuillez consulter the section called "Contrôle de l'accès sur la base des attributs d'une identité avec Verified Permissions".
- La console Lambda fournit un plan Python, que vous pouvez utiliser en choisissant Use a blueprint (Utiliser un plan), puis en choisissant le plan api-gateway-authorizer-python.

Configuration d'un mécanisme d'autorisation Lambda API Gateway

Une fois votre fonction Lambda créée, vous allez la configurer en tant que mécanisme d'autorisation pour votre API. Vous allez ensuite configurer votre méthode pour invoquer votre mécanisme d'autorisation Lambda afin de déterminer si un appelant peut invoquer votre méthode. Vous pouvez créer une fonction Lambda dans le même compte, ou dans un autre compte, que celui dans lequel vous avez créé votre API.

Vous pouvez tester votre mécanisme d'autorisation Lambda à l'aide des outils intégrés de la console API Gateway ou de Postman. Pour apprendre à utiliser Postman pour tester votre fonction

de mécanisme d'autorisation Lambda, consultez the section called "Appel d'une API avec des mécanismes d'autorisation Lambda".

Configuration d'un mécanisme d'autorisation Lambda (console)

La procédure suivante montre comment créer un mécanisme d'autorisation Lambda dans la console d'API REST API Gateway. Pour en savoir plus sur les différents types de mécanismes d'autorisation Lambda, consultez the section called "Choix d'un type de mécanisme d'autorisation 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 Mécanismes d'autorisation.
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 Rôle d'appel Lambda vide pour que la console d'API REST API Gateway définisse une politique basée sur les ressources. La politique autorise API Gateway à invoquer la fonction Lambda du mécanisme d'autorisation. Vous pouvez également choisir de saisir le nom d'un rôle IAM pour autoriser API Gateway à invoquer la fonction du mécanisme d'autorisation Lambda. Pour obtenir un exemple de rôle, consultez Création 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. Quand la mise en cache de la politique 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 mécanisme d'autorisation doit renvoyer une politique applicable à toutes les méthodes d'une API. Pour appliquer une politique spécifique à une méthode, utilisez les variables contextuelles `$context.path` et `$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 Mécanismes d'autorisation.
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 Rôle d'appel Lambda vide pour que la console d'API REST API Gateway définisse une politique basée sur les ressources. La politique autorise API Gateway à invoquer la fonction Lambda du mécanisme d'autorisation. Vous pouvez également choisir de saisir le nom d'un rôle IAM pour autoriser API Gateway à invoquer la fonction du mécanisme d'autorisation Lambda. Pour obtenir un exemple de rôle, consultez Création 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 portant ce nom pour envoyer le jeton d'autorisation au mécanisme d'autorisation 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. Quand

la mise en cache de la politique 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 mécanisme d'autorisation doit renvoyer une politique applicable à toutes les méthodes d'une API. Pour appliquer une politique spécifique à la méthode, vous pouvez désactiver Mise en cache des autorisations.

12. Choisissez Créer un mécanisme d'autorisation.

Une fois que vous avez créé votre mécanisme d'autorisation Lambda, vous pouvez le tester. La procédure suivante montre comment tester votre mécanisme d'autorisation 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 mécanisme d'autorisation.
3. Sous Tester le mécanisme d'autorisation, saisissez une valeur pour votre source d'identité.

   Si vous utilisez l'the section called "Exemple de fonction Lambda de l'autorisateur REQUEST", 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 contextuelles pour l'invocation de test, mais vous pouvez modifier le modèle d'événement de test Mécanisme d'autorisation API Gateway pour votre fonction Lambda. Vous pouvez ensuite tester la fonction de votre mécanisme d'autorisation Lambda avec des variables contextuelles modifiées. Pour plus d'informations, consultez Testing Lambda functions in the console dans le Guide du développeur AWS Lambda .
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 mécanisme d'autorisation.
3. Sous Tester le mécanisme d'autorisation, saisissez une valeur pour votre jeton.

   Si vous utilisez l'the section called "Exemple de fonction Lambda de l'autorisateur TOKEN", procédez comme suit :

   - Pour authorizationToken, saisissez **`allow`**.
4. Choisissez Tester le mécanisme d'autorisation.

   Si votre mécanisme d'autorisation Lambda refuse avec succès une demande dans l'environnement de test, le test répond par une réponse HTTP `200 OK`. En revanche, en dehors de l'environnement de test, API Gateway renvoie une réponse HTTP `403 Forbidden` et la demande de méthode échoue.

Configurer un autorisateur Lambda (AWS CLI)

La commande create-authorizer suivante montre comment créer un mécanisme d'autorisation Lambda à l'aide de l' AWS CLI.

REQUEST authorizer

La commande create-authorizer suivante crée un mécanisme d'autorisation `REQUEST`, et utilise l'en-tête `Authorizer` et la variable de contexte `accountId` 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

La commande create-authorizer suivante crée un mécanisme d'autorisation `TOKEN`, et utilise l'en-tête `Authorization` 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
```

Une fois que vous avez créé votre mécanisme d'autorisation Lambda, vous pouvez le tester. La commande test-invoke-authorizer suivante teste votre mécanisme d'autorisation Lambda :

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

Configuration d'une méthode pour utiliser un mécanisme d'autorisation Lambda (console)

Après avoir configuré votre mécanisme d'autorisation Lambda, vous devez l'attacher à une méthode pour votre API. Si votre mécanisme d'autorisation utilise la mise en cache des autorisations, assurez-vous de mettre à jour la politique afin qu'elle contrôle l'accès à la nouvelle méthode.

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 sélectionnez une nouvelle méthode ou une méthode existante.
4. Dans l'onglet Demande de méthode, sous Paramètres de requête 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. Pour Nom, saisissez le nom de l'en-tête qui correspond au nom Source du jeton que vous avez spécifié lorsque vous avez créé le mécanisme d'autorisation 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 dans 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.

Configurez une méthode pour utiliser un autorisateur Lambda (AWS CLI)

Après avoir configuré votre mécanisme d'autorisation Lambda, vous devez l'attacher à une méthode pour votre API. Vous pouvez créer une méthode ou utiliser une opération de correctif pour associer un mécanisme d'autorisation à une méthode existante. Si votre mécanisme d'autorisation utilise la mise en cache des autorisations, assurez-vous de mettre à jour la politique afin qu'elle contrôle l'accès à la nouvelle méthode.

La commande put-method suivante crée une méthode utilisant un mécanisme d'autorisation 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 met à jour une méthode existante afin qu'elle utilise un mécanisme d'autorisation 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 API Gateway

La section suivante explique le format de l'entrée envoyée par API Gateway à un mécanisme d'autorisation Lambda.

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 **`de saisie de la DEMANDE`**

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 API-dependent.

API Gateway peut ajouter de nouvelles clés au mappage. Pour plus d'informations sur l'entrée de la fonction Lambda dans l'intégration de proxy Lambda, consultez Format d'entrée d'une fonction Lambda pour l'intégration de proxy.

Sortie d'un mécanisme d'autorisation 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 politique (`policyDocument`) contenant la liste des déclarations de politique. 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.

JSON

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

```
      {
        "Action": "execute-api:Invoke",
        "Effect": "Allow|Deny",
        "Resource": "arn:aws:execute-
api:{regionId}:{accountId}:{apiId}/{stage}/{httpVerb}/[{resource}/[{child-
resources}]]"
      }
    ]
  },
  "context": {
    "stringKey": "value",
    "numberKey": "1",
    "booleanKey": "true"
  },
  "usageIdentifierKey": "{api-key}"
}
```

Dans cet exemple, une déclaration de politique indique s'il faut permettre ou interdire (`Effect`) au service d'exécution API Gateway d'appeler (`Action`) la méthode d'API spécifiée (`Resource`). Il se peut que vous deviez contrôler l'accès à plusieurs ressources en fonction de votre mécanisme d'autorisation. Vous pouvez utiliser un caractère générique (*) pour spécifier un type de ressource (méthode). Pour plus d'informations sur le paramétrage des politiques valides pour appeler une API, consultez Référence de déclaration de politique 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 politique 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 de plus amples informations, veuillez consulter Variables context pour les transformations de données.

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 de plus amples informations, veuillez consulter 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`).

JSON

```
{
  "principalId": "user",
  "policyDocument": {
    "Version":"2012-10-17",
    "Statement": [
      {
        "Action": "execute-api:Invoke",
        "Effect": "Deny",
        "Resource": "arn:aws:execute-api:us-west-2:123456789012:ymy8tbxw7b/dev/
GET/"
      }
    ]
  }
```

```
}
```

Appel d'une API avec un mécanisme 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.

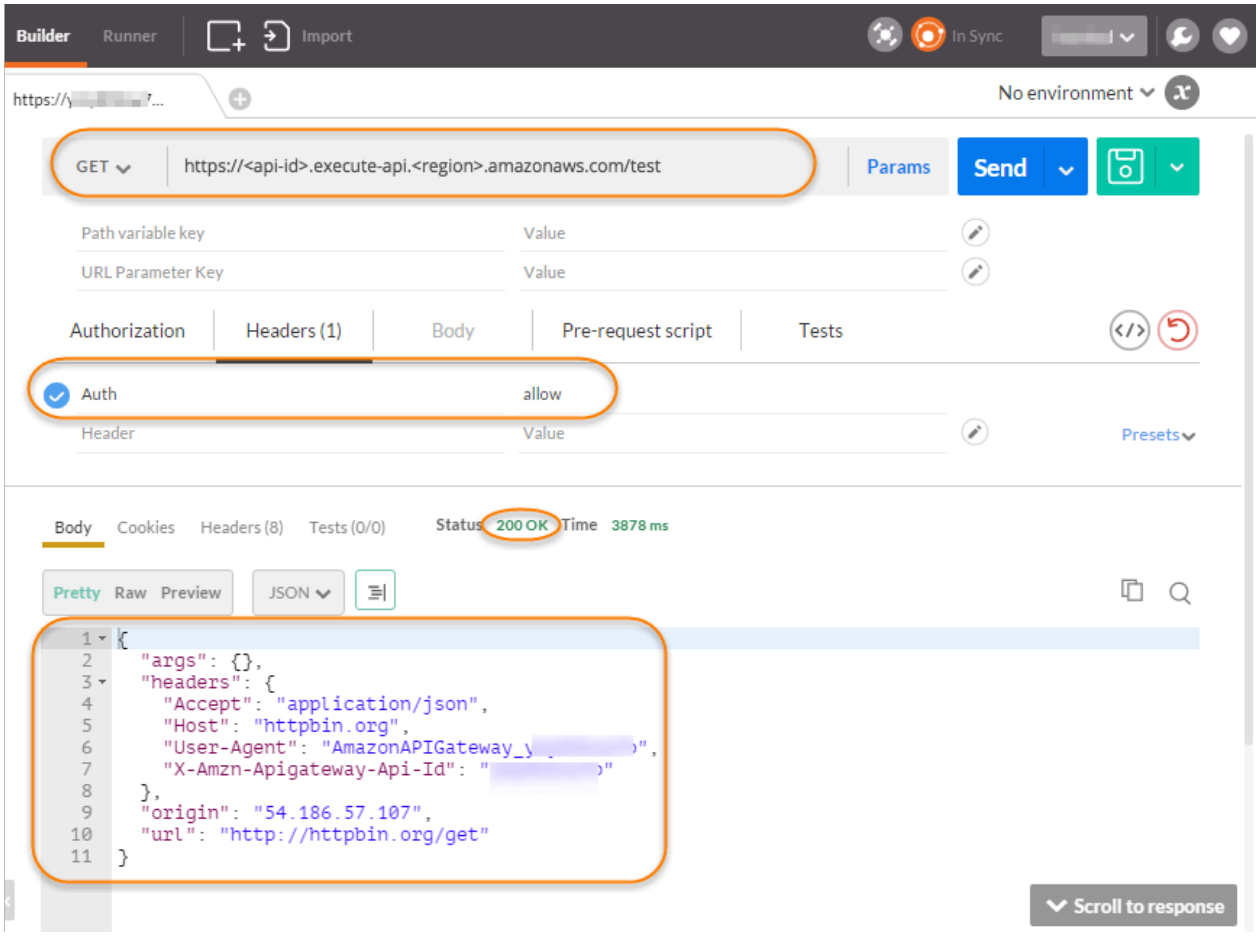
> **Note**
>
> Lors de l'appel d'une méthode activée par l'autorisateur, API Gateway n'enregistre pas l'appel CloudWatch si le jeton requis pour l'**`TOKEN`**autorisateur n'est pas défini, est nul ou est invalidé par l'expression de validation du jeton spécifiée. De même, API Gateway n'enregistre pas 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 l'autorisateur **`TOKEN`** personnalisé

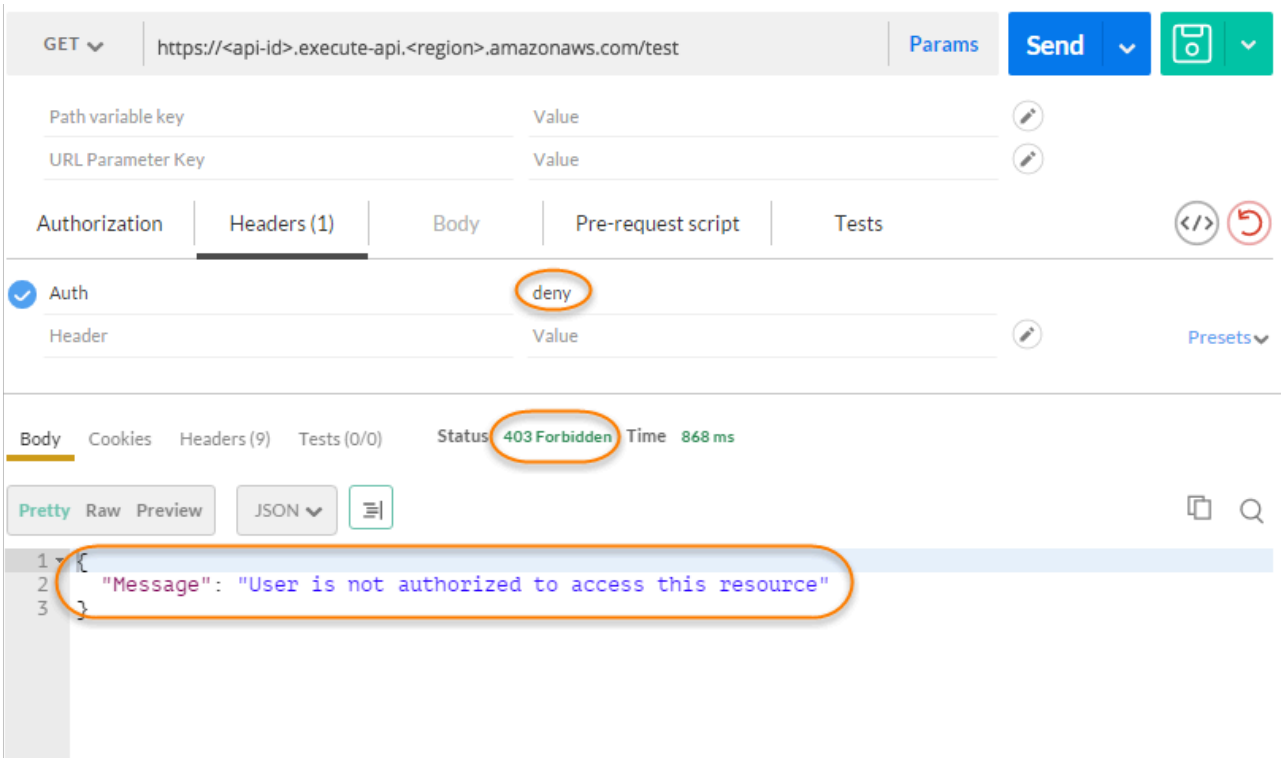
1. Ouvrez Postman, sélectionnez la méthode GET et collez la valeur du champ Invoke URL de l'API dans le champ d'URL adjacent.

   Ajoutez l'en-tête de jeton d'autorisation Lambda et définissez la valeur `allow`. Sélectionnez Send (Envoyer).



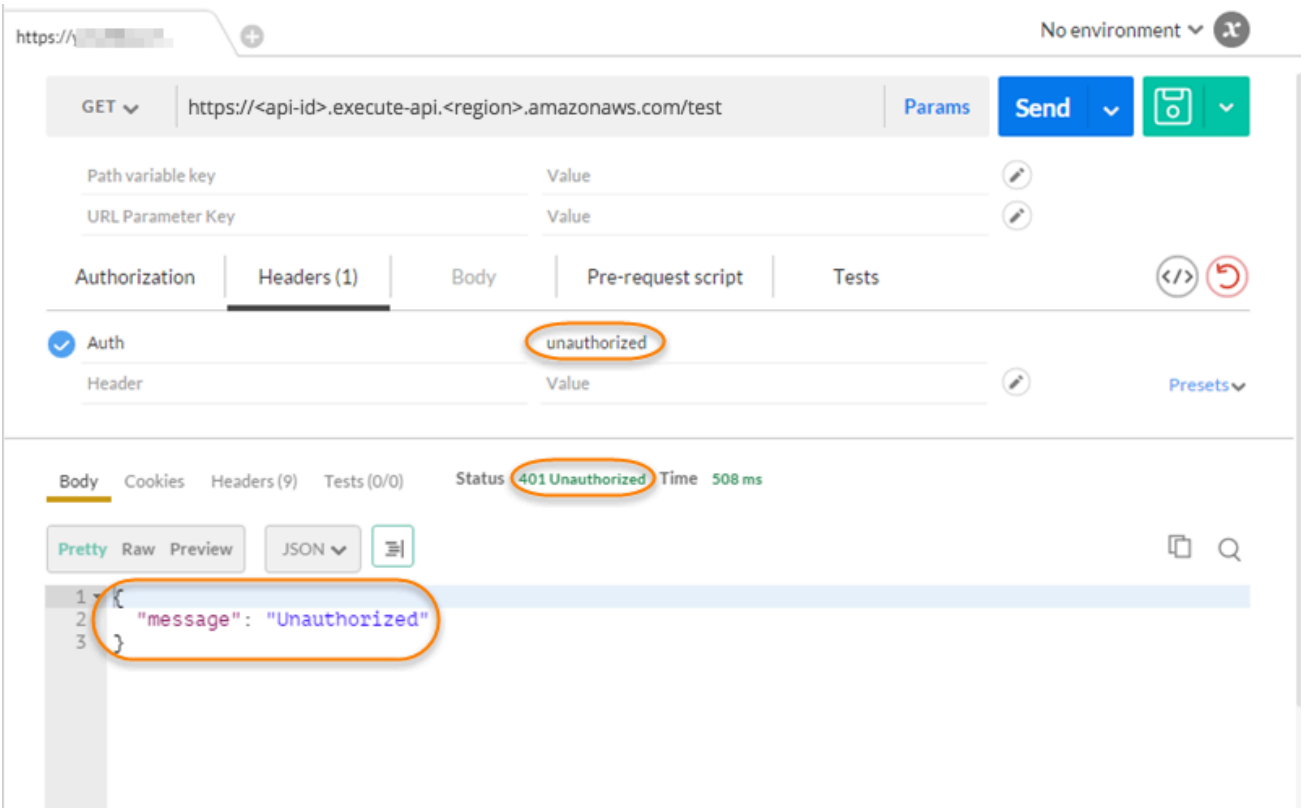
La réponse indique que l'autorisateur Lambda d'API Gateway renvoie une réponse de 200 OK et autorise avec succès l'appel pour accéder au point de terminaison HTTP (http://httpbin.org/get) intégré à la méthode.

2. Toujours dans l’application Postman, remplacez la valeur de l’en-tête de jeton d’autorisation Lambda par `deny`. Sélectionnez Send (Envoyer).



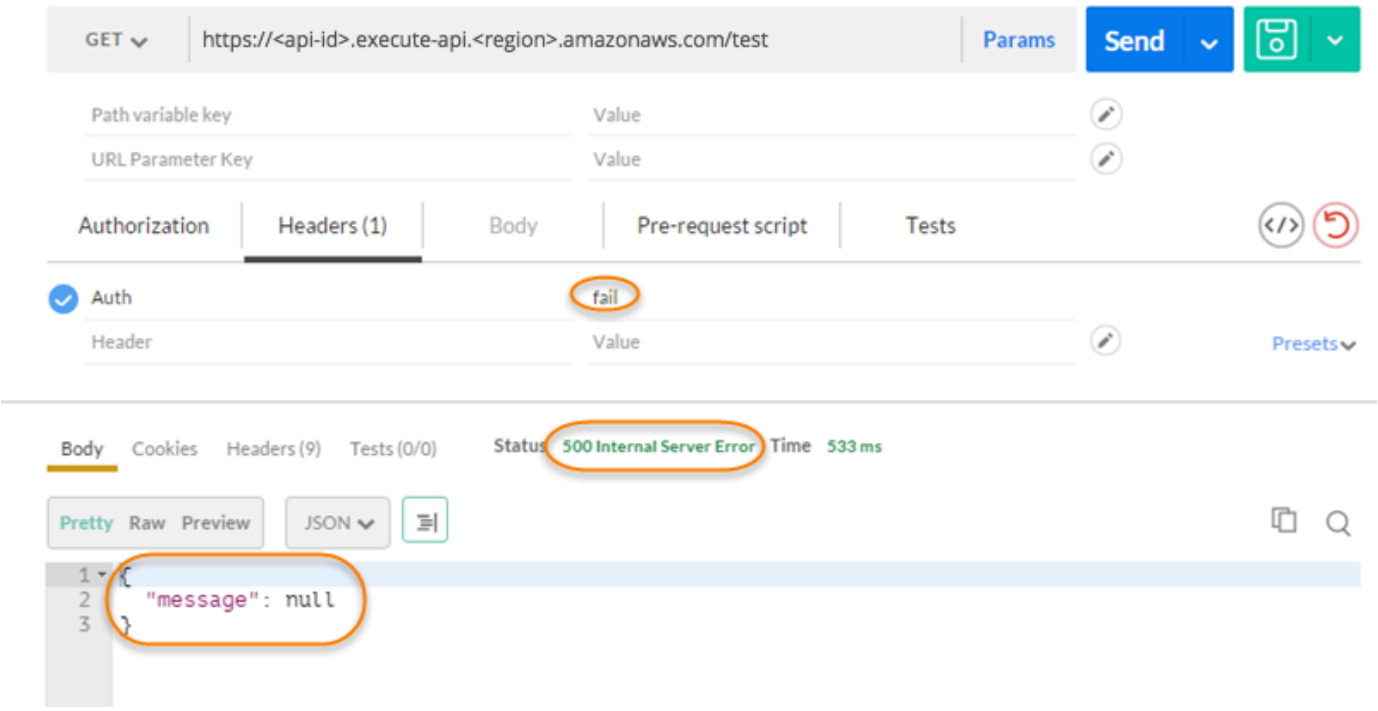
La réponse montre que le mécanisme d'autorisation Lambda API Gateway renvoie une réponse 403 Forbidden (403 Interdit) sans autoriser l'appel à accéder au point de terminaison HTTP.

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 API Gateway 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 politiques 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. Pour Charge utile d'événement Lambda, sélectionnez Jeton pour un mécanisme d'autorisation `TOKEN` ou Effectuer une demande pour un mécanisme d'autorisation `REQUEST`.
8. 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 politique 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 politique 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 politique 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 politique applicable à toutes les méthodes d'une API. Pour appliquer une politique spécifique à la méthode, vous pouvez désactiver Mise en cache des autorisations.

9. Choisissez Créer un mécanisme d'autorisation.
10. 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 sur la base des attributs d'une identité avec Verified Permissions

Utilisez Amazon Verified Permissions pour contrôler l'accès à votre API API Gateway. Lorsque vous utilisez API Gateway avec Verified Permissions, Verified Permissions crée un mécanisme d'autorisation Lambda qui utilise des décisions d'autorisation précises pour contrôler l'accès à votre API. Verified Permissions autorise les appelants sur la base d'un schéma d'entrepôt de politiques et de politiques utilisant le langage de politique Cedar pour définir des autorisations précises pour les utilisateurs de l'application. Pour plus d'informations, consultez Create a policy store with a connected API and identity provider dans le Guide de l'utilisateur Amazon Verified Permissions.

Verified Permissions prend en charge les groupes d'utilisateurs Amazon Cognito ou les fournisseurs d'identité OpenID Connect (OIDC) comme sources d'identité. Verified Permissions présume que le principal a déjà été identifié et authentifié. Verified Permissions n'est pris en charge que pour les API REST régionales et optimisées pour la périphérie.

Création d'un mécanisme d'autorisation Lambda à l'aide de Verified Permissions

Verified Permissions crée un mécanisme d'autorisation Lambda pour déterminer si un principal est autorisé à effectuer une action sur votre API. Vous allez créer la politique Cedar que Verified Permissions utilisera pour effectuer ses tâches d'autorisation.

Voici un exemple de politique Cedar qui autorise l'invocation d'une API sur la base du groupe d'utilisateurs Amazon Cognito `us-east-1_ABC1234` pour le groupe `developer` sur la ressource

`GET /users` d'une API. Verified Permissions détermine l'appartenance du groupe en recherchant l'identité de l'appelant dans le jeton de porteur.

```
permit(
  principal in MyAPI::UserGroup::"us-east-1_ABC1234|developer",
  action in [ MyAPI::Action::"get /users" ],
  resource
  );
```

Verified Permissions peut éventuellement attacher le mécanisme d'autorisation aux méthodes de votre API. Pour les étapes de production de votre API, nous vous recommandons de ne pas autoriser Verified Permissions à attacher le mécanisme d'autorisation à votre place.

La liste suivante indique comment configurer Verified Permissions pour qu'il attache ou non le mécanisme d'autorisation Lambda à la demande de méthode envoyée à votre API.

Verified Permissions attache le mécanisme d'autorisation pour vous (Console de gestion AWS)

Lorsque vous choisissez Créer un entrepôt de politiques dans la console Verified Permissions, sur la page Déployer l'intégration d'application, sélectionnez Maintenant.

Verified Permissions n'attache pas le mécanisme d'autorisation pour vous (Console de gestion AWS)

Lorsque vous choisissez Créer un entrepôt de politiques dans la console Verified Permissions, sur la page Déployer l'intégration d'application, sélectionnez Plus tard.

Verified Permissions crée tout de même un mécanisme d'autorisation Lambda pour vous. Le mécanisme d'autorisation Lambda commence par `AVPAuthorizerLambda-`. Pour apprendre à attacher votre mécanisme d'autorisation à une méthode, consultez the section called "Configuration d'une méthode pour utiliser un mécanisme d'autorisation Lambda (console)".

Verified Permissions attache le mécanisme d'autorisation pour vous (CloudFormation)

Dans le Permissions-generated CloudFormation modèle vérifié, dans la `Conditions` section, définissez `"Ref": "shouldAttachAuthorizer"` sur`true`.

Verified Permissions n'attache pas le mécanisme d'autorisation pour vous (CloudFormation)

Dans le Permissions-generated CloudFormation modèle vérifié, dans la `Conditions` section, définissez `"Ref": "shouldAttachAuthorizer"` sur`false`.

Verified Permissions crée tout de même un mécanisme d'autorisation Lambda pour vous. Le mécanisme d'autorisation Lambda commence par `AVPAuthorizerLambda-`. Pour apprendre à

attacher votre mécanisme d'autorisation à une méthode, consultez the section called "Configurez une méthode pour utiliser un autorisateur Lambda (AWS CLI)".

Appel d'un mécanisme d'autorisation Lambda à l'aide de Verified Permissions

Vous pouvez appeler votre mécanisme d'autorisation Lambda en fournissant une identité ou un jeton d'accès dans l'en-tête `Authorization`. Pour de plus amples informations, veuillez consulter the section called "Appel d'une API avec des mécanismes d'autorisation Lambda".

API Gateway met en cache la politique renvoyée par votre mécanisme d'autorisation Lambda pendant 120 secondes. Vous pouvez modifier la TTL dans la console API Gateway ou à l'aide de l' AWS CLI.

Contrôle de l'accès aux API REST à l'aide de groupes d'utilisateurs Amazon Cognito en tant que mécanisme d'autorisation

Au lieu d'utiliser des rôles et politiques 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 plus d'informations, consultez 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 ou l'API pour créer un groupe d'utilisateurs CLI/SDK, ou utilisez-en un appartenant à un autre compte. AWS
- Utilisez la console API Gateway ou l'API pour créer un autorisateur API Gateway avec le groupe d'utilisateurs choisi. CLI/SDK
- Utilisez la console API Gateway ou l' CLI/SDKAPI pour activer l'autorisateur 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 Amazon Cognito CLI/SDK ou l'API pour connecter un utilisateur au 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 avec le groupe d'utilisateurs et appeler une API intégrée au groupe d'utilisateurs.

Rubriques

- Création d'un groupe d'utilisateurs Amazon Cognito pour une API REST
- Intégration d'une API REST avec un groupe d'utilisateurs Amazon Cognito

- Appel d’une API REST intégrée avec 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 CloudFormation

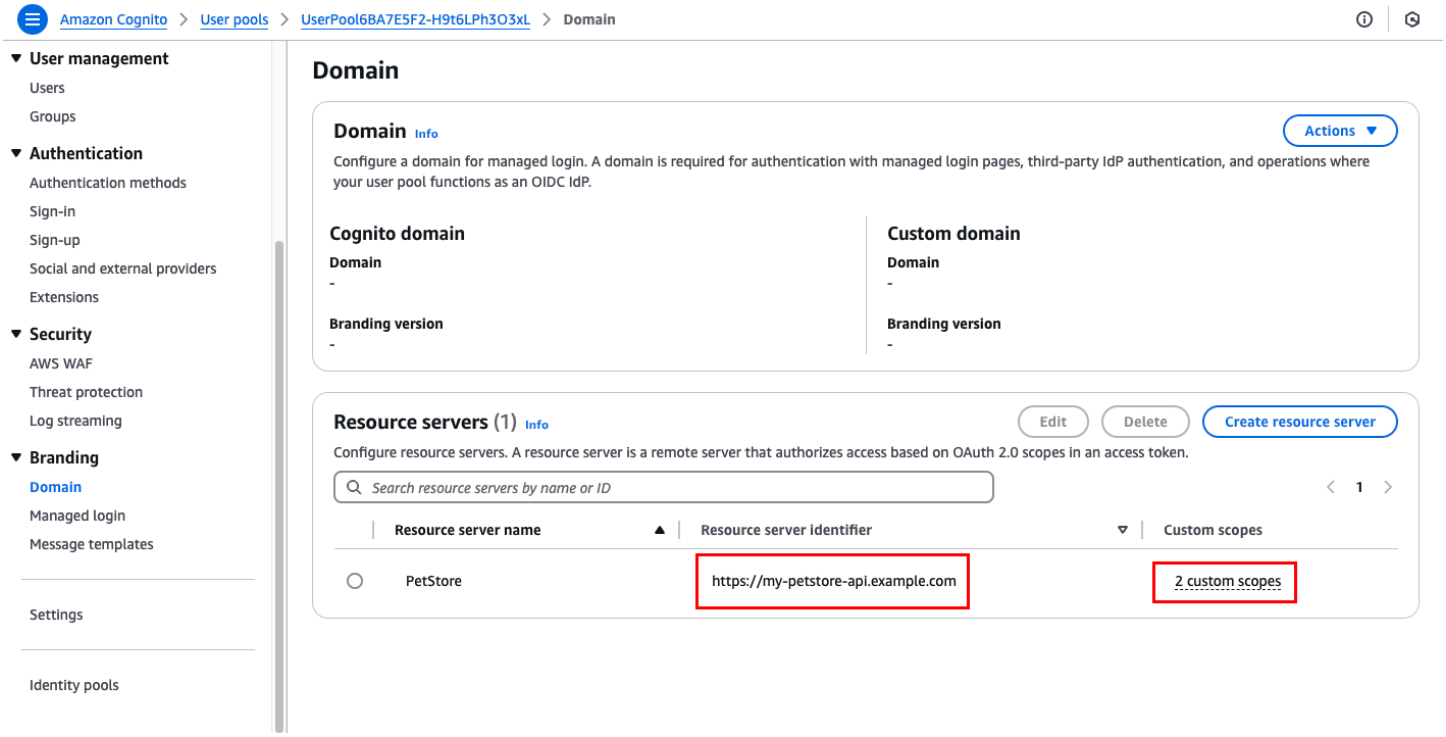
Création d’un groupe d’utilisateurs Amazon Cognito pour une API REST

Avant d’intégrer votre API avec 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 plus d’informations sur l’utilisation de jetons avec des groupes d’utilisateurs Amazon Cognito, consultez Utilisation des jetons avec les groupes d’utilisateurs. Pour plus d’informations sur les serveurs de ressources, consultez 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 avec 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 API Gateway.

Pour créer un autorisateur **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 essayer de l’invoquer en fournissant un jeton d’identité alloué à partir du groupe d’utilisateurs. Vous ne pouvez pas utiliser de jeton d’accès pour essayer d’invoquer votre mécanisme d’autorisation.

   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 autorisateur **`COGNITO_USER_POOLS`** sur des 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 de plus amples informations, veuillez consulter Extensions OpenAPI pour API Gateway.

Appel d'une API REST intégrée avec 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 avec des 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 politiques 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.

Création d’un mécanisme d’autorisation Amazon Cognito entre comptes pour une API REST

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 CloudFormation

Vous pouvez l'utiliser CloudFormation pour créer un groupe d'utilisateurs Amazon Cognito et un autorisateur Amazon Cognito. L'exemple de 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
          - /
```

Intégrations pour REST APIs dans API Gateway

Après avoir configuré une méthode d'API, vous devez l'intégrer avec 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
- Intégrations Lambda pour les API REST dans API Gateway
- Intégrations HTTP pour REST APIs dans API Gateway
- Diffusez la réponse d'intégration pour vos intégrations de proxy dans API Gateway
- Intégrations privées pour REST APIs dans API Gateway
- Intégrations fictives pour REST APIs 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 simulées, 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
- Choix d'un type d'intégration d'API API Gateway
- Configuration de l'intégration de proxy avec une ressource 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 the section called "Mappage de paramètres".

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 politique 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. Pour de plus amples informations, veuillez consulter the section called "Transformations de données".

Choix d'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 simulée, 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 simulée, 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 simulée. Les réponses de passerelle sont d'autres exemples d'intégrations simulées.

Configuration de l'intégration de proxy avec une ressource proxy

Pour configurer une intégration de proxy dans une API API Gateway avec une ressource proxy, exécutez les tâches suivantes :

- Créer une ressource proxy avec une variable de chemin gourmande `{proxy+}`.
- Définir la méthode `ANY` sur la ressource proxy.
- Intégrer la ressource et la méthode avec 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 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 en savoir plus, 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 proxy est plus puissante lorsqu'elle est intégrée avec 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 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 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 proxy `/api/{proxy+}` avec les 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 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 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 avec le 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

- Configuration d’une intégration Lambda
- Configuration d’une intégration HTTP
- Configuration d'une intégration AWS de services
- Configuration d’une intégration simulée

Configuration d’une intégration Lambda

Utilisez une intégration de fonction Lambda pour intégrer votre API avec une fonction Lambda. Au niveau de l’API, il s’agit d’un type d’intégration `AWS` si vous créez une intégration autre que de 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 requête de méthode à l'aide des menus déroulants suivants. Choisissez Paramètres de requête de méthode et configurez votre demande de méthode. Pour plus d'informations, consultez l'étape 3 de la page the section called "Modification d'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 avec 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 Méthode HTTP, 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 requête de méthode à l'aide des menus déroulants suivants. Choisissez Paramètres de requête de méthode et configurez votre

demande de méthode. Pour plus d'informations, consultez l'étape 3 de la page the section called "Modification d'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. Pour Type d'intégration, choisissez Service AWS .
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 Méthode HTTP, 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 avec le nombre souhaité de déclarations d'actions et de ressources. Pour de plus amples informations, veuillez consulter the section called "Fonctionnement d'Amazon API Gateway avec IAM".

    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 :

    JSON

    ```
    {
      "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 requête de méthode à l'aide des menus déroulants suivants. Choisissez Paramètres de requête de méthode et configurez votre

demande de méthode. Pour plus d'informations, consultez l'étape 3 de la page the section called "Modification d'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.

Configuration d'une intégration simulée

Utilisez une configuration simulée si vous souhaitez qu'API Gateway agisse en tant que 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 simulée

1. Dans le volet Ressources, choisissez Créer une méthode.
2. Pour Type de méthode, sélectionnez une méthode HTTP.
3. Pour Type d'intégration, sélectionnez Mock.
4. (Facultatif) Vous pouvez configurer les paramètres de requête de méthode à l'aide des menus déroulants suivants. Choisissez Paramètres de requête de méthode et configurez votre demande de méthode. Pour plus d'informations, consultez l'étape 3 de la page the section called "Modification d'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 plus d'informations sur la réponse d'erreur de la fonction Lambda, consultez 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 de plus amples informations, veuillez consulter Remplacement des codes d'état et des paramètres de demande et de réponse des API REST dans API Gateway.

Intégrations Lambda pour les API REST 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

- Intégrations de proxy Lambda dans API Gateway
- Configuration d'intégrations personnalisées Lambda dans API Gateway
- Configuration d'un appel asynchrone de la fonction Lambda du backend
- Gestion des erreurs Lambda dans API Gateway

Intégrations de proxy Lambda dans API Gateway

La section suivante explique comment utiliser une intégration de proxy Lambda.

Rubriques

- Présentation de l’intégration de proxy Lambda API Gateway
- Prise en charge des en-têtes à valeurs multiples et des paramètres de chaîne de requête
- Format d’entrée d’une fonction Lambda pour l’intégration de proxy
- Format de sortie d’une fonction Lambda pour l’intégration de proxy
- Configurez l'intégration du proxy Lambda pour API Gateway à l'aide du AWS CLI
- Configuration d’une ressource proxy avec une intégration de proxy Lambda à l’aide d’une définition OpenAPI

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 proxy générique. La ressource 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 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 personnalisées : `/{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 permettre à un superviseur de mettre à jour l'inventaire du `produce` département, vous pouvez configurer une autre méthode unique `PUT /produce/{proxy+}` avec des read/write autorisations. Pour permettre à un caissier de mettre à jour le total cumulé d'un légume, vous pouvez configurer une `POST /produce/vegetables/{proxy+}` méthode avec des read/write autorisations. Pour permettre à un responsable de boutique d'effectuer toutes les actions possibles sur n'importe quel produit disponible, le développeur de la boutique en ligne peut exposer la `ANY /{proxy+}` méthode avec read/write des autorisations. 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 plus d'informations sur la configuration des intégrations de proxy API Gateway, consultez Configuration de l'intégration de proxy avec une ressource 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 en charge plusieurs en-têtes et paramètres de chaîne de requête portant le même nom. Multi-value les en-têtes ainsi que 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 de proxy et Format de sortie d'une fonction Lambda pour l'intégration de proxy.

Format d'entrée d'une fonction Lambda pour l'intégration de 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.

Dans cet exemple, nous supposons que l'invocation d'API Gateway était la suivante :

```
curl 'https://a1b2c3.execute-api.us-east-1.amazonaws.com/my/path?
parameter1=value1&parameter2=value1&parameter2=value2&parameter3=value1,value2'
 -H 'header1: value1' -H 'header2: value1' -H 'header2: value2' -H 'header3:
 value1,value2'
```

Le résultat se présente comme suit :

```
{
  "resource": "/my/path",
  "path": "/my/path",
  "httpMethod": "GET",
  "headers": {
      "header1": "value1",
      "header2": "value2",
      "header3": "value1,value2"
  },
  "multiValueHeaders": {
    "header1": ["value1"],
    "header2": ["value1","value2"],
    "header3": ["value1,value2"]
  },
  "queryStringParameters": {
      "parameter1": "value1",
      "parameter2": "value2",
      "parameter3": "value1,value2"
  },
  "multiValueQueryStringParameters": {
```

```
    "parameter1": ["value1"],
    "parameter2": ["value1","value2"],
    "parameter3": ["value1,value2"]
  },
  "requestContext": {
    "accountId": "123456789012",
    "apiId": "id",
    "authorizer": {
      "claims": null,
      "scopes": null
    },
    "domainName": "id.execute-api.us-east-1.amazonaws.com",
    "domainPrefix": "id",
    "extendedRequestId": "request-id",
    "httpMethod": "GET",
    "identity": {
      "accessKey": null,
      "accountId": null,
      "caller": null,
      "cognitoAuthenticationProvider": null,
      "cognitoAuthenticationType": null,
      "cognitoIdentityId": null,
      "cognitoIdentityPoolId": null,
      "principalOrgId": null,
      "sourceIp": "IP",
      "user": null,
      "userAgent": "user-agent",
      "userArn": null,
      "clientCert": {
        "clientCertPem": "CERT_CONTENT",
        "subjectDN": "www.example.com",
        "issuerDN": "Example issuer",
        "serialNumber": "a1:a1:a1:a1:a1:a1:a1:a1:a1:a1:a1:a1:a1:a1:a1:a1",
        "validity": {
          "notBefore": "May 28 12:30:02 2019 GMT",
          "notAfter": "Aug  5 09:36:04 2021 GMT"
        }
      }
    },
    "path": "/my/path",
    "protocol": "HTTP/1.1",
    "requestId": "id=",
    "requestTime": "04/Mar/2020:19:15:17 +0000",
    "requestTimeEpoch": 1583349317135,
```

```
    "resourceId": null,
    "resourcePath": "/my/path",
    "stage": "$default"
  },
  "pathParameters": null,
  "stageVariables": null,
  "body": "Hello from Lambda!",
  "isBase64Encoded": false
}
```

> **Note**
>
> 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 au mappage.

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 de 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. S'il s'`body`agit d'un blob binaire, vous pouvez l'encoder sous forme de Base64-encoded chaîne en le définissant `true` et `isBase64Encoded` en le configurant `*/*` comme type de média 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, consultez Activation de la prise en charge binaire à l'aide de la console API Gateway. Pour obtenir un exemple de fonction Lambda, consultez Renvoi d'un support binaire d'une intégration de proxy Lambda dans API Gateway.

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 `async` fonction 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.

Configurez l'intégration du proxy Lambda pour API Gateway à l'aide du AWS CLI

Dans cette section, nous vous expliquons comment configurer une API avec l'intégration de proxy Lambda à l'aide de l' AWS CLI. Pour obtenir des instructions détaillées sur l'utilisation de la console API Gateway afin de configurer une ressource proxy avec l'intégration de proxy Lambda, consultez Didacticiel : création d'une API REST avec une intégration de proxy Lambda.

À titre d'exemple, nous utilisons l'exemple suivant de fonction Lambda comme backend de l'API :

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

Si l'on compare ce code à la configuration de l'intégration personnalisée Lambda dans the section called " Configuration d'intégrations personnalisées Lambda ", l'entrée de 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. Utilisez la commande create-rest-api suivante pour créer une API :

```
aws apigateway create-rest-api --name 'HelloWorld (AWS CLI)'
```

   Le résultat se présente comme suit :

```
{
    "name": "HelloWorldProxy (AWS CLI)",
    "id": "te6si5ach7",
    "rootResourceId" : "krznpq9xpg",
    "createdDate": 1508461860
}
```

   Vous allez utiliser l'API `id` (`te6si5ach7`) et `rootResourceId` (`krznpq9xpg`) tout au long de cet exemple.

2. Utilisez la commande create-resource suivante pour créer une ressource API Gateway de `/greeting` :

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

   Le résultat se présente comme suit :

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

   Vous allez utiliser la valeur `id` (`2jf6xt`) de la ressource `{proxy+}` pour créer une méthode sur la ressource `/{proxy+}` à l'étape suivante.

3. Utilisez la commande put-method suivante pour créer une demande de méthode `ANY ANY /{proxy+}` :

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

Le résultat se présente comme 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.

4. Utilisez la commande put-integration suivante 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 \
      --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 politiques 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".

Le résultat se présente comme 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 fournir un rôle IAM pour `credentials`, vous pouvez utiliser la commande add-permission pour ajouter des autorisations basées sur les ressources. C'est ce que fait la console API Gateway.

5. Utilisez la commande create-deployment suivante pour déployer l'API sur une étape `test` :

```
aws apigateway create-deployment  \
      --rest-api-id te6si5ach7 \
      --stage-name test
```

6. 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!
```

Configuration d'une ressource proxy avec une intégration de proxy Lambda à l'aide d'une définition OpenAPI

Pour configurer une ressource 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 avec 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 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 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 de 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 de proxy.

Avec l'intégration de proxy Lambda d'une ressource proxy via la méthode `ANY`, la fonction Lambda du backend unique sert de gestionnaire d'événements pour toutes les demandes via la ressource 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 proxy. La fonction Lambda du backend peut ensuite analyser le chemin d'URL et insérer les tuples d'emplacement dans une table DynamoDB.

Configuration d'intégrations personnalisées Lambda dans API Gateway

Pour montrer comment configurer l'intégration Lambda personnalisée ou sans proxy, nous créons une API API Gateway pour exposer la `GET /greeting?greeter={name}` méthode permettant d'invoquer 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

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

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

```
  let city = event.city === undefined ? 'World' : event.city;
  let time = times.indexOf(event.time)<0 ? 'day' : event.time;
  let day = days.indexOf(event.day)<0 ? null : event.day;

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

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

  // Return a greeting to the caller
  return greeting;
};
```

Python

```
import json


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

    if event['greeter'] == "":
        res['body'] = "Hello, World"
    elif (event['greeter']):
        res['body'] = "Hello, " + event['greeter'] + "!"
    else:
        raise Exception('Missing the required greeter parameter.')

    return res
```

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. Utilisez la commande create-rest-api suivante pour créer une API :

   ```
   aws apigateway create-rest-api --name 'HelloWorld (AWS CLI)'
   ```

   Le résultat se présente comme suit :

   ```
   {
       "name": "HelloWorld (AWS CLI)",
       "id": "te6si5ach7",
       "rootResourceId" : "krznpq9xpg",
       "createdDate": 1508461860
   }
   ```

   Vous allez utiliser l'API `id` (`te6si5ach7`) et `rootResourceId` (`krznpq9xpg`) tout au long de cet exemple.

2. Utilisez la commande create-resource suivante pour créer une ressource API Gateway de `/greeting` :

   ```
   aws apigateway create-resource \
         --rest-api-id te6si5ach7 \
         --parent-id krznpq9xpg \
         --path-part greeting
   ```

   Le résultat se présente comme suit :

   ```
   {
       "path": "/greeting",
   ```

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

Vous allez utiliser la valeur `id` (`2jf6xt`) de la ressource `greeting` pour créer une méthode sur la ressource `/greeting` à l'étape suivante.

3. Utilisez la commande put-method suivante pour créer une demande de méthode d'API `GET /greeting?greeter={name}` :

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

Le résultat se présente comme 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.

4. Utilisez la commande put-method-response suivante pour configurer la réponse `200 OK` à la demande de méthode `GET /greeting?greeter={name}` :

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

5. Utilisez la commande put-integration suivante 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 \
        --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. Le résultat se présente comme 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 politiques 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.

6. Utilisez la commande put-integration-response suivante 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 \
       --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.

Le résultat se présente comme suit :

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

7. Utilisez la commande create-deployment suivante pour déployer l'API sur une étape `test` :

```
aws apigateway create-deployment \
       --rest-api-id te6si5ach7 \
       --stage-name test
```

8. 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 Lambda autre que de proxy 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'appel asynchrone de Lambda à l'aide d'OpenAPI

Pour que tous les appels soient asynchrones :

- Ajoutez l'en-tête `X-Amz-Invocation-Type` à la section x-amazon-apigateway-integration.

```
"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'en-tête `X-Amz-Invocation-Type` à la section x-amazon-apigateway-integration.

```
"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.

Configurez l'invocation asynchrone Lambda à l'aide de CloudFormation

Les CloudFormation modèles suivants montrent comment configurer le `AWS::ApiGateway::Method` pour les 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

Le format AWS Lambda d'une erreur standard est le 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.

Par exemple, considérez 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 définissez ensuite un modèle d'expression régulière sur la `selectionPattern` propriété du code d'état HTTP IntegrationResponseassocié au code d'état 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, utilisez ce qui suit put-integration-responsepour configurer une nouvelle `selectionPattern` expression :

```
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. Si vous utilisez `.+` comme modèle de sélection pour filtrer les réponses, il se peut qu'il ne corresponde pas à une réponse contenant un caractère de nouvelle ligne (\n).

Pour mettre à jour une `selectionPattern` valeur existante à l'aide de AWS CLI, appelez l'update-integration-responseopération pour remplacer la valeur du `/selectionPattern` chemin par l'expression regex spécifiée du `Malformed*` modèle.

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, vous AWS Lambda permet de renvoyer un objet d'erreur personnalisé sous forme de chaîne JSON. L'erreur peut être n'importe quel objet JSON valide. Par exemple, la fonction Lambda JavaScript (Node.js) suivante 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. Toutefois, cette désérialisation s'applique uniquement aux body-to-header mappages pour les réponses d'erreur personnalisées Lambda et ne s'applique pas aux mappages utilisant. body-to-body `$input.body` Avec ces mappages custom-error-body-to -header, le client reçoit les en-têtes suivants dans le cadre de la réponse de méthode, à condition que les `error_type` en-têteserror_status, `error_traceerror_trace_function`, et soient 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.

Intégrations HTTP pour REST APIs dans API Gateway

Vous pouvez intégrer une méthode API avec 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 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 avec 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 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 proxy avec l'intégration HTTP, consultez Didacticiel : 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 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 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 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 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 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 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 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, également connue sous le nom d'intégration sans proxy, vous contrôlez davantage les données à transmettre entre une méthode d'API et une intégration d'API, et leur mode de transmission. 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. Vous allez ensuite configurer selectionPattern pour choisir une réponse d'intégration en fonction de la réponse du backend. Pour une intégration HTTP sans proxy, il s'agit d'une expression

régulière. Par exemple, pour mapper tous les codes d'état de réponse HTTP 2xx d'un point de terminaison HTTP à ce mappage de sortie, utilisez `2\d{2}`.

> **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.

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 plus d'informations sur la configuration des modèles de mappage, consultez Transformations de données pour les API REST dans API Gateway.

Diffusez la réponse d'intégration pour vos intégrations de proxy dans API Gateway

Vous pouvez configurer l’intégration de votre proxy pour contrôler comment API Gateway renvoie votre réponse d’intégration. Par défaut, API Gateway attend de recevoir la réponse complète avant de commencer la transmission. Toutefois, si vous définissez le mode de transfert des réponses de votre intégration sur`STREAM`, API Gateway n'attend pas qu'une réponse soit complètement calculée avant de l'envoyer au client. Le streaming des réponses fonctionne pour tous les types de points de terminaison de l’API REST.

Utilisez le streaming de réponses dans les cas d'utilisation suivants :

- Réduisez le délai d'obtention du premier octet (TTFB) pour les applications d'IA génératives telles que les chatbots.
- Diffusez de gros fichiers d'image, de vidéo ou de musique sans utiliser d'URL pré-signée S3.
- Réalisez des opérations de longue durée tout en signalant les progrès progressifs, tels que les événements envoyés par le serveur (SSE).
- Dépassez la limite de charge utile de réponse de 10 Mo d'API Gateway.
- Dépassez le délai d'expiration de 29 secondes d'API Gateway sans demander d'augmentation du délai d'intégration.
- Recevez une charge utile binaire sans configurer les types de supports binaires.

Considérations relatives au streaming de la charge utile de réponse

Les considérations suivantes peuvent avoir un impact sur votre utilisation du streaming de la charge utile de réponse :

- Vous ne pouvez utiliser le streaming de charge utile de réponse que pour les types d'`AWS_PROXY`intégration `HTTP_PROXY` ou d'intégration. Cela inclut les intégrations de proxy Lambda et les intégrations privées qui utilisent des intégrations. `HTTP_PROXY`
- Le paramètre du mode de transfert par défaut est`BUFFERED`. Pour utiliser le streaming de réponses, vous devez changer le mode de transfert de réponse sur`STREAM`.
- Le streaming des réponses n'est pris en charge que pour les API REST.
- Les demandes de streaming ne sont pas prises en charge.
- Vous pouvez diffuser votre réponse pendant 15 minutes maximum.

- Vos streams sont sujets à des délais d'inactivité. Pour les points de terminaison régionaux ou privés, le délai d'expiration est de 5 minutes. Pour les points de terminaison optimisés pour les bords, le délai d'expiration est de 30 secondes.
- Si vous utilisez le streaming de réponses pour une API REST régionale avec votre propre CloudFront distribution, vous pouvez atteindre un temps d'inactivité supérieur à 30 secondes en augmentant le délai de réponse de votre CloudFront distribution. Pour plus d'informations, consultez la section Délai de réponse.
- Lorsque le mode de transfert des réponses est défini sur`STREAM`, API Gateway ne peut pas prendre en charge les fonctionnalités qui nécessitent de mettre en mémoire tampon l'intégralité de la réponse d'intégration. Pour cette raison, les fonctionnalités suivantes ne sont pas prises en charge par le streaming de réponses :
  - Mise en cache des terminaux
  - Codage du contenu. Si vous souhaitez compresser votre réponse d'intégration, faites-le dans votre intégration.
  - Transformation des réponses avec VTL
- Dans chaque réponse de streaming, les 10 premiers Mo de charge utile de réponse ne sont soumis à aucune restriction de bande passante. Les données de charge utile de réponse supérieures à 10 Mo sont limitées à. 2MB/s
- Lorsque la connexion entre le client et API Gateway ou entre API Gateway et Lambda est fermée en raison d'un délai d'attente, la fonction Lambda peut continuer à s'exécuter. Pour plus d'informations, voir Configurer le délai d'expiration de la fonction Lambda.
- Le streaming des réponses entraîne un coût. Pour plus d’informations, consultez API Gateway Pricing (Tarification d’API Gateway).

Configurer une intégration de proxy HTTP avec le streaming des réponses de charge utile dans API Gateway

Lorsque vous configurez le streaming de la charge utile des réponses, vous spécifiez le mode de transfert des réponses dans la demande d'intégration de votre méthode. Vous configurez ces paramètres dans la demande d'intégration pour contrôler le comportement d'API Gateway avant et pendant la réponse d'intégration. Lorsque vous utilisez le streaming de réponses, vous pouvez configurer le délai d'intégration jusqu'à 15 minutes.

Lorsque vous utilisez le streaming de réponse de charge utile avec une `HTTP_PROXY` intégration, API Gateway n'envoie le code d'état de la réponse HTTP ni aucun en-tête de réponse HTTP tant qu'il n'a pas reçu tous les en-têtes complets.

Création d'une intégration de proxy HTTP avec le streaming de réponse à la charge utile

La procédure suivante explique comment importer une nouvelle API avec la valeur `responseTransferMode` définie sur`STREAM`. Si vous disposez d'une API d'intégration existante et que vous souhaitez la modifier`responseTransferMode`, consultezthe section called “Mettre à jour le mode de transfert des réponses pour une intégration de proxy HTTP”.

Console de gestion AWS

Pour créer une intégration de proxy HTTP avec le streaming de réponses aux charges utiles

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Choisissez une API REST.
3. Choisissez Créer une ressource.
4. Sous Resource Name (Nom de la ressource), entrez **`streaming`**.
5. Choisissez Créer une ressource.
6. Lorsque la ressource /streaming est sélectionnée, choisissez Create method.
7. Pour Type de méthode, choisissez ANY.
8. Pour Type d’intégration, choisissez HTTP.
9. Choisissez l'intégration du proxy HTTP.
10. Pour le mode de transfert Response, choisissez Stream.
11. Pour la méthode HTTP, choisissez une méthode.
12. Pour l'URL du point de terminaison, entrez un point de terminaison d'intégration. Assurez-vous de choisir un point de terminaison qui produit une charge utile importante à renvoyer vers vous.
13. Choisissez Créer une méthode.

Après avoir créé votre méthode, déployez 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 **`prod`**.
4. (Facultatif) Sous Description, entrez une description.
5. Choisissez Déployer.

AWS CLI

Pour créer une nouvelle API avec diffusion en continu de la réponse à la charge utile

1. Copiez le fichier Open API suivant, puis enregistrez-le sous`ResponseStreamDemoSwagger.yaml`. Dans ce fichier, `responseTransferMode` est défini sur`STREAM`. Le point de terminaison d'intégration est défini sur`https://example.com`, mais nous vous recommandons de le modifier pour un point de terminaison qui produit une charge utile importante à vous renvoyer.

```
openapi: "3.0.1"
info:
  title: "ResponseStreamingDemo"
  version: "2025-04-28T17:28:25Z"
servers:
- url: "{basePath}"
  variables:
    basePath:
      default: "prod"
paths:
  /streaming:
    get:
      x-amazon-apigateway-integration:
        httpMethod: "GET"
        uri: "https://example.com"
        type: "http_proxy"
        timeoutInMillis: 900000
        responseTransferMode: "STREAM"
```

2. Utilisez la `import-rest-api` commande suivante pour importer votre définition OpenAPI :

```
aws apigateway import-rest-api \
```

```
  --body 'fileb://~/ResponseStreamDemoSwagger.yaml' \
  --parameters endpointConfigurationTypes=REGIONAL \
  --region us-west-1
```

3. Utilisez la `create-deployment` commande suivante pour déployer votre nouvelle API sur une étape :

```
aws apigateway create-deployment \
  --rest-api-id a1b2c3 \
  --stage-name prod \
  --region us-west-1
```

Mettre à jour le mode de transfert des réponses pour une intégration de proxy HTTP

La procédure suivante montre comment mettre à jour le mode de transfert des réponses pour une intégration de proxy HTTP.

Console de gestion AWS

Pour mettre à jour le mode de transfert des réponses pour une intégration de proxy HTTP

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Choisissez une API REST.
3. Choisissez une méthode.
4. Dans l'onglet Requête d'intégration, sous Paramètres de requête d'intégration, choisissez Modifier.
5. Pour le mode de transfert Response, choisissez Stream.
6. Choisissez Enregistrer.

Après avoir mis à jour votre méthode, déployez 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 **`prod`**.

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

AWS CLI

La `update-integration` commande suivante met à jour le mode de transfert d'une intégration de `BUFFERED` vers`STREAM`. Pour toutes les API existantes, le mode de transfert de réponse pour toutes les intégrations est défini sur. `BUFFERED`

```
aws apigateway update-integration \
 --rest-api-id a1b2c3 \
 --resource-id aaa111 \
 --http-method GET \
 --patch-operations "op='replace',path='/responseTransferMode',value=STREAM" \
 --region us-west-1
```

Vous devez redéployer votre API pour que les modifications prennent effet. Si vous avez personnalisé le délai d'intégration, cette valeur de délai est supprimée, car API Gateway diffuse votre réponse pendant 5 minutes maximum.

La `update-integration` commande suivante met à jour le mode de transfert d'une intégration de `STREAM` vers `BUFFERED` :

```
aws apigateway update-integration \
 --rest-api-id a1b2c3 \
 --resource-id aaa111 \
 --http-method GET \
 --patch-operations "op='replace',path='/responseTransferMode',value=BUFFERED" \
 --region us-west-1
```

Vous devez redéployer votre API pour que les modifications prennent effet.

Configurer une intégration de proxy Lambda avec le streaming des réponses de charge utile dans API Gateway

Vous pouvez diffuser la réponse d'une fonction Lambda pour améliorer les performances du délai d'obtention du premier octet (TTFB) et renvoyer des réponses partielles au client dès qu'elles sont disponibles. API Gateway nécessite que vous utilisiez l'API InvokeWithResponseStreamLambda pour

appeler votre fonction Lambda. API Gateway transmet un objet d'événement à la fonction Lambda. La fonction Lambda du backend analyse les données de la demande entrante pour déterminer la réponse qu'elle renvoie. Pour qu'API Gateway diffuse la sortie Lambda, la fonction Lambda doit générer le format requis par API Gateway.

Différences entre les intégrations de proxy Lambda entre le mode de transfert de réponse en flux et le mode de transfert de réponse en mémoire tampon

La liste suivante décrit les différences entre une intégration de proxy Lambda et une intégration de proxy Lambda pour le streaming de réponses :

- API Gateway utilise l'InvokeWithResponseStreamAPI pour appeler l'intégration du proxy Lambda pour le streaming des réponses. Il en résulte un URI différent, qui est le suivant :

  ```
  arn:aws:apigateway:us-west-1:lambda:path/2021-11-15/functions/arn:aws:lambda:us-west-1:111122223333:function:my-function-name/response-streaming-invocations
  ```

  Cet ARN utilise une date différente pour la version de l'API et une action de service différente par rapport à l'intégration du proxy Lambda.

  Si vous utilisez la console API Gateway pour le streaming des réponses, la console utilise l'URI qui vous convient.
- Dans le cadre d'une intégration de proxy Lambda, API Gateway envoie la réponse au client uniquement après avoir reçu la réponse complète de Lambda. Dans le cadre d'une intégration de proxy Lambda pour le streaming des réponses, API Gateway lance le flux de charge utile après avoir reçu les métadonnées et le délimiteur valides de Lambda.
- L'intégration du proxy Lambda pour le streaming des réponses utilise le même format d'entrée que l'intégration du proxy, mais elle nécessite un format de sortie différent.

Format d'intégration du proxy Lambda pour le streaming des réponses

Lorsqu'API Gateway invoque une fonction Lambda avec flux de réponse, le format d'entrée est le même que celui d'une fonction Lambda pour l'intégration de proxy. Pour de plus amples informations, veuillez consulter the section called “Format d’entrée d’une fonction Lambda pour l’intégration de proxy”.

Lorsque Lambda diffuse une réponse à API Gateway, la réponse doit respecter le format suivant. Ce format utilise un délimiteur pour séparer les métadonnées JSON de la charge utile brute. Dans

ce cas, les données de charge utile sont diffusées telles qu'elles sont transmises par votre fonction Lambda de streaming :

```
{
  "headers": {"headerName": "headerValue", ...},
  "multiValueHeaders": { "headerName": ["headerValue", "headerValue2", ...], ... },
  "cookies" : ["cookie1", "cookie2"],
  "statusCode": httpStatusCode
}<DELIMITER>PAYLOAD1 | PAYLOAD2 | PAYLOAD3
```

Dans la sortie :

- Les `statusCode` touches `headersmultiValueHeaders`,`cookies`, et peuvent ne pas être spécifiées 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 sortie s'attend à ce que les en-têtes contiennent l'un `Transfer-Encoding: chunked` ou `Content-length: number` l'autre. Si votre fonction ne renvoie aucun de ces en-têtes, API Gateway les ajoute `Transfer-Encoding: chunked` à l'en-tête de réponse.
- 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.
- Les métadonnées doivent être au format JSON valide. Seulement `headersmultiValueHeaders`, `cookies` et les `statusCode` touches sont prises en charge.
- Vous devez fournir un délimiteur après le JSON des métadonnées. Le délimiteur doit être de 8 octets nuls et doit apparaître dans les 16 premiers Ko de données de flux.
- API Gateway ne nécessite pas de format spécifique pour la charge utile de réponse de la méthode.

Si vous utilisez une URL de fonction pour diffuser votre fonction Lambda, vous devez modifier l'entrée et la sortie de votre fonction Lambda pour répondre à ces exigences.

Si la sortie de votre fonction Lambda ne répond pas aux exigences de ce format, API Gateway peut toujours appeler votre fonction Lambda. Le tableau suivant présente les combinaisons de paramètres de demande d'intégration d'API et de code de fonction Lambda pris en charge par API Gateway. Cela inclut les combinaisons prises en charge pour le mode de transfert des réponses en mode tampon.

Mode de transfert de réponse	Le code de fonction est conforme au format requis	API d'appel Lambda	Pris en charge par API Gateway
Flux	Oui	InvokeWithResponseStream	Oui. API Gateway diffuse votre réponse.
Flux	Non	InvokeWithResponseStream	Non API Gateway appelle votre fonction Lambda et renvoie une réponse d'erreur de 500.
Flux	Oui	Invoke	Non API Gateway ne prend pas en charge cette configuration d'intégration.
Flux	Non	Invoke	Non API Gateway ne prend pas en charge cette configuration d'intégration.
Mis en mémoire tampon	Oui	InvokeWithResponseStream	Non API Gateway ne prend pas en charge cette configuration d'intégration.
Mis en mémoire tampon	Non	InvokeWithResponseStream	Non API Gateway ne prend pas en charge cette configuration d'intégration.
Mis en mémoire tampon	Oui	Invoke	API Gateway renvoie les en-têtes HTTP et le code d'état, mais pas le corps de la réponse.

Mode de transfert de réponse	Le code de fonction est conforme au format requis	API d'appel Lambda	Pris en charge par API Gateway
Mis en mémoire tampon	Non	Invoke	Oui. Il s'agit d'une intégration de proxy Lambda. Pour plus d'informations, consultez la section Intégration du proxy Lambda.

Configuration d'une intégration de proxy Lambda avec le streaming des réponses de charge utile dans API Gateway

Lorsque vous configurez le streaming de la charge utile de réponse, vous spécifiez le mode de transfert dans la demande d'intégration de votre ressource. Vous configurez ces paramètres dans la demande d'intégration pour contrôler le comportement d'API Gateway avant et pendant la réponse d'intégration.

Exemples de fonctions Lambda pour le streaming des réponses

Votre fonction Lambda doit respecter le. the section called “Format d'intégration du proxy Lambda pour le streaming des réponses” Nous vous recommandons d'utiliser l'un des trois exemples de fonctions Lambda pour tester le streaming des réponses. Lorsque vous créez votre fonction Lambda, veillez à effectuer les opérations suivantes :

- Prévoyez un délai d'attente suffisant pour votre fonction. Nous vous recommandons de configurer un délai d'au moins 1 minute pour en savoir plus sur le streaming des réponses. Lorsque vous créez vos ressources de production, assurez-vous que le délai d'expiration de votre fonction Lambda couvre l'ensemble du cycle de demande. Pour plus d'informations, voir Configurer le délai d'expiration de la fonction Lambda.
- Utilisez le dernier moteur d'exécution de Node.js.
- Utilisez une région dans laquelle le streaming des réponses Lambda est disponible.

Using HttpResponseStream.from

L'exemple de code suivant renvoie les objets de métadonnées JSON et les charges utiles au client à l'aide de la `awslambda.HttpResponseStream()` méthode sans utiliser la méthode du pipeline. Il n'est pas nécessaire de créer le délimiteur. Pour plus d'informations, consultez la section Écrire des fonctions Lambda compatibles avec le streaming des réponses.

```
export const handler = awslambda.streamifyResponse(
  async (event, responseStream, context) => {
    const httpResponseMetadata = {
      "statusCode": 200,
      "headers": {
        "x-foo": "bar"
      },
      "multiValueHeaders": {
        "x-mv1": ["hello", "world"],
        "Set-Cookie": ["c1=blue", "c2=red"]
      }
    };

    responseStream = awslambda.HttpResponseStream.from(responseStream,
 httpResponseMetadata);
    await new Promise(r => setTimeout(r, 1000)); // synthetic delay

    responseStream.write("First payload ");
    await new Promise(r => setTimeout(r, 1000)); // synthetic delay

    responseStream.write("Final payload");
    responseStream.end();
});
```

Using the pipeline method

Lambda recommande que lorsque vous écrivez des fonctions activées pour le streaming de réponses, vous utilisiez le `awslambda.streamifyResponse()` décorateur fourni par les environnements d'exécution natifs de Node.js, ainsi que la méthode. `pipeline()` Lorsque vous utilisez la méthode du pipeline, vous n'avez pas besoin de créer le délimiteur, Lambda le fait pour vous. Pour plus d'informations, consultez la section Écrire des fonctions Lambda compatibles avec le streaming des réponses.

L'exemple de code suivant renvoie les objets de métadonnées JSON et trois charges utiles au client.

```
import { pipeline } from 'node:stream/promises';
import { Readable } from 'node:stream';

export const handler = awslambda.streamifyResponse(
  async (event, responseStream, context) => {
    const httpResponseMetadata = {
      statusCode: 200,
      headers: {
        "Content-Type": "text/plain",
        "X-Custom-Header": "Example-Custom-Header"
      }
    };

    responseStream = awslambda.HttpResponseStream.from(responseStream,
 httpResponseMetadata);

    const dataStream = Readable.from(async function* () {
      yield "FIRST payload\n";
      await new Promise(r => setTimeout(r, 1000));
      yield "SECOND payload\n";
      await new Promise(r => setTimeout(r, 1000));
      yield "THIRD payload\n";
      await new Promise(r => setTimeout(r, 1000));
    }());

    await pipeline(dataStream, responseStream);
  }
);
```

Without using the pipeline method

L'exemple de code suivant renvoie les objets de métadonnées JSON et trois charges utiles au client sans utiliser la `awslambda.HttpResponseStream()` méthode. Sans `awslambda.HttpResponseStream()` cette méthode, vous devez inclure un délimiteur de 8 octets nuls entre les métadonnées et la charge utile.

```
export const handler = awslambda.streamifyResponse(async (event, response, ctx) => {
  response.write('{"statusCode": 200, "headers": {"hdr-x": "val-x"}}');
  response.write("\x00".repeat(8)); // DELIMITER
  await new Promise(r => setTimeout(r, 1000));

  response.write("FIRST payload");
```

```
  await new Promise(r => setTimeout(r, 1000));

  response.write("SECOND payload");
  await new Promise(r => setTimeout(r, 1000));

  response.write("FINAL payload");
  response.end();
});
```

Création d'une intégration de proxy Lambda avec diffusion en continu de la réponse à la charge utile

La procédure suivante montre comment créer une intégration de proxy Lambda avec le streaming de réponse de charge utile. Utilisez l'exemple de fonction Lambda ou créez la vôtre.

Console de gestion AWS

Pour créer une intégration de proxy Lambda avec le streaming de réponse à la charge utile

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Choisissez une API REST.
3. Choisissez Créer une ressource.
4. Sous Resource Name (Nom de la ressource), entrez **`streaming`**.
5. Choisissez Créer une ressource.
6. Lorsque la ressource /streaming est sélectionnée, choisissez Create method.
7. Pour Type de méthode, choisissez ANY.
8. Pour Type d'intégration, choisissez Lambda.
9. Choisissez l'intégration du proxy Lambda.
10. Pour le mode de transfert Response, choisissez Stream.
11. Pour la fonction Lambda, choisissez le nom de votre fonction Lambda.

    La console API Gateway utilise automatiquement InvokeWithResponseStreaml'API pour appeler la fonction Lambda. Vous êtes responsable de l'écriture d'une fonction Lambda compatible avec le streaming des réponses. Pour obtenir un exemple, consultez the section called “Exemples de fonctions Lambda pour le streaming des réponses”.
12. Choisissez Créer une méthode.

Après avoir créé votre méthode, déployez 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 **`prod`**.
4. (Facultatif) Sous Description, entrez une description.
5. Choisissez Déployer.

AWS CLI

La procédure suivante explique comment importer une nouvelle API avec la valeur `responseTransferMode` définie sur`STREAM`. Si vous disposez d'une API d'intégration existante et que vous souhaitez la modifier`responseTransferMode`, consultezthe section called "Mettre à jour le mode de transfert des réponses pour une intégration de proxy Lambda".

Pour créer une nouvelle API avec diffusion en continu de la réponse à la charge utile

1. Copiez le fichier Open API suivant, puis enregistrez-le sous`ResponseStreamDemoSwagger.yaml`. Dans ce fichier, `responseTransferMode` est défini sur`STREAM`, et l'URI d'intégration est défini sur`arn:aws:apigateway:us-west-1:lambda:path/2021-11-15/functions/arn:aws:lambda:us-west-1:111122223333:function:my-function-name/response-streaming-invocations`.

   Remplacez le nom de la fonction par une fonction compatible `my-function` avec le streaming et remplacez les informations d'identification par un rôle IAM doté de politiques permettant au `apigateway` service d'invoquer des fonctions Lambda.

```
openapi: "3.0.1"
info:
  title: "ResponseStreamingDemo"
  version: "2025-04-28T17:28:25Z"
servers:
- url: "{basePath}"
  variables:
    basePath:
      default: "prod"
```

```
paths:
  /lambda:
    get:
      x-amazon-apigateway-integration:
        httpMethod: "POST"
        uri: "arn:aws:apigateway:us-west-1:lambda:path/2021-11-15/functions/
arn:aws:lambda:us-west-1:111122223333:function:my-function-name/response-
streaming-invocations"
        type: "aws_proxy"
        timeoutInMillis: 90000
        responseTransferMode: "STREAM"
        credentials: "arn:aws:iam::111122223333:role/apigateway-lambda-role"
```

Au lieu de fournir un rôle IAM pour les informations d'identification, vous pouvez utiliser la `add-permission` commande Lambda pour ajouter des autorisations basées sur les ressources.

2. Utilisez la `import-rest-api` commande suivante pour importer votre définition OpenAPI :

```
aws apigateway import-rest-api \
  --body 'fileb://~/ResponseStreamDemoSwagger.yaml' \
  --parameters endpointConfigurationTypes=REGIONAL \
  --region us-west-1
```

3. Utilisez la `create-deployment` commande suivante pour déployer votre nouvelle API sur une étape :

```
aws apigateway create-deployment \
  --rest-api-id a1b2c2 \
  --stage-name prod \
  --region us-west-1
```

Mettre à jour le mode de transfert des réponses pour une intégration de proxy Lambda

La procédure suivante montre comment mettre à jour le mode de transfert des réponses pour une intégration de proxy Lambda. Lorsque vous passez du mode de transfert des réponses au streaming, mettez à jour votre fonction Lambda afin qu'elle soit conforme aux exigences du streaming des réponses. Utilisez l'exemple de fonction Lambda ou créez la vôtre.

Console de gestion AWS

Pour mettre à jour le mode de transfert des réponses pour une intégration de proxy Lambda

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Choisissez une API REST.
3. Choisissez une méthode.
4. Dans l'onglet Requête d'intégration, sous Paramètres de requête d'intégration, choisissez Modifier.
5. Pour le mode de transfert Response, choisissez Stream.
6. Pour la fonction Lambda, choisissez le nom de votre fonction Lambda.
7. Choisissez Enregistrer.

Après avoir mis à jour votre méthode, déployez 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 **`prod`**.
4. (Facultatif) Sous Description, entrez une description.
5. Choisissez Déployer.

AWS CLI

1. Mettez à jour votre fonction Lambda pour qu'elle soit compatible avec le streaming.
2. Utilisez la AWS CLI commande suivante pour mettre à jour l'URI d'intégration et le mode de transfert des réponses de votre intégration :

```
aws apigateway update-integration \
 --rest-api-id a1b2c3 \
 --resource-id aaa111 \
 --http-method ANY \
 --patch-operations "[{\"op\":\"replace\",\"path\":\"/uri\",\"value
\":\"arn:aws:apigateway:us-west-1:lambda:path/2021-11-15/functions/
```

```
arn:aws:lambda:us-west-1:111122223333:function:my-function-name/response-
streaming-invocations\"}, {\"op\":\"replace\",\"path\":\"/responseTransferMode
\",\"value\":\"STREAM\"}]" \
 --region us-west-1
```

3. Redéployez l'API pour que les modifications prennent effet.

Résoudre les problèmes liés au streaming des réponses dans API Gateway

Les conseils de dépannage suivants peuvent vous aider à résoudre les problèmes liés à vos API qui utilisent le streaming de réponses.

Résolution de problème généraux

Vous pouvez utiliser l'onglet de test TestInvokeMethodou l'onglet de test de la console pour tester la réponse de votre stream. Les considérations suivantes peuvent avoir un impact sur votre utilisation de test invoke pour le streaming des réponses :

- Lorsque vous testez votre méthode, API Gateway met en mémoire tampon votre charge utile de réponse diffusée en continu. Une fois que l'une des conditions suivantes est remplie, API Gateway renvoie une réponse unique contenant la charge utile mise en mémoire tampon :
  - La demande est complète
  - 35 secondes se sont écoulées
  - Plus de 1 Mo de charge utile de réponse ont été mis en mémoire tampon
- Si plus de 35 secondes s'écoulent avant que votre méthode ne renvoie un statut de réponse HTTP et tous les en-têtes, le statut de réponse renvoyé est 0. TestInvokeMethod
- API Gateway ne produit aucun journal d'exécution.

Après avoir déployé votre API, vous pouvez tester la réponse de votre flux à l'aide d'une commande curl. Nous vous recommandons d'utiliser l'`-i`option permettant d'inclure les en-têtes de réponse du protocole dans la sortie. Pour voir les données de réponse telles qu'elles arrivent, utilisez l'option curl `--no-buffer`

Résoudre les erreurs cURL

Si vous testez une intégration et que vous recevez le message d'erreur`curl: (18) transfer closed with outstanding read data remaining`, assurez-vous que le délai d'expiration de votre intégration est suffisamment long. Si vous utilisez une fonction Lambda, vous devez mettre

à jour le délai de réponse de la fonction Lambda. Pour plus d'informations, voir Configurer le délai d'expiration de la fonction Lambda.

Résoudre les problèmes liés à la journalisation des accès

Vous pouvez utiliser les journaux d'accès pour votre stage d'API REST afin de consigner et de dépanner votre flux de réponses. Outre les variables existantes, vous pouvez utiliser les variables du journal d'accès suivantes :

`$context.integration.responseTransferMode`

Le mode de transfert des réponses de votre intégration. Il peut correspondre à `BUFFERED` ou `STREAMED`.

`$context.integration.timeToAllHeaders`

Le délai entre le moment où API Gateway établit la connexion d'intégration et le moment où il reçoit tous les en-têtes de réponse d'intégration du client.

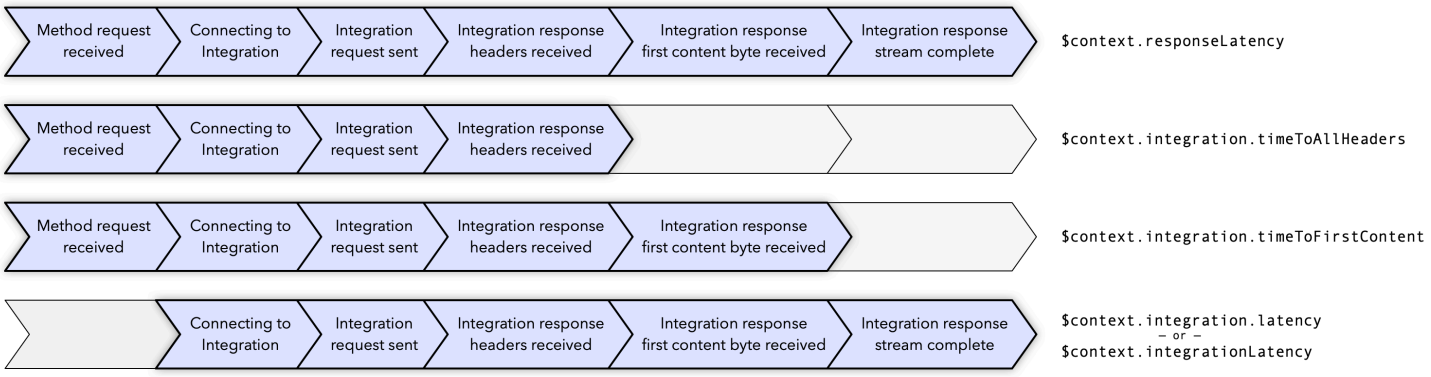
`$context.integration.timeToFirstContent`

Le délai entre le moment où API Gateway établit la connexion d'intégration et le moment où il reçoit les premiers octets de contenu.

`$context.integration.latency` ou `$context.integrationLatency`

L'heure à laquelle API Gateway établit la connexion d'intégration jusqu'à ce que le flux de réponse d'intégration soit terminé.

La figure suivante montre comment ces variables du journal d'accès représentent les différents composants d'un flux de réponses.



Pour plus d'informations sur les journaux d'accès, consultez the section called "CloudWatch journaux". Vous pouvez également l'utiliser X-Ray pour surveiller votre flux de réponses. Pour de plus amples informations, veuillez consulter the section called "X-Ray".

Intégrations privées pour REST APIs dans API Gateway

Utilisez une intégration privée pour exposer vos HTTP/HTTPS ressources au sein d'un Amazon VPC afin que des clients extérieurs au VPC puissent y accéder. Cela étend l'accès à vos ressources VPC privées au-delà des limites du VPC. 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. API Gateway prend en charge les liens VPC V2 pour REST. APIs Les liens VPC V2 vous permettent de créer des intégrations privées qui connectent votre API REST aux équilibreurs de charge d'application sans utiliser de Network Load Balancer. L'utilisation d'un Application Load Balancer vous permet de vous connecter à des applications ECSs basées sur des conteneurs Amazon et à de nombreux autres backends. Les liens VPC V1 sont considérés comme un type d'intégration traditionnel. Bien qu'ils soient pris en charge par API Gateway, nous vous recommandons de ne pas créer de nouveaux liens VPC V1.

Considérations

Les considérations suivantes peuvent avoir un impact sur votre utilisation des intégrations privées :

- Toutes les ressources doivent être détenues par la même personne Compte AWS. Cela inclut l'équilibreur de charge, le lien VPC et l'API REST.
- Par défaut, le trafic d'intégration privée utilise le protocole HTTP. Pour utiliser le protocole HTTPS, `uri`spécifiez un nom de serveur sécurisé, tel que`https://example.com:443/test`.
- Dans le cas d'une intégration privée, API Gateway inclut la partie étape du point de terminaison de l'API dans la demande adressée à vos ressources principales. Par exemple, si vous demandez l'`test`étape d'une API, API Gateway inclut `test/path` dans la demande votre intégration privée. Pour supprimer le nom de l'étape de la demande adressée à vos ressources principales, utilisez le mappage des paramètres pour créer une substitution pour la `$context.requestOverride.path` variable.
- Les intégrations privées avec AWS Cloud Map ne sont pas prises en charge.

Rubriques

- Configuration des liens VPC V2 dans API Gateway

- Configurez une intégration privée
- Intégration privée à l'aide de liaisons VPC V1 (ancienne version)

Configuration des liens VPC V2 dans API Gateway

Les liens VPC vous permettent de créer des intégrations privées qui connectent vos routes d'API aux ressources privées d'un VPC, telles que les équilibreurs de charge d'application ou les applications basées sur des conteneurs Amazon ECS. Une intégration privée utilise un lien VPC V2 pour encapsuler les connexions entre API Gateway et les ressources VPC ciblées. Vous pouvez réutiliser les liens VPC entre différentes ressources et API.

Lorsque vous créez un lien VPC, API Gateway crée et gère des interfaces réseau élastiques pour le lien VPC V2 de votre compte. Ce processus peut prendre quelques minutes. Lorsqu'une liaison VPC V2 est prête à être utilisée, 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 V2 à l'aide du AWS CLI

Pour créer un lien VPC V2, toutes les ressources impliquées doivent appartenir au même AWS compte. La commande create-vpc-link suivante crée un lien VPC :

```
aws apigatewayv2 create-vpc-link --name MyVpcLink \
    --subnet-ids subnet-aaaa subnet-bbbb \
    --security-group-ids sg1234 sg5678
```

> **Note**
>
> Les liens VPC V2 sont immuables. Une fois que vous avez créé un lien VPC V2, vous ne pouvez pas modifier ses sous-réseaux ou ses groupes de sécurité.

Supprimez un lien VPC V2 à l'aide du AWS CLI

La commande delete-vpc-link suivante supprime un lien VPC.

```
aws apigatewayv2 delete-vpc-link --vpc-link-id abcd123
```

Disponibilité par région

Les liaisons VPC V2 sont prises 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
USA Est (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
Asie-Pacifique (Mumbai)	ap-south-1	aps1-az1, aps1-az2, aps1-az3
Asie-Pacifique (Osaka)	ap-northeast-3	apné3-az1, apné3-az2, apné3-az3
Asie-Pacifique (Séoul)	ap-northeast-2	apne2-az1, apne2-az2, apne2-az3

Nom de la région	Région	Zones de disponibilité prises en charge
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 (Espagne)	eu-south-2	eus2-az1, eus2-az2, eus2-az3
Europe (Stockholm)	eu-north-1	eun1-az1, eun1-az2, eun1-az3
Moyen-Orient (Bahreïn)	me-south-1	mes1-az1, mes1-az2, mes1-az3

Nom de la région	Région	Zones de disponibilité prises en charge
Amérique du Sud (São Paulo)	sa-east-1	sae1-az1, sae1-az2, sae1-az3
AWS GovCloud (US-West)	us-gov-west-1	usgw1-az1, usgw1-az2, usgw1-az3

Configurez une intégration privée

Pour créer une intégration privée avec un Application Load Balancer ou un Network Load Balancer, vous devez créer une intégration de proxy HTTP, spécifier le lien VPC V2 à utiliser et fournir l'ARN d'un Network Load Balancer ou d'un Application Load Balancer. Par défaut, le trafic d'intégration privée utilise le protocole HTTP. Pour utiliser le protocole HTTPS, `uri`spécifiez un nom de serveur sécurisé, tel que`https://example.com:443/test`. Pour un didacticiel complet sur la création d'une API REST avec une intégration privée, consultezthe section called "Didacticiel : création d'une API REST avec une intégration privée".

Création d'une intégration privée

La procédure suivante montre comment créer une intégration privée qui se connecte à un équilibreur de charge à l'aide d'un lien VPC V2.

Console de gestion AWS

Pour un didacticiel sur la création d'une intégration privée, voir,the section called "Didacticiel : création d'une API REST avec une intégration privée".

AWS CLI

La commande put-integration suivante crée une intégration privée qui se connecte à un équilibreur de charge à l'aide d'un lien VPC V2 :

```
aws apigateway put-integration \
    --rest-api-id abcdef123 \
    --resource-id aaa000 \
```

```
    --integration-target 'arn:aws:elasticloadbalancing:us-
east-2:111122223333:loadbalancer/app/myLoadBalancerName/1234567891011' \
    --uri 'https://example.com:443/path' \
    --http-method GET \
    --type HTTP_PROXY \
    --integration-http-method GET \
    --connection-type VPC_LINK \
    --connection-id bbb111
```

Au lieu de fournir directement l'ID de connexion, vous pouvez utiliser une variable d'étape à la place. Lorsque vous déployez votre API sur une étape, vous définissez l'ID VPC link V2. La commande put-integration suivante crée une intégration privée à l'aide d'une variable d'étape pour l'ID VPC link V2 :

```
aws apigateway put-integration \
    --rest-api-id abcdef123 \
    --resource-id aaa000 \
    --integration-target 'arn:aws:elasticloadbalancing:us-
east-2:111122223333:loadbalancer/app/myLoadBalancerName/1234567891011' \
    --uri 'https://example.com:443/path' \
    --http-method GET \
    --type HTTP_PROXY \
    --integration-http-method GET \
    --connection-type VPC_LINK \
    --connection-id "\${stageVariables.vpcLinkV2Id}"
```

Assurez-vous de mettre entre guillemets l'expression de la variable d'étape ($ {StageVariables.VPClinkV2ID}) et d'éviter le caractère $.

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 `VpcLinkV2` ou d’une variable d’étape renvoyant à l’ID d’un `VpcLinkV2`.
- Le `uri` paramètre de l'intégration privée pointe vers un HTTP/HTTPS point de terminaison dans le VPC, mais il est utilisé à la place pour configurer l'en-tête de la demande d'`Host`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 du `VpcLinkV2`. 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.vpcLinkIdV2}`) :

```
{
  "swagger": "2.0",
  "info": {
    "version": "2017-11-17T04:40:23Z",
    "title": "MyApiWithVpcLinkV2"
  },
  "host": "abcdef123.execute-api.us-west-2.amazonaws.com",
  "basePath": "/test",
  "schemes": [
    "https"
  ],
  "paths": {
    "/": {
      "get": {
        "produces": [
          "application/json"
        ],
        "responses": {
          "200": {
            "description": "200 response",
            "schema": {
              "$ref": "#/definitions/Empty"
            }
          }
        },
        "x-amazon-apigateway-integration": {
          "responses": {
            "default": {
              "statusCode": "200"
            }
          },
          "uri": "https://example.com:443/path",
```

```
          "passthroughBehavior": "when_no_match",
          "connectionType": "VPC_LINK",
          "connectionId": "${stageVariables.vpcLinkV2Id}",
          "integration-target": "arn:aws:elasticloadbalancing:us-
east-2:111122223333:loadbalancer/app/myLoadBalancerName/1234567891011",
          "httpMethod": "GET",
          "type": "http_proxy"
        }
      }
    }
  },
  "definitions": {
    "Empty": {
      "type": "object",
      "title": "Empty Schema"
    }
  }
}
```

Mettre à jour une intégration privée

L'exemple suivant met à jour le lien VPC V2 pour une intégration privée.

Console de gestion AWS

Pour mettre à jour une intégration privée

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Choisissez une API REST avec une intégration privée.
3. Choisissez la ressource et la méthode qui utilisent une intégration privée.
4. Dans l’onglet Requête d’intégration, sous Paramètres de requête d’intégration, choisissez Modifier.
5. Vous pouvez modifier les paramètres de votre intégration privée. Si vous utilisez actuellement un lien VPC V1, vous pouvez remplacer votre lien VPC par un lien VPC V2.
6. Choisissez Enregistrer.
7. Redéployez l’API pour que les modifications prennent effet.

AWS CLI

La commande update-integration suivante met à jour une intégration privée pour utiliser un lien VPC V2 :

```
aws apigateway update-integration \
    --rest-api-id a1b2c3d4e5 \
    --resource-id a1b2c3 \
    --http-method GET \
    --patch-operations "[{\"op\":\"replace\",\"path\":\"/connectionId\",\"value
\":\"pk0000\"}, {\"op\":\"replace\",\"path\":\"/uri\",\"value\":\"http://
example.com\"}, {\"op\":\"replace\",\"path\":\"/integrationTarget\",\"value
\":\"arn:aws:elasticloadbalancing:us-east-2:111122223333:loadbalancer/app/
myLoadBalancerName/1234567891011\"}]"
```

Intégration privée à l'aide de liaisons VPC V1 (ancienne version)

> **Note**
>
> L'implémentation suivante des intégrations privées utilise les liaisons VPC V1. Les liens VPC V1 sont des ressources héritées. Nous vous recommandons d'utiliser les liens VPC V2 pour REST. APIs

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. Le Network Load Balancer et l'API doivent appartenir à la même entité. Compte AWS

Rubriques

- Configuration d'un Network Load Balancer pour les intégrations privées d'API Gateway (ancienne version)
- Accorder des autorisations à API Gateway pour créer un lien VPC (ancien)

- Configurez une API API Gateway avec des intégrations privées à l'aide de l' AWS CLI (ancienne)
- Comptes API Gateway utilisés pour les intégrations privées (anciens)

Configuration d'un Network Load Balancer pour les intégrations privées d'API Gateway (ancienne version)

> **Note**
>
> L'implémentation suivante des intégrations privées utilise les liaisons VPC V1. Les liens VPC V1 sont des ressources héritées. Nous vous recommandons d'utiliser les liens VPC V2 pour REST. APIs

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 dans lequel vous avez des ressources, il vous suffit de configurer un NLB et un autre. 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 de plus amples informations, veuillez consulter the section called "Didacticiel : création d'une API REST avec une intégration privée". Le Network Load Balancer et l'API doivent appartenir à la même entité. Compte AWS

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 Console de gestion AWS 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 plus d'informations, suivez 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.
      - Utilisez la set-security-groupscommande suivante pour désactiver l'évaluation des groupes de sécurité pour PrivateLink le trafic :

```
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 aucune dépendance à API Gateway CIDRs car elles sont susceptibles de changer sans préavis.

Accorder des autorisations à API Gateway pour créer un lien VPC (ancien)

> **Note**
>
> L'implémentation suivante des intégrations privées utilise les liaisons VPC V1. Les liens VPC V1 sont des ressources héritées. Nous vous recommandons d'utiliser les liens VPC V2 pour REST. APIs

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 politique IAM semblable à celle-ci :

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "apigateway:POST",
                "apigateway:GET",
                "apigateway:PATCH",
                "apigateway:DELETE"
            ],
            "Resource": [
                "arn:aws:apigateway:us-east-1::/vpclinks",
                "arn:aws:apigateway:us-east-1::/vpclinks/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "elasticloadbalancing:DescribeLoadBalancers"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:CreateVpcEndpointServiceConfiguration",
                "ec2:DeleteVpcEndpointServiceConfigurations",
                "ec2:DescribeVpcEndpointServiceConfigurations",
                "ec2:ModifyVpcEndpointServicePermissions"
            ],
            "Resource": "*"
        }
    ]
}
```

Si vous souhaitez activer le balisage pour votre lien VPC, veillez à autoriser les opérations de balisage. Pour de plus amples informations, veuillez consulter the section called “Autoriser les opérations de balisage”.

2. Créez ou choisissez un rôle IAM et attachez-y la politique précédente.
3. Attribuez le rôle IAM à vous-même ou à un utilisateur de votre compte qui crée des liens VPC.

Configurez une API API Gateway avec des intégrations privées à l'aide de l' AWS CLI (ancienne)

> **Note**
>
> L'implémentation suivante des intégrations privées utilise les liaisons VPC V1. Les liens VPC V1 sont des ressources héritées. Nous vous recommandons d'utiliser les liens VPC V2 pour REST. APIs

Le didacticiel suivant montre comment utiliser le AWS CLI pour créer un lien VPC et une intégration privée. Les conditions préalables suivantes sont requises :

- Vous avez besoin d’un équilibreur Network Load Balancer créé et configuré avec votre VPC source comme cible. Pour de plus amples informations, veuillez consulter Configuration d'un Network Load Balancer pour les intégrations privées d'API Gateway (ancienne version). Il doit se trouver dans le même Compte AWS emplacement que votre API. Vous avez besoin de l’ARN de votre équilibreur Network Load Balancer pour créer votre lien VPC.
- Pour créer et gérer un `VpcLink`, vous devez disposer des autorisations pour créer un `VpcLink` dans votre API. Vous n’avez pas besoin d’autorisations pour utiliser le `VpcLink`. Pour de plus amples informations, veuillez consulter Accorder des autorisations à API Gateway pour créer un lien VPC (ancien).

Pour configurer une API avec l'intégration privée à l'aide de AWS CLI

1. Utilisez la create-vpc-linkcommande suivante pour créer un `VpcLink` ciblage du 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`. Vous pouvez le vérifier à l'aide de la get-vpc-linkcommande 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 du `VpcLink` réussie, vous pouvez créer une API et l'intégrer avec la ressource VPC par le biais du `VpcLink`.

Notez la valeur `id` du `VpcLink` que vous venez de créer. Dans cet exemple de sortie, il s'agit de `gim7c3`. Vous en aurez besoin pour configurer l'intégration privée.

2. Utilisez la create-rest-apicommande suivante pour créer une `RestApi`ressource API Gateway :

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

Notez la valeur `id` de `RestApi` et la valeur `rootResourceId` de `RestApi` dans le résultat renvoyé. Vous avez besoin de cette valeur pour effectuer d'autres opérations sur l'API.

Ensuite, vous allez créer une API avec seulement une méthode `GET` sur la ressource racine (`/`) et intégrer la méthode avec le `VpcLink`.

3. Utilisez la commande put-method suivante pour créer la méthode `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 avec le `VpcLink`, vous devez également configurer au moins une réponse de méthode avec le code de statut `200`. Vous allez utiliser l'intégration proxy ici.

4. Après avoir créé la méthode `GET /`, vous devez configurer l'intégration. Pour une intégration privée, vous devez utiliser le paramètre `connection-id` pour fournir l'ID `VpcLink`. Vous pouvez utiliser une variable d'étape ou saisir directement l'ID du `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.

   Use the VPC link ID

   Utilisez la commande put-integration suivante pour utiliser l'ID `VpcLink` directement dans 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 gim7c3
```

Use a stage variable

Utilisez la commande put-integration suivante afin d'utiliser une variable d'étape pour référencer l'ID du lien VPC. Lorsque vous déployez votre API dans une étape, vous devez définir l'ID du lien VPC.

```
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 $.

À tout moment, vous pouvez également mettre à jour l'intégration pour modifier `connection-id`. Utilisez la commande update-integration suivante pour mettre à jour votre intégration :

```
 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`.

Comme vous avez utilisé l'intégration proxy privée, votre API peut maintenant être déployée et testée.

5. Si vous avez utilisé la variable d'étape pour définir `connection-id`, vous devez déployer votre API pour la tester. Utilisez la commande create-deployment suivante pour déployer votre API avec une variable d'étape :

```
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*), utilisez la commande update-stage suivante :

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

Lorsque vous codez en dur la propriété `connection-id` avec l'ID littéral `VpcLink`, vous n'avez pas besoin de déployer votre API pour la tester. Utilisez la test-invoke-methodcommande pour tester votre API avant son déploiement.

6. Utilisez la commande suivante pour appeler votre API :

```
curl -X GET https://abcdef123.execute-api.us-east-2.amazonaws.com/test
```

Vous pouvez également saisir l'URL d'invocation de votre API dans un navigateur web pour afficher le résultat.

Comptes API Gateway utilisés pour les intégrations privées (anciens)

Les comptes API Gateway spécifiques à la région suivants IDs sont automatiquement ajoutés à votre service de point de terminaison VPC `AllowedPrincipals` lorsque vous créez un. `VpcLink`

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

Intégrations fictives pour REST APIs dans API Gateway

Amazon API Gateway prend en charge les intégrations simulées 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 Didacticiel : 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 simulée. Pour cela, vous configurez la demande d'intégration et la réponse d'intégration de la méthode pour associer une réponse à un code de statut donné. Pour une méthode avec l'intégration simulée 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 simulée 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 simulées 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 simulée 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 simulée 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 simulée d'une méthode d'API.

Rubriques

- Activation de l'intégration simulée à l'aide de la console API Gateway

Activation de l'intégration simulée à l'aide de la console API Gateway

La méthode doit être disponible dans API Gateway. Suivez les instructions de la section Didacticiel : création d'une API REST avec une intégration HTTP sans 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 paramètre 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 simulée 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.

Validation des demandes pour les API REST 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
- Modèles de données pour les API REST
- Configuration de la validation de demande de base dans API Gateway
- 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. API Gateway vérifie uniquement si un paramètre existe. Il ne vérifie ni son type ni son format.
- Les données utiles de la demande adhère à la demande de schéma JSON configurée de la méthode pour un type de contenu donné. 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, définissez le type de contenu de votre modèle de données sur `$default`.

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 Comportement des demandes de méthode pour les données utiles sans modèle de mappage pour les API REST dans API Gateway 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 de plus amples informations, veuillez consulter Comportement des demandes de méthode pour les données utiles sans modèle de mappage pour les API REST dans API Gateway.

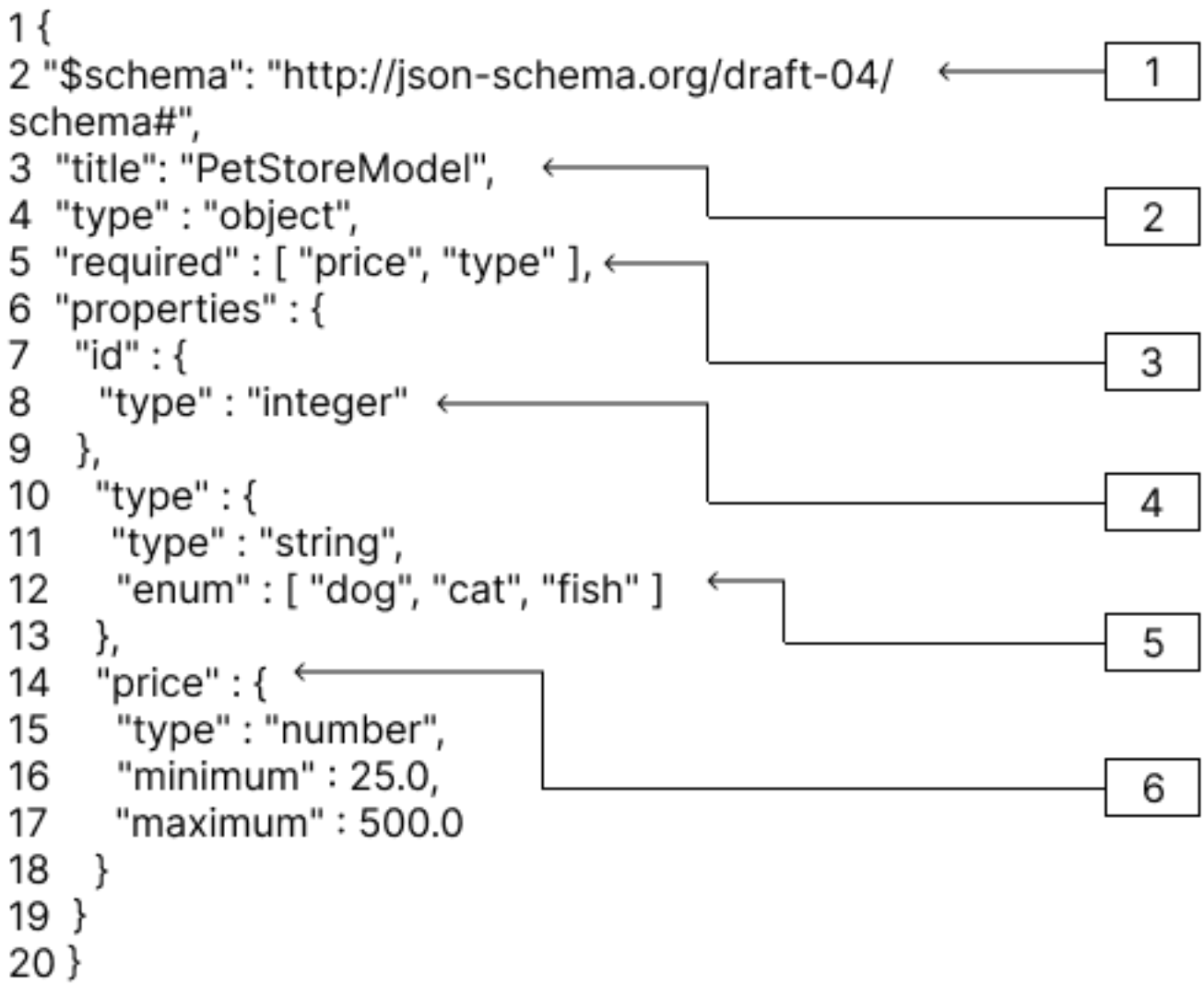
Modèles de données pour les API REST

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, tels que Java ou Swift Objective-C, 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 Transformations de modèles de mappage pour les API REST dans API Gateway.

L'exemple suivant présente les données de sortie d'une réponse d'intégration.

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

L'exemple suivant est un modèle de données utiles qui décrit les données 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

- Pour plus d'informations sur la transformation des données et les modèles de mappage, consultez Transformations de modèles de mappage pour les API REST dans 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 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 avec le 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 CloudFormation. Vous allez utiliser ce modèle pour créer une API incomplète. Vous terminerez les étapes restantes dans la console API Gateway.

Pour créer un CloudFormation pile

1. Ouvrez la 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 CloudFormation vous pouvez créer des ressources IAM dans votre compte.
9. Choisissez Suivant, puis Soumettre.

CloudFormation fournit les ressources spécifiées dans le modèle. La fin du provisionnement de vos ressources peut prendre quelques minutes. Lorsque le statut de votre 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 Créer un modèle.
3. Pour 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 Modèles de données pour les API REST.

Pour configurer la validation des demandes 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 requête 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 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 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 demandes 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 des demandes 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 requête 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`**. 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`.

   Pour Modèle, sélectionnez PetStoreModel.
6. Choisissez Enregistrer.

Pour tester la validation des demandes 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 un CloudFormation pile

1. Ouvrez la CloudFormation console à l'adresse https://console.aws.amazon.com/cloudformation.
2. Sélectionnez votre 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 CloudFormation modèle pour créer une API API Gateway incomplète.

> **Note**
>
> Il ne s'agit pas du même 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 avec le 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 un CloudFormation pile

Téléchargez et décompressez le modèle de création d'application pour 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. Pour obtenir une Windows-integrated version d'Ubuntu et de Bash, installez le sous-système Windows pour Linux. 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 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. 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 CloudFormation pile.

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

3. Lorsque le statut de votre CloudFormation pile est défini`StackStatus: "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 un CloudFormation pile

- Utilisez la commande suivante pour supprimer vos 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 Objet x-amazon-apigateway-request-validators.requestValidator dans le mappage d'Objet x-amazon-apigateway-request-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 modèle de données `RequestBodyModel` 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.

- 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 Propriété x-amazon-apigateway-request-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'Importation d'une API régionale dans API Gateway ou l'Importation d'une API optimisée pour la périphérie 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
    }
  }
}
```

AWS CloudFormation modèle d'un exemple d'API avec validation de demande de base

L' CloudFormation exemple de définition de modèle suivant définit un exemple d'API avec la validation des demandes activée. L'API PetStore est un sous-ensemble de l'API . 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}'
```

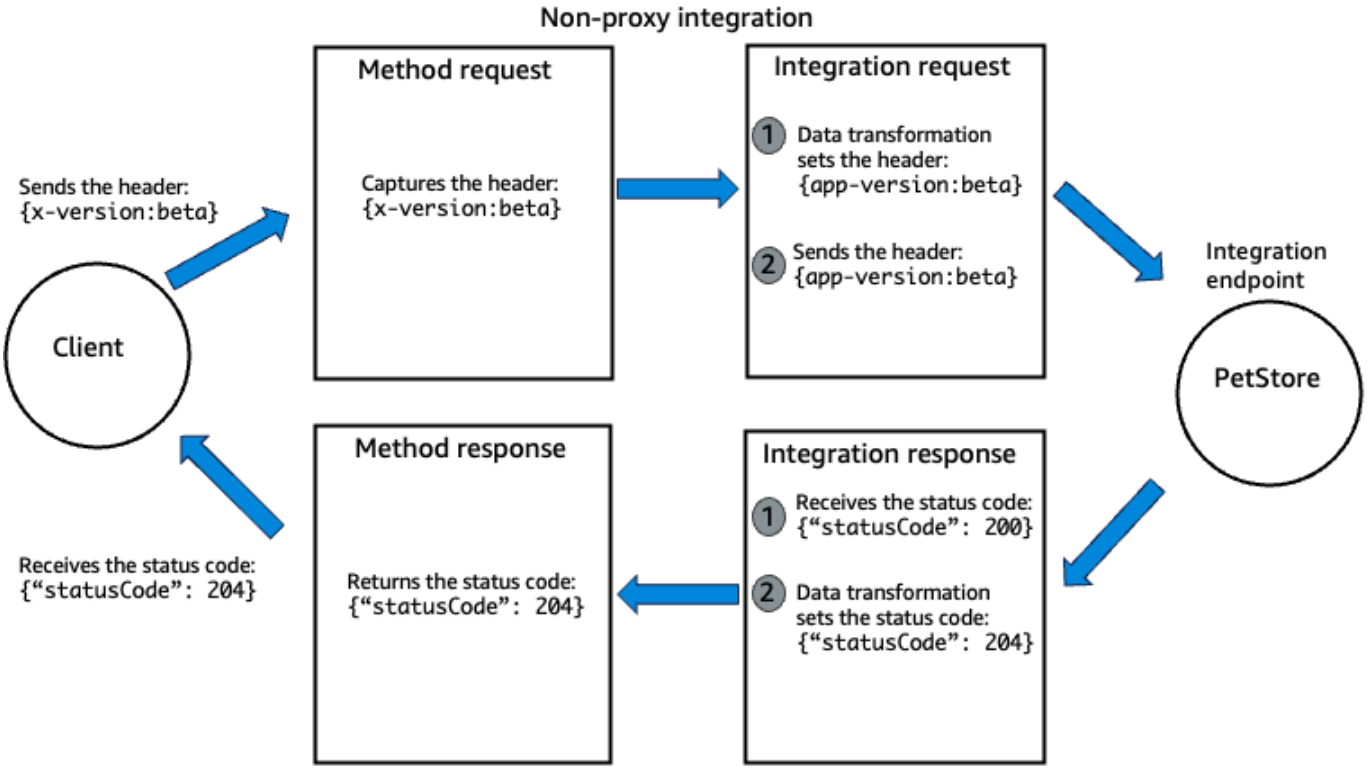
Transformations de données pour les API REST dans API Gateway

> **Note**
>
> Cette section explique les fonctionnalités que vous allez utiliser avec une intégration sans proxy. Toutefois, nous vous recommandons, dans la mesure du possible, d'utiliser une intégration proxy pour votre API REST. Une intégration proxy dispose d'une configuration

d'intégration simplifiée et peut évoluer avec le backend sans qu'il soit nécessaire de modifier la configuration existante. Pour de plus amples informations, veuillez consulter the section called "Choix d'un type d'intégration d'API".

