aws

개발자 가이드

Amazon API Gateway

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

Amazon API Gateway: 개발자 가이드

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

Amazon 상표 및 트레이드 드레스는 Amazon 외 제품 또는 서비스와 함께 사용되어서는 안되며, 고객에게 혼동을 일으키거나 Amazon 브랜드 이미지를 떨어뜨리고 폄하하는 방식으로 이용할 수 없습니다. Amazon이 소유하지 않은 기타 모든 상표는 Amazon과 제휴 관계이거나 관련이 있거나 후원 관계와 관계없이 해당 소유자의 자산입니다.

Table of Contents

Amazon API Gateway란 무엇입니까? 1
API Gateway의 아키텍처 2
API Gateway의 기능 3
API Gateway 사용 사례 3
API Gateway를 사용하여 REST API 생성 3
API Gateway를 사용하여 HTTP API 생성 4
API 게이트웨이를 사용하여 WebSocket API 생성 5
API Gateway를 누가 사용하는가? 5
API Gateway 액세스 6
서버를 사용하지 않는 AWS 인프라의 일부 7
Amazon API Gateway 시작 방법 7
API Gateway 개념 7
REST API와 HTTP API 중에서 선택 12
..... 12
[엔드포인트 유형] 12
보안 13
권한 부여 13
API 관리 14
개발 14
모니터링 15
통합 15
REST API 콘솔 시작하기 16
1단계: Lambda 함수 생성 17
2단계: REST API 생성 18
3단계: Lambda 프록시 통합 생성 18
4단계: API 배포 19
5단계: API 호출 19
(선택 사항) 6단계: 정리 20
사전 조건 22
AWS 계정에 등록 22
관리 사용자 생성 22
시작하기 24
1단계: Lambda 함수 생성 25
2단계: HTTP API 생성 25

3단계: API 테스트 26
(선택 사항) 4단계: 정리 27
다음 단계 28
자습서 및 워크숍 30
REST API 자습서 31
Lambda 통합을 사용하여 API 구축 31
자습서: 예제를 가져와 REST API 생성 54
HTTP 통합을 사용하여 API 빌드 62
자습서: 프라이빗 통합을 사용하여 API 빌드 76
자습서: AWS 통합을 사용하여 API 빌드 78
자습서: 세 가지 통합을 사용한 Calc API 84
자습서: API Gateway에서 REST API를 Amazon S3 프록시로 생성 113
자습서: REST API를 Amazon Kinesis 프록시로 생성 157
프라이빗 REST API 빌드 202
HTTP API 자습서 208
Lambda 및 DynamoDB를 사용한 CRUD API 209
Amazon ECS와의 프라이빗 통합 219
WebSocket API 자습서 226
WebSocket 채팅 앱 226
REST API 작업 232
개발 232
생성 및 구성 233
액세스 제어 278
통합 357
요청 검증 420
데이터 변환 453
게이트웨이 응답 532
CORS 542
이진 미디어 유형 555
Invoke 586
OpenAPI 620
게시 634
REST API 배포 635
사용자 지정 도메인 이름 677
최적화 715
캐시 설정 715

콘텐츠 인코딩 724
배포 729
사용량 계획 730
API 설명서 752
SDK 생성 814
API를 SaaS로 판매 840
보호 844
상호 TLS 845
클라이언트 인증서 851
AWS WAF 892
Throttling 894
프라이빗 API 896
모니터링 906
CloudWatch 지표 907
CloudWatch 로그 915
Firehose 921
X-Ray 922
HTTP API 작업 936
개발 936
HTTP API 만들기 937
라우팅 938
액세스 제어 941
통합 959
CORS 979
파라미터 매핑 982
OpenAPI 988
게시 998
Stages 999
HTTP API에 대한 보안 정책 1001
사용자 지정 도메인 이름 1003
보호 1010
Throttling 1010
상호 TLS 1011
모니터링 1017
지표 1018
로깅 1020

문제 해결 1029
Lambda 통합 1030
JWT 권한 부여자 1032
WebSocket API 작업 1034
API 정보 WebSocket 1034
연결된 사용자 및 클라이언트 앱 관리 1035
백엔드 통합 호출 1038
백엔드 서비스에서 연결된 클라이언트로 데이터 전송 1042
WebSocket 선택 표현식 1042
개발 1051
생성 및 구성 1051
라우팅 1053
액세스 제어 1061
통합 1069
요청 검증 1078
데이터 변환 1081
이진 미디어 유형 1092
Invoke 1092
게시 1095
Stages 1096
WebSocket API 배포 1098
WebSocket API에 대한 보안 정책 1101
사용자 지정 도메인 이름 1102
보호 1107
리전별 계정 수준 조절 1108
라우팅 수준 스로틀 1108
모니터링 1108
지표 1109
로깅 1111
API Gateway ARN 1119
HTTP API 및 WebSocket API 리소스 1119
REST API 리소스 1122
`execute-api`(HTTP API, WebSocket API 및 REST API) 1127
OpenAPI 확장 1128
`x-amazon-apigateway-any-method` 1129
x-amazon-apigateway-any-메서드 예제 1130

x-amazon-apigateway-cors 1131
x-amazon-apigateway-cors 예 1131
x-amazon-apigateway-api-key-source 1132
x-amazon-apigateway-api-키-소스 예제 1132
x-amazon-apigateway-auth 1133
x-amazon-apigateway-auth 예 1134
x-amazon-apigateway-authorizer 1134
x-amazon-apigateway-authorizer REST API의 예시 1137
x-amazon-apigateway-authorizer HTTP API에 대한 예제 1141
x-amazon-apigateway-authtype 1142
x-amazon-apigateway-authtype 예시 1143
다음 사항도 참조하세요. 1145
x-amazon-apigateway-binary-미디어 타입 1145
x-amazon-apigateway-binary-미디어 유형 예제 1145
x-amazon-apigateway-documentation 1145
x-amazon-apigateway-documentation 예 1146
x-amazon-apigateway-endpoint-구성 1147
x-amazon-apigateway-endpoint-구성 예제 1147
x-amazon-apigateway-gateway-응답 1148
x-amazon-apigateway-gateway- 응답 예제 1148
x-amazon-apigateway-gateway- 응답. 게이트웨이 응답 1149
x-amazon-apigateway-gateway- 응답. 게이트웨이 응답 예제 1149
x-amazon-apigateway-gateway- 응답. 응답 파라미터 1150
x-amazon-apigateway-gateway-responses.responseParameter 예제 1150
x-amazon-apigateway-gateway- 응답. 응답 템플릿 1151
x-amazon-apigateway-gateway- response. 응답 템플릿 예제 1151
x-amazon-apigateway-importexport-버전 1152
x-amazon-apigateway-importexport-버전 예제 1152
x-amazon-apigateway-integration 1152
x-amazon-apigateway-integration 예제 1157
x-amazon-apigateway-integrations 1158
x-amazon-apigateway-integrations 예 1159
x-amazon-apigateway-integration. 요청 템플릿 1161
x-amazon-apigateway-integration. 요청 템플릿 예제 1161
x-amazon-apigateway-integration. 요청 매개변수 1161
x-amazon-apigateway-integration.requestParameters 예제 1163

x-amazon-apigateway-integration. 응답 1163
x-amazon-apigateway-integration.responses 예제 1164
x-amazon-apigateway-integration. 응답 1165
x-amazon-apigateway-integration.response 예제 1166
x-amazon-apigateway-integration. 응답 템플릿 1167
x-amazon-apigateway-integration. 응답 템플릿 예제 1167
x-amazon-apigateway-integration. 응답 매개변수 1167
x-amazon-apigateway-integration.responseParameters 예제 1168
x-amazon-apigateway-integration.tlsConfig 1168
x-amazon-apigateway-integration.tlsConfig 예제 1170
x-amazon-apigateway-minimum-압축 크기 1171
x-amazon-apigateway-minimum-압축 크기 예제 1171
x-amazon-apigateway-policy 1171
x-amazon-apigateway-policy 예제 1171
x-amazon-apigateway-request-검증자 1172
x-amazon-apigateway-request-validator 예제 1172
x-amazon-apigateway-request-검증자 1173
x-amazon-apigateway-request-validators 예제 1174
x-amazon-apigateway-request- 유효성 검사기. 유효성 검사기 요청 1174
x-amazon-apigateway-request-validators.requestValidator 예제 1175
x-amazon-apigateway-tag-값 1175
x-amazon-apigateway-tag-value 예제 1175
보안 1177
데이터 보호 1178
데이터 암호화 1178
인터네트워크 트래픽 개인 정보 보호 1179
ID 및 액세스 관리 1180
대상 1180
자격 증명을 통한 인증 1181
정책을 사용한 액세스 관리 1183
Amazon API Gateway에서 IAM을 사용하는 방식 1186
자격 증명 기반 정책 예제 1191
리소스 기반 정책 예제 1199
문제 해결 1199
서비스 연결 역할 사용 1201
로깅 및 모니터링 1205

AWS CloudTrail 작업 1206
AWS Config 작업 1209
규정 준수 확인 1212
복원성 1213
인프라 보안 1214
구성 및 취약성 분석 1214
모범 사례 1215
태그 지정 1217
태그를 지정할 수 있는 API Gateway 리소스 1217
Amazon API Gateway V1 API의 태그 상속 1219
태그 제한 및 사용 규칙 1220
속성 기반 액세스 제어 1220
리소스 태그 기반 작업 한도 지정 1221
리소스 태그 기반 작업 허용 1222
태그 지정 작업 거부 1223
태그 지정 작업 허용 1223
API 참조 1225
할당량 및 중요 정보 1226
리전당 API Gateway 계정 수준 할당량 1226
HTTP API 할당량 1227
..... 1227
WebSocket API 구성 및 실행에 대한 API Gateway 할당량 1229
REST API 구성 및 실행에 대한 API Gateway 할당량 1230
API 생성, 배포 및 관리를 위한 API Gateway 할당량 1233
중요 정보 1236
REST 및 WebSocket API의 중요 정보 1236
WebSocket API의 중요 정보 1236
REST API의 중요 정보 1237
문서 기록 1243
이전 업데이트 1252
AWS 용어집 1261
..... mcclxii

Amazon API Gateway란 무엇입니까?

> **Note**
>
> 이제 재설계된 API Gateway 콘솔 환경을 사용할 수 있습니다. 콘솔을 사용하여 REST API를 생성하는 방법에 대한 튜토리얼은 REST API 콘솔 시작하기 섹션을 참조하세요.

Amazon API Gateway는 규모와 관계없이 REST 및 WebSocket API를 생성, 게시, 유지, 모니터링 및 보호하기 위한 AWS 서비스입니다. API 개발자는 AWS 또는 다른 웹 서비스를 비롯해 AWS 클라우드에 저장된 데이터에 액세스하는 API를 생성할 수 있습니다. API Gateway API 개발자는 자체 클라이언트 애플리케이션에서 사용할 API를 생성할 수 있습니다. 또는 타사 앱 개발자가 API를 사용하도록 제공할 수도 있습니다. 자세한 내용은 the section called “API Gateway를 누가 사용하는가?” 단원을 참조하세요.

API Gateway는 다음과 같은 RESTful API를 생성합니다.

- HTTP 기반.
- 상태 비저장 클라이언트-서버 통신을 활성화합니다.
- 표준 HTTP 메서드(예: GET, POST, PUT, PATCH, DELETE)를 구현합니다.

API Gateway REST API 및 HTTP API에 대한 자세한 내용은, the section called “REST API와 HTTP API 중에서 선택 ”, HTTP API 작업, the section called “API Gateway를 사용하여 REST API 생성” 및 the section called “생성 및 구성” 단원을 참조하세요.

API Gateway는 다음과 같은 WebSocket API를 생성합니다.

- 클라이언트와 서버 간에 상태를 저장하는 전이중 통신을 지원하는 WebSocket 프로토콜 준수.
- 수신 메시지를 메시지 콘텐츠에 따라 라우팅.

API Gateway WebSocket API에 대한 자세한 내용은 the section called “API 게이트웨이를 사용하여 WebSocket API 생성” 및 the section called “API 정보 WebSocket ” 단원을 참조하세요.

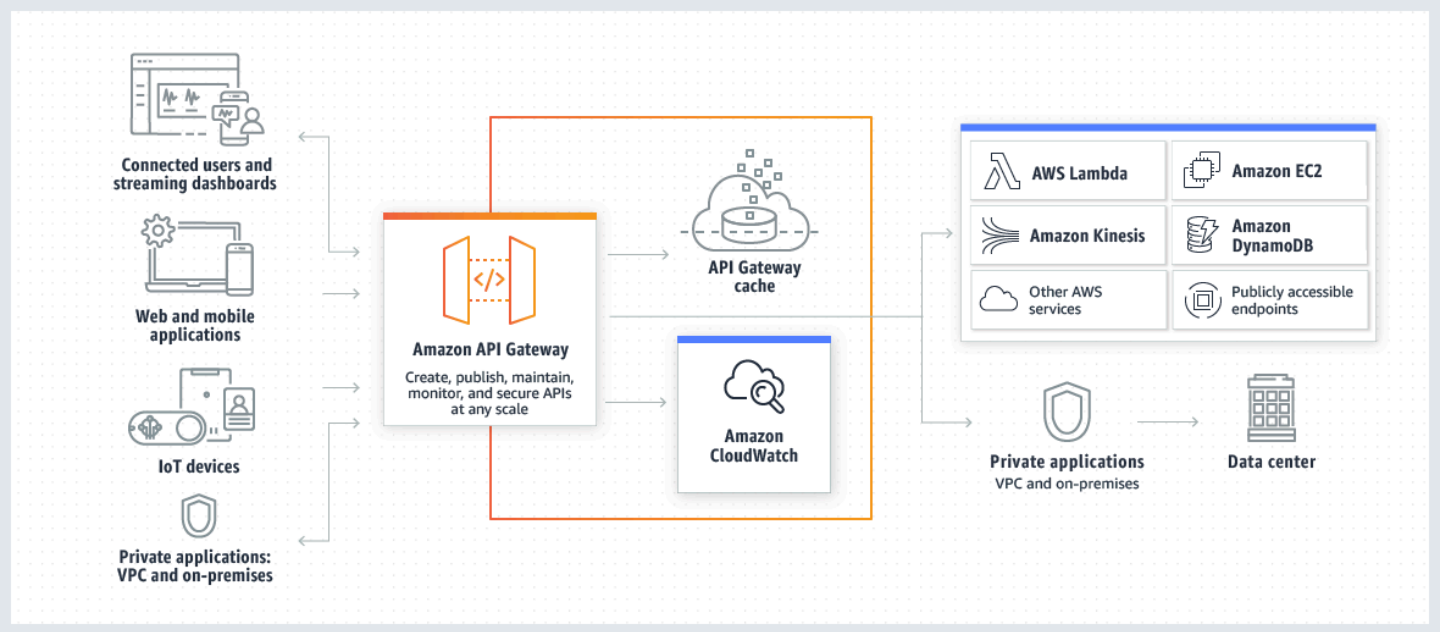
주제

- API Gateway의 아키텍처
- API Gateway의 기능

- API Gateway 사용 사례
- API Gateway 액세스
- 서버를 사용하지 않는 AWS 인프라의 일부
- Amazon API Gateway 시작 방법
- Amazon API Gateway 개념
- REST API와 HTTP API 중에서 선택
- REST API 콘솔 시작하기

API Gateway의 아키텍처

다음 다이어그램은 API Gateway 아키텍처를 보여줍니다.



이 다이어그램은 Amazon API Gateway에서 빌드한 API가 귀사 또는 개발자 고객에게 AWS 서버리스 애플리케이션 빌드를 위한 통합적이고 일관된 개발자 환경을 제공하는 방법을 보여줍니다. API Gateway는 최대 수십만 개의 동시 API 호출 허용 및 처리에 관련된 모든 작업을 다룹니다. 여기에는 트래픽 관리, 권한 부여 및 액세스 제어, 모니터링, API 버전 관리가 포함됩니다.

API Gateway는 애플리케이션이 Amazon Elastic Compute Cloud(Amazon EC2)에서 실행 중인 워크로드, AWS Lambda에서 실행 중인 코드, 웹 애플리케이션, 실시간 통신 애플리케이션과 같은 백엔드 서비스에서 데이터, 비즈니스 로직 또는 기능에 액세스할 수 있게 해주는 "정문" 역할을 합니다.

API Gateway의 기능

Amazon API Gateway는 다음과 같은 기능을 제공합니다.

- 상태 저장(WebSocket) 및 상태 비저장(HTTP 및 REST) API에 대한 지원.
- 강력하고 유연한 인증 메커니즘(예: AWS Identity and Access Management 정책, Lambda 권한 부여자 함수, Amazon Cognito 사용자 풀 등).
- 변경 사항을 안전하게 롤아웃하기 위한 Canary 릴리스 배포.
- API 사용 및 API 변경에 대한 CloudTrail 로깅 및 모니터링.
- 경보 설정 기능을 포함한 CloudWatch 액세스 로깅 및 실행 로깅. 자세한 내용은 the section called “CloudWatch 지표” 및 the section called “지표” 단원을 참조하십시오.
- AWS CloudFormation 템플릿을 사용하여 API 생성을 활성화할 수 있는 기능. 자세한 내용은 Amazon API Gateway 리소스 유형 참조 및 Amazon API Gateway V2 리소스 유형 참조를 참조하세요.
- 사용자 지정 도메인 이름 지원.
- 일반적인 웹 익스플로잇으로부터 API를 보호하기 위해 AWS WAF와 통합.
- 성능 지연 시간 파악 및 학습을 위해 AWS X-Ray와 통합.

API Gateway 기능의 전체 목록은 문서 기록 단원을 참조하세요.

API Gateway 사용 사례

주제

- API Gateway를 사용하여 REST API 생성
- API Gateway를 사용하여 HTTP API 생성
- API 게이트웨이를 사용하여 WebSocket API 생성
- API Gateway를 누가 사용하는가?

API Gateway를 사용하여 REST API 생성

API Gateway REST API는 리소스와 메서드로 구성됩니다. 리소스란 앱이 리소스 경로를 통해 액세스할 수 있는 논리적 엔터티입니다. 메서드는 API 사용자가 제출한 REST API 요청 및 사용자에게 반환되는 응답에 해당합니다.

예를 들어 /incomes는 앱 사용자의 소득을 나타내는 리소스의 경로가 될 수 있습니다. 리소스는 GET, POST, PUT, PATCH, DELETE 등 적절한 HTTP 동사로 정의된 작업을 하나 이상 가질 수 있습니다. 리소스 경로와 작업의 조합으로 API의 메서드를 식별합니다. 예를 들어 POST /incomes 메서드는 호출자가 획득한 소득을 추가하고 GET /expenses 메서드는 호출자로 인해 발생한 것으로 보고된 지출을 쿼리할 수 있습니다.

앱은 백엔드에서 요청된 데이터를 저장하고 가져오는 위치를 알 필요가 없습니다. API Gateway REST API에서 프런트엔드는 메서드 요청 및 메서드 응답으로 캡슐화됩니다. API는 integration requests(통합 요청) 및 integration responses(통합 응답)을 이용해 백엔드와 통신합니다.

예를 들어 DynamoDB가 백엔드일 때, API 개발자는 수신되는 메서드 요청을 선택한 백엔드로 전달하도록 통합 요청을 설정합니다. 이 설정에는 적절한 DynamoDB 작업, 필요한 IAM 역할 및 정책, 필요한 입력 데이터 변환의 사양이 포함됩니다. 백엔드는 API Gateway에 통합 응답으로서 결과를 반환합니다.

통합 응답을 클라이언트에 대한 (해당 HTTP 상태 코드의) 적절한 메서드 응답으로 라우팅하려면, 통합에서 메서드로 필요한 응답 파라미터를 매핑하도록 통합 응답을 구성하면 됩니다. 그런 다음, 필요에 따라 백엔드의 출력 데이터 형식을 프런트엔드의 형식으로 변환합니다. API Gateway를 사용하면 페이로드의 스키마나 모델을 정의하여 본문 매핑 템플릿을 손쉽게 설정할 수 있습니다.

API Gateway는 다음과 같은 REST API 관리 기능을 제공합니다.

- OpenAPI에 대한 API Gateway 확장을 사용하여 SDK 생성 및 API 문서 생성 지원
- HTTP 조절 요청

API Gateway를 사용하여 HTTP API 생성

HTTP API를 사용하면 REST API보다 지연 시간이 짧고 비용이 저렴한 RESTful API를 생성할 수 있습니다.

HTTP API를 사용하여 AWS Lambda 함수 또는 공개 라우팅이 가능한 HTTP 엔드포인트에 요청을 전송할 수 있습니다.

예를 들어 백엔드의 Lambda 함수와 통합되는 HTTP API를 생성할 수 있습니다. 클라이언트가 API를 호출하면 API Gateway는 Lambda 함수에 요청을 전송하고 함수의 응답을 클라이언트에 반환합니다.

HTTP API는 OpenID Connect 및 OAuth 2.0 권한 부여를 지원합니다. 여기에서는 CORS(Cross-Origin Resource Sharing) 및 자동 배포가 기본적으로 지원됩니다.

자세한 내용은 the section called “REST API와 HTTP API 중에서 선택 ” 단원을 참조하십시오.

API 게이트웨이를 사용하여 WebSocket API 생성

WebSocket API에서 클라이언트와 서버는 언제든 서로 메시지를 보낼 수 있습니다. 백엔드 서버는 복잡한 폴링 메커니즘을 구현할 필요 없이 연결된 사용자 및 장치로 데이터를 쉽게 푸시할 수 있습니다.

예를 들어, API Gateway WebSocket API 및 AWS Lambda를 사용하여 서버리스 애플리케이션을 구축하면 채팅룸의 개별 사용자 또는 사용자 그룹과 메시지를 주고 받을 수 있습니다. 또는 메시지 내용을 기반으로 AWS Lambda, Amazon Kinesis 또는 HTTP 엔드포인트 같은 백엔드 서비스를 호출할 수 있습니다.

API Gateway WebSocket API를 사용하면 연결이나 대용량 데이터 교환을 관리하기 위해 서버를 프로비저닝하거나 관리하지 않고도 안전한 실시간 통신 애플리케이션을 구축할 수 있습니다. 목표 사용 사례에는 다음과 같은 실시간 애플리케이션이 포함됩니다.

- 채팅 애플리케이션
- 주식 시세 표시기와 같은 실시간 대시 보드
- 실시간 경고 및 알림

API Gateway는 다음과 같은 WebSocket API 관리 기능을 제공합니다.

- 연결 및 메시지 모니터링 및 제한
- AWS X-Ray를 사용하여 API를 통해 백엔드 서비스로 이동하는 메시지 추적
- HTTP/HTTPS 엔드포인트와 쉽게 통합

API Gateway를 누가 사용하는가?

API Gateway를 사용하는 개발자는 API 개발자와 앱 개발자, 두 가지 유형으로 나뉩니다.

API 개발자는 API를 생성하고 배포하여 API Gateway에서 필요한 기능이 구현되도록 합니다. API 개발자는 AWS 계정 내에서 API를 소유한 사용자여야 합니다.

앱 개발자는 API Gateway에서 API 개발자가 생성한 WebSocket 또는 REST API를 호출하여 AWS 서비스를 호출하기에 효과적인 애플리케이션을 구축합니다.

앱 개발자는 API 개발자의 고객입니다. API에 IAM 권한이 필요하지 않거나 Amazon Cognito 사용자 풀 자격 증명 연동에서 지원하는 타사 연동 자격 증명 공급자를 통해 사용자 권한 부여를 지원하는 경우, 앱 개발자는 AWS 계정을 보유할 필요가 없습니다. 이러한 자격 증명 공급자에는 Amazon, Amazon Cognito 사용자 풀, Facebook 및 Google이 포함됩니다.

API Gateway API 생성 및 관리

API 개발자는 `apigateway`라는 API 관리를 위한 API Gateway 서비스 구성 요소와 함께 작동하여 API를 생성, 구성 및 배포합니다.

API 개발자는 API Gateway 시작하기에 설명된 대로 API Gateway 콘솔을 사용하거나 API 참조을 호출하여 API를 생성하고 관리할 수 있습니다. 이 API를 호출하는 몇 가지 방법이 있습니다. 여기에는 AWS Command Line Interface(AWS CLI)를 사용하거나 AWS SDK를 사용하는 것이 포함됩니다. AWS CloudFormation 템플릿 또는 OpenAPI에 대한 API Gateway 확장 작업(REST API 및 HTTP API의 경우)로 API 생성을 지원할 수도 있습니다.

연결된 제어 서비스 엔드포인트를 비롯하여 API Gateway를 이용할 수 있는 리전 목록은 Amazon API Gateway 엔드포인트 및 할당량을 참조하세요.

API Gateway API 호출

앱 개발자는 `execute-api`라는 API 실행을 위한 API Gateway 서비스 구성 요소와 함께 작동하여 API Gateway에서 생성 또는 배포된 API를 호출합니다. 기본 프로그래밍 엔터티는 생성된 API에 의해 제공됩니다. 이러한 API를 호출하는 몇 가지 방법이 있습니다. 자세한 내용은 Amazon API Gateway에서 REST API 호출 및 WebSocket API 호출 단원을 참조하십시오.

API Gateway 액세스

Amazon API Gateway에 액세스하는 방법은 다음과 같습니다.

- AWS Management Console - AWS Management Console은 API를 생성하고 관리할 수 있는 웹 인터페이스를 제공합니다. 사전 조건의 단계를 완료한 후 https://console.aws.amazon.com/apigateway에서 API Gateway 콘솔에 액세스할 수 있습니다.
- AWS SDK – AWS가 SDK를 제공하는 프로그래밍 언어를 사용하는 경우 SDK를 사용하여 API Gateway에 액세스할 수 있습니다. SDK는 인증을 단순화하고, 개발 환경에 쉽게 통합되며, API Gateway 명령에 액세스할 수 있도록 합니다. 자세한 내용은 Amazon Web Services용 도구를 참조하십시오.
- API Gateway V1 및 V2 API - SDK가 제공되지 않는 프로그래밍 언어를 사용할 경우 Amazon API Gateway 버전 1 API 참조 정보 및 Amazon API Gateway 버전 2 API 참조 정보를 확인하세요.
- AWS Command Line Interface - 자세한 내용은 AWS Command Line Interface 사용 설명서의 AWS Command Line Interface 설정을 참조하세요.
- AWS Tools for Windows PowerShell - 자세한 내용은 AWS Tools for Windows PowerShell 사용 설명서의 AWS Tools for Windows PowerShell 설정을 참조하세요.

서버를 사용하지 않는 AWS 인프라의 일부

API Gateway는 AWS Lambda와 함께 AWS 서버리스 인프라의 앱 페이싱(app-facing) 부분을 구성합니다. 서버리스를 시작하는 방법에 대한 자세한 내용은 서버리스 개발자 안내서를 참조하세요.

앱에서 공개적으로 사용할 수 있는 AWS 서비스를 호출하려면 Lambda를 사용해 필요한 서비스와 상호 작용하고, API Gateway에서 API 메서드를 통해 Lambda 함수를 제공할 수 있습니다. AWS Lambda은 고가용성 컴퓨팅 인프라에서 코드를 실행합니다. 컴퓨팅 리소스의 필요한 실행 및 관리를 수행합니다. 서버리스 애플리케이션을 활성화하기 위해 API Gateway는 AWS Lambda 및 HTTP 엔드포인트와 간소한 프록시 통합을 지원합니다.

Amazon API Gateway 시작 방법

Amazon API 게이트웨이에 대한 소개는 다음을 참조하세요.

- 시작하기에서 HTTP API 생성을 위한 연습을 제공합니다.
- 서버리스 랜드에서 교육용 비디오를 제공합니다.
- 유용하고 간단한 API 숏클립은 짧은 교육용 비디오 시리즈입니다.

Amazon API Gateway 개념

API 게이트웨이

API Gateway는 다음을 지원하는 AWS 서비스입니다.

- 백엔드 HTTP 엔드포인트, AWS Lambda 함수 또는 기타 AWS 서비스를 노출하기 위한 RESTful 애플리케이션 프로그래밍 인터페이스(API)의 생성, 배포 및 관리.
- AWS Lambda 함수 또는 기타 AWS 서비스를 노출하기 위한 WebSocket API의 생성, 배포 및 관리.
- 프런트 엔드 HTTP 및 WebSocket 엔드포인트를 통해 노출된 API 메서드 호출.

API Gateway REST API

백엔드 HTTP 엔드포인트, Lambda 함수 또는 기타 AWS 서비스와 통합되어 있는 HTTP 리소스와 메서드의 모음입니다. 이 모음을 하나 이상의 스테이지로 배포할 수 있습니다. 일반적으로 API 리소스는 애플리케이션 로직에 따른 리소스 트리로 정리되어 있습니다. 각 API 리소스는 API Gateway에서 지원하는 전용 HTTP 동사가 있는 API 메서드를 하나 이상 표시할 수 있습니다. 자세한 정보는 the section called “REST API와 HTTP API 중에서 선택 ”을 참조하십시오.

API Gateway HTTP API

백엔드 HTTP 엔드포인트 또는 Lambda 함수와 통합된 라우팅 및 메서드의 모음입니다. 이 모음을 하나 이상의 스테이지로 배포할 수 있습니다. 각 라우팅은 API Gateway에서 지원되는 고유의 HTTP 동사를 가진 API 메서드를 하나 이상 노출할 수 있습니다. 자세한 정보는 the section called "REST API와 HTTP API 중에서 선택 "을 참조하십시오.

API Gateway WebSocket API

백엔드 HTTP 엔드포인트, Lambda 함수 또는 기타 AWS 서비스와 통합되어 있는 WebSocket 경로와 경로 키의 모음입니다. 이 모음을 하나 이상의 스테이지로 배포할 수 있습니다. API 메서드는 프런트 엔드 WebSocket 연결을 통해 호출되며, 이 엔드포인트를 등록된 사용자 지정 도메인 이름과 연결할 수 있습니다.

API 배포

API Gateway API의 특정 시점 스냅샷. 클라이언트가 사용할 수 있게 하려면 배포가 하나 이상의 API 단계와 연결되어 있어야 합니다.

API 개발자

API Gateway 배포를 소유한 AWS 계정(예: 프로그래밍 액세스도 지원하는 서비스 공급업체)입니다.

API 엔드포인트

특정 리전에 배포되는 API Gateway의 API 호스트 이름. 호스트 이름은 `{api-id}.execute-api.{region}.amazonaws.com` 형식을 갖고 있습니다. 다음 API 엔드포인트 유형이 지원됩니다.

- 엣지 최적화 API 엔드포인트
- 프라이빗 API 엔드포인트
- 리전 API 엔드포인트

API 키

API Gateway가 REST 또는 WebSocket API를 사용하는 앱 개발자를 식별하는 데 사용하는 영숫자 문자열입니다. API Gateway가 사용자 대신 API 키를 생성하거나, 사용자가 CSV 파일에서 API 키를 가져올 수 있습니다. API 키와 Lambda 권한 부여자 또는 사용량 계획을 함께 사용하여 API 액세스를 제어할 수 있습니다.

API 엔드포인트를 참조하십시오.

API 소유자

API 개발자 단원을 참조하십시오.

API 단계

API의 수명 주기 상태에 대한 논리적 참조(예: 'dev', 'prod', 'beta', 'v2')입니다. API 단계는 API ID와 단계 이름으로 식별됩니다.

앱 개발자

API 개발자가 배포한 API와 상호 작용하며 AWS 계정이 있거나 없을 수도 있는 앱 생성자. 앱 개발자는 귀하의 고객입니다. 앱 개발자는 일반적으로 API 키로 식별됩니다.

콜백 URL

새 클라이언트가 WebSocket 연결을 통해 연결되면 API Gateway에서 통합을 호출하여 클라이언트의 콜백 URL을 저장할 수 있습니다. 이 콜백 URL을 사용하여 백엔드 시스템에서 클라이언트로 메시지를 보낼 수 있습니다.

개발자 포털

고객이 API 제품(API 게이트웨이 사용량 계획)을 등록, 검색 및 구독하고, API 키를 관리하고, API에 대한 사용 지표를 볼 수 있게 해주는 애플리케이션입니다.

엣지 최적화 API 엔드포인트

일반적으로 AWS 리전 전체에서 클라이언트 액세스가 용이하도록 CloudFront 배포를 사용할 때 지정된 리전에 배포되는 API Gateway API의 기본 호스트 이름입니다. API 요청은 지리적으로 다양한 클라이언트를 위해 연결 시간을 일반적으로 향상하는 가장 가까운 CloudFront POP(Point of Presence)로 라우팅됩니다.

API 엔드포인트를 참조하십시오.

통합 요청

API Gateway에서 WebSocket API 라우팅 또는 REST API 메서드의 내부 인터페이스로, 여기에서 라우팅 요청의 본문 또는 메서드의 본문 및 파라미터를 백엔드에서 요구하는 형식으로 매핑합니다.

통합 응답

API Gateway에서 WebSocket API 라우팅 또는 REST API 메서드의 내부 인터페이스로, 백엔드에서 클라이언트 앱으로 반환되는 응답 형식으로 받은 상태 코드, 헤더, 페이로드를 여기에서 매핑합니다.

매핑 템플릿

VTL(Velocity Template Language)에 나온 스크립트로, 프런트엔드 데이터 형식으로 된 요청 본문을 백엔드 데이터 형식으로 변환하거나 백엔드 데이터 형식으로 된 응답 본문을 프런트엔드 데이터 형식으로 변환합니다. 통합 요청 또는 통합 응답에 매핑 템플릿이 지정될 수 있습니다. 이들은 실행 시간에 컨텍스트 및 스테이지 변수로 제공되는 데이터를 참조할 수 있습니다.

매핑은 요청에 대해 클라이언트에서 백엔드로 통합을 통해 헤더나 본문을 전달하는 자격 증명 변환만큼 간단할 수 있습니다. 백엔드에서 클라이언트로 페이로드가 전달되는 응답의 경우도 마찬가지입니다.

메서드 요청

API Gateway에 있는 API 메서드의 퍼블릭 인터페이스로, 파라미터와 본문을 정의합니다. 앱 개발자가 API를 통해 백엔드에 액세스하려면 반드시 이 파라미터와 본문을 요청에 포함해 보내야 합니다.

메서드 응답

REST API의 퍼블릭 인터페이스로, 앱 개발자가 API의 응답에 기대하는 본문 모델, 헤더, 상태 코드를 여기서 정의합니다.

모의 통합

모의 통합에서는 통합 백엔드가 없어도 API Gateway에서 직접 API 응답이 생성됩니다. API 개발자는 API Gateway가 모의 통합 요청에 응답하는 방법을 결정합니다. 이를 위해, 메서드의 통합 요청 및 통합 응답을 구성하여 정해진 상태 코드와 응답을 연결합니다.

모델

요청 또는 응답 페이로드의 데이터 구조를 지정하는 데이터 스키마입니다. 모델은 API에서 강력한 형식의 SDK를 생성하는 데 필요합니다. 페이로드를 확인하는 데에도 사용됩니다. 모델은 샘플 매핑 템플릿을 생성해 프로덕션 매핑 템플릿 생성을 시작하기에 편리한 도구입니다. 있으면 유용하기는 하지만 매핑 템플릿을 생성할 때 모델은 필요하지 않습니다.

프라이빗 API

프라이빗 API 엔드포인트를 참조하십시오.

프라이빗 API 엔드포인트

인터페이스 VPC 엔드포인트를 통해 노출되고 클라이언트가 VPC 내부의 프라이빗 API 리소스에 안전하게 액세스할 수 있도록 해주는 API 엔드포인트입니다. 프라이빗 API는 퍼블릭 인터넷과 격리되어 있으며, 액세스 권한이 있는 API Gateway의 VPC 종단점을 사용해야만 액세스할 수 있습니다.

프라이빗 통합

리소스를 퍼블릭 인터넷에 노출하지 않고, 클라이언트가 프라이빗 REST API 엔드포인트를 통해 고객 VPC 내부의 리소스에 액세스할 수 있는 API Gateway 통합 유형입니다.

프록시 통합

간소화된 API Gateway 통합 구성 HTTP 프록시 통합 또는 Lambda 프록시 통합으로 프록시 통합을 설정할 수 있습니다.

HTTP 프록시 통합에서 API Gateway는 프런트 엔드와 HTTP 백엔드 간에 전체 요청 및 응답을 전달합니다. Lambda 프록시 통합에서 API Gateway는 백엔드 Lambda 함수에 대한 입력으로 전체 요청을 전송합니다. 그런 다음 API Gateway가 Lambda 함수 출력을 프런트엔드 HTTP 응답으로 변환합니다.

REST API에서 프록시 통합은 프록시 리소스에서 가장 많이 사용되며, 복잡한 경로 변수(예: `{proxy+}`)와 catch-all ANY 메서드로 표시됩니다.

빠른 생성

빠른 생성을 사용하여 HTTP API 생성을 간소화할 수 있습니다. 빠른 생성은 Lambda 는 HTTP 통합, 기본 catch-all 라우팅, 변경 사항을 자동 배포하도록 구성된 기본 스테이지를 사용하여 API를 생성합니다. 자세한 내용은 the section called “AWS CLI를 사용하여 HTTP API 생성” 단원을 참조하세요.

리전 API 엔드포인트

특정 리전에 배포되고 동일한 AWS 리전에서 EC2 인스턴스와 같은 클라이언트에 서비스를 제공하는 API의 호스트 이름입니다. API 요청은 CloudFront 배포판을 거치지 않고 리전별 API Gateway로 직접 지정됩니다. 리전 내 요청의 경우 리전 엔드포인트는 불필요한 왕복을 CloudFront 배포로 우회합니다.

추가로 리전 엔드포인트에 대한 지연 시간 기반 라우팅을 API 개발자가 동일한 리전 API 엔드포인트 구성을 사용하여 여러 리전에 API를 배포할 수 있게 지원함으로써, 배포된 각 API에 동일한 사용자 지정 도메인 이름을 설정하고 Route 53에서 지연 시간 기반 DNS 레코드를 구성하여 클라이언트 요청을 지연 시간이 가장 짧은 리전에 라우팅합니다.

API 엔드포인트를 참조하십시오.

라우팅

API Gateway에서 WebSocket 경로는 수신 메시지를 메시지 내용에 따라 AWS Lambda 함수 같은 특정 통합으로 연결하는 데 사용됩니다. WebSocket API를 정의할 때 라우팅 키와 통합 백엔드를

지정합니다. 라우팅 키는 메시지 본문의 속성입니다. 수신 메시지에서 라우팅 키가 일치하면 통합 백엔드가 호출됩니다.

일치하지 않는 라우팅 키에 대해 기본 라우팅을 설정하거나, 라우팅을 수행하고 요청을 처리하는 백엔드 구성 요소로 메시지를 전달하는 프록시 모델을 지정할 수도 있습니다.

라우팅 요청

API Gateway에 있는 WebSocket API 메서드의 퍼블릭 인터페이스로, 본문을 정의합니다. 앱 개발자가 API를 통해 백엔드에 액세스하려면 반드시 이 파라미터와 본문을 요청에 포함해 보내야 합니다.

라우팅 응답

WebSocket API의 퍼블릭 인터페이스로, 앱 개발자가 API Gateway에 기대하는 본문 모델, 헤더, 상태 코드를 여기서 정의합니다.

사용량 계획

사용량 계획은 선택한 API 클라이언트에게 하나 이상의 배포된 REST 또는 WebSocket API에 대한 액세스를 제공합니다. 사용량 계획을 사용해 조절 및 할당량 제한을 구성할 수 있으며, 이는 개별 클라이언트 API 키에 적용됩니다.

WebSocket 연결

API Gateway는 클라이언트와 API Gateway 자체 사이에 영구적인 연결을 유지합니다. API Gateway와 Lambda 함수와 같은 백엔드 통합 사이에는 영구적인 연결이 없습니다. 백엔드 서비스는 클라이언트에서 수신한 메시지의 내용에 따라 필요 시 호출됩니다.

REST API와 HTTP API 중에서 선택

REST API와 HTTP API는 모두 RESTful API 제품입니다. REST API는 HTTP API보다 더 많은 기능을 지원하지만 HTTP API는 최소한의 기능으로 설계되어 더 낮은 가격으로 제공될 수 있습니다. API 키, 클라이언트별 제한, 요청 검증, AWS WAF 통합 또는 프라이빗 API 엔드포인트와 같은 기능이 필요한 경우 REST API를 선택합니다. REST API에 포함된 기능이 필요하지 않은 경우 HTTP API를 선택합니다.

다음 섹션에는 REST API 및 HTTP API에서 사용할 수 있는 핵심 기능이 요약되어 있습니다.

[엔드포인트 유형]

엔드포인트 유형은 API Gateway가 API용으로 생성하는 엔드포인트를 나타냅니다. 자세한 설명은 the section called “API 엔드포인트 유형 선택” 섹션을 참조하세요.

엔드포인트 유형	REST API	HTTP API
엣지 최적화	✓	
리전	✓	✓
프라이빗	✓	

보안

API Gateway는 악의적 행위자나 트래픽 스파이크 같은 특정 위협으로부터 API를 보호할 수 있는 여러 가지 방법을 제공합니다. 자세한 내용은 the section called “보호” 및 the section called “보호” 섹션을 참조하세요.

보안 기능	REST API	HTTP API
상호 TLS 인증	✓	✓
백엔드 인증을 위한 인증서	✓	
AWS WAF	✓	

권한 부여

API Gateway는 API 액세스 제어 및 관리에 다중 메커니즘을 지원합니다. 자세한 내용은 the section called “액세스 제어” 및 the section called “액세스 제어” 섹션을 참조하세요.

권한 부여 옵션	REST API	HTTP API
IAM	✓	✓
리소스 정책	✓	
Amazon Cognito	✓	✓ [1]
AWS Lambda 함수를 사용한 사용자 지정 인증	✓	✓

권한 부여 옵션	REST API	HTTP API
JWT(JSON Web Token) [2]		✓

[1] JWT 권한 부여자와 함께 Amazon Cognito를 사용할 수 있습니다.

[2] Lambda 권한 부여자를 사용하여 REST API용 JWT를 검증할 수 있습니다.

API 관리

API 키 및 클라이언트별 속도 제한과 같은 API 관리 기능이 필요한 경우 REST API를 선택합니다. 자세한 내용은 the section called “배포”, the section called “사용자 지정 도메인 이름”, the section called “사용자 지정 도메인 이름” 단원을 참조하세요.

특성	REST API	HTTP API
사용자 지정 도메인	✓	✓
API 키	✓	
클라이언트별 속도 제한	✓	
클라이언트별 사용 제한	✓	

개발

API Gateway API를 개발할 때 API의 여러 특성을 결정합니다. 이러한 특성은 API의 사용 사례에 따라 달라집니다. 자세한 내용은 the section called “개발” 및 the section called “개발” 단원을 참조하세요.

특성	REST API	HTTP API
CORS 구성	✓	✓
테스트 호출	✓	
캐싱	✓	
사용자 제어 배포	✓	✓

특성	REST API	HTTP API
자동 배포		✓
사용자 지정 게이트웨이 응답	✓	
Canary 릴리스 배포	✓	
요청 검증	✓	
요청 파라미터 변환	✓	✓
요청 본문 변환	✓	

모니터링

API Gateway는 API 요청을 로그하고 API를 모니터링하는 몇 가지 옵션을 지원합니다. 자세한 내용은 the section called “모니터링” 및 the section called “모니터링” 섹션을 참조하세요.

기능	REST API	HTTP API
아마존 CloudWatch 메트릭스	✓	✓
로그에서 CloudWatch 로그에 대한 액세스	✓	✓
Amazon Data Firehose에 대한 액세스 로그	✓	
실행 로그	✓	
AWS X-Ray 추적	✓	

통합

통합 기능은 API Gateway API를 백엔드 리소스에 연결합니다. 자세한 내용은 the section called “통합” 및 the section called “통합” 섹션을 참조하세요.

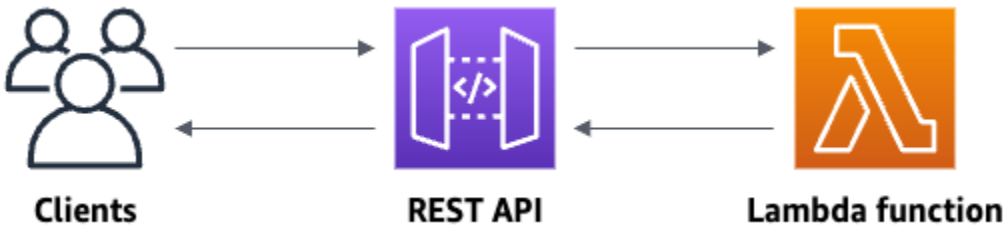
기능	REST API	HTTP API
퍼블릭 HTTP 엔드포인트	✓	✓
AWS 서비스	✓	✓
AWS Lambda 기능	✓	✓
Network Load Balancer와의 프라이빗 통합	✓	✓
Application Load Balancer와의 프라이빗 통합		✓
다음과 같은 비공개 통합 AWS Cloud Map		✓
모의 통합	✓	

REST API 콘솔 시작하기

이 시작 연습에서는 API Gateway REST API 콘솔을 사용하여 서버리스 REST API를 생성합니다. 서버리스 API를 사용하면 서버를 프로비저닝하고 관리하는 데 시간을 소비하는 대신 애플리케이션에 집중할 수 있습니다. 이 연습은 완료하는 데 20분을 넘지 않으며 AWS 프리 티어 내에서 가능합니다.

먼저, Lambda 콘솔을 사용하여 Lambda 함수를 생성합니다. 그런 다음 API Gateway REST API 콘솔을 사용하여 REST API를 생성합니다. API 메서드를 생성하고 Lambda 프록시 통합을 사용하여 Lambda 함수와 통합합니다. 마지막으로, API를 배포 및 테스트합니다.

REST API를 호출하면 API Gateway는 요청을 Lambda 함수로 라우팅합니다. Lambda는 함수를 실행하고 API Gateway에 응답을 반환합니다. 그러고 나면 API Gateway가 응답을 반환합니다.



이 연습을 완료하려면 AWS 계정과 콘솔 액세스 권한이 있는 AWS Identity and Access Management(IAM) 사용자가 있어야 합니다. 자세한 내용은 API Gateway를 시작하기 위한 사전 조건 섹션을 참조하세요.

주제

- 1단계: Lambda 함수 생성
- 2단계: REST API 생성
- 3단계: Lambda 프록시 통합 생성
- 4단계: API 배포
- 5단계: API 호출
- (선택 사항) 6단계: 정리

1단계: Lambda 함수 생성

API의 백엔드에 Lambda 함수를 사용합니다. Lambda는 필요 시에만 코드를 실행하며, 일일 몇 개의 요청에서 초당 수천 개의 요청까지 자동으로 확장이 가능합니다.

이 연습에서는 Lambda 콘솔에서 기본 Node.js 함수를 사용합니다.

Lambda 함수 생성

1. https://console.aws.amazon.com/lambda에서 Lambda 콘솔에 로그인합니다.
2. 함수 생성을 선택합니다.
3. 기본 정보(Basic information)에서 함수 이름(Function name)에 **my-function**을 입력합니다.
4. 함수 생성(Create function)을 선택합니다.

기본 Lambda 함수 코드는 다음과 유사해야 합니다.

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

함수의 응답이 API Gateway에 필요한 형식과 일치하는 한 이 연습의 Lambda 함수를 수정할 수 있습니다.

기본 응답 본문(`Hello from Lambda!`)을 `The API Gateway REST API console is great!`로 바꿉니다. 예제 함수를 호출하면 업데이트된 응답과 함께 `200` 응답이 클라이언트에 반환됩니다.

2단계: REST API 생성

다음으로 루트 리소스(/)를 사용하여 REST API를 생성합니다.

REST API를 생성하려면

1. https://console.aws.amazon.com/apigateway에서 API Gateway 콘솔에 로그인합니다.
2. 다음 중 하나를 수행하세요.
   - 첫 번째 API를 생성하려면 REST API에서 빌드를 선택합니다.
   - 이전에 API를 생성한 경우 API 생성을 선택한 다음 REST API에서 빌드를 선택합니다.
3. API 이름에서 **`my-rest-api`**을(를) 입력합니다.
4. (선택 사항) 설명에 설명을 입력합니다.
5. API 엔드포인트 유형 설정을 지역으로 유지합니다.
6. API 생성(Create API)을 선택합니다.

3단계: Lambda 프록시 통합 생성

다음으로 루트 리소스(/)에 REST API용 API 메서드를 생성하고 프록시 통합을 사용하여 이 메서드를 Lambda 함수와 통합합니다. Lambda 프록시 통합에서 API Gateway는 클라이언트로부터 들어오는 요청을 Lambda 함수에 직접 전달합니다.

Lambda 프록시 통합을 생성하려면

1. / 리소스를 선택한 다음 메서드 생성을 선택합니다.
2. 메서드 유형에서 ANY를 선택합니다.
3. 통합 유형에서 Lambda를 선택합니다.
4. Lambda 프록시 통합을 켭니다.
5. Lambda 함수의 경우 **`my-function`**을 입력한 다음 Lambda 함수를 선택합니다.
6. 메서드 생성을 선택합니다.

4단계: API 배포

다음으로, API 배포를 생성하고 스테이지에 연결합니다.

API를 배포하려면

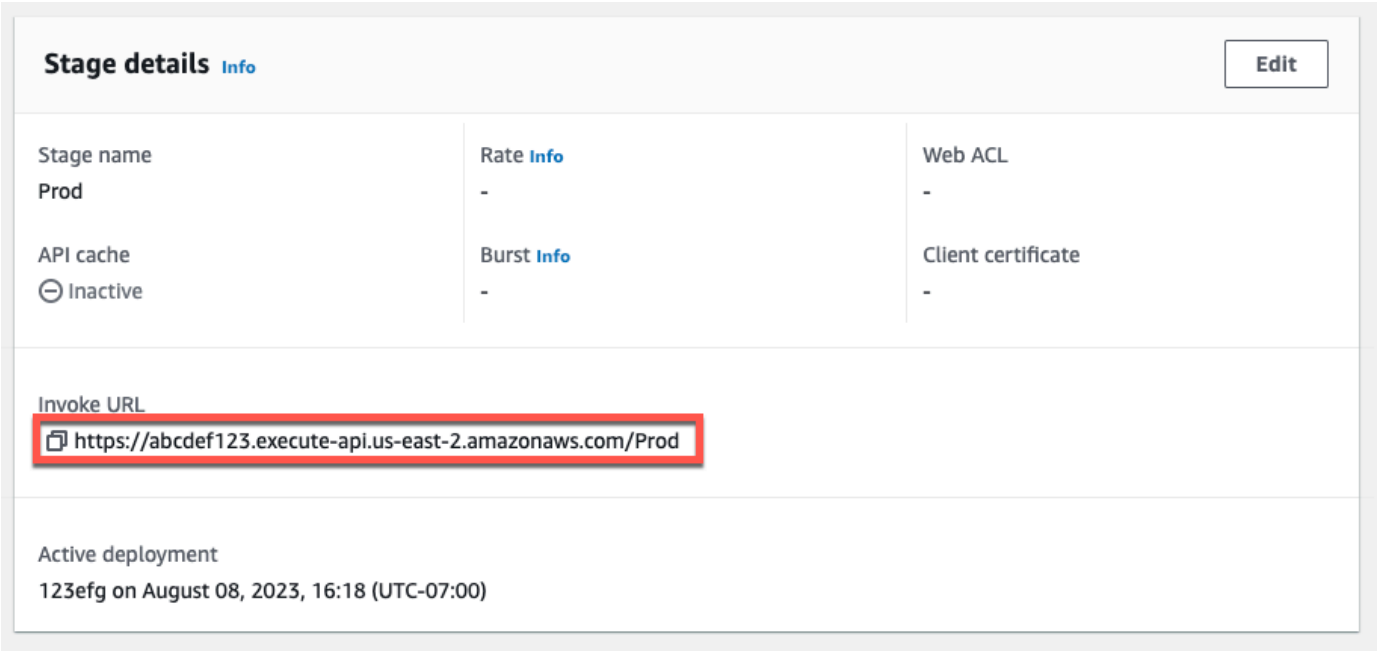
1. Deploy API(API 배포)를 선택합니다.
2. 스테이지에서 새 스테이지를 선택합니다.
3. 단계 이름에 **`Prod`**를 입력합니다.
4. (선택 사항) 설명에 설명을 입력합니다.
5. 배포를 선택합니다.

이제 클라이언트가 API를 호출할 수 있습니다. API를 배포하기 전에 테스트하려는 경우, 선택적으로 ANY 메서드를 선택하고 테스트 탭으로 이동한 다음 테스트를 선택할 수 있습니다.

5단계: API 호출

API를 호출하려면

1. 기본 탐색 창에서 스테이지를 선택합니다.
2. 스테이지 세부 정보에서 복사 아이콘을 선택하여 API의 호출 URL을 복사합니다.



3. 웹 브라우저에 호출 URL을 입력합니다.

전체 URL은 `https://abcdef123.execute-api.us-east-2.amazonaws.com/Prod`와(과) 같아야 합니다.

브라우저가 API에 GET 요청을 보냅니다.

4. API의 응답을 확인합니다. 브라우저에 텍스트 `"The API Gateway REST API console is great!"`이(가) 표시됩니다.

(선택 사항) 6단계: 정리

AWS 계정에 불필요한 비용이 발생하지 않도록 하려면 이 연습의 일부로 생성한 리소스를 삭제하세요. 다음 단계에서는 REST API, Lambda 함수 및 관련 리소스를 삭제합니다.

REST API를 삭제하려면

1. 리소스 창에서 API 작업, API 삭제를 선택합니다.
2. API 삭제 대화 상자에 confirm을 입력한 다음 삭제를 선택합니다.

Lambda 함수를 삭제하려면

1. https://console.aws.amazon.com/lambda에서 Lambda 콘솔에 로그인합니다.
2. 함수 페이지에서 함수를 선택합니다. 작업(Actions), 삭제(Delete)를 선택합니다.
3. 1개 함수 삭제 대화 상자에 **`delete`**를 입력한 후 삭제를 선택합니다.

Lambda 함수의 로그 그룹을 삭제하려면

1. Amazon CloudWatch 콘솔의 로그 그룹 페이지를 엽니다.
2. 로그 그룹 페이지에서 함수의 로그 그룹(`/aws/lambda/my-function`)을 선택합니다. 작업에서 로그 그룹 삭제를 선택합니다.
3. 로그 그룹 삭제(Delete log group(s)) 대화 상자에서 삭제(Delete)를 선택합니다.

Lambda 함수의 실행 역할을 삭제하려면

1. IAM 콘솔에서 역할 페이지를 엽니다.
2. (선택 사항) 역할 페이지의 검색 상자에 **`my-function`**을 입력합니다.
3. 함수의 역할(예: `my-function-31exxmpl`)을 선택한 다음 삭제를 선택합니다.

4. **my-function-*31exxmpl***을 삭제하시겠습니까? 대화 상자에 역할 이름을 입력한 후 삭제를 선택합니다.

> **Tip**
>
> AWS CloudFormation 또는 AWS Serverless Application Model(AWS SAM)을 사용하여 AWS 리소스의 생성 및 정리를 자동화할 수 있습니다. 일부 예제 AWS CloudFormation 템플릿은 awsdocs GitHub 리포지토리의 API Gateway용 예제 템플릿을 참조하세요.

API Gateway를 시작하기 위한 사전 조건

Amazon API Gateway를 처음 사용한다면 먼저 다음 태스크를 완료해야 합니다.

AWS 계정에 등록

AWS 계정가 없는 경우 다음 절차에 따라 계정을 생성합니다.

AWS 계정에 가입하려면

1. https://portal.aws.amazon.com/billing/signup을 엽니다.
2. 온라인 지시 사항을 따릅니다.

   가입 절차 중 전화를 받고 전화 키패드로 확인 코드를 입력하는 과정이 있습니다.

   AWS 계정에 가입하면 AWS 계정 루트 사용자 항목이 생성됩니다. 루트 사용자에게 계정의 모든 AWS 서비스 및 리소스에 대한 액세스 권한이 있습니다. 보안 모범 사례는 관리 사용자에게 관리자 액세스 권한을 할당하고, 루트 사용자만 루트 사용자 액세스 권한이 필요한 작업을 수행하는 것입니다.

가입 프로세스가 완료되면 AWS가 확인 이메일을 전송합니다. 언제든지 https://aws.amazon.com/으로 이동하고 내 계정을 선택하여 현재 계정 활동을 보고 계정을 관리할 수 있습니다.

관리 사용자 생성

AWS 계정에 가입하고, AWS 계정 루트 사용자에 보안 조치를 한 다음, AWS IAM Identity Center를 활성화하여 일상적인 작업에 루트 사용자를 사용하지 않도록 관리 사용자를 생성합니다.

귀하의 AWS 계정 루트 사용자 보호

1. 루트 사용자를 선택하고 AWS 계정 이메일 주소를 입력하여 AWS Management Console에 계정 소유자로 로그인합니다. 다음 페이지에서 암호를 입력합니다.

   루트 사용자를 사용하여 로그인하는 데 도움이 필요하면 AWS 로그인 사용 설명서의 루트 사용자로 로그인을 참조하세요.
2. 루트 사용자에 대해 다중 인증(MFA)을 활성화합니다.

지침은 IAM 사용 설명서의 AWS 계정 루트 사용자용 가상 MFA 디바이스 활성화(콘솔) 섹션을 참조하세요.

관리 사용자 생성

1. IAM Identity Center를 활성화합니다.

   지침은 AWS IAM Identity Center 사용 설명서의 AWS IAM Identity Center 활성화를 참조하세요.
2. IAM Identity Center에서 관리 사용자에게 관리 액세스 권한을 부여합니다.

   IAM Identity Center 디렉터리를 ID 소스로 사용하는 방법에 대한 자습서는 AWS IAM Identity Center 사용 설명서의 Configure user access with the default IAM Identity Center 디렉터리를 참조하세요.

관리 사용자로 로그인

- IAM 자격 증명 센터 사용자로 로그인하려면 IAM 자격 증명 센터 사용자를 생성할 때 이메일 주소로 전송된 로그인 URL을 사용합니다.

  IAM Identity Center 사용자로 로그인하는 데 도움이 필요한 경우 AWS 로그인 사용 설명서의 AWS 액세스 포털에 로그인을 참조하세요.

API Gateway 시작하기

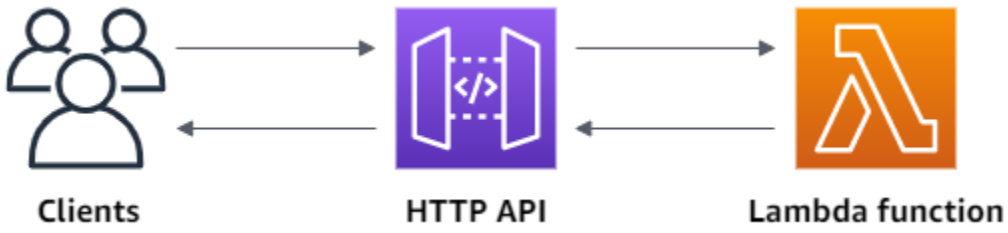
> **Note**
>
> 이제 재설계된 API Gateway 콘솔 환경을 사용할 수 있습니다. 콘솔을 사용하여 REST API를 생성하는 방법에 대한 튜토리얼은 REST API 콘솔 시작하기 섹션을 참조하세요.

이번 시작하기 연습에서는 서버리스 API를 생성합니다. 서버리스 API를 사용하면 서버를 프로비저닝하고 관리하는 데 시간을 소비하는 대신 애플리케이션에 집중할 수 있습니다. 이 연습은 완료하는 데 20분을 넘지 않으며 AWS 프리 티어 내에서 가능합니다.

먼저, AWS Lambda 콘솔을 사용하여 Lambda 함수를 생성합니다. 그런 다음 API Gateway 콘솔을 사용하여 HTTP API를 생성합니다. 그런 다음 API를 호출합니다.

> **Note**
>
> 이 연습에서는 단순화를 위해 HTTP API를 사용합니다. API Gateway는 더 많은 기능을 포함하는 REST API도 지원합니다. 자세한 내용은 the section called “REST API와 HTTP API 중에서 선택 ” 단원을 참조하십시오.

HTTP API를 호출하면 API Gateway는 요청을 Lambda 함수로 라우팅합니다. Lambda는 Lambda 함수를 실행하고 API Gateway에 응답을 반환합니다. 그러고 나면 API Gateway가 응답을 반환합니다.



이 연습을 완료하려면 AWS 계정과 콘솔 액세스 권한이 있는 AWS Identity and Access Management 사용자가 있어야 합니다. 자세한 내용은 사전 조건 단원을 참조하세요.

주제

- 1단계: Lambda 함수 생성
- 2단계: HTTP API 생성

- 3단계: API 테스트
- (선택 사항) 4단계: 정리
- 다음 단계

1단계: Lambda 함수 생성

API의 백엔드에 Lambda 함수를 사용합니다. Lambda는 필요 시에만 코드를 실행하며, 일일 몇 개의 요청에서 초당 수천 개의 요청까지 자동으로 확장이 가능합니다.

이 예제에서는 Lambda 콘솔에서 기본 Node.js 함수를 사용합니다.

Lambda 함수를 만들려면

1. https://console.aws.amazon.com/lambda에서 Lambda 콘솔에 로그인합니다.
2. 함수 생성을 선택합니다.
3. [함수 이름(Function name)]에 **`my-function`**을 입력합니다.
4. 함수 생성을 선택합니다.

예제 함수는 클라이언트에 `200` 응답과 텍스트 `Hello from Lambda!`을(를) 반환합니다.

함수의 응답이 API Gateway에 필요한 형식과 일치하는 한 Lambda 함수를 수정할 수 있습니다.

기본 Lambda 함수 코드는 다음과 유사해야 합니다.

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

2단계: HTTP API 생성

다음으로, HTTP API를 생성합니다. API Gateway는 REST API와 WebSocket API도 지원하지만 이 연습에서는 HTTP API가 가장 좋은 선택입니다. REST API는 HTTP API보다 더 많은 기능을 지원하지만 이 연습에서는 그러한 기능이 필요하지 않습니다. WebSocket API는 최소한의 기능으로 설계되어 더

낮은 가격으로 제공될 수 있습니다. WebSocket API는 전이중 통신을 위해 클라이언트와의 지속적인 연결을 유지하는데, 이 예제에는 필요하지 않습니다.

HTTP API는 Lambda 함수에 대한 HTTP 엔드포인트를 제공합니다. API Gateway는 요청을 Lambda 함수로 라우팅한 다음 함수의 응답을 클라이언트에 반환합니다.

HTTP API 생성하기

1. https://console.aws.amazon.com/apigateway에서 API Gateway 콘솔에 로그인합니다.
2. 다음 중 하나를 수행하십시오.
   - 첫 번째 API를 생성하려면 HTTP API에서 [빌드(Build)]를 선택합니다.
   - 이전에 API를 생성한 경우 [API 생성(Create API)]을 선택한 다음 HTTP API에서 [빌드(Build)]를 선택합니다.
3. [통합(Integrations)]에서 [통합 추가(Add integration)]를 선택합니다.
4. Lambda를 선택합니다.
5. Lambda 함수에서 **my-function**을(를) 입력합니다.
6. API 이름에서 **my-http-api**을(를) 입력합니다.
7. [Next]를 선택합니다.
8. API Gateway가 생성하는 경로를 검토한 후 [다음(Next)]을 선택합니다.
9. API Gateway가 생성하는 단계를 검토한 후 [다음(Next)]을 선택합니다.
10. Create를 선택합니다.

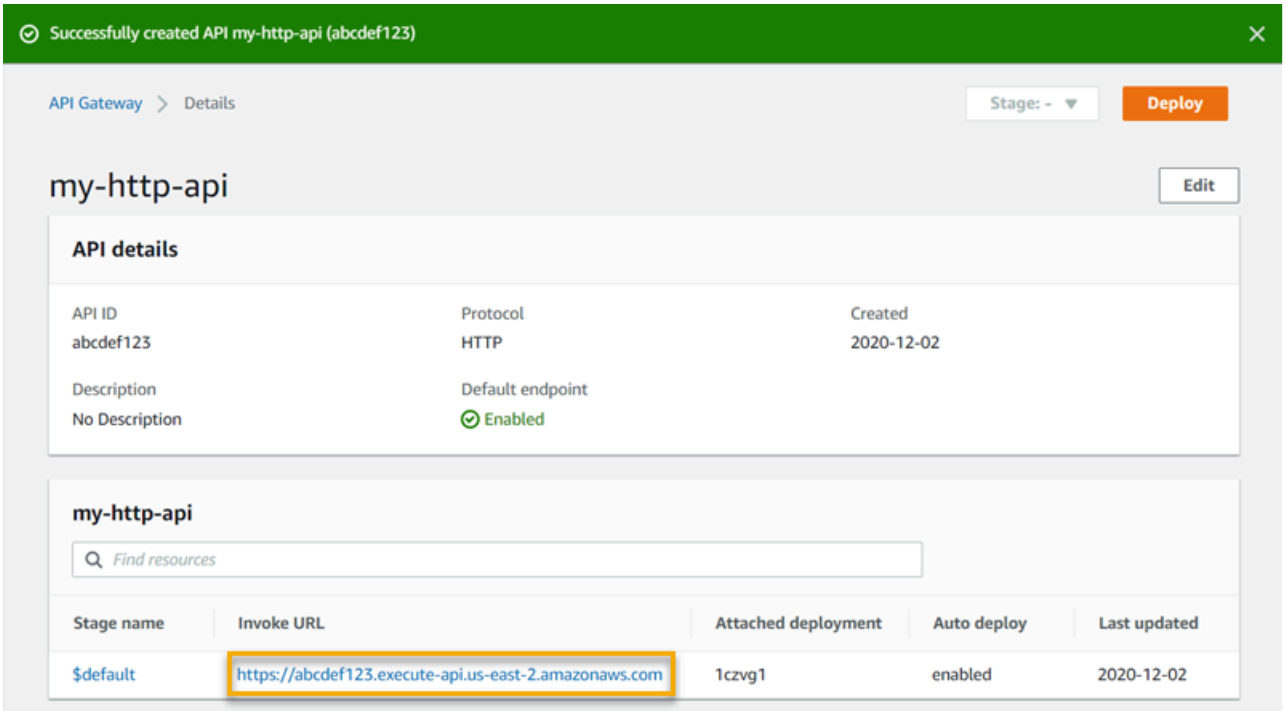
이제 클라이언트에서 요청을 수신할 수 있는 Lambda 통합이 포함된 HTTP API가 생성되었습니다.

3단계: API 테스트

다음으로, API를 테스트하여 제대로 작동하는지 확인합니다. 간단하게 하기 위해, 웹 브라우저를 사용하여 API를 호출합니다.

API 테스트하기

1. https://console.aws.amazon.com/apigateway에서 API Gateway 콘솔에 로그인합니다.
2. API를 선택합니다.
3. API의 호출 URL을 기록해 둡니다.



4. API의 호출 URL을 복사하여 웹 브라우저에 입력합니다. Lambda 함수를 호출할 호출 URL에 Lambda 함수의 이름을 추가합니다. 기본적으로 API Gateway 콘솔은 Lambda 함수(`my-function`)와 이름이 같은 경로를 생성합니다.

   전체 URL은 `https://`*`abcdef123`*`.execute-api.`*`us-east-2`*`.amazonaws.com/`*`my-function`*와(과) 같아야 합니다.

   브라우저가 API에 `GET` 요청을 보냅니다.
5. API의 응답을 확인합니다. 브라우저에 텍스트 `"Hello from Lambda!"`이(가) 표시됩니다.

(선택 사항) 4단계: 정리

불필요한 비용을 방지하려면 이 시작하기 연습의 일부로 생성한 리소스를 삭제하세요. 다음 단계에서는 HTTP API, Lambda 함수 및 관련 리소스를 삭제합니다.

HTTP API 삭제하기

1. https://console.aws.amazon.com/apigateway에서 API Gateway 콘솔에 로그인합니다.
2. API 페이지에서 API를 선택합니다. [Actions]를 선택한 후 [Delete]를 선택합니다.
3. 삭제를 선택합니다.

Lambda 함수 삭제하기

1. https://console.aws.amazon.com/lambda에서 Lambda 콘솔에 로그인합니다.
2. 함수 페이지에서 함수를 선택합니다. [Actions]를 선택한 후 [Delete]를 선택합니다.
3. 삭제를 선택합니다.

Lambda 함수의 로그 그룹 삭제하기

1. Amazon CloudWatch 콘솔에서 로그 그룹 페이지를 엽니다.
2. 로그 그룹 페이지에서 함수의 로그 그룹(`/aws/lambda/my-function`)을 선택합니다. 작업을 선택한 다음 로그 그룹 삭제를 선택합니다.
3. 삭제를 선택합니다.

Lambda 함수의 실행 역할 삭제하기

1. AWS Identity and Access Management 콘솔에서 역할 페이지를 엽니다.
2. 함수의 역할(예: `my-function-31exxmpl`)을 선택합니다.
3. 역할 삭제를 선택합니다.
4. 예, 삭제를 선택합니다.

AWS CloudFormation 또는 AWS SAM을 사용하여 AWS 리소스의 생성 및 정리를 자동화할 수 있습니다. 예제 AWS CloudFormation 템플릿에 대해서는 예제 AWS CloudFormation 템플릿을 참조하세요.

다음 단계

이 예제에서는 AWS Management Console을 사용하여 간단한 HTTP API를 생성했습니다. HTTP API는 Lambda 함수를 호출하고 클라이언트에 응답을 반환합니다.

API Gateway 작업을 계속할 경우의 다음 단계는 아래와 같습니다.

- 다음을 포함한 API 통합의 추가 유형을 구성합니다.
  - HTTP 엔드포인트
  - VPC의 프라이빗 리소스(예: Amazon ECS 서비스)
  - Amazon Simple Queue Service, AWS Step Functions, Kinesis Data Streams와 같은 AWS 서비스
- API에 대한 액세스 제어

- API에 대한 로깅 활성화
- API에 대한 조절 구성
- API에 대한 사용자 지정 도메인 구성

커뮤니티에서 Amazon API Gateway에 대한 도움말을 보려면 API Gateway 토론 포럼을 참조하세요. 이 포럼에 들어갈 때 AWS에서 로그인을 요청할 수 있습니다.

AWS에서 직접 API Gateway에 대한 도움을 받으려면 AWS Support 페이지의 지원 옵션을 참조하세요.

또한 FAQ를 참조하거나 직접 당사에 문의하십시오.

Amazon API Gateway 자습서 및 워크숍

다음 자습서 및 워크숍에는 API Gateway에 대해 알아보는 데 도움이 되는 실습용 과제가 나와 있습니다.

REST API 자습서

- Lambda 통합을 사용하여 API Gateway REST API 구축
- 자습서: 예제를 가져와 REST API 생성
- HTTP 통합을 사용하여 API Gateway REST API 구축
- 자습서: API Gateway 프라이빗 통합으로 REST API 빌드
- 자습서: AWS 통합을 통해 API Gateway REST API 빌드
- 자습서: AWS 서비스 통합 2개와 Lambda 비프록시 통합 1개가 포함된 `Calc` REST API 생성
- 자습서: API Gateway에서 REST API를 Amazon S3 프록시로 생성
- 자습서: API Gateway에서 REST API를 Amazon Kinesis 프록시로 생성
- 자습서: 프라이빗 REST API 빌드

HTTP API 자습서

- 자습서: Lambda 및 DynamoDB를 사용한 CRUD API 구축
- 자습서: Amazon ECS 서비스에 대한 프라이빗 통합을 통해 HTTP API 구축

WebSocket API 자습서

- 자습서: WebSocket API, Lambda 및 DynamoDB를 사용하여 서버리스 채팅 앱 구축

워크숍

- 서버리스 웹 애플리케이션 구축
- 서버리스 애플리케이션을 위한 CI/CD
- 서버리스 보안 워크숍
- 서버리스 ID 관리, 인증 및 권한 부여
- Amazon API Gateway Workshop

Amazon API Gateway REST API 자습서

다음 자습서에는 API Gateway REST API에 대해 알아보는 데 도움이 되는 실습용 과제가 나와 있습니다.

주제

- Lambda 통합을 사용하여 API Gateway REST API 구축
- 자습서: 예제를 가져와 REST API 생성
- HTTP 통합을 사용하여 API Gateway REST API 구축
- 자습서: API Gateway 프라이빗 통합으로 REST API 빌드
- 자습서: AWS 통합을 통해 API Gateway REST API 빌드
- 자습서: AWS 서비스 통합 2개와 Lambda 비프록시 통합 1개가 포함된 Calc REST API 생성
- 자습서: API Gateway에서 REST API를 Amazon S3 프록시로 생성
- 자습서: API Gateway에서 REST API를 Amazon Kinesis 프록시로 생성
- 자습서: 프라이빗 REST API 빌드

Lambda 통합을 사용하여 API Gateway REST API 구축

Lambda 통합을 사용하여 API를 구축하기 위해 Lambda 프록시 통합 또는 Lambda 비 프록시 통합을 사용할 수 있습니다.

Lambda 프록시 통합에서는 통합 Lambda 함수에 대한 입력을 요청 헤더, 경로 변수, 쿼리 문자열 파라미터 및 본문의 조합으로 표현할 수 있습니다. 또한 Lambda 함수는 API 구성 설정을 사용하여 실행 로직에 영향을 줄 수 있습니다. API 개발자의 경우 Lambda 프록시 통합을 간단하게 설정할 수 있습니다. 지정된 리전에서 특정 Lambda 함수를 선택하는 것 외에는 할 일이 거의 없습니다. API Gateway는 통합 요청 및 통합 응답을 구성합니다. 설정을 완료하면 통합 API 메서드는 기존 설정을 수정하지 않고 백엔드와 함께 진화할 수 있습니다. 백엔드 Lambda 함수 개발자가 수신되는 요청 데이터를 구문 분석하고 아무런 문제가 없을 때 원하는 결과와 함께 클라이언트에 응답하거나 무언가 문제가 있을 때 오류 메시지와 함께 응답하기 때문입니다.

Lambda 비 프록시 통합에서는 Lambda 함수에 대한 입력이 통합 요청 페이로드로 제공되는지 확인해야 합니다. 즉, API 개발자는 클라이언트가 요청 파라미터로 제공한 모든 입력 데이터를 적절한 통합 요청 본문에 매핑해야 합니다. 클라이언트가 제공한 요청 본문을 Lambda 함수가 인식하는 형식으로 변환해야 할 수도 있습니다.

주제

- 자습서: Lambda 프록시 통합을 사용하여 Hello World REST API 빌드
- 자습서: 교차 계정 Lambda 프록시 통합을 사용하여 API Gateway REST API 빌드
- 자습서: Lambda 비 프록시 통합을 사용하여 API Gateway REST API 빌드

자습서: Lambda 프록시 통합을 사용하여 Hello World REST API 빌드

Lambda 프록시 통합은 API 메서드나 전체 API를 Lambda 함수와 통합할 수 있는 유연하고 간단한 API Gateway API 통합 유형입니다. Lambda 함수는 Lambda가 지원하는 어떤 언어로도 작성할 수 있습니다. 이것은 프록시 통합이기 때문에 API를 다시 배포할 필요 없이 언제든지 Lambda 함수 구현을 변경할 수 있습니다.

이 자습서에서는 다음 작업을 수행합니다.

- "Hello, World!" 만들기 API의 백엔드가 될 Lambda 함수를 만듭니다.
- "Hello, World!" 만들기 및 테스트 Lambda 프록시 통합을 사용하여 API를 만들고 테스트합니다.

주제

- "Hello, World!" 만들기 Lambda 함수
- "Hello, World!" 만들기 API
- API 배포 및 테스트

"Hello, World!" 만들기 Lambda 함수

“Hello, World!”를 생성하려면 Lambda 콘솔의 Lambda 함수

1. https://console.aws.amazon.com/lambda에서 Lambda 콘솔에 로그인합니다.
2. AWS 탐색 막대에서 지역 (예: 미국 동부 (버지니아 북부)) 을 선택합니다.

   > **Note**
   >
   > Lambda 함수를 생성한 리전을 적어 둡니다. 이 리전은 API를 생성할 때 필요합니다.

3. 탐색 창에서 함수를 선택합니다.
4. 함수 생성을 선택합니다.
5. 새로 작성을 선택합니다.

6. 기본 정보에서 다음과 같이 합니다.

   a. 함수 이름에 **GetStartedLambdaProxyIntegration**을 입력합니다.

   b. 런타임에서 지원되는 최신 Node.js 또는 Python 런타임을 선택합니다.

   c. 권한(Permissions)에서 기본 실행 역할 변경(Change default execution role)을 확장합니다. 실행 역할 드롭다운 목록에서 AWS 정책 템플릿에서 새 역할 생성을 선택합니다.

   d. 역할 이름에 **GetStartedLambdaBasicExecutionRole**을 입력합니다.

   e. [Policy templates] 필드를 비워둡니다.

   f. 함수 생성을 선택합니다.

7. 다음 코드를 복사하여 인라인 코드 편집기의 함수 코드에 붙여 넣습니다.

   Node.js

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

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

Python

```
import json


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

    greeter = 'World'

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

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

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

    if (event['body']) and (event['body'] is not None):
        body = json.loads(event['body'])
        try:
            if (body['greeter']) and (body['greeter'] is not None):
                greeter = body['greeter']
        except KeyError:
```

```
            print('No greeter')

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

    return res
```

8. [Deploy]를 선택합니다.

"Hello, World!" 만들기 API

이제 API Gateway 콘솔을 사용하여 “Hello, World!” Lambda 함수에 대한 API를 만듭니다.

“Hello, World!”를 생성하려면 API

1. https://console.aws.amazon.com/apigateway에서 API Gateway 콘솔에 로그인합니다.
2. API Gateway를 처음 사용하는 경우, 서비스의 기능을 소개하는 페이지가 나타납니다. REST API에서 빌드를 선택합니다. 예제 API 생성 팝업이 나타나면 확인을 선택합니다.

   API Gateway를 처음 사용하는 것이 아닌 경우 API 생성을 선택합니다. REST API에서 빌드를 선택합니다.
3. API 이름에서 **LambdaProxyAPI**을(를) 입력합니다.
4. (선택 사항) 설명에 설명을 입력합니다.
5. API 엔드포인트 유형 설정을 지역으로 유지합니다.
6. API 생성(Create API)을 선택합니다.

API를 생성한 후에는 리소스를 생성합니다. 일반적으로 API 리소스는 애플리케이션 로직에 따른 리소스 트리로 정리되어 있습니다. 이 예시에서는 /helloworld 리소스를 생성합니다.

리소스를 생성하려면

1. / 리소스를 선택한 다음 리소스 생성을 선택합니다.
2. 프록시 리소스는 꺼진 상태로 둡니다.
3. 리소스 경로를 /로 유지합니다.

4. 리소스 이름에 **helloworld**을 입력합니다.
5. 오리진 간 리소스 공유(CORS)를 꺼진 상태로 둡니다.
6. 리소스 생성을 선택합니다.

프록시 통합에서는 전체 요청이 HTTP 메서드를 나타내는 catch-all ANY 메서드를 통해 백엔드 Lambda 함수로 그대로 전송됩니다. 실제 HTTP 메서드는 실행 시 클라이언트가 지정합니다. ANY 메서드를 통해 지원되는 모든 HTTP 메서드인 DELETE, GET, HEAD, OPTIONS, PATCH, POST, PUT에 대해 단일 API 메서드 설정을 사용할 수 있습니다.

ANY 메서드를 생성하려면

1. /helloworld 리소스를 선택한 다음 메서드 생성을 선택합니다.
2. 메서드 유형에서 ANY를 선택합니다.
3. 통합 유형에서 Lambda 함수를 선택합니다.
4. Lambda 프록시 통합을 켭니다.
5. Lambda 함수의 경우, Lambda 함수를 생성한 위치를 선택한 다음 AWS 리전 함수 이름을 입력합니다.
6. 기본 제한 시간 값인 29초를 사용하려면 기본 제한 시간을 활성화된 상태로 유지합니다. 사용자 지정 제한 시간을 설정하려면 기본 제한 시간을 선택하고 50 ~ 29000밀리초 사이의 제한 시간 값을 입력합니다.
7. 메서드 생성을 선택합니다.

API 배포 및 테스트

API를 배포하려면

1. Deploy API(API 배포)를 선택합니다.
2. 스테이지에서 새 스테이지를 선택합니다.
3. 단계 이름에 **test**를 입력합니다.
4. (선택 사항) 설명에 설명을 입력합니다.
5. 배포를 선택합니다.
6. 스테이지 세부 정보에서 복사 아이콘을 선택하여 API의 호출 URL을 복사합니다.

브라우저와 cURL을 사용하여 API의 Lambda 프록시 통합 테스트

브라우저 또는 cURL을 사용하여 API를 테스트할 수 있습니다.

쿼리 문자열 파라미터를 사용하여 GET 요청을 테스트하려면, API의 helloworld 리소스에 대한 URL을 브라우저 주소 표시줄에 입력합니다. 예: https://***r275xc9bmd***.execute-api.***us-east-1***.amazonaws.com/test/helloworld?name=John&city=Seattle

다른 메서드의 경우 POSTMAN 또는 cURL 등의 고급 REST API 테스트 유틸리티를 사용해야 합니다. 이 자습서에서는 cURL을 사용합니다. 아래의 cURL 명령 예제는 컴퓨터에 cURL이 설치되어 있다고 전제합니다.

cURL을 사용하여 배포된 API를 테스트하려면 다음과 같이 합니다.

1. 터미널 창을 엽니다.
2. 다음 cURL 명령을 복사하여 터미널 창에 붙여 넣습니다. 호출 URL을 이전 단계에서 복사한 것으로 바꾸고 URL 끝에 **/helloworld**를 추가합니다.

   > **Note**
   >
   > Windows에서 이 명령을 실행할 경우 다음 구문을 사용하십시오.
   >
   > ```
   > curl -v -X POST "https://r275xc9bmd.execute-api.us-east-1.amazonaws.com/
   > test/helloworld" -H "content-type: application/json" -d "{ \"greeter\":
   >  \"John\" }"
   > ```

   a. ?greeter=John이라는 쿼리 문자열 파라미터와 함께 API를 직접적으로 호출하려면 다음과 같이 합니다.

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

   b. greeter:John이라는 헤더 파라미터와 함께 API를 직접적으로 호출하려면 다음과 같이 합니다.

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

c. `{"greeter":"John"}`이라는 본문과 함께 API를 직접적으로 호출하려면 다음과 같이 합니다.

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

이 모든 경우에 출력되는 것은 다음과 같은 응답 본문이 있는 200 응답입니다.

```
Hello, John!
```

자습서: 교차 계정 Lambda 프록시 통합을 사용하여 API Gateway REST API 빌드

이제 다른 AWS 계정의 AWS Lambda 함수를 API 통합 백엔드로 사용할 수 있습니다. 각 계정은 Amazon API Gateway를 사용할 수 있는 모든 리전에 존재할 수 있습니다. 따라서 여러 API에 대해 Lambda 백엔드 함수를 중앙에서 손쉽게 관리 및 공유할 수 있습니다.

이 단원에서는 Amazon API Gateway 콘솔을 사용하여 교차 계정 Lambda 프록시 통합을 구성하는 방법을 보여줍니다.

API Gateway 교차 계정 Lambda 통합을 위한 API 생성

API를 생성하는 방법

1. https://console.aws.amazon.com/apigateway에서 API Gateway 콘솔에 로그인합니다.
2. API Gateway를 처음 사용하는 경우, 서비스의 기능을 소개하는 페이지가 나타납니다. REST API에서 빌드를 선택합니다. 예제 API 생성 팝업이 나타나면 확인을 선택합니다.

   API Gateway를 처음 사용하는 것이 아닌 경우 API 생성을 선택합니다. REST API에서 빌드를 선택합니다.
3. API 이름에서 **`CrossAccountLambdaAPI`**을(를) 입력합니다.
4. (선택 사항) 설명에 설명을 입력합니다.
5. API 엔드포인트 유형 설정을 지역으로 유지합니다.
6. API 생성(Create API)을 선택합니다.

다른 계정으로 Lambda 통합 함수 생성

이제, 예제 API를 생성한 환경에서 다른 계정으로 Lambda 함수를 만들어 보겠습니다.

다른 계정으로 Lambda 함수 생성

1. API Gateway API를 생성한 환경에서 다른 계정으로 Lambda 콘솔에 로그인합니다.
2. 함수 생성을 선택합니다.
3. 새로 작성을 선택합니다.
4. 새로 작성에서 다음 작업을 수행합니다.
   a. 함수 이름에 이름을 입력합니다.
   b. 런타임 드롭다운 목록에서 지원되는 Node.js 런타임을 선택합니다.
   c. 권한에서 실행 역할 선택 또는 생성을 확장합니다. 역할을 만들거나 기존 역할을 선택할 수 있습니다.
   d. 함수 생성을 선택하여 계속 진행합니다.
5. 아래로 스크롤하여 함수 코드(Function code) 창을 찾습니다.
6. the section called “자습서: Lambda 프록시 통합을 사용한 Hello World API”에서 Node.js 함수 구현을 입력합니다.
7. [Deploy]를 선택합니다.
8. 함수의 정식 ARN을 기록해 둡니다(Lambda 함수 창의 오른쪽 위에). 이것은 나중에 교차 계정 Lambda 통합을 생성할 때 필요합니다.

교차 계정 Lambda 통합 구성

다른 계정으로 Lambda 통합 함수를 생성하면 API Gateway 콘솔을 사용하여 이를 첫 번째 계정의 API에 추가할 수 있습니다.

> **Note**
>
> 교차 리전, 교차 계정 권한 부여자를 구성할 때 대상 함수에 추가된 `sourceArn`은 API의 리전이 아닌 함수에 포함된 리전을 사용합니다.

API를 생성한 후에는 리소스를 생성합니다. 일반적으로 API 리소스는 애플리케이션 로직에 따른 리소스 트리로 정리되어 있습니다. 이 예시에서는 /helloworld 리소스를 생성합니다.

리소스를 생성하려면

1. / 리소스를 선택한 다음 리소스 생성을 선택합니다.
2. 프록시 리소스는 꺼진 상태로 둡니다.
3. 리소스 경로를 /로 유지합니다.
4. 리소스 이름에 **helloworld**을 입력합니다.
5. 오리진 간 리소스 공유(CORS)를 꺼진 상태로 둡니다.
6. 리소스 생성을 선택합니다.

리소스를 생성한 후 GET 메서드를 생성합니다. 다른 계정으로 Lambda 함수와 GET 메서드를 통합합니다.

GET 메서드를 생성하는 방법

1. /helloworld 리소스를 선택한 다음 메서드 생성을 선택합니다.
2. 메서드 유형에서 GET을 선택합니다.
3. 통합 유형에서 Lambda 함수를 선택합니다.
4. Lambda 프록시 통합을 켭니다.
5. Lambda 함수에, 1단계에서 입력한 Lambda 함수의 전체 ARN을 입력합니다.

   Lambda 콘솔 창의 오른쪽 위 모서리에서 함수의 ARN을 찾을 수 있습니다.
6. ARN을 입력하면 `aws lambda add-permission` 명령 문자열이 나타납니다. 이 정책은 첫 번째 계정이 두 번째 계정의 Lambda 함수에 액세스할 수 있는 권한을 부여합니다. `aws lambda add-permission`명령 문자열을 복사하여 두 번째 계정용으로 구성된 AWS CLI 창에 붙여넣습니다.
7. 메서드 생성을 선택합니다.

Lambda 콘솔에서 함수에 대해 업데이트된 정책을 확인할 수 있습니다.

(선택 사항) 업데이트된 정책을 보려면

1. 에 AWS Management Console 로그인하고 https://console.aws.amazon.com/lambda/ 에서 AWS Lambda 콘솔을 엽니다.
2. Lambda 함수를 선택합니다.
3. Permissions를 선택합니다.

`Allow`이 API `Condition` 메서드에 대한 ARN인 `AWS:SourceArn` 절이 포함된 `GET` 정책이 표시되어야 합니다.

자습서: Lambda 비 프록시 통합을 사용하여 API Gateway REST API 빌드

이 연습에서는 API Gateway 콘솔을 사용하여 클라이언트가 Lambda 비 프록시 통합(사용자 지정 통합이라고도 함)을 통해 Lambda 함수를 호출할 수 있도록 하는 API를 빌드합니다. AWS Lambda 및 Lambda 함수에 대한 자세한 내용은 AWS Lambda 개발자 안내서를 참조하세요.

원활한 학습을 위해 최소 API 설정으로 간단한 Lambda 함수를 선택하여 Lambda 사용자 지정 통합을 통해 API Gateway API를 구축하는 단계를 안내합니다. 필요한 경우 일부 로직에 대해 설명합니다. Lambda 사용자 지정 통합에 대한 자세한 예제는 자습서: AWS 서비스 통합 2개와 Lambda 비프록시 통합 1개가 포함된 `Calc` REST API 생성을 참조하십시오.

API를 생성하기 전에 다음에 설명한 대로 AWS Lambda에서 Lambda 함수를 생성하여 Lambda 백엔드를 설정합니다.

주제

- Lambda 비 프록시 통합을 위한 Lambda 함수 만들기
- Lambda 비 프록시 통합을 통해 API 생성
- API 메서드 호출 테스트
- API 배포
- 배포 단계에서 API 테스트
- 정리

Lambda 비 프록시 통합을 위한 Lambda 함수 만들기

> **Note**
>
> Lambda 함수를 생성하면 계정에 요금이 부과될 수 있습니다. AWS

이 단계에서는, "Hello, World!"와 같은 Lambda 사용자 지정 통합에 대한 Lambda 함수를 생성해 봅니다. 이 연습에서는 `GetStartedLambdaIntegration`이라는 함수가 사용됩니다.

이 `GetStartedLambdaIntegration` Lambda 함수의 구현은 다음과 같습니다.

Node.js

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

console.log('Loading function');

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

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

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

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

Python

```
import json

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

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

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

Lambda 사용자 지정 통합의 경우 API Gateway에서 입력을 통합 요청 본문으로 클라이언트의 Lambda 함수에 전달합니다. Lambda 함수 핸들러의 `event` 객체는 입력입니다.

Lambda 함수는 간단합니다. `event`, `name`, `city`, `time` 속성에 대한 입력 `day` 객체를 구문 분석합니다. 그런 다음 인사말을 `{"message":greeting}`의 JSON 객체로 호출자에게 반환합니다. 메시지는

"Good [morning|afternoon|day], [*name*|you] in [*city*|World]. Happy *day*!" 패턴입니다. Lambda 함수에 대한 입력이 다음 JSON 객체라고 가정합니다.

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

자세한 내용은 AWS Lambda 개발자 안내서를 참조하십시오.

또한 함수는 호출을 CloudWatch 통해 Amazon에 실행을 `console.log(...)` 기록합니다. 이는 함수를 디버깅할 때 호출 트레이스에 유용합니다. `GetStartedLambdaIntegration`함수가 호출을 기록하도록 허용하려면 Lambda 함수가 스트림을 생성하고 CloudWatch 스트림에 로그 항목을 추가하도록 Lambda 함수에 대한 적절한 정책이 포함된 IAM 역할을 설정하십시오. Lambda 콘솔은 필요한 IAM 역할과 정책을 생성하도록 안내합니다.

API Gateway 콘솔을 사용하지 않고 API를 설정할 경우(예: OpenAPI 파일에서 API를 가져오는 경우) 필요에 따라 API Gateway에서 Lambda 함수를 호출하기 위한 호출 역할 및 정책을 명시적으로 생성하고 설정해야 합니다. API Gateway API에 대한 Lambda 호출 및 실행 역할을 설정하는 방법에 대한 자세한 내용은 IAM 권한을 사용하여 API에 대한 액세스 제어 단원을 참조하십시오.

Lambda 프록시 통합에 대한 Lambda 함수인 `GetStartedLambdaProxyIntegration`과 달리 Lambda 사용자 지정 통합에 대한 `GetStartedLambdaIntegration` Lambda 함수는 API Gateway API 통합 요청 본문에서 입력만 가져옵니다. 함수는 JSON 객체, 문자열, 숫자, 부울 또는 심지어 이진 BLOB의 출력을 반환할 수 있습니다. 반대로 Lambda 프록시 통합에 대한 Lambda 함수는 모든 요청 데이터에서 입력을 가져올 수 있지만 특정 JSON 객체의 출력을 반환해야 합니다. API Gateway에서 클라이언트 요청을 백엔드로 전달하기 전에 필요한 API 요청 파라미터를 통합 요청 본문에 매핑하는 경우 Lambda 사용자 지정 통합에 대한 `GetStartedLambdaIntegration` 함수는 API 요청 파라미터를 입력으로 가질 수 있습니다. 이를 위해 API 개발자는 API를 생성할 때 매핑 템플릿을 생성하여 API 메서드에서 구성해야 합니다.

이제 `GetStartedLambdaIntegration` Lambda 함수를 생성합니다.

Lambda 사용자 지정 통합을 위한 **`GetStartedLambdaIntegration`** Lambda 함수를 생성하려면

1. https://console.aws.amazon.com/lambda/ 에서 AWS Lambda 콘솔을 엽니다.
2. 다음 중 하나를 수행합니다.

- 시작 페이지가 나타나면 지금 시작을 선택한 다음 함수 만들기를 선택합니다.
- Lambda > 함수(Lambda > Functions) 목록 페이지가 나타나면 함수 만들기(Create function)를 선택합니다.

3. 새로 작성을 선택합니다.
4. 새로 작성 창에서 다음과 같이 합니다.

   a. 이름(Name)에 Lambda 함수 이름으로 **GetStartedLambdaIntegration**을 입력합니다.

   b. 런타임에서 지원되는 최신 Node.js 또는 Python 런타임을 선택합니다.

   c. 권한(Permissions)에서 기본 실행 역할 변경(Change default execution role)을 확장합니다. 실행 역할 드롭다운 목록에서 AWS 정책 템플릿에서 새 역할 생성을 선택합니다.

   d. 역할 이름에 역할의 이름을 입력합니다(예: **GetStartedLambdaIntegrationRole**).

   e. 정책 템플릿에서 단순 마이크로서비스 권한을 선택합니다.

   f. 함수 생성을 선택합니다.

5. 함수 구성 창의 Function code(함수 코드)에서, 다음 필드들을 설정하십시오.

   a. 이 섹션의 시작 부분에 나열된 Lambda 함수 코드를 복사하여 인라인 코드 편집기에 붙여 넣습니다.

   b. 이 섹션의 다른 필드를 모두 기본값으로 그대로 둡니다.

   c. [Deploy]를 선택합니다.

6. 새로 생성된 함수를 테스트하려면 테스트 탭을 선택합니다.

   a. 이벤트 이름에 **HelloWorldTest**를 입력합니다.

   b. 이벤트 JSON에서 기본 코드를 다음과 같이 바꾸십시오.

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

   c. 테스트를 선택하여 함수를 호출합니다. 실행 결과: 성공 섹션이 표시됩니다. 세부 정보를 확장하면 다음 출력이 표시됩니다.

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

```
}
```

출력은 CloudWatch 로그에도 기록됩니다.

측면 연습으로 IAM 콘솔을 사용하여 Lambda 함수 생성의 일부로 생성된 IAM 역할(`GetStartedLambdaIntegrationRole`)을 볼 수 있습니다. 이 IAM 역할에는 두 개의 인라인 정책이 연결되어 있습니다. 하나는 Lambda 실행을 위해 가장 기본적인 권한을 규정합니다. Lambda 함수가 생성된 리전 내 계정의 모든 CloudWatch 리소스에 CloudWatch `CreateLogGroup` 대해 를 호출할 수 있습니다. 또한 이 정책은 `HelloWorldForLambdaIntegration` Lambda 함수에 대한 CloudWatch 스트림 생성 및 이벤트 로깅을 허용합니다.

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

다른 정책 문서는 이 예제에서 사용되지 않은 다른 AWS 서비스를 호출하는 데 적용됩니다. 지금은 건너뛸 수 있습니다.

IAM 역할에는 `lambda.amazonaws.com`인 신뢰할 수 있는 엔터티가 연결되어 있습니다. 다음은 신뢰 관계입니다.

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

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

이 신뢰 관계와 인라인 정책을 조합하면 Lambda 함수가 함수를 호출하여 Logs에 이벤트를 기록할 수 `console.log()` 있습니다. CloudWatch

를 사용하여 Lambda 함수를 생성하지 않은 경우, 다음 예제에 따라 필요한 IAM 역할 및 정책을 생성한 다음 역할을 함수에 수동으로 연결해야 합니다. AWS Management Console

Lambda 비 프록시 통합을 통해 API 생성

Lambda 함수(`GetStartedLambdaIntegration`)가 생성 및 테스트되면 API Gateway API를 통해 함수를 공개할 준비가 된 것입니다. 설명을 위해 일반 HTTP 메서드를 사용하여 Lambda 함수를 공개합니다. 요청 본문, URL 경로 변수, 쿼리 문자열 및 헤더를 사용하여 클라이언트로부터 필수 입력 데이터를 수신합니다. API에 대한 API Gateway 요청 검사기를 설정하여 모든 필수 데이터가 올바르게 정의되고 지정되었는지 확인합니다. 백엔드 Lambda 함수에 따라 클라이언트가 제공한 요청 데이터를 유효한 형식으로 변환하도록 API Gateway에 대한 매핑 템플릿을 구성합니다.

Lambda 비 프록시 통합을 통해 API를 생성하려면

1. https://console.aws.amazon.com/apigateway에서 API Gateway 콘솔에 로그인합니다.
2. API Gateway를 처음 사용하는 경우, 서비스의 기능을 소개하는 페이지가 나타납니다. REST API에서 빌드를 선택합니다. 예제 API 생성 팝업이 나타나면 확인을 선택합니다.

   API Gateway를 처음 사용하는 것이 아닌 경우 API 생성을 선택합니다. REST API에서 빌드를 선택합니다.
3. API 이름에서 **`LambdaNonProxyAPI`**을(를) 입력합니다.
4. (선택 사항) 설명에 설명을 입력합니다.
5. API 엔드포인트 유형 설정을 지역으로 유지합니다.
6. API 생성(Create API)을 선택합니다.

API를 생성한 후에는 /{city} 리소스를 생성합니다. 이는 클라이언트에서 입력을 가져오는 경로 변수를 포함하는 리소스의 예입니다. 나중에 이 경로 변수를 매핑 템플릿을 사용하여 Lambda 함수 입력에 매핑합니다.

리소스를 생성하려면

1. 리소스 생성을 선택합니다.
2. 프록시 리소스는 꺼진 상태로 둡니다.
3. 리소스 경로를 /로 유지합니다.
4. 리소스 이름에 **{city}**을 입력합니다.
5. 오리진 간 리소스 공유(CORS)를 꺼진 상태로 둡니다.
6. 리소스 생성을 선택합니다.

/{city} 리소스를 생성한 후에는 ANY 메서드를 생성합니다. ANY HTTP 동사는 클라이언트가 런타임에 제출하는 유효한 HTTP 메서드의 자리 표시자입니다. 이 예제에서는 Lambda 사용자 지정 통합뿐만 아니라 Lambda 프록시 통합에 사용할 수 있는 ANY 메서드를 보여줍니다.

ANY 메서드를 생성하려면

1. /{city} 리소스를 선택한 다음 메서드 생성을 선택합니다.
2. 메서드 유형에서 ANY를 선택합니다.
3. 통합 유형에서 Lambda 함수를 선택합니다.
4. Lambda 프록시 통합을 비활성화된 상태로 유지합니다.
5. Lambda 함수의 경우, Lambda 함수를 생성한 위치를 선택한 다음 AWS 리전 함수 이름을 입력합니다.
6. 기본 제한 시간 값인 29초를 사용하려면 기본 제한 시간을 활성화된 상태로 유지합니다. 사용자 지정 제한 시간을 설정하려면 기본 제한 시간을 선택하고 `50` ~ `29000`밀리초 사이의 제한 시간 값을 입력합니다.
7. 메서드 생성을 선택합니다.

ANY 메서드를 생성한 후에는 URL 경로 변수, 쿼리 문자열 및 헤더에 대한 요청 검사기를 설정하여 모든 필수 데이터가 올바르게 정의되고 지정되었는지 확인합니다. 이 예시에서는 `time` 쿼리 문자열 파라미터와 `day` 헤더를 생성합니다.

요청 검사기를 켜려면

1. 메서드 요청 탭의 메서드 요청 설정에서 편집을 선택합니다.
2. 요청 검사기에서 쿼리 문자열 파라미터 및 헤더 검사를 선택합니다.
3. URL 쿼리 문자열 파라미터를 선택하고 다음을 수행합니다.

    a. 쿼리 문자열 추가(Add query string)를 선택합니다.
    b. 이름에 **time**를 입력합니다.
    c. 필수를 끕니다.
    d. 캐싱을 꺼진 상태로 둡니다.
4. HTTP 요청 헤더를 선택하고 다음을 수행합니다.

    a. 헤더 추가(Add header)를 선택합니다.
    b. 이름에 **day**를 입력합니다.
    c. 필수를 끕니다.
    d. 캐싱을 꺼진 상태로 둡니다.
5. 저장을 선택합니다.

요청 검사기를 활성화한 후에는 백엔드 Lambda 함수에 필요한 대로 수신된 요청을 JSON 페이로드로 변환하는 본문 매핑 템플릿을 추가하여 ANY 메서드에 대한 통합 요청을 구성합니다.

통합 요청을 구성하려면

1. 통합 요청 탭의 통합 요청 설정에서 편집을 선택합니다.
2. 요청 본문 패스스루에서 정의된 템플릿이 없는 경우(권장)를 선택합니다.
3. 매핑 템플릿을 선택합니다.
4. 매핑 템플릿 추가(Add mapping template)를 선택합니다.
5. 콘텐츠 유형에 **application/json**을 입력합니다.
6. 템플릿 본문에 다음 코드를 입력합니다.

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

```
}
```

7. 저장을 선택합니다.

API 메서드 호출 테스트

API Gateway 콘솔은 배포되기 전에 API 호출을 테스트할 수 있는 테스트 기능을 제공합니다. 콘솔의 테스트 기능을 사용하여 다음 요청을 제출하는 방식으로 API를 테스트합니다.

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

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

이 테스트 요청에서는 ANY를 POST로 설정하고 {city}를 Seattle로 설정하며 Wednesday를 day 헤더 값으로 할당하고 "John"을 callerName 값으로 할당합니다.

ANY 메서드를 테스트하는 방법

1. 테스트 탭을 선택합니다. 탭을 표시하려면 오른쪽 화살표 버튼을 선택해야 할 수도 있습니다.
2. 메서드 유형에서 POST를 선택합니다.
3. 경로의 city에 **Seattle**을 입력합니다.
4. 쿼리 문자열에 **time=morning**를 입력합니다.
5. 헤더에 **day:Wednesday**를 입력합니다.
6. 요청 본문에 **{ "callerName": "John" }**을 입력합니다.
7. 테스트를 선택합니다.

반환된 응답 페이로드가 다음과 같은지 확인합니다.

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

로그를 보고 API Gateway에서 요청 및 응답을 처리하는 방식을 알아볼 수도 있습니다.

```
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
```

로그는 매핑 전에 수신되는 요청과 매핑 후 통합 요청을 보여줍니다. 테스트가 실패하면 로그는 원래 입력이 올바르거나 매핑 템플릿이 올바르게 작동하는지 여부를 평가하는 데 유용합니다.

API 배포

테스트 호출은 시뮬레이션이며 제한이 있습니다. 예를 들어 테스트 호출은 API에서 제정된 권한 부여 메커니즘을 우회합니다. 실시간으로 API 실행을 테스트하려면 먼저 API를 배포해야 합니다. API를 배포하려면 그 시점에 API의 스냅샷을 생성하는 단계를 생성합니다. 단계 이름은 API의 기본 호스트 이름 뒤에 기본 경로도 정의합니다. API의 루트 리소스는 단계 이름 뒤에 추가됩니다. API를 수정할 경우 변경 사항이 적용되기 전에 API를 새 단계 또는 기존 단계에 다시 배포해야 합니다.

API를 단계에 배포하려면

1. Deploy API(API 배포)를 선택합니다.
2. 스테이지에서 새 스테이지를 선택합니다.
3. 단계 이름에 **`test`**를 입력합니다.

   > **Note**
   >
   > 입력은 UTF-8로 인코딩된(현지화되지 않은) 텍스트여야 합니다.

4. (선택 사항) 설명에 설명을 입력합니다.
5. 배포를 선택합니다.

스테이지 세부 정보에서 복사 아이콘을 선택하여 API의 호출 URL을 복사합니다. API 기본 URL의 일반적인 패턴은 `https://`*`api-id`*`.`*`region`*`.amazonaws.com/`*`stageName`*입니다. 예를 들어 `beags1mnid` 리전에서 생성되고 `us-west-2` 단계에 배포된 API(`test`)의 기본 URL은 `https://beags1mnid.execute-api.us-west-2.amazonaws.com/test`입니다.

배포 단계에서 API 테스트

배포된 API를 테스트할 수 있는 방법에는 여러 가지가 있습니다. URL 경로 변수 또는 쿼리 문자열 파라미터만 사용하는 GET 요청의 경우 브라우저에 API 리소스 URL을 입력할 수 있습니다. 다른 메서드의 경우 POSTMAN 또는 cURL 등의 고급 REST API 테스트 유틸리티를 사용해야 합니다.

cURL을 사용하여 API를 테스트하려면

1. 인터넷에 연결된 로컬 컴퓨터에서 터미널 창을 엽니다.
2. `POST /Seattle?time=evening`을 테스트하려면 다음을 수행합니다.

   다음 cURL 명령을 복사하여 터미널 창에 붙여 넣습니다.

   ```
   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"
   }'
   ```

   다음 페이로드와 함께 성공적인 응답을 받아야 합니다.

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

   이 메서드 요청에서 POST를 PUT으로 변경할 경우 동일한 응답을 받습니다.

정리

이제 이 연습을 위해 생성한 Lambda 함수가 더 이상 필요하지 않은 경우 해당 함수를 삭제할 수 있습니다. 함께 제공되는 IAM 리소스도 삭제할 수 있습니다.

> **Warning**
>
> 이 시리즈의 다른 연습을 완료하려면 Lambda 실행 역할 또는 Lambda 호출 역할을 삭제하지 마십시오. API가 의존하는 Lambda 함수를 삭제할 경우 해당 API가 더 이상 작동하지 않습니다. Lambda 함수 삭제는 실행 취소할 수 없습니다. Lambda 함수를 다시 사용하려면 함수를 재생성해야 합니다.
>
> Lambda 함수가 의존하는 IAM 리소스를 삭제할 경우 Lambda 함수가 더 이상 작동하지 않습니다. 따라서 해당 함수에 의존하는 모든 API가 더 이상 작동하지 않게 됩니다. IAM 리소스 삭제는 실행 취소할 수 없습니다. IAM 리소스를 다시 사용하려면 리소스를 재생성해야 합니다.

Lambda 함수를 삭제하려면

1. AWS Management Console 로그인하고 https://console.aws.amazon.com/lambda/ 에서 AWS Lambda 콘솔을 엽니다.
2. 함수 목록에서 GetHelloWorld을 선택하고 작업을 선택한 다음 함수 삭제를 선택합니다. 확인 메시지가 나타나면 삭제를 다시 선택합니다.
3. 함수 목록에서 GetHelloWithName을 선택하고 작업을 선택한 다음 함수 삭제를 선택합니다. 확인 메시지가 나타나면 삭제를 다시 선택합니다.

연결된 IAM 리소스를 삭제하려면

1. https://console.aws.amazon.com/iam/에서 IAM 콘솔을 엽니다.
2. 세부 정보에서 역할을 선택합니다.
3. 역할 목록에서 API를 GatewayLambdaExecRole 선택하고 역할 작업을 선택한 다음 역할 삭제를 선택합니다. 확인 메시지가 나타나면 [Yes, Delete]를 선택합니다.
4. 세부 정보에서 정책을 선택합니다.
5. 정책 목록에서 API를 GatewayLambdaExecPolicy 선택하고 정책 작업을 선택한 다음 삭제를 선택합니다. 확인 메시지가 나타나면 삭제를 선택합니다.

자습서: 예제를 가져와 REST API 생성

Amazon API Gateway 콘솔을 사용하여 PetStore 웹 사이트에 대한 HTTP 통합으로 간단한 REST API를 생성 및 테스트할 수 있습니다. API 정의는 OpenAPI 2.0 파일로 미리 구성되어 있습니다. API 정의를 API Gateway로 로드한 후 API Gateway 콘솔을 사용하여 API의 기본 구조를 검사하거나 API를 간단히 배포 및 테스트할 수 있습니다.

PetStore 예제 API는 클라이언트가 HTTP 백엔드 웹 사이트 `http://petstore-demo-endpoint.execute-api.com/petstore/pets`에 액세스할 수 있도록 다음 메서드를 지원합니다.

> **Note**
>
> 이 튜토리얼에서는 HTTP 엔드포인트를 예로 사용합니다. 자체 API를 생성할 때는 HTTP 통합에 HTTPS 엔드포인트를 사용하는 것이 좋습니다.

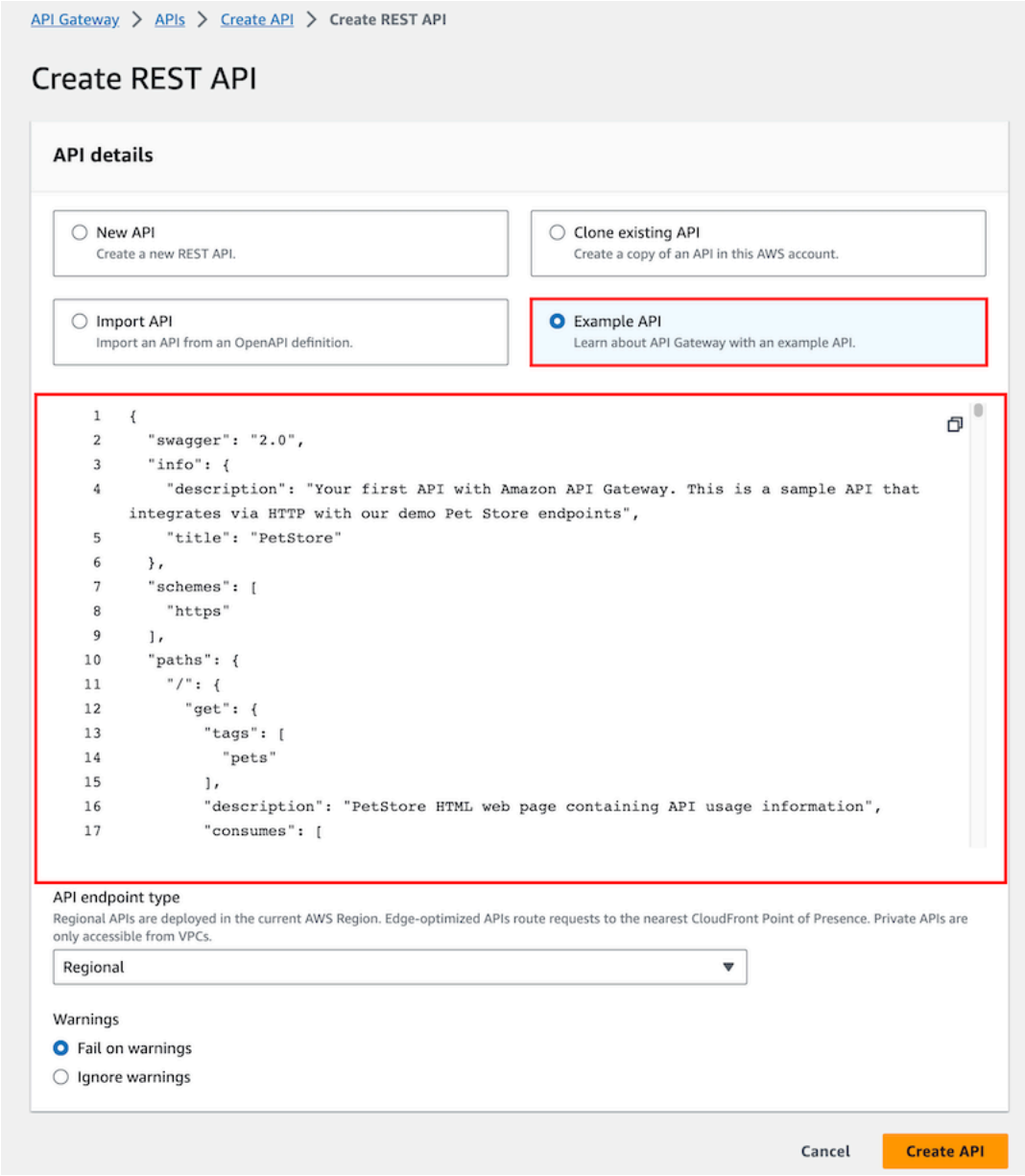
- `GET /`: 백엔드 엔드포인트와 통합되지 않은 API의 루트 리소스에 대한 읽기 액세스용입니다. API Gateway는 PetStore 웹 사이트의 개요로 응답합니다. 이는 `MOCK` 통합 유형의 예제입니다.
- `GET /pets`: 비슷한 이름의 백엔드 `/pets` 리소스와 통합된 API의 `/pets` 리소스에 대한 읽기 액세스용입니다. 백엔드는 PetStore에서 사용 가능한 반려 동물의 페이지를 반환합니다. 이는 `HTTP` 통합 유형의 예제입니다. 통합 엔드포인트의 URL은 `http://petstore-demo-endpoint.execute-api.com/petstore/pets`입니다.
- `POST /pets`: 백엔드 `/pets` 리소스와 통합된 API의 `/petstore/pets` 리소스에 대한 쓰기 액세스용입니다. 올바른 요청을 수신하면 백엔드는 지정된 반려 동물을 PetStore에 추가하고 결과를 호출자에게 반환합니다. 통합은 `HTTP`이기도 합니다.
- `GET /pets/{petId}`: 수신되는 요청 URL의 경로 변수로 지정된 `petId` 값으로 식별되는 반려 동물에 대한 읽기 액세스용입니다. 이 메서드에는 `HTTP` 통합 유형도 있습니다. 백엔드는 PetStore에 있는 지정된 반려 동물을 반환합니다. 백엔드 HTTP 엔드포인트의 URL은 `http://petstore-demo-endpoint.execute-api.com/petstore/pets/`*n*이며 여기서 n은 쿼리된 반려 동물의 식별자로서 정수입니다.

API는 `OPTIONS` 통합 유형의 `MOCK` 메서드를 통해 CORS 액세스를 지원합니다. API Gateway는 CORS 액세스를 지원하는 필수 헤더를 반환합니다.

다음 절차에서는 API Gateway 콘솔을 사용하여 예제 API를 생성하고 테스트하는 단계를 안내합니다.

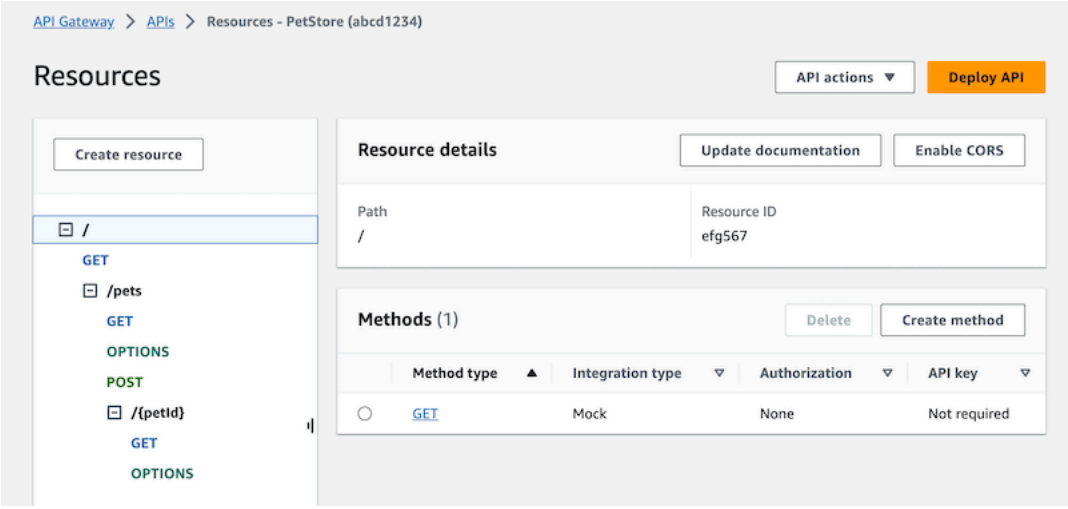
예제 API를 가져와서 구축 및 테스트하려면

1. https://console.aws.amazon.com/apigateway에서 API Gateway 콘솔에 로그인합니다.
2. 다음 중 하나를 수행하십시오.
   - 첫 번째 API를 생성하려면 REST API에서 빌드를 선택합니다.
   - 이전에 API를 생성한 경우 API 생성을 선택한 다음 REST API에서 빌드를 선택합니다.
3. REST API 생성에서 예제 API를 선택한 다음 API 생성을 선택하여 예제 API를 생성합니다.



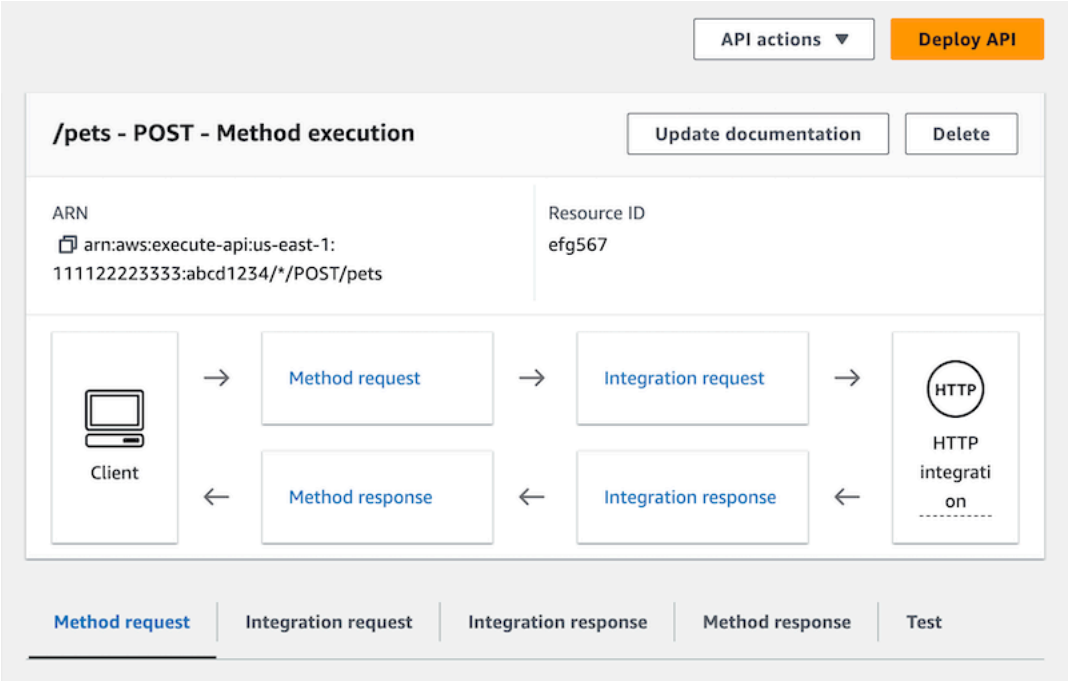
API 생성을 선택하기 전에 OpenAPI 정의를 아래로 스크롤하여 이 예제 API의 세부 정보를 확인할 수 있습니다.

4. 기본 탐색 창에서 리소스를 선택합니다. 새로 생성한 API는 다음과 같이 표시됩니다.



리소스 창은 생성된 API의 구조를 노드 트리 형태로 보여 줍니다. 각 리소스에 정의된 API 메서드가 트리의 엣지입니다. 리소스를 선택하면 모든 메서드가 우측의 방법 테이블에 나열됩니다. 각 메서드와 함께 메서드 유형, 통합 유형, 권한 부여 유형 및 API 키 요구 사항이 표시됩니다.

5. 메서드의 세부 정보를 보거나 메서드 설정을 수정하거나 메서드 호출을 테스트하려면, 메서드 목록이나 리소스 트리에서 메서드 이름을 선택합니다. 여기에서는 그림과 같이 `POST /pets` 메서드를 선택합니다.



결과로 나타나는 메서드 실행 창은 선택된 (`POST /pets`) 메서드의 구조 및 동작에 대한 논리적인 뷰를 보여줍니다.

메서드 요청 및 메서드 응답은 API와 프런트엔드 간의 인터페이스를 나타내며, 통합 요청 및 통합 응답은 API와 백엔드 간의 인터페이스를 나타냅니다.

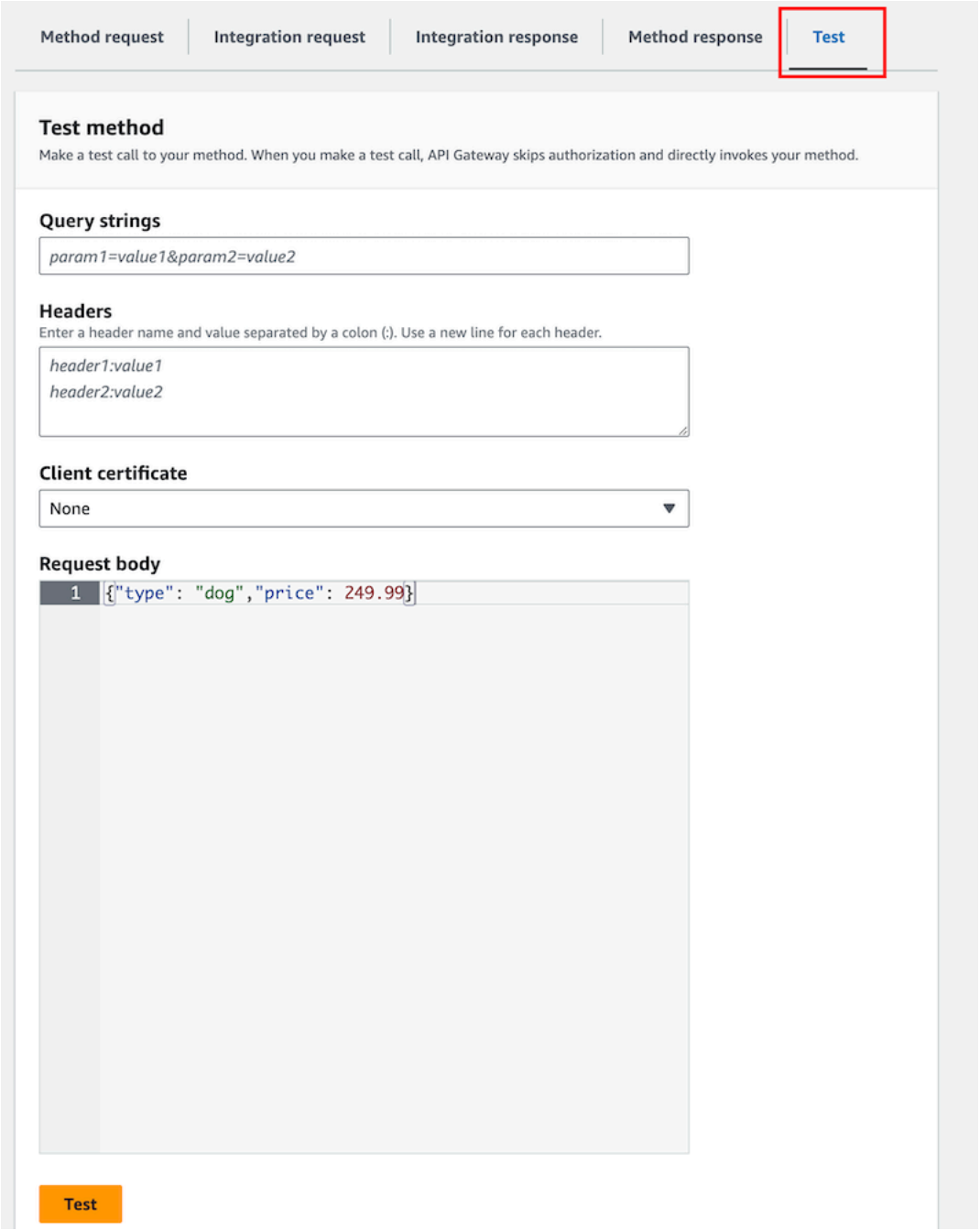
클라이언트는 API를 사용하여 메서드 요청을 통해 백엔드 기능에 액세스합니다. 필요할 경우 수신되는 요청을 백엔드에 전달하기 전에 API Gateway가 통합 요청에서 클라이언트 요청을 백엔드가 수용할 수 있는 형태로 변환합니다. 변환된 요청을 통합 요청이라고 합니다. 마찬가지로 백엔드는 통합 응답에서 응답을 API Gateway로 반환합니다. 그런 다음 API Gateway가 메서드 응답을 라우팅한 후에 클라이언트로 전송합니다. 필요한 경우 API Gateway는 백엔드 응답 데이터를 클라이언트가 기대하는 형태로 변환할 수 있습니다.

API 리소스에서의 POST 메서드의 경우, 메서드 요청의 페이로드가 통합 요청의 페이로드와 같은 형식인 경우 메서드 요청 페이로드를 수정 없이 통합 요청에 전달할 수 있습니다.

GET / 메서드 요청은 MOCK 통합 유형을 사용하며 실제 백엔드 엔드포인트에 연동되지 않습니다. 해당 통합 응답을 설정하여 정적인 HTML 페이지를 반환합니다. 메서드가 호출되면 API Gateway가 요청을 단순히 수용하고 구성된 통합 요청을 메서드 응답 방식으로 클라이언트에 즉시 반환합니다. 모의 통합을 사용하여 백엔드 엔드포인트를 요구하지 않고 API를 테스트할 수 있습니다. 응답 본문 매핑 템플릿에서 생성된 로컬 응답도 모의 통합을 사용하여 제공할 수 있습니다.

API 개발자는 메서드 요청 및 메서드 응답을 구성하여 API의 프런트엔드 상호 작용의 동작을 제어합니다. API의 백엔드 상호 작용 동작은 통합 요청 및 통합 응답을 설정하여 제어합니다. 여기에는 메서드와 해당 통합 간의 데이터 매핑 작업이 수반됩니다. 지금은 API의 종단 간 사용자 경험을 테스트하는 데 중점을 두겠습니다.

6. 테스트 탭을 선택합니다. 탭을 표시하려면 오른쪽 화살표 버튼을 선택해야 할 수도 있습니다.
7. 예를 들어, POST /pets 메서드를 테스트하려면 다음의 **{"type": "dog","price": 249.99}** 페이로드를 요청 본문에 입력한 후 테스트를 선택합니다.



이 입력은 PetStore 웹 사이트 상의 반려 동물 목록에 추가하려는 반려 동물의 속성을 지정합니다.

8. 결과는 다음과 같습니다.

/pets - POST method test results

Request	Latency	Status
/pets	10	200

Response body

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

Response headers

```
{
  "Access-Control-Allow-Origin": "*",
  "Content-Type": "application/json",
  "X-Amzn-Trace-Id": "Root=1-abcdefg1234"
}
```

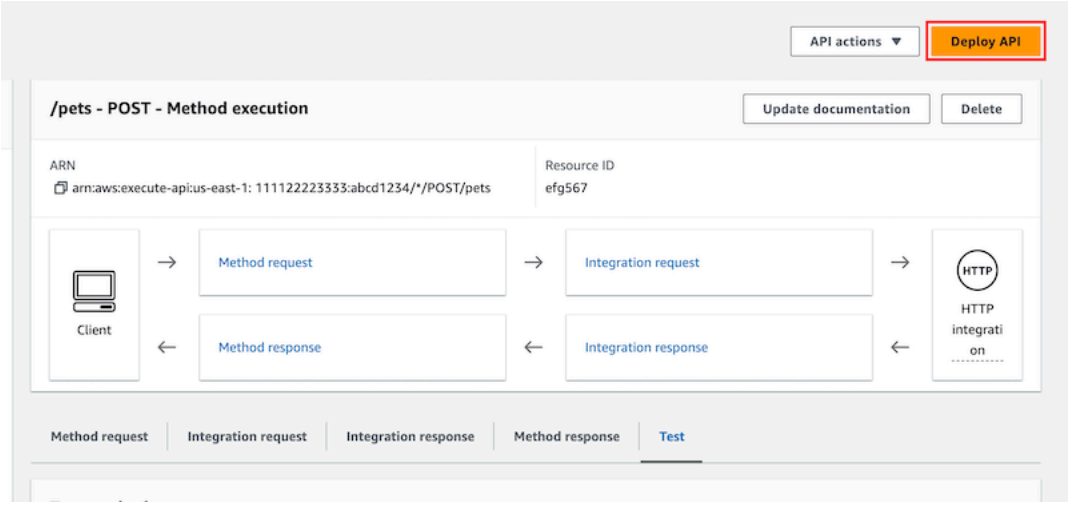
Log

```
Execution log for request 7fb3f811-44c6-4c3f-8842-bfacacf5cf6b
Tue Oct 24 22:44:48 UTC 2023 : Starting execution for request: 7fb3f811-44c6-4c3f-8842-bfacacf5cf6b
Tue Oct 24 22:44:48 UTC 2023 : HTTP Method: POST, Resource Path: /pets
Tue Oct 24 22:44:48 UTC 2023 : Method request path: {}
Tue Oct 24 22:44:48 UTC 2023 : Method request query string: {}
Tue Oct 24 22:44:48 UTC 2023 : Method request headers: {}
Tue Oct 24 22:44:48 UTC 2023 : Method request body before transformations: {"type": "dog","price": 249.99}
Tue Oct 24 22:44:48 UTC 2023 : Endpoint request URI: http://petstore.execute-api.us-east-
1.amazonaws.com/petstore/pets
Tue Oct 24 22:44:48 UTC 2023 : Endpoint request headers: {x-amzn-apigateway-api-id=abcd1234,
Accept=application/json, User-Agent=AmazonAPIGateway_abcd1234, X-Amzn-Trace-Id=Root=1-abcdefg1234, Content-
Type=application/json}
Tue Oct 24 22:44:48 UTC 2023 : Endpoint request body after transformations: {"type": "dog","price": 249.99}
Tue Oct 24 22:44:48 UTC 2023 : Sending request to http://petstore.execute-api.us-east-
1.amazonaws.com/petstore/pets
Tue Oct 24 22:44:48 UTC 2023 : Received response. Status: 200, Integration latency: 4 ms
Tue Oct 24 22:44:48 UTC 2023 : Endpoint response headers: {Date=Tue, 24 Oct 2023 22:44:48 GMT, Content-
Type=application/json; charset=utf-8, Content-Length=81, Connection=keep-alive, X-Powered-By=Express}
Tue Oct 24 22:44:48 UTC 2023 : Endpoint response body before transformations: {
  "pet": {
    "type": "dog",
    "price": 249.99
  },
```

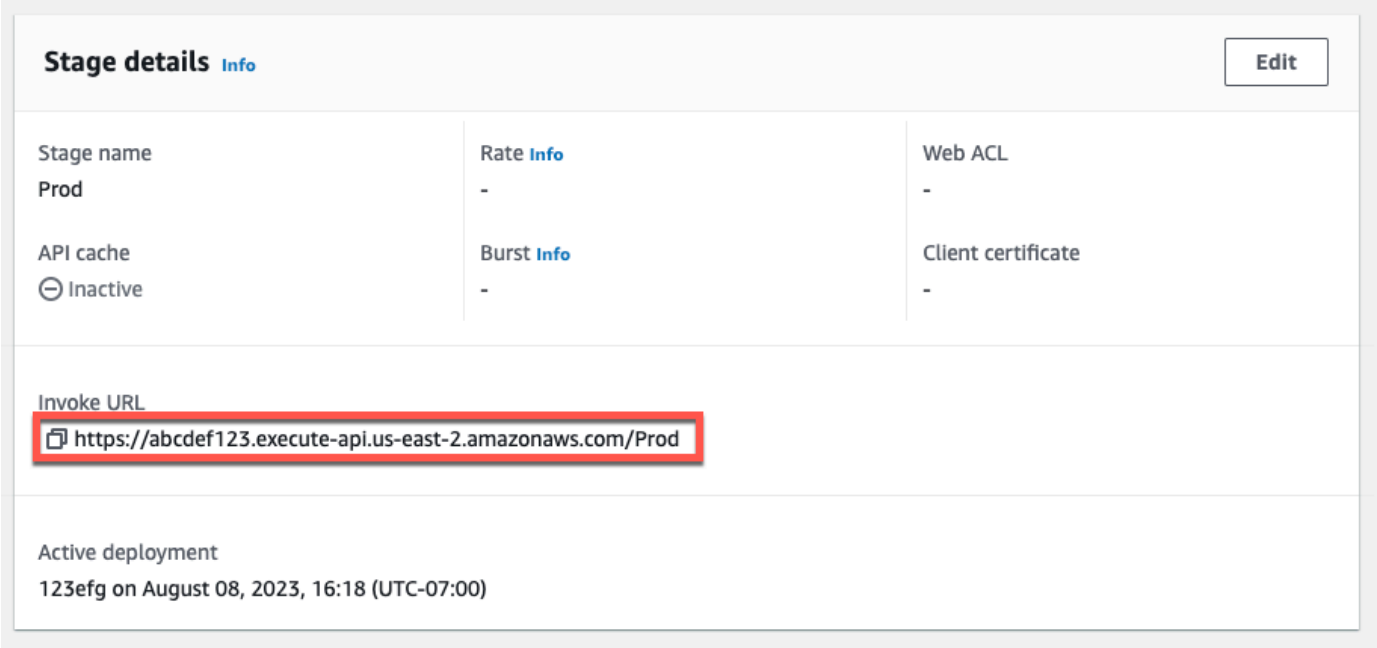
출력의 로그 항목은 메서드 요청에서 통합 요청으로, 그리고 통합 응답에서 메서드 응답으로의 상태 변화를 나타냅니다. 이는 요청 실패를 야기하는 매핑 오류 문제를 해결하는 데 유용할 수 있습니다. 이 예제에서는 어떤 매핑도 적용되지 않습니다. 즉, 메서드 요청 페이로드가 통합 요청을 통해 백엔드로 전달되고, 마찬가지로 백엔드 응답은 통합 응답을 통해 메서드 응답으로 전달됩니다.

API Gateway의 테스트-호출-요청 기능 이외의 클라이언트를 사용하여 API를 테스트하려면 우선 API를 단계에 배포해야 합니다.

9. 샘플 API를 배포하려면 API 배포를 선택합니다.



10. 스테이지에서 새 스테이지를 선택한 다음 **test**를 입력합니다.
11. (선택 사항) 설명에 설명을 입력합니다.
12. [배포]를 선택합니다.
13. 그 결과 표시되는 스테이지 창의 스테이지 세부 정보 아래 URL 호출에 API의 `GET /` 메서드 요청을 호출하는 URL이 표시됩니다.



14. 복사 아이콘을 선택해 API의 호출 URL을 복사하여 웹 브라우저에 입력합니다. 올바른 응답은 통합 응답에서 매핑 템플릿에 의해 생성된 결과를 반환합니다.
15. Stages(단계) 탐색 창에서 테스트 단계를 확장하고 `/pets/{petId}`에 대한 GET을 선택한 다음, `https://`*`api-id`*`.execute-api.`*`region`*`.amazonaws.com/test/pets/{petId}`의 Invoke URL(URL 호출) 값을 복사합니다. `{petId}`는 경로 변수를 나타냅니다.

URL 호출(Invoke URL) 값(이전 단계에서 복사한 값)을 브라우저의 주소 표시줄에 붙여 넣어 `{petId}`을 1(예)로 대체한 다음 엔터 키를 눌러 요청을 제출합니다. 200 OK 응답이 다음의 JSON 페이로드와 함께 반환되어야 합니다.

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

그림에서와 같이 API 메서드 호출이 가능한데, 권한 부여(Authorization) 유형이 `NONE`으로 설정되어 있기 때문입니다. `AWS_IAM` 권한 부여가 사용되었다면 서명 버전 4(SigV4) 프로토콜을 사용하여 요청에 서명해야 합니다. 이러한 요청의 예는 the section called “자습서: HTTP 비 프록시 통합을 사용하여 API 빌드”을 참조하십시오.

HTTP 통합을 사용하여 API Gateway REST API 구축

HTTP 통합을 사용하여 API를 구축하기 위해 HTTP 프록시 통합 또는 HTTP 사용자 지정 통합을 사용할 수 있습니다. 다양하고 강력한 기능을 제공하면서 간소화된 API 설정을 위해 가능하면 언제든 HTTP 프록시 통합을 사용하는 것이 좋습니다. HTTP 사용자 지정 통합은 백엔드에 대한 클라이언트 요청 데이터를 변환하거나 클라이언트에 대한 백엔드 응답 데이터를 변환해야 하는 경우에 효과적일 수 있습니다.

주제

- 자습서: HTTP 프록시 통합을 사용하여 REST API 구축
- 자습서: HTTP 비 프록시 통합을 사용하여 REST API 빌드

자습서: HTTP 프록시 통합을 사용하여 REST API 구축

HTTP 프록시 통합은 웹 애플리케이션이 단일 HTTP 메서드의 간소화된 설정을 통해 전체 웹 사이트와 같은 통합된 HTTP 엔드포인트의 여러 리소스 또는 기능에 액세스할 수 있도록 하는 API를 구축하기 위한 간단하고 강력한 다목적 메커니즘입니다. HTTP 프록시 통합에서 API Gateway는 클라이언트가 제출한 메서드 요청을 백엔드로 전달합니다. 전달된 요청 데이터에는 요청 헤더, 쿼리 문자열 파라미터, URL 경로 변수 및 페이로드가 포함되어 있습니다. 백엔드 HTTP 엔드포인트 또는 웹 서버는 수신되는 요청 데이터를 구문 분석하여 반환하는 응답을 결정합니다. HTTP 프록시 통합을 통해 클라이언

트 및 백엔드가 API 메서드가 설정된 후 API Gateway의 개입 없이 직접 상호 작용할 수 있습니다. 단, the section called “중요 정보”에 나열된 지원되지 않는 문자 등 알려진 문제점의 경우는 예외입니다.

포괄적인 프록시 리소스 `{proxy+}` 및 HTTP 메서드에 대한 catch-all `ANY` 동사를 사용하면 HTTP 프록시 통합을 통해 단일 API 메서드의 API를 생성할 수 있습니다. 메서드는 공개적으로 액세스 가능한 HTTP 리소스의 전체 집합 및 웹 사이트의 작업을 공개합니다. 백엔드 웹 서버가 퍼블릭 액세스를 위해 더 많은 리소스를 열면 클라이언트는 동일한 API 설정으로 이러한 새 리소스를 사용할 수 있습니다. 이렇게 하려면 웹 사이트 개발자는 새 리소스와 각 리소스에 적용 가능한 작업을 클라이언트 개발자에게 정확히 알려야 합니다.

다음 자습서에서는 HTTP 프록시 통합을 간략하게 소개합니다. 자습서에서는 API Gateway 콘솔을 사용해 API를 생성하여 일반 프록시 리소스 `{proxy+}`를 통해 PetStore 웹 사이트와 통합하고 `ANY`의 HTTP 메서드 자리 표시자를 생성합니다.

주제

- API Gateway 콘솔을 사용하여 HTTP 프록시 통합을 통해 API 생성
- HTTP 프록시 통합을 사용하여 API 테스트

API Gateway 콘솔을 사용하여 HTTP 프록시 통합을 통해 API 생성

다음 절차에서는 API Gateway 콘솔을 사용하여 프록시 리소스를 통해 HTTP 백엔드에 대한 API를 생성한 후 테스트하는 단계를 안내합니다. HTTP 백엔드는 `PetStore`의 `http://petstore-demo-endpoint.execute-api.com/petstore/pets` 웹 사이트(자습서: HTTP 비 프록시 통합을 사용하여 REST API 빌드)이며 스크린샷은 API Gateway UI 요소를 보여주기 위한 시각적 보조 도구로 사용됩니다. 처음으로 API Gateway 콘솔을 사용하여 API를 생성하는 경우 먼저 해당 섹션을 따를 수 있습니다.

API를 생성하려면

1. https://console.aws.amazon.com/apigateway에서 API Gateway 콘솔에 로그인합니다.
2. API Gateway를 처음 사용하는 경우, 서비스의 기능을 소개하는 페이지가 나타납니다. REST API에서 빌드를 선택합니다. 예제 API 생성 팝업이 나타나면 확인을 선택합니다.

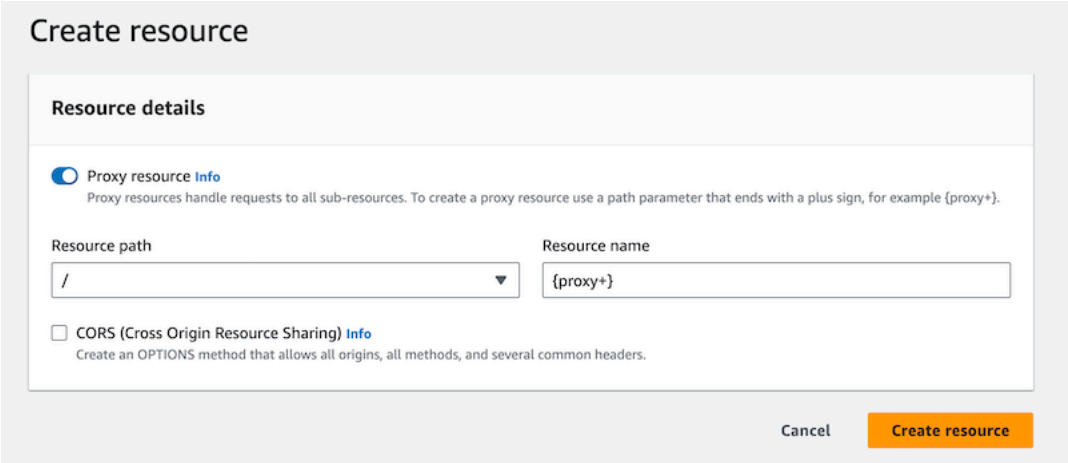
   API Gateway를 처음 사용하는 것이 아닌 경우 API 생성을 선택합니다. REST API에서 [빌드]를 선택합니다.
3. API 이름에서 **`HTTPProxyAPI`**을(를) 입력합니다.
4. (선택 사항) 설명에 설명을 입력합니다.

5. API 엔드포인트 유형 설정을 지역으로 유지합니다.
6. API 생성(Create API)을 선택합니다.

이 단계에서는 `{proxy+}`의 프록시 리소스 경로를 생성합니다. 이는 `http://petstore-demo-endpoint.execute-api.com/` 아래 백엔드 엔드포인트의 자리 표시자입니다. 예를 들어 `petstore`, `petstore/pets`, `petstore/pets/{petId}` 등일 수 있습니다. API Gateway는 `{proxy+}` 리소스 생성 시 ANY 메서드를 생성하며 런타임에 지원되는 HTTP 동사에 대한 자리 표시자 역할을 합니다.

/{proxy+} 리소스를 생성하려면

1. API를 선택합니다.
2. 기본 탐색 창에서 리소스를 선택합니다.
3. 리소스 생성을 선택합니다.
4. 프록시 리소스를 켭니다.
5. 리소스 경로를 /로 유지합니다.
6. 리소스 이름에 **`{proxy+}`**을 입력합니다.
7. 오리진 간 리소스 공유(CORS)를 꺼진 상태로 둡니다.
8. 리소스 생성을 선택합니다.



이 단계에서는 프록시 통합을 사용하여 ANY 메서드를 백엔드 HTTP 엔드포인트와 통합합니다. 프록시 통합에서 API Gateway는 API Gateway의 개입 없이 클라이언트가 제출한 메서드 요청을 백엔드로 전달합니다.

ANY 메서드를 생성하려면

1. /{proxy+} 리소스를 선택합니다.
2. ANY 메서드를 선택합니다.
3. 경고 기호 아래에서 통합 편집을 선택합니다. 통합 없이는 메서드가 있는 API를 배포할 수 없습니다.
4. 통합 유형에서 HTTP를 선택합니다.
5. HTTP 프록시 통합을 켭니다.
6. HTTP 메서드에서 ANY를 선택합니다.
7. 엔드포인트 URL에 **http://petstore-demo-endpoint.execute-api.com/{proxy}**를 입력합니다.
8. 저장을 선택합니다.

HTTP 프록시 통합을 사용하여 API 테스트

특정 클라이언트 요청의 성공 여부는 다음 사항에 따라 달라집니다.

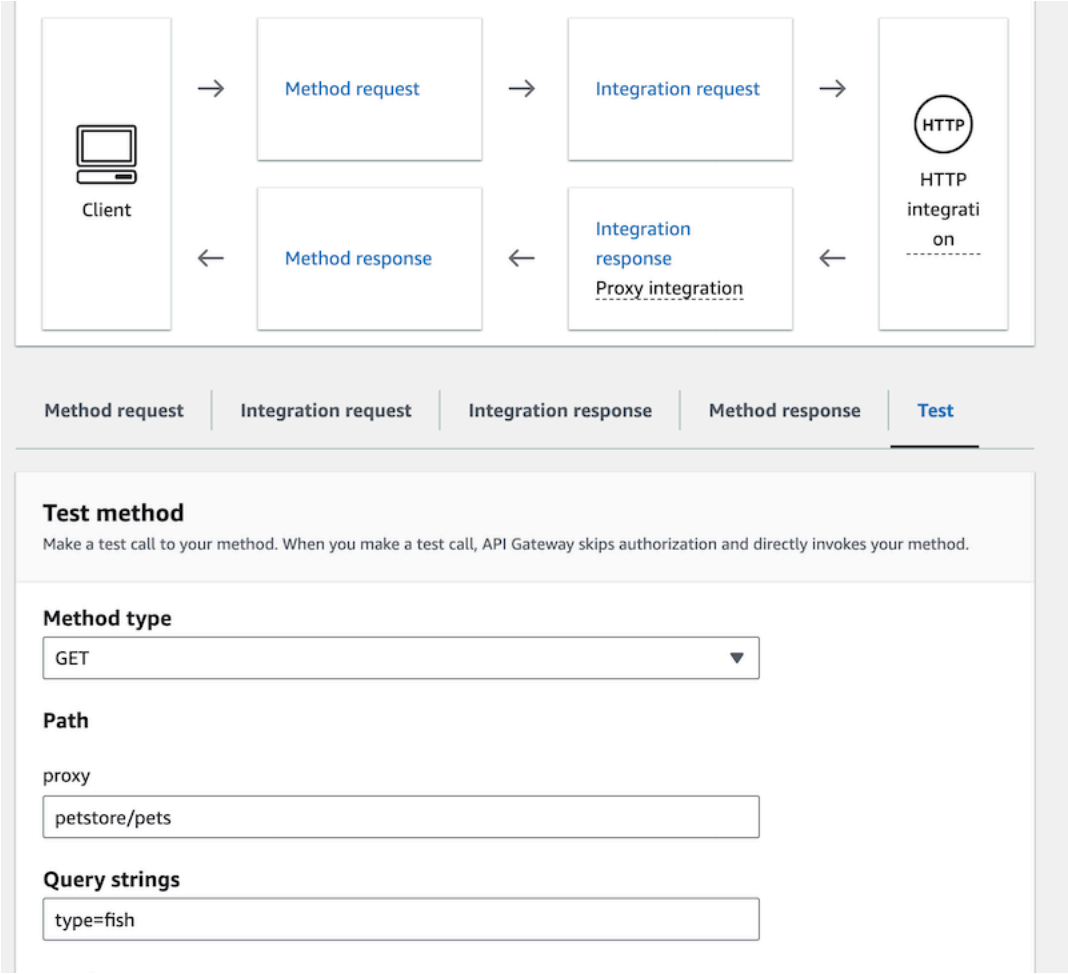
- 백엔드가 해당 백엔드 엔드포인트를 사용할 수 있도록 하고 (그러한 경우) 필수 액세스 권한을 부여한 경우.
- 클라이언트가 올바른 입력을 제공하는 경우.

예를 들어 여기에서 사용된 PetStore API는 `/petstore` 리소스를 공개하지 않습니다. 따라서 `404 Resource Not Found` 오류 메시지가 포함된 `Cannot GET /petstore` 응답이 표시됩니다.

또한 클라이언트는 결과를 올바르게 구문 분석하기 위해 백엔드의 출력 형식을 처리할 수 있어야 합니다. API Gateway는 클라이언트와 백엔드 간의 상호 작용을 원활하게 하기 위해 조정하지 않습니다.

프록시 리소스를 통해 HTTP 프록시 통합을 사용하여 PetStore 웹사이트와 통합된 API를 테스트하려면

1. 테스트 탭을 선택합니다. 탭을 표시하려면 오른쪽 화살표 버튼을 선택해야 할 수도 있습니다.
2. 메서드 유형에서 GET를 선택합니다.
3. 경로의 proxy에 **petstore/pets**를 입력합니다.
4. 쿼리 문자열에 **type=fish**를 입력합니다.
5. 테스트를 선택합니다.



백엔드 웹 사이트가 `GET /petstore/pets?type=fish` 요청을 지원하므로 다음과 비슷한 성공적인 응답을 반환합니다.

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

```
  }
]
```

`GET /petstore` 호출을 시도하면 `404` 오류 메시지와 함께 `Cannot GET /petstore` 응답이 표시됩니다. 백엔드가 지정된 작업을 지원하지 않기 때문입니다. `GET /petstore/pets/1`을 호출하면 요청이 PetStore 웹 사이트에서 지원되므로 다음 페이로드와 함께 `200 OK` 응답이 표시됩니다.

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

브라우저를 사용하여 API를 테스트할 수 있습니다. API를 배포하고 스테이지에 연결하여 API의 호출 URL을 생성합니다.

API를 배포하려면

1. Deploy API(API 배포)를 선택합니다.
2. 스테이지에서 새 스테이지를 선택합니다.
3. 단계 이름에 **`test`**를 입력합니다.
4. (선택 사항) 설명에 설명을 입력합니다.
5. [배포]를 선택합니다.

이제 클라이언트가 API를 호출할 수 있습니다.

API를 호출하려면

1. https://console.aws.amazon.com/apigateway에서 API Gateway 콘솔에 로그인합니다.
2. API를 선택합니다.
3. 기본 탐색 창에서 스테이지를 선택합니다.
4. 스테이지 세부 정보에서 복사 아이콘을 선택하여 API의 호출 URL을 복사합니다.

   API의 호출 URL을 웹 브라우저에 입력합니다.

전체 URL은 `https://abcdef123.execute-api.us-east-2.amazonaws.com/test/petstore/pets?type=fish`와(과) 같아야 합니다.

브라우저가 API에 `GET` 요청을 보냅니다.

5. 결과는 API Gateway 콘솔에서 테스트를 사용할 때 반환되는 결과와 같아야 합니다.

자습서: HTTP 비 프록시 통합을 사용하여 REST API 빌드

이 자습서에서는 Amazon API Gateway 콘솔을 사용하여 API를 완전히 새로 생성합니다. 콘솔을 API 설계 스튜디오로 간주해 이를 사용하여 API 기능의 범위를 정하고, 동작을 시험하고, API를 구축하고, 단계별로 API를 배포합니다.

주제

- HTTP 사용자 지정 통합을 사용하여 API 만들기
- (선택 사항) 요청 파라미터 매핑

HTTP 사용자 지정 통합을 사용하여 API 만들기

이 단원에서는 리소스 생성, 리소스에서 메서드 노출, 원하는 API 동작을 위한 메서드 구성, API 테스트 및 배포 등을 단계별로 살펴봅니다.

이 단계에서는 빈 API를 생성합니다. 다음 단계에서는 비프록시 HTTP 통합을 사용하여 API를 `http://petstore-demo-endpoint.execute-api.com/petstore/pets` 엔드포인트에 연결하는 리소스와 메서드를 생성합니다.

API를 생성하는 방법

1. https://console.aws.amazon.com/apigateway에서 API Gateway 콘솔에 로그인합니다.
2. API Gateway를 처음 사용하는 경우, 서비스의 기능을 소개하는 페이지가 나타납니다. REST API에서 빌드를 선택합니다. 예제 API 생성 팝업이 나타나면 확인을 선택합니다.

   API Gateway를 처음 사용하는 것이 아닌 경우 API 생성을 선택합니다. REST API에서 빌드를 선택합니다.
3. API 이름에서 **`HTTPNonProxyAPI`**을(를) 입력합니다.
4. (선택 사항) 설명에 설명을 입력합니다.
5. API 엔드포인트 유형 설정을 지역으로 유지합니다.

6. API 생성(Create API)을 선택합니다.

리소스 트리에 메서드 없는 루트 리소스(/)가 표시됩니다. 이 연습에서는 PetStore 웹 사이트(petstore-demo-endpointhttp://.execute-api.com/petstore/pets) 의 HTTP 사용자 지정 통합을 사용하여 API를 구축할 것입니다. 설명을 위해 루트의 자식으로 리소스를 만들고 이 /pets 리소스에 GET 메서드를 노출시켜 클라이언트가 웹 사이트에서 사용 가능한 Pets 항목 목록을 검색할 수 있도록 하겠습니다. PetStore

/pets 리소스를 생성하려면

1. / 리소스를 선택한 다음 리소스 생성을 선택합니다.
2. 프록시 리소스는 꺼진 상태로 둡니다.
3. 리소스 경로를 /로 유지합니다.
4. 리소스 이름에 **pets**을 입력합니다.
5. 오리진 간 리소스 공유(CORS)를 꺼진 상태로 둡니다.
6. 리소스 생성을 선택합니다.

이 단계에서는 /pets 리소스의 GET 메서드를 생성합니다. GET 메서드는 http://petstore-demo-endpoint.execute-api.com/petstore/pets 웹 사이트와 통합되어 있습니다. API 메서드의 다른 옵션에는 다음이 포함됩니다.

- POST는 주로 하위 리소스를 생성하는 데 사용됩니다.
- PUT은 주로 기존 리소스를 업데이트하는 데 사용되며, 권장되지는 않지만 하위 리소스를 생성하는 데에도 사용될 수 있습니다.
- DELETE는 리소스를 삭제하는 데 사용됩니다.
- PATCH는 리소스를 업데이트하는 데 사용됩니다.
- HEAD는 주로 시나리오를 테스트하는 데 사용됩니다. GET와 동일하지만 리소스 표현을 반환하지 않습니다.
- OPTIONS는 호출자가 타겟 서비스에 사용 가능한 통신 옵션에 대한 정보를 가져오는 데 사용할 수 있습니다.

통합 요청의 HTTP 메서드(HTTP method)에 백엔드에서 지원하는 항목을 선택해야 합니다. HTTP 또는 Mock integration의 경우, 메서드 요청과 통합 요청이 동일한 HTTP 동사를 사용하는 것이 합리적입니다. 다른 통합 유형 및 메서드 요청의 경우, 통합 요청과 다른 HTTP 동사를 사용할 가능성이 높습

니다. 예를 들어 Lambda 함수를 호출하기 위해 통합 요청은 POST를 이용해 함수를 호출해야 하는 반면, 메서드 요청은 Lambda 함수의 로직에 따라 임의의 HTTP 동사를 사용할 수 있습니다.

/pets 리소스에 **GET** 메서드를 생성하려면

1. /pets 리소스를 선택합니다.
2. 메서드 생성을 선택합니다.
3. 메서드 유형에서 GET을 선택합니다.
4. 통합 유형에서 HTTP 통합을 선택합니다.
5. HTTP 프록시 통합을 꺼진 상태로 둡니다.
6. HTTP 메서드에서 GET을 선택합니다.
7. 엔드포인트 URL에 **http://petstore-demo-endpoint.execute-api.com/petstore/pets**를 입력합니다.
8. 콘텐츠 처리에서 패스스루를 선택합니다.
9. 기본 제한 시간을 켜진 상태로 둡니다.
10. 메서드 생성을 선택합니다.

PetStore 웹사이트에서는 특정 페이지에서 애완동물 유형 (예: “개” 또는 “고양이”) 별로 Pet 항목 목록을 검색할 수 있습니다. 웹사이트는 type 및 page 쿼리 문자열 파라미터를 사용해 이러한 입력을 수락합니다. 이와 같이 메서드 요청에 쿼리 문자열 파라미터를 추가하고, 통합 요청의 해당 쿼리 문자열에 매핑해야 합니다.

쿼리 문자열 파라미터를 **GET** 메서드에 추가하려면

1. 메서드 요청 탭의 메서드 요청 설정에서 편집을 선택합니다.
2. URL 쿼리 문자열 파라미터를 선택하고 다음을 수행합니다.

   a. 쿼리 문자열 추가(Add query string)를 선택합니다.

   b. 이름에 **type**을 입력합니다.

   c. 필수 상태로 유지하고 캐싱을 해제합니다.

   이전 단계를 반복하여 이름이 **page**인 추가 쿼리 문자열을 생성합니다.
3. 저장을 선택합니다.

이제 클라이언트는 요청을 제출할 때 반려 동물 유형과 페이지 번호를 쿼리 문자열 파라미터로 제공할 수 있습니다. 입력 값을 백엔드의 당사 PetStore 웹 사이트에 전달하려면 이러한 입력 매개변수를 통합의 쿼리 문자열 매개변수에 매핑해야 합니다.

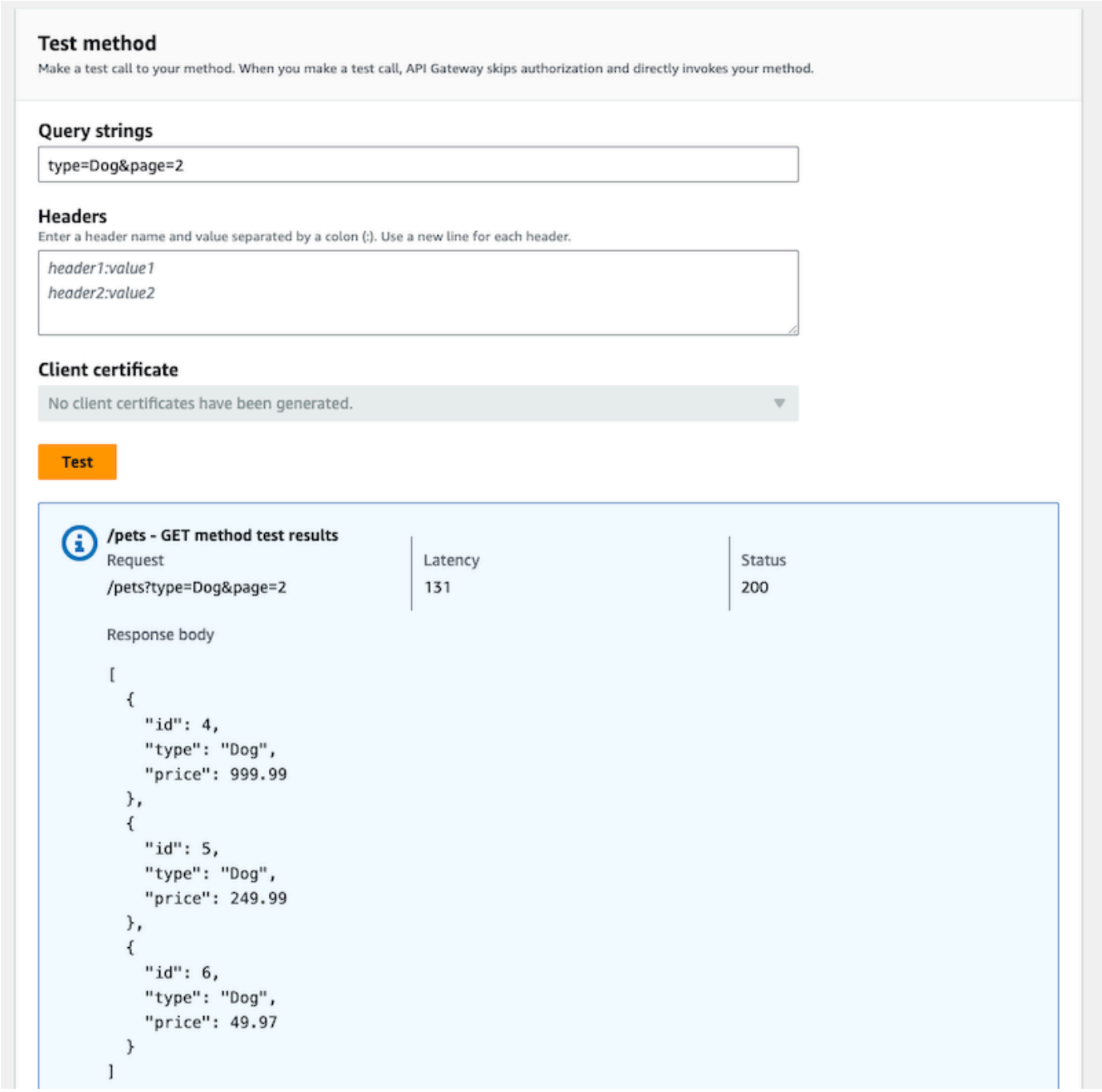
입력 파라미터를 통합 요청에 매핑하려면

1. 통합 요청 탭의 통합 요청 설정에서 편집을 선택합니다.
2. URL 쿼리 문자열 파라미터를 선택하고 다음을 수행합니다.
   a. 쿼리 문자열 파라미터 추가를 선택합니다.
   b. 이름에 **type**를 입력합니다.
   c. 다음에서 매핑됨에 **method.request.querystring.type**을 입력합니다.
   d. 캐싱을 꺼진 상태로 둡니다.
   e. 쿼리 문자열 파라미터 추가를 선택합니다.
   f. 이름에 **page**를 입력합니다.
   g. 다음에서 매핑됨에 **method.request.querystring.page**을 입력합니다.
   h. 캐싱을 꺼진 상태로 둡니다.
3. 저장을 선택합니다.

API를 테스트하려면

1. 테스트 탭을 선택합니다. 탭을 표시하려면 오른쪽 화살표 버튼을 선택해야 할 수도 있습니다.
2. 쿼리 문자열에 **type=Dog&page=2**를 입력합니다.
3. 테스트를 선택합니다.

   그 결과는 다음과 비슷합니다.



테스트에 성공했으므로, API를 배포해 공개적으로 사용 가능하도록 할 수 있습니다.

4. Deploy API(API 배포)를 선택합니다.
5. 스테이지에서 새 스테이지를 선택합니다.
6. 단계 이름에 **`Prod`**를 입력합니다.
7. (선택 사항) 설명에 설명을 입력합니다.
8. 배포를 선택합니다.

9. (선택 사항) 스테이지 세부 정보의 URL 호출에서 복사 아이콘을 선택하여 API의 호출 URL을 복사합니다. 이 URL을 Postman 및 cURL과 같은 도구에서 사용하여 API를 테스트할 수 있습니다.

SDK를 이용해 클라이언트를 생성하면 SDK가 공개한 메서드를 호출하여 요청을 보낼 수 있습니다. 구현 세부 정보는 선택한 AWS SDK를 참조하세요.

> **Note**
>
> API가 변경되면 API를 다시 배포하여 새 기능 또는 업데이트된 기능을 사용할 수 있도록 한 후 요청 URL을 다시 호출해야 합니다.

(선택 사항) 요청 파라미터 매핑

API Gateway API에 대한 요청 파라미터 매핑

이 자습서에서는 항목 ID를 지정하고, 이를 통합 요청 URL의 `{petId}` 경로 파라미터에 매핑하고, 요청을 HTTP 엔드포인트로 보내기 위해 API의 메서드 요청 URL에서 `{id}`의 경로 파라미터를 생성하는 방법을 보여줍니다.

> **Note**
>
> 대문자 대신 소문자를 입력하거나 그 반대로 입력하면 연습 뒷부분에서 오류가 발생할 수 있습니다.

1단계: 리소스 생성

이 단계에서는 경로 파라미터 {petid}를 사용하여 리소스를 생성합니다.

{petId} 리소스를 생성하려면

1. /pets 리소스를 선택한 다음 리소스 생성을 선택합니다.
2. 프록시 리소스는 꺼진 상태로 둡니다.
3. 리소스 경로에서 /pets/를 선택합니다.
4. 리소스 이름에 **`{petId}`**을 입력합니다.

   /pets/{petId}가 표시되도록 `petId`에 중괄호(`{ }`)를 사용합니다.

5. 오리진 간 리소스 공유(CORS)를 꺼진 상태로 둡니다.
6. 리소스 생성을 선택합니다.

2단계: 메서드 생성 및 테스트

이 단계에서는 `{petId}` 경로 파라미터를 사용하여 GET 메서드를 생성합니다.

GET 메서드를 설정하려면

1. /{petId} 리소스를 선택한 다음 메서드 생성을 선택합니다.
2. 메서드 유형에서 GET을 선택합니다.
3. 통합 유형에서 HTTP 통합을 선택합니다.
4. HTTP 프록시 통합을 꺼진 상태로 둡니다.
5. HTTP 메서드에서 GET을 선택합니다.
6. 엔드포인트 URL에 **`http://petstore-demo-endpoint.execute-api.com/petstore/pets/{id}`**를 입력합니다.
7. 콘텐츠 처리에서 패스스루를 선택합니다.
8. 기본 제한 시간을 켜진 상태로 둡니다.
9. 메서드 생성을 선택합니다.

이제 `{petId}` 경로 파라미터를 HTTP 엔드포인트의 `{id}` 경로 파라미터에 매핑합니다.

`{petId}` 경로 파라미터를 매핑하려면

1. 통합 요청 탭의 통합 요청 설정에서 편집을 선택합니다.
2. URL 경로 파라미터를 선택합니다.
3. API Gateway는 통합 요청에 대해 petId라는 경로 파라미터를 생성합니다. 이는 백엔드에서는 작동하지 않습니다. HTTP 엔드포인트는 경로 파라미터로 `{id}`를 사용합니다. petId의 이름을 **`id`**로 변경합니다.

   그렇게 하면 `petId`의 메서드 요청 경로 파리미터가 `id`의 통합 요청 경로 파라미터로 매핑됩니다.
4. 저장을 선택합니다.

이제 메서드를 테스트합니다.

메서드를 테스트하는 방법

1. 테스트 탭을 선택합니다. 탭을 표시하려면 오른쪽 화살표 버튼을 선택해야 할 수도 있습니다.
2. petId의 경로에 **4**를 입력합니다.
3. 테스트를 선택합니다.

   올바르게 수행했다면 응답 본문에 다음이 표시됩니다.

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

3단계: API 배포

이 단계에서는 API를 API Gateway 콘솔 외부에서 호출을 시작할 수 있도록 배포합니다.

API를 배포하려면

1. Deploy API(API 배포)를 선택합니다.
2. 스테이지에서 Prod를 선택합니다.
3. (선택 사항) 설명에 설명을 입력합니다.
4. [Deploy]를 선택합니다.

4단계: API 테스트

이 단계에서는 API Gateway 콘솔 외부로 이동하고 API를 사용하여 HTTP 엔드포인트에 액세스합니다.

1. 기본 탐색 창에서 스테이지를 선택합니다.
2. 스테이지 세부 정보에서 복사 아이콘을 선택하여 API의 호출 URL을 복사합니다.

   다음과 같아야 합니다.

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

3. 이 URL을 새 브라우저 탭의 주소 표시줄에 입력하고 URL에 `/pets/4`를 추가한 다음 요청을 제출합니다.
4. 브라우저에서 다음을 반환합니다.

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

다음 단계

요청 검사를 켜거나, 데이터를 변환하거나, 사용자 지정 게이트웨이 응답을 생성하여 API를 추가로 사용자 지정할 수 있습니다.

API를 사용자 지정하는 더 많은 방법은 다음 자습서를 참조하세요.

- 요청 검증에 대한 자세한 내용은 API Gateway의 기본 요청 확인 설정를 참조하십시오.
- 요청 및 응답 페이로드를 변환하는 방법에 대한 자세한 내용은 API Gateway에서 데이터 변환 설정 섹션을 참조하세요.
- 사용자 지정 게이트웨이 응답을 생성하는 방법에 대한 자세한 내용은 API Gateway 콘솔을 사용하여 REST API에 대한 게이트웨이 응답 설정 섹션을 참조하세요.

자습서: API Gateway 프라이빗 통합으로 REST API 빌드

프라이빗 통합이 포함된 API Gateway API를 생성하여 Amazon Virtual Private Cloud(Amazon VPC) 내의 HTTP/HTTPS 리소스에 액세스할 수 있는 권한을 고객에게 제공할 수 있습니다. 이러한 VPC 리소스는 VPC에서 Network Load Balancer 뒤에 있는 EC2 인스턴스의 HTTP/HTTPS 엔드포인트입니다. Network Load Balancer는 VPC 리소스를 캡슐화하고 수신되는 요청을 대상 지정된 리소스로 라우팅합니다.

클라이언트가 API를 호출하면 미리 구성된 VPC 링크를 통해 API Gateway가 Network Load Balancer에 연결합니다. VPC 링크는 VpcLink의 API Gateway 리소스에 의해 캡슐화됩니다. VPC 링크는 API 메서드 요청을 VPC 리소스에 전달하는 일을 담당하며, 백엔드 응답을 호출자에게 반환합니다. API 개발자에게 `VpcLink`는 통합 엔드포인트와 기능적으로 동등합니다.

프라이빗 통합이 포함된 API를 생성하려면 새 `VpcLink`를 만들거나, 원하는 VPC 리소스를 대상으로 하는 Network Load Balancer와 연결된 기존 VPC 링크를 선택해야 합니다. `VpcLink`를 생성하고 관리

하려면 적절한 권한이 있어야 합니다. 그런 다음 API 메서드를 설정한 다음 VpcLink 또는 HTTP를 통합 유형으로, HTTP_PROXY를 통합 연결 유형으로 설정하고 통합 VPC_LINK에서 VpcLink 식별자를 설정하여 connectionId와 통합합니다.

> **Note**
>
> Network Load Balancer와 API는 동일한 AWS 계정에서 소유해야 합니다.

빠르게 API 생성을 시작해 VPC 리소스에 액세스할 수 있도록 API Gateway 콘솔을 사용하여 프라이빗 통합이 포함된 API를 구축하기 위한 필수 단계를 안내합니다. API를 생성하기 전에 다음을 수행하십시오.

1. VPC 리소스를 만들고, 같은 리전의 계정에서 Network Load Balancer를 만들거나 선택한 다음 Network Load Balancer의 대상으로 리소스를 호스팅하는 EC2 인스턴스를 추가합니다. 자세한 내용은 API Gateway 프라이빗 통합을 위한 Network Load Balancer 설정 단원을 참조하세요.
2. 프라이빗 통합을 위한 VPC 링크를 생성할 수 있는 권한을 부여합니다. 자세한 내용은 VPC 링크를 생성할 수 있는 권한 부여 단원을 참조하세요.

대상 그룹에 구성된 VPC 리소스로 VPC 리소스 및 Network Load Balancer를 생성한 다음, 아래 지침에 따라 API를 생성하고 프라이빗 통합에 있는 VpcLink로 이를 VPC 리소스와 통합합니다.

프라이빗 통합을 통해 API를 생성하려면

1. https://console.aws.amazon.com/apigateway에서 API Gateway 콘솔에 로그인합니다.
2. API Gateway를 처음 사용하는 경우, 서비스의 기능을 소개하는 페이지가 나타납니다. REST API에서 빌드를 선택합니다. 예제 API 생성 팝업이 나타나면 확인을 선택합니다.

   API Gateway를 처음 사용하는 것이 아닌 경우 API 생성을 선택합니다. REST API에서 [빌드]를 선택합니다.
3. 엣지 최적화 또는 리전 REST API를 생성합니다.
4. 해당 API를 선택합니다.
5. 메서드 생성을 선택하고 다음을 수행합니다.

   a. 메서드 유형에서 GET를 선택합니다.

   b. 통합 유형에서 VPC 링크를 선택합니다.

   c. VPC 프록시 통합을 켭니다.

d. HTTP 메서드에서 GET을 선택합니다.

e. VPC 링크에서 [단계 변수 사용]을 선택하고 아래 텍스트 상자에 **${stageVariables.vpcLinkId}**를 입력합니다.

API를 스테이지에 배포한 후 vpcLinkId 스테이지 변수를 정의하고, 그 값을 VpcLink의 ID로 설정합니다.

f. 엔드포인트 URL에 URL(예: http://myApi.example.com)을 입력합니다.

여기서 호스트 이름(예: myApi.example.com)을 사용하여 통합 요청의 Host 헤더를 설정합니다.

g. 메서드 생성을 선택합니다.

프록시 통합의 경우, API를 배포할 준비가 됩니다. 그렇지 않다면 계속해서 적절한 메서드 응답과 통합 응답을 설정해야 합니다.

6. API 배포를 선택하고 다음을 수행합니다.

a. 스테이지에서 새 스테이지를 선택합니다.

b. 스테이지 이름에 스테이지 이름을 입력합니다.

c. (선택 사항) 설명에 설명을 입력합니다.

d. [배포]를 선택합니다.

7. 스테이지 세부 정보 섹션에서 결과 호출 URL을 기록해 둡니다. API 호출 시 필요합니다. 그 전에 vpcLinkId 단계 변수를 설정해야 합니다.

8. 스테이지 창에서 스테이지 변수 탭을 선택한 후 다음을 수행합니다.

a. 변수 관리를 선택한 다음 스테이지 변수 추가를 선택합니다.

b. 이름에 **vpcLinkId**를 입력합니다.

c. 값에 VPC_LINK의 ID(예: *gix6s7*)를 입력합니다.

d. 저장을 선택합니다.

단계 변수를 사용하면 단계 변수 값을 변경하여 API의 다른 VPC 링크로 쉽게 전환할 수 있습니다.

자습서: AWS 통합을 통해 API Gateway REST API 빌드

자습서: Lambda 프록시 통합을 사용하여 Hello World REST API 빌드 및 Lambda 통합을 사용하여 API Gateway REST API 구축 주제에서는 통합된 Lambda 함수를 공개하는 API Gateway API를 생성

하는 방법에 대해 설명합니다. 또한 API Gateway API를 생성하여 Amazon SNS, Amazon S3, Amazon Kinesis, 심지어 AWS Lambda 등의 다른 AWS 서비스를 노출할 수 있습니다. 이는 AWS 통합을 통해 가능합니다. Lambda 통합 또는 Lambda 프록시 통합은 Lambda 함수 호출이 API Gateway API를 통해 공개되는 특수한 경우입니다.

모든 AWS 서비스는 전용 API를 지원하여 해당 기능을 공개합니다. 그러나 애플리케이션 프로토콜 또는 프로그래밍 인터페이스는 서비스마다 다를 수 있습니다. AWS 통합을 사용한 API Gateway API는 클라이언트가 서로 다른 AWS 서비스에 액세스할 수 있도록 일관된 애플리케이션 프로토콜을 제공한다는 이점이 있습니다.

이 연습에서는 Amazon SNS를 공개하는 API를 생성합니다. API와 다른 AWS 서비스를 통합하는 더 많은 예제를 보려면 Amazon API Gateway 자습서 및 워크숍를 참조하십시오.

Lambda 프록시 통합과 달리 다른 AWS 서비스에는 해당 프록시 통합이 없습니다. 따라서 API 메서드는 단일 AWS 작업과 통합됩니다. 유연성을 높이기 위해 프록시 통합과 마찬가지로 Lambda 프록시 통합을 설정할 수 있습니다. 그런 다음 Lambda 함수는 다른 AWS 작업에 대한 요청을 구문 분석하고 처리합니다.

API Gateway는 엔드포인트 시간이 초과되어도 재시도하지 않습니다. API 호출자는 엔드포인트 시간 초과를 처리하기 위한 재시도 로직을 구현해야 합니다.

이 연습은 Lambda 통합을 사용하여 API Gateway REST API 구축의 지침 및 개념을 기반으로 합니다. 아직 그 연습을 완료하지 못했다면 먼저 완료하는 것이 좋습니다.

주제

- 사전 조건
- 1단계: AWS 서비스 프록시 실행 역할 생성
- 2단계: 리소스 생성
- 3단계: GET 메서드 생성
- 4단계: 메서드 설정 지정 및 메서드 테스트
- 5단계: API 배포
- 6단계: API 테스트
- 7단계: 정리

사전 조건

이 연습을 시작하기 전에 다음을 수행합니다.

1. API Gateway를 시작하기 위한 사전 조건의 단계를 수행합니다.
2. `MyDemoAPI`라는 새 API를 생성합니다. 자세한 내용은 자습서: HTTP 비 프록시 통합을 사용하여 REST API 빌드 단원을 참조하세요.
3. API를 `test`이라는 단계에 1회 이상 배포합니다. 자세한 내용은 Lambda 통합을 사용하여 API Gateway REST API 구축의 API 배포를 참조하세요.
4. 의 나머지 단계를 완료합니다Lambda 통합을 사용하여 API Gateway REST API 구축
5. Amazon Simple Notification Service(Amazon SNS)에 하나 이상의 주제를 생성합니다. 배포된 API를 이용해 AWS 계정과 연결된 Amazon SNS에서 주제 목록을 가져옵니다. Amazon SNS에서 주제를 생성하는 방법은 주제 생성을 참조하세요. (5단계에 언급한 주제 ARN을 복사할 필요가 없습니다.)

1단계: AWS 서비스 프록시 실행 역할 생성

API에서 Amazon SNS 작업을 간접적으로 호출하도록 허용하려면 적절한 IAM 정책을 IAM 역할에 연결해야 합니다.

AWS 서비스 프록시 실행 역할을 생성하려면

1. AWS Management Console에 로그인하여 https://console.aws.amazon.com/iam/에서 IAM 콘솔을 엽니다.
2. [Roles]를 선택합니다.
3. 역할 생성(Create role)을 선택합니다.
4. 신뢰할 수 있는 유형의 엔터티 선택에서 AWS 서비스를 선택한 다음 API Gateway를 선택하고 API Gateway가 로그를 CloudWatch Logs로 푸시하도록 허용을 선택합니다.
5. 다음을 선택한 후 다시 다음을 선택합니다.
6. 역할 이름에 **`APIGatewaySNSProxyPolicy`**를 입력한 다음 역할 생성을 선택합니다.
7. 역할 목록에서 방금 생성한 역할을 선택합니다. 스크롤하거나 검색 창을 사용하여 역할을 찾을 수 있습니다.
8. 선택한 역할에 대해 권한 추가 탭을 선택합니다.
9. 드롭다운 목록에서 정책 연결을 선택합니다.
10. 검색 창에 **`AmazonSNSReadOnlyAccess`**를 입력하고 권한 추가를 선택합니다.

> **Note**
>
> 이 자습서에서는 단순화를 위해 관리형 정책을 사용합니다. 가장 좋은 방법은 필요한 최소 권한을 부여하는 자체 IAM 정책을 생성하는 것입니다.

11. 새로 생성된 역할 ARN은 나중에 사용할 수 있도록 기록해 둡니다.

2단계: 리소스 생성

이 단계에서는 AWS 서비스 프록시가 AWS 서비스와 상호 작용할 수 있도록 하는 리소스를 생성합니다.

리소스를 생성하려면

1. https://console.aws.amazon.com/apigateway에서 API Gateway 콘솔에 로그인합니다.
2. API를 선택합니다.
3. 슬래시 하나(/)로 표현된 루트 리소스 /를 선택한 후 리소스 생성을 선택합니다.
4. 프록시 리소스는 꺼진 상태로 둡니다.
5. 리소스 경로를 /로 유지합니다.
6. 리소스 이름에 **mydemoawsproxy**을 입력합니다.
7. 오리진 간 리소스 공유(CORS)를 꺼진 상태로 둡니다.
8. 리소스 생성을 선택합니다.

3단계: GET 메서드 생성

이 단계에서는 AWS 서비스 프록시가 AWS 서비스와 상호 작용할 수 있도록 하는 GET 메서드를 생성합니다.

GET 메서드를 생성하려면

1. /mydemoawsproxy 리소스를 선택한 다음 메서드 생성을 선택합니다.
2. 메서드 유형에서 GET을 선택합니다.
3. 통합 유형에서 AWS 서비스를 선택합니다.
4. AWS 리전에서 Amazon SNS 주제를 생성한 AWS 리전을 선택합니다.
5. AWS 서비스에서 Amazon SNS를 선택합니다.

6. AWS 하위 도메인은 비워 둡니다.
7. HTTP 메서드에서 GET을 선택합니다.
8. 작업 유형에서 작업 이름 사용을 선택합니다.
9. 함수 이름에 **ListTopics**를 입력합니다.
10. 실행 역할에 **APIGatewaySNSProxyPolicy**의 ARN을 입력합니다.
11. 메서드 생성을 선택합니다.

4단계: 메서드 설정 지정 및 메서드 테스트

이제 GET 메서드를 테스트하여 Amazon SNS 주제를 나열하도록 적절히 설정되었는지 확인할 수 있습니다.

GET 메서드를 테스트하는 방법

1. 테스트(Test) 탭을 선택합니다. 탭을 표시하려면 오른쪽 화살표 버튼을 선택해야 할 수도 있습니다.
2. 테스트를 선택합니다.

   결과로 다음과 비슷한 응답이 표시됩니다.

```
{
  "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"
```

```
    }
  }
}
```

5단계: API 배포

이 단계에서는 API를 API Gateway 콘솔 외부에서 호출할 수 있도록 배포합니다.

API를 배포하려면

1. Deploy API(API 배포)를 선택합니다.
2. 스테이지에서 새 스테이지를 선택합니다.
3. 단계 이름에 **test**를 입력합니다.
4. (선택 사항) 설명에 설명을 입력합니다.
5. [Deploy]를 선택합니다.

6단계: API 테스트

이 단계에서는 API Gateway 콘솔 외부로 이동한 후 AWS 서비스 프록시를 사용하여 Amazon SNS 서비스와 상호 작용합니다.

1. 기본 탐색 창에서 스테이지를 선택합니다.
2. 스테이지 세부 정보에서 복사 아이콘을 선택하여 API의 호출 URL을 복사합니다.

   형식은 다음과 같아야 합니다.

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

3. URL을 새 브라우저 탭의 주소 표시줄에 입력합니다.
4. /mydemoawsproxy를 추가하여 URL이 다음과 같이 되도록 합니다.

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

   해당 URL로 이동합니다. 다음과 비슷한 정보가 표시되어야 합니다.

   ```
   {"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"}}
```

7단계: 정리

AWS 서비스 프록시가 작업해야 할 IAM 리소스를 삭제할 수 있습니다.

> **Warning**
>
> AWS 서비스 프록시가 사용하는 IAM 리소스를 삭제하면 AWS 서비스 프록시 및 그것을 사용하는 모든 API는 더 이상 작동하지 않습니다. IAM 리소스 삭제는 실행 취소할 수 없습니다. IAM 리소스를 다시 사용하고 싶다면 재생성해야 합니다.

연결된 IAM 리소스를 삭제하려면

1. https://console.aws.amazon.com/iam/에서 IAM 콘솔을 엽니다.
2. 세부 정보 영역에서 역할을 선택합니다.
3. APIGatewayAWSProxyExecRole을 선택한 다음 역할 작업, 역할 삭제를 선택합니다. 확인 메시지가 나타나면 [Yes, Delete]를 선택합니다.
4. 세부 정보 영역에서 정책을 선택합니다.
5. APIGatewayAWSProxyExecPolicy를 선택한 다음 정책 작업, 삭제를 선택합니다. 확인 메시지가 나타나면 삭제를 선택합니다.

이 연습을 마칩니다. API를 AWS 서비스 프록시로 생성하는 방법에 대한 자세한 설명은 자습서: API Gateway에서 REST API를 Amazon S3 프록시로 생성, 자습서: AWS 서비스 통합 2개와 Lambda 비프록시 통합 1개가 포함된 Calc REST API 생성 또는 자습서: API Gateway에서 REST API를 Amazon Kinesis 프록시로 생성 단원을 참조하세요.

자습서: AWS 서비스 통합 2개와 Lambda 비프록시 통합 1개가 포함된 **Calc** REST API 생성

비 프록시 통합 시작하기 자습서에서는 Lambda Function 통합만 사용합니다. Lambda Function 통합은 AWS Service 통합 유형의 특별한 사례로서, Lambda 함수를 호출하는 데 필요한 리소스 기반 사용 권한을 자동으로 추가하는 등 대부분의 통합 설정을 수행합니다. 여기서 세 가지 통합 중 두 가지

는 AWS Service 통합을 사용합니다. 이 통합 유형은 더 많은 제어 권한을 제공하지만, 적절한 권한이 포함된 IAM 역할 생성 및 지정과 같은 작업을 수동으로 수행해야 합니다.

이 자습서에서는 JSON 형식의 입력 및 출력을 수락 및 반환하여 기본 산술 연산을 구현하는 Calc Lambda 함수를 생성합니다. 그런 다음 REST API를 생성하여 다음과 같이 Lambda 함수와 통합합니다.

1. GET 리소스에 /calc 메서드를 공개하여 Lambda 함수를 호출하고, 입력을 쿼리 문자열 파라미터로 제공합니다(AWS Service 통합).
2. POST 리소스에 /calc 메서드를 공개하여 Lambda 함수를 호출하고, 입력을 메서드 요청 페이로드에 제공합니다(AWS Service 통합).
3. 중첩된 GET 리소스에 /calc/{operand1}/{operand2}/{operator}을 공개하여 Lambda 함수를 호출하고, 입력을 경로 파라미터로 제공합니다(Lambda Function 통합).

이 자습서 외에도, Calc API에 대한 OpenAPI 정의 파일을 검토할 수 있습니다. 이 파일은 the section called “OpenAPI”의 지침에 따라 API Gateway로 가져올 수 있습니다.

주제

- 맡을 수 있는 IAM 역할 생성
- Calc Lambda 함수 생성
- Calc Lambda 함수 테스트
- Calc API를 만듭니다.
- 통합 1: Lambda 함수를 호출하기 위해 쿼리 파라미터를 사용하여 GET 메서드 생성
- 통합 2: Lambda 함수를 호출하기 위해 JSON 페이로드를 사용하여 POST 메서드 생성
- 통합 3: Lambda 함수를 호출하기 위해 경로 파라미터를 사용하여 GET 메서드 생성
- Lambda 함수와 통합된 샘플 API의 OpenAPI 정의

맡을 수 있는 IAM 역할 생성

API가 Calc Lambda 함수를 호출하게 하려면 API Gateway가 맡을 수 있는 IAM 역할이 있어야 합니다. 이 역할은 다음과 같은 신뢰할 수 있는 관계를 가진 IAM 역할입니다.

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

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

생성한 역할에는 InvokeFunctionLambda 권한이 있어야 합니다. 이 권한이 없으면 API 호출자가 `500 Internal Server Error` 응답을 받습니다. 역할에 이 권한을 부여하려면 다음 IAM 정책을 역할에 연결해야 합니다.

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

다음은 이를 수행하는 방법입니다.

API Gateway가 맡을 수 있는 IAM 역할 생성

1. IAM 콘솔에 로그인합니다.
2. [Roles]를 선택합니다.
3. 역할 생성을 선택합니다.
4. [신뢰할 수 있는 유형의 엔터티 선택(Select type of trusted entity)] 아래에서 AWS 서비스를 선택합니다.
5. Choose the service that will use this role(이 역할을 사용할 서비스 선택) 아래에서 Lambda를 선택합니다.
6. Next: Permissions(다음: 권한)를 선택합니다.
7. 정책 생성을 선택합니다.

새로운 정책 생성 콘솔 창이 열립니다. 이 창에서 다음을 수행합니다.

a. JSON 탭에서 기존 정책을 다음으로 바꿉니다.

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

b. 정책 검토(Review policy)를 선택합니다.

c. 정책 검토에서 다음을 수행합니다.

i. 이름에서 **lambda_execute** 같은 사용자 이름을 입력합니다.

ii. 정책 생성을 선택합니다.

8. 원래의 역할 만들기 콘솔 창에서 다음을 수행하십시오.

a. Attach permissions policies(권한 정책 연결)의 드롭다운 목록에서 **lambda_execute** 정책을 선택합니다.

목록에 해당 정책이 표시되지 않으면 목록 상단에 있는 새로 고침 버튼을 선택합니다. (브라우저 페이지를 새로 고치지 마십시오.)

b. 다음: 태그를 선택합니다.

c. 다음: 검토를 선택합니다.

d. 역할 이름에 **lambda_invoke_function_assume_apigw_role** 등의 이름을 입력합니다.

e. 역할 생성을 선택합니다.

9. 역할 목록에서 **lambda_invoke_function_assume_apigw_role**을 선택합니다.

10. 신뢰 관계 탭을 선택합니다.

11. 신뢰 관계 편집(Edit trust relationship)을 선택합니다.

12. 기존 정책을 다음으로 바꿉니다.

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

13. 신뢰 정책 업데이트(Update Trust Policy)를 선택합니다.
14. 방금 생성한 역할의 역할 ARN을 기록해 둡니다. 잠시 후 필요한 정보입니다.

Calc Lambda 함수 생성

다음으로 Lambda 콘솔을 사용하여 Lambda 함수를 생성합니다.

1. Lambda 콘솔에서 함수 생성(Create function)을 선택합니다.
2. 새로 작성을 선택합니다.
3. 이름에 **Calc**를 입력합니다.
4. 런타임에서 지원되는 최신 Node.js 또는 Python 런타임을 선택합니다.
5. 함수 생성을 선택합니다.
6. 선호하는 런타임에 다음의 Lambda 함수를 복사한 다음, Lambda 콘솔에서 코드 편집기에 붙여 넣습니다.

   Node.js

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

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

  const res = {};
  res.a = Number(event.a);
  res.b = Number(event.b);
  res.op = event.op;
  if (isNaN(event.a) || isNaN(event.b)) {
    return "400 Invalid Operand";
  }
  switch (event.op) {
    case "+":
    case "add":
      res.c = res.a + res.b;
      break;
    case "-":
    case "sub":
      res.c = res.a - res.b;
      break;
    case "*":
    case "mul":
      res.c = res.a * res.b;
      break;
    case "/":
    case "div":
      if (res.b == 0) {
        return "400 Divide by Zero";
      } else {
        res.c = res.a / res.b;
      }
      break;
    default:
      return "400 Invalid Operator";
  }

  return res;
};
```

Python

```
import json


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

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

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

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

    return res
```

7. 실행 역할에서 기존 역할 선택(Choose an existing role)을 선택합니다.
8. 앞에서 만든 **lambda_invoke_function_assume_apigw_role** 역할의 ARN을 입력합니다.
9. [Deploy]를 선택합니다.