Si vous utilisez une intégration sans proxy, vous pouvez utiliser deux fonctionnalités d'API Gateway pour transformer votre demande de méthode et votre réponse d'intégration. Vous pouvez transformer votre demande de méthode si elle utilise un format de données utiles différent de celui de la demande d'intégration. Vous pouvez transformer votre réponse d'intégration si elle renvoie un format de données utiles différent de celui que vous devez renvoyer dans la réponse de méthode. Pour plus d'informations sur le cycle de vie des demandes, consultez the section called "Exemple de ressource pour une API REST".

L'exemple suivant montre une transformation de données dans laquelle, pour l'en-tête `"x-version:beta"`, le paramètre d'en-tête `x-version` est transformé en paramètre d'en-tête `app-version`. Les données passent de `x-version` à `app-version` dans la demande d'intégration. Ainsi, le point de terminaison d'intégration reçoit la valeur du paramètre d'en-tête transformé. Lorsque le point de terminaison d'intégration renvoie un code d'état, celui-ci passe de `200` à `204` avant la réponse de méthode.



Pour transformer des données, vous pouvez utiliser les fonctionnalités suivantes :

Mappage de paramètres

Dans un mappage des paramètres, vous pouvez modifier les paramètres du chemin d'URL de la demande d'intégration, les paramètres de la chaîne de requête d'URL ou les valeurs d'en-tête HTTP, mais vous ne pouvez pas modifier les données utiles de la demande d'intégration. Vous pouvez également modifier les valeurs de l'en-tête de réponse HTTP. Utilisez le mappage de paramètres pour créer des valeurs d'en-tête statiques pour le partage des ressources entre origines multiples (CORS).

Vous pouvez utiliser le mappage de paramètres dans votre demande d'intégration pour les intégrations proxy et sans proxy. En revanche, pour utiliser le mappage de paramètres dans une réponse d'intégration, vous avez besoin d'une intégration sans proxy. Le mappage de paramètres ne nécessite aucun script en langage VTL (Velocity Template Language). Pour de plus amples informations, veuillez consulter the section called "Mappage de paramètres".

Transformations de modèles de mappage

Pour transformer un modèle de mappage, vous allez utiliser un modèle de mappage pour mapper les paramètres de chemin d'URL, les paramètres de chaîne de requête d'URL, les en-têtes HTTP, ainsi que le corps de la demande d'intégration ou de la réponse d'intégration. Un modèle de mappage est un script exprimé en langage VTL à l'aide d'expressions JSONPath et appliqué aux données utiles en fonction de l'en-tête `Content-type`.

Un modèle de mappage vous permet d'effectuer les opérations suivantes :

- 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. Pour de plus amples informations, veuillez consulter the section called "Tutoriel : Modifier la demande et la réponse d'intégration pour les intégrations vers AWS services".
- Remplacer de manière conditionnelle les paramètres de demande d'intégration et de réponse d'intégration d'une API, créer des valeurs d'en-tête et remplacer des codes d'état. Pour de plus amples informations, veuillez consulter the section called "Remplacement des codes d'état et des paramètres de demande et de réponse de votre API".

Vous pouvez également spécifier le comportement de votre API lorsqu'un corps de demande d'intégration comporte un en-tête `Content-type` sans modèles de mappage correspondants. C'est ce qu'on appelle le comportement de transmission d'intégration. Pour de plus amples informations, veuillez consulter the section called "Comportement des demandes de méthode pour les données utiles sans modèle de mappage".

Choisissez entre le mappage de paramètres et la transformation du modèle de mappage

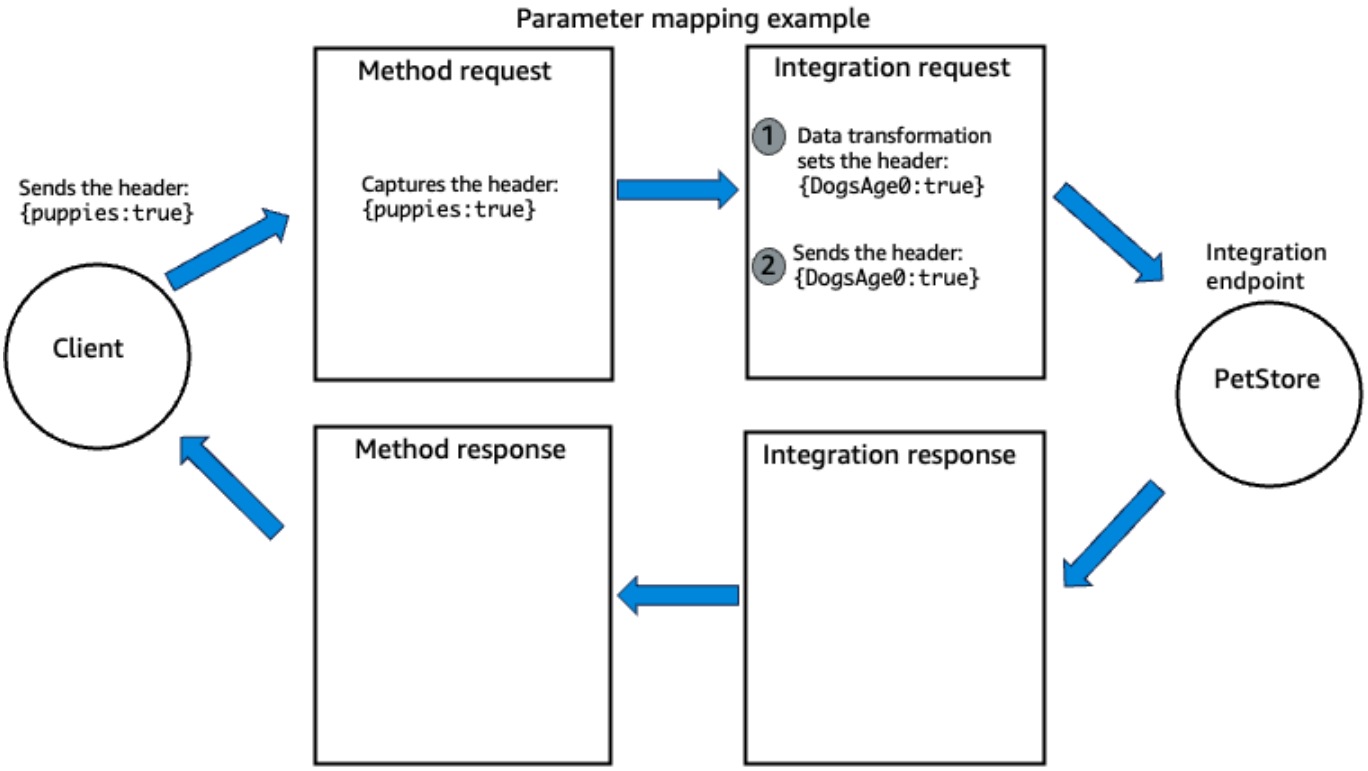
Nous vous recommandons d'utiliser le mappage de paramètres pour transformer vos données lorsque cela est possible. Si votre API vous oblige à modifier le corps ou à effectuer des remplacements et modifications conditionnelles en fonction de la demande d'intégration entrante ou de la réponse d'intégration, et que vous ne pouvez pas utiliser d'intégration proxy, optez pour la transformation du modèle de mappage.

Mappage de paramètres pour les API REST dans API Gateway

> **Note**
>
> Si vous utilisez une API HTTP, consultez the section called "Mappage de paramètres".

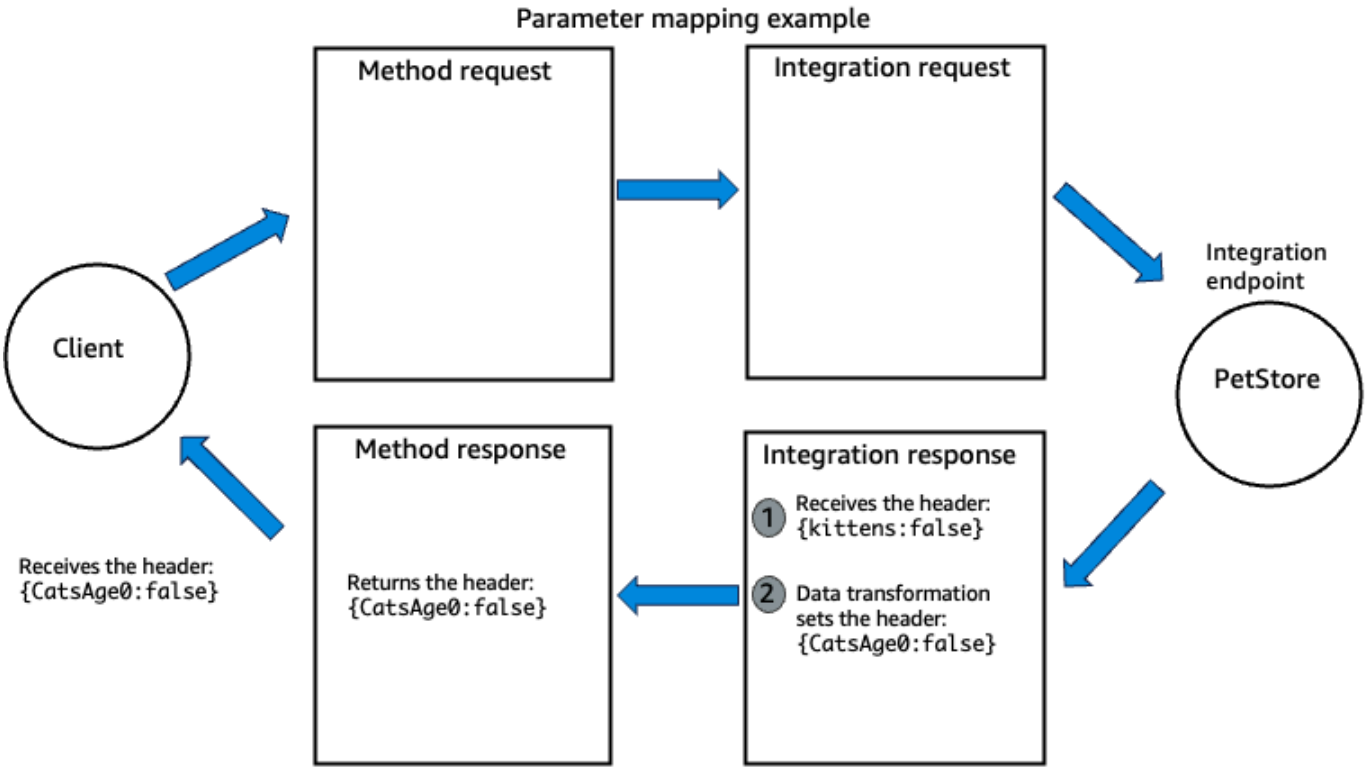
Dans le mappage de paramètres, vous mappez les paramètres de la demande ou de la réponse. Vous pouvez mapper des paramètres à l'aide d'expressions de mappage de paramètres ou de valeurs statiques. Pour obtenir la liste des expressions de mappage, consultez the section called "Référence de la source du mappage de paramètres". Vous pouvez utiliser le mappage de paramètres dans votre demande d'intégration pour les intégrations proxy et sans proxy. En revanche, pour utiliser le mappage de paramètres dans une réponse d'intégration, vous avez besoin d'une intégration sans proxy.

Par exemple, vous pouvez mapper le paramètre d'en-tête `puppies` de la demande de méthode au paramètre d'en-tête `DogsAge0` de la demande d'intégration. Si un client envoie l'en-tête `puppies:true` à votre API, la demande d'intégration envoie alors l'en-tête de demande `DogsAge0:true` au point de terminaison d'intégration. Le schéma suivant illustre le cycle de vie de la demande de cet exemple.



Pour créer cet exemple à l'aide d'API Gateway, consultez the section called "Exemple 1 : association d'un paramètre de demande de méthode à un paramètre de demande d'intégration".

Autre exemple, vous pouvez également mapper le paramètre d'en-tête de réponse d'intégration `kittens` au paramètre d'en-tête de réponse de méthode`CatsAge0`. Ensuite, si le point de terminaison d'intégration renvoie `kittens:false`, le client reçoit l'en-tête `CatsAge0:false`. Le schéma suivant illustre le cycle de vie de la demande de cet exemple.



Rubriques

- Exemples de mappage de paramètres pour les API REST dans API Gateway
- Référence de la source du mappage de paramètres pour les API REST dans API Gateway

Exemples de mappage de paramètres pour les API REST dans API Gateway

Les exemples suivants montrent comment créer des expressions de mappage de paramètres à l'aide de la console API Gateway, d'OpenAPI et de modèles CloudFormation . Pour voir un exemple d'utilisation du mappage de paramètres dans le but de créer les en-têtes CORS requis, consultez the section called "CORS".

Exemple 1 : association d'un paramètre de demande de méthode à un paramètre de demande d'intégration

L'exemple suivant mappe le paramètre d'en-tête `puppies` de la demande de méthode au paramètre d'en-tête `DogsAge0` de la demande d'intégration.

Console de gestion AWS

Pour mapper le paramètre de la demande de méthode

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Choisissez une API REST.
3. Choisissez une méthode.

   Votre méthode doit avoir une intégration sans proxy.
4. Pour Paramètres de requête de méthode, choisissez Modifier.
5. Développez En-têtes de demande HTTP.
6. Sélectionnez Add header.
7. Pour Nom, saisissez **`puppies`**.
8. Choisissez Enregistrer.
9. Choisissez l'onglet Requête d'intégration, puis dans la section Paramètres de requête d'intégration, choisissez Modifier.

   Ajoute Console de gestion AWS automatiquement un mappage de paramètres de `method.request.header.puppies` à `puppies` pour vous, mais vous devez modifier le nom pour qu'il corresponde au paramètre d'en-tête de demande attendu par votre point de terminaison d'intégration.
10. Pour Nom, saisissez **`DogsAge0`**.
11. Choisissez Enregistrer.
12. Redéployez l'API pour que les modifications prennent effet.

Les étapes suivantes montrent comment vérifier que votre mappage de paramètres a été effectué avec succès.

(Facultatif) Testez votre mappage de paramètres.

1. Choisissez l'onglet Test. Vous devrez peut-être choisir la flèche droite pour afficher l'onglet.
2. Pour les en-têtes, saisissez **`puppies:true`**.
3. Sélectionnez Tester).
4. Dans Journaux, le résultat doit se présenter comme suit :

```
Tue Feb 04 00:28:36 UTC 2025 : Method request headers: {puppies=true}
Tue Feb 04 00:28:36 UTC 2025 : Method request body before transformations:
Tue Feb 04 00:28:36 UTC 2025 : Endpoint request URI: http://petstore-demo-
endpoint.execute-api.com/petstore/pets
Tue Feb 04 00:28:36 UTC 2025 : Endpoint request headers: {DogsAge0=true,
 x-amzn-apigateway-api-id=abcd1234, Accept=application/json, User-
Agent=AmazonAPIGateway_aaaaaaa, X-Amzn-Trace-Id=Root=1-abcd-12344}
```

Le paramètre d'en-tête de la demande est passé de `puppies` à `DogsAge0`.

CloudFormation

Dans cet exemple, vous allez utiliser la propriété body pour importer un fichier de définition OpenAPI dans API Gateway.

```
AWSTemplateFormatVersion: 2010-09-09
Resources:
  Api:
    Type: 'AWS::ApiGateway::RestApi'
    Properties:
      Body:
        openapi: 3.0.1
        info:
          title: ParameterMappingExample
          version: "2025-02-04T00:30:41Z"
        paths:
          /pets:
            get:
              parameters:
                - name: puppies
                  in: header
                  schema:
                    type: string
              responses:
                "200":
                  description: 200 response
              x-amazon-apigateway-integration:
                httpMethod: GET
                uri: http://petstore-demo-endpoint.execute-api.com/petstore/pets
                responses:
                  default:
```

```
                    statusCode: "200"
                requestParameters:
                  integration.request.header.DogsAge0: method.request.header.puppies
                passthroughBehavior: when_no_match
                type: http
  ApiGatewayDeployment:
    Type: 'AWS::ApiGateway::Deployment'
    DependsOn: Api
    Properties:
      RestApiId: !Ref Api
  ApiGatewayDeployment20250219:
    Type: 'AWS::ApiGateway::Deployment'
    DependsOn: Api
    Properties:
      RestApiId: !Ref Api
  Stage:
    Type: 'AWS::ApiGateway::Stage'
    Properties:
       DeploymentId: !Ref ApiGatewayDeployment20250219
       RestApiId: !Ref Api
       StageName: prod
```

OpenAPI

```
{
  "openapi" : "3.0.1",
  "info" : {
    "title" : "ParameterMappingExample",
    "version" : "2025-02-04T00:30:41Z"
  },
  "paths" : {
    "/pets" : {
      "get" : {
        "parameters" : [ {
          "name" : "puppies",
          "in" : "header",
          "schema" : {
            "type" : "string"
          }
        } ],
        "responses" : {
          "200" : {
            "description" : "200 response"
```

```
          }
        },
        "x-amazon-apigateway-integration" : {
          "httpMethod" : "GET",
          "uri" : "http://petstore-demo-endpoint.execute-api.com/petstore/pets",
          "responses" : {
            "default" : {
              "statusCode" : "200"
            }
          },
          "requestParameters" : {
            "integration.request.header.DogsAge0" : "method.request.header.puppies"
          },
          "passthroughBehavior" : "when_no_match",
          "type" : "http"
        }
      }
    }
  }
}
```

Exemple 2 : association de plusieurs paramètres de requête de méthode à différents paramètres de demande d'intégration

L'exemple suivant mappe le paramètre `methodRequestQueryParam` de la chaîne de requête de méthode à valeurs multiples au paramètre `integrationQueryParam` de la chaîne de requête de demande d'intégration, et mappe le paramètre d'en-tête `methodRequestHeaderParam` de la demande de méthode au paramètre de chemin `integrationPathParam` de la demande d'intégration.

Console de gestion AWS

Pour mapper les paramètres de requête de méthode

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Choisissez une API REST.
3. Choisissez une méthode.

   Votre méthode doit avoir une intégration sans proxy.

4. Pour Paramètres de requête de méthode, choisissez Modifier.
5. Choisissez Paramètres de chaîne de requête d'URL.
6. Sélectionnez Add query string (Ajouter une chaîne de requêtes).
7. Pour Nom, saisissez **`methodRequestQueryParam`**.
8. Développez En-têtes de demande HTTP.
9. Sélectionnez Add header.
10. Pour Nom, saisissez **`methodRequestHeaderParam`**.
11. Choisissez Enregistrer.
12. Choisissez l'onglet Requête d'intégration, puis dans la section Paramètres de requête d'intégration, choisissez Modifier.
13. Choisissez paramètres de chemin d'URL.
14. Choisissez Ajouter un paramètre de chemin.
15. Pour Nom, saisissez **`integrationPathParam`**.
16. Pour Mappage à partir de, entrez **`method.request.header.methodRequestHeaderParam`**.

    Cette opération mappe l'en-tête de demande de méthode que vous avez spécifié dans la demande de méthode à un nouveau paramètre de chemin de la demande d'intégration.
17. Choisissez Paramètres de chaîne de requête d'URL.
18. Sélectionnez Add query string (Ajouter une chaîne de requêtes).
19. Pour Nom, saisissez **`integrationQueryParam`**.
20. Pour Mappage à partir de, entrez **`method.request.multivaluequerystring.methodRequestQueryParam`**.

    Cette opération mappe le paramètre de chaîne de requête à valeurs multiples à un nouveau paramètre de chaîne de requête de demande d'intégration à valeur unique.
21. Choisissez Enregistrer.
22. Redéployez l'API pour que les modifications prennent effet.

CloudFormation

Dans cet exemple, vous allez utiliser la propriété body pour importer un fichier de définition OpenAPI dans API Gateway.

La définition OpenAPI suivante crée les mappages de paramètres suivants pour une intégration HTTP :

- l'en-tête de la demande de méthode, nommé `methodRequestHeaderParam`, avec le paramètre de chemin de la demande d'intégration, nommé `integrationPathParam` ;
- la chaîne de requête de la demande de méthode à valeurs multiples, nommée `methodRequestQueryParam`, avec la chaîne de requête de la demande d'intégration, nommée `integrationQueryParam`.

```
AWSTemplateFormatVersion: 2010-09-09
Resources:
  Api:
    Type: 'AWS::ApiGateway::RestApi'
    Properties:
      Body:
        openapi: 3.0.1
        info:
          title: Parameter mapping example 2
          version: "2025-01-15T19:12:31Z"
        paths:
          /:
            post:
              parameters:
                - name: methodRequestQueryParam
                  in: query
                  schema:
                    type: string
                - name: methodRequestHeaderParam
                  in: header
                  schema:
                    type: string
              responses:
                "200":
                  description: 200 response
              x-amazon-apigateway-integration:
                httpMethod: GET
                uri: http://petstore-demo-endpoint.execute-api.com/petstore/pets
                responses:
                  default:
                    statusCode: "200"
                requestParameters:
```

```
                  integration.request.querystring.integrationQueryParam:
method.request.multivaluequerystring.methodRequestQueryParam
                  integration.request.path.integrationPathParam:
method.request.header.methodRequestHeaderParam
                requestTemplates:
                  application/json: '{"statusCode": 200}'
                passthroughBehavior: when_no_templates
                timeoutInMillis: 29000
                type: http
 ApiGatewayDeployment:
   Type: 'AWS::ApiGateway::Deployment'
   DependsOn: Api
   Properties:
     RestApiId: !Ref Api
 ApiGatewayDeployment20250219:
   Type: 'AWS::ApiGateway::Deployment'
   DependsOn: Api
   Properties:
     RestApiId: !Ref Api
 Stage:
   Type: 'AWS::ApiGateway::Stage'
   Properties:
      DeploymentId: !Ref ApiGatewayDeployment20250219
      RestApiId: !Ref Api
      StageName: prod
```

OpenAPI

La définition OpenAPI suivante crée les mappages de paramètres suivants pour une intégration HTTP :

- l'en-tête de la demande de méthode, nommé `methodRequestHeaderParam`, avec le paramètre de chemin de la demande d'intégration, nommé `integrationPathParam` ;
- la chaîne de requête de la demande de méthode à valeurs multiples, nommée `methodRequestQueryParam`, avec la chaîne de requête de la demande d'intégration, nommée `integrationQueryParam`.

```
{
  "openapi" : "3.0.1",
  "info" : {
    "title" : "Parameter mapping example 2",
```

```
    "version" : "2025-01-15T19:12:31Z"
  },
  "paths" : {
    "/" : {
      "post" : {
        "parameters" : [ {
          "name" : "methodRequestQueryParam",
          "in" : "query",
          "schema" : {
            "type" : "string"
          }
        }, {
          "name" : "methodRequestHeaderParam",
          "in" : "header",
          "schema" : {
            "type" : "string"
          }
        } ],
        "responses" : {
          "200" : {
            "description" : "200 response"
          }
        },
        "x-amazon-apigateway-integration" : {
          "httpMethod" : "GET",
          "uri" : "http://petstore-demo-endpoint.execute-api.com/petstore/pets",
          "responses" : {
            "default" : {
              "statusCode" : "200"
            }
          },
          "requestParameters" : {
            "integration.request.querystring.integrationQueryParam" :
"method.request.multivaluequerystring.methodRequestQueryParam",
            "integration.request.path.integrationPathParam" :
"method.request.header.methodRequestHeaderParam"
          },
          "requestTemplates" : {
            "application/json" : "{\"statusCode\": 200}"
          },
          "passthroughBehavior" : "when_no_templates",
          "timeoutInMillis" : 29000,
          "type" : "http"
        }
```

```
      }
    }
  }
}
```

Exemple 3 : mappage de champs du corps de la demande JSON aux paramètres de la demande d'intégration

Vous pouvez également mapper les paramètres de la demande d'intégration à partir des champs du corps de la demande JSON à l'aide d'une expression JSONPath. L'exemple suivant mappe le corps de la demande de méthode à un en-tête de la demande d'intégration nommé `body-header`, et mappe une partie du corps de la demande, tel qu'il est exprimé par une expression JSON, à un en-tête de la demande d'intégration nommé `pet-price`.

Pour tester cet exemple, fournissez une entrée contenant une catégorie de prix, par exemple :

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

Console de gestion AWS

Pour mapper les paramètres de requête de méthode

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Choisissez une API REST.
3. Choisissez la méthode `POST`, `PUT`, `PATCH` ou `ANY`.

   Votre méthode doit avoir une intégration sans proxy.
4. Pour Paramètres de requête d'intégration, choisissez Modifier.
5. Sélectionnez Paramètres des en-têtes de requête d'URL.
6. Sélectionnez Ajouter un paramètre d'en-tête de requête.
7. Pour Nom, saisissez **`body-header`**.

8. Pour Mappage à partir de, entrez **`method.request.body`**.

   Cette opération mappe le corps de la demande de méthode à un nouveau paramètre d'en-tête de la demande d'intégration.
9. Sélectionnez Ajouter un paramètre d'en-tête de requête.
10. Pour Nom, saisissez **`pet-price`**.
11. Pour Mappage à partir de, entrez **`method.request.body[0].price`**.

    Cette opération mappe une partie du corps de la demande de méthode à un nouveau paramètre d'en-tête de la demande d'intégration.
12. Choisissez Enregistrer.
13. Redéployez l'API pour que les modifications prennent effet.

CloudFormation

Dans cet exemple, vous allez utiliser la propriété body pour importer un fichier de définition OpenAPI dans API Gateway.

```
AWSTemplateFormatVersion: 2010-09-09
Resources:
  Api:
    Type: 'AWS::ApiGateway::RestApi'
    Properties:
      Body:
        openapi: 3.0.1
        info:
          title: Parameter mapping example 3
          version: "2025-01-15T19:19:14Z"
        paths:
          /:
            post:
              responses:
                "200":
                  description: 200 response
              x-amazon-apigateway-integration:
                httpMethod: GET
                uri: http://petstore-demo-endpoint.execute-api.com/petstore/pets
                responses:
                  default:
                    statusCode: "200"
```

```
                requestParameters:
                  integration.request.header.pet-price: method.request.body[0].price
                  integration.request.header.body-header: method.request.body
                requestTemplates:
                  application/json: '{"statusCode": 200}'
                passthroughBehavior: when_no_templates
                timeoutInMillis: 29000
                type: http
  ApiGatewayDeployment:
    Type: 'AWS::ApiGateway::Deployment'
    DependsOn: Api
    Properties:
      RestApiId: !Ref Api
  ApiGatewayDeployment20250219:
    Type: 'AWS::ApiGateway::Deployment'
    DependsOn: Api
    Properties:
      RestApiId: !Ref Api
  Stage:
    Type: 'AWS::ApiGateway::Stage'
    Properties:
       DeploymentId: !Ref ApiGatewayDeployment20250219
       RestApiId: !Ref Api
       StageName: prod
```

OpenAPI

La définition OpenAPI suivante mappe les paramètres de la demande d'intégration à partir des champs du corps de la demande JSON.

```
{
  "openapi" : "3.0.1",
  "info" : {
    "title" : "Parameter mapping example 3",
    "version" : "2025-01-15T19:19:14Z"
  },
  "paths" : {
    "/" : {
      "post" : {
        "responses" : {
          "200" : {
            "description" : "200 response"
          }
```

```
        },
        "x-amazon-apigateway-integration" : {
          "httpMethod" : "GET",
          "uri" : "http://petstore-demo-endpoint.execute-api.com/petstore/pets",
          "responses" : {
            "default" : {
              "statusCode" : "200"
            }
          },
          "requestParameters" : {
            "integration.request.header.pet-price" : "method.request.body[0].price",
            "integration.request.header.body-header" : "method.request.body"
          },
          "requestTemplates" : {
            "application/json" : "{\"statusCode\": 200}"
          },
          "passthroughBehavior" : "when_no_templates",
          "timeoutInMillis" : 29000,
          "type" : "http"
        }
      }
    }
  }
}
```

Exemple 4 : mappage de la réponse d'intégration à la réponse de méthode

Vous pouvez également mapper la réponse d'intégration à la réponse de méthode. L'exemple suivant mappe le corps de la réponse d'intégration à un en-tête de la réponse de méthode nommé `location`, mappe l'en-tête `x-app-id` de la réponse d'intégration à l'en-tête `id` de la réponse de méthode, et mappe l'en-tête à valeurs multiples `item` de la réponse d'intégration à l'en-tête `items` de la réponse de méthode.

Console de gestion AWS

Pour mapper la réponse d'intégration

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Choisissez une API REST.
3. Choisissez une méthode.

Votre méthode doit avoir une intégration sans proxy.

4. Choisissez l'onglet Méthode de réponse, puis pour Réponse 200, sélectionnez Modifier.
5. Sous Nom de l'en-tête, choisissez Ajouter un en-tête.
6. Créez trois en-têtes nommés **`id`**, **`item`** et **`location`**.
7. Choisissez Enregistrer.
8. Choisissez l'onglet Réponse d'intégration, puis pour Par défaut - Réponse, choisissez Modifier.
9. Sous Mappages d'en-tête, saisissez ce qui suit.
   a. Sous ID, saisissez **`integration.response.header.x-app-id`**.
   b. Sous Élément, saisissez **`integration.response.multivalueheader.item`**.
   c. Sous Emplacement, saisissez **`integration.response.body.redirect.url`**.
10. Choisissez Enregistrer.
11. Redéployez l'API pour que les modifications prennent effet.

CloudFormation

Dans cet exemple, vous allez utiliser la propriété body pour importer un fichier de définition OpenAPI dans API Gateway.

```
AWSTemplateFormatVersion: 2010-09-09
Resources:
  Api:
    Type: 'AWS::ApiGateway::RestApi'
    Properties:
      Body:
        openapi: 3.0.1
        info:
          title: Parameter mapping example
          version: "2025-01-15T19:21:35Z"
        paths:
          /:
            post:
              responses:
                "200":
                  description: 200 response
                  headers:
```

```
                    item:
                      schema:
                        type: string
                    location:
                      schema:
                        type: string
                    id:
                      schema:
                        type: string
              x-amazon-apigateway-integration:
                type: http
                httpMethod: GET
                uri: http://petstore-demo-endpoint.execute-api.com/petstore/pets
                responses:
                  default:
                    statusCode: "200"
                    responseParameters:
                      method.response.header.id: integration.response.header.x-app-
id
                      method.response.header.location:
 integration.response.body.redirect.url
                      method.response.header.item:
 integration.response.multivalueheader.item
                requestTemplates:
                  application/json: '{"statusCode": 200}'
                passthroughBehavior: when_no_templates
                timeoutInMillis: 29000
  ApiGatewayDeployment:
    Type: 'AWS::ApiGateway::Deployment'
    DependsOn: Api
    Properties:
      RestApiId: !Ref Api
  ApiGatewayDeployment20250219:
    Type: 'AWS::ApiGateway::Deployment'
    DependsOn: Api
    Properties:
      RestApiId: !Ref Api
  Stage:
    Type: 'AWS::ApiGateway::Stage'
    Properties:
       DeploymentId: !Ref ApiGatewayDeployment20250219
       RestApiId: !Ref Api
       StageName: prod
```

OpenAPI

La définition d'OpenAPI suivante mappe la réponse d'intégration à la réponse de méthode.

```
{
  "openapi" : "3.0.1",
  "info" : {
    "title" : "Parameter mapping example",
    "version" : "2025-01-15T19:21:35Z"
  },
  "paths" : {
    "/" : {
      "post" : {
        "responses" : {
          "200" : {
            "description" : "200 response",
            "headers" : {
              "item" : {
                "schema" : {
                  "type" : "string"
                }
              },
              "location" : {
                "schema" : {
                  "type" : "string"
                }
              },
              "id" : {
                "schema" : {
                  "type" : "string"
                }
              }
            }
          }
        },
        "x-amazon-apigateway-integration" : {
          "type" : "http",
          "httpMethod" : "GET",
          "uri" : "http://petstore-demo-endpoint.execute-api.com/petstore/pets",
          "responses" : {
            "default" : {
              "statusCode" : "200",
              "responseParameters" : {
```

```
                "method.response.header.id" : "integration.response.header.x-app-
id",
                "method.response.header.location" :
 "integration.response.body.redirect.url",
                "method.response.header.item" :
 "integration.response.multivalueheader.item"
              }
            }
          },
          "requestTemplates" : {
            "application/json" : "{\"statusCode\": 200}"
          },
          "passthroughBehavior" : "when_no_templates",
          "timeoutInMillis" : 29000
        }
      }
    }
  }
}
```

Référence de la source du mappage de paramètres pour les API REST dans API Gateway

Lorsque vous créez un mappage de paramètres, vous spécifiez les paramètres de demande de méthode ou de réponse d'intégration à modifier, et indiquez comment modifier ces paramètres.

Le tableau suivant indique les paramètres de requête de méthode que vous pouvez mapper, ainsi que l'expression permettant ce mappage. Dans ces expressions, *name* est le nom d'un paramètre de demande de méthode. Par exemple, pour mapper le paramètre d'en-tête de demande `puppies`, utilisez l'expression `method.request.header.puppies`. Votre expression doit respecter le modèle d'expression régulière `'^[a-zA-Z0-9._$-]+$]'`. Vous pouvez utiliser le mappage de paramètres dans votre demande d'intégration pour des intégrations avec ou sans proxy.

Source de données mappée	Expression de mappage
Chemin de la demande de méthode	`method.request.path.` *name*
Chaîne de requête de la demande de méthode	`method.request.querystring.` *name*
Multi-value chaîne de requête de méthode	`method.request.multivaluequerystring.` *name*

Source de données mappée	Expression de mappage
En-tête de la demande de méthode	`method.request.header.` *`name`*
Multi-value en-tête de demande de méthode	`method.request.multivalueheader.` *`name`*
Corps de la demande de méthode	`method.request.body`
corps de la demande de méthode (JsonPath)	`method.request.body.` *`JSONPath_EXPRESSION`* . *`JSONPath_EXPRESSION`* est une expression JSONPath pour un champ JSON du corps d'une demande. Pour plus d'informations sur les expressions, consultez Expressions JSONPath.
Variables d'étape	`stageVariables.` *`name`*
Variables de contexte	`context.`*`name`* Le nom doit être l'une des variables de contexte prises en charge.
Valeur statique	*`'static_value'`* . *`static_value`* Il s'agit d'une chaîne littérale qui doit être placée entre guillemets simples. Par exemple, `'https://www.example.com'` .

Le tableau suivant indique les paramètres de réponse d'intégration que vous pouvez mapper, ainsi que l'expression permettant ce mappage. Dans ces expressions, *`name`* est le nom d'un paramètre de réponse d'intégration. Vous pouvez mapper les paramètres d'en-tête d'une réponse de méthode à partir de n'importe quel en-tête ou corps de réponse d'intégration, de variables $context ou de valeurs statiques. Pour utiliser le mappage de paramètres pour une réponse d'intégration, il vous faut une intégration sans proxy.

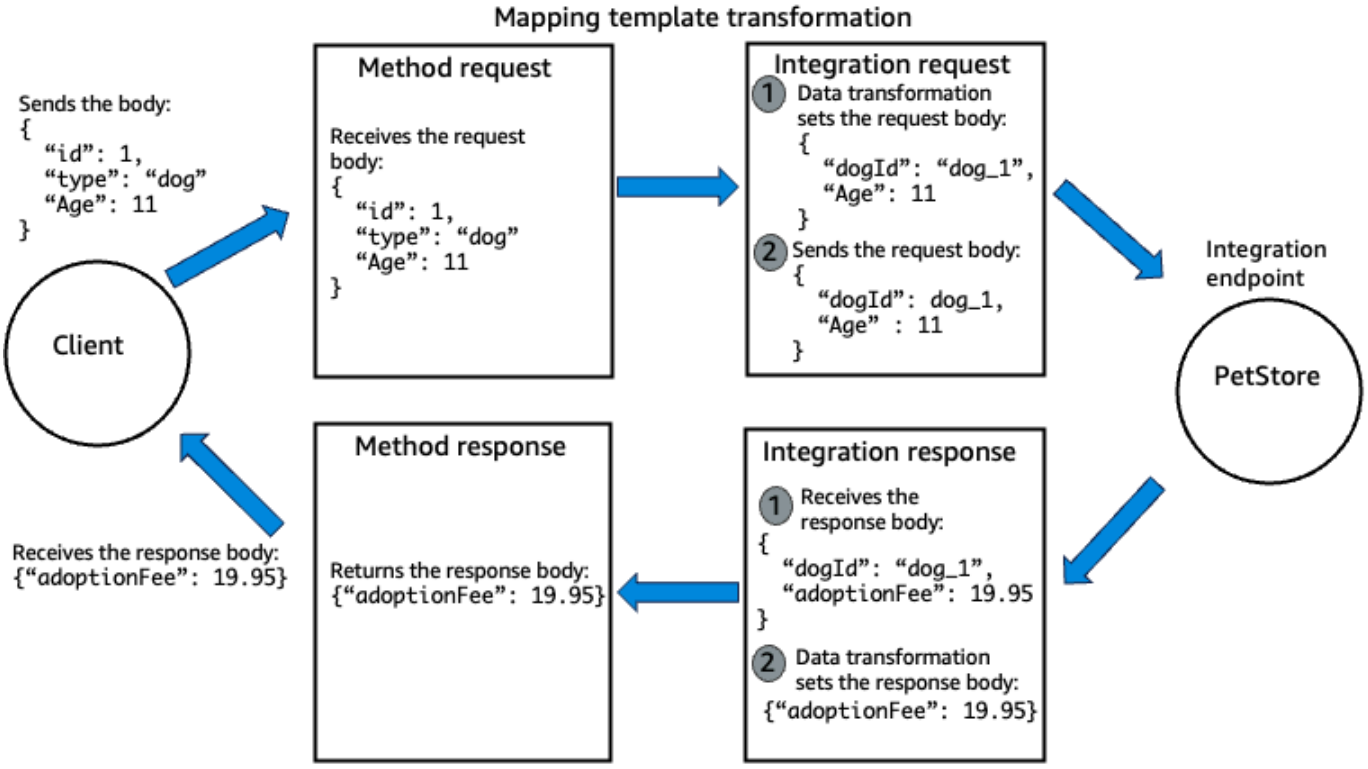
Source de données mappée	Expression de mappage
En-tête de réponse d'intégration	`integration.response.header.` *`name`*
En-tête de réponse d'intégration	`integration.response.multivalueheader.` *`name`*
Corps de réponse intégration	`integration.response.body`
Organisme de réponse à l'intégration (JsonPath)	`integration.response.body.` *`JSONPath_EXPRESSION`* *`JSONPath_EXPRESSION`* est une expression JSONPath pour un champ JSON du corps d'une réponse. Pour plus d'informations sur les expressions, consultez Expressions JSONPath.
Variable d'étape	`stageVariables.` *`name`*
Variable de contexte	`context.`*`name`* Le nom doit être l'une des variables de contexte prises en charge.
Valeur statique	*`'static_value'`* *`static_value`* Il s'agit d'une chaîne littérale qui doit être placée entre guillemets simples. Par exemple, `'https://www.example.com'` .

Transformations de modèles de mappage pour les API REST dans API Gateway

Une transformation de modèle de mappage utilise un modèle de mappage pour modifier votre demande d'intégration ou votre réponse d'intégration. Un modèle de mappage est un script exprimé en langage VTL (Velocity Template Language) appliqué à des données utiles à l'aide d'expressions JSONPath en fonction de l'en-tête `Content-type`. Vous utilisez des modèles de

mappage lorsque vous transformez des modèles de mappage. Cette section décrit les informations conceptuelles relatives aux modèles de mappage.

Le schéma suivant montre le cycle de vie des demandes pour une `POST /pets` ressource intégrée à un point de terminaison PetStore d'intégration. Dans cette API, un utilisateur envoie des données sur un animal de compagnie, et le point de terminaison d'intégration renvoie les frais d'adoption relatifs à cet animal. Dans le cycle de vie de cette demande, les transformations du modèle de mappage filtrent le corps de la demande vers le point de terminaison d'intégration, et filtrent le corps de réponse provenant du point de terminaison d'intégration.



Les sections suivantes expliquent le cycle de vie des demandes et réponses.

Demande de méthode et demande d'intégration

Voici le corps de la demande envoyé à la demande de méthode dans l'exemple précédent :

```
POST /pets
    HTTP/1.1
    Host:abcd1234.us-west-2.amazonaws.com
    Content-type: application/json

  {
```

```
    "id": 1,
    "type": "dog",
    "Age": 11,
  }
```

Le corps de cette demande n'est pas au format approprié pour être utilisé par le point de terminaison d'intégration. API Gateway transforme donc le modèle de mappage. API Gateway effectue uniquement une transformation de modèle de mappage car un modèle de mappage est défini pour le Content-Type `application/json`. Si vous ne définissez pas de modèle de mappage pour l'API Gateway Content-Type, par défaut, transmet le corps de la demande d'intégration au point de terminaison d'intégration. Pour modifier ce comportement, consultez the section called "Comportement des demandes de méthode pour les données utiles sans modèle de mappage".

Le modèle de mappage suivant transforme les données de la demande de méthode dans la demande d'intégration avant qu'elles ne soient envoyées au point de terminaison d'intégration :

```
#set($inputRoot = $input.path('$'))
  {
    "dogId" : "dog_"$elem.id,
    "Age": $inputRoot.Age
  }
```

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. `dog` est une concaténation de la valeur de l'utilisateur `id` et d'une valeur de chaîne.
3. `Age` provient du corps de la demande de méthode.

La sortie suivante est ensuite transmise au point de terminaison d'intégration :

```
{
    "dogId" : "dog_1",
    "Age": 11
  }
```

Réponse d'intégration et réponse de méthode

Une fois la demande envoyée avec succès au point de terminaison d'intégration, celui-ci envoie une réponse à la réponse d'intégration d'API Gateway. Voici un exemple de données de sortie provenant du point de terminaison d'intégration :

```
{
    "dogId" : "dog_1",
    "adoptionFee": 19.95,
}
```

La réponse de méthode attend des données utiles différentes de celles renvoyées par la réponse d'intégration. API Gateway transforme le modèle de mappage. API Gateway effectue uniquement une transformation de modèle de mappage car un modèle de mappage est défini pour le Content-Type `application/json`. Si vous ne définissez pas de modèle de mappage pour API Gateway Content-Type, par défaut, le corps passe par la réponse d'intégration à la réponse de méthode. Pour modifier ce comportement, consultez the section called “Comportement des demandes de méthode pour les données utiles sans modèle de mappage”.

```
#set($inputRoot = $input.path('$'))
  {
    "adoptionFee" : $inputRoot.adoptionFee,
  }
```

La sortie suivante est envoyée à la réponse de méthode :

```
{"adoptionFee": 19.95}
```

Ceci conclut l'exemple de transformation de modèle de mappage. Nous vous recommandons, dans la mesure du possible, d'utiliser une intégration de proxy pour transformer vos données au lieu de transformer le modèle de mappage. Pour de plus amples informations, veuillez consulter the section called “Choix d'un type d'intégration d'API”.

Comportement des demandes de méthode pour les données utiles sans modèle de mappage pour les API REST dans API Gateway

Si votre demande de méthode comporte des données utiles et qu'aucun modèle de mappage n'est défini pour l'en-tête `Content-Type`, vous pouvez choisir de transmettre les données utiles de la demande fournie par le client au backend par l'intermédiaire de la demande d'intégration, sans recourir à une transformation. Ce processus est appelé transfert direct d'intégration.

Le comportement de transmission d'une demande entrante est déterminé par ce paramètre. Trois options sont disponibles :

Lorsqu'aucun modèle ne correspond à l' Content-Type en-tête 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 comprenant des données utiles et un type de contenu qui ne correspond pas au modèle de mappage défini sera rejetée avec une réponse HTTP 415 Unsupported Media Type.

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 transmission possibles.

Exemple 1 : un modèle de mappage est défini dans la demande d'intégration pour le type de contenu `application/json`.

Content-type	Option de transmission	Comportement
Aucune La valeur par défaut d'API Gateway est `application/json`	`WHEN_NO_MATCH`	La charge utile de la demande est transformée à l'aide du modèle.
Aucune La valeur par défaut d'API Gateway est `application/json`	`WHEN_NO_TEMPLATES`	La charge utile de la demande est transformée à l'aide du modèle.
Aucune La valeur par défaut d'API Gateway est `application/json`	`NEVER`	La charge utile de la demande est transformée à l'aide du modèle.
`application/json`	`WHEN_NO_MATCH`	La charge utile de la demande est transformée à l'aide du modèle.
`application/json`	`WHEN_NO_TEMPLATES`	La charge utile de la demande est transformée à l'aide du modèle.
`application/json`	`NEVER`	La charge utile de la demande est transformée à l'aide du modèle.
`application/xml`	`WHEN_NO_MATCH`	La charge utile de la demande n'est pas transformée et est envoyée en l'état au backend.
`application/xml`	`WHEN_NO_TEMPLATES`	La demande est rejetée avec une réponse HTTP

Content-type	Option de transmission	Comportement
		`415 Unsupported Media Type`.
`application/xml`	`NEVER`	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`.

Content-type	Option de transmission	Comportement
Aucune La valeur par défaut d'API Gateway est `application/json`	`WHEN_NO_MATCH`	La charge utile de la demande n'est pas transformée et est envoyée en l'état au backend.
Aucune La valeur par défaut d'API Gateway est `application/json`	`WHEN_NO_TEMPLATES`	La demande est rejetée avec une réponse HTTP `415 Unsupported Media Type`.
Aucune La valeur par défaut d'API Gateway est `application/json`	`NEVER`	La demande est rejetée avec une réponse HTTP `415 Unsupported Media Type`.
`application/json`	`WHEN_NO_MATCH`	La charge utile de la demande n'est pas transformée et est envoyée en l'état au backend.
`application/json`	`WHEN_NO_TEMPLATES`	La demande est rejetée avec une réponse HTTP

Content-type	Option de transmission	Comportement
		`415 Unsupported Media Type`.
`application/json`	`NEVER`	La demande est rejetée avec une réponse HTTP `415 Unsupported Media Type`.
`application/xml`	`WHEN_NO_MATCH`	La charge utile de la demande est transformée à l'aide du modèle.
`application/xml`	`WHEN_NO_TEMPLATES`	La charge utile de la demande est transformée à l'aide du modèle.
`application/xml`	`NEVER`	La charge utile de la demande est transformée à l'aide du modèle.

Exemple 3 : aucun modèle de mappage n'est défini dans la demande d'intégration.

Content-type	Option de transmission	Comportement
Aucune La valeur par défaut d'API Gateway est `application/json`	`WHEN_NO_MATCH`	La charge utile de la demande n'est pas transformée et est envoyée en l'état au backend.
Aucune La valeur par défaut d'API Gateway est `application/json`	`WHEN_NO_TEMPLATES`	La charge utile de la demande n'est pas transformée et est envoyée en l'état au backend.

Content-type	Option de transmission	Comportement
Aucune La valeur par défaut d'API Gateway est `application/json`	`NEVER`	La demande est rejetée avec une réponse HTTP `415 Unsupported Media Type`.
`application/json`	`WHEN_NO_MATCH`	La charge utile de la demande n'est pas transformée et est envoyée en l'état au backend.
`application/json`	`WHEN_NO_TEMPLATES`	La charge utile de la demande n'est pas transformée et est envoyée en l'état au backend.
`application/json`	`NEVER`	La demande est rejetée avec une réponse HTTP `415 Unsupported Media Type`.
`application/xml`	`WHEN_NO_MATCH`	La charge utile de la demande n'est pas transformée et est envoyée en l'état au backend.
`application/xml`	`WHEN_NO_TEMPLATES`	La charge utile de la demande n'est pas transformée et est envoyée en l'état au backend.
`application/xml`	`NEVER`	La demande est rejetée avec une réponse HTTP `415 Unsupported Media Type`.

Autre exemple de modèle de mappage pour les API REST dans API Gateway

L'exemple suivant montre une API d'album photo API Gateway qui utilise des modèles de mappage pour transformer les données de la demande d'intégration et de la réponse d'intégration. Elle utilise

également des modèles de données pour définir les données utiles de la demande de méthode et de la réponse d'intégration. Pour plus d'informations sur les modèles de données, consultez Modèles de données pour les API REST.

Demande de méthode et demande d'intégration

Voici un modèle qui définit le corps de la demande de méthode. Ce modèle d'entrée exige le chargement d'une page de photos et un minimum de 10 photos pour chaque page. Vous pouvez utiliser ce modèle d'entrée pour générer un kit SDK ou utiliser le validateur de demande pour votre API. Dans ce dernier cas, si le corps de la demande de méthode ne respecte pas la structure de données du modèle, API Gateway fait échouer la demande.

```
{
  "$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" }
```

```
            }
          }
        }
      }
    }
  }
}
```

Voici un exemple de corps de demande de méthode qui respecte la structure de données du modèle de données précédent.

```
{
  "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
```

```
      }
    ]
  }
}
```

Dans cet exemple, si le précédent corps de la demande de méthode a été soumis par le client, ce modèle de mappage transforme les données utiles pour qu'elles correspondent au format requis par le point de terminaison d'intégration.

```
#set($inputRoot = $input.path('$'))
{
  "photos": [
#foreach($elem in $inputRoot.photos.photo)
    {
      "id": "$elem.id",
      "photographedBy": "$elem.photographer_first_name $elem.photographer_last_name",
      "title": "$elem.title",
      "ispublic": $elem.ispublic,
      "isfriend": $elem.isfriend,
      "isfamily": $elem.isfamily
    }#if($foreach.hasNext),#end

#end
  ]
}
```

L'exemple suivant montre les données de sortie de la transformation :

```
{
  "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
    }
  ]
}
```

Ces données sont envoyées à la demande d'intégration, puis au point de terminaison d'intégration.

Réponse d'intégration et réponse de méthode

Voici un exemple de modèle de sortie pour les données photo du point de terminaison d'intégration. 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 le transfert de la sortie dans 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" }
        }
      }
    }
  }
}
```

Le point de terminaison d'intégration peut ne pas répondre par une réponse conforme à la structure de données de ce modèle. Par exemple, la réponse d'intégration peut ressembler à ce qui suit :

```
  "photos": [
    {
      "id": "12345678901",
      "photographedBy": "Saanvi Sarkar",
      "title": "Sample photo 1",
      "description": "My sample photo 1",
      "public": true,
      "friend": false,
      "family": false
    },
    {
      "id": "23456789012",
      "photographedBy": "Richard Roe",
      "title": "Sample photo 2",
      "description": "My sample photo 1",
      "public": true,
      "friend": false,
      "family": false
    }
  ]
}
```

L'exemple de modèle de mappage suivant transforme les données de la réponse d'intégration dans le format attendu par la réponse de méthode :

```
#set($inputRoot = $input.path('$'))
{
  "photos": [
#foreach($elem in $inputRoot.photos.photo)
    {
      "id": "$elem.id",
      "photographedBy": "$elem.photographer_first_name $elem.photographer_last_name",
      "title": "$elem.title",
      "ispublic": $elem.public,
      "isfriend": $elem.friend,
      "isfamily": $elem.family
    }#if($foreach.hasNext),#end

#end
  ]
}
```

L'exemple suivant montre les données de sortie de la transformation :

```
{
  "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
    }
  ]
}
```

Ces données sont envoyées à la réponse de méthode, puis renvoyées au client.

Remplacement des codes d'état et des paramètres de demande et de réponse des API REST dans API Gateway

Vous pouvez recourir à la transformation de modèle de mappage pour remplacer n'importe quel type de paramètre de demande, d'en-tête de réponse ou de code d'état de réponse. Vous allez utiliser un modèle de mappage pour :

- Réaliser des mappages de paramètres plusieurs-à-un
- Remplacer les paramètres une fois que les mappages API Gateway standard ont été appliqués
- Mapper les paramètres de façon conditionnelle, en fonction du contenu du corps ou d'autres valeurs de paramètres
- Créer des paramètres par programmation
- Remplacer les codes de statut renvoyés par votre point de terminaison d'intégration

Les remplacements sont définitifs. Un remplacement ne peut être appliqué qu'une seule fois à chaque paramètre. Si vous tentez de remplacer le même paramètre à plusieurs reprises, API

Gateway renvoie une réponse 5XX. 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. Le modèle n'est appliqué qu'après l'analyse de l'ensemble du modèle.

Exemple 1 : remplacement du code d'état en fonction du corps de la réponse d'intégration

L'exemple suivant utilise l'exemple d'API pour remplacer le code d'état en fonction du corps de la réponse d'intégration.

Console de gestion AWS

Pour remplacer un code d'état en fonction du corps de la réponse d'intégration

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 Création.
4. Sous Détails sur l'API, choisissez Exemple d'API.
5. Sélectionnez Create API (Créer une API).

   API Gateway crée un exemple d'API PetStore. Pour récupérer des informations sur un animal de compagnie, vous allez utiliser la demande de méthode d'API `GET /pets/{petId}`, où `{petId}` est un paramètre de chemin correspondant au numéro d'identification de l'animal.

   Dans cet exemple, vous allez remplacer le code de réponse de la méthode `GET` par `400` lorsqu'une condition d'erreur est identifiée.
6. Dans l'arborescence Ressources, sous `GET`, choisissez la méthode sous `/{petId}`.
7. Tout d'abord, vous allez tester l'implémentation actuelle de l'API.

   Choisissez l'onglet Test. Vous devrez peut-être choisir la flèche droite pour afficher l'onglet.
8. Pour petId, saisissez **`-1`**, puis choisissez Tester.

   Corps de la réponse indique une erreur de valeur hors plage :

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

```
    }
  ]
}
```

De plus, la dernière ligne sous Journaux se termine par : `Method completed with status: 200`.

L'intégration a été réalisée avec succès, mais une erreur s'est produite. Vous allez maintenant remplacer le code d'état en fonction de la réponse d'intégration.

9. Dans l'onglet Réponse d'intégration, pour Par défaut - Réponse, choisissez Modifier.
10. Choisissez Modèles de mappage.
11. Sélectionnez Add mapping template.
12. Pour Type de contenu, entrez **`application/json`**.
13. 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
```

Ce modèle de mappage utilise la variable `$context.responseOverride.status` pour remplacer le code d'état par `400` si la réponse d'intégration contient la chaîne `error`.

14. Choisissez Enregistrer.
15. Choisissez l'onglet Test.
16. Pour petId, saisissez **`-1`**.
17. Dans les résultats, le Response Body (Corps de la réponse) indique une erreur de valeur hors plage :

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

Cependant, la dernière ligne sous Journaux se termine maintenant par : `Method completed with status: 400`.

CloudFormation

Dans cet exemple, vous allez utiliser la propriété body pour importer un fichier de définition OpenAPI dans API Gateway.

```
AWSTemplateFormatVersion: 2010-09-09
Resources:
  Api:
    Type: 'AWS::ApiGateway::RestApi'
    Properties:
      Body:
        openapi: 3.0.1
        info:
          title: PetStore Example 1
          description: Example pet store API.
          version: "2025-01-14T00:13:18Z"
        paths:
          /pets/{petId}:
            get:
              parameters:
                - name: petId
                  in: path
                  required: true
                  schema:
                    type: string
              responses:
                "200":
                  description: 200 response
              x-amazon-apigateway-integration:
                httpMethod: GET
                uri: http://petstore.execute-api.us-east-1.amazonaws.com/petstore/
pets/{petId}
                responses:
                  default:
                    statusCode: "200"
                    responseTemplates:
                      application/json: |-
                        #set($inputRoot = $input.path('$'))
                        $input.json("$")
```

```
                        #if($inputRoot.toString().contains("error"))
                        #set($context.responseOverride.status = 400)
                        #end
                requestParameters:
                  integration.request.path.petId: method.request.path.petId
                passthroughBehavior: when_no_match
                type: http
        components:
          schemas:
            Pet:
              type: object
              properties:
                id:
                  type: integer
                type:
                  type: string
                price:
                  type: number
  ApiGatewayDeployment:
    Type: 'AWS::ApiGateway::Deployment'
    DependsOn: Api
    Properties:
      RestApiId: !Ref Api
  ApiGatewayDeployment20250219:
    Type: 'AWS::ApiGateway::Deployment'
    DependsOn: Api
    Properties:
      RestApiId: !Ref Api
  Stage:
    Type: 'AWS::ApiGateway::Stage'
    Properties:
       DeploymentId: !Ref ApiGatewayDeployment20250219
       RestApiId: !Ref Api
       StageName: prod
```

OpenAPI

La définition OpenAPI suivante crée la ressource `GET pets/{petId}` et remplace le code d'état en fonction du corps de la réponse d'intégration.

```
{
  "openapi" : "3.0.1",
  "info" : {
```

```
    "title" : "PetStore Example 1",
    "description" : "Example pet store API.",
    "version" : "2025-01-14T00:13:18Z"
  },
  "paths" : {
    "/pets/{petId}" : {
      "get" : {
        "parameters" : [ {
          "name" : "petId",
          "in" : "path",
          "required" : true,
          "schema" : {
            "type" : "string"
          }
        } ],
        "responses" : {
          "200" : {
            "description" : "200 response"
          }
        },
        "x-amazon-apigateway-integration" : {
          "httpMethod" : "GET",
          "uri" : "http://petstore.execute-api.us-east-1.amazonaws.com/petstore/
pets/{petId}",
          "responses" : {
            "default" : {
              "statusCode" : "200",
              "responseTemplates" : {
                "application/json" : "#set($inputRoot = $input.path('$'))\n
$input.json(\"$\")\n#if($inputRoot.toString().contains(\"error
\"))\n#set($context.responseOverride.status = 400)\n#end"
              }
            }
          },
          "requestParameters" : {
            "integration.request.path.petId" : "method.request.path.petId"
          },
          "passthroughBehavior" : "when_no_match",
          "type" : "http"
        }
      }
    }
  },
  "components" : {
```

```
    "schemas" : {
      "Pet" : {
        "type" : "object",
        "properties" : {
          "id" : {
            "type" : "integer"
          },
          "type" : {
            "type" : "string"
          },
          "price" : {
            "type" : "number"
          }
        }
      }
    }
  }
}
```

Exemple 2 : remplacement de l'en-tête de la demande et création de nouveaux en-têtes

L'exemple suivant utilise l'exemple d'API pour remplacer l'en-tête de la demande et créer de nouveaux en-têtes.

Console de gestion AWS

Pour remplacer l'en-tête de demande d'une méthode en créant un nouvel en-tête

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Choisissez l'exemple d'API que vous avez créé dans le tutoriel précédent. Le nom de l'API doit être PetStore.
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 Nom, saisissez **`header1`**.
7. Choisissez Ajouter un en-tête, puis créez un second en-tête appelé **`header2`**.
8. Choisissez Enregistrer.

Vous pouvez maintenant combiner ces en-têtes en une seule valeur d'en-tête à l'aide d'un modèle de mappage.

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($header1Override = "pets")
#set($header3Value = "$input.params('header1')$input.params('header2')")
$input.json("$")
#set($context.requestOverride.header.header3 = $header3Value)
#set($context.requestOverride.header.header1 = $header1Override)
#set($context.requestOverride.header.multivalueheader=[$header1Override,
 $header3Value])
```

    Ce modèle de mappage remplace `header1` par la chaîne `pets`, et crée un en-tête à valeurs multiples appelé `$header3Value`, qui combine `header1` et `header2`.

12. Choisissez Enregistrer.
13. Choisissez l'onglet Test.
14. Sous En-têtes, copiez le code suivant :

```
header1:header1Val
header2:header2Val
```

15. Sélectionnez Test.

    Dans Journaux, vous devriez voir une entrée incluant ce texte :

```
Endpoint request headers: {header3=header1Valheader2Val,
header2=header2Val, header1=pets, x-amzn-apigateway-api-id=api-id,
Accept=application/json, multivalueheader=pets,header1Valheader2Val}
```

CloudFormation

Dans cet exemple, vous allez utiliser la propriété body pour importer un fichier de définition OpenAPI dans API Gateway.

```
AWSTemplateFormatVersion: 2010-09-09
Resources:
  Api:
    Type: 'AWS::ApiGateway::RestApi'
    Properties:
      Body:
        openapi: 3.0.1
        info:
          title: PetStore Example 2
          description: Example pet store API.
          version: "2025-01-14T00:36:18Z"
        paths:
          /pets:
            get:
              parameters:
                - name: header2
                  in: header
                  schema:
                    type: string
                - name: page
                  in: query
                  schema:
                    type: string
                - name: type
                  in: query
                  schema:
                    type: string
                - name: header1
                  in: header
                  schema:
                    type: string
              responses:
                "200":
                  description: 200 response
              x-amazon-apigateway-integration:
                httpMethod: GET
                uri: http://petstore.execute-api.us-east-1.amazonaws.com/petstore/
pets
                responses:
```

```
                default:
                  statusCode: "200"
              requestParameters:
                integration.request.header.header1: method.request.header.header1
                integration.request.header.header2: method.request.header.header2
                integration.request.querystring.page:
method.request.querystring.page
                integration.request.querystring.type:
method.request.querystring.type
              requestTemplates:
                application/json: |-
                  #set($header1Override = "pets")
                  #set($header3Value =
"$input.params('header1')$input.params('header2')")
                  $input.json("$")
                  #set($context.requestOverride.header.header3 = $header3Value)
                  #set($context.requestOverride.header.header1 = $header1Override)

#set($context.requestOverride.header.multivalueheader=[$header1Override,
$header3Value])
              passthroughBehavior: when_no_match
              type: http
      components:
        schemas:
          Pet:
            type: object
            properties:
              id:
                type: integer
              type:
                type: string
              price:
                type: number
 ApiGatewayDeployment:
   Type: 'AWS::ApiGateway::Deployment'
   DependsOn: Api
   Properties:
     RestApiId: !Ref Api
 ApiGatewayDeployment20250219:
   Type: 'AWS::ApiGateway::Deployment'
   DependsOn: Api
   Properties:
     RestApiId: !Ref Api
 Stage:
```

```
    Type: 'AWS::ApiGateway::Stage'
    Properties:
       DeploymentId: !Ref ApiGatewayDeployment20250219
       RestApiId: !Ref Api
       StageName: prod
```

OpenAPI

La définition OpenAPI suivante crée la ressource `GET pets`, remplace l'en-tête de la demande et crée de nouveaux en-têtes.

```
{
  "openapi" : "3.0.1",
  "info" : {
    "title" : "PetStore Example 2",
    "description" : "Example pet store API.",
    "version" : "2025-01-14T00:36:18Z"
  },
  "paths" : {
    "/pets" : {
      "get" : {
        "parameters" : [ {
          "name" : "header2",
          "in" : "header",
          "schema" : {
            "type" : "string"
          }
        }, {
          "name" : "page",
          "in" : "query",
          "schema" : {
            "type" : "string"
          }
        }, {
          "name" : "type",
          "in" : "query",
          "schema" : {
            "type" : "string"
          }
        }, {
          "name" : "header1",
          "in" : "header",
          "schema" : {
```

```
            "type" : "string"
          }
        } ],
        "responses" : {
          "200" : {
            "description" : "200 response"
          }
        },
        "x-amazon-apigateway-integration" : {
          "httpMethod" : "GET",
          "uri" : "http://petstore.execute-api.us-east-1.amazonaws.com/petstore/
pets",
          "responses" : {
            "default" : {
              "statusCode" : "200"
            }
          },
          "requestParameters" : {
            "integration.request.header.header1" : "method.request.header.header1",
            "integration.request.header.header2" : "method.request.header.header2",
            "integration.request.querystring.page" :
 "method.request.querystring.page",
            "integration.request.querystring.type" :
 "method.request.querystring.type"
          },
          "requestTemplates" : {
            "application/json" : "#set($header1Override = \"pets
\")\n#set($header3Value = \"$input.params('header1')$input.params('header2')\")\n
$input.json(\"$\")\n#set($context.requestOverride.header.header3
 = $header3Value)\n#set($context.requestOverride.header.header1 =
 $header1Override)\n#set($context.requestOverride.header.multivalueheader=[$header1Override,
 $header3Value])"
          },
          "passthroughBehavior" : "when_no_match",
          "type" : "http"
        }
      }
    }
  }
}
```

Pour remplacer un modèle de mappage, ajoutez une ou plusieurs des variables `$context` suivantes. Pour obtenir la liste des variables `$context`, consultez the section called “Variables context pour les transformations de données”.

Tutoriel : Modifier la demande et la réponse d'intégration pour les intégrations vers AWS services

Le didacticiel suivant explique comment utiliser les transformations de modèles de mappage pour configurer des modèles de mappage afin de 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
- Configurez la transformation des données à l'aide du AWS INTERFACE DE LIGNE DE COMMANDE (CLI)
- Transformation des données terminée CloudFormation modèle

Configuration d’une transformation de données à l’aide de la console API Gateway

Dans ce didacticiel, vous allez créer une API incomplète et une table DynamoDB à l’aide du fichier .zip data-transformation-tutorial-console.zip suivant. 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 the section called “Transformations de modèles 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 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 un CloudFormation pile

1. Ouvrez la 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 CloudFormation vous pouvez créer des ressources IAM dans votre compte.
9. Choisissez Suivant, puis Soumettre.

CloudFormation fournit les ressources spécifiées dans le modèle. La fin du provisionnement de vos ressources peut prendre quelques minutes. Lorsque le statut de votre 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 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 the section called “Transformations de modèles 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 à partir 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 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` 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 Content-Types qui n'ont pas de modèle de mappage.

4. Choisissez Modèles de mappage.
5. Type de contenu est défini sur `application/json`. Cela signifie que les types de contenu qui ne le sont pas application/json seront rejetés par l'API. Pour plus d'informations sur les comportements de transmission d'intégration, consultez Comportement des demandes de méthode pour les données utiles sans modèle de mappage pour les API REST dans API Gateway.
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 de plus amples informations, veuillez consulter Validation des demandes pour les API REST dans API Gateway.

7. Choisissez Enregistrer pour enregistrer votre modèle de mappage.

Pour ajouter une méthode et une réponse d'intégration à partir de la méthode **`POST`**

Ils 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 Exemples de mappage de paramètres pour les API REST dans 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 Content-type, 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 Variables context pour les transformations de données.
5. Choisissez Enregistrer pour enregistrer votre modèle de mappage.

Testez 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 https://console.aws.amazon.com/dynamodb/à l'adresse pour vérifier que l'exemple d'élément figure dans votre tableau.

Pour supprimer un CloudFormation pile

1. Ouvrez la CloudFormation console à l'adresse https://console.aws.amazon.com/cloudformation.
2. Sélectionnez votre CloudFormation pile.
3. Choisissez Supprimer, puis confirmez votre choix.

Configurez la transformation des données à l'aide du AWS INTERFACE DE LIGNE DE COMMANDE (CLI)

Dans ce didacticiel, vous allez créer une API incomplète et une table DynamoDB à l'aide du fichier .zip data-transformation-tutorial-cli.zip suivant. 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 the section called "Transformations de modèles 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 un CloudFormation pile

Téléchargez et décompressez le modèle de création d'application pour 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. Pour obtenir une Windows-integrated version d'Ubuntu et de Bash, installez le sous-système Windows pour Linux. 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 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. 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 CloudFormation pile.

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

3. Lorsque le statut de votre CloudFormation pile est défini`StackStatus: "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 the section called “Transformations de modèles 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. 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 un CloudFormation pile

- Utilisez la commande suivante pour supprimer vos CloudFormation ressources.

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

Transformation des données terminée CloudFormation modèle

L'exemple suivant est un 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 the section called “Transformations de modèles 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.

Exemple CloudFormation modèle

```
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
```

Exemples d'utilisation de variables pour transformer des modèles de mappage pour API Gateway

Les exemples suivants montrent comment utiliser les variables `$context`, `input` et `util` dans des modèles de mappage. Vous pouvez utiliser une intégration simulée ou une intégration Lambda sans proxy qui renvoie l'événement d'entrée dans API Gateway. Pour obtenir la liste de toutes les variables prises en charge pour les transformations de données, consultez the section called "Variables pour les transformations de données".

Exemple 1 : Transmettre plusieurs variables **`$context`** au point de terminaison d'intégration

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 :

```
{
    "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"
}
```

Le résultat 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'
}
```

L'une des variables est une clé d'API. Dans cet exemple, on suppose que la méthode exige une clé d'API.

Exemple 2 : transmission de tous les paramètres de la demande au point de terminaison d'intégration via des données utiles JSON

L'exemple suivant transmet tous les paramètres de la demande, y compris les paramètres `path`, `querystring` et `header`, au point de terminaison d'intégration via des données utiles 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
  }
}
```

Si une demande possède les paramètres d'entrée suivants :

- Paramètre de chemin nommé `myparam`
- Paramètres des chaînes de demande `querystring1=value1,value2`
- En-têtes `"header1" : "value1"`.

Le résultat de ce modèle de mappage doit ressembler à ce qui suit :

```
{"params":{"path":{"example2":"myparamm"},"querystring":
{"querystring1":"value1,value2"},"header":{"header1":"value1"}}}
```

Exemple 3 : transmission d'une sous-section d'une demande de méthode au point de terminaison d'intégration

L'exemple suivant utilise le paramètre d'entrée `name` pour récupérer uniquement le paramètre `name`, et le paramètre d'entrée `input.json('$')` pour récupérer l'intégralité du corps de la demande de méthode :

```
{
    "name" : "$input.params('name')",
    "body" : $input.json('$')
}
```

Pour une demande qui inclut les paramètres de chaîne de demande `name=Bella&type=dog` et le corps suivant :

```
{
    "Price" : "249.99",
    "Age": "6"
}
```

Le résultat de ce modèle de mappage doit ressembler à ce qui suit :

```
{
    "name" : "Bella",
```

```
    "body" : {"Price":"249.99","Age":"6"}
}
```

Ce modèle de mappage supprime le paramètre de chaîne de requête `type=dog`.

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 que les données d'entrée JSON puissent être analysées correctement.

L'exemple précédent avec application de `$util.escapeJavaScript($input.json('$'))` 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 4 : utilisation d'une expression JSONPath pour transmettre une sous-section d'une demande de méthode au point de terminaison d'intégration

L'exemple suivant utilise les expressions JSONPath pour récupérer uniquement le paramètre d'entrée `name` et le paramètre `Age` du corps de la demande :

```
{
    "name" : "$input.params('name')",
    "body" : $input.json('$.Age')
}
```

Pour une demande qui inclut les paramètres de chaîne de demande `name=Bella&type=dog` et le corps suivant :

```
{
    "Price" : "249.99",
```

```
    "Age": "6"
}
```

Le résultat de ce modèle de mappage doit ressembler à ce qui suit :

```
{
    "name" : "Bella",
    "body" : "6"
}
```

Ce modèle de mappage supprime le paramètre de chaîne de requête `type=dog` et le champ `Price` du corps.

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 que les données d'entrée JSON puissent être analysées correctement.

L'exemple précédent avec application de `$util.escapeJavaScript($input.json('$.Age'))` 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 5 : utilisation d'une expression JSONPath pour transmettre des informations sur une demande de méthode au point de terminaison d'intégration

L'exemple suivant utilise `$input.params()`, `$input.path()` et `$input.json()` pour envoyer des informations sur une demande de méthode au point de terminaison d'intégration. Ce modèle de mappage utilise la méthode `size()` pour fournir le nombre d'éléments dans une liste.

```
{
```

```
    "id" : "$input.params('id')",
    "count" : "$input.path('$.things').size()",
    "things" : $input.json('$.things')
}
```

Pour une demande qui inclut le paramètre de chemin `123` et le corps suivant :

```
{
      "things": {
            "1": {},
            "2": {},
            "3": {}
      }
}
```

Le résultat 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 que les données d'entrée JSON puissent être analysées correctement.

L'exemple précédent avec application de `$util.escapeJavaScript($input.json('$.things'))` est le suivant :

```
{
     "id" : "$input.params('id')",
     "count" : "$input.path('$.things').size()",
     "things" : "$util.escapeJavaScript($input.json('$.things'))"
}
```

Le résultat de ce modèle de mappage doit ressembler à ce qui suit :

```
{"id":"123","count":"3","things":"{\"1\":{},\"2\":{},\"3\":{}}"}
```

Variables pour les transformations de données pour API Gateway

Lorsque vous créez un mappage de paramètres, vous pouvez utiliser les variables context comme source de données. Lorsque vous transformez un modèle de mappage, vous pouvez utiliser les

variables context, input et util dans les scripts que vous écrivez en langage VTL (Velocity Template Language). Pour des exemples de modèles de mappage qui utilisent ces variables de référence, consultez the section called “Exemples d’utilisation de variables pour transformer des modèles de mappage”.

Pour obtenir la liste des variables de référence pour la journalisation des accès, consultez the section called “Variables pour la journalisation des accès pour API Gateway”.

Variables context pour les transformations de données

Vous pouvez utiliser les variables `$context` suivantes (sensibles à la casse) pour les transformations de données.

Paramètre	Description
`$context.accountId`	L'ID de AWS compte du propriétaire de l'API.
`$context.apiId`	Identifiant qu’API Gateway attribue à votre API.
`$context.authorizer.claims.`*`property`*	Propriété des requêtes renvoyées depuis le groupe d’utilisateurs Amazon Cognito une fois que l’appelant de la méthode a été authentifié avec succès. Pour de plus amples informations, veuillez consulter the section called “Utilisation d’un groupe d’utilisateurs Amazon Cognito en tant que mécanisme d’autorisation pour une API REST”. **Note** L’appel de `$context.authorizer.claims` renvoie la valeur null.
`$context.authorizer.principalId`	Identifiant utilisateur principal associé au jeton envoyé par le client et retourné par un mécanisme d’autorisation Lambda API Gateway (anciennement appelé Custom Authorizer). Pour de plus amples informations,

Paramètre	Description
	veuillez consulter 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"`. En effet*`property`*, le seul caractère spécial pris en charge est le trait de soulignement`(_)`. Pour de plus amples informations, veuillez consulter the section called “Utilisation des mécanismes d’autorisation Lambda”.
`$context.awsEndpointRequestId`	ID de demande du AWS point de terminaison.
`$context.deploymentId`	ID de 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`	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 de plus amples informations, veuillez consulter 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
`$context.identity.cognitoAuthenticationProvider`	Liste séparée par des virgules de tous les fournisseurs d'authentification Amazon Cognito utilisés par l'appelant à l'origine de la demande. Disponible uniquement si la demande a été signée avec les informations d'identification Amazon Cognito. Par exemple, pour une identité provenant d'un groupe d'utilisateurs Amazon Cognito, `cognito-idp.`*`region`*`.amazonaws.com/`*`user_pool_id`* `,cognito-idp.`*`region`*`.amazonaws.com/`*`user_pool_id`* `:CognitoSignIn:`*`token subject claim`* Pour plus d'informations sur les fournisseurs d'authentification Amazon Cognito disponibles, consultez Using Federated Identities dans le Guide du développeur Amazon Cognito.
`$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.
`$context.identity.sourceIp`	Adresse IP source de la connexion TCP immédiate envoyant la demande au point de terminaison API Gateway.
`$context.identity.clientCer t.clientCertPem`	Le certificat PEM-encoded client 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'authent ification 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`	ID du VPC qui fait la demande au point de terminaison API Gateway.
`$context.identity.vpceId`	ID du point de terminaison de VPC qui envoie la demande au point de terminaison API Gateway. Présent uniquement lorsque vous disposez d'une API privée.
`$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 de plus amples informations, veuillez consulter https://docs.aws.amazon.com/IAM/latest/UserGuide/id_users.html.
`$context.isCanaryRequest`	Renvoie `true` si la demande a été dirigée vers le canary et `false` si la demande n'a pas été dirigée vers le canary. Présent uniquement lorsqu'un canary a été activé.
`$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 API Gateway peuvent accepter des HTTP/2 demandes, mais API Gateway envoie des demandes aux intégrations de backend à l'aide de. HTTP/1.1 Par conséquent, le protocole de demande est enregistré HTTP/1.1 même si un client envoie une demande 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.
`$context.requestOverride.header.` *`header_name`*	Remplacement de l'en-tête de la requête. Si ce paramètre est défini, il contient les en-têtes à utiliser à la place des HTTP Headers (En-têtes HTTP) qui sont définis dans le volet Integration Request (Demande d'intégration). Pour de plus amples informations, veuillez consulter Remplacement des codes d'état et des paramètres de demande et de réponse des API REST dans API Gateway.
`$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 de plus amples informations, veuillez consulter Remplacement des codes d'état et des paramètres de demande et de réponse des API REST dans API Gateway.

Paramètre	Description
`$context.requestOverride.querystring.` *querystring_name*	Remplacement de la chaîne d'interrogation de la requête. Si ce paramètre est défini, il contient les chaînes d'interrogation de la requête à utiliser à la place des URL Query String Parameters (Paramètres de chaîne de requête d'URL) qui sont définis dans le volet Integration Request (Demande d'intégration). Pour de plus amples informations, veuillez consulter Remplacement des codes d'état et des paramètres de demande et de réponse des API REST dans API Gateway.
`$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 de plus amples informations, veuillez consulter Remplacement des codes d'état et des paramètres de demande et de réponse des API REST dans API Gateway.
`$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 de plus amples informations, veuillez consulter Remplacement des codes d'état et des paramètres de demande et de réponse des API REST dans API Gateway.

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 `$context.resourcePath` est `/root/child`. Pour de plus amples informations, veuillez consulter Didacticiel : création d'une API REST avec une intégration HTTP sans proxy.
`$context.stage`	Étape de déploiement de la demande d'API (par exemple, `Beta` ou `Prod`).
`$context.wafResponseCode`	Réponse reçue de AWS WAF: `WAF_ALLOW` ou `WAF_BLOCK`. N'est pas défini si l'étape n'est pas associée à une liste ACL web. Pour de plus amples informations, veuillez consulter the section called "AWS WAF".
`$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 de plus amples informations, veuillez consulter the section called "AWS WAF".

Variables d'entrée

Vous pouvez utiliser les variables `$input` suivantes (sensibles à la casse) pour faire référence aux données utiles et aux paramètres de la demande de méthode. Les fonctions suivantes sont disponibles :

Variable et fonction	Description
`$input.body`	Renvoie la charge utile de la demande brute sous forme de chaîne. Vous pouvez utiliser `$input.body` pour conserver des nombres à virgule flottante entiers, tels que `10.00`.
`$input.json(x)`	Cette fonction évalue une expression JSONPath et renvoie les résultats sous la forme une chaîne JSON. Par exemple, `$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`

Variable et fonction	Description
	pour désinfecter le paramètre afin d'éviter une attaque potentielle par injection. Pour un contrôle total de la désinfection des paramètres, utilisez une intégration proxy sans modèle et gérez la désinfection des requêtes dans votre intégration.

Variable et fonction	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').size()` renvoie `"3"`. Pour plus d'informations sur JSONPath, consultez JSONPath ou JSONPath pour Java.

Variables d'étape

Vous pouvez utiliser les variables d'étape suivantes comme espaces réservés pour les ARN et les URL dans les intégrations de méthodes. Pour de plus amples informations, veuillez consulter the section called "Utilisation de variables d'étape".

Syntaxe	Description
`$stageVariables.` *`variable_name`* , `$stageVariables['` *`variable_name`* `']` ou `${stageVariables['` *`variable_name`* `']}`	*`variable_name`* représente le nom d'une variable d'étape.

Variables util

Vous pouvez utiliser les variables util `$util` suivantes (sensibles à la casse) pour les modèles de mappage. 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 :

Fonction	Description
	<pre>"input" : "$util.escapeJavaScript(data).replaceAll("\\'",'''')"</pre>

`$util.parseJson()`	Prend la chaîne JSON (obtenue à l'aide de stringify) et renvoie une représentation objet du résultat. Vous pouvez utiliser le résultat de cette fonction pour accéder aux éléments de la charge utile et les manipuler en mode natif en langage VTL (Apache Velocity Template Language). Par exemple, si vous avez la charge utile suivante :
	<pre>{"errorMessage":"{\"key1\":\"var1\",\"key2\":{\"arr\":[1,2,3]}}"}</pre>et utilisez le modèle de mappage suivant :
	<pre>#set ($errorMessageObj = $util.parseJson($input.path('$.errorMessage')))
	{
	   "errorMessageObjKey2ArrVal" : $errorMessageObj.key2.arr[0]
	}</pre>vous obtenez la sortie suivante :
	<pre>{
	   "errorMessageObjKey2ArrVal" : 1
	}</pre>

`$util.urlEncode()`	Convertit une chaîne au format « application/x-www-form-urlencoded ».

Fonction	Description
`$util.urlDecode()`	Décode une chaîne « application/x -www-form-urlencoded ».
`$util.base64Encode()`	Encode les données dans une chaîne encodée en base64.
`$util.base64Decode()`	Décode les données d'une chaîne encodée en base64.

Réponses de passerelle pour REST APIs dans API Gateway

Une réponse de passerelle est identifiée par un type de réponse défini par API Gateway. La réponse se compose d'un code d'état HTTP, d'un ensemble d'en-têtes supplémentaires qui sont spécifiés par des mappages de paramètres et de données utiles qui sont générées 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 `$context`, consultez Variables context pour les transformations de données. Pour plus d'informations sur `$stageVariables`, consultez Variables d'étape. Pour plus d'informations sur les paramètres de demande de méthode, reportez-vous à la section the section called "Variables d'entrée".

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 réponse de passerelle pour API Gateway

Configurer une réponse de passerelle pour une API REST à l'aide de la console API Gateway

L'exemple suivant montre comment 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. À 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 par 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 réponse de passerelle pour API Gateway

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` .

Type de réponse de passerelle	Code de statut par défaut	Description
`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 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` .

Type de réponse de passerelle	Code de statut par défaut	Description
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 .
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 d'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, cette réponse est définie par défaut sur 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 d'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.

Partage des ressources entre origines multiples pour les API REST dans API Gateway

Cross-origin le partage de ressources (CORS) est une fonctionnalité de sécurité du navigateur qui restreint les requêtes HTTP provenant de plusieurs origines lancées à partir de scripts exécutés dans le navigateur. Pour plus d'informations, consultez What is CORS?

Activation ou non de 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.

Cross-origin Les requêtes HTTP peuvent être divisées en deux types : les requêtes simples et les requêtes 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` cross-origin 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 autres que de 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 simulée.
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 direct sur `NEVER`. Dans ce cas, 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. Pour de plus amples informations, veuillez consulter Comportement des demandes de méthode pour les données utiles sans modèle de mappage pour les API REST dans API Gateway.
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: 'DELETE,GET,HEAD,OPTIONS,PUT,POST,PATCH'`
   - `Access-Control-Allow-Origin: '*'`

Après avoir créé la demande de prévol, vous devez renvoyer l'`Access-Control-Allow-Origin:'origin'`en-tête `Access-Control-Allow-Origin: '*'` ou pour toutes les CORS-enabled méthodes pour au moins les 200 réponses.

Activation de CORS pour les intégrations sans proxy à l'aide du Console de gestion AWS

Vous pouvez utiliser le Console de gestion AWS 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'`Access-Control-Allow-Origin`en-tête de toutes les CORS-enabled méthodes pour au moins les 200 réponses.

Si les types de médias binaires sont définis sur `*/*` pour votre API, lorsqu'API Gateway crée une méthode `OPTIONS`, remplacez `contentHandling` par `CONVERT_TO_TEXT`.

La commande update-integration suivante modifie le paramètre `contentHandling` en `CONVERT_TO_TEXT` pour une demande d'intégration :

```
aws apigateway update-integration \
  --rest-api-id abc123 \
  --resource-id aaa111 \
  --http-method OPTIONS \
  --patch-operations op='replace',path='/contentHandling',value='CONVERT_TO_TEXT'
```

La commande update-integration-response suivante modifie le paramètre `contentHandling` en `CONVERT_TO_TEXT` pour une réponse d'intégration :

```
aws apigateway update-integration-response \
  --rest-api-id abc123 \
  --resource-id aaa111 \
  --http-method OPTIONS \
  --status-code 200 \
  --patch-operations op='replace',path='/contentHandling',value='CONVERT_TO_TEXT'
```

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 de CORS pour une API API Gateway

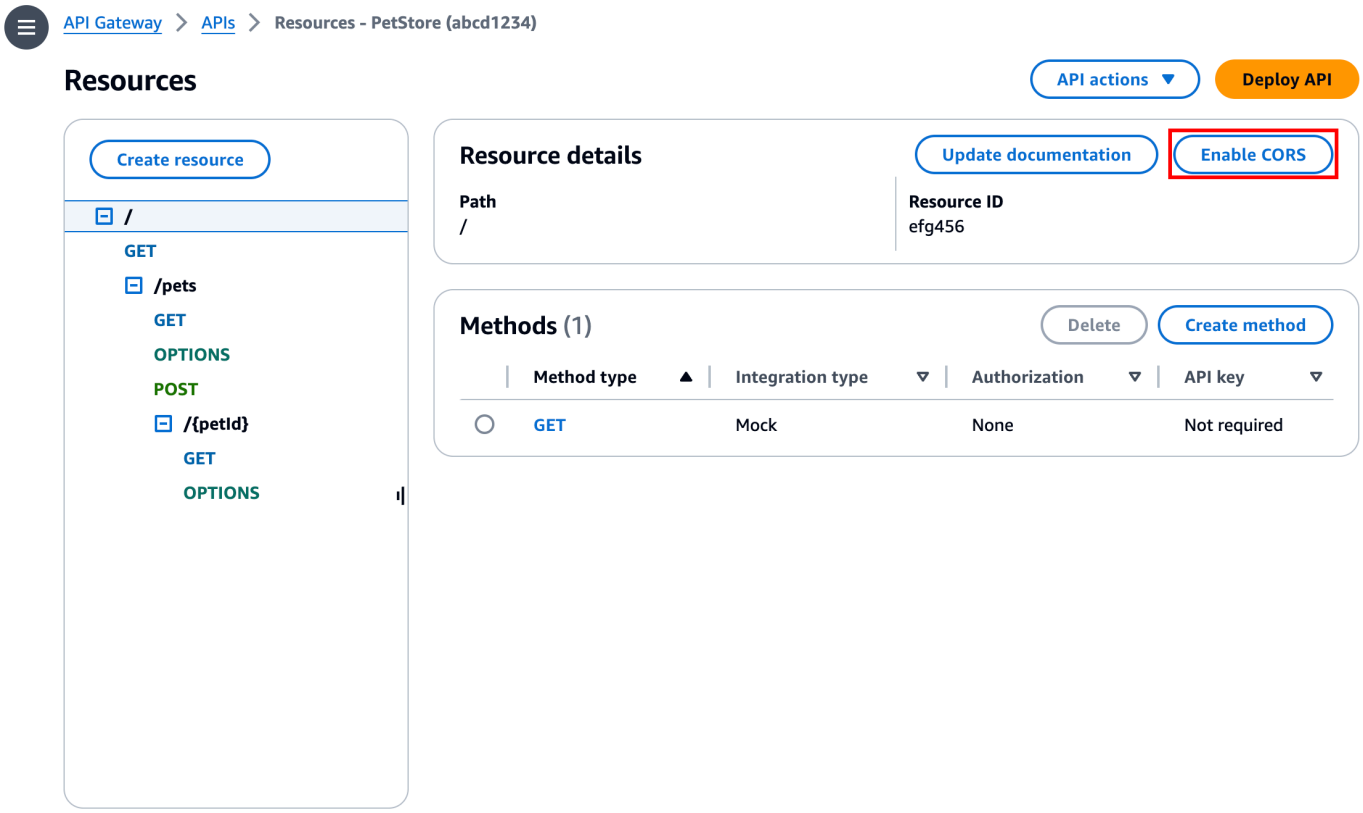
Activation de CORS sur une ressource à l'aide de la console API Gateway

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 la prise en charge de CORS, définissez le comportement de transfert direct sur `NEVER`. Dans ce cas, 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. Pour de plus amples informations, consultez Comportement des demandes de méthode pour les données utiles sans modèle de mappage pour les API REST dans API Gateway.

> **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 de 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 CORS.



5. Dans la zone Activer CORS, procédez comme suit :

a. (Facultatif) Si vous avez créé une réponse de passerelle personnalisée et que vous souhaitez activer la prise en charge de CORS pour une réponse, sélectionnez une réponse de passerelle.

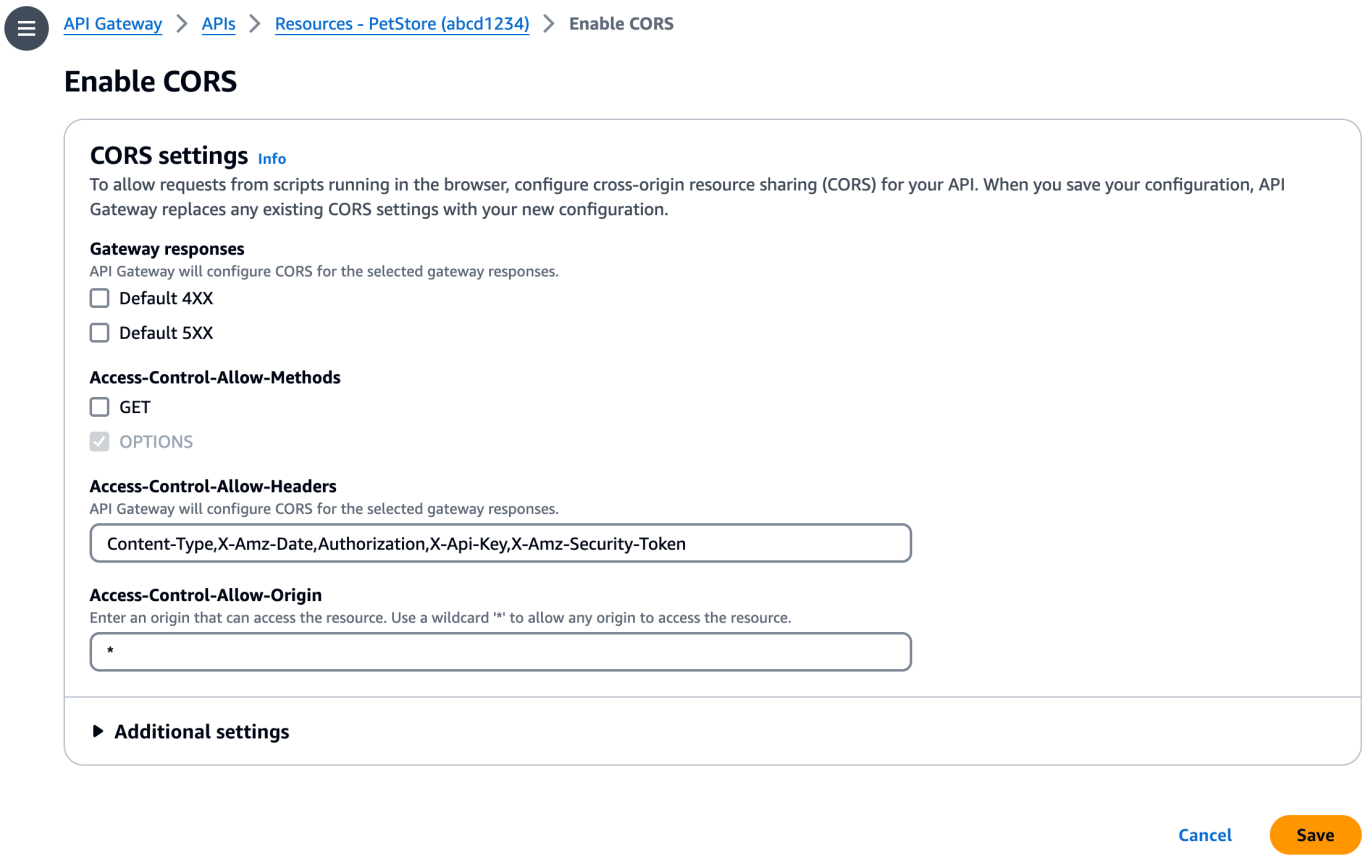
b. Sélectionnez chaque méthode pour activer la prise en charge de CORS. CORS doit être activé sur la méthode `OPTION`.

   Si vous activez la prise en charge de CORS pour une méthode `ANY`, CORS est activé pour toutes les méthodes.

c. Dans le champ de Access-Control-Allow-Headerssaisie, entrez une chaîne statique d'une liste d'en-têtes séparés par des virgules que le client doit soumettre 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 `'*'` comme valeur d'Access-Control-Allow-Originen-tête pour autoriser les demandes d'accès provenant de toutes les origines, ou spécifiez une origine à autoriser 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 de plus amples informations, veuillez consulter the section called "Création d'un déploiement".

> **Note**
>
> Si vous ne parvenez pas à activer la prise en charge de CORS sur votre ressource après avoir suivi la procédure, nous vous recommandons de comparer votre configuration CORS à l'exemple de ressource d'API `/pets`. Pour apprendre à créer l'exemple d'API, consultez the section called "Didacticiel : 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 Access-Control-Allow-Headers à ce que Access-Control-Allow-Origin des en-têtes soient configurés dans chaque méthode d'API qui accepte les requêtes 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 **`d'options`**

L'exemple suivant crée une méthode `OPTIONS` pour une intégration simulée.

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-Originet aux types de réponses.

   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 de CORS pour une API API Gateway

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 de plus amples informations, veuillez consulter Partage des ressources entre origines multiples pour les API REST dans API Gateway.

Types de supports binaires pour les API REST dans API Gateway

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 de proxy ou autre que de proxy.

Si vous utilisez une intégration proxy avec le streaming de réponse à la charge utile, vous n'avez pas besoin de configurer vos types de médias binaires. Pour de plus amples informations, veuillez consulter the section called "Streaming de la réponse d'intégration pour vos intégrations de 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 binaire MediaTypes pour 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 supports binaires incluent `image/png` ou `application/octet-stream`. Vous pouvez utiliser le caractère générique (`*`) pour couvrir plusieurs types de supports.

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 supports binaires de votre API sur `*/*`.

Pour obtenir un exemple de code, consultez the section called "Renvoi d'un support binaire d'une intégration de proxy Lambda dans API Gateway".

Si vous utilisez une intégration de proxy Lambda avec le streaming de réponse à la charge utile, vous n'avez pas besoin de configurer vos types de supports binaires. Pour de plus amples informations, veuillez consulter the section called "Configuration d'une intégration de proxy Lambda avec diffusion des données utiles de la réponse".

Non-proxy intégrations

Pour gérer les charges utiles binaires pour les intégrations sans proxy, vous devez ajouter les types de média à la MediaTypes liste binaire 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.

> **Note**
>
> Pour les intégrations `MOCK` ou privées, la définition des propriétés `contentHandling` n'est pas prise en charge dans la Console de gestion AWS. Vous devez utiliser le AWS CLI CloudFormation, ou un SDK pour définir les `contentHandling` propriétés.

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 `contentHandling` propriété ne sont pas définies, les charges utiles sont traitées sous forme de chaînes JSON UTF-8 codées.
- 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 en savoir plus, 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 plus d'informations sur la façon dont API Gateway gère les charges utiles textuelles et binaires, consultez 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.

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	UTF8-encoded ficelle
Données de texte	Tout type de données	Non défini	`CONVERT_TO_BINARY`	Base64-decoded blob binaire

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	`CONVERT_TO_TEXT`	UTF8-encoded ficelle
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`	Base64-decoded blob binaire
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 de supports correspondants	`CONVERT_TO_BINARY`	Données binaires
Données binaires	Un type de données binaires	Défini avec les types de supports correspondants	`CONVERT_TO_TEXT`	Base64-encoded ficelle

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.

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	UTF8-encoded ficelle
Données texte ou binaires	Un type de texte	Non défini	`CONVERT_TO_BINARY`	Base64-decoded blob
Données texte ou binaires	Un type de texte	Non défini	`CONVERT_TO_TEXT`	UTF8-encoded ficelle
Données de texte	Un type de texte	Défini avec les types de supports correspondants	Non défini	Données de texte

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 de texte	Défini avec les types de supports correspondants	`CONVERT_TO_BINARY`	Base64-decoded blob
Données de texte	Un type de texte	Défini avec les types de supports correspondants	`CONVERT_TO_TEXT`	UTF8-encoded ficelle
Données de texte	Un type binaire	Défini avec les types de supports correspondants	Non défini	Base64-decoded blob
Données de texte	Un type binaire	Défini avec les types de supports correspondants	`CONVERT_TO_BINARY`	Base64-decoded blob
Données de texte	Un type binaire	Défini avec les types de supports correspondants	`CONVERT_TO_TEXT`	UTF8-encoded ficelle
Données binaires	Un type de texte	Défini avec les types de supports correspondants	Non défini	Base64-encoded ficelle
Données binaires	Un type de texte	Défini avec les types de supports correspondants	`CONVERT_TO_BINARY`	Données binaires

Charge utile de la réponse d'intégration	En-tête **Accept** de la demande	**binaryMediaTypes**	**contentHandling**	Charge utile de la réponse de méthode
Données binaires	Un type de texte	Défini avec les types de supports correspondants	CONVERT_TO_TEXT	Base64-encoded ficelle
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	Base64-encoded ficelle

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"
```

```
}
```

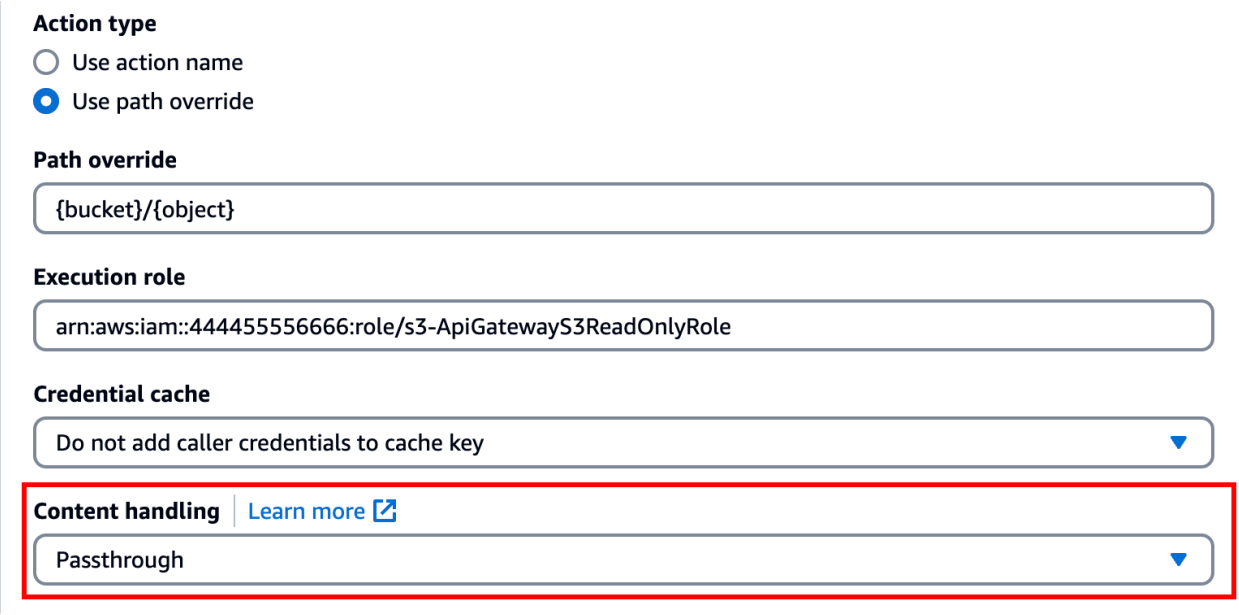
Pour que la charge utile binaire soit passée sans modification, vous devez activer les comportements relais sur l'API.

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 avec 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 avec Amazon S3, consultez Didacticiel : création d'une API REST en tant que proxy Amazon S3.

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 supports dans la section Types de supports binaires.
   d. Choisissez Ajouter un type de support binaire.
   e. Saisissez un type de support requis, par exemple **`image/png`**, dans le champ de texte d'entrée. Si nécessaire, répétez cette étape pour ajouter d'autres types de supports. 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 avec l'action `Object GET` dans Amazon S3.
   c. Dans Gestion de contenu, choisissez une option.

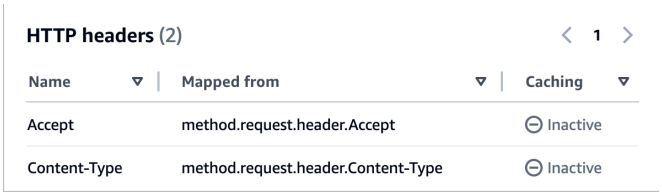


Choisissez Passthrough si vous ne souhaitez pas convertir le corps lorsque le client et le backend acceptent le même format binaire. Choisissez Convertir en texte pour convertir le corps binaire en chaîne encodée en base64 lorsque, par exemple, le backend a besoin qu'une charge utile de requête binaire 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 contenus 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 d'encodages de contenu pour API Gateway

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.

- Propriété x-amazon-apigateway-binary-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 :

- Objet x-amazon-apigateway-integration
- Objet x-amazon-apigateway-integration.response

Renvoi d'un support binaire d'une intégration de proxy Lambda dans API Gateway

Pour renvoyer un support 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 supports binaires de votre API. Lorsque vous configurez les types de médias binaires de votre API, celle-ci traite ce type de contenu comme des données binaires. 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 supports binaires 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 plus d'informations sur les types de supports binaires, consultez Types de supports binaires pour les API REST dans API Gateway.

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 `PUT /lambda?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.

Vous créez la fonction Lambda que votre API appelle, et elle doit renvoyer une chaîne codée en base64 avec l'en-tête `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
```

Invocation d'une API REST dans 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://api-id.execute-api.region.amazonaws.com/stage/
```

où *`api-id`* est l'ID de l'API, *`region`* la région AWS et *`stage`* 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 apprendre à déployer une API, consultezDéploiement d'une API REST dans API Gateway.

Rubriques

- Obtention de l'URL d'invocation d'une API
- Appel d'une API
- Utilisation de la console API Gateway pour tester une méthode API REST
- Utilisation d'un kit SDK Java généré par API Gateway pour une API REST
- Utilisation d'un kit SDK Android généré par API Gateway pour une API REST
- Utiliser un JavaScript SDK généré par API Gateway pour une API REST
- Utilisation d'un kit SDK Ruby généré par API Gateway pour une API REST
- Utiliser le SDK iOS généré par API Gateway pour une API REST dans Objective-C ou Swift

Obtention de l'URL d'invocation d'une API

Vous pouvez utiliser la console, l'AWS CLI ou une définition OpenAPI exportée pour obtenir l'URL d'invocation d'une API.

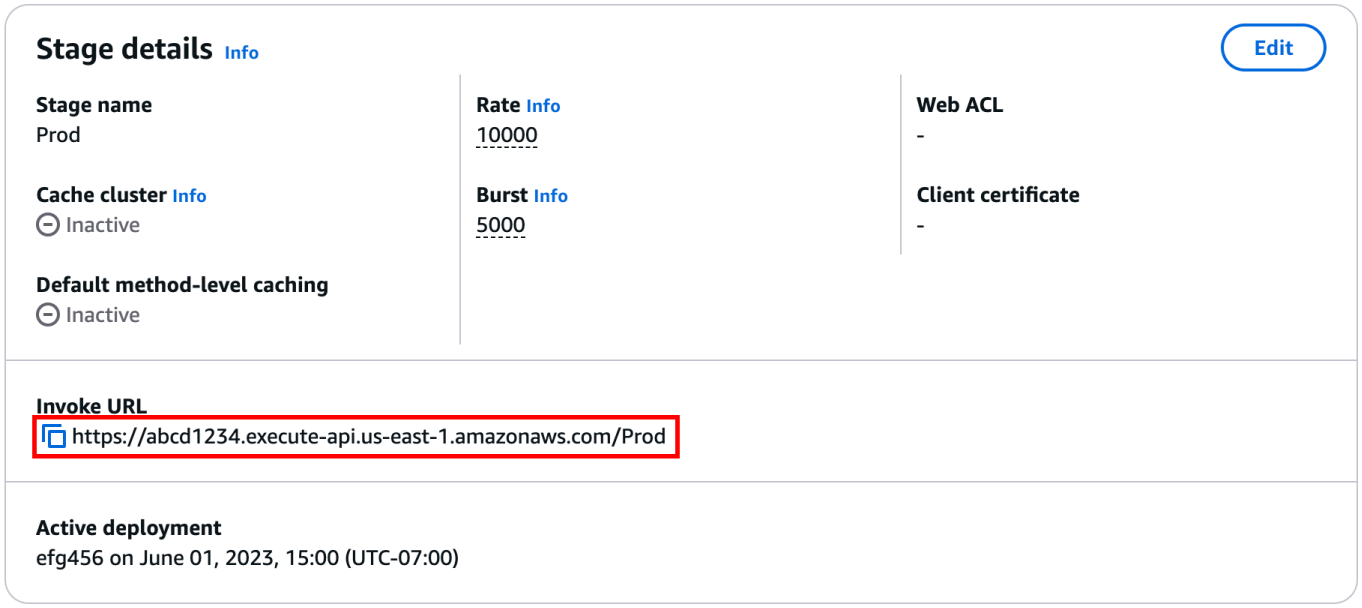
Obtention de l'URL d'invocation d'une API à l'aide de la console

La procédure suivante montre comment obtenir l'URL d'invocation d'une API dans la console API REST.

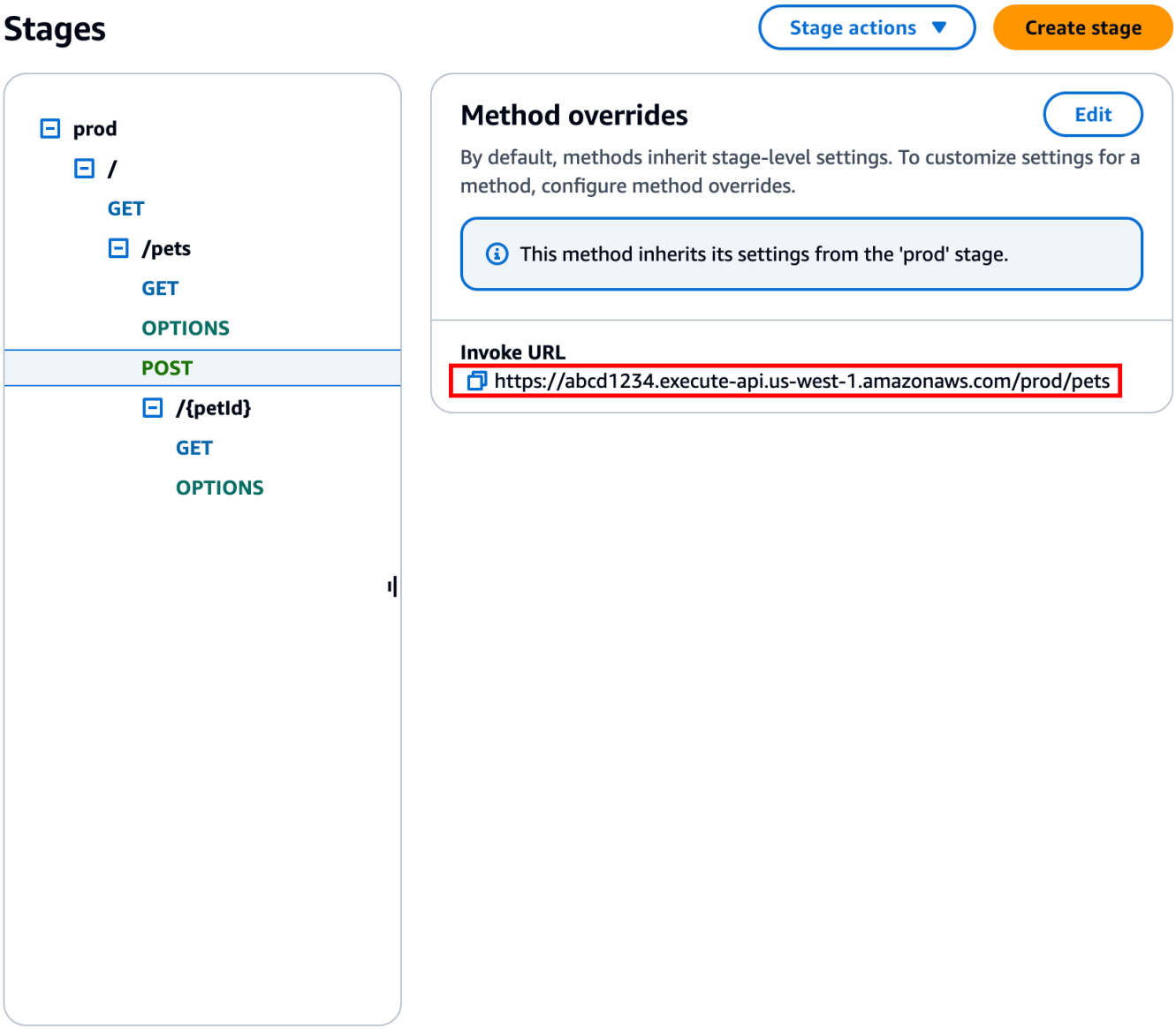
Pour obtenir l'URL d'invocation d'une API à l'aide de la console API REST

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'invocation d'une API pour une autre ressource de votre API, développez l'étape sous le volet de navigation secondaire, puis choisissez une méthode.
6. Choisissez l'icône de copie pour copier l'URL d'invocation de votre API au niveau des ressources.