이 함수에는 a 입력 파라미터의 두 피연산자(b 및 op) 및 연산자(event)가 필요합니다. 입력은 다음 형식의 JSON 객체입니다.

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

이 함수에서 계산된 결과(c) 및 입력을 반환합니다. 입력이 잘못된 경우 함수에서 널 값 또는 "Invalid op" 문자열을 결과로 반환합니다. 출력은 다음 JSON 형식입니다.

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

다음 단계에서 함수를 API와 통합하기 전에 Lambda 콘솔에서 함수를 테스트해야 합니다.

Calc Lambda 함수 테스트

다음은 Lambda 콘솔에서 Calc 함수를 테스트하는 방법입니다.

1. 테스트 탭을 선택합니다.
2. 테스트 이벤트의 이름에 **calc2plus5**를 입력합니다.
3. 테스트 이벤트 정의를 다음과 같이 바꿉니다.

   ```
   {
     "a": "2",
     "b": "5",
     "op": "+"
   }
   ```
4. 저장을 선택합니다.
5. 테스트를 선택합니다.

6. 실행 결과: 성공을 확장합니다. 다음과 같은 모양이어야 합니다.

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

Calc API를 만듭니다.

다음 절차는 방금 만든 Calc Lambda 함수에 대한 API를 만드는 방법을 보여줍니다. 그 다음 단원에서는 이 API에 리소스와 메서드를 추가할 것입니다.

API를 생성하는 방법

1. https://console.aws.amazon.com/apigateway에서 API Gateway 콘솔에 로그인합니다.
2. API Gateway를 처음 사용하는 경우, 서비스의 기능을 소개하는 페이지가 나타납니다. REST API에서 빌드를 선택합니다. 예제 API 생성 팝업이 나타나면 확인을 선택합니다.

   API Gateway를 처음 사용하는 것이 아닌 경우 API 생성을 선택합니다. REST API에서 빌드를 선택합니다.
3. API 이름에서 **LambdaCalc**을(를) 입력합니다.
4. (선택 사항) 설명에 설명을 입력합니다.
5. API 엔드포인트 유형 설정을 지역으로 유지합니다.
6. API 생성(Create API)을 선택합니다.

통합 1: Lambda 함수를 호출하기 위해 쿼리 파라미터를 사용하여 **GET** 메서드 생성

쿼리 문자열 파라미터를 Lambda 함수에 전달하는 GET 메서드를 생성하여, API가 브라우저에서 호출될 수 있도록 합니다. 이 방법은 오픈 액세스를 허용하는 API에 특히 유용합니다.

API를 생성한 후에는 리소스를 생성합니다. 일반적으로 API 리소스는 애플리케이션 로직에 따른 리소스 트리로 정리되어 있습니다. 이 단계에서는 /calc 리소스를 생성합니다.

/calc 리소스를 생성하려면

1. 리소스 생성을 선택합니다.

2. 프록시 리소스는 꺼진 상태로 둡니다.
3. 리소스 경로를 /로 유지합니다.
4. 리소스 이름에 **calc**을 입력합니다.
5. 오리진 간 리소스 공유(CORS)를 꺼진 상태로 둡니다.
6. 리소스 생성을 선택합니다.

쿼리 문자열 파라미터를 Lambda 함수에 전달하는 GET 메서드를 생성하여, API가 브라우저에서 호출될 수 있도록 합니다. 이 방법은 오픈 액세스를 허용하는 API에 특히 유용합니다.

이 메서드에서 Lambda 함수의 경우 임의의 Lambda 함수를 간접적으로 호출할 때 POST 요청을 사용해야 합니다. 이 예제에서는 프런트엔드 메서드 요청에서의 HTTP 메서드가 백엔드에서의 통합 요청과 다를 수 있다는 것을 보여 줍니다.

GET 메서드를 생성하는 방법

1. /calc 리소스를 선택한 다음 메서드 생성을 선택합니다.
2. 메서드 유형에서 GET을 선택합니다.
3. 통합 유형에서 AWS 서비스를 선택합니다.
4. 에서 Lambda 함수를 생성한 AWS 리전 위치를 선택합니다. AWS 리전
5. AWS 서비스에서 Lambda를 선택합니다.
6. AWS 하위 도메인은 비워 둡니다.
7. HTTP 메서드에서 POST를 선택합니다.
8. 작업 유형에서 경로 재정의 사용을 선택합니다. 이 옵션을 사용하면 호출 작업의 ARN을 지정하여 `Calc` 함수를 실행할 수 있습니다.
9. 경로 재정의에서 **2015-03-31/functions/arn:aws:lambda:*us-east-2*:*account-id*:function:Calc/invocations**를 입력합니다. ***account-id***에 ID를 입력합니다. AWS 계정 에 ***us-east-2*** Lambda 함수를 생성한 AWS 리전 위치를 입력합니다.
10. 실행 역할에 **lambda_invoke_function_assume_apigw_role**의 ARN을 입력합니다.
11. 보안 인증 캐시 및 기본 제한 시간 설정을 변경하지 마세요.
12. 메서드 생성을 선택합니다.

이제 /calc 리소스에서 GET 메서드에 대한 쿼리 파라미터를 설정하면 백엔드 Lambda 함수를 대신하여 입력을 수신할 수 있습니다.

쿼리 문자열 파라미터를 설정하려면

1. 메서드 요청 탭의 메서드 요청 설정에서 편집을 선택합니다.
2. 요청 검사기에서 쿼리 문자열 파라미터 및 헤더 검사를 선택합니다. 이렇게 설정하면 클라이언트가 필수 파라미터를 지정하지 않은 경우 오류 메시지가 반환됩니다.
3. URL 쿼리 문자열 파라미터를 선택하고 다음을 수행합니다.

   a. 쿼리 문자열 추가(Add query string)를 선택합니다.

   b. 이름에 **operand1**를 입력합니다.

   c. 필수를 끕니다.

   d. 캐싱을 꺼진 상태로 둡니다.

   같은 단계를 반복하여 **operand2**라는 쿼리 문자열과 **operator**라는 쿼리 문자열을 생성합니다.
4. 저장을 선택합니다.

이제 `Calc` 함수에 필요한 대로 클라이언트가 제공한 쿼리 문자열을 통합 요청 페이로드로 변환하는 매핑 템플릿을 생성합니다. 이 템플릿은 메서드 요청에서 선언된 세 쿼리 문자열 파라미터를 백엔드 Lambda 함수에 대한 입력으로서 JSON 객체의 지정 속성 값에 매핑합니다. 변환된 JSON 객체는 통합 요청 페이로드로 포함됩니다.

입력 파라미터를 통합 요청에 매핑하려면

1. 통합 요청 탭의 통합 요청 설정에서 편집을 선택합니다.
2. 요청 본문 패스스루에서 정의된 템플릿이 없는 경우(권장)를 선택합니다.
3. 매핑 템플릿을 선택합니다.
4. 매핑 템플릿 추가(Add mapping template)를 선택합니다.
5. 콘텐츠 유형에 **application/json**을 입력합니다.
6. 템플릿 본문에 다음 코드를 입력합니다.

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

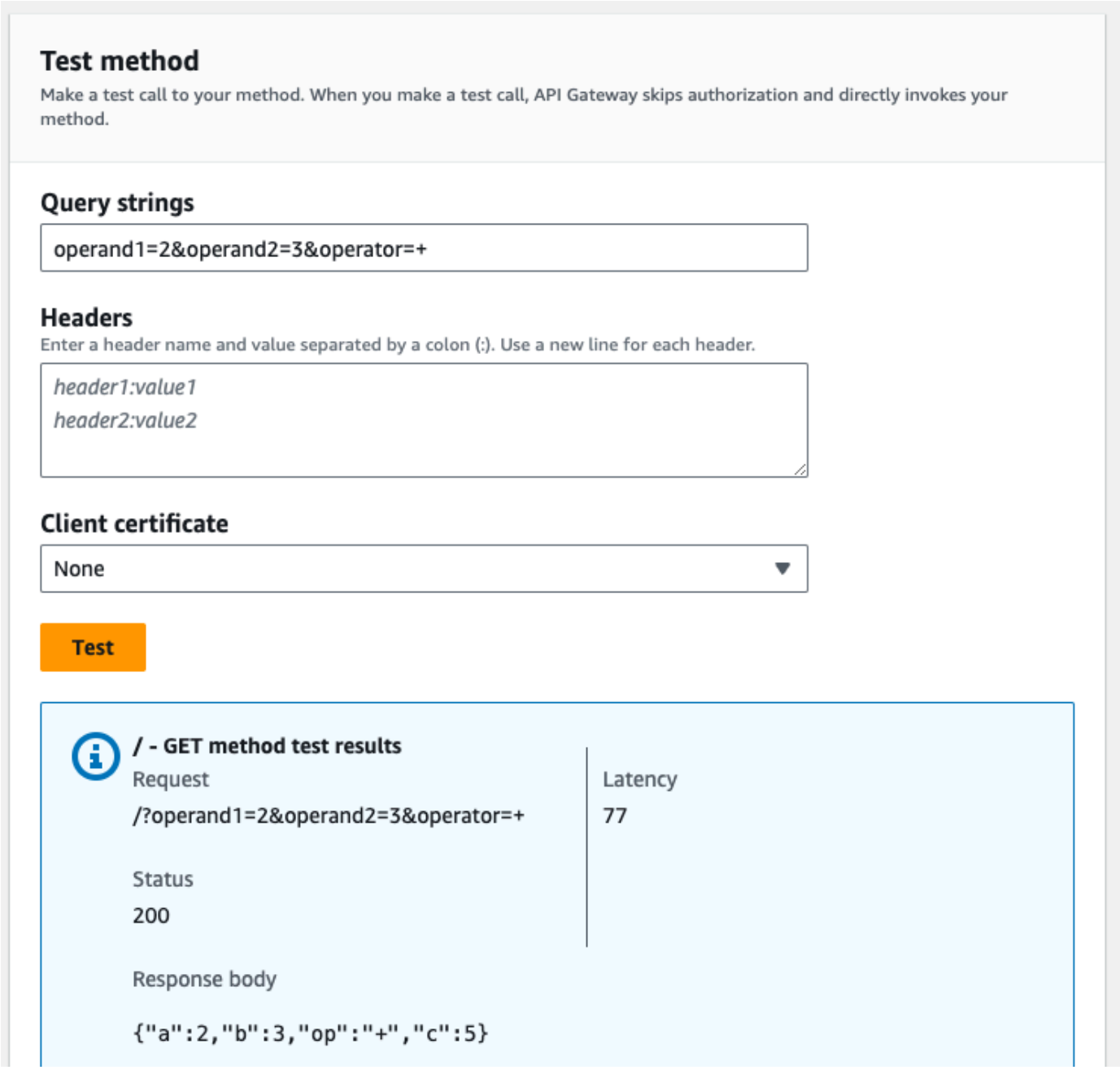
7. 저장을 선택합니다.

이제 GET 메서드를 테스트하여 Lambda 함수를 간접적으로 호출하도록 적절히 설정되었는지 확인할 수 있습니다.

GET 메서드를 테스트하는 방법

1. 테스트 탭을 선택합니다. 탭을 표시하려면 오른쪽 화살표 버튼을 선택해야 할 수도 있습니다.
2. 쿼리 문자열에 **operand1=2&operand2=3&operator=+**를 입력합니다.
3. 테스트를 선택합니다.

   다음과 같은 결과가 나타납니다.



통합 2: Lambda 함수를 호출하기 위해 JSON 페이로드를 사용하여 **POST** 메서드 생성

Lambda 함수를 호출하는 JSON 페이로드로 POST 메서드를 생성하면 클라이언트가 요청 본문의 백엔드 함수에 필요한 입력을 제공해야 합니다. 클라이언트가 올바른 입력 데이터를 업로드하도록 하기 위해 페이로드에서 요청 확인을 활성화합니다.

JSON 페이로드로 **POST** 메서드를 생성하려면

1. /calc 리소스를 선택한 다음 메서드 생성을 선택합니다.
2. 메서드 유형에서 POST를 선택합니다.
3. 통합 유형에서 AWS 서비스를 선택합니다.
4. 에서 Lambda 함수를 생성한 AWS 리전 위치를 선택합니다. AWS 리전
5. AWS 서비스에서 Lambda를 선택합니다.
6. AWS 하위 도메인은 비워 둡니다.
7. HTTP 메서드에서 POST를 선택합니다.
8. 작업 유형에서 경로 재정의 사용을 선택합니다. 이 옵션을 사용하면 호출 작업의 ARN을 지정하여 `Calc` 함수를 실행할 수 있습니다.
9. 경로 재정의에서 **2015-03-31/functions/arn:aws:lambda:*us-east-2*:*account-id*:function:Calc/invocations**를 입력합니다. ***account-id***에 ID를 입력합니다. AWS 계정 에 ***us-east-2*** Lambda 함수를 생성한 AWS 리전 위치를 입력합니다.
10. 실행 역할에 **lambda_invoke_function_assume_apigw_role**의 ARN을 입력합니다.
11. 보안 인증 캐시 및 기본 제한 시간 설정을 변경하지 마세요.
12. 메서드 생성을 선택합니다.

이제 입력 데이터 구조를 설명하고 수신되는 요청의 본문을 검사하기 위한 입력 모델을 생성합니다.

입력 모델을 생성하려면

1. 기본 탐색 창에서 모델을 선택합니다.
2. Create model(모델 생성)을 선택합니다.
3. 이름에 **input**를 입력합니다.
4. 콘텐츠 유형에 **application/json**을 입력합니다.

   일치하는 콘텐츠 유형이 없는 경우 요청 확인이 수행되지 않습니다. 콘텐츠 유형에 관계없이 동일한 모델을 사용하려면 **$default**를 입력합니다.
5. 모델 스키마에서 다음 모델을 입력합니다.

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

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

6. Create model(모델 생성)을 선택합니다.

이제 출력 모델을 생성합니다. 이 모델은 백엔드로부터 계산된 출력의 데이터 구조를 설명합니다. 통합 응답 데이터를 다른 모델에 매핑하는 데 사용할 수 있습니다. 이 자습서에서는 패스스루 동작에 의존하며 이 모델을 사용하지 않습니다.

출력 모델을 생성하려면

1. Create model(모델 생성)을 선택합니다.
2. 이름에 **output**를 입력합니다.
3. 콘텐츠 유형에 **application/json**을 입력합니다.

   일치하는 콘텐츠 유형이 없는 경우 요청 확인이 수행되지 않습니다. 콘텐츠 유형에 관계없이 동일한 모델을 사용하려면 **$default**를 입력합니다.
4. 모델 스키마에서 다음 모델을 입력합니다.

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

5. Create model(모델 생성)을 선택합니다.

이제 결과 모델을 생성합니다. 이 모델은 반환된 응답 데이터의 데이터 구조를 설명합니다. 이는 API에 정의된 입력 스키마와 출력 스키마를 모두 참조합니다.

결과 모델을 생성하려면

1. Create model(모델 생성)을 선택합니다.
2. 이름에 **result**를 입력합니다.

3. 콘텐츠 유형에 **application/json**을 입력합니다.

   일치하는 콘텐츠 유형이 없는 경우 요청 확인이 수행되지 않습니다. 콘텐츠 유형에 관계없이 동일한 모델을 사용하려면 **$default**를 입력합니다.

4. 모델 스키마에서 *restapi-id*를 사용하여 다음 모델을 입력합니다. *resapi-id*는 다음 흐름에서 콘솔 상단에 괄호 안에 표시됩니다. `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. Create model(모델 생성)을 선택합니다.

이제 수신되는 요청의 본문에서 요청 검사를 활성화하도록 POST 메서드의 메서드 요청을 구성합니다.

POST 메서드에서 요청 검증을 활성화하려면

1. 기본 탐색 창에서 리소스를 선택하고 리소스 트리에서 POST 메서드를 선택합니다.
2. 메서드 요청 탭의 메서드 요청 설정에서 편집을 선택합니다.
3. 요청 검사기에서 본문 검사를 선택합니다.
4. 요청 본문을 선택한 다음 모델 추가를 선택합니다.
5. 콘텐츠 유형에 **application/json**을 입력합니다.

   일치하는 콘텐츠 유형이 없는 경우 요청 확인이 수행되지 않습니다. 콘텐츠 유형에 관계없이 동일한 모델을 사용하려면 **$default**를 입력합니다.

6. 모델에서 입력을 선택합니다.
7. 저장을 선택합니다.

이제 POST 메서드를 테스트하여 Lambda 함수를 간접적으로 호출하도록 적절히 설정되었는지 확인할 수 있습니다.

POST 메서드를 테스트하는 방법

1. 테스트 탭을 선택합니다. 탭을 표시하려면 오른쪽 화살표 버튼을 선택해야 할 수도 있습니다.
2. 요청 본문에 다음 JSON 페이로드를 입력합니다.

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

3. 테스트를 선택합니다.

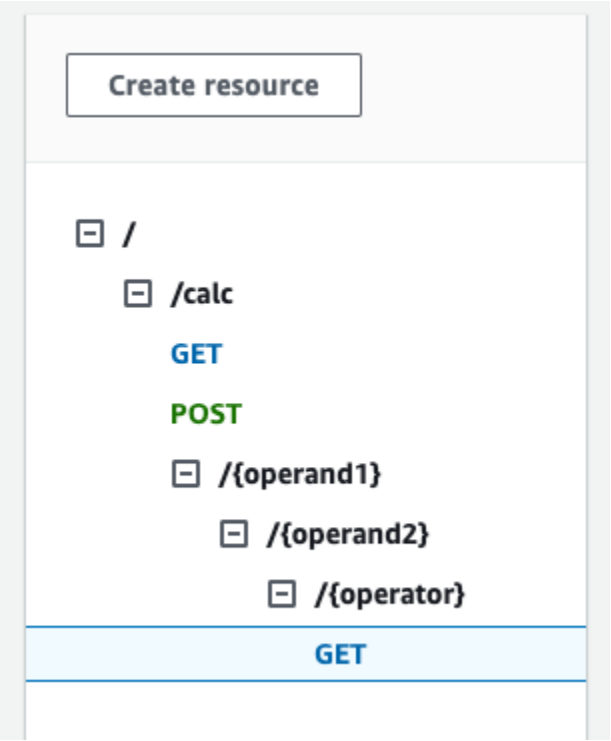
   다음 결과가 표시됩니다.

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

통합 3: Lambda 함수를 호출하기 위해 경로 파라미터를 사용하여 **GET** 메서드 생성

이제 백엔드 Lambda 함수를 호출하는 경로 파라미터 시퀀스가 지정한 리소스에 GET 메서드를 만들어 보겠습니다. 경로 파라미터 값은 Lambda 함수에 대한 입력 데이터를 지정합니다. 받는 경로 파라미터 값을 필수 통합 요청 페이로드에 매핑하는 매핑 템플릿을 사용합니다.

이렇게 만들어진 API 리소스 구조는 다음과 같습니다.



/{operand1}/{operand2}/{operator} 리소스를 생성하려면

1. 리소스 생성을 선택합니다.
2. 리소스 경로에서 /calc를 선택합니다.
3. 리소스 이름에 **{operand1}**을 입력합니다.
4. 오리진 간 리소스 공유(CORS)를 꺼진 상태로 둡니다.
5. 리소스 생성을 선택합니다.
6. 리소스 경로에서 /calc/{operand1}/를 선택합니다.
7. 리소스 이름에 **{operand2}**을 입력합니다.
8. 오리진 간 리소스 공유(CORS)를 꺼진 상태로 둡니다.
9. 리소스 생성을 선택합니다.
10. 리소스 경로에서 /calc/{operand1}/{operand2}/를 선택합니다.
11. 리소스 이름에 **{operator}**을 입력합니다.
12. 오리진 간 리소스 공유(CORS)를 꺼진 상태로 둡니다.
13. 리소스 생성을 선택합니다.

이번에는 API Gateway 콘솔에서 기본으로 제공하는 Lambda 통합 기능을 사용하여 메서드 통합을 설정합니다.

메서드 통합을 설정하려면

1. /{operand1}/{operand2}/{operator} 리소스를 선택한 다음 메서드 생성을 선택합니다.
2. 메서드 유형에서 GET을 선택합니다.
3. 통합 유형에서 Lambda를 선택합니다.
4. Lambda 프록시 통합을 비활성화된 상태로 유지합니다.
5. Lambda 함수의 경우, Lambda 함수를 생성한 위치를 선택하고 AWS 리전 를 입력합니다. **`Calc`**
6. 기본 제한 시간을 켜진 상태로 둡니다.
7. 메서드 생성을 선택합니다.

이제 /calc/{operand1}/{operand2}/{operator} 리소스가 생성될 때 선언된 URL 경로 파라미터를 JSON 객체의 지정 속성 값에 매핑하는 매핑 템플릿을 생성합니다. URL 경로가 URL로 인코딩되어야 하므로 나눗셈 연산자는 `%2F`가 아닌 `/`로 지정해야 합니다. 이 템플릿은 `%2F`를 `''/'`로 변환한 다음 Lambda 함수에 전달합니다.

매핑 템플릿을 생성하려면

1. 통합 요청 탭의 통합 요청 설정에서 편집을 선택합니다.
2. 요청 본문 패스스루에서 정의된 템플릿이 없는 경우(권장)를 선택합니다.
3. 매핑 템플릿을 선택합니다.
4. 콘텐츠 유형에 **`application/json`**을 입력합니다.
5. 템플릿 본문에 다음 코드를 입력합니다.

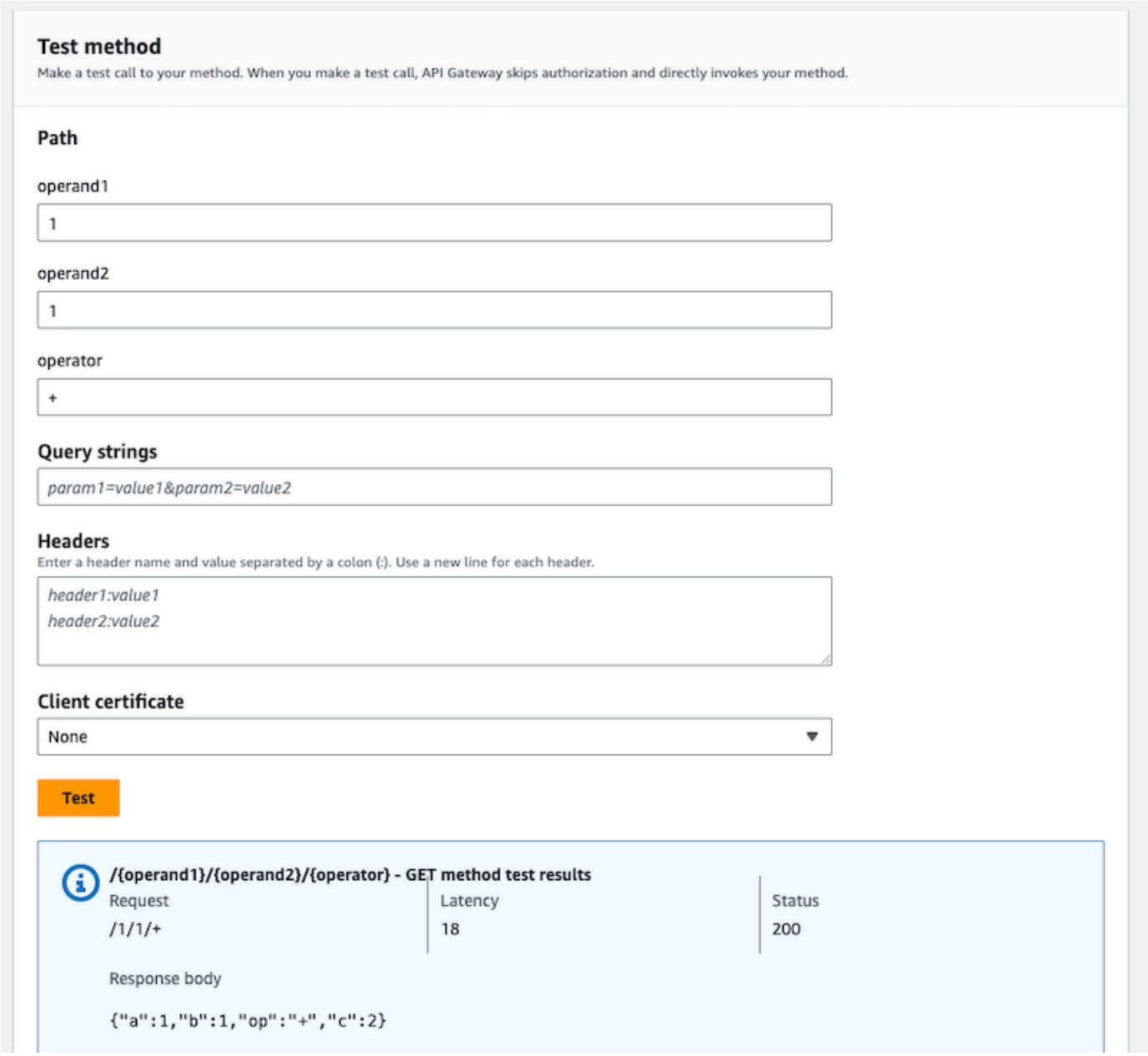
```
{
   "a": "$input.params('operand1')",
   "b": "$input.params('operand2')",
   "op":
 #if($input.params('operator')=='%2F')"/"#{else}"$input.params('operator')"#end
}
```

6. 저장을 선택합니다.

이제 Lambda 함수를 간접적으로 호출하고 매핑 없이 통합 응답을 통해 원본 출력을 전달하도록 적절히 설정되었는지 확인하기 위해 GET 메서드를 테스트할 수 있습니다.

GET 메서드를 테스트하는 방법

1. 테스트 탭을 선택합니다. 탭을 표시하려면 오른쪽 화살표 버튼을 선택해야 할 수도 있습니다.
2. 경로에서 다음을 수행합니다.
   a. operand1에 **1**을 입력합니다.
   b. operand2에 **1**을 입력합니다.
   c. operator에 **+**를 입력합니다.
3. 테스트를 선택합니다.
4. 결과는 다음과 같아야 합니다.



그런 다음 `result` 스키마 이후 메서드 응답 페이로드의 데이터 구조를 모델링합니다.

기본적으로 메서드 응답 본문에는 빈 모델이 지정됩니다. 이 때문에 통합 응답 본문이 매핑되지 않고 통과합니다. 그러나 Java 또는 Objective-C와 같은 강력한 형식의 언어 중 하나에 대한 SDK를 생성하는 경우 결과적으로 SDK 사용자가 빈 객체를 수신하게 됩니다. REST 클라이언트 및 SDK 클라이언트 모두 원하는 결과를 수신하도록 하려면 사전 정의된 스키마를 사용하여 응답 데이터를 모델링해야 합니다. 여기에서는 메서드 응답 본문에 대한 모델을 정의하고 통합 응답 본문을 메서드 응답 본문으로 변환하는 매핑 템플릿을 구성하겠습니다.

메서드 응답을 생성하려면

1. 메서드 응답 탭의 응답 200에서 편집을 선택합니다.
2. 요청 본문에서 모델 추가를 선택합니다.
3. 콘텐츠 유형에 **application/json**을 입력합니다.
4. 모델에서 결과를 선택합니다.
5. 저장을 선택합니다.

메서드 응답 본문에 대한 모델을 설정하여 응답 데이터가 주어진 SDK의 `result` 객체에 캐스팅되도록 합니다. 통합 응답 데이터가 적절히 매핑되게 하려면 매핑 템플릿이 필요합니다.

매핑 템플릿을 생성하려면

1. 통합 응답 탭의 기본값 - 응답에서 편집을 선택합니다.
2. 매핑 템플릿을 선택합니다.
3. 콘텐츠 유형에 **application/json**을 입력합니다.
4. 템플릿 생성에서 결과를 선택합니다.
5. 생성된 매핑 템플릿을 다음과 일치하도록 수정합니다.

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

6. 저장을 선택합니다.

매핑 템플릿을 테스트하려면

1. 테스트 탭을 선택합니다. 탭을 표시하려면 오른쪽 화살표 버튼을 선택해야 할 수도 있습니다.
2. 경로에서 다음을 수행합니다.

a. operand1에 **1**을 입력합니다.

b. operand2에 **2**을 입력합니다.

c. operator에 **+**를 입력합니다.

3. 테스트를 선택합니다.

4. 결과는 다음과 같습니다.

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

이 시점에서는, API Gateway 콘솔에서 테스트 기능을 사용해서만 API를 직접적으로 호출할 수 있습니다. 클라이언트가 호출할 수 있게 하려면 API를 배포해야 합니다. 리소스 또는 메서드를 추가, 수정, 삭제하거나, 데이터 매핑을 업데이트하거나, 단계 설정을 업데이트할 때마다 항상 API를 다시 배포하십시오. 다음과 같이 다시 배포하지 않으면 API의 클라이언트가 새 기능 또는 업데이트를 사용할 수 없습니다.

API를 배포하려면

1. Deploy API(API 배포)를 선택합니다.

2. 스테이지에서 새 스테이지를 선택합니다.

3. 단계 이름에 **Prod**를 입력합니다.

4. (선택 사항) 설명에 설명을 입력합니다.

5. 배포를 선택합니다.

6. (선택 사항) 스테이지 세부 정보의 URL 호출에서 복사 아이콘을 선택하여 API의 호출 URL을 복사합니다. 이 URL을 Postman 및 cURL과 같은 도구에서 사용하여 API를 테스트할 수 있습니다.

> **Note**
>
> 리소스 또는 메서드를 추가, 수정, 삭제하거나, 데이터 매핑을 업데이트하거나, 스테이지 설정을 업데이트할 때마다 항상 API를 다시 배포합니다. 다시 배포하지 않으면 API의 클라이언트가 새 기능 또는 업데이트를 사용할 수 없습니다.

Lambda 함수와 통합된 샘플 API의 OpenAPI 정의

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
    }
  }
}
```

자습서: API Gateway에서 REST API를 Amazon S3 프록시로 생성

API Gateway에서 REST API를 사용하여 Amazon S3를 프록시하는 과정을 보여주는 이 예제 단원에서는 REST API를 생성하여 다음 Amazon S3 작업을 노출하도록 구성하는 방법을 설명합니다.

- API의 루트 리소스에서 GET을 노출하여 호출자의 모든 Amazon S3 버킷을 나열합니다.
- 폴더 리소스에서 GET을 노출하여 Amazon S3 버킷의 모든 객체 목록을 봅니다.
- 폴더/항목 리소스에서 GET을 노출하여 Amazon S3 버킷에서 객체를 보거나 다운로드합니다.

Amazon S3 프록시로 샘플 API의 OpenAPI 정의에 표시된 대로 샘플 API를 Amazon S3 프록시로 가져올 수 있습니다. 이 샘플에는 노출된 메서드가 더 많이 포함되어 있습니다. OpenAPI 정의를 사용하여 API를 가져오는 방법에 대한 지침은 OpenAPI를 사용하여 REST API 구성 단원을 참조하십시오.

> **Note**
>
> API Gateway API를 Amazon S3와 통합하려면 API Gateway와 Amazon S3 서비스를 모두 사용할 수 있는 리전을 선택해야 합니다. 리전 사용 가능성은 Amazon API Gateway 엔드포인트 및 할당량 단원을 참조하십시오.

주제

- API에서 Amazon S3 작업을 호출하도록 허용하는 IAM 권한 설정
- API 리소스를 생성하여 Amazon S3 리소스 표시
- API 메서드를 노출하여 호출자의 Amazon S3 버킷 나열
- API 메서드를 노출하여 Amazon S3 버킷 액세스
- API 메서드를 노출하여 버킷에서 Amazon S3 객체 액세스

- Amazon S3 프록시로 샘플 API의 OpenAPI 정의
- REST API 클라이언트를 사용하여 API 호출

API에서 Amazon S3 작업을 호출하도록 허용하는 IAM 권한 설정

API에서 Amazon S3 작업을 간접적으로 호출하도록 허용하려면 적절한 IAM 정책을 IAM 역할에 연결해야 합니다.

AWS 서비스 프록시 실행 역할을 생성하려면

1. AWS Management Console에 로그인하고 https://console.aws.amazon.com/iam/에서 IAM 콘솔을 엽니다.
2. [Roles]를 선택합니다.
3. 역할 생성을 선택합니다.
4. 신뢰할 수 있는 유형의 엔터티 선택에서 AWS 서비스를 선택한 다음 API Gateway를 선택하고 API Gateway가 로그를 CloudWatch Logs로 푸시하도록 허용을 선택합니다.
5. 다음을 선택한 후 다시 다음을 선택합니다.
6. 역할 이름에 **APIGatewayS3ProxyPolicy**를 입력한 다음 역할 생성을 선택합니다.
7. 역할 목록에서 방금 생성한 역할을 선택합니다. 스크롤하거나 검색 창을 사용하여 역할을 찾을 수 있습니다.
8. 선택한 역할에 대해 권한 추가 탭을 선택합니다.
9. 드롭다운 목록에서 정책 연결을 선택합니다.
10. 검색 창에 **AmazonS3FullAccess**를 입력하고 권한 추가를 선택합니다.

    > **Note**
    >
    > 이 자습서에서는 단순화를 위해 관리형 정책을 사용합니다. 가장 좋은 방법은 필요한 최소 권한을 부여하는 자체 IAM 정책을 생성하는 것입니다.

11. 새로 생성된 역할 ARN은 나중에 사용할 수 있도록 기록해 둡니다.

API 리소스를 생성하여 Amazon S3 리소스 표시

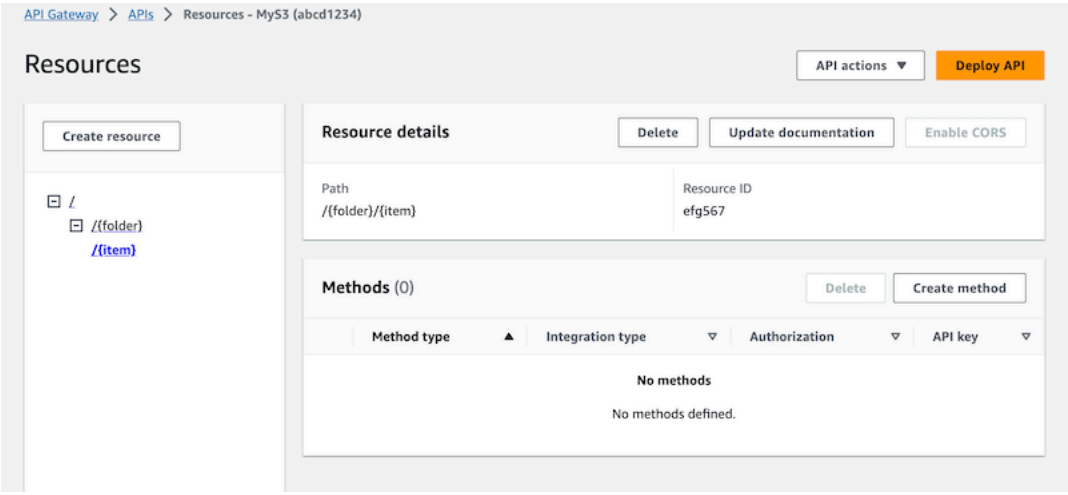
API의 루트(/) 리소스를 인증된 호출자의 Amazon S3 버킷 컨테이너로 사용합니다. 또한 `Folder` 및 `Item` 리소스를 생성하여 각각 특정 Amazon S3 버킷과 특정 Amazon S3 객체를 표시합니다. 호출자는 요청 URL의 일부로 폴더 이름과 객체 키를 경로 파라미터 형식으로 지정합니다.

> **Note**
>
> 객체 키가 /나 다른 특수 문자를 포함하는 객체나에 액세스할 때 문자는 URL로 인코딩되어야 합니다. 예를 들어 `test/test.txt`는 `test%2Ftest.txt`으로 인코딩되어야 합니다.

Amazon S3 서비스 기능을 표시하는 API 리소스를 생성하려면

1. Amazon S3 버킷을 생성한 AWS 리전에서 MyS3라는 API를 생성합니다. 이 API의 루트 리소스 /는 Amazon S3 서비스를 가리킵니다. 이 단계에서는/{folder}와 /{item}이라는 두 개의 추가 리소스를 생성합니다.
2. API의 루트 리소스를 선택한 다음 리소스 생성을 선택합니다.
3. 프록시 리소스는 꺼진 상태로 둡니다.
4. 리소스 경로에서 /를 선택합니다.
5. 리소스 이름에 **`{folder}`**을 입력합니다.
6. 오리진 간 리소스 공유(CORS)를 선택 취소된 상태로 둡니다.
7. 리소스 생성을 선택합니다.
8. /{folder} 리소스를 선택한 다음 리소스 생성을 선택합니다.
9. 이전 단계를 사용하여 이름이 **`{item}`**인 하위 리소스 /{folder}를 생성합니다.

   최종 API는 다음과 같아야 합니다.



API 메서드를 노출하여 호출자의 Amazon S3 버킷 나열

호출자의 Amazon S3 버킷 목록을 가져오는 중에 Amazon S3에서 GET Service 작업을 호출합니다. API의 루트 리소스(/)에서 GET 메서드를 생성합니다. 다음과 같이 Amazon S3와 통합하도록 GET 메서드를 구성합니다.

API의 **`GET /`** 메서드를 생성하고 초기화하는 방법

1. / 리소스를 선택한 다음 메서드 생성을 선택합니다.
2. 메서드 유형에서 GET을 선택합니다.
3. 통합 유형에서 AWS 서비스를 선택합니다.
4. AWS 리전에서 Amazon S3 버킷을 생성한 AWS 리전을 선택합니다.
5. AWS 서비스에서 Amazon Simple Storage Service를 선택합니다.
6. AWS 하위 도메인은 비워 둡니다.
7. HTTP 메서드에서 GET을 선택합니다.
8. 작업 유형에서 경로 재정의 사용을 선택합니다. 경로 재정의를 사용하면 API Gateway가 클라이언트 요청을 해당하는 Amazon S3 REST API path-style request로 Amazon S3로 전달합니다. 여기에서 Amazon S3 리소스가 `s3-host-name/bucket/key` 패턴의 리소스 경로에 의해 표현됩니다. API Gateway는 `s3-host-name`을 설정하고 클라이언트가 지정한 `bucket` 및 `key`를 클라이언트에서 Amazon S3로 전달합니다.
9. 경로 재정의에서 /를 입력합니다.
10. 실행 역할에 **`APIGatewayS3ProxyPolicy`**의 ARN을 입력합니다.
11. 메서드 생성을 선택합니다.

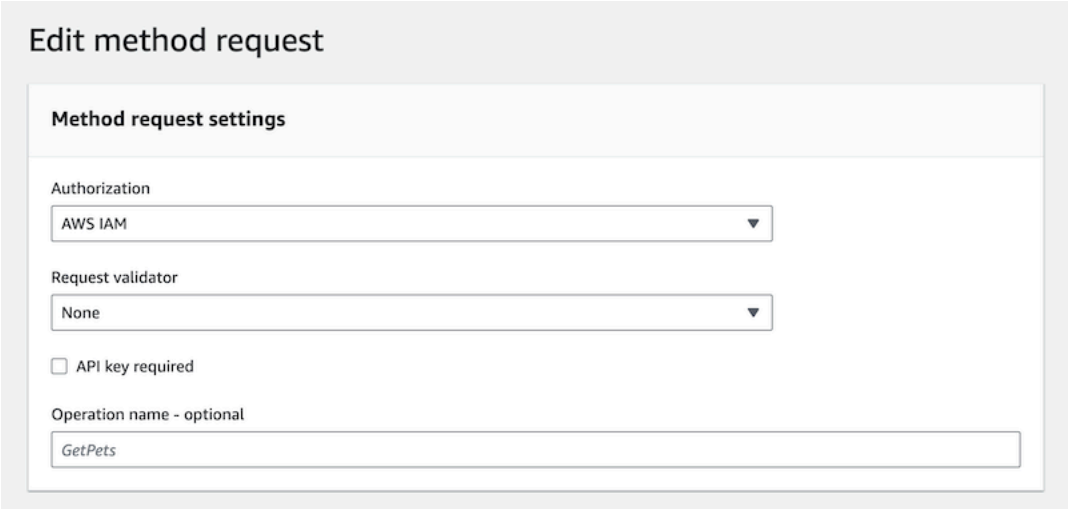
이 설정은 프런트 엔드 `GET https://`*`your-api-host`*`/`*`stage`*`/` 요청을 백엔드 `GET https://`*`your-s3-host`*`/`와 통합합니다.

> **Note**
>
> 초기 설정 후 메서드의 통합 요청(Integration Request) 페이지에서 이러한 설정을 수정할 수 있습니다.

API의 이 메서드를 호출할 수 있는 사람을 제어하려면 메서드 권한 부여 플래그를 켜고 `AWS_IAM`으로 설정합니다.

IAM을 활성화하여 GET / 메서드에 대한 액세스를 제어하는 방법

1. 메서드 요청 탭의 메서드 요청 설정에서 편집을 선택합니다.
2. 권한 부여의 드롭다운 메뉴에서 AWS_IAM을 선택합니다.



3. 저장을 선택합니다.

API에서 성공적인 응답과 예외를 호출자에게 적절히 반환하도록 메서드 응답(Method Response)에서 200, 400 및 500 응답을 선언하겠습니다. 여기에 선언되지 않은 상태 코드의 백엔드 응답은 호출자에게 200 응답으로 반환되도록 200 응답에 대한 기본 매핑을 사용합니다.

GET / 메서드에 대한 응답 유형을 선언하려면

1. 메서드 응답 탭의 응답 200에서 편집을 선택합니다.
2. 헤더 추가를 선택하고 다음을 수행합니다.

   a. 헤더 이름에 **Content-Type**을 입력합니다.

   b. 헤더 추가(Add header)를 선택합니다.

   이 단계를 반복하여 **Timestamp** 헤더와 **Content-Length** 헤더를 생성합니다.
3. 저장을 선택합니다.
4. 메서드 응답 탭의 메서드 응답에서 응답 생성을 선택합니다.
5. HTTP 상태 코드에 400을 입력합니다.

   이 응답에는 헤더를 설정하지 않습니다.

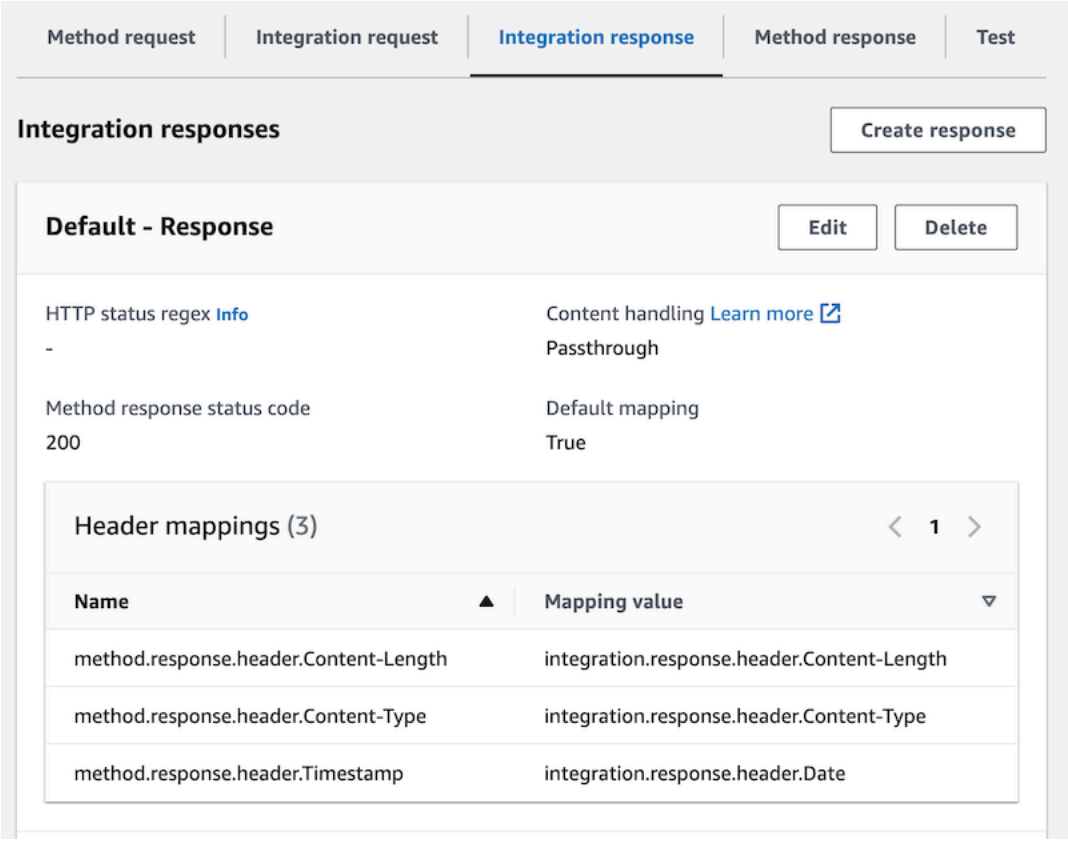
6. 저장을 선택합니다.
7. 다음 단계를 반복하여 500 응답을 생성합니다.

   이 응답에는 헤더를 설정하지 않습니다.

Amazon S3의 통합 응답이 버킷 목록을 XML 페이로드로 반환하고 API Gateway의 기본 메서드 응답이 JSON 페이로드를 반환하므로, 백엔드 Content-Type 헤더 파라미터 값을 해당 프런트엔드 값에 매핑해야 합니다. 그렇지 않으면, 응답 본문이 실제로 XML 문자열일 경우 클라이언트는 콘텐츠 유형에 대한 `application/json`을 수신하게 됩니다. 다음 절차에서는 이렇게 설정하는 방법을 보여줍니다. 또한 클라이언트에게 다른 헤더 파라미터(예: Date 및 Content-Length)를 표시하고자 합니다.

GET / 메서드에 대한 응답 헤더 매핑 설정 방법

1. 통합 응답 탭의 기본값 - 응답에서 편집을 선택합니다.
2. Content-Length 헤더에서 매핑 값으로 **`integration.response.header.Content-Length`**를 입력합니다.
3. Content-Type 헤더에서 매핑 값으로 **`integration.response.header.Content-Type`**을 입력합니다.
4. Timestamp 헤더에서 매핑 값으로 **`integration.response.header.Date`**를 입력합니다.
5. 저장을 선택합니다. 결과는 다음과 같습니다.

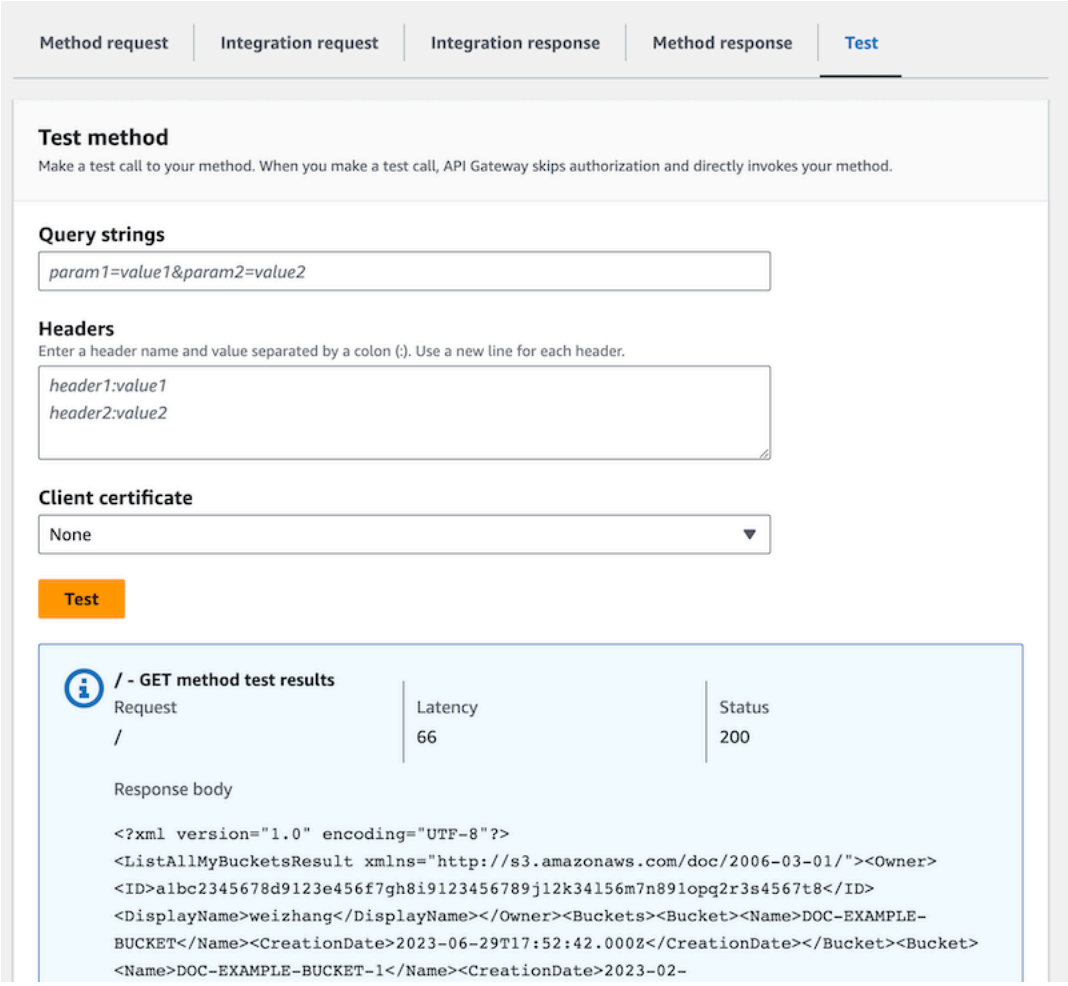


6. 통합 응답 탭의 통합 응답에서 응답 생성을 선택합니다.
7. HTTP 상태 regex에 **4\d{2}**를 입력합니다. 이렇게 하면 모든 4xx HTTP 응답 상태 코드가 메서드 응답에 매핑됩니다.
8. 메서드 응답 상태 코드에서 **400**을 선택합니다.
9. 생성을 선택합니다.
10. 다음 단계를 반복하여 500 메서드 응답의 통합 응답을 생성합니다. HTTP 상태 regex에 **5\d{2}**를 입력합니다.

지금까지 구성한 API를 테스트해 보겠습니다.

GET / 메서드를 테스트하는 방법

1. 테스트 탭을 선택합니다. 탭을 표시하려면 오른쪽 화살표 버튼을 선택해야 할 수도 있습니다.
2. 테스트를 선택합니다. 결과는 다음 이미지와 같아야 합니다.



API 메서드를 노출하여 Amazon S3 버킷 액세스

Amazon S3 버킷을 사용하려면 /{folder} 리소스에서 GET 메서드를 노출하여 버킷의 객체를 나열합니다. 지침은 API 메서드를 노출하여 호출자의 Amazon S3 버킷 나열 단원에 설명한 지침과 비슷합니다. 더 많은 메서드를 보려면 Amazon S3 프록시로 샘플 API의 OpenAPI 정의에서 샘플 API를 가져올 수 있습니다.

폴더 리소스에 GET 메서드를 노출하려면

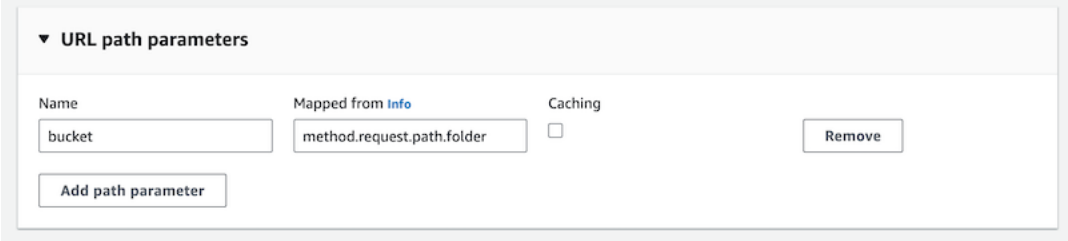
1. /{folder} 리소스를 선택한 다음 메서드 생성을 선택합니다.
2. 메서드 유형에서 GET을 선택합니다.
3. 통합 유형에서 AWS 서비스를 선택합니다.
4. AWS 리전에서 Amazon S3 버킷을 생성한 AWS 리전을 선택합니다.
5. AWS 서비스에서 Amazon Simple Storage Service를 선택합니다.

6. AWS 하위 도메인은 비워 둡니다.
7. HTTP 메서드에서 GET을 선택합니다.
8. 작업 유형에서 경로 재정의 사용을 선택합니다.
9. 경로 재정의에서 **{bucket}**를 입력합니다.
10. 실행 역할에 **APIGatewayS3ProxyPolicy**의 ARN을 입력합니다.
11. 메서드 생성을 선택합니다.

Amazon S3 엔드포인트 URL에 {folder} 경로 파라미터를 설정합니다. 메서드 요청의 {folder} 경로 파라미터를 통합 요청의 {bucket} 경로 파라미터에 매핑해야 합니다.

{folder}를 **{bucket}**에 매핑하려면

1. 통합 요청 탭의 통합 요청 설정에서 편집을 선택합니다.
2. URL 경로 파라미터를 선택한 다음 경로 파라미터 추가를 선택합니다.
3. 이름에 **bucket**를 입력합니다.
4. 다음에서 매핑됨에 **method.request.path.folder**을 입력합니다. 구성은 다음과 비슷해야 합니다.



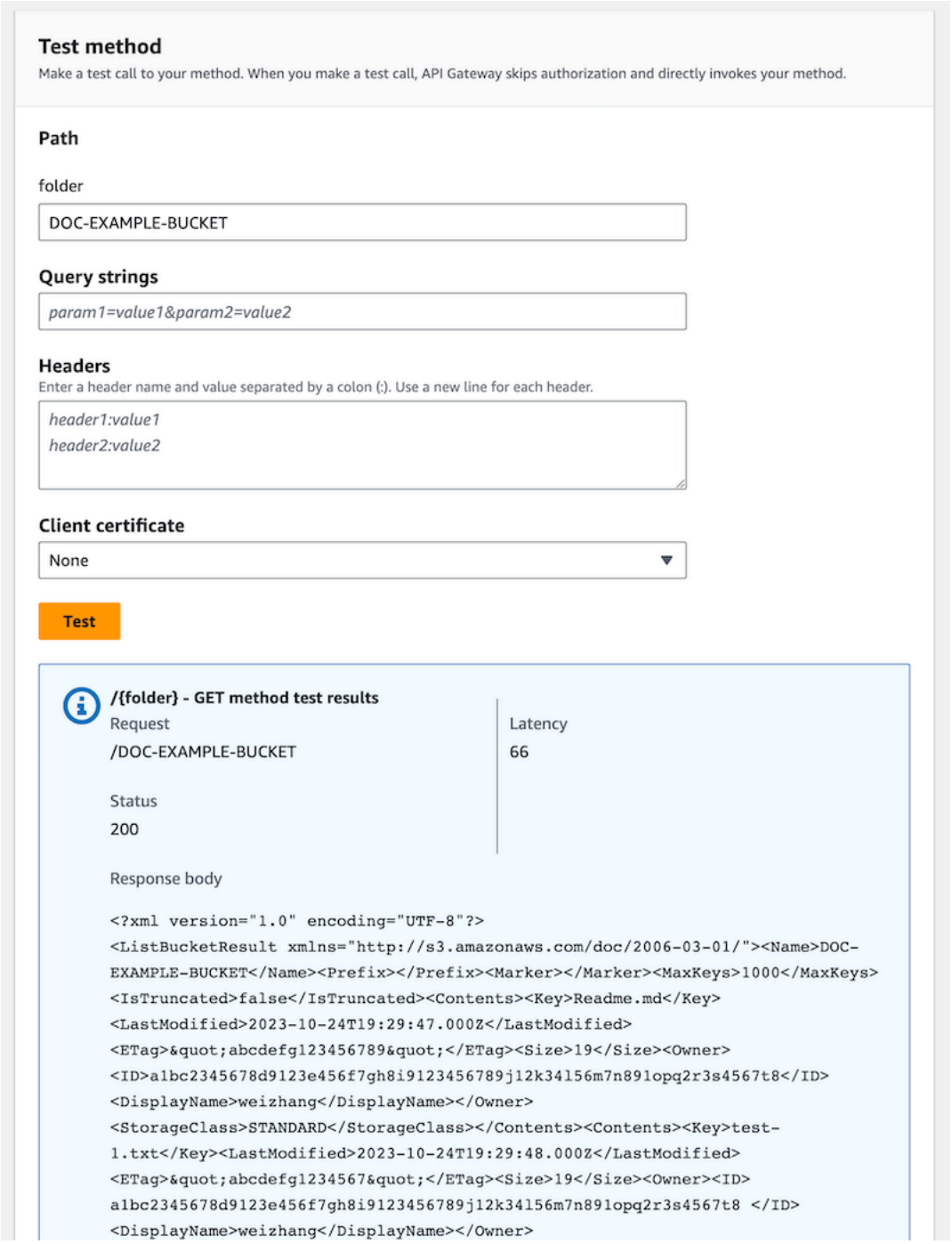
5. 저장을 선택합니다.

이제 API를 테스트합니다.

/{folder} GET 메서드를 테스트하려면

1. 테스트 탭을 선택합니다. 탭을 표시하려면 오른쪽 화살표 버튼을 선택해야 할 수도 있습니다.
2. 경로의 folder에서 버킷 이름을 입력합니다.
3. 테스트를 선택합니다.

   테스트 결과에는 버킷의 객체 목록이 포함됩니다.



API 메서드를 노출하여 버킷에서 Amazon S3 객체 액세스

Amazon S3에서는 지정된 버킷에서 객체를 액세스 및 관리하는 GET, DELETE, HEAD, OPTIONS, POST 및 PUT 작업을 지원합니다. 이 자습서에서는 `{folder}/{item}` 리소스에 `GET` 메서드를 노출하여 버킷에서 이미지를 가져옵니다. `{folder}/{item}` 리소스의 더 많은 적용 사례는 샘플 API를 참조하세요. [Amazon S3 프록시로 샘플 API의 OpenAPI 정의]

아이템 리소스에 GET 메서드 노출

1. /{item} 리소스를 선택한 다음 메서드 생성을 선택합니다.
2. 메서드 유형에서 GET을 선택합니다.
3. 통합 유형에서 AWS 서비스를 선택합니다.
4. AWS 리전에서 Amazon S3 버킷을 생성한 AWS 리전을 선택합니다.
5. AWS 서비스에서 Amazon Simple Storage Service를 선택합니다.
6. AWS 하위 도메인은 비워 둡니다.
7. HTTP 메서드에서 GET을 선택합니다.
8. 작업 유형에서 경로 재정의 사용을 선택합니다.
9. 경로 재정의에 {bucket}/{object}를 입력합니다.
10. 실행 역할에 **APIGatewayS3ProxyPolicy**의 ARN을 입력합니다.
11. 메서드 생성을 선택합니다.

Amazon S3 엔드포인트 URL에 `{folder}` 및 `{item}` 경로 파라미터를 설정합니다. 메서드 요청의 경로 파라미터를 통합 요청의 경로 파라미터에 매핑해야 합니다.

이 단계에서는 다음 작업을 수행합니다.

- 메서드 요청의 `{folder}` 경로 파라미터를 통합 요청의 `{bucket}` 경로 파라미터에 매핑합니다.
- 메서드 요청의 `{item}` 경로 파라미터를 통합 요청의 `{object}` 경로 파라미터에 매핑합니다.

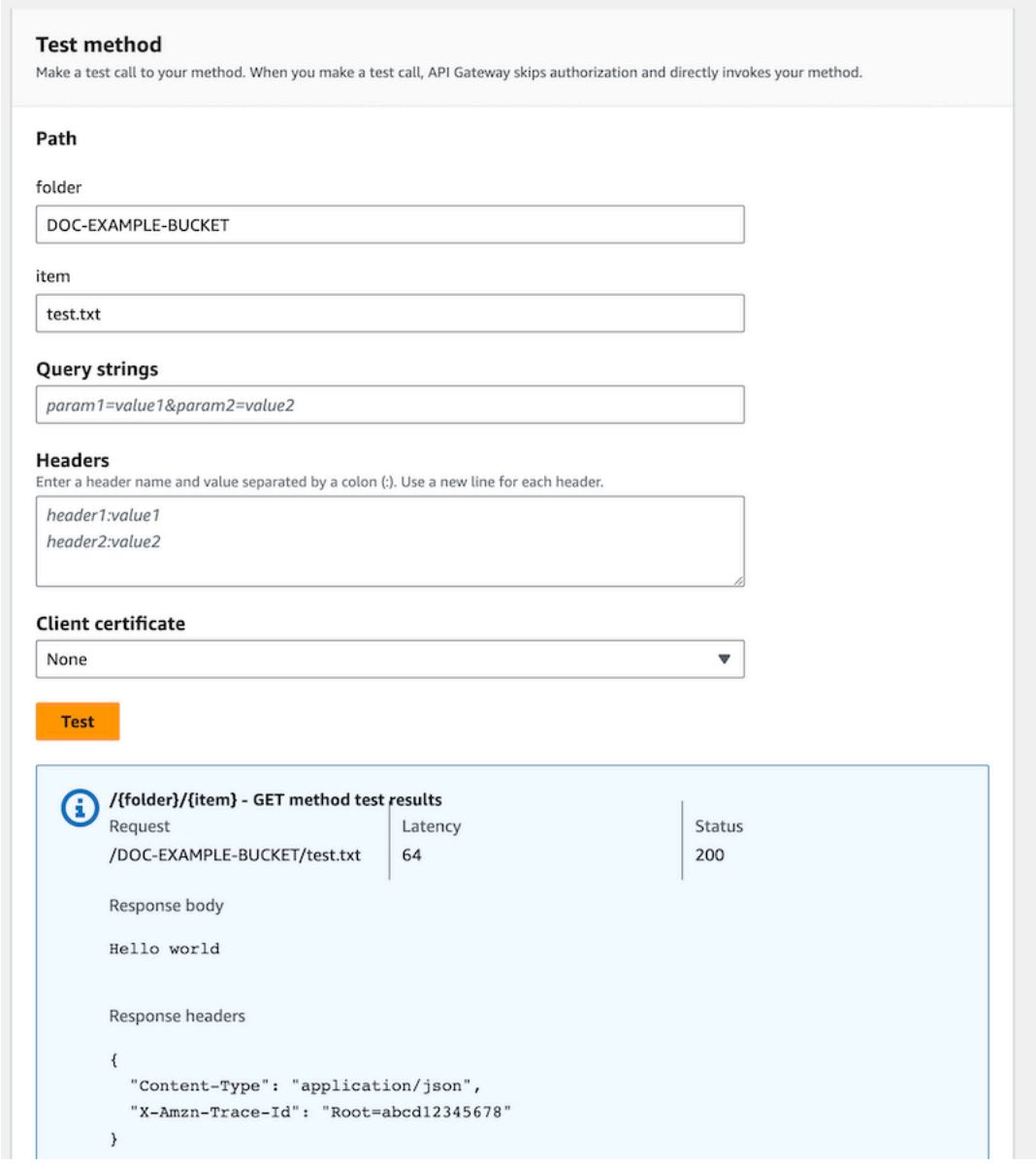
{folder}를 **{bucket}**에 매핑하고 **{item}**을 **{object}**에 매핑하려면

1. 통합 요청 탭의 통합 요청 설정에서 편집을 선택합니다.
2. URL 경로 파라미터를 선택합니다.
3. 경로 파라미터 추가를 선택합니다.
4. 이름에 **bucket**를 입력합니다.
5. 다음에서 매핑됨에 **method.request.path.folder**을 입력합니다.
6. 경로 파라미터 추가를 선택합니다.
7. 이름에 **object**를 입력합니다.
8. 다음에서 매핑됨에 **method.request.path.item**을 입력합니다.
9. 저장을 선택합니다.

`/{folder}/{object}` `GET` 메서드를 테스트하려면

1. 테스트 탭을 선택합니다. 탭을 표시하려면 오른쪽 화살표 버튼을 선택해야 할 수도 있습니다.
2. 경로의 folder에서 버킷 이름을 입력합니다.
3. 경로의 item에서 항목 이름을 입력합니다.
4. 테스트를 선택합니다.

   응답 본문에는 항목의 콘텐츠가 포함됩니다.



이 요청은 (“Hello world”)의 일반 텍스트를 해당 Amazon S3 버킷(DOC-EXAMPLE-BUCKET)에 지정된 파일(test.txt)의 콘텐츠로 올바르게 반환합니다.

API Gateway에서 utf-8 인코딩된 JSON 콘텐츠 이외의 것으로 간주되는 이진 파일을 다운로드 또는 업로드하려면 추가 API 설정이 필요합니다. 개략적으로 설명하면 다음과 같습니다.

S3에서 이진 파일을 다운로드 또는 업로드하려면

1. 해당 파일의 미디어 유형을 API의 binaryMediaTypes에 등록합니다. 이 작업은 콘솔에서 다음과 같이 수행할 수 있습니다.

   a. API에 대한 API 설정을 선택합니다.
   b. 이진 미디어 형식에서 미디어 형식 관리를 선택합니다.
   c. 이진 미디어 형식 추가를 선택한 다음 필요한 미디어 형식(예: `image/png`)을 입력합니다.
   d. 변경 사항 저장을 선택하여 설정을 저장합니다.
2. `Content-Type`(업로드하는 경우) 및/또는 `Accept`(다운로드하는 경우) 헤더를 메서드 요청에 추가하여 클라이언트에게 필요한 이진 미디어 유형을 지정하고 통합 유형에 매핑하도록 요청합니다.
3. 통합 요청(업로드하는 경우) 및 통합 응답(다운로드하는 경우)에서 콘텐츠 처리(Content Handling)를 `Passthrough`로 설정합니다. 해당 콘텐츠 유형에 어떤 매핑 템플릿도 정의되어 있지 않은지 확인하십시오. 자세한 내용은 통합 패스스루 동작 및 VTL 매핑 템플릿 선택 단원을 참조하십시오.

페이로드 크기 제한은 10MB입니다. REST API 구성 및 실행에 대한 API Gateway 할당량을(를) 참조하세요.

Amazon S3에 저장된 파일에서 올바른 콘텐츠 유형이 파일의 메타데이터로 추가되어 있는지 확인하십시오. 스트리밍 가능한 미디어 콘텐츠의 경우, `Content-Disposition:inline`도 메타데이터에 추가되어야 할 수 있습니다.

API Gateway에서의 이진 지원에 대한 자세한 내용은 API Gateway의 콘텐츠 유형 변환 단원을 참조하십시오.

Amazon S3 프록시로 샘플 API의 OpenAPI 정의

다음 OpenAPI 정의에서는 Amazon S3 프록시로 작동하는 API를 설명합니다. 이 API에는 자습서에서 생성한 API보다 더 많은 Amazon S3 작업이 포함되어 있습니다. OpenAPI 정의에는 다음과 같은 메서드가 노출되어 있습니다.

- API의 루트 리소스에서 GET을 노출하여 호출자의 모든 Amazon S3 버킷을 나열합니다.
- 폴더 리소스에서 GET을 노출하여 Amazon S3 버킷의 모든 객체 목록을 봅니다.

- 폴더 리소스에서 PUT을 노출하여 Amazon S3에 버킷을 추가합니다.
- 폴더 리소스에서 DELETE를 노출하여 Amazon S3에서 버킷을 제거합니다.
- 폴더/항목 리소스에서 GET을 노출하여 Amazon S3 버킷에서 객체를 보거나 다운로드합니다.
- 폴더/항목 리소스에서 PUT을 노출하여 Amazon S3 버킷에 객체를 업로드합니다.
- 폴더/항목 리소스에서 HEAD를 노출하여 객체 메타데이터를 Amazon S3 버킷에 가져옵니다.
- 폴더/항목 리소스에서 DELETE를 노출하여 Amazon S3 버킷에서 객체를 제거합니다.

OpenAPI 정의를 사용하여 API를 가져오는 방법에 대한 지침은 OpenAPI를 사용하여 REST API 구성 단원을 참조하십시오.

유사한 API를 생성하는 방법에 대한 지침은 자습서: API Gateway에서 REST API를 Amazon S3 프록시로 생성 섹션을 참조하세요.

AWS IAM 권한 부여를 지원하는 Postman을 사용하여 이 API를 간접적으로 호출하는 방법은 REST API 클라이언트를 사용하여 API 호출 섹션을 참조하세요.

OpenAPI 2.0

```
{
  "swagger": "2.0",
  "info": {
    "version": "2016-10-13T23:04:43Z",
    "title": "MyS3"
  },
  "host": "9gn28ca086.execute-api.{region}.amazonaws.com",
  "basePath": "/S3",
  "schemes": [
    "https"
  ],
  "paths": {
    "/": {
      "get": {
        "produces": [
          "application/json"
        ],
        "responses": {
          "200": {
            "description": "200 response",
            "schema": {
              "$ref": "#/definitions/Empty"
```

```
            },
            "headers": {
              "Content-Length": {
                "type": "string"
              },
              "Timestamp": {
                "type": "string"
              },
              "Content-Type": {
                "type": "string"
              }
            }
          },
          "400": {
            "description": "400 response"
          },
          "500": {
            "description": "500 response"
          }
        },
        "security": [
          {
            "sigv4": []
          }
        ],
        "x-amazon-apigateway-integration": {
          "credentials": "arn:aws:iam::123456789012:role/apigAwsProxyRole",
          "responses": {
            "4\\d{2}": {
              "statusCode": "400"
            },
            "default": {
              "statusCode": "200",
              "responseParameters": {
                "method.response.header.Content-Type":
"integration.response.header.Content-Type",
                "method.response.header.Content-Length":
"integration.response.header.Content-Length",
                "method.response.header.Timestamp":
"integration.response.header.Date"
              }
            },
            "5\\d{2}": {
              "statusCode": "500"
```

```
            }
          },
          "uri": "arn:aws:apigateway:us-west-2:s3:path//",
          "passthroughBehavior": "when_no_match",
          "httpMethod": "GET",
          "type": "aws"
        }
      }
    },
    "/{folder}": {
      "get": {
        "produces": [
          "application/json"
        ],
        "parameters": [
          {
            "name": "folder",
            "in": "path",
            "required": true,
            "type": "string"
          }
        ],
        "responses": {
          "200": {
            "description": "200 response",
            "schema": {
              "$ref": "#/definitions/Empty"
            },
            "headers": {
              "Content-Length": {
                "type": "string"
              },
              "Date": {
                "type": "string"
              },
              "Content-Type": {
                "type": "string"
              }
            }
          },
          "400": {
            "description": "400 response"
          },
          "500": {
```

```
            "description": "500 response"
          }
        },
        "security": [
          {
            "sigv4": []
          }
        ],
        "x-amazon-apigateway-integration": {
          "credentials": "arn:aws:iam::123456789012:role/apigAwsProxyRole",
          "responses": {
            "4\\d{2}": {
              "statusCode": "400"
            },
            "default": {
              "statusCode": "200",
              "responseParameters": {
                "method.response.header.Content-Type":
"integration.response.header.Content-Type",
                "method.response.header.Date": "integration.response.header.Date",
                "method.response.header.Content-Length":
"integration.response.header.content-length"
              }
            },
            "5\\d{2}": {
              "statusCode": "500"
            }
          },
          "requestParameters": {
            "integration.request.path.bucket": "method.request.path.folder"
          },
          "uri": "arn:aws:apigateway:us-west-2:s3:path/{bucket}",
          "passthroughBehavior": "when_no_match",
          "httpMethod": "GET",
          "type": "aws"
        }
      },
      "put": {
        "produces": [
          "application/json"
        ],
        "parameters": [
          {
            "name": "Content-Type",
```

```
            "in": "header",
            "required": false,
            "type": "string"
          },
          {
            "name": "folder",
            "in": "path",
            "required": true,
            "type": "string"
          }
        ],
        "responses": {
          "200": {
            "description": "200 response",
            "schema": {
              "$ref": "#/definitions/Empty"
            },
            "headers": {
              "Content-Length": {
                "type": "string"
              },
              "Content-Type": {
                "type": "string"
              }
            }
          },
          "400": {
            "description": "400 response"
          },
          "500": {
            "description": "500 response"
          }
        },
        "security": [
          {
            "sigv4": []
          }
        ],
        "x-amazon-apigateway-integration": {
          "credentials": "arn:aws:iam::123456789012:role/apigAwsProxyRole",
          "responses": {
            "4\\d{2}": {
              "statusCode": "400"
            },
```

```
            "default": {
              "statusCode": "200",
              "responseParameters": {
                "method.response.header.Content-Type":
"integration.response.header.Content-Type",
                "method.response.header.Content-Length":
"integration.response.header.Content-Length"
              }
            },
            "5\\d{2}": {
              "statusCode": "500"
            }
          },
          "requestParameters": {
            "integration.request.path.bucket": "method.request.path.folder",
            "integration.request.header.Content-Type":
"method.request.header.Content-Type"
          },
          "uri": "arn:aws:apigateway:us-west-2:s3:path/{bucket}",
          "passthroughBehavior": "when_no_match",
          "httpMethod": "PUT",
          "type": "aws"
        }
      },
      "delete": {
        "produces": [
          "application/json"
        ],
        "parameters": [
          {
            "name": "folder",
            "in": "path",
            "required": true,
            "type": "string"
          }
        ],
        "responses": {
          "200": {
            "description": "200 response",
            "schema": {
              "$ref": "#/definitions/Empty"
            },
            "headers": {
              "Date": {
```

```
                "type": "string"
              },
              "Content-Type": {
                "type": "string"
              }
            }
          },
          "400": {
            "description": "400 response"
          },
          "500": {
            "description": "500 response"
          }
        },
        "security": [
          {
            "sigv4": []
          }
        ],
        "x-amazon-apigateway-integration": {
          "credentials": "arn:aws:iam::123456789012:role/apigAwsProxyRole",
          "responses": {
            "4\\d{2}": {
              "statusCode": "400"
            },
            "default": {
              "statusCode": "200",
              "responseParameters": {
                "method.response.header.Content-Type":
"integration.response.header.Content-Type",
                "method.response.header.Date": "integration.response.header.Date"
              }
            },
            "5\\d{2}": {
              "statusCode": "500"
            }
          },
          "requestParameters": {
            "integration.request.path.bucket": "method.request.path.folder"
          },
          "uri": "arn:aws:apigateway:us-west-2:s3:path/{bucket}",
          "passthroughBehavior": "when_no_match",
          "httpMethod": "DELETE",
          "type": "aws"
```

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

```
        "security": [
          {
            "sigv4": []
          }
        ],
        "x-amazon-apigateway-integration": {
          "credentials": "arn:aws:iam::123456789012:role/apigAwsProxyRole",
          "responses": {
            "4\\d{2}": {
              "statusCode": "400"
            },
            "default": {
              "statusCode": "200",
              "responseParameters": {
                "method.response.header.content-type":
"integration.response.header.content-type",
                "method.response.header.Content-Type":
"integration.response.header.Content-Type"
              }
            },
            "5\\d{2}": {
              "statusCode": "500"
            }
          },
          "requestParameters": {
            "integration.request.path.object": "method.request.path.item",
            "integration.request.path.bucket": "method.request.path.folder"
          },
          "uri": "arn:aws:apigateway:us-west-2:s3:path/{bucket}/{object}",
          "passthroughBehavior": "when_no_match",
          "httpMethod": "GET",
          "type": "aws"
        }
      },
      "head": {
        "produces": [
          "application/json"
        ],
        "parameters": [
          {
            "name": "item",
            "in": "path",
            "required": true,
            "type": "string"
```

```
      },
      {
        "name": "folder",
        "in": "path",
        "required": true,
        "type": "string"
      }
    ],
    "responses": {
      "200": {
        "description": "200 response",
        "schema": {
          "$ref": "#/definitions/Empty"
        },
        "headers": {
          "Content-Length": {
            "type": "string"
          },
          "Content-Type": {
            "type": "string"
          }
        }
      },
      "400": {
        "description": "400 response"
      },
      "500": {
        "description": "500 response"
      }
    },
    "security": [
      {
        "sigv4": []
      }
    ],
    "x-amazon-apigateway-integration": {
      "credentials": "arn:aws:iam::123456789012:role/apigAwsProxyRole",
      "responses": {
        "4\\d{2}": {
          "statusCode": "400"
        },
        "default": {
          "statusCode": "200",
          "responseParameters": {
```

```
                "method.response.header.Content-Type":
"integration.response.header.Content-Type",
                "method.response.header.Content-Length":
"integration.response.header.Content-Length"
              }
            },
            "5\\d{2}": {
              "statusCode": "500"
            }
          },
          "requestParameters": {
            "integration.request.path.object": "method.request.path.item",
            "integration.request.path.bucket": "method.request.path.folder"
          },
          "uri": "arn:aws:apigateway:us-west-2:s3:path/{bucket}/{object}",
          "passthroughBehavior": "when_no_match",
          "httpMethod": "HEAD",
          "type": "aws"
        }
      },
      "put": {
        "produces": [
          "application/json"
        ],
        "parameters": [
          {
            "name": "Content-Type",
            "in": "header",
            "required": false,
            "type": "string"
          },
          {
            "name": "item",
            "in": "path",
            "required": true,
            "type": "string"
          },
          {
            "name": "folder",
            "in": "path",
            "required": true,
            "type": "string"
          }
        ],
```

```
        "responses": {
          "200": {
            "description": "200 response",
            "schema": {
              "$ref": "#/definitions/Empty"
            },
            "headers": {
              "Content-Length": {
                "type": "string"
              },
              "Content-Type": {
                "type": "string"
              }
            }
          },
          "400": {
            "description": "400 response"
          },
          "500": {
            "description": "500 response"
          }
        },
        "security": [
          {
            "sigv4": []
          }
        ],
        "x-amazon-apigateway-integration": {
          "credentials": "arn:aws:iam::123456789012:role/apigAwsProxyRole",
          "responses": {
            "4\\d{2}": {
              "statusCode": "400"
            },
            "default": {
              "statusCode": "200",
              "responseParameters": {
                "method.response.header.Content-Type":
"integration.response.header.Content-Type",
                "method.response.header.Content-Length":
"integration.response.header.Content-Length"
              }
            },
            "5\\d{2}": {
              "statusCode": "500"
```

```
            }
          },
          "requestParameters": {
            "integration.request.path.object": "method.request.path.item",
            "integration.request.path.bucket": "method.request.path.folder",
            "integration.request.header.Content-Type":
"method.request.header.Content-Type"
          },
          "uri": "arn:aws:apigateway:us-west-2:s3:path/{bucket}/{object}",
          "passthroughBehavior": "when_no_match",
          "httpMethod": "PUT",
          "type": "aws"
        }
      },
      "delete": {
        "produces": [
          "application/json"
        ],
        "parameters": [
          {
            "name": "item",
            "in": "path",
            "required": true,
            "type": "string"
          },
          {
            "name": "folder",
            "in": "path",
            "required": true,
            "type": "string"
          }
        ],
        "responses": {
          "200": {
            "description": "200 response",
            "schema": {
              "$ref": "#/definitions/Empty"
            },
            "headers": {
              "Content-Length": {
                "type": "string"
              },
              "Content-Type": {
                "type": "string"
```

```
            }
          }
        },
        "400": {
          "description": "400 response"
        },
        "500": {
          "description": "500 response"
        }
      },
      "security": [
        {
          "sigv4": []
        }
      ],
      "x-amazon-apigateway-integration": {
        "credentials": "arn:aws:iam::123456789012:role/apigAwsProxyRole",
        "responses": {
          "4\\d{2}": {
            "statusCode": "400"
          },
          "default": {
            "statusCode": "200"
          },
          "5\\d{2}": {
            "statusCode": "500"
          }
        },
        "requestParameters": {
          "integration.request.path.object": "method.request.path.item",
          "integration.request.path.bucket": "method.request.path.folder"
        },
        "uri": "arn:aws:apigateway:us-west-2:s3:path/{bucket}/{object}",
        "passthroughBehavior": "when_no_match",
        "httpMethod": "DELETE",
        "type": "aws"
      }
    }
  },
  "securityDefinitions": {
    "sigv4": {
      "type": "apiKey",
      "name": "Authorization",
```

```
      "in": "header",
      "x-amazon-apigateway-authtype": "awsSigv4"
    }
  },
  "definitions": {
    "Empty": {
      "type": "object",
      "title": "Empty Schema"
    }
  }
}
```

OpenAPI 3.0

```
{
  "openapi" : "3.0.1",
  "info" : {
    "title" : "MyS3",
    "version" : "2016-10-13T23:04:43Z"
  },
  "servers" : [ {
    "url" : "https://9gn28ca086.execute-api.{region}.amazonaws.com/{basePath}",
    "variables" : {
      "basePath" : {
        "default" : "S3"
      }
    }
  } ],
  "paths" : {
    "/{folder}" : {
      "get" : {
        "parameters" : [ {
          "name" : "folder",
          "in" : "path",
          "required" : true,
          "schema" : {
            "type" : "string"
          }
        } ],
        "responses" : {
          "400" : {
            "description" : "400 response",
            "content" : { }
```

```
        },
        "500" : {
          "description" : "500 response",
          "content" : { }
        },
        "200" : {
          "description" : "200 response",
          "headers" : {
            "Content-Length" : {
              "schema" : {
                "type" : "string"
              }
            },
            "Date" : {
              "schema" : {
                "type" : "string"
              }
            },
            "Content-Type" : {
              "schema" : {
                "type" : "string"
              }
            }
          },
          "content" : {
            "application/json" : {
              "schema" : {
                "$ref" : "#/components/schemas/Empty"
              }
            }
          }
        }
      },
      "x-amazon-apigateway-integration" : {
        "credentials" : "arn:aws:iam::123456789012:role/apigAwsProxyRole",
        "httpMethod" : "GET",
        "uri" : "arn:aws:apigateway:us-west-2:s3:path/{bucket}",
        "responses" : {
          "4\\d{2}" : {
            "statusCode" : "400"
          },
          "default" : {
            "statusCode" : "200",
            "responseParameters" : {
```

```
                "method.response.header.Content-Type" :
"integration.response.header.Content-Type",
                "method.response.header.Date" : "integration.response.header.Date",
                "method.response.header.Content-Length" :
"integration.response.header.content-length"
              }
            },
            "5\\d{2}" : {
              "statusCode" : "500"
            }
          },
          "requestParameters" : {
            "integration.request.path.bucket" : "method.request.path.folder"
          },
          "passthroughBehavior" : "when_no_match",
          "type" : "aws"
        }
      },
      "put" : {
        "parameters" : [ {
          "name" : "Content-Type",
          "in" : "header",
          "schema" : {
            "type" : "string"
          }
        }, {
          "name" : "folder",
          "in" : "path",
          "required" : true,
          "schema" : {
            "type" : "string"
          }
        } ],
        "responses" : {
          "400" : {
            "description" : "400 response",
            "content" : { }
          },
          "500" : {
            "description" : "500 response",
            "content" : { }
          },
          "200" : {
            "description" : "200 response",
```

```
            "headers" : {
              "Content-Length" : {
                "schema" : {
                  "type" : "string"
                }
              },
              "Content-Type" : {
                "schema" : {
                  "type" : "string"
                }
              }
            },
            "content" : {
              "application/json" : {
                "schema" : {
                  "$ref" : "#/components/schemas/Empty"
                }
              }
            }
          }
        },
        "x-amazon-apigateway-integration" : {
          "credentials" : "arn:aws:iam::123456789012:role/apigAwsProxyRole",
          "httpMethod" : "PUT",
          "uri" : "arn:aws:apigateway:us-west-2:s3:path/{bucket}",
          "responses" : {
            "4\\d{2}" : {
              "statusCode" : "400"
            },
            "default" : {
              "statusCode" : "200",
              "responseParameters" : {
                "method.response.header.Content-Type" :
"integration.response.header.Content-Type",
                "method.response.header.Content-Length" :
"integration.response.header.Content-Length"
              }
            },
            "5\\d{2}" : {
              "statusCode" : "500"
            }
          },
          "requestParameters" : {
            "integration.request.path.bucket" : "method.request.path.folder",
```

```
            "integration.request.header.Content-Type" :
"method.request.header.Content-Type"
          },
          "passthroughBehavior" : "when_no_match",
          "type" : "aws"
        }
      },
      "delete" : {
        "parameters" : [ {
          "name" : "folder",
          "in" : "path",
          "required" : true,
          "schema" : {
            "type" : "string"
          }
        } ],
        "responses" : {
          "400" : {
            "description" : "400 response",
            "content" : { }
          },
          "500" : {
            "description" : "500 response",
            "content" : { }
          },
          "200" : {
            "description" : "200 response",
            "headers" : {
              "Date" : {
                "schema" : {
                  "type" : "string"
                }
              },
              "Content-Type" : {
                "schema" : {
                  "type" : "string"
                }
              }
            },
            "content" : {
              "application/json" : {
                "schema" : {
                  "$ref" : "#/components/schemas/Empty"
                }
```

```
            }
          }
        }
      },
      "x-amazon-apigateway-integration" : {
        "credentials" : "arn:aws:iam::123456789012:role/apigAwsProxyRole",
        "httpMethod" : "DELETE",
        "uri" : "arn:aws:apigateway:us-west-2:s3:path/{bucket}",
        "responses" : {
          "4\\d{2}" : {
            "statusCode" : "400"
          },
          "default" : {
            "statusCode" : "200",
            "responseParameters" : {
              "method.response.header.Content-Type" :
"integration.response.header.Content-Type",
              "method.response.header.Date" : "integration.response.header.Date"
            }
          },
          "5\\d{2}" : {
            "statusCode" : "500"
          }
        },
        "requestParameters" : {
          "integration.request.path.bucket" : "method.request.path.folder"
        },
        "passthroughBehavior" : "when_no_match",
        "type" : "aws"
      }
    }
  },
  "/{folder}/{item}" : {
    "get" : {
      "parameters" : [ {
        "name" : "item",
        "in" : "path",
        "required" : true,
        "schema" : {
          "type" : "string"
        }
      }, {
        "name" : "folder",
        "in" : "path",
```

```
          "required" : true,
          "schema" : {
            "type" : "string"
          }
        } ],
        "responses" : {
          "400" : {
            "description" : "400 response",
            "content" : { }
          },
          "500" : {
            "description" : "500 response",
            "content" : { }
          },
          "200" : {
            "description" : "200 response",
            "headers" : {
              "content-type" : {
                "schema" : {
                  "type" : "string"
                }
              },
              "Content-Type" : {
                "schema" : {
                  "type" : "string"
                }
              }
            },
            "content" : {
              "application/json" : {
                "schema" : {
                  "$ref" : "#/components/schemas/Empty"
                }
              }
            }
          }
        },
        "x-amazon-apigateway-integration" : {
          "credentials" : "arn:aws:iam::123456789012:role/apigAwsProxyRole",
          "httpMethod" : "GET",
          "uri" : "arn:aws:apigateway:us-west-2:s3:path/{bucket}/{object}",
          "responses" : {
            "4\\d{2}" : {
              "statusCode" : "400"
```

```
            },
            "default" : {
              "statusCode" : "200",
              "responseParameters" : {
                "method.response.header.content-type" :
"integration.response.header.content-type",
                "method.response.header.Content-Type" :
"integration.response.header.Content-Type"
              }
            },
            "5\\d{2}" : {
              "statusCode" : "500"
            }
          },
          "requestParameters" : {
            "integration.request.path.object" : "method.request.path.item",
            "integration.request.path.bucket" : "method.request.path.folder"
          },
          "passthroughBehavior" : "when_no_match",
          "type" : "aws"
        }
      },
      "put" : {
        "parameters" : [ {
          "name" : "Content-Type",
          "in" : "header",
          "schema" : {
            "type" : "string"
          }
        }, {
          "name" : "item",
          "in" : "path",
          "required" : true,
          "schema" : {
            "type" : "string"
          }
        }, {
          "name" : "folder",
          "in" : "path",
          "required" : true,
          "schema" : {
            "type" : "string"
          }
        } ],
```

```
        "responses" : {
          "400" : {
            "description" : "400 response",
            "content" : { }
          },
          "500" : {
            "description" : "500 response",
            "content" : { }
          },
          "200" : {
            "description" : "200 response",
            "headers" : {
              "Content-Length" : {
                "schema" : {
                  "type" : "string"
                }
              },
              "Content-Type" : {
                "schema" : {
                  "type" : "string"
                }
              }
            },
            "content" : {
              "application/json" : {
                "schema" : {
                  "$ref" : "#/components/schemas/Empty"
                }
              }
            }
          }
        },
        "x-amazon-apigateway-integration" : {
          "credentials" : "arn:aws:iam::123456789012:role/apigAwsProxyRole",
          "httpMethod" : "PUT",
          "uri" : "arn:aws:apigateway:us-west-2:s3:path/{bucket}/{object}",
          "responses" : {
            "4\\d{2}" : {
              "statusCode" : "400"
            },
            "default" : {
              "statusCode" : "200",
              "responseParameters" : {
```

```
                "method.response.header.Content-Type" :
"integration.response.header.Content-Type",
                "method.response.header.Content-Length" :
"integration.response.header.Content-Length"
              }
            },
            "5\\d{2}" : {
              "statusCode" : "500"
            }
          },
          "requestParameters" : {
            "integration.request.path.object" : "method.request.path.item",
            "integration.request.path.bucket" : "method.request.path.folder",
            "integration.request.header.Content-Type" :
"method.request.header.Content-Type"
          },
          "passthroughBehavior" : "when_no_match",
          "type" : "aws"
        }
      },
      "delete" : {
        "parameters" : [ {
          "name" : "item",
          "in" : "path",
          "required" : true,
          "schema" : {
            "type" : "string"
          }
        }, {
          "name" : "folder",
          "in" : "path",
          "required" : true,
          "schema" : {
            "type" : "string"
          }
        } ],
        "responses" : {
          "400" : {
            "description" : "400 response",
            "content" : { }
          },
          "500" : {
            "description" : "500 response",
            "content" : { }
```

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

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

```
                "schema" : {
                  "$ref" : "#/components/schemas/Empty"
                }
              }
            }
          }
        },
        "x-amazon-apigateway-integration" : {
          "credentials" : "arn:aws:iam::123456789012:role/apigAwsProxyRole",
          "httpMethod" : "HEAD",
          "uri" : "arn:aws:apigateway:us-west-2:s3:path/{bucket}/{object}",
          "responses" : {
            "4\\d{2}" : {
              "statusCode" : "400"
            },
            "default" : {
              "statusCode" : "200",
              "responseParameters" : {
                "method.response.header.Content-Type" :
"integration.response.header.Content-Type",
                "method.response.header.Content-Length" :
"integration.response.header.Content-Length"
              }
            },
            "5\\d{2}" : {
              "statusCode" : "500"
            }
          },
          "requestParameters" : {
            "integration.request.path.object" : "method.request.path.item",
            "integration.request.path.bucket" : "method.request.path.folder"
          },
          "passthroughBehavior" : "when_no_match",
          "type" : "aws"
        }
      }
    },
    "/" : {
      "get" : {
        "responses" : {
          "400" : {
            "description" : "400 response",
            "content" : { }
          },
```

```
        "500" : {
          "description" : "500 response",
          "content" : { }
        },
        "200" : {
          "description" : "200 response",
          "headers" : {
            "Content-Length" : {
              "schema" : {
                "type" : "string"
              }
            },
            "Timestamp" : {
              "schema" : {
                "type" : "string"
              }
            },
            "Content-Type" : {
              "schema" : {
                "type" : "string"
              }
            }
          },
          "content" : {
            "application/json" : {
              "schema" : {
                "$ref" : "#/components/schemas/Empty"
              }
            }
          }
        }
      },
      "x-amazon-apigateway-integration" : {
        "credentials" : "arn:aws:iam::123456789012:role/apigAwsProxyRole",
        "httpMethod" : "GET",
        "uri" : "arn:aws:apigateway:us-west-2:s3:path//",
        "responses" : {
          "4\\d{2}" : {
            "statusCode" : "400"
          },
          "default" : {
            "statusCode" : "200",
            "responseParameters" : {
```

```
                "method.response.header.Content-Type" :
 "integration.response.header.Content-Type",
                "method.response.header.Content-Length" :
 "integration.response.header.Content-Length",
                "method.response.header.Timestamp" :
 "integration.response.header.Date"
              }
            },
            "5\\d{2}" : {
              "statusCode" : "500"
            }
          },
          "passthroughBehavior" : "when_no_match",
          "type" : "aws"
        }
      }
    }
  },
  "components" : {
    "schemas" : {
      "Empty" : {
        "title" : "Empty Schema",
        "type" : "object"
      }
    }
  }
}
```

REST API 클라이언트를 사용하여 API 호출

종합 자습서를 제공하기 위해 이제 AWS IAM 권한 부여를 지원하는 Postman을 사용하여 API를 호출하는 방법을 알아봅니다.

Postman을 사용한 Amazon S3 프록시 API 호출 방법

1. API를 배포 또는 재배포합니다. 단계 편집기(Stage Editor) 상단의 URL 호출(Invoke URL) 옆에 표시된 API의 기본 URL를 적어 둡니다.
2. Postman을 시작합니다.
3. 권한 부여(Authorization)를 선택한 다음 `AWS Signature`를 선택합니다. IAM 사용자의 액세스 키 ID 및 보안 액세스 키를 각각 AccessKey 및 SecretKey 입력 필드에 입력합니다. API가 배포되는

AWS 리전을 AWS 리전 텍스트 상자에 입력합니다. 서비스 이름 입력 필드에 `execute-api`을 입력합니다.

IAM Management Console의 IAM 사용자 계정에 있는 보안 자격 증명(Security Credentials) 탭에서 키 페어를 생성할 수 있습니다.

4. `apig-demo-5`라는 버킷을 *`{region}`* 리전에 있는 Amazon S3 계정에 추가하려면:

> **Note**
>
> 버킷 이름이 전역에 걸쳐 고유해야 합니다.

   a. 드롭다운 메서드 목록에서 PUT을 선택하고 메서드 URL(`https://`*`api-id`*`.execute-api.`*`aws-region`*`.amazonaws.com/`*`stage`*`/`*`folder-name`*)을 입력합니다.

   b. `Content-Type` 헤더 값을 `application/xml`로 설정합니다. 콘텐츠 유형을 설정하기 전에 기존 헤더를 모두 삭제해야 할 수 있습니다.

   c. 본문 메뉴 항목을 선택하고 다음의 XML 조각을 요청 본문으로 입력합니다.

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

   d. 전송을 선택하여 요청을 제출합니다. 성공한 경우 페이로드가 비어 있는 `200 OK` 응답을 수신하게 됩니다.

5. 위 지침에 따라 텍스트 파일을 버킷에 추가합니다. URL에서 **`apig-demo-5`**에 대해 `{folder}`의 버킷 이름을, **`Readme.txt`**에 대해 `{item}`의 파일 이름을 지정하고 파일 콘텐츠로(따라서 요청 페이로드가 됨) **`Hello, World!`**의 텍스트 문자열을 요청 페이로드로 제공하면 요청은 다음과 같이 됩니다.