Obtention de l'URL d'invocation d'une API à l'aide de l'AWS CLI

La procédure suivante montre comment obtenir l'URL d'invocation d'une API à l'aide de l'AWS CLI.

Pour obtenir l'URL d'invocation d'une API à l'aide de l'AWS CLI

1. Utilisez la commande suivante pour obtenir la valeur `rest-api-id`. Cette commande renvoie toutes les valeurs `rest-api-id` de votre région. Pour plus d'informations, consultez get-rest-apis.

```
aws apigateway get-rest-apis
```

2. Remplacez la valeur `rest-api-id` de l'exemple par votre valeur `rest-api-id`, remplacez la valeur *`{stage-name}`* de l'exemple par votre valeur *`{stage-name}`*, et remplacez la valeur *`{region}`* par votre région.

```
https://{restapi_id}.execute-api.{region}.amazonaws.com/{stage_name}/
```

Obtention de l'URL d'invocation d'une API à l'aide du fichier de définition OpenAPI exporté de l'API

Vous pouvez également construire l'URL racine en combinant les champs `host` et `basePath` d'un fichier de définition OpenAPI exporté de l'API. Pour apprendre à exporter votre API, consultez the section called "Exportation d'une API REST".

Appel d'une API

Vous pouvez appeler votre API déployée à l'aide d'un navigateur, de curl ou d'autres applications, telles que Postman.

Vous pouvez également utiliser la console API Gateway pour tester un appel d'API. Le test utilise la fonctionnalité `TestInvoke` d'API Gateway, qui permet de tester l'API avant son déploiement. Pour de plus amples informations, consultez the section called "Utilisation de la console pour tester une méthode API REST".

> **Note**
>
> Les valeurs de paramètres de chaîne de requêtes d'une URL d'invocation ne peuvent pas contenir `%%`.

Invocation d'une API à l'aide d'un navigateur Web

Si votre API autorise les accès anonymes, vous pouvez utiliser n'importe quel navigateur web pour invoquer une méthode `GET`. Saisissez l'URL d'invocation complète dans la barre d'adresse du navigateur.

Pour les autres méthodes ou les appels nécessitant une authentification, vous devez spécifier des données utiles ou signer les demandes. Vous pouvez gérer ces appels dans un script sous-jacent d'une page HTML ou dans une application client à l'aide d'un des kits SDK AWS.

Invocation d'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 ressource `getUsers` de l'étape `prod` 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

- Conditions préalables
- Test d'une méthode avec la console API Gateway

Conditions préalables

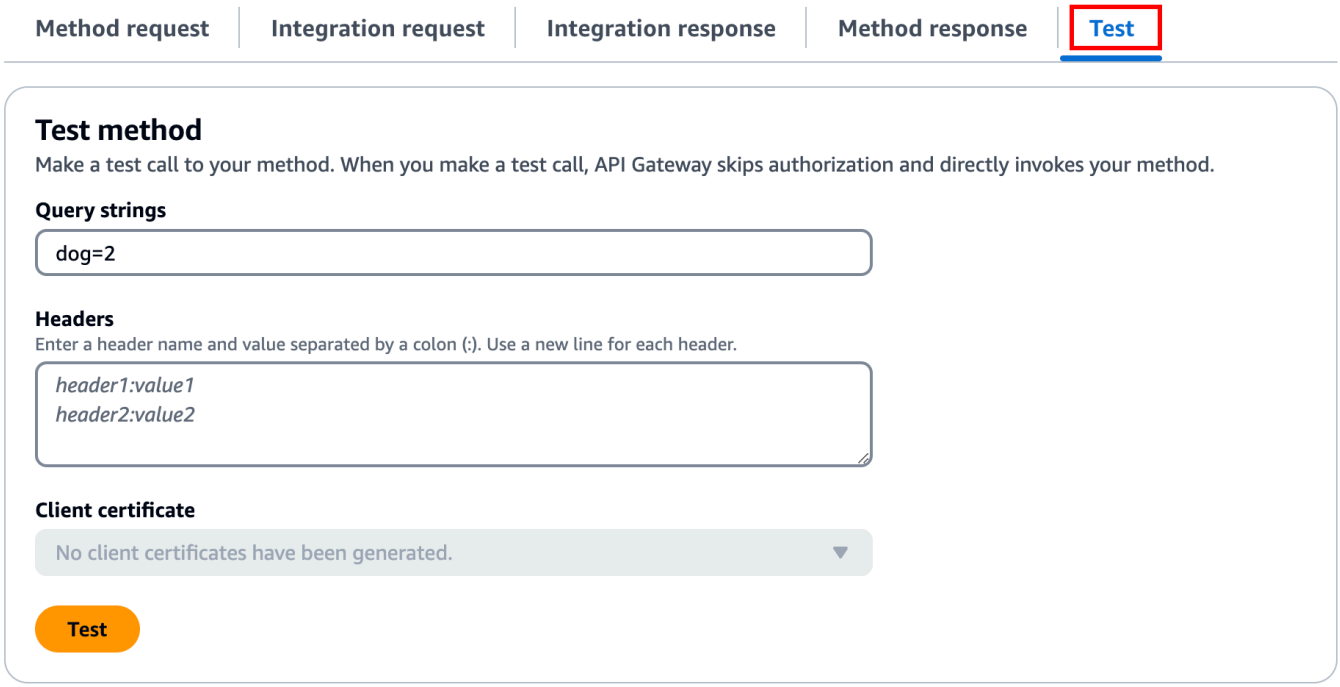
- Vous devez spécifier les paramètres des méthodes que vous souhaitez tester. Suivez les instructions de la section Méthodes pour REST APIs dans API Gateway.

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 la demande de méthode sous sa application/json forme par défaut.

   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 de statut HTTP de la réponse.
   - Latence (ms) correspond au temps écoulé entre la réception de la demande de l'appelant et l'envoi de la réponse.
   - 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 AWS CLI, consultez 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 Générer SDKs pour REST APIs 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 fichier `pom.xml` et un dossier `src` sous 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 Générer SDKs pour REST APIs dans API Gateway.

> **Note**
>
> Le kit SDK généré n’est pas compatible avec Android 4.4 et versions antérieures. Pour de plus amples 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'artefact 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. Exemples :

   ```
   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 classe `ApiClientFactory` pour transmettre un ensemble d'informations d'identification AWS à l'aide du kit 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 d'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

La procédure suivante montre comment utiliser un JavaScript SDK généré par API Gateway.

> **Note**
>
> Ces instructions supposent que vous avez déjà suivi les instructions de Générer SDKs pour REST APIs 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 entre origines multiples (CORS) pour toutes les méthodes que le kit SDK généré par API Gateway appelle. Pour obtenir des instructions, veuillez consulter Partage des ressources entre origines multiples pour les API REST dans API Gateway.

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 d'API avec le kit SDK généré par API Gateway, vous pouvez transmettre la clé d'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 d'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 *methodName* est construit à partir du chemin de ressource 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
```

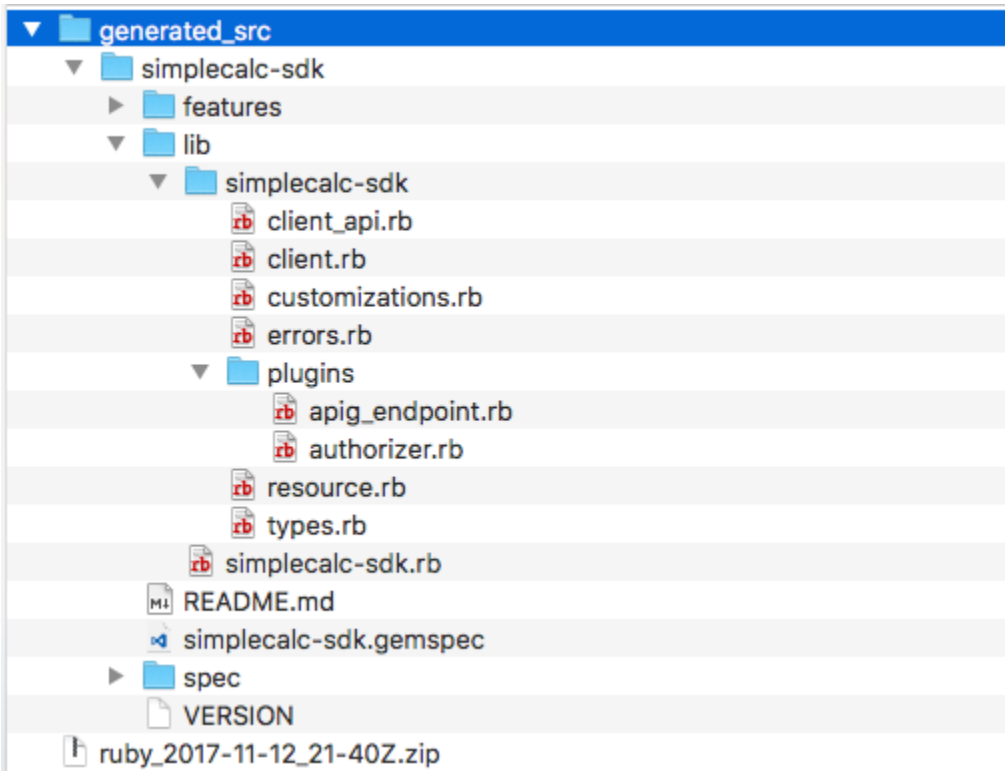
Utilisation d'un kit SDK Ruby généré par API Gateway pour une API REST

La procédure suivante montre comment utiliser un kit SDK Ruby généré par API Gateway.

> **Note**
>
> Ces instructions supposent que vous avez déjà suivi les instructions de Générer SDKs pour REST APIs 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
        }
    }
}
```

Utiliser le SDK iOS généré par API Gateway pour une API REST dans Objective-C ou Swift

Dans ce didacticiel, nous allons montrer comment utiliser un SDK iOS généré par API Gateway pour une API REST dans une application Objective-C ou Swift afin d'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

- Utiliser le SDK iOS (Objective-C) généré pour appeler l'API
- Utilisation d'un kit SDK iOS généré (Swift) pour appeler une API

Utiliser le SDK iOS (Objective-C) généré pour appeler l'API

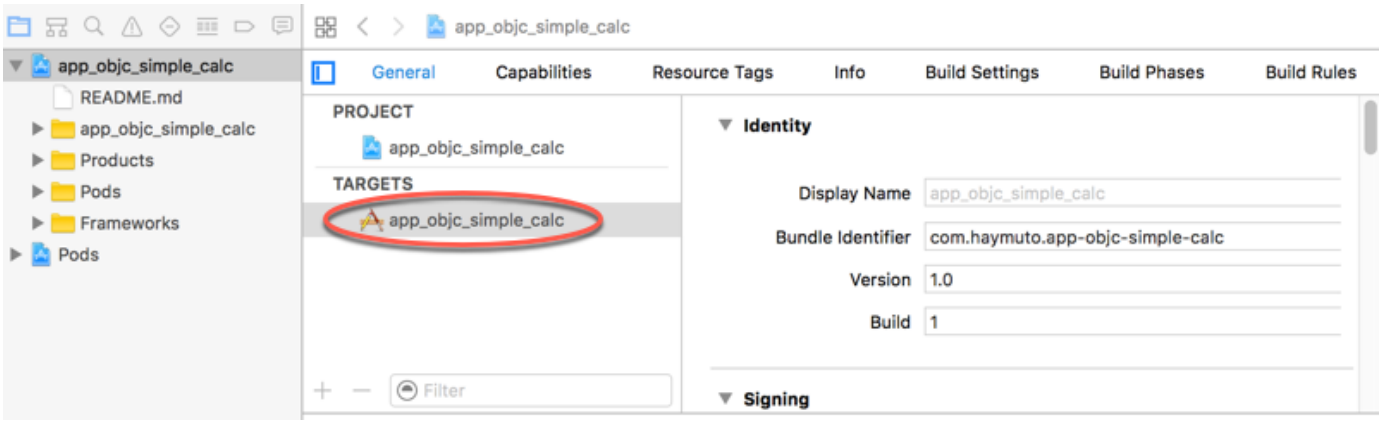
Avant de commencer la procédure suivante, vous devez suivre les étapes Générer SDKs pour REST APIs dans API Gateway décrites dans iOS Objective-C et télécharger le fichier .zip du SDK généré.

Installer la AWS SDK mobile et 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 SDK iOS généré par API Gateway dans Objective-C

1. Extrayez le contenu du fichier API Gateway-generated .zip que vous avez téléchargé précédemment. À l'aide de l'SimpleCalc API, vous souhaiterez peut-être renommer le dossier du SDK décompressé comme suit. **`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 nouveau Objective-C projet iOS. Notez la cible du projet. Vous en avez besoin pour la définir dans le fichier `Podfile`.



3. Pour l'importer Applications mobiles AWS SDK for iOS dans le projet Xcode à l'aide de CocoaPods, procédez comme suit :

   a. Installez 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 :

   ```
   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'
   ```

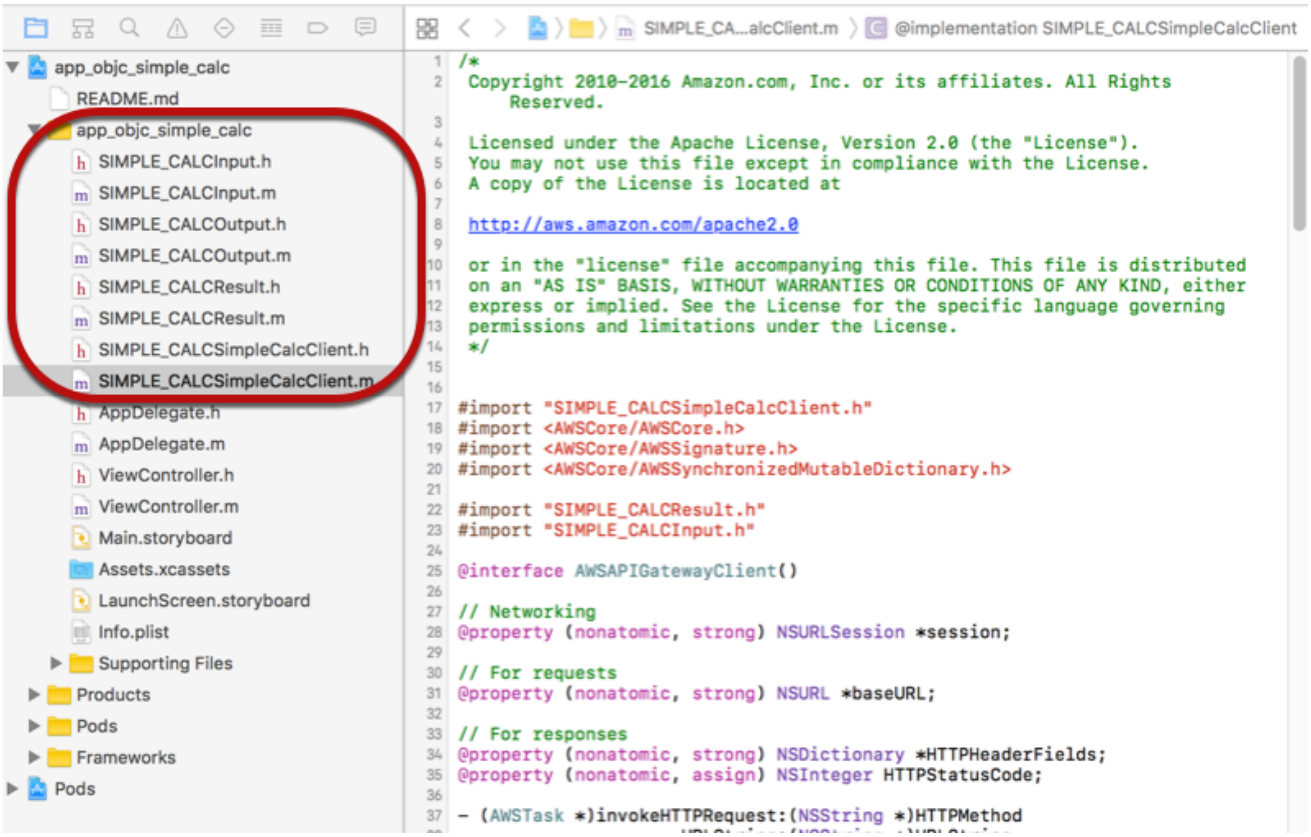
   c. Ouvrez une fenêtre de terminal et exécutez la commande suivante :

   ```
   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 l'importer Applications mobiles AWS 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.

Appelez des méthodes d'API à l'aide du 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, les

`SIMPLE_CALCInput` classes de données correspondantes. `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

  ```
  (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 Objective-C l'application, par exemple, dans le cadre du `viewDidLoad` délégué dans un `ViewController.m` fichier.

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 Générer SDKs pour REST APIs dans API Gateway pour iOS dans Swift et téléchargez le fichier .zip du kit SDK généré.

Rubriques

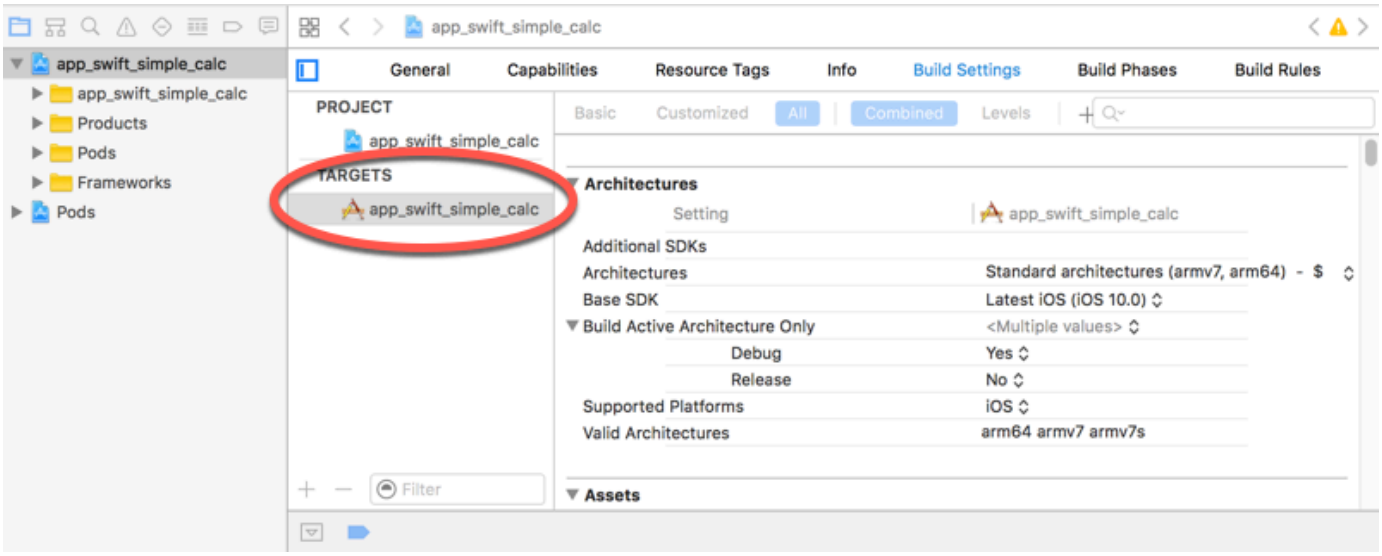
- Installation AWS SDK mobile et Gateway-generated SDK d'API dans un projet Swift
- Appel des méthodes d'API via le kit SDK iOS généré par API Gateway dans un projet Swift

Installation AWS SDK mobile et Gateway-generated SDK d'API 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 API Gateway-generated .zip que vous avez téléchargé précédemment. À l'aide de l'SimpleCalc API, vous souhaiterez peut-être renommer le dossier du SDK décompressé comme suit. **`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'
```

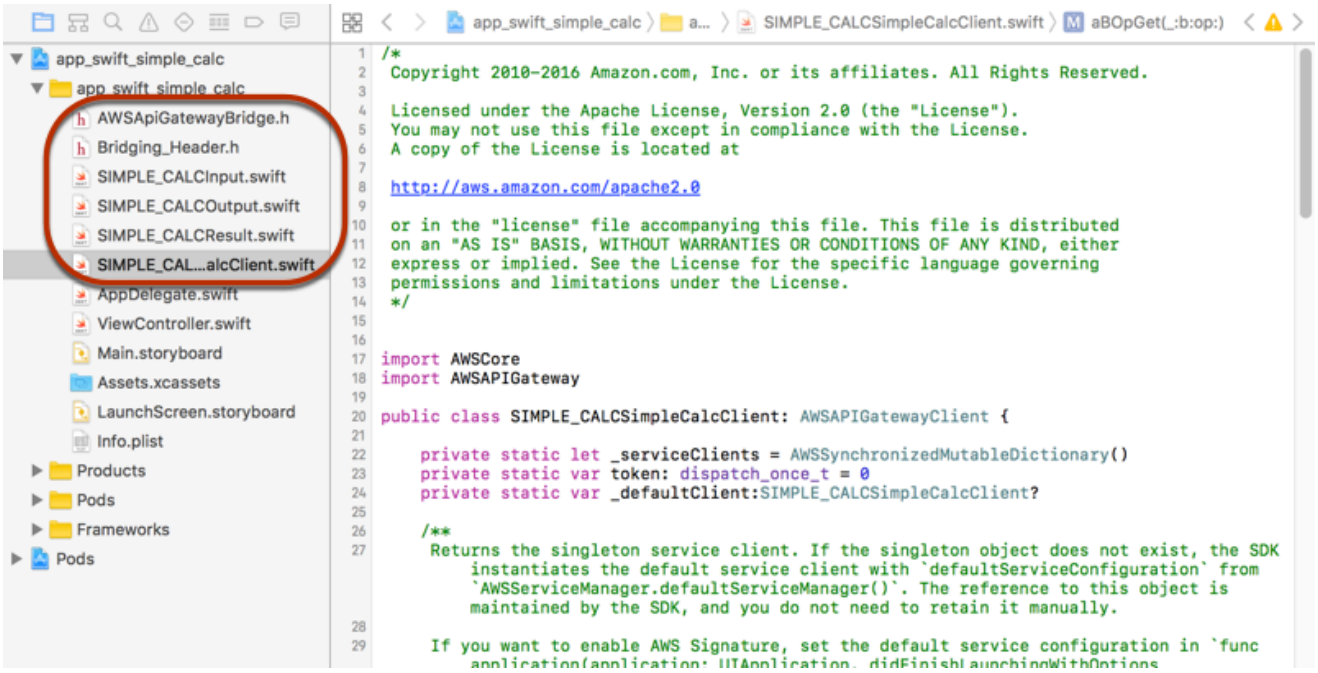
c. Ouvrez une fenêtre de terminal et exécutez la commande suivante dans le répertoire de l'application :

```
pod install
```

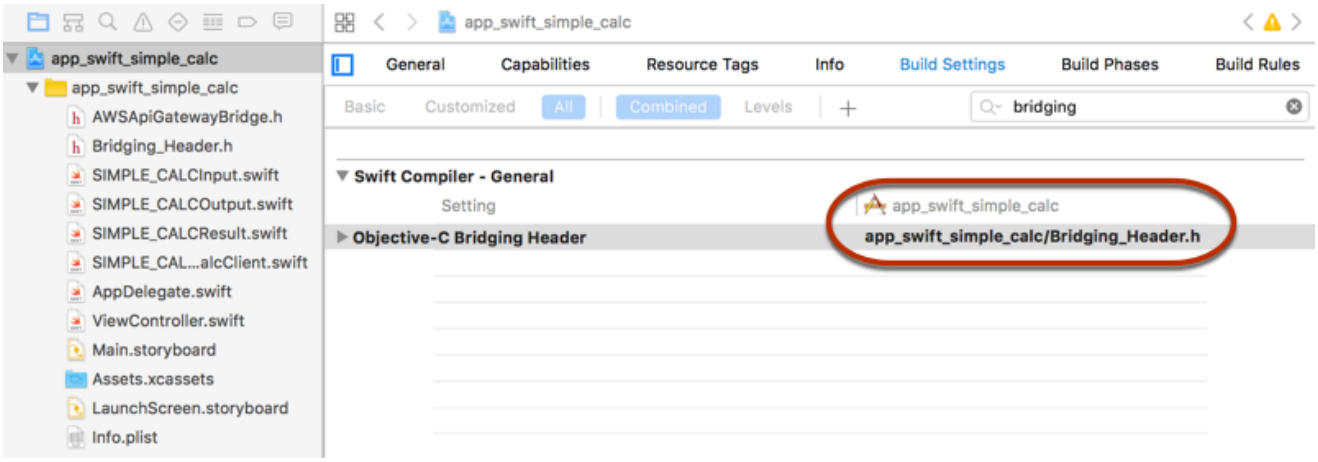
Cela installe le composant API Gateway et tous les composants du SDK AWS mobile dépendants dans le projet de l'application.

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.



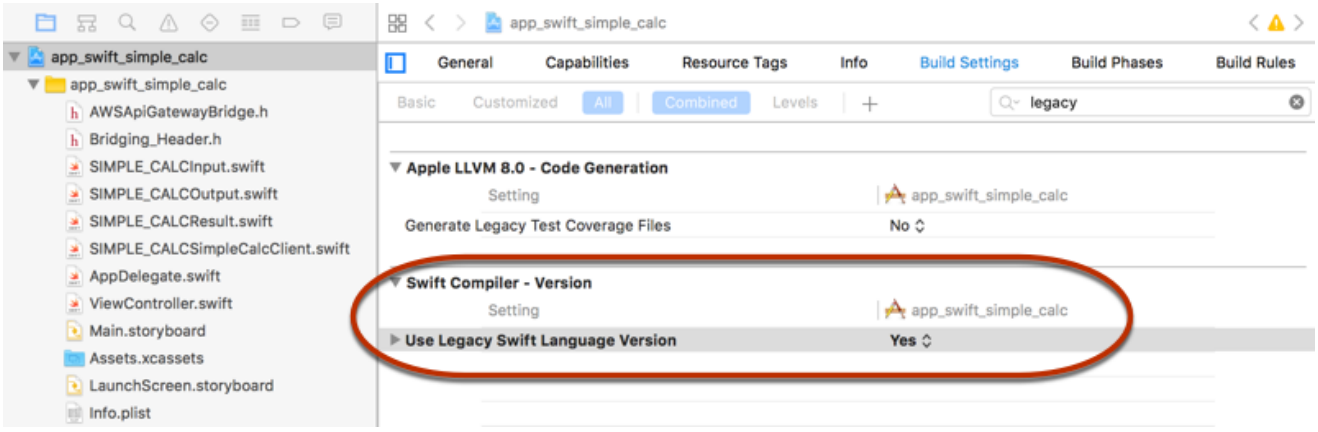
f. Pour permettre d'appeler les Objective-C bibliothèques du SDK AWS mobile depuis votre projet de code Swift, définissez le chemin du `Bridging_Header.h` fichier sur la propriété Objective-C Bridging Header sous le compilateur Swift - Paramètre général de la configuration de votre projet Xcode :



> **Tip**
>
> Vous pouvez taper **`bridging`** dans le champ de recherche de Xcode pour localiser la propriété Objective-C Bridging Header.

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.

Développement d'API REST à l'aide d'OpenAPI dans API Gateway

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 relatives à Amazon API Gateway pour REST APIs. 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 didacticiel sur l'utilisation de la fonction d'importation d'API à partir de la console API Gateway, consultez Didacticiel : création d'une API REST par l'importation d'un exemple.

Rubriques

- Importation d'une API optimisée pour la périphérie dans API Gateway
- Importation d'une API régionale dans API Gateway
- Importation d'un fichier OpenAPI pour mettre à jour une définition d'API existante
- Définissez la propriété OpenAPI basePath
- AWS Variables pour l'importation OpenAPI

- Erreurs et avertissements liés à l'importation de votre API dans API Gateway
- Exportation d'une API REST à partir d'API Gateway

Importation d'une API optimisée pour la périphérie dans API Gateway

Vous pouvez importer un fichier de définition d'API OpenAPI pour créer une API optimisée pour la périphérie 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 didacticiel sur l'utilisation de la fonction d'importation d'API à partir de la console API Gateway, consultez Didacticiel : création d'une API REST par l'importation d'un exemple.

Rubriques

- Importation d'une API optimisée pour la périphérie 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 la périphérie à l'aide de la console API Gateway

Pour importer une API optimisée pour la périphérie 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

La import-rest-apicommande suivante importe une API depuis un fichier de définition OpenAPI pour créer une nouvelle API optimisée pour les périphériques :

```
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'
```

Importation d'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 didacticiel sur l'utilisation de la fonction d'importation d'API à partir de la console API Gateway, consultez Didacticiel : 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
- Importez une API régionale à l'aide du 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.

Importez une API régionale à l'aide du AWS CLI

La import-rest-apicommande suivante importe un fichier de définition OpenAPI et définit le type de point de terminaison sur Regional :

```
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'opération importation vers mise à jour peut se produire dans 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.

> **Important**
>
> Lorsque vous l'utilisez`mode=overwrite`, la définition complète de l'API est remplacée. Toutes les configurations d'API qui ne sont pas explicitement incluses dans le corps de l'importation sont rétablies à leurs valeurs par défaut. Cela inclut des paramètres tels que les politiques de sécurité, les configurations d'autorisation et d'autres API-level propriétés. Pour conserver les configurations existantes, utilisez`mode=merge`, qui préserve les configurations existantes qui ne sont pas spécifiées dans le corps de l'importation. Si vous en utilisez`mode=overwrite`, assurez-vous que votre définition d'OpenAPI inclut toutes les configurations d'API souhaitées, telles que les extensions API Gateway.

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 montre un exemple de demande de remplacement par la charge utile d'une définition : JSON-formatted OpenAPI

```
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.

Définissez la propriété OpenAPI basePath

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 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}`. Le tableau suivant décrit les AWS variables disponibles.

Nom de la variable	Description
`AWS::AccountId`	L'ID du 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`	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 liés à l'importation de votre API dans API Gateway

Lorsque vous importez votre fichier de définition externe dans API Gateway, des avertissements et des erreurs peuvent être générés. Les sections suivantes décrivent les erreurs et les avertissements susceptibles de se produire lors de 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 la fonction d'exportation d'API, vous exportez une API REST existante en envoyant une requête GET et en spécifiant l'API à exporter dans les 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 destinés au type de contenu « application/json » pour qu'API Gateway puisse exporter le modèle. 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 connaître 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
```

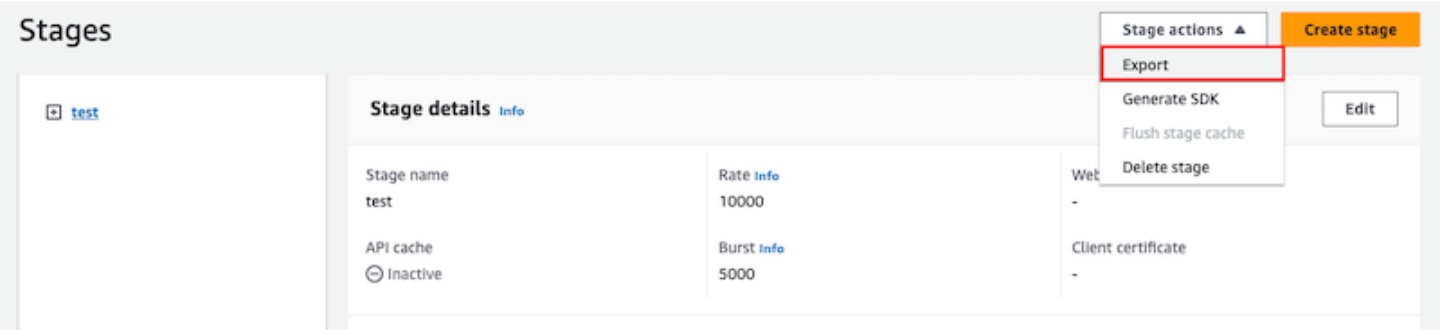
OpenAPI 2.0

```
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 dans une étape, vous pouvez passer à l'exportation de cette API dans 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.

Publiez REST APIs pour que les clients puissent l'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 dans 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 votre API Gateway APIs, le `execute-api.{region}.amazonaws.com` domaine est enregistré dans la liste des suffixes publics (PSL).

Pour plus de sécurité, nous vous recommandons d'utiliser des cookies avec un `__Host-` préfixe si vous devez définir des cookies sensibles dans le nom de domaine par défaut de votre API Gateway APIs. 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 API Gateway
- Nom de domaine personnalisé pour le REST public APIs dans API Gateway

Déploiement d'une API REST dans 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 mécanismes d'autorisation, des ressources de politiques et de tout 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 dans CloudTrail ou CloudWatch, et sélectionner un certificat client pour que le backend authentifie 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 une valeur de 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 AWS SDK.

Rubriques

- Création d'un déploiement pour une API REST dans API Gateway
- Configuration d'une étape pour une API REST dans API Gateway
- Configuration d'un déploiement de la version Canary API Gateway
- Mises à jour de REST APIs nécessitant un redéploiement

Création d'un déploiement pour 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
- Prochaines étapes pour le déploiement de votre API

Création d'un déploiement

Les procédures suivantes montrent comment créer un déploiement pour une API REST.

Console de gestion AWS

Vous devez avoir créé une API REST avant de la déployer pour la première fois. Pour plus d'informations, consultez Développement d'API REST dans API Gateway.

La console API Gateway vous permet de déployer une API en créant un déploiement et en l'associant à une étape nouvelle ou existante.

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).

AWS CLI

Lorsque vous créez un déploiement, vous instanciez la ressource Deployment.

La commande create-deployment suivante crée un déploiement :

```
aws apigateway create-deployment --rest-api-id rest-api-id
```

Vous ne pouvez pas appeler l'API tant que vous n'associez pas ce déploiement à une étape. Si l'étape existe déjà, remplacez sa propriété deploymentId par l'ID du déploiement que vous venez de créer. La commande update-stage suivante met à jour l'étape avec un nouveau déploiement. Dans la console, cette opération s'appelle déploiement actif.

```
aws apigateway update-stage \
   --rest-api-id rest-api-id \
   --stage-name 'stage-name' \
   --patch-operations op='replace',path='/deploymentId',value='deployment-id'
```

Vous pouvez également associer le déploiement que vous êtes en train de créer à une nouvelle étape. La commande create-deployment suivante crée un déploiement et l'associe à une nouvelle étape appelée `beta` :

```
aws apigateway create-deployment \
   --rest-api-id rest-api-id \
   --stage-name beta
```

Pour redéployer une API, exécutez les mêmes étapes. Vous pouvez réutiliser la même étape.

Prochaines étapes pour le déploiement de votre API

Les prochaines étapes pour le déploiement de votre API sont les suivantes.

Modification des paramètres d'étape

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 plus d'informations, consultez Modification des paramètres d'étape.

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.

Choix de différentes combinaisons d'étapes de déploiement

Étant 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.

Transmission des données spécifiques au déploiement à votre API.

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 Utilisation de variables d'étape pour une API REST dans API Gateway.

Configuration d'une étape pour une API REST dans API Gateway

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. La section suivante montre comment créer et configurer votre étape.

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.

Modification 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. Si vous avez modifié le point de terminaison par défaut de votre API REST, lorsque vous mettez à jour une étape, la modification est propagée à toutes les étapes de votre API. Les étapes suivantes vous montrent comment effectuer cette opération dans l'Éditeur d'étape 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 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 de l'API pour l'étape, activez Configurer le cache d'API. 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 au niveau de la méthode par défaut ou le cache au niveau de la méthode pour une méthode spécifique.

Pour plus d'informations sur les paramètres de cache, consultez Paramètres du cache des API REST dans API Gateway.

> **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 pour la configuration et l'exécution d'une API REST dans API Gateway.

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 pour la configuration et l'exécution d'une API REST dans API Gateway.

> **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. Sélectionnez Continuer.
9. Passez en revue vos modifications et choisissez Enregistrer les modifications.
10. 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 Paramètres dans le volet de navigation APIsprincipal. Ensuite, pour le rôle CloudWatch log, entrez l'ARN d'un rôle IAM.
> Pour les scénarios d'application courants, le rôle IAM peut associer la politique gérée d'Amazon APIGateway PushToCloudWatchLogs.
> Le rôle IAM doit également contenir la déclaration de relation d'approbation suivante :
>
> JSON

```
{
  "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.

11. 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.

12. Sélectionnez Data tracing pour qu'API Gateway fasse rapport à CloudWatch la journalisation du suivi des données de votre étape. Cela peut être utile pour résoudre les problèmes APIs, mais peut entraîner l'enregistrement de données sensibles.

    Note

    Nous vous recommandons de ne pas utiliser le suivi des données pour la production APIs.

13. 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.

14. Pour activer la journalisation des accès à une destination, activez Journalisation des accès personnalisée.

15. Pour ARN de destination des journaux d'accès, saisissez l'ARN d'un groupe de journaux ou d'un flux Firehose.

    Le format ARN de Firehose est `arn:aws:firehose:{region}:{account-id}:deliverystream/amazon-apigateway-{your-stream-name}`. Le nom de votre flux Firehose doit être `amazon-apigateway-{your-stream-name}`.

16. Dans Format de journal, entrez un format de journal. Pour en savoir plus sur les exemples de formats de journal, consultez the section called "Formats de journal d'accès".

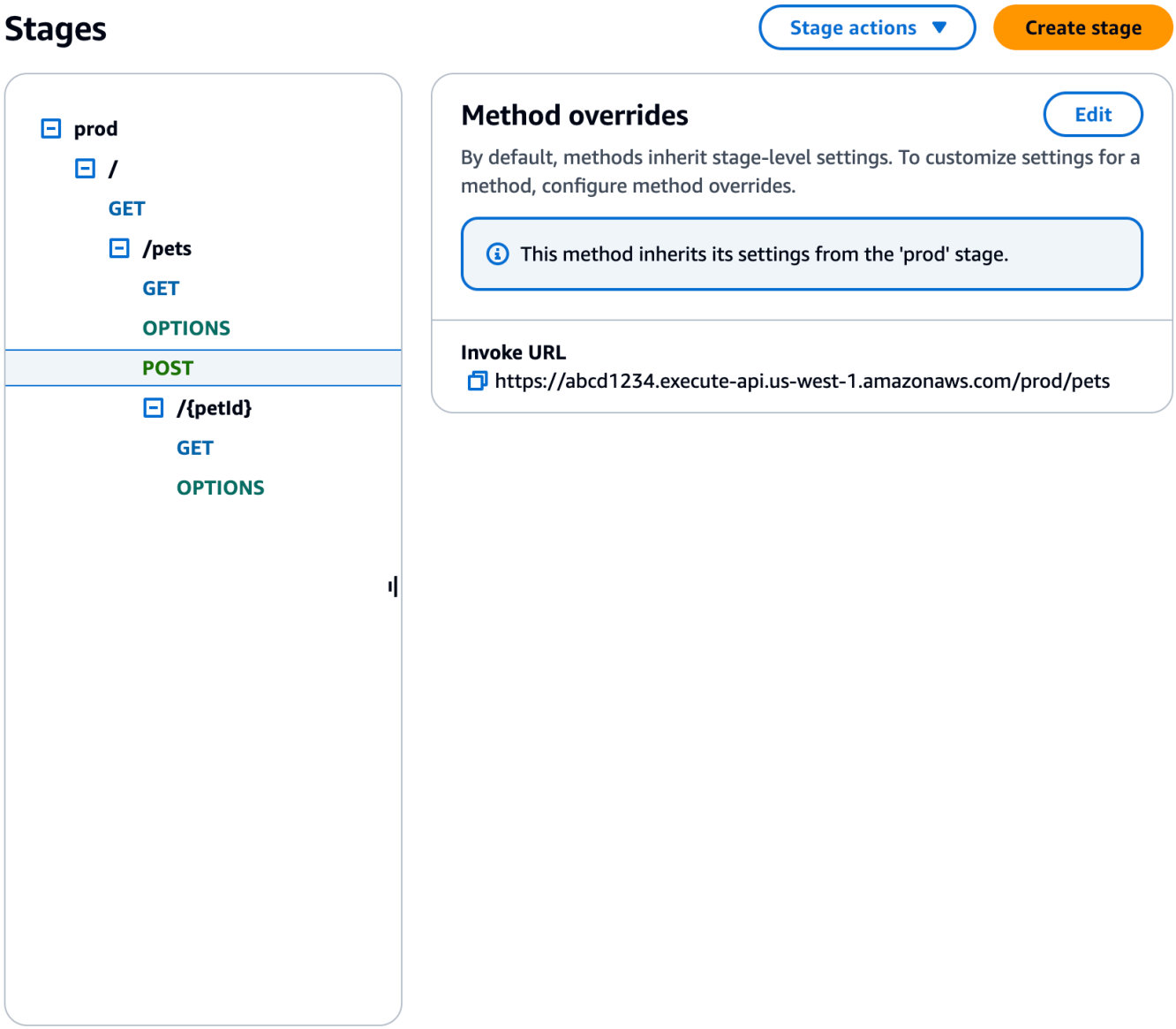
17. Pour activer le suivi AWS X-Ray pour l'étape d'API, sélectionnez Suivi X-Ray. Pour de plus amples informations, veuillez consulter Suivi des demandes des utilisateurs adressées aux API REST à l'aide de X-Ray dans API Gateway.

18. Sélectionnez Save Changes (Enregistrer les modifications). Redéployez votre API pour que les nouveaux paramètres prennent effet.

Remplacement des paramètres au niveau de l'étape

Après avoir personnalisé les paramètres au niveau de l'étape, vous pouvez les remplacer pour chaque méthode d'API. Certaines de ces options peuvent entraîner des frais supplémentaires pour votre Compte AWS.

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 la journalisation du suivi des données pour votre méthode, sélectionnez Suivi des données.

> **Note**
>
> Nous vous recommandons de ne pas utiliser le suivi des données pour la production APIs.

5. 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.
6. 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 “étranglement”.
7. Pour configurer le cache au niveau de la méthode, sélectionnez Activer le cache de la méthode. Si vous modifiez le paramètre de mise en cache au niveau de la méthode par défaut dans Détails de l’étape, ce paramètre n’en sera pas affecté.
8. Choisissez Enregistrer.

Ajouter une API REST API Gateway en tant que cible pour Amazon Bedrock AgentCore Gateway

Une AgentCore passerelle Amazon Bedrock fournit aux développeurs d'agents d'intelligence artificielle un moyen sécurisé d'exposer votre API Gateway REST en APIs tant qu'outils compatibles avec le Model Context Protocol (MCP). AgentCore Gateway utilise des cibles pour définir les outils. Lorsque vous ajoutez votre stage en tant que cible, votre passerelle devient une URL MCP unique qui permet à un agent d'accéder aux outils. Pour plus d'informations, consultez les étapes de l'API REST API Gateway en tant que cibles dans le manuel Amazon Bedrock AgentCore Gateway Developer Guide.

Les cibles d'API Gateway connectent votre AgentCore passerelle aux étapes de votre REST APIs. Vous pouvez inclure l'ensemble de l'étape en tant que cible ou sélectionner des ressources. Après avoir créé la cible API Gateway, AgentCore Gateway traduit les demandes MCP entrantes en requêtes HTTP et gère le formatage des réponses. Les clients MCP peuvent récupérer la documentation de l'API à l'aide de la `tools/list` méthode et l'invoquer APIs à l'aide de la `tools/call` méthode.

Considérations

Les considérations suivantes peuvent avoir un impact sur votre utilisation de l'ajout d'une étape en tant que cible à une AgentCore passerelle :

- Vous devez déjà disposer d'une AgentCore passerelle.
- Seul le REST public APIs est pris en charge.
- Le point de terminaison par défaut de votre API ne peut pas être désactivé.

- Chaque méthode de votre API doit soit avoir un nom d'opération défini, soit vous devez créer un remplacement de nom lorsque vous ajoutez votre étape en tant que cible. Ce nom est utilisé comme nom d'outil que les agents utilisent pour interagir avec votre méthode.
- Vous pouvez utiliser `API_KEYNO_AUTH`, ou des types de fournisseurs `GATEWAY_IAM_ROLE` d'informations d'identification pour l'authentification sortante afin de permettre à votre passerelle d'accéder à votre API. Le fournisseur `API_KEY` d'informations d'identification est défini par AgentCore Gateway. Vous pouvez utiliser votre clé d'API API Gateway existante. Pour plus d'informations, consultez la section Configuration de l'authentification sortante.
- Si vous utilisez un groupe d'utilisateurs Amazon Cognito ou un autorisateur Lambda pour contrôler l'accès à votre API, les clients MCP ne peuvent pas y accéder.
- Votre API doit se trouver dans le même compte et dans la même région que votre AgentCore passerelle.

Ajouter une étape d'une API en tant que cible pour une AgentCore passerelle

La procédure suivante montre comment ajouter une étape d'une API en tant que cible pour une AgentCore passerelle.

Pour ajouter une étape d'une API en tant que cible pour une AgentCore passerelle

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Choisissez une API REST déployée sur une étape.
3. Dans le volet de navigation principal, choisissez Étapes.
4. Choisissez Stage actions, puis Create MCP target.
5. Pour AgentCore Gateway, sélectionnez une AgentCore Gateway.
6. Pour Nom de la cible, entrez un nom de cible.
7. Pour Description de la cible, entrez une description.
8. Conservez l'API et le stage fournis.
9. Pour Select API resources, sélectionnez les ressources de votre API auxquelles les agents utilisant votre AgentCore passerelle peuvent accéder.

   Si vous ne sélectionnez aucune ressource, un agent ne peut pas consulter la documentation ni appeler le point de terminaison.

10. La combinaison de la ressource et de la méthode correspond aux opérations de l'outil. Si votre opération n'a pas de nom, créez un remplacement de nom.

    Vous pouvez également définir un nom d'opération pour une méthode lorsque vous la créez.
11. Pour la configuration de l'authentification sortante, choisissez le rôle IAM, Aucune autorisation ou la clé API.
12. Sélectionnez Créer une cible.

Pour afficher toutes les AgentCore passerelles qui ont accès à votre compte APIs, choisissez la section cibles MCP dans le volet de navigation principal. Dans cette section, vous pouvez créer une cible MCP pour n'importe quelle API de votre région déployée sur une étape. Choisissez Create MCP target et suivez les étapes précédentes.

Vous pouvez également consulter les outils disponibles pour votre cible et modifier votre cible dans la console AgentCore Gateway. Pour plus d'informations, voir Ajouter des cibles à une AgentCore passerelle existante.

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.

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 de 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, l'AWS CLI/le kit 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 plus d'informations sur le balisage des ressources API Gateway, consultez Balisage.

Rubriques

- Configuration des balises pour une étape d'API à l'aide de la console API Gateway
- Configuration de balises pour une étape d'API à l'aide de l'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.

Configuration de balises pour une étape d'API à l'aide de l'AWS CLI

Vous pouvez configurer des balises pour une étape d'API à l'aide de l'AWS CLI en utilisant la commande create-stage ou tag-resource. Vous pouvez supprimer une ou plusieurs balises d'une étape d'API à l'aide de la commande untag-resource.

La commande create-stage suivante ajoute une balise lors de la création d'une étape `test` :

```
aws apigateway create-stage --rest-api-id abc1234 --stage-name test --description
 'Testing stage' --deployment-id efg456 --tag Department=Sales
```

La commande tag-resource suivante ajoute une balise à une étape `prod` :

```
aws apigateway tag-resource --resource-arn arn:aws:apigateway:us-east-2::/
restapis/abc123/stages/prod --tags Department=Sales
```

La commande untag-resource suivante supprime la balise `Department=Sales` de l'étape `test` :

```
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"
    }
}
```

Utilisation de variables d'étape pour une API REST dans API Gateway

Les variables d'étape sont des paires clé-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. Grâce aux étapes de déploiement d'API Gateway, vous pouvez gérer plusieurs étapes de publication pour chaque API et utiliser des variables de phase pour configurer une étape de déploiement d'API afin d'interagir avec différents points de terminaison principaux.

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 the section called "Sortie d'un mécanisme d'autorisation Lambda API Gateway".

Cas d'utilisation des variables d'étape

Voici quelques cas d'utilisation pour vos variables d'étape.

Spécification d'un autre point de terminaison backend

Votre API peut transmettre une demande `GET` en tant que proxy HTTP à l'hôte web backend. Vous pouvez utiliser une variable d'étape afin qu'API Gateway appelle `example.com` lorsque les appelants d'API invoquent votre point de terminaison de production. Ensuite, lorsque les appelants d'API invoquent l'étape bêta, API Gateway appelle un autre hôte web, par exemple `beta.example.com`. De même, vous pouvez utiliser des variables d'étape pour spécifier un nom de fonction AWS Lambda différent pour chaque étape de votre API. Vous ne pouvez pas utiliser une variable d'étape pour définir un autre point de terminaison d'intégration, par exemple pour pointer la demande `GET` vers une intégration de proxy HTTP dans une étape et une intégration de proxy Lambda dans une autre étape.

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. Pour ce faire, vous pouvez également utiliser la AWS CLI commande suivante.

```
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
```

Transmission d'informations à l'aide de modèles de mappage

Vous pouvez également accéder aux variables d'étape dans les modèles de mappage, ou transmettre les paramètres de configuration à votre backend AWS Lambda ou HTTP. 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. 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.

Pour utiliser une variable d'étape, vous devez d'abord la configurer, puis lui attribuer une valeur. Par exemple, pour personnaliser le point de terminaison d'intégration HTTP, créez d'abord la variable d'étape `url`, puis saisissez sa valeur, **`http://${stageVariables.url}`**, dans la demande d'intégration de votre API. Cette valeur dit à API Gateway de remplacer votre variable d'étape `${}` au moment de l'exécution, en fonction de l'étape où se trouve votre API. Pour plus d'informations, consultez the section called "Configuration de variables d'étape pour les API REST dans API Gateway".

Configuration de variables d'étape pour les API REST dans API Gateway

Cette section explique comment définir diverses variables d'étape pour deux étapes de déploiement d'un exemple d'API à l'aide de la console Amazon API Gateway. Pour comprendre comment utiliser les variables d'étape dans API Gateway, nous vous recommandons de suivre toutes les procédures décrites dans cette section.

Conditions préalables

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'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 API Gateway.
- Vous devez avoir créé la première étape pour une API déployée. Suivez les instructions de la section Création d'une étape.

Invocation d'un point de terminaison HTTP via une API avec une variable d'étape

Cette procédure décrit comment créer une variable d'étape pour un point de terminaison HTTP et deux étapes pour votre API. En outre, vous devez créer les variables d'étape `url`, `stageName` et `function` qui sont utilisées dans les procédures suivantes de cette section.

Pour invoquer un point de terminaison HTTP via une API avec une variable d'étape

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 Nom, saisissez **`url`**. Pour Valeur, entrez **`httpbin.org/get`**.
8. Choisissez Ajouter une variable d'étape, puis procédez comme suit :

   Pour Nom, saisissez **`stageName`**. Pour Valeur, entrez **`beta`**.
9. Choisissez Ajouter une variable d'étape, puis procédez comme suit :

   Pour Nom, saisissez **`function`**. Pour Valeur, entrez **`HelloWorld`**.
10. Cliquez sur 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.
13. 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.

14. 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 http://httpbin.org/get. Cette valeur a été affectée à la variable `url` à l'étape beta. Les deux réponses sont identiques.
15. 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.
16. 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 de métadonnées spécifiques à l'étape à un backend HTTP

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 procédure précédente.

Pour transmettre des métadonnées spécifiques à l'étape à un backend HTTP

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 Nom, saisissez **`stageName`**.

   c. Gardez Obligatoire et Mise en cache désactivés.
3. Cliquez sur 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 bêta car le point de terminaison HTTP (tel que spécifié par la `url` variable http://httpbin.org/get « ») accepte les expressions de paramètres de requête et les renvoie en tant qu'`args`objet 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"
}
```

Invocation 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. Vous allez utiliser la variable d'étape `function` déclarée dans la section Invocation d'un point de terminaison HTTP via une API avec une variable d'étape.

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.

Pour invoquer une fonction Lambda via une API avec une variable d'étape

1. Créez une fonction Lambda nommée à **`HelloWorld`** l'aide du Node.js moteur d'exécution 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. Sélectionnez la ressource /lambdav1, puis choisissez Créer une méthode.

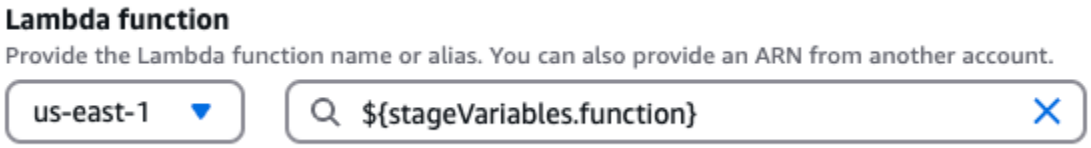
   Ensuite, procédez comme suit :

   a. Pour Type de méthode, sélectionnez GET.

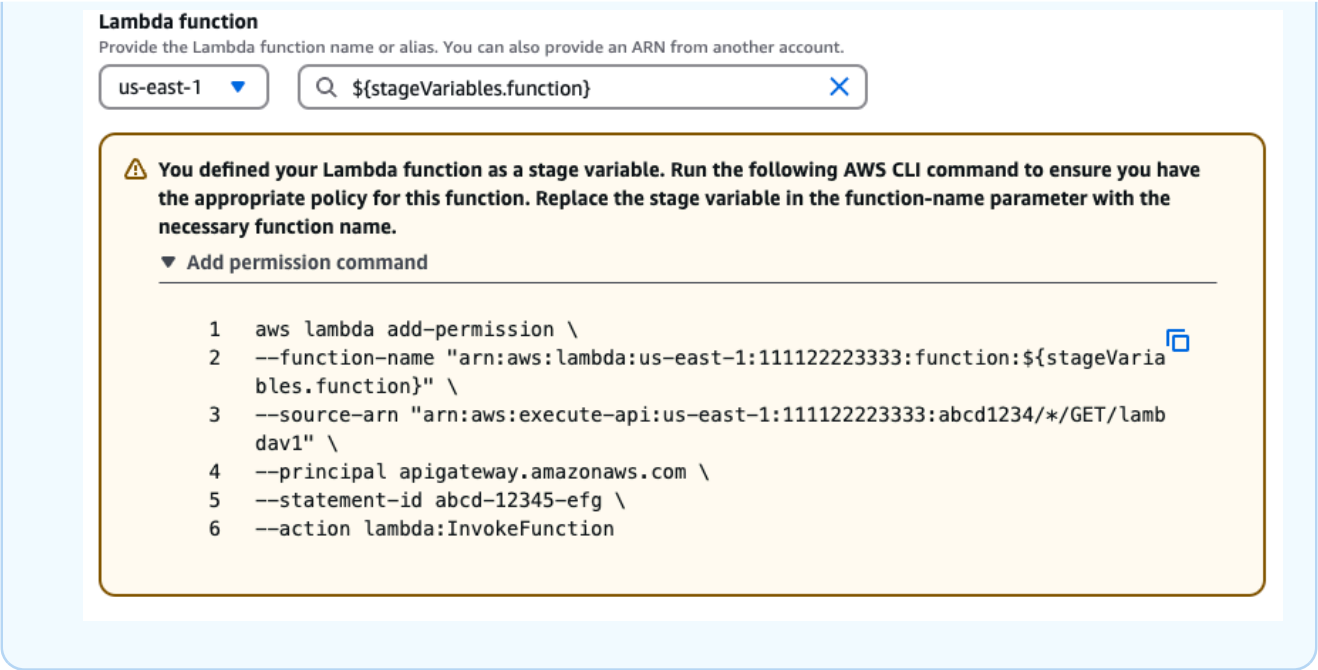
   b. Pour Type d'intégration, sélectionnez Fonction Lambda.

   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 add-permission. Exécutez la commande sur chaque Fonction Lambda qui sera affectée à la variable d'étape `function`. Par exemple, si la valeur `$stageVariables.function` est `HelloWorld`, exécutez la commande AWS CLI suivante :
>
> ```
> aws lambda add-permission --function-name arn:aws:lambda:us-
> east-1:account-id:function:HelloWorld --source-arn arn:aws:execute-
> api:us-east-1:account-id:api-id/*/GET/lambdav1 --principal
>  apigateway.amazonaws.com --statement-id statement-id-guid --action
>  lambda:InvokeFunction
> ```
>
> Si vous ne le faites pas, l'appel de la méthode renverra une réponse `500 Internal Server Error`. Remplacez `${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 dans les 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.

Pour transmettre des métadonnées spécifiques à l'étape à une fonction Lambda via une variable d'étape

1. Sélectionnez la ressource /lambdav1, puis choisissez Créer une méthode.

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 add-permission. 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. Cliquez sur Enregistrer.
7. Déployez l'API dans les 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."
```

Informations de référence sur les variables d'étape API Gateway pour les API REST dans 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 AWS URI ou d'un composant 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 du nom d'une fonction Lambda ou version/alias, comme indiqué 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 mécanisme d’autorisation `COGNITO_USER_POOLS`.

- `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 AWS user/role d'identification, comme illustré 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 politique 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
- Création d'un déploiement de version Canary
- Mise à jour d'une version Canary
- Promotion d'une version Canary
- Désactivation d'une version canary

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 de stage pour les requêtes Canary, s'il useStageCacheest défini et si la mise en cache de l'API est activée sur le stage.

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 CloudWatch journaux Logs en phase de production ainsi qu'à un groupe de CloudWatch journaux Logs spécifique à 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 de Canary peut utiliser le cache de stage, s'il est activé, pour stocker les réponses et utiliser les entrées mises en cache pour renvoyer les résultats aux prochaines demandes Canary, dans un délai préconfiguré time-to-live (TTL).

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 associées 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, vous 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 AWS CLI, du ou d'un AWS SDK.

Rubriques

- Création d'un déploiement Canary à l'aide de la console API Gateway
- Créez un déploiement Canary à l'aide du AWS CLI

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 des API ou associer une ACL AWS WAF Web 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 les 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 "Utilisation de 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.

Créez un déploiement Canary à l'aide du AWS CLI

Pour créer un déploiement canary pour une nouvelle étape

1. Utilisez la commande create-deployment pour créer un déploiement avec deux variables d'étape, mais sans canary :

```
aws apigateway create-deployment \
    --variables sv0=val0,sv1=val1 \
    --rest-api-id abcd1234 \
    --stage-name 'prod'
```

   Le résultat se présente comme suit :

```
{
    "id": "du4ot1",
    "createdDate": 1511379050
```

```
}
```

2. Utilisez la commande create-deployment suivante pour créer un déploiement canary sur l'étape `prod` :

```
aws apigateway create-deployment \
    --rest-api-id abcd1234 \
    --canary-settings
 percentTraffic=10.5,stageVariableOverrides={sv1='val2',sv2='val3'},useStageCache=false
 \
    --stage-name 'prod'
```

Le résultat se présente comme suit :

```
{
    "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.

3. (Facultatif) Utilisez la commande get-stage suivante pour afficher la représentation de l'étape :

```
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 également créer un déploiement de version canary à partir d'un déploiement standard existant en mettant à jour l'étape de façon à activer un canary.

Pour créer un déploiement de version canary à partir d'un déploiement existant

1. Utilisez la commande create-deployment pour créer un déploiement sans canary :

```
aws apigateway create-deployment \
    --variables sv0=val0,sv1=val1 \
    --rest-api-id abcd1234 \
    --stage-name 'beta'
```

2. Utilisez la commande update-stage pour mettre à jour l'étape de façon à activer un canary :

```
aws apigateway update-stage \
    --rest-api-id abcd1234 \
    --stage-name 'beta' \
    --patch-operations '[{
            "op": "replace",
            "value": "0.0",
            "path": "/canarySettings/percentTraffic"
        }, {
            "op": "copy",
            "from": "/canarySettings/stageVariableOverrides",
```

```
            "path": "/variables"
        }, {
            "op": "copy",
            "from": "/canarySettings/deploymentId",
            "path": "/deploymentId"
        }]'
```

Le résultat se présente comme suit :

```
{
    "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 vous venez d'activer un 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. 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 de Canary à l'aide de la console API Gateway, de la commande AWS CLI update-stage ou d'un AWS SDK.

Rubriques

- Mise à jour d'une version Canary à l'aide de la console API Gateway
- Mettez à jour une version de Canary à l'aide du 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 Enregistrer.

Mettez à jour une version de Canary à l'aide du AWS CLI

Pour mettre AWS CLI à jour un canari, utilisez la `update-stage`commande et modifiez l'opération de correctif pour chaque paramètre du canari.

La commande update-stage suivante effectue une mise à jour si le canary utilise le cache de l'étape :

```
aws apigateway update-stage \
    --rest-api-id {rest-api-id} \
    --stage-name '{stage-name}' \
    --patch-operations op=replace,path=/canarySettings/useStageCache,value=true
```

La commande update-stage suivante met à jour le pourcentage de trafic canary :

```
aws apigateway update-stage \
    --rest-api-id {rest-api-id} \
    --stage-name '{stage-name}' \
    --patch-operations op=replace,path=/canarySettings/percentTraffic,value=25.0
```

La commande update-stage suivante met à jour les variables de l'étape. L'exemple suivant montre comment créer une variable d'étape nommée `newVar`, remplacer la variable d'étape `var2` et supprimer la variable d'étape `var1` :

```
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

Lors de la promotion d'une version Canary, cette dernière remplace les paramètres d'étape en cours. 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. Pour promouvoir une version Canary, procédez comme suit.

- Réinitialisez l'ID de déploiement de l'étape avec les paramètres d'ID de déploiement de la version Canary. Cette tâche a 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%.

Rubriques

- Promotion d'une version Canary à l'aide de la console API Gateway
- Faites la promotion d'une version de Canary à l'aide du 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 le canary.
5. Confirmez les modifications à apporter, puis choisissez Promouvoir le 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 du AWS CLI. Pour obtenir un exemple, consultez the section called "Faites la promotion d'une version de Canary à l'aide du AWS CLI".

Faites la promotion d'une version de Canary à l'aide du AWS CLI

Pour promouvoir une version de Canary en version de production à l'aide des AWS CLI commandes, appelez la `update-stage` commande pour copier la version associée au canary `deploymentId` vers la version associée à l'étape`deploymentId`, pour remettre le pourcentage de trafic Canary à zéro (`0.0`) et pour copier toutes les variables d'étape liées au canary dans celles liées à l'étape correspondantes.

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"
    }
}
```

Utilisez la commande update-stage suivante pour promouvoir le canary :

```
aws apigateway update-stage  \
    --rest-api-id {rest-api-id}  \
    --stage-name '{stage-name}'  \
    --patch-operations '[{
        "op": "replace",
        "value": "0.0",
        "path": "/canarySettings/percentTraffic"
      }, {
        "op": "copy",
        "from": "/canarySettings/stageVariableOverrides",
        "path": "/variables"
      }, {
        "op": "copy",
        "from": "/canarySettings/deploymentId",
        "path": "/deploymentId"
      }]'
```

Le résultat 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"
    }
}
```

La promotion d'une version Canary à l'étape ne désactive pas le 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 le déploiement d'une version Canary à l'aide de la console API Gateway AWS CLI, du ou d'un AWS SDK.

Rubriques

- Désactivation d'une version canary à l'aide de la console API Gateway
- Désactivez une version Canary à l'aide du 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. Sélectionnez Delete (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 du AWS CLI. Pour obtenir un exemple, consultez the section called "Désactivez une version Canary à l'aide du AWS CLI".

Désactivez une version Canary à l'aide du AWS CLI

La commande update-stage suivante désactive le déploiement de la version canary :

```
aws apigateway update-stage \
    --rest-api-id abcd1234 \
    --stage-name canary \
    --patch-operations '[{"op":"remove", "path":"/canarySettings"}]'
```

Le résultat se présente comme suit :

```
{
    "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 de REST APIs 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 CloudFormation, 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.

Le tableau suivant décrit les ressources d'API qui nécessitent le redéploiement de votre API lorsque vous les mettez à jour.

Ressource	Remarques
ApiKey	Pour les propriétés applicables et les opérations prises en charge, consultez 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, consultez authorizer:update. La mise à jour nécessite le redéploiement de l'API.
disableExecuteApiPoint de terminaison	La mise à jour nécessite de modifier n'importe quelle étape de l'API (redéploiement de l'API dans une étape, par exemple).
DocumentationPart	Pour les propriétés applicables et les opérations prises en charge, consultez documentationpart:update. La mise à jour nécessite le redéploiement de l'API.

Ressource	Remarques
DocumentationVersion	Pour les propriétés applicables et les opérations prises en charge, consultez documentationversion:update. La mise à jour nécessite le redéploiement de l'API.
GatewayResponse	Pour les propriétés applicables et les opérations prises en charge, consultez gatewayresponse:update. La mise à jour nécessite le redéploiement de l'API.
Integration	Pour les propriétés applicables et les opérations prises en charge, consultez integration:update. La mise à jour nécessite le redéploiement de l'API.
IntegrationResponse	Pour les propriétés applicables et les opérations prises en charge, consultez integrationresponse:update. La mise à jour nécessite le redéploiement de l'API.
Method	Pour les propriétés applicables et les opérations prises en charge, consultez method:update. La mise à jour nécessite le redéploiement de l'API.
MethodResponse	Pour les propriétés applicables et les opérations prises en charge, consultez methodresponse:update. La mise à jour nécessite le redéploiement de l'API.
Modèle	Pour les propriétés applicables et les opérations prises en charge, consultez model:update. La mise à jour nécessite le redéploiement de l'API.
RequestValidator	Pour les propriétés applicables et les opérations prises en charge, consultez requestvalidator:update. La mise à jour nécessite le redéploiement de l'API.
Ressource	Pour les propriétés applicables et les opérations prises en charge, consultez resource:update. La mise à jour nécessite le redéploiement de l'API.
RestApi	Pour les propriétés applicables et les opérations prises en charge, consultez restapi:update. La mise à jour nécessite le redéploiement de l'API. Cela inclut la modification des stratégies de ressources.
VpcLink	Pour les propriétés applicables et les opérations prises en charge, consultez vpclink:update. La mise à jour nécessite le redéploiement de l'API.

Le tableau suivant décrit les configurations d'API qui ne nécessitent pas le redéploiement de votre API lorsque vous les mettez à jour.

Configuration	Remarques
Compte	Pour les propriétés applicables et les opérations prises en charge, consultez 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, consultez deployment:update.
DomainName	Pour les propriétés applicables et les opérations prises en charge, consultez domainname:update. La mise à jour ne nécessite pas le redéploiement de l'API.
BasePathMapping	Pour les propriétés applicables et les opérations prises en charge, consultez basepathmapping:update. La mise à jour ne nécessite pas le redéploiement de l'API.
Type d'adresse IP	La mise à jour ne nécessite pas le redéploiement de l'API.
Étape	Pour les propriétés applicables et les opérations prises en charge, consultez 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, consultez 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, consultez usageplan:update. La mise à jour ne nécessite pas le redéploiement de l'API.

Nom de domaine personnalisé pour le REST public APIs dans API Gateway

Les noms de domaine personnalisés sont plus simples et plus intuitifs URLs et vous pouvez les fournir aux utilisateurs de votre 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* est la AWS région et *stage* est spécifiée par vous lors du déploiement de l'API.

La partie nom d'hôte de l'URL, *api-id*`.execute-api.`*region*`.amazonaws.com`, fait référence à un point de terminaison de l'API. Le nom par défaut du point de terminaison de l'API est généré de manière aléatoire, difficile à mémoriser et peu 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**
>
> Pour plus d'informations sur les noms de domaine personnalisés pour le secteur privé APIs, consultezthe section called "Noms de domaine personnalisés pour les API privées".

Considérations

Les considérations suivantes peuvent avoir une incidence sur votre utilisation des noms de domaine personnalisés :

- Vous pouvez désactiver le point de terminaison par défaut de votre API. Les clients peuvent toujours se connecter à votre point de terminaison par défaut, mais ils recevront un code de statut `403 Forbidden`.
- Un nom de domaine personnalisé régional peut être associé à REST APIs et HTTP APIs. Vous pouvez utiliser l'API Gateway version 2 APIs pour créer et gérer des noms de domaine personnalisés régionaux pour REST APIs.
- Un nom de domaine personnalisé doit être unique au sein d'une région pour tous les AWS comptes.
- Vous pouvez migrer votre nom de domaine personnalisé entre des points de terminaison optimisés pour la périphérie et des points de terminaison régionaux, mais vous ne pouvez pas migrer un domaine personnalisé public vers un nom de domaine personnalisé privé.
- Vous devez créer ou mettre à jour l'enregistrement de ressource de votre fournisseur DNS pour le mapper au point de terminaison de votre API. Sans ce mappage, les demandes d'API destinées au nom de domaine personnalisé ne peuvent pas atteindre API Gateway.

- Vous pouvez prendre en charge un nombre presque infini de noms de domaine sans dépasser le quota par défaut en utilisant un certificat générique. Pour de plus amples informations, veuillez consulter the section called “Noms de domaine personnalisés génériques”.
- Vous pouvez choisir une politique de sécurité pour votre nom de domaine personnalisé. Pour de plus amples informations, veuillez consulter the section called “Sélection d’une politique de sécurité”.
- Pour configurer des mappages d’API à plusieurs niveaux, vous devez utiliser un nom de domaine personnalisé régional et la politique de sécurité TLS 1.2.

Prérequis concernant les noms de domaine personnalisés

Les conditions suivantes sont requises pour créer un nom de domaine personnalisé public ou privé. Pour plus d'informations sur les noms de domaine personnalisés pour le secteur privé APIs, consultezthe section called “Noms de domaine personnalisés pour les API privées”.

Enregistrement d’un nom de domaine

Vous devez avoir un nom de domaine Internet enregistré afin de configurer des noms de domaine personnalisés pour votre APIs. Vous pouvez enregistrer votre nom de domaine Internet avec Amazon Route 53 ou utiliser un bureau d’enregistrement de domaine tiers de votre choix. Votre nom de domaine personnalisé peut être le nom d’un sous-domaine ou le domaine racine (également nommé « zone apex ») d’un domaine Internet enregistré.

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.

Spécification du certificat pour votre nom de domaine personnalisé

Avant de configurer un nom de domaine personnalisé pour une API, vous devez disposer d'un SSL/TLS certificat dans ACM. Si ACM n'est pas disponible dans la AWS région où vous créez votre nom de domaine personnalisé, vous devez importer un certificat dans API Gateway dans cette région.

Pour importer un SSL/TLS certificat, vous devez fournir le corps du SSL/TLS certificat au format PEM, sa clé privée et la chaîne de certificats pour le nom de domaine personnalisé.

Chaque certificat stocké dans ACM est identifié par son ARN. 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 utiliser un certificat AWS géré pour un nom de domaine, il suffit de référencer son ARN.

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 .

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`*`.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` donne lieu à des sous-domaines comme `a.example.com`, `b.example.com` et `c.example.com`. Lorsque vous créez le nom de domaine personnalisé générique, tous ses sous-domaines sont acheminés par le mode de routage du nom de domaine générique. Pour acheminer des sous-domaines vers différents APIs, vous pouvez effectuer l'une des opérations suivantes :

- Utilisez les règles de routage pour acheminer les demandes entrantes `*.example.com` vers différents REST cibles à APIs l'aide de l'`Host`en-tête. Pour de plus amples informations, veuillez consulter the section called "Exemple 4 : règles de routage pour les noms de domaine génériques".
- Créer un nom de domaine pour tous les sous-domaines que vous souhaitez acheminer vers un autre point de terminaison. Dans un seul AWS compte, vous pouvez avoir les deux `*.example.com` eta`.example.com`.

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, consultez Variables pour les transformations de données pour API Gateway.

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.

Étapes suivantes pour les noms de domaine personnalisés

Les étapes suivantes pour les noms de domaine personnalisés sont présentées ci-dessous.

Étapes suivantes

- Pour savoir comment configurer votre SSL/TLS certificat, consultezthe section called “Préparez les certificats dans AWS Certificate Manager”.
- Pour apprendre à créer un nom de domaine personnalisé régional, consultez the section called “Configuration d’un nom de domaine personnalisé régional”.
- Pour apprendre à créer un nom de domaine personnalisé optimisé pour la périphérie, consultez the section called “Configuration d’un nom de domaine personnalisé optimisé pour la périphérie dans API Gateway”.
- Pour apprendre à migrer des noms de domaine personnalisés régionaux et des noms de domaine personnalisés optimisés pour la périphérie, consultez the section called “Migration de noms de domaine personnalisés”.
- Pour apprendre à connecter des étapes d’API à un nom de domaine personnalisé, consultez the section called “Envoyez du trafic vers vous APIs via votre nom de domaine personnalisé dans API Gateway”.
- Pour apprendre à choisir une politique de sécurité pour votre nom de domaine personnalisé, consultez the section called “Sélection d’une politique de sécurité”.
- Pour apprendre à désactiver le point de terminaison par défaut de votre nom de domaine personnalisé, consultez the section called “Désactivation du point de terminaison par défaut”.
- Pour apprendre à utiliser les surveillances de l’état de Route 53 pour contrôler le basculement DNS d’une API API Gateway, consultez the section called “basculement DNS”.

Si c'est la première fois que vous créez un nom de domaine personnalisé, nous vous recommandons de commencer par la section the section called "Préparez les certificats dans AWS Certificate Manager" pour spécifier votre certificat, puis la section the section called "Configuration d'un nom de domaine personnalisé régional" pour créer un nom de domaine personnalisé régional.

Préparez les 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. Pour plus d'informations, consultez le Guide de l'utilisateur AWS Certificate Manager.

Considérations

Voici quelques points à prendre en compte pour votre SSL/TLS certificat.

- Si vous créez un nom de domaine personnalisé optimisé pour les périphériques, API Gateway s'en sert pour prendre en charge les CloudFront certificats pour les noms de domaine personnalisés. Ainsi, les exigences et les contraintes d'un SSL/TLS certificat 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 plus d'informations, voir Accès sécurisé à vos objets et Création de cookies signés URLs et signés.
- Si vous créez un nom de domaine personnalisé régional, la taille maximale de la clé publique est de 2048.
- Pour utiliser un certificat ACM avec un nom de domaine personnalisé régional, vous devez demander ou importer le certificat dans la même région que votre API. Le certificat doit couvrir le nom de domaine personnalisé.
- Pour utiliser un certificat ACM avec un nom de domaine personnalisé optimisé pour la périphérie, vous devez demander ou importer le certificat dans la région `us-east-1` – USA Est (Virginie du Nord).
- Vous devez disposer d'un nom de domaine enregistré, 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 un SSL/TLS certificat dans ACM

Les procédures suivantes indiquent comment créer ou importer un SSL/TLS certificat pour un nom de domaine.

To request a certificate provided by ACM for a domain name

1. Connectez-vous à la console AWS Certificate Manager.
2. Choisissez Request a certificate.
3. Pour Type de certificat, choisissez Demander un certificat public.
4. Choisissez Suivant.
5. Pour Nom de domaine complet, saisissez un nom de domaine personnalisé pour votre API, par exemple, `api.example.com`.
6. Vous pouvez également choisir Ajouter un autre nom à ce certificat.
7. Pour Méthode de validation, choisissez une méthode pour valider la propriété du domaine.
8. Pour Algorithme de la clé, choisissez un algorithme de chiffrement.
9. Cliquez sur Demander.
10. Pour que la demande soit valide et qu'ACM puisse émettre le certificat, le propriétaire inscrit du domaine Internet doit préalablement approuver cette demande. Si vous utilisez Route 53 pour gérer vos enregistrements DNS publics, vous pouvez mettre à jour vos enregistrements directement via la console ACM.

To import into ACM a certificate for a domain name

1. Obtenez un SSL/TLS certificat codé PEM pour votre nom de domaine personnalisé auprès d'une autorité de certification. Pour une liste partielle de ces entités CAs, consultez la liste des CA incluses par 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
   ```

   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 Corps du certificat, saisissez 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. Pour Clé privée du certificat, saisissez 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. Pour Chaîne de certificats, saisissez les certificats intermédiaires au format PEM et, éventuellement, 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 Suivant, puis Suivant.

Une fois que le certificat a été créé ou importé, notez son ARN. Vous en aurez besoin pour configurer le nom de domaine personnalisé.

Configuration d'un nom de domaine personnalisé régional dans API Gateway

Utilisez un nom de domaine personnalisé régional pour créer une URL de base d'API conviviale. Avec un nom de domaine personnalisé régional, vous pouvez mapper les étapes d'API HTTP et REST au même nom de domaine personnalisé et utiliser l'authentification TLS mutuelle.

Considérations

Voici quelques points à prendre en compte pour votre nom de domaine personnalisé régional :

- Vous devez fournir un certificat ACM spécifique à la région. Ce certificat doit se trouver dans la même région que votre API. Pour plus d'informations sur la création ou le chargement d'un

certificat de nom de domaine personnalisé, consultez the section called “Préparez les certificats dans AWS Certificate Manager”.

- 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. Le rôle lié à un service est requis pour attacher votre certificat ACM à votre point de terminaison régional. Le rôle est nommé AWSServiceRoleForAPIGateway et la politique gérée APIGatewayServiceRolePolicy lui est attachée. Pour plus d’informations sur l’utilisation du rôle lié à un service, consultez Utilisation des rôles liés à un service.
- Après avoir créé votre nom de domaine personnalisé régional, vous devez créer un enregistrement DNS pour pointer le nom de domaine personnalisé vers le 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 un enregistrement CNAME ou A Alias. Si vous utilisez Route 53 comme fournisseur DNS, créez un enregistrement A Alias. Si vous utilisez un fournisseur DNS tiers, utilisez un enregistrement CNAME. Si vous utilisez un enregistrement CNAME et créez un point de terminaison de VPC d’interface API Gateway avec le DNS privé activé pour une API privée, vous ne pouvez pas résoudre le nom de domaine personnalisé dans le VPC qui héberge votre API privée.

Création d’un nom de domaine personnalisé régional

La procédure suivante montre comment créer un nom de domaine personnalisé régional. Après avoir effectué cette procédure, vous allez créer une règle de routage pour acheminer les étapes de votre API vers votre nom de domaine personnalisé.

Console de gestion AWS

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. Pour Mode de routage, choisissez Règles de routage uniquement.

   Dans ce mode de routage, vous pouvez uniquement envoyer du trafic depuis votre nom de domaine personnalisé vers votre APIs en utilisant des règles de routage. Pour de plus amples

informations, veuillez consulter the section called “Envoyez du trafic vers vous APIs via votre nom de domaine personnalisé dans API Gateway”.

6. Pour Version TLS minimale, sélectionnez une version.
7. Sous Configuration du point de terminaison, pour Type de point de terminaison d’API, choisissez Régional.
8. Choisissez un certificat ACM. Le certificat doit se trouver dans la même région que l’API.
9. Choisissez Créer.

AWS CLI

La create-domain-namecommande suivante crée un nom de domaine personnalisé :

```
aws apigatewayv2 create-domain-name \
    --domain-name 'regional.example.com' \
    --domain-name-configurations CertificateArn=arn:aws:acm:us-
west-2:123456789012:certificate/123456789012-1234-1234-1234-12345678 \
    --routing-mode ROUTING_RULE_ONLY
```

Le résultat se présente comme suit :

```
{
    "ApiMappingSelectionExpression": "$request.basepath",
    "DomainName": "regional.example.com",
    "DomainNameConfigurations": [
        {
            "ApiGatewayDomainName": "d-numh1z56v6.execute-api.us-
west-2.amazonaws.com",
            "CertificateArn": "arn:aws:acm:us-
west-2:123456789012:certificate/123456789012-1234-1234-1234-12345678",
            "DomainNameStatus": "AVAILABLE",
            "EndpointType": "REGIONAL",
            "HostedZoneId": "Z2OJLYMUO9EFXC",
            "SecurityPolicy": "TLS_1_2"
        }
        "RoutingMode": "ROUTING_RULE_ONLY"
    ]
}
```

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.

Création d'une règle de routage pour votre nom de domaine personnalisé régional

Après avoir créé votre nom de domaine personnalisé, vous configurez la manière dont le trafic est acheminé de votre nom de domaine personnalisé vers votre APIs. Comme vous avez défini le mode de routage sur`ROUTING_RULE_ONLY`, vous utilisez des règles de routage pour acheminer les demandes entrantes vers votre nom de domaine personnalisé vers votre APIs.

Dans cet exemple, vous allez créer une règle fourre-tout qui achemine toutes les demandes entrantes de votre nom de domaine personnalisé vers une étape de votre API. Vous pouvez également configurer des règles de routage en fonction de différentes conditions d'en-tête et de chemin. Pour de plus amples informations, veuillez consulter the section called "règles de routage".

Console de gestion AWS

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. Dans l'onglet Détails du routage, choisissez Ajouter une règle de routage.
4. Choisissez Ajouter une nouvelle condition pour ajouter une nouvelle condition.
5. Conservez cette règle sans condition. Cette opération achemine toutes les demandes de votre nom de domaine personnalisé vers votre API et votre étape cibles.
6. Sous Action, utilisez le menu déroulant pour sélectionner votre API et votre étape cibles.
7. Choisissez Suivant.
8. Dans le champ Priorité, saisissez **`100`**.

   API Gateway évalue les règles par ordre de priorité, de la valeur la plus basse à la valeur la plus élevée. Comme il s'agit d'une règle fourre-tout, utilisez une priorité élevée afin qu'API Gateway puisse faire correspondre toutes les règles supplémentaires créées en premier.
9. Choisissez Créer une règle de routage.

AWS CLI

La commande `create-routing-rule` suivante crée une règle de routage fourre-tout :

```
aws apigatewayv2 create-routing-rule \
  --domain-name 'regional.example.com' \
  --priority 100 \
  --conditions  \
  --actions '[{
    "InvokeApi": {
      "ApiId": "a1b2c3",
      "Stage": "prod"
    }
  }]'
```

Vous pouvez modifier le mode de routage et créer de nouvelles règles à tout moment. Pour de plus amples informations, veuillez consulter the section called “Envoyez du trafic vers vous APIs via votre nom de domaine personnalisé dans API Gateway”.

Création d’un enregistrement DNS pour votre nom de domaine personnalisé régional

Après avoir créé votre nom de domaine personnalisé et créé les mappages de chemins de base, vous allez créer un enregistrement DNS pour pointer votre nom de domaine personnalisé vers le nom de domaine régional que vous venez de créer.

Console de gestion AWS

Pour utiliser le Console de gestion AWS, suivez la documentation de Route 53 sur la configuration de Route 53 pour acheminer le trafic vers API Gateway.

AWS CLI

Pour configurer vos enregistrements DNS de manière à mapper le nom de domaine personnalisé régional au nom d’hôte de l’ID de la zone hébergée, commencez par créer un fichier JSON qui contient la configuration pour créer un enregistrement DNS pour le nom de domaine régional.

L’exemple `setup-dns-record.json` suivant montre comment créer un enregistrement DNS `A` 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`) configuré dans le cadre 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 la zone hébergée régionale Route 53 IDs dans Amazon API Gateway Endpoints and Quotas.

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

Ce qui suit change-resource-record-setscrée un enregistrement DNS pour votre nom de domaine personnalisé régional :

```
aws route53 change-resource-record-sets \
    --hosted-zone-id Z2OJLYMUO9EFXC \
    --change-batch file://path/to/your/setup-dns-record.json
```

Remplacez `hosted-zone-id` par l'ID de la zone hébergée Route 53 du jeu d'enregistrements DNS de votre compte. La valeur du paramètre `change-batch` pointe vers un fichier JSON (*`setup-dns-record.json`*) dans un dossier (*`path/to/your`*).

Configuration d'un nom de domaine personnalisé optimisé pour la périphérie dans API Gateway

Lorsque vous créez un nom de domaine personnalisé pour une API optimisée pour les périphériques, API Gateway configure une CloudFront distribution 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 . Ce mappage concerne les demandes d'API liées au nom de domaine personnalisé à acheminer vers API Gateway via la distribution mappée CloudFront .

Considérations

Voici quelques points à prendre en compte pour votre nom de domaine personnalisé optimisé pour la périphérie :

- 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.

  Les autorisations suivantes sont requises pour mettre à jour CloudFront les distributions :

  JSON

  ```
  {
      "Version":"2012-10-17",
      "Statement": [
           {
              "Sid": "AllowCloudFrontUpdateDistribution",
              "Effect": "Allow",
              "Action": [
                  "cloudfront:updateDistribution"
              ],
              "Resource": [
                  "*"
              ]
          }
      ]
  }
  ```

- Vous devez demander ou importer un certificat pour votre nom de domaine personnalisé optimisé pour la périphérie dans la région `us-east-1` USA Est (Virginie du Nord).
- 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 CloudTrail, vous devez utiliser cet ID de compte API Gateway. Pour de plus amples informations, veuillez consulter Enregistrez la création d'un nom de domaine personnalisé dans CloudTrail.
- 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

- Il faut compter environ 40 minutes pour qu'un nom de domaine personnalisé optimisé pour la périphérie soit prêt.
- Après avoir créé votre nom de domaine personnalisé optimisé pour les périphériques, vous devez créer un enregistrement DNS pour mapper le nom de domaine personnalisé au nom de CloudFront distribution.

Création d'un nom de domaine personnalisé optimisé pour la périphérie

La procédure suivante décrit comment créer un nom de domaine personnalisé optimisé pour la périphérie pour une API.

Console de gestion AWS

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 Ajouter un nom de domaine.
4. Pour Nom de domaine, entrez un nom de domaine.
5. Pour le mode de routage, choisissez API_MAPPING_ONLY.
6. Pour Type de point de terminaison d'API, choisissez Optimisé pour la périphérie.
7. Choisissez une version TLS minimale.
8. Choisissez un certificat ACM.
9. Choisissez Ajouter un nom de domaine.

REST API

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 affiché dans le résultat. Vous en aurez besoin à l'étape suivante pour définir la cible de l'alias d'enregistrement A du domaine personnalisé dans votre DNS.

Pour des exemples de code de cet appel d'API REST, consultez domainname:create.

Il faut environ 40 minutes pour qu'un nom de domaine personnalisé optimisé pour les périphériques soit prêt, mais la console affiche immédiatement le nom de domaine de CloudFront distribution associé, sous la forme de`distribution-id`.`cloudfront.net`, ainsi que l'ARN du certificat. En attendant, vous pouvez créer un mappage de chemin de base ou une règle de routage, puis configurer l'alias d'enregistrement DNS pour mapper le nom de domaine personnalisé au nom de domaine de CloudFront distribution associé.

Configuration du mappage de chemin de base d'une API à l'aide d'un nom de domaine personnalisé servant de nom d'hôte

Comme vous avez défini le mode de routage sur`API_MAPPING_ONLY`, vous pouvez utiliser le mappage de chemins de base pour utiliser un seul nom de domaine personnalisé comme nom d'hôte de plusieurs APIs. Le mappage de chemin de base rend une API accessible par la combinaison du nom de domaine personnalisé et du chemin de base associé.

Par exemple, si vous avez créé dans API Gateway une API nommée `PetStore` et une autre nommée `Dogs` et que vous avez configuré un nom de domaine personnalisé `api.example.com`, vous pouvez définir l'URL de l'API `PetStore` comme `https://api.example.com`.

Cette opération associe l'API `PetStore` au chemin de base d'une chaîne vide. Si vous définissez l'URL de l'API `PetStore` comme `https://api.example.com/PetStore`, l'API `PetStore` est associée au chemin de base `PetStore`. Vous pouvez affecter un chemin de base `MyDogList` pour l'API `Dogs`. L'URL de `https://api.example.com/MyDogList` est alors l'URL racine de l'API `Dogs`.

Pour configurer des mappages d'API à plusieurs niveaux, vous ne pouvez utiliser qu'un nom de domaine personnalisé régional. Les noms de domaine personnalisés optimisés pour la périphérie ne sont pas pris en charge. Pour de plus amples informations, veuillez consulter the section called "Mappages d'API".

La procédure suivante configure les mappages d'API pour mapper les chemins de votre nom de domaine personnalisé à vos étapes d'API.

Console de gestion AWS

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

2. Sélectionnez Custom Domain Names dans le panneau de navigation principal de la console API Gateway.
3. Choisissez un nom de domaine personnalisé.
4. Choisissez Configurer les mappages d'API.
5. Choisissez Ajouter un nouveau mappage.
6. Spécifiez l'API, l'étape, et le chemin (facultatif) pour le mappage.
7. Choisissez Enregistrer.

REST API

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, consultez basepathmapping:create.

Pour plus de flexibilité quant à la manière dont vous acheminez le trafic vers votre APIs, vous pouvez modifier le mode de routage en `ROUTING_RULE_ONLY` ou `ROUTING_RULE_THEN_API_MAPPING` créer une règle de routage. Pour de plus amples informations, veuillez consulter the section called "Envoyez du trafic vers vous APIs via votre nom de domaine personnalisé dans API Gateway".

Création d'un enregistrement DNS pour votre nom de domaine personnalisé optimisé pour la périphérie

Après avoir créé votre nom de domaine personnalisé optimisé pour la périphérie, configurez l'alias de l'enregistrement DNS.

Nous vous recommandons d'utiliser Route 53 pour créer un alias d'enregistrement A pour votre nom de domaine personnalisé et de 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 plus d'informations, consultez Choix entre des jeux d'enregistrements de ressources avec ou sans alias dans le guide du développeur Amazon Route 53.

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.

Enregistrez la création d'un nom de domaine personnalisé dans 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 journaux sont disponibles dans `us-east-1`. Ces CloudFront distributions étant détenues par API Gateway, chacune des CloudFront distributions signalées est identifiée par l'un des comptes API Gateway spécifiques à la région suivants IDs, au lieu de l'ID 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
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

Région	ID de compte
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
me-south-1	855739686837
me-central-1	614065512851

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 décrites dans the section called “Pour créer ou importer un SSL/TLS certificat dans ACM”.

La procédure suivante explique comment effectuer la rotation d'un certificat pour un nom de domaine.

> **Note**
>
> Il faut compter environ 40 minutes pour effectuer la rotation d'un certificat importé dans ACM.

Console de gestion AWS

1. Demandez ou importez un certificat dans ACM.
2. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
3. Sélectionnez Custom Domain Names dans le panneau de navigation principal de la console API Gateway.
4. Choisissez un nom de domaine personnalisé optimisé pour la périphérie.
5. Pour Configuration du point de terminaison, choisissez Modifier.
6. Pour Certificat ACM, choisissez un certificat dans la liste déroulante.
7. Sélectionnez Enregistrer les modifications pour commencer la rotation du certificat pour le nom de domaine personnalisé.

REST API

Appelez l'action domainname:update, en spécifiant l'ARN du nouveau certificat ACM pour le nom de domaine personnalisé spécifié.

AWS CLI

Ce qui suit update-domain-namemet à jour le certificat ACM pour un nom de domaine optimisé pour les périphériques.

```
aws apigateway update-domain-name \
    --domain-name edge.example.com \
    --patch-operations "op='replace',path='/certificateArn',value='arn:aws:acm:us-
east-2:111122223333:certificate/CERTEXAMPLE123EXAMPLE'"
```

Ce qui suit update-domain-namemet à jour le certificat ACM pour un nom de domaine régional.

```
aws apigateway update-domain-name \
    --domain-name regional.example.com \
```

```
    --patch-operations "op='replace',path='/
regionalCertificateArn',value='arn:aws:acm:us-east-2:111122223333:certificate/
CERTEXAMPLE123EXAMPLE'"
```

Appel de votre API avec des noms de domaine personnalisés lorsque vous utilisez un mappage de chemin de base

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 contrastent un ensemble de valeurs par défaut URLs et URLs de personnalisation correspondantes de deux APIs (`udxjefetqf3duz`) dans une région spécifiée (`us-east-1`), et d'un nom de domaine personnalisé donné (`api.example.com`).

ID d'API	Étape	URL par défaut	Chemin de base	URL personnalisé
udxjef	prod	https://udxjef.execute-api.us-east-1.amazonaws.com/produit	/petstore	https://api.example.com/animalerie
udxjef	tst	https://udxjef.execute-api.us-east-1.amazonaws.com/tst	/petdepot	https://api.example.com/dépôt pour animaux
qf3duz	dev	https://qf3duz.execute-api.us-east-1.amazonaws.com/dev	/bookstore	https://api.example.com/librairie
qf3duz	tst	https://qf3duz.execute-api.us-east-1	/bookstand	https://api.example.com/kiosque

ID d'API	Étape	URL par défaut	Chemin de base	URL personnalisé
		.amazonaws.com/tst		

Pour plus de flexibilité quant à la manière dont vous acheminez le trafic vers votre APIs, vous pouvez créer une règle de routage. Pour de plus amples informations, veuillez consulter the section called "Envoyez du trafic vers vous APIs via votre nom de domaine personnalisé dans API Gateway".

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.

Migration d'un nom de domaine personnalisé vers un autre type de point de terminaison d'API dans API Gateway

Vous pouvez migrer votre nom de domaine personnalisé entre des points de terminaison optimisés pour la périphérie et régionaux. Vous ne pouvez pas migrer un nom de domaine personnalisé public vers un nom de domaine personnalisé privé. 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. Enfin, vous supprimez le point de terminaison obsolète du nom de domaine personnalisé.

Considérations

Voici les points à prendre en compte pour la migration de votre domaine personnalisé entre un point de terminaison d'API régional et un point de terminaison d'API optimisé pour la périphérie :

- Un nom de domaine personnalisé optimisé pour la périphérie nécessite un certificat fourni par ACM de la région `us-east-1` - USA Est (Virginie du Nord). Ce certificat est distribué à tous les emplacements géographiques.
- Un nom de domaine personnalisé régional nécessite un certificat fourni par ACM de la même région qui héberge 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 un nouveau certificat ACM de la région locale de l'API.

- La migration entre un nom de domaine personnalisé optimisé pour la périphérie et un nom de domaine personnalisé régional peut prendre jusqu'à 60 secondes. Le délai de migration dépend également du moment où vous mettez à jour vos enregistrements DNS.
- Vous ne pouvez ajouter une configuration de point de terminaison supplémentaire que si le mode d'accès au point de terminaison est défini sur`BASIC`. Une fois que vous avez deux configurations de point de terminaison, vous ne pouvez pas modifier le mode d'accès au point de terminaison. Pour de plus amples informations, veuillez consulter the section called "Mode d'accès au point de terminaison".
- Si votre nom de domaine personnalisé utilise une politique de sécurité qui commence par`SecurityPolicy_`, lorsque vous ajoutez un nouveau type de configuration de point de terminaison, le mode d'accès au point de terminaison est le même pour les deux types de point de terminaison, et vous devez choisir une politique de sécurité commençant par `SecurityPolicy_` pour le nouveau type de configuration de point de terminaison.

Migration de noms de domaine personnalisés

> **Note**
>
> Pour effectuer la migration, veillez à supprimer le point de terminaison obsolète de votre nom de domaine personnalisé.

La procédure suivante explique comment migrer un nom de domaine personnalisé optimisé pour la périphérie vers un nom de domaine personnalisé régional.

Console de gestion AWS

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 un nom de domaine personnalisé optimisé pour la périphérie.
4. Pour Configuration du point de terminaison, choisissez Modifier.
5. Choisissez Ajouter un point de terminaison régional.
6. Pour Certificat ACM, choisissez un certificat.

Le certificat régional doit se situer dans la même région que l'API régionale.

7. Sélectionnez Enregistrer les modifications.
8. Configurez un enregistrement DNS pour pointer le nom de domaine personnalisé régional vers ce nom d'hôte régional. Pour plus d'informations, consultez Configuring Route 53 to route traffic to API Gateway.
9. Après avoir vérifié que votre configuration DNS utilise le bon point de terminaison, vous allez supprimer la configuration du point de terminaison optimisé pour la périphérie. Choisissez votre nom de domaine personnalisé et, pour Configuration du point de terminaison optimisé pour la périphérie, choisissez Supprimer.
10. Confirmez votre choix et supprimez le point de terminaison.

AWS CLI

La update-domain-namecommande suivante fait migrer un nom de domaine personnalisé optimisé pour les périphériques vers un nom de domaine personnalisé régional :

```
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.

Le résultat se présente comme 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 la périphérie. La update-domain-namecommande suivante supprime le nom de domaine personnalisé optimisé pour les périphériques :

```
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"}
        ]'
```

La procédure suivante explique comment migrer un nom de domaine personnalisé optimisé pour les périphériques qui utilise une politique de sécurité améliorée vers un nom de domaine personnalisé régional qui utilise également une stratégie de sécurité améliorée.

Console de gestion AWS

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 un nom de domaine personnalisé optimisé pour la périphérie.
4. Pour Configuration du point de terminaison, choisissez Modifier.
5. Choisissez Ajouter un point de terminaison régional.
6. Pour Certificat ACM, choisissez un certificat.

Le certificat régional doit se situer dans la même région que l'API régionale.

7. Pour Politique de sécurité, choisissez une stratégie de sécurité commençant par`SecurityPolicy_`.
8. Pour le mode d'accès aux terminaux, choisissez Basic.
9. Sélectionnez Enregistrer les modifications.
10. Configurez un enregistrement DNS pour pointer le nom de domaine personnalisé régional vers ce nom d'hôte régional. Pour plus d'informations, consultez Configuring Route 53 to route traffic to API Gateway.
11. Après avoir vérifié que votre configuration DNS utilise le bon point de terminaison, vous allez supprimer la configuration du point de terminaison optimisé pour la périphérie. Choisissez votre nom de domaine personnalisé et, pour Configuration du point de terminaison optimisé pour la périphérie, choisissez Supprimer.
12. Confirmez votre choix et supprimez le point de terminaison.

AWS CLI

La update-domain-namecommande suivante fait migrer un nom de domaine personnalisé optimisé pour les périphériques vers un nom de domaine personnalisé régional :

```
aws apigateway update-domain-name \
    --domain-name 'api.example.com' \
    --patch-operations  '[
        { "op":"add", "path": "/endpointConfiguration/types","value": "REGIONAL" },
        { "op":"replace", "path": "/securityPolicy",
 "value":"SecurityPolicy_TLS13_1_3_2025_09"},
        { "op":"add", "path": "/regionalCertificateArn", "value": "arn:aws:acm:us-
west-2:123456789012:certificate/cd833b28-58d2-407e-83e9-dce3fd852149" }
      ]'
```

Le certificat régional doit se situer dans la même région que l'API régionale.

Le résultat se présente comme 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"
        ]
    },
    "securityPolicy": "SecurityPolicy_TLS13_1_3_2025_09",
    "endpointAccessMode": "BASIC",
    "regionalCertificateArn": "arn:aws:acm:us-west-2:123456789012:certificate/
cd833b28-58d2-407e-83e9-dce3fd852149",
    "regionalDomainName": "d-fdisjghyn6.execute-api.us-west-2.amazonaws.com"
}
```

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 la périphérie. La update-domain-namecommande suivante supprime le nom de domaine personnalisé optimisé pour les périphériques :

```
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"}
        ]'
```

La procédure suivante montre comment migrer un nom de domaine personnalisé régional vers un nom de domaine personnalisé optimisé pour la périphérie.

Console de gestion AWS

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

2. Dans le panneau de navigation, sélectionnez Noms de domaine personnalisés.
3. Choisissez un nom de domaine personnalisé régional.
4. Pour Configuration du point de terminaison, choisissez Modifier.
5. Choisissez Ajouter un point de terminaison optimisé pour les périphéries.
6. Pour Certificat ACM, choisissez un certificat.

   Le certificat du domaine optimisé pour la périphérie doit être créé dans la région `us-east-1`.
7. Choisissez Enregistrer.
8. Configurez un enregistrement DNS pour pointer le nom de domaine personnalisé optimisé pour la périphérie vers ce nom d'hôte optimisé pour la périphérie. Pour plus d'informations, consultez Configuring Route 53 to route traffic to API Gateway.
9. Après avoir vérifié que votre configuration DNS utilise le bon point de terminaison, vous allez supprimer la configuration du point de terminaison régional. Choisissez votre nom de domaine personnalisé, puis pour Configuration du point de terminaison régional, choisissez Supprimer.
10. Confirmez votre choix et supprimez le point de terminaison.

AWS CLI

La update-domain-namecommande suivante fait migrer votre nom de domaine personnalisé régional vers un nom de domaine personnalisé optimisé pour les périphériques :

```
aws apigateway update-domain-name \
    --domain-name 'api.example.com' \
    --patch-operations  '[
        { "op":"add", "path": "/endpointConfiguration/types","value": "EDGE" },
        { "op":"add", "path": "/certificateName", "value": "edge-cert" },
 {"op":"add", "path": "/certificateArn", "value": "arn:aws:acm:us-
east-1:738575810317:certificate/34a95aa1-77fa-427c-aa07-3a88bd9f3c0a"}
      ]'
```

Le certificat du domaine optimisé pour la périphérie doit être créé dans la région `us-east-1`.

Le résultat se présente comme 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 la périphérie vers ce nom de domaine de distribution. Cela permet au trafic lié au nom de domaine personnalisé optimisé pour la périphérie d'être acheminé vers le nom d'hôte d'API optimisé pour la périphérie.

Une fois l'enregistrement DNS défini, vous pouvez supprimer le type de point de terminaison `REGION` du nom de domaine personnalisé. La update-domain-namecommande suivante supprime le type de point de terminaison régional :

```
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 se présente comme 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"
    }
}
```

Envoyez du trafic vers vous APIs via votre nom de domaine personnalisé dans API Gateway

Lorsque vous configurez le mode de routage pour votre nom de domaine personnalisé, vous définissez la manière dont le trafic entrant est dirigé vers votre APIs. Vous envoyez du trafic à votre adresse à APIs l'aide de règles de routage, de mappages d'API ou de règles de routage et de mappages d'API. La section suivante explique quand utiliser des règles de routage et des mappages d'API, et comment définir le mode de routage de votre nom de domaine personnalisé.

Quand utiliser des règles de routage

Lorsque vous utilisez des règles de routage, vous dirigez les demandes entrantes répondant à certaines conditions vers des APIs étapes REST spécifiques. Par exemple, une règle peut acheminer une demande vers l'étape `production` de votre API REST `users` si elle contient l'en-tête `version:v1` et le chemin de base `/users`. Utilisez les règles de routage pour créer des topologies de routage dynamiques avancées qui prennent en charge des cas d'utilisation tels que le A/B test ou l'augmentation de l'utilisation des nouvelles versions de votre APIs.

Lorsque vous acheminez le trafic vers une API REST, nous vous recommandons d'utiliser des règles de routage pour votre nom de domaine personnalisé. Vous pouvez recréer n'importe quel mappage d'API à l'aide de règles de routage. Pour de plus amples informations, veuillez consulter the section called "Recréation d'un mappage d'API à l'aide de règles de routage".

Pour REST APIs, vous pouvez également utiliser conjointement des règles de routage et des mappages d'API. Le cas échéant, API Gateway évalue toujours les règles de routage avant les mappages d'API. Utilisez conjointement des règles de routage et des mappages d'API pour migrer vos noms de domaine personnalisés actuels ou pour explorer les règles de routage.

Considérations relatives aux règles de routage

Les considérations suivantes peuvent avoir un impact sur votre utilisation des règles de routage :

- WebSocket ou le protocole HTTP APIs ne sont pas pris en charge comme cible APIs pour les règles de routage.
- Si votre nom de domaine personnalisé comporte des mappages d'API à la fois vers REST et HTTP APIs, les règles de routage ne sont pas prises en charge.
- Pour un domaine personnalisé privé, vous pouvez créer une règle de routage vers une API REST privée. Pour un domaine public personnalisé, vous pouvez créer une règle de routage vers une API régionale ou optimisée pour la périphérie.
- Pour un domaine personnalisé privé, vous ne pouvez pas créer de règle de routage vers une API privée. Pour un nom de domaine personnalisé privé, vous ne pouvez pas créer de règle de routage vers une API publique.

Choix entre les règles de routage et les mappages d’API

Nous vous recommandons d’utiliser les règles de routage dans la mesure du possible. Utilisez uniquement les mappages d'API pour envoyer du trafic vers un HTTP ou une WebSocket API.

Définition du mode de routage de votre nom de domaine personnalisé

Vous pouvez choisir le mode de routage qu'API Gateway utilise pour acheminer le trafic vers votre APIs. Pour de plus amples informations, veuillez consulter the section called “Envoyez du trafic vers vous APIs via votre nom de domaine personnalisé dans API Gateway”. Cette section décrit les modes de routage des noms de domaine personnalisés. Vous devez définir un mode de routage pour votre nom de domaine personnalisé afin d'acheminer le trafic vers votre APIs. Les modèles de routage pris en charge sont les suivants :

- ROUTING_RULE_THEN_ API_MAPPING — Utilisez ce mode pour envoyer du trafic vers vous APIs avec des règles de routage et des mappages d'API. Dans ce mode, toutes les règles de routage ont la priorité sur les mappages d’API. Pour voir un exemple de ce mode, consultez the section called “Exemple 2 : règles de routage et mappages d’API”.
- ROUTING_RULE_ONLY — Utilisez ce mode pour autoriser uniquement les règles de routage à envoyer du trafic vers votre. APIs Lorsque votre nom de domaine personnalisé utilise ce mode, vous ne pouvez pas créer de mappage d'API, mais vous pouvez utiliser la get-api-mappingscommande pour les afficher. Les appelants de l’API ne peuvent pas utiliser de mappages d’API pour accéder à ce nom de domaine.
- API_MAPPING_ONLY — Utilisez ce mode pour autoriser uniquement les mappages d'API à envoyer du trafic vers votre. APIs Si votre nom de domaine personnalisé utilise ce mode, vous ne pouvez pas créer de règles de routage, mais vous pouvez utiliser la commande `list-routing-`

`rules` pour les afficher. Les appelants de l'API ne peuvent pas utiliser de règles de routage pour accéder à ce nom de domaine.

Il s'agit du mode de routage par défaut pour tous vos noms de domaine existants et pour tous les nouveaux noms de domaine que vous créez.

Lorsque vous créez un nom de domaine personnalisé avec `apigateway`, `API_MAPPING_ONLY` est appelé `BASE_PATH_MAPPING_ONLY` et `ROUTING_RULE_THEN_API_MAPPING` est appelé `ROUTING_RULE_THEN_BASE_PATH_MAPPING`. Ce comportement n'est présent que pour AWS CLI CloudFormation, ou aucun SDKs, pas dans le Console de gestion AWS.

La procédure suivante montre comment modifier le mode de routage d'un nom de domaine personnalisé existant. Lorsque vous modifiez le mode de routage de votre nom de domaine personnalisé, les appelants de l'API ne peuvent pas accéder à votre nom de domaine en utilisant des modes de routage non pris en charge.

Console de gestion AWS

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 un nom de domaine personnalisé.
4. Sous Détails du domaine, choisissez Modifier.
5. Pour le mode de routage, choisissez API_MAPPINGROUTING_RULE_THEN_.
6. Choisissez Enregistrer.

Si vous remplacez le mode de routage par `ROUTING_RULE_ONLY` ou `API_MAPPING_ONLY`, les mappages d'API ou les règles de routage que vous avez créés sont supprimés de la page Informations de nom de domaine de la console. Si vous modifiez le mode de routage pour prendre en charge soit des règles de routage soit des mappages d'API, ces ressources seront renvoyées.

AWS CLI - apigatewayv2

La update-domain-namecommande suivante met à jour un nom de domaine pour utiliser le mode de routage `ROUTING_RULE_THEN_API_MAPPING` :

```
aws apigatewayv2 update-domain-name \
  --domain-name 'api.example.com' \
```

```
  --routing-mode "ROUTING_RULE_THEN_API_MAPPING"
```

Le résultat se présente comme suit :

```
{
"ApiMappingSelectionExpression": "$request.basepath",
"DomainName": "api.example.com",
"DomainNameArn": "arn:aws:apigateway:us-west-2::/domainnames/api.example.com",
"DomainNameConfigurations": [
  {
      "ApiGatewayDomainName": "d-abcdefg.execute-api.us-west-2.amazonaws.com",
      "CertificateArn": "arn:aws:acm:us-west-2:111122223333:certificate/
abcdefg-123456-abcdefg",
      "DomainNameStatus": "AVAILABLE",
      "EndpointType": "REGIONAL",
      "HostedZoneId": "Z2OJLYMUO9EFXC",
      "SecurityPolicy": "TLS_1_2"
   }
 ],
"RoutingMode": "ROUTING_RULE_THEN_API_MAPPING",
"Tags": {}
}
```

AWS CLI - apigateway

La update-domain-namecommande suivante met à jour un nom de domaine personnalisé privé pour utiliser le mode de routage ROUTING_RULE_THEN_BASE_PATH_MAPPING :

```
aws apigateway update-domain-name \
  --domain-name 'private.example.com' \
  --patch-operations "op='replace',path='/
routingMode',value='ROUTING_RULE_THEN_BASE_PATH_MAPPING'"
```

Le résultat se présente comme suit :

```
{
"domainName": "private.example.com",
"domainNameId": "abcd1234",
"domainNameArn": "arn:aws:apigateway:us-west-2:111122223333:/domainnames/
private.example.com+abcd1234",
"certificateArn": "arn:aws:acm:us-west-2:111122223333:certificate/
a1b2c3d4-5678-90ab-cdef",
```

```
"certificateUploadDate": "2024-09-10T10:31:20-07:00",
"endpointConfiguration": {
  "types": [
    "PRIVATE"
   ],
  "ipAddressType": "dualstack"
  },
"domainNameStatus": "AVAILABLE",
"securityPolicy": "TLS_1_2",
"policy": "...",
"routingMode" : "ROUTING_RULE_THEN_BASE_PATH_MAPPING"
}
```

Règles de routage pour connecter des étapes d'API REST à un nom de domaine personnalisé

Une règle de routage est un ensemble de conditions qui invoquent une action lorsqu'elles sont réunies. Par exemple, une règle peut acheminer une demande entrante d'un nom de domaine personnalisé qui contient l'en-tête `Hello:World` et le chemin de base `users` vers l'étape `production` d'une API REST.

Les règles sont évaluées par ordre de priorité. Si vous définissez le mode de routage sur `ROUTING_RULE_THEN_API_MAPPING`, API Gateway évalue toujours toutes les règles de routage avant de passer aux mappages d'API. La liste suivante décrit les conditions, les actions et les priorités d'une règle de routage.

Conditions

Lorsque les conditions d'une règle sont satisfaites, ses actions sont effectuées. API Gateway prend en charge jusqu'à deux conditions d'en-tête et une condition de chemin. API Gateway évalue conjointement les conditions d'en-tête et les conditions de chemin de base.

Vous pouvez créer une règle sans condition. Lorsqu'API Gateway évalue cette règle, l'action est toujours exécutée. Vous pouvez créer une règle sans condition comme règle fourre-tout.

Pour plus d'informations sur les conditions d'en-tête, consultez the section called "Correspondance des conditions d'en-tête". Pour plus d'informations sur les conditions de chemin, consultez the section called "Correspondance des conditions de chemin de base".

Actions

Les actions sont le résultat de la correspondance entre les conditions et la règle de routage. Actuellement, la seule action prise en charge consiste à invoquer une étape d'une API REST.

Chaque règle peut comporter une action.

Priority

La priorité détermine l'ordre d'évaluation des règles, de la valeur la plus basse à la valeur la plus élevée. Les règles ne peuvent pas avoir la même priorité.

Vous pouvez définir la priorité entre 1 et 1 000 000. Si une règle a une priorité de 1, API Gateway l'évalue en premier. Lorsque vous créez une règle, nous vous recommandons d'ajouter des espaces entre les priorités. Vous pourrez ainsi modifier la priorité des règles et en ajouter de nouvelles. Pour de plus amples informations, veuillez consulter the section called "Modification de la priorité d'une règle de routage".

Pour voir des exemples d'évaluation des règles de routage par API Gateway, consultez the section called "Exemples d'évaluation de règles de routage par API Gateway".

Types de conditions des règles de routage API Gateway

La section suivante décrit les types de conditions des règles de routage. API Gateway applique une règle que si toutes les conditions sont réunies.

Correspondance des conditions d'en-tête

Lorsque vous créez une condition d'en-tête, vous pouvez faire correspondre le nom de l'en-tête et la valeur globale de l'en-tête, par exemple `Hello:World`. API Gateway utilise une correspondance littérale pour valider les conditions de correspondance des en-têtes. Votre condition peut utiliser jusqu'à deux en-têtes, séparés par `AND`. Par exemple, votre condition peut être remplie si une demande entrante comprend `Hello:World` et `x-version:beta`.

La correspondance du nom de l'en-tête n'est pas sensible à la casse, mais la valeur globale de l'en-tête l'est. `Hello:World` correspond à `hello:World`, mais pas `Hello:world`.

Pour connaître la liste des valeurs d'en-tête restreintes, consultez the section called "Restrictions".

Utilisation de caractères génériques avec des conditions d'en-tête

Les caractères génériques ne peuvent être utilisés que dans la valeur globale de l'en-tête, et doivent être `*prefix-match`, `suffix-match*` ou `*contains*`. Le tableau suivant présente des exemples d'utilisation des caractères génériques pour la correspondance des conditions d'en-tête.

Conditions d'en-tête	Demandes conformes à la règle de routage	Demandes non conformes à la règle de routage
`x-version: a*`	• `x-version: account` • `x-version: alpha`	• `x-version: backup` • `x-version: beta` • `x-version: users`
`x-version: *a`	• `x-version: alpha` • `x-version: beta`	• `x-version: account` • `x-version: backup` • `x-version: users`
`x-version: *a*`	• `x-version: account` • `x-version: alpha` • `x-version: backup` • `x-version: beta`	• `x-version: users`
`x-version: *a*` et `x-version: *b*`	• `x-version: backup` • `x-version: beta`	• `x-version: account` • `x-version: alpha` • `x-version: users`
`x-version: b*` et `x-version: *a`	• `x-version: beta`	• `x-version: account` • `x-version: alpha` • `x-version: backup` • `x-version: users`
`x-version: *`	• `x-version: account` • `x-version: alpha` • `x-version: backup` • `x-version: beta` • `x-version: users`	Aucune

Si vous créez des conditions pour plusieurs valeurs d'en-tête, par exemple `Accept:application/json,text/xml`, nous vous recommandons d'utiliser `*contains*` pour vos conditions d'en-tête et d'éviter de créer des conditions à l'aide de la virgule (`,`).

Étant donné qu'API Gateway fait correspondre littéralement les conditions d'en-tête, les correspondances sémantiques peuvent être acheminées différemment. Le tableau suivant montre les différence de résultat des règles de routage.

Conditions d'en-tête	Demandes conformes à la règle de routage	Demandes non conformes à la règle de routage
`Accept: *json`	• `Accept:application/json Accept:text/xml`	• `Accept:application/json,text/xml`
`Accept: *json*`	• `Accept:application/json Accept:text/xml` • `Accept:application/json,text/xml`	Aucune

Correspondance des conditions de chemin de base

Lorsque vous créez une condition de chemin de base, si la demande entrante contient le chemin que vous avez spécifié, la règle correspond. La correspondance étant sensible à la casse, le chemin `New/Users` ne correspondra pas à `new/users`.

Vous ne pouvez créer une condition de chemin de base que pour un seul chemin de base.

Pour obtenir la liste des restrictions des conditions de chemin de base, consultez the section called "Restrictions".

Suppression du chemin de base avec les conditions du chemin de base

Lorsque vous créez une condition de trajectoire de base, vous pouvez choisir de supprimer la trajectoire de base. Lorsque vous supprimez le chemin de base, API Gateway supprime le chemin de base entrant correspondant lorsqu'il invoque l'API cible. C'est le même comportement que lorsque vous utilisez un mappage d'API. Si vous ne supprimez pas le chemin de base, API Gateway transmet l'intégralité du chemin de base à l'API cible. Nous vous recommandons de supprimer le chemin de base uniquement lorsque vous recréez un mappage d'API.

Le tableau suivant présente des exemples d'évaluation de la condition de suppression du chemin de base par API Gateway.

Condition	Suppression du chemin de base	Demande entrante	Résultat
Si le chemin de base contient `PetStoreShopper/dogs`	True	`GET https://example.com/PetStoreShopper/dogs`	API Gateway appelle la méthode `GET` de la ressource `/`.
Si le chemin de base contient `PetStoreShopper/dogs` .	False	`GET https://example.com/PetStoreShopper/dogs`	API Gateway appelle la méthode `GET` de la ressource `PetStoreShopper/dogs` .
Si le chemin de base contient `PetStoreShopper`	True	`GET https://example.com/PetStoreShopper/dogs`	API Gateway appelle la méthode `GET` de la ressource `dogs`.
Si le chemin de base contient `PetStoreShopper`	False	`GET https://example.com/PetStoreShopper/dogs`	API Gateway appelle la méthode `GET` de la ressource `PetStoreShopper/dogs` .
Si le chemin de base contient `PetStoreShopper`	True	`GET https://example.com/PetStoreShopper?birds=available`	API Gateway appelle la méthode `GET` de la ressource `/` avec le paramètre de chaîne de requête `birds=available` .
Si le chemin de base contient `PetStoreShopper`	False	`GET https://example.com/PetStoreShopper?`	API Gateway appelle la méthode `GET` de la ressource `/PetStoreShopper` avec le paramètre de

Condition	Suppression du chemin de base	Demande entrante	Résultat
		`birds=available`	chaîne de requête `birds=available` .

Restrictions

- L'API cible et le nom de domaine personnalisé doivent se trouver dans le même AWS compte.
- Chaque règle peut comporter une API cible.
- Pour un nom de domaine personnalisé privé, vous ne pouvez créer une règle de routage que vers une API privée. Pour un nom de domaine personnalisé public, vous ne pouvez créer une règle de routage que vers une API publique. Vous ne pouvez pas mélanger des ressources publiques et privées.
- Si votre nom de domaine personnalisé comporte des mappages d'API à des API REST et HTTP, les règles de routage ne sont pas prises en charge.
- La priorité maximale est de 1 000 000.
- Restrictions applicables aux en-têtes :
  - Chaque condition `anyOf` ne peut contenir qu'une seule valeur d'en-tête.
  - Les seuls caractères autorisés pour les noms d'en-tête et les valeurs globales des en-têtes sont spécifiés par RFC 7230, à savoir `a-z`, `A-Z`, `0-9` et les caractères spéciaux suivants : `*?-!#$%&'.^_`|~`.
  - Vous pouvez utiliser un caractère générique dans la valeur globale de l'en-tête, mais celui-ci doit être `*prefix-match`, `suffix-match*` ou `*contains*`. Vous ne pouvez pas utiliser `*` au milieu d'une valeur globale d'en-tête.
  - Les noms d'en-tête génériques ne sont pas pris en charge.
  - Le nom de l'en-tête doit contenir moins de 40 caractères.
  - La valeur globale de l'en-tête doit être inférieure à 128 caractères.
  - La valeur globale de l'en-tête pour une correspondance infixe doit être inférieure à 40 caractères.
  - Les en-têtes suivants ne sont pas pris en charge en tant que conditions :
    - `access-control-*`
    - `apigw-*`
    - `Authorization`

  - `Connection`
  - `Content-Encoding`
  - `Content-Length`
  - `Content-Location`
  - `Forwarded`
  - `Keep-Alive`
  - `Origin`
  - `Proxy-Authenticate`
  - `Proxy-Authorization`
  - `TE`
  - `Trailers`
  - `Transfer-Encoding`
  - `Upgrade`
  - `x-amz-*`
  - `x-amzn-*`
  - `x-apigw-api-id`
  - `X-Forwarded-For`
  - `X-Forwarded-Host`
  - `X-Forwarded-Proto`
  - `x-restAPI`
  - `Via`
- Restrictions applicables aux chemins de base :
  - Le nom du chemin de base doit contenir moins de 128 caractères.
  - Le chemin de base ne doit contenir que des lettres, des chiffres et les caractères suivants : `$-_. +!*'()/`.

    Ces caractères ne sont pas pris en charge pour les expressions régulières.
  - Le chemin de base ne peut ni commencer ni se terminer par une barre oblique inversée (\).

Exemples d'évaluation de règles de routage par API Gateway

La section suivante présente quatre exemples d'évaluation de règles de routage et de mappages d'API par API Gateway.

Exemple 1 : règles de routage uniquement

Dans cet exemple, le mode de routage du nom de domaine personnalisé `https://petstore.example.com` est défini sur `ROUTING_RULE_ONLY`, et ce dernier possède les règles et priorités de routage suivantes.

ID de la règle	Priority	Conditions	Action
`abc123`	10	Si la demande contient l'en-tête `Hello:World`	API cible 1
`zzz000`	50	Si la demande contient les en-têtes `Accept:image/webp` et `Pet:Dog-*` , et si le chemin de base contient `PetStoreShopper`	API cible 2
`efg456`	100	Aucune	API cible 3

Le tableau suivant montre comment API Gateway applique les règles de routage précédentes à des exemples de demande.

Requête	API sélectionnée	Explication
`https://petstore.example.com -h "Hello:World"`	API cible 1	La demande correspond à la règle de routage `abc123`.
`https://petstore.example.com/PetStor`	API cible 1	API Gateway évalue toutes les règles de routage par

Requête	API sélectionnée	Explication
`eShopper -h "Hello:World", "Pet:Dog-Bella", "Accept:image/webp"`		ordre de priorité. La règle de routage `abc123` a la priorité la plus élevée et les conditions correspondent. API Gateway invoque donc l'API cible 1. Bien que les conditions de la demande correspondent également à la règle de routage `zzz000`, API Gateway n'évalue aucune autre règle de routage après avoir identifié une correspondance.
`https://petstore.example.com/PetStoreShopper -h "Pet:Dog-Bella", "Accept:image/webp"`	API cible 2	La demande correspond à la règle de routage `zzz000`. Il y a correspondance, car `Pet:Dog-Bella` correspond à la chaîne `Pet:Dog-*`
`https://petstore.example.com/PetStoreShopper -h "Pet:Dog-Bella"`	API cible 3	La demande ne correspond pas à la règle de routage `abc123`. La demande ne correspond pas à la règle de routage `zzz000`, car tous les en-têtes requis ne sont pas présents. La règle de priorité suivante correspond à toutes les demandes entrantes. API Gateway invoque donc l'API cible 3.

Exemple 2 : règles de routage et mappages d'API

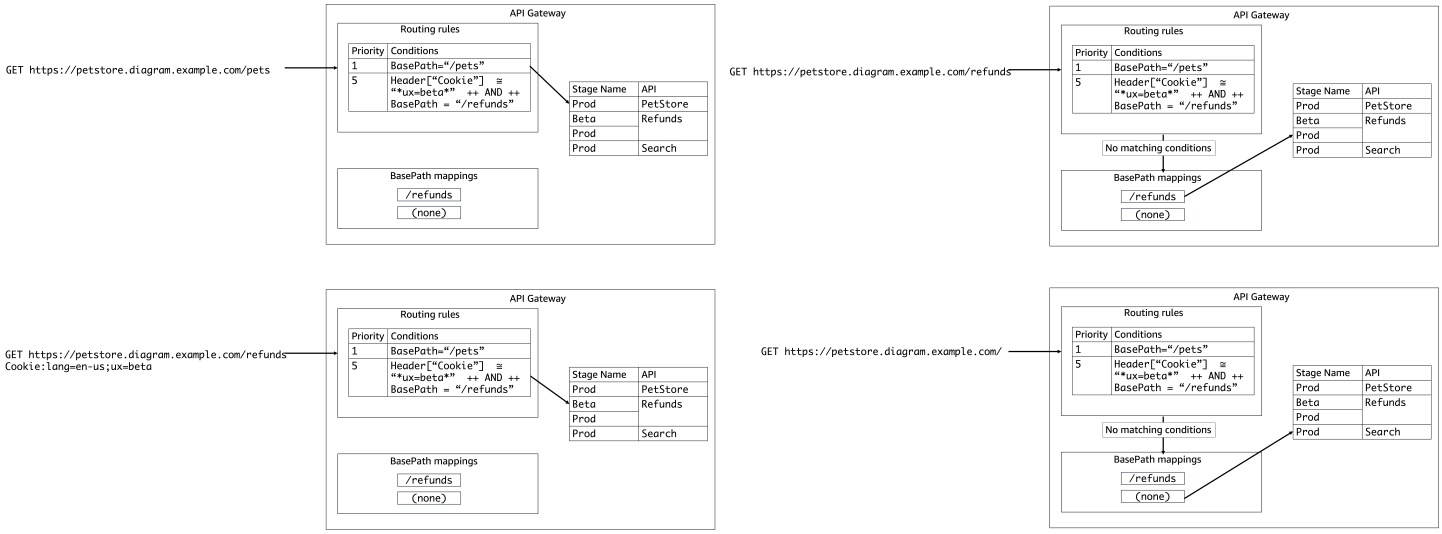
Dans cet exemple, le mode de routage du nom de domaine personnalisé `https://petstore.diagram.example.com` est défini sur `ROUTING_RULE_THEN_API_MAPPING`, et ce dernier possède les règles de routage et mappages d'API suivants.

ID de la règle	Priority	Conditions	Action
`abc123`	1	Si la demande contient `pets`	Invoquer l'étape `Prod` de l'API `PetStore`.
`000zzz`	5	Si la demande contient l'en-tête `Cookie:*ux=beta*` et si le chemin de base contient `/refunds`	Invoquer l'étape `Beta` de l'API `Refunds`.

Le tableau suivant présente les mappages d'API pour `https://petstore.backup.example.com`.

Mappage d'API	API sélectionnée
`/refunds`	Invoquer l'étape `Prod` de l'API `Refunds`.
`(none)`	Invoquer l'étape `Prod` de l'API `Search`.

Le schéma suivant montre comment API Gateway applique les règles de routage et les mappages d'API précédents à des exemples de demande. Ces exemples sont résumés dans le tableau qui suit ce schéma.



Le tableau suivant montre comment API Gateway applique les règles de routage et les mappages d'API précédents à des exemples de demande.

Requête	API sélectionnée	Explication
`https://petstore.diagram.com/pets`	Étape `Prod` de l'API `PetStore`.	La demande correspond à la règle de routage `abc123`.
`https://petstore.diagram.example.com/refunds -h "Cookie:lang=en-us;ux=beta"`	Étape `Beta` de l'API `Refunds`.	La demande correspond à la règle de routage `000zzz`. L'en-tête `Cookie` contient la bonne correspondance `*contains*` et le bon chemin de base pour cette condition.
`https://petstore.diagram.example.com/refunds`	Étape `Prod` de l'API `Refunds`.	La demande ne possède pas les en-têtes requis pour correspondre à la règle de routage `zzz000`. Si API Gateway ne parvient pas à identifier de correspondance avec la règle de routage, il passe aux mappages d'API. API Gateway peut mapper

Requête	API sélectionnée	Explication
		le chemin de base à l'étape `Prod` de l'API `Refunds`.
`https://petstore.diagram.example.com/`	Étape `Prod` de l'API `Search`.	La demande correspond au mappage de l'API vers le chemin vide (`none`).

Exemple 3 : règles de routage et mappages d'API à plusieurs niveaux

Dans cet exemple, le mode de routage du nom de domaine personnalisé `https://petstore.backup.example.com` est défini sur `ROUTING_RULE_THEN_API_MAPPING`, et ce dernier possède les règles de routage et mappages d'API suivants.

Le tableau suivant présente les règles de routage pour `https://petstore.backup.example.com`.

ID de la règle	Priority	Conditions	Action
`abc123`	10	Si la demande contient l'en-tête `Hello:World`	API cible 1
`000zzz`	50	Si la demande contient les en-têtes `Accept:image/webp` et `Pet:Dog-*`, et si le chemin de base contient `PetStoreShopper`	API cible 2

Le tableau suivant présente les mappages d'API pour `https://petstore.backup.example.com`.

Mappage d'API	API sélectionnée
`PetStoreShopper`	API cible 3
`PetStoreShopper/cats`	API cible 4

Le tableau suivant montre comment API Gateway applique les règles de routage et les mappages d'API précédents à des exemples de demande.

Requête	API sélectionnée	Explication
`https://petstore.example.com/PetStoreShopper -h "Accept:image/webp", "Pet:Cats"`	API cible 3	La demande ne possède pas les en-têtes requis pour correspondre à la règle de routage `zzz000`. Si API Gateway ne parvient pas à identifier de correspondance avec la règle de routage, il passe aux mappages d'API. API Gateway peut mapper le chemin de base à l'API cible 3.
`https://petstore.example.com/PetStoreShopper/cats -h "Hello:World"`	API cible 1	La demande correspond à la règle de routage `abc123`. Si le mode de routage est défini sur `ROUTING_RULE_THEN_API_MAPPING` , les règles de routage ont toujours la priorité sur les mappages d'API.
`https://petstore.example.com/Admin -h "Pet:Dog-Bella"`	Aucune	La demande ne correspond à aucune règle de routage ni à aucun mappage d'API. En l'absence de règle de routage par défaut, API Gateway rejette l'appel et envoie à

Requête	API sélectionnée	Explication
		l'appelant le code d'état `403 Forbidden` .

Exemple 4 : règles de routage pour les noms de domaine génériques

Dans cet exemple, le nom de domaine personnalisé `https://*.example.com` est un nom de domaine générique. Le caractère générique prend en charge tous les sous-domaines renvoyant vers le même domaine. Les exemples de règles de routage suivants modifient ce comportement afin d'autoriser les sous-domaines à acheminer le trafic vers différentes API cibles à l'aide de l'en-tête `Host`.

Le tableau suivant présente les règles de routage pour `https://*.example.com`.

ID de la règle	Priority	Conditions	Action
`abc123`	10	Si la demande contient l'en-tête `Host:a.example.com`	API cible 1
`000zzz`	50	Si la demande contient l'en-tête `Host:b.example.com`	API cible 2
`efg456`	500	Aucune	API cible 3

Le tableau suivant montre comment API Gateway applique les règles de routage précédentes à des exemples de demande.

Requête	API sélectionnée	Explication
`https://a.example.com`	API cible 1	L'en-tête `Host` est `a.example.com` . Cette

Requête	API sélectionnée	Explication
		demande correspond à la règle de routage `abc123`.
`https://b.example.com`	API cible 2	L'en-tête `Host` est `b.example.com` . Cette demande correspond à la règle de routage `000zzz`.
`https://testing.example.com`	API cible 3	Cette demande correspond à la règle de routage fourre-tout `efg456`.

Méthodes d'utilisation des règles de routage

Vous pouvez créer une règle de routage à l'aide du Console de gestion AWS SDK ou de n'importe quel autre AWS SDK. AWS CLI Après avoir créé une règle, vous pouvez modifier sa priorité.

Création d'une règle de routage

La procédure suivante explique comment créer une règle de routage pour un nom de domaine personnalisé avec un mode de routage défini sur `ROUTING_RULE_THEN_API_MAPPING` ou `ROUTING_RULE_ONLY`.

Console de gestion AWS

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 un nom de domaine personnalisé.
4. Dans l'onglet Détails du routage, choisissez Ajouter une règle de routage.
5. Choisissez Ajouter une nouvelle condition pour ajouter une nouvelle condition.

   Vous pouvez ajouter une condition d'en-tête ou de chemin de base. Pour faire correspondre toutes les demandes entrantes à votre nom de domaine personnalisé, n'ajoutez pas de condition.
6. Sous Action, utilisez le menu déroulant pour sélectionner votre API et votre étape cibles.

7. Choisissez Suivant.
8. Dans le champ Priorité, saisissez un nombre pour votre priorité.

   API Gateway évalue les règles par ordre de priorité, de la valeur la plus basse à la valeur la plus élevée.

   Si vous créez une règle sans condition, nous vous recommandons d'utiliser une priorité élevée.
9. Choisissez Créer une règle de routage.

AWS CLI

La commande create-routing-rule suivante crée une règle de routage avec une priorité de 50. Dans cet exemple, API Gateway achemine toutes les demandes entrantes contenant les en-têtes `Hello:World` et `x-version:beta`, ainsi que le chemin de base `PetStoreShopper` vers l'API cible `a1b2c3`.

```
aws apigatewayv2 create-routing-rule \
 --domain-name 'api.example.com' \
 --priority 50 \
 --conditions '[
   {
     "MatchHeaders": {
       "AnyOf": [
         {
           "Header": "Hello",
           "ValueGlob": "World"
         }
       ]
     }
   },
   {
     "MatchHeaders": {
       "AnyOf": [
         {
           "Header": "x-version",
           "ValueGlob": "beta"
         }
       ]
     }
   },
```

```
    {
      "MatchBasePaths": {
        "AnyOf": [
          "PetStoreShopper"
        ]
      }
    }
  ]'\
  --actions '[
  {
    "InvokeApi": {
      "ApiId": "a1b2c3",
      "Stage": "prod"
    }
  }
 ]'
```

Le résultat se présente comme suit :

```
{
    "Actions": [
        {
            "InvokeApi": {
                "ApiId": "a1b2c3",
                "Stage": "prod",
                "StripBasePath": false
            }
        }
    ],
    "Conditions": [
        {
            "MatchHeaders": {
                "AnyOf": [
                    {
                        "Header": "Hello",
                        "ValueGlob": "World"
                    }
                ]
            }
        },
        {
            "MatchHeaders": {
                "AnyOf": [
```

```
                    {
                        "Header": "x-version",
                        "ValueGlob": "beta"
                    }
                ]
            }
        },
        {
            "MatchBasePaths": {
                "AnyOf": [
                    "PetStoreShopper"
                ]
            }
        }
    ],
    "Priority": 50,
    "RoutingRuleArn": "arn:aws:apigateway:us-west-2:111122223333:/domainnames/
api.example.com/routingrules/abc123",
    "RoutingRuleId": "abc123"
}
```

Modification de la priorité d'une règle de routage

Vous pouvez modifier la priorité d'une règle de routage. Cette modification prend effet immédiatement et peut avoir un impact sur la manière dont les utilisateurs d'API invoquent vos noms de domaine personnalisés. Lorsque vous définissez les priorités de vos règles de routage, nous vous recommandons de laisser des espaces entre les règles.

Par exemple, imaginons que vous définissiez deux règles de routage, une règle `abc123` avec une priorité de 50 et une règle `zzz000` avec une priorité de 150. Pour modifier leur priorité des règle afin qu'API Gateway évalue la règle `zzz000` en premier, vous pouvez définir la priorité de la règle `zzz000` sur 30.

Console de gestion AWS

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 un nom de domaine personnalisé.

4. Sous l'onglet Détails du routage, choisissez votre règle de routage, puis sélectionnez Modifier.
5. Choisissez Suivant.
6. Sous Priorité, saisissez la nouvelle priorité.
7. Sélectionnez Enregistrer les modifications.

AWS CLI

La commande put-routing-rule suivante modifie la priorité d'une règle de routage `abc123`.

```
aws apigatewayv2 put-routing-rule \
 --domain-name 'api.example.com' \
 --priority 30 \
 --routing-rule-id abc123 \
 --conditions '[
   {
     "MatchHeaders": {
       "AnyOf": [
         {
           "Header": "Hello",
           "ValueGlob": "World"
         }
       ]
     }
   },
   {
     "MatchHeaders": {
       "AnyOf": [
         {
           "Header": "x-version",
           "ValueGlob": "beta"
         }
       ]
     }
   },
   {
     "MatchBasePaths": {
       "AnyOf": [
         "PetStoreShopper"
       ]
     }
```

```
    }
  ]'\
  --actions '[
  {
    "InvokeApi": {
      "ApiId": "a1b2c3",
      "Stage": "prod"
    }
  }
 ]'
```

Le résultat se présente comme suit :

```
{
    "Actions": [
        {
            "InvokeApi": {
                "ApiId": "a1b2c3",
                "Stage": "prod",
                "StripBasePath": false
            }
        }
    ],
    "Conditions": [
        {
            "MatchHeaders": {
                "AnyOf": [
                    {
                        "Header": "Hello",
                        "ValueGlob": "World"
                    }
                ]
            }
        },
        {
            "MatchHeaders": {
                "AnyOf": [
                    {
                        "Header": "x-version",
                        "ValueGlob": "beta"
                    }
                ]
            }
```

```
        },
        {
            "MatchBasePaths": {
                "AnyOf": [
                    "PetStoreShopper"
                ]
            }
        }
    ],
    "Priority": 38,
    "RoutingRuleArn": "arn:aws:apigateway:us-west-2:111122223333:/domainnames/
api.example.com/routingrules/abc123",
    "RoutingRuleId": "abc123"
}
```

Recréation d'un mappage d'API à l'aide de règles de routage

Vous pouvez recréer un mappage d'API à l'aide de règles de routage. Pour recréer un mappage d'API, assurez-vous d'activer la suppression du chemin de base. Cela préserve le comportement des mappages d'API. Pour de plus amples informations, veuillez consulter the section called "Suppression du chemin de base avec les conditions du chemin de base".

Le tutoriel suivant explique comment recréer le mappage d'API `https://` *`api.example.com`*`/orders/v2/items/categories/5` en tant que règle de routage, et comment mettre à jour vos journaux d'accès pour enregistrer l'ID de la règle de routage utilisée par API Gateway pour envoyer le trafic à votre API.

Console de gestion AWS

Pour définir le mode de routage sur ROUTING_RULE_THEN_API_MAPPING

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 votre nom de domaine personnalisé.
4. Sous Détails du domaine, choisissez Modifier.
5. Sous Mode de routage, choisissez ROUTING_RULE_THEN_API_MAPPING.
6. Choisissez Enregistrer.

Après avoir défini le mode de routage, vous allez créer la règle de routage.

Pour créer la règle de routage

1. Dans l'onglet Détails du routage, choisissez Ajouter une règle de routage.
2. Choisissez Ajouter une condition, puis Chemin.
3. Sous Chemin, saisissez **`orders/v2/items/categories/5`**.
4. Sous Chemin de base de la bande, choisissez Actif.
5. Sous API cible, choisissez votre API cible.
6. Sous Étape cible, choisissez votre étape cible.
7. Choisissez Suivant.
8. Sous Priorité, saisissez une priorité.

   Même si vous conservez votre mappage d'API, API Gateway utilisera toujours la nouvelle règle de routage, car les règles de routage ont toujours la priorité sur les mappages d'API.
9. Sélectionnez Enregistrer les modifications.

Après avoir créé la règle de routage, mettez à jour le format du journal d'accès pour votre étape, ou créez un journal afin de vérifier qu'API Gateway utilise votre règle de routage pour envoyer le trafic à votre API.

Pour mettre à jour vos journaux d'accès

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.
4. Sous Journaux et traçage, choisissez Modifier.

   Si vous n'avez pas de groupe de journaux, consultez the section called "CloudWatch journaux".
5. Ajoutez **`$context.customDomain.routingRuleIdMatched`** à votre format de journal.

   Ce groupe de journaux enregistre l'ID de la règle de routage utilisée par API Gateway pour envoyer le trafic à votre API. Pour de plus amples informations, veuillez consulter the section called "Je ne sais pas comment API Gateway a envoyé le trafic à mes API".

6. Choisissez Enregistrer.

Après avoir mis à jour vos journaux d'accès, invoquez votre nom de domaine personnalisé. Voici un exemple de commande curl permettant d'invoquer le nom de domaine personnalisé `https://api.example.com` avec le chemin de base `orders/v2/items/categories/5`.

```
curl "https://api.example.com/orders/v2/items/categories/5"
```

Après avoir invoqué avec succès votre nom de domaine personnalisé, vérifiez que CloudWatch Logs affiche le`routingRuleIdMatched`. Pour savoir comment utiliser la console CloudWatch Logs pour afficher un groupe de journaux, consultezthe section called "Afficher les événements du journal dans la CloudWatch console".

AWS CLI

1. Utilisez la commande update-domain-name suivante pour mettre à jour le nom de domaine `api.example.com` afin qu'il utilise le mode de routage `ROUTING_RULE_THEN_API_MAPPING`.

```
aws apigatewayv2 update-domain-name \
  --domain-name 'api.example.com' \
  --routing-mode ROUTING_RULE_THEN_API_MAPPING
```

2. Utilisez la commande create-routing-rule suivante pour créer une règle de routage dans le but de recréer le mappage d'API `https://api.example.com/orders/v2/items/categories/5`.

```
aws apigatewayv2 create-routing-rule \
  --domain-name 'api.example.com' \
  --priority 50 \
  --conditions '[
  {
    "MatchBasePaths": {
      "AnyOf": [
        "orders/v2/items/categories/5"
      ]
    }
  }
]' \
  --actions '[
  {
```

```
    "InvokeApi": {
      "ApiId": "a1b2c3",
      "Stage": "prod",
      "StripBasePath": true
    }
  }
]'
```

3. Utilisez la commande update-stage suivante pour mettre à jour le format des journaux d'accès afin qu'il inclut la variable `$context.customDomain.routingRuleIdMatched`. Cette variable enregistre l'ID de la règle de routage utilisée par API Gateway pour envoyer le trafic à votre API. Vous allez utilisez ce journal pour vérifier qu'API Gateway utilise votre règle de routage pour envoyer le trafic à votre API. Pour de plus amples informations, veuillez consulter the section called "Je ne sais pas comment API Gateway a envoyé le trafic à mes API".

```
aws apigateway update-stage \
  --rest-api-id a1bc2c3 \
  --stage-name prod \
  --patch-operations "op=replace,path=/accessLogSettings/format,value='\
$context.path \$context.customDomain.routingRuleIdMatched \$context.requestId \
$context.extendedRequestId'"
```

   Si vous n'avez pas de groupe de journaux, consultez the section called "CloudWatch journaux".

4. Utilisez l'exemple de commande curl suivant pour invoquer votre nom de domaine personnalisé à l'aide du chemin de base `orders/v2/items/categories/5`.

```
curl "https://api.example.com/orders/v2/items/categories/5
```

5. Utilisez la commande filter-log-events suivante pour obtenir les événements du groupe de journaux `access-log-group-orders` contenant l'ID de la règle de routage `abc123`.

```
aws logs filter-log-events --log-group-name access-log-group-orders --filter-
pattern abc123
```

   Vous avez la confirmation qu'API Gateway a utilisé la règle de routage pour envoyer le trafic à votre API.

Résolution des problèmes liés aux règles de routage

Les conseils suivants peuvent vous aider à résoudre les problèmes liés à votre règles de routage.

Je ne sais pas comment API Gateway a envoyé le trafic à mes API

Vous pouvez utiliser les journaux d'accès de l'étape de votre API REST afin d'enregistrer et de résoudre les problèmes liés à vos règles de routage. Vous pouvez consulter l'ID de la règle de routage utilisée par API Gateway pour envoyer le trafic à votre API à l'aide de la variable `$context.customDomain.routingRuleIdMatched`. Pour afficher le mappage d'API utilisé par API Gateway pour envoyer le trafic à votre API, utilisez la variable `$context.customDomain.basePathMatched`.

Pour enregistrer vos règles de routage, vous devez configurer un ARN de rôle CloudWatch Logs approprié pour votre compte et créer un groupe de journaux.

L'exemple de groupe de journaux d'accès suivant récupère les informations pertinentes pour résoudre les problèmes liés aux règles de routage et aux mappages d'API. API Gateway renseigne uniquement la variable context correspondant au mécanisme de routage utilisé. Sinon, la variable context est `-`.

CLF

```
$context.path $context.customDomain.routingRuleIdMatched
 $context.customDomain.basePathMatched $context.requestId $context.extendedRequestId
```

JSON

```
{"requestPath": "$context.path", "routingRuleId" :
 "$context.customDomain.routingRuleIdMatched", "API mapping" :
 "$context.customDomain.basePathMatched", "requestId":"$context.requestId",
 "extendedRequestId":"$context.extendedRequestId"}
```

XML

```
<request id="$context.requestId"> <requestPath>$context.path</requestPath>
 <ruleId>$context.customDomain.routingRuleIdMatched</ruleId> <ApiMapping>
$context.customDomain.basePathMatched</ApiMapping> <extendedRequestId>
$context.extendedRequestId</extendedRequestId> </request>
```

CSV

```
$context.path,$context.customDomain.routingRuleIdMatched,
$context.customDomain.basePathMatched,$context.requestId,$context.extendedRequestId
```

Nous vous recommandons également de vérifier le mode de routage de votre nom de domaine personnalisé. Pour de plus amples informations, veuillez consulter the section called “Définition du mode de routage de votre nom de domaine personnalisé”.

Je ne parviens pas à activer les règles de routage de mon nom de domaine personnalisé

API Gateway peut vous envoyer l’erreur suivante :

```
Your account doesn’t have permission to use RoutingRules.
This might be caused by an IAM policy in your account with a deny statement on
 BasePathMapping or ApiMapping.
To grant permission for this account to use RoutingRules, use the UpdateAccount API.
This will impact any existing IAM policies that deny access to BasePathMapping or
 ApiMapping.
See API Gateway documentation for further details.
```

Vous recevrez cette erreur si vous avez ou avez eu une politique IAM qui refuse l'accès à BasePathMappingou ApiMapping. Lorsque vous activez les règles de routage d’un nom de domaine personnalisé, même si votre politique continue de refuser l’accès à `BasePathMapping` ou `ApiMapping`, celle-ci peut être utilisée pour accéder à `RoutingRule`. L’utilisateur peut ainsi modifier le comportement de routage de votre nom de domaine personnalisé.

Par exemple, si vous aviez une politique comme suit :

```
{
    "Sid": "DenyCreatingApiMappings",
    "Effect": "Deny",
    "Action": "apigateway:POST",
    "Resource": [
        "arn:aws:apigateway:us-west-2::/domainnames/example.com/apimappings"
    ]
}
```

Si vous activez les règles de routage pour `example.com`, cette politique continuera à refuser l’accès à la création d’un `ApiMapping`, mais ne refusera pas l’accès à la création d’une `RoutingRule`.

Nous vous recommandons d'auditer les politiques IAM de votre compte. L'exemple de politique suivant refuse l'accès à la création d'un `ApiMapping`, d'un `BasePathMapping` et d'une `RoutingRule` :

```
{
    "Sid": "DenyCreatingBasePathMappingsApiMappings",
    "Effect": "Deny",
    "Action": "apigateway:POST",
    "Resource": [
        "arn:aws:apigateway:us-west-2::/domainnames/example.com/basepathmappings",
        "arn:aws:apigateway:us-west-2::/domainnames/example.com/apimappings"
    ]
},
{
    "Sid": "DenyCreatingRoutingRules",
    "Effect": "Deny",
    "Action": "apigateway:CreateRoutingRule",
    "Resource": [
        "arn:aws:apigateway:us-west-2:111122223333:/domainnames/example.com/
routingrules/*"
    ]
}
```

Après avoir vérifié que toutes vos politiques ont été mises à jour, vous pouvez mettre à jour les paramètres au niveau du compte de votre API afin d'activer les règles de routage pour une région.

Utilisez la commande update-account suivante pour mettre à jour les paramètres au niveau du compte de votre API pour une région :

```
aws apigateway update-account --patch-operations 'op=remove,path=/
features,value=BlockedForRoutingRules' --region us-west-2
```

Après avoir mis à jour les paramètres au niveau du compte de votre API, vous pouvez modifier le mode de routage de votre nom de domaine personnalisé. Vous pouvez également continuer à utiliser des politiques IAM pour refuser l'accès aux `RoutingRules`, au `ApiMapping` ou au `BasePathMapping`.

Utilisation de mappages d'API pour connecter des étapes d'API REST à un nom de domaine personnalisé

Les mappages d'API vous permettent de connecter des étapes d'API à un nom de domaine personnalisé. Ils envoient le trafic à 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 `https://api.example.com/orders` à l'étape `production` d'une API.

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".

Demandes entrantes de votre nom de domaine personnalisé

Lorsque vous mappez un nom de domaine personnalisé à une étape de votre API, API Gateway supprime le chemin de base entrant. Le chemin de base mappé de l'invocation de l'API est supprimé. Par exemple, si votre chemin de base `https://api.example.com/orders/shop/5` était mappé à l'étape `test` et que vous utilisiez la demande `https://api.example.com/orders/shop/5/hats`, API Gateway invoquerait la ressource `/hats` de l'étape `test` de votre API, et non la ressource `orders/shop/5/hats`.

Mappage des demandes d'API

L'exemple suivant explique comment API Gateway évalue les mappages d'API.

Vous pouvez créer un mappage d'API à l'aide de mappages à un niveau, par exemple un mappage d'API entre `orders` et l'étape `beta` d'une API, et un mappage d'API entre `shipping` et l'étape `alpha` d'une API. Pour les noms de domaine personnalisés régionaux avec une politique de sécurité TLS 1.2, API Gateway prend en charge les mappages d'API à plusieurs niveaux. Vous pouvez créer un mappage d'API entre `orders/v1/items` et l'étape `alpha` d'une API, et entre `orders/v2/items` et l'étape `beta` d'une API. Lorsque vous créez un mappage à plusieurs niveaux, API Gateway achemine les demandes vers le mappage d'API au chemin de correspondance le plus long.

Vous pouvez créer un mappage d'API vers le chemin vide (`none`). Si aucun chemin ne correspond à la demande, API Gateway envoie celle-ci au chemin vide (`none`).

Dans cet exemple, le nom de domaine personnalisé `https://api.example.com` possède les mappages d'API suivants.

Mappage d'API	API sélectionnée
`(none)`	API 1
`orders`	API 2
`orders/v1/items`	API 3
`orders/v2/items`	API 4
`orders/v1/items/categories`	API 5

Le tableau suivant montre comment API Gateway applique les mappages d'API précédents à des exemples de demande.

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. Consultez the

Requête	API sélectionnée	Explication
		section called “Demandes entrantes de votre nom de domaine personnalisé”.
`https://api.example.com/orders/v1/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 au préfixe de correspondance 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 figurer 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. Cette limite n'inclut pas les mappages d'API à un niveau tels que `/prod`.
- Vous ne pouvez mapper les API HTTP à un nom de domaine personnalisé régional qu'à l'aide de la politique de sécurité TLS 1.2.
- Vous ne pouvez pas mapper les WebSocket API au même nom de domaine personnalisé qu'une API HTTP ou une API REST.
- Après avoir créé vos mappages d'API, vous devez créer ou mettre à jour l'enregistrement de ressource de votre fournisseur DNS pour le mapper au point de terminaison de votre API.
- Si vous créez un mappage d'API à plusieurs niveaux, API Gateway convertit tous les noms d'en-tête en minuscules.

Création d'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. Le mode de routage de votre nom de domaine personnalisé doit être défini sur `ROUTING_RULE_THEN_API_MAPPING` ou `API_MAPPING_ONLY`. Pour plus d'informations sur la configuration du mode de routage, consultez the section called "Définition du mode de routage de votre nom de domaine personnalisé".

Pour des exemples AWS Serverless Application Model de modèles qui créent toutes les ressources, voir Sessions avec SAM activé GitHub.

Console de gestion AWS

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 un nom de domaine personnalisé.
4. Sous l'onglet Détails du routage, choisissez Configuration des mappages d'API.
5. Saisissez l'API, l'étape et le chemin du mappage.
6. Cliquez sur Enregistrer.

AWS CLI

La commande create-api-mapping 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
```

CloudFormation

L' 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
```

Types d'adresses IP pour les noms de domaine personnalisés dans API Gateway

Lorsque vous créez un nom de domaine personnalisé, vous spécifiez le type d'adresses IP qui peuvent appeler votre domaine. Vous pouvez choisir IPv4 de résoudre IPv4 les adresses pour

appeler votre domaine, ou vous pouvez choisir DualStack pour autoriser IPv4 les deux IPv6 adresses à appeler votre domaine. Nous vous recommandons de définir le type d'adresse IP sur dualstack pour éviter l'épuisement de l'espace IP ou pour améliorer votre niveau de sécurité. Pour plus d'informations sur les avantages d'un type d'adresse IP à double pile, reportez-vous IPv6 à la section suivante. AWS

Vous pouvez modifier le type d'adresse IP en mettant à jour la configuration du point de terminaison de votre nom de domaine.

Considérations relatives aux types d'adresses IP

Les considérations suivantes peuvent avoir un impact sur votre utilisation des types d'adresses IP.

- Le type d'adresse IP par défaut pour les noms de domaine personnalisés API Gateway destinés au public APIs est IPv4.
- Les noms de domaine personnalisés privés ne peuvent avoir qu'un type d'adresse IP à double pile.
- Il n'est pas nécessaire que le même type d'adresse IP soit associé à votre nom de domaine personnalisé pour tous APIs . Si vous désactivez votre point de terminaison d'API par défaut, cela peut affecter la manière dont les appelants peuvent appeler votre domaine.

Modifier le type d'adresse IP d'un nom de domaine personnalisé

Vous pouvez modifier le type d'adresse IP en mettant à jour la configuration du point de terminaison du nom de domaine. Vous pouvez mettre à jour la configuration du point de terminaison à l'aide du Console de gestion AWS AWS CLI, du CloudFormation, ou d'un AWS SDK.

Console de gestion AWS

Pour modifier le type d'adresse IP d'un nom de domaine personnalisé

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Choisissez un nom de domaine public personnalisé.
3. Choisissez Configuration du point de terminaison.
4. Pour le type d'adresse IP, sélectionnez Dualstack IPv4ou Dualstack.
5. Choisissez Enregistrer.

AWS CLI

La update-domain-namecommande suivante met à jour une API pour qu'elle ait un type d'adresse IP dualstack :

```
aws apigateway update-domain-name \
    --domain-name dualstack.example.com \
    --patch-operations "op='replace',path='/endpointConfiguration/
ipAddressType',value='dualstack'"
```

Le résultat se présente comme suit :

```
{
    "domainName": "dualstack.example.com",
    "certificateUploadDate": "2025-02-04T14:46:10-08:00",
    "regionalDomainName": "d-abcd1234.execute-api.us-east-1.amazonaws.com",
    "regionalHostedZoneId": "Z3LQWSYCGH4ADY",
    "regionalCertificateArn": "arn:aws:acm:us-east-1:111122223333:certificate/
a1b2c3d4-5678-90ab-cdef",
    "endpointConfiguration": {
        "types": [
            "REGIONAL"
        ],
        "ipAddressType": "dualstack"
    },
    "domainNameStatus": "AVAILABLE",
    "securityPolicy": "TLS_1_2",
    "tags": {}
}
```

Choisissez une politique de sécurité pour votre domaine personnalisé dans API Gateway

Une politique de sécurité est une combinaison prédéfinie d'une version minimale du protocole TLS et de suites de chiffrement offertes par API Gateway. Lorsque vos clients établissent une liaison TLS vers votre API ou nom de domaine personnalisé, la politique de sécurité applique la version TLS et la suite de chiffrement acceptées par API Gateway. Les politiques de sécurité protègent vos API et vos noms de domaine personnalisés contre les problèmes de sécurité réseau tels que la falsification et l'écoute clandestine entre un client et un serveur.

API Gateway prend en charge les politiques de sécurité existantes et les politiques de sécurité améliorées. `TLS_1_0`et `TLS_1_2` sont des politiques de sécurité héritées. Utilisez ces politiques de sécurité pour les charges de travail généralisées ou pour commencer à créer une API. Toute politique qui commence par `SecurityPolicy_` est une politique de sécurité renforcée. Utilisez-les pour les charges de travail régulées, la gouvernance avancée ou pour utiliser la cryptographie post-quantique. Lorsque vous utilisez une politique de sécurité améliorée, vous devez également définir le mode d'accès aux points de terminaison pour une meilleure gouvernance. Pour de plus amples informations, veuillez consulter the section called "Mode d'accès au point de terminaison".

Considérations

Voici les considérations relatives aux politiques de sécurité relatives aux noms de domaine personnalisés pour les API REST dans API Gateway :

- Vous ne pouvez pas activer le protocole TLS mutuel sur un nom de domaine qui utilise une politique de sécurité renforcée.
- Vous ne pouvez pas associer une API HTTP à un nom de domaine qui utilise une politique de sécurité renforcée.
- Si vous activez le mappage de chemins de base à plusieurs niveaux vers une API REST qui utilise une politique de sécurité renforcée, vous ne pouvez pas créer de mappage de chemin de base vers une API HTTP pour le même nom de domaine.
- Votre API peut être mappée à un nom de domaine personnalisé avec une politique de sécurité différente de celle de votre API. Lorsque vous invoquez ce nom de domaine personnalisé, API Gateway utilise la politique de sécurité du domaine personnalisé pour négocier le handshake TLS. La désactivation du point de terminaison de votre API par défaut peut avoir une incidence sur la manière dont les appelants peuvent invoquer votre API.
- API Gateway prend en charge les politiques de sécurité sur toutes les API. Toutefois, vous ne pouvez choisir une politique de sécurité que pour les API REST. API Gateway prend uniquement en charge la politique `TLS_1_2` de sécurité pour le HTTP ou WebSocket les API.
- API Gateway ne prend pas en charge la mise à jour d'une politique de sécurité pour un nom de domaine comportant plusieurs types de points de terminaison. Si vous avez plusieurs types de point de terminaison pour un nom de domaine, supprimez l'un d'entre eux pour mettre à jour la politique de sécurité.

Comment API Gateway applique les politiques de sécurité

L'exemple suivant montre comment API Gateway applique les politiques de sécurité en utilisant la politique de `SecurityPolicy_TLS13_1_3_2025_09` sécurité comme exemple.

La politique `SecurityPolicy_TLS13_1_3_2025_09` de sécurité accepte le trafic TLS 1.3 et rejette le trafic TLS 1.2 et TLS 1.0. Pour le trafic TLS 1.3, la politique de sécurité accepte les suites de chiffrement suivantes :

- `TLS_AES_128_GCM_SHA256`
- `TLS_AES_256_GCM_SHA384`
- `TLS_CHACHA20_POLY1305_SHA256`

API Gateway n'accepte aucune autre suite de chiffrement. Par exemple, la politique de sécurité rejetterait tout trafic TLS 1.3 utilisant la suite de `AES128-SHA` chiffrement.

Pour contrôler le protocole TLS et les chiffrements utilisés par les clients pour accéder à votre API Gateway, vous pouvez utiliser les variables de `$context.cipherSuite` contexte `$context.tlsVersion` et dans vos journaux d'accès. Pour de plus amples informations, veuillez consulter the section called “Surveillance”.

Pour connaître les politiques de sécurité par défaut pour toutes les API REST et les noms de domaine personnalisés, voirthe section called “Politiques de sécurité par défaut”. Pour connaître les politiques de sécurité prises en charge pour toutes les API REST et les noms de domaine personnalisés, consultezthe section called “Politiques de sécurité prises en charge”.

Modifier la politique de sécurité de votre nom de domaine personnalisé

Si vous modifiez votre politique de sécurité, la mise à jour prend environ 15 minutes. Vous pouvez surveiller votre nom `lastUpdateStatus` de domaine personnalisé. Au fur et à mesure que votre nom de domaine personnalisé `lastUpdateStatus` est mis à jour, il est `PENDING` et une fois terminé, il le sera`AVAILABLE`.

Lorsque vous utilisez une politique de sécurité commençant par`SecurityPolicy_`, vous devez également activer le mode d'accès aux terminaux. Pour de plus amples informations, veuillez consulter the section called “Mode d’accès au point de terminaison”.

Console de gestion AWS

Pour modifier la politique de sécurité d'un nom de domaine personnalisé

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Choisissez un nom de domaine personnalisé qui envoie le trafic vers les API REST.

   Assurez-vous qu'un seul type de point de terminaison est associé à votre nom de domaine personnalisé.
3. Choisissez Paramètres de nom de domaine personnalisés, puis sélectionnez Modifier.
4. Pour Politique de sécurité, sélectionnez une nouvelle stratégie.
5. Pour le mode d'accès aux terminaux, choisissez Strict.
6. Sélectionnez Enregistrer les modifications.

AWS CLI

La commande update-domain-name suivante met à jour un nom de domaine afin d'utiliser la politique de sécurité : `SecurityPolicy_TLS13_1_3_2025_09`

```
aws apigateway update-domain-name \
    --domain-name example.com \
    --patch-operations '[
        {
            "op": "replace",
            "path": "/securityPolicy",
            "value": "SecurityPolicy_TLS13_1_3_2025_09"
        },
        {
            "op": "replace",
            "path": "/endpointAccessMode",
            "value": "STRICT"
        }
    ]'
```

Le résultat se présente comme suit :

```
{
   "domainName": "example.com",
```

```
   "endpointConfiguration": {
      "types": [ "REGIONAL" ],
      "ipAddressType": "dualstack"
   },
   "regionalCertificateArn": "arn:aws:acm:us-west-2:111122223333:certificate/
a1b2c3d4-5678-90ab-cdef",
   "securityPolicy": "SecurityPolicy_TLS13_1_3_2025_09",
   "endpointAccessMode": "STRICT"
}
```

Informations sur les API HTTP et les WebSocket API

Pour plus d'informations sur les API et WebSocket API HTTP, consultez the section called “Politique de sécurité pour le protocole HTTP APIs dans API Gateway” etthe section called “Politique de sécurité pour WebSocket APIs in API Gateway”.

Désactivation du point de terminaison par défaut pour les 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 de statut `403 Forbidden`. La désactivation du point de terminaison par défaut affecte toutes les étapes de l’API. Ce paramètre prend effet lorsque vous mettez à jour un paramètre sur une étape, par exemple lorsque vous mettez à jour le déploiement sur la phase.

La procédure suivante explique comment désactiver le point de terminaison par défaut pour une API REST.

Console de gestion AWS

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 Paramètres de l’API.
4. Choisissez une API.
5. Dans l’onglet Détails de l’API, choisissez Modifier.

6. Pour Point de terminaison par défaut, sélectionnez Inactif.
7. Cliquez sur Enregistrer les modifications.
8. Dans le volet de navigation principal, choisissez Ressources.
9. Sélectionnez Deploy API (Déployer une API).
10. Redéployez votre API sur une étape ou mettez à jour un paramètre d'une étape pour que la mise à jour prenne effet.

AWS CLI

La commande update-rest-api suivante désactive le point de terminaison par défaut :

```
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 commande create-deployment suivante crée un déploiement et l'associe à une étape :

```
aws apigateway create-deployment \
    --rest-api-id abcdef123 \
    --stage-name dev
```

Configuration de surveillances de l'état personnalisées pour le basculement DNS d'une API API Gateway

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, votre propre API Gateway APIs peut être interrompue

avant cette date. Pour contrôler directement le basculement du DNS, configurez des contrôles de santé Route 53 personnalisés pour votre API Gateway APIs. 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 : tester le basculement
- Étape 4 : retourner 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 “Prérequis concernant les noms de domaine personnalisés”.
- 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 APIs en deux Régions AWS
- Noms de domaine personnalisés API Gateway portant le même nom sur deux Régions AWS
- Mappages d'API Gateway qui connectent votre API Gateway APIs 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 CloudFormation de modèles qui créent ces ressources, voir `primary.yaml`et `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 : tester le 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 : retourner 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.

Optimisez les performances de REST APIs

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 politiques 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

- Paramètres du cache des API REST dans API Gateway
- Compression de charge utile pour REST APIs dans API Gateway

Paramètres du cache des API REST dans API Gateway

Vous pouvez activer la mise en cache des API dans 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 la période de durée de vie (TTL) spécifiée, 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, U.S. Health Insurance Portability and Accountability Act de 1996 (HIPAA), et utilisation de AWS services

pour traiter, stocker et transmettre des informations de santé protégées (PHI), voir la vue d'ensemble 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 en charge, consultez la section ClusterSize relative au cache dans 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 ?).

Console de gestion AWS

Dans la console API Gateway, la mise en cache est configurée à la page Étapes. Vous allez configurer le cache de l'étape et spécifier un paramètre de cache au niveau de la méthode par défaut. Si vous activez le cache au niveau de la méthode par défaut, la mise en cache au niveau de la méthode est activée pour toutes les méthodes `GET` de votre étape, à moins que cette méthode comporte un remplacement de méthode. Toutes les méthodes `GET` supplémentaires que vous déployez pour votre étape disposeront d'un cache au niveau de la méthode. Pour configurer le paramètre de mise en cache au niveau de la méthode pour des méthodes spécifiques de votre étape, vous pouvez utiliser des remplacements de méthode. Pour plus d'informations sur les remplacements de méthode, consultez the 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 :

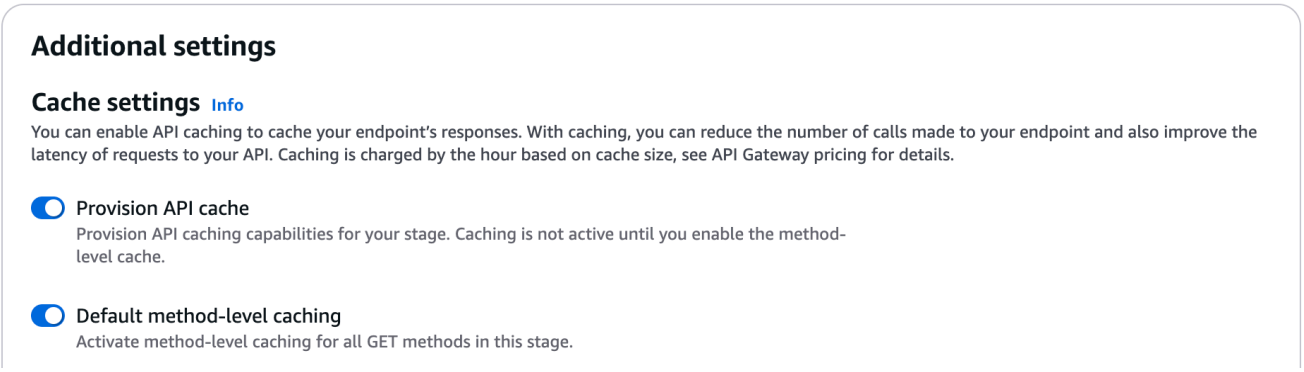
1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Choisissez Stages (Étapes).
3. Dans la liste Stages (Étapes) de l'API, choisissez l'étape.
4. Dans la section Détails de l'étape, choisissez Modifier.
5. Sous Paramètres supplémentaires, pour Paramètres de cache, activez Configurer le cache d'API.

   Cette opération configure un cluster de cache pour votre étape.

6. Pour activer la mise en cache pour votre étape, activez Mise en cache au niveau de la méthode par défaut.

   La mise en cache au niveau de la méthode est alors activée pour toutes les méthodes `GET` de votre étape. Toutes les méthodes `GET` supplémentaires que vous déployez pour cette étape disposeront d'un cache au niveau de la méthode.

   > **Note**
   >
   > S'il existe déjà un paramètre de cache au niveau de la méthode, celui-ci ne sera pas affecté par la modification du paramètre de cache au niveau de la méthode par défaut.



7. Sélectionnez Enregistrer les modifications.

AWS CLI

La commande update-stage suivante met à jour une étape afin de configurer un cache, et active la mise en cache au niveau de la méthode de toutes les méthodes `GET` de votre étape :

```
aws apigateway update-stage \
    --rest-api-id a1b2c3 \
    --stage-name 'prod' \
    --patch-operations file://patch.json
```

Le contenu de `patch.json` est le suivant :

```
[
     {
```

```
          "op": "replace",
          "path": "/cacheClusterEnabled",
          "value": "true"
     },
     {
          "op": "replace",
          "path": "/cacheClusterSize",
          "value": "0.5"
     },
     {
        "op": "replace",
        "path": "/*/*/caching/enabled",
        "value": "true"
     }
]
```

> **Note**
>
> S'il existe déjà un paramètre de cache au niveau de la méthode, celui-ci ne sera pas affecté par la modification du paramètre de cache au niveau de la méthode par défaut.

> **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 champ Cluster de cache passe de `Create in progress` à `Active`. Lorsqu'un cache est supprimé, la valeur du champ 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 étape, la valeur 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 étape, la valeur Mise en cache au niveau de la méthode par défaut devient `Inactive`.
> S'il existe déjà un paramètre de cache au niveau de la méthode, il ne sera pas affecté par la modification du statut du cache.

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**
>
> N'utilisez pas 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.

Remplacement de la mise en cache au niveau de l'étape 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 d'une méthode spécifique. Vous pouvez également modifier la période TTL, ou activer ou désactiver le chiffrement des réponses mises en cache.

Si vous prévoyez qu'une méthode que vous mettez en cache recevra des données sensibles dans ses réponses, chiffrez les données du cache. Vous devrez peut-être le faire pour vous conformer aux différents cadres de conformité. Pour plus d'informations, consultez Amazon API Gateway controls dans le Guide de l'utilisateur AWS Security Hub .

Console de gestion AWS

Si vous modifiez le paramètre de mise en cache au niveau de la méthode par défaut dans Détails de l'étape, les paramètres de cache au niveau de la méthode ayant des remplacements n'en seront pas affectés.

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 configuré un cluster de cache pour votre étape.

7. Choisissez Enregistrer.

AWS CLI

La commande update-stage suivante désactive le cache uniquement pour la méthode `GET /pets` :

```
aws apigateway update-stage /
    --rest-api-id a1b2c3 /
    --stage-name 'prod' /
    --patch-operations file://patch.json
```

Le contenu de `patch.json` est le suivant :

```
[{
        "op": "replace",
        "path": "/~1pets/GET/caching/enabled",
        "value": "false"
}]
```

Utilisation de paramètres de méthode ou d'intégration en tant que clés de cache pour indexer les réponses mises en cache

Vous pouvez utiliser un paramètre de méthode ou d'intégration en tant que clés de cache pour indexer les réponses mises en cache. Cela inclut les en-têtes personnalisés, les chemins d'URL ou les chaînes de requête. Vous pouvez spécifier tout ou partie de ces paramètres comme clé de cache, mais vous devez spécifier au moins une valeur. Si vous disposez d'une clé de cache, API Gateway met en cache les réponses de chaque valeur de clé séparément, y compris en l'absence de la clé de cache.

> **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 :

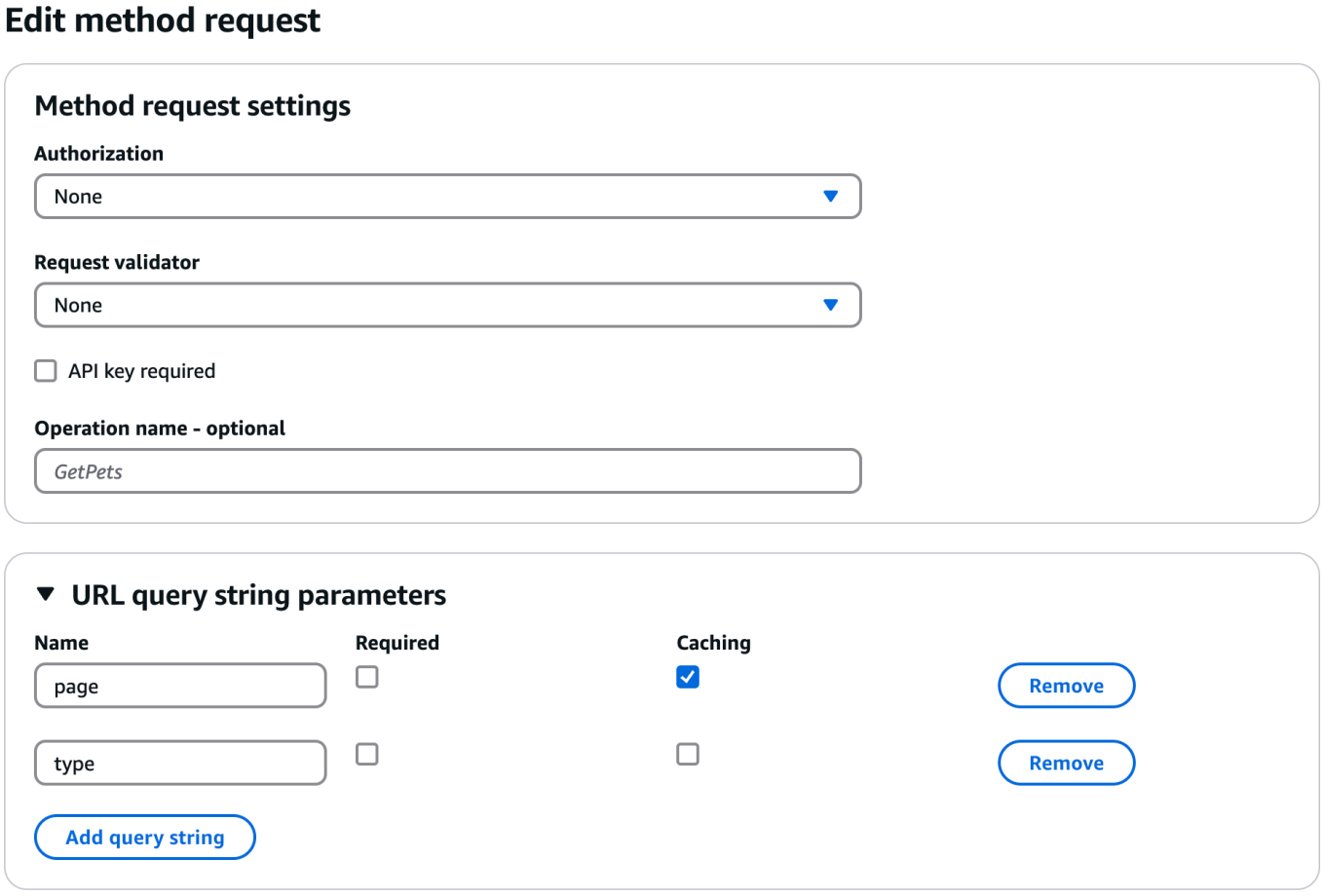
```
GET /users?type=admin&department=A HTTP/1.1
host: example.com
...
```

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
...
```

Console de gestion AWS

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.



AWS CLI

La commande put-method suivante crée une méthode GET et exige le paramètre de chaîne de requête `type` :

```
aws apigateway put-method /
    --rest-api-id a1b2c3 /
    --resource-id aaa111 /
```

```
    --http-method GET /
    --authorization-type "NONE" /
    --request-parameters "method.request.querystring.type=true"
```

La commande put-integration suivante crée une intégration pour la méthode GET avec un point de terminaison HTTP, et indique qu'API Gateway met en cache le paramètre de demande de méthode `type` :

```
aws apigateway put-integration /
    --rest-api-id a1b2c3 /
    --resource-id aaa111 /
    --http-method GET /
    --type HTTP /
    --integration-http-method GET /
    --uri 'https://example.com' /
    --cache-key-parameters "method.request.querystring.type"
```

Pour spécifier à API Gateway de mettre en cache un paramètre de demande d'intégration, utilisez `integration.request.location.name` comme paramètre de clé de cache.

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.

Console de gestion AWS

Pour vider le cache de l'étape d'API

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Choisissez une API dotée d'une étape avec un cache.
3. Dans le volet de navigation principal, choisissez Étapes, puis sélectionnez une étape avec un cache.
4. Choisissez le menu Actions d'étape, puis sélectionnez Vider le cache d'étape.

AWS CLI

La commande flush-stage-cache suivante vide le cache de l'étape :

```
aws apigateway flush-stage-cache \
    --rest-api-id a1b2c3 \
    --stage-name prod
```

> **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 politique au format suivant à un rôle d'exécution IAM pour l'utilisateur.

> **Note**
>
> Cross-account l'invalidation du cache n'est pas prise en charge.

JSON

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

```
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "execute-api:InvalidateCache"
            ],
            "Resource": [
                "arn:aws:execute-api:us-east-1:111111111111:api-id/stage-name/
GET/resource-path-specifier"
            ]
        }
    ]
}
```

Cette politique 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 plus d'informations sur la définition d'autorisations pour le service d'exécution API Gateway, consultez Contrôle de l'accès à une API REST avec des autorisations IAM.

Si vous n'imposez aucune politique `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 politique est en place, la mise en cache est activée et l'autorisation est requise.

Vous pouvez définir la manière dont API Gateway traite les demandes non autorisées en choisissant l'une des options suivantes :

Faire échouer la requête avec le code d'état 403

API Gateway renvoie une réponse `403 Unauthorized`.

Pour définir cette option à l'aide de l'API, utilisez `FAIL_WITH_403`.

Ignorer l'en-tête de contrôle du cache et ajouter un avertissement dans l'en-tête de la réponse

API Gateway 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`.

Ignorer l’en-tête de contrôle du cache

API Gateway traite la demande sans ajouter d’en-tête d’avertissement dans la réponse.

Pour définir cette option à l'aide de l'API, utilisez `SUCCEED_WITHOUT_RESPONSE_HEADER`.

Vous pouvez définir le comportement de traitement des demandes non autorisées à l’aide de la console API Gateway ou de l’ AWS CLI.

Console de gestion AWS

Pour spécifier le mode de traitement des demandes non autorisées

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Choisissez une API dotée d’une étape avec un cache.
3. Dans le volet de navigation principal, choisissez Étapes, puis sélectionnez une étape avec un cache.
4. Sous Détails de l’étape, choisissez Modifier.
5. Sous Traitement des requêtes non autorisées, sélectionnez une option.
6. Sélectionnez Continuer.
7. Passez en revue vos modifications et choisissez Enregistrer les modifications.

AWS CLI

La commande update-stage suivante met à jour une étape afin qu’elle gère les demandes non autorisées en rejetant la demande avec le code d’état 403 :

```
aws apigateway update-stage /
    --rest-api-id a1b2c3 /
    --stage-name 'prod' /
    --patch-operations 'op=replace,path=/*/*/caching/
unauthorizedCacheControlHeaderStrategy,value="FAIL_WITH_403"'
```

CloudFormation exemple d'une étape avec un cache

Le CloudFormation modèle suivant crée un exemple d'API, fournit un cache de `0.5` Go pour la `Prod` phase et active la mise en cache au niveau de la méthode pour toutes les méthodes. `GET`

> **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 pour l'offre gratuite AWS . Pour plus d'informations, consultez API Gateway Pricing (Tarification d'API Gateway).

Exemple CloudFormation modèle

```
AWSTemplateFormatVersion: 2010-09-09
Resources:
  Api:
    Type: 'AWS::ApiGateway::RestApi'
    Properties:
      Name: cache-example
  PetsResource:
    Type: 'AWS::ApiGateway::Resource'
    Properties:
      RestApiId: !Ref Api
      ParentId: !GetAtt Api.RootResourceId
      PathPart: 'pets'
  PetsMethodGet:
    Type: 'AWS::ApiGateway::Method'
    Properties:
      RestApiId: !Ref Api
      ResourceId: !Ref PetsResource
      HttpMethod: GET
      ApiKeyRequired: true
      AuthorizationType: NONE
      Integration:
        Type: HTTP_PROXY
        IntegrationHttpMethod: GET
        Uri: http://petstore-demo-endpoint.execute-api.com/petstore/pets/
  ApiDeployment:
    Type: 'AWS::ApiGateway::Deployment'
    DependsOn:
      - PetsMethodGet
```

```
    Properties:
      RestApiId: !Ref Api
  ApiStage:
    Type: 'AWS::ApiGateway::Stage'
    Properties:
      StageName: Prod
      Description: Prod Stage with a cache
      RestApiId: !Ref Api
      DeploymentId: !Ref ApiDeployment
      CacheClusterEnabled: True
      CacheClusterSize: 0.5
      MethodSettings:
        - ResourcePath: /*
          HttpMethod: '*'
          CachingEnabled: True
```

Compression de charge utile pour REST APIs dans API Gateway

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 des données utiles pour une API dans API Gateway
- Appel d'une méthode d'API avec des données utiles compressées dans API Gateway
- Réception d'une réponse d'API avec des données utiles compressées dans API Gateway

Activation de la compression des données utiles pour une API dans API Gateway

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 du 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 du AWS CLI

La create-rest-apicommande suivante crée une API avec compression de charge utile :

```
aws apigateway create-rest-api \
    --name "My test API" \
    --minimum-compression-size 0
```

La update-rest-apicommande suivante active la compression de charge utile pour une API existante :

```
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.

La update-rest-apicommande suivante désactive la compression de la charge utile :

```
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 des données utiles compressées dans API Gateway

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 des données utiles compressées dans API Gateway

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

permet d'activer la compression des données utiles 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.

Distribuez votre REST APIs aux clients dans API Gateway

Cette section fournit des informations sur la distribution de votre API Gateway APIs à vos clients. La distribution de votre API consiste à générer des applications SDKs pour que vos clients puissent les télécharger et les intégrer à leurs applications clientes, à documenter votre API afin que les clients sachent comment l'appeler depuis leurs applications clientes et à rendre votre API disponible dans le cadre de l'offre de produits.

Rubriques

- Plans d'utilisation et clés d'API pour les API REST dans API Gateway
- Documentation sur les API REST dans API Gateway
- Générer SDKs pour REST APIs dans API Gateway
- Vendez votre API Gateway APIs via AWS Marketplace

Plans d'utilisation et clés d'API pour les API REST dans API Gateway

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 de clés d'API pour les API REST dans 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 pour développeurs pour publier vos API, notez que les clients peuvent s'abonner à toutes les API d'un plan d'utilisation donné, même si vous ne les avez pas rendues visibles pour eux.

- 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.
- Une fois que vous avez ajouté une clé d'API à un plan d'utilisation, l'opération de mise à jour peut prendre quelques minutes.

Choix d'une source de clé d'API dans API Gateway

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 "Sortie d'un mécanisme d'autorisation 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".

La procédure ci-dessous montre comment choisir une source de clé d'API pour une API.

Console de gestion AWS

Pour choisir une source de clé d'API pour une API

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.

AWS CLI

La commande update-rest-api suivante met à jour une API afin de définir la source de clé d'API sur `AUTHORIZER` :

```
aws apigateway update-rest-api --rest-api-id 1234123412 --patch-operations
 op=replace,path=/apiKeySource,value=AUTHORIZER
```

Pour que le client soumette une clé d'API, définissez `value` sur `HEADER` dans la commande précédente.

REST API

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 d'API, définissez la `value` sur `AUTHORIZER` dans l'entrée `patchOperations` précédente.

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 valeur `Key` doit comprendre entre 20 et 128 caractères. Une valeur `Name` ne peut pas dépasser 1 024 caractères.

Un fichier de clé d'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 d'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”.

Configuration de clés d’API pour les API REST dans API Gateway

Pour configurer les clés d’API, procédez comme suit :

- Configurez des méthodes API pour exiger une clé d’API.
- Créez ou importez une clé d’API pour l’API dans une région.

Avant de configurer des clés d’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, consultez Développement d’API REST dans API Gateway et Déploiement d’une API REST dans API Gateway, respectivement.

Une fois votre clé d’API créée, vous devez l’associer à un plan d’utilisation. Pour de plus amples informations, veuillez consulter Configuration de plans d’utilisation pour les API REST dans API Gateway.

> **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 d’API sur une méthode
- Création d’une clé d’API
- Importation de clés d’API

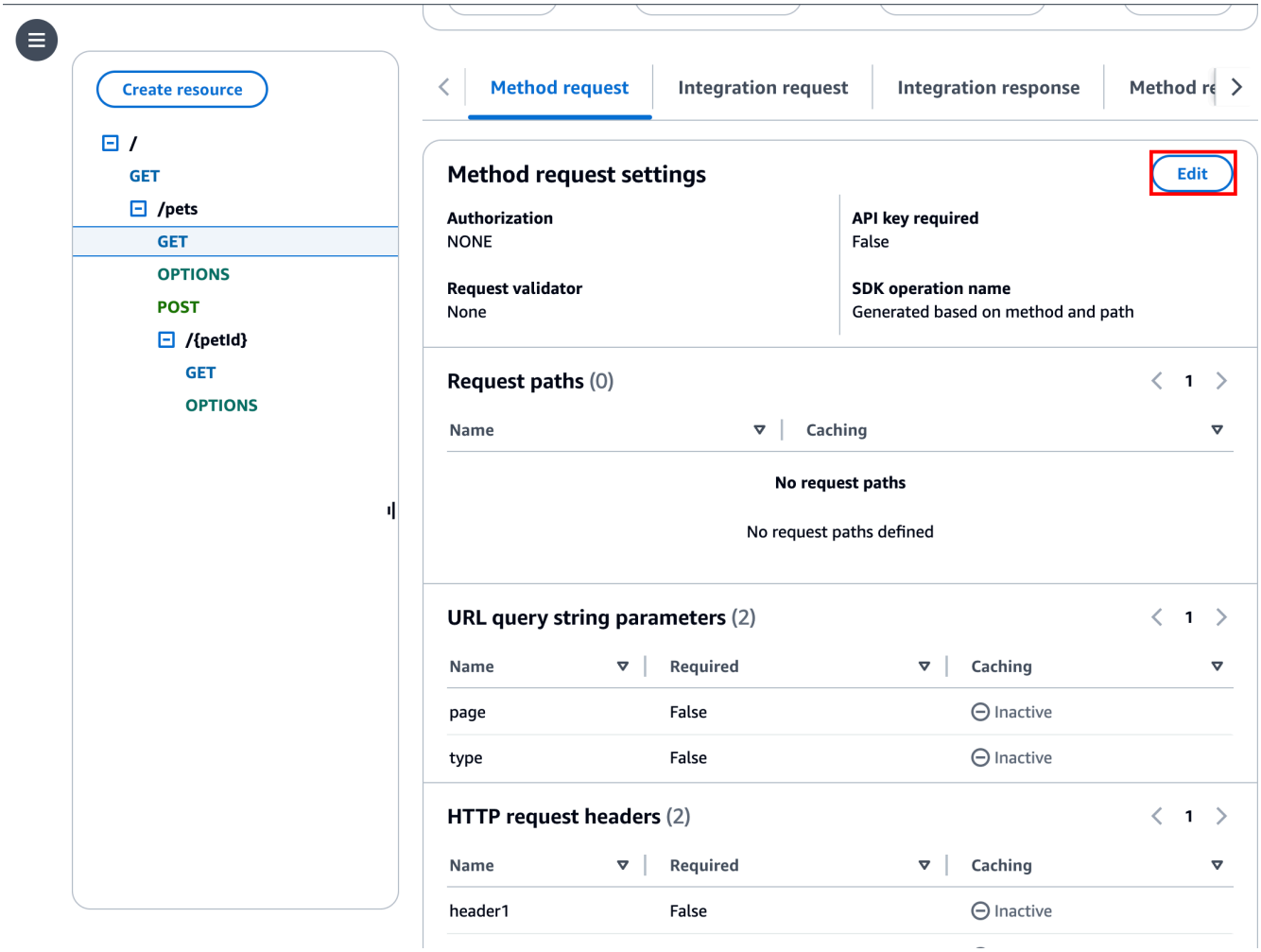
Exigence d'une clé d’API sur une méthode

La procédure suivante explique comment configurer une méthode d’API pour exiger une clé d’API.

Console de gestion AWS

Pour configurer une méthode d'API pour exiger une clé d'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 d'API obligatoire.
7. Choisissez Enregistrer.
8. Déployez ou redéployez l'API pour que l'exigence prenne effet.

Si l'option Clé d'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.

AWS CLI

La commande put-method suivante crée une méthode PUT qui nécessite une clé d'API :

```
aws apigateway put-method \
    --rest-api-id 1234123412 \
    --resource-id a1b2c3 \
    --http-method PUT \
    --authorization-type "NONE" \
    --api-key-required
```

La commande update-method suivante met à jour une méthode existante pour qu'elle exige une clé d'API :

```
aws apigateway update-method \
    --rest-api-id 1234123412 \
    --resource-id a1b2c3 \
    --http-method PUT \
    --patch-operations op="replace",path="/apiKeyRequired",value="true"
```

REST API

Pour exiger une clé d'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 d'une clé d'API

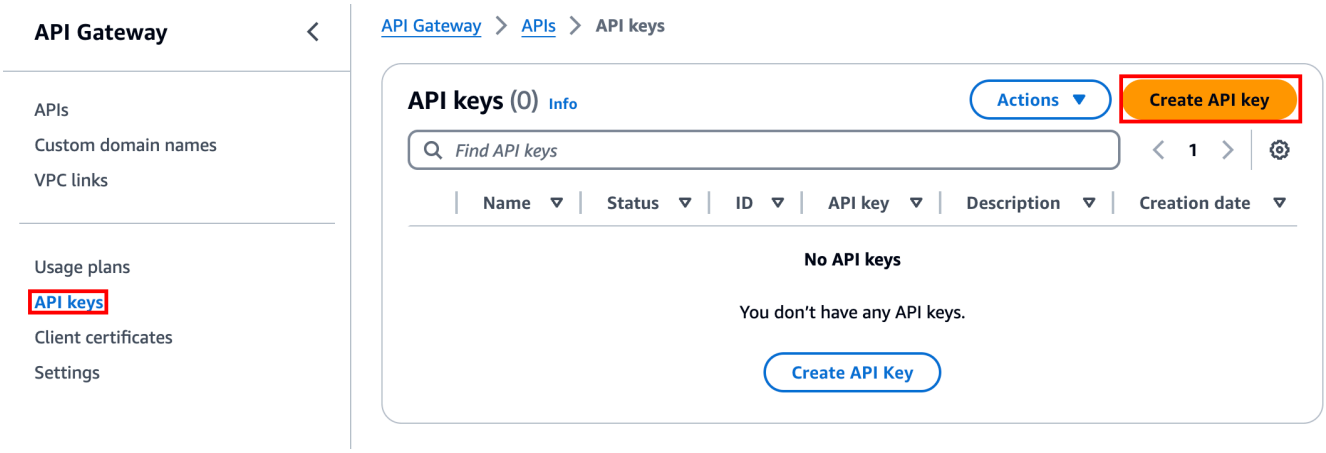
La procédure suivante montre comment créer une clé d'API. Si vous désirez importer votre clé d'API, ignorez cette étape.

Console de gestion AWS

Pour créer une clé d'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.

AWS CLI

La commande create-key suivante crée une clé d'API :

```
aws apigateway create-api-key \
   --name 'Dev API key' \
   --description 'API key for Devs' \
   --enabled
```

REST API

Appelez `apikey:create` pour créer une clé d'API.

Importation de clés d'API

La procédure suivante explique comment importer des clés d'API. Si vous avez déjà créé une clé d'API, ignorez cette étape.

Console de gestion AWS

Pour importer des clés d'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 d'API.
5. Pour charger un fichier de clés séparées par des virgules, sélectionnez 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.

AWS CLI

La commande import-api-keys suivante importe une clé d'API :

```
aws apigateway import-api-key \
    a--body fileb://keys.csv \
    --format csv
```

REST 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. Une fois votre API créée, vous allez configurer un plan d'utilisation. Pour de plus amples informations, veuillez consulter Configuration de plans d'utilisation pour les API REST dans API Gateway.

Configuration de plans d’utilisation pour les API REST dans API Gateway

Avant de créer un plan d’utilisation, assurez-vous d’avoir configuré vos clés d’API. Pour de plus amples informations, veuillez consulter Configuration de clés d’API pour les API REST dans API Gateway.

Rubriques

- Migration de votre API vers les plans d’utilisation par défaut (si nécessaire)
- Création d'un plan d'utilisation
- Ajout d’une étape d’API à un plan d’utilisation.
- Ajout d’une clé d’API à 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 d’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`propriété `apiStages` 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"
    }
]
```

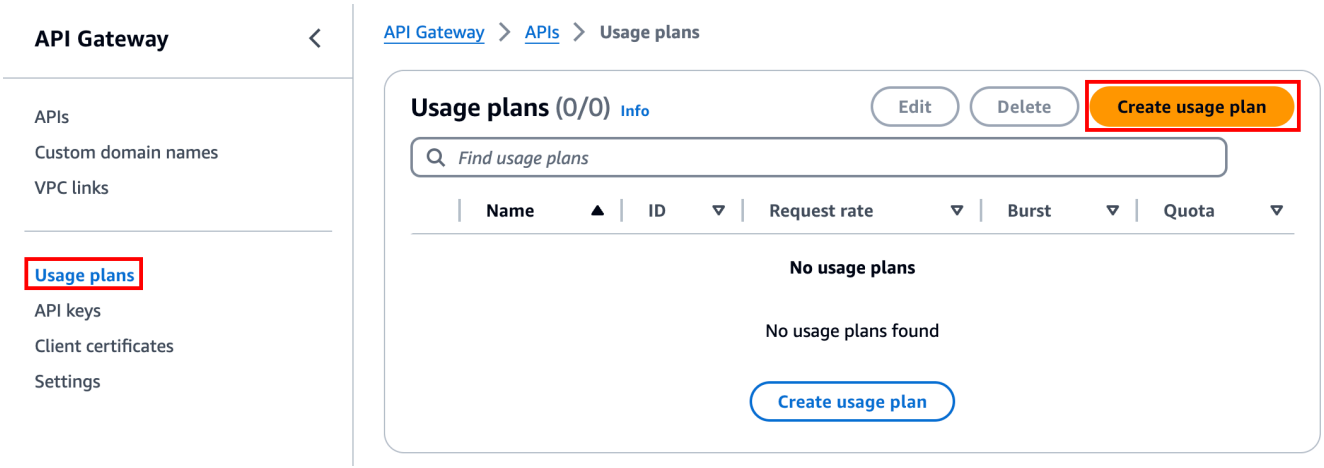
Création d'un plan d'utilisation

La procédure suivante explique comment créer un plan d'utilisation.

Console de gestion AWS

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.

AWS CLI

La commande create-usage-plan suivante crée un plan d'utilisation qui se réinitialise au début du mois :

```
aws apigateway create-usage-plan \
    --name "New Usage Plan" \
    --description "A new usage plan" \
    --throttle burstLimit=10,rateLimit=5 \
    --quota limit=500,offset=0,period=MONTH
```

REST API

Appelez `usageplan:create` pour créer un plan d'utilisation.

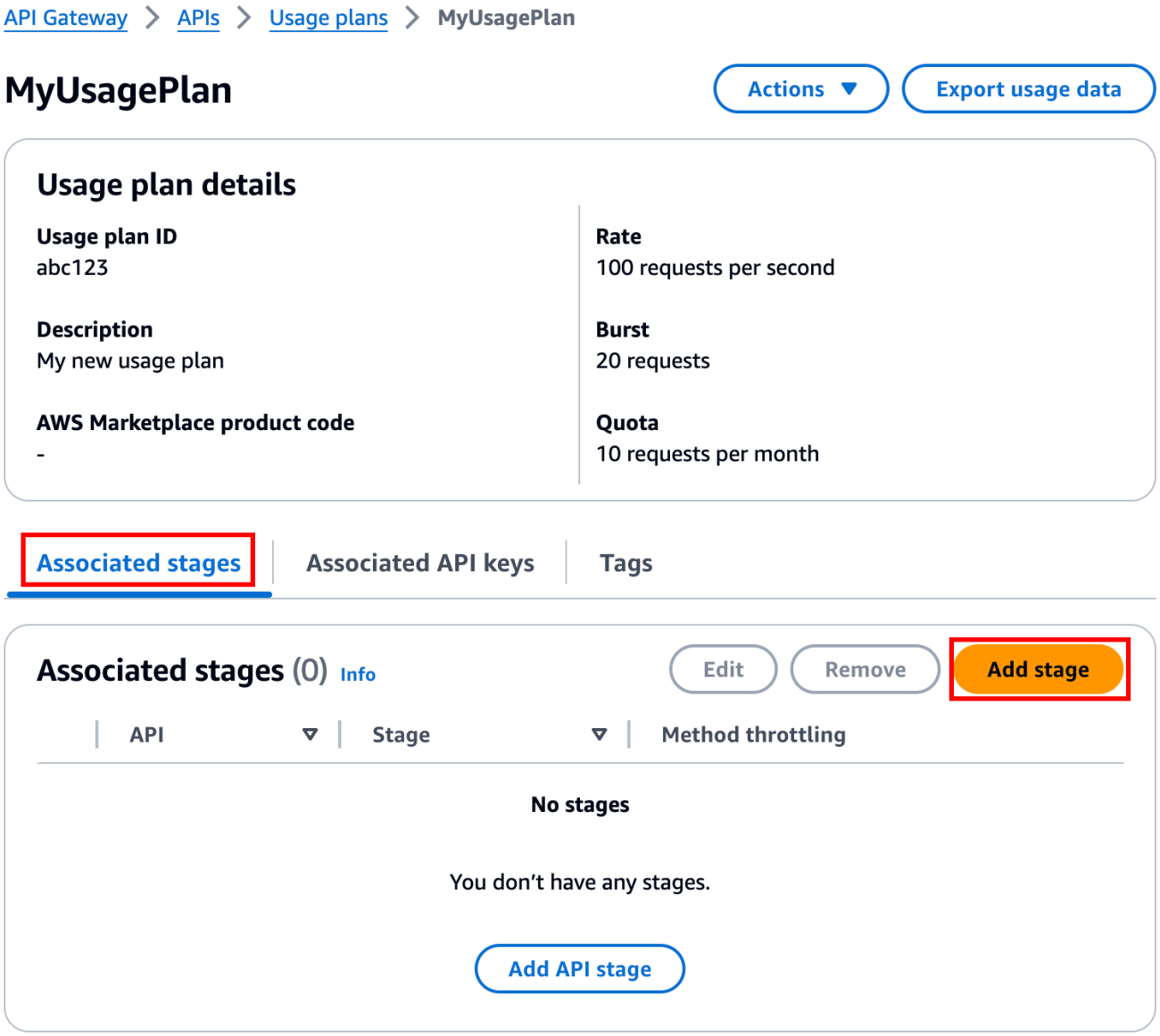
Ajout d'une étape d'API à un plan d'utilisation.

La procédure suivante explique comment ajouter une étape à un plan d'utilisation.

Console de gestion AWS

Pour ajouter une étape à un 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 Method-level la régulation, 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.

AWS CLI

La commande update-usage-plan suivante ajoute l'étape `Prod` d'une API à un plan d'utilisation :

```
aws apigateway update-usage-plan \
    --usage-plan-id abc123 \
    --patch-operations op="add",path="/apiStages",value="a1b1c2:Prod"
```

REST API

Appelez `usageplan:update` pour mettre à jour un plan d'utilisation.

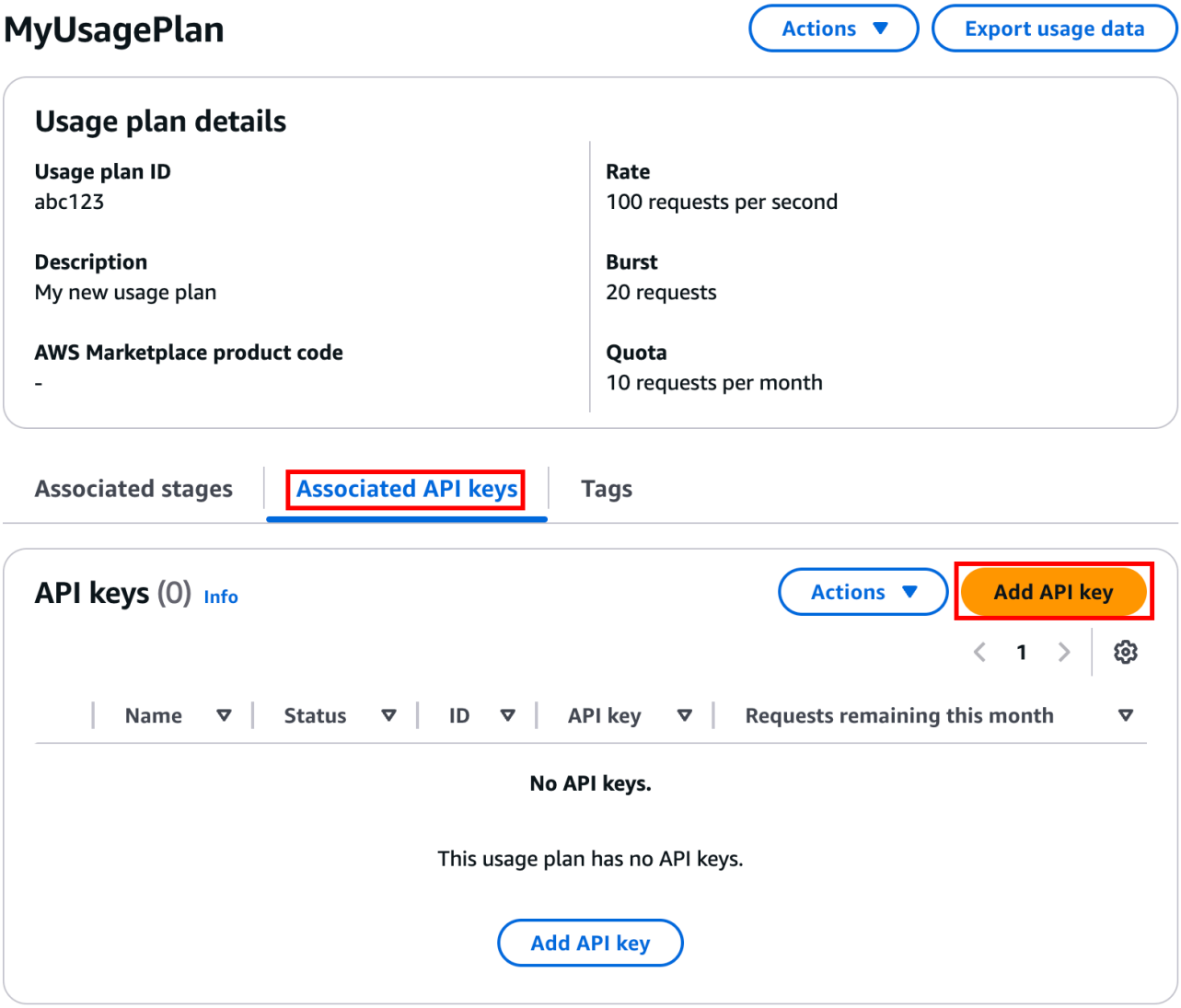
Ajout d'une clé d'API à un plan d'utilisation

La procédure suivante montre comment ajouter une clé d'API à un plan d'utilisation.

Console de gestion AWS

Pour ajouter une clé à un 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 Création d'une clé d'API.

3. Choisissez Ajouter une clé d'API.

AWS CLI

La commande create-usage-plan-key suivante associe une clé d'API existante à un plan d'utilisation :

```
aws apigateway create-usage-plan-key \
    --usage-plan-id a1b2c3 \
    --key-type "API_KEY" \
    --key-id aaa111bbb
```

REST API

Appelez `usageplankey:create` pour associer une clé d'API existante à un plan d'utilisation.

Vous pouvez également associer directement vos clés d'API à un plan d'utilisation au moment de leur importation. 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 d'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 d'API.

L'exemple suivant de requête `apikey:import` ajoute trois clés d'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 d'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.

Gestion d’un plan d’utilisation pour les API REST dans API Gateway

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.

Console de gestion AWS

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.

```
{
    "px1KW6...qBazOJH": [
      [
        0,
        5000
      ],
      [
        0,
        5000
      ],
```

```
      [
        0,
        10
      ]
    ]
  }
```

Les données d'utilisation de cet exemple présentent les données d'utilisation quotidiennes d'un client d'API, identifié par la clé d'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.

AWS CLI

Les exemples update-usage-plan suivants montrent comment ajouter, supprimer ou modifier les paramètres de limitation au niveau de la méthode dans un plan d'utilisation.

> **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}}"'
```

REST API

Appelez `usageplan:update` pour gérer un plan d'utilisation.

Créez et configurez des clés d'API et des plans d'utilisation avec CloudFormation

Vous pouvez l'utiliser CloudFormation pour exiger des clés d'API sur les méthodes d'API et créer un plan d'utilisation pour une API. L'exemple de 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 afin d’exiger une clé d’API pour invoquer la méthode. Définissez ensuite `api_key` dans la définition de la sécurité. Après avoir créé votre API, ajoutez la nouvelle étape d’API à votre plan d’utilisation.

L’exemple suivant crée une API et exige une clé d’API pour les méthodes `POST` et `GET` :

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"
      }
    }
  }
}
```

Test de plans d'utilisation pour les API REST dans API Gateway

À titre d'exemple, utilisons l' PetStore API, qui a été créée dansDidacticiel : création d'une API REST par l'importation d'un exemple. Supposons que l'API est configurée pour utiliser la clé d'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 de statut `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 d'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. En revanche, il consigne un message d'erreur `Usage Plan check failed due to an internal error` dans CloudWatch. Vous pouvez ignorer ces erreurs occasionnelles.

Appel d'une méthode à l'aide d'une clé d'API

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.
5. Appelez l'API à l'aide de la clé d'API. L'exemple de commande curl suivant invoque la méthode `GET` sur la ressource `getUsers` de l'étape `prod` d'une API à l'aide d'une clé d'API.

```
curl -H "X-API-Key: abcd1234" 'https://b123abcde4.execute-api.us-
west-2.amazonaws.com/prod/getUsers'
```

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. Un appel peut se présenter comme suit :

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 apprendre à créer un mécanisme d'autorisation par jeton, consultez the section called "Exemple de fonction Lambda de l'autorisateur TOKEN".
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.

Documentation sur les API REST dans API Gateway

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, ou utiliser l'un des kits AWS SDK, l'AWS CLI pour API Gateway ou 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 ou How to Streamline API Development on Amazon API Gateway en utilisant SmartBear le blog SwaggerHub on the 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 dans API Gateway

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. Cela inclut tout UTF-8 le contenu et toutes les principales

langues de localisation de votre documentation. 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 propage les attributs de OpenAPI-compliant contenu dans les définitions d'entités d'API à partir des `DocumentationPart` instances 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**	**state**	**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 les entités

Cas	`path`	`state`	`name`	Remar
				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**	**state**	**name**	Remar
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	state	name	Remar
				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`	N/A	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
	"path": "*resource_path* ", "method": "*http_verb* " }, ... }		/ et *, respectivement.	path par préfixe et correspondance de method quelle que soit la valeur.
Paramètre de la demande	{ "location": { "type": "QUERY_PARAMETER", "path": "*resource_path* ", "method": "*HTTP_verb* ", "name": "*query_parameter_name* " }, ... }	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`	N/A	Non
Modèle	`{ "location": { "type": "MODEL", "name": "model_name " }, ... }`	`type`	N/A	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 Didacticiel : création d’une API REST par l’importation d’un exemple.

Rubriques

- Documenter l'entité API
- Documenter une entité RESOURCE
- Documenter une entité METHOD
- Documenter une entité QUERY_PARAMETER
- Documenter une entité PATH_PARAMETER
- Documenter une entité REQUEST_HEADER
- Documenter une entité REQUEST_BODY
- Documenter une entité RESPONSE
- Documenter une entité RESPONSE_HEADER
- Documenter une entité RESPONSE_BODY
- Documenter une entité MODÈLE
- Documenter une entité AUTORISER

Documenter 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.

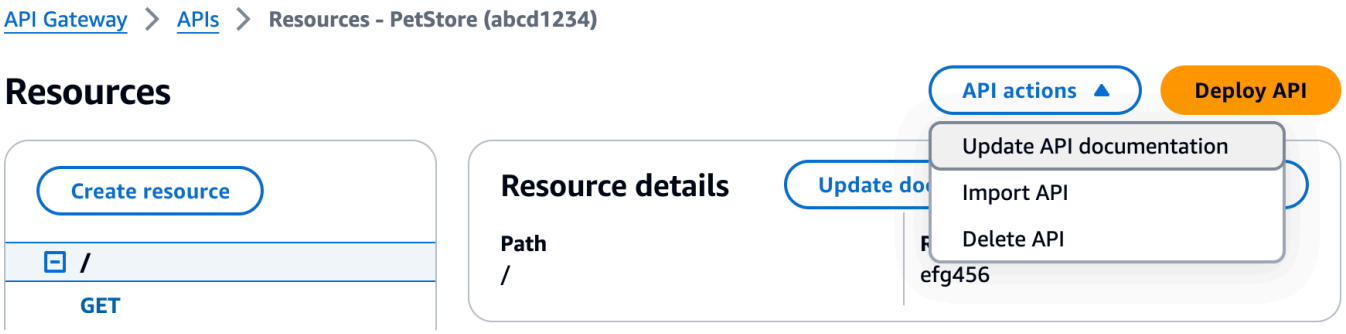
```
{
  "info": {
    "description": "Your first API Gateway API.",
    "contact": {
        "name": "John Doe",
        "email": "john.doe@api.com"
    }
  }
}
```

> **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.

Documenter une entité `RESOURCE`

Pour ajouter une nouvelle partie de la documentation pour une entité `RESOURCE`, procédez comme suit :

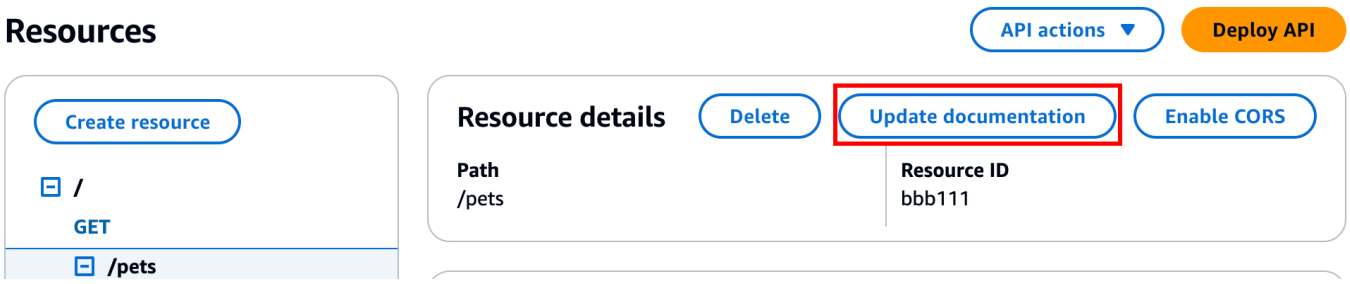
1. Dans le volet de navigation principal, choisissez Documentation, puis Créer une partie de la documentation.
2. Pour Type de documentation, sélectionnez Ressource.
3. Pour Chemin, entrez un chemin.
4. Entrez une description dans l'éditeur de texte, par exemple :

```
{
    "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.

Documenter une entité **METHOD**

Pour ajouter une nouvelle partie de la documentation pour une entité `METHOD`, procédez comme suit :

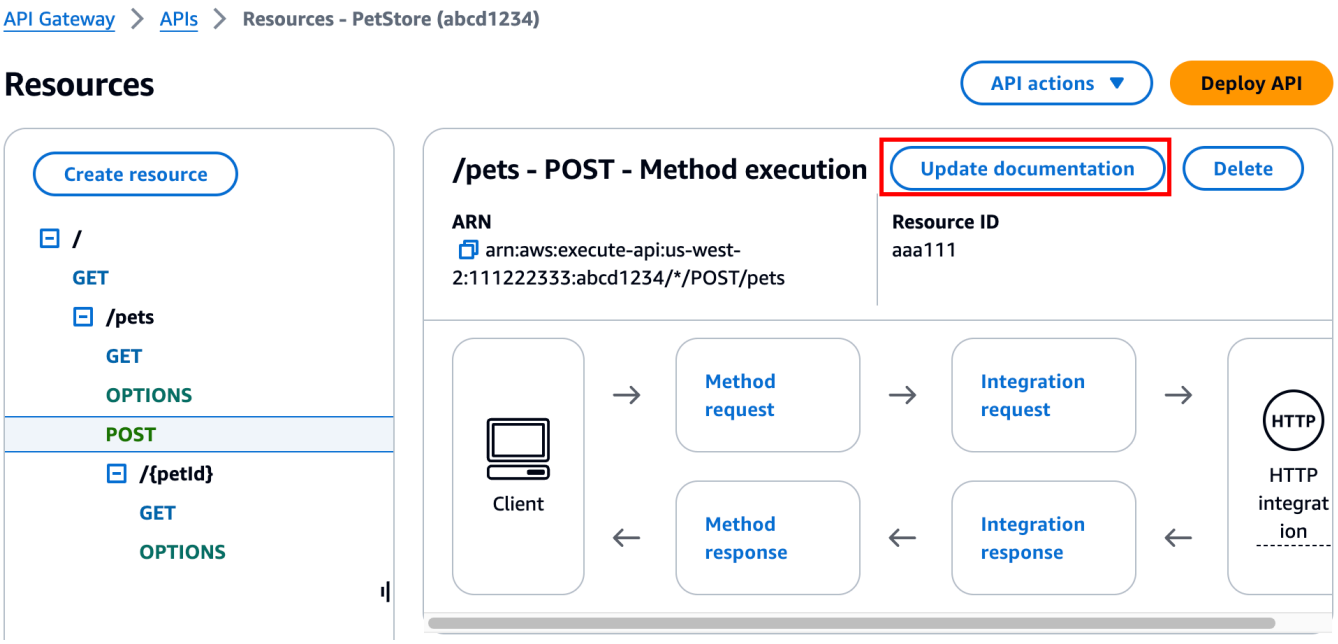
1. Dans le volet de navigation principal, choisissez Documentation, puis Créer une partie de la documentation.
2. Pour Type de documentation, sélectionnez Méthode.
3. Pour Chemin, entrez un chemin.
4. Pour Méthode, sélectionnez un verbe HTTP.
5. Entrez une description dans l'éditeur de texte, par exemple :

```
{
  "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.

Documenter 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.

Documenter 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.

Documenter 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.

Documenter 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.

Documenter 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.

Documenter 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.

Documenter 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.

Documenter une entité **MODÈLE**

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."
  }
}
```

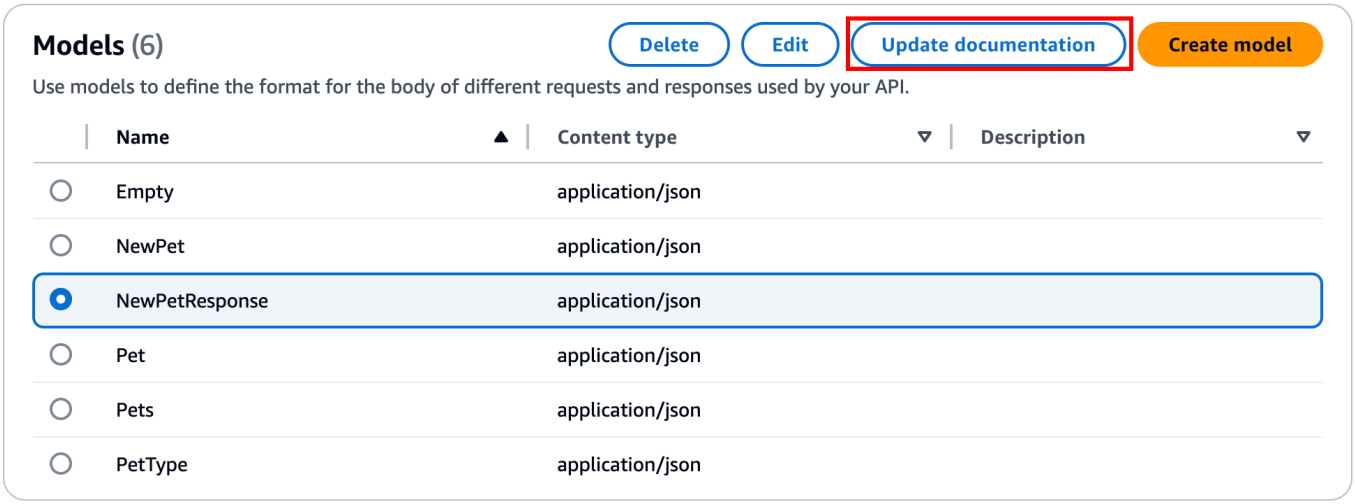
Lorsque l'API est exportée, les propriétés de `DocumentationPart` remplacent les valeurs dans le schéma d'origine.

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.

Documenter une **`entité AUTORISER`**

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 Didacticiel : création d'une API REST par l'importation d'un exemple.

Rubriques

- Documenter l'entité API
- Documenter une entité RESOURCE

- Documenter une entité METHOD
- Documenter une entité QUERY_PARAMETER
- Documenter une entité PATH_PARAMETER
- Documenter une entité REQUEST_BODY
- Documenter une entité REQUEST_HEADER
- Documenter une entité RESPONSE
- Documenter une entité RESPONSE_HEADER
- Documenter une entité AUTORISER
- Documenter une entité MODÈLE
- Mise à jour de certaines parties de la documentation
- Liste des parties de la documentation

Documenter 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.

Documenter 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 de 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.

Documenter 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. 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/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}"
}
```

Documenter 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`.