```
PUT /S3/apig-demo-5/Readme.txt HTTP/1.1
Host: 9gn28ca086.execute-api.{region}.amazonaws.com
Content-Type: application/xml
X-Amz-Date: 20161015T062647Z
Authorization: AWS4-HMAC-SHA256 Credential=access-key-id/20161015/{region}/execute-
api/aws4_request, SignedHeaders=content-length;content-type;host;x-amz-date,
 Signature=ccadb877bdb0d395ca38cc47e18a0d76bb5eaf17007d11e40bf6fb63d28c705b
Cache-Control: no-cache
Postman-Token: 6135d315-9cc4-8af8-1757-90871d00847e
```

```
Hello, World!
```

모든 것이 순조롭게 완료되면 비어 있는 페이로드가 포함된 `200 OK` 응답을 수신할 것입니다.

6. 방금 `Readme.txt` 버킷에 추가한 `apig-demo-5` 파일의 콘텐츠를 얻으려면 다음과 같이 GET 요청을 합니다.

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

성공한 경우 페이로드처럼 `200 OK` 텍스트 문자열이 포함된 `Hello, World!` 응답을 수신하게 됩니다.

7. `apig-demo-5` 버킷에 있는 항목을 나열하려면 다음의 요청을 제출합니다.

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

성공적으로 수행했을 경우, 이 요청을 제출하기 전에 더 많은 파일을 해당 버킷에 추가하지 않았다면, 지정된 버킷에 단일 항목을 보여 주는 XML 페이로드가 포함된 `200 OK` 응답을 수신하게 됩니다

```
<?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**
>
> 이미지를 업로드하거나 다운로드하려면 콘텐츠 처리를 CONVERT_TO_BINARY로 설정해야 합니다.

자습서: API Gateway에서 REST API를 Amazon Kinesis 프록시로 생성

이 페이지에서는 AWS 유형의 통합을 통해 REST API를 생성하고 구성하여 Kinesis에 액세스하는 방법을 설명합니다.

> **Note**
>
> API Gateway API를 Kinesis와 통합하려면 API Gateway와 Kinesis 서비스를 모두 사용할 수 있는 리전을 선택해야 합니다. 리전 사용 가능성은 서비스 엔드포인트 및 할당량 단원을 참조하십시오.

이 예시에서는 클라이언트가 다음 작업을 수행하도록 허용하는 예제 API를 생성합니다.

1. Kinesis에서 사용자의 사용 가능한 스트림 나열
2. 지정된 스트림 생성, 설명 또는 삭제
3. 지정된 스트림에서 데이터 레코드 읽기 또는 쓰기

위의 작업을 수행하기 위해 API에서는 다양한 리소스에 대해 각각 다음을 호출하는 메서드를 노출합니다.

1. Kinesis의 `ListStreams` 작업
2. `CreateStream`, `DescribeStream` 또는 `DeleteStream` 작업
3. Kinesis의 `GetRecords` 또는 `PutRecords`(`PutRecord` 포함) 작업

특히, API를 다음과 같이 빌드합니다.

- API `/streams` 리소스에 HTTP GET 메서드를 노출하고 이 메서드를 Kinesis의 ListStreams작업과 통합하여 호출자 계정의 스트림을 나열합니다.
- API `/streams/{stream-name}` 리소스에 HTTP POST 메서드를 노출하고 이 메서드를 Kinesis의 CreateStream작업과 통합하여 호출자 계정에 명명된 스트림을 생성합니다.
- API `/streams/{stream-name}` 리소스에 HTTP GET 메서드를 노출하고 이 메서드를 Kinesis의 DescribeStream작업과 통합하여 호출자 계정의 명명된 스트림을 설명합니다.
- API `/streams/{stream-name}` 리소스에 HTTP DELETE 메서드를 노출하고 이 메서드를 Kinesis의 DeleteStream작업과 통합하여 호출자 계정에서 스트림을 삭제합니다.
- API `/streams/{stream-name}/record` 리소스에 HTTP PUT 메서드를 노출하고 이 메서드를 Kinesis의 PutRecord작업과 통합합니다. 그러면 클라이언트가 단일 데이터 레코드를 명명된 스트림에 추가할 수 있습니다.
- API `/streams/{stream-name}/records` 리소스에 HTTP PUT 메서드를 노출하고 이 메서드를 Kinesis의 PutRecords작업과 통합합니다. 그러면 클라이언트가 데이터 레코드 목록을 명명된 스트림에 추가할 수 있습니다.
- API `/streams/{stream-name}/records` 리소스에 HTTP GET 메서드를 노출하고 이 메서드를 Kinesis의 GetRecords작업과 통합합니다. 그러면 클라이언트가 지정된 샤드 반복기를 사용하여 명명된 스트림에 데이터 레코드를 나열할 수 있습니다. 샤드 반복기는 순차적으로 데이터 레코드 읽기를 시작할 샤드 위치를 지정합니다.
- API `/streams/{stream-name}/sharditerator` 리소스에 HTTP GET 메서드를 노출하고 이 메서드를 Kinesis의 GetShardIterator작업과 통합합니다. 이 헬퍼 메서드는 Kinesis의 `ListStreams` 작업에 제공되어야 합니다.

여기에 제공된 지침을 다른 Kinesis 작업에 적용할 수 있습니다. Kinesis 작업의 전체 목록은 Amazon Kinesis API 참조를 참조하십시오.

API Gateway 콘솔을 사용하여 샘플 API를 생성하는 대신 API Gateway API 가져오기를 사용하여 샘플 API를 API Gateway로 가져올 수 있습니다. API 가져오기를 사용하는 방법에 대한 자세한 내용은 OpenAPI를 사용하여 REST API 구성 단원을 참조하세요.

API에서 Kinesis 액세스를 위한 IAM 역할 및 정책 생성

API에서 Kinesis 작업을 간접적으로 호출하도록 허용하려면 적절한 IAM 정책을 IAM 역할에 연결해야 합니다.

AWS 서비스 프록시 실행 역할을 만들려면

1. 에 AWS Management Console 로그인하고 https://console.aws.amazon.com/iam/ 에서 IAM 콘솔을 엽니다.
2. [Roles]를 선택합니다.
3. 역할 생성을 선택합니다.
4. 신뢰할 수 있는 개체 유형 선택에서 AWS 서비스를 선택한 다음 API Gateway를 선택하고 API Gateway에서 로그를 로그로 푸시할 수 있도록 허용을 CloudWatch 선택합니다.
5. 다음을 선택한 후 다시 다음을 선택합니다.
6. 역할 이름에 **APIGatewayKinesisProxyPolicy**를 입력한 다음 역할 생성을 선택합니다.
7. 역할 목록에서 방금 생성한 역할을 선택합니다. 스크롤하거나 검색 창을 사용하여 역할을 찾을 수 있습니다.
8. 선택한 역할에 대해 권한 추가 탭을 선택합니다.
9. 드롭다운 목록에서 정책 연결을 선택합니다.
10. 검색 창에 **AmazonKinesisFullAccess**를 입력하고 권한 추가를 선택합니다.

> **Note**
>
> 이 자습서에서는 단순화를 위해 관리형 정책을 사용합니다. 가장 좋은 방법은 필요한 최소 권한을 부여하는 자체 IAM 정책을 생성하는 것입니다.

11. 새로 생성된 역할 ARN은 나중에 사용할 수 있도록 기록해 둡니다.

API를 Kinesis 프록시로 생성하기 시작

다음 단계에 따라 API Gateway 콘솔에서 API를 생성합니다.

Kinesis용 AWS 서비스 프록시로 API를 생성하려면

1. https://console.aws.amazon.com/apigateway에서 API Gateway 콘솔에 로그인합니다.
2. API Gateway를 처음 사용하는 경우, 서비스의 기능을 소개하는 페이지가 나타납니다. REST API에서 빌드를 선택합니다. 예제 API 생성 팝업이 나타나면 확인을 선택합니다.

   API Gateway를 처음 사용하는 것이 아닌 경우 API 생성을 선택합니다. REST API에서 빌드를 선택합니다.
3. 새 API(New API)를 선택합니다.
4. API 이름에 **KinesisProxy**를 입력합니다. 다른 모든 필드에 대한 기본값을 그대로 유지합니다.
5. (선택 사항) 설명에 설명을 입력합니다.
6. API 생성(Create API)을 선택합니다.

API가 생성되면 API Gateway 콘솔에 리소스(Resources) 페이지가 표시됩니다. 이 페이지에는 API의 루트(/) 리소스만 포함되어 있습니다.

Kinesis에 스트림 나열

Kinesis는 다음의 REST API 호출을 통해 `ListStreams` 작업을 지원합니다.

```
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>

{
   ...
}
```

위의 REST API 요청에서 작업은 `Action` 쿼리 파라미터에 지정합니다. 또는 다음과 같이 `X-Amz-Target` 헤더에서 작업을 지정할 수도 있습니다.

```
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
{
   ...
}
```

이 자습서에서는 쿼리 파라미터를 사용하여 작업을 지정합니다.

API에 Kinesis 작업을 표시하려면 `/streams` 리소스를 API의 루트에 추가합니다. 그런 다음 리소스에 `GET` 메서드를 설정하고 그 메서드를 Kinesis의 `ListStreams` 작업과 통합합니다.

다음 절차에서는 API Gateway 콘솔을 사용하여 Kinesis 스트림을 나열하는 방법에 대해 설명합니다.

API Gateway 콘솔을 사용하여 Kinesis 스트림을 나열하려면

1. `/` 리소스를 선택한 다음 리소스 생성을 선택합니다.
2. 리소스 이름에 **`streams`**을 입력합니다.
3. 오리진 간 리소스 공유(CORS)를 꺼진 상태로 둡니다.
4. 리소스 생성을 선택합니다.
5. `/streams` 리소스를 선택한 후 메서드 생성을 선택하고 다음을 수행합니다.
   a. 메서드 유형에서 GET을 선택합니다.

      > **Note**
      >
      > 클라이언트에 의해 호출된 메서드에 대한 HTTP 동사는 백엔드에서 요구하는 통합에 대한 HTTP 동사와 다를 수 있습니다. 여기에서는 `GET`을 선택했는데, 목록 스트림이 본질적으로 읽기(READ) 작업이기 때문입니다.

   b. 통합 유형에서 AWS 서비스를 선택합니다.
   c. 에서 Kinesis 스트림을 생성한 AWS 리전 위치를 선택합니다. AWS 리전
   d. AWS 서비스에서 Kinesis를 선택합니다.
   e. AWS 하위 도메인은 비워 둡니다.
   f. HTTP 메서드(HTTP method)에 대해 POST를 선택합니다.

> **Note**
>
> 여기에서는 POST를 선택했는데, Kinesis의 경우 `ListStreams` 작업이 POST로 호출되기 때문입니다.

g. 작업 유형에서 사용자 작업 이름을 선택합니다.

h. 함수 이름에 **`ListStreams`**를 입력합니다.

i. 실행 역할에 실행 역할의 ARN을 입력합니다.

j. 콘텐츠 처리에서 패스스루의 기본값을 유지합니다.

k. 메서드 생성을 선택합니다.

6. 통합 요청 탭의 통합 요청 설정에서 편집을 선택합니다.
7. 요청 본문 패스스루에서 정의된 템플릿이 없는 경우(권장)를 선택합니다.
8. URL 요청 헤더 파라미터를 선택하고 다음을 수행합니다.

   a. 요청 헤더 파라미터 추가를 선택합니다.

   b. 이름에 **`Content-Type`**를 입력합니다.

   c. 다음에서 매핑됨에 **`'application/x-amz-json-1.1'`**을 입력합니다.

   `Content-Type` 헤더를 `'application/x-amz-json-1.1'`의 정적 값으로 설정하는 요청 파라미터 매핑을 사용하여 입력이 특정 버전의 JSON임을 Kinesis에 알립니다.
9. 매핑 템플릿을 선택한 다음 매핑 템플릿 추가를 선택하고 다음을 수행합니다.

   a. 콘텐츠 유형에 **`application/json`**을 입력합니다.

   b. 템플릿 본문에 **`{}`**를 입력합니다.

   c. 저장을 선택합니다.

   ListStreams요청은 다음 JSON 형식의 페이로드를 사용합니다.

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

하지만 속성은 선택 사항입니다. 여기에서는 기본값을 사용하기 위해 비어 있는 JSON 페이로드를 선택했습니다.

10. Kinesis에서 `ListStreams` 작업을 호출하는 GET 메서드를 /streams 리소스에서 테스트합니다.

    테스트 탭을 선택합니다. 탭을 표시하려면 오른쪽 화살표 버튼을 선택해야 할 수도 있습니다.

    테스트를 선택하여 메서드를 테스트합니다.

    Kinesis에서 이미 두 개의 스트림("myStream" 및 "yourStream")을 생성했다면, 테스트가 다음의 페이로드가 포함된 200 OK 응답을 반환할 것입니다.

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

Kinesis에서 스트림 생성, 설명, 삭제

Kinesis에서 스트림을 생성, 설명, 삭제하는 동안 각각 다음 Kinesis REST API 요청을 생성합니다.

```
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"
}
```

필요한 입력을 메서드 요청의 JSON 페이로드로 수락하고 페이로드를 통합 요청으로 전달하는 API를 빌드할 수 있습니다. 하지만 메서드와 통합 요청 사이와 메서드와 통합 응답 사이에 더 많은 데이터 매핑 예제를 제공하기 위해 API를 약간 다르게 생성합니다.

리소스에GET,POST, Delete HTTP 메서드를 노출합니다. to-be-named Stream {stream-name} 경로 변수를 스트림 리소스의 자리 표시자로 사용하고 이들 API 메서드를 각각 Kinesis의 DescribeStream, CreateStream 및 DeleteStream 작업과 통합합니다. 클라이언트에서 다른 입력 데이터를 헤더, 쿼리 파라미터 또는 메서드 요청의 페이로드로 전달하도록 해야 합니다, 데이터를 필요한 통합 요청 페이로드로 변환하는 매핑 템플릿을 제공합니다.

{stream-name} 리소스를 생성하려면

1. /stream 리소스를 선택한 다음 리소스 생성을 선택합니다.
2. 프록시 리소스는 꺼진 상태로 둡니다.
3. 리소스 경로에서 /streams를 선택합니다.
4. 리소스 이름에 **{stream-name}**을 입력합니다.
5. 오리진 간 리소스 공유(CORS)를 꺼진 상태로 둡니다.
6. 리소스 생성을 선택합니다.

스트림 리소스에서 GET 메서드를 구성하고 테스트하는 방법

1. /{stream-name} 리소스를 선택한 다음 [메소드 생성] 을 선택합니다.
2. 메서드 유형에서 GET을 선택합니다.
3. 통합 유형에서 AWS 서비스를 선택합니다.
4. 에서 Kinesis 스트림을 생성한 AWS 리전 위치를 선택합니다. AWS 리전
5. AWS 서비스에서 Kinesis를 선택합니다.
6. AWS 하위 도메인은 비워 둡니다.
7. HTTP 메서드에 대해 POST를 선택합니다.
8. 작업 유형에서 사용자 작업 이름을 선택합니다.
9. 함수 이름에 **DescribeStream**를 입력합니다.
10. 실행 역할에 실행 역할의 ARN을 입력합니다.
11. 콘텐츠 처리에서 패스스루의 기본값을 유지합니다.
12. 메서드 생성을 선택합니다.
13. 통합 요청 섹션에서 다음 URL 요청 헤더 파라미터를 추가합니다.

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

    작업 절차는 `GET /streams` 메서드에 대한 요청 파라미터 매핑을 설정하는 절차와 같습니다.
14. 다음의 본문 매핑 템플릿을 추가하여 데이터를 `GET /streams/{stream-name}` 메서드 요청에서 `POST /?Action=DescribeStream` 통합 요청으로 매핑합니다.

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

    이 매핑 템플릿은 메서드 요청의 `DescribeStream` 경로 파라미터 값으로부터 Kinesis의 `stream-name` 작업에 대한 필수 통합 요청 페이로드를 생성합니다.
15. Kinesis에서 `DescribeStream` 작업을 간접적으로 호출하는 `GET /stream/{stream-name}` 메서드를 테스트하려면 테스트 탭을 선택합니다.
16. 경로의 stream-name에 기존 Kinesis 스트림의 이름을 입력합니다.
17. 테스트를 선택합니다. 테스트가 성공하면 다음과 비슷한 페이로드와 함께 200 OK 응답이 반환됩니다.

```
{
  "StreamDescription": {
    "HasMoreShards": false,
    "RetentionPeriodHours": 24,
    "Shards": [
      {
        "HashKeyRange": {
          "EndingHashKey": "680564733841876926926749214863536422 90",
          "StartingHashKey": "0"
        },
        "SequenceNumberRange": {
          "StartingSequenceNumber":
 "49559266461454070523309915164834022007924120923395850242"
        },
        "ShardId": "shardId-000000000000"
      },
      ...
      {
        "HashKeyRange": {
          "EndingHashKey": "340282366920938463463374607431768211455",
          "StartingHashKey": "272225893536750770770699685945414569164"
        },
        "SequenceNumberRange": {
          "StartingSequenceNumber":
 "49559266461543273504104037657400164881014714369419771970"
        },
        "ShardId": "shardId-000000000004"
      }
    ],
    "StreamARN": "arn:aws:kinesis:us-east-1:12345678901:stream/myStream",
    "StreamName": "myStream",
    "StreamStatus": "ACTIVE"
  }
}
```

API를 배포한 후에 이 API 메서드에 대해 REST 요청을 할 수 있습니다.

```
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
```

스트림 리소스에서 POST 메서드를 구성하고 테스트하는 방법

1. /{stream-name} 리소스를 선택한 다음 [메소드 생성] 을 선택합니다.
2. 메서드 유형에서 POST를 선택합니다.
3. 통합 유형에서 AWS 서비스를 선택합니다.
4. 에서 Kinesis 스트림을 생성한 AWS 리전 위치를 선택합니다. AWS 리전
5. AWS 서비스에서 Kinesis를 선택합니다.
6. AWS 하위 도메인은 비워 둡니다.
7. HTTP 메서드에 대해 POST를 선택합니다.
8. 작업 유형에서 사용자 작업 이름을 선택합니다.
9. 함수 이름에 **CreateStream**를 입력합니다.
10. 실행 역할에 실행 역할의 ARN을 입력합니다.
11. 콘텐츠 처리에서 패스스루의 기본값을 유지합니다.
12. 메서드 생성을 선택합니다.
13. 통합 요청 섹션에서 다음 URL 요청 헤더 파라미터를 추가합니다.

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

    작업 절차는 `GET /streams` 메서드에 대한 요청 파라미터 매핑을 설정하는 절차와 같습니다.
14. 다음의 본문 매핑 템플릿을 추가하여 데이터를 `POST /streams/{stream-name}` 메서드 요청에서 `POST /?Action=CreateStream` 통합 요청으로 매핑합니다.

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

    앞서 다룬 매핑 템플릿에서, 클라이언트가 메서드 요청 페이로드에서 값을 지정하지 않은 경우 `ShardCount`를 고정값 5로 설정합니다.

15. Kinesis에서 `CreateStream` 작업을 간접적으로 호출하는 `POST /stream/{stream-name}` 메서드를 테스트하려면 테스트 탭을 선택합니다.
16. 경로의 stream-name에 새 Kinesis 스트림의 이름을 입력합니다.
17. 테스트를 선택합니다. 테스트가 성공하면 아무 데이터 없이 200 OK 응답이 반환됩니다.

    API를 배포한 후에 스트림 리소스 상에서 POST 메서드에 대한 REST API 요청을 만들어 Kinesis에서 `CreateStream` 작업을 호출할 수도 있습니다.

```
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
}
```

스트림 리소스에서 DELETE 메서드를 구성하고 테스트하는 방법

1. /{stream-name} 리소스를 선택한 다음 [메소드 생성] 을 선택합니다.
2. 메서드 유형에서 DELETE를 선택합니다.
3. 통합 유형에서 AWS 서비스를 선택합니다.
4. 에서 Kinesis 스트림을 생성한 AWS 리전 위치를 선택합니다. AWS 리전
5. AWS 서비스에서 Kinesis를 선택합니다.
6. AWS 하위 도메인은 비워 둡니다.
7. HTTP 메서드에 대해 POST를 선택합니다.
8. 작업 유형에서 사용자 작업 이름을 선택합니다.
9. 함수 이름에 **`DeleteStream`**를 입력합니다.
10. 실행 역할에 실행 역할의 ARN을 입력합니다.
11. 콘텐츠 처리에서 패스스루의 기본값을 유지합니다.
12. 메서드 생성을 선택합니다.
13. 통합 요청 섹션에서 다음 URL 요청 헤더 파라미터를 추가합니다.

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

작업 절차는 GET /streams 메서드에 대한 요청 파라미터 매핑을 설정하는 절차와 같습니다.

14. 다음의 본문 매핑 템플릿을 추가하여 데이터를 DELETE /streams/{stream-name} 메서드 요청에서 POST /?Action=DeleteStream의 해당 통합 요청으로 매핑합니다.

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

이 매핑 템플릿은 클라이언트에서 제공한 DELETE /streams/{stream-name}의 URL 경로 이름으로부터 stream-name 작업에 대한 필수 입력을 생성합니다.

15. Kinesis에서 DeleteStream 작업을 간접적으로 호출하는 DELETE /stream/{stream-name} 메서드를 테스트하려면 테스트 탭을 선택합니다.
16. 경로의 stream-name에 기존 Kinesis 스트림의 이름을 입력합니다.
17. 테스트를 선택합니다. 테스트가 성공하면 아무 데이터 없이 200 OK 응답이 반환됩니다.

API를 배포한 후에 스트림 리소스 상에서 다음과 같은 DELETE 메서드에 대한 REST API 요청을 만들어 Kinesis에서 DeleteStream 작업을 호출할 수도 있습니다.

```
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

{}
```

Kinesis의 스트림에서 레코드 가져오기 및 추가

Kinesis에서 스트림을 생성한 후 데이터 레코드를 스트림에 추가하고 스트림에서 데이터를 읽을 수 있습니다. 데이터 레코드를 추가하려면 Kinesis에서 PutRecords또는 PutRecord작업을 호출해야 합니다. 첫 번째 작업은 여러 레코드를 추가하고 두 번째 작업은 단일 레코드를 스트림에 추가합니다.

```
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"
}
```

또는

```
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"
}
```

여기서 StreamName은 레코드를 추가할 타겟 스트림을 식별합니다. StreamName, Data 및 PartitionKey는 필요한 입력 데이터입니다. 이 예에서는 모든 선택적 입력 데이터에 대해 기본값을 사용하며 메서드 요청에 대한 입력에서 해당 값을 명시적으로 지정하지 않습니다.

Kinesis에서 데이터를 읽는 것은 작업을 호출하는 것과 같습니다. 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
}
```

샤드 반복기를 가져오기 위한 다음 Kinesis 작업에 표시된 대로 레코드를 가져올 소스 스트림은 필수 ShardIterator 값으로 지정됩니다.

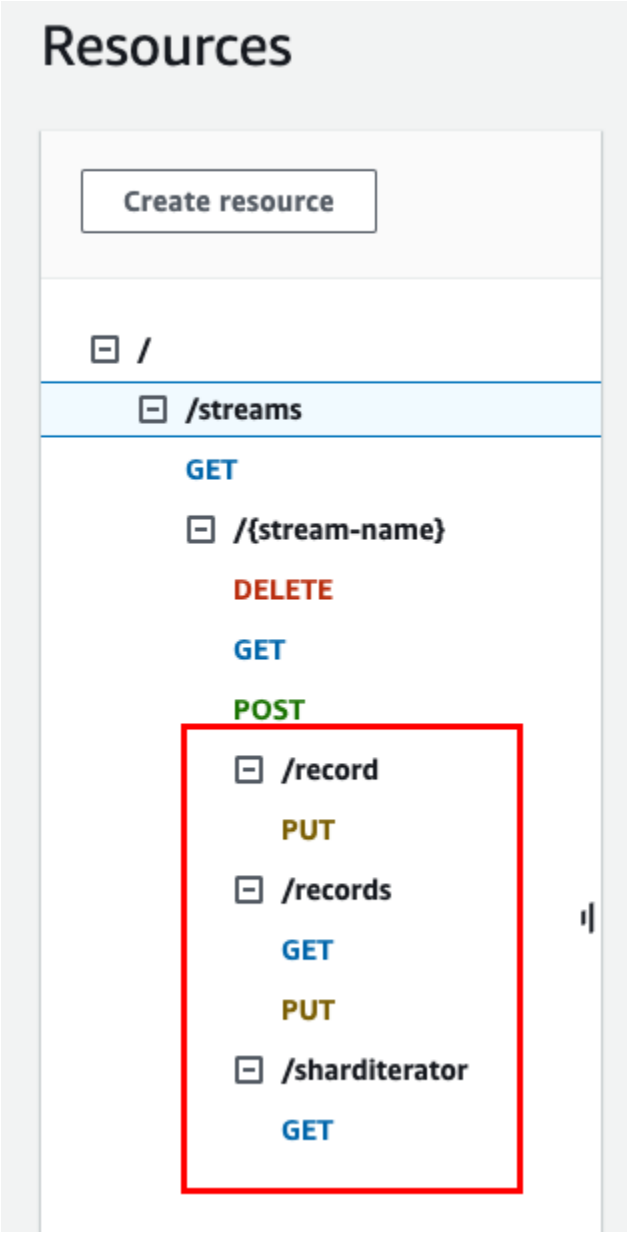
```
POST /?Action=GetShardIterator HTTP/1.1
Host: kinesis.region.domain
Authorization: AWS4-HMAC-SHA256 Credential=..., ...
...
Content-Type: application/x-amz-json-1.1
Content-Length: PayloadSizeBytes

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

GetRecords 및 PutRecords 작업의 경우 명명된 스트림 리소스(GET)에 추가되는 PUT 리소스에 각각 /records 및 /{stream-name} 메서드를 노출합니다. 마찬가지로 PutRecord 리소스에 PUT 작업을 /record 메서드로 노출합니다.

GetRecords 작업은 ShardIterator 헬퍼 작업을 호출하여 GetShardIterator 값을 입력으로 가져오므로 GET 리소스(ShardIterator)에 /sharditerator 헬퍼 메서드를 노출합니다.

/record, /records 및 /sharditerator 리소스를 생성하려면

1. /{stream-name} 리소스를 선택한 다음 리소스 생성을 선택합니다.
2. 프록시 리소스는 꺼진 상태로 둡니다.
3. 리소스 경로에서 `/{stream-name}`를 선택합니다.
4. 리소스 이름에 **`record`**을 입력합니다.
5. 오리진 간 리소스 공유(CORS)를 꺼진 상태로 둡니다.
6. 리소스 생성을 선택합니다.
7. 이전 단계를 반복하여 /records와 /sharditerator 리소스를 생성합니다. 최종 API는 다음과 같아야 합니다.



다음 네 가지 절차에서는 각 메서드를 설정하는 방법, 메서드 요청에서 통합 요청으로 데이터를 매핑하는 방법, 메서드를 테스트하는 방법을 설명합니다.

Kinesis에서 **PutRecord**를 호출할 수 있도록 **PUT /streams/{stream-name}/record** 메서드를 설정하고 테스트하려면:

1. /record를 선택한 다음 [메서드 생성] 을 선택합니다.
2. 메서드 유형에서 PUT을 선택합니다.
3. 통합 유형에서 AWS 서비스를 선택합니다.

4. 에서 Kinesis 스트림을 생성한 AWS 리전 위치를 선택합니다. AWS 리전
5. AWS 서비스에서 Kinesis를 선택합니다.
6. AWS 하위 도메인은 비워 둡니다.
7. HTTP 메서드에 대해 POST를 선택합니다.
8. 작업 유형에서 사용자 작업 이름을 선택합니다.
9. 함수 이름에 **PutRecord**를 입력합니다.
10. 실행 역할에 실행 역할의 ARN을 입력합니다.
11. 콘텐츠 처리에서 패스스루의 기본값을 유지합니다.
12. 메서드 생성을 선택합니다.
13. 통합 요청 섹션에서 다음 URL 요청 헤더 파라미터를 추가합니다.

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

작업 절차는 `GET /streams` 메서드에 대한 요청 파라미터 매핑을 설정하는 절차와 같습니다.

14. 다음의 본문 매핑 템플릿을 추가하여 데이터를 `PUT /streams/{stream-name}/record` 메서드 요청에서 `POST /?Action=PutRecord`의 해당 통합 요청으로 매핑합니다.

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

이 매핑 템플릿에서는 메서드 요청 페이로드의 형식이 다음과 같다고 가정합니다.

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

이 데이터는 다음의 JSON 스키마에 의해 모델링할 수 있습니다.

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

```
  "properties": {
      "Data": { "type": "string" },
      "PartitionKey": { "type": "string" }
  }
}
```

이 스키마를 포함하는 모델을 생성한 다음 그 모델을 사용하면 매핑 템플릿을 용이하게 생성할 수 있습니다. 하지만 모델을 사용하지 않고 매핑 템플릿을 생성할 수도 있습니다.

15. PUT /streams/{stream-name}/record 메서드를 테스트하려면 stream-name 경로 변수를 기존 스트림 이름으로 설정하고, 필수 형식의 페이로드를 공급한 다음, 메서드 요청을 제출합니다. 테스트가 성공하면 다음 형식의 페이로드가 포함된 200 OK 응답이 반환됩니다.

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

Kinesis에서 **PUT /streams/{stream-name}/records**를 호출할 수 있도록 **PutRecords** 메서드를 설정하고 테스트하려면

1. /records 리소스를 선택한 다음 생성 방법을 선택합니다.
2. 메서드 유형에서 PUT을 선택합니다.
3. 통합 유형에서 AWS 서비스를 선택합니다.
4. 에서 Kinesis 스트림을 생성한 AWS 리전 위치를 선택합니다. AWS 리전
5. AWS 서비스에서 Kinesis를 선택합니다.
6. AWS 하위 도메인은 비워 둡니다.
7. HTTP 메서드에 대해 POST를 선택합니다.
8. 작업 유형에서 사용자 작업 이름을 선택합니다.
9. 함수 이름에 **PutRecords**를 입력합니다.
10. 실행 역할에 실행 역할의 ARN을 입력합니다.
11. 콘텐츠 처리에서 패스스루의 기본값을 유지합니다.
12. 메서드 생성을 선택합니다.
13. 통합 요청 섹션에서 다음 URL 요청 헤더 파라미터를 추가합니다.

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

작업 절차는 GET /streams 메서드에 대한 요청 파라미터 매핑을 설정하는 절차와 같습니다.

14. 다음의 매핑 템플릿을 추가하여 데이터를 PUT /streams/{stream-name}/records 메서드 요청에서 POST /?Action=PutRecords의 해당 통합 요청으로 매핑합니다.

```
{
    "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
    ]
}
```

이 매핑 템플릿에서는 메서드 요청 페이로드를 다음의 JSON 스키마에 의해 모델링할 수 있다고 가정합니다.

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

이 스키마를 포함하는 모델을 생성한 다음 그 모델을 사용하면 매핑 템플릿을 용이하게 생성할 수 있습니다. 하지만 모델을 사용하지 않고 매핑 템플릿을 생성할 수도 있습니다.

이 자습서에서는 약간 다른 두 가지 페이로드 형식을 사용하여 API 개발자가 백엔드 데이터 형식을 클라이언트에 표시하거나 클라이언트로부터 숨길 수 있음을 보여 줬습니다. 형식 하나는 `PUT /streams/{stream-name}/records` 메서드(위)에 사용되고, 다른 형식은 `PUT /streams/{stream-name}/record` 메서드(이전 절차)에 사용됩니다. 프로덕션 환경에서는 두 형식의 일관성을 유지해야 합니다.

15. `PUT /streams/{stream-name}/records` 메서드를 테스트하려면 `stream-name` 경로 변수를 기존 스트림으로 설정하고, 다음의 페이로드를 공급한 다음, 메서드 요청을 제출합니다.

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

테스트가 성공하면 다음과 출력과 비슷한 페이로드를 포함하는 200 OK 응답이 반환됩니다.

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

```
  ]
}
```

Kinesis에서 **GET /streams/{stream-name}/sharditerator**를 호출할 수 있도록 **GetShardIterator** 메서드를 설정하고 테스트하려면

GET /streams/{stream-name}/sharditerator 메서드는 GET /streams/{stream-name}/records 메서드를 호출하기 전에 필수 샤드 반복자를 취득하는 헬퍼 반복자입니다.

1. /sharditerator 리소스를 선택한 다음, 메서드 생성을 선택합니다.
2. 메서드 유형에서 GET을 선택합니다.
3. 통합 유형에서 AWS 서비스를 선택합니다.
4. 에서 Kinesis 스트림을 생성한 AWS 리전 위치를 선택합니다. AWS 리전
5. AWS 서비스에서 Kinesis를 선택합니다.
6. AWS 하위 도메인은 비워 둡니다.
7. HTTP 메서드에 대해 POST를 선택합니다.
8. 작업 유형에서 사용자 작업 이름을 선택합니다.
9. 함수 이름에 **GetShardIterator**를 입력합니다.
10. 실행 역할에 실행 역할의 ARN을 입력합니다.
11. 콘텐츠 처리에서 패스스루의 기본값을 유지합니다.
12. 메서드 생성을 선택합니다.

    GetShardIterator 작업은 ShardId 값의 입력이 요구됩니다. 클라이언트에서 제공한 ShardId 값을 전달하려면 다음 단계에 나온 것처럼 shard-id 쿼리 파라미터를 메서드 요청에 추가합니다.
13. 메서드 요청 탭의 메서드 요청 설정에서 편집을 선택합니다.
14. URL 쿼리 문자열 파라미터를 선택하고 다음을 수행합니다.
    a. 쿼리 문자열 추가(Add query string)를 선택합니다.
    b. 이름에 **shard-id**를 입력합니다.
    c. 필수 상태로 유지하고 캐싱을 해제합니다.
    d. 저장을 선택합니다.

다음의 매핑 템플릿에서 shard-id 쿼리 파라미터 값을 Kinesis에서의 GetShardIterator 작업에 대한 입력으로 JSON 페이로드의 ShardId 속성 값으로 설정합니다.

15. 통합 요청 섹션에서, 메서드 요청의 shard-id 및 stream-name 파라미터에서 GetShardIterator 작업에 대한 필수 입력(ShardId 및 StreamName)을 생성하도록 다음과 같은 매핑 템플릿을 추가합니다. 이에 더해, 매핑 템플릿은 기본값으로 ShardIteratorType을 TRIM_HORIZON로 설정합니다.

```
{
    "ShardId": "$input.params('shard-id')",
    "ShardIteratorType": "TRIM_HORIZON",
    "StreamName": "$input.params('stream-name')"
}
```

16. API Gateway 콘솔에서 테스트(Test) 옵션을 사용하여 기존 스트림 이름을 stream-name 경로(Path) 변수 값으로 입력하고, shard-id 쿼리 문자열(Query string)을 기존 ShardId 값(예: shard-000000000004)으로 설정한 다음, 테스트(Test)를 선택합니다.

    올바른 응답 페이로드는 다음 출력과 유사합니다.

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

    [ShardIterator] 값을 기록해 둡니다. 스트림에서 레코드를 가져올 때 필요합니다.

Kinesis에서 **GET /streams/{stream-name}/records** 작업을 호출할 수 있도록 **GetRecords** 메서드를 구성하고 테스트하려면

1. /records 리소스를 선택한 다음 생성 방법을 선택합니다.
2. 메서드 유형에서 GET을 선택합니다.
3. 통합 유형에서 AWS 서비스를 선택합니다.
4. 에서 Kinesis 스트림을 생성한 AWS 리전 위치를 선택합니다. AWS 리전
5. AWS 서비스에서 Kinesis를 선택합니다.
6. AWS 하위 도메인은 비워 둡니다.
7. HTTP 메서드에 대해 POST를 선택합니다.

8. 작업 유형에서 사용자 작업 이름을 선택합니다.
9. 함수 이름에 **GetRecords**를 입력합니다.
10. 실행 역할에 실행 역할의 ARN을 입력합니다.
11. 콘텐츠 처리에서 패스스루의 기본값을 유지합니다.
12. 메서드 생성을 선택합니다.

    GetRecords 작업은 ShardIterator 값의 입력이 요구됩니다. 클라이언트에서 제공한 ShardIterator 값을 전달하려면 다음 단계에 나온 것처럼 Shard-Iterator 헤더 파라미터를 메서드 요청에 추가합니다.
13. 메서드 요청 탭의 메서드 요청 설정에서 편집을 선택합니다.
14. HTTP 요청 헤더를 선택하고 다음을 수행합니다.

    a. 헤더 추가(Add header)를 선택합니다.

    b. 이름에 **Shard-Iterator**를 입력합니다.

    c. 필수 상태로 유지하고 캐싱을 해제합니다.

    d. 저장을 선택합니다.
15. 통합 요청 섹션에서 다음의 본문 매핑 템플릿을 설정하여 Shard-Iterator 헤더 파라미터 값을 Kinesis에서의 GetRecords 작업에 대한 JSON 페이로드의 ShardIterator 속성 값에 매핑합니다.

    ```
    {
        "ShardIterator": "$input.params('Shard-Iterator')"
    }
    ```
16. API Gateway 콘솔에서 테스트 옵션을 사용하여 기존 스트림 이름을 stream-name 경로 변수 값으로 입력하고, Shard-Iterator 헤더를 GET /streams/{stream-name}/sharditerator 메서드(위)의 테스트 실행에서 얻은 ShardIterator 값으로 설정한 다음, 테스트를 선택합니다.

    올바른 응답 페이로드는 다음 출력과 유사합니다.

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

Kinesis 프록시로 샘플 API의 OpenAPI 정의

다음은 이 자습서에서 Kinesis 프록시로서 샘플 API에 대한 OpenAPI 정의입니다.

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"
        ],
```

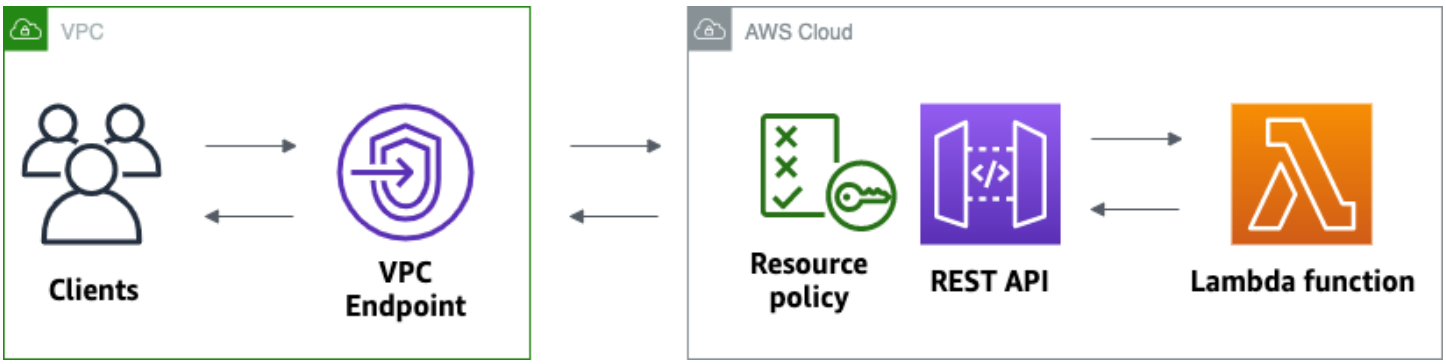
```
        "produces": [
          "application/json"
        ],
        "parameters": [
          {
            "name": "stream-name",
            "in": "path",
            "required": true,
            "type": "string"
          },
          {
            "name": "shard-id",
            "in": "query",
            "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"
              }
            }
          }
        }
      }
    }
  }
}
```

자습서: 프라이빗 REST API 빌드

이 자습서에서는 프라이빗 REST API를 생성합니다. 클라이언트는 Amazon VPC 내에서만 API에 액세스할 수 있습니다. API는 일반적인 보안 요구 사항인 공용 인터넷과 격리됩니다.

이 자습서는 완료되는데 약 30분이 걸립니다. 먼저 AWS CloudFormation 템플릿을 사용하여 Amazon VPC, VPC 엔드포인트, AWS Lambda 함수를 생성하고 API 테스트에 사용할 Amazon EC2 인스턴스를 시작합니다. 다음으로 를 사용하여 프라이빗 API를 만들고 VPC 엔드포인트에서만 액세스를 허용하는 리소스 정책을 연결합니다. AWS Management Console 마지막으로 API를 테스트합니다.



이 자습서를 완료하려면 AWS 계정과 콘솔 액세스 권한이 있는 AWS Identity and Access Management 사용자가 필요합니다. 자세한 설명은 사전 조건 섹션을 참조하세요.

이 자습서에서는 를 사용합니다 AWS Management Console이 API와 모든 관련 리소스를 생성하는 AWS CloudFormation 템플릿은 template.yaml을 참조하십시오.

주제

- 1단계: 종속성 생성
- 2단계: 프라이빗 API 생성
- 3단계: 메서드 및 통합 생성
- 4단계: 리소스 정책 연결
- 5 단계: API 배포
- 6단계: API에 공개적으로 액세스할 수 없는지 확인
- 7단계: VPC의 인스턴스에 연결하고 API 호출
- 8단계: 정리
- 다음 단계: 다음을 사용하여 자동화하십시오. AWS CloudFormation

1단계: 종속성 생성

이 템플릿을 다운로드하고 압축을 풉니다. AWS CloudFormation 템플릿을 사용하여 Amazon VPC, VPC 엔드포인트 및 API의 백엔드 역할을 하는 Lambda 함수를 포함하여 프라이빗 API에 대한 모든 종속성을 생성합니다. 나중에 프라이빗 API를 생성합니다.

스택을 AWS CloudFormation 만들려면

1. https://console.aws.amazon.com/cloudformation 에서 AWS CloudFormation 콘솔을 엽니다.
2. 스택 생성을 선택한 다음 새 리소스 사용(표준)을 선택합니다.

3. 템플릿 지정에서 템플릿 파일 업로드를 선택합니다.
4. 다운로드한 템플릿을 선택합니다.
5. 다음을 선택합니다.
6. 스택 이름에 **private-api-tutorial**을(를) 입력하고 다음을 선택합니다.
7. 스택 옵션 구성에서 다음을 선택합니다.
8. [Capacabilities] 의 경우 계정에서 IAM 리소스를 생성할 AWS CloudFormation 수 있음을 확인하시기 바랍니다.
9. 스택 생성을 선택합니다.

AWS CloudFormation API의 종속성을 프로비저닝합니다. 몇 분 정도 걸릴 수 있습니다. AWS CloudFormation 스택 상태가 CREATE_COMPLETE인 경우 출력을 선택합니다. VPC 엔드포인트 ID를 적어둡니다. 이 자습서의 이후 단계를 위해 필요합니다.

2단계: 프라이빗 API 생성

프라이빗 API를 생성하여 VPC 내의 클라이언트만 액세스할 수 있도록 합니다.

프라이빗 API 생성

1. https://console.aws.amazon.com/apigateway에서 API Gateway 콘솔에 로그인합니다.
2. API 생성(Create API)을 선택한 다음 REST API에 대해 빌드(Build)를 선택합니다.
3. API 이름에서 **private-api-tutorial**을(를) 입력합니다.
4. API 엔드포인트 유형에서 프라이빗을 선택합니다.
5. VPC 엔드포인트 ID의 경우 스택의 출력에서 VPC 엔드포인트 ID를 입력합니다. AWS CloudFormation
6. API 생성(Create API)을 선택합니다.

3단계: 메서드 및 통합 생성

GET 메서드와 Lambda 통합을 생성하여 API에 대한 GET 요청을 처리합니다. 클라이언트가 API를 호출하면 API Gateway는 1단계에서 생성한 Lambda 함수로 요청을 보낸 다음 클라이언트에 대한 응답을 반환합니다.

메서드 및 통합 생성

1. https://console.aws.amazon.com/apigateway에서 API Gateway 콘솔에 로그인합니다.

2. API를 선택합니다.
3. / 리소스를 선택한 다음 메서드 생성을 선택합니다.
4. 메서드 유형에서 GET을 선택합니다.
5. 통합 유형에서 Lambda 함수를 선택합니다.
6. Lambda 프록시 통합을 켭니다. Lambda 프록시 통합을 통해 API Gateway는 정의된 구조를 사용하여 이벤트를 Lambda에 보내고 Lambda 함수의 응답을 HTTP 응답으로 변환합니다.
7. Lambda 함수의 경우 1단계에서 템플릿으로 AWS CloudFormation 생성한 함수를 선택합니다. 함수의 이름은 **private-api-tutorial**로 시작합니다.
8. 메서드 생성을 선택합니다.

4단계: 리소스 정책 연결

클라이언트가 VPC 엔드포인트를 통해서만 API를 호출하도록 허용하는 리소스 정책을 API에 연결합니다. API에 대한 액세스를 추가로 제한하기 위해 VPC 엔드포인트에 대한 VPC 엔드포인트 정책을 구성할 수도 있지만 이 자습서에는 필요하지 않습니다.

리소스 정책 연결

1. https://console.aws.amazon.com/apigateway에서 API Gateway 콘솔에 로그인합니다.
2. API를 선택합니다.
3. 리소스 정책, 정책 생성을 차례로 선택합니다.
4. 다음 정책을 입력합니다. *VPCEID# ## ###* VPC 엔드포인트 ID로 교체하십시오. AWS CloudFormation

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

```
        },
        {
            "Effect": "Allow",
            "Principal": "*",
            "Action": "execute-api:Invoke",
            "Resource": "execute-api:/*"
        }
    ]
}
```

5. 변경 사항 저장를 선택합니다.

5 단계: API 배포

그런 다음 API를 배포하여 Amazon VPC의 클라이언트가 사용할 수 있도록 합니다.

API 배포

1. https://console.aws.amazon.com/apigateway에서 API Gateway 콘솔에 로그인합니다.
2. API를 선택합니다.
3. Deploy API(API 배포)를 선택합니다.
4. 스테이지에서 새 스테이지를 선택합니다.
5. 단계 이름에 **`test`**를 입력합니다.
6. (선택 사항) 설명에 설명을 입력합니다.
7. [Deploy]를 선택합니다.

이제 API를 테스트 할 준비가 되었습니다.

6단계: API에 공개적으로 액세스할 수 없는지 확인

`curl`를 사용하여 Amazon VPC 외부에서 API를 호출할 수 없는지 확인합니다.

API 테스트하기

1. https://console.aws.amazon.com/apigateway에서 API Gateway 콘솔에 로그인합니다.
2. API를 선택합니다.
3. 기본 탐색 창에서 스테이지를 선택한 후 테스트 스테이지를 선택합니다.

4. 스테이지 세부 정보에서 복사 아이콘을 선택하여 API의 호출 URL을 복사합니다. 이 URL은 `https://abcdef123.execute-api.us-west-2.amazonaws.com/test`와 같은 형식입니다. 1단계에서 생성한 VPC 엔드포인트에는 프라이빗 DNS가 활성화되어 있으므로 제공된 URL을 사용하여 API를 호출할 수 있습니다.
5. curl을 사용하여 VPC 외부에서 API를 호출합니다.

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

   Curl은 API의 엔드포인트를 확인할 수 없음을 나타냅니다. 다른 응답을 받은 경우 2단계로 돌아가서 API의 엔드포인트 유형에 대해 프라이빗(Private)으로 선택해야 합니다.

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

그런 다음 VPC의 Amazon EC2 인스턴스에 연결하여 API를 호출합니다.

7단계: VPC의 인스턴스에 연결하고 API 호출

그런 다음 Amazon VPC 내에서 API를 테스트합니다. 프라이빗 API에 액세스하려면 VPC의 Amazon EC2 인스턴스에 연결한 다음 curl을 사용하여 API를 호출합니다. Systems Manager 세션 관리자를 사용하여 브라우저에서 인스턴스에 연결합니다.

API 테스트하기

1. https://console.aws.amazon.com/ec2/에서 Amazon EC2 콘솔을 엽니다.
2. 인스턴스를 선택합니다.
3. 1단계에서 AWS CloudFormation 템플릿으로 만든 이름이 지정된 private-api-tutorial인스턴스를 선택합니다.
4. 연결(Connect)을 선택한 다음 세션 관리자(Session Manager)를 선택합니다.
5. 연결(Connect)을 선택하여 인스턴스에 대한 브라우저 기반 세션을 시작합니다.
6. 세션 관리자 세션에서 curl을 사용하여 API를 호출합니다. Amazon VPC에서 인스턴스를 사용하고 있기 때문에 API를 호출할 수 있습니다.

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

   응답 `Hello from Lambda!`이 수신되는지 확인합니다.

Amazon VPC 내에서만 액세스할 수 있는 API를 성공적으로 생성한 다음 작동하는지 확인했습니다.

8단계: 정리

불필요한 비용을 방지하려면 이 자습서의 일부로 생성한 리소스를 삭제합니다. 다음 단계는 REST API와 AWS CloudFormation 스택을 삭제합니다.

REST API 삭제

1. https://console.aws.amazon.com/apigateway에서 API Gateway 콘솔에 로그인합니다.
2. API 페이지에서 API를 선택합니다. API 작업을 선택하고 API 삭제를 선택한 다음 선택을 확인합니다.

AWS CloudFormation 스택을 삭제하려면

1. https://console.aws.amazon.com/cloudformation 에서 AWS CloudFormation 콘솔을 엽니다.
2. AWS CloudFormation 스택을 선택합니다.
3. 삭제를 선택한 다음 해당 선택을 확인합니다.

다음 단계: 다음을 사용하여 자동화하십시오. AWS CloudFormation

이 자습서와 관련된 모든 AWS 리소스의 생성 및 정리를 자동화할 수 있습니다. 전체 예제 AWS CloudFormation 템플릿에 대해서는 template.yaml을 참조하세요.

Amazon API Gateway HTTP API 자습서

다음 자습서에는 API Gateway HTTP API에 대해 알아보는 데 도움이 되는 실습용 과제가 나와 있습니다.

주제

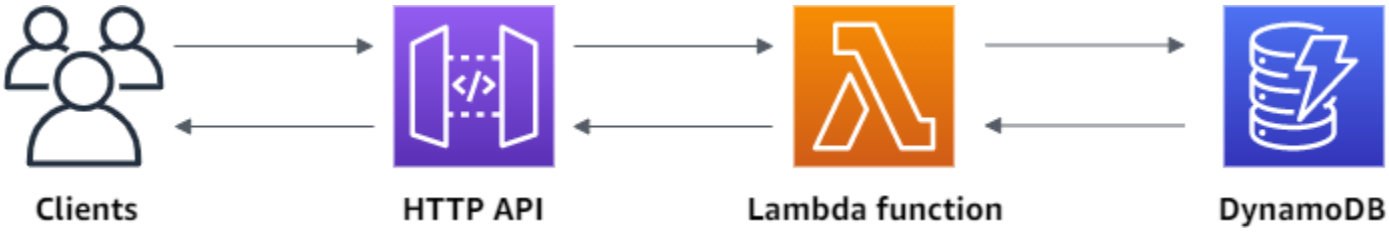
- 자습서: Lambda 및 DynamoDB를 사용한 CRUD API 구축
- 자습서: Amazon ECS 서비스에 대한 프라이빗 통합을 통해 HTTP API 구축

자습서: Lambda 및 DynamoDB를 사용한 CRUD API 구축

이 자습서에서는 DynamoDB 테이블에서 항목을 생성, 읽기, 업데이트 및 삭제하는 서버리스 API를 생성합니다. DynamoDB는 완전관리형 NoSQL 데이터베이스 서비스로서 원활한 확장성과 함께 빠르고 예측 가능한 성능을 제공합니다. 이 자습서를 완료하는 데 약 30분이 소요되며 AWS 프리 티어 내에서 이를 수행할 수 있습니다.

먼저 DynamoDB 콘솔을 사용하여 DynamoDB 테이블을 생성합니다. 그런 다음, AWS Lambda 콘솔을 사용하여 Lambda 함수를 생성합니다. 그런 다음 API Gateway 콘솔을 사용하여 HTTP API를 생성합니다. 마지막으로 API를 테스트합니다.

HTTP API를 호출하면 API Gateway는 요청을 Lambda 함수로 라우팅합니다. Lambda 함수는 DynamoDB와 상호 작용하고 API Gateway에 대한 응답을 반환합니다. 그러고 나면 API Gateway가 응답을 반환합니다.



이 연습을 완료하려면 AWS 계정과 콘솔 액세스 권한이 있는 AWS Identity and Access Management 사용자가 있어야 합니다. 자세한 내용은 사전 조건 단원을 참조하세요.

이 자습서에서는 를 사용합니다AWS Management Console 이 API 및 모든 관련 리소스를 생성하는 AWS SAM 템플릿은 `template.yaml` 단원을 참조하세요.

주제

- 1단계: DynamoDB 테이블 생성
- 2단계: Lambda 함수 생성
- 3단계: HTTP API 생성
- 4단계: 경로 생성
- 5단계: 통합 생성

- 6단계: 경로에 통합 연결
- 7단계: API 테스트
- 8단계: 정리
- 다음 단계: AWS SAM 또는 AWS CloudFormation을 통한 자동화

1단계: DynamoDB 테이블 생성

DynamoDB 테이블을 사용하여 API에 대한 데이터를 저장합니다.

각 항목에는 테이블의 파티션 키로 사용하는 고유 ID가 있습니다.

DynamoDB 테이블을 생성하려면

1. https://console.aws.amazon.com/dynamodb/에서 DynamoDB 콘솔을 엽니다.
2. [Create table]을 선택합니다.
3. 테이블 이름에 **`http-crud-tutorial-items`**을(를) 입력합니다.
4. 파티션 키(Partition key)에 **`id`**를 입력합니다.
5. Create table(테이블 만들기)을 선택합니다.

2단계: Lambda 함수 생성

API의 백엔드에 Lambda 함수를 생성합니다. 이 Lambda 함수는 DynamoDB에서 항목을 생성, 읽기, 업데이트 및 삭제합니다. 이 함수는 API Gateway의 이벤트를 사용하여 DynamoDB와 상호 작용하는 방법을 결정합니다. 단순화를 위해 이 자습서에서는 단일 Lambda 함수를 사용합니다. 가장 좋은 방법은 각 경로에 대해 별도의 함수를 생성하는 것입니다.

Lambda 함수를 만들려면

1. https://console.aws.amazon.com/lambda에서 Lambda 콘솔에 로그인합니다.
2. 함수 생성을 선택합니다.
3. [함수 이름(Function name)]에 **`http-crud-tutorial-function`**을 입력합니다.
4. 사용 권한에서 [기본 실행 역할 변경(Change default execution role)]을 선택합니다.
5. AWS 정책 템플릿에서 새 역할 생성을 선택합니다.
6. [역할 이름(Role name)]에 **`http-crud-tutorial-role`**을 입력합니다.

7. 정책 템플릿에서 **Simple microservice permissions**을(를) 선택합니다. 이 정책은 Lambda 함수에 DynamoDB와 상호 작용할 수 있는 권한을 부여합니다.

> **Note**
>
> 이 자습서에서는 단순화를 위해 관리형 정책을 사용합니다. 가장 좋은 방법은 필요한 최소 권한을 부여하는 자체 IAM 정책을 생성하는 것입니다.

8. 함수 생성을 선택합니다.
9. 콘솔의 코드 편집기에서 `index.mjs`을(를)열고 내용을 다음 코드로 바꿉니다. [배포(Deploy)]를 선택하여 함수를 업데이트합니다.

```
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,
  };
};
```

3단계: HTTP API 생성

HTTP API는 Lambda 함수에 대한 HTTP 엔드포인트를 제공합니다. 이 단계에서는 빈 API를 생성합니다. 다음 단계에서는 API와 Lambda 함수를 연결하기 위한 경로 및 통합을 구성합니다.

HTTP API 생성하기

1. https://console.aws.amazon.com/apigateway에서 API Gateway 콘솔에 로그인합니다.
2. [API 생성(Create API)]을 선택한 다음 HTTP API에 대해 [빌드(Build)]를 선택합니다.
3. API 이름에서 **`http-crud-tutorial-api`**을(를) 입력합니다.
4. [Next]를 선택합니다.
5. 경로 구성에 대해 [다음(Next)]을 선택하여 경로 생성을 건너뜁니다. 나중에 루트를 생성합니다.
6. API Gateway가 생성하는 단계를 검토한 후 [다음(Next)]을 선택합니다.
7. Create를 선택합니다.

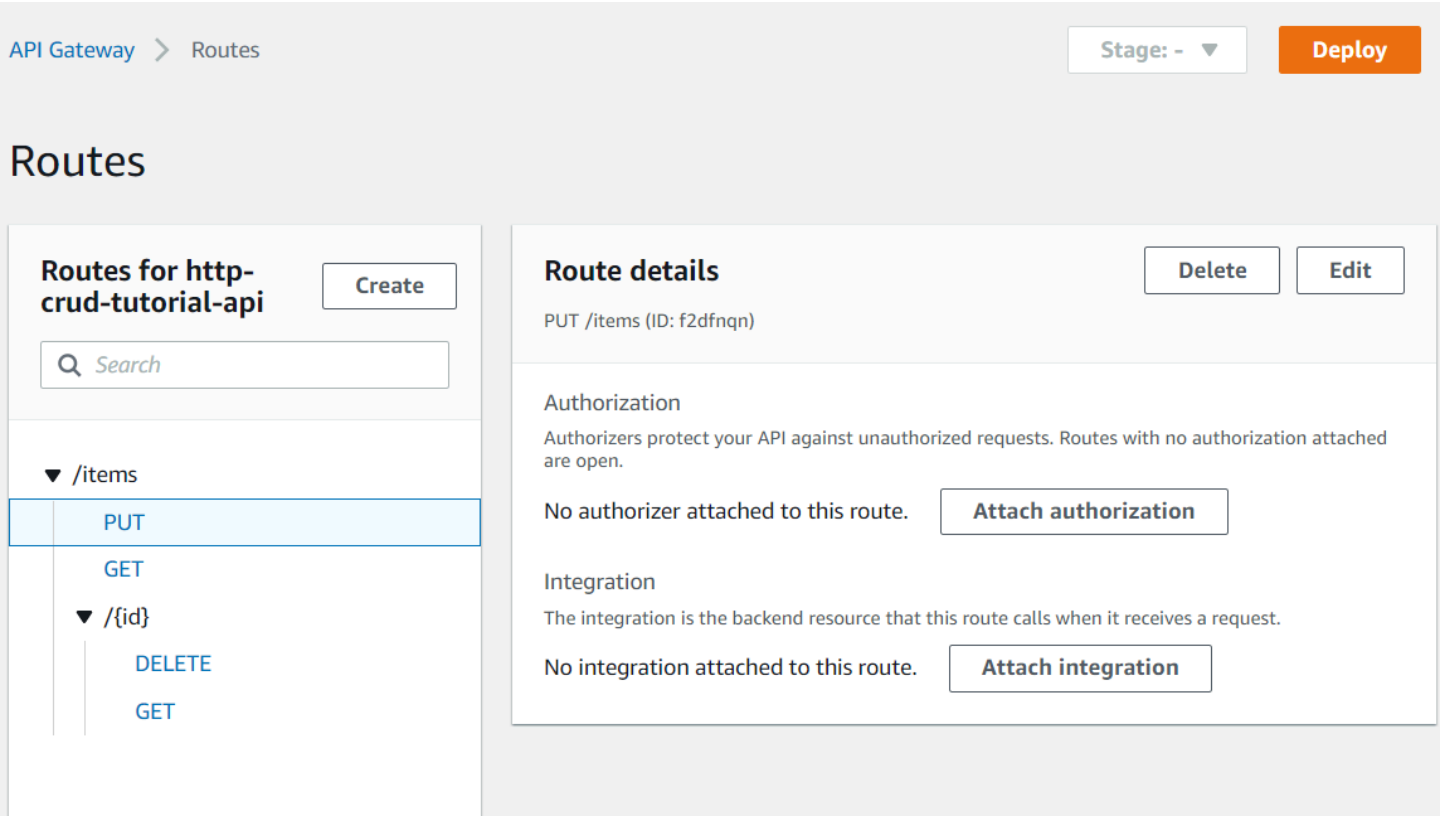
4단계: 경로 생성

경로는 수신 API 요청을 백엔드 리소스로 보내는 방법입니다. 경로는 HTTP 메서드와 리소스 경로(예: `GET /items`), 이렇게 두 부분으로 구성됩니다. 이 예제 API의 경우 네 개의 경로를 생성합니다.

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

경로 생성

1. https://console.aws.amazon.com/apigateway에서 API Gateway 콘솔에 로그인합니다.
2. API를 선택합니다.
3. Routes(라우팅)를 선택합니다.
4. Create를 선택합니다.
5. 메서드(Method)에서 **GET**을(를) 선택합니다.
6. 경로에 **/items/{id}**을(를) 입력합니다. 경로 끝의 {id}은(는) 클라이언트가 요청을 할 때 API Gateway가 요청 경로에서 검색하는 경로 파라미터입니다.
7. Create를 선택합니다.
8. GET /items, DELETE /items/{id} 및 PUT /items에 대해 4-7단계를 반복합니다.



5단계: 통합 생성

백엔드 리소스에 경로를 연결하는 통합을 생성합니다. 이 예제 API에 대해 모든 경로에 사용할 Lambda 통합을 하나 생성합니다.

통합 생성하기

1. https://console.aws.amazon.com/apigateway에서 API Gateway 콘솔에 로그인합니다.
2. API를 선택합니다.
3. [통합(Integrations)]을 선택합니다.
4. [통합 관리(Manage integrations)]를 선택한 다음 [생성(Create)]을 선택합니다.
5. [경로에 이 통합 연결(Attach this integration to a route)]을 건너뜁니다. 이후 단계에서 이 작업을 완료합니다.
6. 통합 유형에서 Lambda 함수(Lambda Function)를 선택합니다.
7. Lambda 함수에서 **`http-crud-tutorial-function`**을(를) 입력합니다.
8. Create를 선택합니다.

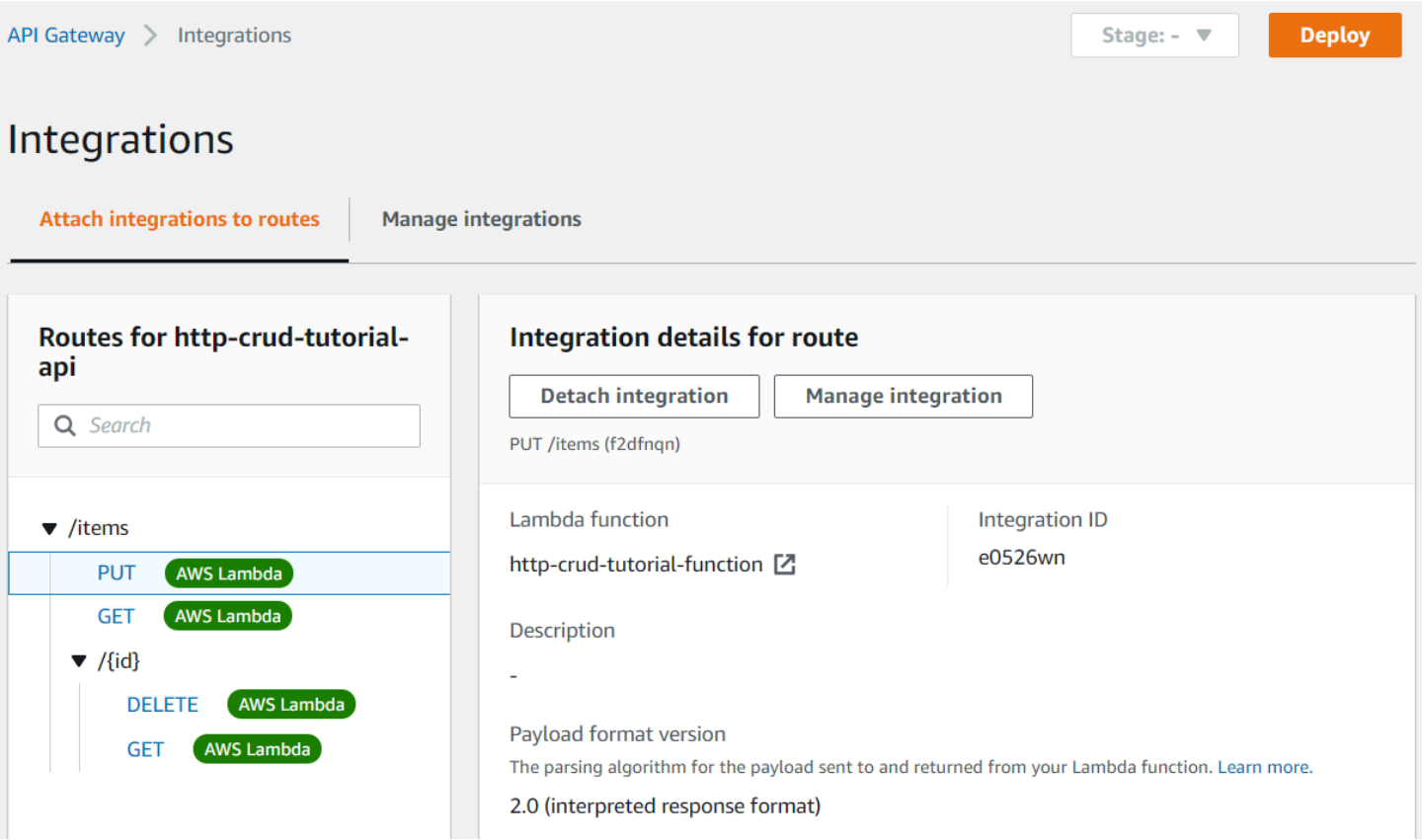
6단계: 경로에 통합 연결

이 예제 API의 경우 모든 경로에 대해 Lambda 통합을 사용합니다. 모든 API의 경로에 통합을 연결한 후 클라이언트가 경로를 호출하면 Lambda 함수가 호출됩니다.

경로에 통합 연결하기

1. https://console.aws.amazon.com/apigateway에서 API Gateway 콘솔에 로그인합니다.
2. API를 선택합니다.
3. [통합(Integrations)]을 선택합니다.
4. 경로를 선택합니다.
5. 기존 통합 선택에서 **`http-crud-tutorial-function`**을(를) 선택합니다.
6. [통합 연결(Attach integration)]을 선택합니다.
7. 모든 경로에 대해 4-6단계를 반복합니다.

모든 경로가 AWS Lambda 통합이 연결되어 있음을 보여줍니다.



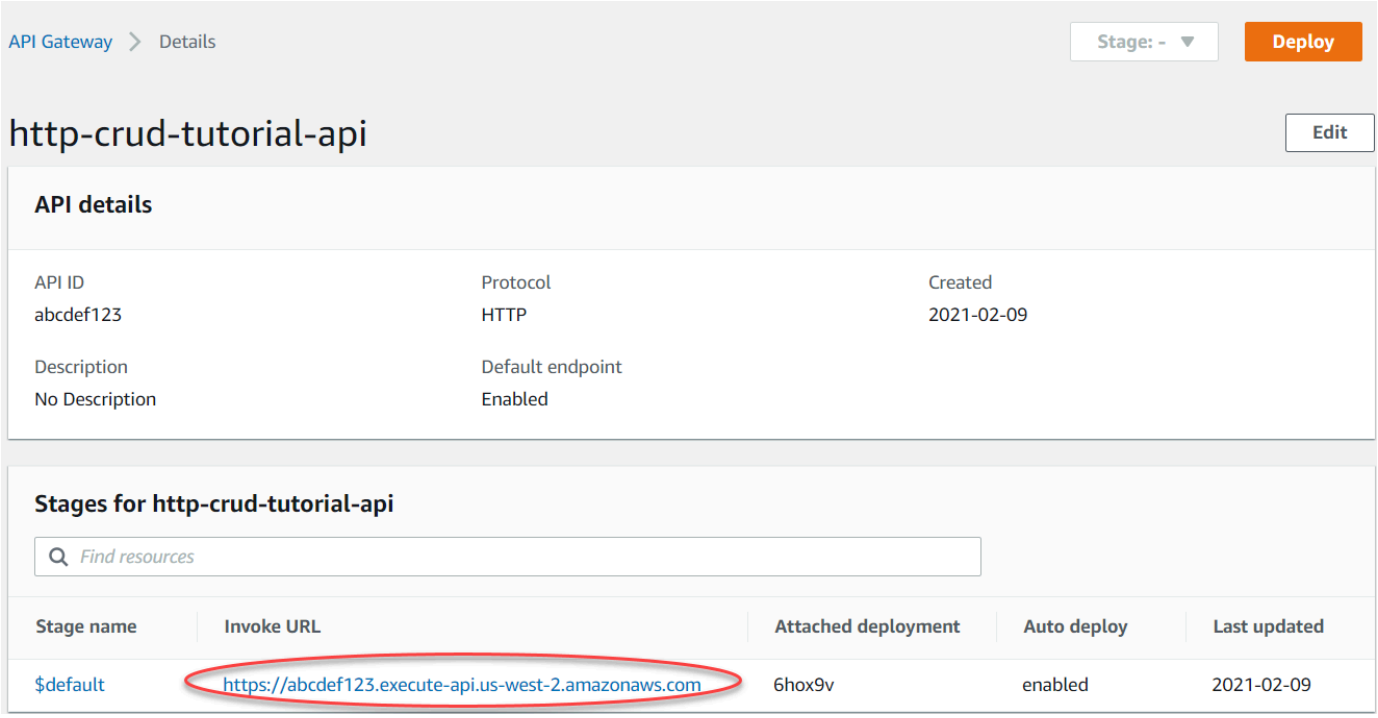
이제 HTTP API에 경로 및 통합이 연결되었으므로 API를 테스트할 수 있습니다.

7단계: API 테스트

API가 작동하는지 확인하려면 curl을 사용합니다.

API를 호출할 URL 가져오기

1. https://console.aws.amazon.com/apigateway에서 API Gateway 콘솔에 로그인합니다.
2. API를 선택합니다.
3. API의 호출 URL을 기록해 둡니다. 세부 정보 페이지의 [호출 URL(Invoke URL)] 아래에 나타납니다.



4. API의 호출 URL을 복사합니다.

   전체 URL은 `https://abcdef123.execute-api.us-west-2.amazonaws.com`처럼 보입니다.

항목 생성 또는 업데이트하기

- 다음 명령을 사용하여 항목을 생성하거나 업데이트합니다. 명령에는 항목의 ID, 가격 및 이름이 포함된 요청 본문이 포함됩니다.

```
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
```

모든 항목 가져오기

- 다음 명령을 사용하여 모든 항목을 나열합니다.

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

항목 가져오기

- 다음 명령을 사용하여 ID별로 항목을 가져옵니다.

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

항목 삭제 방법

1. 다음 명령을 사용하여 항목을 삭제합니다.

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

2. 항목이 삭제되었는지 확인하려면 모든 항목을 가져옵니다.

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

8단계: 정리

불필요한 비용을 방지하려면 이 시작하기 연습의 일부로 생성한 리소스를 삭제하세요. 다음 단계에서는 HTTP API, Lambda 함수 및 관련 리소스를 삭제합니다.

DynamoDB 테이블을 삭제하려면

1. https://console.aws.amazon.com/dynamodb/에서 DynamoDB 콘솔을 엽니다.
2. 테이블을 선택합니다.
3. [Delete Table]을 선택합니다.
4. 선택을 확인하고 [삭제(Delete)]를 선택합니다.

HTTP API 삭제하기

1. https://console.aws.amazon.com/apigateway에서 API Gateway 콘솔에 로그인합니다.
2. API 페이지에서 API를 선택합니다. [Actions]를 선택한 후 [Delete]를 선택합니다.
3. 삭제를 선택합니다.

Lambda 함수 삭제하기

1. https://console.aws.amazon.com/lambda에서 Lambda 콘솔에 로그인합니다.
2. 함수 페이지에서 함수를 선택합니다. [Actions]를 선택한 후 [Delete]를 선택합니다.
3. 삭제를 선택합니다.

Lambda 함수의 로그 그룹 삭제하기

1. Amazon CloudWatch 콘솔에서 로그 그룹 페이지를 엽니다.
2. 로그 그룹 페이지에서 함수의 로그 그룹(`/aws/lambda/http-crud-tutorial-function`)을 선택합니다. 작업을 선택한 다음 로그 그룹 삭제를 선택합니다.
3. 삭제를 선택합니다.

Lambda 함수의 실행 역할 삭제하기

1. AWS Identity and Access Management 콘솔에서 역할 페이지를 엽니다.
2. 함수의 역할(예: `http-crud-tutorial-role`)을 선택합니다.
3. 역할 삭제를 선택합니다.
4. 예, 삭제를 선택합니다.

다음 단계: AWS SAM 또는 AWS CloudFormation을 통한 자동화

AWS CloudFormation 또는 AWS SAM을 사용하여 AWS 리소스의 생성 및 정리를 자동화할 수 있습니다. 이 자습서의 AWS SAM 템플릿 예는 `template.yaml` 단원을 참조하세요.

예제 AWS CloudFormation 템플릿에 대해서는 예제 AWS CloudFormation 템플릿을 참조하세요.

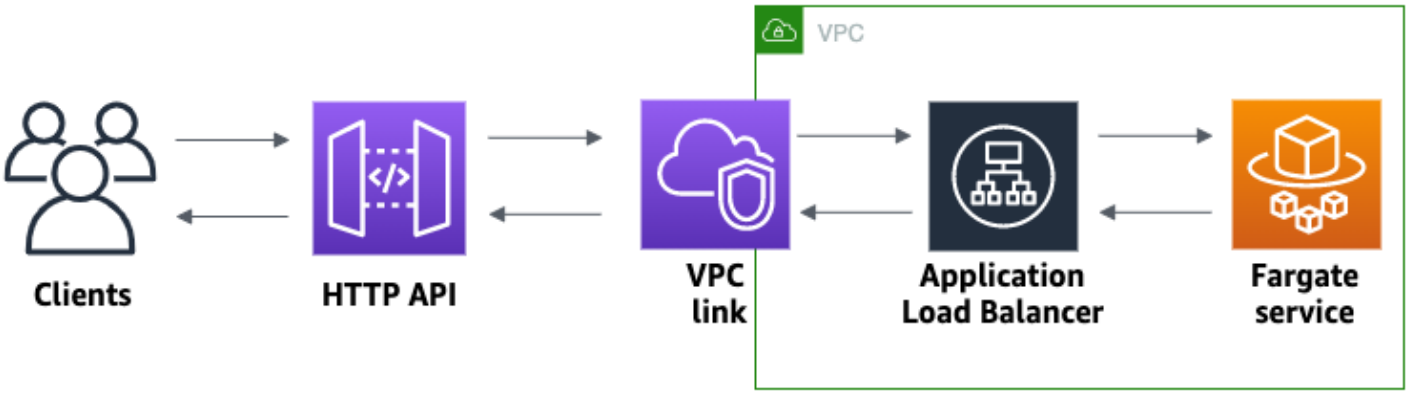
자습서: Amazon ECS 서비스에 대한 프라이빗 통합을 통해 HTTP API 구축

이 자습서에서는 Amazon VPC에서 실행되는 Amazon ECS 서비스에 연결하는 서버리스 API를 생성합니다. Amazon VPC 외부의 클라이언트는 API를 사용하여 Amazon ECS 서비스에 액세스할 수 있습니다.

이 자습서는 완료하는데 약 1시간이 걸립니다. 먼저 AWS CloudFormation 템플릿을 사용하여 Amazon VPC와 Amazon ECS 서비스를 생성합니다. 그런 다음 API Gateway 콘솔을 사용하여 VPC 링크를 생성합니다. VPC 링크를 통해 API Gateway는 Amazon VPC에서 실행되는 Amazon ECS 서비스에 액세

스할 수 있습니다. 그런 다음 VPC 링크를 사용하여 Amazon ECS 서비스에 연결하는 HTTP API를 생성합니다. 마지막으로 API를 테스트합니다.

HTTP API를 호출하면 API Gateway는 VPC 링크를 통해 요청을 Amazon ECS 서비스로 라우팅한 다음 서비스에서 응답을 반환합니다.



이 자습서를 완료하려면 AWS 계정과 콘솔 액세스 AWS Identity and Access Management 권한이 있는 사용자가 필요합니다. 자세한 설명은 사전 조건 섹션을 참조하세요.

이 자습서에서는 를 사용합니다 AWS Management Console이 API와 모든 관련 리소스를 생성하는 AWS CloudFormation 템플릿은 template.yaml을 참조하십시오.

주제

- 1단계: Amazon ECS 서비스 생성
- 2단계: VPC 링크 생성
- 3단계: HTTP API 생성
- 4단계: 경로 생성
- 5단계: 통합 생성
- 6단계: API 테스트
- 7단계: 정리
- 다음 단계: 다음을 사용하여 자동화하십시오. AWS CloudFormation

1단계: Amazon ECS 서비스 생성

Amazon ECS는 클러스터에서 Docker 컨테이너를 손쉽게 실행, 중지 및 관리할 수 있게 해 주는 컨테이너 관리 서비스입니다. 이 자습서에서는 Amazon ECS에서 관리하는 서버리스 인프라에서 클러스터를 실행합니다.

이 AWS CloudFormation 템플릿을 다운로드하고 압축을 풀면 Amazon VPC를 포함하여 서비스에 대한 모든 종속성이 생성됩니다. 이 템플릿을 사용하여 Application Load Balancer를 사용하는 Amazon ECS 서비스를 생성합니다.

스택을 생성하려면 AWS CloudFormation

1. https://console.aws.amazon.com/cloudformation 에서 AWS CloudFormation 콘솔을 엽니다.
2. 스택 생성을 선택한 다음 새 리소스 사용(표준)을 선택합니다.
3. 템플릿 지정에서 템플릿 파일 업로드를 선택합니다.
4. 다운로드한 템플릿을 선택합니다.
5. 다음을 선택합니다.
6. 스택 이름에 **http-api-private-integrations-tutorial**을(를) 입력하고 다음을 선택합니다.
7. 스택 옵션 구성에서 다음을 선택합니다.
8. [Capacabilities] 의 경우 계정에서 IAM 리소스를 생성할 AWS CloudFormation 수 있음을 확인하시기 바랍니다.
9. 스택 생성을 선택합니다.

AWS CloudFormation ECS 서비스를 프로비저닝합니다. 몇 분 정도 걸릴 수 있습니다. AWS CloudFormation 스택 상태가 CREATE_COMPLETE이면 다음 단계로 넘어갈 준비가 된 것입니다.

2단계: VPC 링크 생성

VPC 링크를 사용하면 API Gateway가 Amazon VPC의 프라이빗 리소스에 액세스할 수 있습니다. VPC 링크를 사용하면 클라이언트가 HTTP API를 통해 Amazon ECS 서비스에 액세스할 수 있습니다.

VPC 링크 생성

1. https://console.aws.amazon.com/apigateway에서 API Gateway 콘솔에 로그인합니다.
2. 기본 탐색 창에서 VPC 링크를 선택한 다음 [Create] 를 선택합니다.

   기본 탐색 창을 열려면 메뉴 아이콘을 선택해야 할 수 있습니다.
3. VPC 링크 버전 선택(Choose a VPC link version)에서 HTTP API용 VPC 링크(VPC link for HTTP APIs)를 선택합니다.
4. 이름에 **private-integrations-tutorial**를 입력합니다.

5. VPC에서 1단계에서 생성한 VPC를 선택합니다. 이름은 로 시작해야 PrivateIntegrationsStack합니다.
6. 서브넷(Subnets)의 경우 VPC에 있는 두 개의 프라이빗 서브넷을 선택합니다. 이름은 `PrivateSubnet`(으)롤 끝납니다.
7. Create를 선택합니다.

VPC 링크를 생성한 후 API Gateway는 탄력적 네트워크 인터페이스를 프로비저닝하여 VPC에 액세스합니다. 이 프로세스는 몇 분 정도 걸릴 수 있습니다. 그동안 API를 생성할 수 있습니다.

3단계: HTTP API 생성

HTTP API는 Amazon ECS 서비스에 대한 HTTP 엔드포인트를 제공합니다. 이 단계에서는 빈 API를 생성합니다. 4단계와 5단계에서는 API와 Amazon ECS 서비스를 연결하기 위한 경로 및 통합을 구성합니다.

HTTP API 생성하기

1. https://console.aws.amazon.com/apigateway에서 API Gateway 콘솔에 로그인합니다.
2. [API 생성(Create API)]을 선택한 다음 HTTP API에 대해 [빌드(Build)]를 선택합니다.
3. API 이름에서 **`http-private-integrations-tutorial`**을(를) 입력합니다.
4. [Next]를 선택합니다.
5. 경로 구성에 대해 [다음(Next)]을 선택하여 경로 생성을 건너뜁니다. 나중에 루트를 생성합니다.
6. API Gateway가 생성하는 스테이지를 검토합니다. API Gateway는 자동 배포가 활성화된 `$default` 스테이지를 생성합니다. 이 스테이지는 이 자습서에 가장 적합합니다. [Next]를 선택합니다.
7. Create를 선택합니다.

4단계: 경로 생성

경로는 수신 API 요청을 백엔드 리소스로 보내는 방법입니다. 경로는 HTTP 메서드와 리소스 경로(예: `GET /items`), 이렇게 두 부분으로 구성됩니다. 이 예제 API의 경우 1개의 경로를 생성합니다.

경로 생성

1. https://console.aws.amazon.com/apigateway에서 API Gateway 콘솔에 로그인합니다.

2. API를 선택합니다.
3. Routes(라우팅)를 선택합니다.
4. Create를 선택합니다.
5. 메서드(Method)에서 **ANY**을(를) 선택합니다.
6. 경로에 **/{proxy+}**을(를) 입력합니다. 경로 끝의 {proxy+}은(는) 복잡한 경로 변수입니다. API Gateway는 API에 대한 모든 요청을 이 경로로 보냅니다.
7. Create를 선택합니다.

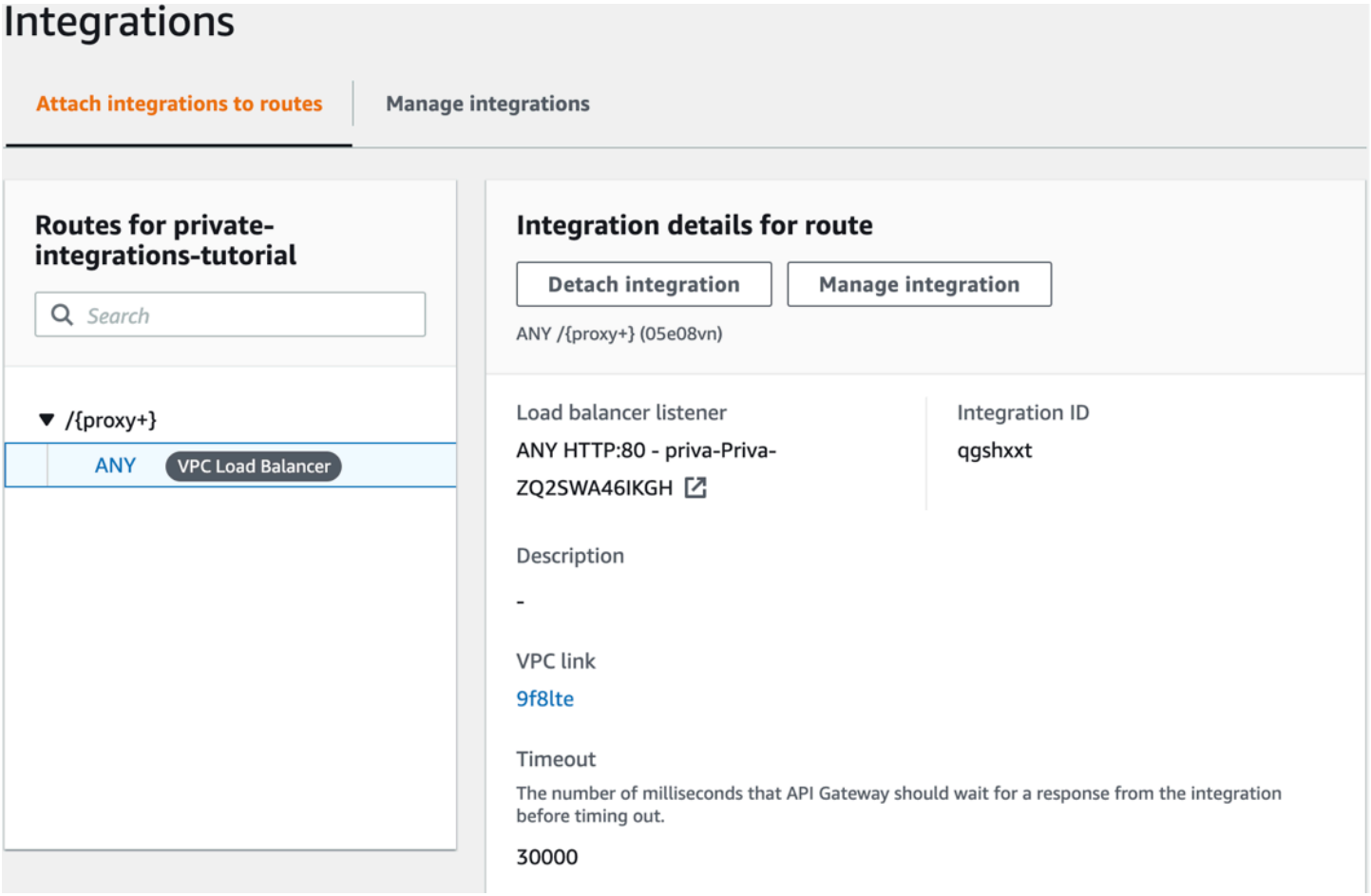
5단계: 통합 생성

백엔드 리소스에 경로를 연결하는 통합을 생성합니다.

통합 생성하기

1. https://console.aws.amazon.com/apigateway에서 API Gateway 콘솔에 로그인합니다.
2. API를 선택합니다.
3. [통합(Integrations)]을 선택합니다.
4. [통합 관리(Manage integrations)]를 선택한 다음 [생성(Create)]을 선택합니다.
5. 이 통합을 경로에 연결(Attach this integration to a route)에서 이전에 생성한 ANY /{proxy+} 경로를 선택합니다.
6. 통합 유형(Integration type)에서 프라이빗 리소스(Private resource)를 선택합니다.
7. 통합 세부 정보(Integration details)에서 수동 선택(Select manually)을 선택합니다.
8. 대상 서비스(Target service)에서 ALB/NLB를 선택합니다.
9. 로드 밸런서(Load balancer)의 경우 1단계에서 AWS CloudFormation 템플릿으로 생성한 로드 밸런서를 선택합니다. 이름은 HTTP-Priva로 시작해야 합니다.
10. 리스너(Listener)에서 **HTTP 80**을(를) 선택합니다.
11. VPC 링크(VPC link)의 경우 2단계에서 생성한 VPC 링크를 선택합니다. 이름은 `private-integrations-tutorial`이어야 합니다.
12. Create를 선택합니다.

경로 및 통합이 올바르게 설정되었는지 확인하려면 경로에 통합 연결(Attach integrations to routes)을 선택합니다. 콘솔에 VPC Load Balancer에 대한 통합의 `ANY /{proxy+}` 경로가 있음이 표시됩니다.



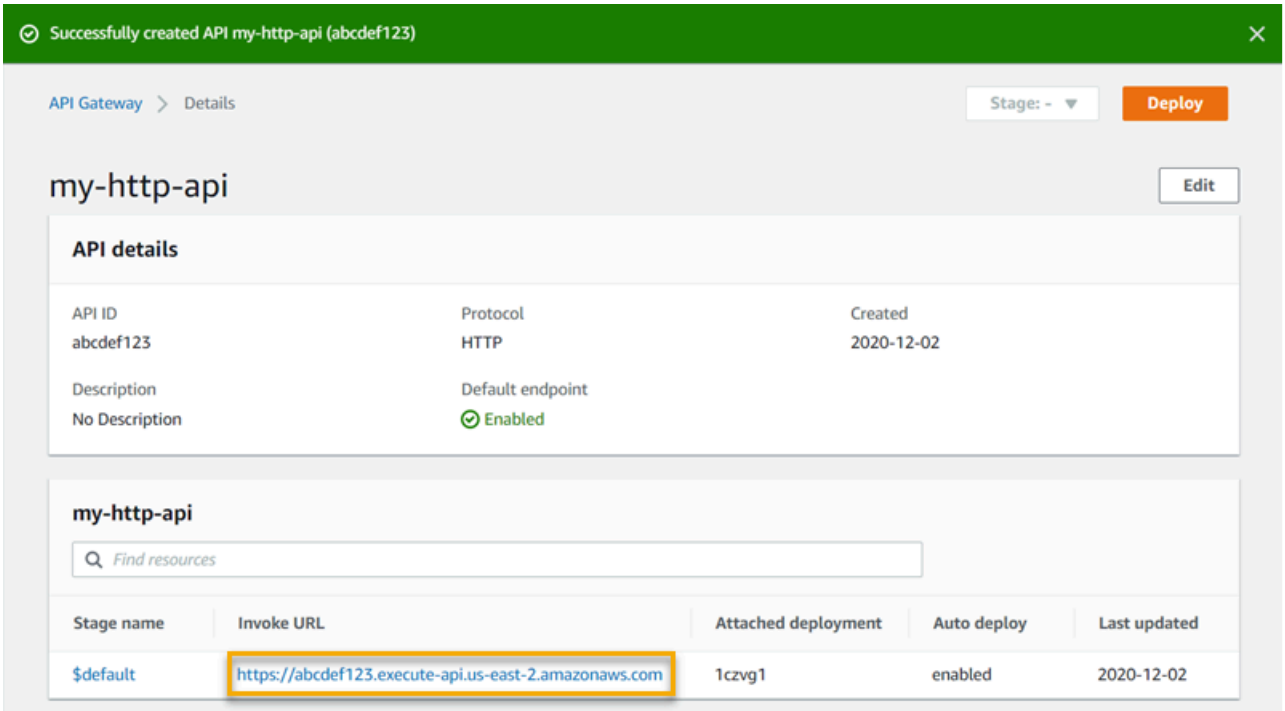
이제 API를 테스트 할 준비가 되었습니다.

6단계: API 테스트

다음으로, API를 테스트하여 제대로 작동하는지 확인합니다. 간단하게 하기 위해, 웹 브라우저를 사용하여 API를 호출합니다.

API 테스트하기

1. https://console.aws.amazon.com/apigateway에서 API Gateway 콘솔에 로그인합니다.
2. API를 선택합니다.
3. API의 호출 URL을 기록해 둡니다.



4. 웹 브라우저에서 API의 호출 URL로 이동합니다.

   전체 URL은 `https://abcdef123.execute-api.us-east-2.amazonaws.com`와(과) 같아야 합니다.

   브라우저가 API에 GET 요청을 보냅니다.

5. API의 응답이 앱이 Amazon ECS에서 실행 중임을 알려주는 시작 메시지인지 확인합니다.

   시작 메시지가 표시되면 Amazon VPC에서 실행되는 Amazon ECS 서비스를 성공적으로 생성하고, VPC 링크와 함께 API Gateway HTTP API를 사용하여 Amazon ECS 서비스에 액세스한 것입니다.

7단계: 정리

불필요한 비용을 방지하려면 이 자습서의 일부로 생성한 리소스를 삭제합니다. 다음 단계는 VPC 링크, AWS CloudFormation 스택 및 HTTP API를 삭제합니다.

HTTP API 삭제하기

1. https://console.aws.amazon.com/apigateway에서 API Gateway 콘솔에 로그인합니다.
2. API 페이지에서 API를 선택합니다. 작업(Actions)을 선택하고 삭제(Delete)를 선택한 다음 선택을 확인합니다.

VPC 링크 삭제

1. https://console.aws.amazon.com/apigateway에서 API Gateway 콘솔에 로그인합니다.
2. VPC 링크(VPC link)를 선택합니다.
3. VPC 링크를 선택하고 삭제(Delete)를 선택한 다음, 해당 선택을 확인합니다.

스택을 삭제하려면 AWS CloudFormation

1. https://console.aws.amazon.com/cloudformation 에서 AWS CloudFormation 콘솔을 엽니다.
2. AWS CloudFormation 스택을 선택합니다.
3. 삭제를 선택한 다음 해당 선택을 확인합니다.

다음 단계: 다음을 사용하여 자동화하십시오. AWS CloudFormation

이 자습서와 관련된 모든 AWS 리소스의 생성 및 정리를 자동화할 수 있습니다. 전체 예제 AWS CloudFormation 템플릿에 대해서는 template.yaml을 참조하세요.

Amazon API Gateway WebSocket API 자습서

다음 자습서에는 API Gateway WebSocket API에 대해 알아보는 데 도움이 되는 실습용 과제가 나와 있습니다.

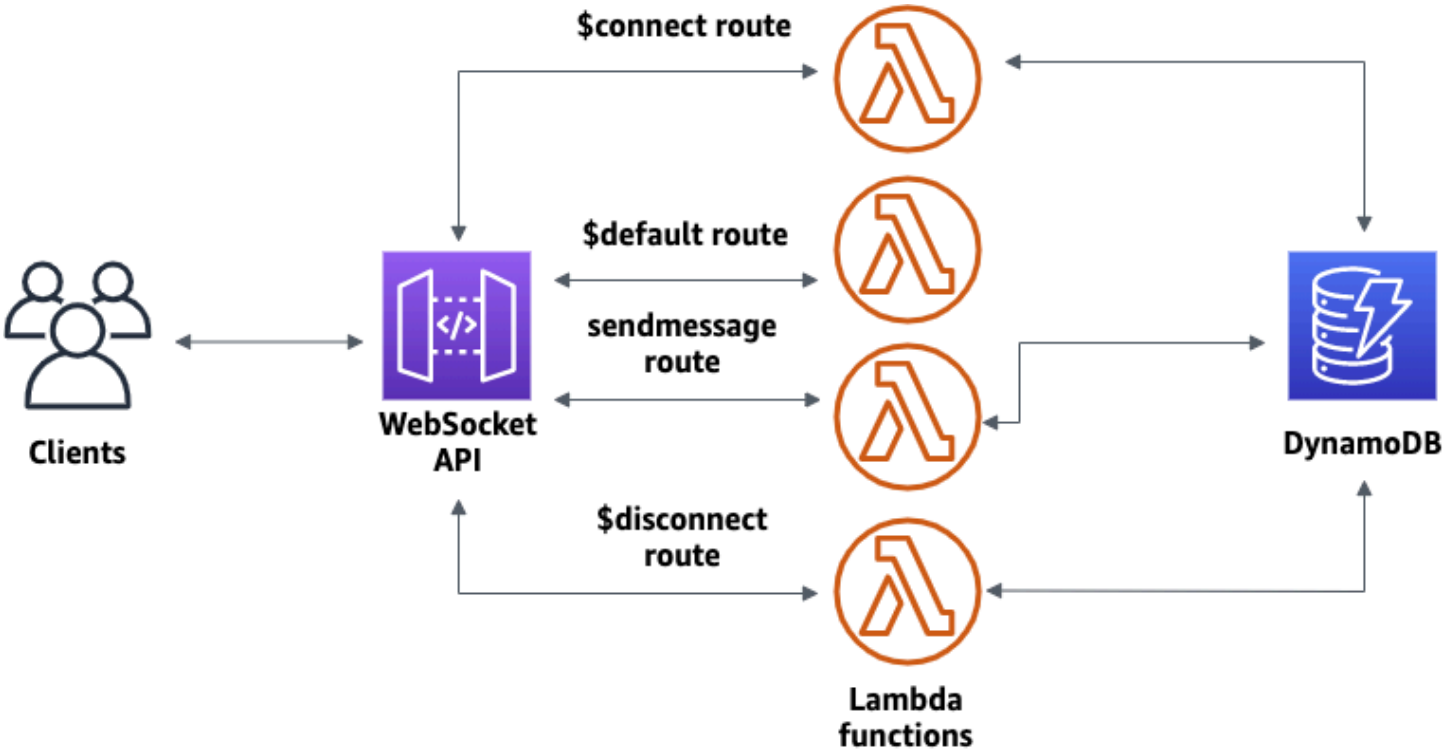
주제

- 자습서: WebSocket API, Lambda 및 DynamoDB를 사용하여 서버리스 채팅 앱 구축

자습서: WebSocket API, Lambda 및 DynamoDB를 사용하여 서버리스 채팅 앱 구축

이 자습서에서는 API를 사용하여 서버리스 채팅 애플리케이션을 만들어 보겠습니다. WebSocket WebSocket API를 사용하면 클라이언트 간 양방향 통신을 지원할 수 있습니다. 클라이언트는 업데이트를 폴링하지 않고 메시지를 수신할 수 있습니다.

이 자습서는 완료되는데 약 30분이 걸립니다. 먼저 AWS CloudFormation 템플릿을 사용하여 API 요청을 처리하는 Lambda 함수와 클라이언트 ID를 저장하는 DynamoDB 테이블을 생성합니다. 그런 다음 API Gateway 콘솔을 사용하여 Lambda 함수와 통합되는 WebSocket API를 생성합니다. 마지막으로 API를 테스트하여 메시지가 송신 및 수신되는지 확인합니다.



이 자습서를 완료하려면 콘솔 액세스 권한이 있는 AWS 계정과 AWS Identity and Access Management 사용자가 필요합니다. 자세한 설명은 사전 조건 섹션을 참조하세요.

또한 API에 연결하기 위해 `wscat`가 필요합니다. 자세한 설명은 the section called “`wscat`를 사용하여 WebSocket API에 연결하고 메시지 보내기” 섹션을 참조하세요.

주제

- 1단계: Lambda 함수 및 DynamoDB 테이블 생성
- 2단계: API 생성 WebSocket
- 3단계: API 테스트
- 4단계: 정리
- 다음 단계: 다음을 사용하여 자동화하십시오. AWS CloudFormation

1단계: Lambda 함수 및 DynamoDB 테이블 생성

에 대한 AWS CloudFormation앱 생성 템플릿을 다운로드하고 압축을 풉니다. 이 템플릿을 사용하여 앱의 클라이언트 ID를 저장할 Amazon DynamoDB 테이블을 생성합니다. 연결된 각 클라이언트에는 테이블의 파티션 키로 사용될 고유 ID가 있습니다. 또한 이 템플릿은 DynamoDB에서 클라이언트 연결을 업데이트하고 연결된 클라이언트로의 메시지 전송을 처리하는 Lambda 함수를 생성합니다.

스택을 AWS CloudFormation 만들려면

1. https://console.aws.amazon.com/cloudformation 에서 AWS CloudFormation 콘솔을 엽니다.
2. 스택 생성을 선택한 다음 새 리소스 사용(표준)을 선택합니다.
3. 템플릿 지정에서 템플릿 파일 업로드를 선택합니다.
4. 다운로드한 템플릿을 선택합니다.
5. 다음을 선택합니다.
6. 스택 이름에 **websocket-api-chat-app-tutorial**을(를) 입력하고 다음을 선택합니다.
7. 스택 옵션 구성에서 다음을 선택합니다.
8. [Capacabilities] 의 경우 계정에서 IAM 리소스를 생성할 AWS CloudFormation 수 있다는 점을 확인하십시오.
9. 스택 생성을 선택합니다.

AWS CloudFormation 템플릿에 지정된 리소스를 프로비저닝합니다. 리소스 프로비저닝을 완료하는 데 몇 분 정도 걸릴 수 있습니다. AWS CloudFormation 스택 상태가 CREATE_COMPLETE이면 다음 단계로 넘어갈 준비가 된 것입니다.

2단계: API 생성 WebSocket

클라이언트 연결을 처리하고 1단계에서 생성한 Lambda 함수로 요청을 라우팅하는 WebSocket API를 생성합니다.

API를 생성하려면 WebSocket

1. https://console.aws.amazon.com/apigateway에서 API Gateway 콘솔에 로그인합니다.
2. API 생성을 선택합니다. 그런 다음 WebSocket API의 경우 빌드를 선택합니다.
3. API 이름에서 **websocket-chat-app-tutorial**을(를) 입력합니다.
4. 경로 선택 표현식(Route selection expression)에 **request.body.action**을 입력합니다. 경로 선택 표현식에 따라 클라이언트가 메시지를 보낼 때 API Gateway에서 호출할 경로가 결정됩니다.
5. 다음을 선택합니다.
6. 사전 정의된 경로(Predefined routes)에서 $connect 추가(Add $connect), $disconnect 추가(Add $disconnect), $default 추가(Add $default)를 선택합니다. $connect 및 $disconnect 경로는 클라이언트가 API에 연결하거나 API에서 연결을 끊을 때 API Gateway가 자동으로 호출하는 특수 경로입니다. API Gateway는 요청과 일치하는 다른 경로가 없을 때 `$default` 경로를 호출합니다.

7. 사용자 지정 경로(Custom routes)에서 사용자 지정 경로 추가(Add custom routes)를 선택합니다. 경로 키(Route key)에 **sendmessage**를 입력합니다. 이 사용자 지정 경로는 연결된 클라이언트로 전송되는 메시지를 처리합니다.
8. 다음을 선택합니다.
9. 각 경로 및 통합 유형(Integration type)에 대한 통합 연결(Attach integrations)에서 Lambda를 선택합니다.

   Lambda의 경우, 1단계에서 생성한 Lambda 함수를 선택하십시오. AWS CloudFormation 각 함수의 이름은 경로와 일치합니다. 예를 들어 $connect 경로에서 이름이 **websocket-chat-app-tutorial-ConnectHandler**인 함수를 선택합니다.
10. API Gateway가 생성하는 스테이지를 검토합니다. 기본적으로 API Gateway는 스테이지 이름 `production`을 생성하여 API를 해당 스테이지에 자동으로 배포합니다. 다음을 선택합니다.
11. 생성 및 배포(Create and deploy)를 선택합니다.

3단계: API 테스트

다음으로, API를 테스트하여 제대로 작동하는지 확인합니다. `wscat` 명령을 사용하여 API에 연결합니다.

API의 호출 URL을 가져오려면

1. https://console.aws.amazon.com/apigateway에서 API Gateway 콘솔에 로그인합니다.
2. API를 선택합니다.
3. 스테이지(Stages)를 선택한 다음 프로덕션(production)을 선택합니다.
4. API의 URL을 기록해 두십시오. WebSocket URL은 `wss://`*`abcdef123`*`.execute-api.`*`us-east-2`*`.amazonaws.com/production`와 같아야 합니다.

API에 연결하려면

1. API에 연결하려면 다음 명령을 사용합니다. API에 연결하면 API Gateway에서 `$connect` 경로를 호출합니다. 이 경로가 호출되면 API Gateway가 DynamoDB에 연결 ID를 저장하는 Lambda 함수를 호출합니다.

   ```
   wscat -c wss://abcdef123.execute-api.us-west-2.amazonaws.com/production
   ```

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

2. 새 터미널을 열고 다음 파라미터를 사용하여 wscat 명령을 다시 실행합니다.