Documenter 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. 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/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}"
}
```

Documenter 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. 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/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}"
}
```

Documenter 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. 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/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}"
}
```

Documenter 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. Par exemple :

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

Documenter 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.

Documenter une **entité AUTORISER**

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.

Documenter une entité **`MODÈLE`**

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 personnalisée). 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`.

À 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 OpenAPI-compliant attribut défini dans la `properties` carte d'une partie de la documentation, API Gateway insère l'attribut dans la définition de l'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, l'interface utilisateur OpenAPI) analyse les définitions des entités de l'API pour extraire les attributs de documentation, les attributs autres que ceux d'une instance `DocumentationPart` « » ne sont pas disponibles pour l'outil. OpenAPI-compliant `properties` 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 documentation à importer dans l'extension Objet x-amazon-apigateway-documentation d'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 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 de plus amples informations, veuillez

consulter the section called “Erreurs et avertissements liés à l’importation de votre API dans API Gateway”.

6. Choisissez Importer.

Contrôle de l’accès à la documentation de l’API dans API Gateway

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 accorder à votre équipe de documentation l'accès nécessaire pour 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.

JSON

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

    {
      "Sid": "StmtDocPartsAddEditViewDelete",
      "Effect": "Allow",
      "Action": [
        "apigateway:GET",
        "apigateway:PUT",
        "apigateway:POST",
        "apigateway:PATCH",
        "apigateway:DELETE"
      ],
      "Resource": [
        "arn:aws:apigateway:us-east-1:111111111111:/restapis/*/documentation/*"
      ]
    }
  ]
}
```

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".

Générer SDKs pour REST APIs 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 devez générer votre kit SDK après avoir créé, testé et déployé votre API dans 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 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 dans API Gateway
- Génération du kit SDK Android d'une API dans API Gateway
- Génération du kit SDK iOS d'une API dans API Gateway
- Génération du JavaScript SDK d'une API REST dans API Gateway
- Génération du kit SDK Ruby d'une API dans API Gateway
- Génération SDKs pour une API à l'aide de AWS CLI commandes dans API Gateway

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";
  }

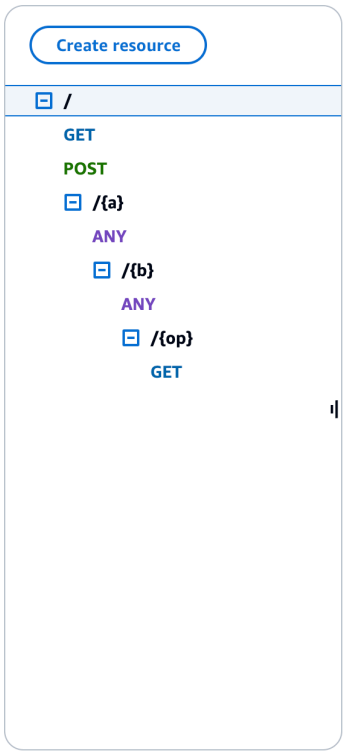
  const res = {};
  res.a = Number(event.a);
  res.b = Number(event.b);
```

```
  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 :



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'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 Créer un modèle.
3. Pour 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 Créer un modèle.
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 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 dans API Gateway

La procédure suivante montre comment générer le kit SDK Java d'une API dans API Gateway.

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 dans API Gateway

La procédure suivante montre comment générer le kit SDK Android d'une API dans API Gateway.

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. Ceci est utilisé dans le `pom.xml` fichier et doit suivre un*`major`*.*`minor`*.*`patch`*motif (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 dans API Gateway

La procédure suivante montre comment générer le kit SDK iOS d'une API dans API Gateway.

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 Utiliser le 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 Utiliser le 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 dans API Gateway

La procédure suivante montre comment générer le JaveScript SDK d'une API dans API Gateway.

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 dans API Gateway

La procédure suivante montre comment générer le kit SDK Ruby d'une API dans API Gateway.

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 SDKs pour une API à l'aide de AWS CLI commandes dans API Gateway

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 de plus amples informations, veuillez consulter Invocation d'une API REST dans API Gateway.

Vendez votre API Gateway APIs via AWS Marketplace

Après avoir créé, testé et déployé le vôtre APIs, vous pouvez le regrouper dans un plan d'utilisation d'API Gateway et le vendre 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 APIs le vôtre 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 AWS Marketplace l'intégration avec API Gateway
- Gestion de l'abonnement des clients aux plans d'utilisation

Initialisation de AWS Marketplace l'intégration avec API Gateway

Les tâches suivantes concernent l'initialisation unique de AWS Marketplace l'intégration avec API Gateway, qui vous permet de vendre votre produit en APIs 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 le price-per-request et est utilisée par API Gateway pour mesurer les demandes qui vous sont adressées APIs.

Création du rôle de mesure

Créez un rôle IAM nommé `ApiGatewayMarketplaceMeteringRole` avec la politique d'exécution et la politique d'approbation suivantes. Ce rôle permet à API Gateway d'envoyer des métriques d'utilisation AWS Marketplace en votre nom.

Politique d'exécution du rôle de mesure

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Action": [
        "aws-marketplace:BatchMeterUsage",
        "aws-marketplace:ResolveCustomer"
      ],
      "Resource": "*",
      "Effect": "Allow"
    }
  ]
}
```

Politique de la relation d'approbation du rôle de mesure

JSON

```
{
  "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 est terminée, l'application du portail des développeurs doit présenter au client la clé d'API associée et informer le client que la clé d'API doit être incluse dans l'`x-api-key`en-tête des demandes adressées au APIs.

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 vos API REST dans API Gateway

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 politiques 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

- Exigez des certificats clients pour votre API grâce à l'authentification TLS mutuelle dans API Gateway
- Présenter les certificats clients aux services principaux avec un protocole TLS mutuel dans API Gateway
- AWS WAF À utiliser pour protéger votre REST APIs dans API Gateway
- Limitez les requêtes adressées à votre REST APIs pour améliorer le débit dans API Gateway
- API REST privées dans API Gateway

Exigez des certificats clients pour votre API grâce à l'authentification TLS mutuelle dans API Gateway

L'authentification TLS mutuelle nécessite une authentification bidirectionnelle entre le client et le serveur. Avec le TLS mutuel, les clients doivent présenter X.509 des certificats pour vérifier leur identité afin d'accéder à votre API. L'authentification TLS mutuelle est une exigence courante pour l'Internet des objets (IOT) et les applications inter-entreprises.

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 “Désactivation du point de terminaison par défaut”.

Rubriques

- Conditions préalables pour l’authentification TLS mutuelle
- Configuration de l’authentification TLS mutuelle pour un nom de domaine personnalisé
- Appeler une API à l’aide d’un nom de domaine personnalisé qui nécessite une authentification TLS mutuelle
- Mise à jour de votre magasin de confiance
- Désactiver l’authentification TLS mutuelle
- Résolution des problèmes d’authentification TLS mutuelle pour votre API REST

Conditions préalables pour l’authentification TLS mutuelle

Pour configurer des authentifications TLS mutuelles, vous avez besoin des éléments suivants :

- Un nom de domaine personnalisé régional
- 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 AWS Certificate Manager certificats 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.

À l'aide d'un fichier importé ou Autorité de certification privée AWS certificat

Pour utiliser un certificat importé dans ACM ou un certificat depuis Autorité de certification privée AWS 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'en avez pas encore`ownershipVerificationCertificate`, rendez-vous sur https://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 le protocole TLS mutuel, créez une banque de X.509 certificats fiables 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 magasin de confiance :

- SHA-256 ou plus fort
- RSA-2048 ou plus fort
- 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
X.509 syntaxe	Le certificat doit répondre aux exigences de X.509 syntaxe.
Intégrité	Le contenu du certificat ne doit pas avoir été modifié par rapport à celui signé par l'autorit é 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 certifica ts peuvent avoir une longueur de chaîne maximale de quatre.

Chargement du 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 cp 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 REST, vous devez utiliser un nom de domaine personnalisé régional pour votre API, avec une politique de sécurité `TLS_1_2`. Pour plus d'informations sur le choix d'une politique de sécurité, consultez the section called "Sélection 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. La commande create-domain-name suivante crée un nom de domaine personnalisé avec une authentification TLS mutuelle.

```
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 code de statut `403`. 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 commande update-domain-name suivante met à jour un nom de domaine personnalisé de manière à ce qu'il utilise la nouvelle version du magasin de confiance.

```
aws apigateway update-domain-name \
    --domain-name api.example.com \
    --patch-operations op='replace',path='/mutualTlsAuthentication/
truststoreVersion',value='abcdef123'
```

Désactiver l'authentification TLS mutuelle

La commande update-domain-name suivante désactive l'authentification TLS mutuelle :

```
aws apigateway update-domain-name \
    --domain-name api.example.com \
    --patch-operations op='replace',path='/mutualTlsAuthentication/
truststoreUri',value=''
```

Résolution des problèmes d'authentification TLS mutuelle pour votre API REST

Les rubriques suivantes fournissent des conseils de dépannage pour les erreurs et problèmes que vous pouvez rencontrer lorsque vous activez l'authentification TLS mutuelle.

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 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. Vous trouverez des informations supplémentaires sur le renouvellement du certificat dans le guide : Résolution des problèmes dus au renouvellement des certificats gérés d'ACM.

Résolution des problèmes liés au renvoi d'un certificat incorrect

Lors de la migration d'un certificat dédié d'un nom de domaine complet vers un nom de domaine client générique, API Gateway peut renvoyer le certificat du nom de domaine complet au lieu du nom de domaine générique.

La commande suivante indique le certificat qui est renvoyé par API Gateway :

```
openssl s_client -connect hostname:port
```

Si le certificat obtenu correspond au nom de domaine client générique, contactez Support pour ouvrir un ticket.

Présenter les certificats clients aux services principaux avec un protocole TLS mutuel dans API Gateway

API Gateway peut présenter un certificat client à votre backend lors de la prise de contact TLS. Votre backend peut ensuite vérifier que les demandes proviennent d'API Gateway. Cela crée une connexion TLS mutuelle entre API Gateway et votre service principal.

Vous pouvez choisir entre deux approches :

- AWS Certificate Manager Certificats (ACM) — Importez votre propre CA-signed certificat depuis ACM et joignez-le à votre étape d'API. Avec cette approche, vous bénéficiez de chaînes de certificats complètes, d'un renouvellement automatique et d'une intégration à votre infrastructure PKI existante.
- Gateway-generated Certificats d'API : générez un certificat auto-signé dans API Gateway. Cette approche ne nécessite aucune infrastructure de certificats externe. Le certificat expire au bout de 365 jours et doit faire l'objet d'une rotation manuelle.

Rubriques

- Comment fonctionne l'authentification par certificat client principal
- Utilisez votre propre certificat ACM pour le TLS mutuel du backend dans API Gateway
- Utiliser un Gateway-generated certificat d'API pour l'authentification du backend dans API Gateway
- Autorités de certification prises en charge par API Gateway pour les intégrations HTTP et les intégrations de proxy HTTP

Comment fonctionne l'authentification par certificat client principal

Avec le protocole TLS mutuel sortant (mTLS), API Gateway agit comme un client TLS. Il présente un certificat client à votre serveur principal lorsqu'il se connecte. Votre backend vérifie ce certificat pour confirmer que les demandes proviennent d'API Gateway.

Les mTL sortants sont différents des mTL entrants. Les mTLS entrants nécessitent que les clients API présentent des certificats à API Gateway. Le mTLS sortant sécurise le lien entre API Gateway et votre backend.

Séquence de prise de contact TLS

Les étapes suivantes montrent comment API Gateway configure une connexion TLS mutuelle avec votre backend :

1. API Gateway envoie un ClientHello message au backend.
2. Le backend envoie son certificat de serveur à API Gateway.
3. API Gateway vérifie le certificat du serveur.
4. Le backend envoie un CertificateRequest à API Gateway.
5. API Gateway envoie la chaîne de certificats client au backend.
6. Le backend vérifie le certificat client par rapport à son magasin de confiance.
7. La connexion TLS mutuelle est prête. API Gateway transmet la demande d'API.

Si le backend ne demande pas de certificat client à l'étape 4, API Gateway termine la prise de contact TLS sans en envoyer un. La connexion utilise le protocole TLS unidirectionnel standard.

Comparez les options de certificat

Le tableau suivant compare les deux approches de configuration des certificats clients principaux.

ACM-managed par rapport aux Gateway-generated certificats API

Fonctionnalité	ACM-managed certificats	Gateway-generated Certificats API
Source du certificat	Votre PKI ou Autorité de certification privée AWS	API Gateway auto-signée
Configuration	Référencez le certificat sur scène par son ARN ACM ; gérez le cycle de vie du certificat à l'aide des API ACM.	Gérez sur la page Certificats clients ou à l'aide des API de certificats clients API Gateway.
Renouvellement	Cela dépend de la façon dont le certificat a été délivré. Pour les certificats émis par ACM Autorité de certification privée AWS, ACM les renouvelle automatiquement et API Gateway propage	Manuel. Le certificat expire au bout de 365 jours et doit faire l'objet d'une rotation manuelle.

Fonctionnalité	ACM-managed certificats	Gateway-generated Certificats API
	la mise à jour sans redéploie ment ni interruption de service. Pour les certificats que vous importez dans ACM, vous devez les réimporter avant leur expiratio n ; API Gateway propage ensuite la réimportation automatiquement.	
Support de chaîne	Chaîne complète (jusqu'à 5 certificats)	Certificat auto-signé unique
Modèle de confiance du backend	Le backend fait confiance à votre autorité de certification	Le backend doit épingler le certificat API Gateway

Utilisez votre propre certificat ACM pour le TLS mutuel du backend dans API Gateway

Vous pouvez configurer API Gateway pour présenter votre propre CA-signed certificat aux services principaux. Importez votre certificat dans AWS Certificate Manager (ACM) ou émettez-en un via Autorité de certification privée AWS. Liez ensuite l'ARN du certificat ACM à votre étape d'API.

Conditions préalables

Avant de configurer un certificat client ACM, vous devez disposer des éléments suivants :

- AWS Certificate Manager accès dans la même région que votre API.
- Autorisations IAM : `acm:ImportCertificate` et `acm:DescribeCertificate` (pour l'importation de l'option A), ou `acm:RequestCertificate` (pour l'option B), ou `acm-pca:IssueCertificateacm-pca:GetCertificate`, et `acm:ImportCertificate` (pour l'option C).
- Une API REST déployée sur une scène.

Étape 1 : Importez un certificat ou émettez-en un via Autorité de certification privée AWS

Vous pouvez soit importer un certificat à partir de votre PKI existante, soit émettre un nouveau certificat via Autorité de certification privée AWS. Les deux chemins produisent un ARN de certificat ACM que vous utiliserez à l'étape suivante.

> **Note**
>
> Les certificats publics ACM ne sont pas pris en charge pour l'authentification du client principal. À compter du 11 juin 2025, AWS Certificate Manager n'émettra plus de certificats publics avec l'utilisation `clientAuth` étendue des clés (EKU). Cette fonctionnalité nécessite cet EKU. Utilisez donc un certificat que vous importez dans ACM ou un certificat émis via. Autorité de certification privée AWS

> **Note**
>
> Créez le certificat ACM dans la même AWS région que l'API REST qui l'utilisera. Les certificats ACM sont des ressources régionales. Le certificat doit donc exister dans la région de votre API.

Option A : Importer depuis votre PKI existante

Pour importer un certificat client et sa clé privée dans ACM, exécutez la commande suivante. Pour plus d'informations, consultez la section Importation de certificats dans le AWS Certificate Manager Guide de l'utilisateur.

```
aws acm import-certificate \
  --certificate fileb://client-cert.pem \
  --private-key fileb://private-key.pem \
  --certificate-chain fileb://ca-chain.pem \
  --region region
```

La commande renvoie l'ARN du certificat ACM. Enregistrez cette valeur pour l'étape suivante.

Option B : demandez un certificat via Autorité de certification privée AWS (ACM-managed)

Pour demander un certificat privé qu'ACM gère et peut renouveler automatiquement, exécutez la commande suivante. Pour plus d'informations, consultez la section Demande de certificat privé dans le Guide de AWS Certificate Manager l'utilisateur.

```
aws acm request-certificate \
  --domain-name www.example.com \
  --certificate-authority-arn arn:aws:acm-pca:us-east-1:123456789012:certificate-
authority/12345678-1234-1234-1234-123456789012 \
```

```
  --region region
```

La commande renvoie l'ARN du certificat ACM. Enregistrez cette valeur pour l'étape suivante.

Option C : Émettre par Autorité de certification privée AWS et importez dans ACM

Si vous avez besoin de contrôler directement les paramètres du certificat (tels que des extensions personnalisées ou des algorithmes de signature), vous pouvez émettre un certificat via ACM, Autorité de certification privée AWS puis l'importer dans ACM. Les certificats importés de cette manière ne sont pas renouvelés automatiquement par ACM. Assurez-vous que le certificat répond aux exigences du certificat. Pour plus d'informations sur l'émission de certificats privés, consultez la section Émission d'un certificat d'entité finale privée dans le Guide de l'Autorité de certification privée AWS utilisateur.

```
aws acm-pca issue-certificate \
  --certificate-authority-arn arn:aws:acm-pca:us-east-1:123456789012:certificate-
authority/12345678-1234-1234-1234-123456789012 \
  --csr fileb://csr.pem \
  --signing-algorithm SHA256WITHRSA \
  --validity Value=365,Type=DAYS
```

> **Récupérez et importez le certificat**
>
> La `issue-certificate` commande renvoie un ARN de Autorité de certification privée AWS certificat, et non un ARN ACM. Pour utiliser ce certificat avec API Gateway, récupérez-le à l'aide de, `aws acm-pca get-certificate` puis importez-le dans ACM à l'aide `aws acm import-certificate` de. L'importation produit l'ARN du certificat ACM que vous utiliserez à l'étape suivante. Lorsque vous exécutez`aws acm import-certificate`, définissez `--region` la région de votre API pour que le certificat ACM y soit créé.

Étape 2 : Configuration d'un stage d'API pour utiliser le certificat ACM

Une fois que vous avez un ARN de certificat ACM, configurez votre étape d'API pour présenter le certificat à votre backend.

Pour configurer une scène (console)

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

2. Choisissez votre API REST.
3. Choisissez Stages (Étapes).
4. Dans la section Détails de l'étape, choisissez Modifier.
5. Pour le certificat client, sélectionnez votre certificat ACM dans la liste déroulante.
6. Sélectionnez Enregistrer les modifications.

Pour configurer une étape (AWS CLI)

Exécutez la commande suivante :

```
aws apigateway update-stage \
  --rest-api-id abc123 \
  --stage-name prod \
  --patch-operations op='replace',path=/clientCertificateId,value=arn:aws:acm:us-
east-1:123456789012:certificate/12345678-1234-1234-1234-123456789012
```

> **Note**
>
> API Gateway utilise le même `clientCertificateId` champ pour les Gateway-generated certificats ACM et API. Lorsque vous fournissez un ARN de certificat ACM, API Gateway détecte automatiquement le format et utilise le ACM-managed flux de travail.

Étape 3 : Vérifiez la configuration

Pour vérifier qu'API Gateway envoie le certificat à votre backend, procédez comme suit :

> **Le backend doit demander un certificat client**
>
> Votre backend doit être configuré pour demander un certificat client lors de la prise de contact TLS. Si le backend n'en demande pas, API Gateway ne présente pas le certificat.

1. Invoquez le point de terminaison de votre API.
2. Vérifiez que votre backend obtient le certificat client lors de la prise de contact TLS.
3. Vérifiez que votre backend accepte le certificat et renvoie une réponse correcte.

Si le backend rejette le certificat, vérifiez que la chaîne de certificats peut être validée par rapport au trust store du backend.

Exigences du certificat

Le certificat Leaf que vous configurez doit répondre aux exigences suivantes.

Exigences relatives aux certificats clients ACM

Exigence	Description
Longueur maximale de la chaîne	5 certificats
Validité	Le certificat ne doit pas avoir expiré ou être encore valide lorsque vous le configurez
Région	Le certificat ACM doit se trouver dans la même région que l'API
Compte	Le certificat ACM doit être associé au même compte que l'API
Utilisation étendue des clés (EKU)	Le cas échéant, doit inclure`clientAuth` . En cas d'absence, le certificat est accepté.
Utilisation des clés (KU)	Le cas échéant, doit inclure `digitalSignature` ou`keyAgreement` . En cas d'absence, le certificat est accepté.
Algorithme clé	Doit être l'un des suivants : RSA 2048, RSA 3072, RSA 4096, ECDSA (EC_Prime256v1), ECDSA (EC_SecP384R1) ou ECDSA P-256 (EC_SecP521R1) P-384 P-521
État du certificat ACM	Doit être `ISSUED`

> **Note**
>
> API Gateway ne valide pas la chaîne de confiance entre le certificat initial et le certificat intermédiaire. API Gateway ne valide pas non plus l'intention du certificat ni les contraintes de base (telles que`CA:TRUE`) sur les certificats intermédiaires. Votre backend effectue ces validations lors de la prise de contact TLS.

Renouvellement et propagation des certificats

Lorsqu'un certificat est modifié dans ACM, API Gateway détecte la mise à jour et propage automatiquement le nouveau certificat. Vous n'avez pas besoin de redéployer votre stage et votre API ne subit aucune interruption pendant la rotation.

La propagation des certificats est finalement cohérente. Pendant la mise à jour, votre backend peut recevoir l'ancien ou le nouveau certificat jusqu'à la fin de la propagation.

Le mode de renouvellement du certificat dépend de la manière dont il a été délivré :

- Certificats émis via Autorité de certification privée AWS (ACM-managed) (Option B) — ACM renouvelle automatiquement ces certificats. API Gateway détecte le renouvellement et se met à jour automatiquement.
- Certificats émis Autorité de certification privée AWS et importés (option C) — ACM ne renouvelle pas automatiquement les certificats importés. Vous devez réimporter le certificat renouvelé. Une fois que vous avez réimporté le certificat, API Gateway détecte le changement et le met à jour automatiquement.
- Certificats importés depuis votre PKI (option A) — Vous devez réimporter le certificat renouvelé dans ACM. Une fois que vous avez réimporté le certificat, API Gateway détecte le changement et le met à jour automatiquement.

ACM envoie des notifications d'expiration des certificats via Amazon EventBridge. Vous pouvez utiliser ces notifications pour configurer des alarmes avant l'expiration d'un certificat.

Comportement et limites des certificats ACM

Affichage du certificat configuré

Les certificats ACM n'apparaissent pas dans les réponses `GetClientCertificate` ni dans les réponses de `GetClientCertificates` l'API. Pour afficher l'ARN du certificat ACM configuré sur une scène, utilisez GetStage. Pour afficher les détails du certificat, utilisez les API ACM DescribeCertificate et GetCertificate.

Réutilisation à travers les étapes

Vous pouvez associer le même certificat ACM à plusieurs étapes. Chaque étape référence indépendamment le certificat par son ARN.

Les API de certificats clients ne s'appliquent pas aux certificats ACM

Les certificats ACM ne sont pas des Gateway-managed ressources d'API. Les `DeleteClientCertificate` API`GetClientCertificate`,`UpdateClientCertificate`, et renvoient une valeur `NotFoundException` lorsqu'elles sont appelées avec un ARN de certificat ACM. Utilisez les API ACM pour gérer le cycle de vie des certificats.

Nettoyage automatique des associations de certificats

Lorsque vous supprimez un certificat ACM d'un stage, que vous mettez à jour un stage pour utiliser un autre certificat, ou que vous supprimez un stage ou une API REST, API Gateway nettoie automatiquement l'association de certificats. Aucune action manuelle n'est requise.

Supprimer le certificat ACM

ACM ne vous permet pas de supprimer un certificat alors qu'API Gateway y est activement associé. Pour supprimer le certificat d'ACM, supprimez-le d'abord de toutes les étapes qui le référencent.

Utiliser un Gateway-generated certificat d'API pour l'authentification du backend dans API Gateway

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".

> **Tip**
>
> Pour utiliser votre propre CA-signed certificat d'ACM au lieu d'un certificat Gateway-generated auto-signé par API, consultez. the section called "Utilisez votre propre certificat ACM"

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

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 un REST ou une WebSocket API.
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 PEM-encoded publique.

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 un WebSocket REST ou une API pour lequel vous souhaitez utiliser le certificat client. Vérifiez que l'API a été déployée dans 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.

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. Choisissez une méthode pour l'API REST. 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 l' AWS CLI.

Avant de configurer un serveur HTTPS principal pour vérifier le certificat SSL client d'API Gateway, vous devez avoir obtenu la clé PEM-encoded privée et un certificat côté serveur fournis par une autorité de certification fiable.

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 les intégrations HTTP et les intégrations 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 Node.js serveur du backend, vous pouvez configurer le serveur de la manière suivante :

```
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 modules client-certificate-auth pour authentifier les demandes des clients à l'aide de certificats. PEM-encoded

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.

Faites pivoter un certificat client expirant à l'aide du Console de gestion AWS

La procédure suivante montre comment effectuer la rotation d'un certificat client dans la console pour une API déployée précédemment.

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.

Faites pivoter un certificat client expirant à l'aide du AWS CLI

Vous pouvez vérifier la date d'expiration du certificat en appelant le client Certificate:by-id de l'API REST API Gateway ou en utilisant la AWS CLI commande get-client-certificate et en inspectant la propriété ExpirationDate renvoyée. https://docs.aws.amazon.com/apigateway/latest/api/API_ClientCertificate.html#expirationDate

Pour faire pivoter un certificat client, procédez comme suit :

1. Générez un nouveau certificat client en appelant la commande clientcertificate:generate de l'API REST API Gateway ou la commande generate-client-certificate de l' AWS CLI . 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"
       }
     ]
   }
   ```

   Vous pouvez également utiliser la commande update-stage.

   Si vous utilisez une WebSocket API, utilisez la commande `apigatewayv2` 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 client CertificateId (`a1b2c3`) de l'ancien certificat :

   ```
   DELETE /clientcertificates/a1b2c3
   ```

   Vous pouvez également appeler la commande delete-client-certificate :

```
aws apigateway delete-client-certificate --client-certificate-id a1b2c3
```

Autorités de certification prises en charge par API Gateway pour les intégrations HTTP et les intégrations de proxy HTTP

La liste suivante présente les autorités de certification prises en charge par API Gateway pour les intégrations HTTP, les intégrations de proxy HTTP 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
```

AWS WAF À utiliser pour protéger votre REST APIs dans API Gateway

AWS WAF est un pare-feu pour applications Web qui aide à protéger les applications Web APIs 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 Getting Started with AWS WAF et les listes de contrôle d'accès Web (Web ACLs).

   > **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'applications Web (AWS WAF), sélectionnez votre liste 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

La associate-web-aclcommande suivante associe une ACL AWS WAFV2 Web pour une application régionale à un stage d'API API Gateway existant :

```
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
```

La associate-web-aclcommande suivante associe une ACL AWS WAF Classic Regional Web à un stage d'API API Gateway existant :

```
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'
)
```

Limitez les requêtes adressées à votre REST APIs pour améliorer le débit dans API Gateway

Vous pouvez configurer la régulation et les quotas pour éviter qu'ils ne soient submergés par un trop grand nombre de demandes. APIs 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. Plus précisément, API Gateway examine le taux et le nombre de demandes soumises par rapport APIs à l'ensemble 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 cibles pour les différentes étapes ou méthodes d'API afin d'améliorer les APIs performances globales de l'ensemble de 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
- Limitation au niveau du compte par région
- Configuration des cibles de limitation au niveau de l'API et de l'étape dans un plan d'utilisation
- Configuration de cibles de limitation au niveau de l'étape
- Configuration des cibles de limitations au niveau de la méthode dans un plan d'utilisation

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 les types de paramètres de limitation pour votre API et la façon dont API Gateway les applique.

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 tous les APIs membres d'un compte dans une région spécifiée. La limite de débit au niveau du compte peut être augmentée sur demande. Des limites plus élevées sont possibles avec APIs des délais d'attente plus courts et des charges utiles plus faibles. Pour demander une augmentation des limitations au niveau du compte par Région, contactez le Centre de support AWS. Pour de plus amples informations, veuillez consulter 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 politique d'utilisation comme identifiant de client. Notez que ces limites ne peuvent pas être supérieures aux limites par compte.

API Gateway applique vos paramètres de limitation 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

Par défaut, API Gateway limite le nombre de demandes stables par seconde (RPS) sur l'ensemble APIs d'un AWS compte, par région. Cela limite également la rafale (c'est-à-dire la taille maximale du bucket) sur l'ensemble APIs du AWS compte, 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 SDKs, et le Console de gestion AWS pour créer un plan d'utilisation. Pour plus d'informations sur la création du plan d'utilisation, consultez ???.

Configuration de cibles de limitation au niveau de l'étape

Vous pouvez utiliser la AWS CLI et le SDKs Console de gestion AWS pour créer des cibles de régulation au niveau de l'étape.

Pour plus d'informations sur la façon d'utiliser le pour Console de gestion AWS 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 API Gateway

Une API privée est une API REST qui ne peut être appelée que d'un VPC Amazon. Vous pouvez accéder à votre API à l'aide d'un point de terminaison de VPC d'interface, qui est une interface réseau de point de terminaison que vous créez dans votre VPC. Les points de terminaison de l'interface sont AWS PrivateLink alimentés par une technologie qui vous permet d'accéder à des AWS services en toute confidentialité à l'aide d'adresses IP privées.

Vous pouvez également l'utiliser 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 à destination de votre API privée utilise des connexions sécurisées et est isolé de l'Internet public. Le trafic ne quitte pas le réseau Amazon.

Bonnes pratiques relatives aux API privées

Nous vous recommandons d'utiliser les bonnes pratiques ci-dessous lorsque vous créez votre API privée :

- Utilisez un point de terminaison de VPC unique pour accéder à plusieurs API privées. Cela permet de réduire le nombre de points de terminaison de VPC dont vous pourriez avoir besoin.
- Associez votre point de terminaison de VPC à votre API. Cela permet de créer un enregistrement DNS d'alias Route 53 et de simplifier l'invocation de votre API privée.
- Activez le DNS privé pour votre VPC. Lorsque vous activez le DNS privé pour votre VPC, vous pouvez invoquer votre API au sein d'un VPC sans transmettre l'en-tête `Host` ou `x-apigw-api-id`.

  Si vous activez le DNS privé, vous ne pouvez pas accéder au point de terminaison par défaut pour les API publiques. Pour accéder au point de terminaison par défaut pour les API publiques, vous pouvez désactiver le DNS privé, créer une zone hébergée privée pour chaque API privée de votre

VPC, puis configurer les enregistrements requis dans Route 53. Cela permet de résoudre votre API privée et de continuer à invoquer le point de terminaison public par défaut de votre VPC. Pour plus d'informations, consultez Création d'une zone hébergée privée.

- Restreignez l'accès de votre API privée à des VPC ou points de terminaison de VPC spécifiques. Ajoutez la condition `aws:SourceVpc` ou `aws:SourceVpce` à la politique de ressources de votre API pour restreindre l'accès.
- Pour sécuriser au maximum le périmètre de données, vous pouvez créer une politique de point de terminaison de VPC. Cela permet de contrôler l'accès aux points de terminaison de VPC qui peuvent invoquer 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.
- Vous ne pouvez pas convertir une API privée en une API optimisée pour la périphérie.
- Les API privées prennent uniquement en charge le protocole TLS 1.2. Les versions antérieures de TLS ne sont pas prises en charge.
- Si vous faites une demande en utilisant le HTTP/2 protocole, la demande est forcée pour utiliser HTTP/1.1 le protocole.
- Vous ne pouvez pas définir de type d'adresse IP pour les API privées afin d'autoriser uniquement les adresses IPv4 à invoquer votre API privée. Seule l'option Double pile est prise en charge. Pour de plus amples informations, veuillez consulter the section called "Types d'adresses IP pour les API REST dans API Gateway".
- Pour envoyer du trafic à l'aide de votre API privée, vous pouvez utiliser tous les types d'adresses IP pris en charge par Amazon VPC. Vous pouvez envoyer du trafic à double pile et IPv6 en configurant les paramètres de votre point de terminaison de VPC. Vous ne pouvez pas le modifier à l'aide d'API Gateway. Pour plus d'informations, consultez Ajout de la prise en charge d'IPv6 pour votre VPC.
- 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, voir Accéder 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 découvrir comment créer une API privée et associer un point de terminaison de VPC, consultez the section called “Création d’une API privée”. Pour suivre un didacticiel dans lequel vous créez des dépendances CloudFormation et une API privée dans le Console de gestion AWS, consultezthe section called “Didacticiel : création d’une API REST privée”.

Création d’une API privée

Avant de créer une API privée, vous allez créer un point de terminaison de VPC pour API Gateway. Ensuite, vous allez créer votre API privée et y attacher une politique de ressources. Si vous le souhaitez, vous pouvez associer votre point de terminaison de VPC à votre API privée pour simplifier l’invocation de votre API. Enfin, vous allez déployer votre API.

Les procédures suivantes décrivent la marche à suivre. Vous pouvez créer une API REST privée à l'aide du Console de gestion AWS, AWS CLI ou d'un AWS SDK.

Conditions préalables

Pour effectuer ces étapes, vous devez disposer d’un VPC entièrement configuré. Pour apprendre à créer un VPC, consultez 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. Ainsi, vous pouvez invoquer votre API au sein d’un VPC sans avoir à transmettre l’hôte ou l’en-tête `x-apigw-api-id`.

Pour activer le DNS privé, les attributs `enableDnsSupport` et `enableDnsHostnames` de votre VPC doivent être définis sur `true`. Pour plus d’informations, consultez Prise en charge du DNS dans votre VPC et Mise à jour de la prise en charge de DNS pour votre VPC.

Étape 1 : créer un point de terminaison de VPC pour API Gateway dans votre VPC

La procédure suivante montre comment créer un point de terminaison de 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 domaine `execute-api` est le service des composants API Gateway pour l’exécution de l’API.

Lorsque vous créez votre point de terminaison de VPC pour API Gateway, vous devez spécifier les paramètres DNS. Si vous désactivez le DNS privé, vous ne pouvez accéder à votre API que via le DNS public. Pour de plus amples informations, veuillez consulter the section called “Problème : je ne parviens pas à me connecter à mon API publique à partir d’un point de terminaison de VPC API Gateway”.

Console de gestion AWS

Pour créer un point de terminaison de VPC d'interface pour API Gateway

1. Connectez-vous au Console de gestion AWS et ouvrez la console Amazon VPC à https://console.aws.amazon.com/vpc/ l'adresse.
2. Dans le panneau de navigation, sous Cloud privé virtuel, choisissez Points de terminaison.
3. Choisissez Créer un point de terminaison.
4. (Facultatif) Pour Balise de nom, saisissez un nom permettant d'identifier votre point de terminaison de VPC.
5. Pour Service category (Catégorie de service), choisissez Services AWS .
6. Sous Services, dans la barre de recherche, saisissez **`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 devrait ressembler à `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 Activer le nom de DNS privé, choisissez Paramètres supplémentaires, puis décochez Activer le nom de DNS privé.
9. Pour Sous-réseaux, choisissez les zones de disponibilité dans lesquelles vous avez créé les interfaces réseau du point de terminaison. 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 Politique, effectuez l'une des actions suivantes :
    - Si vous n'avez pas créé votre API privée ou si vous ne souhaitez pas configurer de politique personnalisée de point de terminaison de VPC, choisissez Accès complet.
    - Si vous avez déjà créé une API privée et que vous souhaitez configurer une politique personnalisée de point de terminaison de VPC, vous pouvez saisir une politique personnalisée de point de terminaison de VPC. Pour de plus amples informations, veuillez consulter the section called "Utilisation de politiques de point de terminaison de VPC pour des API privées".

Vous pouvez mettre à jour la politique de point de terminaison de VPC après avoir créé votre point de terminaison de VPC. Pour plus d'informations, consultez Mise à jour d'une politique de point de terminaison de VPC.

12. Choisissez Créer un point de terminaison.
13. Copiez l'ID du point de terminaison de VPC obtenu. Vous pourriez en avoir besoin lors des étapes suivantes.

AWS CLI

La commande create-vpc-endpoint suivante crée un point de terminaison de 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 du point de terminaison de VPC obtenu. Vous pourriez en avoir besoin lors des étapes suivantes.

Étape 2 : créer une API privée

Après avoir créé votre point de terminaison de VPC, vous allez créer une API REST privée. La procédure suivante montre comment créer une API privée.

Console de gestion AWS

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 ID de points de terminaison de VPC, saisissez un ID de point de terminaison de VPC.

   Si vous associez un ID de point de terminaison de VPC à votre API privée, vous pouvez invoquer votre API depuis votre VPC sans remplacer l'en-tête `Host` ni transmettre l'en-tête `x-apigw-api-id header`. Pour plus d'informations, consultez the section called "(Facultatif) Association d'un point de terminaison de VPC à une API privée ou dissociation d'un point de terminaison de VPC d'une API privée".
8. Pour Type d'adresse IP, choisissez Double pile.
9. Sélectionnez Create API (Créer une API).

Après avoir effectué les étapes précédentes, vous pouvez suivre les instructions de la section the section called "Prise en main de la console d'API REST" pour configurer des méthodes et des intégrations pour cette API, mais vous ne pouvez pas déployer votre API. Pour déployer votre API, suivez l'étape 3 et attachez une politique de ressources à votre API.

AWS CLI

La commande create-rest-api suivante crée 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"], "ipAddressType":
 "dualstack" }'
```

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"
        ],
        "ipAddressType": "dualstack"
    },
    "id": "0qzs2sy7bh",
    "name": "Simple PetStore (AWS CLI, Private)"
```

```
}
```

Après avoir effectué les étapes précédentes, vous pouvez suivre les instructions de la section the section called “Tutoriel : Création d'une API REST à l'aide de AWS SDK ou AWS CLI” pour configurer des méthodes et des intégrations pour cette API, mais vous ne pouvez pas déployer votre API. Pour déployer votre API, suivez l’étape 3 et attachez 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 de la section the section called “Tutoriel : Création d'une API REST à l'aide de AWS SDK ou AWS CLI” pour configurer des méthodes et des intégrations pour cette API, mais vous ne pouvez pas déployer votre API. Pour déployer votre API, suivez l’étape 3 et attachez 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 de la section the section called “Tutoriel : Création d'une API REST à l'aide de AWS SDK ou AWS CLI” pour configurer des méthodes et des intégrations pour cette API, mais vous ne pouvez pas déployer votre API. Pour déployer votre API, suivez l’étape 3 et attachez 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 autoriser vos VPC et vos points de terminaison de VPC à accéder à vos API privées. Vous pouvez accorder l'accès à un point de terminaison VPC depuis n'importe quel AWS compte.

Votre politique de ressources doit contenir la condition `aws:SourceVpc` ou `aws:SourceVpce` pour restreindre l’accès. Nous vous recommandons d’identifier des VPC et des points de terminaison de VPC spécifiques et de ne pas créer de politique de ressources autorisant l’accès à tous les VPC et points de terminaison de VPC.

La procédure suivante montre comment attacher une politique de ressources à votre API.

Console de gestion AWS

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 Create Policy (Créer une politique).
5. Choisissez Sélectionnez un modèle, puis VPC source.
6. Remplacez `{{vpcID}}` (y compris les accolades) par l’ID de votre VPC.

7. Sélectionnez Enregistrer les modifications.

AWS CLI

La commande update-rest-api suivante attache une stratégie 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 de VPC. Pour contrôler quelles sont les ressources qui accèdent à votre point de terminaison de VPC, attachez une politique de point de terminaison de VPC à votre point de terminaison de VPC Pour de plus amples informations, veuillez consulter the section called "Utilisation de politiques de point de terminaison de VPC pour des API privées".

(Facultatif) Association d'un point de terminaison de VPC à une API privée ou dissociation d'un point de terminaison de VPC d'une API privée

Lorsque vous associez un point de terminaison de VPC à votre API privée, API Gateway génère un enregistrement DNS d'alias Route 53. Vous pouvez utiliser cet enregistrement pour invoquer vos API privées comme vous le faites avec vos API publiques sans remplacer un en-tête `Host` ni transmettre un en-tête `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 (Console de gestion AWS)

Vous pouvez associer un point de terminaison de VPC à votre API privée lorsque vous la créez ou après l'avoir créée. La procédure suivante montre comment associer un point de terminaison de VPC à une API déjà créée.

Pour associer un point de terminaison de 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 de VPC sélectionnez les ID de point de terminaison de VPC supplémentaires.
8. Choisissez Enregistrer.
9. Redéployez l'API pour que les modifications prennent effet.

Dissociate a VPC endpoint (Console de gestion AWS)

Pour dissocier un point de terminaison de 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 politique 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 commande create-rest-api suivante associe des points de terminaison de VPC lors 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 commande update-rest-api suivante associe des points de terminaison de 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 commande update-rest-api suivante dissocie un point de terminaison de 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 API et l'associer à une étape. La procédure suivante montre comment déployer votre API privée.

Console de gestion AWS

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 Déployer.

AWS CLI

La commande create-deployment suivante déploie une API privée :

```
aws apigateway create-deployment --rest-api-id a1b2c3 \
  --stage-name test \
  --stage-description 'Private API test stage' \
  --description 'First deployment'
```

Résolution des problèmes de votre API privée

La section suivante fournit des conseils pour résoudre les erreurs et les problèmes que vous pouvez rencontrer lorsque vous créez une API privée.

Problème : je ne parviens pas à me connecter à mon API publique à partir d'un point de terminaison de 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 ne pas activer 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 publique d'API Gateway à partir de votre point de terminaison de 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 la périphérie, 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 : pourquoi ne puis-je pas me connecter à mon API publique à partir d'un point de terminaison de VPC API Gateway ?

Problème : Mon API renvoie **{"Message » :"Utilisateur : anonymous n'est pas autorisé à exécuter : execute-API:Invoke on resource : arn:aws:execute-api:us-east-1 : ********/****/****/ "}**

Dans votre politique de ressources, si vous définissez le AWS principal sur un principal, comme suit :

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": [
                    "arn:aws:iam::111122223333:role/developer",
                    "arn:aws:iam::111122223333:role/Admin"
                ]
            },
            "Action": "execute-api:Invoke",
            "Resource": [
                "execute-api:/stage/GET/pets"
            ]
        }
    ]
}
```

Vous devez utiliser l'autorisation `AWS_IAM` pour chaque méthode de votre API, sinon votre API renvoie le message d'erreur précédent. Pour plus d'instructions sur l'activation de l'autorisation `AWS_IAM` pour une méthode, consultez the section called "Méthodes".

Problème : je ne sais pas si le point de terminaison de mon VPC est associé à mon API

Si vous associez un point de terminaison de VPC à votre API privée ou l'en dissociez, vous devez redéployer votre API. L'opération de mise à jour peut prendre quelques minutes du fait de la propagation DNS. Pendant ce temps, votre API est disponible, mais la propagation DNS pour les URL DNS nouvellement générées peut encore être en cours. Si après plusieurs minutes, vos nouvelles URL ne sont pas résolues dans DNS, nous vous recommandons de redéployer votre API.

Noms de domaine personnalisés pour les API privées dans API Gateway

Vous pouvez créer un nom de domaine personnalisé pour vos API privées. Utilisez un nom de domaine personnalisé privé pour fournir aux appelants d'API une URL plus simple et plus intuitive. Avec un nom de domaine personnalisé privé, vous pouvez réduire la complexité, configurer des mesures de sécurité lors de la prise de contact TLS et contrôler le cycle de vie du certificat de votre nom de domaine à l'aide de AWS Certificate Manager (ACM). Pour de plus amples informations, veuillez consulter the section called "Sécurisation de la clé privée du certificat pour votre nom de domaine personnalisé".

Les noms de domaine personnalisés des API privées n'ont pas besoin d'être propres à un compte. Vous pouvez créer `example.private.com` dans les comptes 111122223333 et 555555555555, à condition que votre certificat ACM couvre le nom de domaine. Pour identifier un nom de domaine personnalisé privé, utilisez l'ARN du nom de domaine en question. Cet identifiant est propre aux noms de domaine personnalisés privés.

Lorsque vous créez un nom de domaine personnalisé privé dans API Gateway, vous êtes un fournisseur d'API. Vous pouvez fournir votre nom de domaine personnalisé privé à d'autres personnes à Comptes AWS l'aide d'API Gateway ou AWS Resource Access Manager (AWS RAM).

Lorsque vous invoquez un nom de domaine personnalisé privé, vous êtes un utilisateur d'API. Vous pouvez utiliser un nom de domaine personnalisé privé provenant du vôtre Compte AWS ou d'un autre Compte AWS.

Lorsque vous utilisez un nom de domaine personnalisé privé, vous créez une association d'accès au nom de domaine entre le point de terminaison d'un VPC et un nom de domaine personnalisé privé. Avec une association d'accès au nom de domaine, les utilisateurs d'API peuvent invoquer votre nom de domaine personnalisé privé tout en étant isolés de l'Internet public. Pour de plus amples informations, veuillez consulter the section called "Fournisseurs d'API et utilisateurs d'API".

Sécurisation de la clé privée du certificat pour votre nom de domaine personnalisé

Lorsque vous demandez un SSL/TLS certificat à l'aide d'ACM pour créer votre nom de domaine personnalisé pour les API privées, ACM génère une paire de public/private clés. Lorsque vous importez un certificat, vous générez la paire de clés. La clé publique devient partie intégrante du certificat. Pour stocker la clé privée en toute sécurité, ACM crée une autre clé en utilisant AWS KMS, appelée clé KMS, avec l'alias aws/acm. AWS KMS utilise cette clé pour chiffrer la clé privée de votre certificat. Pour plus d'informations, consultez Protection des données dans AWS Certificate Manager dans le Guide de l'utilisateur AWS Certificate Manager .

API Gateway utilise le gestionnaire de connexions AWS TLS, un service accessible uniquement à Services AWS, pour sécuriser et utiliser les clés privées de votre certificat. Lorsque vous utilisez votre certificat ACM pour créer un nom de domaine personnalisé API Gateway, API Gateway associe votre certificat à AWS TLS Connection Manager. Pour ce faire, nous créons une subvention AWS KMS pour votre clé AWS gérée. Cette autorisation permet à TLS Connection Manager de AWS KMS déchiffrer la clé privée de votre certificat. TLS Connection Manager utilise le certificat et la clé privée déchiffrée (texte en clair) pour établir une connexion sécurisée (SSL/TLS session) avec les clients des services API Gateway. Lorsque le certificat est dissocié d'un service API Gateway, l'autorisation est supprimée. Pour plus d'informations, consultez Grants dans le Guide du développeur AWS Key Management Service .

Pour de plus amples informations, veuillez consulter the section called "Chiffrement au repos".

Considérations relatives aux noms de domaine personnalisés privés

Les considérations suivantes peuvent avoir un impact sur votre utilisation des noms de domaine personnalisés privés :

- API Gateway a besoin d'environ 15 minutes pour configurer votre nom de domaine personnalisé privé.
- Si vous mettez à jour votre certificat ACM, il faut environ 15 minutes à API Gateway pour effectuer la mise à jour. Pendant ce temps, votre nom de domaine est dans l'état `UPDATING` et vous pouvez toujours y accéder.
- Pour invoquer un nom de domaine personnalisé privé, vous devez créer une association d'accès au nom de domaine personnalisé. Une fois l'association d'accès au nom de domaine créée, il faut compter environ 15 minutes pour qu'elle soit prête.
- L'ARN du nom de domaine personnalisé privé contient le *`account-id`* et le*`domain-name-id`*. Lorsque vous créez un nom de domaine, API Gateway utilise le format ARN d'`arn:`*`partition`*`:apigateway:`*`region`*`::/domainnames/domain-name`. Lorsque

vous accédez à un nom de domaine personnalisé privé, vous utilisez le format ARN d'`arn:`*`partition`*`:apigateway:`*`region`*`:`*`account-id`*`:/domainnames/domain-name+`*`domain-name-id`*.

Vous devrez peut-être modifier vos autorisations IAM pour autoriser l'accès à un nom de domaine privé après l'avoir créé.

- Vous ne pouvez pas invoquer des noms de domaines personnalisés privés portant le même nom à partir du même point de terminaison de VPC. Par exemple, si vous souhaitez invoquer `arn:aws:apigateway:us-west-2:111122223333:/domainnames/private.example.com+abcd1234` et `arn:aws:apigateway:us-west-2:111122223333:/domainnames/private.example.com+xyz000`, associez chaque nom de domaine personnalisé privé à un point de terminaison de VPC différent.
- Les certificats génériques sont pris en charge, par exemple un certificat pour `*.private.example.com`.
- Les noms de domaine personnalisés génériques ne sont pas pris en charge.
- Seuls les certificats RSA avec une longueur de clé de 2 048 bits et les certificats ECDSA avec des longueurs de clé de 256 et 384 bits sont pris en charge.
- Vous ne pouvez pas définir de type d'adresse IP pour les API privées afin d'autoriser uniquement les adresses IPv4 à invoquer votre API privée. Seule l'option Double pile est prise en charge. Pour de plus amples informations, veuillez consulter the section called "Types d'adresses IP pour les API REST dans API Gateway".
- Pour envoyer du trafic à l'aide de votre API privée, vous pouvez utiliser tous les types d'adresses IP pris en charge par Amazon VPC. Vous pouvez envoyer du trafic à double pile et IPv6 en configurant les paramètres de votre point de terminaison de VPC. Vous ne pouvez pas le modifier à l'aide d'API Gateway. Pour plus d'informations, consultez Ajout de la prise en charge d'IPv6 pour votre VPC.
- Multi-level le mappage de chemin de base, tel que le mappage de votre API privée vers `developers/feature`, n'est pas pris en charge, mais vous pouvez utiliser une règle de routage pour créer une condition de chemin à plusieurs niveaux. Pour de plus amples informations, veuillez consulter the section called "Envoyez du trafic vers vous APIs via votre nom de domaine personnalisé dans API Gateway".
- Vous ne pouvez pas définir de version TLS minimale pour votre nom de domaine personnalisé privé. Tous les noms de domaine personnalisés privés ont la politique de sécurité de `TLS-1-2`.
- Vous pouvez utiliser une politique de point de terminaison de VPC pour contrôler l'accès à un nom de domaine personnalisé privé. Pour plus d'informations, consultez les exemples 4 et 5 de la

section the section called “Utilisation de politiques de point de terminaison de VPC pour des API privées”.

- Vous devez créer une politique de ressources distincte pour votre API privée et pour votre nom de domaine personnalisé privé. Pour invoquer un nom de domaine personnalisé privé, un utilisateur d’API doit avoir accès à la politique de ressources du nom de domaine personnalisé privé, à la politique de ressources de l’API privée et à toute politique ou autorisation de point de terminaison de VPC sur l’API privée.

Considérations relatives à l’utilisation de noms de domaine personnalisés privés avec d’autres ressources API Gateway

Les considérations suivantes peuvent avoir un impact sur la manière dont vous utilisez les noms de domaine personnalisés privés avec d’autres ressources API Gateway :

- Vous ne pouvez pas envoyer du traffic d’un nom de domaine personnalisé privé vers une API publique.
- Lorsqu’une API privée est mappée à un nom de domaine personnalisé privé, vous ne pouvez pas modifier le type de point de terminaison de l’API.
- Vous ne pouvez pas migrer un nom de domaine personnalisé public vers un nom de domaine personnalisé privé.
- Si vous utilisez un point de terminaison de VPC pour accéder à un nom de domaine personnalisé public, ne l’utilisez pas pour créer une association d’accès au nom de domaine avec un nom de domaine personnalisé privé.

Différences entre les noms de domaine personnalisés privés et les noms de domaine personnalisés publics

Les différences entre les noms de domaine personnalisés privés et publics sont décrites ci-dessous :

- Les noms de domaine personnalisés privés n’ont pas besoin d’être propres à un compte.
- Un nom de domaine privé possède un identifiant de nom de domaine. Cet ID identifie de manière unique un nom de domaine personnalisé privé. Il n’est pas généré pour les noms de domaine personnalisés publics.
- Lorsque vous utilisez le AWS CLI pour mettre à jour ou supprimer votre nom de domaine personnalisé privé, vous devez fournir l'ID du nom de domaine. Si vous possédez un nom de domaine personnalisé privé `example.com` et un nom de domaine public personnalisé

`example.com` et que vous ne fournissez pas l'ID du nom de domaine, API Gateway modifiera ou supprimera votre nom de domaine public personnalisé public.

Prochaines étapes pour les noms de domaine personnalisés pour les API privées

Pour plus d'informations sur les tâches d'un fournisseur d'API et d'un utilisateur d'API, consultez the section called "Fournisseurs d'API et utilisateurs d'API".

Pour obtenir des instructions sur la création d'un nom de domaine personnalisé privé que vous pouvez invoquer vous-même Compte AWS, consultezthe section called "Didacticiel : création et invocation d'un nom de domaine personnalisé pour les API privées".

Pour obtenir des instructions sur la fourniture Compte AWS d'un autre accès à votre nom de domaine personnalisé privé, consultezthe section called "Fournisseur d'API : partagez votre nom de domaine personnalisé privé en utilisant AWS RAM". Pour obtenir des instructions sur l'association de votre point de terminaison VPC à un nom de domaine personnalisé privé dans un autre Compte AWS, consultez. the section called "Utilisateur d'API : association de votre point de terminaison de VPC à un nom de domaine personnalisé privé partagé avec vous"

Tâches des fournisseurs d'API et des utilisateurs d'API pour les noms de domaine personnalisés pour les API privées

Lorsque vous créez un nom de domaine personnalisé privé, vous êtes un fournisseur d'API. Lorsque vous invoquez un nom de domaine personnalisé privé, vous êtes un utilisateur d'API. Vous pouvez utiliser un nom de domaine personnalisé privé provenant du vôtre Compte AWS ou d'un autre Compte AWS.

La section suivante explique les tâches requises par le fournisseur d'API et l'utilisateur d'API pour utiliser un nom de domaine personnalisé privé. Si vous souhaitez invoquer vous-même un nom de domaine personnalisé privé Compte AWS, vous êtes à la fois le fournisseur et le consommateur de l'API. Si vous souhaitez invoquer un domaine personnalisé privé dans un autre domaine Compte AWS, en fonction de la relation de confiance entre le fournisseur d'API et le consommateur d'API dans AWS Organizations, vous AWS RAM pouvez effectuer certaines tâches à votre place.

Tâches d'un fournisseur d'API

Les fournisseurs d'API créent des API privées et les mappent à des noms de domaine personnalisés.

Les fournisseurs d'API gèrent deux politiques de ressources pour protéger leurs noms de domaine personnalisés privés. La première politique concerne le service `execute-api` et contrôle les points

de terminaison de VPC qui peuvent invoquer votre nom de domaine personnalisé privé. Dans la configuration du nom de domaine personnalisé privé, elle est baptisée `policy`.

La deuxième politique concerne le service Amazon API Gateway Management et contrôle quels points de terminaison VPC dans d'autres pays Comptes AWS peuvent former une association d'accès au nom de domaine avec votre nom de domaine personnalisé privé. Un point de terminaison de VPC doit former une association d’accès au nom de domaine avec un nom de domaine personnalisé privé pour l’invoquer. Dans la configuration du nom de domaine personnalisé privé, cette politique est baptisée `managementPolicy`. Vous pouvez utiliser notre AWS RAM API Gateway pour mettre à jour cette politique. Si vous ne prévoyez pas d'autoriser les points de terminaison VPC dans un autre utilisateur Comptes AWS à invoquer votre nom de domaine personnalisé, vous ne modifiez pas le. `managementPolicy`

Si vous êtes un fournisseur d’API, vous devez :

1. Créer une API privée.
2. Mettre à jour la politique `policy` de votre API privée pour autoriser votre point de terminaison de VPC à accéder à votre API privée.
3. Créer un nom de domaine personnalisé privé.
4. Mettre à jour la politique `policy` de votre nom de domaine personnalisé privé pour autoriser votre point de terminaison de VPC à accéder à votre nom de domaine personnalisé privé.
5. Créer un mappage du chemin de base ou une règle de routage pour envoyer le trafic de votre API privée vers votre nom de domaine personnalisé privé. Pour de plus amples informations, veuillez consulter the section called “Envoyez du trafic vers vous APIs via votre nom de domaine personnalisé dans API Gateway”.

Si vous souhaitez autoriser les utilisateurs d'API d'autres utilisateurs Comptes AWS à accéder à votre nom de domaine personnalisé privé, procédez comme suit :

1. Mettez à jour la politique `managementPolicy` de votre nom de domaine personnalisé privé pour autoriser les utilisateurs d’API d’autres comptes à associer les points de terminaison de leurs VPC à votre nom de domaine personnalisé privé. Pour ce faire, utilisez les méthodes suivantes :

   AWS RAM

   Avec AWS RAM, si le fournisseur d'API et le consommateur d'API font partie de la même organisation qui les utilise AWS Organizations, le partage des ressources entre le fournisseur et le consommateur est automatiquement accepté. Dans le cas contraire, vous devez attendre

que l'utilisateur d'API accepte le partage de ressources. Nous vous recommandons de l'utiliser AWS RAM pour partager votre nom de domaine personnalisé privé.

API Gateway

Avec API Gateway, seul le AWS CLI est pris en charge. Vous devez mettre à jour votre nom de domaine personnalisé privé à l'aide d'une opération de correctif et fournir votre propre document de politique pour la politique `managementPolicy`.

2. Mettez à jour la politique `policy` de votre nom de domaine personnalisé privé et toutes les API privées qui y sont mappées pour autoriser l'accès au point de terminaison du VPC de l'utilisateur d'API.

Pour obtenir des instructions sur la façon de fournir votre API à une autre Compte AWS personne, consultezthe section called "Fournisseur d'API : partagez votre nom de domaine personnalisé privé en utilisant AWS RAM".

Tâches d'un utilisateur d'API

Les utilisateurs d'API associent les points de terminaison de leurs VPC à un nom de domaine ARN pour invoquer un nom de domaine personnalisé privé. Les utilisateurs d'API n'ont pas besoin de créer d'API API Gateway.

Si vous êtes un utilisateur d'API, procédez comme suit :

1. Créez un point de terminaison de VPC avec le DNS privé activé dans Amazon VPC.
2. (Facultatif, s' AWS RAM il est utilisé) Acceptez un partage de ressources de domaine personnalisé privé AWS RAM dans les 12 heures suivant le partage de ressources. Si vous et le fournisseur d'API appartenez à la même organisation, le partage des ressources est automatiquement accepté.
3. Obtenez l'ARN du nom de domaine personnalisé privé. L'URL du nom de domaine personnalisé privé n'étant pas unique, vous allez utiliser l'ARN du nom de domaine personnalisé privé pour former l'association d'accès au nom de domaine entre votre point de terminaison de VPC et le nom de domaine personnalisé privé. Vous pouvez l'utiliser AWS RAM pour récupérer l'ARN du nom de domaine personnalisé privé.
4. Associez l'ARN du nom de domaine personnalisé privé à votre point de terminaison de VPC dans API Gateway. Cette opération permet de créer une connexion sécurisée entre votre point de terminaison de VPC et le nom de domaine personnalisé privé. Le trafic ne quitte pas le réseau Amazon.

5. Attendez que le fournisseur d'API autorise votre point de terminaison de VPC à accéder au nom de domaine personnalisé privé et à toutes les API privées mappées au nom de domaine personnalisé privé. Si vous êtes à la fois le fournisseur et l'utilisateur d'API, vous autorisez votre point de terminaison de VPC à invoquer l'accès.
6. Créez une zone hébergée privée Route 53 et un enregistrement Route 53 pour résoudre le nom de domaine personnalisé privé dans Route 53.

Pour savoir comment utiliser une API dans une autre Compte AWS, consultezthe section called "Utilisateur d'API : association de votre point de terminaison de VPC à un nom de domaine personnalisé privé partagé avec vous".

Didacticiel : création et invocation d'un nom de domaine personnalisé pour les API privées

Dans ce didacticiel, vous allez créer un nom de domaine personnalisé privé que vous pouvez invoquer dans un VPC de votre propre compte. Pour ce faire, vous êtes le fournisseur d'API et l'utilisateur d'API. Vous avez besoin d'une API privée et d'un point de terminaison de VPC existants pour effectuer ce didacticiel. Si vous utilisez un point de terminaison de VPC pour accéder à un nom de domaine personnalisé public, ne l'utilisez pas pour ce didacticiel ou pour créer une association d'accès à un nom de domaine.

Étape 1 : créer un nom de domaine personnalisé privé

Vous allez créer votre nom de domaine personnalisé privé en spécifiant le nom de domaine, le certificat ACM et la politique du service `execute-api` pour contrôler les points de terminaison de VPC qui peuvent l'invoquer.

Console de gestion AWS

Pour créer un nom de domaine personnalisé privé

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Dans le panneau de navigation, sélectionnez Noms de domaine personnalisés.
3. Choisissez Ajouter un nom de domaine.
4. Pour Nom de domaine, entrez un nom de domaine.

   Votre certificat ACM doit couvrir ce nom de domaine, mais il n'est pas nécessaire que celui-ci soit unique.

5. Sélectionnez Privé.
6. Sous Mode de routage, choisissez Mappages d'API uniquement.
7. Pour Certificat ACM, sélectionnez un certificat.
8. Choisissez Ajouter un nom de domaine.

API Gateway fournit un nom de domaine avec une politique `deny` qui refuse toutes les ressources. Il s'agit de la politique de ressources pour le service `execute-api`. Vous devez mettre à jour cette politique de ressources afin d'accorder l'accès à vos points de terminaison de VPC pour invoquer votre nom de domaine personnalisé privé.

Pour mettre à jour votre politique de ressources

1. Choisissez l'onglet Stratégie de ressources, puis choisissez Modifier la stratégie de ressources.
2. Saisissez la politique de ressources suivante dans l'éditeur de code. Remplacez le point de terminaison VPC *`vpce-abcd1234efg`* par votre propre ID de point de terminaison VPC.

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

```
            }
        }
    ]
}
```

3. Sélectionnez Enregistrer les modifications.

AWS CLI

Lorsque vous créez un nom de domaine personnalisé privé à l'aide de AWS CLI, vous fournissez une politique de ressources pour que le `execute-api` service accorde l'accès aux points de terminaison VPC afin d'invoquer votre nom de domaine personnalisé privé, à l'aide du paramètre. `--policy file://policy.json` Vous pourrez modifier cette politique ultérieurement.

Dans cet exemple, vous allez attacher la politique de ressources suivante en tant que `policy` en chargeant les paramètres d'un fichier. Copiez et enregistrez ce fichier sous `policy.json`. Cette politique autorise uniquement le trafic entrant vers un nom de domaine personnalisé privé depuis le point de terminaison *`vpce-abcd1234efg`* VPC :

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

```
            }
        }
    ]
}
```

La commande create-domain-name suivante crée un nom de domaine personnalisé privé :

```
aws apigateway create-domain-name \
    --domain-name 'private.example.com' \
    --certificate-arn 'arn:aws:acm:us-west-2:111122223333:certificate/
a1b2c3d4-5678-90ab-cdef' \
    --security-policy 'TLS_1_2' \
    --endpoint-configuration '{"types":["PRIVATE"]}' \
    --policy file://policy.json
```

La sortie se présente comme suit :

```
{
    "domainName": "private.example.com",
    "domainNameId": "abcd1234",
    "domainNameArn": "arn:aws:apigateway:us-west-2:111122223333:/domainnames/
private.example.com+abcd1234",
    "certificateArn": "arn:aws:acm:us-west-2:111122223333:certificate/
a1b2c3d4-5678-90ab-cdef",
    "certificateUploadDate": "2024-09-10T10:31:20-07:00",
    "endpointConfiguration": {
        "types": [
            "PRIVATE"
        ]
    },
    "domainNameStatus": "AVAILABLE",
    "securityPolicy": "TLS_1_2",
    "routingMode" : "API_MAPPING_ONLY",
    "policy": "..."
}
```

Étape 2 : créer un mappage de chemin de base pour mapper votre API privée à votre nom de domaine personnalisé privé

Après avoir créé votre nom de domaine personnalisé privé, vous devez le mapper à une API privée. Le mappage de chemin de base rend une API accessible en combinant le nom de domaine personnalisé privé et un chemin de base associé. Nous vous recommandons d'utiliser un seul nom de domaine personnalisé privé comme nom d'hôte de plusieurs API privées.

Tous les fournisseurs d'API doivent créer un mappage de chemin de base, même si vous n'avez pas l'intention d'invoquer votre propre API. Vous devez également autoriser l'accès aux points de terminaison de VPC pour invoquer toutes les API privées que vous mappez à votre nom de domaine personnalisé privé.

Console de gestion AWS

Pour créer un mappage de chemin de base

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Dans le panneau de navigation, sélectionnez Noms de domaine personnalisés.
3. Choisissez un nom de domaine personnalisé privé.
4. Dans l'onglet Mappages d'API, choisissez Configurer les mappages.
5. Choisissez Ajouter un nouveau mappage.
6. Entrez une API, une Étape et, éventuellement, un Chemin d'accès.
7. Choisissez Enregistrer.

AWS CLI

La commande create-base-path-mapping suivante crée un mappage entre une API privée et un nom de domaine personnalisé privé :

```
aws apigateway create-base-path-mapping \
    --domain-name-id abcd1234 \
    --domain-name 'private.example.com' \
    --rest-api-id a1b2c3 \
    --stage prod \
    --base-path v1
```

Le résultat se présente comme suit :

```
{
    "basePath": "v1",
    "restApiId": "a1b2c3",
    "stage": "prod"
}
```

Pour plus de flexibilité dans l'acheminement du trafic vers vos API, vous pouvez remplacer le mode de routage par `ROUTING_RULE_ONLY` ou `ROUTING_RULE_THEN_API_MAPPING` et créer une règle de routage. Pour de plus amples informations, veuillez consulter the section called "Envoyez du trafic vers vous APIs via votre nom de domaine personnalisé dans API Gateway".

> **Note**
>
> Si vous souhaitez que d'autres Comptes AWS personnes invoquent votre nom de domaine personnalisé privé, une fois ce didacticiel terminé, suivez les étapes décrites dansthe section called "Fournisseur d'API : partagez votre nom de domaine personnalisé privé en utilisant AWS RAM".

Étape 3 : créer une association d'accès au nom de domaine entre votre nom de domaine personnalisé et un point de terminaison de VPC

Ensuite, vous allez créer une association d'accès au nom de domaine entre votre nom de domaine personnalisé et un point de terminaison de VPC. Votre point de terminaison de VPC utilise l'association d'accès au nom de domaine pour invoquer votre nom de domaine personnalisé privé tout en étant isolé de l'Internet public.

Console de gestion AWS

Pour créer une association d'accès au nom de domaine

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Dans le panneau de navigation, sélectionnez Noms de domaine personnalisés.
3. Choisissez un nom de domaine personnalisé privé.

4. Dans l'onglet Partage des ressources, pour Associations d'accès aux noms de domaine, choisissez Créer une association d'accès aux noms de domaine.
5. Pour ARN du nom de domaine, sélectionnez votre nom de domaine.
6. Pour ID de point de terminaison de VPC, sélectionnez l'ID de point de terminaison de VPC auquel vous avez donné accès à l'étape 1.
7. Choisissez Association d'accès aux noms de domaine.

Vous pouvez également créer votre association d'accès au nom de domaine à l'aide de la page Associations d'accès aux noms de domaine de la console.

AWS CLI

La commande `create-domain-name-access-association` suivante crée une association d'accès au nom de domaine entre votre nom de domaine personnalisé privé et votre point de terminaison de VPC.

```
aws apigateway create-domain-name-access-association \
    --domain-name-arn arn:aws:apigateway:us-west-2:111122223333:/domainnames/
private.example.com+abcd1234 \
    --access-association-source vpce-abcd1234efg \
    --access-association-source-type VPCE \
    --region us-west-2
```

Le résultat se présente comme suit :

```
{
    "domainNameAccessAssociationARN": "arn:aws:apigateway:us-west-2:111122223333:/
domainnameaccessassociations/domainname/private.example.com+abcd1234/vpcesource/
vpce-abcd1234efg",
    "accessAssociationSource": "vpce-abcd1234efg",
    "accessAssociationSourceType": "VPCE",
    "domainNameARN" : "arn:aws:apigateway:us-west-2:111122223333:/domainnames/
private.example.com+abcd1234"
}
```

Une fois l'association d'accès au nom de domaine créée, il faut compter environ 15 minutes pour qu'elle soit prête. Pendant que vous patientez, vous pouvez suivre les étapes suivantes.

Étape 4 : créer une zone hébergée Route 53

Après avoir mis à jour votre politique de ressources et associé votre nom de domaine personnalisé privé à votre point de terminaison de VPC, vous allez créer une zone hébergée privée dans Route 53 pour résoudre votre nom de domaine personnalisé. Une zone hébergée est un conteneur qui comporte des informations sur la façon dont vous souhaitez acheminer le trafic pour un domaine dans un ou plusieurs VPC, sans exposer vos ressources à Internet. Pour plus d'informations, veuillez onsulter Utilisation des zones hébergées privées.

Console de gestion AWS

Pour l'utiliser Console de gestion AWS, consultez la section Création d'une zone hébergée privée dans le guide du développeur Amazon Route 53.

Pour Nom, utilisez le nom de votre nom de domaine personnalisé privé. Pour ID de VPC, utilisez le VPC contenant le point de terminaison de VPC que vous avez utilisé lors des étapes précédentes.

AWS CLI

La commande create-hosted-zone suivante crée une zone hébergée privée :

```
aws route53 create-hosted-zone --name private.example.com \
    --caller-reference 2014-04-01-18:47 \
    --hosted-zone-config Comment="command-line version",PrivateZone=true \
    --vpc VPCRegion=us-west-2,VPCId=vpc-abcd1234
```

La sortie contient l'ID de la zone hébergée. Vous allez l'utiliser lors des étapes suivantes.

Étape 5 : créer un enregistrement DNS Route 53

Après avoir créé la zone hébergée, vous allez créer un enregistrement pour résoudre votre nom de domaine personnalisé privé. Vous allez utiliser l'ID de la zone hébergée que vous avez créée à l'étape précédente. Dans cet exemple, vous allez créer un type d'enregistrement A. Si vous utilisez IPv6 pour votre point de terminaison de VPC, créez un type d'enregistrement AAAA. Si vous utilisez une double pile pour votre point de terminaison de VPC, créez un enregistrement de type AAAA et un type d'enregistrement A.

Console de gestion AWS

Pour l'utiliser Console de gestion AWS, consultez la section Routage du trafic vers une API Amazon API Gateway à l'aide de votre nom de domaine.

Utilisez Création rapide et activez Alias. Pour le point de terminaison, utilisez le nom DNS du point de terminaison du VPC.

AWS CLI

Pour configurer vos enregistrements DNS de manière à ce qu'ils mappent le nom de domaine personnalisé privé au nom d'hôte de l'ID de la zone hébergée, vous allez créer un fichier JSON qui contient la configuration pour créer un enregistrement DNS pour le nom de domaine privé.

L'enregistrement `setup-dns-record.json` suivant montre comment créer un enregistrement DNS A pour mapper un nom de domaine personnalisé privé à son nom d'hôte privé. Vous allez fournir le nom DNSName de l'ID DNS de votre VPC et l'ID de la zone hébergée que vous avez créée à l'étape précédente.

```
{
  "Changes": [
    {
      "Action": "UPSERT",
      "ResourceRecordSet": {
        "Name": "private.example.com",
        "Type": "A",
        "AliasTarget": {
          "DNSName": "vpce-abcd1234.execute-api.us-west-2.vpce.amazonaws.com",
          "HostedZoneId": "Z2OJLYMUO9EFXC",
          "EvaluateTargetHealth": false
        }
      }
    }
  ]
}
```

La commande change-resource-record-sets suivante crée un enregistrement DNS pour votre nom de domaine personnalisé privé :

```
aws route53 change-resource-record-sets \
    --hosted-zone-id ZABCDEFG1234 \
```

```
    --change-batch file://path/to/your/setup-dns-record.json
```

Remplacez `hosted-zone-id` par l'ID de la zone hébergée Route 53 du jeu d'enregistrements DNS de votre compte. La valeur du paramètre `change-batch` pointe vers un fichier JSON.

Si vous n'avez pas l'intention d'invoquer votre propre nom de domaine personnalisé privé, vous pouvez supprimer ces ressources, après avoir vérifié que votre nom de domaine personnalisé privé fonctionne.

Étape 6 : invoquer votre nom de domaine personnalisé privé

Vous pouvez désormais invoquer votre nom de domaine personnalisé privé dans votre propre Compte AWS. Dans votre VPC, utilisez la commande curl suivante pour accéder à votre nom de domaine personnalisé privé.

```
curl https://private.example.com/v1
```

Pour découvrir d'autres façons d'invoquer votre API privée, consultez the section called "Invocation d'une API privée à l'aide d'un nom de domaine personnalisé".

Étape 7 : nettoyer

Pour éviter des coûts inutiles, supprimez l'association entre votre point de terminaison de VPC et votre nom de domaine personnalisé privé, puis supprimez votre nom de domaine personnalisé privé.

Console de gestion AWS

Pour supprimer l'association d'accès au nom de domaine

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Dans le panneau de navigation principal, sélectionnez Associations d'accès aux noms de domaine.
3. Sélectionnez votre association d'accès au nom de domaine, puis choisissez Supprimer.
4. Confirmez votre choix, puis choisissez Supprimer.

Après avoir supprimé votre association d'accès au nom de domaine, vous pouvez supprimer votre nom de domaine personnalisé privé.

Pour supprimer votre nom de domaine personnalisé privé

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Dans le panneau de navigation, sélectionnez Noms de domaine personnalisés.
3. Choisissez votre nom de domaine personnalisé privé.
4. Sélectionnez Delete (Supprimer).
5. Confirmez votre choix, puis choisissez Supprimer.

Si nécessaire, vous pouvez également supprimer votre point de terminaison de VPC. Pour plus d'informations, veuillez consulter la section Suppression d'un point de terminaison d'interface.

AWS CLI

Pour nettoyer

1. La commande `delete-access-association` suivante supprime l'association d'accès au nom de domaine :

```
aws apigateway delete-domain-name-access-association \
    --domain-name-access-association-arn 'arn:aws:apigateway:us-
west-2:111122223333:/domainnameaccessassociations/domainname/private.example.com
+abcd1234/vpcesource/vpce-abcd1234efg' \
    --region us-west-2
```

2. La commande `delete-domain-name` suivante supprime votre nom de domaine personnalisé privé. Cette commande supprime également tous les mappages de chemins de base.

```
aws apigateway delete-domain-name \
    --domain-name test.private.com \
    --domain-name-id abcd1234
```

Si nécessaire, vous pouvez également supprimer votre point de terminaison de VPC. Pour plus d'informations, veuillez consulter la section Suppression d'un point de terminaison d'interface.

Bonnes pratiques

Nous vous recommandons d'utiliser les bonnes pratiques ci-dessous lorsque vous créez votre nom de domaine personnalisé privé :

- Utilisez un mappage de chemin de base ou des règles de routage pour envoyer le trafic d'un nom de domaine personnalisé privé vers plusieurs API privées.
- Lorsqu'un point de terminaison de VPC n'a plus besoin d'accéder à un nom de domaine personnalisé privé, supprimez l'association. En outre, supprimez le point de terminaison de VPC de la politique `policy` pour le service `execute-api` pour le domaine personnalisé privé.
- Configurez au moins deux zones de disponibilité par point de terminaison de VPC.
- Désactivez le point de terminaison par défaut. Nous vous recommandons de désactiver le point de terminaison par défaut pour permettre à vos clients d'appeler votre API à partir du nom de domaine personnalisé uniquement. Pour de plus amples informations, veuillez consulter the section called "Désactivation du point de terminaison par défaut".
- Nous vous recommandons de configurer une zone hébergée privée Route 53 et un A-type enregistrement lorsque vous configurez votre nom de domaine personnalisé privé. Si vous n'avez pas l'intention d'invoquer votre propre nom de domaine personnalisé privé, vous pouvez supprimer ces ressources ultérieurement.

Utilisation de noms de domaine personnalisés privés intercomptes

Cette section explique comment utiliser des noms de domaine personnalisés intercomptes. Vous pouvez fournir un nom de domaine personnalisé privé à un autre Compte AWS et en utiliser un autre Compte AWS pour invoquer un nom de domaine personnalisé privé.

Vous pouvez partager votre nom de domaine personnalisé privé Compte AWS avec un autre utilisateur à l'aide de AWS Resource Access Manager notre API Gateway. AWS Resource Access Manager (AWS RAM) vous aide à partager vos ressources en toute sécurité entre Comptes AWS et au sein de votre organisation ou de vos unités organisationnelles (UO). Pour plus d'informations, voir Qu'est-ce que c'est AWS Resource Access Manager ?

Pour savoir comment partager un nom de domaine personnalisé privé Compte AWS avec un autre utilisateur AWS RAM, consultezthe section called "Fournisseur d'API : partagez votre nom de domaine personnalisé privé en utilisant AWS RAM".

Pour savoir comment partager un nom de domaine personnalisé privé avec un autre à Compte AWS l'aide d'API Gateway, consultezthe section called “Fournisseur d'API : partagez votre nom de domaine personnalisé privé à l'aide de l'API Gateway AWS CLI”.

Pour savoir comment utiliser un nom de domaine personnalisé privé dans un autre Compte AWS, consultezthe section called “Utilisateur d’API : association de votre point de terminaison de VPC à un nom de domaine personnalisé privé partagé avec vous”.

Bonnes pratiques relatives à l’utilisation de noms de domaine personnalisés intercomptes

Nous recommandons les bonnes pratiques ci-dessous concernant l’utilisation de noms de domaine personnalisés privés intercomptes :

- AWS RAM Utilisez-le pour partager vos noms de domaine personnalisés privés. Lorsque vous l'utilisez AWS RAM, vous pouvez réduire les frais opérationnels et vous n'avez pas à créer un `managementPolicy` pour le service Amazon API Gateway Management.
- Utilisez le paramètre `resource-owner` lorsque vous recensez vos noms de domaine personnalisés privés ou vos associations d’accès aux noms de domaine. Utilisez le paramètre `resource-owner` pour ne recenser que les ressources vous appartenant ou appartenant à d’autres Comptes AWS.

  L’exemple suivant montre comment obtenir toutes les associations d’accès aux noms de domaine que vous possédez :

  ```
  aws apigateway get-domain-name-access-associations --resource-owner SELF
  ```

  Utilisez `--resource-owner OTHER_ACCOUNTS` pour recenser toutes les associations d’accès au nom de domaine que d’autres comptes ont formées avec votre nom de domaine personnalisé privé.

Fournisseur d'API : partagez votre nom de domaine personnalisé privé en utilisant AWS RAM

Vous pouvez fournir aux utilisateurs de l'API Comptes AWS un autre accès à votre nom de domaine personnalisé privé. Dans cette section, vous apprendrez comment partager votre nom de domaine personnalisé privé à l'aide de votre nom de domaine personnalisé privé AWS RAM et comment contrôler l'accès à celui-ci.

Considérations relatives au partage de votre nom de domaine personnalisé privé

Les considérations suivantes peuvent impacter la façon dont vous autorisez l'accès à votre nom de domaine personnalisé privé à l'aide d' AWS RAM. Pour savoir comment partager votre nom de domaine personnalisé privé sans l'utiliser AWS RAM, consultezthe section called "Fournisseur d'API : partagez votre nom de domaine personnalisé privé à l'aide de l'API Gateway AWS CLI".

- Les noms de domaine personnalisés privés sont partagés au Région AWS niveau. Le nom de domaine personnalisé privé et le point de terminaison de VPC doivent se trouver dans la même Région AWS.
- Vous pouvez utiliser un partage de ressources avec plusieurs principaux, et après avoir créé le partage de ressources, vous pouvez y ajouter d'autres principaux. Nous vous recommandons, dans la mesure du possible, de réutiliser votre partage de ressources.
- Vous devez toujours autoriser le point de terminaison de VPC de l'utilisateur d'API à invoquer votre nom de domaine personnalisé privé et toutes les API privées qui y sont mappées.
- Si le consommateur d'API et le fournisseur d'API font partie de la même organisation qui les utilise AWS Organizations, le partage des ressources est automatiquement accepté. Vous devez tout de même créer le partage de ressources à l'aide d' AWS RAM.
- Si le consommateur d'API et le fournisseur d'API font partie de la même organisation AWS Organizations et que le partage des ressources au sein de votre organisation est activé, tous les principaux membres de l'organisation avec laquelle vous partagez ont automatiquement accès aux partages de ressources. Aucune invitation n'est nécessaire et vous pouvez ignorer le partage des ressources.
- Si l'utilisateur d'API n'accepte pas le partage de ressources dans les 12 heures, le fournisseur d'API doit à nouveau partager la ressource.
- Après avoir créé le partage de ressources, AWS RAM mettez à jour le `managementPolicy` service Amazon API Gateway Management pour votre nom de domaine personnalisé privé afin d'empêcher l'accès aux principaux sans `allow` accès explicite. Pour plus d'informations, consultez Determining whether a request is allowed or denied within an account dans le Guide de l'utilisateur IAM.

  La politique `managementPolicy` mise à jour ressemble à ce qui suit :

  JSON

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

```
    "Id": "abcd1234-1234-abcd-abcd-1234abcdefg",
    "Statement": [
        {
            "Sid":
 "APIGatewayPrivateDomainNameManagementPolicyDefaultPermission-org",
            "Effect": "Allow",
            "Principal": "*",
            "Action": "apigateway:CreateAccessAssociation",
            "Resource": "arn:aws:apigateway:us-west-2:111122223333:/
domainnames/private.example.com+abcd1234",
            "Condition": {
                "StringEquals": {
                    "aws:PrincipalOrgID": "o-1234abcd"
                },
                "StringNotEquals": {
                    "aws:PrincipalAccount": "111122223333"
                }
            }
        }
    ]
}
```

AWS RAM a empêché les mandants sans `allow` accès explicite de créer des associations d'accès avec votre nom de domaine personnalisé privé, en ajoutant ce qui suit :

```
"StringNotEquals": {
    "aws:PrincipalAccount": "111122223333"
}
```

Vous pouvez toujours utiliser le principal dans la section Compte AWS qui a créé le nom de domaine personnalisé privé pour créer des associations d'accès au nom de domaine.

Autorisation d’autres comptes à créer des associations d’accès au nom de domaine avec votre nom de domaine personnalisé privé

Tout d'abord, vous accordez l'accès Compte AWS à un autre pour créer des associations d'accès au nom de domaine avec votre nom de domaine personnalisé privé.

Console de gestion AWS

Pour utiliser le Console de gestion AWS, reportez-vous à la section Création d'un partage de ressources AWS RAM dans le Guide de AWS RAM l'utilisateur.

Pour Sélectionner le type de ressource, choisissez Domaines personnalisés privés API Gateway.

AWS CLI

La commande create-resource-share suivante crée un partage de ressources pour votre nom de domaine personnalisé privé. L'association entre la ressource et le mandataire peut prendre quelques minutes. Pour les principaux, fournissez un ID de compte ou un ID d'organisation, comme `arn:aws:organizations::123456789012:organization/o-1234abcd`. Vous pouvez fournir plusieurs principaux pour votre partage de ressources.

```
aws ram create-resource-share \
    --region us-west-2 \
    --name privateCustomDomain-resource-share \
    --permission-arns arn:aws:ram::aws:permission/
APIGatewayPrivateDomainNameManagementPolicyDefaultPermission \
    --resource-arns arn:aws:apigateway:us-west-2:111122223333:/domainnames/
private.example.com+abcd1234 \
    --principals 222222222222
```

Une fois que vous avez fourni l'accès à un autre compte Compte AWS, les utilisateurs d'API de ce compte doivent créer une association d'accès au nom de domaine entre leur point de terminaison VPC et votre nom de domaine personnalisé privé. Vous ne pouvez pas la créer pour eux. Pour de plus amples informations, veuillez consulter the section called “Association de votre point de terminaison de VPC à un nom de domaine personnalisé privé partagé”.

Autorisation d'autres comptes à invoquer votre nom de domaine personnalisé privé

Ensuite, vous allez autoriser le point de terminaison de VPC de l'utilisateur d'API à invoquer votre nom de domaine personnalisé privé et toutes les API privées qui y sont mappées.

Console de gestion AWS

Pour autoriser les points de terminaison de VPC d'autres comptes à invoquer votre nom de domaine personnalisé privé

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Dans le panneau de navigation, sélectionnez Noms de domaine personnalisés.
3. Choisissez le nom de domaine personnalisé privé que vous avez partagé avec d'autres personnes Comptes AWS.
4. Dans l'onglet Stratégie de ressources, choisissez Modifier la stratégie de ressources.
5. Ajoutez l'ID de point de terminaison de VPC de l'utilisateur d'API à votre politique de ressources.

   Vous trouverez l'ID de point de terminaison de VPC de l'utilisateur d'API dans la section Associations d'accès aux noms de domaine de l'onglet Partage des ressources sur la page Détails du domaine de votre nom de domaine personnalisé privé.
6. Sélectionnez Enregistrer les modifications.

AWS CLI

La politique `policy` suivante pour le service `execute-api` autorise le trafic entrant vers un nom de domaine personnalisé privé depuis les points de terminaison de VPC *`vpce-abcd1234efg`* et *`vpce-xyz000abc`*.

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

```
            "Principal": "*",
            "Action": "execute-api:Invoke",
            "Resource": [
                "execute-api:/*"
            ],
            "Condition" : {
                "StringNotEquals": {
                    "aws:SourceVpce": [
                    "vpce-abcd1234",
                    "vpce-xyzz0000"
                    ]
                }
            }
        }
    ]
}
```

La commande update-domain-name suivante utilise une opération de correctif pour mettre à jour la politique `policy` pour un nom de domaine personnalisé privé :

```
aws apigateway update-domain-name
    --domain-name private.example.com \
    --domain-name-id abcd1234 \
    --patch-operations op=replace,path=/policy,value='"{\"Version\": \"2012-10-17\",
       \"Statement\": [{\"Effect\": \"Allow\",\"Principal\": \"*\",\"Action\":
 \"execute-api:Invoke\",\"Resource\":[\"execute-api:/*\"]},{\"Effect\": \"Deny\",
\"Principal\": \"*\",\"Action\": \"execute-api:Invoke\",\"Resource\":[\"execute-
api:/*\"],\"Condition\":{\"StringNotEquals\":[\"vpce-abcd1234efg\", \"vpce-xyz000abc
\"]}}}]}"
```

Fournisseur d'API : arrêtez de partager un nom de domaine personnalisé privé en utilisant AWS RAM

Pour arrêter de partager votre nom de domaine personnalisé privé, vous devez d'abord empêcher l'utilisateur d'API de créer d'autres associations d'accès au nom de domaine en dissociant le partage de ressources. Ensuite, vous devez rejeter l'association d'accès au nom de domaine et supprimer le point de terminaison de VPC du client d'API de votre politique `policy` pour le service `execute-api`. L'utilisateur d'API peut ensuite supprimer son association d'accès au nom de domaine.

Arrêt du partage de votre nom de domaine personnalisé privé

Tout d'abord, vous arrêtez d'utiliser le partage de ressources AWS RAM.

Console de gestion AWS

Pour utiliser le Console de gestion AWS, voir Mettre à jour un partage de ressources dans AWS RAM.

AWS CLI

La méthode disassociate-resource-share suivante dissocie un partage de ressources pour votre nom de domaine personnalisé privé.

```
aws ram disassociate-resource-share \
    --region us-west-2 \
    --resource-arns arn:aws:apigateway:us-west-2:111122223333:/domainnames/
private.example.com+abcd1234 \
    --principals 222222222222
```

Refus de l'association d'accès au nom de domaine

Lorsque vous arrêtez de partager votre ressource en utilisant AWS RAM, vous rejetez l'association d'accès au nom de domaine entre un point de terminaison VPC d'un autre compte et votre nom de domaine personnalisé privé.

> **Note**
>
> Vous ne pouvez pas refuser une association d'accès au nom de domaine de votre propre compte. Pour arrêter le partage de ressources, supprimez l'association d'accès au nom de domaine. Pour plus d'informations, consultez Delete a domain name access association.

Lorsque vous rejetez une association d'accès au nom de domaine avec un point de terminaison de VPC, si un utilisateur d'API essaie d'appeler votre nom de domaine personnalisé privé, API Gateway rejette l'appel et renvoie le code de statut `403`.

Console de gestion AWS

Pour refuser une association d'accès au nom de domaine

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Dans le panneau de navigation, sélectionnez Noms de domaine personnalisés.

3. Choisissez le nom de domaine personnalisé privé que vous avez partagé avec d'autres personnes Comptes AWS.
4. Sous Partage de ressources, choisissez l'association d'accès au nom de domaine que vous souhaitez refuser.
5. Choisissez Refuser l'association.
6. Confirmez votre choix, puis choisissez Refuser.

AWS CLI

La commande `reject-domain-name-access-association` suivante refuse l'association d'accès au nom de domaine entre le point de terminaison de VPC et votre nom de domaine personnalisé privé :

```
aws apigateway reject-domain-name-access-association \
    --domain-name-access-association-arn arn:aws:apigateway:us-west-2:444455556666:/
domainnameaccessassociations/domainname/private.example.com+abcd1234/vpcesource/
vpce-abcd1234efg \
    --domain-name-arn arn:aws:apigateway:us-west-2:111122223333:/domainnames/
private.example.com+abcd1234
```

Refus de l'invocation de votre nom de domaine personnalisé privé par le fournisseur d'API

Après avoir refusé l'association d'accès au nom de domaine, vous devez supprimer le point de terminaison de VPC de votre politique `policy` pour le service `execute-api`.

Console de gestion AWS

Pour supprimer le point de terminaison de VPC de l'utilisateur d'API de votre politique de ressources

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Dans le panneau de navigation, sélectionnez Noms de domaine personnalisés.
3. Choisissez le nom de domaine personnalisé privé que vous avez partagé avec d'autres personnes Comptes AWS.
4. Dans l'onglet Stratégie de ressources choisissez Modifier.
5. Supprimez le point de terminaison de VPC de la politique.

6. Sélectionnez Enregistrer les modifications.

AWS CLI

La commande update-domain-name suivante utilise une opération de correctif pour mettre à jour la politique `policy` pour le service `execute-api` pour un nom de domaine personnalisé privé. Cette nouvelle politique `policy` supprime un ID de point de terminaison de VPC supplémentaire ajouté à la section the section called “Autorisation d’autres comptes à invoquer votre nom de domaine personnalisé privé” :

```
aws apigateway update-domain-name
    --domain-name private.example.com \
    --domain-name-id abcd1234 \
    --patch-operations op=replace,path=/policy,value='"{\"Version\": \"2012-10-17\",
       \"Statement\": [{\"Effect\": \"Allow\",\"Principal\": \"*\",\"Action\":
 \"execute-api:Invoke\",\"Resource\":[\"execute-api:/*\"]},{\"Effect\": \"Deny\",
\"Principal\": \"*\",\"Action\": \"execute-api:Invoke\",\"Resource\":[\"execute-
api:/*\"],\"Condition\":{\"StringNotEquals\":{\"aws:SourceVpce\": \"vpce-abcd1234efg
\"}}}]}"
```

L’utilisateur d’API doit ensuite supprimer l’association d’accès au nom de domaine. Vous ne pouvez pas le faire pour lui. Pour de plus amples informations, veuillez consulter the section called “Utilisateur d’API : suppression de votre association d’accès au nom de domaine avec un nom de domaine personnalisé privé”.

Fournisseur d'API : partagez votre nom de domaine personnalisé privé à l'aide de l'API Gateway AWS CLI

Vous pouvez partager un nom de domaine personnalisé privé à l'aide de l'API Gateway AWS CLI, mais nous vous recommandons de l'utiliser AWS RAM pour réduire vos frais d'exploitation. Pour obtenir des instructions sur la façon AWS RAM de partager votre nom de domaine personnalisé privé, consultezthe section called “Fournisseur d'API : partagez votre nom de domaine personnalisé privé en utilisant AWS RAM”.

Pour partager un nom de domaine personnalisé privé à l'aide de l'API Gateway AWS CLI, vous accordez un autre Comptes AWS accès pour créer des associations d'accès au nom de domaine et invoquer votre nom de domaine personnalisé privé. Pour ce faire, vous devez mettre à jour la politique `managementPolicy` pour le service API Gateway Management et la politique `policy` pour

le service `execute-api` pour votre nom de domaine personnalisé privé. Vous devez également accorder un accès au point de terminaison de VPC de l'utilisateur d'API dans la politique de ressources pour toutes les API privées mappées à votre nom de domaine personnalisé privé.

L'utilisateur d'API doit tout de même créer une association d'accès au nom de domaine de son propre compte entre le point de terminaison de son VPC et votre nom de domaine personnalisé privé. Vous ne pouvez pas le faire pour lui.

Octroi d'un accès à votre nom de domaine personnalisé privé

Pour accorder un accès à votre nom de domaine personnalisé privé

1. Pour mettre à jour la politique `managementPolicy` pour le service API Gateway Management, vous devez créer un fichier JSON contenant l'opération de correctif permettant de mettre à jour la politique. Ce qui suit `patch-managementPolicy.json` remplace la politique actuelle par un exemple de politique qui accorde l'accès à Comptes AWS 111122223333 et 444455556666 pour créer des associations d'accès au nom de domaine `managementPolicy` avec le nom de domaine personnalisé privé. `private.example.com`

   ```
   [{
       "op": "replace",
       "path": "/managementPolicy",
       "value": "{\"Version\":\"2012-10-17\",       \"Statement\":[{\"Effect
   \":\"Allow\",\"Principal\":{\"AWS\":[\"arn:aws:iam::111122223333:root
   \", \"arn:aws:iam::444455556666:root\"]},\"Action\":
   \"apigateway:CreateAccessAssociation\",\"Resource\":\"arn:aws:apigateway:us-
   west-2:111122223333:/domainnames/private.example.com+abcd1234\"}]}"
   }]
   ```

   La commande update-domain-name suivante met à jour la politique `managementPolicy` à l'aide du correctif `patch-managementPolicy.json`.

   ```
   aws apigateway update-domain-name \
       --domain-name private.example.com \
       --domain-name-id abcd1234 \
       --patch-operations file://patch-managementPolicy.json
   ```

   Une fois que vous avez accordé l'accès, vous devez informer l'utilisateur d'API qu'il peut former l'association d'accès au nom de domaine. Si vous l'utilisez AWS RAM, AWS RAM effectuera cette étape pour vous.

2. Pour mettre à jour la politique `policy` du service `execute-api`, vous devez créer un fichier JSON contenant l'opération de correctif permettant de mettre à jour la politique. Le correctif `patch-policy.json` suivant remplace la politique `policy` actuelle par un exemple de politique qui autorise deux points de terminaison de VPC à invoquer le nom de domaine personnalisé privé `private.example.com`.

```
[{
    "op": "replace",
    "path": "/policy",
    "value": "{\"Version\": \"2012-10-17\",       \"Statement\": [{\"Effect\":
 \"Allow\",\"Principal\": \"*\",\"Action\": \"execute-api:Invoke\",\"Resource\":
 \"arn:aws:execute-api:us-west-2:111122223333:/domainnames/private.example.com
+abcd1234\"},{\"Effect\": \"Deny\",\"Principal\": \"*\",\"Action\": \"execute-
api:Invoke\",\"Resource\": \"arn:aws:execute-api:us-west-2:111122223333:/
domainnames/private.example.com+abcd1234\",\"Condition\": {\"StringNotEquals\":
 {\"aws:SourceVpce\": [\"vpce-abcd1234\",\"vpce-xyzz0000\"]}}}]}"
}]
```

Utilisez la commande update-domain-name suivante pour mettre à jour la politique `policy` à l'aide du correctif `patch-policy.json`.

```
aws apigateway update-domain-name \
    --domain-name private.example.com \
    --domain-name-id abcd1234 \
    --patch-operations file://patch-policy.json
```

Refus de l'accès à votre nom de domaine personnalisé privé

Pour arrêter de partager votre nom de domaine personnalisé privé, vous devez refuser l'association d'accès au nom de domaine entre votre nom de domaine personnalisé privé et le point de terminaison de VPC de l'utilisateur d'API.

Pour refuser l'accès à votre nom de domaine personnalisé privé

1. La commande `reject-domain-name-access-association` suivante rejette l'association d'accès au nom de domaine.

```
aws apigateway reject-domain-name-access-association \
```

```
    --domain-name-access-association-arn arn:aws:apigateway:us-
west-2:444455556666:/domainnameaccessassociations/domainname/private.example.com
+abcd1234/vpcesource/vpce-abcd1234 \
    --domain-name-arn arn:aws:apigateway:us-west-2:111122223333:/domainnames/
private.example.com+abcd1234
```

2. Modifiez la politique `patch-managementPolicy.json` pour supprimer l'accès au compte du fournisseur d'API et créer une association d'accès au nom de domaine avec votre nom de domaine personnalisé privé. Le correctif `patch-managementPolicy.json` suivant supprime un compte de la politique `managementPolicy` :

```
[{
     "op": "replace",
     "path": "/managementPolicy",
     "value": "{\"Version\":\"2012-10-17\",       \"Statement\":[{\"Effect\":
\"Allow\",\"Principal\":\"*\",\"Action\":\"apigateway:CreateAccessAssociation
\",\"Resource\":\"arn:aws:apigateway:us-west-2:111122223333:/domainnames/
private.example.com+abcd1234\"}]}"
}]
```

La commande update-domain-name suivante met à jour la politique `managementPolicy` à l'aide du correctif `patch-managementPolicy.json`.

```
aws apigateway update-domain-name \
    --domain-name private.example.com \
    --domain-name-id abcd1234 \
    --patch-operations file://patch-managementPolicy.json
```

3. Modifiez le correctif `patch-policy.json` pour supprimer l'accès du point de terminaison de VPC au fournisseur d'API permettant d'invoquer votre nom de domaine personnalisé privé. Le correctif `patch-policy.json` suivant supprime l'ID de point de terminaison de VPC de la politique `policy` :

```
[{
    "op": "replace",
    "path": "/policy",
    "value": "{\"Version\":\"2012-10-17\",       \"Statement\":[{\"Effect\":
\"Allow\",\"Principal\":\"*\",\"Action\":\"execute-api:Invoke\",\"Resource\":
\"arn:aws:execute-api:us-west-2:111122223333:/domainnames/private.example.com
+abcd1234\"},{\"Effect\":\"Deny\",\"Principal\":\"*\",\"Action\":\"execute-
api:Invoke\",\"Resource\":\"arn:aws:execute-api:us-west-2:111122223333:/
```

```
domainnames/private.example.com+abcd1234\",\"Condition\":{\"StringNotEquals\":
{\"aws:SourceVpce\":\"vpce-abcd1234\"}}}]}"
}]
```

La commande update-domain-name suivante met à jour la politique `policy` à l'aide du correctif `patch-policy.json`.

```
aws apigateway update-domain-name \
    --domain-name private.example.com \
    --domain-name-id abcd1234 \
    --patch-operations file://patch-policy.json
```

Exemples de politiques utilisées dans cette procédure

La section suivante présente les exemples de politiques utilisées dans la procédure précédente.

L'exemple suivant concerne la politique `managementPolicy` du service Amazon API Gateway Management. Cette politique accorde l'accès à Comptes AWS 111122223333 et 444455556666 pour créer des associations d'accès au nom de domaine avec le nom de domaine personnalisé privé. `private.example.com`

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": [
                    "111122223333",
                    "444455556666"
                ]
            },
            "Action": "apigateway:CreateAccessAssociation",
            "Resource": "arn:aws:apigateway:us-west-2:111122223333:/domainnames/
private.example.com+a1b2c3"
        }
    ]
}
```

L'exemple suivant concerne la politique `policy` pour le service `execute-api`. Cette politique accorde aux points de terminaison de VPC `vpce-abcd1234` et `vpce-xyzz0000` un accès leur permettant d'invoquer le nom de domaine personnalisé privé.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": "*",
            "Action": "execute-api:Invoke",
            "Resource": "arn:aws:execute-api:us-west-2:111122223333:/domainnames/
private.example.com+abcd1234"
        },
        {
            "Effect": "Deny",
            "Principal": "*",
            "Action": "execute-api:Invoke",
            "Resource": "arn:aws:execute-api:us-west-2:111122223333:/domainnames/
private.example.com+abcd1234",
            "Condition": {
                "StringNotEquals": {
                    "aws:SourceVpce": [
                        "vpce-abcd1234",
                        "vpce-xyzz0000"
                    ]
                }
            }
        }
    ]
}
```

Utilisateur d'API : association de votre point de terminaison de VPC à un nom de domaine personnalisé privé partagé avec vous

La procédure suivante montre comment utiliser un nom de domaine privé dans un autre Compte AWS. En fonction de votre relation de confiance avec le fournisseur d'API, certaines tâches AWS RAM peuvent être effectuées à votre place.

Lorsque vous utilisez un nom de domaine Compte AWS différent d'un nom de domaine personnalisé privé, vous pouvez uniquement associer votre point de terminaison VPC à un nom de domaine personnalisé privé et l'invoquer. Vous ne pouvez pas voir la politique `policy` ni aucun autre paramètre du nom de domaine personnalisé privé.

Conditions préalables

Les conditions préalables suivantes sont requises pour utiliser un nom de domaine personnalisé privé dans un autre Compte AWS :

- Un VPC et un point de terminaison de VPC pour le service `execute-api`. Les valeurs `enableDnsHostnames` et `enableDnsSupport` de votre VPC doivent être définies sur `true`.
- Nous vous recommandons de configurer au moins deux zones de disponibilité par point de terminaison de VPC.

(Facultatif) Acceptez le partage de ressources du domaine personnalisé privé

Si votre fournisseur d'API AWS RAM créait un partage de ressources, vous disposez de 12 heures pour l'accepter. Si vous faites partie de la même organisation AWS Organizations que le fournisseur d'API, le partage est automatiquement accepté. Si vous faites partie d'une organisation qui a activé le partage automatique de ressources, celles-ci sont automatiquement partagées avec vous.

Console de gestion AWS

Pour l'utiliser Console de gestion AWS, voir Accepter et rejeter les invitations à partager des ressources dans le Guide de l'AWS RAM utilisateur.

AWS CLI

Pour trouver toutes les ressources partagées avec vous, utilisez la commande get-resource-share-invitations suivante :

```
aws ram get-resource-share-invitations \
    --region us-west-2
```

Utilisez l'ARN de partage de ressources obtenu pour accepter l'invitation de partage de ressources. La commande accept-resource-share-invitation suivante accepte le partage de ressources.

```
aws ram accept-resource-share-invitation \
```

```
    --resource-share-invitation-arn arn:aws:ram:us-west-2:123456789012:resource-
share-invitation/1e3477be-4a95-46b4-bbe0-c4001EXAMPLE \
    --region us-west-2
```

Association de votre point de terminaison de VPC à un nom de domaine personnalisé privé partagé

Comme les noms de domaine personnalisés privés ne sont pas uniques, vous devez associer votre point de terminaison de VPC à l'ARN unique du nom de domaine personnalisé. Après avoir créé votre association d'accès au nom de domaine, il peut s'écouler jusqu'à 15 minutes avant que votre point de terminaison de VPC n'invoque avec succès votre nom de domaine personnalisé privé. Si vous utilisez un point de terminaison de VPC pour accéder à un nom de domaine personnalisé public, ne l'utilisez pas pour créer une association d'accès à un nom de domaine.

Console de gestion AWS

Pour associer votre point de terminaison de VPC à un nom de domaine personnalisé privé partagé

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Dans le panneau de navigation principal, sélectionnez Associations d'accès aux noms de domaine.
3. Choisissez Créer une association d'accès au nom de domaine.
4. Pour ARN du nom de domaine, sélectionnez l'ARN du nom de domaine que le fournisseur d'API a partagé avec vous.

   L'ARN du nom de domaine peut ne pas apparaître dans la liste déroulante. Vous pouvez utiliser la AWS RAM console pour afficher les noms de domaine partagés avec vous, puis copier l'ARN du nom de domaine et le saisir dans ce champ.
5. Pour ID de point de terminaison de VPC, sélectionnez l'ID de point de terminaison de VPC avec lequel vous souhaitez former l'association d'accès au nom de domaine.
6. Choisissez Créer une association d'accès au nom de domaine.

AWS CLI

Comme les noms de domaine personnalisés privés ne sont pas uniques, vous devez associer votre point de terminaison de VPC à l'ARN unique du nom de domaine personnalisé. Pour rechercher l'ARN du nom de domaine, utilisez l'une des commandes suivantes.

1. AWS RAM

   La commande list-resources suivante recense les ressources partagées avec vous. Le fournisseur d'API doit avoir eu AWS RAM l'habitude de partager son domaine personnalisé privé avec vous pour utiliser cette commande.

   ```
   aws ram list-resources \
       --resource-owner OTHER-ACCOUNTS \
       --region us-west-2
       --resource-type apigateway:Domainnames
   ```

   API Gateway

   La `get-domain-names` commande suivante répertorie tous les noms de domaine personnalisés privés appartenant à d'autres personnes avec Comptes AWS lesquels vous pouvez créer des associations d'accès aux noms de domaine.

   ```
   aws apigateway get-domain-names \
       --resource-owner OTHER_ACCOUNTS \
       --region us-west-2
   ```

2. Après avoir récupéré l'ARN, utilisez API Gateway pour créer l'association d'accès au nom de domaine entre votre point de terminaison de VPC et un nom de domaine personnalisé privé partagé. Utilisez la commande `create-domain-name-access-association` suivante :

   ```
   aws apigateway create-domain-name-access-association \
       --access-association-source-type VPCE \
       --access-association-source 'vpce-1a2b3c4d5e6f1a2b3' \
       --domain-name-arn arn:aws:apigateway:us-west-2:111122223333:/domainnames/
   private.example.com+abcd1234"
   ```

   Le résultat se présente comme suit :

   ```
   {
   ```

```
    "domainNameAccessAssociationARN": "arn:aws:apigateway:us-
west-2:444455556666:/domainnameaccessassociations/domainname/private.example.com
+abcd1234/vpcesource/vpce-abcd1234efg",
    "accessAssociationSource": "vpce-1a2b3c4d5e6f1a2b3",
    "accessAssociationSourceType": "VPCE",
    "domainNameARN" : "arn:aws:apigateway:us-west-1:111122223333:/domainnames/
private.example.com+a1b2c3"
}
```

Après avoir associé votre point de terminaison de VPC au nom de domaine personnalisé privé, vérifiez que votre fournisseur d'API a mis à jour la politique de son nom de domaine personnalisé privé de manière à autoriser le point de terminaison de votre VPC à invoquer son nom de domaine. Pour de plus amples informations, veuillez consulter the section called "Autorisation d'autres comptes à invoquer votre nom de domaine personnalisé privé".

Création d'une zone hébergée Route 53

Pour résoudre le nom de domaine personnalisé privé, vous devez créer une zone hébergée privée Route 53. Une zone hébergée est un conteneur qui comporte des informations sur la façon dont vous souhaitez acheminer le trafic pour un domaine dans un ou plusieurs VPC, sans exposer vos ressources à Internet. Pour plus d'informations, veuillez onsulter Utilisation des zones hébergées privées.

Console de gestion AWS

Pour l'utiliser Console de gestion AWS, consultez la section Création d'une zone hébergée privée dans le guide du développeur Amazon Route 53.

Pour Nom, utilisez le nom du nom de domaine personnalisé privé. Pour ID du VPC, utilisez le VPC contenant le point de terminaison de VPC que vous avez utilisé pour votre association d'accès au nom de domaine.

AWS CLI

La commande create-hosted-zone suivante crée une zone hébergée privée :

```
aws route53 create-hosted-zone --name private.example.com \
    --caller-reference 2014-04-01-18:47 \
    --hosted-zone-config Comment="command-line version",PrivateZone=true \
    --vpc VPCRegion=us-west-2,VPCId=vpc-abcd1234
```

La sortie contient l'ID de la zone hébergée. Vous allez l'utiliser lors des étapes suivantes.

Création d'un enregistrement DNS Route 53

Après avoir créé la zone hébergée, vous devez créer un enregistrement pour résoudre le domaine personnalisé privé. Dans cet exemple, vous allez créer un type d'enregistrement A. Si vous utilisez IPv6 pour votre point de terminaison de VPC, créez un type d'enregistrement AAAA. Si vous utilisez une double pile pour votre point de terminaison de VPC, créez un enregistrement de type AAAA et un type d'enregistrement A.

Console de gestion AWS

Pour l'utiliser Console de gestion AWS, consultez la section Routage du trafic vers une API Amazon API Gateway à l'aide de votre nom de domaine.

Utilisez Création rapide et activez Alias. Pour le point de terminaison, utilisez le nom DNS du point de terminaison du VPC.

AWS CLI

Pour configurer vos enregistrements DNS de manière à ce qu'ils mappent le nom de domaine personnalisé privé au nom d'hôte de l'ID de la zone hébergée, commencez par créer un fichier JSON qui contient la configuration pour créer un enregistrement DNS pour le nom de domaine privé.

L'enregistrement `setup-dns-record.json` suivant montre comment créer un enregistrement DNS `A` pour mapper un nom de domaine personnalisé privé à son nom d'hôte privé. Vous allez fournir le nom `DNSName` de l'ID DNS de votre VPC et l'ID de la zone hébergée que vous avez créée à l'étape précédente.