```
wscat -c wss://abcdef123.execute-api.us-west-2.amazonaws.com/production
```

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

이렇게 하면 메시지를 교환할 수 있는 연결된 두 클라이언트가 제공됩니다.

메시지를 전송하려면

- API Gateway는 API의 경로 선택 표현식을 기반으로 호출할 경로를 결정합니다. API의 경로 선택 표현식은 `$request.body.action`입니다. 따라서 API Gateway는 다음 메시지가 전송될 때 `sendmessage` 경로를 호출합니다.

```
{"action": "sendmessage", "message": "hello, everyone!"}
```

호출된 경로와 연결된 Lambda 함수는 DynamoDB에서 클라이언트 ID를 수집합니다. 그런 다음 이 함수는 API Gateway Management API를 호출하고 해당 클라이언트에 메시지를 보냅니다. 연결된 모든 클라이언트는 다음과 같은 메시지를 받습니다.

```
< hello, everyone!
```

API의 $default 경로를 호출하려면

- API Gateway는 클라이언트가 정의된 경로와 일치하지 않는 메시지를 전송하는 경우 API의 기본 경로를 호출합니다. `$default` 경로와 연결된 Lambda 함수는 API Gateway Management API를 사용하여 연결에 대한 클라이언트 정보를 보냅니다.

```
test
```

```
Use the sendmessage route to send a message. Your info:
 {"ConnectedAt":"2022-01-25T18:50:04.673Z","Identity":
{"SourceIp":"192.0.2.1","UserAgent":null},"LastActiveAt":"2022-01-25T18:50:07.642Z","connec
```

API 연결을 해제하려면

- API 연결을 해제하려면 **CTRL+C**를 누릅니다. 클라이언트가 API에서 연결을 해제하면 API Gateway에서 API의 `$disconnect` 경로를 호출합니다. API의 `$disconnect` 경로에 대한 Lambda 통합이 DynamoDB에서 연결 ID를 제거합니다.

4단계: 정리

불필요한 비용을 방지하려면 이 자습서의 일부로 생성한 리소스를 삭제합니다. 다음 단계는 AWS CloudFormation 스택과 WebSocket API를 삭제합니다.

WebSocket API를 삭제하려면

1. https://console.aws.amazon.com/apigateway에서 API Gateway 콘솔에 로그인합니다.
2. API(APIs) 페이지에서 `websocket-chat-app-tutorial` API를 선택합니다. 작업(Actions)을 선택하고 삭제(Delete)를 선택한 다음 선택을 확인합니다.

AWS CloudFormation 스택을 삭제하려면

1. https://console.aws.amazon.com/cloudformation 에서 AWS CloudFormation 콘솔을 엽니다.
2. AWS CloudFormation 스택을 선택합니다.
3. 삭제를 선택한 다음 해당 선택을 확인합니다.

다음 단계: 다음을 사용하여 자동화하십시오. AWS CloudFormation

이 자습서와 관련된 모든 AWS 리소스의 생성 및 정리를 자동화할 수 있습니다. 이 API와 모든 관련 리소스를 생성하는 AWS CloudFormation 템플릿은 chat-app.yaml을 참조하십시오.

REST API 작업

API Gateway의 REST API는 백엔드 HTTP 엔드포인트, Lambda 함수 또는 기타 AWS 서비스와 통합되어 있는 리소스 및 메서드의 모음입니다. API Gateway 기능을 사용하여 생성에서 프로덕션 API 모니터링에 이르기까지 API 수명 주기의 모든 측면을 지원할 수 있습니다.

API Gateway REST API는 클라이언트가 서비스에 요청을 보내고 서비스가 동기식으로 응답하는 요청/응답 모델을 사용합니다. 이러한 종류의 모델은 동기식 통신을 사용하는 다양한 종류의 애플리케이션에 적합합니다.

주제

- API Gateway에서 REST API 개발
- 고객이 호출할 수 있도록 REST API 게시
- REST API의 성능 최적화
- 클라이언트에 REST API 배포
- REST API 보호
- REST API 모니터링

API Gateway에서 REST API 개발

이 단원에서는 API Gateway API를 개발하는 동안 필요한 API Gateway 기능에 대해 자세히 설명합니다.

API Gateway API를 개발할 때 API의 여러 특성을 결정합니다. 이러한 특성은 API의 사용 사례에 따라 달라집니다. 예를 들어, 특정 클라이언트만 API를 호출하도록 허용하거나 모든 사용자가 API를 사용할 수 있도록 할 수 있습니다. API 호출로 Lambda 함수를 실행하거나, 데이터베이스 쿼리를 작성하거나, 애플리케이션을 호출할 수 있습니다.

주제

- Amazon API Gateway에서 REST API 생성
- API Gateway의 REST API에 대한 액세스 제어 및 관리
- REST API 통합 설정
- API Gateway에서 요청 확인 사용
- REST API에 대한 데이터 변환 설정

- API Gateway의 게이트웨이 응답
- REST API 리소스에 대한 CORS 활성화
- REST API에 대한 이진 미디어 유형 작업
- Amazon API Gateway에서 REST API 호출
- OpenAPI를 사용하여 REST API 구성

Amazon API Gateway에서 REST API 생성

Amazon API Gateway에서 Amazon API Gateway 리소스로 알려진 프로그래밍 가능 엔터티 모음으로 REST API를 구축합니다. 예를 들어 `RestApi` 리소스를 사용하여 `Resource` 엔터티 모음을 포함할 수 있는 API를 나타냅니다. 각 `Resource` 엔터티마다 `Method` 리소스가 하나 이상 있을 수 있습니다. 요청 파라미터 및 본문에 표시된 `Method`는 클라이언트가 노출된 `Resource`에 액세스할 수 있도록 애플리케이션 프로그래밍 인터페이스를 정의하고 클라이언트가 제출하는 수신 요청을 나타냅니다. 이후 `Integration` 리소스를 생성하여 수신 요청을 특정 통합 엔드포인트 URI로 전송함으로써 `Method`를 통합 엔드포인트라고도 하는 백엔드 엔드포인트에 통합합니다. 필요한 경우 요청 파라미터 또는 본문을 변환하여 백엔드 요구 사항을 충족합니다. 응답에 대해 `MethodResponse` 리소스를 생성하여 클라이언트가 수신하는 요청 응답을 표시하고 `IntegrationResponse` 리소스를 생성하여 백엔드에서 반환되는 요청 응답을 표시할 수 있습니다. 통합 응답을 구성함으로써 클라이언트에게 데이터를 반환하기 전에 백엔드 응답 데이터를 변환하거나 클라이언트에게 백엔드 응답을 있는 그대로 전달할 수 있습니다.

고객이 API를 이해할 수 있도록 지원하기 위해 API 생성 중에 또는 API가 생성된 후에 API 설명서를 제공할 수 있습니다. 이를 활성화하려면 지원되는 API 엔터티에 대한 `DocumentationPart` 리소스를 추가하세요.

클라이언트가 API를 호출하는 방법을 제어하려면 IAM 권한, Lambda 권한 부여자, 또는 Amazon Cognito 사용자 풀을 사용합니다. API 사용을 측정하려면 API 요청을 조절하도록 사용량 계획을 설정하세요. API를 생성 또는 업데이트할 때 이를 활성화할 수 있습니다.

API Gateway 콘솔, API Gateway REST API AWS CLI, 또는 AWS SDK 중 하나를 사용하여 이러한 작업 및 기타 작업을 수행할 수 있습니다. 다음으로 이 작업을 수행하는 방법에 대해 알아보겠습니다.

주제

- API Gateway API에 대해 설정할 엔드포인트 유형 선택
- API Gateway에서 REST API 설정 초기화
- API Gateway에서 REST API 메서드 설정

API Gateway API에 대해 설정할 엔드포인트 유형 선택

API 엔드포인트 유형은 API의 호스트 이름을 나타냅니다. API 엔드포인트 유형은 대다수 API 트래픽의 출처에 따라 엣지 최적화, 리전 또는 프라이빗 엔드포인트일 수 있습니다.

엣지 최적화 API 엔드포인트

엣지 최적화 API 엔드포인트는 일반적으로 요청을 가장 가까운 POP (CloudFront Point of Presence)로 라우팅하므로 클라이언트가 지리적으로 분산되어 있는 경우에 도움이 될 수 있습니다. 이것은 API Gateway REST API의 기본 엔드포인트 유형입니다.

엣지 최적화 API는 HTTP 헤더의 이름을 대문자로 처리합니다(예: `Cookie`).

CloudFront 요청을 오리진에 전달하기 전에 HTTP 쿠키를 쿠키 이름을 기준으로 자연스러운 순서로 정렬합니다. 쿠키를 CloudFront 처리하는 방법에 대한 자세한 내용은 쿠키 기반 콘텐츠 캐싱을 참조하십시오.

엣지 최적화 API를 위해 사용되는 사용자 지정 도메인 이름은 모든 리전에 적용됩니다.

리전 API 엔드포인트

리전 API 엔드포인트는 동일 리전의 클라이언트를 위한 것입니다. EC2 인스턴스를 실행하는 클라이언트가 동일 리전에서 API를 호출하거나 API가 수요가 큰 소수의 클라이언트에게 서비스를 제공하기 위한 것이라면 리전 API는 연결 오버헤드를 줄입니다.

리전 API에 대해 API가 배포된 리전별로 고유한 사용자 지정 도메인 이름이 사용됩니다. 여러 리전에 배포된 리전 API를 배포하는 경우, 모든 리전에서 동일한 사용자 지정 도메인 이름을 사용할 수 있습니다. Amazon Route 53과 함께 사용자 지정 도메인을 사용하여 지연 시간 기반 라우팅 등의 작업을 수행할 수 있습니다. 자세한 내용은 the section called “리전 사용자 지정 도메인 이름 설정” 및 the section called “에지 최적화 사용자 지정 도메인 이름 생성” 단원을 참조하세요.

리전 API 엔드포인트는 모든 헤더 이름을 그대로 전달합니다.

프라이빗 API 엔드포인트

프라이빗 API 엔드포인트는 인터페이스 VPC 종단점을 사용해서 Amazon Virtual Private Cloud(VPC)에서만 액세스할 수 있는 API 엔드포인트입니다. 이때 이 엔드포인트는 사용자가 VPC에서 만든 엔드포인트 네트워크 인터페이스(ENI)입니다. 자세한 내용은 the section called “프라이빗 API” 단원을 참조하세요.

프라이빗 API 엔드포인트는 모든 헤더 이름을 그대로 전달합니다.

API Gateway에서 퍼블릭 또는 프라이빗 API 엔드포인트 유형 변경

API 엔드포인트 유형을 변경하려면 API의 구성을 업데이트해야 합니다. API Gateway 콘솔, AWS CLI 또는 API Gateway용 AWS SDK를 사용하여 기존 API 유형을 변경할 수 있습니다. 이 시간 동안 API를 사용할 수 있지만 현재 변경이 완료될 때까지 엔드포인트 유형을 다시 변경할 수는 없습니다.

다음과 같은 엔드포인트 유형 변경이 지원됩니다.

- 엣지 최적화에서 리전 또는 프라이빗으로 변경
- 리전에서 엣지 최적화 또는 프라이빗으로 변경
- 프라이빗에서 리전으로 변경

프라이빗 API를 엣지 최적화된 API로 바꿀 수는 없습니다.

퍼블릭 API의 유형을 엣지 최적화에서 리전으로, 또는 리전에서 엣지 최적화로 변경할 때는 엣지 최적화된 API가 리전 API와 다르게 동작할 수 있다는 점에 유의해야 합니다. 예를 들어 엣지 최적화 API는 `Content-MD5` 헤더를 제거합니다. 백엔드로 전달되는 모든 MD5 해시 값은 요청 문자열 파라미터 또는 본문 속성으로 표현될 수 있습니다. 그러나 리전 API는 헤더 이름을 다른 이름으로 다시 매핑하기는 하지만 이 헤더를 빠져나갑니다. 그 차이를 이해하면 엣지 최적화 API를 리전 API로 또는 리전 API를 엣지 최적화 API로 업데이트하는 방법을 결정하는 데 도움이 됩니다.

주제

- API Gateway 콘솔을 사용하여 API 엔드포인트 유형 변경
- AWS CLI를 사용하여 API 엔드포인트 유형 변경

API Gateway 콘솔을 사용하여 API 엔드포인트 유형 변경

API의 API 엔드포인트 유형을 변경하려면 다음 단계 세트 중 하나를 수행하세요.

지역에서 엣지 최적화로 또는 그 반대로 퍼블릭 엔드포인트를 전환하려면

1. https://console.aws.amazon.com/apigateway에서 API Gateway 콘솔에 로그인합니다.
2. REST API를 선택합니다.
3. API 설정을 선택합니다.
4. API 세부 정보 섹션에서 편집을 선택합니다.
5. API 엔드포인트 유형으로 엣지 최적화 또는 지역을 선택합니다.
6. 변경 사항 저장을 선택합니다.

7. API를 다시 배포하여 변경 사항을 적용합니다.

프라이빗 엔드포인트를 리전 엔드포인트로 전환하려면

1. https://console.aws.amazon.com/apigateway에서 API Gateway 콘솔에 로그인합니다.
2. REST API를 선택합니다.
3. API의 리소스 정책에서 VPC 또는 VPC 엔드포인트에 대한 언급을 삭제하여 VPC 내부는 물론 VPC 외부에서 오는 API 호출도 성공하도록 합니다.
4. API 설정을 선택합니다.
5. API 세부 정보 섹션에서 편집을 선택합니다.
6. API 엔드포인트 유형에서 지역을 선택합니다.
7. 변경 사항 저장을 선택하여 업데이트를 시작합니다.
8. API에서 리소스 정책을 제거합니다.
9. API를 다시 배포하여 변경 사항을 적용합니다.

AWS CLI를 사용하여 API 엔드포인트 유형 변경

AWS CLI을 사용하여 API ID가 *{api-id}*인 엣지 최적화 API를 업데이트하려면 다음과 같이 update-rest-api를 호출합니다.

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

성공적인 응답에는 다음과 같이 `200 OK` 상태 코드와 페이로드가 있습니다.

```
{

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

반대로 다음과 같이 리전 API를 엣지 최적화 API로 업데이트합니다.

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

put-rest-api는 API 정의를 업데이트하기 위한 것이므로 API 엔드포인트 유형을 업데이트하는 데는 적용할 수 없습니다.

API Gateway에서 REST API 설정 초기화

API 게이트웨이 콘솔, API 게이트웨이 REST API, API 게이트웨이용 AWS SDK 등을 사용하여 REST API를 생성할 수 있습니다. AWS Command Line Interface

API Gateway REST API, API Gateway용 AWS SDK 또는 를 사용하여 REST API를 생성하는 경우 기본 구성은 엣지에 최적화된 API입니다. AWS Command Line Interface API 엔드포인트 유형에 대한 자세한 내용은 을 참조하십시오. the section called “API 엔드포인트 유형 선택”

API를 스테이지에 배포하면 API Gateway에서 API의 기본 URL을 생성합니다. 기본 URL에 대한 자세한 내용은 을 참조하십시오the section called “REST API 배포”. 사용자 지정 도메인 이름(예: `apis.example.com`)을 API의 호스트 이름으로 할당하고 `https://apis.example.com/myApi` 형식의 기본 URL로 API를 호출할 수 있습니다. 사용자 지정 도메인 이름에 대한 자세한 내용은 을 참조하십시오the section called “사용자 지정 도메인 이름”.

다음 예제 중 하나를 사용하여 REST API를 만드는 방법을 익히는 것이 좋습니다.

주제

- API Gateway 콘솔을 사용하여 API 설정
- AWS CLI 명령을 사용하여 엣지 최적화 API 설정
- Node.js용 AWS SDK를 사용한 엣지 최적화 API 설정
- OpenAPI 정의를 가져와 엣지 최적화 API 설정
- API Gateway에서 리전 API 설정

API Gateway 콘솔을 사용하여 API 설정

REST API Gateway 콘솔을 사용하여 REST API Gateway를 생성하는 방법을 배우려면 다음 자습서 중에서 선택하는 것이 좋습니다.

Lambda 함수에 이벤트를 전달하는 REST API를 생성하려면 을 선택하십시오. the section called “REST API 콘솔 시작하기”

Lambda 함수에 대한 통합 요청 페이로드를 구성하는 REST API를 생성하려면 을 선택하십시오. the section called “자습서: Lambda 비 프록시 통합을 사용하여 API 빌드”

HTTP 엔드포인트와 통합되는 REST API를 생성하려면 을 선택하십시오. the section called “자습서: HTTP 프록시 통합을 사용하여 REST API 구축”

HTTP 엔드포인트에 대한 통합 요청을 구성하는 REST API를 생성하려면 을 선택합니다the section called “자습서: HTTP 비 프록시 통합을 사용하여 API 빌드”.

예제 API를 가져오려면 을 선택합니다the section called “자습서: 예제를 가져와 REST API 생성”.

또는 외부 API 정의(예: OpenAPI에 대한 API Gateway 확장 작업이 포함된 OpenAPI 2.0으로 표시된 정의)를 업로드하는 API Gateway API 가져오기 기능을 사용하여 API를 설정할 수 있습니다. 자습서: 예제를 가져와 REST API 생성에서 제공된 예에서는 API 가져오기 기능을 사용합니다.

AWS CLI 명령을 사용하여 엣지 최적화 API 설정

를 사용하여 API를 AWS CLI 설정하려면, `create-resource` 또는`create-rest-api`, `get-resources` `put-method` `put-method-responseput-integration`, 및 `put-integration-response` 명령을 사용해야 합니다. 다음 절차는 이러한 AWS CLI 명령을 사용하여 HTTP 통합 유형의 단순 PetStore API를 생성하는 방법을 보여줍니다.

를 사용하여 간단한 PetStore API를 만들려면 AWS CLI

1. `create-rest-api` 명령을 호출하여 특정 리전(`RestApi`)에서 `us-west-2`를 설정합니다.

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

다음은 이 명령의 출력입니다.

```
{
    "id": "vaz7da96z6",
    "name": "Simple PetStore (AWS CLI)",
    "createdDate": "2022-12-15T08:07:04-08:00",
    "apiKeySource": "HEADER",
    "endpointConfiguration": {
        "types": [
```

```
            "EDGE"
        ]
    },
    "disableExecuteApiEndpoint": false
}
```

새로 생성한 id의 반환된 RestApi를 기록해 둡니다. 이 값은 API의 다른 부분을 설정할 때 필요합니다.

2. get-resources 명령을 호출하여 RestApi의 루트 리소스 식별자를 검색합니다.

```
aws apigateway get-resources --rest-api-id vaz7da96z6 --region us-west-2
```

다음은 이 명령의 출력입니다.

```
{
    "items": [
        {
            "id": "begaltmsm8",
            "path": "/"
        }
    ]
}
```

루트 리소스 Id를 기록해 둡니다. 이 값은 API의 리소스 트리 설정과 메서드 및 통합 구성을 시작할 때 필요합니다.

3. create-resource 명령을 호출하여 루트 리소스(pets) 아래에 하위 리소스(begaltmsm8)를 추가합니다.

```
aws apigateway create-resource --rest-api-id vaz7da96z6 \
      --region us-west-2 \
      --parent-id begaltmsm8 \
      --path-part pets
```

다음은 이 명령의 출력입니다.

```
{
    "id": "6sxz2j",
    "parentId": "begaltmsm8",
    "pathPart": "pets",
```

```
    "path": "/pets"
}
```

루트 아래에 하위 리소스를 추가하려면 루트 리소스 Id를 parentId 속성 값으로 지정합니다. 이와 마찬가지로 pets 리소스 아래에 하위 리소스를 추가하려면 다음과 같이 parent-id 값을 pets의 id 리소스 6sxz2j로 교체하면서 이전 단계를 반복합니다.

```
aws apigateway create-resource --rest-api-id vaz7da96z6 \
      --region us-west-2 \
      --parent-id 6sxz2j \
      --path-part '{petId}'
```

경로 부분을 경로 파라미터로 만들려면 중괄호 {}로 묶습니다. 이 명령이 제대로 실행되면 다음과 같은 응답을 반환합니다.

```
{
    "id": "rjkmth",
    "parentId": "6sxz2j",
    "path": "/pets/{petId}",
    "pathPart": "{petId}"
}
```

/pets(6sxz2j) 및 /pets/{petId}(rjkmth) 두 리소스를 생성했으므로 이제 이 리소스의 메서드를 설정하는 단계로 이동할 수 있습니다.

4. put-method 명령을 호출하여 GET 리소스에 /pets HTTP 메서드를 추가합니다. 이렇게 하면 오픈 액세스로 Method의 API GET /pets가 생성되어 /pets의 ID 값에 따라 6sxz2j 리소스를 참조합니다.

```
aws apigateway put-method --rest-api-id vaz7da96z6 \
       --resource-id 6sxz2j \
       --http-method GET \
       --authorization-type "NONE" \
       --region us-west-2
```

다음은 이 명령이 제대로 실행되었을 경우의 출력 결과입니다.

```
{
    "httpMethod": "GET",
    "authorizationType": "NONE",
```

```
    "apiKeyRequired": false
}
```

이 메서드는 authorization-type이 NONE으로 설정되었으므로 오픈 액세스를 위한 것입니다. 인증된 사용자만 메서드를 호출하도록 허용하려면 IAM 역할과 정책, Lambda 권한 부여자(이전에는 사용자 지정 권한 부여자라고 함) 또는 Amazon Cognito 사용자 풀을 사용할 수 있습니다. 자세한 내용은 the section called “액세스 제어” 단원을 참조하세요.

/pets/{petId} 리소스(rjkmth)에 대한 읽기 액세스를 허용하려면 다음과 같이 그 리소스에 GET HTTP 메서드를 추가하여 Method의 API GET /pets/{petId}를 생성합니다.

```
aws apigateway put-method --rest-api-id vaz7da96z6 \
       --resource-id rjkmth --http-method GET \
       --authorization-type "NONE" \
       --region us-west-2 \
       --request-parameters method.request.path.petId=true
```

다음은 이 명령이 제대로 실행되었을 경우의 출력 결과입니다.

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

petId의 메서드 요청 경로 파라미터를 필수 요청 파라미터로 지정하여 동적으로 설정된 값이 해당 통합 요청 파라미터에 매핑되고 백엔드로 전달되도록 해야 합니다.

5. put-method-response 명령을 호출하여 GET /pets 메서드의 200 OK 응답을 설정함으로써 /pets의 ID 값에 따라 6sxz2j 리소스를 지정합니다.

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

다음은 이 명령의 출력입니다.

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

이와 마찬가지로 GET /pets/{petId} 메서드의 200 OK 응답을 설정하려면 다음과 같이 하여 /pets/{petId}의 리소스 ID 값에 따라 rjkmth 리소스를 지정합니다.

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

API에 대해 간단한 클라이언트 인터페이스를 설정한 후에는 백엔드로 API 메서드의 통합을 설정하는 단계로 나아갈 수 있습니다.

6. put-integration 명령을 호출하여 Integration 메서드에 지정된 HTTP 엔드포인트로 GET /pets을 설정합니다. /pets 리소스는 다음과 같이 리소스 ID 6sxz2j로 식별됩니다.

```
aws apigateway put-integration --rest-api-id vaz7da96z6 \
       --resource-id 6sxz2j --http-method GET --type HTTP \
       --integration-http-method GET \
       --uri 'http://petstore-demo-endpoint.execute-api.com/petstore/pets' \
       --region us-west-2
```

다음은 이 명령의 출력입니다.

```
{
    "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": []
}
```

uri의 통합 http://petstore-demo-endpoint.execute-api.com/petstore/pets는 GET /pets 메서드의 통합 엔드포인트를 지정합니다.

이와 마찬가지로 다음과 같이 GET /pets/{petId} 메서드에 대한 통합 요청을 생성합니다.

```
aws apigateway put-integration \
        --rest-api-id vaz7da96z6 \
        --resource-id rjkmth \
        --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
```

여기에서 통합 엔드포인트(uri의 http://petstore-demo-endpoint.execute-api.com/petstore/pets/{id}) 역시 경로 파라미터(id)를 사용합니다. 그 값은 상응하는 {petId}의 메서드 요청 경로 파라미터로부터 매핑됩니다. 매핑은 request-parameters의 일부로 정의됩니다. 이 매핑이 여기에 정의되지 않은 경우 클라이언트는 메서드 호출 시 오류 응답을 받습니다.

다음은 이 명령의 출력입니다.

```
{
    "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": []
}
```

7. put-integration-response 명령을 호출하여 HTTP 백엔드와 통합된 IntegrationResponse 메서드의 GET /pets를 생성합니다.

```
aws apigateway put-integration-response --rest-api-id vaz7da96z6 \
       --resource-id 6sxz2j --http-method GET \
       --status-code 200 --selection-pattern ""  \
```

```
       --region us-west-2
```

다음은 이 명령의 출력입니다.

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

이와 마찬가지로 다음과 같이 put-integration-response 명령을 호출하여 IntegrationResponse 메서드의 GET /pets/{petId}를 생성합니다.

```
aws apigateway put-integration-response --rest-api-id vaz7da96z6 \
      --resource-id rjkmth --http-method GET
      --status-code 200 --selection-pattern ""
      --region us-west-2
```

이전 단계를 통해 고객이 PetStore 웹사이트에서 이용 가능한 애완동물을 쿼리하고 지정된 식별자의 개별 애완동물을 볼 수 있는 간단한 API 설정을 마쳤습니다. 고객이 이를 호출할 수 있게 하려면 API를 배포해야 합니다.

8. 예를 들어, stage를 호출하여 API를 create-deployment 단계에 배포합니다.

```
aws apigateway create-deployment --rest-api-id vaz7da96z6 \
       --region us-west-2 \
       --stage-name test \
       --stage-description 'Test stage' \
       --description 'First deployment'
```

다음은 이 명령의 출력입니다.

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

브라우저에 `https://vaz7da96z6.execute-api.us-west-2.amazonaws.com/test/pets` URL을 입력하고 `vaz7da96z6`을 API의 ID로 대체하여 이 API를 테스트할 수 있습니다. 예상 결과는 다음과 같습니다.

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

`GET /pets/{petId}` 메서드를 테스트하려면 브라우저에 `https://vaz7da96z6.execute-api.us-west-2.amazonaws.com/test/pets/3`를 입력합니다. 다음과 같은 응답을 받게 됩니다.

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

Node.js용 AWS SDK를 사용한 엣지 최적화 API 설정

예를 들어, Node.js AWS SDK를 사용하여 API Gateway API를 생성하는 데 AWS SDK를 사용하는 방법을 설명합니다. 개발 환경 설정 방법을 포함하여 AWS SDK를 사용하는 자세한 내용은 SDK를 참조하십시오.AWS

Node.js AWS SDK를 사용하여 API를 설정하려면`createRestApi`, `createResource` 또는`getResources`,, 및 함수를 호출해야 합니다. `putMethod` `putMethodResponse` `putIntegration` `putIntegrationResponse`

다음 절차는 이러한 SDK 명령어를 사용하여 `GET /pets` 및 `GET /pets/{petId}` 메서드를 지원하는 간단한 PetStore API를 설정하는 필수 단계를 안내합니다.

Node.js 용 AWS SDK를 사용하여 간단한 PetStore API를 설정하려면

1. SDK 인스턴스화:

```
var AWS = require('aws-sdk');

AWS.config.region = 'us-west-2';
var apig = new AWS.APIGateway({apiVersion: '2015/07/09'});
```

2. `createRestApi` 함수를 호출하여 `RestApi` 엔터티를 설정합니다.

```
apig.createRestApi({
 name: "Simple PetStore (node.js SDK)",
 binaryMediaTypes: [
  '*'
 ],
 description: "Demo API created using the AWS SDK for node.js",
 version: "0.00.001"
}, function(err, data){
 if (!err) {
  console.log(data);
 } else {
  console.log('Create API failed:\n', err);
 }
});
```

이 함수는 다음 결과와 비슷한 출력을 반환합니다.

```
{
  id: 'iuo308uaq7',
  name: 'PetStore (node.js SDK)',
  description: 'Demo API created using the AWS SDK for node.js',
  createdDate: 2017-09-05T19:32:35.000Z,
  version: '0.00.001',
  binaryMediaTypes: [ '*' ]
}
```

그 결과로 얻는 API의 식별자는 `iuo308uaq7`입니다. API 설정을 계속하려면 이를 제공해야 합니다.

3. `getResources` 함수를 호출하여 `RestApi`의 루트 리소스 식별자를 검색합니다.

```
apig.getResources({
 restApiId: 'iuo308uaq7'
}, function(err, data){
 if (!err) {
  console.log(data);
 } else {
  console.log('Get the root resource failed:\n', err);
 }
})
```

이 함수는 다음 결과와 비슷한 출력 결과를 반환합니다.

```
{
    "items": [
        {
            "path": "/",
            "id": "s4fb0trnk0"
        }
    ]
}
```

루트 리소스 식별자는 `s4fb0trnk0`입니다. 이것은 다음 작업인 API 리소스 트리 구축의 시작점입니다.

4. `createResource` 함수를 호출하여 API에 대해 `/pets` 리소스를 설정함으로써 `s4fb0trnk0` 속성에 루트 리소스 식별자(`parentId`)를 지정합니다.

```
apig.createResource({
 restApiId: 'iuo308uaq7',
 parentId: 's4fb0trnk0',
 pathPart: 'pets'
}, function(err, data){
 if (!err) {
  console.log(data);
 } else {
  console.log("The '/pets' resource setup failed:\n", err);
 }
```

```
})
```

이 작업이 성공하면 결과는 다음과 같습니다.

```
{
    "path": "/pets",
    "pathPart": "pets",
    "id": "8sxa2j",
    "parentId": "s4fb0trnk0'"
}
```

/pets/{petId} 리소스를 설정하려면 다음과 같이 createResource 함수를 호출하여 /pets 속성에 새로 생성된 8sxa2j 리소스(parentId)를 지정합니다.

```
apig.createResource({
 restApiId: 'iuo308uaq7',
 parentId: '8sxa2j',
 pathPart: '{petId}'
}, function(err, data){
 if (!err) {
  console.log(data);
 } else {
  console.log("The '/pets/{petId}' resource setup failed:\n", err);
 }
})
```

이 작업이 성공하면 다음과 같이 새로 생성된 리소스 id 값을 반환합니다.

```
{
    "path": "/pets/{petId}",
    "pathPart": "{petId}",
    "id": "au5df2",
    "parentId": "8sxa2j"
}
```

이 절차 내내 /pets의 리소스 ID를 지정하여 8sxa2j 리소스를 참조하고 /pets/{petId}의 리소스 ID를 지정하여 au5df2 리소스를 참조합니다.

5. putMethod 함수를 호출하여 GET 리소스(/pets)에 8sxa2j HTTP 메서드를 추가합니다. 이를 통해 오픈 액세스로 GET /pets Method를 설정합니다.

```
apig.putMethod({
 restApiId: 'iuo308uaq7',
 resourceId: '8sxa2j',
 httpMethod: 'GET',
 authorizationType: 'NONE'
}, function(err, data){
 if (!err) {
  console.log(data);
 } else {
  console.log("The 'GET /pets' method setup failed:\n", err);
 }
})
```

이 함수는 다음 결과와 비슷한 출력 결과를 반환합니다.

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

오픈 액세스로 GET의 API 메서드를 설정하는 /pets/{petId} 리소스(au5df2)에 GET /pets/{petId} HTTP 메서드를 추가하려면 다음과 같이 putMethod 함수를 호출합니다.

```
apig.putMethod({
 restApiId: 'iuo308uaq7',
 resourceId: 'au5df2',
 httpMethod: 'GET',
 authorizationType: 'NONE',
      requestParameters: {
        "method.request.path.petId" : true
      }
}, function(err, data){
 if (!err) {
  console.log(data);
 } else {
  console.log("The 'GET /pets/{petId}' method setup failed:\n", err);
 }
})
```

이 함수는 다음 결과와 비슷한 출력 결과를 반환합니다.

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

앞의 예제와 같이 requestParameters 속성을 설정하여 클라이언트에서 제공한 petId 값을 백엔드로 매핑 및 전달해야 합니다.

6. putMethodResponse 함수를 호출하여 GET /pets 메서드에 메서드 응답을 설정합니다.

```
apig.putMethodResponse({
 restApiId: 'iuo308uaq7',
 resourceId: "8sxa2j",
 httpMethod: 'GET',
 statusCode: '200'
}, function(err, data){
 if (!err) {
  console.log(data);
 } else {
  console.log("Set up the 200 OK response for the 'GET /pets' method failed:\n",
 err);
 }

})
```

이 함수는 다음 결과와 비슷한 출력 결과를 반환합니다.

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

GET /pets/{petId} 메서드의 200 OK 응답을 설정하려면 putMethodResponse 함수를 호출하여 /pets/{petId} 속성에 au5df2 리소스 식별자(resourceId)를 지정합니다.

```
apig.putMethodResponse({
 restApiId: 'iuo308uaq7',
 resourceId: "au5df2",
 httpMethod: 'GET',
 statusCode: '200'
}, function(err, data){
 if (!err) {
  console.log(data);
 } else {
  console.log("Set up the 200 OK response for the 'GET /pets/{petId}' method
 failed:\n", err);
 }

})
```

7. putIntegration 함수를 호출하여 Integration 메서드에 지정된 HTTP 엔드포인트로 GET /pets을 설정함으로써 /pets 속성에 8sxa2j 리소스 식별자(parentId)를 제공합니다.

```
apig.putIntegration({
 restApiId: 'iuo308uaq7',
 resourceId: '8sxa2j',
 httpMethod: 'GET',
 type: 'HTTP',
 integrationHttpMethod: 'GET',
 uri: 'http://perstore-demo-endpoint.execute-api.com/pets'
}, function(err, data){
 if (!err) {
  console.log(data);
 } else {
  console.log("Set up the integration of the 'GET /' method of the API failed:\n",
 err);
 }

})
```

이 함수는 다음과 비슷한 출력 결과를 반환합니다.

```
{
    "httpMethod": "GET",
    "passthroughBehavior": "WHEN_NO_MATCH",
    "cacheKeyParameters": [],
```

```
    "type": "HTTP",
    "uri": "http://petstore-demo-endpoint.execute-api.com/petstore/pets",
    "cacheNamespace": "8sxa2j"
}
```

백엔드의 GET /pets/{petId}의 HTTP 엔드포인트로 http://perstore-demo-endpoint.execute-api.com/pets/{id} 메서드의 통합을 설정하려면 다음과 같이 putIntegration 함수를 호출하여 /pets/{petId} 속성에 API의 au5df2 리소스 식별자(parentId)를 제공합니다.

```
apig.putIntegration({
 restApiId: 'iuo308uaq7',
 resourceId: 'au5df2',
 httpMethod: 'GET',
 type: 'HTTP',
 integrationHttpMethod: 'GET',
 uri: 'http://perstore-demo-endpoint.execute-api.com/pets/{id}',
       requestParameters: {
          "integration.request.path.id": "method.request.path.petId"
       }
}, function(err, data){
 if (!err) {
  console.log(data);
 } else {
  console.log("The 'GET /pets/{petId}' method integration setup failed:\n", err);
 }

})
```

이 함수가 성공적으로 실행되면 다음과 비슷한 출력 결과를 반환합니다.

```
{
    "httpMethod": "GET",
    "passthroughBehavior": "WHEN_NO_MATCH",
    "cacheKeyParameters": [],
    "type": "HTTP",
    "uri": "http://petstore-demo-endpoint.execute-api.com/petstore/pets/{id}",
    "cacheNamespace": "au5df2",
    "requestParameters": {
       "integration.request.path.id": "method.request.path.petId"
    }
```

```
}
```

8. putIntegrationResponse 함수를 호출하여 GET /pets 메서드에 대한 200 OK 통합 응답을 설정함으로써 /pets 속성에 API의 8sxa2j 리소스 식별자(resourceId)를 지정합니다.

```
apig.putIntegrationResponse({
 restApiId: 'iuo308uaq7',
 resourceId: '8sxa2j',
 httpMethod: 'GET',
 statusCode: '200',
 selectionPattern: ''
}, function(err, data){
 if (!err) {
  console.log(data);
 } else
  console.log("The 'GET /pets' method integration response setup failed:\n", err);

})
```

이 함수는 다음 결과와 비슷한 출력 결과를 반환합니다.

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

GET /pets/{petId} 메서드의 200 OK 통합 응답을 설정하여 putIntegrationResponse 함수를 호출함으로써 /pets/{petId} 속성에 API의 au5df2 리소스 식별자(resourceId)를 지정합니다.

```
apig.putIntegrationResponse({
 restApiId: 'iuo308uaq7',
 resourceId: 'au5df2',
 httpMethod: 'GET',
 statusCode: '200',
 selectionPattern: ''
}, function(err, data){
 if (!err) {
  console.log(data);
 } else
```

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

})
```

9. API를 배포하기 전에 API 호출을 테스트해봅니다. GET /pets 메서드 호출을 테스트하려면 다음과 같이 testInvokeMethod를 호출하여 /pets 속성에 8sxa2j 리소스 식별자(resourceId)를 지정합니다.

```
apig.testInvokeMethod({
 restApiId: 'iuo308uaq7',
 resourceId: '8sxa2j',
 httpMethod: "GET",
 pathWithQueryString: '/'
}, function(err, data){
 if (!err) {
  console.log(data)
 } else {
  console.log('Test-invoke-method on 'GET /pets' failed:\n', err);
 }
})
```

GET /pets/{petId} 메서드 호출을 테스트하려면 다음과 같이 testInvokeMethod를 호출하여 /pets/{petId} 속성에 au5df2 리소스 식별자(resourceId)를 지정합니다.

```
apig.testInvokeMethod({
 restApiId: 'iuo308uaq7',
 resourceId: 'au5df2',
 httpMethod: "GET",
 pathWithQueryString: '/'
}, function(err, data){
 if (!err) {
  console.log(data)
 } else {
  console.log('Test-invoke-method on 'GET /pets/{petId}' failed:\n', err);
 }
})
```

10. 끝으로 고객이 호출할 수 있게 API를 배포합니다.

```
apig.createDeployment({
```

```
 restApiId: 'iuo308uaq7',
 stageName: 'test',
 stageDescription: 'test deployment',
 description: 'API deployment'
}, function(err, data){
 if (err) {
  console.log('Deploying API failed:\n', err);
 } else {
  console.log("Deploying API succeeded\n", data);
 }
})
```

OpenAPI 정의를 가져와 엣지 최적화 API 설정

해당하는 API Gateway API 엔터티의 OpenAPI 정의를 지정하고 OpenAPI 정의를 API Gateway로 가져와 API Gateway에서 API를 설정할 수 있습니다.

다음 OpenAPI 정의는 PetStore 웹 사이트의 HTTP 엔드포인트와 통합된 `GET /` 메서드만 백엔드에 노출하고 응답을 반환하는 간단한 API를 설명합니다. `200 OK`

OpenAPI 2.0

```
{
  "swagger": "2.0",
  "info": {
    "title": "Simple PetStore (OpenAPI)"
  },
  "schemes": [
    "https"
  ],
  "paths": {
    "/pets": {
      "get": {
        "responses": {
          "200": {
            "description": "200 response"
          }
        },
        "x-amazon-apigateway-integration": {
          "responses": {
            "default": {
              "statusCode": "200"
```

```
            }
          },
          "uri": "http://petstore-demo-endpoint.execute-api.com/petstore/pets",
          "passthroughBehavior": "when_no_match",
          "httpMethod": "GET",
          "type": "http"
        }
      }
    },
    "/pets/{petId}": {
      "get": {
        "parameters": [
          {
            "name": "petId",
            "in": "path",
            "required": true,
            "type": "string"
          }
        ],
        "responses": {
          "200": {
            "description": "200 response"
          }
        },
        "x-amazon-apigateway-integration": {
          "responses": {
            "default": {
              "statusCode": "200"
            }
          },
          "requestParameters": {
            "integration.request.path.id": "method.request.path.petId"
          },
          "uri": "http://petstore-demo-endpoint.execute-api.com/petstore/pets/{id}",
          "passthroughBehavior": "when_no_match",
          "httpMethod": "GET",
          "type": "http"
        }
      }
    }
  }
}
```

다음 절차에서는 API Gateway 콘솔을 사용하여 이러한 OpenAPI 정의를 API Gateway로 가져오는 방법을 설명합니다.

API Gateway 콘솔을 이용해 간단한 OpenAPI 정의를 가져오려면

1. https://console.aws.amazon.com/apigateway에서 API Gateway 콘솔에 로그인합니다.
2. API 생성을 선택한 다음 REST API에 대해 가져오기를 선택합니다.
3. 앞서 파일에 OpenAPI 정의를 저장한 경우, 파일 선택을 선택합니다. OpenAPI 정의를 복사하여 가져온 텍스트 편집기에 붙여 넣을 수도 있습니다.
4. API 엔드포인트 유형에서 엣지 최적화를 선택합니다.
5. OpenAPI 정의를 가져오려면 API 생성을 선택합니다.

를 사용하여 OpenAPI 정의를 가져오려면 OpenAPI 정의를 파일에 저장한 후 다음 명령을 실행합니다. 이때 해당 영역을 사용하고 절대 OpenAPI 파일 경로가 다음과 같다고 가정합니다. AWS CLIus-west-2 `file:///path/to/API_OpenAPI_template.json`

```
aws apigateway import-rest-api --body 'file:///path/to/API_OpenAPI_template.json' --
region us-west-2
```

API Gateway에서 리전 API 설정

API 요청이 API가 배포되는 리전 내의 EC2 인스턴스 또는 서비스에서 주로 발신되는 경우 리전 API 엔드포인트는 일반적으로 연결 지연 시간을 줄이므로 그러한 시나리오에 권장됩니다.

> **Note**
>
> API 클라이언트가 지리적으로 분산되어 있는 경우, API Gateway가 API를 서비스 제어 CloudFront 배포와 연결하지 않도록 하기 위해 리전 API 엔드포인트를 자체 Amazon CloudFront 배포와 함께 사용할 수 있습니다. 이 사용 사례에 대한 자세한 내용은 고유의 CloudFront 배포를 사용하여 API Gateway를 설정하려면 어떻게 해야 합니까?를 참조하세요.

리전 API를 생성하려면 엣지 최적화 API 생성의 단계를 따르되 `REGIONAL` 유형을 API endpointConfiguration의 유일한 옵션으로 명시적으로 설정해야 합니다.

다음 예제에서는 API Gateway 콘솔, AWS CLI, Node.js JavaScript용 AWS SDK를 사용하여 리전 API를 생성하는 방법을 보여줍니다.

주제

- API Gateway 콘솔을 사용하여 리전 API 생성
- AWS CLI를 사용하여 리전 API 생성
- JavaScript용 AWS SDK를 사용하여 리전 API 생성
- 리전 API 테스트

API Gateway 콘솔을 사용하여 리전 API 생성

API Gateway 콘솔을 사용하여 리전 API를 생성하려면

1. https://console.aws.amazon.com/apigateway에서 API Gateway 콘솔에 로그인합니다.
2. 다음 중 하나를 수행하십시오.
   - 첫 번째 API를 생성하려면 REST API에서 빌드를 선택합니다.
   - 이전에 API를 생성한 경우 API 생성을 선택한 다음 REST API에서 빌드를 선택합니다.
3. 이름에 이름을 입력합니다.
4. (선택 사항) 설명에 설명을 입력합니다.
5. API 엔드포인트 유형 설정을 지역으로 유지합니다.
6. API 생성(Create API)을 선택합니다.

AWS CLI를 사용하여 리전 API 생성

AWS CLI를 사용하여 리전 API를 생성하려면 다음과 같이 `create-rest-api` 명령을 호출하십시오.

```
aws apigateway create-rest-api \
        --name 'Simple PetStore (AWS CLI, Regional)' \
        --description 'Simple regional PetStore API' \
        --region us-west-2 \
        --endpoint-configuration '{ "types": ["REGIONAL"] }'
```

성공적인 응답은 다음과 비슷한 페이로드를 반환합니다.

```
{
    "createdDate": "2017-10-13T18:41:39Z",
    "description": "Simple regional PetStore API",
    "endpointConfiguration": {
        "types": "REGIONAL"
```

```
    },
    "id": "0qzs2sy7bh",
    "name": "Simple PetStore (AWS CLI, Regional)"
}
```

여기서부터는 the section called “AWS CLI 명령을 사용하여 엣지 최적화 API 설정”에 제공된 지침에 따라 이 API에 대해 메서드와 통합을 설정할 수 있습니다.

JavaScript용 AWS SDK를 사용하여 리전 API 생성

JavaScript용 AWS SDK를 사용하여 리전 API 생성:

```
apig.createRestApi({
 name: "Simple PetStore (node.js SDK, regional)",
 endpointConfiguration: {
  types: ['REGIONAL']
 },
 description: "Demo regional API created using the AWS SDK for node.js",
 version: "0.00.001"
}, function(err, data){
 if (!err) {
  console.log('Create API succeeded:\n', data);
  restApiId = data.id;
 } else {
  console.log('Create API failed:\n', err);
 }
});
```

성공적인 응답은 다음과 비슷한 페이로드를 반환합니다.

```
{
    "createdDate": "2017-10-13T18:41:39Z",
    "description": "Demo regional API created using the AWS SDK for node.js",
    "endpointConfiguration": {
        "types": "REGIONAL"
    },
    "id": "0qzs2sy7bh",
    "name": "Simple PetStore (node.js SDK, regional)"
}
```

이전 단계를 완료한 후에 the section called “Node.js용 AWS SDK를 사용한 엣지 최적화 API 설정”의 지침에 따라 이 API에 대해 메서드와 통합을 설정할 수 있습니다.

리전 API 테스트

일단 배포된 리전 API의 기본 URL 호스트 이름은 그 형식이 다음과 같습니다.

```
{restapi-id}.execute-api.{region}.amazonaws.com
```

API를 호출할 기본 URL은 다음과 같습니다.

```
https://{restapi-id}.execute-api.{region}.amazonaws.com/{stage}
```

이 예제에서는 `GET /pets` 및 `GET /pets/{petId}`를 설정했다는 가정 하에 브라우저에 다음 URL을 입력하여 API를 테스트할 수 있습니다.

```
https://0qzs2sy7bh.execute-api.us-west-2.amazonaws.com/test/pets
```

및

```
https://0qzs2sy7bh.execute-api.us-west-2.amazonaws.com/test/pets/1
```

또는 다음과 같이 cURL 명령을 사용할 수 있습니다.

```
curl -X GET https://0qzs2sy7bh.execute-api.us-west-2.amazonaws.com/test/pets
```

및

```
curl -X GET https://0qzs2sy7bh.execute-api.us-west-2.amazonaws.com/test/pets/2
```

API Gateway에서 REST API 메서드 설정

API Gateway에서 API 메서드는 메서드 요청 및 메서드 응답을 구현합니다. API 메서드를 설정하여 클라이언트가 백엔드에 있는 서비스에 대한 액세스 요청을 제출하기 위해 해야 할 것을 정의하고 클라이언트가 그 반대 급부로 수신하는 응답을 정의합니다. 입력의 경우, 메서드 요청 파라미터 또는 적용 가능 페이로드를 선택하여 클라이언트가 실행 시간에 필수 또는 선택 데이터를 제공하도록 할 수 있습니다. 출력의 경우, 메서드 응답 상태 코드, 헤더 및 적용 가능 본문을 백엔드 응답 데이터가 클라이언트에 반환되기 전에 이 데이터를 매핑할 대상으로 결정합니다. 클라이언트 개발자가 해당 API의 동작과 입력 및 출력 형식을 이해하는 데 도움을 주기 위해 API를 문서화하고 잘못된 요청에 대해 적절한 오류 메시지를 제공할 수 있습니다.

API 메서드 요청은 HTTP 요청입니다. 메서드 요청을 설정하려면 HTTP 메서드(또는 동사), API 리소스 경로, 헤더, 적용 가능한 쿼리 문자열 파라미터를 구성합니다. 또한 HTTP 메서드가 POST, PUT 또는 PATCH인 경우 페이로드를 구성합니다. 예를 들어 PetStore 샘플 API를 사용하여 애완동물을 가져오려면 의 GET /pets/{petId} API 메서드 요청을 정의합니다. 여기서 {petId} 는 런타임에 숫자를 받을 수 있는 경로 매개변수입니다.

```
GET /pets/1
Host: apigateway.us-east-1.amazonaws.com
...
```

클라이언트가 잘못된 경로를 지정하는 경우(예: /pet/1 대신에 /pets/one 또는 /pets/1을 지정) 예외가 발생합니다.

API 메서드 응답은 특정 상태 코드의 HTTP 응답입니다. 비 프록시 통합의 경우에는 메서드 응답을 설정하여 매핑의 필수 또는 선택 대상을 지정해야 합니다. 이를 통해 통합 응답 헤더 또는 본문을 연결된 메서드 응답 헤더 또는 본문으로 변환합니다. 매핑은 통합을 통해 있는 그대로 헤더나 본문을 전달하는 자격 증명 변환만큼 간단할 수 있습니다. 예를 들어 다음 200 메서드 응답에서는 성공적인 통합 응답을 있는 그대로 패스스루하는 예를 보여줍니다.

```
200 OK
Content-Type: application/json
...

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

원칙적으로 백엔드에서 오는 특정 응답에 상응하는 메서드 응답을 정의할 수 있습니다. 일반적으로 이 작업에는 2XX, 4XX 및 5XX 응답이 수반됩니다. 그러나 이렇게 하는 것은 백엔드가 반환하는 모든 응답을 미리 알 수 없는 경우가 많으므로 실용적 방법이 되지 못할 수 있습니다. 실제로 한 가지 메서드 응답을 기본값으로 지정하여 백엔드에서 오는 알 수 없거나 매핑되지 않은 응답을 처리할 수 있습니다. 500 응답을 기본값으로 지정하는 것이 좋습니다. 어떤 경우에도 비 프록시 통합에 대해 메서드 응답을 최소 한 개 설정해야 합니다. 그렇게 하지 않으면 백엔드에서 요청이 성공하더라도 API Gateway는 클라이언트에게 500 오류 응답을 반환합니다.

해당 API에 대해 Java SDK와 같은 강력한 형식의 SDK를 지원하려면 메서드 요청에 대해 입력용 데이터 모델을 정의하고 메서드 응답에 대해 출력용 데이터 모델을 정의해야 합니다.

주제

- API Gateway에서 메서드 요청 설정
- API Gateway에서 메서드 응답 설정
- API Gateway 콘솔을 사용하여 메서드 설정

API Gateway에서 메서드 요청 설정

메서드 요청을 설정하려면 RestApi리소스를 생성한 후 다음 작업을 수행해야 합니다.

1. 새 API 생성 또는 기존 API 리소스 엔터티 선택.
2. 새로운 또는 선택된 API `Resource`에서 특정 HTTP 동사인 API 메서드 리소스를 생성. 이 작업은 다음과 같은 하위 작업으로 나눌 수 있습니다.
   - HTTP 메서드를 메서드 요청에 추가
   - 요청 파라미터 구성
   - 요청 본문에 모델을 정의
   - 권한 부여 체계 적용
   - 요청 검증 활성화

다음 메서드를 사용하여 이 작업을 수행할 수 있습니다.

- API Gateway 콘솔
- AWS CLI 명령 (리소스 생성 및 풋 메소드)
- AWS SDK 함수 (예: Node.js, CreateResource 및 PutMethod)
- API Gateway REST API(resource:create 및 method:put).

이러한 도구의 사용 예시는 API Gateway에서 REST API 설정 초기화를 참조하십시오.

주제

- API 리소스 설정
- HTTP 메서드 설정
- 메서드 요청 파라미터 설정
- 메서드 요청 모델 설정
- 메서드 요청 권한 부여 설정

- 메서드 요청 검증 설정

API 리소스 설정

API Gateway API에서 계층 꼭대기에 있는 루트 리소스(/)와 함께 주소 지정 가능 리소스를 API 리소스 엔터티의 트리로 노출합니다. 루트 리소스는 API 엔드포인트 및 단계 이름으로 구성된 API의 기본 URL에 상대적입니다. API Gateway 콘솔에서 이 기본 URI는 URI 호출(Invoke URI)로 참조되고 API가 배포된 후에 API의 단계 편집기에 표시됩니다.

API 엔드포인트는 기본 호스트 이름 또는 사용자 지정 도메인 이름일 수 있습니다. 기본 호스트 이름은 다음 형식으로 되어 있습니다.

```
{api-id}.execute-api.{region}.amazonaws.com
```

이 형식에서 *{api-id}*는 API Gateway가 생성하는 API 식별자를 나타냅니다. *{region}* 변수는 API를 생성할 때 선택한 AWS 리전(예: us-east-1)을 나타냅니다. 사용자 지정 도메인 이름은 유효한 인터넷 도메인에 속한, 사용자에게 친숙한 이름입니다. 예를 들어 example.com의 인터넷 도메인을 등록하였다면 모든 *.example.com은 유효한 사용자 지정 도메인 이름입니다. 자세한 내용은 사용자 지정 도메인 이름 생성 단원을 참조하십시오.

PetStore 샘플 API의 경우 루트 리소스 (/) 는 펫 스토어를 노출합니다. /pets 리소스는 반려 동물 스토어에서 사용 가능한 반려 동물 모음을 나타냅니다. /pets/{petId}는 특정 식별자(petId)의 개별 반려 동물을 공개합니다. {petId}의 경로 파라미터는 요청 파라미터의 일부입니다.

API 리소스를 설정하려면 기존 리소스를 상위 리소스로 선택한 후 이 상위 리소스 아래에 하위 리소스를 생성합니다. 상위 리소스인 루트 리소스로 시작하여 이 상위 리소스에 리소스를 추가하고 이 하위 리소스에 또 다른 리소스를 새로운 상위 리소스로 추가하는 등의 작업을 상위 식별자에 이르기까지 수행합니다. 그 다음에 이름이 지정된 리소스를 상위 리소스에 추가합니다.

를 사용하면 get-resources 명령을 호출하여 API의 어떤 리소스를 사용할 수 있는지 확인할 수 있습니다. AWS CLI

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

그 결과 현재 사용 가능한 API 리소스의 목록을 얻을 수 있습니다. PetStore 샘플 API의 경우 이 목록은 다음과 같습니다.

```
{
```

```
    "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"
        }
    ]
}
```

각 항목은 리소스의 식별자(id)와 루트 리소스를 제외한 바로 위에 있는 상위 리소스(parentId)뿐 아니라 리소스 이름(pathPart)도 나열합니다. 루트 리소스는 상위 리소스가 없다는 점에서 특별합니다. 리소스를 상위 리소스로 선택한 후에 다음 명령을 호출하여 하위 리소스를 추가합니다.

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

예를 들어 PetStore 웹 사이트에 판매용 애완동물 사료를 추가하려면 food to와 parent-id to를 설정하여 루트 (/) path-part 에 food 리소스를 추가합니다svzr2028x8. 그 결과는 다음과 같습니다.

```
{
    "path": "/food",
    "pathPart": "food",
    "id": "xdsvhp",
```

```
    "parentId": "svzr2028x8"
}
```

프록시 리소스를 사용하여 API 설정 간소화

비즈니스가 성장함에 따라 PetStore 소유자는 판매용으로 식품, 장난감 및 기타 애완동물 관련 품목을 추가하기로 결정할 수 있습니다. 이를 지원하기 위해 루트 리소스 아래에 /food, /toys 및 기타 리소스를 추가할 수 있습니다. 각 판매 범주에 /food/{type}/{item}, /toys/{type}/{item} 등 더 많은 리소스를 추가하고 싶을 수도 있습니다. 이는 번거로운 일일 수 있습니다. 리소스 경로에 중간 계층 {subtype}을 추가하여 경로 계층을 /food/{type}/{subtype}/{item}, /toys/{type}/{subtype}/{item} 등으로 변경하기로 한 경우, 그러한 변경으로 인해 기존 API 설정이 손상됩니다. 이를 방지하기 위해 API Gateway 프록시 리소스를 사용하여 한꺼번에 일련의 API 리소스를 공개할 수 있습니다.

API Gateway는 프록시 리소스를 해당 요청을 제출할 때 지정할 리소스의 자리 표시자로 정의합니다. 프록시 리소스는 복잡한 경로 파라미터로 자주 참조되는 {proxy+}의 특별 경로 파라미터로 표시됩니다. + 기호는 어떤 하위 리소스이든지 거기에 추가된 것을 나타냅니다. /parent/{proxy+} 자리 표시자는 /parent/*의 경로 패턴과 일치하는 모든 리소스를 나타냅니다. 복잡한 경로 변수 이름인 proxy는 일반 경로 파라미터 이름을 처리할 때와 같은 방식으로 다른 문자열로 대체할 수 있습니다.

를 사용하여 다음 AWS CLI명령을 호출하여 루트 () /{proxy+} 아래에 프록시 리소스를 설정합니다.

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

그 결과는 다음과 비슷합니다.

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

PetStore API의 예를 들면, /{proxy+}를 사용하여 /pets 및 /pets/{petId}를 모두 나타낼 수 있습니다. 이 프록시 리소스는, 등, 또는 /food/{type}/{item}/toys/{type}/{item}, 등과 같은 다른 (기존 또는 to-be-added/food/{type}/{subtype}/{item}) 리소스를 참조할 수도 있습니

다. /toys/{type}/{subtype}/{item} 백엔드 개발자는 리소스 계층을 결정하고 클라이언트 개발자는 이를 이해할 책임이 있습니다. API Gateway는 클라이언트가 제출한 것은 무엇이든 백엔드에 전달할 뿐입니다.

API에는 프록시 리소스가 둘 이상 있을 수 있습니다. 예를 들어 다음 프록시 리소스는 API 내에서 허용됩니다.

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

프록시 리소스에 비 프록시 형제 리소스가 있는 경우 형제 리소스는 프록시 리소스의 표시에서 제외됩니다. 앞 예제의 경우, /{proxy+}는 /parent[/*] 리소스를 제외한 루트 리소스 아래의 모든 리소스를 가리킵니다. 즉 동일한 리소스 계층 수준에서는 특정 리소스에 대한 메서드 요청이 일반 리소스에 대한 메서드 요청에 우선합니다.

프록시 리소스에는 하위 리소스가 전혀 없을 수 있습니다. {proxy+} 뒤의 모든 API 리소스는 중복되며 의미가 명확하지 않습니다. 다음 프록시 리소스는 API 내에서 허용되지 않습니다.

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

HTTP 메서드 설정

API 메서드 요청은 API Gateway Method 리소스로 캡슐화됩니다. 메서드 요청을 설정하려면 먼저 Method 리소스를 인스턴스화하여 HTTP 메서드를 최소 한 개 설정하고 그 메서드에 권한 부여 유형을 설정해야 합니다.

프록시 리소스와 긴밀히 연결된 API Gateway는 ANY의 HTTP 메서드를 지원합니다. 이 ANY 메서드는 실행 시간에 제공될 모든 HTTP 메서드를 나타냅니다. 이를 통해 DELETE, GET, HEAD, OPTIONS, PATCH, POST 및 PUT의 모든 지원되는 HTTP 메서드에 대해 단일 API 메서드 설정을 사용할 수 있습니다.

비 프록시 리소스에서도 ANY 메서드를 설정할 수 있습니다. ANY 메서드를 프록시 리소스와 결합하여 API의 모든 리소스에 대해 지원되는 모든 HTTP 메서드에 대한 단일 API 메서드 설정을 가져옵니다. 뿐만 아니라 백엔드는 기존 API 설정을 손상하지 않고도 발전할 수 있습니다.

API 메서드를 설정하기 전에 누가 메서드를 호출할 수 있는지 고려합니다. 계획에 따라 권한 부여 유형을 설정합니다. 오픈 액세스의 경우 그 유형을 NONE으로 설정합니다. IAM 권한을 사용하려면 권한 부

여 유형을 AWS_IAM으로 설정합니다. Lambda 권한 부여자 함수를 사용하려면 이 속성을 CUSTOM으로 설정합니다. Amazon Cognito 사용자 풀을 사용하려면 권한 부여 유형을 COGNITO_USER_POOLS로 설정합니다.

다음 AWS CLI 명령은 IAM 권한을 사용하여 액세스를 제어하여 지정된 리소스 (6sxz2j) 에 대해 ANY 동사의 메서드 요청을 생성하는 방법을 보여줍니다.

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

다른 권한 부여 유영으로 API 메서드 요청을 생성하려면 the section called “메서드 요청 권한 부여 설정”을 참조하십시오.

메서드 요청 파라미터 설정

메서드 요청 파라미터는 클라이언트가 메서드 요청을 완료하는 데 필요한 입력 데이터 또는 실행 컨텍스트를 제공하는 방식입니다. 메서드 파라미터는 경로 파라미터, 헤더 또는 쿼리 문자열 파라미터일 수 있습니다. 메서드 요청을 설정하는 과정에서 필수 요청 파라미터를 선언하여 클라이언트가 사용할 수 있게 해야 합니다. 비 프록시 통합의 경우 이러한 요청 파라미터를 백엔드 요구 사항과 호환되는 형식으로 변환할 수 있습니다.

예를 들어 GET /pets/{petId} 메서드 요청의 경우 {petId} 경로 변수는 필수 요청 파라미터입니다. AWS CLI의 put-method 명령을 호출할 때 이 경로 파라미터를 선언할 수 있습니다. 이를 코드로 나타내면 다음과 같습니다.

```
aws apigateway put-method --rest-api-id vaz7da96z6 \
       --resource-id rjkmth \
       --http-method GET \
       --authorization-type "NONE" \
       --region us-west-2 \
       --request-parameters method.request.path.petId=true
```

파라미터가 필요하지 않은 경우 false에서 request-parameters로 설정할 수 있습니다. 예를 들어 GET /pets 메서드가 type의 선택적 쿼리 문자열 파라미터와 breed의 선택적 헤더 파라미터를 사용하는 경우 /pets 리소스 id가 6sxz2j라는 가정 하에 다음과 같은 CLI 명령을 사용하여 두 파라미터를 선언할 수 있습니다.

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

이처럼 축약된 형식 대신에 다음과 같이 JSON 문자열을 사용하여 `request-parameters` 값을 설정할 수 있습니다.

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

클라이언트는 이 설정을 사용하여 반려 동물을 유형별로 쿼리할 수 있습니다.

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

또한 클라이언트는 다음과 같이 푸들 종의 개를 쿼리할 수 있습니다.

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

메서드 요청 파리미터를 통합 요청 파라미터로 매핑하는 방법에 대한 자세한 내용은 the section called “통합”를 참조하십시오.

메서드 요청 모델 설정

페이로드로 입력 데이터를 받아들일 수 있는 API 메서드의 경우에는 모델을 사용할 수 있습니다. 모델은 JSON 스키마 draft 4로 표시되고 요청 본문의 데이터 구조를 설명합니다. 클라이언트는 모델을 통해 메서드 요청 페이로드를 입력으로 구성하는 방법을 결정할 수 있습니다. 더 중요한 것은 API Gateway가 모델을 사용하여 요청의 유효성을 검증하고 SDK를 생성하며 API Gateway 콘솔에서 통합을 설정하기 위한 매핑 템플릿을 초기화한다는 것입니다. 모델을 만드는 방법에 대한 자세한 내용은 데이터 모델 이해를 참조하세요.

콘텐츠 유형에 따라 메서드 페이로드에는 다양한 형식이 있을 수 있습니다. 모델은 적용된 페이로드의 미디어 유형에 대해 인덱싱됩니다. API Gateway는 `Content-Type` 요청 헤더를 사용하여 콘텐츠 유형을 결정합니다. 메서드 요청 모델을 설정하려면 명령을 호출할 때 해당 "*<media-type>*":"*<model-name>*" 형식의 키-값 쌍을 `requestModels` 맵에 추가하십시오. AWS CLI `put-method`

콘텐츠 유형에 관계없이 동일한 모델을 사용하려면 `$default`을(를) 키로 지정합니다.

예를 들어, PetStore 예제 API의 `POST /pets` 메서드 요청의 JSON 페이로드에 모델을 설정하려면 다음 명령을 호출하면 됩니다. AWS CLI

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

여기에서 `petModel`은 반려 동물을 설명하는 `name` 리소스의 `Model` 속성 값입니다. 실제 스키마 정의는 `schema` 리소스의 `Model` 속성의 JSON 문자열 값으로 표시됩니다.

API의 Java 또는 기타 강력한 형식의 SDK에서 입력 데이터는 스키마 정의에서 파생된 `petModel` 클래스로 캐스팅됩니다. 요청 모델을 사용하는 경우, 생성된 SDK 내 입력 데이터는 기본 `Empty` 모델에서 파생된 `Empty` 클래스로 캐스팅됩니다. 이 경우 클라이언트는 필수 입력을 제공하기 위해 정확한 데이터 클래스를 인스턴스화할 수는 없습니다.

메서드 요청 권한 부여 설정

API 메서드를 호출할 수 있는 사용자를 관리하기 위해 메서드에서 권한 부여 유형을 구성할 수 있습니다. 이 유형을 사용하여 IAM 역할과 정책(`AWS_IAM`), Amazon Cognito 사용자 풀(`COGNITO_USER_POOLS`) 또는 Lambda 권한 부여자(`CUSTOM`) 등 지원되는 권한 부여자 중 하나를 적용할 수 있습니다.

IAM 권한을 사용하여 API 메서드에 대한 액세스 권한을 부여하려면 **`AWS_IAM`**에 `authorization-type` 입력 속성을 설정합니다. 이 옵션이 설정되면 API Gateway는 호출자의 보안 인증 정보를 바탕으로 요청에 대한 호출자의 서명을 확인합니다. 확인된 사용자에게 메서드를 호출할 수 있는 권한이 있는 경우 그 요청은 수락됩니다. 그렇지 않다면 요청은 거부되고 호출자는 인증되지 않은 오류 응답을 받습니다. 호출자에게 API 메서드를 호출할 권한이 없는 한 메서드 호출은 성공하지 못합니다. 다음 IAM 정책은 호출자에게 동일한 AWS 계정에서 생성된 모든 API 메서드를 호출할 수 있는 권한을 부여합니다.

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

```
        {
            "Effect": "Allow",
            "Action": [
                "execute-api:Invoke"
            ],
            "Resource": "arn:aws:execute-api:*:*:*"
        }
    ]
}
```

자세한 설명은 the section called “IAM 권한 사용” 섹션을 참조하세요.

현재는 API 소유자 AWS 계정내의 사용자, 그룹 및 역할에만 이 정책을 부여할 수 있습니다. 작업을 호출하는 데 필요한 권한을 AWS 계정 가진 API 소유자 내에서 역할을 맡을 AWS 계정 수 있는 경우에만 다른 사용자가 API 메서드를 호출할 수 있습니다. `execute-api:Invoke` 교차 계정 권한에 대한 자세한 내용은 IAM 역할 사용 단원을 참조하세요.

서명 버전 4 서명을 구현하는 Postman과 같은 AWS SDK 또는 REST API 클라이언트를 사용할 AWS CLI수 있습니다.

Lambda 권한 부여자를 사용하여 API 메서드에 대한 액세스 권한을 부여하려면 `CUSTOM`에 `authorization-type` 입력 속성을 설정하고 이미 존재하는 Lambda 권한 부여자의 `id` 속성 값에 `authorizer-id` 입력 속성을 설정합니다. 참조된 Lambda 권한 부여자는 `TOKEN` 또는 `REQUEST` 유형일 수 있습니다. Lambda 권한 부여자를 생성하는 방법은 the section called “Lambda 권한 부여자 사용” 단원을 참조하세요.

Amazon Cognito 사용자 풀을 사용하여 API 메서드에 대한 액세스 권한을 부여하려면 `COGNITO_USER_POOLS`에 `authorization-type` 입력 속성을 설정하고 이미 생성된 `COGNITO_USER_POOLS` 권한 부여자의 `id` 속성 값에 `authorizer-id` 입력 속성을 설정합니다. Amazon Cognito 사용자 풀 권한 부여자 생성에 대한 자세한 내용은 the section called “REST API에 대한 권한 부여자로 Amazon Cognito 사용자 풀 사용” 단원을 참조하세요.

메서드 요청 검증 설정

API 메서드 요청을 설정할 때 요청 검증을 활성화할 수 있습니다. 먼저 다음과 같이 요청 검사기를 생성해야 합니다.

```
aws apigateway create-request-validator \
    --rest-api-id 7zw9uyk9kl \
    --name bodyOnlyValidator \
    --validate-request-body  \
```

```
    --no-validate-request-parameters
```

이 CLI 명령은 본문 전용 요청 검사기를 생성합니다. 예제 출력 내용은 다음과 같습니다.

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

이 요청 검사기를 통해 다음과 같이 요청 검증을 메서드 요청 설정의 일부로 활성화할 수 있습니다.

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

요청 파라미터는 요청 검증에 포함되려면 필수로 선언되어야 합니다. 해당 페이지에 대한 쿼리 문자열 파라미터가 요청 검증에 사용되는 경우 이전 예제의 `request-parameters` 맵은 `'{"method.request.querystring.type": false, "method.request.querystring.page":true}'`로 지정되어야 합니다.

API Gateway에서 메서드 응답 설정

API 메서드 응답은 클라이언트가 수신할 API 메서드 요청의 출력을 캡슐화합니다. 출력 데이터에는 HTTP 상태 코드, 일부 헤더, 그리고 때로는 본문이 포함됩니다.

비 프록시 통합의 경우, 지정된 응답 파라미터 및 본문은 연결된 통합 응답 데이터로부터 매핑되거나 매핑에 따라 특정 정적 값이 할당될 수 있습니다. 이 매핑은 통합 응답에 지정되어 있습니다. 매핑은 통합 응답을 있는 그대로 통과하는 동일한 변환일 수 있습니다.

프록시 통합의 경우, API Gateway는 백엔드 응답을 메서드 응답으로 자동 전달합니다. API 메서드 응답을 설정할 필요는 없습니다. 그러나 Lambda 프록시 통합의 경우, 통합 응답을 메서드 응답에 성공적으로 매핑하려면 Lambda 함수는 API Gateway에 대해 이 출력 형식의 결과를 반환해야 합니다.

프로그래밍 측면에서 메서드 응답 설정은 API Gateway의 MethodResponse리소스를 생성하고 StatusCode, Response Parameter, ResponseModels의 속성을 설정하는 것과 같습니다.

API 메서드에 대해 상태 코드를 설정할 때는 예기치 않은 상태 코드의 모든 통합 응답을 처리하기 위해 상태 코드 하나를 기본값으로 선택해야 합니다. 500을 기본값으로 설정하는 것이 타당합니다. 왜냐하면 이것은 그렇지 않았다면 매핑되지 않았을 응답을 서버 측 오류로 캐스팅하는 것에 해당하기 때문입니다. 지침을 제공하기 위해, API Gateway 콘솔은 200 응답을 기본값으로 설정합니다. 그러나 이를 500 응답으로 재설정할 수 있습니다.

메서드 응답을 설정하려면 메서드 요청을 생성했어야 합니다.

메서드 응답 상태 코드 설정

메서드 응답의 상태 코드는 응답의 유형을 정의합니다. 예를 들어 200, 400 및 500 응답은 각각 성공, 클라이언트 측 오류 및 서버 측 오류 응답을 나타냅니다.

메서드 응답 상태 코드를 설정하려면 HTTP 상태 코드에 `statusCode` 속성을 설정합니다. 다음 AWS CLI 명령은 200의 메서드 응답을 생성합니다.

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

메서드 응답 파라미터 설정

메서드 응답 파라미터는 연결된 메서드 요청에 대한 응답으로 클라이언트가 어떤 헤더를 수신하는지 정의합니다. 응답 파라미터는 API Gateway가 API 메서드의 통합 응답에 미리 지정된 매핑에 따라 통합 응답 파라미터를 매핑하는 대상을 지정하기도 합니다.

메서드 응답 파라미터를 설정하려면 responseParameters의 `MethodResponse` 맵에 `"{parameter-name}":"{boolean}"` 형식의 키 값 페어를 추가합니다. 다음 CLI 명령은 헤더 설정의 my-header 예를 보여줍니다.

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

```
        --status-code 200  \
        --response-parameters method.response.header.my-header=false
```

메서드 응답 모델 설정

메서드 응답 모델은 메서드 응답 본문의 형식을 정의합니다. 응답 모델을 설정하기 전에 먼저 API Gateway에 모델을 생성해야 합니다. 이를 위해 `create-model` 명령을 호출할 수 있습니다. 다음 예시에서는 `PetStorePet` 메서드 요청에 대한 응답의 본문을 설명하기 위해 `GET /pets/{petId}` 모델을 생성하는 방법을 보여줍니다.

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

그 결과 API Gateway `Model` 리소스가 생성됩니다.

페이로드 형식을 정의하도록 메서드 응답 모델을 설정하려면 리소스 맵에 “application/json”:PetStorePet” 키-값 쌍을 추가하십시오. `requestModelsMethodResponse` 의 다음 AWS CLI 명령은 이 작업을 수행하는 방법을 보여줍니다`put-method-response`.

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

메서드 응답 모델 설정은 강력한 형식의 API용 SDK를 생성할 때 필요합니다. 이를 통해 출력은 Java 또는 Objective-C의 적절한 클래스에 캐스팅됩니다.

API Gateway 콘솔을 사용하여 메서드 설정

API 메서드를 설정하기 전에 다음 사항을 확인합니다.

- API Gateway에 사용 가능한 메서드가 있어야 합니다. 자습서: HTTP 비 프록시 통합을 사용하여 REST API 빌드의 지침을 따르세요.
- 메서드가 Lambda 함수와 통신하게 하려면 이미 IAM에서 Lambda 호출 역할과 Lambda 실행 역할을 생성했어야 합니다. 또한 AWS Lambda에서 메서드가 통신할 Lambda 함수를 생성했어야 합니다. 역할 및 함수를 생성하려면 Lambda 통합을 사용하여 API Gateway REST API 구축의 Lambda 비 프록시 통합을 위한 Lambda 함수 만들기 지침을 따르세요.
- 메서드가 HTTP 또는 HTTP 프록시 통합과 통신하도록 하려면 해당 메서드와 통신할 HTTP 엔드포인트 URL을 이미 생성하고 그에 대한 액세스 권한을 확보했어야 합니다.
- HTTP 및 HTTP 프록시 엔드포인트의 인증서를 API Gateway에서 지원하는지 확인하세요. 자세한 내용은 HTTP 및 HTTP 프록시 통합에 대한 API Gateway 지원 인증 기관 단원을 참조하세요.

주제

- API Gateway 콘솔에서 API Gateway 메서드 요청 설정
- API Gateway 콘솔에서 API Gateway 메서드 응답 설정

API Gateway 콘솔에서 API Gateway 메서드 요청 설정

API Gateway 콘솔을 사용하여 API의 메서드 요청/응답을 지정하고 메서드의 요청 승인 방식을 구성하려면 다음 지침을 따르세요.

> **Note**
>
> 이 지침에서는 API Gateway 콘솔을 사용하여 API 통합 요청 설정 단원의 단계를 이미 완료한 것으로 가정합니다. 그 절차는 Lambda 통합을 사용하여 API Gateway REST API 구축에 제공된 설명을 보충하는 용도로 사용하는 것이 좋습니다.

1. 리소스 창에서 방법을 선택한 다음 메서드 요청 탭을 선택합니다.

2. 메서드 요청 설정 섹션에서 편집을 선택합니다.
3. 권한 부여에서 사용 가능한 권한 부여자를 선택합니다.

   a. 모든 사용자에게 메서드에 대한 오픈 액세스를 활성화하려면 없음을 선택합니다. 기본값 설정이 변경되지 않은 경우 이 단계는 건너뛸 수 있습니다.

   b. IAM 권한을 사용하여 메서드에 대한 클라이언트 액세스 권한을 제어하려면 AWS_IAM을 선택합니다. 이 선택을 통해 올바른 IAM 정책이 연결된 IAM 역할의 사용자만 이 메서드를 호출할 수 있습니다.

      IAM 역할을 생성하려면 다음과 같은 형식의 액세스 정책을 지정합니다.

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

      이 액세스 정책에서 *resource-statement*는 인증 설정(Authorization Settings) 섹션에 있는 ARN 필드의 값입니다. IAM 권한 설정에 대한 자세한 내용은 IAM 권한을 사용하여 API에 대한 액세스 제어를 참조하세요.

      IAM 역할을 생성하려면 Lambda 통합을 사용하여 API Gateway REST API 구축의 Lambda 함수 생성 섹션에 나와 있는 "Lambda 호출 역할 및 정책 생성 방법" 및 "Lambda 실행 역할 및 정책 생성 방법"의 지침을 적절히 적용합니다.

   c. Lambda 권한 부여자를 사용하려면 토큰 또는 요청 권한 부여자를 선택합니다. 이 선택이 드롭다운 메뉴에 표시되도록 하기 위해서는 Lambda 권한 부여자를 생성합니다. Lambda 권한 부여자를 생성하는 방법에 대한 자세한 내용은 API Gateway Lambda 권한 부여자 사용 단원을 참조하세요.

   d. Amazon Cognito 사용자 풀을 사용하려면 Cognito 사용자 풀 권한 부여자에서 사용할 수 있는 사용자 풀을 선택합니다. 이 선택이 드롭다운 메뉴에 표시되도록 하기 위해서는 Amazon

Cognito에 사용자 풀을 생성하고 API Gateway에 Amazon Cognito 사용자 풀 권한 부여자를 생성합니다. Amazon Cognito 사용자 풀 권한 부여자를 생성하는 방법에 대한 자세한 내용은 Amazon Cognito 사용자 풀을 권한 부여자로 사용하여 REST API에 대한 액세스 제어 단원을 참조하세요.

4. 요청 검증을 지정하려면 요청 검사기 드롭다운 메뉴에서 값을 선택합니다. 요청 검증을 비활성화하려면 없음을 선택합니다. 각 옵션에 대한 자세한 내용은 API Gateway에서 요청 확인 사용를 참조하십시오.
5. API 키를 요구하려면 API 키가 필요함을 선택합니다. API는 활성화되면 사용량 계획에서 클라이언트 트래픽을 조절하는 데 사용됩니다.
6. (선택 사항) API Gateway에서 생성된 이 API의 Java SDK에서 작업 이름을 할당하려면 작업 이름에 이름을 입력합니다. 예를 들어 `GET /pets/{petId}`의 메서드 요청에 대해 이에 상응하는 Java SDK 작업 이름은 기본적으로 `GetPetsPetId`입니다. 이 이름은 메서드의 HTTP 동사(`GET`)와 리소스 경로 변수 이름(`Pets` 및 `PetId`)으로부터 구성됩니다. 작업 이름을 `getPetById`로 설정하면 SDK 작업 이름은 `GetPetById`가 됩니다.
7. 메서드에 쿼리 문자열 파라미터를 추가하려면 다음을 수행합니다.

   a. URL 쿼리 문자열 파라미터를 선택한 다음, 쿼리 문자열 추가를 선택합니다.

   b. 이름에 쿼리 문자열 파라미터의 이름을 입력합니다.

   c. 새로 생성된 쿼리 문자열 파라미터가 요청 검증에 사용되는 경우 필수를 선택합니다. 요청 검증에 대한 자세한 내용은 API Gateway에서 요청 확인 사용를 참조하십시오.

   d. 새로 생성된 쿼리 문자열 파라미터가 캐싱 키의 일부로 사용되는 경우 캐싱을 선택합니다. 캐싱에 대한 자세한 정보는 메서드/통합 파라미터를 캐시 키로 사용하여 캐시된 응답 인덱싱을 참조하십시오.