```
{
  "Changes": [
    {
      "Action": "UPSERT",
      "ResourceRecordSet": {
        "Name": "private.example.com",
        "Type": "A",
        "AliasTarget": {
          "DNSName": "vpce-abcd1234.execute-api.us-west-2.vpce.amazonaws.com",
          "HostedZoneId": "Z2OJLYMUO9EFXC",
          "EvaluateTargetHealth": false
        }
```

```
      }
    }
  ]
}
```

La commande change-resource-record-sets suivante crée un enregistrement DNS pour le nom de domaine personnalisé privé :

```
aws route53 change-resource-record-sets \
    --hosted-zone-id ZABCDEFG1234 \
    --change-batch file://path/to/your/setup-dns-record.json
```

Remplacez `hosted-zone-id` par l'ID de la zone hébergée Route 53 du jeu d'enregistrements DNS de votre compte. La valeur du paramètre `change-batch` pointe vers un fichier JSON.

Prochaines étapes pour un utilisateur d'API

Vous pouvez désormais invoquer l'API privée dans votre propre Compte AWS. Dans votre VPC, vous pouvez utiliser la commande curl suivante pour accéder à votre nom de domaine personnalisé privé.

```
curl https://private.example.com/v1
```

Pour découvrir d'autres façons d'invoquer votre API privée, consultez the section called "Invocation d'une API privée à l'aide d'un nom de domaine personnalisé".

Utilisateur d'API : suppression de votre association d'accès au nom de domaine avec un nom de domaine personnalisé privé

Si vous êtes un utilisateur d'API, vous pouvez à tout moment supprimer l'association d'accès. Le fournisseur d'API ne peut pas la supprimer pour vous.

Nous vous recommandons de toujours supprimer une association d'accès à un nom de domaine lorsque vous ne l'utilisez plus.

Console de gestion AWS

Pour supprimer l'association d'accès au nom de domaine

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

2. Dans le panneau de navigation principal, sélectionnez Associations d'accès aux noms de domaine.
3. Sélectionnez votre association d'accès au nom de domaine, puis choisissez Supprimer.
4. Confirmez votre choix, puis choisissez Supprimer.

AWS CLI

La commande `delete-access-association` suivante supprime l'association d'accès :

```
aws apigateway delete-domain-name-access-association \
    --domain-name-access-association-arn 'arn:aws:apigateway:us-
west-2:444455556666:/domainnameaccessassociations/domainname/private.example.com
+abcd1234/vpcesource/vpce-abcd1234efg'
```

Créez un nom de domaine personnalisé pour les API privées à l'aide de CloudFormation

L'exemple de CloudFormation modèle suivant crée une API privée et un nom de domaine personnalisé privé, mappe l'API privée au nom de domaine personnalisé, puis crée une association d'accès au nom de domaine. Vous devez fournir votre point de terminaison de VPC, votre nom de domaine et votre ARN de certificat.

Les considérations suivantes peuvent avoir une incidence sur votre utilisation de CloudFormation pour créer un nom de domaine personnalisé privé :

- Vous ne pouvez pas rejeter une association d'accès à un nom de domaine en utilisant CloudFormation. Pour rejeter une association d'accès à un nom de domaine, utilisez l' AWS CLI.
- Utilisez cette `AWS::ApiGateway::DomainNameV2` CloudFormation propriété pour créer un nom de domaine personnalisé privé.
- Utilisez cette `AWS::ApiGateway:BasePathMappingV2` CloudFormation propriété pour créer un mappage de chemin de base.

```
AWSTemplateFormatVersion: 2010-09-09
Parameters:
  EndpointID:
    Type: String
    Default: vpce-abcd1234567efg
```

```
    Description: A VPC endpoint with enableDnsHostnames and enableDnsSupport set to
 true.
  DomainName:
    Type: String
    Default: private.example.com
    Description: A domain name that you own.
  CertificateArn:
    Type: String
    Default: arn:aws:acm:us-west-2:123456789:certificate/abcd-000-1234-0000-000000abcd
    Description: An ACM certificate that covers the domain name.
Resources:
  PrivateApi:
    Type: 'AWS::ApiGateway::RestApi'
    Properties:
      EndpointConfiguration:
        Types:
          - PRIVATE
        VpcEndpointIds:
          - !Ref EndpointID
      Name: private-api
      Policy:
        Statement:
          - Action: 'execute-api:Invoke'
            Effect: Allow
            Principal: '*'
            Resource: 'execute-api:/*'
          - Action: 'execute-api:Invoke'
            Condition:
              StringNotEquals:
                'aws:SourceVpce': !Ref EndpointID
            Effect: Deny
            Principal: '*'
            Resource: 'execute-api:/*'
        Version: 2012-10-17
  PrivateApiDeployment:
    Type: 'AWS::ApiGateway::Deployment'
    Properties:
      RestApiId: !Ref PrivateApi
      Description: Private API deployment
    DependsOn:
      - PrivateApiMethod
  PrivateApiStage:
    Type: 'AWS::ApiGateway::Stage'
    Properties:
```

```
      RestApiId: !Ref PrivateApi
      DeploymentId: !Ref PrivateApiDeployment
      StageName: prod
  PrivateApiMethod:
    Type: 'AWS::ApiGateway::Method'
    Properties:
      HttpMethod: ANY
      ResourceId: !GetAtt PrivateApi.RootResourceId
      RestApiId: !Ref PrivateApi
      AuthorizationType: NONE
      Integration:
        Type: MOCK
        RequestTemplates:
          application/json: "{\"statusCode\": 200}"
        IntegrationResponses:
          - StatusCode: '200'
      MethodResponses:
        - StatusCode: '200'
  PrivateDomainName:
    Type: AWS::ApiGateway::DomainNameV2
    Properties:
      DomainName: !Ref DomainName
      CertificateArn: !Ref CertificateArn
      EndpointConfiguration:
        Types:
          - PRIVATE
      SecurityPolicy: TLS_1_2
      Policy:
        Statement:
            - Action: 'execute-api:Invoke'
              Effect: Allow
              Principal: '*'
              Resource: 'execute-api:/*'
            - Action: 'execute-api:Invoke'
              Condition:
                StringNotEquals:
                  'aws:SourceVpce': !Ref EndpointID
              Effect: Deny
              Principal: '*'
              Resource: 'execute-api:/*'
        Version: 2012-10-17
  PrivateBasePathMapping:
    Type: AWS::ApiGateway::BasePathMappingV2
    DependsOn:
```

```
      - PrivateApiStage
    Properties:
      BasePath: prod
      DomainNameArn: !GetAtt PrivateDomainName.DomainNameArn
      RestApiId: !Ref PrivateApi
      Stage: prod
  DomainNameAccessAssociation:
    Type: AWS::ApiGateway::DomainNameAccessAssociation
    Properties:
      DomainNameArn: !GetAtt PrivateDomainName.DomainNameArn
      AccessAssociationSource: !Ref EndpointID
      AccessAssociationSourceType: VPCE
```

Invocation d’une API privée

Vous ne pouvez invoquer une API privée que depuis un VPC à l’aide d’un point de terminaison de VPC. Votre API privée doit disposer d’une politique de ressources qui autorise des VPC et des points de terminaison de VPC spécifiques à invoquer votre API.

Si vous invoquez une API privée sans utiliser de nom de domaine personnalisé ou de noms DNS privés et que vos API ou nom de domaine utilisent une politique de sécurité commençant par`SecurityPolicy_`, vous devez définir le mode d'accès des terminaux sur`BASIC`. Pour de plus amples informations, veuillez consulter the section called “Mode d’accès au point de terminaison”.

Invocation d’une API privée à l’aide d’un nom de domaine personnalisé

Pour invoquer une API privée à l’aide d’un nom de domaine personnalisé, votre point de terminaison de VPC a besoin d’une association d’accès au nom de domaine avec un nom de domaine personnalisé, et le nom de domaine personnalisé doit autoriser l’accès au point de terminaison de VPC pour l’invoquer. Pour de plus amples informations, veuillez consulter the section called “Noms de domaine personnalisés pour les API privées”.

Il n'y a aucune différence entre l'invocation d'un nom de domaine personnalisé privé dans un VPC propre Compte AWS ou dans un autre. Compte AWS

Utilisation de votre nom de domaine personnalisé

Dans votre VPC, vous pouvez invoquer votre API à l’aide du nom de domaine personnalisé. L’exemple suivant est une commande curl permettant d’invoquer votre nom de domaine personnalisé privé :

```
curl https://private.example.com
```

Utilisation de noms d'hôte DNS privés spécifiques au point de terminaison

Vous pouvez invoquer votre API en utilisant le nom de domaine personnalisé et le nom d'hôte DNS privé spécifique au point de terminaison.

```
curl https://private-dns-hostname.execute-api.region.vpce.amazonaws.com/basepath -H
 'Host:custom-domain-name'
```

L'exemple suivant est une commande curl permettant d'invoquer votre nom de domaine personnalisé à l'aide d'un nom d'hôte DNS privé spécifique au point de terminaison :

```
curl https://vpce-123456-abc000.execute-api.us-east-2.vpce.amazonaws.com/test -H
 'Host:private.example.com'
```

Invocation d'une API privée sans utiliser de nom de domaine personnalisé

Pour invoquer votre API privée sans utiliser de nom de domaine personnalisé, vous devez identifier les noms DNS de votre API. La procédure suivante montre comment rechercher vos noms DNS.

Console de gestion AWS

Pour rechercher les noms DNS

1. Connectez-vous au Console de gestion AWS et ouvrez la console Amazon VPC à https://console.aws.amazon.com/vpc/ l'adresse.
2. Dans le panneau de navigation de gauche, choisissez Points de terminaison, puis choisissez votre point de terminaison de VPC d'interface pour API Gateway.
3. Dans le volet Détails, cinq valeurs sont répertoriées dans le champ Noms DNS. Les trois premières sont les noms DNS publics de votre API. Les deux autres sont ses noms DNS privés.

AWS CLI

Utilisez la commande describe-vpc-endpoints suivante pour répertorier vos valeurs DNS.

```
aws ec2 describe-vpc-endpoints --vpc-endpoint-ids vpce-01234567abcdef012
```

Les trois premières sont les noms DNS publics de votre API. Les deux autres sont ses noms DNS privés.

Invocation d'une API privée à l'aide d'un alias Route53

Vous pouvez associer un point de terminaison de VPC à votre API privée ou l'en dissocier. Pour de plus amples informations, veuillez consulter the section called "(Facultatif) Association d'un point de terminaison de VPC à une API privée ou dissociation d'un point de terminaison de VPC d'une API privée".

Après avoir associé vos points de terminaison de VPC à votre API privée, vous pouvez utiliser l'URL de base suivante pour invoquer l'API :

```
https://{rest-api-id}-{vpce-id}.execute-api.{region}.amazonaws.com/{stage}
```

Par exemple, si vous avez configuré la méthode `GET /pets` pour l'étape `test`, que l'ID de votre API REST était `01234567ab`, que l'ID de votre point de terminaison de VPC était `vpce-01234567abcdef012` et que votre région était `us-west-2`, vous pouvez invoquer votre API comme suit :

```
curl -v https://01234567ab-vpce-01234567abcdef012.execute-api.us-west-2.amazonaws.com/
test/pets
```

Invocation d'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 les noms DNS privés suivants :

```
{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 méthode `GET /pets` pour l'étape `test`, que l'ID de votre API REST était `01234567ab` et que votre région était `us-west-2`, vous pouvez invoquer 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 invoquer 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 de VPC, vous ne pourrez pas accéder au point de terminaison par défaut pour les API publiques. Pour plus d'informations, consultez Pourquoi ne puis-je pas me connecter à mon API publique à partir d'un point de terminaison de VPC API Gateway ?

Invoquez une API privée à l'aide de Direct Connect

Vous pouvez l'utiliser 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.

Invocation d'une API privée à l'aide de noms d'hôte 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 de 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 configurez la méthode `GET /pets` pour l'étape `test` et que l'ID de votre API REST était `abc1234`, que son nom d'hôte DNS public était `vpce-def-01234567` et que votre région était `us-west-2`, vous pouvez invoquer votre API privée à l'aide de son ID VPCe en utilisant l'en-tête `Host` dans 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 invoquer votre API privée via son ID d'API en utilisant l'en-tête `x-apigw-api-id` dans 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 API Gateway

Dans cette section, vous pouvez apprendre à surveiller votre API à l'aide de CloudWatch métriques, de 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 pouvez également journaliser 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
> - 431 Erreurs trop nombreuses dans les champs de l'en-tête de requête
> - 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, consultez the section called "Utilisation de la console pour tester une méthode API REST".

Rubriques

- Surveillez l'exécution de l'API REST avec CloudWatch les métriques Amazon

- Configuration de la CloudWatch journalisation pour les API REST dans API Gateway
- Suivi des demandes des utilisateurs adressées aux API REST à l'aide de X-Ray dans API Gateway

Surveillez 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 métriques IntegrationLatency pour mesurer la réactivité du serveur principal.
- Surveillez les métriques Latency pour mesurer la réactivité globale de vos appels d'API.
- Surveillez les métriques CacheHitCount et CacheMissCount pour optimiser les capacités de cache afin d'atteindre les performances souhaitées.

Rubriques

- Dimensions et métriques Amazon API Gateway
- Afficher les métriques CloudWatch avec le tableau de bord 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 intégrés AWS à API Gateway

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 Surveillez 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 compte les codes d'état de réponse de la passerelle modifiés comme 4 XXError erreurs. 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 les métriques CloudWatch avec le tableau de bord API dans API Gateway

Vous pouvez utiliser le tableau de bord API de la console API Gateway pour afficher les métriques CloudWatch 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'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 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 métriques CloudWatch au niveau de la méthode, assurez-vous d'avoir activé CloudWatch Logs à ce niveau-là. Pour de plus amples informations sur la configuration de la journalisation au niveau de la méthode, veuillez consulter Remplacement des paramètres au niveau de l'étape.

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 de la méthode pour votre API, activez les métriques détaillées. Pour de plus amples informations, veuillez consulter 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 votre nom APIs et le nom de la métrique.
6. Pour voir les métriques par API spécifique, choisissez le volet By Api Name. Sélectionnez ensuite votre nom APIs et le nom de la métrique.

Pour afficher les métriques à l'aide de la AWS CLI

1. Utilisez la commande list-metrics 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. Utilisez la get-metrics-statisticscommande suivante pour afficher la moyenne sur une période en utilisant des intervalles de 5 minutes :

   ```
   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

La section suivante explique les prérequis nécessaires et explique comment afficher les événements du journal d'API Gateway dans la CloudWatch console.

Conditions préalables

1. Vous devez avoir créé une API dans API Gateway. Suivez les instructions de la section Développement d'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 API Gateway et Invocation d'une API REST dans API Gateway.
3. Les CloudWatch journaux doivent être activés pour une étape. Suivez les instructions de la section Configuration de la CloudWatch journalisation pour les 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 intégrés AWS à API Gateway

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 de plus amples informations, veuillez consulter Surveillez 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 les API REST dans API Gateway

Pour vous aider à résoudre les problèmes liés à l'exécution des requêtes ou à l'accès des clients à votre API, vous pouvez autoriser Amazon CloudWatch Logs à enregistrer les appels d'API. Pour plus d'informations sur CloudWatch, consultezthe section called "CloudWatch métriques".

Il existe deux types de connexion à l'API CloudWatch :

- Journalisation des exécutions : API Gateway enregistre les actions entreprises pour traiter les demandes d'API, y compris les erreurs et les traces d'exécution. Vous pouvez également configurer la diffusion d'Amazon CloudWatch Logs pour acheminer les journaux d'exécution vers vos propres destinations.
- Enregistrement des accès : vous enregistrez qui a accédé à votre API et comment. Vous créez votre propre groupe de journaux, vous choisissez un format de journal et vous spécifiez les `$context` variables à inclure.

Vous pouvez activer la journalisation de l'exécution et la journalisation de l'accès indépendamment l'une de l'autre.

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 politique AmazonAPIGatewayPushToCloudWatchLogs gérée possède toutes les autorisations requises.

> **Note**
>
> API Gateway lance des appels AWS Security Token Service afin d'assumer le rôle IAM. Assurez-vous donc que cette option AWS STS est activée pour la région. Pour plus d'informations, consultez la section Gestion AWS des STS dans une AWS région.

Pour accorder ces autorisations à votre compte, créez un rôle IAM avec `apigateway.amazonaws.com` comme entité de confiance, associez la politique précédente au rôle IAM et définissez l'ARN du rôle IAM sur la WatchRoleArn propriété cloud de votre compte. https://docs.aws.amazon.com/apigateway/latest/api/API_GetAccount.html Vous devez définir la WatchRoleArn propriété cloud 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 des AWS STS dans une AWS région dans le Guide de l'utilisateur IAM.

Rubriques

- Journalisation des exécutions pour les API REST
- Journalisation des accès pour les API REST

Journalisation des exécutions pour les API REST

La journalisation des exécutions enregistre les actions entreprises par API Gateway pour traiter une demande d'API. Les données enregistrées incluent les erreurs, les traces d'exécution, les valeurs des paramètres de demande et de réponse ou les charges utiles, les données utilisées par les

autorisateurs Lambda, si des clés d'API sont requises et si les plans d'utilisation sont activés. API Gateway supprime les en-têtes d'autorisation, les valeurs des clés d'API et d'autres paramètres de demande sensibles similaires des données journalisées.

Pour renforcer votre niveau de sécurité, nous vous recommandons d'utiliser la journalisation des exécutions au niveau `ERROR` ou `INFO`. Vous devrez peut-être le faire pour vous conformer aux différents cadres de conformité. Pour plus d'informations, consultez Amazon API Gateway controls dans le Guide de l'utilisateur AWS Security Hub .

Journalisation des exécutions standard

Lorsque vous activez la journalisation des exécutions sur une étape, API Gateway crée et gère automatiquement un groupe de CloudWatch journaux de journaux. Le groupe de journaux est nomméAPI-Gateway-Execution-Logs_{rest-api-id}/{stage_name}. Au sein de chaque groupe de journaux, les journaux sont divisés en flux de journaux classés par date du dernier événement.

Avec la journalisation d'exécution standard :

- Les événements du journal sont tronqués à 1 Ko.
- API Gateway gère le nom du groupe de journaux. Vous ne pouvez pas choisir le nom du groupe de journaux ou envoyer des journaux directement à Amazon S3 ou Firehose.

Pour activer la journalisation des exécutions standard, réglez le paramètre `loggingLevel` on your stage sur `ERROR` ou`INFO`. Aucune autre configuration n'est requise.

Permissions

Pour activer CloudWatch les journaux, vous devez autoriser API Gateway à lire et à écrire les journaux CloudWatch de votre compte. La politique AmazonAPIGatewayPushToCloudWatchLogs gérée possède toutes les autorisations requises.

Créez un rôle IAM avec `apigateway.amazonaws.com` comme entité de confiance, associez la politique précédente et définissez l'ARN du rôle IAM sur la WatchRoleArn propriété cloud de votre compte. Vous devez définir cette propriété séparément pour chaque AWS région dans laquelle vous souhaitez activer CloudWatch les journaux.

> **Note**
>
> API Gateway appelle AWS Security Token Service pour assumer le rôle IAM. Assurez-vous donc que cette option AWS STS est activée pour la région. Pour plus d'informations, consultez la section Gestion AWS des STS dans une AWS région.

Activer la journalisation des exécutions (console)

Avant de commencer, déployez l'API sur une étape et configurez 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 doté des autorisations appropriées. Vous devez effectuer cette opération une fois pour chaque application Compte AWS qui crée des API à l'aide d'API Gateway.
4. Dans le volet de navigation principal, choisissez API, choisissez votre API REST, puis choisissez Stages.
5. Choisissez une étape, puis dans la section Journaux et suivi, choisissez Modifier.
6. Sélectionnez un niveau de journalisation dans le menu déroulant CloudWatch Journaux :
   - Désactivé : API Gateway n'enregistre pas les demandes pour cette étape.
   - Erreurs uniquement : API Gateway enregistre uniquement les erreurs.
   - Erreurs et journaux d'informations : API Gateway enregistre tous les événements.
7. (Facultatif) Sélectionnez Suivi des données pour enregistrer les données complètes des demandes et des réponses. Cela peut entraîner l'enregistrement de données sensibles.

   > **Note**
   >
   > Nous vous recommandons de ne pas activer Suivi des données pour les API de production.

8. (Facultatif) Sélectionnez Mesures détaillées pour activer les CloudWatch mesures détaillées. Pour plus d'informations, consultez the section called “CloudWatch métriques”.

9. Sélectionnez Save Changes (Enregistrer les modifications).

Il n'est pas nécessaire de redéployer l'API lorsque vous mettez à jour les paramètres de stage.

Activez la journalisation des exécutions (CloudFormation)

Utilisez la `MethodSettings` propriété de la ressource de scène pour activer la journalisation de l'exécution.

```
TestStage:
  Type: AWS::ApiGateway::Stage
  Properties:
    StageName: test
    RestApiId: !Ref MyAPI
    DeploymentId: !Ref Deployment
    MethodSettings:
      - ResourcePath: "/*"
        HttpMethod: "*"
        LoggingLevel: INFO
```

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 le Guide de l'utilisateur AWS CloudFormation .

Livraison CloudWatch des journaux d'exécution par Amazon Logs

La diffusion CloudWatch d'Amazon Logs fournit un autre chemin pour les journaux d'exécution. Au lieu d'écrire dans un groupe de Gateway-managed journaux d'API, vous configurez une diffusion qui envoie des journaux d'exécution vers des destinations que vous possédez et contrôlez.

Avec la livraison Amazon CloudWatch Logs :

- Les événements du journal peuvent atteindre 1 Mo, ce qui permet de capturer des données de demande et de réponse plus complètes. Les charges utiles supérieures à 1 Mo sont toujours tronquées.
- Vous choisissez la destination : vos propres groupes de CloudWatch journaux Logs, vos compartiments Amazon S3 ou vos flux Firehose.
- Vous pouvez envoyer des journaux vers plusieurs destinations simultanément.
- Vous contrôlez la configuration des groupes de journaux, y compris la dénomination et la conservation.

> **Important**
>
> CloudWatch La livraison des journaux remplace la journalisation d'exécution standard. Lorsque vous configurez une diffusion, API Gateway arrête d'écrire dans le groupe de journaux géré automatiquement. Vous ne pouvez pas utiliser les deux chemins en même temps pour la même étape.

Lorsque vous créez une diffusion, API Gateway achemine les journaux d'exécution vers les destinations que vous avez configurées au lieu du groupe de journaux géré automatiquement. Lorsque vous supprimez une livraison, les journaux d'exécution reprennent automatiquement leur transfert vers le groupe de journaux géré automatiquement. Aucune modification de configuration d'API Gateway n'est nécessaire au-delà de la condition préalable de paramétrage `loggingLevel` sur `ERROR` ou`INFO`.

> **Note**
>
> La livraison des grumes constitue le meilleur effort. Dans de rares cas, certains événements du journal peuvent ne pas être transmis.

> **Warning**
>
> Avant d'activer la remise des journaux, mettez à jour les tableaux de bord, les alarmes ou les filtres d'abonnement qui font référence au groupe de journaux géré automatiquement ()`API-Gateway-Execution-Logs_{rest-api-id}/{stage_name}`. Une fois la livraison active, ce groupe de journaux ne reçoit plus de nouveaux événements.

Le tableau suivant compare la journalisation d'exécution standard à la diffusion d'Amazon CloudWatch Logs.

	Journalisation des exécutions standard	Livraison Amazon CloudWatch Logs
Taille maximale des événements de journalisation	1 Ko	1 Mo
Destinations	CloudWatch Journaux (groupe de journaux géré)	CloudWatch Logs, Amazon S3, Firehose (vos propres ressources)
Destinations multiples	Non	Oui
Nom du groupe de journaux	Corrigé par API Gateway	À vous de choisir
Configuration	Automatique lorsque la journalisation est activée	Nécessite la création d'une livraison Amazon CloudWatch Logs
Prérequis	Réglez `loggingLevel` sur `ERROR` ou `INFO`	Réglez `loggingLevel` sur `ERROR` ou `INFO`

> **Note**
>
> Si vous définissez cette `loggingLevel` valeur `OFF` après avoir configuré une diffusion, aucun journal d'exécution n'est généré ou transmis. Vos destinations de livraison restent configurées mais ne reçoivent aucune donnée.

Pour créer une diffusion, consultezthe section called “Créer une livraison de journaux”.

Création d'une livraison de journal pour les journaux d'exécution de l'API REST

Vous pouvez acheminer les journaux d'exécution vers vos propres groupes de CloudWatch journaux, vos compartiments Amazon S3 ou vos flux Firehose. Pour ce faire, créez une livraison Amazon CloudWatch Logs. Cela implique trois étapes : créer une source de livraison, créer une destination de livraison et les associer à une livraison.

Conditions préalables

Avant de créer une livraison de journaux, assurez-vous de disposer des éléments suivants :

- La journalisation de l'exécution est activée sur le stage de votre API REST. Définissez `loggingLevel` sur `ERROR` ou `INFO`. Si la journalisation n'est pas activée, la création de la source de diffusion échoue.
- Une ressource de destination : un groupe de CloudWatch journaux Logs, un compartiment Amazon S3 ou un flux Firehose dont vous êtes propriétaire.
- Autorisations pour appeler les API de livraison Amazon CloudWatch Logs. Pour plus d'informations, consultez la section Activer la journalisation à partir AWS des services dans le guide de l'utilisateur Amazon CloudWatch Logs.

Gateway-specific Valeurs d'API pour la livraison d'Amazon CloudWatch Logs

Lorsque vous créez une diffusion pour les journaux d'exécution d'API Gateway, utilisez les valeurs suivantes :

ARN des ressources

```
arn:aws:apigateway:{region}:{account-id}:/restapis/{rest-api-id}/
stages/{stage-name}
```

Type de journal

```
EXECUTION_LOGS
```

Créez une livraison de journaux (AWS CLI)

L'exemple suivant crée une diffusion qui envoie des journaux d'exécution depuis une API REST vers un groupe de CloudWatch journaux de journaux dont vous êtes le propriétaire.

1. Création de la source de diffusion

   Enregistrez votre API REST en tant que source de diffusion.

```
aws logs put-delivery-source \
    --name my-apigw-source \
    --log-type EXECUTION_LOGS \
```

```
    --resource-arn arn:aws:apigateway:{region}:{account-id}:/restapis/{rest-api-
id}/stages/{stage-name}
```

2. Créez la destination de livraison

   Enregistrez votre groupe de journaux en tant que destination de livraison.

```
aws logs put-delivery-destination \
    --name my-log-destination \
    --delivery-destination-configuration
 "destinationResourceArn=arn:aws:logs:{region}:{account-id}:log-group:{log-group-
name}"
```

3. Créez la livraison

   Liez la source à la destination.

```
aws logs create-delivery \
    --delivery-source-name my-apigw-source \
    --delivery-destination-arn arn:aws:logs:{region}:{account-id}:delivery-
destination:my-log-destination
```

Pour envoyer des journaux à Amazon S3 ou Firehose, spécifiez l'ARN de ressource approprié dans la `put-delivery-destination` commande. Pour plus d'informations sur le flux de livraison complet d'Amazon CloudWatch Logs, consultez la section Activer la journalisation à partir AWS des services dans le guide de l'utilisateur Amazon CloudWatch Logs.

Pour envoyer des journaux vers plusieurs destinations, créez des livraisons supplémentaires en utilisant la même source de livraison avec différentes destinations.

Champs d'enregistrement

Avec la diffusion d'Amazon CloudWatch Logs, les journaux d'exécution incluent des champs structurés que vous pouvez personnaliser en fonction de la destination. Les champs suivants sont disponibles :

- `resource_arn`— L'ARN de l'étape API
- `event_timestamp`— L'heure de l'événement de journalisation
- `api_id`— L'identifiant de l'API REST
- `stage`— Le nom de scène

- `resource_path`— Le chemin de ressource de la demande
- `http_method`— La méthode HTTP
- `payload`— Le contenu du journal d'exécution (identique à celui des journaux d'exécution standard)

Vous pouvez sélectionner les champs à inclure et le format de sortie pour chaque destination à l'aide du `--record-fields` paramètre activé `create-delivery` et du `--output-format` paramètre activé`put-delivery-destination`.

Les formats de sortie disponibles dépendent de la destination :

- CloudWatch Journaux — json, en clair
- Amazon S3 — JSON, brut, W3C, parquet
- Firehose — json, clair

> **Correspond à la sortie standard du journal d'exécution**
>
> Si vous souhaitez que le résultat de livraison corresponde à la journalisation d'exécution standard (charge utile uniquement, pas de champs supplémentaires), ajoutez `--output-format "plain"` à `put-delivery-destination` et `--record-fields "payload" --field-delimiter ""` à`create-delivery`.

Créez une livraison de journaux (CloudFormation)

Lorsque vous assemblez l'ARN de scène pour les CloudFormation modèles, utilisez le modèle suivant :

```
  DeliverySource:
    Type: AWS::Logs::DeliverySource
    Properties:
      Name: my-apigw-source
      LogType: EXECUTION_LOGS
      ResourceArn: !Sub "arn:${AWS::Partition}:apigateway:${AWS::Region}:
${AWS::AccountId}:/restapis/${MyApi}/stages/${MyStageName}"
```

Créer une livraison de journal (console API Gateway)

Vous pouvez créer une livraison de journaux dans la section Journaux et suivi de la page des détails de votre étape.

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Choisissez votre API REST, puis choisissez Stages.
3. Choisissez une étape.
4. Dans la section Journaux et suivi, sous Destinations de livraison des journaux, choisissez Ajouter une destination.
5. Choisissez un type de destination (CloudWatch Logs, Amazon S3 ou Firehose) et spécifiez la ressource de destination.
6. Cliquez sur Enregistrer.

> **Important**
>
> Vous devez activer la journalisation des exécutions sur la scène avant de pouvoir ajouter des destinations de livraison. Si la journalisation est désactivée, la configuration de diffusion ne génère pas de journaux.

> **Note**
>
> Pour configurer les destinations de livraison entre comptes, utilisez l' AWS interface de ligne de commande. Vous devez créer une politique de destination de livraison dans le compte de destination avant de créer la livraison dans le compte source. Pour plus d'informations, consultez l'exemple Cross-account de livraison dans le guide de l'utilisateur Amazon CloudWatch Logs.

Vérifier la livraison du journal d'exécution

Après avoir créé une livraison de journaux, vérifiez qu'elle est active et que les journaux d'exécution arrivent à destination.

Pour confirmer qu'une livraison est active, utilisez la `describe-deliveries` commande suivante :

```
aws logs describe-deliveries
```

Envoyez ensuite une demande de test et vérifiez votre destination :

1. Envoyez une demande de test à votre API REST.

   ```
   curl https://{rest-api-id}.execute-api.{region}.amazonaws.com/{stage}/{resource}
   ```

2. Attendez 1 à 2 minutes pour que les journaux se propagent.
3. Vérifiez votre destination pour les événements du journal :

   - CloudWatch Journaux : ouvrez la console CloudWatch Journaux et accédez à votre groupe de journaux. Vous devriez voir un flux de journaux avec les événements récents.
   - Amazon S3 : ouvrez la console Amazon S3 et accédez à votre compartiment. Recherchez les objets dont le préfixe de chemin inclut votre ID d'API REST.
   - Firehose : vérifiez les statistiques du flux de diffusion dans la console Firehose pour confirmer que les données circulent.

Résolution des problèmes

Si les journaux n'arrivent pas à destination, vérifiez les points suivants :

- La journalisation est activée — Vérifiez que `loggingLevel` c'est réglé `INFO` sur `ERROR` ou sur scène. Si c'est le cas`OFF`, aucun journal n'est généré, quelle que soit la configuration de diffusion.
- État de la livraison : utilisez la commande `get-delivery` CLI pour vérifier l'état de la livraison et confirmer qu'il est actif.

  ```
  aws logs get-delivery --id {delivery-id}
  ```

- Autorisations : vérifiez que la politique relative aux ressources de votre destination autorise Amazon CloudWatch Logs à y écrire. Pour plus d'informations, consultez la section Activer la journalisation à partir AWS des services dans le guide de l'utilisateur Amazon CloudWatch Logs.

Supprimer une livraison de journal pour les journaux d'exécution de l'API REST

Lorsque vous supprimez une livraison de journal, API Gateway reprend l'écriture des journaux d'exécution dans le groupe de journaux géré automatiquement (`API-Gateway-Execution-`

`Logs_{rest-api-id}/{stage_name}`), tant que la journalisation des exécutions est toujours activée sur la scène.

Supprimer une livraison de journal (AWS CLI)

Pour supprimer une livraison, vous supprimez la livraison, puis supprimez éventuellement la source et la destination de la livraison.

1. Supprimer la livraison

```
aws logs delete-delivery --id {delivery-id}
```

2. (Facultatif) Supprimer la source de diffusion

```
aws logs delete-delivery-source --name my-apigw-source
```

3. (Facultatif) Supprimer la destination de livraison

```
aws logs delete-delivery-destination --name my-log-destination
```

> **Note**
>
> La suppression de la source ou de la destination de livraison est facultative. Vous pouvez les réutiliser pour créer de nouvelles livraisons ultérieurement.

Supprimer une livraison de journal (console API Gateway)

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Choisissez votre API REST, puis choisissez Stages.
3. Choisissez une étape.
4. Dans la section Journaux et suivi, sous Destinations de livraison des journaux, sélectionnez la destination que vous souhaitez supprimer.
5. Sélectionnez Supprimer.

Maintenir la continuité des journaux

> **Tip**
>
> Pour conserver les journaux dans le même groupe de journaux et conserver les paramètres de conservation et les filtres d'abonnement existants, définissez le groupe de journaux géré automatiquement (`API-Gateway-Execution-Logs_{rest-api-id}/{stage_name}`) comme destination de diffusion. Les noms des flux de journaux seront différents de ceux créés par la journalisation standard.

Journalisation des accès pour les API REST

Dans la journalisation des accès, vous enregistrez qui a accédé à votre API et comment l'appelant a accédé à l'API. Vous pouvez créer votre propre groupe de journaux ou choisir un groupe de journaux existant. Pour spécifier les détails d'accès, vous devez sélectionner des variables `$context`, un format de journal et une destination de groupe de journaux.

Formats de journal d'accès

Le format du journal d'accès doit inclure au moins `$context.requestId` ou `$context.extendedRequestId`. Il est recommandé d'inclure les deux dans le format de votre journal.

`$context.requestId`

Cela journalise la valeur dans l'en-tête `x-amzn-RequestId`. 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 ID de demande écrasés qui ne sont pas au format UUID par *`UUID`*`_REPLACED_INVALID_REQUEST_ID` dans vos journaux d'accès.

`$context.extendedRequestId`

extendedRequestID est un ID 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 vous aider à résoudre les problèmes liés à votre API. Pour plus d'informations, consultez the section called "Variables pour la journalisation des accès pour API Gateway".

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 `accessLogSettings/destinationArn` propriété 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 un format choisi dans la `accessLogSetting/format` propriété 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
```

Configurer la journalisation des accès à l'aide de la console API Gateway

Avant de configurer la journalisation des accès, déployez l'API sur une étape et configurez 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 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.
3. Dans le volet de navigation principal, choisissez Étapes.
4. Dans la section Journaux et suivi, choisissez Modifier.
5. Activez Journalisation des accès personnalisée.
6. 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`.
7. 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 "Formats de journal d'accès".
8. Cliquez sur Enregistrer les modifications.

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.

Configurez la journalisation des accès à l'aide CloudFormation

Utilisez l'exemple de CloudFormation modèle suivant pour créer un groupe de CloudWatch journaux Amazon Logs et configurer la journalisation 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 le Guide de l'utilisateur AWS CloudFormation .

```
 TestStage:
   Type: AWS::ApiGateway::Stage
   Properties:
     StageName: test
     RestApiId: !Ref MyAPI
     DeploymentId: !Ref Deployment
     AccessLogSetting:
       DestinationArn: !GetAtt MyLogGroup.Arn
       Format: $context.extendedRequestId $context.identity.sourceIp
$context.identity.caller $context.identity.user [$context.requestTime]
"$context.httpMethod $context.resourcePath $context.protocol" $context.status
$context.responseLength $context.requestId
 MyLogGroup:
   Type: AWS::Logs::LogGroup
   Properties:
     LogGroupName: !Join
       - '-'
       - - !Ref MyAPI
         - access-logs
```

Journalisation des appels d'API REST adressés à Amazon Data Firehose dans API Gateway

Pour déboguer les problèmes d'accès client à votre API, vous pouvez journaliser les appels d'API adressés à Amazon Data Firehose. Pour plus d'informations sur Firehose, consultez What Is 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 des journaux Firehose pour API Gateway
- Autorisations pour la journalisation de Firehose
- Configuration de la journalisation des accès de Firehose à l'aide de la console API Gateway

Formats des journaux 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 de Firehose est activée pour 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` et la politique 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 flux Firehose doit être `amazon-apigateway-{your-stream-name}`.

Configuration de la journalisation des accès de Firehose à l'aide de la console API Gateway

Pour configurer la journalisation d'API, vous devez avoir déployé l'API dans une étape. Vous devez également avoir créé un flux 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 pour un flux Firehose :
   a. Activez Journalisation des accès personnalisée.
   b. Pour ARN de destination des journaux 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 flux 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 “Formats de journal d'accès”.

6. Sélectionnez Enregistrer les modifications.

API Gateway est maintenant prêt à journaliser dans Firehose 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 les variables de l’étape.

Variables pour la journalisation des accès pour API Gateway

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 pouvez utiliser les variables `$context` suivantes (sensibles à la casse).

Pour obtenir la liste des variables de référence pour les transformations de données, consultez the section called “Variables pour les transformations de données”.

Paramètre	Description
`$context.accountId`	ID de AWS compte du propriétaire de l'API.
`$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 de statut renvoyé à la suite d’une tentative d’autorisation.

Paramètre	Description
`$context.authorizer.claims.` *`property`*	Propriété des requêtes renvoyées depuis le groupe d'utilisateurs Amazon Cognito une fois que l'appelant de la méthode a été authentifié avec succès. Pour de plus amples informations, veuillez consulter the section called "Utilisation d'un groupe d'utilisateurs Amazon Cognito en tant que mécanisme d'autorisation pour une API REST". **Note** L'appel de `$context.authorizer.claims` renvoie la valeur null.
`$context.authorizer.error`	Message d'erreur renvoyé par un mécanisme d'autorisation.
`$context.authorizer.integrationLatency`	Latence d'intégration du mécanisme d'autorisation en millisecondes (ms).
`$context.authorizer.integrationStatus`	Code de statut renvoyé par un mécanisme d'autorisation Lambda.
`$context.authorizer.latency`	Latence du mécanisme d'autorisation en millisecondes (ms).
`$context.authorizer.principalId`	Identifiant utilisateur principal associé au jeton envoyé par le client et retourné par un mécanisme d'autorisation Lambda API Gateway (anciennement appelé Custom Authorizer). Pour de plus amples informations, veuillez consulter the section called "Utilisation des mécanismes d'autorisation Lambda".

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"`. En effet*`property`*, le seul caractère spécial pris en charge est le trait de soulignement(`_`). Pour de plus amples informations, veuillez consulter the section called "Utilisation des mécanismes d'autorisation Lambda".
`$context.authorizer.requestId`	ID de demande du AWS point de terminaison.
`$context.authorizer.status`	Code de statut renvoyé par un mécanisme d'autorisation.
`$context.authenticate.error`	Message d'erreur renvoyé à la suite d'une tentative d'authentification.
`$context.authenticate.latency`	Latence d'authentification en millisecondes.

Paramètre	Description
`$context.authenticate.status`	Code de statut renvoyé à la suite d'une tentative d'authentification.
`$context.awsEndpointRequestId`	ID de demande du AWS point de terminaison.
`$context.cipherSuite`	Chiffre, au format IANA, négocié lors de la prise de contact TLS entre le client et API Gateway.
`$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".
`$context.customDomain.routingRuleIdMatched`	Règle de routage à laquelle une demande entrante a correspondu. Applicable lorsqu'un client utilise un nom de domaine personnalisé pour accéder à une API. Pour en savoir plus, veuillez consulter la section the section called "règles de routage".
`$context.deploymentId`	ID de déploiement de l'API.
`$context.domainName`	Nom de domaine complet utilisé pour invoquer l'API. Il doit être identique à l'en-tête `Host` entrant.
`$context.domainPrefix`	Première étiquette de `$context.domainName`.

Paramètre	Description
`$context.endpointType`	Type de point de terminaison de l'API.
`$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".
`$context.error.validationErrorString`	Chaîne contenant un message d'erreur de validation détaillé.
`$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.

Paramètre	Description
`$context.httpMethod`	Méthode HTTP utilisée. Les valeurs valides sont les suivantes : `DELETE`, `GET`, `HEAD`, `OPTIONS`, `PATCH`, `POST` et `PUT`.
`$context.identity.accountId`	L'ID de AWS compte associé à la demande.
`$context.identity.apiKey`	Pour les méthodes d'API qui nécessitent une clé d'API, cette variable est la clé d'API associée à la demande de méthode. Pour les méthodes qui ne nécessitent aucune clé d'API, cette variable est null. Pour de plus amples informations, veuillez consulter the section called "Plans d'utilisation".
`$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
`$context.identity.cognitoAuthenticationProvider`	Liste séparée par des virgules de tous les fournisseurs d'authentification Amazon Cognito utilisés par l'appelant à l'origine de la demande. Disponible uniquement si la demande a été signée avec les informations d'identification Amazon Cognito. Par exemple, pour une identité provenant d'un groupe d'utilisateurs Amazon Cognito, `cognito-idp.`*`region`*`.amazonaws.com/`*`user_pool_id`*`,cognito-idp.`*`region`*`.amazonaws.com/`*`user_pool_id`*`:CognitoSignIn:`*`token subject claim`* Pour plus d'informations sur les fournisseurs d'authentification Amazon Cognito disponibles, consultez Using Federated Identities dans le Guide du développeur Amazon Cognito.
`$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.
`$context.identity.sourceIp`	Adresse IP source de la connexion TCP immédiate envoyant la demande au point de terminaison API Gateway.
`$context.identity.clientCer t.clientCertPem`	Le certificat PEM-encoded client 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.
`$context.identity.clientCer t.subjectDN`	Nom distinctif de l'objet du certificat présenté par un client. Présent lorsqu'un client accède à une API à l'aide d'un nom de domaine personnalisé pour lequel l'authentification TLS mutuelle est activée.
`$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.
`$context.identity.clientCer t.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'authent ification TLS mutuelle est activée.

Paramètre	Description
`$context.identity.clientCert.validity.notBefore`	Date avant laquelle le certificat n'est pas valide. Présent lorsqu'un client accède à une API à l'aide d'un nom de domaine personnalisé pour lequel l'authentification TLS mutuelle est activée.
`$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.
`$context.identity.vpcId`	ID du VPC qui fait la demande au point de terminaison API Gateway.
`$context.identity.vpceId`	ID du point de terminaison de VPC qui envoie la demande au point de terminaison API Gateway. Présent uniquement lorsque vous disposez d'une API privée.
`$context.identity.user`	Identifiant principal de l'utilisateur qui sera autorisé à accéder aux ressources. Pris en charge pour les ressources qui utilisent l'autorisation IAM.
`$context.identity.userAgent`	En-tête `User-Agent` de l'appelant d'API.
`$context.identity.userArn`	ARN (Amazon Resource Name) de l'utilisateur identifié après l'authentification. Pour de plus amples informations, veuillez consulter https://docs.aws.amazon.com/IAM/latest/UserGuide/id_users.html.
`$context.integration.error`	Message d'erreur renvoyé à partir d'une intégration.

Paramètre	Description
`$context.integration.integrationStatus`	Pour l'intégration du proxy Lambda, le code d'état est renvoyé par le code de fonction Lambda principal AWS Lambda, et non par le code de fonction Lambda principal.
`$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.responseTransferMode`	Le mode de transfert des réponses de votre intégration. Il peut correspondre à `BUFFERED` ou `STREAMED`.
`$context.integration.status`	Code de statut renvoyé à partir d’une intégration. Pour les intégrations de proxy Lambda, code de statut que votre code de fonction Lambda renvoie.
`$context.integration.timeToAllHeaders`	Le délai entre le moment où API Gateway établit la connexion d'intégration et le moment où il reçoit tous les en-têtes de réponse d'intégration du client.
`$context.integration.timeToFirstContent`	Le délai entre le moment où API Gateway établit la connexion d'intégration et le moment où il reçoit les premiers octets de contenu.
`$context.integrationLatency`	Latence d’intégration en millisecondes (ms).

Paramètre	Description
`$context.integrationStatus`	Pour l'intégration du proxy Lambda, ce paramètre représente le code d'état renvoyé par le code de fonction Lambda principal AWS Lambda, et non par le code de fonction Lambda principal.
`$context.isCanaryRequest`	Renvoie `true` si la demande a été dirigée vers le canary et `false` si la demande n'a pas été dirigée vers le canary. Présent uniquement lorsqu'un canary a été activé.
`$context.path`	Chemin d'accès de la demande. Par exemple, pour une URL de demande autre que de proxy de `https://{rest-api-id}.execute-api.{region}.amazonaws.com/{stage}/root/child` , la valeur `$context.path` est `/{stage}/root/child` .
`$context.protocol`	Protocole de demande, par exemple, `HTTP/1.1`. **Note** Les API API Gateway peuvent accepter des HTTP/2 demandes, mais API Gateway envoie des demandes aux intégrations de backend à l'aide de. HTTP/1.1 Par conséquent, le protocole de demande est enregistré HTTP/1.1 même si un client envoie une demande 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.
`$context.requestOverride.header.` *`header_name`*	Remplacement de l'en-tête de la requête. Si ce paramètre est défini, il contient les en-têtes à utiliser à la place des HTTP Headers (En-têtes HTTP) qui sont définis dans le volet Integration Request (Demande d'intégration). Pour de plus amples informations, veuillez consulter Remplacement des codes d'état et des paramètres de demande et de réponse des API REST dans API Gateway.
`$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 de plus amples informations, veuillez consulter Remplacement des codes d'état et des paramètres de demande et de réponse des API REST dans API Gateway.

Paramètre	Description
`$context.requestOverride.querystring.` *querystring_name*	Remplacement de la chaîne d'interrogation de la requête. Si ce paramètre est défini, il contient les chaînes d'interrogation de la requête à utiliser à la place des URL Query String Parameters (Paramètres de chaîne de requête d'URL) qui sont définis dans le volet Integration Request (Demande d'intégration). Pour de plus amples informations, veuillez consulter Remplacement des codes d'état et des paramètres de demande et de réponse des API REST dans API Gateway.
`$context.responseLatency`	Latence de la réponse en millisecondes (ms).
`$context.responseLength`	La longueur de la charge utile de la réponse en octets.
`$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 de plus amples informations, veuillez consulter Remplacement des codes d'état et des paramètres de demande et de réponse des API REST dans API Gateway.

Paramètre	Description
`$context.responseOverride.status`	Remplacement du code de statut de la réponse. Si ce paramètre est défini, il contient le code du statut qui est renvoyé à la place du Method response status (Statut de la réponse de méthode) qui est défini comme le Default mapping (Mappage par défaut) dans le volet Integration Response (Réponse d'intégration). Pour de plus amples informations, veuillez consulter Remplacement des codes d'état et des paramètres de demande et de réponse des API REST dans API Gateway.
`$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 de plus amples informations, veuillez consulter Didacticiel : création d'une API REST avec une intégration HTTP sans proxy.
`$context.stage`	Étape de déploiement de la demande d'API (par exemple, `Beta` ou `Prod`).
`$context.status`	Statut de la réponse de la méthode.

Paramètre	Description
`$context.tlsVersion`	Version TLS négociée lors de la prise de contact TLS entre le client et API Gateway.
`$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.
`$context.xrayTraceId`	L'ID de trace de la X-Ray trace. Pour de plus amples informations, veuillez consulter the section called “Configurez AWS X-Ray”.
`$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 de plus amples informations, veuillez consulter 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 de plus amples informations, veuillez consulter the section called “AWS WAF”.

Suivi des demandes des utilisateurs adressées aux API REST à l’aide de X-Ray dans API Gateway

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 Régions AWS où X-Ray est disponible.

Comme X-Ray vous offre une vue de bout en bout de la totalité d'un processus de demande, vous pouvez analyser les latences de vos API et de leurs services backend. 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 avec 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 API Gateway REST APIs
- Utilisation AWS X-Ray cartes de service et vues de suivi avec API Gateway
- Configuration des règles AWS X-Ray d'échantillonnage pour API Gateway APIs
- AWS X-Ray traces pour Amazon API Gateway APIs

Configuration AWS X-Ray avec API Gateway REST APIs

Dans cette section, vous trouverez des informations détaillées sur la configuration AWS X-Rayavec API Gateway REST APIs.

Rubriques

- Modes de suivi X-Ray pour API Gateway
- Autorisations pour le suivi X-Ray
- Activation du suivi X-Ray dans la console API Gateway
- Activation du AWS X-Ray suivi à l'aide de l'API Gateway CLI

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` et la politique 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**
  >
  > 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` et la politique 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.

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 dans 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

La commande create-stage suivante crée une étape avec un suivi X-Ray actif :

```
aws apigateway create-stage \
    --rest-api-id rest-api-id \
    --stage-name stage-name \
    --deployment-id deployment-id \
    --region region \
    --tracing-enabled=true
```

Le résultat se présente comme suit :

```
{
    "tracingEnabled": true,
    "stageName": stage-name,
    "cacheClusterEnabled": false,
    "cacheClusterStatus": "NOT_AVAILABLE",
    "deploymentId": deployment-id,
    "lastUpdatedDate": 1533849811,
    "createdDate": 1533849811,
    "methodSettings": {}
}
```

La commande create-stage suivante crée une étape sans suivi X-Ray actif :

```
aws apigateway create-stage \
    --rest-api-id rest-api-id \
    --stage-name stage-name \
    --deployment-id deployment-id \
    --region region \
```

```
    --tracing-enabled=false
```

Le résultat se présente comme suit :

```
{
    "tracingEnabled": false,
    "stageName": stage-name,
    "cacheClusterEnabled": false,
    "cacheClusterStatus": "NOT_AVAILABLE",
    "deploymentId": deployment-id,
    "lastUpdatedDate": 1533849811,
    "createdDate": 1533849811,
    "methodSettings": {}
}
```

La commande update-stage suivante active le suivi X-Ray pour une API déployée :

```
aws apigateway update-stage \
    --rest-api-id rest-api-id \
    --stage-name stage-name \
    --patch-operations op=replace,path=/tracingEnabled,value=true
```

La commande update-stage suivante désactive le suivi X-Ray pour une API déployée :

```
aws apigateway update-stage \
    --rest-api-id rest-api-id \
    --stage-name stage-name \
    --region region \
    --patch-operations op=replace,path=/tracingEnabled,value=false
```

Le résultat se présente comme suit :

```
{
    "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'API X-Ray avec la AWS CLI.

Utilisation AWS X-Ray cartes de service et vues de suivi 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.

Rubriques

- Exemple de plan X-Ray de service
- Exemple de vue de X-Ray suivi

Exemple de plan X-Ray de service

AWS X-Ray les cartes de service affichent des informations sur votre API et tous ses services en aval. Lorsque cette option X-Ray est activée pour une étape d'API dans API Gateway, vous verrez un nœud dans la carte des services contenant des informations sur le temps total passé dans le service API Gateway. 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.

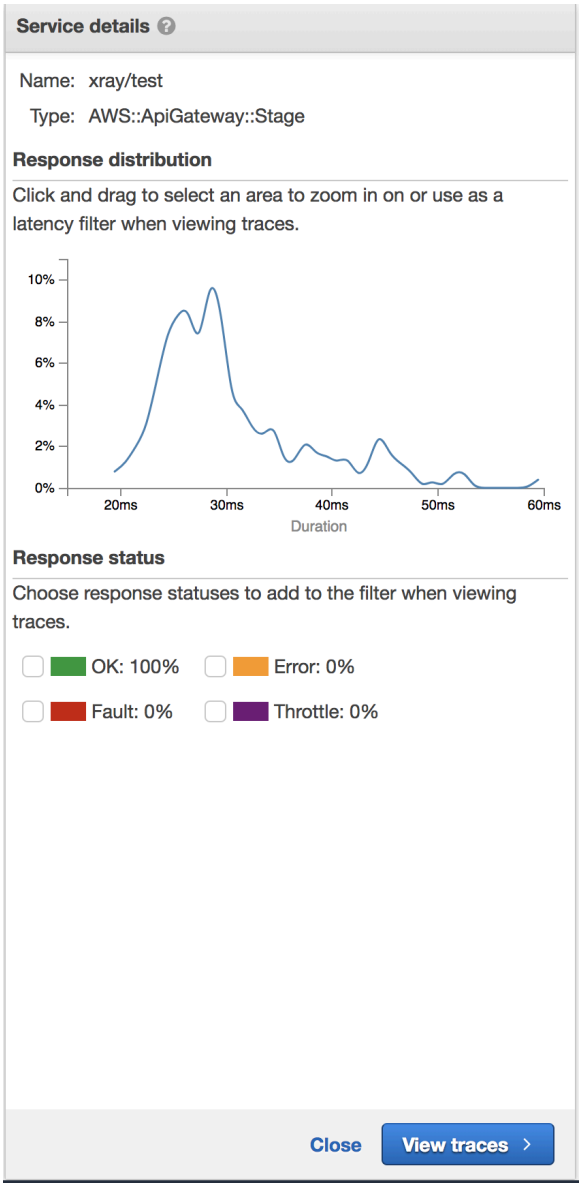
L'exemple suivant montre une cartographie des services pour l'étape `test` d'une API appelée `xray`. Cette API possède deux intégrations Lambda. Les nœuds représentent le service API Gateway et les deux fonctions Lambda.

Pour une explication détaillée de la structure de la carte de service, voir Utiliser la carte de X-Ray suivi.



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.

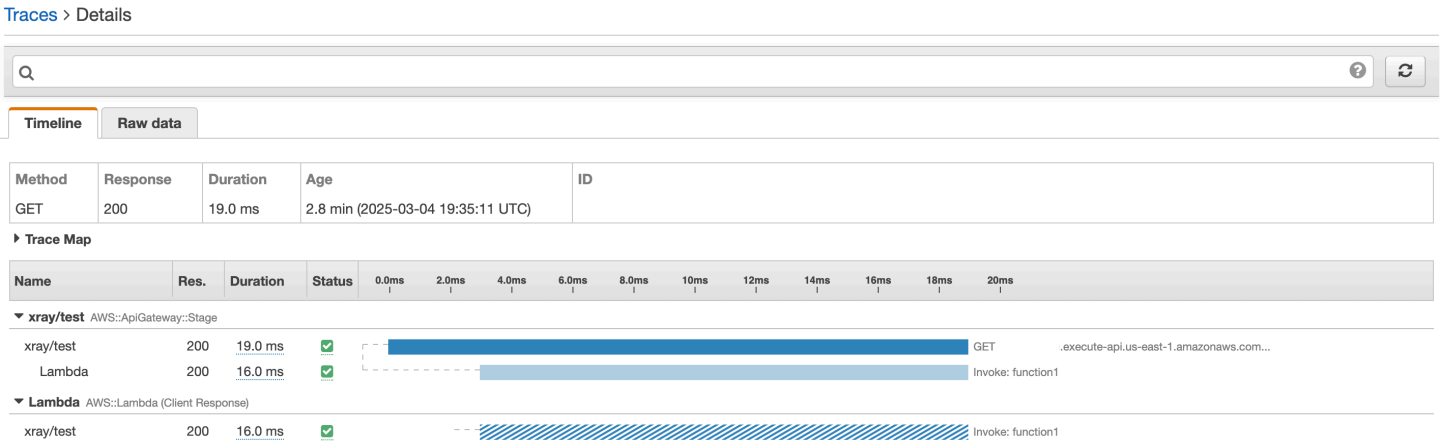
Si vous choisissez un nœud ou une arête sur une carte de X-Ray service, la X-Ray console affiche un histogramme de distribution de latence. Vous pouvez utiliser un histogramme de latence pour voir de combien de temps a besoin un service pour traiter ses requêtes. Voici un histogramme de l'étape d'API Gateway nommée `xray/test` dans la précédente carte de service. Pour obtenir une explication détaillée des histogrammes de distribution des latences, consultez Use Latency Histograms.



Exemple de vue de X-Ray suivi

Le schéma suivant présente une vue du suivi générée pour l'exemple d'API décrit ci-dessus, avec une fonction de backend 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, consultez View traces and trace details.



Configuration des règles AWS X-Ray d'échantillonnage pour API Gateway APIs

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, consultez les rubriques suivantes dans le Guide du développeur X-Ray :

- Configure sampling rules
- Utilisation de règles d’échantillonnage avec l’API X-Ray

Rubriques

- Valeurs des options des règles d'échantillonnage X-Ray pour API Gateway APIs
- Exemples de règles d’échantillonnage X-Ray

Valeurs des options des règles d'échantillonnage X-Ray pour API Gateway APIs

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 obtenir de plus amples informations et une explication détaillée de l’utilisation des paramètres Réservoir et Fréquence, consultez Configure sampling rules.

- 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 — **`{}`**

AWS X-Ray traces pour Amazon API Gateway APIs

Cette section décrit les segments de AWS X-Ray trace, les sous-segments et les autres champs de suivi pour Amazon API Gateway APIs.

Avant de lire cette section, consultez les rubriques suivantes dans le Guide du développeur X-Ray :

- Utilisez un Console de gestion AWS
- X-Ray segment documents

- Concepts

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 Configure sampling rules.

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"
}
```

Pour plus d'informations sur les annotations, consultez X-Ray segment documents, puis choisissez Documents de segment X-Ray, Annotations.

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 Configure sampling rules.

```
"aws": {
    "xray": {
        "sampling_rule_name": "Default"
    },
    "api_gateway": {
        "account_id": "123412341234",
```

```
        "rest_api_id": "a1b2c3d4e5",
        "stage": "dev",
        "request_id": "a1b2c3d4-a1b2-a1b2-a1b2-a1b2c3d4e5f6"
    }
}
```

Pour plus d'informations sur l'objet `aws`, consultez X-Ray segment documents, puis choisissez Documents de segment X-Ray, Données de ressources AWS .

Présentation des suivis

Voici un segment de suivi pour une étape d'API Gateway. Pour obtenir une explication détaillée des champs qui constituent le segment de suivi, consultez Documents de segment X-Ray.

```
        {
            "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"
        }
```

Portails API Gateway

Vous pouvez utiliser un portail API Gateway pour partager votre documentation APIs et celle des API avec les utilisateurs d'API sur une page Web. Un portail contient un ensemble de produits de portail. Chaque produit de portail est un regroupement logique de REST APIs et contient la documentation que vous créez et publiez pour vos utilisateurs d'API. Les pages de produits d'un portail contiennent la documentation personnalisée au niveau du produit du portail. Les pages de point de terminaison REST du produit contiennent la documentation de chacun des REST APIs ainsi que le chemin, la méthode et l'étape d'une API REST. La combinaison des pages produit et des pages de point de terminaison REST du produit fournit la documentation complète permettant aux utilisateurs d'API d'apprendre comment commencer à utiliser votre REST APIs.

Utilisez un portail pour effectuer les opérations suivantes :

- Utilisez API Gateway pour l'ensemble du cycle de vie de vos API, de leur création APIs à leur documentation et à leur distribution.
- Partagez les produits du portail entre différents AWS comptes.
- Personnalisez et partagez un emplacement central pour les produits de votre portail et fournissez des pages de produits.

Considérations

Les considérations suivantes peuvent avoir un impact sur votre utilisation des portails API Gateway :

- Vous ne pouvez créer des portails que pour partager REST APIs.
- Les portails sont créés au niveau de la AWS région. Votre portail ne peut contenir que des fichiers REST situés dans la même région APIs que celle dans laquelle vous avez créé le portail.
- Vous ne pouvez utiliser le Console de gestion AWS AWS CLI, ou que AWS SDKs pour créer et partager vos portails.
- Vous ne pouvez contrôler l'accès à votre portail qu'à l'aide des groupes d'utilisateurs Amazon Cognito. Les utilisateurs de votre portail peuvent soit se connecter directement par le biais d'un groupe d'utilisateurs, soit se fédérer via un fournisseur d'identité (IdP) tiers. Si vous utilisez un groupe d'utilisateurs Amazon Cognito, vous ne pouvez pas définir de secret client pour votre groupe d'utilisateurs. Pour plus d'informations, consultez la section Groupes d'utilisateurs Amazon Cognito.

Produits du portail dans API Gateway

Un produit de portail représente un service ou une fonctionnalité que vous souhaitez partager. Le produit de votre portail est un ensemble de points de terminaison REST et de pages de produits. Les points de terminaison REST du produit sont les points d'accès à votre produit de portail. Ils comprennent le chemin et la méthode d'une API REST ainsi que l'étape sur laquelle elle est déployée. Les pages de produits sont des documents que vous fournissez pour expliquer comment les utilisateurs d'API peuvent utiliser les points de terminaison de vos produits. Un produit de portail peut contenir l'intégralité de la `Prod` phase sous la forme d'un ensemble de points de terminaison REST du produit ou simplement la `GET /pets` ressource déployée sur la `Prod` scène sous la forme d'un point de terminaison REST de produit unique.

Votre produit de portail est personnalisable. Vous pouvez ajouter une documentation personnalisée, renommer les points de terminaison REST du produit, réorganiser l'ordre d'affichage, ajouter de nouvelles sections et partager des produits entre comptes. AWS Pour que les modifications que vous apportez à votre produit de portail prennent effet, vous devez republier tous les portails qui utilisent votre produit de portail.

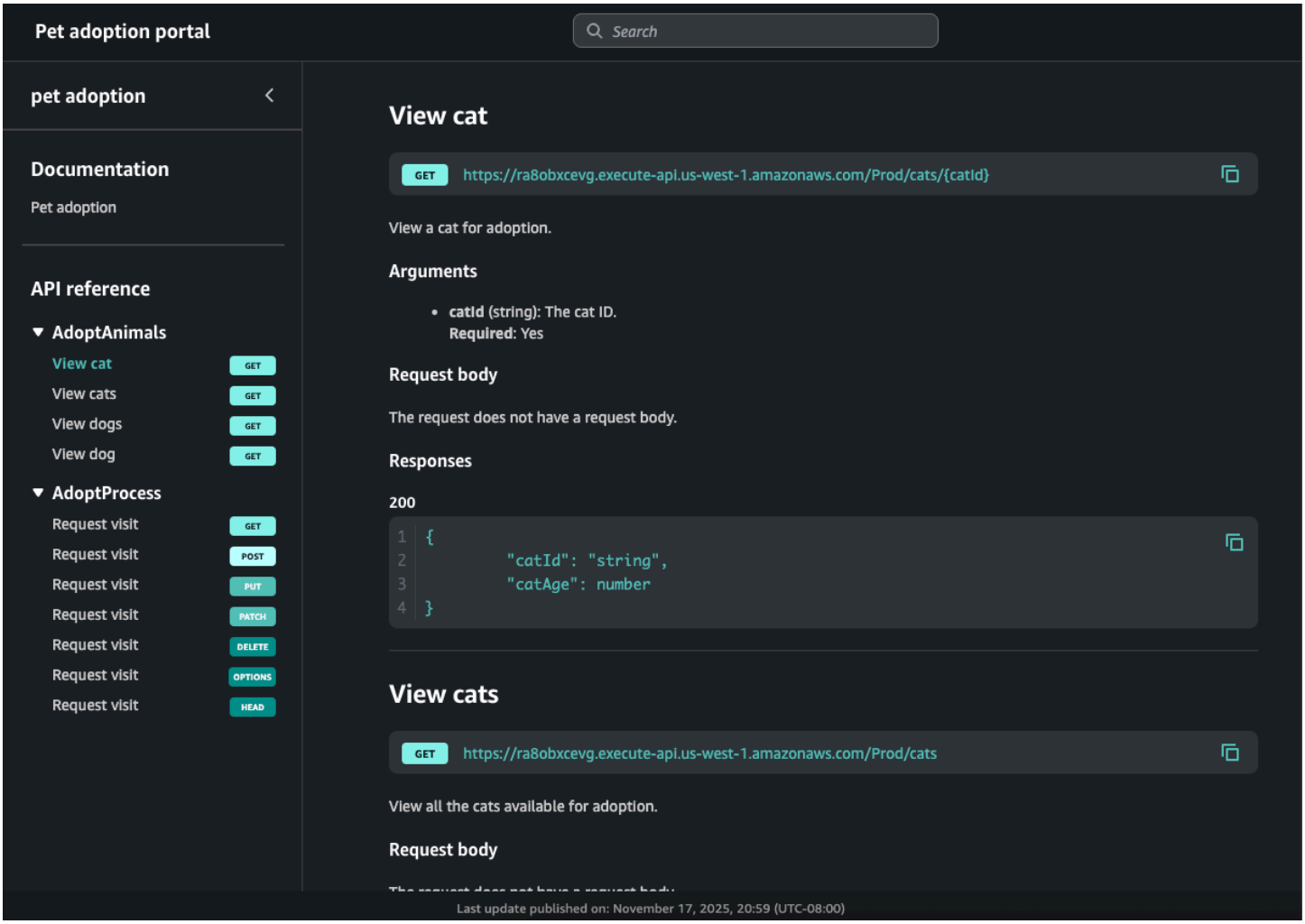
Exemple de produit de portail d'adoption d'animaux de compagnie

Par exemple, plusieurs API REST peuvent représenter un service d'adoption d'animaux de compagnie. Vous pouvez utiliser API Gateway pour créer un produit de `pet adoption` portail. Ce produit de portail aiderait les clients à découvrir les API qu'ils devraient utiliser pour rencontrer et adopter des animaux de compagnie. Ce produit de portail utilise des API REST que vous avez déjà créées, mais il vous permet de les regrouper et de les organiser. Vous pouvez également fournir de la documentation sur les conditions générales d'utilisation de votre produit de portail d'adoption pour animaux de compagnie et permettre aux clients d'essayer vos API. Ces informations sont toutes stockées dans votre produit de portail.

Le tableau suivant présente trois API qui représentent un produit de portail d'adoption pour animaux de compagnie, ainsi que les noms d'opérations de point de terminaison REST et les noms de section de page correspondants.

IDENTIFIANT DE L'API REST	Chemin et méthode de l'API REST	Étape de l'API REST	Nom de l'opération	Section de page
kf5387miad	GET /dogs	Prod	View dogs	AdoptAnimals
kf5387miad	GET /dogs/{dogId}	Prod	View dog	AdoptAnimals
ra8obxcevg	GET /cats	Prod	View cats	AdoptAnimals
ra8obxcevg	GET /cats/{catId}	Prod	View cat	AdoptAnimals
h0rpx9cm62	ANY /user/{userId}/{petId+}	Bêta	Request visit	AdoptProcess

Dans cet exemple, deux API REST, `ra8obxcevg` et`kf5387miad`, sont regroupées dans la `AdoptAnimals` section. Le résultat de cette navigation de commande ressemblerait à ce qui suit dans un portail :



Comme il s'agit d'une ANY méthode fourre-tout, le portail affiche toutes les méthodes HTTP prises en charge. Ce portail contient également de la documentation créée par le propriétaire du produit du portail.

Étapes suivantes

Pour commencer à utiliser les produits du portail, vous pouvez effectuer les opérations suivantes :

- Pour créer un produit de portail, voirthe section called “Création d’un produit de portail”.
- Pour en savoir plus sur la fonctionnalité d'essai, voirthe section called “Activez Essayez-le pour un point de terminaison REST du produit sur votre portail”.
- Pour en savoir plus sur les pages de produits, consultezthe section called “Création d'une page produit”.
- Pour en savoir plus sur le partage de votre produit de portail, consultezthe section called “Partagez les produits du portail”.

Après avoir créé un produit de portail, vous pouvez le publier sur un portail. Pour de plus amples informations, veuillez consulter the section called “Création d'un portail”.

Création d'un produit de portail dans API Gateway

La procédure suivante indique comment créer un produit de portail. Un portail est un ensemble de produits de portail. Après avoir créé votre produit de portail, vous créez des points de terminaison REST et des pages de produits. Pour en savoir plus sur les produits du portail, consultezthe section called “Produit de portail”.

Considérations

Les considérations suivantes peuvent avoir une incidence sur la façon dont vous créez un produit de portail :

- Votre produit de portail peut contenir des API REST privées et publiques. Les API privées ne sont pas prises en charge pour la fonctionnalité d'essai et ont donc une incidence visuelle sur votre portail. En tant que propriétaire du portail, vous devrez peut-être fournir une documentation expliquant cela.
- Si vous créez votre produit de portail à l' AWS CLI aide AWS des SDK, votre portail ne comportera aucun point de terminaison ni aucune page de produit. Vous devez ajouter ces ressources à l'aide de la console AWS CLI or. Pour savoir comment créer un point de terminaison REST pour un produit, consultezthe section called “Création d'un point de terminaison REST du produit”. Pour savoir comment créer une page produit, consultezthe section called “Création d'une page produit”.

Création d’un produit de portail

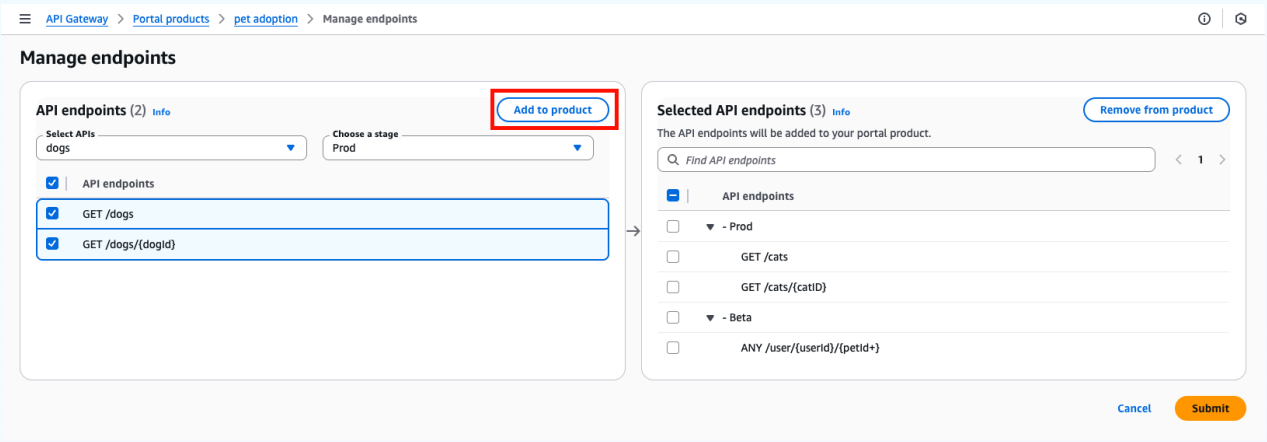
La procédure suivante indique comment créer un produit de portail.

Pour créer un produit de portail

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Dans le volet de navigation principal, sélectionnez Portal products.
3. Choisissez Créer un produit.
4. Dans Nom du produit, entrez le nom du produit de votre portail.
5. Pour Description du produit, entrez une description.

6. Choisissez Suivant.
7. Pour sélectionner les points de terminaison REST de votre produit, sous Points de terminaison d'API, choisissez une API, puis choisissez une étape.
8. Pour ajouter un point de terminaison aux points de terminaison REST de votre produit, sélectionnez le point de terminaison de l'API, puis choisissez Ajouter au produit.

> **Note**
>
> Ne choisissez pas Suivant sans d'abord sélectionner Ajouter au produit.



Le point de terminaison de l'API apparaîtra dans la liste des points de terminaison de l'API sélectionnés.

9. Choisissez Suivant.
10. Vérifiez votre sélection et choisissez Créer un produit.

Une fois que vous avez créé votre produit de portail à l'aide de la console, toutes vos pages de produit et les pages de point de terminaison REST de vos produits sont des brouillons et n'apparaîtront pas dans un portail. Pour que vos pages de produits et les pages de point de terminaison REST de vos produits soient visibles par les consommateurs, vous devez ajouter votre brouillon à une section. Si vous créez votre produit de portail à l'aide AWS des SDK AWS CLI ou, vous ajoutez le brouillon à la section de la AWS CLI commande. Quelle que soit la manière dont vous ajoutez votre brouillon à une section, vous devez publier le portail qui utilise les produits de votre portail pour qu'il soit visible par les consommateurs.

Pour ajouter votre brouillon à une section de page

1. Vos brouillons sont répertoriés dans l'onglet Documentation. Il existe des pages de documentation de projet pour vos pages de produit et des pages de référence d'API de brouillon pour les pages de point de terminaison REST de votre produit. Choisissez Draft API reference pages.
2. Choisissez un brouillon de page de référence d'API.

   Si vous n'avez aucun point de terminaison REST de produit, vous n'aurez aucun brouillon de pages de référence d'API. Pour savoir comment créer un point de terminaison REST pour un produit, consultezthe section called “Création d'un point de terminaison REST du produit”.
3. Choisissez Modifier la page.
4. Sur cette page, vous pouvez remplacer toute partie de la documentation d'API existante ou utiliser la documentation d'API Gateway. Pour permettre aux consommateurs de voir le contenu de la page de point de terminaison REST de votre produit, entrez un nom sous Nom de section. S'il s'agit du `/pets-GET` point de terminaison, le nom de la page pourrait êtrePets.
5. Sélectionnez Enregistrer les modifications.
6. Le nouveau nom de page que vous avez créé apparaît dans la section des pages de référence de l'API.

   Pour que la nouvelle page soit visible par les consommateurs, vous devez tout de même republier votre portail. Pour de plus amples informations, veuillez consulter the section called “Publier un portail”.

Création d'un point de terminaison REST du produit dans API Gateway

Un point de terminaison REST du produit est un point d'accès à votre produit de portail. Chaque point de terminaison comprend le chemin et la méthode d'une API REST et l'étape sur laquelle elle est déployée. Le point de terminaison inclut toute documentation que vous avez définie pour votre API REST, ou peut être mis à jour avec une documentation personnalisée. Lorsque vous créez le point de terminaison REST de votre produit, vous configurez les paramètres suivants :

- Nom de l'opération du point de terminaison REST du produit. Si votre API REST possède le chemin et la méthode`GET /pets`, dans un portail, le nom est`/pets`. À l'aide du nom de l'opération, vous pouvez remplacer le nom du point de terminaison REST du produit par`Pet summaries`. Cela n'a aucun impact sur votre API existante.

- La fonctionnalité d'essai pour le point de terminaison de votre produit. Cela permet aux utilisateurs d'API de tester votre APIs application sur votre portail. Si vous créez un point de terminaison de produit pour une API privée, la fonctionnalité d'essai est définie sur`UNAVAILABLE`. Pour de plus amples informations, veuillez consulter the section called “Activez Essayez-le pour un point de terminaison REST du produit sur votre portail”.
- Si des parties de documentation sont actuellement définies pour votre API REST, vous pouvez soit importer la documentation, soit remplacer votre documentation existante et créer une nouvelle documentation au niveau du produit à l'aide GitHub de Flavored Markdown. Les modifications de documentation ne sont pas répercutées dans les parties de la documentation de votre API API Gateway. Si vous importez les parties de votre documentation actuelle, API Gateway synchronise les modifications apportées à votre API avec le point de terminaison REST de votre produit.

Création d'un point de terminaison REST du produit

La procédure suivante montre comment ajouter un point de terminaison REST de produit à un produit de portail existant. Pour savoir comment créer ensemble un produit de portail et un point de terminaison REST de produit, voirthe section called “Création d’un produit de portail”.

Pour créer un point de terminaison REST de produit pour un portail de produit

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Dans le volet de navigation principal, sélectionnez Portal products.
3. Choisissez un produit de portail.
4. Choisissez l'onglet Points de terminaison associés, puis choisissez Ajouter des points de terminaison.
5. Pour sélectionner les points de terminaison REST de votre produit, sous Points de terminaison d'API, choisissez une API, puis choisissez une étape.
6. Pour ajouter un point de terminaison aux points de terminaison REST de votre produit, sélectionnez le point de terminaison de l'API, puis choisissez Ajouter au produit.

   Le point de terminaison de l'API apparaîtra dans la liste des points de terminaison de l'API sélectionnés.
7. Sélectionnez Envoyer.

8. La page de point de terminaison REST de votre produit est désormais un brouillon. Choisissez Draft API reference pages, puis choisissez la page de point de terminaison REST de votre produit pour finaliser le contenu de la page.
9. Choisissez Modifier la page.
10. Pour remplacer la documentation existante d'API Gateway, dans le champ Source de documentation, activez l'option Remplacer la documentation et procédez comme suit :
    a. Pour Afficher le contenu, choisissez Create override.
    b. Dans Nom de l'opération, entrez un nouveau nom d'opération.
    c. Pour le corps de la page, entrez votre documentation personnalisée.
    d. (Facultatif) Pour Endpoint, entrez un point de terminaison personnalisé qui apparaît sur votre portail.
11. Pour permettre aux clients d'invoquer votre API sur votre portail, sélectionnez la fonctionnalité Essayer. Pour de plus amples informations, veuillez consulter the section called “Activez Essayez-le pour un point de terminaison REST du produit sur votre portail”.
12. Dans Nom de section, entrez un nom ou choisissez une section existante. Lorsque vous ajoutez le point de terminaison REST de votre produit à une section, les consommateurs peuvent consulter votre contenu sur un portail publié.
13. Sélectionnez Enregistrer les modifications.

Pour que la nouvelle page soit visible par les consommateurs, vous devez tout de même republier votre portail. Pour de plus amples informations, veuillez consulter the section called “Publier un portail”.

Création d'une page produit dans API Gateway

Une page produit aide les consommateurs à comprendre et à utiliser vos produits. La documentation se situe au niveau du produit, de sorte que tous les portails qui incluent votre produit disposent de cette documentation. API Gateway prend en charge deux types de pages de produits rédigées en GitHub Flavored Markdown :

Page Présentation

Lorsque vous créez votre produit, API Gateway génère automatiquement une page de présentation de votre produit en fonction des informations que vous fournissez lors de la création

de votre produit. Cette page comporte une mise en forme supplémentaire qui ne peut pas être modifiée. Vous pouvez modifier le texte ou supprimer la page du produit portail.

Pages personnalisées

Une page personnalisée est une page écrite en Markdown. Vous pouvez consulter cette section pour consulter les conditions générales ou suivre un didacticiel sur l'utilisation de votre produit. Vous pouvez transformer n'importe quelle page personnalisée en page d'aperçu. Lorsque vous transformez une page personnalisée en page d'aperçu, le contenu de la page est mis en forme pour correspondre à la page d'aperçu.

Considérations

Les considérations suivantes peuvent avoir une incidence sur votre utilisation des pages de produits :

- Cette documentation n'est pas au niveau de l'API et n'inclut aucune méthode, aucune ressource ou aucun paramètre de demande. Pour savoir comment modifier la documentation au niveau de l'API d'un produit, consultez. the section called “Mettre à jour un terminal REST du produit”
- Vous ne pouvez pas charger directement un `.md` fichier sur votre produit.
- Vous pouvez modifier l'ordre des pages de produits en modifiant la liste des pages dans l'ordre d'affichage de votre produit de portail. Pour de plus amples informations, veuillez consulter the section called “Mettre à jour un produit de portail”.
- Vous devez republier tous les portails qui incluent votre produit pour que les modifications apportées à la documentation prennent effet.
- Si plusieurs portails utilisent le même produit de portail, les pages de produits sont les mêmes sur tous les portails. Pour avoir les mêmes points de terminaison REST mais des pages de produit différentes, créez plusieurs produits.

Création d'une page produit

La procédure suivante indique comment créer une page produit.

Pour créer une page produit

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Dans le volet de navigation principal, sélectionnez Portal products.

3. Choisissez un produit.
4. Sous Documentation, choisissez Créer une page personnalisée.
5. Pour Titre de page, entrez le titre de la page.
6. Pour Body, entrez votre documentation supplémentaire à l'aide de GitHub Flavored Markdown.

   La section Aperçu montre comment votre contenu apparaît sur un portail. Le style visuel final peut changer en fonction des paramètres de votre portail.
7. Choisissez Créer une page de produit.
8. Votre page produit est désormais un brouillon. Choisissez Brouillon de pages de documentation, puis choisissez votre page produit pour finaliser le contenu de la page.
9. Choisissez Modifier la page.
10. Dans Nom de section, entrez un nom ou choisissez une section existante. Lorsque vous ajoutez votre page produit à une section, les consommateurs peuvent consulter votre contenu sur un portail publié.
11. Sélectionnez Enregistrer les modifications.

Pour que la nouvelle page soit visible par les consommateurs, vous devez tout de même republier votre portail. Pour de plus amples informations, veuillez consulter the section called “Publier un portail”.

Mettre à jour un produit de portail dans API Gateway

Lorsque vous mettez à jour un produit de portail, vous pouvez modifier le nom, la description ou l'ordre d'affichage des points de terminaison REST et des pages de produits de votre produit. Pour modifier l'ordre d'affichage, vous devez modifier la liste des pages pour les pages de produits et le nom de section pour les points de terminaison des produits. Si vous le faites à l'aide du AWS CLI ou SDKs, vous spécifiez l'ARN de la page du produit ou l'ARN du point de terminaison REST du produit. Vous devez republier tous les portails qui utilisent vos produits pour les utilisateurs de vos API afin de voir les modifications reflétées dans un portail.

Mettre à jour un produit de portail

La procédure suivante montre comment modifier la liste des sections pour réorganiser les points de terminaison REST du produit tels qu'ils apparaîtront sur un portail. Dans cette procédure, nous partons du principe que vous disposez d'au moins deux points de terminaison REST dans votre produit de portail.

Pour mettre à jour un produit de portail

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Dans le volet de navigation principal, sélectionnez Portal products.
3. Choisissez un produit.
4. Dans l'onglet Documentation, pour les pages de produits, sélectionnez Gérer.
5. Faites glisser et déposez les points de terminaison REST du produit pour les réorganiser. Vous pouvez également déplacer les points de terminaison REST et les pages de produits vers la section des pages de brouillon. Les pages des brouillons ne seront pas visibles sur votre portail.
6. (Facultatif) Renommez les noms des points de terminaison REST du produit ou ajoutez une nouvelle section de référence d'API. Ces modifications n'auront aucun impact sur votre REST APIs.
7. Sélectionnez Enregistrer les modifications.

Mettre à jour le point de terminaison REST d'un produit dans API Gateway

Lorsque vous mettez à jour un point de terminaison REST du produit, vous pouvez modifier tous les paramètres du point de terminaison du produit, à l'exception du chemin et de la méthode d'une API REST et de l'étape sur laquelle elle est déployée. Toutes les modifications apportées aux points de terminaison de vos produits sont mises à jour automatiquement, mais vous devez republier tous les portails qui utilisent vos produits pour que les modifications soient reflétées dans un portail.

Mettre à jour un terminal REST du produit

La procédure suivante montre comment mettre à jour le point de terminaison REST d'un produit afin de remplacer la documentation d'API actuelle par une documentation de référence d'API personnalisée.

Pour mettre à jour un point de terminaison REST du produit

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Dans le volet de navigation principal, sélectionnez Portal products.
3. Choisissez un produit.

4. Dans l'onglet Documentation, sous les pages de référence de l'API, choisissez le nom d'un point de terminaison REST du produit, tel que /dogs - GET.
5. Pour Aperçu, choisissez Modifier la page.
6. Pour Source de documentation, activez Remplacer la documentation.
7. Pour Afficher le contenu, choisissez Remplacer le contenu existant.

   Si vous choisissez Supprimer tout le contenu, celui-ci est supprimé de la page, mais vous pouvez choisir Remplacer le contenu existant pour accéder à nouveau à la documentation.
8. Dans le corps de la page, entrez une documentation de référence d'API personnalisée à l'aide de GitHub Flavored Markdown.

   Cette documentation ne sera pas repropagée vers API Gateway.
9. Sélectionnez Enregistrer les modifications.
10. Vous devez republier tous les portails qui utilisent ce produit de portail pour que les modifications prennent effet.

Mettre à jour une page produit dans API Gateway

Lorsque vous mettez à jour une page produit, vous pouvez modifier tous les paramètres de la page produit. Toutes les modifications apportées à votre page produit sont mises à jour automatiquement, mais vous devez republier tous les portails qui utilisent vos produits pour que les modifications soient reflétées dans un portail.

Partagez les produits du portail dans API Gateway

Vous pouvez partager un produit du portail entre les Comptes AWS utilisateurs AWS RAM. Lorsque vous partagez un produit de portail, l'autre compte peut utiliser votre produit de portail sur son propre portail. Grâce aux produits de portail partagé, vous pouvez créer un catalogue unique des API de votre organisation et appliquer des normes de gouvernance dans l'ensemble de votre écosystème d'API. Dans le même temps, le partage des produits du portail permet aux fournisseurs d'API de développer, de tester et de gérer des API sur leurs propres comptes.

Considérations

Les considérations suivantes peuvent avoir une incidence sur la manière dont vous partagez les ressources du portail :

- Lorsque vous partagez votre produit avec un autre compte, celui-ci ne peut modifier aucune propriété de votre API REST. Cela inclut les points de terminaison de l'intégration, la stratégie d'autorisation ou la configuration de l'étape.
- Lorsque vous ajoutez le produit de portail d'un autre compte à votre portail, le propriétaire du produit de portail ne peut ni afficher ni contrôler aucune autre propriété de votre portail. Le propriétaire du produit du portail sait uniquement que le produit est utilisé sur votre portail.
- Les produits du portail API Gateway sont partagés au Région AWS niveau.
- Vous pouvez utiliser un partage de ressources avec plusieurs principaux, et après avoir créé le partage de ressources, vous pouvez y ajouter d'autres principaux. Nous vous recommandons, dans la mesure du possible, de réutiliser votre partage de ressources.
- Si les deux comptes sont utilisés par la même organisation AWS Organizations, le partage des ressources est automatiquement accepté. Vous devez tout de même créer le partage de ressources à l'aide d' AWS RAM.
- Si les deux comptes appartiennent à la même organisation AWS Organizations et que le partage des ressources au sein de votre organisation est activé, tous les principaux membres de l'organisation avec lesquels vous partagez ont automatiquement accès aux partages de ressources. Aucune invitation n'est nécessaire et vous pouvez ignorer le partage des ressources.
- Si le compte avec lequel vous avez partagé le produit n'accepte pas le partage des ressources dans les 12 heures, vous devez partager à nouveau la ressource.
- Une fois que vous avez créé le partage de ressources, AWS RAM mettez à jour la politique de partage des produits de votre produit afin d'empêcher l'accès aux principaux sans `allow` accès explicite. Pour plus d'informations, consultez Determining whether a request is allowed or denied within an account dans le Guide de l'utilisateur IAM.

  La politique de ressources mise à jour ressemblera à ce qui suit :

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "apigateway:GetProduct",
                "apigateway:ListEndpoints",
                "apigateway:ListPages",
                "apigateway:GetEndpoint",
                "apigateway:GetPage"
```

```
            ],
            "Resource": [
                "arn:aws:apigateway:us-east-1:111122223333:/portalproducts/product-
id",
                "arn:aws:apigateway:us-east-1:111122223333:/portalproducts/product-
id/*"
            ]
        }
    ]
}
```

AWS RAM a empêché les donneurs d'ordre qui n'étaient pas explicitement autorisés à accéder à leur portail d'ajouter votre produit à leur portail, en ajoutant ce qui suit :

```
"StringNotEquals": { "aws:PrincipalAccount": "555555555555" }
```

Pour savoir comment partager un produit, consultezthe section called “Partagez le produit de votre portail avec le propriétaire du portail”. Pour savoir comment ajouter un produit partagé avec vous à votre portail, consultezthe section called “Ajouter un produit de portail partagé à votre portail”.

Partagez le produit de votre portail avec le propriétaire du portail dans API Gateway

En tant que propriétaire de produit, vous pouvez partager votre produit avec un propriétaire de portail d'un autre Compte AWS utilisateur AWS RAM. Vous conservez le contrôle total de toutes les pages de produits et des pages de points de terminaison REST des produits.

Considérations

Les considérations suivantes peuvent avoir une incidence sur la manière dont vous partagez les produits du portail :

- Vous pouvez partager ou annuler le partage des produits de votre portail à tout moment. Si vous annulez le partage d'un produit de portail alors qu'il est utilisé dans un portail publié, le propriétaire du portail ne pourra pas consulter le produit de portail, consulter les mises à jour qui y ont été apportées, republier ou prévisualiser le portail tant qu'il n'aura pas supprimé le produit de portail.
- Vous pouvez consulter les produits du portail que vous avez partagés avec d'autres comptes.
- Vous ne pouvez modifier aucun portail contenant votre produit de portail, sauf si vous créez le portail vous-même.

Partagez le produit de votre portail

La procédure suivante montre comment créer un partage de ressources.

Console de gestion AWS

Pour utiliser le Console de gestion AWS, reportez-vous à la section Création d'un partage de ressources AWS RAM dans le Guide de AWS RAM l'utilisateur.

Pour Sélectionner le type de ressource, choisissez API Gateway Product.

AWS CLI

La commande create-resource-share suivante crée un partage de ressources pour votre nom de domaine personnalisé privé. L'association entre la ressource et le mandataire peut prendre quelques minutes. Pour les principaux, fournissez un ID de compte ou un ID d'organisation, comme `arn:aws:organizations::123456789012:organization/o-1234abcd`. Vous pouvez fournir plusieurs principaux pour votre partage de ressources.

```
aws ram create-resource-share \
 --region us-west-2 \
 --name portal-product-resource-share \
 --permission-arns arn:aws:ram::aws:permission/
AWSRAMDefaultPermissionAPIGatewayDeveloperPortalProduct \
 --resource-arns arn:aws:apigateway:us-west-2:111122223333:/portalproducts/
p000000000 \
 --principals 222222222222
```

Pour annuler le partage de votre produit de portail, AWS RAM supprimez le partage de ressources.

À tout moment, vous pouvez modifier la politique de partage des produits afin de modifier les principaux utilisateurs autorisés à utiliser les produits de votre portail sur leurs portails.

Pour mettre à jour la politique de partage des produits

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Dans le volet de navigation principal, sélectionnez les produits Portal.
3. Choisissez un produit.
4. Dans l'onglet Partage de produits, pour Politique de partage de produits, choisissez Gérer la politique de partage de produits.

5. Si vous n'avez jamais partagé votre produit auparavant, votre politique de partage de produits interdira tout accès aux autres comptes des produits de votre portail. Vous devez mettre à jour la politique pour autoriser l'accès de certains comptes à votre produit de portail.
6. Après avoir mis à jour votre politique de partage de produits, choisissez Enregistrer les modifications.

Ajouter un produit de portail partagé à votre portail dans API Gateway

En tant que propriétaire du portail, vous pouvez utiliser les produits du portail partagés avec vous par d'autres comptes de votre portail. Vous conservez le contrôle total de votre portail.

Considérations

Les considérations suivantes peuvent avoir une incidence sur la manière dont vous partagez les ressources du portail :

- Vous devez accepter le partage des ressources du produit par le propriétaire du produit pour utiliser son produit sur votre portail. Si vous et le propriétaire du produit appartenez à la même organisation, vous AWS RAM pouvez effectuer certaines étapes de partage pour vous.
- Si le propriétaire du produit annule le partage d'un produit alors qu'il se trouve sur votre portail publié, le produit sera toujours visible sur le portail, mais vous ne pourrez pas le publier à nouveau tant que vous n'aurez pas supprimé ce produit. En tant que propriétaire du portail, il est de votre responsabilité de supprimer le produit de votre portail, car il ne peut pas le faire à votre place.
- Vous pouvez consulter les produits du portail, les pages de produits et les pages de points de terminaison REST des produits, mais vous ne pouvez modifier aucune de ces ressources.
- Si un produit est partagé avec vous, vous ne pouvez pas le partager avec un autre compte.

(Facultatif) Acceptez le partage des ressources

Une fois que votre responsable de produit a créé un partage de ressources, vous disposez de 12 heures pour l'accepter. Si vous faites partie de la même organisation AWS Organizations que le propriétaire du produit, le partage est automatiquement accepté. Si vous faites partie d’une organisation qui a activé le partage automatique de ressources, celles-ci sont automatiquement partagées avec vous.

Console de gestion AWS

Pour l'utiliser Console de gestion AWS, voir Accepter et rejeter les invitations à partager des ressources dans le Guide de l'AWS RAM utilisateur.

AWS CLI

Pour trouver toutes les ressources partagées avec vous, utilisez la commande get-resource-share-invitations suivante :

```
aws ram get-resource-share-invitations \
    --region us-west-2
```

Utilisez l'ARN de partage de ressources obtenu pour accepter l'invitation de partage de ressources. La commande accept-resource-share-invitation suivante accepte le partage de ressources.

```
aws ram accept-resource-share-invitation \
    --resource-share-invitation-arn arn:aws:ram:us-west-2:123456789012:resource-
share-invitation/1e3477be-4a95-46b4-bbe0-c4001EXAMPLE \
    --region us-west-2
```

Ajouter un produit partagé à votre portail

Après avoir accepté le partage des ressources, vous ajoutez le produit à votre portail.

Console de gestion AWS

Pour ajouter un produit partagé à votre portail

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 Portals.
3. Choisissez un portail.
4. Dans l'onglet Produits, pour les produits Portal, choisissez Ajouter des produits.
5. Ajoutez un produit à votre portail. Les produits partagés sont affichés comme `shared` dans la liste des produits.
6. Sélectionnez Enregistrer les modifications.

AWS CLI

Pour trouver tous les produits du portail que vous avez partagés, utilisez la `get-portal-products` commande suivante :

```
aws apigatewayv2 get-portal-products \
    --resource-owner OTHER_ACCOUNTS \
    --region us-west-2
```

Pour ajouter un produit de portail partagé à votre portail, utilisez la `update-portal` commande suivante :

```
aws apigateway update-portal \
    --included-portal-product-arns arn:aws:apigateway:us-west-2:111122223333:/
portalproducts/p000000000 \
    --region us-west-2
```

Activez, essayez-le pour le point de terminaison REST d'un produit API Gateway sur votre portail

Utilisez try it pour permettre à un utilisateur d'API d'appeler le point de terminaison de votre produit depuis votre portail. Lorsqu'un utilisateur de l'API utilise Try it, il saisit les paramètres de demande de méthode et invoque le point de terminaison de votre produit. API Gateway appelle ensuite votre API sur l'Internet public et renvoie la réponse dans la fenêtre d'essai. Vous pouvez utiliser une page de produit personnalisée pour fournir des informations supplémentaires sur les paramètres requis pour accéder à votre API.

API Gateway applique les limites suivantes pour protéger votre APIs :

- API Gateway n'autorise que 3 requêtes par seconde adressées à votre API.
- API Gateway utilise un délai d'expiration intégré de 29 000 ms. Votre API actuelle a peut-être un délai d'expiration plus élevé, mais API Gateway n'applique pas ce délai lorsqu'un client essaie de l'utiliser.
- API Gateway limite la charge utile de réponse à 6 Mo.

Considérations

Les considérations suivantes peuvent avoir une incidence sur la façon dont vous utilisez Try it :

- Essayez, il n'est pas pris en charge lorsque vous prévisualisez un portail.
- Essayez qu'il n'est pas pris en charge pour le REST APIs avec les fonctionnalités suivantes :
  - Privé APIs
  - APIs qui utilisent le protocole TLS mutuel
  - APIs qui utilisent des certificats SSL privés ou auto-signés

  En tant que propriétaire du portail, vous êtes responsable de la communication avec les utilisateurs de votre API. La raison pour laquelle le bouton « essayer » n'existe pas pour les applications APIs REST non prises en charge. API Gateway ne vous explique pas cela.

Activez Essayez-le pour un point de terminaison REST du produit

La procédure suivante indique comment activer Try it pour un point de terminaison de produit.

Pour savoir comment utiliser Try it in a portal, voirthe section called “Utiliser un portail”.

Activez Essayez-le pour un point de terminaison REST du produit

1. Dans le volet de navigation principal, sélectionnez Portal products.
2. Choisissez un produit.
3. Dans l'onglet Documentation, sous les pages de référence de l'API, choisissez le nom d'un point de terminaison REST du produit, tel que /dogs - GET.
4. Choisissez Modifier la page.
5. Pour permettre aux clients d'invoquer votre API sur votre portail, sélectionnez la fonctionnalité Essayer.
6. Sélectionnez Enregistrer les modifications.
7. Vous devez republier tous les portails qui utilisent ce produit de portail pour que les modifications prennent effet.

Supprimer un produit de portail dans API Gateway

Lorsque vous supprimez un produit du portail, il ne peut pas être récupéré.

Supprimer un produit de portail

La procédure suivante indique comment supprimer un produit de portail.

Pour supprimer un produit de portail

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Dans le volet de navigation principal, sélectionnez Portal products.
3. Choisissez un produit de portail.
4. Sélectionnez Delete (Supprimer).
5. Confirmez votre choix et choisissez Supprimer.
6. Vous devez republier tous les portails qui utilisent ce produit de portail pour que les modifications prennent effet.

Création d'un portail dans API Gateway

Un portail est un ensemble de produits. Vous contrôlez l'autorisation, l'image de marque et la publication de votre portail. Lorsque vous créez un portail, son statut de publication est`Disabled`. Cela signifie que votre portail n'est pas détectable sur le Web. Pour permettre aux consommateurs d'accéder à votre portail, vous devez le publier. Avant de publier votre portail, vous pouvez également le prévisualiser.

API Gateway prend en charge les portails sur les navigateurs suivants :

- Firefox
- Google Chrome
- Microsoft Edge
- Safari

Choisissez un nom de domaine

Par défaut, l'URL de votre portail est hébergée sur le nom de domaine Amazon suivant :

```
https://p-portalId.apigw-portal.us-east-1.on.aws
```

Pour personnaliser le nom de domaine, vous pouvez fournir un nom de domaine que vous possédez et un certificat SSL pour le domaine personnalisé. Le certificat doit être géré par AWS Certificate

Manager(ACM) dans l'est des États-Unis (Virginie du Nord) et être pris en charge par Amazon CloudFront. Pour plus d'informations, consultez la section Exigences relatives à l'utilisation de SSL/TLS certificats avec CloudFront.

Lorsque vous utilisez un nom de domaine personnalisé, API Gateway crée une CloudFront distribution, sécurisée en transit avec votre certificat ACM. API Gateway fournit ensuite un enregistrement d'alias, que vous ajoutez à votre configuration DNS. L'enregistrement d'alias est le nom de domaine appartenant à Amazon. Vous devez ajouter cet enregistrement d'alias pour diriger le trafic vers la CloudFront distribution de votre domaine personnalisé. Le nom de domaine que vous fournissez doit être unique et ne pas être déjà utilisé par la liste des alias d'une CloudFront distribution existante.

Si vous utilisez votre propre nom de domaine, API Gateway utilise la politique `TLS_1_2` de sécurité pour protéger votre portail. Pour plus d'informations, voir Protocoles et chiffrements pris en charge entre les lecteurs. Vous ne pouvez pas modifier cette valeur.

Création d'un portail

Vous utilisez la console API Gateway pour créer et configurer votre portail. Les procédures suivantes vous guident à travers toutes les étapes requises. Vous pouvez également choisir Passer à la révision et créer à la fin de la première procédure pour créer un portail vide.

La procédure suivante indique comment créer un portail. Vous devez d'abord définir les détails de votre portail.

Pour définir les détails de votre portail

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Dans le volet de navigation principal, sélectionnez Portals.
3. Choisissez Créer un portail.
4. Dans Détails du portail, dans Nom du portail, entrez un nom.
5. Pour Description du portail, entrez une description.
6. Pour l'autorisation du portail, sélectionnez le mode d'autorisation de votre portail.
    - Si c'est la première fois que vous créez un portail, nous vous recommandons de choisir Aucun. Si vous prévoyez d'ajouter une production APIs à votre portail, utilisez Exiger une autorisation.

- Pour obliger les utilisateurs du portail à s'authentifier pour utiliser votre portail, choisissez Exiger une autorisation, puis procédez comme suit :

  a. Pour le groupe d'utilisateurs Amazon Cognito, sélectionnez votre groupe d'utilisateurs.

     Le groupe d'utilisateurs doit se trouver dans la même région que votre portail.

  b. Pour ID client de l'application, choisissez l'ID du client de l'application.

     Dans le client de votre application, vous devez définir l'URL de rappel sur l'URL par défaut du portail.

  c. Pour le domaine du groupe d'utilisateurs, entrez le domaine du groupe d'utilisateurs. Pour plus d'informations, consultez la section Configuration d'un domaine de groupe d'utilisateurs.

7. Pour la configuration du domaine, sélectionnez le nom de domaine de votre portail.
   - Si c'est la première fois que vous créez un portail, nous vous recommandons de choisir le domaine par défaut. Une fois que vous aurez publié votre portail, il sera disponible à l'adresse`https://p-`*`portalId`*`.apigw-portal.`*`us-east-1`*`.on.aws`.
   - Pour fournir un nom de domaine personnalisé dont vous êtes le propriétaire, sélectionnez Nom de domaine personnalisé et procédez comme suit :

     a. Pour Nom de domaine, entrez un nom de domaine complet.

     b. Pour le certificat ACM, choisissez un certificat ACM qui couvre le nom de domaine.

        Vous devez ajouter l'enregistrement d'alias fourni par API Gateway à votre enregistrement DNS afin de desservir le trafic provenant de votre portail.

8. Pour ajouter un moniteur d'application CloudWatch RUM à votre portail, pour le moniteur d'application RUM, choisissez un moniteur d'application. Pour plus d'informations, voir CloudWatch RUM.
9. Choisissez Suivant.

Après avoir défini les détails de votre portail, vous ajoutez des produits de portail. Vous pouvez modifier les produits du portail à tout moment, mais vous devez republier votre portail pour que les modifications prennent effet.

Pour sélectionner les produits de votre portail

1. Dans la liste des ressources, sélectionnez les produits à ajouter à votre portail.

2. Choisissez Suivant.

Après avoir sélectionné vos produits, vous personnalisez le design du portail. Pour télécharger un logo de portail, vous devez charger une image sur Amazon S3. Vous pouvez modifier ces valeurs à tout moment, mais vous devez republier votre portail pour que les modifications prennent effet.

(Facultatif) Pour personnaliser votre portail

1. Pour le logo du portail, indiquez l'URI S3 de votre fichier.

   Seulement `.svg.png`, ou `.jpg` sont pris en charge.
2. Pour personnaliser le design de votre portail, deux options s'offrent à vous :

   a. Vous pouvez choisir le thème du portail pour appliquer un thème de couleur prédéfini sur votre portail.

   b. Vous pouvez utiliser les sélecteurs de couleur pour sélectionner toutes les options de conception de votre portail.
3. Choisissez Suivant.

Après avoir sélectionné le design du portail, passez en revue les informations relatives au portail.

Pour consulter votre portail

- Vérifiez vos paramètres sur la page de révision et de création. Lorsque vous êtes satisfait de vos paramètres, choisissez Create (Créer).

Si vous utilisez un groupe d'utilisateurs Amazon Cognito pour contrôler l'accès à votre portail, vous devez définir l'URL de rappel sur l'URL par défaut du portail dans l'application cliente. Pour plus d'informations, consultez la section Paramètres spécifiques à l'application avec les clients d'applications.

Votre portail ne sera plus accessible aux consommateurs une fois que vous l'aurez créé. Vous devez publier votre portail pour qu'il soit accessible sur Internet. Pour de plus amples informations, veuillez consulter the section called “Publier un portail”.

Mettre à jour un portail dans API Gateway

Vous pouvez mettre à jour le thème, le logo ou les produits du portail dans votre portail. Lorsque vous mettez à jour votre portail, les modifications ne prennent effet que lorsque vous le republiez.

Mettre à jour un portail

La procédure suivante indique comment mettre à jour la conception d'un portail.

Pour mettre à jour la conception d'un portail

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Dans le volet de navigation principal, sélectionnez Portals.
3. Choisissez un portail.
4. Dans l'onglet Conception du portail, choisissez Modifier.
5. Pour mettre à jour le logo du portail, choisissez Mettre à jour le logo du portail. Vous pouvez soit supprimer le logo existant, soit utiliser un nouveau logo.
6. Apportez n'importe quelle mise à jour du thème ou de l'arrière-plan à la conception de votre portail.
7. Sélectionnez Enregistrer les modifications.
8. (Facultatif) Choisissez Générer un aperçu du portail pour afficher vos modifications.

   Après avoir confirmé vos modifications, publiez votre portail.
9. Choisissez Publier le portail.
10. Entrez une description pour la mise à jour de votre portail et choisissez Publier.

Aperçu d'un portail dans API Gateway

Vous pouvez prévisualiser votre portail avant de le publier. API Gateway crée une URL éphémère que vous pouvez utiliser pour prévisualiser la version de votre portail que les clients verront une fois votre portail publié.

Considérations

Les considérations suivantes peuvent avoir une incidence sur la façon dont vous prévisualisez un portail :

- Essayez, il n'est pas activé pour votre portail de prévisualisation. Pour de plus amples informations, veuillez consulter the section called “Activez Essayez-le pour un point de terminaison REST du produit sur votre portail”.

Afficher un aperçu d'un portail

La procédure suivante indique comment prévisualiser un portail.

Pour prévisualiser un portail

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Dans le volet de navigation principal, sélectionnez Portals.
3. Choisissez un portail.
4. Choisissez Générer un aperçu du portail.

   N'actualisez pas votre page.
5. Une fois qu'API Gateway a généré votre aperçu, choisissez Open portal preview.

   L'aperçu de votre portail s'ouvrira dans un nouvel onglet.

Publier un portail dans API Gateway

Pour que les utilisateurs d'API puissent accéder à votre portail, vous devez le publier. L'URL d'un portail peut être découverte par n'importe qui sur Internet. Nous vous recommandons de prévisualiser et de sécuriser votre portail avant de le publier.

Considérations

API Gateway peut prendre quelques minutes pour publier votre portail. Vous pouvez surveiller l'état de publication dans la console.

Publier un portail

La procédure suivante indique comment publier un portail.

Pour publier un portail

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Dans le volet de navigation principal, sélectionnez Portals.
3. Choisissez un portail.
4. Choisissez Publier le portail.
5. (Facultatif) Dans Description des modifications, entrez une description de votre modification.

   Lorsque vous publiez un portail, nous vous recommandons de toujours fournir une brève description de vos modifications.
6. Choisissez Publish.

   API Gateway met quelques minutes à terminer la publication de votre portail. API Gateway fournit un lien vers votre portail lorsqu'il est disponible.

Pour supprimer votre portail, vous devez d'abord le désactiver. Pour de plus amples informations, veuillez consulter the section called “Désactiver un portail”.

Utiliser un portail API Gateway

Une fois que vous avez publié votre portail, il est disponible sur le Web. Pour utiliser votre portail, copiez le nom de domaine par défaut du portail ou le nom de domaine et saisissez-le dans un navigateur. Si votre portail utilise un nom de domaine, le nom de domaine par défaut ne peut pas être utilisé pour accéder à votre portail.

Utiliser un portail

La procédure suivante montre comment accéder à un portail et utiliser Try it pour le point de terminaison REST d'un produit.

Pour utiliser un portail

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Dans le volet de navigation principal, sélectionnez Portals.
3. Choisissez un portail.
4. Copiez le nom de domaine par défaut du portail ou le nom de domaine et saisissez-le dans un navigateur.
5. Sur la page d'accueil du portail, vous pouvez rechercher des produits du portail et découvrir des pages de produits. Utilisez la recherche en haut du portail pour rechercher l'ensemble du contenu de votre portail. Utilisez la barre de recherche située sous Produits pour rechercher des produits du portail.
6. Choisissez un produit de portail.
7. Choisissez un point de terminaison REST du produit.
8. Dans la fenêtre Try it, entrez les paramètres pour essayer d'appeler votre API.
9. Choisissez Try it. API Gateway invoquera votre API et affichera les résultats dans la fenêtre.

Résoudre les problèmes liés à votre portail

Vous trouverez ci-dessous des conseils de résolution des erreurs et des problèmes que vous pourriez rencontrer lors de l'utilisation d'un portail.

Impossible d'accéder au site Web du portail

Si vous recevez un message d'erreur indiquant que le site Web du portail n'est pas accessible, assurez-vous d'avoir publié votre portail. Si vous avez utilisé un nom de domaine, assurez-vous d'avoir ajouté l'enregistrement d'alias fourni par API Gateway pour diriger le trafic vers la CloudFront distribution de votre domaine personnalisé.

L'essai a échoué

Si vous ne parvenez pas à utiliser Try it pour le point de terminaison REST d'un produit, assurez-vous que l'état Try it est activé`ACTIVE`. En outre, assurez-vous d'avoir saisi les bons paramètres pour appeler votre API.

Les pages n'apparaissent pas dans votre portail

Si vous avez créé une page de point de terminaison REST ou une page de produit et qu'elle n'est pas visible sur votre portail, assurez-vous que vos pages ne sont pas des brouillons. Pour vérifier cela, procédez comme suit :

Pour vérifier que vos pages ne figurent pas dans les brouillons

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Dans le volet de navigation principal, sélectionnez Portal products.
3. Choisissez votre produit de portail.
4. Dans l'onglet Documentation, vérifiez que vos pages ne figurent pas dans la section Brouillon de documentation ou dans la section Brouillon de pages de référence d'API, mais dans les pages de documentation personnalisée ou les pages de référence d'API.

Désactiver un portail dans API Gateway

Lorsque vous désactivez un portail, celui-ci n'est plus disponible sur le Web. Après avoir désactivé un portail, vous pouvez soit le supprimer, soit le republier si nécessaire.

Désactiver un portail

La procédure suivante indique comment désactiver un portail.

Pour désactiver un portail

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Dans le volet de navigation principal, sélectionnez Portals.
3. Choisissez un portail.
4. Choisissez Actions, puis sélectionnez Désactiver le portail.
5. Confirmez votre choix et choisissez Désactiver le portail.

   Le statut de publication deviendra Désactivé.

Pour supprimer le statut de désactivation du portail, publiez-le à nouveau.

Supprimer un portail dans API Gateway

Lorsque vous supprimez un portail, il ne peut pas être récupéré et vous devez d'abord le désactiver. Si vous souhaitez supprimer votre portail du Web, vous pouvez le désactiver. Cela vous permet de modifier votre portail et de le republier ultérieurement.

Supprimer un portail

La procédure suivante indique comment supprimer un portail.

Pour supprimer un portail

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Dans le volet de navigation principal, sélectionnez Portals.
3. Choisissez un portail.
4. Si votre portail n'est pas désactivé, désactivez-le en choisissant Actions, Désactiver le portail.

   La désactivation de votre portail prend quelques minutes. Vous pouvez surveiller l'état de publication pour voir si le statut de publication de votre portail est désactivé.
5. Après avoir dépublié votre portail, vous pouvez le supprimer. Pour supprimer votre portail, choisissez Actions, Supprimer le portail. Confirmez votre choix et choisissez Supprimer.

API HTTP API Gateway

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 de plus amples 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 fonctions AWS Lambda ou à n'importe quel point de terminaison HTTP routable. Par exemple, vous pouvez créer une API HTTP qui s'intègre avec 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 gestion AWS, de l'AWS CLI, des API, de CloudFormation ou des SDK.

Rubriques

- Développement du protocole HTTP APIs dans API Gateway
- Publication d'API HTTP invocables par les clients
- Protection de vos API HTTP dans API Gateway
- Surveiller le protocole HTTP APIs dans API Gateway
- Résolution des problèmes des API HTTP dans API Gateway

Développement du protocole HTTP APIs dans API Gateway

Cette section fournit des détails sur les fonctionnalités d'API Gateway dont vous avez besoin pour développer votre API Gateway APIs.

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
- Création de routes pour HTTP APIs dans API Gateway
- Types d’adresses IP pour les API HTTP dans API Gateway
- Contrôler et gérer l'accès au protocole HTTP APIs dans API Gateway
- Création d’intégrations pour les API HTTP dans API Gateway
- Configurer CORS pour HTTP APIs dans API Gateway
- Transformation des demandes et des réponses d’API HTTP dans API Gateway
- Utiliser les définitions OpenAPI pour HTTP APIs dans API Gateway

Création d’une API HTTP

Pour créer une API fonctionnelle, vous devez disposer d’au moins une route, 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, consultez Mise en route.

Rubriques

- Créez une API HTTP à l'aide du Console de gestion AWS
- Création d'une API HTTP à l'aide de la AWS CLI

Créez une API HTTP à l'aide du Console de gestion AWS

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 create-api suivante utilise la création rapide pour créer une API qui s'intègre avec une fonction Lambda sur le backend.

> **Note**
>
> Pour appeler une intégration Lambda, API Gateway doit disposer des autorisations requises. Vous pouvez utiliser une politique 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
```

Création de routes pour HTTP APIs dans API Gateway

Route les demandes d'API entrantes directes vers les ressources dorsales. Les routes se composent de deux parties : une méthode HTTP et un chemin de ressource (, par exemple, `GET /pets`. Vous

pouvez définir des méthodes HTTP spécifiques pour votre route. 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 une route `$default` qui agit comme un fourre-tout pour les demandes qui ne correspondent à aucune autre route.