   쿼리 문자열 파라미터를 제거하려면 제거를 선택합니다.
8. 메서드에 헤더 파라미터를 추가하려면 다음을 수행합니다.

   a. HTTP 요청 헤더를 선택한 다음, 헤더 추가를 선택합니다.

   b. 이름에 헤더 이름을 입력합니다.

   c. 새로 생성된 헤더가 요청 검증에 사용되는 경우 필수를 선택합니다. 요청 검증에 대한 자세한 내용은 API Gateway에서 요청 확인 사용를 참조하십시오.

   d. 새로 생성된 헤더가 캐싱 키의 일부로 사용되는 경우 캐싱을 선택합니다. 캐싱에 대한 자세한 정보는 메서드/통합 파라미터를 캐시 키로 사용하여 캐시된 응답 인덱싱을 참조하십시오.

헤더를 제거하려면 제거를 선택합니다.

9. POST, PUT 또는 PATCH HTTP 동사로 메서드 요청의 페이로드 형식을 선언하려면 요청 본문을 선택하고 다음과 같이 합니다.

   a. 모델 추가를 선택합니다.

   b. 콘텐츠 유형에 MIME 유형(예: `application/json`)을 입력합니다.

   c. 모델의 경우 드롭다운 메뉴에서 모델을 선택합니다. API에 대해 현재 사용할 수 있는 모델에는 기본 `Empty` 및 `Error` 모델뿐 아니라 이미 생성해 API의 모델 모음에 추가한 모든 모델도 포함됩니다. 모델 생성에 대한 자세한 내용은 데이터 모델 이해을 참조하십시오.

   > **Note**
   >
   > 모델은 클라이언트에게 페이로드의 예상 데이터 형식을 알려주는 데 유용합니다. 스켈레톤 매핑 템플릿을 생성하는 것이 도움이 됩니다. Java, C#, Objective-C, Swift 등의 언어에서 API의 강력한 형식 SDK를 생성하는 것이 중요합니다. 이는 페이로드에 대해 요청 검증이 활성화된 경우에만 필요합니다.

10. 저장을 선택합니다.

API Gateway 콘솔에서 API Gateway 메서드 응답 설정

API 메서드는 응답이 하나 이상일 수 있습니다. 각 응답은 HTTP 상태 코드로 인덱싱됩니다. 기본적으로 API Gateway 콘솔은 `200` 응답을 메서드 응답에 추가합니다. 예를 들어 메서드가 `201`을 대신 반환하도록 응답을 수정할 수 있습니다. 또한 다른 응답을 추가할 수도 있는데, 예를 들면 액세스 거부의 경우 `409`를, 초기화되지 않은 단계 변수가 사용된 경우 `500`을 추가할 수 있습니다.

API Gateway 콘솔을 사용하여 API 메서드에 대한 응답을 수정, 삭제 또는 추가하려면 다음 지침을 따르세요.

1. 리소스 창에서 방법을 선택한 다음 메서드 응답 탭을 선택합니다. 탭을 표시하려면 오른쪽 화살표 버튼을 선택해야 할 수도 있습니다.
2. 메서드 응답 설정 섹션에서 응답 생성을 선택합니다.
3. HTTP 상태 코드에 HTTP 상태 코드(예: `200`, `400`, `500`)를 입력합니다.

백엔드 반환 응답에 그에 상응하는 메서드 응답이 정의되어 있지 않으면 API Gateway는 클라이언트에게 응답을 반환하는 데 실패합니다. 그 대신에 `500 Internal server error` 오류 응답을 반환합니다.

4. 헤더 추가(Add header)를 선택합니다.
5. 헤더 이름에 이름을 입력합니다.

   백엔드에서 클라이언트로 헤더를 반환하려면 메서드 응답에 헤더를 추가합니다.
6. 메서드 응답 본문의 형식을 정의하려면 모델 추가를 선택합니다.

   콘텐츠 유형에 응답 페이로드의 미디어 유형을 입력하고 모델 드롭다운 메뉴에서 모델을 선택합니다.
7. 저장을 선택합니다.

기존 응답을 수정하려면 메서드 응답으로 이동한 다음 편집을 선택합니다. HTTP 상태 코드를 변경하려면 삭제를 선택하고 새 메서드 응답을 생성합니다.

백엔드에서 반환되는 응답마다 메서드 응답으로 구성된 호환 가능한 응답이 있어야 합니다. 그러나 클라이언트에게 반환되기 전에 백엔드로부터 오는 결과를 메서드 응답에 매핑하지 않으면 메서드 응답 헤더 및 페이로드 모델을 구성하는 것은 선택 사항입니다. 또한 메서드 응답 페이로드 모델은 강력한 형식의 API용 SDK를 생성하는 경우에 중요합니다.

API Gateway의 REST API에 대한 액세스 제어 및 관리

API Gateway는 API 액세스 제어 및 관리에 다중 메커니즘을 지원합니다.

다음 메커니즘은 인증 및 권한 부여에 사용할 수 있습니다.

- 리소스 정책으로 리소스 기반 정책을 만들어 특정 소스의 IP 주소 또는 VPC 종단점의 API 및 메서드 액세스 권한을 부여하거나 거부할 수 있습니다. 자세한 내용은 the section called “API Gateway 리소스 정책 사용” 섹션을 참조하세요.
- 표준 AWS IAM 역할 및 정책은 전체 API 또는 개별 메서드에 적용할 수 있는 유연하고 강력한 액세스 제어를 제공합니다. IAM 역할 및 정책을 사용하여 API를 생성하고 관리할 수 있는 사용자와 API를 호출할 수 있는 사용자를 관리할 수 있습니다. 자세한 내용은 the section called “IAM 권한 사용” 섹션을 참조하세요.
- IAM 태그는 액세스 제어를 위해 IAM 정책과 함께 사용할 수 있습니다. 자세한 내용은 the section called “속성 기반 액세스 제어” 섹션을 참조하세요.

- 인터페이스 VPC 종단점에 대한 엔드포인트 정책을 통해 프라이빗 API의 보안을 향상시키기 위해 인터페이스 VPC 종단점에 IAM 리소스 정책을 연결할 수 있습니다. 자세한 내용은 the section called "프라이빗 API에 대한 VPC 종단점 정책 사용" 섹션을 참조하세요.
- Lambda 권한 부여자는 보유자 토큰 인증과 헤더, 경로, 쿼리 문자열, 단계 변수, 맥락 변수 요청 파라미터에 서술된 정보를 이용하여 REST API 메서드에 대한 액세스를 제어하는 Lambda 함수입니다. Lambda 권한 부여자는 REST API 메서드를 호출할 수 있는 사람을 제어하는 데 사용됩니다. 자세한 내용은 the section called "Lambda 권한 부여자 사용" 섹션을 참조하세요.
- Amazon Cognito 사용자 풀로 REST API에 대해 사용자 지정이 가능한 인증 및 권한 부여 솔루션을 만들 수 있습니다. Amazon Cognito 사용자 풀은 REST API 메서드를 호출할 수 있는 사람을 제어하는 데 사용됩니다. 자세한 내용은 the section called "REST API에 대한 권한 부여자로 Amazon Cognito 사용자 풀 사용" 섹션을 참조하세요.

다음 메커니즘은 액세스 제어와 관련된 다른 작업을 수행할 때 사용할 수 있습니다.

- CORS(Cross-origin 리소스 공유)로 REST API가 교차 도메인 리소스 요청에 응답하는 방식을 제어할 수 있습니다. 자세한 내용은 the section called "CORS" 섹션을 참조하세요.
- 클라이언트 측 SSL 인증서를 사용하여 백엔드 시스템에 대한 HTTP 요청이 API Gateway에서 시작된 것인지 확인할 수 있습니다. 자세한 내용은 the section called "클라이언트 인증서" 섹션을 참조하세요.
- AWS WAF는 일반적인 웹 익스플로잇으로부터 API Gateway API를 보호하기 위해 사용할 수 있습니다. 자세한 내용은 the section called "AWS WAF" 섹션을 참조하세요.

다음 메커니즘은 권한이 있는 클라이언트에게 부여한 액세스 권한을 추적하고 제한하는 데 사용할 수 있습니다.

- 사용량 계획을 통해 고객에게 API 키를 제공할 수 있으며 그 후에 각 API 키에 대하여 API 단계와 메서드를 추적하고 사용량을 제한할 수 있습니다. 자세한 내용은 the section called "사용량 계획" 섹션을 참조하세요.

API Gateway 리소스 정책을 사용하여 API에 대한 액세스 제어

Amazon API Gateway 리소스 정책은 지정된 보안 주체(보통 IAM 역할 또는 그룹)가 API를 호출할 수 있는지 여부를 제어하기 위해 API에 연결하는 JSON 정책 문서입니다. API Gateway 리소스 정책을 사용하여 다음과 같은 엔터티의 안전한 API 호출을 허용할 수 있습니다.

- 지정된 AWS 계정의 사용자

- 지정된 소스 IP 주소 범위 또는 CIDR 블록.
- 지정된 가상 사설 클라우드(VPC) 또는 VPC 종단점(계정 무관).

프라이빗, 엣지 최적화, 리전 등 API Gateway의 모든 API 엔드포인트 유형에 대해 리소스 정책을 사용할 수 있습니다.

프라이빗 API의 경우 VPC 종단점 정책과 함께 리소스 정책을 사용하여 어떤 주체가 어떤 리소스 및 작업에 대한 액세스를 가질지를 제어할 수 있습니다. 자세한 설명은 the section called “프라이빗 API에 대한 VPC 종단점 정책 사용” 섹션을 참조하세요.

AWS Management Console, AWS CLI 또는 AWS SDK를 사용하여 API에 리소스 정책을 연결할 수 있습니다.

API Gateway 리소스 정책은 IAM 자격 증명 기반 정책과 다릅니다. IAM 정책은 IAM 엔터티(사용자, 그룹 또는 역할)에 연결되어 이들 엔터티가 어떤 리소스에 대해 어떤 조치를 취할 수 있는지 정의합니다. API Gateway 리소스 정책은 리소스에 연결됩니다. 자격 증명 기반 정책과 리소스 정책 간의 차이점에 대한 자세한 내용은 자격 증명 기반 정책 및 리소스 기반 정책 단원을 참조하세요.

API Gateway 리소스 정책을 IAM 정책과 함께 사용할 수 있습니다.

주제

- Amazon API Gateway에 대한 액세스 정책 언어 개요
- API Gateway 리소스 정책이 권한 부여 워크플로우에 미치는 영향
- API Gateway 리소스 정책 예제
- API Gateway 리소스 정책을 생성하여 API에 연결
- AWS API Gateway 리소스 정책에서 사용할 수 있는 조건 키

Amazon API Gateway에 대한 액세스 정책 언어 개요

이 페이지에서는 Amazon API Gateway 리소스 정책에 사용되는 기본 요소를 설명합니다.

IAM 정책과 동일한 구문을 사용하여 리소스 정책이 지정됩니다. 전체 정책 언어에 대한 정보는 IAM 사용 설명서의 IAM 정책 개요 및 AWS Identity and Access Management 정책 참조를 참조하세요.

AWS 서비스가 특정 요청의 허용 또는 거부 여부를 결정하는 방법에 대한 자세한 내용은 요청의 허용 또는 거부 여부 결정을 참조하십시오.

액세스 정책의 공통 요소

가장 기본적인 경우, 리소스 정책에는 다음 요소가 포함되어 있습니다.

- 리소스 – API는 권한을 허용 또는 거부할 수 있는 Amazon API Gateway 리소스입니다. 정책에서는 Amazon 리소스 이름(ARN)을 사용하여 리소스를 식별해야 합니다. 리소스 정책을 저장할 때 API Gateway가 전체 ARN으로 자동으로 확장하는 약식 구문을 사용할 수도 있습니다. 자세한 내용은 API Gateway 리소스 정책 예제를 참조하세요.

  전체 `Resource` 요소의 형식은 API Gateway의 API 실행 권한 리소스 형식 단원을 참조하세요.
- 작업 - 각 리소스에 대해 Amazon API Gateway는 작업의 집합을 지원합니다. 작업 키워드를 사용하여 허용(또는 거부)할 리소스 작업을 식별합니다.

  예를 들어 `execute-api:Invoke` 권한은 사용자가 클라이언트 요청 시 API를 호출할 수 있도록 허용합니다.

  `Action` 요소의 형식은 API Gateway의 API 실행 권한 작업 형식 단원을 참조하세요.
- 효과 - 사용자가 특정 작업을 요청하는 경우의 효과입니다. 이는 `Allow` 또는 `Deny` 중에 하나가 될 수 있습니다. 다른 정책에서 액세스 권한을 부여하는 경우라도 사용자가 해당 리소스에 액세스할 수 없도록 하기 위해 리소스에 대한 권한을 명시적으로 거부할 수도 있습니다.

  > **Note**
  >
  > "암시적 거부"는 "기본 거부"와 동일합니다.
  > “암시적 거부”는 “명시적 거부”와 다릅니다. 자세한 내용은 기본 거부와 명시적 거부의 차이 단원을 참조하세요.
- 보안 주체 - 문에서의 작업 및 리소스에 액세스할 수 있는 계정 또는 사용자입니다. 리소스 정책에서 보안 주체는 이 권한을 수신하는 사용자나 계정입니다.

다음의 리소스 정책 예시는 이전의 공통 정책 요소를 나타냅니다. 이 정책은 소스 IP 주소가 *`123.4.5.6/24`* 주소 블록에 속하는 사용자에게 지정된 *`region`*의 지정된 *`account-id`*에서 API에 대한 액세스 권한을 부여합니다. 이 정책은 사용자의 소스 IP가 해당 범위 내에 있지 않으면 API에 대한 모든 액세스를 거부합니다.

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

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

API Gateway 리소스 정책이 권한 부여 워크플로우에 미치는 영향

API Gateway가 API에 연결된 리소스 정책을 평가할 때 그 결과는, 다음 단원의 흐름 차트에 서명된 것처럼, APi에 대하여 정의한 인증 유형의 영향을 받습니다.

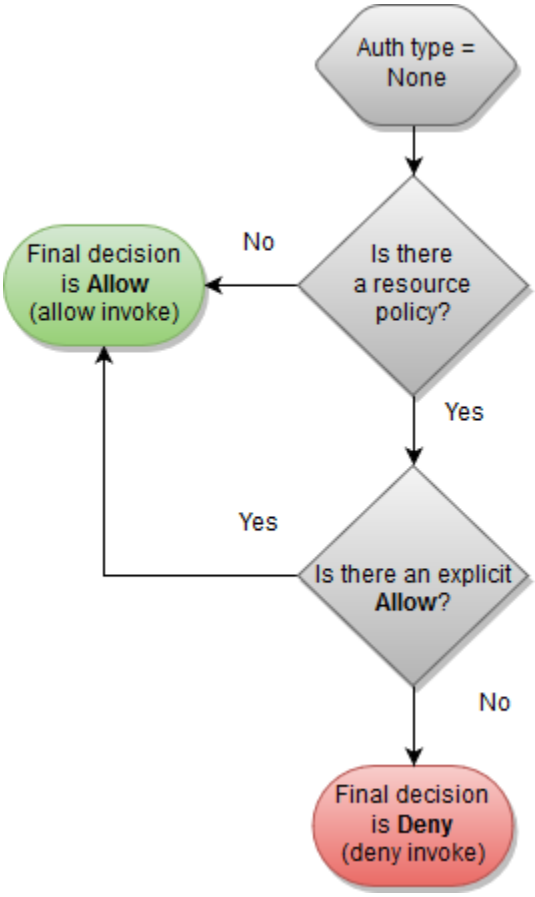
주제

- API Gateway 리소스 정책만
- Lambda 권한 부여자 및 리소스 정책
- IAM 인증 및 리소스 정책
- Amazon Cognito 인증 및 리소스 정책
- 정책 평가 결과표

API Gateway 리소스 정책만

이 워크플로우에서 API Gateway 리소스 정책은 API에 연결되지만 API에 대하여 정의되는 인증 유형은 없습니다. 정책 평가를 위해서는 호출자의 인바운드 기준에 따른 명시적 허용이 필요합니다. 묵시적 거부 또는 명시적 거부는 호출자에 대한 거부로 이어집니다.

다음은 이러한 리소스 정책의 예입니다.

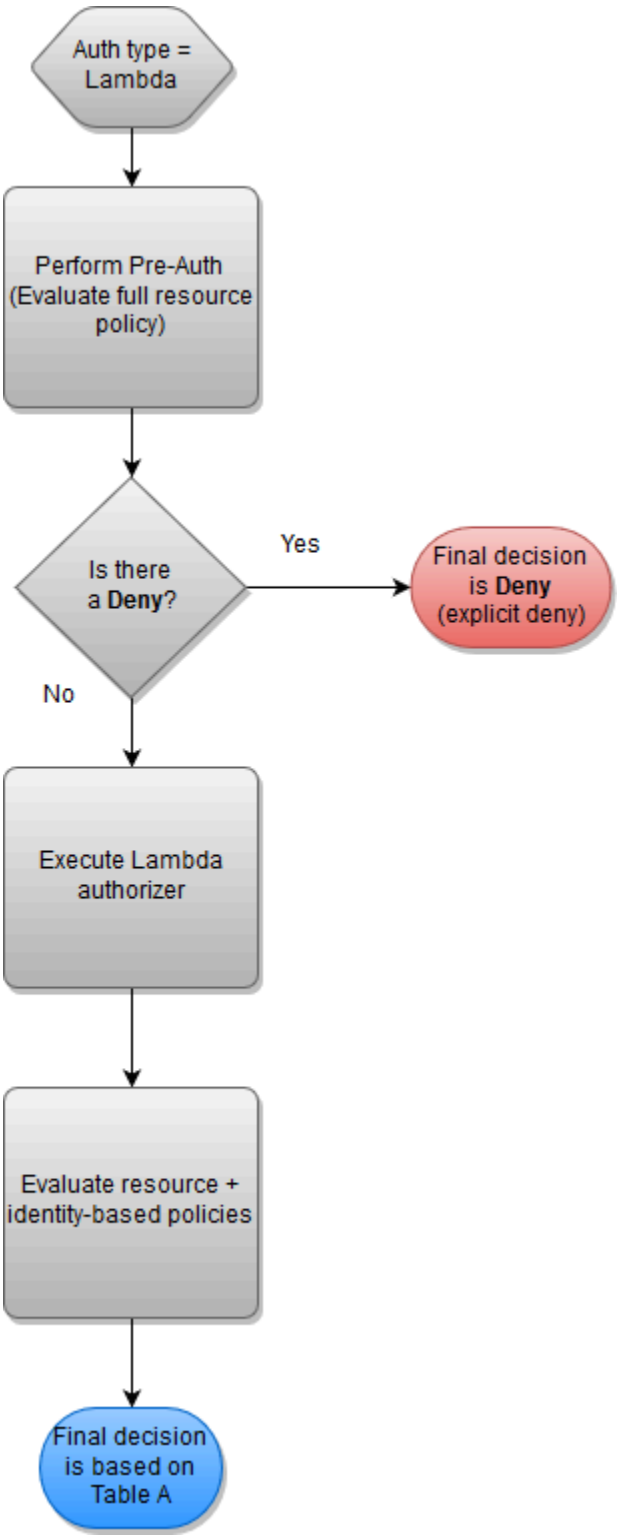



```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": "*",
            "Action": "execute-api:Invoke",
            "Resource": "arn:aws:execute-api:region:account-id:api-id/",
            "Condition": {
                "IpAddress": {
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

Lambda 권한 부여자 및 리소스 정책

이 워크플로우에서 Lambda 권한 부여자는 리소스 정책과 함께 API에 대하여 구성됩니다. 리소스 정책은 2단계로 평가됩니다. Lambda 권한 부여자를 호출하기 전에 API Gateway는 먼저 정책을 평가하고 명시적 거부를 확인합니다. 명시적 거부가 확인되면 호출자는 즉시 액세스를 거부당합니다. 그렇지 않으면 Lambda 권한 보유자가 호출되어 리소스 정책과 결합하여 평가되는 정책 문서를 반환합니다. 결과는 테이블 A(이 항목의 끝 부분)에 따라 결정됩니다.

다음의 리소스 정책 예제는 그 VPC 엔드포인트 ID가 *vpce-1a2b3c4d*인 VPC 엔드포인트로부터의 호출만을 허용합니다. “인증 전” 평가 과정 중에는 예제에 나온 VPC 종단점에서의 호출만이 다음 단계로 넘어가 Lambda 권한 부여자를 평가하는 것이 허용됩니다. 나머지 모든 호출은 차단됩니다.



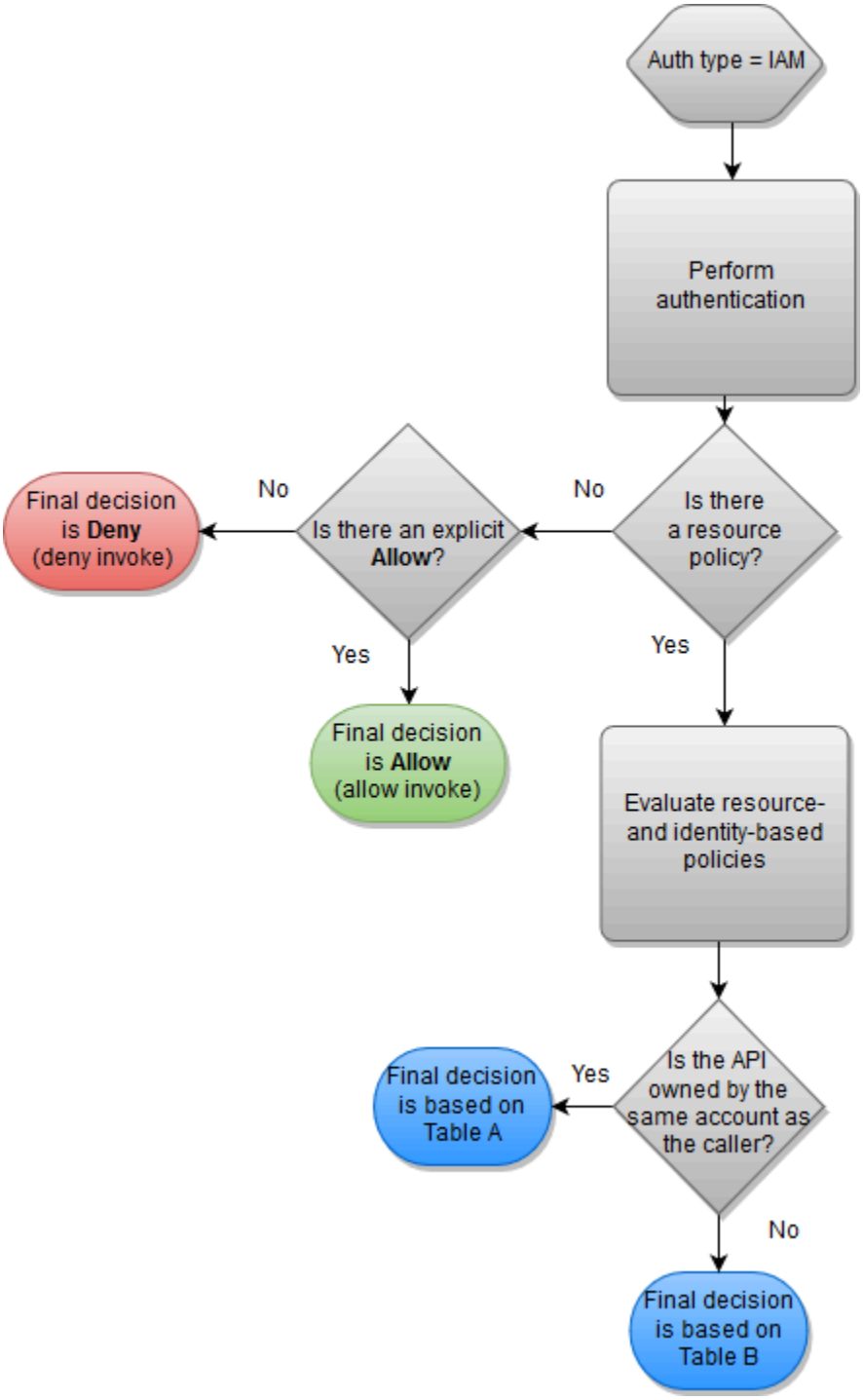

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
```

```
            "Effect": "Deny",
            "Principal": "*",
            "Action": "execute-api:Invoke",
            "Resource": [
                "arn:aws:execute-api:region:account-id:api-id/"
            ],
            "Condition" : {
                "StringNotEquals": {
                    "aws:SourceVpce": "vpce-1a2b3c4d"
                }
            }
        }
    ]
}
```

IAM 인증 및 리소스 정책

이 워크플로에서 리소스 정책 외에도 API에 대한 IAM 인증을 구성합니다. IAM 서비스로 사용자를 인증한 후에는 API가 사용자에게 연결된 정책과 리소스 정책을 모두 평가합니다. 그 결과는 호출자가 동일한 AWS 계정에 있는지 또는 API 소유자의 별도 AWS 계정에 있는지에 따라 달라집니다.

호출자와 API 소유자가 각기 다른 계정에 있는 경우, IAM 정책과 리소스 정책 모두 호출자가 진행하도록 명시적으로 허용합니다. (이 항목의 끝 부분에 있는 테이블 B 를 참조). 그러나 호출자와 API 소유자가 동일한 AWS 계정에 있는 경우, IAM 사용자 정책 또는 리소스 정책 중 하나에서 호출자가 진행하도록 명시적으로 허용합니다. 아래 테이블 A를 참조하십시오.



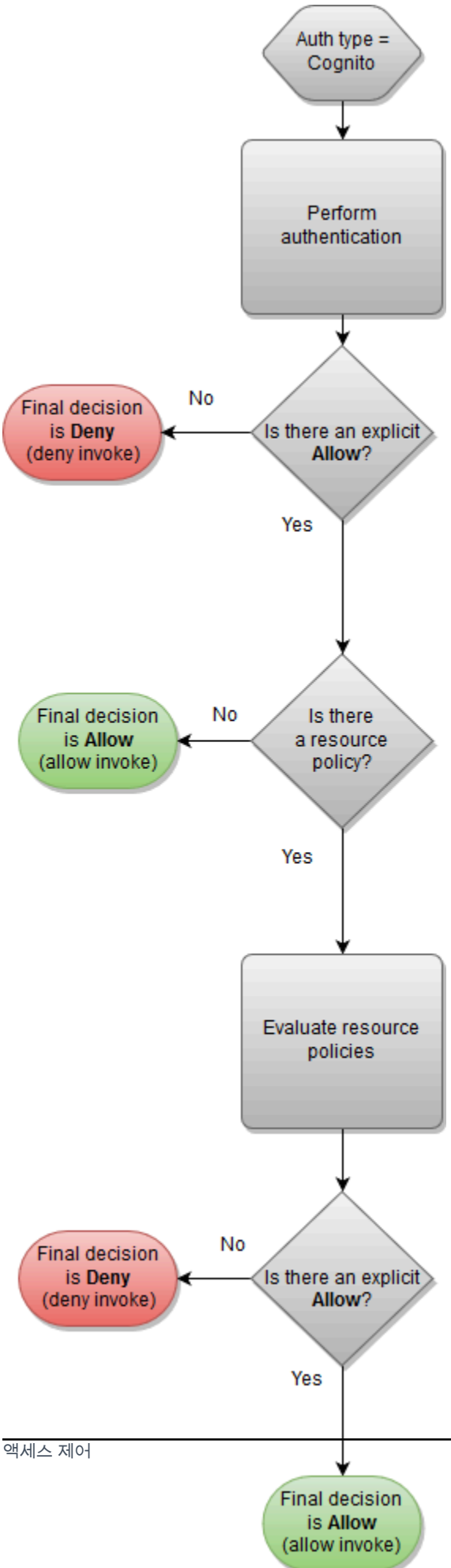
다음은 교차 계정 리소스 정책의 예입니다. IAM 정책에 허용 효과가 포함되어 있다고 가정할 때, 이 리소스 정책은 VPC ID가 *vpc-2f09a348*인 VPC로부터의 호출만을 허용합니다 (이 항목의 끝 부분에 있는 테이블 B 를 참조).

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

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

Amazon Cognito 인증 및 리소스 정책

이 워크플로에서는 리소스 정책 외에 API에 대해 Amazon Cognito 사용자 풀이 구성됩니다. API Gateway는 먼저 Amazon Cognito를 통해 호출자에게 권한을 부여하려 합니다. 이는 보통 호출자가 제공하는 JWT 토큰을 통해 수행됩니다. 인증에 성공하면, 리소스 정책이 독립적으로 평가되고 명시적 허용이 필요해집니다. 거부 또는 “허용도 거부도 아님”인 경우에는 거부됩니다. 다음은 Amazon Cognito 사용자 풀과 함께 사용될 수 있는 리소스 정책의 예제입니다.


Auth type = Cognito
Perform authentication
Is there an explicit Allow?
No
Final decision is Deny (deny invoke)
Yes
Is there a resource policy?
No
Final decision is Allow (allow invoke)
Yes
Evaluate resource policies
Is there an explicit Allow?
No
Final decision is Deny (deny invoke)
Yes
Final decision is Allow (allow invoke)

다음은 Amazon Cognito 인증 토큰에 허용이 포함되어 있다는 가정하에 지정된 소스 IP로부터의 호출만을 허용하는 리소스 정책의 예제입니다 (이 항목의 끝 부분에 있는 테이블 A 참조).

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

정책 평가 결과표

테이블 A에는 API Gateway API에 대한 액세스가 IAM 정책(또는 Lambda 또는 Amazon Cognito 사용자 풀 권한 부여자) 및 API Gateway 리소스 정책 모두에 의해 제어되고, 이러한 두 정책이 모두 동일한 AWS 계정에 있을 때 결과로 수행되는 동작이 나와 있습니다.

테이블 A: 계정 A가 자신이 소유한 API를 호출

IAM 정책(또는 Lambda 또는 Amazon Cognito 사용자 풀 권한 부여자)	API Gateway 리소스 정책	결과적 동작
허용	허용	허용
허용	허용도 거부도 아님	허용
허용	거부	명시적 거부
허용도 거부도 아님	허용	허용
허용도 거부도 아님	허용도 거부도 아님	암시적 거부

IAM 정책(또는 Lambda 또는 Amazon Cognito 사용자 풀 권한 부여자)	API Gateway 리소스 정책	결과적 동작
허용도 거부도 아님	거부	명시적 거부
거부	허용	명시적 거부
거부	허용도 거부도 아님	명시적 거부
거부	거부	명시적 거부

테이블 B에는 API Gateway API에 대한 액세스가 IAM 정책(또는 Lambda 또는 Amazon Cognito 사용자 풀 권한 부여자) 및 API Gateway 리소스 정책 모두에 의해 제어되고, 이 두 정책이 서로 다른 AWS 계정에 있을 때 결과로 수행되는 동작이 나와 있습니다. 허용되지도 거부되지도 않으면, 교차 계정 액세스가 거부됩니다. 왜냐하면 교차 계정 액세스를 위해서는 리소스 정책과 IAM 정책(또는 Lambda 또는 Amazon Cognito 사용자 풀 권한 부여자) 모두에서 액세스를 명시적으로 허용해야 하기 때문입니다.

테이블 B: 계정 B가

IAM 정책(또는 Lambda 또는 Amazon Cognito 사용자 풀 권한 부여자)	API Gateway 리소스 정책	결과적 동작
허용	허용	허용
허용	허용도 거부도 아님	암시적 거부
허용	거부	명시적 거부
허용도 거부도 아님	허용	암시적 거부
허용도 거부도 아님	허용도 거부도 아님	암시적 거부
허용도 거부도 아님	거부	명시적 거부
거부	허용	명시적 거부
거부	허용도 거부도 아님	명시적 거부

IAM 정책(또는 Lambda 또는 Amazon Cognito 사용자 풀 권한 부여자)	API Gateway 리소스 정책	결과적 동작
거부	거부	명시적 거부

API Gateway 리소스 정책 예제

이 페이지에서는 API Gateway 리소스 정책의 일반적인 사용 사례에 대한 몇 가지 예제를 제시합니다.

다음 예제 정책은 간소화된 구문을 사용하여 API 리소스를 지정합니다. 이 간소화된 구문은 전체 Amazon 리소스 이름(ARN)을 지정하는 대신 API 리소스를 참조할 수 있는 약식 방법입니다. API Gateway는 정책을 저장할 때 약어 구문을 전체 ARN으로 변환합니다. 예를 들어 리소스 정책에서 `execute-api:/stage-name/GET/pets` 리소스를 지정할 수 있습니다. API Gateway는 리소스 정책을 저장할 때 이 리소스를 `arn:aws:execute-api:us-east-2:123456789012:aabbccddee/stage-name/GET/pets`로 변환합니다. API Gateway는 현재 지역, AWS 계정 ID, 리소스 정책이 연결된 REST API의 ID를 사용하여 전체 ARN을 구축합니다. `execute-api:/*`를 사용하여 현재 API의 모든 단계, 메서드 및 경로를 나타낼 수 있습니다. 액세스 정책 언어에 대한 자세한 내용은 Amazon API Gateway에 대한 액세스 정책 언어 개요 단원을 참조하세요.

주제

- 예: 다른 AWS 계정의 역할이 API를 사용하도록 허용
- 예제: 소스 IP 주소 또는 범위에 따라 API 트래픽 거부
- 예: 프라이빗 API를 사용할 때 소스 IP 주소 또는 범위를 기반으로 API 트래픽 거부
- 예제: 소스 VPC 또는 VPC 종단점에 따라 프라이빗 API 트래픽 허용

예: 다른 AWS 계정의 역할이 API를 사용하도록 허용

다음 예제 리소스 정책은 서명 버전 4 (SigV4) 프로토콜을 통해 한 AWS AWS 계정의 API 액세스 권한을 다른 계정의 두 역할에 부여합니다. 구체적으로, 로 식별된 AWS 계정의 개발자 및 관리자 *account-id-2* 역할에는 계정의 `pets` 리소스 (API) 에서 `GET` 작업을 실행할 수 AWS 있는 작업이 부여됩니다. `execute-api:Invoke`

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

```
            "Effect": "Allow",
            "Principal": {
                "AWS": [
                    "arn:aws:iam::account-id-2:role/developer",
                    "arn:aws:iam::account-id-2:role/Admin"
                ]
            },
            "Action": "execute-api:Invoke",
            "Resource": [
                "execute-api:/stage/GET/pets"
            ]
        }
    ]
}
```

예제: 소스 IP 주소 또는 범위에 따라 API 트래픽 거부

아래의 리소스 정책 예제에서는 2개의 지정된 소스 IP 주소 블록에서 API로 들어오는 트래픽을 거부(차단)합니다.

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

```
        }
    ]
}
```

예: 프라이빗 API를 사용할 때 소스 IP 주소 또는 범위를 기반으로 API 트래픽 거부

아래의 리소스 정책 예제에서는 2개의 지정된 소스 IP 주소 블록에서 프라이빗 API로 들어오는 트래픽을 거부(차단)합니다. 프라이빗 API를 사용할 때 `execute-api`의 VPC 종단점은 원래 소스 IP 주소를 다시 기록합니다. `aws:VpcSourceIp` 조건은 원래 요청자 IP 주소를 기준으로 요청을 필터링합니다.

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

예제: 소스 VPC 또는 VPC 종단점에 따라 프라이빗 API 트래픽 허용

다음 예제의 리소스 정책에서는 지정된 Virtual Private Cloud(VPC) 또는 VPC 종단점에서 오는 수신 트래픽만 프라이빗 API에 액세스하도록 허용합니다.

이 리소스 정책 예제는 소스 VPC를 지정합니다.

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

이 리소스 정책 예제는 소스 VPC 종단점을 지정합니다.

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

```
            "Action": "execute-api:Invoke",
            "Resource": [
                "execute-api:/*"
            ],
            "Condition" : {
                "StringNotEquals": {
                    "aws:SourceVpce": "vpce-1a2b3c4d"
                }
            }
        }
    ]
}
```

API Gateway 리소스 정책을 생성하여 API에 연결

사용자가 API 실행 서비스를 호출하여 API에 액세스하도록 허용하려면 API Gateway 리소스에 대한 액세스를 제어하는 API Gateway 리소스 정책을 생성하고 이를 API에 연결해야 합니다.

> **Important**
>
> API Gateway 리소스 정책을 업데이트하려면 `apigateway:UpdateRestApiPolicy` 권한 외에 `apigateway:PATCH` 권한도 있어야 합니다.

리소스 정책은 API를 생성할 때 API에 연결하거나 나중에 연결할 수 있습니다. 프라이빗 API의 경우, 리소스 정책을 프라이빗 API에 연결할 때까지는 해당 API에 대한 모든 호출이 실패합니다.

> **Important**
>
> API가 생성된 후에 리소스 정책을 업데이트하려면 업데이트된 정책을 연결한 후에 변경 내용이 전파되도록 API를 배포해야 합니다. 정책을 업데이트 또는 저장하는 것만으로는 API의 실행 시간 동작이 변경되지 않습니다. API 배포에 대한 자세한 내용은 Amazon API Gateway에서 REST API 배포 단원을 참조하세요.

조건, 소스 VPC AWS 계정, 소스 VPC 엔드포인트 또는 IP 범위를 비롯한 IAM 조건 요소로 액세스를 제어할 수 있습니다. 정책의 `Principal`가 `"*"`로 설정된 경우, 리소스 정책과 함께 다른 권한 부여 유형을 사용할 수 있습니다. `Principal`가 `"AWS"`로 설정된 경우에는 보안되지 않은 리소스를 포함하여 `AWS_IAM` 권한 부여를 통해 보호되지 않는 모든 리소스에 대해 권한 부여가 실패합니다.

다음 단원에서는 자체적으로 API Gateway 리소스 정책을 생성하여 이를 API에 연결하는 방법을 설명합니다. 정책을 연결하면 정책의 해당 권한이 API의 메서드에 적용됩니다.

> **Important**
>
> API Gateway 콘솔을 사용하여 배포된 API에 리소스 정책을 연결하거나 기존의 리소스 정책을 업데이트할 때는 API를 해당 콘솔에 다시 배포해야 변경 사항이 적용됩니다.

주제

- API Gateway 리소스 정책 연결(콘솔)
- API Gateway 리소스 정책 (AWS CLI) 연결
- API Gateway 리소스 정책 연결 ()AWS CloudFormation

API Gateway 리소스 정책 연결(콘솔)

AWS 관리 콘솔을 사용하여 API Gateway API에 리소스 정책을 연결할 수 있습니다.

API Gateway API에 리소스 정책을 연결하려면

1. https://console.aws.amazon.com/apigateway에서 API Gateway 콘솔에 로그인합니다.
2. REST API를 선택합니다.
3. 기본 탐색 창에서 리소스 정책을 선택합니다.
4. 정책 생성을 선택합니다.
5. (선택 사항) 예제 정책을 생성하려면 템플릿 선택을 선택합니다.

   여기 나온 정책 예제에서는 자리 표시자가 이중 중괄호("{{*placeholder*}}")로 묶여 있습니다. 중괄호를 포함한 각각의 자리 표시자를 필요한 정보로 바꿉니다.
6. 템플릿 예제 중 하나를 사용하지 않는 경우에는 리소스 정책을 입력합니다.
7. 변경 사항 저장를 선택합니다.

API Gateway 콘솔에서 이미 배포한 API라면 다시 배포해야 리소스 정책이 적용됩니다.

API Gateway 리소스 정책 (AWS CLI) 연결

를 AWS CLI 사용하여 새 API를 만들고 여기에 리소스 정책을 연결하려면 다음과 같이 `create-rest-api`명령을 호출합니다.

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

를 사용하여 기존 API에 리소스 정책을 AWS CLI 연결하려면 다음과 같이 update-rest-api명령을 호출합니다.

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

API Gateway 리소스 정책 연결 ()AWS CloudFormation

를 사용하여 AWS CloudFormation 리소스 정책을 사용하여 API를 생성할 수 있습니다. 다음 예시에서는 예제 리소스 정책을 사용하여 REST API를 만듭니다the section called “예제: 소스 IP 주소 또는 범위에 따라 API 트래픽 거부”.

예제 AWS CloudFormation 템플릿

```
AWSTemplateFormatVersion: 2010-09-09
Resources:
  Api:
    Type: 'AWS::ApiGateway::RestApi'
    Properties:
      Name: testapi
      Policy:
        Statement:
          - Action: 'execute-api:Invoke'
            Effect: Allow
            Principal: '*'
            Resource: 'execute-api/*'
          - Action: 'execute-api:Invoke'
            Effect: Deny
            Principal: '*'
            Resource: 'execute-api/*'
            Condition:
              IpAddress:
                'aws:SourceIp': ["192.0.2.0/24", "198.51.100.0/24" ]
        Version: 2012-10-17
  Resource:
```

```
    Type: 'AWS::ApiGateway::Resource'
    Properties:
      RestApiId: !Ref Api
      ParentId: !GetAtt Api.RootResourceId
      PathPart: 'helloworld'
  MethodGet:
    Type: 'AWS::ApiGateway::Method'
    Properties:
      RestApiId: !Ref Api
      ResourceId: !Ref Resource
      HttpMethod: GET
      ApiKeyRequired: false
      AuthorizationType: NONE
      Integration:
        Type: MOCK
  ApiDeployment:
    Type: 'AWS::ApiGateway::Deployment'
    DependsOn:
      - MethodGet
    Properties:
      RestApiId: !Ref Api
      StageName: test
```

AWS API Gateway 리소스 정책에서 사용할 수 있는 조건 키

다음 표에는 각 권한 부여 유형별로 API Gateway의 API에 대한 리소스 정책에 사용할 수 있는 AWS 조건 키가 나와 있습니다.

AWS 조건 키에 대한 자세한 내용은 AWS 글로벌 조건 컨텍스트 키를 참조하십시오.

조건 키 표

조건 키	기준	**AuthN**이 필요합니까?	권한 부여 유형
aws:CurrentTime	없음	아니요	모두
aws:EpochTime	없음	아니요	모두
aws:Token IssueTime	임시 보안 자격 증명을 사용해 서명된 요청에만 키가 존재합니다.	예	IAM

조건 키	기준	**AuthN**이 필요합니까?	권한 부여 유형
aws:MultiFactorAuthPresent	임시 보안 자격 증명을 사용해 서명된 요청에만 키가 존재합니다.	예	IAM
aws:MultiFactorAuthAge	MFA가 요청에 존재하는 경우에만 키가 존재합니다.	예	IAM
aws:PrincipalAccount	없음	예	IAM
aws:PrincipalArn	없음	예	IAM
aws:PrincipalOrgID	이 키는 보안 주체가 조직의 멤버인 경우에만 요청 컨텍스트에 포함됩니다.	예	IAM
aws:PrincipalOrgPaths	이 키는 보안 주체가 조직의 멤버인 경우에만 요청 컨텍스트에 포함됩니다.	예	IAM
aws:PrincipalTag	이 키는 보안 주체가 태그가 연결된 IAM 사용자를 사용하는 경우 요청 컨텍스트에 포함됩니다. 연결된 태그 또는 세션 태그가 있는 IAM 역할을 사용하는 보안 주체에 포함됩니다.	예	IAM
aws:PrincipalType	없음	예	IAM

조건 키	기준	**AuthN**이 필요합니까?	권한 부여 유형
aws:Referer	HTTP 헤더에서 호출자가 값을 제공하는 경우에만 키가 존재합니다.	아니요	모두
aws:SecureTransport	없음	아니요	모두
aws:SourceArn	없음	아니요	모두
aws:SourceIp	없음	아니요	모두
aws:SourceVpc	이 키는 프라이빗 API에만 사용할 수 있습니다.	아니요	모두
aws:SourceVpce	이 키는 프라이빗 API에만 사용할 수 있습니다.	아니요	모두
aws:UserAgent	HTTP 헤더에서 호출자가 값을 제공하는 경우에만 키가 존재합니다.	아니요	모두
aws:userid	없음	예	IAM
aws:username	없음	예	IAM

IAM 권한을 사용하여 API에 대한 액세스 제어

다음 두 API Gateway 구성 요소 프로세스에 대한 액세스를 제어하여 IAM 권한을 통해 Amazon API Gateway API에 대한 액세스를 제어합니다.

- API Gateway에서 API를 생성, 배포, 관리하려면 API 개발자에게 API Gateway의 API 관리 구성 요소에서 지원되는 필수 작업을 수행할 수 있는 권한을 부여해야 합니다.

- 배포된 API를 호출하거나 API 캐시를 새로 고치려면 API Gateway의 API 실행 구성 요소에서 지원되는 필요한 IAM 작업을 수행할 권한을 API 호출자에게 부여해야 합니다.

두 프로세스에 대한 액세스 제어에는 다음에 설명하는 서로 다른 권한 모델이 포함됩니다.

API 생성 및 관리를 위한 API Gateway 권한 모델

API 개발자가 API Gateway에서 API를 생성하고 관리할 수 있도록 허용하려면 지정된 API 개발자가 필수 API 엔터티를 생성, 업데이트, 배포, 보기 또는 삭제할 수 있도록 허용하는 IAM 권한 정책을 생성해야 합니다. 사용자, 역할 또는 그룹에 권한 정책을 연결합니다.

액세스 권한을 제공하려면 사용자, 그룹 또는 역할에 권한을 추가합니다.

- AWS IAM Identity Center의 사용자 및 그룹:

  권한 세트를 생성합니다. AWS IAM Identity Center 사용 설명서의 권한 세트 생성의 지침을 따르세요.
- 자격 증명 공급자를 통해 IAM에서 관리되는 사용자:

  아이덴티티 페더레이션을 위한 역할을 생성합니다. IAM 사용 설명서의 서드 파티 자격 증명 공급자의 역할 만들기(연합)의 지침을 따르세요.
- IAM 사용자:
  - 사용자가 맡을 수 있는 역할을 생성합니다. IAM 사용 설명서에서 IAM 사용자의 역할 생성의 지침을 따르세요.
  - (권장되지 않음) 정책을 사용자에게 직접 연결하거나 사용자를 사용자 그룹에 추가합니다. IAM 사용 설명서에서 사용자(콘솔)에 권한 추가의 지침을 따르세요.

이 권한 모델을 사용하는 방법에 대한 자세한 내용은 the section called “API Gateway 자격 증명 기반 정책” 단원을 참조하십시오.

API 호출을 위한 API Gateway 권한 모델

API 호출자가 API를 호출하거나 캐싱을 새로 고치도록 허용하려면 지정된 API 호출자에게 사용자 인증이 사용되는 API 메서드를 호출하도록 허용하는 IAM 정책을 생성해야 합니다. API 개발자는 메서드의 `authorizationType` 속성을 `AWS_IAM`으로 설정하여 인증받을 사용자의 보안 인증 정보를 제출하도록 호출자에게 요구합니다. 그런 다음 정책을 사용자, 역할 또는 그룹에 연결할 수 있습니다.

이 IAM 권한 정책 설명에서, IAM `Resource` 요소는 지정된 HTTP 동사와 API Gateway 리소스 경로로 식별할 수 있는 배포된 API 메서드 목록을 포함합니다. IAM `Action` 요소는 필요한 API

Gateway API 실행 작업을 포함합니다. 이러한 작업은 `execute-api:Invoke` 또는 `execute-api:InvalidateCache`를 포함합니다. 여기서 `execute-api`는 API Gateway의 기본 API 실행 구성 요소를 지정합니다.

이 권한 모델을 사용하는 방법에 대한 자세한 내용은 API 호출을 위한 액세스 제어 단원을 참조하십시오.

API가 백엔드에서 AWS 서비스(예: AWS Lambda)와 통합될 경우 API Gateway에도 API 호출자를 대신하여 통합된 AWS 리소스(예: Lambda 함수 호출)에 액세스할 권한이 있어야 합니다. 이러한 권한을 부여하려면 API Gateway용 AWS 서비스 유형의 IAM 역할을 생성합니다. IAM 관리 콘솔에서 이 역할을 생성하면 이 결과 역할에는 API Gateway를 역할을 수임하도록 허용되는 신뢰할 수 있는 개체로 선언하는 다음 IAM 신뢰 정책이 포함됩니다.

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

CLI create-role 명령 또는 동등한 SDK 메서드를 호출하여 IAM 역할을 생성할 경우 `assume-role-policy-document`의 입력 파라미터로 상기 신뢰 정책을 제공해야 합니다. IAM 관리 콘솔에서 직접, 또는 AWS CLI create-policy 명령 또는 동등한 SDK 메서드를 호출하여 그러한 정책을 생성하려 하지 마세요.

API Gateway가 통합된 AWS 서비스를 호출하려면, 이 역할에 통합된 AWS 서비스를 호출하는 데 적절한 IAM 권한 정책도 연결해야 합니다. 예를 들어 Lambda 함수를 호출하려면 IAM 역할에 다음 IAM 권한 정책을 포함해야 합니다.

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

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

Lambda는 신뢰 정책과 권한 정책을 결합한 리소스 기반 액세스 정책을 지원합니다. API Gateway 콘솔을 사용하여 API를 Lambda 함수와 통합할 경우 콘솔에서 사용자의 동의를 받아 Lambda 함수에 대한 리소스 기반 권한을 자동으로 설정하므로 이 IAM 역할을 명시적으로 설정하라는 메시지가 표시되지 않습니다.

> **Note**
>
> AWS 서비스에 대한 액세스 제어를 적용하려면 권한 정책이 호출자의 사용자 또는 그룹에 직접 연결되는 호출자 기반 권한 모델을 사용하거나 권한 정책이 API Gateway에서 가정할 수 있는 IAM 역할에 연결되는 역할 기반 권한 모델을 사용할 수 있습니다. 두 모델의 권한 정책은 다를 수 있습니다. 예를 들어 호출자 기반 정책에서는 액세스를 차단하고 역할 기반 정책에서는 액세스를 허용할 수 있습니다. 이를 활용하여 사용자에게 API Gateway API를 통해서만 AWS 서비스에 액세스하도록 요구할 수 있습니다.

API 호출을 위한 액세스 제어

이 단원에서는 API Gateway에서 배포된 API를 호출할 수 있는 사용자를 제어하는 IAM 정책 설명을 작성하는 방법을 알아봅니다. 또한 API 실행 서비스와 관련된 `Action` 및 `Resource` 필드의 형식을 비롯하여 정책 설명 참조도 찾을 수 있습니다. the section called “리소스 정책이 권한 부여 워크플로우에 미치는 영향”의 IAM 섹션도 알아보아야 합니다.

프라이빗 API의 경우 API Gateway 리소스 정책과 VPC 종단점 정책을 조합하여 사용해야 합니다. 자세한 정보는 다음 주제를 참조하세요.

- the section called “API Gateway 리소스 정책 사용”
- the section called “프라이빗 API에 대한 VPC 종단점 정책 사용”

IAM 정책을 사용하여 API Gateway API 메서드를 호출할 수 있는 사용자 제어

IAM 권한을 사용하여 배포된 API를 호출할 수 있는 사용자와 호출할 수 없는 사용자를 제어하려면 필요한 권한을 사용하여 IAM 정책 문서를 생성합니다. 그런 정책 문서에 대한 템플릿은 다음과 같습니다.

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

여기서 포함된 권한을 부여할지 취소할지 여부에 따라 *Permission*은 Allow 또는 Deny로 대체됩니다. *Execution-operation*은 API 실행 서비스에서 지원되는 작업으로 대체됩니다. *METHOD_HTTP_VERB*는 지정된 리소스에서 지원되는 HTTP 동사를 나타냅니다. *Resource-path*는 해당 *METHOD_HTTP_VERB*를 지원하는 배포된 API Resource 인스턴스의 URL 경로에 대한 자리 표시자입니다. 자세한 내용은 API Gateway에서 API 실행을 위한 IAM 정책의 설명 참조 섹션을 참조하세요.

> **Note**
>
> IAM 정책을 적용하려면 메서드의 AWS_IAM 속성에 대한 authorizationType을 설정하여 API 메서드에서 IAM 인증을 활성화해야 합니다. 그렇지 않은 경우 API 메서드에 공개적으로 액세스할 수 있습니다.

예를 들어 사용자에게 지정된 API 노출 반려 동물 목록을 볼 수 있는 권한을 부여하고, 목록에 반려 동물을 추가할 수 있는 권한을 거부하려면 IAM 정책에 다음 명령문을 포함하세요.

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

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

사용자에게 GET /pets/*{petId}*로 구성된 API를 통해 공개된 특정 반려 동물을 볼 수 있는 권한을 부여하려면, IAM 정책에 다음 명령문을 포함합니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "execute-api:Invoke"
      ],
      "Resource": [
        "arn:aws:execute-api:us-east-1:account-id:api-id/*/GET/pets/a1b2"
      ]
    }
  ]
}
```

API Gateway에서 API 실행을 위한 IAM 정책의 설명 참조

다음은 API 실행을 위한 액세스 권한에 대한 IAM 정책 설명의 작업 및 리소스 형식입니다.

API Gateway의 API 실행 권한 작업 형식

API 실행 Action 표현식의 일반적인 형식은 다음과 같습니다.

```
execute-api:action
```

여기서 *##*은 사용 가능한 API 실행 작업입니다.

- *는 다음의 모든 작업을 나타냅니다.
- 호출은 클라이언트 요청에 따라 API를 호출하는 데 사용됩니다.
- InvalidateCache는 클라이언트 요청에 따라 API 캐시를 무효화하는 데 사용됩니다.

API Gateway의 API 실행 권한 리소스 형식

API 실행 `Resource` 표현식의 일반적인 형식은 다음과 같습니다.

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

여기서 각 항목은 다음과 같습니다.

- *region*은 메서드에 대해 배포된 API에 해당하는 AWS 리전(예: **us-east-1** 또는 ***(모든 AWS 리전))입니다.
- *account-id*는 REST API 소유자의 12자리 AWS 계정 ID입니다.
- *api-id*는 API Gateway에서 메서드에 대한 API에 할당한 식별자입니다.
- *stage-name*은 메서드와 연결된 스테이지의 이름입니다.
- *HTTP-VERB*는 메서드에 대한 HTTP 동사입니다. GET, POST, PUT, DELETE, PATCH 중 하나일 수 있습니다.
- *resource-path-specifier*는 원하는 메서드의 경로입니다.

> **Note**
>
> 와일드카드(*)를 지정하는 경우 `Resource` 표현식에서 표현식의 나머지에 와일드카드가 적용됩니다.

다음은 리소스 표현식의 몇 가지 예입니다.

- **arn:aws:execute-api:*:*:***는 모든 단계의 모든 리소스 경로와 모든 AWS 리전의 모든 API를 나타내며,
- **arn:aws:execute-api:us-east-1:*:***는 모든 단계의 모든 리소스 경로와 `us-east-1`의 AWS 리전의 모든 API를 나타내며,

- **arn:aws:execute-api:us-east-1:*:*api-id*/***는 모든 단계의 모든 리소스 경로와 AWS 리전(us-east-1)에서 식별자가 *api-id*인 API를 나타냅니다.
- **arn:aws:execute-api:us-east-1:*:*api-id*/test/***는 `test` 단계의 리소스 경로와 AWS 리전(us-east-1)에서 식별자가 *api-id*인 API를 나타냅니다.

자세한 내용은 API Gateway Amazon 리소스 이름(ARN) 참조 단원을 참조하십시오.

API 실행 권한에 대한 IAM 정책 예

권한 모델 및 기타 배경 정보는 API 호출을 위한 액세스 제어 단원을 참조하십시오.

다음 정책 설명은 사용자에게 `mydemoresource` 경로를 따라 `test` 단계에서 식별자가 a123456789인 API에 대해 POST 메서드를 호출할 수 있는 권한을 부여합니다. 여기서는 해당 API가 AWS 리전(us-east-1)에 배포된 것으로 가정합니다.

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

다음 예제 정책 설명은 사용자에게 모든 단계의 `petstorewalkthrough/pets` 리소스 경로에서 해당 API가 배포된 a123456789 리전에서 식별자가 AWS인 API에 대해 모든 메서드를 호출할 수 있는 권한을 부여합니다.

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

```
      ],
      "Resource": [
        "arn:aws:execute-api:*:*:a123456789/*/*/petstorewalkthrough/pets"
      ]
    }
  ]
}
```

정책을 생성하여 사용자에 연결

사용자가 API 관리 서비스 또는 API 실행 서비스를 호출할 수 있도록 허용하려면 API Gateway 엔터티에 대한 액세스를 제어하는 IAM 정책을 생성해야 합니다.

JSON 정책 편집기를 사용하여 정책을 생성하려면

1. AWS Management Console에 로그인하여 https://console.aws.amazon.com/iam/에서 IAM 콘솔을 엽니다.
2. 왼쪽의 탐색 창에서 정책(Policies)을 선택합니다.

   정책(Policies)을 처음으로 선택하는 경우 관리형 정책 소개 페이지가 나타납니다. 시작하기(Get Started)를 선택합니다.
3. 페이지 상단에서 정책 생성을 선택합니다.
4. 정책 편집기 섹션에서 JSON 옵션을 선택합니다.
5. 다음 JSON 정책 문서를 입력합니다.

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

```
      ],
      "Resource" : [
        "resource-statement"
      ]
    }
  ]
}
```

6. 다음(Next)을 선택합니다.

> **Note**
>
> 언제든지 시각적 편집기 옵션과 JSON 편집기 옵션을 서로 전환할 수 있습니다. 그러나 변경을 적용하거나 시각적 편집기에서 다음을 선택한 경우 IAM은 시각적 편집기에 최적화되도록 정책을 재구성할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 정책 재구성을 참조하세요.

7. 검토 및 생성 페이지에서 생성하는 정책에 대한 정책 이름과 설명(선택 사항)을 입력합니다. 이 정책에 정의된 권한을 검토하여 정책이 부여한 권한을 확인합니다.
8. 정책 생성(Create policy)을 선택하고 새로운 정책을 저장합니다.

이 설명에서 필요에 따라 *action-statement* 및 *resource-statement*를 대체하고 다른 설명을 추가하여, 사용자가 관리하도록 허용할 API Gateway 엔터티, 사용자가 호출할 수 있는 API 또는 두 항목을 모두 지정합니다. 기본적으로 해당하는 명시적 `Allow` 설명이 없는 한, 용자에게는 권한이 없습니다.

방금 IAM 정책을 생성했지만, 연결하기 전까지는 아무런 효과가 없습니다.

액세스 권한을 제공하려면 사용자, 그룹 또는 역할에 권한을 추가합니다.

- AWS IAM Identity Center의 사용자 및 그룹:

  권한 세트를 생성합니다. AWS IAM Identity Center 사용 설명서의 권한 세트 생성의 지침을 따르세요.

- 자격 증명 공급자를 통해 IAM에서 관리되는 사용자:

  아이덴티티 페더레이션을 위한 역할을 생성합니다. IAM 사용 설명서의 서드 파티 자격 증명 공급자의 역할 만들기(연합)의 지침을 따르세요.

- IAM 사용자:

- 사용자가 맡을 수 있는 역할을 생성합니다. IAM 사용 설명서에서 IAM 사용자의 역할 생성의 지침을 따르세요.
- (권장되지 않음) 정책을 사용자에게 직접 연결하거나 사용자를 사용자 그룹에 추가합니다. IAM 사용 설명서에서 사용자(콘솔)에 권한 추가의 지침을 따르세요.

IAM 정책 문서를 IAM 그룹에 연결하려면

1. 기본 탐색 창에서 그룹을 선택합니다.
2. 선택한 그룹 아래에서 권한 탭을 선택합니다.
3. [Attach policy]를 선택합니다.
4. 앞서 생성한 정책 문서를 선택한 후 정책 연결을 선택합니다.

API Gateway에서 다른 AWS 서비스를 자동으로 호출하도록 하려면 Amazon API Gateway 유형의 IAM 역할을 생성합니다.

Amazon API Gateway 유형의 역할을 생성하려면

1. 기본 탐색 창에서 역할을 선택합니다.
2. 새 역할 생성을 선택합니다.
3. 역할 이름에 이름을 입력한 후 다음 단계를 선택합니다.
4. 역할 유형 선택(Select Role Type)의 AWS 서비스 역할(AWS Service Roles)에서 Amazon API Gateway 옆에 있는 선택(Select)을 선택합니다.
5. 사용 가능한 관리형 IAM 권한 정책을 선택합니다. 예를 들어 API Gateway가 CloudWatch에 메트릭을 기록하도록 하려면, 정책 연결에서 AmazonAPIGatewayPushToCloudWatchLog를 선택한 후 다음 단계를 선택합니다.
6. 신뢰할 수 있는 주체(Trusted Entities)에서 apigateway.amazonaws.com이 항목으로 나열되는지 확인한 후 역할 생성을 선택합니다.
7. 새로 생성한 역할에 대해 권한 탭을 선택한 다음, 정책 연결을 선택합니다.
8. 앞서 생성한 사용자 지정 IAM 정책 문서를 선택한 후 정책 연결을 선택합니다.

API Gateway의 프라이빗 API에 대한 VPC 종단점 정책 사용

인터페이스 VPC 엔드포인트를 사용하도록 API Gateway를 구성하여 프라이빗 API의 보안을 향상할 수 있습니다. 인터페이스 엔드포인트는 사설 IP 주소를 사용하여 비공개로 AWS PrivateLink 서비스에

액세스할 수 있게 해주는 AWS 기술을 통해 제공됩니다. VPC 종단점에 대한 자세한 내용은 인터페이스 엔드포인트 생성을 참조하십시오.

VPC 종단점 정책은 인터페이스 VPC 종단점에 연결하여 엔드포인트에 대한 액세스를 제어할 수 있는 IAM 리소스 정책입니다. 자세한 정보는 VPC 종단점을 통해 서비스에 대한 액세스 제어를 참조하십시오. 엔드포인트 정책을 사용하여 내부 네트워크에서 프라이빗 API에 액세스하는 트래픽을 제한할 수 있습니다. VPC 종단점을 통해 액세스할 수 있는 특정 프라이빗 API에 대한 액세스를 허용 또는 거부하도록 선택할 수 있습니다.

VPC 종단점 정책은 API Gateway 리소스 정책과 함께 사용할 수 있습니다. 리소스 정책에서는 해당 리소스에 액세스할 수 있는 보안 주체를 지정합니다. 엔드포인트 정책은 VPC 종단점을 통해 호출할 수 있는 프라이빗 API를 지정합니다. 리소스 정책에 대한 자세한 내용은 the section called “API Gateway 리소스 정책 사용” 단원을 참조하십시오.

VPC 엔드포인트 정책 고려 사항

- 정책에서 IAM 보안 주체를 제한하는 경우 메서드의 `authorizationType`을 `AWS_IAM` 또는 `NONE`으로 설정해야 합니다.
- 호출자의 ID는 `Authorization` 헤더 값을 기반으로 평가됩니다. 사용자의 `authorizationType`에 따라 다르지만 이로 인해 `403 IncompleteSignatureException` 또는 `403 InvalidSignatureException` 오류가 발생할 수 있습니다. 다음 표에는 각 `authorizationType`의 `Authorization` 헤더 값이 나와 있습니다.

authorizationType	**Authorization** 헤더 평가 여부	허용되는 **Authorization** 헤더 값
`NONE`(기본 전체 액세스 정책 사용)	아니요	전달되지 않음
`NONE`(사용자 지정 액세스 정책 사용)	예	유효한 SigV4 값이어야 함
`IAM`	예	유효한 SigV4 값이어야 함
`CUSTOM` 또는 `COGNITO_USER_POOLS`	아니요	전달되지 않음

VPC 종단점 정책 예제

Amazon API Gateway에 대한 Amazon Virtual Private Cloud 엔드포인트 정책을 생성하여 다음을 지정할 수 있습니다.

- 작업을 수행할 수 있는 보안 주체.
- 수행할 수 있는 작업입니다.
- 수행되는 작업을 가질 수 있는 리소스입니다.

VPC 종단점에 정책을 첨부하려면 VPC 콘솔을 사용해야 합니다. 자세한 정보는 VPC 종단점을 통해 서비스에 대한 액세스 제어를 참조하십시오.

예제 1: 두 API에 대한 액세스 권한을 부여하는 VPC 종단점 정책

다음 예제 정책은 정책이 첨부된 VPC 종단점을 통해 두 개의 특정 API로만 액세스 권한을 부여합니다.

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

예제 2: GET 메서드에 대한 액세스 권한을 부여하는 VPC 종단점 정책

다음 예제 정책은 해당 정책이 연결된 VPC 종단점을 통해 특정 API에 대한 GET 메서드에 대한 액세스 권한을 사용자에게 부여합니다.

```
{
    "Statement": [
        {
```

```
            "Principal": "*",
            "Action": [
                "execute-api:Invoke"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:execute-api:us-east-1:123412341234:a1b2c3d4e5/stageName/GET/*"
            ]
        }
    ]
}
```

예제 3: 특정 API에 특정 사용자 액세스 권한을 부여하는 VPC 종단점 정책

다음 예제 정책은 정책이 연결된 VPC 종단점을 통한 특정 API에 대한 특정 사용자 액세스 권한을 부여합니다.

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

API Gateway에서 태그를 사용하여 REST API에 대한 액세스 제어

IAM 정책에서 속성 기반 액세스 제어를 사용하여 REST API 액세스 권한을 미세 조정할 수 있습니다.

자세한 정보는 the section called “속성 기반 액세스 제어”을 참조하십시오.

API Gateway Lambda 권한 부여자 사용

Lambda 권한 부여자(이전 사용자 지정 권한 부여자)는 Lambda 함수를 사용하여 API 메서드에 대한 액세스를 제어하는 API Gateway 기능입니다.

Lambda 권한 부여자는 OAuth 또는 SAML과 같은 보유자 토큰 인증 전략을 사용하거나 요청 파라미터를 사용하여 호출자의 자격 증명을 확인하는 사용자 지정 인증 체계를 구현하려는 경우에 유용합니다.

클라이언트가 API의 메서드 중 하나를 요청할 경우 API Gateway는 해당 Lambda 권한 부여자를 호출합니다. 이 권한 부여자는 호출자의 자격 증명을 입력으로 받아서 IAM 정책을 출력으로 반환합니다.

Lambda 권한 부여자는 두 가지 유형이 있습니다.

- 토큰 기반 Lambda 권한 부여자(`TOKEN` 권한 부여자라고도 함)는 보유자 토큰(JSON Web Token(JWT) 또는 OAuth 토큰)에 있는 호출자의 자격 증명을 받습니다. 예제 신청서는 오픈 뱅킹 브라질 - 승인 샘플 온을 참조하십시오 GitHub.
- 요청 파라미터 기반 Lambda 권한 부여자(`REQUEST` 권한 부여자라고도 함)는 헤더, 쿼리 문자열 파라미터, `stageVariables` 및 `$context` 변수의 조합으로 호출자의 자격 증명을 수신합니다.

  WebSocket API의 경우 요청 매개변수 기반 권한 부여자만 지원됩니다.

API를 생성한 계정과 다른 AWS 계정의 AWS Lambda 함수를 사용할 수 있습니다. 자세한 설명은 the section called “교차 계정 Lambda 권한 부여자 구성” 섹션을 참조하세요.

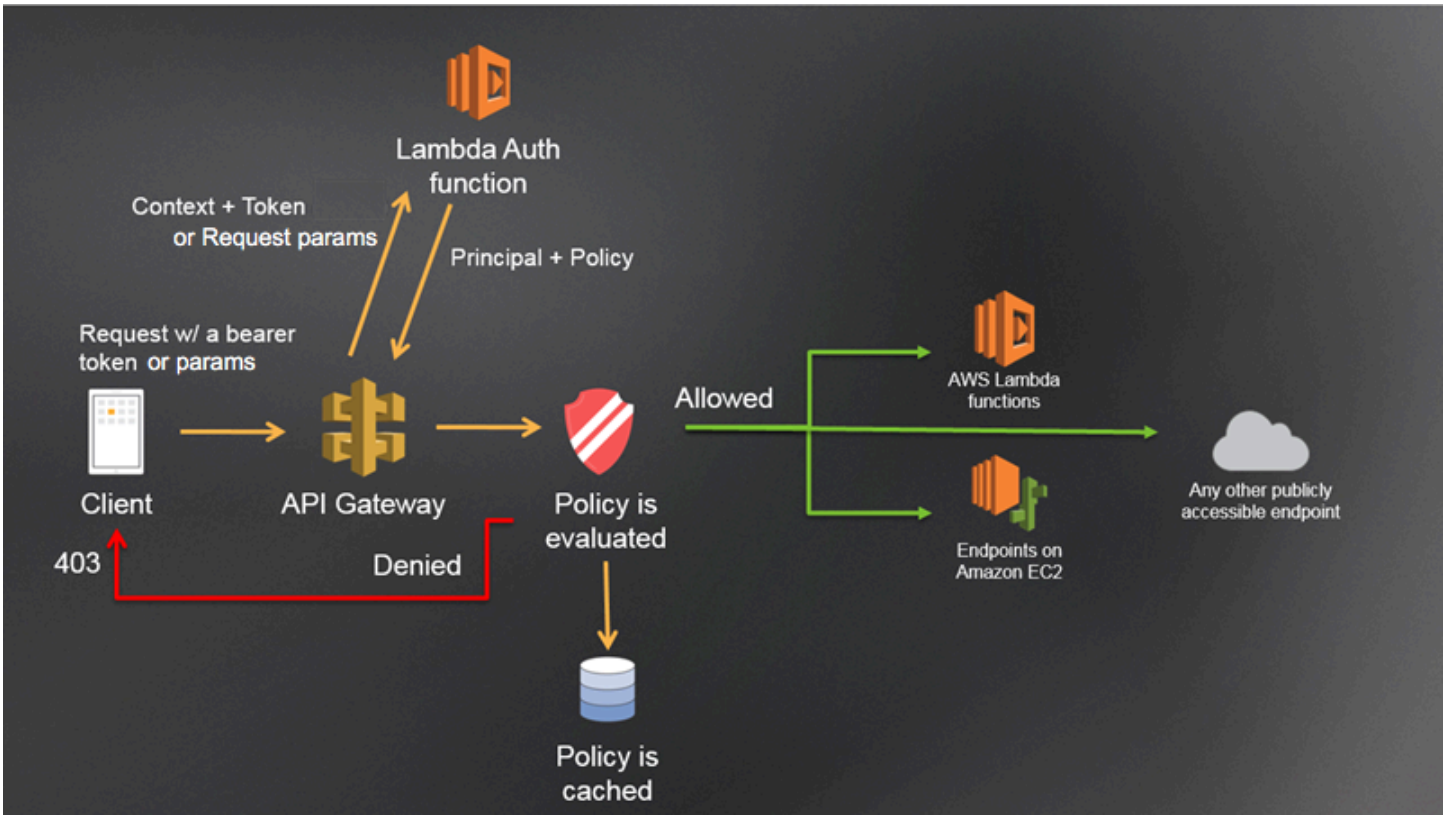
Lambda 함수의 예는 -blueprints on을 참조하십시오. aws-apigateway-lambda-authorizer GitHub

주제

- Lambda 권한 부여자 인증 워크플로우
- API Gateway Lambda 권한 부여자 생성 단계
- Lambda 콘솔에서 API Gateway Lambda 권한 부여자 함수 생성
- API Gateway 콘솔을 사용하여 Lambda 권한 부여자 구성
- Amazon API Gateway Lambda 권한 부여자에 대한 입력
- Amazon API Gateway Lambda 권한 부여자의 출력
- API Gateway Lambda 권한 부여자를 사용하여 API 호출
- 교차 계정 Lambda 권한 부여자 구성

Lambda 권한 부여자 인증 워크플로우

다음 다이어그램은 Lambda 권한 부여자에 대한 인증 워크플로를 보여 줍니다.



API Gateway Lambda 권한 부여 워크플로

1. 클라이언트가 API Gateway API 메서드의 메서드를 호출하고 보유자 토큰이나 요청 파라미터를 전달합니다.
2. API Gateway는 메서드에 대해 Lambda 권한 부여자가 구성되어 있는지 확인합니다. 구성되어 있으면 API Gateway가 Lambda 함수를 호출합니다.
3. Lambda 함수는 다음과 같은 방법을 통해 호출자를 인증합니다.
   - OAuth 공급자를 호출하여 OAuth 액세스 토큰을 받습니다.
   - SAML 공급자를 호출하여 SAML 어설션을 받습니다.
   - 요청 파라미터 값을 기반으로 IAM 정책을 생성합니다.
   - 데이터베이스에서 자격 증명을 가져옵니다.
4. 호출이 성공하면 Lambda 함수는 한 개 이상의 IAM 정책과 보안 주체 식별자를 포함하는 출력 객체를 반환하여 액세스를 부여합니다.
5. API Gateway가 정책을 평가합니다.

- 액세스가 거부되면 API Gateway는 적절한 HTTP 상태 코드(예: `403 ACCESS_DENIED`)를 반환합니다.
- 액세스가 허용되면 API Gateway가 메서드를 호출합니다. 권한 부여자 설정에서 캐싱이 활성화된 경우, API Gateway는 Lambda 권한 부여자 함수를 다시 호출할 필요가 없도록 정책을 캐싱합니다.

6. Lambda 함수가 응답을 반환하는 경우 호출이 실패할 수 있습니다. `401 Unauthorized` `401 Unauthorized`게이트웨이 응답을 사용자 지정할 수 있습니다. 자세한 내용은 the section called "게이트웨이 응답"를 참조하세요.

API Gateway Lambda 권한 부여자 생성 단계

Lambda 권한 부여자를 생성하려면 다음 작업을 수행해야 합니다.

1. the section called "Lambda 콘솔에서 Lambda 권한 부여자 함수 생성"에 설명된 대로 Lambda 콘솔에서 Lambda 권한 부여자 함수를 생성합니다. 블루프린트 예제 중 하나를 시작점으로 사용하여 입력 및 출력을 원하는 대로 사용자 지정할 수 있습니다.
2. the section called "콘솔을 사용하여 Lambda 권한 부여자 구성"에서 설명한 대로 Lambda 함수를 API Gateway 권한 부여자로 구성하고, API 메서드가 이를 요구하도록 구성합니다. 또는 교차 계정 Lambda 권한 부여자가 필요할 경우 the section called "교차 계정 Lambda 권한 부여자 구성" 단원을 참조하세요.

   > **Note**
   >
   > AWS CLI 또는 AWS SDK를 사용하여 권한 부여자를 구성할 수도 있습니다.

3. the section called "Lambda 권한 부여자를 사용하여 API 호출"에서 설명한 대로 Postman을 사용하여 권한 부여자를 테스트합니다.

Lambda 콘솔에서 API Gateway Lambda 권한 부여자 함수 생성

Lambda 권한 부여자를 구성하기 전에 먼저, 호출자를 권한 부여하고 필요한 경우 인증하기 위한 로직을 구현하는 Lambda 함수를 생성해야 합니다. Lambda 콘솔은 Python 블루프린트를 제공합니다. Python 블루프린트는 블루프린트 사용을 선택하고 블루프린트를 선택하여 사용할 수 있습니다. api-gateway-authorizer-python 그렇지 않으면, GitHub awslabs 리포지토리의 블루프린트 중 하나를 출발점으로 사용하는 것이 좋습니다.

이 단원에서 설명하는 Lambda 권한 부여자 함수 예제는 다른 서비스를 호출하지 않으므로 기본으로 제공되는 `AWSLambdaBasicExecutionRole`을 사용할 수 있습니다. 자체 API Gateway Lambda 권한 부여자를 위한 Lambda 함수를 생성할 때, 다른 서비스를 호출하는 경우 Lambda 함수에 IAM 실행 역할을 할당해야 합니다. AWS 역할을 생성하려면 AWS Lambda 실행 역할의 지침을 따르십시오.

Lambda 함수의 예시를 더 보려면 -blueprints on을 참조하십시오. aws-apigateway-lambda-authorizer GitHub 예제 애플리케이션은 오픈 뱅킹 브라질 - 인증 샘플 온을 참조하십시오. GitHub

예제: 토큰 기반의 Lambda 권한 부여자 함수 생성

토큰 기반의 Lambda 권한 부여자 함수를 만들려면 Lambda 콘솔에서 최근 런타임에 대해 다음 Node.js 코드를 입력합니다. 그런 다음 API Gateway 콘솔에서 권한 부여자를 테스트합니다.

토큰 기반 Lambda 권한 부여자 함수를 생성하려면

1. Lambda 콘솔에서 함수 생성(Create function)을 선택합니다.
2. 새로 작성을 선택합니다.
3. 함수 이름을 입력합니다.
4. 런타임에서 지원되는 최신 Node.js 또는 Python 런타임을 선택합니다.
5. 함수 생성을 선택합니다.
6. 다음 코드를 복사하여 코드 편집기에 붙여 넣습니다.

   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
```

7. 배포를 선택합니다.

Lambda 함수를 생성한 후 API Gateway 콘솔에서 토큰 기반 Lambda 권한 부여자를 생성하고 테스트합니다.

토큰 기반 Lambda 권한 부여자를 생성하려면

1. 단순 API가 없을 경우 API Gateway 콘솔에서 만듭니다.
2. API 목록에서 해당 API를 선택합니다.
3. 권한 부여자를 선택합니다.
4. 권한 부여자 생성을 선택합니다.
5. 권한 부여자 이름에 이름을 입력합니다.
6. 권한 부여자 유형으로 Lambda를 선택합니다.
7. Lambda 함수의 경우, Lambda 권한 부여 함수를 생성한 위치를 선택한 다음 AWS 리전 함수 이름을 입력합니다.
8. Lambda 호출 역할을 비워 둡니다.
9. Lambda 이벤트 페이로드로 토큰을 선택합니다.
10. 토큰 소스에 **authorizationToken**을 입력합니다.
11. 권한 부여자 생성을 선택합니다.

권한 부여자를 테스트하려면

1. 권한 부여자의 이름을 선택합니다.
2. 테스트 권한 부여자에서 authorizationToken 값에 **allow**를 입력합니다.
3. 테스트 권한 부여자를 선택합니다.

이 예제에서는 API가 메서드 요청을 받을 때 API Gateway가 소스 토큰을 이 Lambda 권한 부여자 함수의 `event.authorizationToken` 속성에 전달합니다. Lambda 권한 부여자 함수는 토큰을 읽고 다음과 같이 동작합니다.

- 토큰 값이 `'allow'`인 경우 권한 부여자 함수 테스트는 `200 OK` HTTP 응답 및 다음과 같은 IAM 정책을 반환하고 메서드 요청이 성공합니다.

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

- 토큰 값이 `'deny'`인 경우 권한 부여자 함수 테스트는 `200 OK` HTTP 응답 및 다음과 같은 `Deny` IAM 정책을 반환하고 메서드 요청이 실패합니다.

```
{
  "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**
>
> 테스트 환경 밖에서는 권한 부여자 함수가 `403 Forbidden` HTTP 응답을 반환하고 메서드 요청이 실패합니다.

- 토큰 값이 `'unauthorized'`이나 빈 문자열인 경우 권한 부여자 함수 테스트는 `401 Unauthorized` HTTP 응답을 반환하고 메서드 호출이 실패합니다.
- 그 외 토큰의 경우 클라이언트는 `500 Invalid token` 응답을 수신하고, 메서드 호출은 실패합니다.

> **Note**
>
> 프로덕션 코드에서는 권한 부여를 허용하기 전에 사용자를 인증해야 할 수 있습니다. 이 경우, 해당 공급자의 설명서에서 설명한 대로 인증 공급자를 호출하여 Lambda 함수에 인증 로직을 추가할 수 있습니다.

Lambda 권한 부여자 함수는 IAM 정책을 반환하는 것 외에 호출자의 보안 주체 ID도 반환해야 합니다. 선택적으로 통합 백엔드로 전달될 수 있는 추가 정보를 포함하는 `context` 객체를 반환할 수도 있습니다. 자세한 설명은 Amazon API Gateway Lambda 권한 부여자의 출력 섹션을 참조하세요.

예제: 요청 기반의 Lambda 권한 부여자 함수 생성

요청 기반의 Lambda 권한 부여자 함수를 만들려면 Lambda 콘솔에서 최근 런타임에 대해 다음 Node.js 코드를 입력합니다. 그런 다음 API Gateway 콘솔에서 권한 부여자를 테스트합니다.

1. Lambda 콘솔에서 함수 생성(Create function)을 선택합니다.
2. 새로 작성을 선택합니다.
3. 함수 이름을 입력합니다.
4. 런타임에서 지원되는 최신 Node.js 또는 Python 런타임을 선택합니다.
5. 함수 생성을 선택합니다.
6. 다음 코드를 복사하여 코드 편집기에 붙여 넣습니다.

Node.js

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

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

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

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

    // Perform authorization to return the Allow policy for correct parameters
 and
    // the 'Unauthorized' error, otherwise.
    var authResponse = {};
    var condition = {};
    condition.IpAddress = {};

    if (headers.headerauth1 === "headerValue1"
        && queryStringParameters.QueryString1 === "queryValue1"
        && stageVariables.StageVar1 === "stageValue1") {
        callback(null, generateAllow('me', event.methodArn));
```

```
    }  else {
        callback("Unauthorized");
    }
}

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

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

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

Python

```
# A simple request-based authorizer example to demonstrate how to use request
# parameters to allow or deny a request. In this example, a request is
```

```
# authorized if the client-supplied headerauth1 header, QueryString1
# query parameter, and stage variable of StageVar1 all match
# specified values of 'headerValue1', 'queryValue1', and 'stageValue1',
# respectively.

import json


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

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

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

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

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

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

    if (headers['headerauth1'] == "headerValue1" and
 queryStringParameters["QueryString1"]
            == "queryValue1" and stageVariables["StageVal1"] == "stageValue1"):
        response = generateAllow('me', event['methodArn'])
        print('authorized')
        return json.loads(response)
    else:
```

```
        print('unauthorized')
        raise Exception('Unauthorized') # Return a 401 Unauthorized response
        return 'unauthorized'

    # Help function to generate IAM policy


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

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

    authResponse_JSON = json.dumps(authResponse)

    return authResponse_JSON


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


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

7. 배포를 선택합니다.

Lambda 함수를 생성한 후 API Gateway 콘솔에서 요청 기반 Lambda 권한 부여자를 생성하고 테스트합니다.

요청 기반 Lambda 권한 부여자를 생성하려면

1. 단순 API가 없을 경우 API Gateway 콘솔에서 만듭니다.
2. API 목록에서 해당 API를 선택합니다.
3. 권한 부여자를 선택합니다.
4. 권한 부여자 생성을 선택합니다.
5. 권한 부여자 이름에 이름을 입력합니다.
6. 권한 부여자 유형으로 Lambda를 선택합니다.
7. Lambda 함수의 경우, Lambda 권한 부여 함수를 생성한 위치를 선택한 다음 AWS 리전 함수 이름을 입력합니다.
8. Lambda 호출 역할을 비워 둡니다.
9. Lambda 이벤트 페이로드로 요청을 선택합니다.
10. 자격 증명 소스 유형에서 다음을 입력합니다.
    a. 헤더를 선택하고 **headerauth1**을 입력한 다음 파라미터 추가를 선택합니다.
    b. 자격 증명 소스 유형에서 쿼리 문자열을 선택한 다음 **QueryString1**을 입력하고 파라미터 추가를 선택합니다.
    c. 자격 증명 소스 유형에서 스테이지 변수를 선택하고 **StageVar1**을 입력합니다.
11. 권한 부여자 생성을 선택합니다.

권한 부여자를 테스트하려면

1. 권한 부여자의 이름을 선택합니다.
2. 테스트 권한 부여자에서 다음을 입력합니다.
    a. 헤더를 선택하고 **headerValue1**을 입력한 다음 파라미터 추가를 선택합니다.
    b. 자격 증명 소스 유형에서 쿼리 문자열을 선택한 다음 **queryValue1**을 입력하고 파라미터 추가를 선택합니다.
    c. 자격 증명 소스 유형에서 스테이지 변수를 선택하고 **stageValue1**을 입력합니다.
3. 테스트 권한 부여자를 선택합니다.

이 예제에서 Lambda 권한 부여자 함수는 입력 파라미터를 확인하고 다음과 같이 동작합니다.

- 필요한 모든 파라미터 값이 예상 값과 일치하면 권한 부여자 함수가 `200 OK` HTTP 응답 및 다음과 같은 IAM 정책을 반환하고 메서드 요청이 성공합니다.

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

- 일치하지 않으면 권한 부여자 함수가 `401 Unauthorized` HTTP 응답을 반환하고 메서드 호출이 실패합니다.

> **Note**
>
> 프로덕션 코드에서는 권한 부여를 허용하기 전에 사용자를 인증해야 할 수 있습니다. 이 경우, 해당 공급자의 설명서에서 설명한 대로 인증 공급자를 호출하여 Lambda 함수에 인증 로직을 추가할 수 있습니다.

Lambda 권한 부여자 함수는 IAM 정책을 반환하는 것 외에 호출자의 보안 주체 ID도 반환해야 합니다. 선택적으로 통합 백엔드로 전달될 수 있는 추가 정보를 포함하는 `context` 객체를 반환할 수도 있습니다. 자세한 설명은 Amazon API Gateway Lambda 권한 부여자의 출력 섹션을 참조하세요.

API Gateway 콘솔을 사용하여 Lambda 권한 부여자 구성

Lambda 함수를 생성하고 함수가 작동하는지 확인한 후, API Gateway 콘솔에서 다음 단계를 사용해 API Gateway Lambda 권한 부여자(이전에는 사용자 지정 권한 부여자라고 함)를 구성합니다.

API Gateway 콘솔을 사용하여 Lambda 권한 부여자를 구성하려면

1. API Gateway 콘솔에 로그인합니다.
2. 새 API를 생성하거나 기존 API를 선택한 후 권한 부여자를 선택합니다.
3. 권한 부여자 생성을 선택합니다.

4. 권한 부여자 이름에 권한 부여자의 이름을 입력합니다.
5. 권한 부여자 유형으로 Lambda를 선택합니다.
6. Lambda 함수의 경우, Lambda 권한 부여 함수를 생성한 위치를 선택한 다음 AWS 리전 함수 이름을 입력합니다.
7. API Gateway 콘솔이 리소스 기반 정책을 설정하도록 Lambda 호출 역할을 비워 둡니다. 이 정책은 API Gateway 권한 부여자 Lambda 함수를 호출할 권한을 부여합니다. API Gateway가 권한 부여자 Lambda 함수를 호출하도록 허용하는 IAM 역할의 이름을 입력할 수도 있습니다. 이러한 역할의 예는 맡을 수 있는 IAM 역할 생성를 참조하세요.
8. Lambda 이벤트 페이로드에서 TOKEN 권한 부여자는 토큰을, REQUEST 권한 부여자는 요청을 선택합니다. (이는 유형 속성을 TOKEN 또는 REQUEST로 설정하는 것과 같습니다.)
9. 이전 단계에서의 선택에 따라 다음 중 하나를 수행합니다.

   a. 토큰 옵션의 경우 다음을 수행합니다.

      - 토큰 소스에 인증 토큰이 포함된 헤더 이름을 입력합니다. API 클라이언트는 권한 부여 토큰을 Lambda 권한 부여자에게 전송하려면 이 이름의 헤더를 포함해야 합니다.
      - 토큰 검증의 경우 명령문을 입력할 수도 있습니다. RegEx API Gateway는 이 표현식에 대해 입력 토큰의 초기 확인을 수행하고 확인이 성공할 경우 권한 부여자를 호출합니다. 이렇게 하면 API 호출을 줄일 수 있습니다.
      - 권한 부여자에 의해 생성된 권한 부여 정책을 캐싱하려면 권한 부여 캐싱을 설정해 둡니다. 정책 캐싱을 활성화한 경우 TTL 값을 수정할 수 있습니다. TTL을 0으로 설정하면 정책 캐싱이 비활성화됩니다. 정책 캐싱을 비활성화한 경우 토큰 소스에 지정된 헤더 이름이 캐시 키가 됩니다. 요청에서 이 헤더에 여러 값이 전달되면 모든 값이 순서가 유지된 상태로 캐시 키가 됩니다.

        > **Note**
        >
        > 기본 TTL 값은 300초입니다. 최대 TTL 값은 3600초이며, 이 제한은 늘릴 수 없습니다.

   b. 요청 옵션의 경우 다음을 수행합니다.

      - 자격 증명 소스 유형에서 파리미터 유형을 선택합니다. 지원되는 파라미터 유형은 Header, Query string, Stage variable 및 Context입니다. 자격 증명 소스를 더 추가하려면 파라미터 추가를 선택합니다.

- 권한 부여자에 의해 생성된 권한 부여 정책을 캐싱하려면 권한 부여 캐싱을 설정해 둡니다. 정책 캐싱을 활성화한 경우 TTL 값을 수정할 수 있습니다. TTL을 0으로 설정하면 정책 캐싱이 비활성화됩니다.

  API Gateway는 지정된 자격 증명 원본을 요청 권한 부여자 캐싱 키로 사용합니다. 캐싱을 활성화한 경우 API Gateway가 런타임 시 모든 지정된 자격 증명 원본이 존재하는 것을 성공적으로 확인한 경우에만 권한 부여자의 Lambda 함수를 호출합니다. 지정된 자격 증명 원본이 없거나, null이거나, 비어 있을 경우 API Gateway가 권한 부여자 Lambda 함수를 호출하지 않고 `401 Unauthorized` 응답을 반환합니다.

  여러 개의 자격 증명 소스가 정의된 경우, 이들은 모두 권한 부여자의 캐시 키를 도출하는 데 사용됩니다. 캐시 키 부분을 변경하면 권한 부여자가 캐시된 정책 문서를 폐기하고 새로 생성합니다. 요청에서 여러 값이 있는 헤더가 전달되면 모든 값이 순서가 유지된 상태로 캐시 키의 일부가 됩니다.

- 캐싱을 비활성화한 경우 자격 증명 소스를 지정할 필요가 없습니다. API Gateway는 요청을 권한 부여자 Lambda 함수에 직접 전달합니다.

> **Note**
>
> 캐싱을 활성화하려면 권한 부여자가 API 전체의 모든 메서드에 적용 가능한 정책을 반환해야 합니다. 메서드별 정책을 적용하려는 경우 권한 부여자 캐싱을 해제할 수 있습니다.

10. 권한 부여자 생성을 선택합니다.
11. API에 대해 권한 부여자가 생성되면, 메서드에 대해 권한 부여자를 구성하기 전에 권한 부여자를 테스트할 수 있습니다. 권한 부여자를 테스트하려면 권한 부여자의 이름을 선택합니다.
12. a. `TOKEN` 권한 부여자의 경우 토큰 값에 유효한 토큰을 입력합니다. 테스트 권한 부여자를 선택합니다. 토큰이 권한 부여자의 토큰 소스 설정에서 지정된 헤더로 Lambda 함수에 전달됩니다.

    b. `REQUEST` 권한 부여자의 경우 자격 증명 소스 유형에서 파라미터 유형을 선택하고 값을 입력합니다. 파라미터를 더 추가하려면 파라미터 추가를 선택합니다. 테스트 권한 부여자를 선택합니다.

API Gateway 콘솔을 사용하는 것 외에도 AWS CLI 또는 API Gateway용 AWS SDK를 사용하여 권한 부여자 호출을 테스트할 수 있습니다. 를 사용하여 이 작업을 수행하려면 을 참조하십시오. AWS CLItest-invoke-authorizer

> **Note**
>
> 메서드 실행 test-invoke와 권한 부여자 test-invoke는 독립된 프로세스입니다.
> API Gateway 콘솔을 사용하여 메서드 호출을 테스트하려면 콘솔을 사용하여 REST API 메서드 테스트 단원을 참조하세요. 를 사용하여 메서드 호출을 AWS CLI테스트하려면 을 참조하십시오 test-invoke-method.
> 메서드 및 구성된 권한 부여자 호출을 테스트하려면 API를 배포한 후, cURL 또는 Postman을 사용해 필요한 토큰 또는 요청 파라미터를 제공하여 메서드를 호출합니다.

다음 절차에서는 Lambda 권한 부여자를 사용하도록 API 메서드를 구성하는 방법을 보여줍니다.

Lambda 권한 부여자를 사용하도록 API 메서드를 구성하려면

1. 리소스를 선택합니다. 새 메서드를 선택하거나 기존 메서드를 선택합니다. 필요하다면 새 리소스를 만듭니다.
2. 메서드 요청 탭의 메서드 요청 설정에서 편집을 선택합니다.
3. 권한 부여자의 경우 드롭다운 메뉴에서 방금 생성한 Lambda 권한 부여자를 선택합니다.
4. (선택 사항) 권한 부여 토큰을 백엔드로 전달하려면 HTTP 요청 헤더를 선택합니다. 헤더 추가를 선택한 다음 인증 헤더의 이름을 추가합니다. 이름에서, API에 대한 Lambda 권한 부여자를 생성할 때 지정한 토큰 소스 이름과 일치하는 헤더 이름을 입력합니다. `REQUEST` 권한 부여자에는 이 단계가 적용되지 않습니다.
5. 저장을 선택합니다.
6. Deploy API(API 배포)를 선택하여 단계에 API를 배포합니다. Invoke URL(URL 호출) 값을 메모합니다. API 호출 시 필요합니다. 스테이지 변수를 사용하는 `REQUEST` 권한 부여자의 경우 스테이지 페이지가 열려 있는 동안 필요한 스테이지 변수를 지정하고 값을 지정해야 합니다.