> **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, la route `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 gourmande capture toutes les ressources enfants d'une route. Pour créer une variable de chemin gourmande, 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 la route **`$default`**

La route `$default` attrape les demandes qui ne correspondent pas explicitement aux autres routes de votre API.

Lorsque la route `$default` reçoit une demande, API Gateway envoie le chemin de demande complet à l'intégration. Par exemple, vous pouvez créer une API avec seulement une route `$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 Création d'intégrations de proxy HTTP pour HTTP APIs.

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 une route. API Gateway sélectionne la route avec la correspondance la plus spécifique, en utilisant les priorités suivantes :

1. Correspondance complète pour une route et une méthode.
2. Correspondance pour une route et une méthode avec une variable de chemin gourmande ({proxy+}).
3. La route $default

Si aucune route 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 de routes 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 routes de l'exemple.

Requête	Route sélectionnée	Explication
`GET https://api-id.execute-api.region.amazonaws.com/pets/dog/1`	`GET /pets/dog/1`	La demande correspond entièrement à cette route statique.
`GET https://api-id.execute-api.region.amazonaws.com/pets/dog/2`	`GET /pets/dog/{id}`	La demande correspond entièrement à cette route.
`GET https://api-id.execute-api.region.amazonaws.com/pets/cat/1`	`GET /pets/{proxy+}`	La demande ne correspond pas entièrement à une route. La route 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 une route. Les routes avec des variables de chemin gourmandes ont une priorité plus élevée que la route `$default`.

Types d'adresses IP pour les API HTTP dans API Gateway

Lorsque vous créez une API, vous spécifiez le type d'adresses IP qui peut invoquer votre API. Vous avez le choix entre IPv4 (pour autoriser les adresses IPv4 à invoquer votre API) et Dualstack (pour autoriser les adresses IPv4 et IPv6 à invoquer votre API). Vous pouvez définir le type d'adresse IP sur Dualstack pour éviter l'épuisement de l'espace IP ou renforcer votre niveau de sécurité. Pour plus d'informations sur les avantages d'un type d'adresse IP à double pile, consultez IPv6 sur AWS.

Considérations relatives aux types d'adresses IP

Les considérations suivantes peuvent avoir une incidence sur votre utilisation des types d'adresses IP :

- Le type d'adresses IP par défaut pour toutes les API HTTP est IPv4.
- Si vous remplacez le type d'adresse IP IPv4 d'une API existante par un type d'adresse IP à double pile, vérifiez que toutes les stratégies d'accès à vos API ont été mises à jour pour prendre en compte les appels IPv6. La modification du type d'adresse IP prend effet immédiatement.
- Votre API peut être mappée à un nom de domaine personnalisé présentant un type d'adresse IP différent de celui de votre API. La désactivation du point de terminaison de votre API par défaut peut avoir une incidence sur la manière dont les appelants peuvent invoquer votre API.

Modification du type d'adresse IP d'une API HTTP

Vous pouvez modifier le type d'adresse IP en mettant à jour la configuration de l'API. Vous pouvez mettre à jour la configuration de l'API à l'aide de la Console de gestion AWS, de l'AWS CLI, d'CloudFormation ou d'un kit SDK AWS. Si vous modifiez le type d'adresse IP de l'API, vous n'avez pas besoin de redéployer votre API pour que les modifications prennent effet.

Console de gestion AWS

Pour modifier le type d'adresse IP d'une API HTTP

1. Connectez-vous à la console API Gateway à l'adresse : https://console.aws.amazon.com/apigateway.
2. Choisissez une API HTTP.
3. Pour Paramètres de l'API, choisissez Modifier.
4. Pour Type d'adresse IP, choisissez IPv4 ou Dualstack.
5. Choisissez Save (Enregistrer).

   La modification de la configuration de votre API prend effet immédiatement.

AWS CLI

La commande update-api suivante met à jour une API pour qu'elle possède le type d'adresse IP Dualstack :

```
aws apigatewayv2 update-api \
    --api-id abcd1234 \
    --ip-address-type dualstack
```

Le résultat se présente comme suit :

```
{
    "ApiEndpoint": "https://abcd1234.execute-api.us-east-1.amazonaws.com",
    "ApiId": "abcd1234",
    "ApiKeySelectionExpression": "$request.header.x-api-key",
    "CreatedDate": "2025-02-04T22:20:20+00:00",
    "DisableExecuteApiEndpoint": false,
    "Name": "My-HTTP-API",
    "ProtocolType": "HTTP",
    "RouteSelectionExpression": "$request.method $request.path",
    "Tags": {},
    "NotificationUris": [],
    "IpAddressType": "dualstack"
}
```

Contrôler et gérer l'accès au protocole HTTP APIs 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 à. APIs Pour de plus amples informations, veuillez consulter Contrôlez l'accès au HTTP APIs avec des AWS Lambda autorisateurs.
- Les autorisateurs JWT utilisent des jetons Web JSON pour contrôler l'accès à. APIs Pour de plus amples informations, veuillez consulter Contrôle de l'accès aux API HTTP avec des mécanismes d'autorisation JWT dans API Gateway.
- Les rôles et politiques AWS IAM standard offrent des contrôles d'accès flexibles et robustes. Vous pouvez utiliser les rôles et les politiques IAM pour contrôler qui peut les créer et les gérer APIs, ainsi que les personnes habilitées à les invoquer. Pour de plus amples informations, veuillez consulter Contrôlez l'accès au HTTP APIs avec l'autorisation IAM dans API Gateway.

Pour renforcer votre posture de sécurité, nous vous recommandons de configurer un mécanisme d'autorisation pour toutes les routes de votre API HTTP. Vous devrez peut-être le faire pour vous

conformer aux différents cadres de conformité. Pour plus d'informations, consultez Amazon API Gateway controls dans le Guide de l'utilisateur AWS Security Hub .

Contrôlez l'accès au HTTP APIs avec des AWS Lambda autorisateurs

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 Console de gestion AWS utilisée par défaut. Si vous créez un autorisateur Lambda à l'aide du AWS CLI, ou d'un SDK CloudFormation, vous devez spécifier un. `authorizerPayloadFormatVersion` Les valeurs prises en charge sont `1.0` et `2.0`.

Si vous avez besoin de compatibilité avec REST APIs, 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 politique IAM qui autorise ou refuse l'accès à votre route d'API. Vous pouvez utiliser la syntaxe de politique standard IAM dans la politique. Pour obtenir des exemples de politiques 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

JSON

```
{
  "principalId": "abcdef",
  "policyDocument": {
    "Version":"2012-10-17",
    "Statement": [
      {
        "Action": "execute-api:Invoke",
        "Effect": "Allow|Deny",
        "Resource": "arn:aws:execute-api:{regionId}:{accountId}:{apiId}/{stage}/
{httpVerb}/[{resource}/[{child-resources}]]"
      }
    ]
  },
  "context": {
    "exampleKey": "exampleValue"
  }
}
```

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 politique IAM qui utilise la syntaxe de politique 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",
  "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`.

Le tableau suivant décrit les sources d'identité prises en charge pour un mécanisme d'autorisation Lambda.

Type	Exemple	Remarques
Valeur d'en-tête	$request.header. *`name`*	Les noms d'en-tête ne sont pas sensibles à la casse.
Valeur de chaîne de requête	$request.querystring. *`name`*	Les noms de chaîne de requête sont sensibles à la casse.
Variable de contexte	$contexte. *`variableName`*	Valeur d'une variable de contexte prise en charge.
Variable d'étape	$StageVariables. *`variableName`*	Valeur d'une variable d'étape.

Vous pouvez également renvoyer directement `{"errorMessage" : "Unauthorized"}` de votre fonction Lambda pour renvoyer une erreur `401` à vos clients. Si vous renvoyez directement une

erreur `401` de votre fonction Lambda à vos clients, ne spécifiez aucune source d'identité lorsque vous créez votre mécanisme d'autorisation Lambda.

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 politique IAM. En fonction de votre mécanisme d'autorisation, votre politique IAM peut avoir besoin de contrôler l'accès à plusieurs éléments.

Par défaut, API Gateway utilise la réponse du mécanisme d'autorisation mise en cache pour toutes les routes 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 politique de ressources de la fonction ou d'un rôle IAM. La commande create-authorizer suivante crée un mécanisme d'autorisation 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 add-permission suivante met à jour la stratégie de ressources de la fonction Lambda afin qu'elle autorise API Gateway à invoquer la fonction. 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 route de manière à l'utiliser. La commande update-route suivante ajoute le mécanisme d'autorisation Lambda à la route. Si votre mécanisme d'autorisation Lambda utilise la mise en cache des stratégies, veillez à mettre à jour la politique afin qu'elle contrôle l'accès à la route supplémentaire.

```
aws apigatewayv2 update-route \
    --api-id abcdef123 \
    --route-id abc123 \
    --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 politique 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 des mécanismes d'autorisation JWT dans API Gateway

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 une route 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 route d'une API ou utiliser le même mécanisme d'autorisation pour plusieurs routes.

> **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 routes pour qu'elles 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.

Autorisation des demandes d'API avec un mécanisme d'autorisation JWT

API Gateway utilise le workflow général suivant pour autoriser les demandes de routes configurées 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 `jwks_uri` de l'émetteur. Actuellement, seuls les RSA-based algorithmes sont pris en charge. API Gateway met en cache la clé publique pendant deux heures au maximum, dans la mesure du possible. En cas de panne de cache, API Gateway extrait la clé publique auprès de l'émetteur via le réseau, ce qui ajoute de la latence à la demande qui la déclenche. Les API dont le taux de

demandes est faible ou intermittent sont plus susceptibles de subir ces récupérations. Comme une clé publique peut rester en cache jusqu'à deux heures, prévoyez un délai de grâce lors de la rotation des clés, pendant lequel l'ancienne et la nouvelle clé 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 ne valide `client_id` que si `aud` n'est pas présent. Si `aud` et `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 la route.

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 la route 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, consultez 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.

Création d'un mécanisme d'autorisation JWT à l'aide de la console

La procédure suivante montre comment créer un mécanisme d'autorisation JWT à l'aide de la console.

Pour créer un mécanisme d'autorisation 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 panneau de navigation principal, choisissez Autorisation.
4. Choisissez l'onglet Gérer les mécanismes d'autorisation.
5. Cliquez sur Créer.
6. Pour Type du mécanisme d'autorisation, choisissez JWT.
7. Configurez votre mécanisme d'autorisation JWT et spécifiez une source d'identité qui définit la source du jeton.
8. Cliquez sur Créer.

Créez un autorisateur JWT à l'aide du AWS CLI

La commande create-authorizer suivante crée un mécanisme d'autorisation JWT. Pour `jwt-configuration`, spécifiez les paramètres `Audience` et `Issuer` pour votre fournisseur d'identité. Si vous utilisez Amazon Cognito comme fournisseur d'identité, `IssuerUrl` est `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 à l'aide de AWS CloudFormation

Le CloudFormation modèle suivant crée une API HTTP avec un autorisateur JWT qui utilise Amazon Cognito comme fournisseur d'identité.

Le résultat du 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 invoquer l'API avec le jeton d'accès, remplacez le symbole # de l'URL par un ? pour utiliser le jeton comme paramètre de chaîne de requête.

Exemple 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
```

Mise à jour d’une route pour utiliser un mécanisme d’autorisation JWT

Vous pouvez utiliser la console AWS CLI, le SDK ou un AWS SDK pour mettre à jour un itinéraire afin d'utiliser un autorisateur JWT.

Mise à jour d'une route pour utiliser un mécanisme d'autorisation JWT à l'aide de la console

La procédure suivante montre comment mettre à jour un mécanisme d'autorisation JWT à l'aide de la console.

Pour créer un mécanisme d'autorisation 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 panneau de navigation principal, choisissez Autorisation.
4. Choisissez une méthode, puis sélectionnez votre mécanisme d'autorisation dans le menu déroulant, puis choisissez Attacher le mécanisme d'autorisation.

Mettez à jour un itinéraire pour utiliser un autorisateur JWT à l'aide du AWS CLI

La commande update-route suivante met à jour une route de manière à ce qu'elle utilise un mécanisme d'autorisation JWT :

```
aws apigatewayv2 update-route \
   --api-id api-id  \
   --route-id route-id  \
   --authorization-type JWT \
   --authorizer-id authorizer-id \
   --authorization-scopes user.email
```

Contrôlez l'accès au HTTP APIs avec l'autorisation IAM dans API Gateway

Vous pouvez activer l'autorisation IAM pour les routes 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 la route.

L'autorisation IAM pour HTTP APIs est similaire à celle pour REST APIs.

> **Note**
>
> Les politiques relatives aux ressources ne sont actuellement pas prises en charge pour le protocole HTTP APIs.

Pour des exemples de politiques IAM qui accordent aux clients l'autorisation d'invoquer APIs, consultezthe section called “Contrôler l’accès pour l’appel d’une API”.

Activation d’une autorisation IAM pour une route

La commande update-route 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
```

Création d’intégrations pour les API HTTP dans API Gateway

Les intégrations connectent une route aux ressources backend. Les API HTTP prennent en charge les intégrations de proxy Lambda, de service AWS et de proxy HTTP. Par exemple, vous pouvez configurer une demande `POST` sur la route `/signup` de votre API pour l’intégrer avec une fonction Lambda qui gère l’inscription des clients.

Rubriques

- Création d'intégrations de AWS Lambda proxy pour HTTP APIs dans API Gateway
- Création d'intégrations de proxy HTTP pour HTTP APIs
- Création d'intégrations de AWS services pour HTTP APIs dans API Gateway
- Création d'intégrations privées pour HTTP APIs dans API Gateway

Création d'intégrations de AWS Lambda proxy pour HTTP APIs dans API Gateway

Une intégration de proxy Lambda vous permet d’intégrer une route API avec 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 des données utiles spécifie le format de l’événement qu’API Gateway envoie à une intégration Lambda, et comment API Gateway interprète la réponse de Lambda. Si vous ne spécifiez pas de version de format de charge utile, la dernière version est Console de gestion AWS utilisée par défaut. Si vous créez une intégration Lambda à l'aide du AWS CLI 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 configuration de la valeur `payloadFormatVersion`, consultez create-integration. Pour plus d'informations sur la détermination de la valeur `payloadFormatVersion` d'une intégration existante, consultez get-integration.

Différences des formats de données utiles

La liste suivante indique les différences entre les versions de format de données utiles `1.0` et `2.0` :

- 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` inclut le champ `rawPath`. Si vous utilisez un mappage d'API pour connecter votre étape à un nom de domaine personnalisé, `rawPath` ne fournira pas la valeur du mappage d'API. Utilisez les formats `1.0` et `path` pour accéder 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 des formats de données utiles

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 de format `1.0`, 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": { "content-type": "application/json" } }`
`{ "message": "Hello from Lambda!" }`	`{ "isBase64Encoded": false, "statusCode": 200, "body": "{ \"message\": \"Hello from Lambda!\" }", "headers": { "content-type": "application/json" } }`

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!"
}
```

Création d'intégrations de proxy HTTP pour HTTP APIs

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 gourmande, 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 avec 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.

Création d'intégrations de AWS services pour HTTP APIs dans API Gateway

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, consultez the section called “AWSRéférence des intégrations de service ”.

Paramètres de demande de mappage

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, consultez the section called "AWSRéférence des intégrations de service ".

Le tableau suivant décrit les paramètres de demande de mappage pris en charge.

Type	Exemple	Remarques
Valeur d'en-tête	$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"` .
Valeur de chaîne de requête	$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"` .
Paramètre de chemin	$request.path. *name*	Valeur d'un paramètre de chemin dans la demande. Par exemple, si la route est `/pets/{petId}` , vous pouvez mapper le paramètre `petId` de la demande avec *`$request.path.petId`* .
Transmission de corps de demande	$request.body	API Gateway transmet l'ensemble du corps de la demande.
Corps de la demande	$request.body. *name*	Expression de chemin JSON. La descente récursive

Type	Exemple	Remarques
		($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	$contexte. *variableName*	Valeur d'une variable de contexte prise en charge.
Variable d'étape	$StageVariables. *variableName*	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, consultez 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 les autorisations nécessaires pour 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 "AWSRéférence des intégrations de service ".

La commande create-integration 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 en utilisant CloudFormation

L'exemple suivant montre un 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 1.0
- SQS-SendMessage 1.0
- SQS-ReceiveMessage 1.0
- SQS-DeleteMessage 1.0
- SQS-PurgeQueue 1.0
- AppConfig-GetConfiguration 1.0
- Kinesis-PutRecord 1.0
- StepFunctions-StartExecution 1.0
- StepFunctions-StartSyncExecution 1.0
- StepFunctions-StopExecution 1.0

EventBridge-PutEvents 1.0

Envoie des événements personnalisés à Amazon EventBridge afin qu'ils puissent être mis en correspondance avec les règles.

Paramètre	Obligatoire
Detail	Vrai
DetailType	Vrai
Source	Vrai
Heure	Faux

Paramètre	Obligatoire
EventBusName	Faux
Ressources	Faux
Région	Faux
TraceHeader	Faux

Pour en savoir plus, consultez PutEvents dans le document Référence des API Amazon EventBridge.

SQS-SendMessage 1.0

Remet un message dans la file d'attente spécifiée.

Paramètre	Obligatoire
QueueUrl	Vrai
MessageBody	Vrai
DelaySeconds	Faux
MessageAttributes	Faux
MessageDeduplicationId	Faux
MessageGroupId	Faux
MessageSystemAttributes	Faux
Région	Faux

Pour en savoir plus, consultez SendMessage dans le document Référence des API Amazon Simple Queue Service.

SQS-ReceiveMessage 1.0

Extrait un ou plusieurs messages (jusqu'à 10) de la file d'attente spécifiée.

Paramètre	Obligatoire
QueueUrl	Vrai
AttributeNames	Faux
MaxNumberOfMessages	Faux
MessageAttributeNames	Faux
ReceiveRequestAttemptId	Faux
VisibilityTimeout	Faux
WaitTimeSeconds	Faux
Région	Faux

Pour en savoir plus, consultez ReceiveMessage dans le document Référence des API Amazon Simple Queue Service.

SQS-DeleteMessage 1.0

Supprime le message spécifié de la file d'attente désignée.

Paramètre	Obligatoire
ReceiptHandle	Vrai
QueueUrl	Vrai
Région	Faux

Pour en savoir plus, consultez DeleteMessage dans le document Référence des API Amazon Simple Queue Service.

SQS-PurgeQueue 1.0

Supprime tous les messages de la file d'attente spécifiée.

Paramètre	Obligatoire
QueueUrl	Vrai
Région	Faux

Pour en savoir plus, consultez PurgeQueue dans le document Référence des API Amazon Simple Queue Service.

AppConfig-GetConfiguration 1.0

Reçoit des informations sur une configuration.

Paramètre	Obligatoire
Application	Vrai
Environnement	Vrai
Configuration	Vrai
ClientId	Vrai
ClientConfigurationVersion	Faux
Région	Faux

Pour en savoir plus, consultez GetConfiguration dans la Référence de l'API AWS AppConfig.

Kinesis-PutRecord 1.0

Écrit un enregistrement de données unique dans Amazon Kinesis Data Stream.

Paramètre	Obligatoire
StreamName	Vrai
non structurées	Vrai

Paramètre	Obligatoire
PartitionKey	Vrai
SequenceNumberForOrdering	Faux
ExplicitHashKey	Faux
Région	Faux

Pour en savoir plus, consultez PutRecord dans le document Référence des API Amazon Kinesis Data Streams.

StepFunctions-StartExecution 1.0

Démarre l'exécution d'une machine d'état.

Paramètre	Obligatoire
StateMachineArn	Vrai
Nom	Faux
Entrée	Faux
Région	Faux

Pour en savoir plus, consultez StartExecution dans la Référence de l'API AWS Step Functions.

StepFunctions-StartSyncExecution 1.0

Démarre une exécution de machine d'état synchrone.

Paramètre	Obligatoire
StateMachineArn	Vrai
Nom	Faux

Paramètre	Obligatoire
Entrée	Faux
Région	Faux
TraceHeader	Faux

Pour en savoir plus, consultez StartSyncExecution dans la Référence de l'API AWS Step Functions.

StepFunctions-StopExecution 1.0

Arrête une exécution.

Paramètre	Obligatoire
ExecutionArn	Vrai
Cause	Faux
Erreur	Faux
Région	Faux

Pour en savoir plus, consultez StopExecution dans la Référence de l'API AWS Step Functions.

Création d'intégrations privées pour HTTP APIs dans API Gateway

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.

> **Note**
>
> Pour créer une intégration privée, vous devez d'abord créer un lien VPC. Les liens VPC V2 sont désormais pris en charge à la fois pour HTTP et REST. APIs Pour en savoir plus sur les liens VPC V2, consultez. the section called "Liaisons VPC V2"

Après avoir créé un lien VPC V2, 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

Considérations

Les considérations suivantes peuvent avoir un impact sur votre utilisation des intégrations privées :

- Toutes les ressources doivent être détenues par la même personne Compte AWS. Cela inclut 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. Pour utiliser le protocole HTTPS, spécifiez un `tlsConfig`. Pour ce faire Console de gestion AWS, lorsque vous créez votre intégration privée, choisissez Paramètres avancés, puis entrez un nom de serveur sécurisé.
- 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 d'Application Load Balancer ou de Network Load Balancer

Avant de créer une intégration privée, vous devez créer un lien VPC V2. Pour en savoir plus sur les liens VPC V2, consultez. the section called "Liaisons VPC V2"

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 create-integration 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 V2. Pour de plus amples informations sur les liens VPC, veuillez consulter the section called “Liaisons VPC V2”.

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.

La commande create-integration suivante crée une intégration privée qui utilise la découverte de AWS Cloud Map 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
```

Configurer CORS pour HTTP APIs dans API Gateway

Le partage des ressources entre origines multiples (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, consultez What is CORS?

CORS est généralement nécessaire pour créer des applications Web dont l'accès est APIs hébergé 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 de route OPTIONS configurée 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.

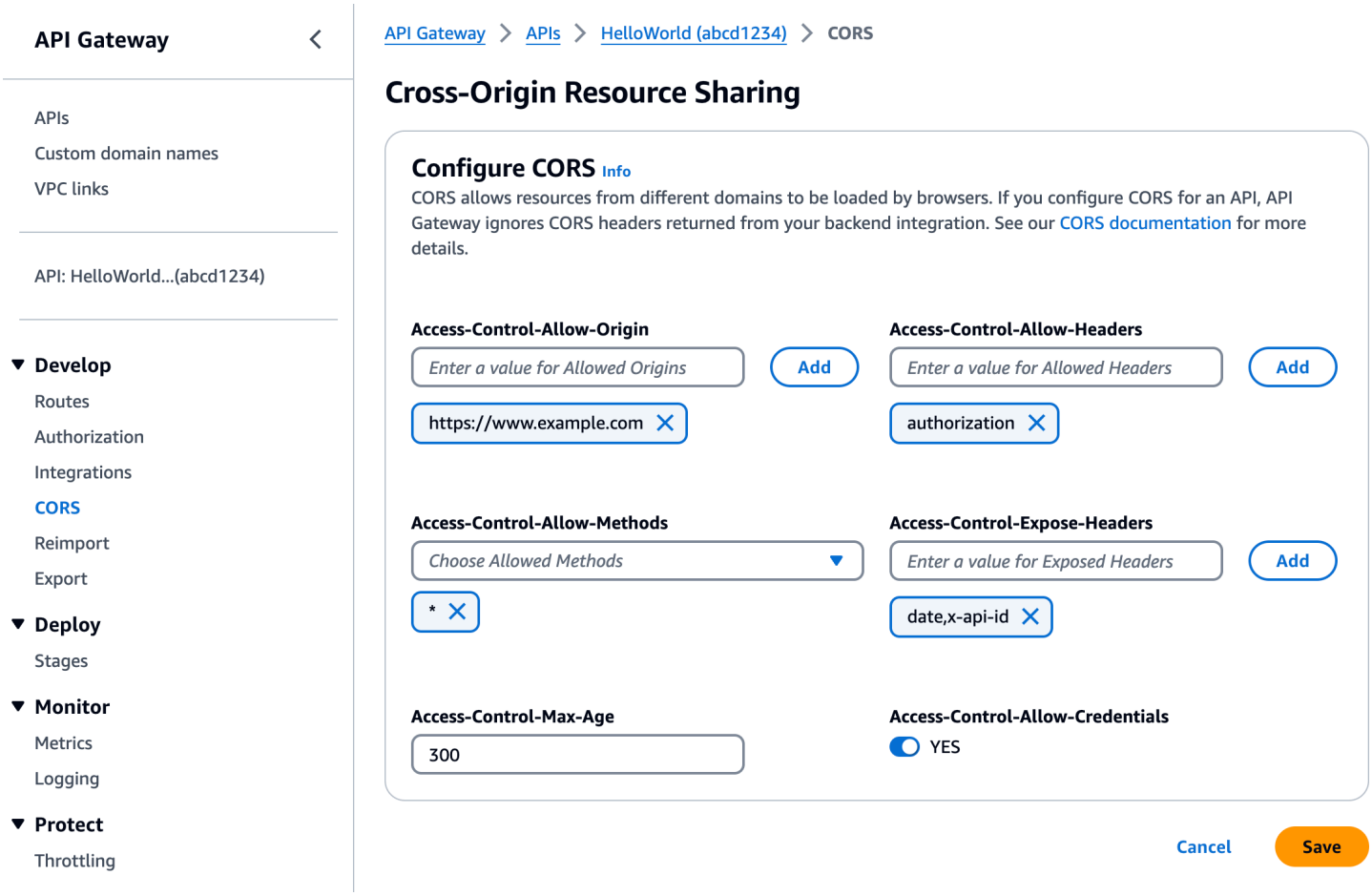
Vous pouvez spécifier les paramètres suivants dans une configuration CORS. Pour ajouter ces paramètres à l'aide de la console API HTTP API Gateway, choisissez Ajouter après avoir entré votre valeur.

En-têtes CORS	Propriété de configuration CORS	Exemples de valeur
Access-Control-Allow-Origin	allowOrigins	• `https://www.example.com` • `*` (autoriser toutes les origines)

En-têtes CORS	Propriété de configuration CORS	Exemples de valeur
		• `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`. Pour la méthode `OPTIONS`, votre demande doit contenir un en-tête `origin` et un en-tête `Access-Control-Request-Method`.

Votre configuration CORS peut s’apparenter à ce qui suit :



Configuration de CORS pour une API HTTP avec une route **$default** et un mécanisme d'autorisation

Vous pouvez activer CORS et configurer l'autorisation pour n'importe quelle route d'une API HTTP. Lorsque vous activez CORS et l'autorisation pour la route `$default`, certaines considérations particulières sont à prendre en compte. La route `$default` attrape les demandes pour toutes les méthodes et routes que vous n'avez pas explicitement définies, y compris les demandes `OPTIONS`. Pour prendre en charge les demandes `OPTIONS` non autorisées, ajoutez une route `OPTIONS /{proxy+}` à votre API qui ne nécessite pas d'autorisation et attachez une intégration à la route. La priorité de la route `OPTIONS /{proxy+}` est supérieure à celle de la route `$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.

Configurer CORS pour une API HTTP à l'aide de la CLI AWS

La commande update-api suivante active les demandes CORS de `https://www.example.com` :

Example

```
aws apigatewayv2 update-api --api-id api-id --cors-configuration AllowOrigins="https://
www.example.com"
```

Pour plus d'informations, consultez CORS dans le document Référence des API Amazon API Gateway Version 2.

Transformation des demandes et des réponses d'API HTTP dans API Gateway

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.

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.

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 corps de la requête

Type	Syntaxe	Remarques
		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 de statut d'une réponse avant qu'API Gateway ne renvoie la réponse aux clients.

Vous configurez les paramètres de réponse pour chaque code de statut 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.

Type	Syntaxe
En-tête	`append|overwrite|remove:header.` *`headername`*
Code de statut	`overwrite:statuscode`

Le tableau suivant présente les valeurs prises en charge que vous pouvez mapper aux paramètres.

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, par exemple `"header1" : "value1,value2"` .

Type	Syntaxe	Remarques
		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.
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 exemples AWS CLI suivants configurent les mappages de paramètres. Pour des exemples de modèles CloudFormation, consultez GitHub.

Ajouter un en-tête à une demande d'API

La commande create-integration suivante ajoute un en-tête nommé `header1` à une demande d'API avant que celle-ci 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

La commande create-integration suivante renomme un en-tête de requête `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

La commande create-integration suivante configure les paramètres de réponse pour une intégration. Lorsque les intégrations renvoient un code de statut 500, API Gateway modifie le code de statut en 403 et ajoute `header11` à la réponse. Lorsque l'intégration renvoie un code de statut 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

La commande update-integration suivante supprime les paramètres de requête précédemment configurés pour `append:header.header1`. Il supprime également les paramètres de réponse précédemment configurés pour un code de statut 200.

```
aws apigatewayv2 update-integration \
    --api-id abcdef123 \
    --integration-id hijk456 \
    --request-parameters '{"append:header.header1" : ""}' \
    --response-parameters '{"200" : {}}'
```

Utiliser les définitions OpenAPI pour HTTP APIs dans API Gateway

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, consultez 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, consultez Exportation d'une API REST à partir d'API Gateway.

> **Note**
>
> Le protocole HTTP APIs prend en charge les mêmes AWS variables que le protocole REST APIs. Pour en savoir plus, consultez la section AWS Variables pour l'importation OpenAPI.

Importer des informations de validation

Lorsque vous importez une API, API Gateway fournit trois catégories d'informations de validation.

Info (Infos)

Une propriété est valide selon la spécification OpenAPI, mais elle n'est pas prise en charge pour le protocole HTTP. APIs

Par exemple, l'extrait OpenAPI 3.0 suivant produit des informations lors de l'importation, car le protocole HTTP APIs ne prend pas en charge la validation des demandes. 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 génère des avertissements lors de l'importation, car HTTP ne prend en APIs charge que les intégrations de proxy Lambda et de 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 le protocole HTTP ne APIs prend 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 import-api 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"
        ]
      }
    }
  }
}
```

Exporter le protocole HTTP APIs 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 plus d'informations sur l'importation de définitions d'API, consultez Importation d'une API HTTP.

Exporter une définition OpenAPI 3.0 d'une étape à l'aide de la CLI AWS

La commande export-api suivante exporte une définition OpenAPI d'une étape d'API nommée `prod` dans un fichier YAML nommé `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 export-api suivante exporte une définition OpenAPI d'une API HTTP vers un fichier JSON nommé `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
```

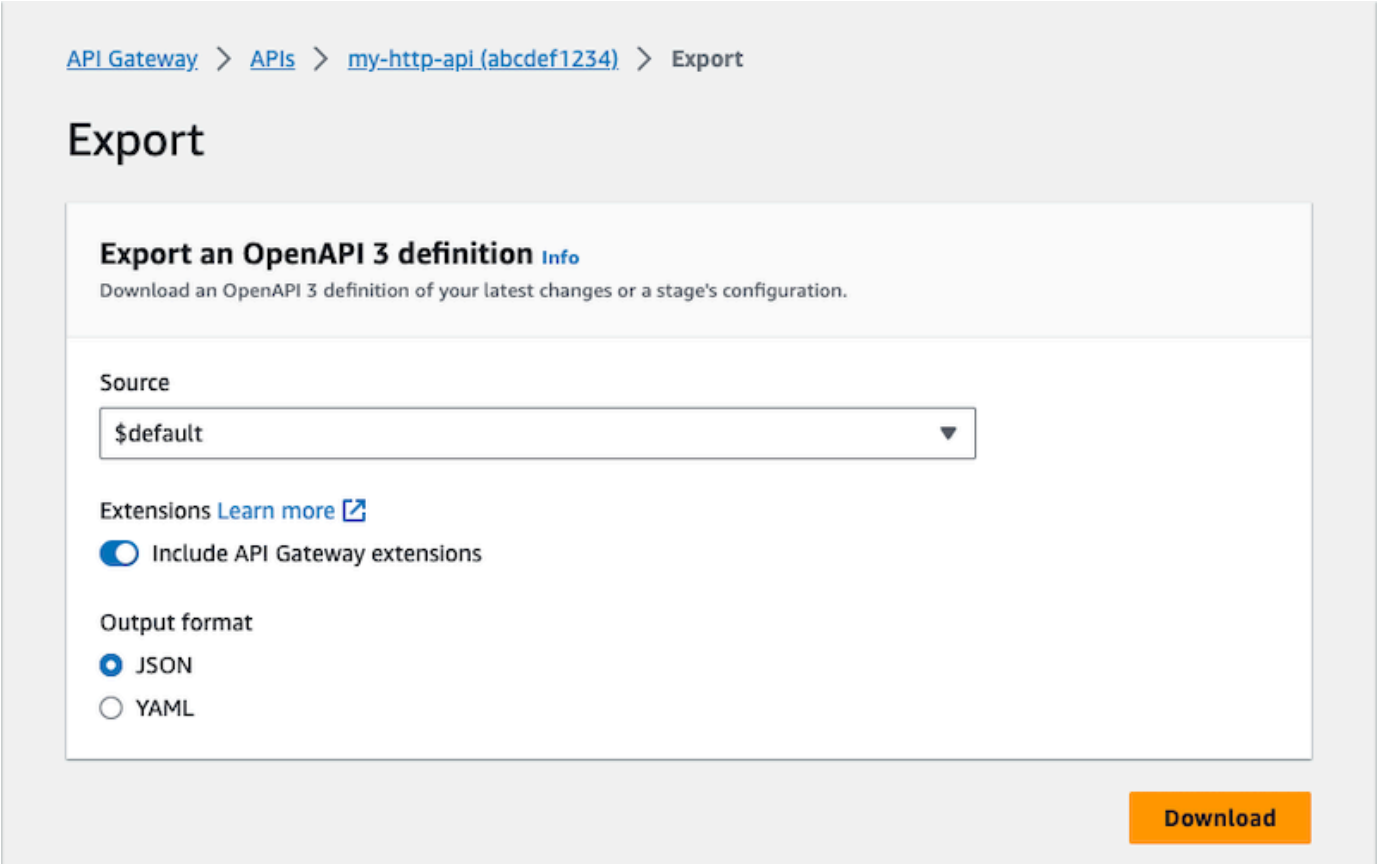
Pour plus d'informations, consultez ExportAPI dans le document Référence des API Amazon API Gateway Version 2.

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 invocables 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

- Étapes du protocole HTTP APIs dans API Gateway
- Politique de sécurité pour le protocole HTTP APIs dans API Gateway
- Noms de domaine personnalisés pour les API HTTP dans API Gateway

Étapes du protocole HTTP APIs dans API Gateway

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 dans 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.

Utiliser des variables d'étape pour le protocole HTTP APIs dans API Gateway

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.

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".

Exemple : utilisation d'une variable d'étape pour personnaliser le point de terminaison d'intégration HTTP

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.

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. La commande add-permission suivante configure l'autorisation pour la fonction Lambda :

```
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 de stage API Gateway pour HTTP APIs dans API Gateway

Vous pouvez utiliser les variables d'étape API Gateway pour HTTP APIs dans les cas suivants.

Intégration HTTP URIs

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 le protocole HTTP APIs dans API Gateway

API Gateway applique une politique de sécurité `TLS_1_2` pour tous les points de terminaison des API HTTP.

Une politique de sécurité est une combinaison prédéfinie d'une version minimale du protocole TLS et de suites de chiffrement offerte 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 politique 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 HTTP APIs

Le tableau suivant décrit les protocoles TLS pris en charge pour le protocole HTTP APIs.

Protocoles TLS	Politique de sécurité TLS_1_2
TLSv13.	Oui
TLSv12.	Oui

Le tableau suivant décrit les chiffrements TLS disponibles pour la politique de sécurité TLS 1_2 pour HTTP. APIs

Chiffrements TLS	Politique de sécurité TLS_1_2
TLS-AES-128-GCM- SHA256	Oui

Chiffrements TLS	Politique de sécurité TLS_1_2
TLS-AES-256-GCM- SHA384	Oui
TLS- CHACHA20 - - POLY1305 SHA256	Oui
ECDHE-ECDSA- -GCM- AES128 SHA256	Oui
ECDHE-RSA- -GCM- AES128 SHA256	Oui
ECDHE-ECDSA- - AES128 SHA256	Oui
ECDHE-RSA- - AES128 SHA256	Oui
ECDHE-ECDSA- -GCM- AES256 SHA384	Oui
ECDHE-RSA- -GCM- AES256 SHA384	Oui
ECDHE-ECDSA- - AES256 SHA384	Oui
ECDHE-RSA- - AES256 SHA384	Oui
AES128-GCM- SHA256	Oui
AES128-SHA256	Oui
AES256-GCM- SHA384	Oui
AES256-SHA256	Oui

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 de chiffrement, consultez the section called “Noms de chiffrement OpenSSL et RFC”.

Informations sur REST APIs et WebSocket APIs

Pour plus d'informations sur REST APIs et WebSocket APIs, voir the section called “Sélection d’une politique de sécurité” etthe section called “Politique de sécurité pour WebSocket APIs in API Gateway”.

Noms de domaine personnalisés pour les API HTTP dans API Gateway

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`* est la région AWS et *`stage`* est spécifié par vous lors du déploiement de l’API.

La partie nom d’hôte de l’URL, *`api-id`*`.execute-api.`*`region`*`.amazonaws.com`, fait référence à un point de terminaison de l’API. Le nom par défaut du point de terminaison de l’API est généré de manière aléatoire, difficile à mémoriser et peu 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
```

Considérations

Les considérations suivantes peuvent avoir une incidence sur votre utilisation d’un nom de domaine personnalisé.

- Un nom de domaine personnalisé régional peut être associé à des API REST et HTTP. Vous pouvez utiliser les API API Gateway version 2 pour créer et gérer des noms de domaine personnalisés régionaux pour les API REST.
- Pour la version TLS minimale, seule la version TLS 1.2 est prise en charge.

- Vous devez créer ou mettre à jour l'enregistrement de ressource de votre fournisseur DNS pour le mapper au point de terminaison de votre API. Sans ce mappage, les demandes d'API destinées au nom de domaine personnalisé ne peuvent pas atteindre API Gateway.
- Vous pouvez prendre en charge un nombre presque infini de noms de domaine sans dépasser le quota par défaut en utilisant un certificat générique. Pour de plus amples informations, consultez the section called "Noms de domaine personnalisés génériques".

Prérequis

Les conditions suivantes sont requises pour créer un nom de domaine personnalisé.

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. Vous pouvez enregistrer votre nom de domaine Internet avec Amazon Route 53 ou utiliser un bureau d'enregistrement de domaine tiers de votre choix. Votre nom de domaine personnalisé peut être le nom d'un sous-domaine ou le domaine racine (également nommé « zone apex ») d'un domaine Internet enregistré.

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.

Certificats pour les noms de domaine personnalisés

Avant de configurer un nom de domaine personnalisé pour une API, vous devez avoir un certificat SSL/TLS prêt dans ACM. Si ACM n'est pas disponible dans la région AWS où vous créez votre nom de domaine personnalisé, 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. 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 utiliser un certificat géré par AWS pour un nom de domaine, indiquez simplement son ARN.

Si votre application utilise l'épinglage de certificat, parfois appelé épinglage SSL, pour épingler un certificat ACM, l'application ne pourra peut-être pas se connecter à votre domaine une fois qu'AWS aura renouvelé le certificat. Pour plus d'informations, consultez la section Problèmes d'épinglage de certificat dans le Guide de l'utilisateur AWS Certificate Manager.

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 compte AWS, vous pouvez configurer *.example.com et a.example.com pour qu'ils se comportent 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 un nom de domaine personnalisé générique si un autre compte AWS a créé un nom de domaine personnalisé en conflit avec ce nom. 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 AWSCentre de support pour demander une exception.

Étapes suivantes pour les noms de domaine personnalisés

Pour configurer un nom de domaine personnalisé pour une API HTTP, utilisez la documentation de la section API REST du Guide du développeur API Gateway.

D'abord, spécifiez un certificat pour votre nom de domaine personnalisé. Pour de plus amples informations, consultez the section called "Préparez les certificats dans AWS Certificate Manager".

Ensuite, créez un nom de domaine personnalisé régional. Pour de plus amples informations, consultez the section called “Configuration d’un nom de domaine personnalisé régional”.

Mappage des étapes d’API à un nom de domaine personnalisé 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.

Requête	API sélectionnée	Explication
		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 sur le même compte AWS.
- 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. Cette limite n'inclut pas les mappages d'API à un niveau tels que `/prod`.
- Vous ne pouvez mapper les API HTTP à un nom de domaine personnalisé régional qu'à l'aide de la politique de sécurité TLS 1.2.
- Vous ne pouvez pas mapper les API WebSocket au même nom de domaine personnalisé qu'une API HTTP ou une API REST.
- Si vous créez un mappage d'API à plusieurs niveaux, API Gateway convertit tous les noms d'en-tête en minuscules.

Création d'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 accéder à des exemples de modèles AWS Serverless Application Model qui créent toutes les ressources, consultez Séances avec SAM sur GitHub.

Console de gestion AWS

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 Save (Enregistrer).

AWS CLI

La commande create-api-mapping 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
```

CloudFormation

L'exemple CloudFormation 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`. Lorsque vous désactivez le point de terminaison par défaut, toutes les étapes d'une API sont affectées.

La procédure suivante explique comment désactiver le point de terminaison par défaut pour une API HTTP.

Console de gestion AWS

1. Connectez-vous à la console API Gateway à l'adresse : https://console.aws.amazon.com/apigateway.
2. Choisissez une API HTTP.
3. Choisissez l'ID de votre API pour ouvrir la page Détails de l'API.
4. Dans l'onglet Détails de l'API, choisissez Modifier.
5. Pour Point de terminaison par défaut, sélectionnez Désactiver.
6. Choisissez Save (Enregistrer).

   Si vous activez les déploiements automatiques pour votre étape, vous n'avez pas besoin de redéployer votre API pour que la modification prenne effet. Sinon, vous devez redéployer votre API.
7. (Facultatif) Choisissez Déployer, puis redéployez votre API ou créez une étape pour que la modification prenne effet.

AWS CLI

La commande update-domain-name 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 commande create-deployment suivante crée un déploiement :

```
aws apigatewayv2 create-deployment \
    --api-id abcdef123 \
    --stage-name dev
```

Types d'adresses IP des noms de domaine personnalisés pour les API HTTP

Lorsque vous créez une API, vous spécifiez le type d'adresses IP qui peut invoquer votre domaine. Vous avez le choix entre IPv4 (pour autoriser les adresses IPv4 à invoquer votre domaine) et Dualstack (pour autoriser les adresses IPv4 et IPv6 à invoquer votre domaine). Nous vous recommandons de définir le type d'adresse IP sur Dualstack pour éviter l'épuisement de l'espace IP ou renforcer votre niveau de sécurité. Pour plus d'informations sur les avantages d'un type d'adresse IP à double pile, consultez IPv6 sur AWS.

Considérations relatives aux types d'adresses IP

Les considérations suivantes peuvent avoir une incidence sur votre utilisation des types d'adresses IP.

- Le type d'adresse IP par défaut des noms de domaine personnalisés API Gateway est IPv4.
- Il n'est pas nécessaire que votre nom de domaine personnalisé possède le même type d'adresse IP pour toutes les API qui y sont mappées. La désactivation du point de terminaison de votre API par défaut peut avoir une incidence sur la manière dont les appelants peuvent invoquer votre API.

Modification du type d'adresse IP d'un nom de domaine personnalisé

Vous pouvez modifier le type d'adresse IP en mettant à jour la configuration de point de terminaison du domaine. Vous pouvez mettre à jour la configuration du point de terminaison du domaine à l'aide de la Console de gestion AWS, de l'AWS CLI, d'CloudFormation ou d'un kit SDK AWS.

Console de gestion AWS

Pour modifier le type d'adresse IP d'un nom de domaine personnalisé

1. Connectez-vous à la console API Gateway à l'adresse : https://console.aws.amazon.com/apigateway.
2. Choisissez un nom de domaine personnalisé public.
3. Sélectionnez Configuration du point de terminaison.
4. Pour Type d'adresse IP, choisissez IPv4 ou Dualstack.
5. Choisissez Save (Enregistrer).

AWS CLI

La commande update-domain-name suivante met à jour une API pour qu'elle possède le type d'adresse IP Dualstack :

```
aws apigatewayv2 update-domain-name \
   --domain-name dualstack.example.com \
   --domain-name-configurations CertificateArn=arn:aws:acm:us-
east-1:111122223333:certificate/abcd1234-5678-abc,IpAddressType=dualstack
```

Le résultat se présente comme suit :

```
{
    "ApiMappingSelectionExpression": "$request.basepath",
    "DomainName": "dualstack.example.com",
    "DomainNameConfigurations": [
        {
            "ApiGatewayDomainName": "d-abcd1234.execute-api.us-
east-1.amazonaws.com",
            "CertificateArn": "arn:aws:acm:us-east-1:111122223333:certificate/
abcd1234-5678-abc",
            "DomainNameStatus": "AVAILABLE",
```

```
            "EndpointType": "REGIONAL",
            "HostedZoneId": "Z3LQWSYCGH4ADY",
            "SecurityPolicy": "TLS_1_2",
            "IpAddressType": "dualstack"
        }
    ],
    "Tags": {}
}
```

Protection de vos API HTTP dans API Gateway

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 politiques 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

- Limitez les requêtes adressées à votre HTTP APIs pour un meilleur débit dans API Gateway
- Comment activer l'authentification TLS mutuelle pour votre protocole HTTP APIs dans API Gateway

Limitez les requêtes adressées à votre HTTP APIs pour un meilleur débit dans API Gateway

Vous pouvez configurer la régulation pour éviter qu'ils ne soient submergés par un trop grand nombre de demandes. APIs 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. Plus précisément, API Gateway examine le taux et le nombre de demandes soumises par rapport APIs à l'ensemble 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 cibles pour les différentes étapes ou itinéraires d'API afin d'améliorer les APIs performances globales de l'ensemble de votre compte.

Limitation au niveau du compte par région

Par défaut, API Gateway limite le nombre de demandes stables par seconde (RPS) sur l'ensemble APIs d'un AWS compte, par région. Cela limite également la rafale (c'est-à-dire la taille maximale du bucket) sur l'ensemble APIs du AWS compte, 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 tous les APIs membres d'un compte dans une région spécifiée. La limite de débit au niveau du compte peut être augmentée sur demande. Des limites plus élevées sont possibles avec APIs des délais d'attente plus courts et des charges utiles plus faibles. Pour demander une augmentation des limitations au niveau du compte par Région, contactez le Centre de support AWS. Pour de plus amples informations, veuillez consulter 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 update-stage suivante configure une limitation personnalisée pour l'étape et la route spécifiées d'une API :

```
aws apigatewayv2 update-stage \
    --api-id a1b2c3d4 \
    --stage-name dev \
    --route-settings '{"GET /pets":
{"ThrottlingBurstLimit":100,"ThrottlingRateLimit":2000}}'
```

Comment activer l'authentification TLS mutuelle pour votre protocole HTTP APIs dans API Gateway

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 "Désactivation du 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 Autorité de certification privée AWS certificat ou d'un produit importé

Pour utiliser un certificat importé dans ACM ou un certificat provenant Autorité de certification privée AWS 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'en avez pas encore`ownershipVerificationCertificate`, rendez-vous sur https://console.aws.amazon.com/acm/pour en créer 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'autorit é 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 certifica ts peuvent avoir une longueur de chaîne maximale de quatre.

Chargement du 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 cp 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 plus d'informations sur la création et la configuration d'un nom de domaine personnalisé, consultez the section called "Configuration d'un nom de domaine personnalisé régional".

> **Note**
>
> Le protocole TLS mutuel n'est pas pris en charge pour le mode privé APIs.

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. Ce qui suit create-domain-namecrée un nom de domaine personnalisé avec le protocole TLS mutuel :

```
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 code de statut `403`. 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 update-domain-namecommande 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.

La update-domain-namecommande suivante met à jour un nom de domaine personnalisé afin de supprimer le truststore de votre nom de domaine personnalisé :

```
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 d'authentification TLS mutuelle pour votre API HTTP

Les rubriques suivantes fournissent des conseils de dépannage pour les erreurs et problèmes que vous pouvez rencontrer lorsque vous activez l'authentification TLS mutuelle.

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 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 et que sa validation a échoué car le nouveau certificat possède un SAN (nom alternatif du sujet) qui n'est pas couvert par le `ownershipVerificationCertificate` ou le sujet ou SANs le certificat ne couvre 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.

Surveiller le protocole HTTP APIs dans API Gateway

Vous pouvez utiliser CloudWatch les métriques et les CloudWatch journaux pour surveiller le protocole HTTP APIs. 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 des métriques CloudWatch pour les API HTTP dans API Gateway
- Configuration de la journalisation pour HTTP APIs dans API Gateway

Surveillance des métriques CloudWatch pour les API HTTP dans API Gateway

Vous pouvez surveiller l'exécution de l'API avec CloudWatch, qui recueille les données brutes d'API Gateway et les traite en métriques lisibles et disponibles presque en temps réel. 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 des métriques API Gateway sont automatiquement envoyées à CloudWatch toutes les minutes. Pour surveiller vos métriques, créez un tableau de bord CloudWatch pour votre API. Pour plus d'informations sur la façon de créer un tableau de bord CloudWatch, consultez Création d'un tableau de bord CloudWatch dans le Guide de l'utilisateur Amazon CloudWatch. Pour plus d'informations, consultez la rubrique Qu'est-ce qu'Amazon CloudWatch ? du Guide de l'utilisateur Amazon CloudWatch.

Les métriques suivantes sont prises en charge pour les API HTTP. Vous pouvez également activer des métriques détaillées pour écrire des métriques de niveau route sur Amazon CloudWatch.

Métrique	Description
4xx	Nombre d'erreurs côté client capturées dans une période donnée.

Métrique	Description
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.
Données traitées	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, Stage	Filtre les métriques API Gateway pour trouver une

Dimension	Description
	étape d'API portant l'ID d'API et l'ID d'étape spécifiés.
ApiId, Method, Resource, Stage	Filtre les métriques API Gateway pour trouver une méthode d'API portant l'ID d'API, l'ID d'étape, le chemin de ressource et l'ID de routage spécifiés. API Gateway n'envoie pas ces métriques à moins que vous n'ayez explicitement activé les métriques CloudWatch détaillées. Pour ce faire, appelez l'action UpdateStage de l'API REST d'API Gateway V2 pour mettre à jour la propriété `detailedMetricsEnabled` sur `true`. Sinon, vous pouvez appeler la commande update-stage de l'AWS CLI pour mettre à jour la propriété `DetailedMetricsEnabled` sur `true`. L'activation de ces métriques implique des frais supplémentaires pour votre compte. Pour plus d'informations, consultez Tarification Amazon CloudWatch.

Configuration de la journalisation pour HTTP APIs dans API Gateway

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 améliorer votre niveau de sécurité, nous vous recommandons d'écrire des journaux dans CloudWatch Logs pour toutes les étapes de votre API HTTP. Vous devrez peut-être le faire pour vous conformer aux différents cadres de conformité. Pour plus d'informations, consultez Amazon API Gateway controls dans le Guide de l'utilisateur AWS Security Hub .

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

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogGroups",
                "logs:DescribeLogStreams",
                "logs:GetLogEvents",
                "logs:FilterLogEvents"
            ],
            "Resource": "arn:aws:logs:us-east-2:123456789012:log-group:*"
        },
        {
            "Effect": "Allow",
```

```
            "Action": [
                "logs:CreateLogDelivery",
                "logs:PutResourcePolicy",
                "logs:UpdateLogDelivery",
                "logs:DeleteLogDelivery",
                "logs:CreateLogGroup",
                "logs:DescribeResourcePolicies",
                "logs:GetLogDelivery",
                "logs:ListLogDeliveries"
            ],
            "Resource": "*"
        }
    ]
}
```

Création d'un groupe de journaux et activation de la journalisation pour HTTP APIs

Vous pouvez créer un groupe de journaux et activer la journalisation des accès à l'aide du Console de gestion AWS ou du AWS CLI.

Console de gestion AWS

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 create-log-groupcommande 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 : ::log-group :. *region account-id log-group-name*

La commande update-stage suivante active la journalisation pour l'étape `$default` 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 pour les API HTTP

Vous pouvez utiliser les variables suivantes pour personnaliser les journaux d'accès des API HTTP. Pour en savoir plus sur les journaux d'accès pour HTTP APIs, consultezConfiguration de la journalisation pour HTTP APIs dans API Gateway.

Paramètre	Description
`$context.accountId`	ID de AWS compte du propriétaire de l'API.
`$context.apiId`	Identifiant qu'API Gateway attribue à votre API.
`$context.authorizer.claims.` *`property`*	Propriété des requêtes renvoyées depuis le jeton JWT (JSON Web Token) une fois que l'appelant de la méthode a été authentifié avec succès, par exemple `$context.authorizer.claims.username` . Pour de plus amples informations, veuillez consulter Contrôle de l'accès aux API HTTP avec des mécanismes d'autorisation JWT dans API Gateway. **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.` *`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 plus d'informations, consultez 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
`$context.httpMethod`	Méthode HTTP utilisée. Les valeurs valides sont les suivantes : `DELETE`, `GET`, `HEAD`, `OPTIONS`, `PATCH`, `POST` et `PUT`.
`$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 de tous les 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 groupe 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 plus d'informations sur les fournisseurs d'authentification Amazon Cognito disponibles, consultez Using Federated Identities dans le Guide 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` .
`$context.integration.status`	Code de statut renvoyé à partir d'une intégration. Pour les intégrations de proxy Lambda, code de statut que votre code de fonction Lambda renvoie.
`$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` .
`$context.protocol`	Protocole de demande, par exemple, `HTTP/1.1`. **Note** API Gateway APIs peut accepter les requêtes HTTP/2, mais API Gateway envoie des demandes aux intégrati ons de backend à l'aide du protocole HTTP/1.1. Par conséquent, le protocole de requête est enregistré comme HTTP/1.1 même si un client envoie une requête qui utilise HTTP/2.
`$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.

Paramètre	Description
`$context.routeKey`	La clé de routage de la demande d'API, par exemple `/pets`.
`$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 des API HTTP dans API Gateway

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

Vous trouverez ci-dessous des conseils de résolution des erreurs et des problèmes que vous pourriez rencontrer lors de l'utilisation AWS Lambda intégrations du protocole HTTP APIs.

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 Console de gestion AWS

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 : ::log-group :. *region account-id 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`**

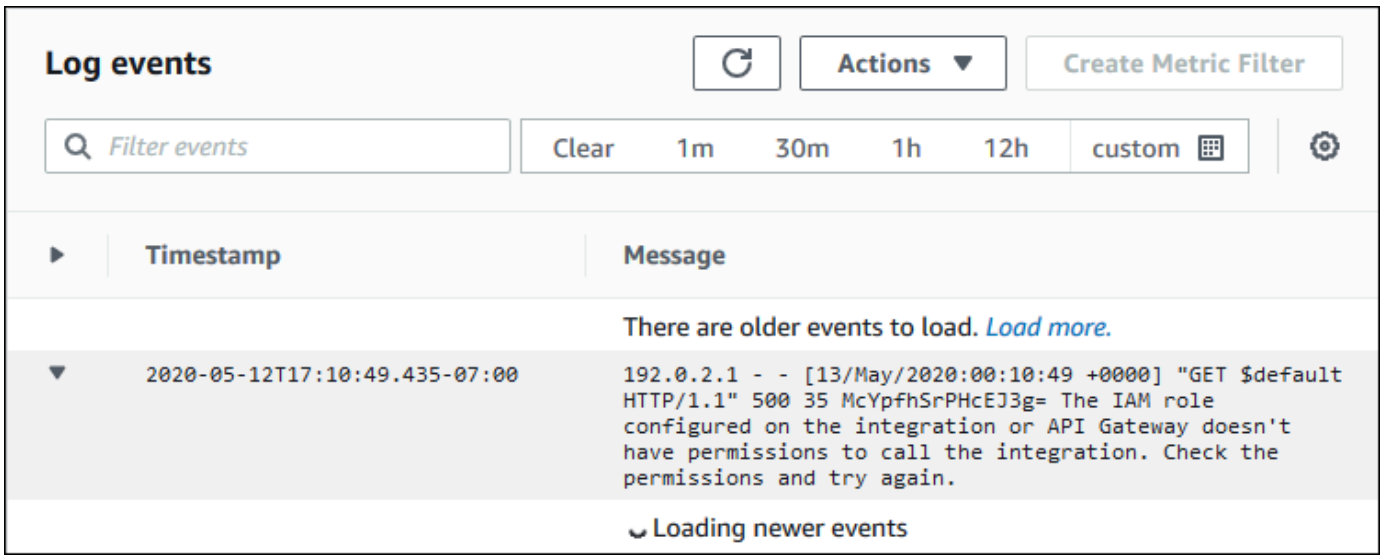
1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Choisissez votre API HTTP.
3. Sous Moniteur, choisissez Journalisation.
4. Sélectionnez une étape de votre API.
5. Choisissez Modifier, puis activez la journalisation des accès.
6. Pour Destination du 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 CloudFormation, ou d'un SDK, vous devez autoriser API Gateway à appeler la fonction. Les commandes add-permission suivantes autorisent différentes routes d'API HTTP à invoquer une fonction Lambda.

Example Exemple : pour l'étape **`$default`** et la route **`$default`** 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'étape **`prod`** et la route **`test`** 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 politique 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

Vous trouverez ci-dessous des conseils de résolution des erreurs et des problèmes que vous pourriez rencontrer lors de l'utilisation des autorisateurs JSON Web Token (JWT) avec HTTP. APIs

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 ne valide `client_id` que si `aud` n'est pas présent. Si `aud` et `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 Web jwt.io peut être JWTs débogué dans le navigateur. L'OpenID Foundation tient à jour une liste de bibliothèques avec lesquelles travailler. JWTs

Pour plus d'informations sur les mécanismes d'autorisation JWT, consultez Contrôle de l'accès aux API HTTP avec des mécanismes d'autorisation JWT dans API Gateway.

API WebSocket API Gateway

Une API WebSocket dans API Gateway est un ensemble de routes WebSocket qui sont intégrées avec les points de terminaison HTTP backend, les fonctions Lambda ou 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 des interactions client/service plus riches car les services peuvent transmettre des données aux clients sans exiger d'eux qu'ils fassent une demande explicite. Les API WebSocket 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 WebSocket chat avec une WebSocket API, Lambda et DynamoDB.

Dans cette section, vous pouvez apprendre à développer, publier, protéger et surveiller vos API WebSocket à l'aide d'API Gateway.

Rubriques

- Présentation des WebSocket API dans API Gateway
- Développez WebSocket APIs dans API Gateway
- Publication d'API WebSocket invocables par les clients
- Protégez votre WebSocket APIs API Gateway
- Surveillance des API WebSocket dans API Gateway

Présentation des WebSocket 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. (Non-JSON les messages sont dirigés vers un `$default` itinéraire 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.

Utiliser des routes pour une WebSocket API

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 connectées".

Pour plus d'informations sur la route `$default` et les routes personnalisées, consultez la section the section called "Invocation de votre intégration backend".

Envoi de données à des applications client connectées

Les services backend peuvent envoyer des données vers des applications client connectées. Vous pouvez envoyer des données en procédant de la façon suivante :

- Utilisez une intégration pour 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 de plus amples informations, veuillez consulter the section called "Utilisation des commandes `@connections` dans votre service backend".

WebSocket Codes d'état de l'API

Les API WebSocket API Gateway utilisent les codes d'état suivants pour les communications entre le serveur et le client, comme décrit dans le registre des numéros de code de WebSocket fermeture :

1001

API Gateway renvoie ce code d'état lorsque le client est inactif pendant 10 minutes ou a atteint la durée maximale de connexion de 2 heures.

1003

API Gateway renvoie ce code d'état lorsqu'un point de terminaison reçoit un type de médias binaires. Les types de média binaires ne sont pas pris en charge pour les WebSocket API.

1005

API Gateway renvoie ce code d'état si le client envoie un frame de fermeture sans code de fermeture.

1006

API Gateway renvoie ce code d'état en cas de fermeture inattendue de la connexion, telle que la fermeture de la connexion TCP sans cadre de WebSocket fermeture.

1008

API Gateway renvoie ce code d'état lorsqu'un point de terminaison reçoit un trop grand nombre de demandes d'un client en particulier.

1009

API Gateway renvoie ce code d'état lorsqu'un point de terminaison reçoit un message trop volumineux pour être traité.

1011

API Gateway renvoie ce code d'état en cas d'erreur interne du serveur.

1012

API Gateway renvoie ce code d'état si le service redémarre.

Gestion des utilisateurs connectés et des applications clientes : **routes $connect** et **$disconnect**

La section suivante décrit comment utiliser les `$disconnect` routes `$connect` et pour votre WebSocket API.

Rubriques

- La route $connect
- Transmission des informations de connexion depuis la route $connect
- La route $disconnect

La **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, consultez Configurer 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 des identifiants de connexion, des noms de domaine ou d'autres informations de connexion dans une base de données (par exemple, Amazon DynamoDB). L'ID de connexion est une chaîne de longueur variable dont la longueur et le format peuvent changer.

Transmission des informations de connexion depuis la **`route $connect`**

Vous pouvez utiliser des intégrations de proxy et autres que de 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 autre que de proxy

- Vous pouvez accéder aux informations de connexion avec une intégration autre que de 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 autre que de 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 autre que de 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.

La `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".

Invoquez l'intégration de votre backend avec la route **$default** et les routes personnalisées dans API Gateway

La section suivante décrit comment invoquer votre intégration principale à l'aide de la `$default` route ou d'une route personnalisée pour une WebSocket API.

Rubriques

- Utilisation de routes pour traiter les messages
- La 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 request/response modèle 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 routés vers des intégrations spécifiques en fonction du contenu du message. Non-JSON les messages sont transmis au backend par la `$default` route.

> **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 supports 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.

La `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 "Vue d'ensemble des WebSocket 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.

WebSocket expressions 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*.

Rubriques

- Expressions de sélection de la réponse de routage
- Expressions de sélection de la clé d'API
- Expressions de sélection du mappage d'API
- WebSocket résumé de l'expression de sélection

Expressions de sélection de la réponse de routage

Une réponse de routage est utilisée pour modéliser une réponse du serveur principal au client. En effet WebSocket APIs, 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 sélectionnée 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`.

WebSocket résumé de l'expression de sélection

Le tableau suivant récapitule les cas d'utilisation des expressions de sélection dans WebSocket APIs :

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 que route fourre-tout.	WebSocket Acheminez les messages en fonction du contexte d'une demande du client.
`Route.ModelSelectionExpression`	Clé pour `Route.RequestModels`	Facultatif. Si elle est fournie pour une intégration autre que de proxy, la validatio	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
		n du modèle a lieu. `$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.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 prises 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}` , `${integration.response.multivalueheader.headerName}` , et les valeurs statiques sont prises en charge. `$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.	

Développez WebSocket APIs dans API Gateway

Cette section fournit des détails sur les fonctionnalités d'API Gateway dont vous avez besoin pour développer votre API Gateway APIs.

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 WebSocket APIs dans API Gateway
- Types d'adresses IP pour les API WebSocket dans API Gateway
- Création de routes pour WebSocket APIs in API Gateway
- Contrôler et gérer l'accès WebSocket APIs à API Gateway
- Intégrations pour WebSocket APIs in API Gateway
- Demande de validation pour WebSocket APIs in API Gateway
- Transformations de données pour WebSocket APIs dans API Gateway
- Types de supports binaires pour les API WebSocket dans API Gateway
- Invoquer WebSocket APIs

Création WebSocket APIs 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 APIs ne supporte que TLS 1.2 et TLS 1.3. Les versions antérieures de TLS ne sont pas prises en charge.

Création d'une WebSocket API à l'aide de AWS CLI commandes

La commande create-api suivante crée une API avec l'expression de sélection de route `$request.body.action` :

```
aws apigatewayv2 --region us-east-1 create-api --name "myWebSocketApi3" --protocol-type
 WEBSOCKET --route-selection-expression '$request.body.action'
```

Le résultat se présente comme suit :

```
{
    "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.

Types d'adresses IP pour les API WebSocket dans API Gateway

Lorsque vous créez une API, vous spécifiez le type d'adresses IP qui peut invoquer votre API. Vous avez le choix entre IPv4 (pour autoriser les adresses IPv4 à invoquer votre API) et Dualstack (pour autoriser les adresses IPv4 et IPv6 à invoquer votre API). Nous vous recommandons de définir le type d'adresse IP sur Dualstack pour éviter l'épuisement de l'espace IP ou renforcer votre niveau de sécurité. Pour plus d'informations sur les avantages d'un type d'adresse IP à double pile, consultez IPv6 sur AWS.

Considérations relatives aux types d'adresses IP

Les considérations suivantes peuvent avoir une incidence sur votre utilisation des types d'adresses IP :

- Le type d'adresses IP par défaut pour toutes les API WebSocket est IPv4.
- Si vous remplacez le type d'adresse IP IPv4 d'une API existante par un type d'adresse IP à double pile, vérifiez que toutes les stratégies d'accès à vos API ont été mises à jour pour prendre en compte les appels IPv6. La modification du type d'adresse IP prend effet immédiatement.
- Votre API peut être mappée à un nom de domaine personnalisé présentant un type d'adresse IP différent de celui de votre API. La désactivation du point de terminaison de votre API par défaut peut avoir une incidence sur la manière dont les appelants peuvent invoquer votre API.

Modification du type d'adresse IP d'une API WebSocket

Vous pouvez modifier le type d'adresse IP en mettant à jour la configuration de l'API. Vous pouvez mettre à jour la configuration de l'API à l'aide de la Console de gestion AWS, de l'AWS CLI, d'CloudFormation ou d'un kit SDK AWS. Si vous modifiez le type d'adresse IP de l'API, vous n'avez pas besoin de redéployer votre API pour que les modifications prennent effet.

Console de gestion AWS

Pour modifier le type d'adresse IP d'une API WebSocket

1. Connectez-vous à la console API Gateway à l'adresse : https://console.aws.amazon.com/apigateway.
2. Choisissez une API WebSocket.
3. Choisissez Paramètres de l'API, puis Modifier.
4. Pour Type d'adresse IP, choisissez IPv4 ou Dualstack.
5. Choisissez Save (Enregistrer).

   La modification de la configuration de votre API prend effet immédiatement.

AWS CLI

La commande update-api suivante met à jour une API pour qu'elle possède le type d'adresse IP Dualstack :

```
aws apigatewayv2 update-api \
    --api-id abcd1234 \
    --ip-address-type dualstack
```

Le résultat se présente comme suit :

```
{
    "ApiEndpoint": "https://abcd1234.execute-api.us-east-1.amazonaws.com",
    "ApiId": "abcd1234",
    "ApiKeySelectionExpression": "$request.header.x-api-key",
    "CreatedDate": "2025-02-04T22:20:20+00:00",
    "DisableExecuteApiEndpoint": false,
    "Name": "My-WebSocket-API",
    "ProtocolType": "WEBSOCKET",
    "RouteSelectionExpression": "$request.method $request.path",
    "Tags": {},
    "NotificationUris": [],
    "IpAddressType": "dualstack"
}
```

Création de routes pour WebSocket APIs in API Gateway

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 WebSocket based APIs, 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, il `request.body` fait référence à la charge utile JSON de votre message et `.action` est une JSONPathexpression. 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 JSONPath expression renvoie un tableau de deux éléments, celui-ci sera présenté 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 le n' JSONPath est pas trouvé, la variable est résolue sous la forme « ».
`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 plus d'informations, consultez the section called "Vue d'ensemble des WebSocket 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 de plus amples informations, veuillez consulter 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

La commande create-route suivante crée une route :

```
aws apigatewayv2 --region us-east-1 create-route --api-id aabbccddee --route-key
 $default
```

Le résultat se présente comme suit :

```
{
    "ApiKeyRequired": false,
    "AuthorizationType": "NONE",
    "RouteKey": "$default",
    "RouteId": "1122334"
}
```

Spécification des paramètres de requête de routage 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 plus d'informations, consultez the section called "La `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 plus d'informations sur les mécanismes d'autorisation Lambda, consultez 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 mécanisme d'autorisation 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.

- Clé d'API obligatoire : si vous le souhaitez, vous pouvez exiger une clé d'API pour une route `$connect` de l'API. Vous pouvez utiliser des clés d'API ainsi que des plans d'utilisation pour contrôler et suivre l'accès à votre APIs. 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, procédez comme suit :

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 requête de routage, 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 WebSocket APIs 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 ne pouvez définir la réponse de l'`$default`itinéraire que pour WebSocket APIs. Vous pouvez utiliser une réponse d'intégration pour manipuler la réponse d'un service backend.

Pour de plus amples informations, veuillez consulter 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 avec une intégration de la fonction Lambda par 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 dans 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

La create-route-responsecommande suivante crée une réponse d'itinéraire pour l'`$default`itinéraire. Vous pouvez identifier l'ID de l'API et l'ID du routage à l'aide des commandes get-apis et get-routes.

```
aws apigatewayv2 create-route-response \
    --api-id aabbccddee \
    --route-id 1122334  \
    --route-response-key '$default'
```

Le résultat se présente comme suit :

```
{
    "RouteResponseId": "abcdef",
    "RouteResponseKey": "$default"
}
```

Configurer 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 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'invocation WebSocket APIs, consultezInvoquer WebSocket APIs.

Contrôler et gérer l'accès WebSocket APIs à 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 politiques IAM pour contrôler qui peut les créer et les gérer APIs, ainsi que les personnes habilitées à les invoquer. Pour de plus amples informations, veuillez consulter Contrôler l'accès à l' WebSocket APIs aide d'une autorisation IAM.
- Les balises IAM peuvent être utilisées avec des politiques 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 autorisateurs Lambda sont des fonctions Lambda qui contrôlent l'accès à. APIs Pour de plus amples informations, veuillez consulter Contrôlez l'accès à l' WebSocket APIs aide des AWS Lambda autorisateurs REQUEST.

Pour améliorer votre niveau de sécurité, nous vous recommandons de configurer un autorisateur pour l'`$connect`itinéraire sur tous vos WebSocket APIs itinéraires. Vous devrez peut-être le faire pour vous conformer aux différents cadres de conformité. Pour plus d’informations, consultez Amazon API Gateway controls dans le Guide de l’utilisateur AWS Security Hub .

Rubriques

- Contrôler l'accès à l' WebSocket APIs aide d'une autorisation IAM
- Contrôlez l'accès à l' WebSocket APIs aide des AWS Lambda autorisateurs REQUEST

Contrôler l'accès à l' WebSocket APIs aide d'une autorisation IAM

L'autorisation IAM dans WebSocket APIs est similaire à celle pour REST APIs, 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'`@connections`API utilise le même format ARN que REST APIs :

```
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 politique 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.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "execute-api:Invoke"
            ],
            "Resource": [
                "arn:aws:execute-api:us-east-1:111122223333:api-id/prod/*"
            ]
        },
        {
            "Effect": "Deny",
            "Action": [
                "execute-api:Invoke"
            ],
            "Resource": [
                "arn:aws:execute-api:us-east-1:111122223333:api-id/prod/secret"
            ]
```

```
        },
        {
            "Effect": "Deny",
            "Action": [
                "execute-api:ManageConnections"
            ],
            "Resource": [
                "arn:aws:execute-api:us-east-1:111122223333:api-id/*"
            ]
        }
    ]
}
```

Contrôlez l'accès à l' WebSocket APIs aide des AWS Lambda autorisateurs REQUEST

Une fonction d'autorisation Lambda dans WebSocket APIs est similaire à celle de REST APIs, 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 `event.requestContext` sont différentes de celles de REST APIs.

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 REST dans : APIs the section called “Exemples supplémentaires de fonctions de mécanisme 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(null, generateDeny('me', event.methodArn));
    }
}

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

```
        statementOne.Resource = resource;
        policyDocument.Statement[0] = statementOne;
        authResponse.policyDocument = policyDocument;
    }
   // Optional output with custom properties of the String, Number or Boolean type.
   authResponse.context = {
       "stringKey": "stringval",
       "numberKey": 123,
       "booleanKey": true
    };
   return authResponse;
}

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

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

Python

```
# A simple REQUEST authorizer example to demonstrate how to use request
# parameters to allow or deny a request. In this example, a request is
# authorized if the client-supplied HeaderAuth1 header and QueryString1 query
 parameter
# in the request context match the specified values of
# of 'headerValue1' and 'queryValue1' respectively.

import json


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

    # Retrieve request parameters from the Lambda function input:
    headers = event['headers']
    queryStringParameters = event['queryStringParameters']
    stageVariables = event['stageVariables']
    requestContext = event['requestContext']

    # Parse the input for the parameter values
```

```
    tmp = event['methodArn'].split(':')
    apiGatewayArnTmp = tmp[5].split('/')
    awsAccountId = tmp[4]
    region = tmp[3]
    ApiId = apiGatewayArnTmp[0]
    stage = apiGatewayArnTmp[1]
    route = apiGatewayArnTmp[2]

    # Perform authorization to return the Allow policy for correct parameters
    # and the 'Unauthorized' error, otherwise.

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

    if (headers['HeaderAuth1'] ==
            "headerValue1" and queryStringParameters["QueryString1"] ==
 "queryValue1"):
        response = generateAllow('me', event['methodArn'])
        print('authorized')
        return json.loads(response)
    else:
        response = generateDeny('me', event['methodArn'])
        print('unauthorized')
        return json.loads(response)

    # Help function to generate IAM policy


def generatePolicy(principalId, effect, resource):
    authResponse = {}
    authResponse['principalId'] = principalId
    if (effect and resource):
        policyDocument = {}
        policyDocument['Version'] = '2012-10-17       '
        policyDocument['Statement'] = []
        statementOne = {}
        statementOne['Action'] = 'execute-api:Invoke'
        statementOne['Effect'] = effect
        statementOne['Resource'] = resource
        policyDocument['Statement'] = [statementOne]
        authResponse['policyDocument'] = policyDocument

    authResponse['context'] = {
```

```
        "stringKey": "stringval",
        "numberKey": 123,
        "booleanKey": True
    }

    authResponse_JSON = json.dumps(authResponse)

    return authResponse_JSON


def generateAllow(principalId, resource):
    return generatePolicy(principalId, 'Allow', resource)


def generateDeny(principalId, resource):
    return generatePolicy(principalId, 'Deny', resource)
```

Pour configurer la fonction Lambda précédente en tant que fonction d'`REQUEST`autorisation pour une WebSocket API, suivez la même procédure que pour REST. APIs

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
```

Intégrations pour WebSocket APIs in API Gateway

Après avoir configuré une route d'API, vous devez l'intégrer avec 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

- Configurer une demande d'intégration d' WebSocket API dans API Gateway
- Configurer une réponse d'intégration d' WebSocket API dans API Gateway

Configurer 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 avec le serveur principal.
- Spécifier le point de terminaison du backend à appeler. WebSocket APIs 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 avec un point de terminaison HTTP existant. Pour de plus amples informations, veuillez consulter Intégrations HTTP pour REST APIs 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 de plus amples informations, veuillez consulter Intégrations fictives pour REST APIs 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 Configurez une intégration privée.
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 Méthode HTTP, 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 dans 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. Utilisez la commande create-integration suivante pour créer l'intégration simulée.

```
aws apigatewayv2 --region us-east-1 create-integration --api-id aabbccddee --cli-
input-json file://integration-params.json
```

   Le résultat se présente comme suit :

```
{
    "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 en utilisant le AWS CLI.

1. Créez une fonction Lambda dans la console Lambda et attribuez-lui un rôle d'exécution Lambda de base.
2. Utilisez la commande create-integration suivante pour créer l'intégration.

```
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
```

Le résultat se présente comme suit :

```
{
    "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 WebSocket APIs

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 route `$connect` et de l'événement d'entrée de la route `$disconnect` 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
  }
```

Configurer une réponse d'intégration d' WebSocket API dans API Gateway

La section suivante fournit un bref aperçu des réponses d'intégration pour une WebSocket API et explique comment configurer une réponse d'intégration pour une WebSocket API.

Rubriques

- Présentation des réponses d’intégration
- Réponses d’intégration pour une communication bidirectionnelle
- Configuration d’une réponse d’intégration à l’aide de la console API Gateway
- Configurez une réponse d'intégration à l'aide du AWS CLI

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'un itinéraire est configuré pour une communication bidirectionnelle, une réponse d'intégration vous permet de configurer des transformations sur la charge utile du message renvoyé, de la même manière que les réponses d'intégration pour REST. APIs
- 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 effectué. 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 de statut 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

La create-integration-responsecommande suivante crée une réponse `$default` d'intégration :

```
aws apigatewayv2 create-integration-response \
    --api-id vaz7da96z6 \
    --integration-id a1b2c3 \
    --integration-response-key '$default'
```

Demande de validation pour WebSocket APIs in API Gateway

Vous pouvez configurer API Gateway afin qu'il exécute la validation d'une requête 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 requêtes 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 autres que de 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 requête. 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 requête. 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 "Modèles de données pour les API REST".

Configuration de la validation des requêtes à l'aide de la console API Gateway

L'exemple suivant vous montre comment configurer une validation des demandes sur une route.

Vous devez d'abord créer un modèle, puis une route. Ensuite, vous devez configurer la validation des demandes sur la route que vous venez de créer. Enfin, vous devez déployer et tester 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 de plus amples informations, veuillez consulter 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 Créer un modèle.
5. Pour 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 requête 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, sélectionnez 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 allez configurer la validation des demandes pour la route `sendMessage`.

Pour configurer une validation des demandes

1. Dans l'onglet Requête de routage, sous Paramètres de requête de routage, choisissez Modifier.
2. Pour Expression de sélection de modèle, saisissez **`${request.body.messageType}`**.

   API Gateway utilise la propriété `messageType` pour valider la requête entrante.

3. Choisissez Ajouter un modèle de requête.
4. Pour Clé de modèle, saisissez **`email`**.
5. Pour Modèle, choisissez emailModel.

   API Gateway valide les messages entrants avec la propriété `messageType` définie sur `email` pour ce modèle.

   > **Note**
   >
   > Si API Gateway ne parvient pas à faire correspondre l'expression de sélection du modèle à une clé de modèle, le modèle `$default` est sélectionné. S'il n'y a pas de modèle `$default`, la validation échoue. Pour la production APIs, nous vous recommandons de créer un `$default` modèle.

6. Sélectionnez Enregistrer les modifications.

Au cours de cette étape, vous allez déployer et tester 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 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 commande wscat 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 rejettera la requête.

Utilisez la commande suivante pour envoyer une demande valide à votre API.

```
{"action": "sendMessage", "messageType": "email", "address":
 "mary_major@example.com"}
```

Transformations de données pour WebSocket APIs dans API Gateway

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 serveur principal 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 les transformations de modèles de mappage pour mapper les données utiles d'une demande de méthode à la demande d'intégration correspondante, et d'une réponse d'intégration à la réponse de méthode correspondante. Vous allez créer un modèle de mappage et spécifier une expression de sélection de modèle afin de déterminer le modèle à utiliser pour 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 dans le langage VTL (Velocity Template Language) et appliqué à la charge utile à l'aide d'JSONPath expressions. Pour plus d'informations sur les modèles de mappage API Gateway, consultez Transformations de modèles de mappage pour les API REST dans API Gateway.

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 plus d'informations sur les modèles API Gateway, consultez Modèles de données pour les API REST.

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.

Configurer le mappage des données pour WebSocket APIs dans API Gateway

Le mappage de données vous permet de mapper des données d'une demande de routage vers une intégration backend.

> **Note**
>
> Le mappage de données pour WebSocket APIs n'est pas pris en charge dans le Console de gestion AWS. 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.

Le tableau suivant présente les expressions de mappage des données des demandes d'intégration. Dans le tableau, *`PARAM_NAME`* est le nom d'un paramètre de demande de routage du type de paramètre donné. Elle doit correspondre à l'expression régulière`'^[a-zA-Z0-9._$-]+$]'`. *`JSONPath_EXPRESSION`*est une JSONPath expression pour un champ JSON du corps de la requête.

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`* Il s'agit d'une chaîne littérale qui doit être placée entre guillemets simples.

Lorsque vous créez un mappage de données, AWS CLI assurez-vous de suivre le format correct pour utiliser des littéraux avec des chaînes dans le AWS CLI. Pour plus d'informations, consultez Using quotation marks and literals with strings in the AWS CLI dans le Guide d'utilisateur AWS Command Line Interface .

Exemples

Les AWS CLI exemples suivants configurent les mappages de données. Pour un exemple CloudFormation de modèle, voir `websocket-data-mapping.yaml`.

Mapper ConnectionID d'un client à un en-tête dans une demande d'intégration

La commande update-integration suivante mappe l'`connectionId` d'un client à un en-tête `connectionId` de la demande adressée 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

L'exemple suivant mappe un paramètre de chaîne de requête `authToken` à un en-tête `authToken` de la demande d'intégration.

1. Utilisez la commande update-route suivante pour ajouter le paramètre de chaîne de requête `authToken` aux paramètres de demande de la route.

   ```
   aws apigatewayv2 update-route --route-id 0abcdef \
       --api-id a1b2c3d4 \
       --request-parameters '{"route.request.querystring.authToken": {"Required":
    false}}'
   ```

2. Utilisez la commande update-integration suivante pour mapper le paramètre de chaîne de requête à l'en-tête `authToken` de la demande adressée à l'intégration backend.

   ```
   aws apigatewayv2 update-integration \
       --integration-id abc123 \
       --api-id a1b2c3d4 \
       --request-parameters
    'integration.request.header.authToken'='route.request.querystring.authToken'
   ```

3. (Facultatif) Si nécessaire, utilisez ce qui suit delete-route-request-parameterpour supprimer le paramètre de chaîne de `authToken` requête des 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'
   ```

WebSocket Référence du modèle de mappage d'API pour API Gateway

Cette section résume l'ensemble de variables actuellement prises en charge pour les WebSocket API dans API Gateway.

Paramètre	Description
`$context.connectionId`	Un identifiant unique pour la connexion que vous pouvez utiliser pour rappeler le client. L'ID de connexion est une chaîne de longueur variable dont la longueur et le format peuvent changer.
`$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 identifiant généré automatiquement pour l'appel d'API, qui contient des informations plus utiles pour debugging/troubleshooting.
`$context.apiId`	Identifiant qu'API Gateway attribue à votre API.
`$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 mécanisme d'autorisation personnalisée).

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"`.
`$context.error.messageString`	Valeur entre guillemets de `$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

Paramètre	Description
`$context.identity.caller`	Identifiant principal de l'appelant effectuant la demande.
`$context.identity.cognitoAuthenticationProvider`	Liste séparée par des virgules de tous les 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 groupe 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 plus d'informations sur les fournisseurs d'authentification Amazon Cognito disponibles, consultez Using Federated Identities dans le Guide 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`	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 le nom d'une 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.

Types de supports binaires pour les API WebSocket dans API Gateway

Les API WebSocket d'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 de 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 WebSocket APIs

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 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 l' 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-expression}`* est une JSONPath expression et *`{route-key}`* une clé de route pour l'API. Par exemple :

   ```
   {"action":"action1"}
   {"message":"test response body"}
   ```

   Pour plus d'informations sur JSONPath, voir JSONPathou 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 de plus amples informations, veuillez consulter 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,
  };
};
```

Si vous utilisez un nom de domaine personnalisé pour votre WebSocket API, supprimez la `stage` variable de votre code de fonction.

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'API WebSocket invocables 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 dans 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

- Création d'étapes pour WebSocket APIs in API Gateway
- Déploiement WebSocket APIs dans API Gateway
- Politique de sécurité pour WebSocket APIs in API Gateway
- Noms de domaine personnalisés pour les API WebSocket dans API Gateway

Création d'étapes pour WebSocket APIs in API Gateway

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 dans 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. La commande add permission suivante ajoute les autorisations requises :

```
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

Intégration HTTP URIs

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>}`

Déploiement WebSocket APIs 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 :
>
> - Routes
> - 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 REST APIs. 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 plus d'informations sur les étapes, consultez the section called “Configuration d'une étape”.

Rubriques

- Créez un déploiement d' WebSocket API 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' WebSocket API à l'aide du AWS CLI

La commande create-deployment suivante crée un déploiement :

```
aws apigatewayv2 --region us-east-1 create-deployment --api-id aabbccddee
```

Le résultat se présente comme suit :

```
{
    "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.

La commande create-stage suivante crée une étape et l'associe au déploiement :

```
aws apigatewayv2 --region us-east-1 create-stage --api-id aabbccddee --deployment-id
 fedcba --stage-name test
```

Le résultat se présente comme suit :

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

Vous pouvez également réutiliser une étape existante en mettant à jour la `deploymentId` propriété de l'étape avec le nouvel ID de déploiement (*`deployment-id`*). La commande update-stage suivante met à jour l'ID de déploiement de l'étape :

```
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 WebSocket APIs in API Gateway

API Gateway applique une politique de sécurité `TLS_1_2` pour tous les points de terminaison de WebSocket l'API.

Une politique de sécurité est une combinaison prédéfinie d'une version minimale du protocole TLS et de suites de chiffrement offerte 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 politique 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 WebSocket APIs

Le tableau suivant décrit les protocoles TLS pris en charge pour WebSocket APIs.

Protocoles TLS	Politique de sécurité TLS_1_2
TLSv13.	Oui

Protocoles TLS	Politique de sécurité TLS_1_2
TLSv12.	Oui

Le tableau suivant décrit les chiffrements TLS disponibles pour la politique de sécurité TLS 1_2 pour. WebSocket APIs

Chiffrements TLS	Politique de sécurité TLS_1_2
TLS_AES_128_GCM_ SHA256	Oui
TLS_AES_256_GCM_ SHA384	Oui
TLS_ _ _ CHACHA20 POLY1305 SHA256	Oui
ECDHE-ECDSA- -GCM- AES128 SHA256	Oui
ECDHE-RSA- -GCM- AES128 SHA256	Oui
ECDHE-ECDSA- - AES128 SHA256	Oui
ECDHE-RSA- - AES128 SHA256	Oui
ECDHE-ECDSA- -GCM- AES256 SHA384	Oui
ECDHE-RSA- -GCM- AES256 SHA384	Oui
ECDHE-ECDSA- - AES256 SHA384	Oui
ECDHE-RSA- - AES256 SHA384	Oui
AES128-GCM- SHA256	Oui
AES128-SHA256	Oui
AES256-GCM- SHA384	Oui
AES256-SHA256	Oui

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 de chiffrement, consultez the section called “Noms de chiffrement OpenSSL et RFC”.

Informations sur REST APIs et HTTP APIs

Pour plus d'informations sur REST APIs et HTTP APIs, consultez the section called “Sélection d’une politique de sécurité” etthe section called “Politique de sécurité pour le protocole HTTP APIs dans API Gateway”.

Noms de domaine personnalisés pour les API WebSocket dans API Gateway

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`* est la région AWS et *`stage`* est spécifié par vous lors du déploiement de l’API.

La partie nom d’hôte de l’URL, *`api-id`*`.execute-api.`*`region`*`.amazonaws.com`, fait référence à un point de terminaison de l’API. Le nom par défaut du point de terminaison de l’API est généré de manière aléatoire, difficile à mémoriser et peu 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
```

Considérations

Les considérations suivantes peuvent avoir une incidence sur votre utilisation d’un nom de domaine personnalisé.

- Si vous mappez un nom de domaine personnalisé à une API WebSocket, vous ne pouvez pas le mapper à une API REST ou à une API HTTP.
- Seuls les noms de domaine personnalisés régionaux sont pris en charge.
- Pour la version TLS minimale, seule la version TLS 1.2 est prise en charge.
- Vous devez créer ou mettre à jour l'enregistrement de ressource de votre fournisseur DNS pour le mapper au point de terminaison de votre API. Sans ce mappage, les demandes d'API destinées au nom de domaine personnalisé ne peuvent pas atteindre API Gateway.
- Vous pouvez prendre en charge un nombre presque infini de noms de domaine sans dépasser le quota par défaut en utilisant un certificat générique. Pour de plus amples informations, consultez the section called "Noms de domaine personnalisés génériques".

Prérequis

Les conditions suivantes sont requises pour créer un nom de domaine personnalisé.

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. Vous pouvez enregistrer votre nom de domaine Internet avec Amazon Route 53 ou utiliser un bureau d'enregistrement de domaine tiers de votre choix. Votre nom de domaine personnalisé peut être le nom d'un sous-domaine ou le domaine racine (également nommé « zone apex ») d'un domaine Internet enregistré.

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.

Certificats pour les noms de domaine personnalisés

Avant de configurer un nom de domaine personnalisé pour une API, vous devez avoir un certificat SSL/TLS prêt dans ACM. Si ACM n'est pas disponible dans la région AWS où vous créez votre nom de domaine personnalisé, 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. 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 utiliser un certificat géré par AWS pour un nom de domaine, indiquez simplement son ARN.

Si votre application utilise l'épinglage de certificat, parfois appelé épinglage SSL, pour épingler un certificat ACM, l'application ne pourra peut-être pas se connecter à votre domaine une fois qu'AWS aura renouvelé le certificat. Pour plus d'informations, consultez la section Problèmes d'épinglage de certificat dans le Guide de l'utilisateur AWS Certificate Manager.

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 compte AWS, vous pouvez configurer `*.example.com` et `a.example.com` pour qu'ils se comportent 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, consultez Variables pour les transformations de données pour API Gateway.

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 générique si un autre compte AWS a créé un nom de domaine personnalisé en conflit avec ce nom. 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 AWSCentre de support pour demander une exception.

Étapes suivantes pour les noms de domaine personnalisés

Pour configurer un nom de domaine personnalisé pour une API HTTP, utilisez la documentation de la section API REST du Guide du développeur API Gateway.

D'abord, spécifiez un certificat pour votre nom de domaine personnalisé. Pour de plus amples informations, consultez the section called "Préparez les certificats dans AWS Certificate Manager". Ensuite, créez un nom de domaine personnalisé régional. Pour de plus amples informations, consultez the section called "Configuration d'un nom de domaine personnalisé régional".

Mappage des étapes d'API à un nom de domaine personnalisé pour les API WebSocket

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 sur le même compte AWS.
- 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 mapper les API WebSocket au même nom de domaine personnalisé qu'une API HTTP ou une API REST.
- Si vous créez un mappage d'API à plusieurs niveaux, API Gateway convertit tous les noms d'en-tête en minuscules.

Création d'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".

Console de gestion AWS

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 Save (Enregistrer).

AWS CLI

La commande create-api-mapping 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
```

CloudFormation

L'exemple CloudFormation suivant crée un mappage d'API.

```
MyApiMapping:
  Type: 'AWS::ApiGatewayV2::ApiMapping'
  Properties:
    DomainName: api.example.com
    ApiMappingKey: 'v1'
    ApiId: !Ref MyApi
    Stage: !Ref MyStage
```

Types d'adresses IP des noms de domaine personnalisés pour les API WebSocket

Lorsque vous créez un domaine personnalisé, vous devez spécifier le type d'adresse IP pouvant invoquer votre domaine. Vous avez le choix entre IPv4 (pour autoriser les adresses IPv4 à invoquer votre domaine) et Dualstack (pour autoriser les adresses IPv4 et IPv6 à invoquer votre domaine). Nous vous recommandons de définir le type d'adresse IP sur Dualstack pour éviter l'épuisement de l'espace IP ou renforcer votre niveau de sécurité. Pour plus d'informations sur les avantages d'un type d'adresse IP à double pile, consultez IPv6 sur AWS.

Considérations relatives aux types d'adresses IP

Les considérations suivantes peuvent avoir une incidence sur votre utilisation des types d'adresses IP.

- Le type d'adresse IP par défaut des noms de domaine personnalisés API Gateway est IPv4.
- Il n'est pas nécessaire que votre nom de domaine personnalisé possède le même type d'adresse IP pour toutes les API qui y sont mappées. La désactivation du point de terminaison de votre API par défaut peut avoir une incidence sur la manière dont les appelants peuvent invoquer votre API.

Modification du type d'adresse IP d'un nom de domaine personnalisé

Vous pouvez modifier le type d'adresse IP en mettant à jour la configuration de point de terminaison du nom de domaine. Vous pouvez mettre à jour la configuration du point de terminaison à l'aide de la Console de gestion AWS, de l'AWS CLI, d'CloudFormation ou d'un kit SDK AWS.

Console de gestion AWS

Pour modifier le type d'adresse IP d'un nom de domaine personnalisé

1. Connectez-vous à la console API Gateway à l'adresse : https://console.aws.amazon.com/apigateway.
2. Choisissez un nom de domaine personnalisé public.
3. Sélectionnez Configuration du point de terminaison.
4. Pour Type d'adresse IP, choisissez IPv4 ou Dualstack.
5. Choisissez Save (Enregistrer).

AWS CLI

La commande update-domain-name suivante met à jour une API pour qu'elle possède le type d'adresse IP Dualstack :

```
aws apigatewayv2 update-domain-name \
   --domain-name dualstack.example.com \
   --domain-name-configurations CertificateArn=arn:aws:acm:us-
east-1:111122223333:certificate/abcd1234-5678-abc,IpAddressType=dualstack
```

Le résultat se présente comme suit :

```
{
    "ApiMappingSelectionExpression": "$request.basepath",
    "DomainName": "dualstack.example.com",
    "DomainNameConfigurations": [
        {
            "ApiGatewayDomainName": "d-abcd1234.execute-api.us-
east-1.amazonaws.com",
            "CertificateArn": "arn:aws:acm:us-east-1:111122223333:certificate/
abcd1234-5678-abc",
            "DomainNameStatus": "AVAILABLE",
            "EndpointType": "REGIONAL",
            "HostedZoneId": "Z3LQWSYCGH4ADY",
            "SecurityPolicy": "TLS_1_2",
            "IpAddressType": "dualstack"
        }
    ],
    "Tags": {}
```

```
}
```

Désactiver le point de terminaison par défaut pour WebSocket APIs

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`. Lorsque vous désactivez le point de terminaison par défaut, toutes les étapes d'une API sont affectées.

La procédure suivante montre comment désactiver le point de terminaison par défaut pour une WebSocket API.

Console de gestion AWS

1. Connectez-vous à la console API Gateway à l'adresse https://console.aws.amazon.com/apigateway.
2. Choisissez une WebSocket API.
3. Choisissez Paramètres de l'API.
4. Dans l'onglet Détails de l'API, choisissez Modifier.
5. Pour Point de terminaison par défaut, sélectionnez Inactif.
6. Sélectionnez Enregistrer les modifications.
7. Dans le volet de navigation principal, sélectionnez Routes.
8. Choisissez Déployer, puis redéployez votre API ou créez une étape pour que la modification prenne effet.

AWS CLI

La commande update-api suivante désactive le point de terminaison par défaut d'une API : WebSocket

```
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
```

Protégez votre WebSocket APIs API Gateway

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 ou la définition de limitations. Pour plus d'informations sur la génération de certificats SSL, consultez the section called " Gateway-generated Certificats API". Le reste de cette section porte sur la définition de limitations.

Vous pouvez configurer la régulation pour éviter qu'ils ne soient submergés par un trop grand nombre de demandes. APIs 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. Plus précisément, API Gateway examine le taux et le nombre de demandes soumises par rapport APIs à l'ensemble 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 cibles pour les différentes étapes ou itinéraires d'API afin d'améliorer les APIs performances globales de l'ensemble de votre compte.

Limitation au niveau du compte par région

Par défaut, API Gateway limite le nombre de demandes stables par seconde (RPS) sur l'ensemble APIs d'un AWS compte, par région. Cela limite également la rafale (c'est-à-dire la taille maximale du bucket) sur l'ensemble APIs du AWS compte, 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 tous les APIs membres d'un compte dans une région spécifiée. La limite de débit au niveau du compte peut être augmentée sur demande. Des limites plus élevées sont possibles avec APIs des délais d'attente plus courts et des charges utiles plus faibles. Pour demander une augmentation des limitations au niveau du compte par Région, contactez le Centre de support AWS. Pour de plus amples informations, veuillez consulter 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 update-stage suivante configure une limitation personnalisée pour l'étape et la route spécifiées d'une API :

```
aws apigatewayv2 update-stage \
    --api-id a1b2c3d4 \
    --stage-name dev \
    --route-settings '{"messages":
{"ThrottlingBurstLimit":100,"ThrottlingRateLimit":2000}}'
```

Surveillance des API WebSocket dans API Gateway

Vous pouvez utiliser les métriques CloudWatch et CloudWatch Logs pour surveiller les API WebSocket. 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

- Surveillez WebSocket l'exécution des API à l'aide de CloudWatch métriques
- Configuration de la journalisation WebSocket APIs dans API Gateway

Surveillez WebSocket l'exécution des API à l'aide de CloudWatch métriques

Vous pouvez utiliser les CloudWatch métriques Amazon pour effectuer un suivi WebSocket APIs. La configuration est similaire à celle utilisée pour REST APIs. Pour de plus amples informations, veuillez consulter Surveillez l'exécution de l'API REST avec CloudWatch les métriques Amazon.

Les mesures suivantes sont prises en charge pour WebSocket APIs :

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.

Métrique	Description
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 simulées.

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 API Gateway pour trouver une méthode d’API portant l’ID d’API, l’ID d’étape, le chemin de ressource et l’ID de routage spécifiés. API Gateway n'enverra pas ces métriques à moins que vous n'ayez explicite

Dimension	Description
	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 `detailedMetricsEnabled` propriété sur`true`. Vous pouvez également appeler la AWS CLI commande update-stage pour mettre à jour la `DetailedMetricsEnabled` 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.

Configuration de la journalisation WebSocket APIs dans API Gateway

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.

Pour renforcer votre niveau de sécurité, nous vous recommandons d'utiliser la journalisation des exécutions au niveau `ERROR` ou `INFO`. Vous devrez peut-être le faire pour vous conformer aux différents cadres de conformité. Pour plus d'informations, consultez Amazon API Gateway controls dans le Guide de l'utilisateur AWS Security Hub .

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 "Configurer la journalisation des accès 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 de statut 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 de l'autorisation Lambda en ms.
`$context.authorizer.integrationStatus`	Code de statut renvoyé par un mécanisme d'autorisation Lambda.
`$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 de statut 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'autoris

Paramètre	Description
	ation API Gateway Lambda. (Un mécanisme d'autorisation Lambda était auparavant connu sous le nom de mécanisme d'autorisation personnalisé.)
`$context.authorizer.` *`property`*	Valeur obtenue à l'aide de stringify de la paire clé-valeur spécifiée du mappage `context` renvoyé par une fonction du mécanisme d'autorisation Lambda API Gateway. Par exemple, si le mécanisme d'autorisation retourne le mappage `context` suivant : `"context" : { "key": "value", "numKey": 1, "boolKey": true }` l'appel de `$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.
`$context.authenticate.status`	Code de statut renvoyé à la suite d'une tentative d'authentification.
`$context.connectedAt`	Temps de connexion au format Epoch.