Amazon API Gateway Lambda 권한 부여자에 대한 입력

`TOKEN` 입력 형식

`TOKEN` 유형 Lambda 권한 부여자(이전에는 사용자 지정 권한 부여자라고 함)의 경우 API에 대해 권한 부여자를 구성할 때 사용자 지정 헤더를 토큰 원본으로 지정해야 합니다. API 클라이언트는 수신되는 요청에 있는 해당 헤더의 권한 부여 토큰을 전달해야 합니다. 메서드 요청을 수신하면 API Gateway는 사용자 지정 헤더로부터 토큰을 추출합니다. 그런 다음 토큰을 Lambda 함수 내 `event` 객체의 `authorizationToken` 속성으로 전달하고 메서드 ARN을 `methodArn` 속성으로 전달합니다.

```
{
    "type":"TOKEN",
    "authorizationToken":"{caller-supplied-token}",
    "methodArn":"arn:aws:execute-api:{regionId}:{accountId}:{apiId}/{stage}/{httpVerb}/
[{resource}/[{child-resources}]]"
}
```

이 예제에서 type 속성은 권한 부여자 유형(TOKEN 권한 부여자)을 지정합니다. *{caller-supplied-token}*의 출처는 클라이언트 요청의 권한 부여 헤더이며 모든 문자열 값이 될 수 있습니다. methodArn은 수신 메서드 요청의 ARN이며 API Gateway에서 Lambda 권한 부여자 구성에 따라 채워집니다.

REQUEST 입력 형식

REQUEST 유형의 Lambda 권한 부여자의 경우, API Gateway가 요청 파라미터를 event 객체의 일부로 Lambda 함수에 전달합니다. 이러한 요청 파라미터에는 헤더, 경로 파라미터, 쿼리 문자열 파라미터, 단계 변수 및 일부 요청 컨텍스트 변수가 포함됩니다. API 호출자는 경로 파라미터, 헤더 및 쿼리 문자열 파라미터를 설정할 수 있습니다. API 개발자는 API를 배포할 때 단계 변수를 설정해야 하며, API Gateway는 런타임 시 요청 컨텍스트를 제공합니다.

> **Note**
>
> 경로 파라미터는 Lambda 권한 부여자 함수에 요청 파라미터로 전달될 수 있지만, 자격 증명 원본으로 사용될 수는 없습니다.

다음 예제는 프록시 통합을 포함하는 API 메서드(REQUEST)에 대한 GET /request 권한 부여자에 대한 입력입니다.

```
{
  "type": "REQUEST",
  "methodArn": "arn:aws:execute-api:us-east-1:123456789012:abcdef123/test/GET/request",
  "resource": "/request",
  "path": "/request",
  "httpMethod": "GET",
  "headers": {
    "X-AMZ-Date": "20170718T062915Z",
    "Accept": "*/*",
    "HeaderAuth1": "headerValue1",
    "CloudFront-Viewer-Country": "US",
```

```
    "CloudFront-Forwarded-Proto": "https",
    "CloudFront-Is-Tablet-Viewer": "false",
    "CloudFront-Is-Mobile-Viewer": "false",
    "User-Agent": "..."
  },
  "queryStringParameters": {
    "QueryString1": "queryValue1"
  },
  "pathParameters": {},
  "stageVariables": {
    "StageVar1": "stageValue1"
  },
  "requestContext": {
    "path": "/request",
    "accountId": "123456789012",
    "resourceId": "05c7jb",
    "stage": "test",
    "requestId": "...",
    "identity": {
      "apiKey": "...",
      "sourceIp": "...",
      "clientCert": {
        "clientCertPem": "CERT_CONTENT",
        "subjectDN": "www.example.com",
        "issuerDN": "Example issuer",
        "serialNumber": "a1:a1:a1:a1:a1:a1:a1:a1:a1:a1:a1:a1:a1:a1:a1:a1",
        "validity": {
          "notBefore": "May 28 12:30:02 2019 GMT",
          "notAfter": "Aug  5 09:36:04 2021 GMT"
        }
      }
    },
    "resourcePath": "/request",
    "httpMethod": "GET",
    "apiId": "abcdef123"
  }
}
```

`requestContext`는 키-값 페어의 맵이며 $context 변수에 해당합니다. 결과는 API 종속적입니다. API Gateway는 새 키를 맵에 추가할 수 있습니다. Lambda 프록시 통합에서의 Lambda 함수에 대한 자세한 내용은 프록시 통합에 대한 Lambda 함수의 입력 형식 단원을 참조하세요.

Amazon API Gateway Lambda 권한 부여자의 출력

Lambda 권한 부여자 함수의 출력은 사전과 같은 객체이며, 이 객체에는 보안 주체 ID(`principalId`)와 정책 설명 목록이 들어 있는 정책 문서(`policyDocument`)가 포함되어 있어야 합니다. 또한 키-값 페어를 포함하는 `context` 맵을 포함할 수 있습니다. API가 사용량 계획을 사용하는 경우 (`apiKeySource`가 `AUTHORIZER`로 설정됨), Lambda 권한 부여자 함수는 `usageIdentifierKey` 속성 값으로 사용량 계획의 API 키 중 하나를 반환해야 합니다.

다음은 이 출력의 예입니다.

```
{
  "principalId": "yyyyyyyy", // The principal user identification associated with the
 token sent by the client.
  "policyDocument": {
    "Version": "2012-10-17",
    "Statement": [
      {
        "Action": "execute-api:Invoke",
        "Effect": "Allow|Deny",
        "Resource": "arn:aws:execute-
api:{regionId}:{accountId}:{apiId}/{stage}/{httpVerb}/[{resource}/[{child-resources}]]"
      }
    ]
  },
  "context": {
    "stringKey": "value",
    "numberKey": "1",
    "booleanKey": "true"
  },
  "usageIdentifierKey": "{api-key}"
}
```

여기서 정책 설명은 API Gateway 실행 서비스에서 지정된 API 메서드(`Effect`)를 호출(`Action`)하도록 허용할지 거부할지(`Resource`) 여부를 지정합니다. 와일드카드(`*`)를 사용하여 리소스 유형(메서드)를 지정할 수 있습니다. API 호출에 대한 유효한 정책을 설정하는 방법은 API Gateway에서 API 실행을 위한 IAM 정책의 설명 참조 단원을 참조하세요.

권한 부여가 활성화된 메서드 ARN의 경우(예: `arn:aws:execute-api:{regionId}:{accountId}:{apiId}/{stage}/{httpVerb}/[{resource}/[{child-resources}]]`) 최대 길이는 1,600바이트입니다. 경로 파라미터 값은 실행 시간에 값 크기가 결정되

기 때문에 ARN 길이가 한도를 초과하게 될 가능성이 있습니다. 이런 일이 발생하면 API 클라이언트는 `414 Request URI too long` 응답을 받습니다.

뿐만 아니라 권한 부여자의 정책 설명 출력 내용과 마찬가지로 리소스 ARN은 현재 512자로 제한되어 있습니다. 이런 이유 때문에 요청 URI에서는 길이가 상당히 긴 JWT 토큰이 있는 URI를 사용해서는 안 됩니다. 그 대신에 요청 헤더에서 JWT 토큰을 안전하게 전달할 수도 있습니다.

`principalId` 변수를 사용하여 매핑 템플릿에서 `$context.authorizer.principalId` 값에 액세스할 수 있습니다. 이 기능은 값을 백엔드에 전달하려는 경우에 유용합니다. 자세한 설명은 $context데이터 모델, 권한 부여자, 매핑 템플릿, 액세스 로깅용 변수 CloudWatch 섹션을 참조하세요.

각각 `stringKey`, `numberKey` 또는 `booleanKey`를 호출하여 매핑 템플릿에서 `"value"` 맵의 `"1"`, `"true"` 또는 `context` 값(예: `$context.authorizer.stringKey`, `$context.authorizer.numberKey` 또는 `$context.authorizer.booleanKey`)에 액세스할 수 있습니다. 반환되는 값은 모두 문자열화됩니다. JSON 객체 또는 어레이를 `context` 맵의 유효한 키 값으로 설정할 수 없습니다.

`context` 맵을 사용하면 통합 요청 매핑 템플릿을 사용해 캐시된 자격 증명을 권한 부여자에서 백엔드로 반환할 수 있습니다. 그러면 캐시된 자격 증명을 사용해 모든 요청에 대해 보안 키에 액세스하고 권한 부여 토큰을 열 필요를 줄일 수 있으므로 백엔드가 개선된 사용자 경험을 제공할 수 있습니다.

Lambda 프록시 통합의 경우 API Gateway가 `context` 객체를 입력 `event`의 일부로 Lambda 권한 부여자에서 백엔드 Lambda 함수로 직접 전달합니다. `$event.requestContext.authorizer.`*`key`*를 호출하여 Lambda 함수에서 `context` 키-값 페어를 검색할 수 있습니다.

`{api-key}`는 API 단계의 사용량 계획에서 API 키를 뜻합니다. 자세한 설명은 the section called “사용량 계획” 섹션을 참조하세요.

다음 예제는 Lambda 권한 부여자의 출력을 보여줍니다. 예제 출력에는 AWS 계정 () 의 API (Deny) dev 단계에서 GET 메서드에 대한 () 호출을 차단하기 위한 정책 명령문이 포함되어 있습니다. ymy8tbxw7b 123456789012

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

```
      {
        "Action": "execute-api:Invoke",
        "Effect": "Deny",
        "Resource": "arn:aws:execute-api:us-west-2:123456789012:ymy8tbxw7b/dev/GET/"
      }
    ]
  }
}
```

API Gateway Lambda 권한 부여자를 사용하여 API 호출

Lambda 권한 부여자(이전에는 사용자 지정 권한 부여자라고 함)를 구성하고 API를 배포했으면 Lambda 권한 부여자가 활성화된 상태에서 API를 테스트해야 합니다. 이를 위해서는 REST 클라이언트(예: cURL 또는 Postman)가 필요합니다. 다음 예제에서는 Postman을 사용합니다.

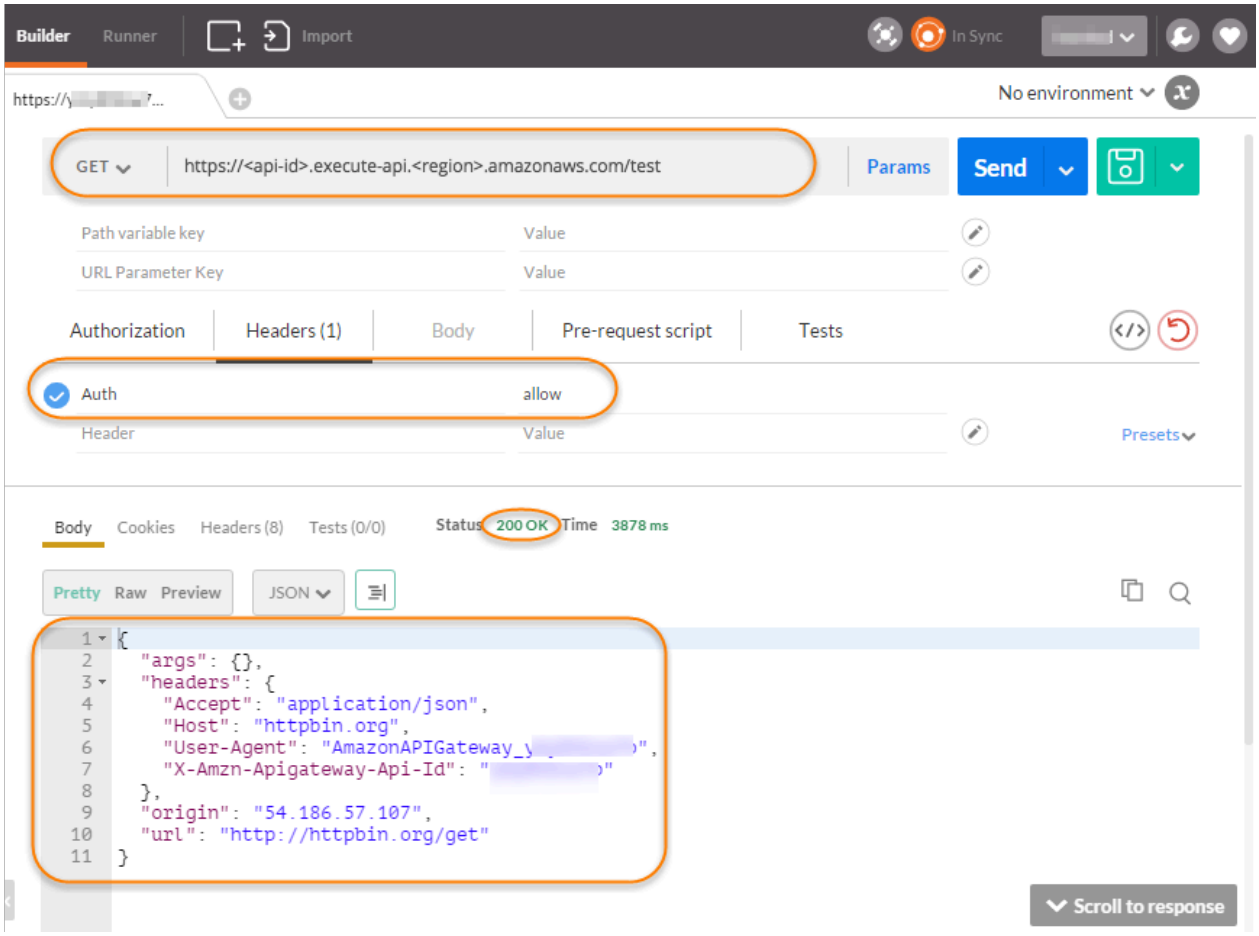
> **Note**
>
> 권한 부여자가 활성화된 메서드를 호출할 때 권한 부여자에 대한 필수 토큰이 설정되지 않았거나, null이거나, 지정된 토큰 검증 표현식에 의해 무효화된 CloudWatch 경우 API Gateway는 호출을 기록하지 않습니다. **TOKEN** 마찬가지로, REQUEST 권한 부여자의 필수 ID 소스가 설정되지 않았거나, null이거나, 비어 있는 CloudWatch 경우 API Gateway는 호출을 기록하지 않습니다.

다음에서는 Postman을 사용하여 Lambda TOKEN 권한 부여자로 API를 호출하거나 테스트하는 방법을 보여줍니다. 이 방법은 필요한 경로, 헤더 또는 쿼리 문자열 파라미터를 명시적으로 지정할 경우 Lambda REQUEST 권한 부여자를 사용해 API를 호출하는 데 적용할 수 있습니다.

사용자 지정 **TOKEN** 권한 부여자를 사용해 API를 호출하려면

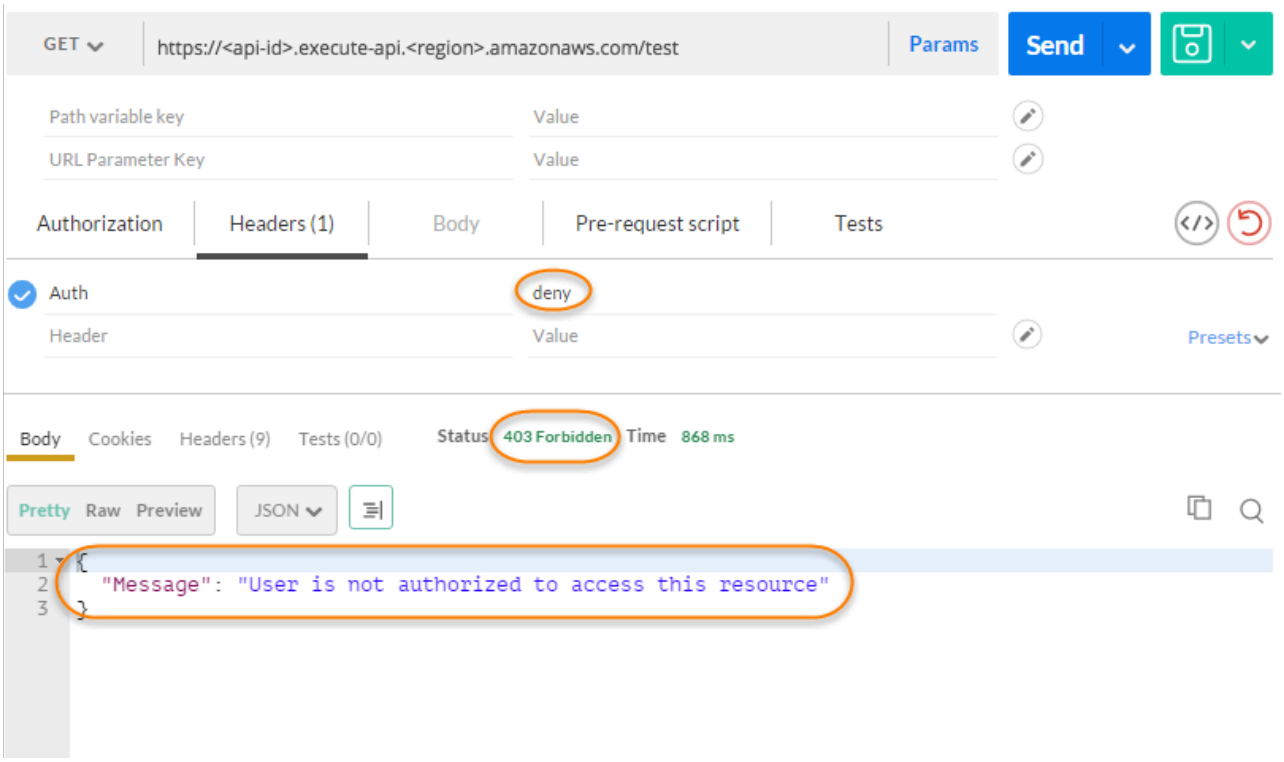
1. Postman을 열고 GET 메서드를 선택한 후 API의 Invoke URL(URL 호출)을 인접한 URL 필드에 붙여 넣습니다.

   Lambda 권한 부여 토큰 헤더를 추가하고 값을 allow로 설정합니다. Send(전송)를 선택합니다.



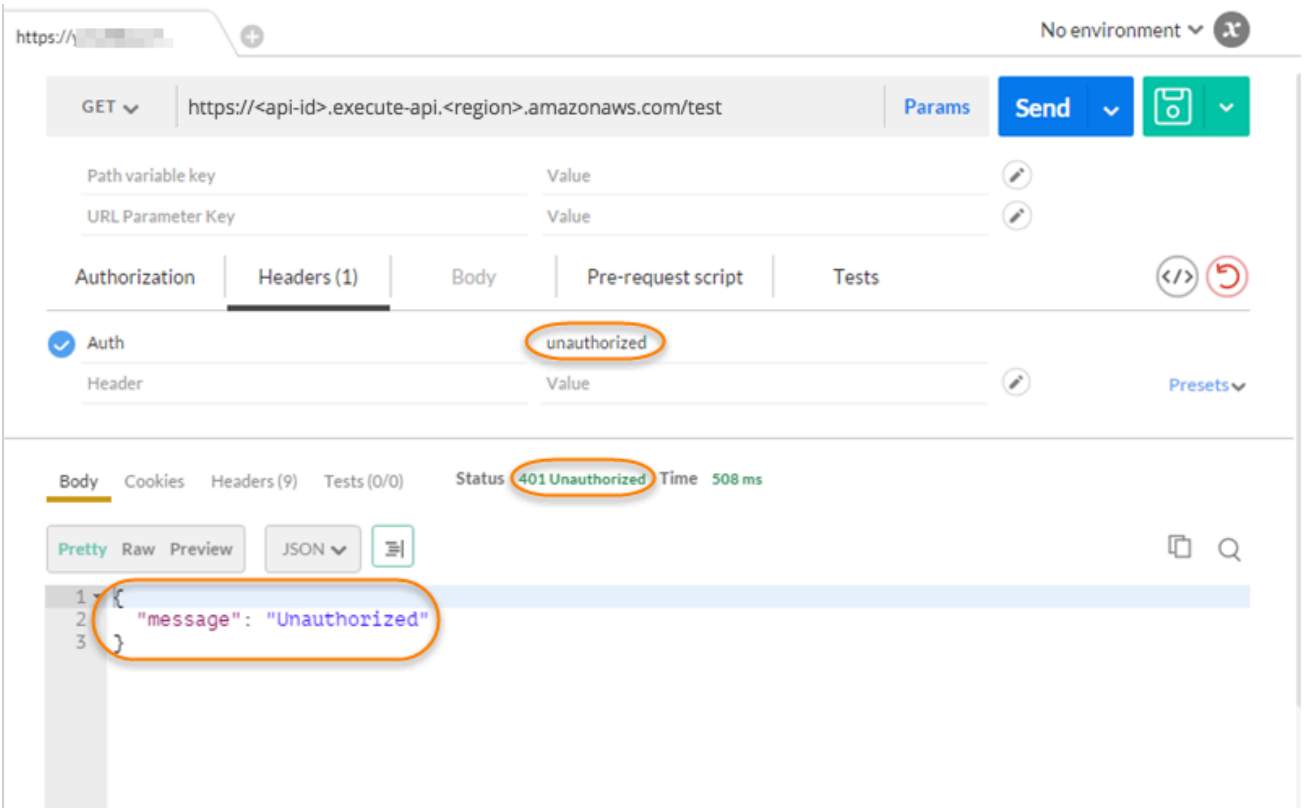
응답을 보면, API Gateway Lambda 권한 부여자가 200 OK 응답을 반환하고, 메서드와 통합된 HTTP 엔드포인트(http://httpbin.org/get)에 액세스할 수 있는 호출 권한을 부여함을 알 수 있습니다.

2. Postman에 머물면서 Lambda 권한 부여 토큰 헤더 값을 deny로 변경합니다. Send(전송)를 선택합니다.



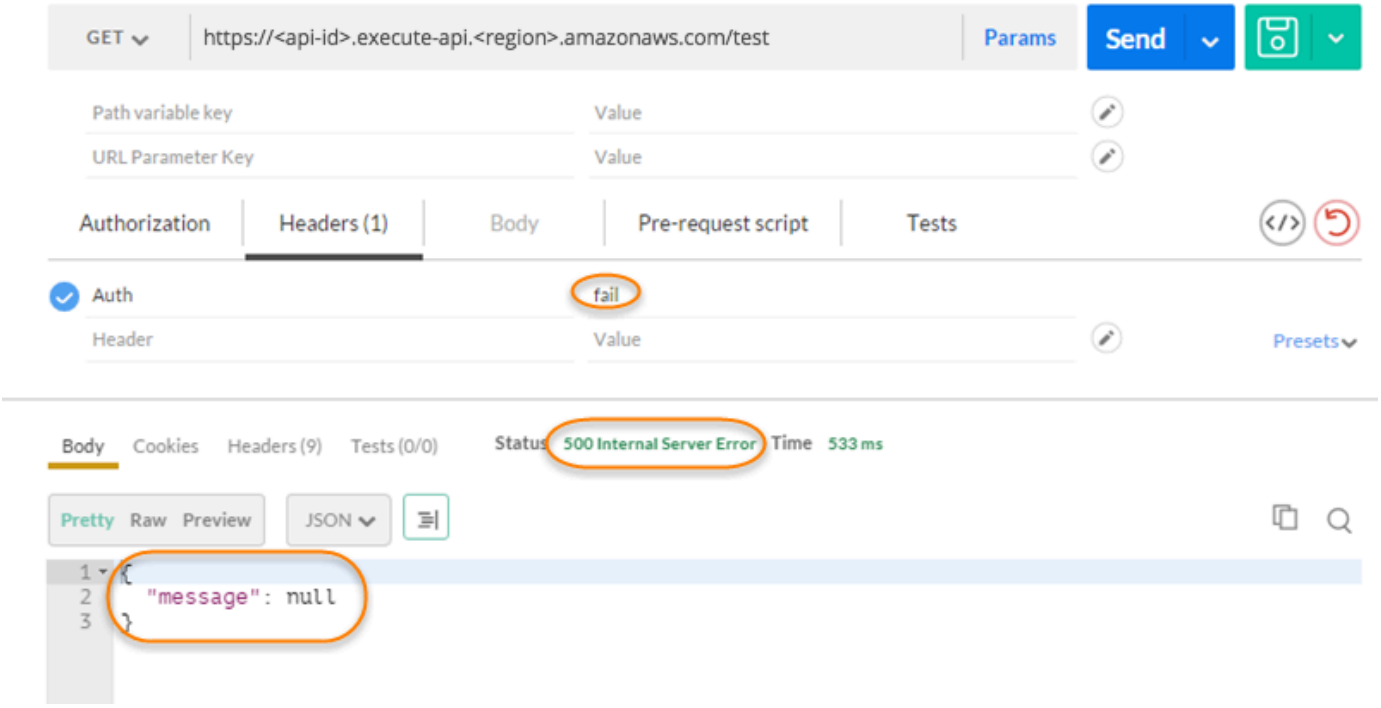
응답을 보면, API Gateway Lambda 권한 부여자가 403 Forbidden 응답을 반환하고, HTTP 엔드포인트에 액세스할 수 있는 호출 권한을 부여하지 않음을 알 수 있습니다.

3. Postman에서 Lambda 권한 부여 토큰 헤더 값을 `unauthorized`로 변경하고 전송을 선택합니다.



응답을 보면, API Gateway가 401 Unauthorized 응답을 반환하고, HTTP 엔드포인트에 액세스할 수 있는 호출 권한을 부여하지 않음을 알 수 있습니다.

4. 이제 Lambda 권한 부여 토큰 헤더 값을 `fail`로 변경합니다. Send(전송)를 선택합니다.



응답을 보면, API Gateway가 500 Internal Server Error 응답을 반환하고, HTTP 엔드포인트에 액세스할 수 있는 호출 권한을 부여하지 않음을 알 수 있습니다.

교차 계정 Lambda 권한 부여자 구성

이제 다른 AWS 계정의 AWS Lambda 함수를 API 권한 부여 함수로 사용할 수도 있습니다. 각 계정은 Amazon API Gateway를 사용할 수 있는 모든 리전에 존재할 수 있습니다. Lambda 권한 부여자 함수는 OAuth 또는 SAML 같은 보유자 토큰 인증 전략을 사용할 수 있습니다. 따라서 여러 API Gateway API에 대해 Lambda 권한 부여자 함수를 중앙에서 손쉽게 관리 및 공유할 수 있습니다.

이 단원에서는 Amazon API Gateway 콘솔을 사용하여 교차 계정 Lambda 권한 부여자 함수를 구성하는 방법을 보여줍니다.

이 지침에서는 이미 한 AWS 계정에는 API Gateway API가 있고 다른 계정에는 Lambda 권한 부여 함수가 있다고 가정합니다.

API Gateway 콘솔을 사용하여 교차 계정 Lambda 권한 부여자 구성

API를 포함하고 있는 계정에서 Amazon API Gateway 콘솔에 로그인하고 다음을 수행합니다.

1. API를 선택하고 기본 탐색 창에서 권한 부여자를 선택합니다.
2. 권한 부여자 생성을 선택합니다.
3. 권한 부여자 이름에 권한 부여자의 이름을 입력합니다.
4. 권한 부여자 유형으로 Lambda를 선택합니다.
5. Lambda 함수에서 두 번째 계정에서 생성한 Lambda 권한 부여자 함수의 정식 ARN을 입력합니다.

   > **Note**
   >
   > Lambda 콘솔 창의 오른쪽 위 모서리에서 함수의 ARN을 찾을 수 있습니다.

6. `aws lambda add-permission` 명령 문자열과 함께 경고가 표시됩니다. 이 정책은 권한 부여자 Lambda 함수를 호출할 API Gateway 권한을 부여합니다. 나중에 사용하기 위해 명령을 복사하여 저장합니다. 권한 부여자를 생성한 후 명령을 실행합니다.
7. API Gateway 콘솔이 리소스 기반 정책을 설정하도록 Lambda 호출 역할을 비워 둡니다. 이 정책은 API Gateway 권한 부여자 Lambda 함수를 호출할 권한을 부여합니다. API Gateway가 권한 부여자 Lambda 함수를 호출하도록 허용하는 IAM 역할을 입력할 수도 있습니다. 이러한 역할의 예는 맡을 수 있는 IAM 역할 생성를 참조하세요.

8. Lambda 이벤트 페이로드에서 TOKEN 권한 부여자는 토큰을, REQUEST 권한 부여자는 요청을 선택합니다.
9. 이전 단계에서의 선택에 따라 다음 중 하나를 수행합니다.

   a. 토큰 옵션의 경우 다음을 수행합니다.

   - 토큰 소스에 인증 토큰이 포함된 헤더 이름을 입력합니다. API 클라이언트는 권한 부여 토큰을 Lambda 권한 부여자에게 전송하려면 이 이름의 헤더를 포함해야 합니다.
   - 선택적으로 토큰 검증의 경우 명령문을 입력합니다. RegEx API Gateway는 이 표현식에 대해 입력 토큰의 초기 확인을 수행하고 확인이 성공할 경우 권한 부여자를 호출합니다. 이렇게 하면 API 호출을 줄일 수 있습니다.
   - 권한 부여자에 의해 생성된 권한 부여 정책을 캐싱하려면 권한 부여 캐싱을 설정해 둡니다. 정책 캐싱을 활성화한 경우 TTL 값을 수정할 수 있습니다. TTL을 0으로 설정하면 정책 캐싱이 비활성화됩니다. 정책 캐싱을 비활성화한 경우 토큰 소스에 지정된 헤더 이름이 캐시 키가 됩니다. 요청에서 이 헤더에 여러 값이 전달되면 모든 값이 순서가 유지된 상태로 캐시 키가 됩니다.

     > **Note**
     >
     > 기본 TTL 값은 300초입니다. 최대 TTL 값은 3600초이며, 이 제한은 늘릴 수 없습니다.

   b. 요청 옵션의 경우 다음을 수행합니다.

   - 자격 증명 소스 유형에서 파리미터 유형을 선택합니다. 지원되는 파라미터 유형은 Header, Query string, Stage variable 및 Context입니다. 자격 증명 소스를 더 추가하려면 파라미터 추가를 선택합니다.
   - 권한 부여자에 의해 생성된 권한 부여 정책을 캐싱하려면 권한 부여 캐싱을 설정해 둡니다. 정책 캐싱을 활성화한 경우 TTL 값을 수정할 수 있습니다. TTL을 0으로 설정하면 정책 캐싱이 비활성화됩니다.

     API Gateway는 지정된 자격 증명 원본을 요청 권한 부여자 캐싱 키로 사용합니다. 캐싱을 활성화한 경우 API Gateway가 런타임 시 모든 지정된 자격 증명 원본이 존재하는 것을 성공적으로 확인한 경우에만 권한 부여자의 Lambda 함수를 호출합니다. 지정된 자격 증명 원본이 없거나, null이거나, 비어 있을 경우 API Gateway가 권한 부여자 Lambda 함수를 호출하지 않고 401 Unauthorized 응답을 반환합니다.

여러 개의 자격 증명 소스가 정의된 경우, 이들은 모두 권한 부여자의 캐시 키를 도출하는 데 사용됩니다. 캐시 키 부분을 변경하면 권한 부여자가 캐시된 정책 문서를 폐기하고 새로 생성합니다. 요청에서 여러 값이 있는 헤더가 전달되면 모든 값이 순서가 유지된 상태로 캐시 키의 일부가 됩니다.

- 캐싱을 비활성화한 경우 자격 증명 소스를 지정할 필요가 없습니다.

> **Note**
>
> 캐싱을 활성화하려면 권한 부여자가 API 전체의 모든 메서드에 적용 가능한 정책을 반환해야 합니다. 메서드별 정책을 적용하려는 경우 권한 부여자 캐싱을 해제할 수 있습니다.

10. 권한 부여자 생성을 선택합니다.
11. 이전 단계에서 복사한 `aws lambda add-permission` 명령 문자열을 두 번째 계정용으로 구성된 AWS CLI 창에 붙여넣습니다. `AUTHORIZER_ID`를 권한 부여자의 ID로 바꿉니다. 이렇게 하면 첫 번째 계정이 두 번째 계정의 Lambda 권한 부여자 함수에 액세스할 수 있는 권한이 부여됩니다.

Amazon Cognito 사용자 풀을 권한 부여자로 사용하여 REST API에 대한 액세스 제어

IAM 역할 및 정책 또는 Lambda 권한 부여자(이전에는 사용자 지정 권한 부여자라고 함)를 사용하는 것 말고도 Amazon Cognito 사용자 풀을 사용하여 Amazon API Gateway에서 API에 액세스할 수 있는 사람을 제어할 수도 있습니다.

API에 Amazon Cognito 사용자 풀을 사용하려면 먼저 `COGNITO_USER_POOLS` 유형의 권한 부여자를 생성한 다음 해당 권한 부여자를 사용하도록 API 메서드를 구성해야 합니다. API가 배포된 후 클라이언트는 먼저 사용자 풀에 사용자를 로그인하고, 해당 사용자의 자격 증명 또는 액세스 토큰을 획득한 다음 일반적으로 요청의 `Authorization` 헤더로 설정되는 토큰 1개를 사용하여 API 메서드를 호출해야 합니다. API 호출은 필요한 토큰이 제공되고 제공된 토큰이 유효한 경우에만 성공하며, 그렇지 않으면 권한이 부여될 수 있는 자격 증명이 클라이언트에게 없기 때문에 호출할 권한이 클라이언트에게 부여되지 않습니다.

자격 증명 토큰은 로그인한 사용자의 자격 증명 클레임을 기반으로 API 호출 권한을 부여하는 데 사용됩니다. 액세스 토큰은 지정된 액세스 보호 리소스의 사용자 지정 범위를 기반으로 API 호출 권한을 부여하는 데 사용됩니다. 자세한 내용은 사용자 풀에 토큰 사용과 리소스 서버 및 사용자 지정 범위를 참조하세요.

API를 위한 Amazon Cognito 사용자 풀을 생성하고 구성하려면 다음 작업을 수행하세요.

- Amazon Cognito 콘솔, CLI/SDK 또는 API를 사용하여 사용자 풀을 생성하거나 다른 AWS 계정이 소유한 사용자 풀을 사용합니다.
- API Gateway 콘솔, CLI/SDK 또는 API를 사용하여 선택한 사용자 풀이 포함된 API Gateway 권한 부여자를 생성합니다.
- 선택한 API 메서드에서 API Gateway 콘솔, CLI/SDK 또는 API를 사용하여 권한 부여자를 활성화합니다.

사용자 풀이 활성화된 상태에서 API 메서드를 호출하기 위해 API 클라이언트는 다음 작업을 수행합니다.

- Amazon Cognito CLI/SDK 또는 API를 사용하여 선택한 사용자 풀에 사용자를 로그인하고 자격 증명 토큰 또는 액세스 토큰을 획득합니다. SDK 사용에 대한 자세히 내용은 AWS SDK를 사용한 Amazon Cognito 코드 예시를 참조하세요.
- 클라이언트별 프레임워크를 사용하여 배포된 API Gateway API를 호출하고 `Authorization` 헤더에서 적절한 토큰을 제공합니다.

API 개발자는 클라이언트 개발자에게 사용자 풀 ID와 클라이언트 ID 및 가능한 경우, 사용자 풀의 일부로 정의되는 연결된 클라이언트 비밀번호를 제공해야 합니다.

> **Note**
>
> 사용자가 Amazon Cognito 자격 증명을 사용하여 로그인하고 IAM 역할 권한과 함께 사용할 임시 자격 증명도 획득하게 하려면 Amazon Cognito 연동 ID를 사용합니다. 각 API 리소스 엔드포인트 HTTP 메서드에 대해 권한 부여 유형을 설정하고 `Method Execution` 범주를 `AWS_IAM`으로 설정합니다.

이 단원에서는 사용자 풀을 생성하고, API Gateway API를 사용자 풀과 통합하고, 사용자 풀에 통합된 API를 호출하는 방법을 설명합니다.

주제

- REST API를 위한 Amazon Cognito 사용자 풀 권한 부여자를 생성할 수 있는 권한 획득
- REST API에 대한 Amazon Cognito 사용자 풀 생성
- REST API와 Amazon Cognito 사용자 풀 통합
- Amazon Cognito 사용자 풀과 통합된 REST API 호출

- API Gateway 콘솔을 사용하여 REST API를 위한 교차 계정 Amazon Cognito 권한 부여자 구성
- AWS CloudFormation을 사용하여 REST API를 위한 Amazon Cognito 권한 부여자 생성

REST API를 위한 Amazon Cognito 사용자 풀 권한 부여자를 생성할 수 있는 권한 획득

Amazon Cognito 사용자 풀이 포함된 권한 부여자를 생성하려면 선택한 Amazon Cognito 사용자 풀이 포함된 권한 부여자를 생성하거나 업데이트할 수 있는 `Allow` 권한이 있어야 합니다. 다음 IAM 정책 문서는 그러한 권한의 예시를 보여줍니다.

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

```
                    ]
                }
            }
        }
    ]
}
```

정책은 소속 IAM 그룹 또는 할당된 IAM 역할에 연결되어야 합니다.

이전 정책 문서에서 `apigateway:POST` 작업은 새로운 권한 부여자를 생성하는 것이고, `apigateway:PATCH` 작업은 기존 권한 부여자를 업데이트하는 것입니다. `Resource` 값의 처음 2개의 와일드카드(*) 문자를 각각 재정의함으로써 정책을 특정 리전 또는 특정 API로 제한할 수 있습니다.

여기에 사용된 `Condition` 절은 `Allowed` 권한을 지정된 사용자 풀로 제한합니다. `Condition` 절이 존재하면 조건에 일치하지 않는 모든 사용자 풀에 대한 액세스가 거부됩니다. 권한에 `Condition` 절이 없으면 모든 사용자 풀에 대한 액세스가 허용됩니다.

`Condition` 절을 설정하는 옵션은 다음과 같습니다.

- `ArnLike` 또는 `ArnEquals` 조건 표현식을 설정하여 특정 사용자 풀만을 보유한 `COGNITO_USER_POOLS` 권한 부여자 생성 또는 업데이트를 허용할 수 있습니다.
- `ArnNotLike` 또는 `ArnNotEquals` 조건식을, 조건식에서 지정되지 않은 모든 사용자 풀이 포함된 `COGNITO_USER_POOLS` 권한 부여자 생성 또는 업데이트를 허용하도록 설정할 수 있습니다.
- `Condition` 절을 생략하여 모든 AWS 계정 및 모든 리전의 모든 사용자 풀이 포함된 `COGNITO_USER_POOLS` 권한 부여자 생성 또는 업데이트를 허용할 수 있습니다.

Amazon 리소스 이름(ARN) 조건식에 대한 자세한 내용은 Amazon 리소스 이름 조건 연산자 단원을 참조하세요. 예에 나온 것처럼 `apigateway:CognitoUserPoolProviderArn`은 `COGNITO_USER_POOLS` 유형의 API Gateway 권한 부여자와 함께 사용할 수 있거나 사용할 수 없는 `COGNITO_USER_POOLS` 사용자 풀의 ARN 목록입니다.

REST API에 대한 Amazon Cognito 사용자 풀 생성

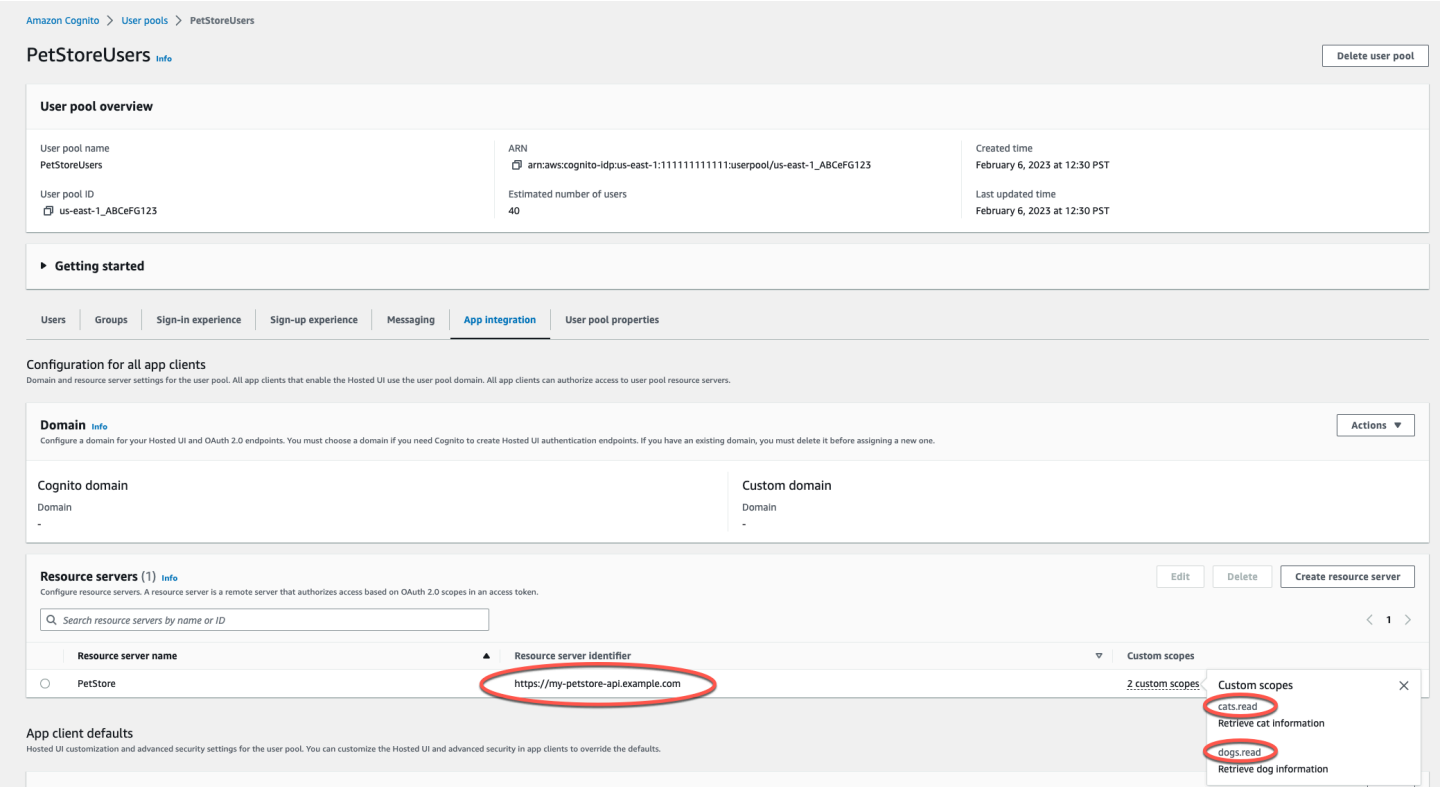
API를 사용자 풀과 통합하기 전에 Amazon Cognito에서 사용자 풀을 생성해야 합니다. 사용자 풀 구성은 Amazon Cognito의 모든 리소스 할당량을 준수해야 합니다. 그룹, 사용자 및 역할과 같은 모든 사용자 정의 Amazon Cognito 변수가 영숫자 문자만 사용해야 합니다. 사용자 풀을 생성하는 방법에 대한 지침은 Amazon Cognito 개발자 가이드에서 자습서: 사용자 풀 생성을 참조하세요.

사용자 풀 ID, 클라이언트 ID 및 모든 클라이언트 비밀번호를 적어 둡니다. 사용자가 사용자 풀에 등록하고, 사용자 풀에 로그인하고, 사용자 풀에 구성된 API 메서드를 호출하기 위해 요청에 포함할 자격

증명 또는 액세스 토큰을 획득하려면 클라이언트가 Amazon Cognito에 이러한 ID 및 클라이언트 비밀번호를 제공해야 합니다. 또한 다음에 설명한 대로 사용자 풀을 API Gateway에서 권한 부여자로 구성할 경우 사용자 풀 이름을 지정해야 합니다.

액세스 토큰을 사용하여 API 메서드 호출 권한을 부여하는 경우, 사용자 풀과의 앱 통합을 구성하여 지정된 리소스 서버에서 원하는 사용자 지정 범위를 설정해야 합니다. Amazon Cognito 사용자 풀을 통해 토큰 사용에 대한 자세한 내용은 사용자 풀을 통해 토큰 사용을 참조하세요. 리소스 서버에 대한 자세한 내용은 사용자 풀의 리소스 서버 정의를 참조하세요.

구성된 리소스 서버 식별자와 사용자 지정 범위 이름을 적어 둡니다. 이 정보는 `COGNITO_USER_POOLS` 권한 부여자가 사용하는 OAuth Scopes(OAuth 범위)의 액세스 범위 전체 이름을 생성하는 데 필요합니다.



REST API와 Amazon Cognito 사용자 풀 통합

API Gateway에서 Amazon Cognito 사용자 풀을 생성한 다음 이 사용자 풀을 사용하는 `COGNITO_USER_POOLS` 권한 부여자를 생성해야 합니다. 다음 절차에서는 API Gateway 콘솔에서 이 작업을 수행하는 방법을 보여줍니다.

> **Note**
>
> `CreateAuthorizer` 작업을 통해 여러 사용자 풀을 사용하는 `COGNITO_USER_POOLS` 권한 부여자를 생성할 수 있습니다. `COGNITO_USER_POOLS` 권한 부여자 한 명에 최대 1,000개의 사용자 풀을 사용할 수 있습니다. 이 한도는 늘릴 수 없습니다.

> **Important**
>
> 아래 절차를 수행한 후 변경 사항을 전파하기 위해 API를 배포 또는 재배포할 필요가 있습니다. API 배포에 대한 자세한 내용은 Amazon API Gateway에서 REST API 배포 단원을 참조하세요.

API Gateway 콘솔을 사용하여 **`COGNITO_USER_POOLS`** 권한 부여자를 생성하려면

1. API Gateway에서 새 API를 생성하거나 기존 API를 선택합니다.
2. 기본 탐색 창에서 권한 부여자를 선택합니다.
3. 권한 부여자 생성을 선택합니다.
4. 사용자 풀을 사용하도록 새 권한 부여자를 구성하려면 다음을 수행합니다.
   a. 권한 부여자 이름에 이름을 입력합니다.
   b. 권한 부여자 유형으로 Cognito를 선택합니다.
   c. Cognito 사용자 풀의 경우 Amazon Cognito를 생성한 AWS 리전을 선택하고 사용 가능한 사용자 풀을 선택합니다.
   d. 토큰 소스에 헤더 이름으로 **`Authorization`**을 입력하여 사용자가 성공적으로 로그인할 때 Amazon Cognito에서 반환하는 자격 증명 또는 액세스 토큰을 전달합니다.
   e. (선택 사항) 토큰 검증 필드에 정규식을 입력하여 Amazon Cognito를 통해 요청에 대한 권한이 부여되기 전에 자격 증명 토큰의 aud(대상) 필드를 검증합니다. 액세스 토큰을 사용할 경우, 액세스 토큰에 aud 필드가 포함되지 않으므로 이 유효성 검사는 요청을 거부합니다.
   f. 권한 부여자 생성을 선택합니다.
5. `COGNITO_USER_POOLS` 권한 부여자를 생성한 후 필요할 경우, 사용자 풀에서 프로비저닝한 자격 증명 토큰을 제공하여 호출을 테스트할 수 있습니다. Amazon Cognito 자격 증명 SDK를 호출하여 사용자 로그인을 수행함으로써 이 자격 증명 토큰을 얻을 수 있습니다. `InitiateAuth` 작업을 사용해도 됩니다. 권한 부여 범위를 구성하지 않으면 API Gateway는 제공된 토큰을 자격 증명 토큰으로 취급합니다.

이전 절차에서는 새로 생성된 Amazon Cognito 사용자 풀을 사용하는 COGNITO_USER_POOLS 권한 부여자를 생성합니다. API 메서드에서 권한 부여자를 활성화하는 방법에 따라 통합된 사용자 풀에서 프로비저닝되는 자격 증명 토큰 또는 액세스 토큰을 사용할 수 있습니다.

메서드에서 **COGNITO_USER_POOLS** 권한 부여자를 구성하려면

1. 리소스를 선택합니다. 새 메서드를 선택하거나 기존 메서드를 선택합니다. 필요하다면 리소스를 생성합니다.
2. 메서드 요청 탭의 메서드 요청 설정에서 편집을 선택합니다.
3. 권한 부여자의 경우 드롭다운 메뉴에서 방금 생성한 Amazon Cognito 사용자 풀 권한 부여자를 선택합니다.
4. 자격 증명 토큰을 사용하려면 다음 작업을 수행하세요.

   a. 권한 부여 범위를 비워둡니다.

   b. 필요에 따라 통합 요청에서 본문 매핑 템플릿에 `$context.authorizer.claims['`*property-name*`']` 또는 `$context.authorizer.claims.`*property-name* 표현식을 추가하여, 사용자 풀에서 백엔드로 지정된 자격 증명 클레임 속성을 전달합니다. sub 또는 custom-sub처럼 단순한 속성 이름의 경우, 두 표기법이 동일합니다. custom:role처럼 복잡한 속성 이름의 경우, 점 표기법을 사용할 수 없습니다. 예를 들어 다음 매핑 표현식은 클레임의 표준 필드인 sub 및 email를 백엔드로 전달합니다.

```
{
 "context" : {
  "sub" : "$context.authorizer.claims.sub",
  "email" : "$context.authorizer.claims.email"
 }
}
```

   사용자 풀을 구성할 때 사용자 지정 클레임 필드를 선언했다면 동일한 패턴에 따라 사용자 지정 필드에 액세스할 수 있습니다. 다음 예에서는 클레임의 사용자 지정 role 필드를 가져옵니다.

```
{
 "context" : {
  "role" : "$context.authorizer.claims.role"
    }
}
```

사용자 지정 클레임 필드가 `custom:role`로 선언되면 다음 예를 사용하여 클레임의 속성을 가져옵니다.

```
{
 "context" : {
  "role" : "$context.authorizer.claims['custom:role']"
    }
}
```

5. 액세스 토큰을 사용하려면 다음 작업을 수행하세요.

   a. 권한 부여 범위에 Amazon Cognito 사용자 풀이 생성되었을 때 구성된 범위의 전체 이름을 하나 이상 입력합니다. 예를 들어 REST API에 대한 Amazon Cognito 사용자 풀 생성에서 제시된 예를 따르면, 스코프 중 하나는 `https://my-petstore-api.example.com/cats.read`입니다.

      런타임 시 이 단계의 메서드에서 지정되는 범위가 수신되는 토큰에서 클레임되는 범위와 일치하면 메서드 호출이 성공합니다. 그렇지 않으면 호출은 `401 Unauthorized` 응답과 함께 실패합니다.

   b. 저장을 선택합니다.

6. 선택한 다른 메서드에 대해 이러한 단계를 반복합니다.

`COGNITO_USER_POOLS` 권한 부여자의 경우, OAuth Scopes 옵션이 지정되지 않으면 API Gateway는 제공된 토큰을 자격 증명 토큰으로 취급하고 클레임된 자격 증명을 사용자 풀의 자격 증명과 대조하여 확인합니다. 그렇지 않으면 API Gateway는 제공된 토큰을 액세스 토큰으로 취급하고 토큰에서 클레임된 액세스 범위를 메서드에서 선언된 권한 부여 범위와 대조하여 확인합니다.

API Gateway 콘솔을 사용하는 대신 OpenAPI 정의 파일을 지정하고 API 정의를 API Gateway로 가져와 메서드에서 Amazon Cognito 사용자 풀을 활성화할 수도 있습니다.

OpenAPI 정의 파일을 사용하여 COGNITO_USER_POOLS 권한 부여자를 가져오려면

1. API에 OpenAPI 정의 파일을 생성(또는 내보내기)합니다.
2. `COGNITO_USER_POOLS` 권한 부여자(`MyUserPool`) JSON 정의를 다음과 같이 OpenAPI 3.0의 `securitySchemes` 섹션 또는 Open API 2.0의 `securityDefinitions` 섹션의 일부로 지정합니다.

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. 메서드 권한 부여에 자격 증명 토큰을 사용하려면 루트 리소스에서 다음 GET 메서드에 나온 것처럼 `{ "MyUserPool": [] }`을 메서드의 `security` 정의에 추가합니다.

```
"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. 메서드 권한 부여에 액세스 토큰을 사용하려면 위의 보안 정의를 `{ "MyUserPool": [resource-server/scope, ...] }`로 변경합니다.

```
  "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. 필요에 따라 적절한 OpenAPI 정의 또는 확장자를 사용해 다른 API 구성을 설정할 수 있습니다. 자세한 내용은 OpenAPI에 대한 API Gateway 확장 작업 단원을 참조하세요.

Amazon Cognito 사용자 풀과 통합된 REST API 호출

사용자 풀 권한 부여자를 구성한 상태로 메서드를 호출하려면 클라이언트에서 다음을 수행해야 합니다.

- 사용자가 사용자 풀을 사용하여 로그인하도록 설정합니다.
- 사용자가 사용자 풀에 로그인하도록 설정합니다.
- 사용자 풀에서 로그인한 사용자의 자격 증명 또는 액세스 토큰을 가져옵니다.
- `Authorization` 헤더(또는 권한 부여자를 생성할 때 지정한 다른 헤더)에 토큰을 포함합니다.

AWS Amplify를 사용하여 이러한 작업을 수행할 수 있습니다. 자세한 내용은 Amazon Cognito와 웹 및 모바일 앱 통합을 참조하세요.

- Android의 경우 Android용 Amplify 시작하기를 참조하세요.
- iOS를 사용하려면 iOS용 Amplify 시작하기를 참조하세요.
- JavaScript를 사용하려면 Javascript용 Amplify 시작하기를 참조하세요.

API Gateway 콘솔을 사용하여 REST API를 위한 교차 계정 Amazon Cognito 권한 부여자 구성

이제 다른 AWS 계정의 Amazon Cognito 사용자 풀도 API 권한 부여자로 사용할 수 있습니다. 각 계정은 Amazon API Gateway를 사용할 수 있는 모든 리전에 존재할 수 있습니다. Amazon Cognito 사용자 풀은 OAuth 또는 SAML 같은 보유자 토큰 인증 전략을 사용할 수 있습니다. 따라서 여러 API Gateway API에 대해 Amazon Cognito 사용자 풀 권한 부여자를 중앙에서 손쉽게 관리 및 공유할 수 있습니다.

이 단원에서는 Amazon API Gateway 콘솔을 사용하여 교차 계정 Amazon Cognito 사용자 풀을 구성하는 방법을 보여줍니다.

이 지침에서는 한 AWS 계정에서 API Gateway API를, 다른 계정에서는 Amazon Cognito 사용자 풀을 이미 생성했다고 가정합니다.

API Gateway 콘솔을 사용하여 교차 계정 Amazon Cognito 권한 부여자 구성

API를 포함하고 있는 계정에서 Amazon API Gateway 콘솔에 로그인하고 다음을 수행합니다.

1. API Gateway에서 새 API를 생성하거나 기존 API를 선택합니다.
2. 기본 탐색 창에서 권한 부여자를 선택합니다.
3. 권한 부여자 생성을 선택합니다.

4. 사용자 풀을 사용하도록 새 권한 부여자를 구성하려면 다음을 수행합니다.

   a. 권한 부여자 이름에 이름을 입력합니다.

   b. 권한 부여자 유형으로 Cognito를 선택합니다.

   c. Cognito 사용자 풀에서는 두 번째 계정에 있는 사용자 풀의 전체 ARN을 입력합니다.

   > **Note**
   >
   > Amazon Cognito 콘솔에서 일반 설정 창의 풀 ARN 필드에서 사용자 풀의 ARN을 찾을 수 있습니다.

   d. 토큰 소스에 헤더 이름으로 **`Authorization`**을 입력하여 사용자가 성공적으로 로그인할 때 Amazon Cognito에서 반환하는 자격 증명 또는 액세스 토큰을 전달합니다.

   e. (선택 사항) 토큰 검증 필드에 정규식을 입력하여 Amazon Cognito를 통해 요청에 대한 권한이 부여되기 전에 자격 증명 토큰의 aud(대상) 필드를 검증합니다. 액세스 토큰을 사용할 경우, 액세스 토큰에 aud 필드가 포함되지 않으므로 이 유효성 검사는 요청을 거부합니다.

   f. 권한 부여자 생성을 선택합니다.

AWS CloudFormation을 사용하여 REST API를 위한 Amazon Cognito 권한 부여자 생성

AWS CloudFormation을 사용하여 Amazon Cognito 사용자 풀 및 Amazon Cognito 권한 부여자를 생성합니다. 예제 AWS CloudFormation 템플릿은 다음을 수행합니다.

- Amazon Cognito 사용자 풀을 생성합니다. 클라이언트는 먼저 사용자를 사용자 풀에 로그인시키고 자격 증명 또는 액세스 토큰을 얻어야 합니다. 액세스 토큰을 사용하여 API 메서드 호출 권한을 부여하는 경우, 사용자 풀과의 앱 통합을 구성하여 지정된 리소스 서버에서 원하는 사용자 지정 범위를 설정해야 합니다.
- `GET` 메서드를 사용하여 API Gateway API를 생성합니다.
- `Authorization` 헤더를 토큰 소스로 사용하는 Amazon Cognito 권한 부여자를 생성합니다.

```
AWSTemplateFormatVersion: 2010-09-09
Resources:
  UserPool:
    Type: AWS::Cognito::UserPool
    Properties:
      AccountRecoverySetting:
        RecoveryMechanisms:
```

```
          - Name: verified_phone_number
            Priority: 1
          - Name: verified_email
            Priority: 2
      AdminCreateUserConfig:
        AllowAdminCreateUserOnly: true
      EmailVerificationMessage: The verification code to your new account is {####}
      EmailVerificationSubject: Verify your new account
      SmsVerificationMessage: The verification code to your new account is {####}
      VerificationMessageTemplate:
        DefaultEmailOption: CONFIRM_WITH_CODE
        EmailMessage: The verification code to your new account is {####}
        EmailSubject: Verify your new account
        SmsMessage: The verification code to your new account is {####}
    UpdateReplacePolicy: Retain
    DeletionPolicy: Retain
  CogAuthorizer:
    Type: AWS::ApiGateway::Authorizer
    Properties:
      Name: CognitoAuthorizer
      RestApiId:
        Ref: Api
      Type: COGNITO_USER_POOLS
      IdentitySource: method.request.header.Authorization
      ProviderARNs:
        - Fn::GetAtt:
            - UserPool
            - Arn
  Api:
    Type: AWS::ApiGateway::RestApi
    Properties:
      Name: MyCogAuthApi
  ApiDeployment:
    Type: AWS::ApiGateway::Deployment
    Properties:
      RestApiId:
        Ref: Api
    DependsOn:
      - CogAuthorizer
      - ApiGET
  ApiDeploymentStageprod:
    Type: AWS::ApiGateway::Stage
    Properties:
      RestApiId:
```

```
        Ref: Api
      DeploymentId:
        Ref: ApiDeployment
      StageName: prod
  ApiGET:
    Type: AWS::ApiGateway::Method
    Properties:
      HttpMethod: GET
      ResourceId:
        Fn::GetAtt:
          - Api
          - RootResourceId
      RestApiId:
        Ref: Api
      AuthorizationType: COGNITO_USER_POOLS
      AuthorizerId:
        Ref: CogAuthorizer
      Integration:
        IntegrationHttpMethod: GET
        Type: HTTP_PROXY
        Uri: http://petstore-demo-endpoint.execute-api.com/petstore/pets
Outputs:
  ApiEndpoint:
    Value:
      Fn::Join:
        - ""
        - - https://
          - Ref: Api
          - .execute-api.
          - Ref: AWS::Region
          - "."
          - Ref: AWS::URLSuffix
          - /
          - Ref: ApiDeploymentStageprod
          - /
```

REST API 통합 설정

API 메서드를 설정한 후 이를 백엔드의 엔드포인트와 통합해야 합니다. 백엔드 엔드포인트는 통합 엔드포인트로도 참조되며, Lambda 함수, HTTP 웹 페이지 또는 AWS 서비스 작업일 수 있습니다.

API 메서드와 마찬가지로 API 통합에는 통합 요청 및 통합 응답이 있습니다. 통합 요청은 백엔드가 수신하는 HTTP 요청을 캡슐화합니다. 클라이언트가 제출하는 메서드 요청과는 다를 수도 있고, 다르지 않을 수도 있습니다. 통합 응답은 백엔드에서 반환되는 출력을 캡슐화하는 HTTP 응답입니다.

통합 요청 설정에 수반되는 작업: 클라이언트가 제출한 메서드 요청을 백엔드로 전달하는 방식을 구성, 필요한 경우 요청 데이터를 통합 요청 데이터로 변환하는 방식을 구성, 호출할 Lambda 함수 지정, 수신 요청을 전송할 HTTP 서버 지정, 또는 호출할 AWS 서비스 작업 지정.

통합 응답을 설정하려면(비 프록시 통합에만 적용) 백엔드에서 반환된 결과를 특정 상태 코드의 메서드 응답에 전달하는 방식을 구성하고, 지정된 통합 응답 파라미터를 사전 구성된 메서드 응답 파라미터로 변환하는 방식을 구성하고, 지정된 본문 매핑 템플릿에 따라 통합 응답 본문을 메서드 응답 본문으로 매핑하는 방식을 구성해야 합니다.

프로그램에 따라 통합 요청은 `Integration` 리소스에 의해 캡슐화되고 통합 응답은 API Gateway의 `IntegrationResponse` 리소스에 의해 캡슐화됩니다.

통합 요청을 설정하려면 `Integration` 리소스를 생성한 후 이를 사용해 통합 엔드포인트 URL을 구성합니다. 그 다음에 백엔드에 액세스할 수 있는 IAM 권한을 설정하고 매핑을 지정하여 수신 요청 데이터를 백엔드로 전달하기 전에 변환합니다. 비프록시 통합에 대해 통합 응답을 설정하려면 `IntegrationResponse` 리소스를 생성한 후 이를 사용하여 대상 메서드 응답을 설정합니다. 그 다음에 백엔드 출력을 메서드 응답으로 매핑하는 방식을 구성합니다.

주제

- API Gateway에서 통합 요청을 설정
- API Gateway에서 통합 응답 설정
- API Gateway에서 Lambda 통합 설정
- API Gateway에서 HTTP 통합 설정
- API Gateway 프라이빗 통합 설정
- API Gateway에서 모의 통합 설정

API Gateway에서 통합 요청을 설정

통합 요청을 설정하려면 다음과 같은 필수 및 선택 작업을 수행합니다.

1. 메서드 요청 데이터가 백엔드로 전달되는 방식을 결정하는 통합 유형을 선택합니다.
2. 모의가 아닌 통합의 경우, `MOCK` 통합을 제외한 HTTP 메서드와 대상 통합 엔드포인트의 URI를 지정합니다.

3. Lambda 함수 및 기타 AWS 서비스 작업을 통한 통합의 경우, API Gateway에 대한 필수 권한이 있는 IAM 역할을 설정하여 사용자를 대신해 백엔드를 호출합니다.
4. 비 프록시 통합의 경우에는 필수 파라미터 매핑을 설정하여 사전 정의 메서드 요청 파라미터를 적절한 통합 요청 파라미터로 매핑합니다.
5. 비 프록시 통합의 경우, 필요한 본문 매핑을 설정하여 지정된 매핑 템플릿에 따라 특정 콘텐츠 유형의 수신 메서드 요청 본문을 매핑합니다.
6. 비 프록시 통합의 경우, 조건을 지정하여 그 조건에 따라 수신 메서드 요청 데이터가 백엔드로 있는 그대로 패스스루되도록 합니다.
7. 선택 사항으로 바이너리 페이로드에 대해 형식 변환을 처리하는 방식을 지정합니다.
8. 선택 사항으로 캐시 네임스페이스 이름과 캐시 키 파라미터를 선언하여 API 캐싱을 활성화합니다.

이 작업에는 API Gateway의 통합 리소스를 생성하고 적절한 속성 값을 설정하는 과정이 수반됩니다. API Gateway 콘솔, AWS CLI 명령, AWS SDK 또는 API Gateway REST API를 사용하면 됩니다.

주제

- API 통합 요청의 기본 작업
- API Gateway API 통합 유형 선택
- 프록시 리소스를 사용하여 프록시 통합 설정
- API Gateway 콘솔을 사용하여 API 통합 요청 설정

API 통합 요청의 기본 작업

통합 요청은 API Gateway가 백엔드에 제출하는 HTTP 요청으로서 클라이언트가 제출한 요청 데이터를 전달하고 필요한 경우 이 데이터를 변환합니다. 통합 요청의 HTTP 메서드(또는 동사) 및 URI는 백엔드(즉 통합 엔드포인트)의 지시를 받습니다. 이들은 각각 메서드 요청의 HTTP 메서드 및 URI와 같거나 다를 수 있습니다.

예를 들어 Lambda 함수가 Amazon S3에서 가져온 파일을 반환하면 이 작업을 직관적으로 GET 메서드 요청으로 클라이언트에게 공개할 수 있습니다. 이는 그에 상응하는 통합 요청이 POST 요청을 사용해 Lambda 함수를 호출할 것을 요구하더라도 가능합니다. HTTP 엔드포인트의 경우, 메서드 요청과 그에 상응하는 통합 요청이 모두 동일한 HTTP 동사를 사용할 확률이 높습니다. 하지만 이것이 필수적인 것은 아닙니다. 다음 메서드 요청을 통합할 수 있습니다.

```
GET /{var}?query=value
Host: api.domain.net
```

다음 통합 요청을 사용:

```
POST /
Host: service.domain.com
Content-Type: application/json
Content-Length: ...

{
   path: "{var}'s value",
   type: "value"
}
```

API 개발자라면 메서드 요청을 위한 HTTP 동사 및 URI는 해당 요구 사항에 적합한 것은 무엇이든지 사용할 수 있습니다. 그러나 통합 엔드포인트의 요구 사항을 따라야 합니다. 메서드 요청 데이터가 통합 요청 데이터와 다르다면 메서드 요청 데이터에서 통합 요청 데이터로 매핑을 제공하여 그 차이를 해소할 수 있습니다.

이전 예제에서 매핑은 `{var}` 메서드 요청의 경로 변수(`query`)와 쿼리 파라미터(`GET`) 값을 `path` 및 `type`의 통합 요청 페이로드 속성 값으로 변환합니다. 매핑 가능한 다른 요청 데이터는 요청 헤더와 본문을 포함합니다. 이에 대해서는 API Gateway 콘솔을 사용하여 요청 및 응답 데이터 매핑 설정에 설명되어 있습니다.

HTTP 또는 HTTP 프록시 통합 요청을 설정할 때 백엔드 HTTP 엔드포인트 URL을 통합 요청 URI 값으로 할당합니다. 예를 들어 PetStore API에서 반려 동물 페이지를 얻어오라는 메서드 요청에는 다음과 같은 통합 요청 URI가 있습니다.

```
http://petstore-demo-endpoint.execute-api.com/petstore/pets
```

Lambda 또는 Lambda 프록시 통합을 설정할 때 Lambda 함수 호출을 위한 Amazon 리소스 이름(ARN)을 통합 요청 URI 값으로 할당합니다. 이 ARN의 형식은 다음과 같습니다.

```
arn:aws:apigateway:api-region:lambda:path//2015-03-31/functions/arn:aws:lambda:lambda-region:account-id:function:lambda-function-name/invocations
```

`arn:aws:apigateway:`*`api-region`*`:lambda:path/` 이후 부분, 즉 `/2015-03-31/functions/arn:aws:lambda:`*`lambda-region`*`:`*`account-id`*`:function:`*`lambda-function-name`*`/invocations`는 Lambda 호출 작업의 REST API URI 경로입니다. API Gateway 콘솔을 사용하여 Lambda 통합을 설정하면 API Gateway는 사용자에게 리전에서 *`lambda-function-name`*을 선택하라는 메시지를 표시한 후 ARN을 생성하여 이를 통합 URI에 할당합니다.

다른 AWS 서비스 작업으로 통합 요청을 설정할 때 통합 요청 URI는 Lambda Invoke 작업이 있는 통합과 유사한 ARN이기도 합니다. 예를 들어 Amazon S3의 GetBucket 작업과 통합하는 경우, 통합 요청 URI는 다음 형식으로 된 ARN입니다.

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

통합 요청 URI는 그 작업을 지정할 경로 규칙에 관한 것입니다. 여기에서 *{bucket}*은 버킷 이름의 자리 표시자입니다. 또는 AWS 서비스 작업을 그 이름으로 참조할 수 있습니다. 작업 이름을 사용하면 Amazon S3의 GetBucket 작업에 대한 통합 요청 URI는 다음과 같아집니다.

```
arn:aws:apigateway:api-region:s3:action/GetBucket
```

작업 기반 통합 요청 URI를 사용하는 경우 *{bucket}* 작업의 입력 형식에 따라 버킷 이름({ Bucket: "*{bucket}*" })을 통합 요청 본문(GetBucket)에 지정해야 합니다.

AWS 통합의 경우에는 API Gateway가 통합 작업을 호출할 수 있게 허용하도록 자격 증명을 구성해야 합니다. IAM 역할을 새로 만들거나 기존 것에서 선택하여 API Gateway가 작업을 호출한 후 그 ARN을 사용해 그 역할을 지정하도록 할 수 있습니다. 다음은 이 ARN의 예를 보여줍니다.

```
arn:aws:iam::account-id:role/iam-role-name
```

이 IAM 역할에는 그 작업이 실행되도록 허용하는 정책이 포함되어야 합니다. 또한 그 역할을 맡을 수 있도록 (그 역할의 신뢰 관계에) 신뢰할 수 있는 엔터티로 선언된 API Gateway가 있어야 합니다. 그러한 권한은 그 작업 자체에 대해 부여할 수 있습니다. 이 권한을 리소스 기반 권한이라고 합니다. Lambda 통합의 경우, Lambda의 addPermission 작업을 호출하여 리소스 기반 권한을 설정한 후 API Gateway 통합 요청에서 credentials를 null로 설정합니다.

지금까지 기본적인 통합 설정을 다루었습니다. 고급 설정에는 메서드 요청 데이터를 통합 요청 데이터로 매핑하는 작업이 수반됩니다. 통합 응답에 대한 기본 설정에 대해 설명한 후에는 API Gateway 콘솔을 사용하여 요청 및 응답 데이터 매핑 설정에서 고급 주제를 다룹니다. 또한 페이로드를 패스스루하고 콘텐츠 인코딩을 처리하는 방법도 다룹니다.

API Gateway API 통합 유형 선택

작업 중인 통합 엔드포인트의 유형과 데이터가 통합 엔드포인트로(부터) 전달되는 방식에 따라 API 통합 유형을 선택합니다. Lambda 함수의 경우 Lambda 프록시 통합 또는 Lambda 사용자 지정 통합이 있을 수 있습니다. HTTP 엔드포인트의 경우에는 HTTP 프록시 통합 또는 HTTP 사용자 지정 통합이 있을 수 있습니다. AWS 서비스 작업의 경우에는 비프록시 유형의 AWS 통합만 있습니다. API Gateway도

모의 통합을 지원하는데, 이 경우 API Gateway는 메서드 요청에 응답하기 위한 통합 엔드포인트의 역할을 합니다.

Lambda 사용자 지정 통합은 AWS 통합의 특수 사례로서, 이 경우 통합 엔드포인트는 Lambda 서비스의 function-invoking 작업에 해당합니다.

프로그래밍 방식으로 `Integration` 리소스의 `type` 속성을 설정하여 통합 유형을 선택합니다. Lambda 프록시 통합의 경우, 그 값은 `AWS_PROXY`입니다. Lambda 사용자 지정 통합 및 기타 모든 AWS 통합의 경우에는 `AWS`입니다. HTTP 프록시 통합 및 HTTP 통합의 경우에는 그 값이 각각 `HTTP_PROXY` 및 `HTTP`입니다. 모의 통합의 경우 `type` 값은 `MOCK`입니다.

Lambda 프록시 통합은 단일 Lambda 함수를 사용하여 간소화된 통합 설정을 지원합니다. 이 설정은 간단하며 기존 설정을 손상하지 않고 백엔드와 함께 진화할 수 있습니다. 이러한 이유로 Lambda 함수와의 통합을 적극 권장합니다.

이와 대조적으로 Lambda 사용자 지정 통합을 사용하면 입력 및 출력 데이터 형식의 유사한 요구 사항이 있는 다양한 통합 엔드포인트에 대해 구성된 매핑 템플릿을 재사용할 수 있습니다. 이 설정은 더 복잡하며 더 고급 애플리케이션 시나리오에 권장됩니다.

이와 마찬가지로 HTTP 프록시 통합에는 간소화된 통합 설정이 있으며 기존 설정을 손상하지 않고 백엔드와 함께 진화할 수 있습니다. HTTP 사용자 지정 통합은 설정하기가 더 복잡하지만 다른 통합 엔드포인트에 대해 구성된 매핑 템플릿을 재사용할 수 있습니다.

다음 목록은 지원되는 통합 유형을 요약한 것입니다.

- `AWS`: 이 통합 유형을 사용하면 API에서 AWS 서비스 작업을 공개할 수 있습니다. `AWS` 통합의 경우, 통합 요청과 통합 응답을 모두 구성하고 메서드 요청에서 통합 요청으로, 통합 응답에서 메서드 응답으로 필수적인 데이터 매핑을 설정해야 합니다.
- `AWS_PROXY`: 이러한 유형의 통합을 사용해 유연하고 다목적이며 간소화된 통합 설정으로 API 메서드가 Lambda 함수 호출 작업에 통합되도록 할 수 있습니다. 이러한 통합은 통합된 Lambda 기능과 클라이언트 간의 직접적인 상호 작용에 의존합니다.

  Lambda 프록시 통합으로도 알려진 이러한 유형의 통합에서는 통합 요청 또는 통합 응답을 설정하지 않습니다. API Gateway는 클라이언트로부터 입력으로 수신되는 요청을 백엔드 Lambda 함수로 전달합니다. 통합된 Lambda 함수는 이 형식의 입력을 받아들여 그 입력을 요청 헤더, URL 경로 변수, 쿼리 문자열 파라미터 및 적용 가능한 본문 등 사용 가능한 모든 소스로부터 구문 분석합니다. 이 함수는 이 출력 형식에 따라 결과를 반환합니다.

  이것은 API Gateway를 통해 Lambda 함수를 호출할 때 선호되는 통합 유형이며 함수 호출 작업 외의 Lambda 작업을 포함한 기타 모든 AWS 서비스 작업에는 적용할 수 없습니다.

- HTTP: 이 통합 유형을 사용하면 API에서 백엔드의 HTTP 엔드포인트를 공개할 수 있습니다. HTTP 사용자 지정 통합으로도 알려진 HTTP 통합에서는 통합 요청과 통합 응답을 둘 다 설정해야 합니다. 메서드 요청에서 통합 요청으로, 그리고 통합 응답에서 메서드 응답으로 필수적인 데이터 매핑을 설정해야 합니다.
- HTTP_PROXY: HTTP 프록시 통합을 사용하여 클라이언트는 단일 API 메서드에서 간소화된 통합 설정을 통해 백엔드 HTTP 엔드포인트에 액세스할 수 있습니다. 통합 요청 또는 통합 응답은 설정하지 않습니다. API Gateway는 클라이언트로부터 수신되는 요청을 HTTP 엔드포인트로 전달하고 HTTP 엔드포인트에서 발신되는 응답을 클라이언트로 전달합니다.
- MOCK: 이 통합 유형을 사용하면 API Gateway에서 요청을 백엔드로 전송하지 않고 응답을 반환할 수 있습니다. 이것은 백엔드 사용에 대한 요금을 발생시키지 않으면서 통합 설정을 테스트하고 협력을 통한 API의 발전을 활성화하는 데 사용할 수 있으므로 API 테스트에 유용합니다.

  협력을 통한 발전을 통해 팀은 MOCK 통합을 사용해 다른 팀이 소유한 API 구성 요소의 시뮬레이션을 설정함으로써 자신의 개발 노력을 격리시킬 수 있습니다. 또한 API 메서드가 CORS 액세스를 허용하는지 확인하기 위해 CORS 관련 헤더를 반환하는 데도 유용합니다. 사실 API Gateway 콘솔은 OPTIONS 메서드를 통합하여 모의 통합으로 CORS를 지원합니다. 게이트웨이 응답은 모의 통합의 다른 예입니다.

프록시 리소스를 사용하여 프록시 통합 설정

프록시 리소스를 사용하여 API Gateway API에서 프록시 통합을 설정하려면 다음 작업을 수행합니다.

- {*proxy*+}의 복잡한 경로 변수를 사용하여 프록시 리소스를 생성합니다.
- 프록시 리소스에서 ANY 메서드를 설정합니다.
- HTTP 또는 Lambda 통합 유형을 사용하여 리소스 및 메서드를 백엔드와 통합합니다.

> **Note**
>
> 복잡한 경로 변수, ANY 메서드 및 프록시 통합 유형은 일반적으로 함께 사용되지만 독립적인 기능입니다. 복잡한 리소스에 대해 특정 HTTP 메서드를 구성하거나 비 프록시 통합 유형을 프록시 리소스에 적용할 수 있습니다.

API Gateway는 Lambda 프록시 통합 또는 HTTP 프록시 통합으로 메서드를 처리할 때 특정 제한 및 제약을 적용합니다. 자세한 내용은 the section called “중요 정보” 단원을 참조하세요.

> **Note**
>
> 패스스루를 포함하는 프록시 통합을 사용할 경우 API Gateway에서는 페이로드의 콘텐츠 유형이 지정되지 않은 경우 기본 `Content-Type:application/json` 헤더를 반환합니다.

프록시 리소스는 HTTP 프록시 통합이나 Lambda 프록시 통합을 사용하여 백엔드와 통합될 때 가장 강력합니다.

프록시 리소스를 사용한 HTTP 프록시 통합

API Gateway REST API에서 `HTTP_PROXY`로 지정되는 HTTP 프록시 통합은 메서드 요청을 백엔드 HTTP 엔드포인트와 통합하기 위한 것입니다. API Gateway에서는 이 통합 유형을 사용하여 프런트엔드와 백엔드 사이에 전체 요청 및 응답을 전달할 뿐이며 특정 제약 및 제한의 적용을 받습니다.

> **Note**
>
> HTTP 프록시 통합은 다중 값 헤더와 쿼리 문자열을 지원합니다.

HTTP 프록시 통합을 프록시 리소스에 적용할 경우 단일 통합 설정으로 HTTP 백엔드의 일부 또는 전체 엔드포인트 계층을 공개하도록 API를 설정할 수 있습니다. 예를 들어 웹사이트의 백엔드가 루트 노드(`/site`)에서 $/site/a_0/a_1/.../a_N$, $/site/b_0/b_1/.../b_M$과 같은 여러 트리 노드 분기로 구성된다고 가정합니다. `ANY`의 프록시 리소스에 대한 `/api/{proxy+}` 메서드를 URL 경로가 `/site/{proxy}`인 백엔드 엔드포인트와 통합할 경우, 단일 통합 요청으로 $[a_0, a_1, ..., a_N, b_0, b_1, ...b_M, ...]$에 대해 모든 HTTP 작업(GET, POST 등)을 지원할 수 있습니다. 그 대신에 프록시 통합을 특정 HTTP 메서드(예: `GET`)에 적용할 경우 그 결과로 얻는 통합 요청은 그러한 백엔드 노드 중 하나에서 지정된(예: `GET`) 작업으로 작동합니다.

프록시 리소스를 사용한 Lambda 프록시 통합

API Gateway REST API에서 `AWS_PROXY`로 지정되는 Lambda 프록시 통합은 백엔드에서 메서드 요청을 Lambda 함수와 통합하기 위한 것입니다. 이 통합 유형을 통해 API Gateway에서는 기본 매핑 템플릿을 적용하여 전체 요청을 Lambda 함수로 보내고 Lambda 함수의 출력을 HTTP 응답으로 변환합니다.

마찬가지로 Lambda 프록시 통합을 `/api/{proxy+}`의 프록시 리소스에 적용하여 백엔드 Lambda 함수가 `/api` 아래의 API 리소스 변경에 개별적으로 대응하도록 단일 통합을 설정할 수 있습니다.

API Gateway 콘솔을 사용하여 API 통합 요청 설정

API 메서드 설정은 메서드를 정의하고 그 동작을 설명합니다. 메서드를 설정하려면 해당 메서드를 노출시킬 루트('/') 등의 리소스, HTTP 메서드(GET, POST 등) 및 대상 백엔드와의 통합 방식을 지정해야 합니다. 메서드 요청 및 응답은 호출하는 앱과의 관계를 규정하며, 이를 통해 API는 어떤 파라미터를 받게 되고 응답은 어떤 유형인지 지정합니다.

다음 절차에서는 API Gateway 콘솔을 사용하여 메서드 설정을 지정하는 방법을 설명합니다.

1. 리소스 창에서 메서드 생성을 선택합니다.
2. 메서드 유형에서 HTTP 메서드를 선택합니다.
3. 통합 유형에서 다음 중 하나를 선택합니다.

   - API를 Lambda 함수에 통합할 예정이라면 Lambda 함수를 선택합니다. API 수준에서 비프록시 통합을 생성하는 경우 AWS 통합 유형이고, 프록시 통합을 생성하는 경우 AWS_PROXY 통합 유형입니다.
   - API가 HTTP 엔드포인트와 통합될 예정이라면 HTTP를 선택합니다. API 수준에서 이는 HTTP 통합 유형입니다.
   - API가 AWS 서비스와 직접 통합될 예정이라면 AWS 서비스를 선택합니다. API 수준에서 이는 AWS 통합 유형입니다. Lambda 함수 옵션은 Lambda 함수 호출을 위한 AWS 통합의 특수한 경우입니다.

     API Gateway API를 설정하여 다음 중 하나를 수행하려면:

     - 새 Lambda 함수를 생성합니다.
     - Lambda 함수에 리소스 권한을 설정합니다.
     - 기타 Lambda 서비스 작업을 수행합니다.

     AWS 서비스를 선택해야 합니다.
   - API Gateway가 정적 응답을 반환하는 백엔드로 작동하기를 원할 경우 Mock을 선택합니다. API 수준에서 이는 MOCK 통합 유형입니다. 일반적으로 API가 최종적인 것은 아니지만 API 응답을 생성하여 테스트를 위해 종속 팀을 차단하고자 하는 경우 MOCK 통합을 사용할 수 있습니다. OPTION 메서드의 경우 API Gateway가 MOCK 통합을 기본값으로 설정하여 적용된 API 리소스에 대한 CORS 사용 헤더를 반환합니다. 이 옵션을 선택하는 경우 이 주제와 관련된 나머지 부분을 건너뛰고 API Gateway에서 모의 통합 설정을 참조하세요.
4. Lambda 함수를 선택한 경우 다음을 수행합니다.

a. Lambda 프록시 통합을 사용하려면 Lambda 프록시 통합을 활성화합니다. Lambda 프록시 통합에 대한 자세한 내용은 the section called “ Lambda 프록시 통합 이해 ” 섹션을 참조하세요.

b. Lambda 함수에서 Lambda 함수의 이름을 입력합니다. API와 다른 리전에서 Lambda 함수를 사용하는 경우, 드롭다운 메뉴에서 리전을 선택하고 Lambda 함수의 이름을 입력합니다. 교차 계정 Lambda 함수를 사용하는 경우 함수 ARN을 입력합니다.

   리전 이름 및 식별자 목록은 Amazon Web Services 일반 참조의 AWS Lambda를 참조하세요.

c. 기본 제한 시간 값인 29초를 사용하려면 기본 제한 시간을 활성화된 상태로 유지합니다. 사용자 지정 제한 시간을 설정하려면 기본 제한 시간을 선택하고 `50` ~ `29000`밀리초 사이의 제한 시간 값을 입력합니다.

d. 메서드 생성을 선택합니다.

5. HTTP 를 선택한 경우 다음을 수행합니다.

a. HTTP 프록시 통합을 사용하려면 HTTP 프록시 통합을 활성화합니다. HTTP 프록시 통합에 대한 자세한 내용은 the section called “API Gateway에서 HTTP 프록시 통합 설정” 섹션을 참조하세요.

b. HTTP 메서드(HTTP method)에서 HTTP 백엔드의 메서드와 가장 가까운 HTTP 메서드 유형을 선택합니다.

c. 엔드포인트 URL에서 이 메서드를 사용하고자 하는 HTTP 백엔드의 URL을 입력합니다.

d. 콘텐츠 처리에서 콘텐츠 처리 동작을 선택합니다.

e. 기본 제한 시간 값인 29초를 사용하려면 기본 제한 시간을 활성화된 상태로 유지합니다. 사용자 지정 제한 시간을 설정하려면 기본 제한 시간을 선택하고 `50` ~ `29000`밀리초 사이의 제한 시간 값을 입력합니다.

f. 메서드 생성을 선택합니다.

6. 모의(Mock)를 선택한 경우 다음을 수행합니다.

- 메서드 생성을 선택합니다.

7. AWS 서비스를 선택한 경우 다음을 수행합니다.

a. AWS 리전에서 이 메서드를 사용하여 작업을 호출하고자 하는 AWS 리전을 선택합니다.

b. AWS 서비스에서 이 메서드로 호출하려는 AWS 서비스를 선택합니다.

c. AWS 하위 도메인에 AWS 서비스가 사용하는 하위 도메인을 입력합니다. 일반적으로 이 필드는 비워 둡니다. 일부 AWS 서비스는 호스트의 일부로 하위 도메인을 지원할 수 있습니다. 사용 가능 여부, 그리고 사용 가능할 경우 세부 정보는 서비스 설명서를 참조하세요.

d. HTTP 메서드(HTTP method)에서 작업에 해당하는 HTTP 메서드 유형을 선택합니다. HTTP 메서드 유형에 대해서는 AWS 서비스에서 선택한 AWS 서비스에 관한 API 참조 문서를 참조하세요.

e. 작업 유형에서 API 작업을 사용하려면 작업 이름 사용을 선택하고 사용자 지정 리소스 경로를 사용하려면 경로 재정의 사용을 선택합니다. 사용 가능한 작업 및 사용자 지정 리소스 경로에 대해서는 AWS 서비스에서 선택한 AWS 서비스에 관한 API 참조 문서를 참조하세요.

f. 작업 이름 또는 경로 재정의를 입력합니다.

g. 실행 역할에 메서드가 작업을 호출할 때 사용할 IAM 역할의 ARN을 입력합니다.

IAM 역할을 생성하려는 경우 the section called “1단계: AWS 서비스 프록시 실행 역할 생성”의 지침을 조정할 수 있습니다. 원하는 작업 개수 및 리소스 설명으로 다음 형식의 액세스 정책을 지정합니다.

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

작업 및 리소스 설명문 구문은 AWS 서비스에서 선택한 AWS서비스에 관한 문서를 참조하세요.

IAM 역할의 신뢰 관계에 대해 다음과 같이 지정합니다. 이렇게 하면 API Gateway가 AWS 계정을 대신해 작업을 수행할 수 있습니다.

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

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

h. 기본 제한 시간 값인 29초를 사용하려면 기본 제한 시간을 활성화된 상태로 유지합니다. 사용자 지정 제한 시간을 설정하려면 기본 제한 시간을 선택하고 `50` ~ `29000`밀리초 사이의 제한 시간 값을 입력합니다.

i. 메서드 생성을 선택합니다.

API Gateway에서 통합 응답 설정

비 프록시 통합의 경우에는 통합 응답을 하나 이상 설정하고 이를 기본 응답으로 만들어 백엔드에서 클라이언트로 반환되는 결과를 전달해야 합니다. 그 결과를 있는 그대로 패스스루하거나 통합 응답 데이터를 메서드 응답 데이터로 변환(둘이 서로 다른 형식인 경우)할 수 있습니다.

프록시 통합의 경우, API Gateway는 백엔드 출력을 클라이언트에게 HTTP 응답으로 자동 전달합니다. 통합 응답이나 메서드 응답은 설정하지 않습니다.

통합 응답을 설정하려면 다음과 같은 필수 및 선택 작업을 수행합니다.

1. 통합 응답 데이터가 매핑된 메서드 응답의 HTTP 상태 코드를 지정합니다. 이 항목은 필수입니다.
2. 이 통합 응답이 표시할 백엔드 출력을 선택하기 위해 표현식을 정의합니다. 이를 비워둘 경우, 응답은 아직 구성되지 않은 응답을 파악하는 데 사용되는 기본 응답입니다.
3. 필요한 경우 키-값 페어로 구성된 매핑을 선언하여 지정된 통합 응답 파라미터를 특정 메서드 응답 파라미터로 매핑합니다.
4. 필요한 경우 본문 매핑 템플릿을 추가하여 특정 통합 응답 페이로드를 지정된 메서드 응답 페이로드로 변환합니다.
5. 필요한 경우 바이너리 페이로드에 대해 형식 변환을 처리하는 방식을 지정합니다.

통합 응답은 백엔드 응답을 캡슐화하는 HTTP 응답입니다. HTTP 엔드포인트의 경우, 백엔드 응답은 HTTP 응답입니다. 통합 응답 상태 코드는 백엔드에서 반환하는 상태 코드를 받아들일 수 있고, 통합 응답 본문은 백엔드에서 반환하는 페이로드입니다. Lambda 엔드포인트의 경우, 백엔드 응답은 Lambda 함수에서 반환되는 출력입니다. Lambda 통합의 경우, Lambda 함수 출력은 `200 OK` 응답으

로 반환됩니다. 페이로드에는 JSON 문자열이나 JSON 객체, 또는 JSON 객체인 오류 메시지 등 JSON 데이터가 결과로 포함될 수 있습니다. 정규식을 selectionPattern 속성에 할당하여 오류 응답을 적절한 HTTP 오류 응답으로 매핑할 수 있습니다. Lambda 함수 오류 응답에 대한 자세한 내용은 API Gateway에서 Lambda 오류 처리를 참조하세요. Lambda 프록시 통합의 경우, Lambda 함수는 다음 형식의 출력을 반환해야 합니다.

```
{
    statusCode: "...",            // a valid HTTP status code
    headers: {
        custom-header: "..."      // any API-specific custom header
    },
    body: "...",                  // a JSON string.
    isBase64Encoded:  true|false  // for binary support
}
```

Lambda 함수 응답을 적당한 HTTP 응답으로 매핑할 필요가 없습니다.

클라이언트에게 결과를 반환하려면 통합 응답을 설정하여 엔드포인트 응답을 있는 그대로 그에 상응하는 메서드 응답으로 패스스루합니다. 또는 엔드포인트 응답 데이터를 메서드 응답 데이터에 매핑할 수 있습니다. 매핑할 수 있는 응답 데이터에는 응답 상태 코드, 응답 헤더 파라미터 및 응답 본문이 포함됩니다. 반환된 상태 코드에 대해 메서드 응답이 정의되지 않은 경우, API Gateway는 500 오류를 반환합니다. 자세한 내용은 매핑 템플릿을 사용한 API의 요청 및 응답 파라미터와 상태 코드 재정의 섹션을 참조하세요.

API Gateway에서 Lambda 통합 설정

Lambda 프록시 통합 또는 Lambda 비프록시(사용자 지정) 통합을 사용하여 API 메서드를 Lambda 함수와 통합할 수 있습니다.

Lambda 프록시 통합에서는 필요한 설정이 간단합니다. 통합의 HTTP 메서드를 POST로, 통합 엔드포인트 URI를 특정 Lambda 함수의 Lambda 함수 호출 작업의 ARN으로 설정하고, 사용자를 대신해 Lambda 함수를 호출할 수 있는 API Gateway 권한을 부여합니다.

Lambda 비 프록시 통합에서는, 프록시 통합 설정 단계 외에도 수신 요청 데이터가 통합 요청에 매핑되는 방식과 그 결과로 얻는 통합 응답 데이터가 메서드 응답에 매핑되는 방식도 지정합니다.

주제

- API Gateway에서 Lambda 프록시 통합 설정
- API Gateway에서 Lambda 사용자 지정 통합 설정

- 백엔드 Lambda 함수의 비동기 호출 설정
- API Gateway에서 Lambda 오류 처리

API Gateway에서 Lambda 프록시 통합 설정

주제

- API Gateway Lambda 프록시 통합 이해
- 다중 값 헤더 및 쿼리 문자열 파라미터에 대한 지원
- Lambda 프록시 통합을 사용하여 프록시 리소스 설정
- 를 사용하여 Lambda 프록시 통합을 설정합니다. AWS CLI
- 프록시 통합에 대한 Lambda 함수의 입력 형식
- 프록시 통합에 대한 Lambda 함수의 출력 형식

API Gateway Lambda 프록시 통합 이해

Amazon API Gateway의 Lambda 프록시 통합은 단일 API 메서드의 설정을 통해 API를 구축하기 위한 간단하고 강력하며 민첩한 메커니즘입니다. Lambda 프록시 통합을 사용하면 클라이언트가 백엔드에서 단일 Lambda 함수를 호출할 수 있습니다. 함수는 다른 Lambda 함수 호출을 포함하여 다른 AWS 서비스의 많은 리소스 또는 기능에 액세스합니다.

Lambda 프록시 통합에서 클라이언트가 API 요청을 제출하면 API Gateway는 통합된 Lambda 함수로 이벤트 객체를 전달합니다. 단, 요청 파라미터의 순서는 유지되지 않습니다. 요청 데이터에는 요청 헤더, 쿼리 문자열 파라미터, URL 경로 변수, 페이로드 및 API 구성 데이터가 포함되어 있습니다. 구성 데이터에는 현재 배포 단계 이름, 단계 변수, 사용자 ID 또는 권한 부여 컨텍스트(있는 경우)가 포함될 수 있습니다. 백엔드 Lambda 함수는 수신되는 요청 데이터를 구문 분석하여 반환하는 응답을 결정합니다. API Gateway에서 Lambda 출력을 클라이언트에 대한 API 응답으로 전달하려면 Lambda 함수에서 이 형식으로 결과를 반환해야 합니다.

API Gateway는 Lambda 프록시 통합을 위해 클라이언트와 백엔드 Lambda 함수 사이에 개입하지 않기 때문에 클라이언트와 통합된 Lambda 함수는 API의 기존 통합 설정을 깨지 않고 서로의 변화에 맞게 조정할 수 있습니다. 이렇게 하려면 클라이언트가 백엔드 Lambda 함수에 의해 제정된 애플리케이션 프로토콜을 따라야 합니다.