Paramètre	Description
`$context.connectionId`	ID unique pour la connexion qui peut être utilisé pour effectuer un rappel au client.
`$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
`$context.identity.cognitoAuthenticationProvider`	Liste séparée par des virgules de tous les fournisseurs d'authentification Amazon Cognito utilisés par l'appelant à l'origine de la demande. Disponible uniquement si la demande a été signée avec les informations d'identification Amazon Cognito. Par exemple, pour une identité provenant d'un groupe d'utilisateurs Amazon Cognito, `cognito-idp.`*`region`*`.amazonaws.com/`*`user_pool_id`* `,cognito-idp.`*`region`*`.amazonaws.com/`*`user_pool_id`* `:CognitoSignIn:`*`token subject claim`* Pour plus d'informations sur les fournisseurs d'authentification Amazon Cognito disponibles, consultez Using Federated Identities dans le Guide du développeur Amazon Cognito.
`$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
`$context.integration.status`	Code de statut renvoyé à partir d'une intégration. Pour les intégrations de proxy Lambda, code de statut que votre code de fonction Lambda renvoie. Équivalent à `$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 Amazon Resource Names (ARNs) pour les ressources API Gateway. Pour en savoir plus sur l'utilisation ARNs dans Gestion des identités et des accès AWS les politiques, consultez Fonctionnement d'Amazon API Gateway avec IAM etContrôle de l’accès à une API REST 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`
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`
Cors	`arn:partition :apigatew ay: region::/apis/api-id/cors`
Déploiement	`arn:partition :apigatew ay: region::/apis/api-id/deployme nts/ id`

Ressource	ARN
Déploiements	arn:*partition* :apigateway: *region*::/apis/*api-id*/deployments
ExportedAPI	arn:*partition* :apigateway: *region*::/apis/*api-id*/exports/ *specification*
Integration	arn:*partition* :apigateway: *region*::/apis/*api-id*/integrations/ *integration-id*
Intégrations	arn:*partition* :apigateway: *region*::/apis/*api-id*/integrations
IntegrationResponse	arn:*partition* :apigateway: *region*::/apis/*api-id*/integrationresponses/ *integration-response*
IntegrationResponses	arn:*partition* :apigateway: *region*::/apis/*api-id*/integrationresponses
Modèle	arn:*partition* :apigateway: *region*::/apis/*api-id*/models/*id*
Modèles	arn:*partition* :apigateway: *region*::/apis/*api-id*/models
ModelTemplate	arn:*partition* :apigateway: *region*::/apis/*api-id*/models/*id*/template
Acheminement	arn:*partition* :apigateway: *region*::/apis/*api-id*/routes/*id*

Ressource	ARN
Routes	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*
Étape	arn:*partition* :apigatew ay: *region*::/apis/*api-id*/ stages/*stage-name*
Stages	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
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

Ressource	ARN
DocumentationPart	arn:*partition* :apigatew ay: *region*::/restapis/ *api-id*/ documentation/parts/ *id*
DocumentationParts	arn:*partition* :apigatew ay: *region*::/restapis/ *api-id*/ documentation/parts
DocumentationVersion	arn:*partition* :apigatew ay: *region*::/restapis/ *api-id*/ documentation/versions/ *version*
DocumentationVersions	arn:*partition* :apigatew ay: *region*::/restapis/ *api-id*/ documentation/versions
GatewayResponse	arn:*partition* :apigatew ay: *region*::/restapis/ *api-id*/ gatewayresponses/ *response-type*
GatewayResponses	arn:*partition* :apigatew ay: *region*::/restapis/ *api-id*/ gatewayresponses
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*

Ressource	ARN
Method	arn:*partition* :apigatew ay: *region*::/restapis/ *api-id*/ resources/ *resource-id* /methods/ *http-method*
MethodResponse	arn:*partition* :apigatew ay: *region*::/restapis/ *api-id*/ resources/ *resource-id* /methods/ *http-method* /responses/ *status-co de*
Modèle	arn:*partition* :apigatew ay: *region*::/restapis/ *api-id*/ models/*model-name*
Modèles	arn:*partition* :apigatew ay: *region*::/restapis/ *api-id*/ models
RequestValidator	arn:*partition* :apigatew ay: *region*::/restapis/ *api-id*/ requestvalidators/ *id*
RequestValidators	arn:*partition* :apigatew ay: *region*::/restapis/ *api-id*/ requestvalidators
Ressource	arn:*partition* :apigatew ay: *region*::/restapis/ *api-id*/ resources/ *id*
Ressources	arn:*partition* :apigatew ay: *region*::/restapis/ *api-id*/ resources

Ressource	ARN
RestApi	arn:*partition* :apigatew ay: *region*::/restapis/ *api-id*
RestApis	arn:*partition* :apigatew ay: *region*::/restapis
Étape	arn:*partition* :apigatew ay: *region*::/restapis/ *api-id*/ stages/*stage-name*
Stages	arn:*partition* :apigatew ay: *region*::/restapis/ *api-id*/ stages
Étiquettes	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

Ressource	ARN
VpcLink	arn:*partition* :apigatew ay: *region*::/vpclinks/ *vpclink-id*
VpcLinks	arn:*partition* :apigatew ay: *region*::/vpclinks

Ressources du nom de domaine

Ressource	ARN
ApiMapping	arn:*partition* :apigatew ay: *region*::/domainnames/ *domain-na me* /apimappings/ *id*
ApiMappings	arn:*partition* :apigatew ay: *region*::/domainnames/ *domain-na me* /apimappings
BasePathMapping	arn:*partition* :apigatew ay: *region*::/domainnames/ *domain-na me* /basepathmappings/ *basepath*
BasePathMappings	arn:*partition* :apigatew ay: *region*::/domainnames/ *domain-na me* /basepathmappings
DomainName	arn:*partition* :apigatew ay: *region*::/domainnames/ *domain-na me*
DomainNameAccessAssociation	arn:*partition* :apigatew ay: *region*:*account-id* :/domainn ameaccessassociations/domai nname/ *domain-name* /vpcesour ce/ *vpce-source-id*

Ressource	ARN
DomainNameAccessAssociations	`arn:partition :apigatew ay: region:account-id :/domainn ameaccessassociations`
DomainNames	`arn:partition :apigatew ay: region::/domainnames`
RoutingRule	`arn:partition :apigatew ay: region:account-id :/domainn ames/ domain-name /routingr ules/ routing-rule-id`

execute-api(HTTP APIs WebSocket APIs et REST APIs)

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 la `$default` route pour HTTP APIs est`arn:partition:execute-api:region:account-id:api-id/*/$default`.

** Cet ARN s'applique uniquement lors de la définition de la condition `SourceArn` dans la politique de ressources pour une fonction de mécanisme d'autorisation Lambda. Pour obtenir un exemple, consultez the section called "Création d'un mécanisme d'autorisation Lambda".

Extensions OpenAPI pour API Gateway

Les extensions API Gateway prennent en charge l'autorisation AWS spécifique et les intégrations d' Gateway-specific API 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 plus d'informations sur la procédure d'exportation d'une API, consultez Exportation d'une API REST à partir d'API Gateway et Exporter le protocole HTTP APIs depuis API Gateway.

Rubriques

- Objet x-amazon-apigateway-any-method
- Objet x-amazon-apigateway-cors
- Propriété x-amazon-apigateway-api-key-source
- Objet x-amazon-apigateway-auth
- Objet x-amazon-apigateway-authorizer
- Propriété x-amazon-apigateway-authtype
- Propriété x-amazon-apigateway-binary-media-types
- Objet x-amazon-apigateway-documentation
- mode d'accès à x-amazon-apigateway-endpoint-endpoint
- Objet x-amazon-apigateway-endpoint-configuration
- Objet x-amazon-apigateway-gateway-responses
- Objet x-amazon-apigateway-gateway-responses.gatewayResponse
- Objet x-amazon-apigateway-gateway-responses.responseParameters
- Objet x-amazon-apigateway-gateway-responses.responseTemplates
- x-amazon-apigateway-importexport-version
- Objet x-amazon-apigateway-integration
- Objet x-amazon-apigateway-integrations

- Objet x-amazon-apigateway-integration.requestTemplates
- Objet x-amazon-apigateway-integration.requestParameters
- Objet x-amazon-apigateway-integration.responses
- Objet x-amazon-apigateway-integration.response
- Objet x-amazon-apigateway-integration.responseTemplates
- Objet x-amazon-apigateway-integration.responseParameters
- Objet x-amazon-apigateway-integration.tlsConfig
- x-amazon-apigateway-minimum-compression-size
- x-amazon-apigateway-policy
- Propriété x-amazon-apigateway-request-validator
- Objet x-amazon-apigateway-request-validators
- Objet x-amazon-apigateway-request-validators.requestValidator
- Politique de sécurité de x-amazon-apigateway-
- Propriété x-amazon-apigateway-tag-value

Objet x-amazon-apigateway-any-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.

Nom de la propriété	Type	Description
`isDefaultRoute`	`Boolean`	Spécifie si une route est la route `$default`. Prise en charge uniquement pour les API HTTP. Pour en savoir plus, consultez la section Création de routes pour HTTP APIs dans API Gateway.

Nom de la propriété	Type	Description
`x-amazon-apigateway-integration`	Objet x-amazon-apigateway-integration	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`.

Exemples x-amazon-apigateway-any-method

L'exemple suivant intègre la méthode `ANY` sur une ressource 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 avec 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
      }
   }
}
```

Objet x-amazon-apigateway-cors

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 Configurer CORS pour HTTP APIs dans API Gateway.

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.

Exemple x-amazon-apigateway-cors

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"
    ]
}
```

Propriété x-amazon-apigateway-api-key-source

Spécifiez la source qui recevra une clé API pour limiter les méthodes d'API qui requièrent une clé. Cette API-level propriété est un `String` type. Pour plus d'informations sur la configuration d'une méthode pour exiger une clé d'API, consultez the section called "Configuration d'une méthode pour utiliser des clés d'API avec une définition OpenAPI".

Spécifiez la source de la clé d'API pour les demandes. Les valeurs valides sont :

- `HEADER` pour recevoir la clé d'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 mécanisme d'autorisation personnalisée).

Exemple x-amazon-apigateway-api-key-source

L'exemple suivant définit l'en-tête `X-API-Key` en tant que source de clé d'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",
   .
   .
   .
}
```

Objet x-amazon-apigateway-auth

Définit un type d'autorisation à appliquer pour l'autorisation des appels de méthode dans API Gateway.

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.

Exemple x-amazon-apigateway-auth

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"
```

```
        }
      }
    }
  }
}
```

Objet x-amazon-apigateway-authorizer

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 d'OpenAPI 2 et au schéma de sécurité d'OpenAPI 3.

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. Pour les API HTTP, spécifiez `request` pour un mécanisme

Nom de la propriété	Type	Description
		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 politique 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 plus d'informations, consultez 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]+ ». Prise en charge uniquement pour les mécanismes d'autorisation `TOKEN` pour les 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` .

Exemples x-amazon-apigateway-authorizer pour les 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" : "custom", // Specifies the authorization
mechanism for the client.
     "x-amazon-apigateway-authorizer" : {       // An API Gateway Lambda authorizer
definition
       "type" : "token",                         // Required property and the value
must "token"
       "authorizerUri" : "arn:aws:apigateway:us-east-1:lambda:path/2015-03-31/
functions/arn:aws:lambda:us-east-1:account-id:function:function-name/invocations",
       "authorizerCredentials" : "arn:aws:iam::account-id:role",
       "identityValidationExpression" : "^x-[a-z]+",
       "authorizerResultTtlInSeconds" : 60
     }
   }
 }
```

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"
        }
      }
    }
```

Exemples x-amazon-apigateway-authorizer pour les 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
    }
  }
}
```

Propriété x-amazon-apigateway-authtype

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.

Exemple de propriété x-amazon-apigateway-authtype

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

Propriété x-amazon-apigateway-binary-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.

Exemple x-amazon-apigateway-binary-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" ]
```

Objet x-amazon-apigateway-documentation

Définit les parties de la documentation à importer dans API Gateway. Cet objet est un objet JSON qui contient un tableau d'instances `DocumentationPart`.

Nom de la propriété	Type	Description
`documentationParts`	`Array`	Un tableau des instances `DocumentationPart` exportées ou importées.

Nom de la propriété	Type	Description
`version`	`String`	L'identifiant de version de l'instantané des parties de la documentation exportées.

Exemple x-amazon-apigateway-documentation

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
      }
    ]
  }
}
```

mode d'accès à x-amazon-apigateway-endpoint-endpoint

Spécifie le mode d'accès au point de terminaison pour une API REST ou un nom de domaine personnalisé. Pour en savoir plus sur le mode d'accès aux terminaux, consultezMode d'accès au point de terminaison.

Exemple de mode d'accès x-amazon-apigateway-endpoint-point

L'exemple suivant spécifie le mode d'accès au terminal sur strict.

```
"x-amazon-apigateway-endpoint-access-mode": "STRICT"
```

Objet x-amazon-apigateway-endpoint-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.

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

Nom de la propriété	Type	Description
		pris en charge pour le type de point de terminaison `PRIVATE` des API REST.
`ipAddressType`	`string`	Types d'adresses IP pouvant invoquer une API HTTP. Utilisez `ipv4` pour autoriser les types d'adresses IPv4 à invoquer une API HTTP. Utilisez `dualstack` pour autoriser les types d'adresses IPv4 et IPv6 à invoquer une API HTTP. Seules les API HTTP sont prises en charge.

Exemples x-amazon-apigateway-endpoint-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
}
```

L'exemple suivant définit le type d'adresse IP sur « double pile » pour une API HTTP.

```
"x-amazon-apigateway-endpoint-configuration": {
    "ipAddressType": "dualstack"
}
```

Objet x-amazon-apigateway-gateway-responses

Définit les réponses de passerelle pour une API sous la forme d'une carte GatewayResponse chaîne-à-carte de paires clé-valeur. L'extension s'applique à la structure OpenAPI au niveau racine.

Nom de la propriété	Type	Description
responseType	x-amazon-apigateway-gateway-responses.gatewayResponse	A `GatewayResponse` pour la valeur spécifiée*responseType* .

Exemple x-amazon-apigateway-gateway-responses

L'exemple d'extension API Gateway vers OpenAPI suivant définit une GatewayResponses carte contenant deux GatewayResponse instances, 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 }"
      }
    }
  }
}
```

Objet x-amazon-apigateway-gateway-responses.gatewayResponse

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.

Nom de la propriété	Type	Description
responseParameters	x-amazon-apigateway-gateway-responses.responseParameters	Spécifie les GatewayResponse paramètres, 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.responseTemplates	Spécifie les modèles de mappage de la réponse de passerelle. Les modèles ne sont pas traités par le moteur VTL.
statusCode	`string`	Code de statut HTTP pour la réponse de passerelle.

Exemple x-amazon-apigateway-gateway-responses.gatewayResponse

L'exemple suivant de l'extension API Gateway pour OpenAPI définit un GatewayResponse pour personnaliser la `INVALID_API_KEY` réponse afin de renvoyer le code d'état de`456`, la valeur d'`api-key`en-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\" }"
      }
    }
```

Objet x-amazon-apigateway-gateway-responses.responseParameters

Définit un mappage chaîne à chaîne de paires clé-valeur pour générer des paramètres de réponse de passerelle à partir des paramètres de demande entrante ou à l'aide de chaînes littérales. Prise en charge uniquement pour les API REST.

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 de paramètres pour les API REST dans API Gateway.

Exemple x-amazon-apigateway-gateway-responses.responseParameters

L'exemple d'extensions OpenAPI suivant montre une expression de mappage de paramètres de GatewayResponse réponse pour activer la prise en charge CORS des ressources sur les `*.example.domain` domaines.

```
    "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
    }
```

Objet x-amazon-apigateway-gateway-responses.responseTemplates

Définit les modèles de GatewayResponse mappage, sous la forme d'une carte chaîne à chaîne 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 stringifié pour des substitutions de variables simples. Un modèle de mappage `GatewayResponse` n'est pas traité par le moteur VTL (Velocity Template Language).

Nom de la propriété	Type	Description
content-type	`string`	Corps de modèle de mappage `GatewayResponse` prenant en charge uniquement les substitutions de variables simples pour personnaliser un corps de réponse de passerelle.

Exemple x-amazon-apigateway-gateway-responses.responseTemplates

L'exemple d'extensions OpenAPI suivant montre un modèle de GatewayResponse mappage 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 GatewayResponse mappage 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 plus d'informations, consultez exportVersion dans la Référence d'API API Gateway Version 2.

x-amazon-apigateway-importexport-version example

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": { ...
```

Objet x-amazon-apigateway-integration

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.

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`	Un groupe de API-speci fic balises contenant des paramètres mis en cache associés.
`connectionId`	`string`	L'ID de a VpcLink pour l'intégration privée.

Nom de la propriété	Type	Description
`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 ajoutées manuellement pour permettre à l'API d'accéder à la ressource. Pour plus d'informations, consultez Accord d'autorisations à l'aide d'une politique 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`.
`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, consultez the section called "AWSRéférence des intégrations de service ".

Nom de la propriété	Type	Description
`integrationTarget`	`string`	L'écouteur ALB ou NLB auquel envoyer la demande. Pris en charge uniquement pour les intégrations privées qui utilisent des liens VPC V2. Pour plus d'informations, consultez the section called "Liaisons VPC V2".
`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 plus d'informations, consultez Integration.passthroughBehavior.

Nom de la propriété	Type	Description
`payloadFormatVersion`	`string`	Spécifie le format de la charge utile envoyée à une intégrati on. 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 intégrati ons” et the section called “AWSRéférence des intégrati ons de service ”.

Nom de la propriété	Type	Description
`requestParameters`	Objet x-amazon-apigateway-integration.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`	Objet x-amazon-apigateway-integration.requestTemplates	Modèles de mappage pour la charge utile des demandes des types MIME spécifiés.

Nom de la propriété	Type	Description
`responses`	Objet x-amazon-apigateway-integration.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.
`responseTransferMode`	`string`	Mode de transfert de réponse de l'intégration. Utilisez cette `BUFFERED` option pour qu'API Gateway attende de recevoir la réponse complète avant de commencer la transmiss ion. Permet `STREAM` à API Gateway de renvoyer des réponses partielles au client dès qu'elles sont disponibles. Pour en savoir plus, consultez the section called "Streamin g de la réponse d'intégra tion pour vos intégrations de proxy".
`timeoutInMillis`	`integer`	Délais d'expiration de l'intégra tion 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 plus d'informations sur les types d'intégrations, consultez 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`.

Exemples d'objet x-amazon-apigateway-integration

Pour les API HTTP, vous pouvez définir des intégrations dans la section composants de votre définition OpenAPI. Pour en savoir plus, consultez la section Objet x-amazon-apigateway-integrations.

```
"x-amazon-apigateway-integration": {
    "$ref": "#/components/x-amazon-apigateway-integrations/integration1"
}
```

L'exemple suivant crée une intégration avec une fonction Lambda. À des fins de démonstration, les exemples de modèles `responseTemplates` de mappage présentés dans `requestTemplates` et parmi les exemples ci-dessous sont supposés s'appliquer à la JSON-formatted charge utile 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 de`<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.

Objet x-amazon-apigateway-integrations

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.

Nom de la propriété	Type	Description
integration	Objet x-amazon-apigateway-integration	Collection d'objets d'intégration.

Exemple x-amazon-apigateway-integrations

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"
            }
        }
    }
}
```

Objet x-amazon-apigateway-integration.requestTemplates

Spécifie les modèles de mappage pour la charge utile des demandes des types MIME spécifiés.

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 the section called “Transformations de modèles de mappage”.

Exemple d’objet x-amazon-apigateway-integration.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> "
}
```

Objet x-amazon-apigateway-integration.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.

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 the section called “AWS intégrations de services”.

Exemple X-Amazon-APIGateway-Integration.RequestParameters

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 ou 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"
},
```

Objet x-amazon-apigateway-integration.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.

Nom de la propriété	Type	Description
`Response status pattern`	Objet x-amazon-apigateway-integration.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 de statut 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 à une défaillance lorsque l'exécution de la fonction Lambda génère une exception .

Nom de la propriété	Type	Description
		Note Le nom de la *Response status pattern* propriété fait référence à un code d'état de réponse ou à une expressio n régulière décrivant un groupe de codes d'état de réponse. Il ne correspond à aucun identifiant de IntegrationResponse ressource dans l'API REST API Gateway.

Exemple x-amazon-apigateway-integration.responses

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"
        }
    }
}
```

Objet x-amazon-apigateway-integration.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.

Nom de la propriété	Type	Description
`statusCode`	`string`	Code de statut 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`	Objet x-amazon-apigateway-integration.responseTemplates	Spécifie les modèles de mappage spécifiques au type MIME pour la charge utile de la réponse.
`responseParameters`	Objet x-amazon-apigateway-integration.responseParameters	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.

Exemple x-amazon-apigateway-integration.response

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"
    }
}
```

Objet x-amazon-apigateway-integration.responseTemplates

Spécifie les modèles de mappage pour la charge utile des réponses des types MIME spécifiés.

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 the section called "Transformations de modèles de mappage". Un exemple de cela *MIME type* est`application/json` .

Exemple d'objet x-amazon-apigateway-integration.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> "
}
```

Objet x-amazon-apigateway-integration.responseParameters

Spécifie les mappages des paramètres de réponse d'intégration aux paramètres de réponse de méthode. Vous pouvez mapper `header`, `body`, ou des valeurs statiques au type `header` de la réponse de la méthode. Prise en charge uniquement pour les API REST.

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.

Exemple X-Amazon-APIGateway-Integration.ResponseParameters

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"
}
```

Objet x-amazon-apigateway-integration.tlsConfig

Spécifie la configuration TLS pour une intégration.

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, qui comprend la vérification de la date d'expiration du certificat, du nom d'hôte, de la présence d'une autorité de certification racine et de l'intégrité de la chaîne de certificats. La validation de la chaîne de certificats inclut l'application des contraintes de nom X509v3. Si vous rencontrez des erreurs de validation lorsque cette `insecureSkipVerification` option est activée, vérifiez que vos certificats CA n'incluent pas d'extensions Name Constraints, ou réémettez-les sans cette extension. Le certificat racine

Nom de la propriété	Type	Description
		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`. 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 activez `insecureSkipVerification` , vous augmentez le risque d'attaques de type homme du milieu (HDM).

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.

Exemples x-amazon-apigateway-integration.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-compression-size

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 Compression de charge utile pour REST APIs dans API Gateway.

x-amazon-apigateway-minimum-compression-size example

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 politique de ressources pour une API REST. Pour plus d'informations sur les politiques de ressources, consultez Contrôle d'accès à une API REST avec des politiques de ressources API Gateway. Pour obtenir des exemples de politique de ressources, consultez Exemples de politiques de ressources API Gateway.

Exemple de politique x-amazon-apigateway-policy

L'exemple suivant spécifie une politique de ressources pour une API REST. La politique 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/*`, à l'aide de la région actuelle, de votre identifiant de AWS compte et de 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"
                }
            }
        }
    ]
}
```

Propriété x-amazon-apigateway-request-validator

Indique un valideur de demande, en référençant un *request_validator_name* de la mappe Objet x-amazon-apigateway-request-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 API-level validateur s'applique à toutes les méthodes sauf s'il est remplacé par le validateur au niveau de la méthode.

Exemple de validateur x-amazon-apigateway-request-validator

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",
        ...
     }
}
```

Objet x-amazon-apigateway-request-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.

Nom de la propriété	Type	Description
request_validator_name	Objet x-amazon-apigateway-request-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 Propriété x-amazon-apigateway-request-validator.

Exemple de x-amazon-apigateway-request-validators

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
    }
  },
  ...
}
```

Objet x-amazon-apigateway-request-validators.requestValidator

Spécifie les règles de validation d'un valideur de demande dans le cadre de la définition de mappe Objet x-amazon-apigateway-request-validators.

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.

Exemple X-Amazon-APIGateway-Request-Validators.RequestValidator

L'exemple suivant illustre un valideur de demande parameter-only :

```
"params-only": {
    "validateRequestBody" : false,
    "validateRequestParameters" : true
}
```

Politique de sécurité de x-amazon-apigateway-

Spécifie une politique de sécurité pour une API REST. Si vous créez une politique de sécurité qui commence par`"SecurityPolicy_"`, vous devez également définir le mode d'accès aux terminaux.

Pour en savoir plus sur les politiques de sécurité, consultezPolitiques de sécurité pour les API REST dans API Gateway.

Exemple de politique de sécurité x-amazon-apigateway-

L'exemple suivant spécifie `SecurityPolicy_TLS13_1_3_2025_09` une API REST.

```
"x-amazon-apigateway-security-policy": "SecurityPolicy_TLS13_1_3_2025_09"
```

Propriété x-amazon-apigateway-tag-value

Spécifie la valeur d'une AWS balise pour une API HTTP. Vous pouvez utiliser la `x-amazon-apigateway-tag-value` propriété dans le cadre de l'objet de balise OpenAPI au niveau racine pour spécifier des AWS balises pour 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 plus d'informations sur le balisage, consultez Ajout de balises à vos ressources API Gateway.

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.

Exemple x-amazon-apigateway-tag-value

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

- Protection des données dans Amazon API Gateway
- Identity and Access Management pour Amazon API Gateway
- Journalisation et surveillance dans Amazon API Gateway
- Validation de conformité pour Amazon API Gateway
- Résilience dans Amazon API Gateway
- Sécurité de l'infrastructure dans Amazon API Gateway
- Analyse des vulnérabilités dans Amazon API Gateway

- Bonnes pratiques de sécurité dans Amazon API Gateway

Protection des données dans Amazon API Gateway

Le modèle de responsabilité AWS partagée (modèle de) s'applique à la protection des données dans Amazon API Gateway. Comme décrit dans ce modèle, AWS est chargé de protéger l'infrastructure mondiale qui gère tous les AWS Cloud. 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 des Services AWS que vous utilisez. Pour plus d'informations sur la confidentialité des données, consultez la FAQ sur la confidentialité des données et les . Pour plus d'informations sur la protection des données en Europe, consultez le Centre du règlement général sur la protection des données (RGPD).

À des fins de protection des données, nous vous recommandons de protéger les Compte AWS informations d'identification et de configurer les utilisateurs individuels avec AWS IAM Identity Center ou Gestion des identités et des accès AWS (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.
- SSL/TLS À utiliser pour communiquer avec AWS les ressources. Nous exigeons TLS 1.2 et recommandons TLS 1.3.
- Configurez l'API et la journalisation de l'activité des utilisateurs avec AWS CloudTrail. Pour plus d'informations sur l'utilisation des CloudTrail sentiers pour capturer AWS des activités, consultez la section Utilisation des CloudTrail sentiers dans le guide de AWS CloudTrail l'utilisateur.
- Utilisez des solutions de AWS chiffrement, ainsi que tous les contrôles de sécurité par défaut qu'ils contiennent 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 par la norme FIPS 140-3 pour accéder AWS via une interface de ligne de commande ou une API, utilisez un point de terminaison FIPS. Pour plus d'informations sur les points de terminaison FIPS disponibles, consultez Norme FIPS (Federal Information Processing Standard) 140-3.

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 Nom. Cela inclut lorsque vous travaillez avec API Gateway ou une autre solution Services AWS à l'aide de la console, de l'API ou AWS des SDK. AWS CLI Toutes les données que vous entrez 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 Paramètres du cache des API REST dans API Gateway.

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 et déchiffrement de la clé privée de votre certificat

Lorsque vous créez un nom de domaine personnalisé pour des API privées, votre certificat ACM et votre clé privée sont chiffrés à l'aide d'une clé KMS gérée par AWS portant l'alias aws/acm. Vous pouvez afficher l'ID de clé portant cet alias dans la console AWS KMS sous Clés gérées par AWS.

API Gateway n'accède pas directement à vos ressources ACM. Il utilise AWS TLS Connection Manager pour sécuriser les clés privées de votre certificat et y accéder. Lorsque vous utilisez votre certificat ACM pour créer un nom de domaine personnalisé API Gateway pour des API privées, API Gateway associe votre certificat à AWS TLS Connection Manager. Pour cela, une autorisation est créée dans AWS KMS pour votre clé gérée par AWS avec le préfixe aws/acm. Une autorisation est un instrument de politique qui autorise TLS Connection Manager à utiliser des clés KMS dans les opérations de chiffrement. Elle permet au principal bénéficiaire (TLS Connection Manager) d'appeler les opérations d'autorisation spécifiées sur la clé KMS pour déchiffrer la clé privée de votre certificat. TLS Connection Manager utilise ensuite le certificat et la clé privée déchiffrée (texte brut) pour établir une connexion sécurisée (session SSL/TLS) avec les clients des services API Gateway. Lorsque le certificat est dissocié d'un nom de domaine personnalisé API Gateway pour les API privées, l'autorisation est supprimée.

Si vous souhaitez supprimer l'accès à la clé KMS, nous vous recommandons de remplacer ou de supprimer le certificat du service à l'aide de la Console de gestion AWS ou de la commande `update-service` de l'AWS CLI.

Contexte de chiffrement pour API Gateway

Un contexte de chiffrement est un ensemble optionnel de paires clé-valeur contenant des informations contextuelles sur l'utilisation possible de vos clés privées. AWS KMS lie le contexte de chiffrement aux données chiffrées et l'utilise comme données authentifiées supplémentaires pour prendre en charge le chiffrement authentifié.

Lorsque vos clés TLS sont utilisées avec API Gateway et TLS Connection Manager, le nom de votre service API Gateway est inclus dans le contexte de chiffrement utilisé pour chiffrer votre clé au repos. Vous pouvez vérifier le nom de domaine personnalisé API Gateway pour lequel votre certificat et votre clé privée sont utilisés en consultant le contexte de chiffrement dans vos journaux CloudTrail, comme indiqué dans la section suivante, ou en consultant l'onglet Ressources associées de la console ACM.

Pour déchiffrer des données, le même contexte de chiffrement est inclus dans la demande. API Gateway utilise le même contexte de chiffrement dans toutes les opérations cryptographiques AWS KMS, où la clé est `aws:apigateway:arn` et la valeur est l'Amazon Resource Name (ARN) de la ressource `PrivateDomainName` d'API Gateway.

L'exemple suivant présente le contexte de chiffrement dans la sortie d'une opération telle que `CreateGrant`.

```
"constraints": {
"encryptionContextEquals": {
"aws:acm:arn": "arn:aws:acm:us-west-2:859412291086:certificate/9177097a-
f0ae-4be1-93b1-19f911ea4f88",
"aws:apigateway:arn": "arn:aws:apigateway:us-west-2:859412291086:/domainnames/denytest-
part1.pdx.sahig.people.aws.dev+cbaeumzjhg"
}
},
"operations": [
"Decrypt"
],
"granteePrincipal": "tlsconnectionmanager.amazonaws.com"
```

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 Choisissez 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 politique de sécurité pour la distribution CloudFront avec TLS 1.1 ou supérieur, en fonction de vos exigences de sécurité et de conformité.

Pour plus d'informations sur la protection des données, consultez Protection de vos API REST dans API Gateway 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 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

Gestion des identités et des accès AWS (IAM) est un outil Service AWS qui permet à un administrateur de contrôler en toute sécurité l'accès aux AWS ressources. 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 outil que vous pouvez utiliser sans frais supplémentaires.

Rubriques

- Public ciblé
- Authentification par des identités
- Gestion de l’accès à l’aide de politiques
- Fonctionnement d'Amazon API Gateway avec IAM
- Exemples de stratégie basée sur l'identité d'Amazon API Gateway
- Exemples de politique basée sur les ressources d'Amazon API Gateway
- Résolution des problèmes d'identité et d'accès à Amazon API Gateway
- Utilisation de rôles liés à un service pour API Gateway

Public ciblé

La façon dont vous utilisez Gestion des identités et des accès AWS (IAM) varie en fonction de votre rôle :

- Utilisateur du service : demandez des autorisations à votre administrateur si vous ne pouvez pas accéder aux fonctionnalités (voir Résolution des problèmes d'identité et d'accès à Amazon API Gateway)
- Administrateur du service : déterminez l’accès des utilisateurs et soumettez les demandes d’autorisation (voir Fonctionnement d'Amazon API Gateway avec IAM)
- Administrateur IAM : rédigez des politiques pour gérer l’accès (voir Exemples de stratégie basée sur l'identité d'Amazon API Gateway)

Authentification par des identités

L'authentification est la façon dont vous vous connectez à AWS l'aide de vos informations d'identification. Vous devez être authentifié en tant qu'utilisateur IAM ou en assumant un rôle IAM. Utilisateur racine d'un compte AWS

Vous pouvez vous connecter en tant qu'identité fédérée à l'aide d'informations d'identification provenant d'une source d'identité telle que AWS IAM Identity Center (IAM Identity Center), d'une authentification unique ou d'informations d'identification. Google/Facebook Pour plus d'informations sur la connexion, consultez Connexion à votre Compte AWS dans le Guide de l'utilisateur Connexion à AWS .

Pour l'accès par programmation, AWS fournit un SDK et une CLI pour signer les demandes de manière cryptographique. Pour plus d'informations, consultez Signature AWS Version 4 pour les demandes d'API dans le Guide de l'utilisateur IAM.

Compte AWS utilisateur root

Lorsque vous créez un Compte AWS, vous commencez par une seule identité de connexion appelée utilisateur Compte AWS root qui dispose d'un accès complet à toutes Services AWS les ressources. Il est vivement déconseillé d'utiliser l'utilisateur racine pour vos tâches quotidiennes. Pour les tâches qui requièrent des informations d'identification de l'utilisateur racine, consultez Tâches qui requièrent les informations d'identification de l'utilisateur racine dans le Guide de l'utilisateur IAM.

Utilisateurs et groupes IAM

Un utilisateur IAM est une identité qui dispose d'autorisations spécifiques pour une seule personne ou application. Nous vous recommandons d'utiliser ces informations d'identification temporaires au lieu des utilisateurs IAM avec des informations d'identification à long terme. Pour plus d'informations, voir Exiger des utilisateurs humains qu'ils utilisent la fédération avec un fournisseur d'identité pour accéder à AWS l'aide d'informations d'identification temporaires dans le guide de l'utilisateur IAM.

Les groupes IAM spécifient une collection d'utilisateurs IAM et permettent de gérer plus facilement les autorisations pour de grands ensembles d'utilisateurs. Pour plus d'informations, consultez Cas d'utilisation pour les utilisateurs IAM dans le Guide de l'utilisateur IAM.

Rôles IAM

Un rôle IAM est une identité dotée d'autorisations spécifiques qui fournit des informations d'identification temporaires. Vous pouvez assumer un rôle en passant d'un rôle d'utilisateur à un rôle

IAM (console) ou en appelant une opération d' AWS API AWS CLI ou d'API. Pour plus d'informations, consultez Méthodes pour endosser un rôle dans le Guide de l'utilisateur IAM.

Les rôles IAM sont utiles pour l'accès des utilisateurs fédérés, les autorisations temporaires des utilisateurs IAM, les accès intercompte, les accès entre services et les applications exécutées sur Amazon EC2. Pour plus d'informations, consultez Accès intercompte aux ressources dans IAM dans le Guide de l'utilisateur IAM.

Gestion de l'accès à l'aide de politiques

Vous contrôlez l'accès en AWS créant des politiques et en les associant à AWS des identités ou à des ressources. Une politique définit les autorisations lorsqu'elles sont associées à une identité ou à une ressource. AWS évalue ces politiques lorsqu'un directeur fait une demande. La plupart des politiques sont stockées AWS sous forme de documents JSON. Pour plus d'informations les documents de politique JSON, consultez Vue d'ensemble des politiques JSON dans le Guide de l'utilisateur IAM.

À l'aide de politiques, les administrateurs précisent qui a accès à quoi en définissant 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. Un administrateur IAM crée des politiques IAM et les ajoute aux rôles, que les utilisateurs peuvent ensuite assumer. Les politiques IAM définissent les autorisations quelle que soit la méthode que vous utilisez pour exécuter l'opération.

Politiques basées sur l'identité

Les stratégies basées sur l'identité sont des documents de stratégie d'autorisations JSON que vous attachez à une identité (utilisateur, groupe ou rôle). Ces politiques contrôlent les actions que peuvent exécuter ces identités, sur quelles ressources et dans quelles conditions. Pour découvrir comment créer une politique basée sur l'identité, consultez Définition d'autorisations IAM personnalisées avec des politiques gérées par le client dans le Guide de l'utilisateur IAM.

Les politiques basées sur l'identité peuvent être des politiques intégrées (intégrées directement dans une seule identité) ou des politiques gérées (politiques autonomes associées à plusieurs identités). Pour découvrir comment choisir entre des politiques gérées et 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. Les exemples incluent les politiques de confiance de rôle IAM et les stratégies 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. Vous devez spécifier un principal dans une politique basée sur les ressources.

Les politiques basées sur les ressources sont des politiques en ligne situées dans ce service. Vous ne pouvez pas utiliser les politiques AWS gérées par IAM dans une stratégie basée sur les ressources.

Listes de contrôle d'accès (ACLs)

Les listes de contrôle d'accès (ACLs) contrôlent les principaux (membres du compte, utilisateurs ou rôles) autorisés à accéder à une ressource. ACLs sont similaires aux politiques basées sur les ressources, bien qu'elles n'utilisent pas le format de document de politique JSON.

Amazon S3 et AWS WAF Amazon VPC sont des exemples de services compatibles. ACLs Pour en savoir plus ACLs, consultez la présentation de la liste de contrôle d'accès (ACL) dans le guide du développeur Amazon Simple Storage Service.

Autres types de politique

AWS prend en charge des types de politiques supplémentaires qui peuvent définir les autorisations maximales accordées par les types de politiques les plus courants :

- Limites d'autorisations : une limite des autorisations définit le nombre maximum d'autorisations qu'une politique basée sur l'identité peut accorder à une entité IAM. Pour plus d'informations, consultez Limites d'autorisations pour des entités IAM dans le Guide de l'utilisateur IAM.
- Politiques de contrôle des services (SCPs) — Spécifiez les autorisations maximales pour une organisation ou une unité organisationnelle dans AWS Organizations. Pour plus d'informations, consultez Politiques de contrôle de service dans le Guide de l'utilisateur AWS Organizations .
- Politiques de contrôle des ressources (RCPs) : définissez le maximum d'autorisations disponibles pour les ressources de vos comptes. Pour plus d'informations, voir Politiques de contrôle des ressources (RCPs) dans le guide de AWS Organizations l'utilisateur.
- Politiques de session : politiques avancées que vous passez en tant que paramètre lorsque vous créez par programmation une session temporaire pour un rôle ou un utilisateur fédéré. Pour plus d'informations, consultez Politiques de session dans le Guide de l'utilisateur IAM.

Plusieurs types de politique

Lorsque plusieurs types de politiques s'appliquent à la requête, les autorisations en résultant sont plus compliquées à comprendre. Pour savoir comment AWS déterminer s'il faut autoriser une demande lorsque plusieurs types de politiques sont impliqués, consultez la section Logique d'évaluation des politiques 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 une vue d'ensemble du fonctionnement d'API Gateway et des autres AWS services avec IAM, consultez la section AWS Services That Work with IAM dans le guide de l'utilisateur d'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 montre une politique basée sur l'identité qui permet à un utilisateur de créer ou de mettre à jour uniquement un REST privé. APIs Pour obtenir plus d'exemples, consultez the section called "Exemples de stratégies basées sur l'identité".

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "ScopeToPrivateApis",
      "Effect": "Allow",
      "Action": [
        "apigateway:PATCH",
        "apigateway:POST",
        "apigateway:PUT"
      ],
      "Resource": [
        "arn:aws:apigateway:us-east-1::/restapis",
        "arn:aws:apigateway:us-east-1::/restapis/??????????"
      ],
      "Condition": {
        "ForAllValues:StringEqualsIfExists": {
          "apigateway:Request/EndpointType": "PRIVATE",
          "apigateway:Resource/EndpointType": "PRIVATE"
        }
      }
    },
    {
      "Sid": "AllowResourcePolicyUpdates",
      "Effect": "Allow",
      "Action": [
          "apigateway:UpdateRestApiPolicy"
      ],
      "Resource": [
          "arn:aws:apigateway:us-east-1::/restapis/*"
      ]
    }
  ]
}
```

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.

Le format `apigateway:`*`action`* de l'`Action`expression de gestion des API *`action`* est l'une des actions API Gateway suivantes : GET, POST, PUT, DELETE, PATCH (pour mettre à jour les ressources) ou *, qui correspond à toutes les 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 spécifiques d'API Gateway, consultez la référence d'API (REST APIs) d'Amazon API Gateway version 1 et la référence d'API Amazon API Gateway version 2 (WebSocket et HTTP APIs).

Pour de plus amples informations, veuillez consulter the section called “Exemples de stratégies basées sur l'identité”.

Ressources

Les administrateurs peuvent utiliser les politiques AWS JSON 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 auxquels l’action s’applique. Il est recommandé de définir une ressource à l’aide de son Amazon Resource Name (ARN). Pour les actions qui ne sont pas compatibles avec les autorisations de niveau ressource, 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 ressources REST APIs et les sous-ressources appartenant à un compte spécifique, utilisez le caractère générique (*) :

```
"Resource": "arn:aws:apigateway:us-east-2::/restapis/*"
```

Pour obtenir une liste des types de ressources API Gateway et de leurs ARNs caractéristiques, consultezRéférence Amazon Resource Name (ARN) API Gateway.

Clés de condition

Les administrateurs peuvent utiliser les politiques AWS JSON 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` indique à quel moment les instructions s'exécutent en fonction de critères définis. 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. Pour voir toutes les clés de condition AWS globales, voir les clés de contexte de condition AWS globales 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 politiques d'autorisation basées sur les ressources pour REST. APIs 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 politiques de ressources API Gateway”.

Exemples

Pour accéder à des exemples de stratégies API Gateway basées sur les ressources, consultez Exemples de politiques 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 “Attribute-based contrôle d'accès”.

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 de votre AWS compte qui possède des 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' AWS STS API telles que AssumeRoleou GetFederationToken.

API Gateway prend en charge l'utilisation des informations d'identification temporaires.

Rôles liés à un service

Les rôles liés aux AWS services permettent aux services d'accéder aux ressources 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 à un 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 du Console de gestion AWS AWS CLI, ou AWS SDKs. Un administrateur IAM doit créer des politiques 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
- Autorisation accordée aux utilisateurs pour afficher leurs propres autorisations
- Autorisations de lecture simple
- Créer uniquement des mécanismes d'autorisation REQUEST ou JWT
- Exiger que le point de terminaison execute-api par défaut soit désactivé
- Permettre aux utilisateurs de créer ou de mettre à jour uniquement un REST privé APIs
- Exiger que les routes d'API disposent d'une autorisation
- Empêcher un utilisateur de créer ou de mettre à jour un lien VPC
- Exemples de politiques d’utilisation des règles de routage

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 :

- Commencez AWS par les politiques gérées et passez aux autorisations du moindre privilège : pour commencer à accorder des autorisations à vos utilisateurs et à vos charges de travail, utilisez les politiques AWS gérées qui accordent des autorisations pour de nombreux cas d'utilisation courants. Ils sont disponibles dans votre Compte AWS. Nous vous recommandons de réduire davantage les autorisations en définissant des politiques gérées par les AWS clients spécifiques à vos cas d'utilisation. Pour plus d’informations, consultez politiques gérées par AWS ou politiques gérées par AWS pour les activités professionnelles dans le Guide de l’utilisateur IAM.
- Accordez 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 d’IAM pour appliquer des autorisations, consultez politiques et autorisations dans IAM dans le Guide de l’utilisateur IAM.
- Utilisez 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 par le biais d'un service spécifique Service AWS, tel que CloudFormation. Pour plus d'informations, consultez Conditions pour éléments de politique JSON IAM dans le Guide de l'utilisateur IAM.

- Utilisez l'Analyseur d'accès IAM pour valider vos politiques IAM afin de garantir des autorisations sécurisées et fonctionnelles : l'Analyseur d'accès IAM 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 plus d'informations, consultez Validation de politiques avec IAM Access Analyzer dans le Guide de l'utilisateur IAM.
- Exiger l'authentification multifactorielle (MFA) : si vous avez un scénario qui nécessite des utilisateurs IAM ou un utilisateur root, activez l'authentification MFA pour une sécurité accrue. Compte AWS Pour exiger la MFA lorsque des opérations d'API sont appelées, ajoutez des conditions MFA à vos politiques. Pour plus d'informations, consultez Sécurisation de l'accès aux API avec 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 politique inclut les autorisations permettant d'effectuer cette action sur la console ou par programmation à l'aide de l'API AWS CLI or 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 politique autorise un utilisateur à obtenir des informations sur toutes les ressources d'un HTTP ou d'une WebSocket API dont l'identifiant est `a123456789` in the AWS Region of `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.

JSON

```
{
  "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 politique permet à un utilisateur de créer uniquement APIs avec `REQUEST` ou avec `JWT` des autorisateurs, y compris par 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.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "OnlyAllowSomeAuthorizerTypes",
      "Effect": "Allow",
      "Action": [
        "apigateway:PUT",
        "apigateway:POST",
        "apigateway:PATCH"
      ],
      "Resource": [
        "arn:aws:apigateway:us-east-1::/apis",
        "arn:aws:apigateway:us-east-1::/apis/??????????",
        "arn:aws:apigateway:us-east-1::/apis/*/authorizers",
        "arn:aws:apigateway:us-east-1::/apis/*/authorizers/*"
      ],
      "Condition": {
        "ForAllValues:StringEqualsIfExists": {
          "apigateway:Request/AuthorizerType": [
            "REQUEST",
            "JWT"
          ]
        }
      }
    }
  ]
```

```
}
```

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.

Comme la `apis/*` ressource capture également des sous-ressources telles que les autorisateurs ou les méthodes, nous la définissons explicitement à l' APIs aide d'une simple `Deny` instruction.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "DisableExecuteApiEndpoint",
      "Effect": "Allow",
      "Action": [
        "apigateway:PATCH",
        "apigateway:POST",
        "apigateway:PUT"
      ],
      "Resource": [
        "arn:aws:apigateway:us-east-1::/apis",
        "arn:aws:apigateway:us-east-1::/apis/*"
      ],
      "Condition": {
        "BoolIfExists": {
          "apigateway:Request/DisableExecuteApiEndpoint": true
        }
      }
    },
```

```
    {
      "Sid": "ScopeDownToJustApis",
      "Effect": "Deny",
      "Action": [
        "apigateway:PATCH",
        "apigateway:POST",
        "apigateway:PUT"
      ],
      "Resource": [
        "arn:aws:apigateway:us-east-1::/apis/*/*"
      ]
    }
  ]
}
```

Permettre aux utilisateurs de créer ou de mettre à jour uniquement un REST privé APIs

Cet exemple de politique utilise des clés de condition pour obliger un utilisateur à créer uniquement `PRIVATE` APIs et pour empêcher les mises à jour susceptibles de modifier le type `PRIVATE` d'une API, 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 le matcher (?), qui n'est pas gourmand, pour établir une correspondance explicite avec l'API afin d'empêcher l'autorisation de ressources non IDs liées à l'API, telles que les autorisateurs.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "ScopePutToPrivateApis",
            "Effect": "Allow",
            "Action": [
                "apigateway:PUT"
```

```
            ],
            "Resource": [
                "arn:aws:apigateway:us-east-1::/restapis",
                "arn:aws:apigateway:us-east-1::/restapis/??????????"
            ],
            "Condition": {
                "ForAllValues:StringEquals": {
                    "apigateway:Resource/EndpointType": "PRIVATE"
                }
            }
        },
        {
            "Sid": "ScopeToPrivateApis",
            "Effect": "Allow",
            "Action": [
                "apigateway:DELETE",
                "apigateway:PATCH",
                "apigateway:POST"
            ],
            "Resource": [
                "arn:aws:apigateway:us-east-1::/restapis",
                "arn:aws:apigateway:us-east-1::/restapis/??????????"
            ],
            "Condition": {
                "ForAllValues:StringEquals": {
                    "apigateway:Request/EndpointType": "PRIVATE",
                    "apigateway:Resource/EndpointType": "PRIVATE"
                }
            }
        },
        {
            "Sid": "AllowResourcePolicyUpdates",
            "Effect": "Allow",
            "Action": [
                "apigateway:UpdateRestApiPolicy"
            ],
            "Resource": [
                "arn:aws:apigateway:us-east-1::/restapis/*"
            ]
        }
    ]
}
```

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.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "AllowUpdatesOnApisAndRoutes",
      "Effect": "Allow",
      "Action": [
        "apigateway:POST",
        "apigateway:PATCH",
        "apigateway:PUT"
      ],
      "Resource": [
        "arn:aws:apigateway:us-east-1::/apis",
        "arn:aws:apigateway:us-east-1::/apis/??????????",
        "arn:aws:apigateway:us-east-1::/apis/*/routes",
        "arn:aws:apigateway:us-east-1::/apis/*/routes/*"
      ]
    },
    {
      "Sid": "DenyUnauthorizedRoutes",
      "Effect": "Deny",
      "Action": [
        "apigateway:POST",
        "apigateway:PATCH",
        "apigateway:PUT"
      ],
      "Resource": [
        "arn:aws:apigateway:us-east-1::/apis",
        "arn:aws:apigateway:us-east-1::/apis/*"
      ],
      "Condition": {
        "ForAnyValue:StringEqualsIgnoreCase": {
```

```
          "apigateway:Request/RouteAuthorizationType": "NONE"
        }
      }
    }
  ]
}
```

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.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "DenyVPCLink",
      "Effect": "Deny",
      "Action": [
        "apigateway:POST",
        "apigateway:PUT",
        "apigateway:PATCH"
      ],
      "Resource": [
        "arn:aws:apigateway:us-east-1::/vpclinks",
        "arn:aws:apigateway:us-east-1::/vpclinks/*"
      ]
    }
  ]
}
```

Exemples de politiques d’utilisation des règles de routage

Les exemples de politiques suivants montrent comment utiliser les clés de RoutingRule condition pour contrôler la manière dont les utilisateurs peuvent acheminer le trafic de leurs noms de domaine personnalisés vers leur REST APIs. Vous pouvez utiliser ces exemples pour créer des politiques

détaillées concernant les règles de routage que les utilisateurs peuvent définir. Pour de plus amples informations, veuillez consulter the section called “règles de routage”.

Méthode pour empêcher un utilisateur de modifier la façon dont un nom de domaine personnalisé achemine une demande

Cette politique empêche un utilisateur de créer ou de mettre à jour `BasePathMapping`, `ApiMapping` ou `RoutingRule`. Toutes ces ressources peuvent modifier la manière dont un nom de domaine personnalisé achemine les demandes APIs.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "DenyAccessBasePathMappingsApiMappingsRoutingRules",
      "Effect": "Deny",
      "Action": "apigateway:*",
      "Resource": [
        "arn:aws:apigateway:us-east-1::/domainnames/example.com/basepathmappings/
*",
        "arn:aws:apigateway:us-east-1::/domainnames/example.com/apimappings/*",
        "arn:aws:apigateway:us-east-1:111122223333:/domainnames/example.com/
routingrules/*"
      ]
    }
  ]
}
```

Autorisation d’un utilisateur à mettre à jour certaines priorités d’une règle de routage

Cette politique autorise un utilisateur à modifier la priorité d’une règle de routage entre 1 001 et 2 000. Vous pouvez utiliser cette règle pour séparer vos règles de production des règles de priorité inférieure, puis autoriser les utilisateurs à modifier les règles de priorité inférieure sans affecter les règles de production.

JSON

```
{
```

```
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "UpdatingRoutingRulePriorityBetween1001And2000",
      "Effect": "Allow",
      "Action": "apigateway:UpdateRoutingRule",
      "Resource": "arn:aws:apigateway:us-east-1:111122223333:/domainnames/
example.com/routingrules/*",
      "Condition": {
        "NumericGreaterThanEquals": {
          "apigateway:Resource/Priority": 1001,
          "apigateway:Request/Priority": 1001
        },
       "NumericLessThanEquals": {
          "apigateway:Resource/Priority": 2000,
          "apigateway:Request/Priority": 2000
        }
      }
    }
  ]
}
```

Autorisation d'un utilisateur à mettre à jour une règle de routage ou un mappage de chemin de base pour une certaine valeur de chemin de base

Cette politique autorise un utilisateur à mettre à jour un mappage de chemin de base uniquement pour les chemins de base commençant par `orders`, ou à mettre à jour une règle de routage correspondant à un chemin de base commençant par `orders`. Dans cette politique, un utilisateur peut mettre à jour un mappage de chemin de base ou une règle de routage pour `orders/create` ou `orders123`, mais pas pour `payment/orders`.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
      {
        "Sid": "AllowUpdateRoutingRuleUnderPathOrders",
        "Effect": "Allow",
        "Action": "apigateway:UpdateRoutingRule",
```

```
        "Resource": "arn:aws:apigateway:us-east-1:111122223333:/domainnames/
example.com/routingrules/*",
        "Condition": {
            "ForAllValues:StringLike": {
                "apigateway:Request/ConditionBasePaths": ["orders*"],
                "apigateway:Resource/ConditionBasePaths": ["orders*"]
            },
            "Null":{
                 "apigateway:Request/ConditionBasePaths":"false",
                 "apigateway:Resource/ConditionBasePaths":"false"
          }
        }
      }
  ]
}
```

Autorisation d'un utilisateur à mettre à jour des valeurs spécifiques du mode de routage

Cette politique autorise un utilisateur à remplacer le mode de routage par `API_MAPPING_ONLY` et `ROUTING_RULE_THEN_API_MAPPING` uniquement. Pour plus d'informations sur le mode de routage, consultez the section called "Définition du mode de routage de votre nom de domaine personnalisé".

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
        {
           "Sid": "AllowUpdateRoutingModeToAnythingWithApiMapping",
           "Effect": "Allow",
           "Action": ["apigateway:PATCH"],
           "Resource": "arn:aws:apigateway:us-east-1::/domainnames/example.com",
              "Condition": {
                "StringLike": {
                    "apigateway:Request/RoutingMode":"*API_MAPPING*"
                }
           }
       }
    ]
}
```

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 politiques 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 effectuer iam : PassRole
- Je souhaite autoriser des personnes extérieures à mon AWS compte 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 because no identity-based policy
 allows the GetWidget action
```

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 besoin d'aide, contactez votre AWS administrateur. Votre administrateur vous a fourni vos informations d’identification de connexion.

Je ne suis pas autorisé à effectuer 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 vous Services AWS permettent de transmettre un rôle existant à ce service au lieu de créer un nouveau rôle de service ou un rôle lié à un service. Pour ce faire, vous devez 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 n'est pas autorisée à transmettre le rôle au service.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

Dans ce cas, les politiques de Mary doivent être mises à jour pour lui permettre d’exécuter l’action `iam:PassRole`.

Si vous avez besoin d'aide, contactez votre AWS administrateur. Votre administrateur vous a fourni vos informations d’identification de connexion.

Je souhaite autoriser des personnes extérieures à mon AWS compte 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 pourront 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 (ACLs), vous pouvez utiliser ces politiques pour autoriser les utilisateurs à accéder à vos ressources.

Pour plus d’informations, 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 fournir l'accès à vos ressources sur celles Comptes AWS que vous possédez, consultez la section Fournir l'accès à un utilisateur IAM dans un autre utilisateur Compte AWS que vous possédez dans le Guide de l'utilisateur IAM.

- Pour savoir comment fournir l'accès à vos ressources à des tiers Comptes AWS, consultez la section Fournir un accès à des ressources Comptes AWS détenues par 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 en savoir plus sur la différence entre l'utilisation des rôles et des politiques basées sur les ressources pour l'accès intercompte, consultez Accès intercompte aux ressources dans IAM dans le Guide de l'utilisateur IAM.

Utilisation de rôles liés à un service pour API Gateway

Amazon API Gateway utilise des rôles Gestion des identités et des accès AWS 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 politique d'approbation et la politique d'autorisation. De plus, cette politique 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 à un 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 AWSService RoleFor APIGateway lié à un service fait confiance aux services suivants pour assumer le rôle :

- `ops.apigateway.amazonaws.com`

La politique d'autorisations liée au rôle permet à API Gateway de réaliser les actions suivantes sur les ressources spécifiées :

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "elasticloadbalancing:AddListenerCertificates",
                "elasticloadbalancing:RemoveListenerCertificates",
                "elasticloadbalancing:ModifyListener",
                "elasticloadbalancing:DescribeListeners",
                "elasticloadbalancing:DescribeLoadBalancers",
                "xray:PutTraceSegments",
                "xray:PutTelemetryRecords",
                "xray:GetSamplingTargets",
                "xray:GetSamplingRules",
                "logs:CreateLogDelivery",
                "logs:GetLogDelivery",
                "logs:UpdateLogDelivery",
                "logs:DeleteLogDelivery",
                "logs:ListLogDeliveries",
                "servicediscovery:DiscoverInstances"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "firehose:DescribeDeliveryStream",
                "firehose:PutRecord",
                "firehose:PutRecordBatch"
```

```
            ],
            "Resource": "arn:aws:firehose:*:*:deliverystream/amazon-apigateway-*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "acm:DescribeCertificate",
                "acm:GetCertificate"
            ],
            "Resource": "arn:aws:acm:*:*:certificate/*"
        },
        {
            "Effect": "Allow",
            "Action": "ec2:CreateNetworkInterfacePermission",
            "Resource": "arn:aws:ec2:*:*:network-interface/*"
        },
        {
            "Effect": "Allow",
            "Action": "ec2:CreateTags",
            "Resource": "arn:aws:ec2:*:*:network-interface/*",
            "Condition": {
                "ForAllValues:StringEquals": {
                    "aws:TagKeys": [
                        "Owner",
                        "VpcLinkId"
                    ]
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:ModifyNetworkInterfaceAttribute",
                "ec2:DeleteNetworkInterface",
                "ec2:AssignPrivateIpAddresses",
                "ec2:CreateNetworkInterface",
                "ec2:DeleteNetworkInterfacePermission",
                "ec2:DescribeNetworkInterfaces",
                "ec2:DescribeAvailabilityZones",
                "ec2:DescribeNetworkInterfaceAttribute",
                "ec2:DescribeVpcs",
                "ec2:DescribeNetworkInterfacePermissions",
                "ec2:UnassignPrivateIpAddresses",
                "ec2:DescribeSubnets",
```

```
                "ec2:DescribeRouteTables",
                "ec2:DescribeSecurityGroups"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "servicediscovery:GetNamespace",
            "Resource": "arn:aws:servicediscovery:*:*:namespace/*"
        },
        {
            "Effect": "Allow",
            "Action": "servicediscovery:GetService",
            "Resource": "arn:aws:servicediscovery:*:*:service/*"
        }
    ]
}
```

Vous devez configurer les autorisations de manière à permettre à une entité IAM (comme un utilisateur, un 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 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' Console de gestion AWS 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 à un service pour vous.

Modification d'un rôle lié à un service pour API Gateway

API Gateway ne vous permet pas de modifier le rôle AWSService RoleFor APIGateway 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 en savoir plus, consultez 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 AWSService RoleFor APIGateway

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 à un service.
3. Utilisez la console pour supprimer la ressource.
4. Répétez la procédure pour supprimer tous les APIs noms de domaine personnalisés ou les liens VPC qui utilisent le rôle lié à un service.

Pour supprimer manuellement le rôle lié au service à l’aide d’IAM

Utilisez la console IAM, le AWS CLI, ou l' AWS API pour supprimer le rôle lié au AWSService RoleFor APIGateway service. Pour plus d’informations, consultez Suppression d’un rôle lié à un service dans le Guide de l’utilisateur IAM.

Régions prises en charge pour les rôles liés à un 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 politique `AWSServiceRoleForAPIGateway`.	La politique `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 est un enjeu important pour assurer la fiabilité, la disponibilité et les performances d'API Gateway et de vos solutions AWS. Vous devez collecter les données de surveillance de toutes les parties de votre solution AWS afin de pouvoir déboguer plus facilement une éventuelle défaillance à plusieurs points. AWS fournit plusieurs outils pour surveiller vos ressources API Gateway et répondre aux incidents potentiels :

Amazon CloudWatch Logs

Pour vous aider à déboguer les problèmes liés à l'exécution de requêtes ou à l'accès client à votre API, vous pouvez activer CloudWatch Logs pour journaliser les appels d'API. Pour de plus amples informations, veuillez consulter the section called "CloudWatch journaux".

Alarmes Amazon CloudWatch

Les alarmes Amazon CloudWatch permettent de surveiller une métrique unique sur une période de temps que vous spécifiez. Si la métrique dépasse un seuil donné, une notification est envoyée à une rubrique Amazon Simple Notification Service ou à une politique AWS Auto Scaling. Les alarmes CloudWatch n'appellent pas d'actions 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, consultez the section called "CloudWatch métriques".

Accès à la journalisation de Firehose

Pour vous aider à résoudre des problèmes liés à l'accès client à votre API, vous pouvez activer Firehose afin de journaliser les appels d'API. Pour de plus amples informations, consultez the section called "Firehose".

AWS CloudTrail

CloudTrail fournit un enregistrement des actions réalisées par un utilisateur, un rôle ou un service AWS dans API Gateway. Avec les 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 effectuée, l'auteur et la date de la demande, ainsi que d'autres détails. Pour de plus amples informations, veuillez consulter the section called "Travailler avec CloudTrail".

AWS X-Ray

X-Ray est un service AWS qui collecte des données sur les demandes traitées par votre application et les utilise pour construire une cartographie des services qui vous permet d'identifier les problèmes liés à votre application et les opportunités d'optimisation. Pour de plus amples informations, veuillez consulter the section called "Configurez AWS X-Ray".

AWS Config

AWS Config fournit une vue détaillée de la configuration des ressources AWS 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 utiliser AWS Config pour définir des règles qui évaluent les configurations de ressources pour assurer la conformité des données. Les règles AWS Config représentent les paramètres de configuration idéaux pour vos ressources API Gateway. Si une ressource enfreint une règle et est signalée comme non conforme, AWS Config peut vous avertir à l'aide d'une rubrique Amazon Simple Notification Service (Amazon SNS). Pour plus d'informations, consultez the section called "Travailler avec 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 adressés au service API

Gateway APIs. 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 du centre d'identité IAM.
- 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 Console de gestion AWS sont multirégionaux. Vous ne pouvez créer un journal de suivi en une ou plusieurs régions à l'aide de l' 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 journal de suivi pour une seule région, il convient de n'afficher que les événements enregistrés dans le journal de suivi pour une seule région Région AWS.

Pour plus d'informations sur les journaux de suivi, consultez Créez un journal de suivi dans vos Compte AWS et Création d'un journal de suivi pour une organisation dans le AWS CloudTrail Guide de 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 suivre les 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, AWS Config Rules cette fonctionnalité vous permet de définir des règles de configuration et de détecter, suivre et signaler automatiquement les violations de ces règles. En suivant les modifications apportées à ces propriétés de configuration des ressources, vous pouvez également créer des AWS Config règles déclenchées par des modifications pour vos ressources API Gateway et tester vos configurations de ressources par rapport aux meilleures pratiques.

Vous pouvez l'activer AWS Config dans votre compte à l'aide de la AWS Config console ou du 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, ces ressources API Gateway seront automatiquement enregistrées dans votre compte. Support pour Amazon API Gateway in AWS Config est disponible dans toutes les régions AWS publiques 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
- Con AWS Config figuration
- Configuration AWS Config pour enregistrer les ressources d'API Gateway
- Afficher les détails de configuration d'API Gateway dans la AWS Config console
- Évaluation des ressources d'API Gateway à l'aide de AWS Config règles

Types de ressources pris en charge

Les types de ressources API Gateway suivants sont intégrés AWS Config et sont documentés dans AWS Config Supported AWS Resource Types and Resource Relationships :

- `AWS::ApiGatewayV2::Api`(WebSocket et API HTTP)
- `AWS::ApiGateway::RestApi` (API REST)
- `AWS::ApiGatewayV2::Stage`(WebSocket et étape de l'API HTTP)
- `AWS::ApiGateway::Stage` (étape de l'API REST)

Pour plus d'informations à ce sujet AWS Config, consultez le guide du AWS Config développeur. 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. Dans le cas contraire, les attributs seront modifiés dans la AWS Config console, mais les paramètres de propriété précédents seront toujours en vigueur ; le comportement d'exécution de l'API restera inchangé.
>
> - **`AWS::ApiGateway::RestApi`** – `binaryMediaTypes`, `minimumCompressionSize`, `apiKeySource`

- **AWS::ApiGatewayV2::Api** – apiKeySelectionExpression

Con AWS Config figuration

Pour la configuration initiale AWS Config, consultez les rubriques suivantes du guide du AWS Config développeur.

- Configuration à l' AWS Config aide de la console
- Configuration à l' AWS Config aide du AWS CLI

Configuration AWS Config pour enregistrer les ressources d'API Gateway

Par défaut, AWS Config enregistre les modifications de configuration pour tous les types de ressources régionales pris en charge qu'il découvre dans la région dans laquelle votre environnement s'exécute. Vous pouvez le personnaliser AWS Config pour enregistrer uniquement les modifications relatives à des types de ressources spécifiques ou les modifications apportées aux ressources globales.

Pour en savoir plus sur les ressources régionales par rapport aux ressources mondiales et sur la façon de personnaliser votre AWS Config configuration, consultez la section Sélection des ressources AWS Config enregistrées.

Afficher les détails de configuration d'API Gateway dans la AWS Config console

Vous pouvez utiliser la AWS Config console pour rechercher des ressources API Gateway et obtenir des informations actuelles et historiques sur leurs configurations. La procédure ci-après illustre comment rechercher des informations sur une API Amazon API Gateway.

Pour rechercher une ressource API Gateway dans la console de AWS configuration

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 Type de ressource, faites défiler l'écran jusqu'à APIGateway APIGateway V2, puis choisissez un ou plusieurs types de ressources API Gateway.
5. Choisissez Recherche.

6. Choisissez un ID de ressource dans la liste des ressources qui AWS Config s'affiche. AWS Config affiche les détails de configuration et 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 les moyens de rechercher une ressource et d'afficher les informations sur cette page, consultez la section Affichage AWS des configurations et de l'historique des ressources dans le guide du AWS Config développeur.

Évaluation des ressources d'API Gateway à l'aide de AWS Config règles

Vous pouvez créer des AWS Config règles qui représentent les paramètres de configuration idéaux pour vos ressources API Gateway. Vous pouvez utiliser des règles AWS Config gérées prédéfinies ou définir des règles personnalisées. AWS Config suit en permanence les modifications apportées à la configuration de vos ressources afin de déterminer si ces modifications enfreignent l'une des conditions de vos règles. La AWS Config console indique l'état de conformité de vos règles et de vos ressources.

Si une ressource enfreint une règle et est signalée comme non conforme, vous AWS Config pouvez vous en informer à l'aide d'une rubrique Amazon Simple Notification Service Developer Guide (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 développeur AWS Config.

Validation de conformité pour Amazon API Gateway

Notre nouvelle expérience AWS d'inscription n'est pas conçue pour les charges de travail réglementées. Si vous utilisez notre nouvelle interface d' AWS inscription, mais que vous souhaitez l'utiliser AWS pour des charges de travail réglementées, vous pouvez vous inscrire à des fonctionnalités AWS (avancées) ou activer des fonctionnalités avancées pour votre AWS environnement.

Pour savoir si un s' Service AWS inscrit dans le cadre de programmes de conformité spécifiques, consultez Services AWS la section Étendue par programme de conformité et choisissez le

programme de conformité qui vous intéresse. Pour des informations générales, voir Programmes de AWS conformité Programmes .

Vous pouvez télécharger des rapports d'audit tiers à l'aide de AWS Artifact. Pour plus d'informations, consultez la section Téléchargement de rapports dans la section AWS Artifact .

Votre responsabilité en matière de conformité lors de l'utilisation Services AWS dépend de la sensibilité de vos données, des objectifs de conformité de votre entreprise et des lois et réglementations applicables. Pour plus d'informations sur votre responsabilité en matière de conformité lors de l'utilisation Services AWS, consultez AWS la documentation de sécurité.

Résilience dans Amazon API Gateway

L'infrastructure mondiale AWS s'articule autour 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 davantage disponibles, tolérantes aux pannes et ont une plus grande capacité de mise à l'échelle que les infrastructures traditionnelles à un ou plusieurs centres de données.

En tant que service régional entièrement géré, API Gateway fonctionne dans plusieurs zones de disponibilité de chaque région, en utilisant la redondance des zones de disponibilité pour minimiser les défaillances de l'infrastructure en tant que catégorie de risque pour la disponibilité. API Gateway est conçu pour récupérer automatiquement après la défaillance d'une zone de disponibilité.

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 Limitez les requêtes adressées à votre REST APIs pour améliorer le débit dans API Gateway.

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 API Gateway dans une région secondaire. Pour obtenir un exemple, consultez 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 AWS mondial. Pour plus d'informations sur les services AWS de sécurité et sur la manière dont AWS l'infrastructure est protégée, consultez la section Sécurité du AWS cloud. Pour concevoir votre AWS environnement en utilisant les meilleures pratiques en matière de sécurité de l'infrastructure, consultez la section Protection de l'infrastructure dans le cadre AWS bien architecturé du pilier de sécurité.

Vous utilisez des appels d'API AWS publiés 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 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.

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 politiques basées sur les ressources pour contrôler l'accès depuis des points de terminaison Amazon Virtual Private Cloud (Amazon VPC) spécifiques ou spécifiques. VPCs En fait, cela isole l'accès réseau à une ressource API Gateway donnée uniquement du VPC spécifique au sein AWS du réseau.

Analyse des vulnérabilités dans Amazon API Gateway

La configuration et les contrôles informatiques sont une responsabilité partagée entre AWS et vous, notre client. Pour plus d'informations, consultez 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 politiques 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 les politiques IAM pour mettre en œuvre l'accès avec le moindre privilège lors de la création, de la lecture, de la mise à jour ou de la suppression d'API Gateway APIs. Pour en savoir plus, veuillez consulter la section Identity and Access Management pour Amazon API Gateway. API Gateway propose plusieurs options pour contrôler l'accès à APIs ce que vous créez. Pour en savoir plus, consultez Contrôler et gérer l'accès à REST APIs dans API Gateway, Contrôler et gérer l'accès WebSocket APIs à API Gateway et Contrôle de l’accès aux API HTTP avec des mécanismes d’autorisation JWT dans API Gateway.

Mise en œuvre de la journalisation

Utilisez CloudWatch Logs ou Amazon Data Firehose pour enregistrer les demandes adressées à votre. APIs Pour en savoir plus, consultez Surveillance des API REST dans API Gateway, Configuration de la journalisation WebSocket APIs dans API Gateway et Configuration de la journalisation pour HTTP APIs dans API Gateway.

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 pas d'actions 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 en savoir plus, consultez the section called “Travailler avec AWS Config”.

Utiliser AWS Security Hub CSPM

Surveillez votre utilisation d’API Gateway, car elle est liée aux bonnes pratiques de sécurité, en utilisant AWS Security Hub CSPM. Security Hub CSPM utilise des contrôles de sécurité pour évaluer les configurations des ressources et les normes de sécurité afin de vous aider à vous conformer aux différents cadres de conformité. Pour plus d'informations sur l'utilisation de Security Hub CSPM pour évaluer les ressources d'API Gateway, consultez les contrôles Amazon API Gateway dans le guide de l'AWS Security Hub utilisateur.

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 politique Gestion des identités et des accès AWS (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 plus d'informations, consultez 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 AWS Tagging Strategies.

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`
- `DomainNameAccessAssociation`
- `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 balise est définie sur une `RestApi` ressource, cette balise est héritée par les ressources enfants suivantes de celle-ci `RestApi` pour le contrôle Attribute-based d'accès :
  - `Authorizer`
  - `Deployment`
  - `Documentation`
  - `GatewayResponse`
  - `Integration`

- `Method`
- `Model`
- `Resource`
- `ResourcePolicy`
- `Setting`
- `Stage`

- Si une balise est définie sur une ressource `DomainName`, les ressources `BasePathMapping`, `ApiMapping` et `RoutingRule` situées sous celle-ci héritent de cette balise.
- 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. Pour utiliser des balises d'allocation des coûts, nous vous recommandons de créer des balises sur des ressources enfants, telles que la ressource `Stage`.
> 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 peut hériter de la balise de sa ressource `RestApi` parent.

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 d'identification doit être unique, et chaque clé d'identification peut avoir une seule valeur.
- La longueur maximale de la clé de balise est de 128 caractères Unicode UTF-8.
- La longueur maximale de la valeur de balise est de 256 caractères Unicode UTF-8.
- Les caractères autorisés pour les clés et les valeurs sont les lettres, les chiffres, les espaces représentables dans UTF-8 et les caractères 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 politique pour mettre des balises en majuscule et mettre en œuvre cette politique 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 Gestion des identités et des accès AWS politiques font partie de la syntaxe que vous utilisez pour spécifier les autorisations d'accès aux ressources API Gateway. Pour plus d'informations sur la spécification de politiques IAM, consultez the section called "Utilisation des autorisations IAM". Dans API Gateway, les ressources et certaines actions peuvent comporter des balises. Lorsque vous créez une politique 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 de balises pour le contrôle d'accès basé sur les attributs peut permettre un contrôle plus fin que le contrôle, ainsi qu'un API-level contrôle plus dynamique que le contrôle d'accès basé sur les ressources. Il est possible de créer des politiques 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 politiques 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 politiques 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`.

JSON

```
{
  "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:StringEqualsIgnoreCase": {
          "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.

JSON

```
{
    "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.

JSON

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

```
                "apigateway:PUT",
                "apigateway:POST",
                "apigateway:DELETE"
            ],
            "Resource": [
                "arn:aws:apigateway:*::/tags/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "apigateway:GET",
                "apigateway:PATCH"
            ],
            "Resource": [
                "arn:aws:apigateway:*::*"
            ]
        }
    ]
}
```

Références d’API

Amazon API Gateway permet APIs de créer et de déployer votre propre protocole HTTP et WebSocket APIs. En outre, les API Gateway APIs sont disponibles en standard AWS SDKs.

Si vous utilisez un langage pour lequel un AWS SDK existe, vous préférerez peut-être utiliser le SDK plutôt que d'utiliser directement le REST API APIs Gateway. Ils SDKs simplifient l'authentification, s'intègrent facilement à votre environnement de développement et fournissent un accès facile aux commandes API Gateway.

Voici où se trouve la documentation AWS SDKs 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

Points de terminaison de service API Gateway

Un point de terminaison est une URL qui sert de point d'entrée à un service AWS Web. API Gateway prend en charge les types de point de terminaison suivants :

- the section called “IPv4 points de terminaison”
- Points de terminaison Dualstack (compatibles avec et) IPv4 IPv6
- points de terminaison FIPS.

Lorsque vous soumettez une demande, vous pouvez spécifier le point de terminaison et la région à utiliser. Si vous ne spécifiez aucun point de IPv4 terminaison, celui-ci est utilisé par défaut. Pour utiliser un autre type de point de terminaison, vous devez le spécifier dans votre demande. Pour obtenir un exemple de la façon de procéder, consultez the section called “Spécification des points de terminaison”. Pour obtenir un tableau des points de terminaison disponibles, consultez Points de terminaison Amazon API Gateway.

IPv4 points de terminaison

IPv4 les terminaux ne prennent en charge que IPv4 le trafic. IPv4 les points de terminaison sont disponibles pour toutes les régions.

Si vous spécifiez le point de terminaison général, `apigateway.amazonaws.com`, nous utilisons le point de terminaison pour `us-east-1`. Pour utiliser une autre région, spécifiez son point de terminaison associé. Par exemple, si vous le spécifiez `apigateway.us-east-2.amazonaws.com` comme point de terminaison, nous dirigeons votre demande vers le point de terminaison `us-east-2`.

IPv4 les noms des points de terminaison utilisent la convention de dénomination suivante :

- `apigateway.`*`region`*`.amazonaws.com`

Par exemple, le nom du IPv4 point de terminaison de la `eu-west-1` région est`apigateway.eu-west-1.amazonaws.com`.

Points de terminaison à double pile (IPv4 et IPv6)

Les points de terminaison Dualstack prennent en charge à la fois IPv4 le trafic et le trafic. IPv6 Lorsque vous envoyez une demande à un point de terminaison à double pile, l'URL du point de terminaison correspond à une adresse IPv6 ou à une IPv4 adresse, selon le protocole utilisé par votre réseau et votre client.

Les noms des points de terminaison à double pile utilisent la convention d'affectation de noms suivante :

- `apigateway.`*`region`*`.api.aws`

Par exemple, le nom du point de terminaison à double pile de la région `eu-west-1` est `apigateway.eu-west-1.api.aws`.

Spécification des points de terminaison

Les exemples suivants montrent comment spécifier un point de terminaison pour la `us-east-2` région à l'aide du AWS CLI for`apigateway`.

- Double pile

```
aws apigateway get-rest-apis --region us-east-2 --endpoint-url https://apigateway.us-east-2.api.aws
```

- IPv4

```
aws apigateway get-rest-apis --region us-east-2 --endpoint-url https://apigateway.us-east-2.amazonaws.com
```

Les exemples suivants montrent comment spécifier un point de terminaison pour la `us-east-2` région à l'aide du AWS CLI for`apigatewayv2`.

- Double pile

```
aws apigatewayv2 get-apis --region us-east-2 --endpoint-url https://apigateway.us-east-2.api.aws
```

- IPv4

```
aws apigatewayv2 get-apis --region us-east-2 --endpoint-url https://apigateway.us-east-2.amazonaws.com
```

Utilisation des IPv6 adresses dans les politiques IAM

Si vous utilisez des politiques utilisateur IAM ou des politiques de ressources API Gateway pour contrôler l'accès à API Gateway ou à toute autre API Gateway APIs, vérifiez que vos politiques sont mises à jour pour inclure les plages d' IPv6 adresses. Les politiques qui ne sont pas mises à jour pour gérer IPv6 les adresses peuvent avoir un impact sur l'accès du client à API Gateway lorsqu'il commence à utiliser le point de terminaison dualstack. Pour plus d’informations sur la gestion des autorisations d’accès avec IAM, consultez Identity and Access Management pour Amazon API Gateway.

Les stratégies IAM qui permettent de filtrer les adresses IP utilisent des opérateurs de condition d'adresse IP. La politique d'identité suivante permet aux adresses IP de la `54.240.143.*` plage d'obtenir des informations sur toutes les ressources d'un HTTP ou d'une WebSocket API avec l'identifiant `dea123456789`. Toute adresse IP située en dehors de cette plage se verra refuser l’accès aux informations sur toutes les ressources de l’API. Comme toutes les IPv6 adresses se situent en dehors de la plage autorisée, cette politique empêche IPv6 les adresses d'accéder aux informations relatives à l'API.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "IPAllow",
      "Effect": "Allow",
      "Principal": "*",
      "Action": "apigateway:GET",
      "Resource": "arn:aws:apigateway:us-east-1::/apis/a123456789/*",
      "Condition": {
         "IpAddress": {"aws:SourceIp": "54.240.143.0/24"}
      }
    }
  ]
}
```

Vous pouvez modifier l'`Condition`élément de la politique d'API pour autoriser à la fois les plages d'adresses IPv4 IPv6 (`54.240.143.0/24`2001:DB8:1234:5678::/64) et (), comme indiqué dans l'exemple suivant. Vous pouvez utiliser le même type de bloc `Condition` que celui indiqué dans l’exemple pour mettre à jour vos stratégies d’utilisateur IAM et stratégies de ressources API Gateway.

```
"Condition": {
  "IpAddress": {
     "aws:SourceIp": [
       "54.240.143.0/24",
        "2001:DB8:1234:5678::/64"
      ]
   }
 }
```

Quotas Amazon API Gateway

Les quotas suivants s'appliquent à tous les types d'API Amazon API Gateway.

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
Limiter le quota par compte, par région sur HTTP APIs APIs WebSocket APIs, REST et callback WebSocket APIs	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
Quota de limitation sans contrôle d'accès par compte et par région pour un portail	250 000 demandes par seconde	Non
Quota de limitation avec contrôle d'accès par compte et par région pour un portail	10 000 demandes par seconde	Non

* Pour les régions suivantes, le quota de limitation par défaut est de 2 500 RPS et le quota en mode rafale par défaut de 1 250 RPS : Afrique (Le Cap), Europe (Milan), Asie-Pacifique (Jakarta), Moyen-Orient (EAU), Asie-Pacifique (Hyderabad), Asie-Pacifique (Melbourne), Europe (Espagne), Europe (Zurich), Israël (Tel Aviv), Canada-Ouest (Calgary), Asie-Pacifique (Malaisie), Asie-Pacifique (Thaïlande) et Mexique (Centre).

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 API SDKs. 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
CreateRestApipour une API régionale ou privée	1 requête toutes les 3 secondes par compte	Non
CreateRestApipour une API optimisée pour les périphériques	1 requête toutes les 30 secondes par compte	Non

Action	Quota par défaut	Peut être augmenté
CreateVpcLink(V2)	1 demande toutes les 15 secondes par compte	Non
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
ImportRestApipour une API régionale ou privée	1 requête toutes les 3 secondes par compte	Non
ImportRestApipour une 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
Créer un portail	1 demande toutes les 3 secondes	Non
Portail de mise à jour	2 demandes par minute	Non
Obtenir le portail	10 requêtes par seconde	Non
Portails de listes	10 requêtes par seconde	Non
Portail de publication	2 demandes par minute	Non
DeletePortal	2 demandes par minute	Non
PreviewPortal	1 demande toutes les 3 secondes	Non
DisablePortal	2 demandes par minute	Non
GetPortalProduct	10 requêtes par seconde	Non
ListPortalProduct	5 demandes par seconde	Non
CreatePortalProduct	2 demandes par seconde	Non
UpdatePortalProduct	0,5 demande par seconde	Non
DeletePortalProduct	1 demande par seconde	Non
PutPortalProductSharingPolicy	1 demande toutes les 3 secondes	Non
GetPortalProductSharingPolicy	10 requêtes par seconde	Non

Action	Quota par défaut	Peut être augmenté
DeletePortalProductSharingPolicy	1 demande toutes les 3 secondes	Non
CreateProductRestEndpointPage	1 demande toutes les 3 secondes	Non
GetProductRestEndpointPage	10 requêtes par seconde	Non
UpdateProductRestEndpointPage	1 demande toutes les 3 secondes	Non
DeleteProductRestEndpointPage	1 demande toutes les 3 secondes	Non
ListProductRestEndpointPages	10 requêtes par seconde	Non
CreateProductPage	1 demande toutes les 3 secondes	Non
GetProductPage	10 requêtes par seconde	Non
UpdateProductPage	1 demande toutes les 3 secondes	Non
DeleteProductPage	1 demande toutes les 3 secondes	Non
ListProductPages	10 requêtes par seconde	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

Quotas pour la configuration et l'exécution d'une API REST dans API Gateway

Les quotas suivants s'appliquent à la configuration et à l'exécution d'une API REST dans Amazon API Gateway. S'il s'agit d'un quota par API, il ne peut être augmenté que pour certaines API spécifiques, et non pour toutes les API d'un compte.

Pour plus d'informations sur les quotas au niveau du compte consultez Quotas et remarques importantes.

Ressource ou opération	Quota par défaut	Peut être augmenté
Noms de domaine personnalisés publics par compte et par région	120	Oui
Noms de domaine personnalisés privés par compte et par région	50	Oui
API régionales par région	600	Non
Edge-optimized API par région	120	Non
API privées par compte et par région	600	Non
Associations d'accès aux noms de domaine par compte	100	Oui

Ressource ou opération	Quota par défaut	Peut être augmenté
Multi-level Mappages d'API par domaine	200	Non
Règles de routage par domaine	50	Oui
Longueur, en caractères, de l'URL pour une API optimisée pour les périphériques	8192	Non
Longueur, en caractères, de l'URL d'une API régionale	10240	Non
Longueur, en caractères, de l'URL d'une API privée	8192	Non
Longueur, en caractères, de la politique 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 Gateway-generated clients API par compte et par région	60	Oui
Certificats clients ACM par compte et par région	500	Non
Autorisateurs par API (AWS Lambda et Amazon Cognito)	10	Oui Pour augmenter ce quota, contactez le Centre de support AWS

Ressource ou opération	Quota par défaut	Peut être augmenté
Parties de la documentation par API	2000	Oui Pour augmenter ce quota, contactez le Centre de support AWS
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 d'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 pour les API régionales	50 millisecondes à 29 secondes pour tous les types d'intégration, y compris Lambda, proxy Lambda, HTTP, proxy HTTP et intégrations. AWS	Oui *
Délai d'intégration pour les API optimisées pour la périphérie	50 millisecondes à 29 secondes pour tous les types d'intégration, y compris Lambda, proxy Lambda, HTTP, proxy HTTP et intégrations. AWS	Non
Délai d'intégration pour les API privées	50 millisecondes à 29 secondes pour tous les types d'intégration, y compris Lambda, proxy Lambda, HTTP, proxy HTTP et intégrations. AWS	Oui *

Ressource ou opération	Quota par défaut	Peut être augmenté
Taille totale combinée de toutes les valeurs d'en-tête, y compris les noms d'en-tête , les valeurs, les terminaisons de ligne et les espaces blancs	20480 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
Nombre d'itérations dans une boucle `#foreach ... #end` dans les modèles de mappage	1 000	Non
Taille maximale du modèle de mappage	300 Ko	Non

Ressource ou opération	Quota par défaut	Peut être augmenté
Taille du modèle de journal d'accès	3 KO	Non
Longueur de l'ARN de la méthode	1 600 octets. Si votre méthode contient un paramètre de chemin et qu'un client utilise une valeur supérieure à la longueur de l'ARN, votre API renverra une réponse `414 Request URI too long`.	Non
Method-level paramètres de limitation pour une étape d'un plan d'utilisation	20	Oui Pour augmenter ce quota, contactez le Centre de support AWS
Taille de modèle par API	400 Ko	Non
Nombre de certificats d'un magasin de confiance	1 000 certificats jusqu'à une taille totale d'objet de 1 Mo.	Non
Délai d'expiration des connexions inactives	310 secondes	Non

Ressource ou opération	Quota par défaut	Peut être augmenté
Taille maximale du fichier de définition de l'API lors de l'utilisation de restapi:import ou de restapi:put	6 Mo	Non

* Vous ne pouvez pas définir le délai d'attente de l'intégration à moins de 50 millisecondes. Vous pouvez augmenter le délai d'intégration à plus de 29 secondes, mais cela peut nécessiter une réduction du quota d' Region-levelaccélération pour votre compte.

Bonnes pratiques pour réduire vos quotas

Les bonnes pratiques suivantes peuvent vous aider à réduire votre nombre actuel de ressources pour ne pas avoir à augmenter votre quota. Assurez-vous que ces suggestions fonctionnent pour l'architecture de votre API.

API par région

Pour réduire le nombre d'API par région, exportez toutes les API inutilisées, puis supprimez-les d'API Gateway. Pour de plus amples informations, veuillez consulter the section called "Exportation d'une API REST".

Étapes par API

Pour réduire le nombre d'étapes par API, divisez votre API en plusieurs API.

Ressources par API

Utilisez les chemins `{proxy+}` pour réduire le nombre de ressources. Pour de plus amples informations, veuillez consulter the section called "Configuration d'intégrations de proxy avec une ressource proxy".

Mappages d'API

Pour réduire le nombre de mappages d'API pour un nom de domaine personnalisé, utilisez des mappages d'API à un niveau tels que `/prod`. Les mappages d'API à un niveau ne sont pas pris en compte dans le quota de mappages des API.

Mécanismes d'autorisation par API

Pour réduire le nombre de mécanismes d'autorisation par API, réutilisez les mécanismes d'autorisation pour toutes les méthodes d'API.

Parties de la documentation par API

Pour réduire le nombre de parties de la documentation par API, utilisez `ignore=documentation` lorsque vous importez votre API. Pour de plus amples informations, veuillez consulter the section called "Importation de la documentation d'une API".

Vous pouvez également utiliser l'héritage de contenu pour autoriser certaines parties de la documentation à hériter du contenu d'une entité API d'une spécification plus générale. Pour de plus amples informations, veuillez consulter the section called "Représentation de la documentation de l'API dans API Gateway".

Longueur, en caractères, de la politique de ressources API Gateway

Pour réduire la longueur d'une politique de ressources, AWS WAFV2 utilisez-la pour protéger votre API. Pour de plus amples informations, veuillez consulter the section called "AWS WAF".

Si votre politique contient des adresses IP, vous pouvez également utiliser des plages plutôt que des valeurs spécifiques pour définir les adresses IP.

Plans d'utilisation par clé d'API

Pour réduire le nombre de plans d'utilisation par clé d'API, utilisez une clé d'API par plan d'utilisation et associez votre plan d'utilisation à plusieurs API. Nous vous déconseillons d'utiliser une même clé d'API pour plusieurs plans d'utilisation.

Quotas pour la configuration et l'exécution d'une API HTTP dans API Gateway

Les quotas suivants s'appliquent à la configuration et à l'exécution d'une API HTTP dans Amazon API Gateway. Si le quota est défini par API, il ne peut être augmenté que pour des

applications spécifiques APIs, et non pour toutes les API APIs d'un compte. Pour plus d'informations sur les quotas au niveau du compte, consultez Quotas et remarques importantes

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 mappages d'API. Cette limite n'inclut pas les mappages d'API à un niveau tels que `/prod`.	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

Ressource ou opération	Quota par défaut	Peut être augmenté
Entrée dans le CloudWatch journal Amazon Logs	1 Mo	Non
Mécanismes d'autorisation par API	10	Oui Pour augmenter ce quota, contactez le Centre de support AWS
Audiences par mécanisme d'autorisation	50	Non
Portées par route	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

Ressource ou opération	Quota par défaut	Peut être augmenté
Délai d’expiration du point de terminaison de découverte OpenID Connect	1500 ms	Non
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

Bonnes pratiques pour réduire vos quotas

Les bonnes pratiques suivantes peuvent vous aider à réduire votre nombre actuel de ressources pour ne pas avoir à augmenter votre quota. Assurez-vous que ces suggestions fonctionnent pour l'architecture de votre API.

APIs par région

Pour réduire le nombre de APIs par région, exportez les éléments non utilisés, APIs puis supprimez-les d'API Gateway. Pour de plus amples informations, veuillez consulter the section called "Exportation".

Étapes par API

Pour réduire le nombre d'étapes par API, divisez votre API en plusieurs étapes APIs.

Mécanismes d'autorisation par API

Pour réduire le nombre de mécanismes d'autorisation par API, réutilisez les mécanismes d'autorisation pour toutes les méthodes d'API.

Quotas pour la configuration des portails dans API Gateway

Les quotas suivants s'appliquent à la création de portails dans API Gateway. Pour de plus amples informations, veuillez consulter Portails API Gateway.

Ressource ou opération	Quota par défaut	Peut être augmenté
Portails par compte	15	Oui
Produits de portail par portail	200	Non
Produits du portail par compte	500	Non
Pages de point de terminaison REST	40	Oui

Ressource ou opération	Quota par défaut	Peut être augmenté
du produit par produit de portail		
Pages de produits par produit du portail	40	Oui
Taille du logo	200 Ko	Non
Taille de page de documentation par page de point de terminaison REST du produit	32 000 caractères	Non
Taille de page personnalisée pour les pages de produits	32 000 caractères	Non
Noms de domaine personnalisés par portail	1	Non
Autorisateurs par portail	1	Non

Quotas pour la configuration et l'exécution d'une passerelle WebSocket intégrée à l'API

Les quotas suivants s'appliquent à la configuration et à l'exécution d'une WebSocket API dans Amazon API Gateway. Si le quota est défini par API, il ne peut être augmenté que pour des

applications spécifiques APIs, et non pour toutes les API APIs d'un compte. Pour plus d'informations sur les quotas au niveau du compte, consultez Quotas et remarques importantes

Ressource ou opération	Quota par défaut	Peut être augmenté
Nouvelles connexions par seconde par compte (dans l'ensemble WebSocket APIs) par région	500	Oui
Connexions simultanées	Ne s'applique pas *	Non applicable
AWS Lambda autorisateurs par API	10	Oui Pour augmenter ce quota, contactez le Centre de support AWS
AWS Lambda taille du résultat de l'autorisateur	8 Ko	Non
Routes par API	300	Oui

Ressource ou opération	Quota par défaut	Peut être augmenté
Intégrations par API	300	Oui Pour augmenter ce quota, contactez le Centre de support AWS
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
Taille maximale du modèle de mappage	300 Ko	Non
Durée de connexion pour WebSocket l'API	2 heures	Non

Ressource ou opération	Quota par défaut	Peut être augmenté
Délai d'expiration de connexion inactive	10 minutes	Non
Longueur, en caractères, de l'URL d'une WebSocket API	4096	Non
Taille du modèle de journal d'accès	3 KO	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 de 32 Ko ou moins chacune. 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.

Remarques importantes concernant Amazon API Gateway

La section suivante détaille les remarques susceptibles d'avoir un impact sur votre façon d'utiliser API Gateway.

Rubriques

- Remarques importantes concernant Amazon API Gateway pour le protocole HTTP APIs
- Remarques importantes concernant Amazon API Gateway pour HTTP et WebSocket APIs
- Remarques importantes concernant Amazon API Gateway pour REST et WebSocket APIs

- Remarques importantes relatives à Amazon API Gateway pour WebSocket APIs
- Remarques importantes relatives à Amazon API Gateway pour REST APIs

Remarques importantes concernant Amazon API Gateway pour le protocole HTTP APIs

- Le protocole HTTP APIs traduit `X-Forwarded-*` les en-têtes entrants en `Forwarded` en-tête standard et ajoute l'adresse IP, l'hôte et le protocole de sortie.
- API Gateway ajoute l'en-tête Content-type à votre demande s'il n'y a pas de données utiles et si Content-Length est égal à 0.

Remarques importantes concernant Amazon API Gateway pour HTTP et WebSocket APIs

- Signature Version 4A n'est pas officiellement prise en charge par Amazon API Gateway pour HTTP et WebSocket APIs.

Remarques importantes concernant Amazon API Gateway pour REST et WebSocket APIs

- API Gateway ne prend pas en charge le partage d'un nom de domaine personnalisé entre REST et WebSocket APIs.
- 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 l'utilisateur CloudWatch .) 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 relatives à Amazon API Gateway pour WebSocket APIs

- 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 relatives à Amazon API Gateway pour REST APIs

- Le caractère pipe en texte brut (`|`) et l'accolade (`{` ou `}`) ne sont pas pris en charge pour les chaînes de requête des 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.
- REST APIs décode les paramètres de requête codés en URL avant de les transmettre aux intégrations du backend. Pour les paramètres de requête UTF-8, REST APIs décode les paramètres puis les transmet au format Unicode aux intégrations du backend. APIs Encodez en URL REST le caractère de pourcentage (`%`) avant de le transmettre 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 majeure 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`. Cependant, les authentifications OAuth 2 et HTTP Basic sont prises en charge via les autorisateurs Lambda ; la configuration OpenAPI est réalisée via les extensions des fournisseurs.
  - Le `deprecated` champ n'est pas pris en charge et est supprimé lors de l'exportation APIs.
  - 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 pas pris en charge dans aucun objet de schéma.
  - La balise `example` n'est pas prise en charge.
  - Le champ `nullable` n'est pas pris 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.
  - Pour OpenAPI 3.0, `oneOf`, `anyOf` et `allOf` ne peuvent pas utiliser `$ref` dans une définition du même schéma. Vous pouvez soit saisir directement votre schéma, soit définir une ressource de modèle API Gateway distincte. Pour de plus amples informations, veuillez consulter the section called "Création de modèles plus complexes".
  - `oneOf` n'est pas pris en charge pour la génération d'OpenAPI 2.0 ou du kit SDK.
  - Le champ `readOnly` n'est pas pris en charge.
  - `$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	Dropped/Passthrough/Passthrough
`Accept-Charset`	Transmettre	Transmettre
`Accept-Encoding`	Transmettre	Transmettre
`Authorization`	Transmettre *	Remappé
`Connection`	Passthrough/Passthrough/Dropped	Remappé
`Content-Encoding`	Passthrough/Dropped/Passthrough	Transmettre
`Content-Length`	Transmette (généré sur la base du corps)	Transmettre
`Content-MD5`	A abandonné	Remappé
`Content-Type`	Transmettre	Transmettre

Nom de l'en-tête	Requête (**http**/**http_proxy** /**lambda**)	Réponse (**http**/**http_proxy** /**lambda**)
`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
`Referer`	Transmettre	Transmettre
`Server`	A abandonné	Remappé écrasé
`TE`	A abandonné	A abandonné
`Transfer-Encoding`	Dropped/Dropped/Exception	A abandonné
`Trailer`	A abandonné	A abandonné

Nom de l'en-tête	Requête (**http**/**http_proxy** /**lambda**)	Réponse (**http**/**http_proxy** /**lambda**)
`Upgrade`	A abandonné	A abandonné
`User-Agent`	Transmettre	Remappé
`Via`	Dropped/Dropped/Passthrough	Passthrough/ Dropped/ Dropped
`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 API Java, Android et iOS générées par API Gateway, le JavaScript SDK SDKs 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 transmis 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 : 2 décembre 2025

Modification	Description	Date
Ajouter une API REST API Gateway en tant que cible pour Amazon Bedrock AgentCore Gateway	API Gateway prend désormais en charge l'ajout d'une API REST comme cible pour Amazon Bedrock AgentCore Gateway. Pour plus d'informations, consultez Ajouter une API REST API Gateway en tant que cible pour Amazon Bedrock AgentCore Gateway.	2 décembre 2025
Intégration privée avec Application Load Balancer	API Gateway prend désormais en charge les intégrations privées avec les équilibreurs de charge d'application pour REST. APIs Pour plus d'informations, consultez Intégrations privées pour REST. APIs	21 novembre 2025
Politiques de sécurité pour les noms de domaine REST APIs et personnalisés	API Gateway prend désormais en charge les politiques de sécurité TLS améliorées sur REST APIs et les noms de domaine personnalisés. Pour plus d'informations,	19 novembre 2025

	consultez la section Politiques de sécurité pour REST APIs dans API Gateway.	
Streaming de réponses pour REST APIs	API Gateway REST prend APIs désormais en charge le streaming progressif des charges utiles de réponse vers les clients au fur et à mesure de leur disponibilité. Pour plus d'informations, voir Diffuser la réponse d'intégration pour vos intégrations de proxy.	19 novembre 2025
Portails pour développeurs	API Gateway prend désormais en charge les portails, qui constituent un emplacement centralisé permettant à vos clients de découvrir et de tester vos produits APIs et ceux du portail représentant un service ou une fonctionnalité que vous souhaitez partager. Pour plus d'informations, consultez la section Portails API Gateway.	19 novembre 2025
SIGv4a pour REST APIs	API Gateway prend désormais en charge l'utilisation de Signature version 4a pour REST APIs. Pour plus d’informations, consultez le modèle d’autorisation API Gateway pour l’invocation d’une API.	19 août 2025

Règles de routage pour REST APIs	API Gateway prend désormais en charge les règles de routage pour REST APIs à l'aide de noms de domaine personnalisés. Vous pouvez acheminer les demandes entrantes de manière dynamique en fonction des valeurs d'en-tête HTTP et/ou des chemins de base d'URL. Pour plus d'informations, voir Envoyer du trafic vers vous APIs via votre nom de domaine personnalisé.	3 juin 2025
Ajout de la prise en charge des terminaux à double pile (IPv4 et IPv6)	API Gateway prend désormais en charge les points de terminaison à double pile pour les noms de domaine REST WebSocket APIs, HTTP et personnalisés. Pour plus d'informations, consultez Types d'adresses IP.	28 mars 2025
Noms de domaine personnalisés pour les particuliers APIs	Ajout de la prise en charge des noms de domaine personnalisés pour les particuliers APIs. Pour plus d'informations, consultez la section Noms de domaine personnalisés pour les domaines privés APIs.	21 novembre 2024

Ajout de la prise en charge de TLS 1.3	API Gateway prend désormais en charge le protocole TLS 1.3 sur les réseaux REST APIs APIs, HTTP et WebSocket APIs. Pour plus d'informations, consultez Choosing a security policy for your custom domain in API Gateway.	15 février 2024
Mises à jour de l' WebSocket API REST et de la console	Informations de console mises à jour pour REST APIs et WebSocket APIs.	10 décembre 2023
Mise à jour de la documentation	Informations conceptuelles mises à jour et création de nouveaux didacticiels pour les transformations de données et les sujets de validation des demandes pour API Gateway REST APIs. Pour en savoir plus, consultez Utilisation de la validation de demande dans API Gateway et Configuration de transformations de données pour les API REST.	22 juin 2023

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	Résumés des fonctionnalités de base mis à jour pour l'API REST et l'API APIs HTTP. Pour plus d'informations, consultez Choisir entre l'API REST et l'API HTTP APIs.	31 mai 2022
Mise à jour de la politique gérée	Ajout de la prise en charge de `acm:GetCertificate` pour la politique `AWSServiceRoleForAPIGateway` . Pour plus d'informations, consultez Utilisation de rôles liés au service pour API Gateway.	12 Juillet 2021
Mappage des paramètres pour HTTP APIs	Ajout de la prise en charge du mappage des paramètres pour HTTP APIs. 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 REST. APIs Pour plus d'informations, consultez 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 REST APIs et HTTP APIs. Pour plus d'informations, consultez 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 des AWS Lambda autorisateurs pour le protocole HTTP APIs. Pour plus d'informations, consultez la section Utilisation des AWS Lambda autorisateurs pour le protocole HTTP APIs.	9 septembre 2020
Intégrations des AWS services d'API HTTP	Ajout du support pour les intégrations de AWS services pour HTTP APIs. Pour plus d'informations, consultez la section Utilisation des intégrations de AWS services pour HTTP APIs.	20 août 2020

Domaines personnalisés contenant des caractères génériques des API HTTP	Ajout de la prise en charge des noms de domaine personnalisés avec caractères génériques pour HTTP APIs. 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 plus d'informations, consultez Utiliser le portail de développement sans serveur pour cataloguer votre API Gateway APIs.	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 de la prise en charge de l'exportation des définitions HTTP d'OpenAPI 3.0. APIs 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	HTTP a été publié APIs en version générale. Pour plus d'informations, consultez la section Utilisation du protocole HTTP APIs.	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
HTTP APIs	HTTP a été publié APIs en version bêta. Pour plus d'informations, consultez la section HTTP APIs.	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 Amazon Data Firehose	Ajout de la prise en charge d'Amazon Data Firehose en tant que destination des données de journalisation des accès. Pour plus d'informations, consultez Using Amazon Data Firehose as a Destination for API Gateway Access Logging.	15 octobre 2019
Alias Route53 pour invoquer le mode privé APIs	Ajout de la prise en charge des enregistrements DNS supplémentaires de l'alias Route53 pour appeler le mode privé. APIs 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 WebSocket APIs	Ajout de la prise en charge du contrôle d'accès basé sur des balises pour WebSocket APIs. 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 des versions de Transport Layer Security (TLS) pour APIs les déploiements 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
Politiques de point de terminaison VPC pour le secteur privé APIs	Ajout de la prise en charge de l'amélioration de la sécurité du privé APIs en associant des politiques de point de terminaison aux points de terminaison VPC d'interface. Pour plus d'informations, consultez Utiliser les politiques de point de terminaison VPC pour le mode privé APIs dans API Gateway.	4 juin 2019
Documentation mise à jour	Réécriture de Mise en route avec Amazon API Gateway. Déplacement des didacticiels dans Didacticiels Amazon API Gateway.	29 mai 2019
Contrôle d'accès basé sur des balises pour REST APIs	Ajout de la prise en charge du contrôle d'accès basé sur des balises pour REST. APIs Pour plus d'informations, consultez Utilisation des balises avec des politiques 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 ?, Didacticiel : Création d'une API avec l'intégration de proxy HTTP, Didacticiel : 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 d'un panneau d'administration et d'autres améliorations pour faciliter la publication APIs sur le portail des développeurs Amazon API Gateway. Pour plus d'informations, voir Utiliser un portail pour développeurs pour cataloguer votre APIs.	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 CloudFormation	L'API API Gateway V2 a été ajoutée à la référence du CloudFormation modèle. Pour de plus amples informations, veuillez consulter Référence des types de ressources Amazon API Gateway V2.	7 février 2019
Support pour WebSocket APIs	Ajout du support pour WebSocket APIs. 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 plus d'informations, consultez Utiliser un portail pour développeurs pour cataloguer votre API Gateway APIs.	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 de développement entièrement personnalisable sous forme d'application sans serveur que vous pouvez déployer pour publier votre API Gateway APIs. Pour plus d'informations, consultez Utiliser un portail pour développeurs pour cataloguer votre API Gateway APIs.	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 politiques 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 lors de leur transfert APIs 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 de APIs par compte. Limites de débit d'API accrues pour Create/Import/Deploy APIs. 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
Privé APIs	Ajout du support pour le mode privé APIs, que vous exposez via les points de terminaison VPC de l'interface. Le trafic vers votre réseau privé APIs 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	2 avril 2018

Modification	Description	Date de modification
	“Configuration d’un mécanisme d’autorisation Lambda entre comptes”, the section called “Didacticiel : création d’une API REST Gateway 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”.	
Politiques relatives aux ressources pour APIs	Utilisez les politiques de ressources API Gateway pour permettre aux utilisateurs d’un autre compte AWS d’accéder 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 “Utilisation des politiques 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 plus d’informations, consultez the section called “Configuration de 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 d’API provenant d’un mécanisme d’autorisation personnalisée	Renvoyez une clé d’API personnalisée d’un mécanisme d’autorisation personnalisée à API Gateway pour appliquer un plan d’utilisation pour les méthodes d’API qui nécessitent la clé. Pour de plus amples informations, veuillez consulter the section called “Choix d’une source de clé d’API dans API Gateway”.	19 décembre 2017

Modification	Description	Date de modification
Autorisation avec OAuth 2 portées	Activez l'autorisation d'invocation de méthode en utilisant OAuth 2 portées avec l'`COGNITO_USER_POOLS` autorisateur. 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 HTTP/HTTPS ressources d'un Amazon VPC depuis l'extérieur du VPC via une ressource. VpcLink Pour plus d’informations, consultez the section called “Didacticiel : création d’une API REST avec une intégration privée” et the section called “Configurez une 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 de plus amples informations, veuillez consulter the section called “Configuration du déploiement d’une version Canary”.	28 novembre 2017
Journalisation 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 dans API Gateway" 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, tel qu'une EC2 instance, s'exécute dans la même AWS région que celle où l'API est déployée. Comme pour une API optimisée pour la périphérie, vous pouvez créer un nom de domaine personnalisé pour une API régionale. Pour plus d'informations, consultez 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 Limitez les requêtes adressées à votre REST APIs pour améliorer le débit dans API Gateway.	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 Validation des demandes pour les API REST 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 vous référez ensuite à l'ARN du certificat lorsque vous définissez un nom de domaine personnalisé pour votre APIs. Pour de plus amples informations, veuillez consulter Nom de domaine personnalisé pour le REST public APIs dans API Gateway.	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 Vendez votre produit APIs 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 dans API Gateway.	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 politique IAM. Pour de plus amples informations, veuillez consulter Sortie d'un mécanisme d'autorisation 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 de plus amples informations, veuillez consulter Types de supports binaires pour les API REST dans API Gateway.	17 novembre 2016

Modification	Description	Date de modification
Activation de l'intégration de proxy à un backend HTTP ou Lambda via une ressource 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 avec 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 proxy.	20 septembre 2016
Étendre les offres de produits sélectionnés APIs dans API Gateway pour vos clients en proposant 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 Plans d'utilisation et clés d'API pour les API REST dans API Gateway.	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 aux 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 de plus amples informations, veuillez consulter 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 Didacticiel : création d'une API REST par l'importation d'un exemple et Didacticiel : création d'une API REST avec une intégration HTTP sans 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 Développement d'API REST à l'aide d'OpenAPI dans API Gateway.	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 Variables pour les transformations de données pour API Gateway.	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 Limitez les requêtes adressées à votre REST APIs pour améliorer le débit dans API Gateway.	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 politique 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 Développement d'API REST à l'aide d'OpenAPI dans API Gateway 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 Exemples de mappage de paramètres pour les API REST dans 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 Utilisation de variables d'étape pour une API REST dans API Gateway.	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 Partage des ressources entre origines multiples pour les API REST dans API Gateway.	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 Utiliser un Gateway-generated certificat d'API pour l'authentification du backend dans API Gateway.	22 septembre 2015
Intégration simulée 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, à l'encodage JavaScript URLs et aux chaînes de caractères. Pour de plus amples informations, veuillez consulter Variables pour les transformations de données pour API Gateway.	28 août 2015
Intégration de Swagger	Utilisez l'outil d'importation Swagger pour importer les GitHub définitions d'API Swagger dans Amazon API Gateway. En savoir plus sur Extensions OpenAPI pour API Gateway la création et le déploiement ainsi que sur APIs les méthodes à l'aide de l'outil d'importation. Avec l'outil d'importation Swagger, vous pouvez également mettre à jour les fichiers existants. APIs	21 juillet 2015
Référence de modèle de mappage	Découvrez le paramètre `$input` et ses fonctions dans Variables pour les transformations de données pour API Gateway.	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

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.