모든 API 메서드에 대해 Lambda 프록시 통합을 설정할 수 있습니다. 그러나 Lambda 프록시 통합은 일반 프록시 리소스가 포함된 API 메서드에 대해 구성될 때 더 강력합니다. 일반 프록시 리소스는 `{proxy+}`의 특수 템플릿 경로 변수, catch-all ANY 메서드 자리 표시자 또는 둘 다로 표시될 수 있습니

다. 클라이언트는 수신되는 요청의 백엔드 Lambda 함수에 대한 입력을 요청 파라미터 또는 해당 페이로드로 전달할 수 있습니다. 요청 파라미터에는 헤더, URL 경로 변수, 쿼리 문자열 파라미터 및 해당 페이로드가 포함되어 있습니다. 통합된 Lambda 함수는 필요한 입력이 누락된 경우 요청을 처리하고 의미 있는 오류 메시지로 클라이언트에 응답하기 전에 모든 입력 소스를 확인합니다.

일반 HTTP 메서드 ANY 및 일반 리소스 `{proxy+}`와 통합된 API 메서드를 호출할 때 클라이언트는 ANY 대신 특정 HTTP 메서드를 사용하여 요청을 제출합니다. 또한 클라이언트는 `{proxy+}` 대신 특정 URL 경로를 지정하고 필수 헤더, 쿼리 문자열 파라미터 또는 해당 페이로드를 포함합니다.

다음 목록에는 Lambda 프록시 통합을 통해 다양한 API 메서드의 런타임 동작이 요약되어 있습니다.

- `ANY /{proxy+}`: 클라이언트는 특정 HTTP 메서드를 선택하고 특정 리소스 경로 계층을 설정해야 하며 헤더, 쿼리 문자열 파라미터 및 해당 페이로드를 설정하여 데이터를 통합된 Lambda 함수에 대한 입력으로 전달할 수 있습니다.
- `ANY /res`: 클라이언트는 특정 HTTP 메서드를 선택해야 하고 헤더, 쿼리 문자열 파라미터 및 해당 페이로드를 설정하여 데이터를 통합된 Lambda 함수에 대한 입력으로 전달할 수 있습니다.
- `GET|POST|PUT|... /{proxy+}`: 클라이언트는 특정 리소스 경로 계층, 헤더, 쿼리 문자열 파라미터 및 해당 페이로드를 설정하여 데이터를 통합된 Lambda 함수에 대한 입력으로 전달할 수 있습니다.
- `GET|POST|PUT|... /res/{path}/...`: 클라이언트는 특정 경로 세그먼트(`{path}` 변수의 경우)를 선택해야 하고 요청 헤더, 쿼리 문자열 파라미터 및 해당 페이로드를 설정하여 입력 데이터를 통합된 Lambda 함수에 전달할 수 있습니다.
- `GET|POST|PUT|... /res`: 클라이언트는 요청 헤더, 쿼리 문자열 파라미터 및 해당 페이로드를 선택하여 입력 데이터를 통합된 Lambda 함수에 전달할 수 있습니다.

프록시 리소스 `{proxy+}` 및 사용자 지정 리소스 `{custom}` 모두 템플릿 경로 변수로 표현됩니다. 그러나 `{proxy+}`는 경로 계층을 따라 모든 리소스를 참조할 수 있지만 `{custom}`은 특정 경로 세그먼트만 참조합니다. 예를 들어 식품 매장은 부서 이름, 생산 카테고리 및 제품 유형별로 온라인 제품 재고를 구성할 수 있습니다. 식품 매장의 웹 사이트는 사용자 지정 리소스의 템플릿 경로 변수인 `/{department}/{produce-category}/{product-type}`을 사용하여 사용 가능한 제품을 나타낼 수 있습니다. 예를 들어 사과는 `/produce/fruit/apple`로, 당근은 `/produce/vegetables/carrot`으로 표현됩니다. 또한 `/{proxy+}`를 사용하여 온라인 매장에서 쇼핑하는 동안 고객이 검색할 수 있는 부서, 생산 카테고리 또는 제품 유형을 나타낼 수 있습니다. 예를 들어 `/{proxy+}`는 다음 항목 중 하나를 참조할 수 있습니다.

- `/produce`
- `/produce/fruit`

- `/produce/vegetables/carrot`

고객이 사용 가능한 제품, 제품 카테고리 및 관련 매장 부서를 검색할 수 있게 하려면 읽기 전용 권한으로 `GET /{proxy+}`의 단일 메서드를 공개할 수 있습니다. 마찬가지로 감독자가 produce 부서의 재고를 업데이트할 수 있게 하려면 읽기/쓰기 권한으로 `PUT /produce/{proxy+}`의 또 다른 단일 메서드를 설정할 수 있습니다. 점원이 야채의 총 누계를 업데이트할 수 있게 하려면 읽기/쓰기 권한으로 `POST /produce/vegetables/{proxy+}` 메서드를 설정할 수 있습니다. 매장 관리자가 사용 가능한 제품에 대해 가능한 조치를 수행할 수 있게 하려면 온라인 매장 개발자는 읽기/쓰기 권한으로 `ANY /{proxy+}` 메서드를 공개할 수 있습니다. 어떤 경우에도 런타임 시 고객 또는 직원은 선택한 부서에서 지정된 유형의 특정 제품, 선택한 부서의 특정 생산 카테고리 또는 특정 부서를 선택해야 합니다.

API Gateway 프록시 통합 설정에 대한 자세한 내용은 프록시 리소스를 사용하여 프록시 통합 설정를 참조하세요.

프록시 통합을 위해서는 클라이언트가 백엔드 요구 사항에 대해 보다 자세한 지식을 갖추고 있어야 합니다. 따라서 최적의 앱 성능과 사용자 경험을 보장하려면 백엔드 개발자는 클라이언트 개발자에게 백엔드 요구 사항을 명확하게 전달하고 요구 사항이 충족되지 않으면 강력한 오류 피드백 메커니즘을 제공해야 합니다.

다중 값 헤더 및 쿼리 문자열 파라미터에 대한 지원

API Gateway는 이제 동일한 이름을 지닌 복수의 헤더 및 쿼리 문자열 파라미터를 지원합니다. 다중 값 헤더와 단일 값 헤더 및 파라미터는 동일한 요청 및 응답에서 결합될 수 있습니다. 자세한 내용은 프록시 통합에 대한 Lambda 함수의 입력 형식 및 프록시 통합에 대한 Lambda 함수의 출력 형식 섹션을 참조하세요.

Lambda 프록시 통합을 사용하여 프록시 리소스 설정

Lambda 프록시 통합 유형을 사용하여 프록시 리소스를 설정하려면 복잡한 경로 파라미터(예: `/parent/{proxy+}`)를 사용하여 API 리소스를 생성한 후 이 리소스를 `arn:aws:lambda:us-west-2:123456789012:function:SimpleLambda4ProxyResource` 메서드의 Lambda 함수 백엔드(예: ANY)와 통합합니다. 복잡한 경로 파라미터는 API 리소스 경로의 끝에 와야 합니다. 비 프록시 리소스를 사용할 때와 마찬가지로 API Gateway 콘솔을 사용하거나 OpenAPI 정의 파일을 가져오거나 API Gateway REST API를 직접 호출하여 프록시 리소스를 설정할 수 있습니다.

다음 OpenAPI API 정의 파일은 `SimpleLambda4ProxyResource`라는 Lambda 함수에 통합된 프록시 리소스를 포함하는 API의 예를 보여줍니다.

OpenAPI 3.0

```
{
   "openapi": "3.0.0",
   "info": {
      "version": "2016-09-12T17:50:37Z",
      "title": "ProxyIntegrationWithLambda"
   },
   "paths": {
      "/{proxy+}": {
         "x-amazon-apigateway-any-method": {
            "parameters": [
               {
                  "name": "proxy",
                  "in": "path",
                  "required": true,
                  "schema": {
                     "type": "string"
                  }
               }
            ],
            "responses": {},
            "x-amazon-apigateway-integration": {
               "responses": {
                  "default": {
                     "statusCode": "200"
                  }
               },
               "uri": "arn:aws:apigateway:us-east-1:lambda:path/2015-03-31/
functions/arn:aws:lambda:us-east-1:123456789012:function:SimpleLambda4ProxyResource/
invocations",
               "passthroughBehavior": "when_no_match",
               "httpMethod": "POST",
               "cacheNamespace": "roq9wj",
               "cacheKeyParameters": [
                  "method.request.path.proxy"
               ],
               "type": "aws_proxy"
            }
         }
      }
   },
   "servers": [
      {
```

```
        "url": "https://gy415nuibc.execute-api.us-east-1.amazonaws.com/{basePath}",
        "variables": {
           "basePath": {
             "default": "/testStage"
           }
        }
      }
   ]
}
```

OpenAPI 2.0

```
{
  "swagger": "2.0",
  "info": {
    "version": "2016-09-12T17:50:37Z",
    "title": "ProxyIntegrationWithLambda"
  },
  "host": "gy415nuibc.execute-api.us-east-1.amazonaws.com",
  "basePath": "/testStage",
  "schemes": [
    "https"
  ],
  "paths": {
    "/{proxy+}": {
      "x-amazon-apigateway-any-method": {
        "produces": [
          "application/json"
        ],
        "parameters": [
          {
            "name": "proxy",
            "in": "path",
            "required": true,
            "type": "string"
          }
        ],
        "responses": {},
        "x-amazon-apigateway-integration": {
          "responses": {
            "default": {
              "statusCode": "200"
            }
```

```
          },
          "uri": "arn:aws:apigateway:us-east-1:lambda:path/2015-03-31/functions/
arn:aws:lambda:us-east-1:123456789012:function:SimpleLambda4ProxyResource/
invocations",
          "passthroughBehavior": "when_no_match",
          "httpMethod": "POST",
          "cacheNamespace": "roq9wj",
          "cacheKeyParameters": [
            "method.request.path.proxy"
          ],
          "type": "aws_proxy"
        }
      }
    }
  }
}
```

Lambda 프록시 통합을 사용하여 API Gateway에서는 런타임에 수신 요청을 Lambda 함수의 입력 `event` 파라미터에 매핑합니다. 입력에는 요청 메서드, 경로, 헤더, 쿼리 문자열 파라미터, 페이로드, 연결된 컨텍스트, 정의된 단계 변수 등이 포함됩니다. 입력 형식에 대한 설명은 프록시 통합에 대한 Lambda 함수의 입력 형식 단원을 참조하세요. API Gateway에서 Lambda 출력을 HTTP 응답에 성공적으로 매핑하도록 하려면 Lambda 함수에서 프록시 통합에 대한 Lambda 함수의 출력 형식에 설명된 형식으로 결과를 출력해야 합니다.

`ANY` 메서드를 통한 프록시 리소스의 Lambda 프록시 통합에서는 단일 백엔드 Lambda 함수가 프록시 리소스를 통해 모든 요청에 대한 이벤트 핸들러 역할을 합니다. 예를 들어 트래픽 패턴을 기록하려면 프록시 리소스에 대한 URL 경로에 `/state/city/street/house`를 포함하는 요청을 제출하여 모바일 디바이스에서 시/도, 시, 거리, 빌딩 등 위치 정보를 전송하도록 할 수 있습니다. 그러면 백엔드 Lambda 함수에서 URL 경로를 구문 분석한 다음 위치 튜플을 DynamoDB 테이블에 삽입할 수 있습니다.

를 사용하여 Lambda 프록시 통합을 설정합니다. AWS CLI

이 섹션에서는 Lambda AWS CLI 프록시 통합을 사용하여 API를 설정하는 방법을 보여줍니다.

> **Note**
>
> API Gateway 콘솔을 사용하여 Lambda 프록시 통합을 통해 프록시 리소스를 구성하는 방법에 대한 자세한 내용은 자습서: Lambda 프록시 통합을 사용하여 Hello World REST API 빌드 단원을 참조하세요.

예를 들면 다음과 같은 샘플 Lambda 함수를 API의 백엔드로 사용합니다.

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

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

이를 Lambda 사용자 지정 통합 설정과 비교하여 이 Lambda 함수에 대한 입력을 요청 파라미터 및 본문으로 표현할 수 있습니다. 클라이언트가 동일한 입력 데이터를 전달하도록 허용하는 더 많은 위도가 있습니다. 여기에서 클라이언트는 greeter의 이름을 쿼리 문자열 파라미터, 헤더 또는 본문 속성으로 전달할 수 있습니다. 함수는 Lambda 사용자 지정 통합도 지원합니다. API 설정은 더 간단합니다. 메서드 응답 또는 통합 응답은 전혀 구성하지 않습니다.

를 사용하여 Lambda 프록시 통합을 설정하려면 AWS CLI

1. 다음과 같이 `create-rest-api` 명령을 호출하여 API를 생성합니다.

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

다음과 같이 그 결과로 얻은 응답에 있는 API의 id 값(te6si5ach7)을 적어 둡니다.

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

이 단원 전체에서 API id가 필요합니다.

2. 다음과 같이 get-resources 명령을 호출하여 루트 리소스 id를 가져옵니다.

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

이 작업이 성공하면 그 응답은 다음과 같습니다.

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

루트 리소스 id 값(krznpq9xpg)을 기록해 둡니다. 다음 단계 및 그 이후에 이 값이 필요합니다.

3. 다음과 같이 create-resource를 호출하여 /greeting의 API Gateway 리소스를 생성합니다.

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

성공적인 응답은 다음과 같습니다.

```
{
    "path": "/{proxy+}",
```

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

{proxy+} 리소스의 id 값(2jf6xt)을 기록해 둡니다. 다음 단계에서 /{proxy+} 리소스에 메서드를 생성하려면 이 값이 필요합니다.

4. 다음과 같이 put-method를 호출하여 ANY의 ANY /{proxy+} 메서드 요청을 생성합니다.

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

성공적인 응답은 다음과 같습니다.

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

이 API 메서드를 통해 클라이언트는 백엔드에서 Lambda 함수의 인사를 수신 또는 발신할 수 있습니다.

5. put-integration을 호출하여 ANY /{proxy+}라는 Lambda 함수로 HelloWorld 메서드의 통합을 설정합니다. 이 함수는 "Hello, {name}!" 파라미터가 제공되면 greeter이라는 메시지로, 쿼리 문자열 파라미터가 설정되어 있지 않으면 "Hello, World!"라는 메시지로 요청에 응답합니다.

```
aws apigateway put-integration \
        --region us-west-2 \
        --rest-api-id te6si5ach7 \
        --resource-id 2jf6xt \
        --http-method ANY \
        --type AWS_PROXY \
        --integration-http-method POST \
        --uri arn:aws:apigateway:us-west-2:lambda:path/2015-03-31/functions/
arn:aws:lambda:us-west-2:123456789012:function:HelloWorld/invocations \
```

```
        --credentials arn:aws:iam::123456789012:role/apigAwsProxyRole
```

> **Important**
>
> Lambda 통합의 경우에는 함수 호출을 위한 Lambda 서비스 작업의 사양에 따라 통합 요청에 대해 POST의 HTTP 메서드를 사용해야 합니다. apigAwsProxyRole의 IAM 역할에는 apigateway 서비스가 Lambda 함수를 호출하도록 허용하는 정책이 있어야 합니다. IAM 권한에 대한 자세한 내용은 the section called “ API 호출을 위한 API Gateway 권한 모델” 단원을 참조하세요.

성공적인 출력 결과는 다음과 같습니다.

```
{
    "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"
}
```

credentials에 IAM 역할을 제공하는 대신 add-permission 명령을 호출하여 리소스 기반 권한을 추가할 수 있습니다. 이 작업은 API Gateway 콘솔이 수행합니다.

6. 다음과 같이 create-deployment를 호출하여 API를 test 단계에 배포합니다.

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

7. 터미널에서 다음 cURL 명령을 사용하여 API를 테스트합니다.

   ?greeter=jane이라는 쿼리 문자열 파라미터와 함께 API를 호출:

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

   greeter:jane이라는 헤더 파라미터와 함께 API를 호출:

```
curl -X GET https://te6si5ach7.execute-api.us-west-2.amazonaws.com/test/hi \
  -H 'content-type: application/json' \
  -H 'greeter: jane'
```

`{"greeter":"jane"}`이라는 본문과 함께 API를 호출:

```
curl -X POST https://te6si5ach7.execute-api.us-west-2.amazonaws.com/test/hi \
  -H 'content-type: application/json' \
  -d '{ "greeter": "jane" }'
```

이 모든 경우에 출력되는 것은 다음과 같은 응답 본문이 있는 200 응답입니다.

```
Hello, jane!
```

프록시 통합에 대한 Lambda 함수의 입력 형식

Lambda 프록시 통합을 사용할 경우 API Gateway에서 전체 클라이언트 요청을 백엔드 Lambda 함수의 입력 `event` 파라미터에 매핑합니다. 다음 예제에서는 API Gateway가 Lambda 프록시 통합으로 보내는 이벤트의 구조를 보여줍니다.

```
{
  "resource": "/my/path",
  "path": "/my/path",
  "httpMethod": "GET",
  "headers": {
    "header1": "value1",
    "header2": "value1,value2"
  },
  "multiValueHeaders": {
    "header1": [
      "value1"
    ],
    "header2": [
      "value1",
      "value2"
    ]
  },
  "queryStringParameters": {
    "parameter1": "value1,value2",
```

```
    "parameter2": "value"
  },
  "multiValueQueryStringParameters": {
    "parameter1": [
      "value1",
      "value2"
    ],
    "parameter2": [
      "value"
    ]
  },
  "requestContext": {
    "accountId": "123456789012",
    "apiId": "id",
    "authorizer": {
      "claims": null,
      "scopes": null
    },
    "domainName": "id.execute-api.us-east-1.amazonaws.com",
    "domainPrefix": "id",
    "extendedRequestId": "request-id",
    "httpMethod": "GET",
    "identity": {
      "accessKey": null,
      "accountId": null,
      "caller": null,
      "cognitoAuthenticationProvider": null,
      "cognitoAuthenticationType": null,
      "cognitoIdentityId": null,
      "cognitoIdentityPoolId": null,
      "principalOrgId": null,
      "sourceIp": "IP",
      "user": null,
      "userAgent": "user-agent",
      "userArn": null,
      "clientCert": {
        "clientCertPem": "CERT_CONTENT",
        "subjectDN": "www.example.com",
        "issuerDN": "Example issuer",
        "serialNumber": "a1:a1:a1:a1:a1:a1:a1:a1:a1:a1:a1:a1:a1:a1:a1:a1",
        "validity": {
          "notBefore": "May 28 12:30:02 2019 GMT",
          "notAfter": "Aug  5 09:36:04 2021 GMT"
        }
```

```
      }
    },
    "path": "/my/path",
    "protocol": "HTTP/1.1",
    "requestId": "id=",
    "requestTime": "04/Mar/2020:19:15:17 +0000",
    "requestTimeEpoch": 1583349317135,
    "resourceId": null,
    "resourcePath": "/my/path",
    "stage": "$default"
  },
  "pathParameters": null,
  "stageVariables": null,
  "body": "Hello from Lambda!",
  "isBase64Encoded": false
}
```

> **Note**
>
> 입력에서:
>
> - `headers` 키에는 단일 값 헤더만 있습니다.
> - `multiValueHeaders` 키에는 다중 값 헤더와 단일 값 헤더가 있을 수 있습니다.
> - `headers` 및 `multiValueHeaders` 모두에 대해 값을 지정한 경우, API Gateway는 이들 헤더를 단일 목록으로 병합합니다. 동일한 키-값 페어가 양쪽 모두에 지정된 경우, `multiValueHeaders`의 값만이 병합된 목록에 표시됩니다.

백엔드 Lambda 함수에 대한 입력에서 `requestContext` 객체는 키-값 페어의 맵입니다. 각 페어에서 키는 $context 변수 속성의 이름이고, 값은 그 속성의 값입니다. API Gateway는 새 키를 맵에 추가할 수 있습니다.

활성화된 기능에 따라 `requestContext` 맵은 API마다 다를 수 있습니다. 예를 들어 앞의 예제에서는 권한 부여 유형이 지정되지 않았으므로 `$context.authorizer.*` 또는 `$context.identity.*` 속성이 없습니다. 권한 부여 유형이 지정된 경우에는 다음과 같이 API Gateway가 `requestContext.identity` 객체의 통합 엔드포인트에 권한이 부여된 사용자 정보를 전달합니다.

- 권한 부여 유형이 `AWS_IAM`인 경우, 권한 부여된 사용자 정보에 `$context.identity.*` 속성이 포함됩니다.

- 권한 부여 유형이 COGNITO_USER_POOLS(Amazon Cognito 권한 부여자)인 경우, 권한 부여된 사용자 정보에 $context.identity.cognito* 및 $context.authorizer.claims.* 속성이 포함됩니다.
- 권한 부여 유형이 CUSTOM(Lambda 권한 부여자)인 경우, 권한 부여된 사용자 정보에 $context.authorizer.principalId 및 기타 적용되는 $context.authorizer.* 속성이 포함됩니다.

프록시 통합에 대한 Lambda 함수의 출력 형식

Lambda 프록시 통합을 사용할 경우 API Gateway에서는 백엔드 Lambda 함수에 다음 JSON 형식에 따라 출력을 반환하도록 요구합니다.

```
{
    "isBase64Encoded": true|false,
    "statusCode": httpStatusCode,
    "headers": { "headerName": "headerValue", ... },
    "multiValueHeaders": { "headerName": ["headerValue", "headerValue2", ...], ... },
    "body": "..."
}
```

출력에서:

- 출력에서 추가 응답 헤더가 반환되지 않을 경우 headers 및 multiValueHeaders 키가 비지정 상태일 수 있습니다.
- headers 키에는 단일 값 헤더만 있습니다.
- multiValueHeaders 키에는 다중 값 헤더와 단일 값 헤더가 있을 수 있습니다. multiValueHeaders 키를 사용하여 모든 단일 값 헤더를 포함하는 모든 추가 헤더를 지정할 수 있습니다.
- headers 및 multiValueHeaders 모두에 대해 값을 지정한 경우, API Gateway는 이들 헤더를 단일 목록으로 병합합니다. 동일한 키-값 페어가 양쪽 모두에 지정된 경우, multiValueHeaders의 값만이 병합된 목록에 표시됩니다.

Lambda 프록시 통합에 대해 CORS를 활성화하려면 출력 headers에 Access-Control-Allow-Origin:*domain-name*을 추가해야 합니다. *domain-name*은 모든 도메인 이름에 대해 *일 수 있습니다. 출력 body는 프런트엔드에 메서드 응답 페이로드로 마샬링됩니다. body가 이진 BLOB인 경우 isBase64Encoded를 true으로 설정하고, 이진 미디어 유형을 */*로 구성하여 Base64로 인코딩된

문자열로 인코딩할 수 있습니다. 그렇지 않은 경우 `false`로 설정하거나 지정되지 않은 상태로 둘 수 있습니다.

> **Note**
>
> 이진 지원 활성화에 대한 자세한 내용은 API Gateway 콘솔을 사용하여 이진 지원 활성화 단원을 참조하세요. Lambda 함수의 예는 Lambda 프록시 통합에서 이진 미디어 반환를 참조하세요.

함수 출력의 형식이 다른 경우 API Gateway에서는 `502 Bad Gateway` 오류 응답을 반환합니다.

Node.js에서 Lambda 함수의 응답을 반환하려면 다음과 같은 명령어를 사용할 수 있습니다.

- 성공 결과를 반환하려면 `callback(null, {"statusCode": 200, "body": "results"})`을 호출합니다.
- 예외를 발생하려면 `callback(new Error('internal server error'))`를 호출합니다.
- 클라이언트 측 오류의 경우(예: 필요한 파라미터가 없는 경우) `callback(null, {"statusCode": 400, "body": "Missing parameters of ..."})`를 호출하여 예외를 발생하지 않고 오류를 반환할 수 있습니다.

Node.js의 Lambda `async` 함수에서 이와 동일한 구문은 다음과 같습니다.

- 성공 결과를 반환하려면 `return {"statusCode": 200, "body": "results"}`을 호출합니다.
- 예외를 발생하려면 `throw new Error("internal server error")`를 호출합니다.
- 클라이언트 측 오류의 경우(예: 필요한 파라미터가 없는 경우) `return {"statusCode": 400, "body": "Missing parameters of ..."}`를 호출하여 예외를 발생하지 않고 오류를 반환할 수 있습니다.

API Gateway에서 Lambda 사용자 지정 통합 설정

Lambda 사용자 지정 통합을 설정하는 방법을 보여주기 위해 API Gateway API를 생성하여 Lambda 함수를 호출할 `GET /greeting?greeter={name}` 메서드를 공개합니다. 다음 예시 Lambda 함수 중 하나를 API에 사용하십시오.

다음 예시 Lambda 함수 중 하나를 사용하십시오.

Node.js

```
export const handler = function(event, context, callback) {
    var res ={
        "statusCode": 200,
        "headers": {
            "Content-Type": "*/*"
        }
    };
    if (event.greeter==null) {
        callback(new Error('Missing the required greeter parameter.'));
    } else if (event.greeter === "") {
        res.body = "Hello, World";
        callback(null, res);
    } else {
        res.body = "Hello, " + event.greeter +"!";
        callback(null, res);
    }
};
```

Python

```
import json


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

    if event['greeter'] == "":
        res['body'] = "Hello, World"
    elif (event['greeter']):
        res['body'] = "Hello, " + event['greeter'] + "!"
    else:
        raise Exception('Missing the required greeter parameter.')

    return res
```

이 함수는 "Hello, {name}!" 파라미터 값이 비어 있지 않은 문자열인 경우 greeter이라는 메시지로 응답합니다. "Hello, World!" 값이 빈 문자열이면 greeter라는 메시지를 반환합니다. 수신 요청에 greeter 파라미터가 설정되지 않은 경우 함수는 "Missing the required greeter parameter."라는 오류 메시지를 반환합니다. 이 함수의 이름을 HelloWorld로 지정합니다.

Lambda 콘솔에서 또는 AWS CLI를 사용하여 이를 생성할 수 있습니다. 이 단원에서는 다음 ARN을 사용하여 이 함수를 참조합니다.

```
arn:aws:lambda:us-east-1:123456789012:function:HelloWorld
```

백엔드에 설정된 Lambda 함수를 사용하여 API를 설정하는 단계로 이동합니다.

를 사용하여 Lambda 사용자 지정 통합을 설정하려면 AWS CLI

1. 다음과 같이 create-rest-api 명령을 호출하여 API를 생성합니다.

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

   다음과 같이 그 결과로 얻은 응답에 있는 API의 id 값(te6si5ach7)을 적어 둡니다.

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

   이 단원 전체에서 API id가 필요합니다.
2. 다음과 같이 get-resources 명령을 호출하여 루트 리소스 id를 가져옵니다.

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

   이 작업이 성공하면 그 응답은 다음과 같습니다.

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

```
    ]
}
```

루트 리소스 id 값(krznpq9xpg)을 기록해 둡니다. 다음 단계 및 그 이후에 이 값이 필요합니다.

3. 다음과 같이 create-resource를 호출하여 /greeting의 API Gateway 리소스를 생성합니다.

```
aws apigateway create-resource --rest-api-id te6si5ach7 \
      --region us-west-2 \
      --parent-id krznpq9xpg \
      --path-part greeting
```

성공적인 응답은 다음과 같습니다.

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

greeting 리소스의 id 값(2jf6xt)을 기록해 둡니다. 다음 단계에서 /greeting 리소스에 메서드를 생성하려면 이 값이 필요합니다.

4. put-method를 호출하여 GET /greeting?greeter={name}의 API 메서드 요청을 생성합니다.

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

성공적인 응답은 다음과 같습니다.

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

```
        "method.request.querystring.greeter": false
    }
}
```

이 API 메서드를 통해 클라이언트는 백엔드에서 Lambda 함수의 인사를 수신할 수 있습니다. greeter 파라미터는 선택 사항인데, 그 이유는 백엔드가 익명 호출자 또는 자기 동일 호출자를 처리해야 하기 때문입니다.

5. 다음과 같이 put-method-response 함수를 호출하여 200 OK의 메서드 요청에 GET /greeting?greeter={name} 응답을 설정합니다.

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

6. put-integration을 호출하여 GET /greeting?greeter={name}라는 Lambda 함수로 HelloWorld 메서드의 통합을 설정합니다. 이 함수는 "Hello, {name}!" 파라미터가 제공되면 greeter이라는 메시지로, 쿼리 문자열 파라미터가 설정되어 있지 않으면 "Hello, World!"라는 메시지로 요청에 응답합니다.

```
aws apigateway put-integration \
        --region us-west-2 \
        --rest-api-id te6si5ach7 \
        --resource-id 2jf6xt \
        --http-method GET \
        --type AWS \
        --integration-http-method POST \
        --uri arn:aws:apigateway:us-east-1:lambda:path/2015-03-31/functions/
arn:aws:lambda:us-east-1:123456789012:function:HelloWorld/invocations \
        --request-templates '{"application/json":"{\"greeter\":
\"$input.params('greeter')\"}"}' \
        --credentials arn:aws:iam::123456789012:role/apigAwsProxyRole
```

여기에 제공된 매핑 템플릿은 greeter 쿼리 문자열 파라미터를 JSON 페이로드의 greeter 속성으로 변환합니다. 이것이 필요한 이유는 Lambda 함수에 대한 입력은 본문에 표시되어야 하기 때문입니다.

> **Important**
>
> Lambda 통합의 경우에는 함수 호출을 위한 Lambda 서비스 작업의 사양에 따라 통합 요청에 대해 POST의 HTTP 메서드를 사용해야 합니다. `uri` 파라미터는 함수 호출 작업의 ARN입니다.
> 성공적인 출력 결과는 다음과 같습니다.

```
{
    "passthroughBehavior": "WHEN_NO_MATCH",
    "cacheKeyParameters": [],
    "uri": "arn:aws:apigateway:us-east-1:lambda:path/2015-03-31/functions/
arn:aws:lambda:us-east-1:123456789012:function:HelloWorld/invocations",
    "httpMethod": "POST",
    "requestTemplates": {
        "application/json": "{\"greeter\":\"$input.params('greeter')\"}"
    },
    "cacheNamespace": "krznpq9xpg",
    "credentials": "arn:aws:iam::123456789012:role/apigAwsProxyRole",
    "type": "AWS"
}
```

`apigAwsProxyRole`의 IAM 역할에는 `apigateway` 서비스가 Lambda 함수를 호출하도록 허용하는 정책이 있어야 합니다. `credentials`에 IAM 역할을 제공하는 대신 add-permission 명령을 호출하여 리소스 기반 권한을 추가할 수 있습니다. API Gateway 콘솔은 이 방식으로 권한을 추가합니다.

7. `put-integration-response`를 호출하여 통합 응답을 설정함으로써 클라이언트에게 Lambda 함수 출력을 `200 OK` 메서드 응답으로 전달합니다.

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

빈 문자열에 선택 패턴을 설정하면 `200 OK` 응답이 기본값입니다.

올바른 응답은 다음과 유사합니다.

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

8. 다음과 같이 `create-deployment`를 호출하여 API를 `test` 단계에 배포합니다.

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

9. 터미널에서 다음 cURL 명령을 사용하여 API를 테스트합니다.

```
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'
```

백엔드 Lambda 함수의 비동기 호출 설정

Lambda 비프록시(사용자 지정)에서, 백엔드 Lambda 함수는 기본적으로 동기적으로 호출됩니다. 이 동작은 대부분의 REST API 작업에 적합합니다. 그러나 일부 애플리케이션은 일반적으로 별도의 백엔드 구성 요소에 의해 (배치 작업 또는 장기 지연 작업으로) 비동기적으로 작업을 수행해야 합니다. 이 경우 백엔드 Lambda 함수는 비동기적으로 호출되며 프런트엔드 REST API 메서드가 결과를 반환하지 않습니다.

Lambda 호출 유형을 `'Event'`로 지정하여 Lambda 함수를 Lambda 비프록시 통합이 비동기적으로 호출되도록 구성할 수 있습니다. 구성 방법은 다음과 같습니다.

API Gateway 콘솔에서 Lambda 비동기 호출 구성

모든 호출이 비동기식 인 경우:

- 통합 요청에서 정적 값이 `'Event'`인 `X-Amz-Invocation-Type` 헤더를 추가합니다.

호출이 비동기식인지 동기식인지 여부를 클라이언트가 결정하려면 다음을 수행하세요.

1. 메서드 요청에서 InvocationType 헤더를 추가합니다.
2. 통합 요청에서 매핑 표현식이 method.request.header.InvocationType인 X-Amz-Invocation-Type 헤더를 추가합니다.
3. 클라이언트는 API 요청에 비동기식 호출의 경우 InvocationType: Event 헤더를, 동기식 호출의 경우 InvocationType: RequestResponse 헤더를 포함할 수 있습니다.

OpenAway를 사용하여 Lambda 비동기 호출 구성

모든 호출이 비동기식 인 경우:

- 섹션에 X-Amz-Invocation-Type 헤더를 추가합니다. 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"
        }
```

호출이 비동기식인지 동기식인지 여부를 클라이언트가 결정하려면 다음을 수행하세요.

1. 모든 OpenAPI Path Item Object(OpenAPI 경로 항목 객체)에 다음 헤더를 추가합니다.

```
"parameters" : [ {
"name" : "InvocationType",
"in" : "header",
"schema" : {
  "type" : "string"
}
```

```
} ]
```

2. x-amazon-apigateway-integration섹션에 `X-Amz-Invocation-Type` 헤더를 추가합니다.

```
"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. 클라이언트는 API 요청에 비동기식 호출의 경우 `InvocationType: Event` 헤더를, 동기식 호출의 경우 `InvocationType: RequestResponse` 헤더를 포함할 수 있습니다.

API Gateway에서 Lambda 오류 처리

Lambda 사용자 지정 통합의 경우에는 Lambda가 통합 응답에서 반환한 오류를 클라이언트에 대한 표준 HTTP 오류 응답으로 매핑해야 합니다. 그렇게 하지 않으면 Lambda 오류는 기본적으로 API 사용자가 직관적으로 이해하기 어려운 `200 OK` 응답으로 반환됩니다.

Lambda가 반환할 수 있는 오류 유형은 표준 오류와 사용자 지정 오류, 두 가지입니다. API에서 이 두 가지 오류를 다르게 처리해야 합니다.

Lambda 프록시 통합의 경우, Lambda는 다음 형식의 출력을 반환해야 합니다.

```
{
  "isBase64Encoded" : "boolean",
  "statusCode": "number",
  "headers": { ... },
  "body": "JSON string"
}
```

이 출력에서 statusCode는 일반적으로 클라이언트 오류의 경우 4XX, 서버 오류의 경우 5XX입니다. API Gateway는 지정된 statusCode에 따라 Lambda 오류를 HTTP 오류 응답에 매핑함으로써 이러한 오류를 처리합니다. API Gateway에서 클라이언트에 대한 응답의 일부로 오류 유형(예: InvalidParameterException)을 전달하려면 Lambda 함수가 headers 속성에 헤더(예: "X-Amzn-ErrorType":"InvalidParameterException")를 포함해야 합니다.

주제

- API Gateway에서 표준 Lambda 오류 처리
- API Gateway에서 사용자 지정 Lambda 오류 처리

API Gateway에서 표준 Lambda 오류 처리

표준 AWS Lambda의 오류 형식은 다음과 같습니다.

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

여기서 errorMessage는 오류의 문자열 표현식입니다. errorType은 언어 종속적 오류 또는 예외 유형입니다. stackTrace는 문자열 표현식 목록으로, 오류를 유발하는 스택 추적을 보여 줍니다.

예를 들어 다음과 같은 JavaScript(Node.js) Lambda 함수를 생각해 보세요.

```
export const handler = function(event, context, callback) {
    callback(new Error("Malformed input ..."));
};
```

이 함수는 오류 메시지 Malformed input ...이 포함된 다음과 같은 표준 Lambda 오류를 반환합니다.

```
{
  "errorMessage": "Malformed input ...",
  "errorType": "Error",
  "stackTrace": [
    "export const handler (/var/task/index.js:3:14)"
```

```
  ]
}
```

마찬가지로 다음과 같은 Python Lambda 함수를 생각해 보세요. 동일한 `Malformed input ...` 오류 메시지와 함께 `Exception`을 야기하는 함수입니다.

```
def lambda_handler(event, context):
    raise Exception('Malformed input ...')
```

이 함수는 다음과 같은 표준 Lambda 오류를 반환합니다.

```
{
  "stackTrace": [
    [
      "/var/task/lambda_function.py",
      3,
      "lambda_handler",
      "raise Exception('Malformed input ...')"
    ]
  ],
  "errorType": "Exception",
  "errorMessage": "Malformed input ..."
}
```

`errorType` 및 `stackTrace` 속성 값은 언어 종속적입니다. 또한 이 표준 오류는 `Error` 객체의 확장이거나 `Exception` 클래스의 하위 클래스인 모든 오류 객체에 적용됩니다.

표준 Lambda 오류를 메서드 응답에 매핑하려면, 먼저 주어진 Lambda 오류의 HTTP 상태 코드를 결정해야 합니다. 그런 다음 주어진 HTTP 상태 코드와 연결된 IntegrationResponse의 `selectionPattern` 속성에서 정규 표현식 패턴을 설정합니다. API Gateway 콘솔에서 이 `selectionPattern`은 각 통합 응답 아래의 통합 응답 섹션에 Lambda 오류 Regex로 표시됩니다.

> **Note**
>
> API Gateway는 응답 매핑을 위해서 Java 패턴 스타일 정규식을 사용합니다. 자세한 내용은 Oracle 설명서의 패턴 단원을 참조하십시오

예를 들어, `selectionPattern`를 사용하여 새 AWS CLI 표현식을 설정하려면 put-integration-response 명령을 호출합니다.

```
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
```

메서드 응답에 해당하는 오류 코드(400)도 설정해야 합니다. 그렇지 않으면 API Gateway는 실행 시간에 잘못된 구성 오류 응답을 내보냅니다.

> **Note**
>
> 실행 시간에 API Gateway는 Lambda 오류의 `errorMessage`를 메시지를 `selectionPattern` 속성의 정규식 패턴과 비교합니다. 양쪽이 일치하면 API Gateway는 Lambda 오류를 해당 HTTP 상태 코드의 HTTP 응답으로 반환합니다. 일치하지 않으면 API Gateway는 오류를 기본 응답으로 반환하고, 만일 기본 응답이 구성되어 있지 않으면 잘못된 구성 예외로 내보냅니다.
>
> 지정된 응답에 대해 `selectionPattern` 값을 `.*`로 설정하면 이 응답이 기본 응답으로 다시 설정됩니다. 이는 해당 선택 패턴이 null(즉. 지정되지 않은 오류 메시지)을 포함하는 모든 오류 메시지와 일치하기 때문입니다. 그 결과로 나타나는 매핑은 기본 매핑을 무시합니다.

`selectionPattern`를 사용하여 기존의 AWS CLI 값을 업데이트하려면 integrationresponse:update 작업을 호출하여 `/selectionPattern` 경로 값을 `Malformed*` 패턴의 지정된 regex 표현식으로 바꿉니다.

API Gateway 콘솔을 사용하여 `selectionPattern` 표현식을 설정하려면, 지정된 HTTP 상태 코드의 통합 응답을 설정하거나 업데이트할 때 Lambda 오류 Regex 텍스트 상자에 원하는 표현식을 입력합니다.

API Gateway에서 사용자 지정 Lambda 오류 처리

AWS Lambda에서는 사용자 지정 오류 객체를 앞 단원에서 다룬 표준 오류 대신 JSON 문자열로 반환할 수 있습니다. 유효한 모든 JSON 객체를 오류로 사용할 수 있습니다. 예를 들어 다음 JavaScript(Node.js) Lambda 함수는 사용자 지정 오류를 반환합니다.

```
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));
};
```

myErrorObj을 호출하여 함수를 종료하려면 먼저 callback 객체를 JSON 문자열로 변환해야 합니다. 그렇지 않으면 myErrorObj가 "[object Object]"의 문자열로 반환됩니다. API 메서드가 앞의 Lambda 함수와 통합되면 API Gateway는 다음과 같은 페이로드의 통합 응답을 받습니다.

```
{
    "errorMessage": "{\"errorType\":\"InternalServerError\",\"httpStatus\":500,
\"requestId\":\"e5849002-39a0-11e7-a419-5bb5807c9fb2\",\"trace\":{\"function\":
\"abc()\",\"line\":123,\"file\":\"abc.js\"}}"
}
```

통합 응답과 마찬가지로 이 오류 응답을 그대로 메서드 응답으로 전달할 수 있습니다. 아니면 매핑 템플릿을 사용하여 페이로드를 다른 형식으로 변환할 수도 있습니다. 예를 들어, 500 상태 코드의 메서드 응답이라면 다음과 같은 본문 매핑 템플릿을 고려해 보십시오.

```
{
    errorMessage: $input.path('$.errorMessage');
}
```

이 템플릿은 사용자 지정 오류 JSON 문자열이 포함된 통합 응답 본문을 다음과 같은 메서드 응답 본문으로 변환합니다. 이 메서드 응답 본문에는 사용자 지정 오류 JSON 객체가 들어 있습니다.

```
{
    "errorMessage" : {
        errorType : "InternalServerError",
        httpStatus : 500,
        requestId : context.awsRequestId,
        trace : {
            "function": "abc()",
            "line": 123,
            "file": "abc.js"
        }
    }
```

```
};
```

API 요구 사항에 따라 사용자 지정 오류 속성 중 일부 또는 전부를 메서드 응답 헤더 파라미터로 전달해야 할 수 있습니다. 그러려면 통합 응답 본문의 사용자 지정 오류 매핑을 메서드 응답 헤더에 적용하면 됩니다.

예를 들어, 다음 OpenAPI 확장은 각각 `errorMessage.errorType`, `errorMessage.httpStatus`, `errorMessage.trace.function` 및 `errorMessage.trace` 속성에서 `error_type`, `error_status`, `error_trace_function` 및 `error_trace` 헤더로 매핑을 정의합니다.

```
"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"
          },
          ...
        }
    }
}
```

API Gateway는 실행 시간에 헤더 매핑을 수행할 때 `integration.response.body` 파라미터를 역직렬화합니다. 그러나 이런 역직렬화는 Lambda 사용자 지정 오류 응답의 본문-헤더 매핑에만 적용되며, `$input.body`를 사용하는 본문-본문 매핑에는 적용되지 않습니다. 이러한 사용자 지정 오류의 본문-헤더 매핑을 사용할 경우, 클라이언트가 받는 메서드 응답에 다음 헤더가 포함되어 있습니다. 단, 메서드 요청에 `error_status`, `error_trace`, `error_trace_function`, `error_type` 헤더가 선언되어 있어야 합니다.

```
"error_status":"500",
"error_trace":"{\"function\":\"abc()\",\"line\":123,\"file\":\"abc.js\"}",
"error_trace_function":"abc()",
"error_type":"InternalServerError"
```

통합 응답 본문의 `errorMessage.trace` 속성은 복잡한 속성입니다. `error_trace` 헤더에 JSON 문자열로 매핑됩니다.

API Gateway에서 HTTP 통합 설정

HTTP 프록시 통합 또는 HTTP 사용자 지정 통합을 사용하여 API 메서드를 HTTP 엔드포인트와 통합할 수 있습니다.

API Gateway는 80, 443, 1024-65535 엔드포인트 포트를 지원합니다.

프록시 통합을 사용하면 설정이 간단합니다. 콘텐츠 인코딩 또는 캐싱에 관심이 없다면 백엔드 요구 사항에 따라 HTTP 메서드 및 HTTP 엔드포인트 URI를 설정하기만 하면 됩니다.

사용자 지정 통합은 설정이 이보다 복잡합니다. 프록시 통합 설정 단계 외에도 수신 요청 데이터가 통합 요청에 매핑되는 방식과 그 결과로 얻는 통합 응답 데이터가 메서드 응답에 매핑되는 방식을 지정해야 합니다.

주제

- API Gateway에서 HTTP 프록시 통합 설정
- API Gateway에서 HTTP 사용자 지정 통합 설정

API Gateway에서 HTTP 프록시 통합 설정

HTTP 프록시 통합 유형을 사용하여 프록시 리소스를 설정하려면 복잡한 경로 파라미터(예: `/parent/{proxy+}`)를 사용하여 API 리소스를 생성한 후 이 리소스를 `https://petstore-demo-endpoint.execute-api.com/petstore/{proxy}` 메서드의 HTTP 백엔드 엔드포인트(예: ANY)와 통합합니다. 복잡한 경로 파라미터는 리소스 경로의 끝에 와야 합니다.

비 프록시 리소스를 사용할 때와 마찬가지로 API Gateway 콘솔을 사용하거나, OpenAPI 정의 파일을 가져오거나, API Gateway REST API를 직접 호출하여 HTTP 프록시 통합을 통해 프록시 리소스를 설정할 수 있습니다. API Gateway 콘솔을 사용하여 HTTP 프록시 통합을 통해 프록시 리소스를 구성하는 방법에 대한 자세한 내용은 자습서: HTTP 프록시 통합을 사용하여 REST API 구축 단원을 참조하세요.

다음 OpenAPI 정의 파일은 PetStore 웹사이트에 통합된 프록시 리소스를 포함하는 API의 예를 보여줍니다.

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"
        }
      }
    }
  }
}
```

이 예에서 캐시 키는 프록시 리소스의 `method.request.path.proxy` 경로 파라미터에 선언됩니다. 이는 API Gateway 콘솔을 사용하여 API를 생성할 때의 기본 설정입니다. API의 기본 경로(`/test`, 단계에 해당함)는 웹사이트의 PetStore 페이지(`/petstore`)에 매핑됩니다. 단일 통합 요청으로 API의 복잡한 경로 변수와 완전 포착(catch-all) ANY 메서드를 사용하여 전체 PetStore 웹사이트를 미러링합니다. 다음 예에서는 이 미러링을 보여줍니다.

- **ANY**을(를) **GET**으로 설정하고 **{proxy+}**을(를) **pets**로 설정

  프런트엔드에서 시작되는 메서드 요청:

  ```
  GET https://4z9giyi2c1.execute-api.us-west-2.amazonaws.com/test/pets HTTP/1.1
  ```

  백엔드로 전송되는 통합 요청:

  ```
  GET http://petstore-demo-endpoint.execute-api.com/petstore/pets HTTP/1.1
  ```

  ANY 메서드와 프록시 리소스의 런타임 인스턴스가 모두 유효합니다. 이 호출은 백엔드에서 반환되는 반려 동물의 첫 번째 배치를 포함하는 페이로드와 함께 `200 OK` 응답을 반환합니다.

- **ANY**을(를) **GET**으로 설정하고 **{proxy+}**을(를) **pets?type=dog**로 설정

  ```
  GET https://4z9giyi2c1.execute-api.us-west-2.amazonaws.com/test/pets?type=dog
   HTTP/1.1
  ```

백엔드로 전송되는 통합 요청:

```
GET http://petstore-demo-endpoint.execute-api.com/petstore/pets?type=dog HTTP/1.1
```

ANY 메서드와 프록시 리소스의 런타임 인스턴스가 모두 유효합니다. 이 호출은 백엔드에서 반환되는 지정된 개의 첫 번째 배치를 포함하는 페이로드와 함께 200 OK 응답을 반환합니다.

- **ANY**을(를) **GET**으로 설정하고 **{proxy+}**을(를) **pets/{petId}**로 설정

  프런트엔드에서 시작되는 메서드 요청:

  ```
  GET https://4z9giyi2c1.execute-api.us-west-2.amazonaws.com/test/pets/1 HTTP/1.1
  ```

  백엔드로 전송되는 통합 요청:

  ```
  GET http://petstore-demo-endpoint.execute-api.com/petstore/pets/1 HTTP/1.1
  ```

  ANY 메서드와 프록시 리소스의 런타임 인스턴스가 모두 유효합니다. 이 호출은 백엔드에서 반환되는 지정된 반려 동물을 포함하는 페이로드와 함께 200 OK 응답을 반환합니다.

- **ANY**을(를) **POST**으로 설정하고 **{proxy+}**을(를) **pets**로 설정

  프런트엔드에서 시작되는 메서드 요청:

  ```
  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
  }
  ```

  백엔드로 전송되는 통합 요청:

  ```
  POST http://petstore-demo-endpoint.execute-api.com/petstore/pets HTTP/1.1
  Content-Type: application/json
  Content-Length: ...

  {
  ```

```
  "type" : "dog",
  "price" : 1001.00
}
```

ANY 메서드와 프록시 리소스의 런타임 인스턴스가 모두 유효합니다. 이 호출은 백엔드에서 반환되는 새로 생성된 반려 동물을 포함하는 페이로드와 함께 `200 OK` 응답을 반환합니다.

- **ANY**을(를) **GET**으로 설정하고 **{proxy+}**을(를) **pets/cat**로 설정

  프런트엔드에서 시작되는 메서드 요청:

  ```
  GET https://4z9giyi2c1.execute-api.us-west-2.amazonaws.com/test/pets/cat
  ```

  백엔드로 전송되는 통합 요청:

  ```
  GET http://petstore-demo-endpoint.execute-api.com/petstore/pets/cat
  ```

  프록시 리소스 경로의 런타임 인스턴스가 백엔드 엔드포인트와 일치하지 않고 결과 요청이 잘못되었습니다. 따라서 `400 Bad Request` 응답이 다음 오류 메시지와 함께 반환됩니다.

  ```
  {
    "errors": [
      {
        "key": "Pet2.type",
        "message": "Missing required field"
      },
      {
        "key": "Pet2.price",
        "message": "Missing required field"
      }
    ]
  }
  ```

- **ANY**을(를) **GET**으로 설정하고 **{proxy+}**을(를) **null**로 설정

  프런트엔드에서 시작되는 메서드 요청:

  ```
  GET https://4z9giyi2c1.execute-api.us-west-2.amazonaws.com/test
  ```

  백엔드로 전송되는 통합 요청:

```
GET http://petstore-demo-endpoint.execute-api.com/petstore/pets
```

타겟 리소스는 프록시 리소스의 상위이지만, ANY 메서드의 런타임 인스턴스가 해당 리소스의 API에 정의되어 있지 않습니다. 따라서 이 GET 요청은 API Gateway에서 반환되는 403 Forbidden 오류 메시지와 함께 Missing Authentication Token 응답을 반환합니다. API가 상위 리소스(ANY)에 GET 또는 / 메서드를 공개할 경우 이 호출은 백엔드에서 반환되는 404 Not Found 메시지와 함께 Cannot GET /petstore 응답을 반환합니다.

클라이언트 요청에 대해 타겟 엔드포인트 URL이 잘못되거나 HTTP 동사가 유효하지만 지원되지 않는 경우 백엔드에서 404 Not Found 응답을 반환합니다. 지원되지 않는 HTTP 메서드의 경우 403 Forbidden 응답이 반환됩니다.

API Gateway에서 HTTP 사용자 지정 통합 설정

HTTP 사용자 지정 통합에서는 API 메서드와 API 통합 간에 전달할 데이터와 그 데이터를 전달하는 방식을 제어할 수 있는 권한이 더 많습니다. 이러한 제어는 데이터 매핑을 통해 할 수 있습니다.

메서드 요청 설정의 일부로 메서드 리소스에 대한 requestParameters속성을 설정입니다. 이를 통해 클라이언트로부터 프로비저닝되는 메서드 요청 파라미터 중 어떤 것이 백엔드로 디스패치되기 전에 통합 요청 파라미터 또는 적용 가능한 본문 속성에 매핑되도록 할지 선언합니다. 그런 다음 통합 요청 설정의 일부로 해당 통합 리소스의 requestParameters 속성을 설정하여 파라미터 대 파라미터 매핑을 지정합니다. 또한 requestTemplates 속성을 설정하여 지원되는 각 콘텐츠 유형에 매핑 템플릿을 지정합니다. 매핑 템플릿은 메서드 요청 파라미터 또는 본문을 통합 요청 본문으로 매핑합니다.

마찬가지로 메서드 응답 설정의 일부로 MethodResponse 리소스에 대한 responseParameters 속성을 설정합니다. 이를 통해 클라이언트로 디스패치될 메서드 응답 파라미터 중 어떤 것이 백엔드로부터 반환된 통합 응답 파라미터 또는 적용 가능한 특정 본문 속성으로부터 매핑되도록 할지 선언합니다. 그런 다음 통합 응답 설정의 일부로 해당 IntegrationResponse 리소스에 대한 responseParameters 속성을 설정하여 파라미터 대 파라미터 매핑을 지정합니다. 또한 requestTemplates 맵을 설정하여 지원되는 각 콘텐츠 유형에 매핑 템플릿을 지정합니다. 매핑 템플릿은 통합 응답 파라미터 또는 통합 응답 본문 속성을 메서드 응답 본문으로 매핑합니다.

매핑 템플릿 설정에 대한 자세한 내용은 REST API에 대한 데이터 변환 설정 단원을 참조하세요.

API Gateway 프라이빗 통합 설정

API Gateway 프라이빗 통합으로 VPC 외부 클라이언트의 액세스를 위해 Amazon VPC 내의 HTTP/HTTPS 리소스를 간단히 공개할 수 있습니다. 프라이빗 VPC 리소스에 대한 액세스를 VPC 경계 너머

까지 확장하려면 프라이빗 통합이 포함된 API를 생성할 수 있습니다. API Gateway가 지원하는 권한 부여 방법을 사용하여 API에 대한 액세스를 제어할 수 있습니다.

프라이빗 통합을 생성하려면 먼저 Network Load Balancer를 생성해야 합니다. Network Load Balancer에는 요청을 VPC의 리소스로 라우팅하는 리스너가 있어야 합니다. API의 가용성을 개선하려면 Network Load Balancer에서 AWS 리전의 한 개 이상의 가용 영역에 있는 리소스로 트래픽을 라우팅해야 합니다. 그런 다음 API와 Network Load Balancer를 연결하는 데 사용하는 VPC 링크를 생성합니다. VPC 링크를 생성한 후 프라이빗 통합을 생성하여 VPC 링크 및 Network Load Balancer를 통해 API에서 VPC의 리소스로 트래픽을 라우팅합니다.

> **Note**
>
> Network Load Balancer와 API는 동일한 AWS 계정에서 소유해야 합니다.

API Gateway 프라이빗 통합을 사용하면 프라이빗 네트워크 구성이나 기술별 어플라이언스에 대한 상세한 지식이 없어도 VPC 내의 HTTP/HTTPS 리소스에 대한 액세스를 활성화할 수 있습니다.

주제

- API Gateway 프라이빗 통합을 위한 Network Load Balancer 설정
- VPC 링크를 생성할 수 있는 권한 부여
- API Gateway 콘솔을 사용하여 프라이빗 통합이 포함된 API Gateway API 설정
- AWS CLI를 사용하여 프라이빗 통합이 포함된 API Gateway API 설정
- OpenAPI를 사용하여 프라이빗 통합이 포함된 API 설정
- 프라이빗 통합에 사용되는 API Gateway 계정

API Gateway 프라이빗 통합을 위한 Network Load Balancer 설정

다음 절차에서는 Amazon EC2 콘솔을 사용하여 API Gateway 프라이빗 통합을 위해 Network Load Balancer(NLB)를 설정하는 단계를 설명하고, 각 단계의 상세한 지침을 위한 참조를 제공합니다.

리소스를 보유한 각 VPC에 대해 하나의 NLB와 하나의 VPCLink만 구성하면 됩니다. NLB는 NLB당 여러 리스너와 대상 그룹을 지원합니다. 각 서비스를 NLB에서 특정 리스너로 구성하고 단일 VPCLink를 사용해 NLB에 연결할 수 있습니다. API Gateway에서 프라이빗 통합을 생성하는 경우, 각 서비스에 할당된 특정 포트를 사용하여 각 서비스를 정의합니다. 자세한 내용은 the section called “자습서: 프라이빗 통합을 사용하여 API 빌드” 섹션을 참조하세요.

> **Note**
>
> Network Load Balancer와 API는 동일한 AWS 계정에서 소유해야 합니다.

API Gateway 콘솔을 사용하여 프라이빗 통합을 위한 Network Load Balancer를 생성하려면

1. AWS Management Console에 로그인하고 https://console.aws.amazon.com/ec2/에서 Amazon EC2 콘솔을 엽니다.
2. Amazon EC2 인스턴스에서 웹 서버를 설정합니다. 설정 예제는 Amazon Linux 2에 LAMP 웹 서버 설치 섹션을 참조하세요.
3. Network Load Balancer를 생성하고 대상 그룹에 EC2 인스턴스를 등록한 다음 대상 그룹을 Network Load Balancer의 리스너에 추가합니다. 자세한 내용은 Network Load Balancer 시작하기의 지침을 따르세요.
4. Network Load Balancer가 생성된 후 다음 작업을 수행합니다.

   a. Network Load Balancer의 ARN을 기록해 둡니다. API를 Network Load Balancer 뒤의 VPC 리소스와 통합하기 위해 API Gateway에서 VPC 링크를 생성하려면 이 정보가 필요합니다.

   b. PrivateLink에 대한 보안 그룹 평가를 해제합니다. 다음 명령어를 사용하여 PrivateLink 트래픽에 대한 인바운드 규칙을 해제합니다.

```
aws elbv2 set-security-groups --load-balancer-
arn arn:aws:elasticloadbalancing:us-east-2:111122223333:loadbalancer/net/my-
loadbalancer/abc12345 \
  --security-groups sg-123345a --enforce-security-group-inbound-rules-on-
private-link-traffic off
```

> **Note**
>
> API Gateway CIDR은 예고 없이 변경될 수 있으므로 종속성을 추가하지 마세요.

VPC 링크를 생성할 수 있는 권한 부여

본인 또는 본인 계정의 사용자가 VPC 링크를 생성 및 유지 관리하려면 VPC 엔드포인트 서비스 구성을 생성, 삭제, 확인하고, VPC 엔드포인트 서비스 권한을 변경하고, 로드 밸런서를 검사할 수 있는 권한이 있어야 합니다. 이러한 권한을 부여하려면 다음 단계를 사용하세요.

VPC 링크를 생성, 업데이트 및 삭제할 수 있는 권한을 부여하려면

1. 다음과 비슷한 IAM 정책을 생성합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "apigateway:POST",
                "apigateway:GET",
                "apigateway:PATCH",
                "apigateway:DELETE"
            ],
            "Resource": [
                "arn:aws:apigateway:us-east-1::/vpclinks",
                "arn:aws:apigateway:us-east-1::/vpclinks/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "elasticloadbalancing:DescribeLoadBalancers"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:CreateVpcEndpointServiceConfiguration",
                "ec2:DeleteVpcEndpointServiceConfigurations",
                "ec2:DescribeVpcEndpointServiceConfigurations",
                "ec2:ModifyVpcEndpointServicePermissions"
            ],
            "Resource": "*"
        }
    ]
}
```

2. IAM 역할을 만들거나 선택하여 앞의 정책을 이 역할에 연결합니다.
3. IAM 역할을 본인 또는 VPC 링크를 생성하는 본인 계정의 사용자에게 할당합니다.

API Gateway 콘솔을 사용하여 프라이빗 통합이 포함된 API Gateway API 설정

API Gateway 콘솔을 사용하여 프라이빗 통합이 포함된 API를 설정하는 지침은 자습서: API Gateway 프라이빗 통합으로 REST API 빌드 단원을 참조하세요.

AWS CLI를 사용하여 프라이빗 통합이 포함된 API Gateway API 설정

프라이빗 통합이 포함된 API를 생성하려면 먼저 VPC 리소스를 설정하고 Network Load Balancer를 생성하고 VPC 소스를 대상으로 구성해야 합니다. 요구 사항이 충족되지 않으면 API Gateway 프라이빗 통합을 위한 Network Load Balancer 설정에 따라 VPC 리소스를 설치하고, NLB를 생성하고, VPC 리소스를 Network Load Balancer의 대상으로 설정합니다.

> **Note**
>
> Network Load Balancer와 API는 동일한 AWS 계정에서 소유해야 합니다.

`VpcLink`를 생성하고 관리하려면 적절한 권한을 구성해야 합니다. 자세한 내용은 VPC 링크를 생성할 수 있는 권한 부여 섹션을 참조하세요.

> **Note**
>
> API에 `VpcLink`를 만들 수 있는 권한만 있으면 됩니다. `VpcLink`를 사용하기 위한 권한은 필요하지 않습니다.

Network Load Balancer가 생성된 후 ARN을 적어 둡니다. 이것은 프라이빗 통합을 위한 VPC 링크를 생성하는 데 필요합니다.

AWS CLI를 사용하여 프라이빗 통합이 포함된 API를 생성하려면

1. 지정된 Network Load Balancer를 대상 지정하는 `VpcLink`를 생성합니다.

```
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
```

   이 명령의 출력은 요청 수신을 확인하고 생성 중인 `VpcLink`의 `PENDING` 상태를 보여줍니다.

```
{
    "status": "PENDING",
    "targetArns": [
        "arn:aws:elasticloadbalancing:us-east-2:123456789012:loadbalancer/net/my-
vpclink-test-nlb/1234567890abcdef"
    ],
    "id": "gim7c3",
    "name": "my-test-vpc-link"
}
```

API Gateway가 VpcLink 생성을 완료하는 데 2~4분이 걸립니다. 작업이 성공적으로 완료되면 status는 AVAILABLE이 됩니다. 다음 CLI 명령을 호출하여 이를 확인할 수 있습니다.

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

작업이 실패하면 오류 메시지가 포함된 FAILED와 함께 statusMessage 상태를 얻게 됩니다. 예를 들어 이미 VPC 종단점에 연결된 Network Load Balancer를 사용하여 VpcLink를 생성하려는 경우, statusMessage 속성에서 다음을 얻게 됩니다.

```
"NLB is already associated with another VPC Endpoint Service"
```

VpcLink가 성공적으로 생성되어야 API를 생성하고 VpcLink를 통해 VPC 리소스와 통합할 수 있습니다.

새로 생성된 id의 VpcLink 값을 적어 둡니다(앞의 출력에서는 *gim7c3*). 프라이빗 통합을 설정하려면 이 값이 필요합니다.

2. API Gateway RestApi 리소스를 생성하여 API를 설정합니다.

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

반환된 결과에서 RestApi의 id 값을 적어 둡니다. 이 작업은 API에 대한 추가 작업을 실행하는 데 필요합니다.

예시를 위해 루트 리소스(GET)에 / 메서드만 있는 API를 생성해 이 메서드를 VpcLink와 통합하겠습니다.

3. GET / 메서드를 설정합니다. 먼저 루트 리소스(/) 식별자를 가져옵니다.

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

출력에서 id 경로의 / 값을 적어 둡니다. 이 예에서는 이 값이 *skpp60rab7*라고 가정합니다.

API 메서드 GET /에 대한 메서드 요청을 설정합니다.

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

VpcLink에 프록시 통합을 사용하지 않는 경우, 상태 코드가 200 이상인 메서드 응답도 설정해야 합니다. 여기서는 프록시 통합을 사용합니다.

4. HTTP_PROXY 유형의 프라이빗 통합을 설정하고 다음과 같이 put-integration 명령을 호출합니다.

```
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
```

프라이빗 통합의 경우, connection-type을 VPC_LINK로 설정하고, connection-id를 VpcLink의 식별자 또는 VpcLink ID를 참조하는 단계 변수로 설정해야 합니다. uri 파라미터는 요청을 엔드포인트로 라우팅하는 데 사용되지 않지만 Host 헤더 설정 및 인증서 유효성 검사에 사용됩니다.

명령은 다음 출력을 반환합니다.

```
{
    "passthroughBehavior": "WHEN_NO_MATCH",
    "timeoutInMillis": 29000,
    "connectionId": "gim7c3",
    "uri": "http://my-vpclink-test-nlb-1234567890abcdef.us-east-2.amazonaws.com",
```

```
    "connectionType": "VPC_LINK",
    "httpMethod": "GET",
    "cacheNamespace": "skpp60rab7",
    "type": "HTTP_PROXY",
    "cacheKeyParameters": []
}
```

통합을 생성할 때 단계 변수를 사용하여 connectionId 속성을 설정합니다.

```
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}"
```

단계 변수 표현식(${stageVariables.vpcLinkId})을 큰따옴표로 묶고 $ 문자를 이스케이프합니다.

또는 통합을 업데이트하여 단계 변수로 connectionId 값을 재설정할 수 있습니다.

```
 aws apigateway update-integration \
    --rest-api-id abcdef123 \
    --resource-id skpp60rab7 \
    --http-method GET \
    --patch-operations '[{"op":"replace","path":"/
connectionId","value":"${stageVariables.vpcLinkId}"}]'
```

문자열화된 JSON 목록을 patch-operations 파라미터 값으로 사용해야 합니다.

VpcLink의 단계 변수 값을 재설정하여 단계 변수를 통해 API를 다른 VPC 또는 Network Load Balancer와 통합할 수 있습니다.

프라이빗 프록시 통합을 사용했기 때문에 이제 API를 배포 및 테스트 실행할 준비가 됐습니다. 비프록시 통합의 경우, HTTP 사용자 지정 통합이 포함된 API를 설정할 때처럼 메서드 응답과 통합 응답도 설정해야 합니다.

5. API를 테스트하려면 API를 배포하세요. 이것은 VpcLink ID의 자리 표시자로 단계 변수를 사용한 경우에 필요합니다. 단계 변수가 포함된 API를 배포하려면 다음과 같이 create-deployment 명령을 호출하세요.

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

다른 VpcLink ID(예: *asf9d7*)로 단계 변수를 업데이트하라면 update-stage 명령을 호출합니다.

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

다음 명령을 사용하여 API를 호출합니다.

```
curl -X GET https://abcdef123.execute-api.us-east-2.amazonaws.com/test
```

또는 API의 호출 URL을 웹 브라우저에서 입력하여 결과를 볼 수 있습니다.

connection-id ID 리터럴로 VpcLink 속성을 하드코드하면 test-invoke-method를 호출하여 API가 배포되기 전에 API 호출을 테스트할 수도 있습니다.

OpenAPI를 사용하여 프라이빗 통합이 포함된 API 설정

API의 OpenAPI 파일을 가져와 프라이빗 통합이 포함된 API를 설정할 수 있습니다. 다음만 제외하면 설정은 HTTP 통합이 포함된 API의 OpenAPI 정의와 비슷합니다.

- 명시적으로 connectionType을 VPC_LINK로 설정해야 합니다.
- 명시적으로 connectionId를 VpcLink의 ID 또는 VpcLink의 ID를 참조하는 단계 변수로 설정해야 합니다.
- 프라이빗 통합의 uri 파라미터는 VPC의 HTTP/HTTPS 엔드포인트를 가리키지만 그 대신 통합 요청의 Host 헤더를 설정하는 데 사용됩니다.
- VPC에 HTTPS 엔드포인트가 있는 프라이빗 통합의 uri 파라미터는 VPC 엔드포인트에 설치된 인증서에 있는 이름과 대조하여 지정된 도메인 이름을 확인하는 데 사용됩니다.

단계 변수를 사용하여 VpcLink ID를 참조할 수 있습니다. 또는 ID 값을 직접 connectionId에 할당할 수 있습니다.

다음 JSON 형식의 OpenAPI 파일은 단계 변수(${stageVariables.vpcLinkId})가 참조하는 VPC 링크가 포함된 API의 예를 보여 줍니다.

OpenAPI 2.0

```
{
  "swagger": "2.0",
  "info": {
    "version": "2017-11-17T04:40:23Z",
    "title": "MyApiWithVpcLink"
  },
  "host": "p3wocvip9a.execute-api.us-west-2.amazonaws.com",
  "basePath": "/test",
  "schemes": [
    "https"
  ],
  "paths": {
    "/": {
      "get": {
        "produces": [
          "application/json"
        ],
        "responses": {
          "200": {
            "description": "200 response",
            "schema": {
              "$ref": "#/definitions/Empty"
            }
          }
        },
        "x-amazon-apigateway-integration": {
          "responses": {
            "default": {
              "statusCode": "200"
            }
          },
          "uri": "http://my-vpclink-test-nlb-1234567890abcdef.us-
east-2.amazonaws.com",
          "passthroughBehavior": "when_no_match",
          "connectionType": "VPC_LINK",
```

```
          "connectionId": "${stageVariables.vpcLinkId}",
          "httpMethod": "GET",
          "type": "http_proxy"
        }
      }
    }
  },
  "definitions": {
    "Empty": {
      "type": "object",
      "title": "Empty Schema"
    }
  }
}
```

프라이빗 통합에 사용되는 API Gateway 계정

다음 리전별 API Gateway 계정 ID는 `VpcLink`을 생성할 때 `AllowedPrincipals`로 VPC 종단점 서비스에 자동으로 추가됩니다 .

Region	계정 ID
us-east-1	392220576650
us-east-2	718770453195
us-west-1	968246515281
us-west-2	109351309407
ca-central-1	796887884028
eu-west-1	631144002099
eu-west-2	544388816663
eu-west-3	061510835048
eu-central-1	474240146802
eu-central-2	166639821150

Region	계정 ID
eu-north-1	394634713161
eu-south-1	753362059629
eu-south-2	359345898052
ap-northeast-1	969236854626
ap-northeast-2	020402002396
ap-northeast-3	360671645888
ap-southeast-1	195145609632
ap-southeast-2	798376113853
ap-southeast-3	652364314486
ap-southeast-4	849137399833
ap-south-1	507069717855
ap-south-2	644042651268
ap-east-1	174803364771
sa-east-1	287228555773
me-south-1	855739686837
me-central-1	614065512851

API Gateway에서 모의 통합 설정

Amazon API Gateway는 API 메서드에 대한 모의 통합을 지원합니다. 이 기능을 통해 API 개발자는 통합 백엔드가 없어도 API Gateway에서 직접 API 응답을 생성할 수 있습니다. API 개발자라면 이 기능을 사용하여 프로젝트 개발이 완료되기 전에 API로 작업해야 하는 종속 팀에 대한 차단을 해제할 수 있습니다. 또한 이 기능을 활용하여 API의 개요와 탐색 기능을 제공하는 API에 대해 시작 페이지를 프로비

저닝할 수 있습니다. 이러한 시작 페이지의 예는 자습서: 예제를 가져와 REST API 생성에서 설명한 예제 API의 루트 리소스에 대한 GET 메서드의 통합 요청 및 응답을 참조하세요.

API 개발자는 API Gateway가 모의 통합 요청에 응답하는 방법을 결정합니다. 이를 위해, 메서드의 통합 요청 및 통합 응답을 구성하여 정해진 상태 코드와 응답을 연결합니다. 모의 통합이 포함된 메서드가 200 응답을 반환하려면 다음을 반환하도록 통합 요청 본문 매핑 템플릿을 구성해야 합니다.

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

예를 들어 다음의 본문 매핑 템플릿이 포함되도록 200 통합 응답을 구성하십시오.

```
{
    "statusCode": 200,
    "message": "Go ahead without me."
}
```

마찬가지로 메서드가 예를 들어 500 오류 응답을 반환하려면 다음을 반환하도록 통합 요청 본문 매핑 템플릿을 설정해야 합니다.

```
{"statusCode": 500}
```

예를 들어 다음의 매핑 템플릿을 사용하여 500 통합 응답을 설정하십시오.

```
{
    "statusCode": 500,
    "message": "The invoked method is not supported on the API resource."
}
```

또는 통합 요청 매핑 템플릿을 정의하지 않고 모의 통합의 메서드가 기본 통합 응답을 반환하도록 할 수도 있습니다. 기본 통합 응답은 정의되지 않은 HTTP status regex(HTTP 상태 regex)가 포함된 응답입니다. 적절한 패스스루 동작이 설정되어야 합니다.

> **Note**
>
> 모의 통합은 대규모 응답 템플릿을 지원하지 않습니다. 사용 사례에 필요한 경우 대신 Lambda 통합 사용을 고려해야 합니다.

통합 요청 매핑 템플릿을 사용하여 애플리케이션 로직을 삽입하면 특정 조건에 따라 반환할 모의 통합 응답을 결정할 수 있습니다. 예를 들어 수신되는 요청에서 scope 쿼리 파라미터를 사용하여 성공 응답을 반환할지 오류 응답을 반환할지 결정할 수 있습니다.

```
{
  #if( $input.params('scope') == "internal" )
    "statusCode": 200
  #else
    "statusCode": 500
  #end
}
```

이런 방법으로 모의 통합의 메서드는 오류 응답을 포함한 다른 유형의 호출은 거부하면서 내부 호출을 통과시킵니다.

이 단원에서는 API Gateway 콘솔을 사용하여 API 메서드에 대한 모의 통합을 활성화하는 방법에 대해 설명합니다.

주제

- API Gateway 콘솔을 사용하여 모의 통합 활성화

API Gateway 콘솔을 사용하여 모의 통합 활성화

API Gateway에 사용 가능한 메서드가 있어야 합니다. 자습서: HTTP 비 프록시 통합을 사용하여 REST API 빌드의 지침을 따르세요.

1. API 리소스를 선택하고 메서드 생성을 선택합니다.

   메서드를 생성하려면 다음을 수행합니다.

   a. 메서드 유형에서 메서드를 선택합니다.
   b. 통합 유형에서 Mock을 선택합니다.
   c. 메서드 생성을 선택합니다.
   d. 메서드 요청 탭의 메서드 요청 설정에서 편집을 선택합니다.
   e. URL 쿼리 문자열 파라미터를 선택합니다. 쿼리 문자열 추가를 선택하고 이름에 **scope**를 입력합니다. 쿼리 파라미터에 따라 호출자가 내부 호출자인지 아닌지 결정됩니다.
   f. 저장을 선택합니다.
2. 메서드 응답 탭에서 응답 생성을 선택하고 다음을 수행합니다.

a. HTTP 상태에 **500**을 입력합니다.

b. 저장을 선택합니다.

3. 통합 요청 탭의 통합 요청 설정에서 편집을 선택합니다.

4. 매핑 템플릿을 선택한 후 다음을 수행합니다.

a. 매핑 템플릿 추가(Add mapping template)를 선택합니다.

b. 콘텐츠 유형에 **application/json**을 입력합니다.

c. 템플릿 본문에 다음을 입력합니다.

```
{
  #if( $input.params('scope') == "internal" )
    "statusCode": 200
  #else
    "statusCode": 500
  #end
}
```

d. 저장을 선택합니다.

5. 통합 응답 탭의 기본값 - 응답에서 편집을 선택합니다.

6. 매핑 템플릿을 선택한 후 다음을 수행합니다.

a. 콘텐츠 유형에 **application/json**을 입력합니다.

b. 템플릿 본문에 다음을 입력합니다.

```
{
    "statusCode": 200,
    "message": "Go ahead without me"
}
```

c. 저장을 선택합니다.

7. 응답 생성을 선택합니다.

500 응답을 생성하려면 다음을 수행합니다.

a. HTTP 상태 regex에 **5\d{2}**를 입력합니다.

b. 메서드 응답 상태에서 **500**을 선택합니다.

c. 저장을 선택합니다.

d. 5\d{2} - 응답에서 편집을 선택합니다.

e. 매핑 템플릿을 선택한 다음 매핑 템플릿 추가를 선택합니다.

f. 콘텐츠 유형에 **application/json**을 입력합니다.

g. 템플릿 본문에 다음을 입력합니다.

```
{
    "statusCode": 500,
    "message": "The invoked method is not supported on the API resource."
}
```

h. 저장을 선택합니다.

8. 테스트(Test) 탭을 선택합니다. 탭을 표시하려면 오른쪽 화살표 버튼을 선택해야 할 수도 있습니다. Mock 통합을 테스트하려면 다음을 수행합니다.

a. 쿼리 문자열에 `scope=internal`을 입력합니다. 테스트를 선택합니다. 테스트 결과에 다음이 표시됩니다.

```
Request: /?scope=internal
Status: 200
Latency: 26 ms
Response Body

{
  "statusCode": 200,
  "message": "Go ahead without me"
}

Response Headers

{"Content-Type":"application/json"}
```

b. `Query strings`에 `scope=public`을 입력하거나 비워 둡니다. 테스트를 선택합니다. 테스트 결과에 다음이 표시됩니다.

```
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"}
```

먼저 메서드 응답에 헤더를 추가한 다음 통합 응답에서 헤더 매핑을 설정하면 모의 통합 응답에서 헤더를 반환할 수도 있습니다. 이는 사실 API Gateway 콘솔이 CORS 필수 헤더를 반환하여 CORS 지원을 활성화하는 방법입니다.

API Gateway에서 요청 확인 사용

통합 요청을 진행하기 전에 API 요청의 기본 확인을 수행하도록 API Gateway를 구성할 수 있습니다. 검증이 실패하면 API Gateway는 즉시 요청을 실패시키고 호출자에게 400 오류 응답을 반환하고 검증 결과를 Logs에 CloudWatch 게시합니다. 이렇게 하면 불필요한 백엔드 호출이 줄어듭니다. 무엇보다, 해당 애플리케이션 고유의 확인 작업에 집중할 수 있게 됩니다. 필요한 요청 파라미터가 유효하며 Null이 아님을 확인하거나 더 복잡한 데이터 확인에 대한 모델 스키마를 지정하여 요청 본문을 확인할 수 있습니다.

주제

- API Gateway의 기본 요청 확인 개요
- 데이터 모델 이해
- API Gateway의 기본 요청 확인 설정
- 기본 요청 확인이 설정된 샘플 API의 OpenAPI 정의
- AWS CloudFormation 기본 요청 검증이 포함된 샘플 API의 템플릿

API Gateway의 기본 요청 확인 개요

API Gateway는 기본 요청 확인을 수행할 수 있으므로 사용자가 백엔드에서 앱별 확인에 집중할 수 있습니다. 확인에서 API Gateway는 다음 조건 중 하나 또는 둘 다를 확인합니다.

- URI의 필수 요청 파라미터, 쿼리 문자열, 수신 요청의 헤더가 포함되어 있고 비어 있지 않습니다.

- 요청 페이로드는 해당 메서드에 구성된 JSON 스키마 요청을 준수합니다.

확인을 활성화하려면 요청 검사기에 확인 규칙을 지정하고, API의 요청 검사기 맵에 검사기를 추가하고, 개별 API 메서드에 검사기를 할당합니다.

> **Note**
>
> 요청 본문 확인과 통합 패스스루 동작은 별도의 주제입니다. 요청 페이로드에 일치하는 모델 스키마가 없으면 패스스루를 선택하거나 원래 페이로드를 차단할 수 있습니다. 자세한 설명은 통합 패스스루 동작 섹션을 참조하세요.

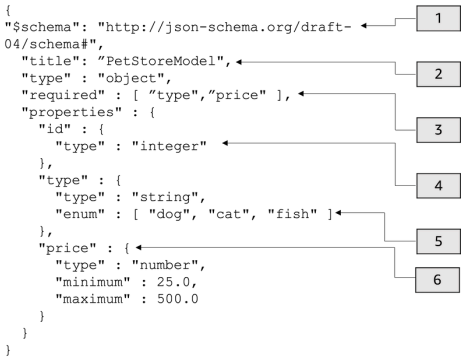
데이터 모델 이해

API Gateway에서 모델은 페이로드의 데이터 구조를 정의합니다. API Gateway에서 모델은 JSON 스키마 드래프트 4를 사용하여 정의됩니다. 다음 JSON 객체는 Pet Store 예시의 샘플 데이터입니다.

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

데이터에는 반려동물의 `id`, `type` 및 `price`가 포함되어 있습니다. 이 데이터의 모델을 통해 다음을 수행할 수 있습니다.

- 기본 요청 확인을 사용합니다.
- 데이터 변환을 위한 매핑 템플릿을 생성합니다.
- SDK를 생성할 때 사용자 정의 데이터 유형(UDT)을 생성합니다.



이 모델의 경우:

1. $schema 객체는 유효한 JSON 스키마 버전 식별자를 나타냅니다. 이 스키마는 JSON 스키마 드래프트 v4입니다.
2. title 객체는 사람이 읽을 수 있는 모델 식별자입니다. 이 제목은 PetStoreModel입니다.
3. 기본 요청 확인에서는 required 확인 키워드에 type 및 price가 필요합니다.
4. 모델의 properties은 id, type, price입니다. 각 객체에는 모델에 설명된 속성이 있습니다.
5. type 객체에는 dog, cat, 또는 fish 값만 있을 수 있습니다.
6. price 객체는 숫자이며 minimum 25, maximum 500으로 제한됩니다.

PetStore 모델

```
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 }
```

이 모델의 경우:

1. 2번째 줄에서 $schema 객체는 유효한 JSON 스키마 버전 식별자를 나타냅니다. 이 스키마는 JSON 스키마 드래프트 v4입니다.
2. 3번째 줄에서 title 객체는 사람이 읽을 수 있는 모델 식별자입니다. 이 제목은 PetStoreModel입니다.
3. 5번째 줄에서 기본 요청 확인에서는 required 확인 키워드에 type 및 price가 필요합니다.
4. 6~17번째 줄에서 모델의 properties은 id, type, price입니다. 각 객체에는 모델에 설명된 속성이 있습니다.
5. 12번째 줄에서 type 객체에는 dog, cat 또는 fish 값만 있을 수 있습니다.
6. 14~17번째 줄에서 price 객체는 숫자이며 minimum 25, maximum 500으로 제한됩니다.

더 복잡한 모델 생성

$ref 프리미티브를 사용하여 더 긴 모델에 대해 재사용 가능한 정의를 만들 수 있습니다. 예를 들어, price 객체를 설명하는 definitions 섹션에서 Price라는 정의를 생성할 수 있습니다. $ref의 값은 Price 정의입니다.

```
{
  "$schema" : "http://json-schema.org/draft-04/schema#",
  "title" : "PetStoreModelReUsableRef",
  "required" : ["price", "type" ],
  "type" : "object",
  "properties" : {
    "id" : {
      "type" : "integer"
    },
    "type" : {
      "type" : "string",
      "enum" : [ "dog", "cat", "fish" ]
    },
    "price" : {
        "$ref": "#/definitions/Price"
    }
  },
  "definitions" : {
```

```
      "Price": {
        "type" : "number",
        "minimum" : 25.0,
        "maximum" : 500.0
            }
      }
}
```

외부 모델 파일에 정의된 다른 모델 스키마를 참조할 수도 있습니다. `$ref` 속성 값을 모델의 위치로 설정합니다. 다음 예시에서는 `Price` 모델이 `a1234` API의 `PetStorePrice` 모델에 정의되어 있습니다.

```
{
  "$schema" : "http://json-schema.org/draft-04/schema#",
  "title" : "PetStorePrice",
  "type": "number",
  "minimum": 25,
  "maximum": 500
}
```

더 긴 모델은 `PetStorePrice` 모델을 참조할 수 있습니다.

```
{
  "$schema" : "http://json-schema.org/draft-04/schema#",
  "title" : "PetStoreModelReusableRefAPI",
  "required" : [ "price", "type" ],
  "type" : "object",
  "properties" : {
    "id" : {
      "type" : "integer"
    },
    "type" : {
      "type" : "string",
      "enum" : [ "dog", "cat", "fish" ]
    },
    "price" : {
        "$ref": "https://apigateway.amazonaws.com/restapis/a1234/models/PetStorePrice"
    }
  }
}
```

출력 데이터 모델 사용

데이터를 변환하면 통합 응답에서 페이로드 모델을 정의할 수 있습니다. 페이로드 모델은 SDK를 생성할 때 사용할 수 있습니다. Java, Objective-C 또는 Swift 등 강력한 형식의 언어에서 객체는 사용자 정의 데이터 형식(UDT)에 해당합니다. SDK를 생성할 때 데이터 모델과 함께 제공하면 API Gateway가 이러한 UDT를 생성합니다. 데이터 변환에 대한 자세한 내용은 매핑 템플릿 이해 섹션을 참조하세요.

출력 데이터

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

출력 모델

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

이 모델에서
는 `PetStoreOutputModel[i].description` 및 `PetStoreOutputModel[i].askingPrice` 속성을 읽음으로써 `description` 및 `askingPrice` 속성 값을 검색하도록 SDK를 호출할 수 있습니다. 모델이 제공되지 않는 경우, API Gateway는 빈 모델을 사용하여 기본 UDT를 생성합니다.

다음 단계

- 이 섹션에서는 이 주제에 제시된 개념에 대해 자세히 알아보는 데 사용할 수 있는 리소스를 제공합니다.

  요청 확인 자습서를 따라 수행할 수 있습니다.
  - API Gateway 콘솔을 사용하여 요청 확인 설정
  - 를 사용하여 기본 요청 검증을 설정합니다. AWS CLI
  - OpenAPI 정의를 사용하여 기본 요청 확인 설정
- 데이터 변환 및 매핑 템플릿(매핑 템플릿 이해)에 대한 자세한 정보를 얻을 수 있습니다.
- 더 높은 수준의 사진 앨범 예시 모델도 볼 수 있습니다. 예제 사진 섹션을 참조하십시오.

API Gateway의 기본 요청 확인 설정

이 섹션에서는 콘솔과 OpenAPI 정의를 사용하여 API Gateway에 대한 요청 검증을 설정하는 방법을 보여줍니다. AWS CLI

주제

- API Gateway 콘솔을 사용하여 요청 확인 설정
- 를 사용하여 기본 요청 검증을 설정합니다. AWS CLI
- OpenAPI 정의를 사용하여 기본 요청 확인 설정

API Gateway 콘솔을 사용하여 요청 확인 설정

API Gateway 콘솔을 사용하여 API 요청에 대한 세 가지 검사기 중 하나를 선택하여 요청을 확인할 수 있습니다.

- 본문을 확인합니다.
- 쿼리 문자열 파라미터 및 헤더를 확인합니다.
- 본문, 쿼리 문자열 파라미터 및 헤더를 확인합니다.

API 메서드에 검증기 중 하나를 적용하면 API Gateway 콘솔이 API 맵에 검증기를 추가합니다. RequestValidators

이 자습서를 따라하려면 AWS CloudFormation 템플릿을 사용하여 불완전한 API Gateway API를 생성해 보겠습니다. 이 API에는 GET 및 POST 메서드가 포함된 /validator 리소스가 있습니다. 두 메서드 모두 http://petstore-demo-endpoint.execute-api.com/petstore/pets HTTP 엔드포인트와 통합됩니다. 두 가지 유형의 요청 확인을 구성합니다.

- GET 메서드에서 URL 쿼리 문자열 파라미터에 대한 요청 확인을 구성합니다.
- POST 메서드에서 요청 본문에 대한 요청 확인을 구성합니다.

이렇게 하면 특정 API 호출만 API로 전달될 수 있습니다.

의 앱 생성 템플릿을 다운로드하고 압축을 풉니다. AWS CloudFormation이 템플릿을 사용하여 불완전한 API를 만들 수 있습니다. API Gateway 콘솔에서 나머지 단계를 완료하게 됩니다.

스택을 AWS CloudFormation 만들려면

1. https://console.aws.amazon.com/cloudformation 에서 AWS CloudFormation 콘솔을 엽니다.
2. 스택 생성을 선택한 다음 새 리소스 사용(표준)을 선택합니다.
3. 템플릿 지정에서 템플릿 파일 업로드를 선택합니다.
4. 다운로드한 템플릿을 선택합니다.
5. 다음을 선택합니다.
6. 스택 이름에 **request-validation-tutorial-console**을(를) 입력하고 다음을 선택합니다.
7. 스택 옵션 구성에서 다음을 선택합니다.
8. [Capacabilities] 의 경우 계정에서 IAM 리소스를 생성할 AWS CloudFormation 수 있다는 점을 확인하십시오.
9. 스택 생성을 선택합니다.

AWS CloudFormation 템플릿에 지정된 리소스를 프로비저닝합니다. 리소스 프로비저닝을 완료하는 데 몇 분 정도 걸릴 수 있습니다. AWS CloudFormation 스택 상태가 CREATE_COMPLETE이면 다음 단계로 넘어갈 준비가 된 것입니다.

새로 생성한 API를 선택하는 방법

1. 새로 생성한 **request-validation-tutorial-console** 스택을 선택합니다.
2. 리소스를 선택합니다.
3. 물리적 ID에서 API를 선택합니다. 이 링크는 API Gateway 콘솔로 연결됩니다.

GET 및 POST 메서드를 수정하기 전에 모델을 만들어야 합니다.

모델을 생성하는 방법

1. 수신 요청의 본문에서 요청 확인을 사용하려면 모델이 필요합니다. 모델을 생성하려면 기본 탐색 창에서 모델을 선택합니다.
2. Create model(모델 생성)을 선택합니다.
3. 이름에 **PetStoreModel**를 입력합니다.
4. 콘텐츠 유형에 **application/json**을 입력합니다. 일치하는 콘텐츠 유형이 없는 경우 요청 확인이 수행되지 않습니다. 콘텐츠 유형에 관계없이 동일한 모델을 사용하려면 **$default**를 입력합니다.
5. 설명에 모델 설명으로 **My PetStore Model**을 입력합니다.
6. 모델 스키마의 경우 다음 모델을 코드 편집기에 붙여 넣고 생성을 선택합니다.

```
{
  "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
    }
  }
}
```

모델에 대한 자세한 내용은 데이터 모델 이해 섹션을 참조하세요.

GET 메서드에 대한 요청 검사를 구성하려면

1. 기본 탐색 창에서 리소스를 선택하고 GET 메서드를 선택합니다.
2. 메서드 요청 탭의 메서드 요청 설정에서 편집을 선택합니다.
3. 요청 검사기에서 쿼리 문자열 파라미터 및 헤더 검사를 선택합니다.
4. URL 쿼리 문자열 파라미터에서 다음을 수행합니다.
   a. 쿼리 문자열 추가(Add query string)를 선택합니다.
   b. 이름에 **petType**를 입력합니다.
   c. 필수를 끕니다.
   d. 캐싱을 꺼진 상태로 둡니다.
5. 저장을 선택합니다.
6. 통합 요청 탭의 통합 요청 설정에서 편집을 선택합니다.
7. URL 쿼리 문자열 파라미터에서 다음을 수행합니다.
   a. 쿼리 문자열 추가(Add query string)를 선택합니다.
   b. 이름에 **petType**를 입력합니다.
   c. 다음에서 매핑됨에 **method.request.querystring.petType**을 입력합니다. 이렇게 하면 **petType**이 반려동물의 유형에 매핑됩니다.

      데이터 매핑에 대한 자세한 내용은 데이터 매핑 자습서를 참조하세요.
   d. 캐싱을 꺼진 상태로 둡니다.
8. 저장을 선택합니다.

GET 메서드에 대한 요청 검사를 테스트하려면

1. 테스트 탭을 선택합니다. 탭을 표시하려면 오른쪽 화살표 버튼을 선택해야 할 수도 있습니다.

2. 쿼리 문자열에 **petType=dog**를 입력하고 테스트를 선택합니다.
3. 메서드 테스트에서 200 OK와 dog 목록이 반환됩니다.

   이 출력 데이터를 변환하는 방법에 대한 자세한 내용은 데이터 매핑 자습서를 참조하세요.
4. **petType=dog**를 제거하고 테스트를 선택합니다.
5. 메서드 테스트에서 다음 오류 메시지와 함께 400 오류가 반환됩니다.

```
{
  "message": "Missing required request parameters: [petType]"
}
```

POST 메서드에 대한 요청 검사를 구성하려면

1. 기본 탐색 창에서 리소스를 선택하고 POST 메서드를 선택합니다.
2. 메서드 요청 탭의 메서드 요청 설정에서 편집을 선택합니다.
3. 요청 검사기에서 본문 검사를 선택합니다.
4. 요청 본문에서 모델 추가를 선택합니다.
5. 콘텐츠 유형에는 을 입력하고 **application/json** 모델에서는 을 선택합니다. PetStoreModel
6. 저장을 선택합니다.

POST 메서드에 대한 요청 검사를 테스트하려면

1. 테스트 탭을 선택합니다. 탭을 표시하려면 오른쪽 화살표 버튼을 선택해야 할 수도 있습니다.
2. 요청 본문에서 다음을 코드 편집기에 붙여 넣습니다.

```
{
  "id": 2,
  "name": "Bella",
  "type": "dog",
  "price": 400
}
```

   테스트를 선택합니다.
3. 메서드 테스트에서 200 OK와 함께 성공 메시지가 반환됩니다.
4. 요청 본문에서 다음을 코드 편집기에 붙여 넣습니다.

```
{
  "id": 2,
  "name": "Bella",
  "type": "dog",
  "price": 4000
}
```

테스트를 선택합니다.

5. 메서드 테스트에서 다음 오류 메시지와 함께 400 오류가 반환됩니다.

```
{
 "message": "Invalid request body"
}
```

테스트 로그 하단에 잘못된 요청 본문의 이유가 반환됩니다. 이 경우 반려동물의 가격이 모델에 명시된 최대 가격을 벗어났습니다.

AWS CloudFormation 스택을 삭제하려면

1. https://console.aws.amazon.com/cloudformation 에서 AWS CloudFormation 콘솔을 엽니다.
2. AWS CloudFormation 스택을 선택합니다.
3. 삭제를 선택한 다음 해당 선택을 확인합니다.

다음 단계

- 출력 데이터를 변환하고 더 많은 데이터 매핑을 수행하는 방법에 대한 자세한 내용은 데이터 매핑 자습서를 참조하세요.
- AWS CLI를 사용하여 유사한 단계를 수행하려면 AWS CLI를 사용하여 기본 요청 확인 설정 자습서를 따라 하세요.

를 사용하여 기본 요청 검증을 설정합니다. AWS CLI

AWS CLI를 사용하여 검사기를 만들어 요청 확인을 설정할 수 있습니다. 이 자습서를 따라하려면 AWS CloudFormation 템플릿을 사용하여 불완전한 API Gateway API를 생성해 보겠습니다.

> **Note**
>
> 이 AWS CloudFormation 템플릿은 콘솔 자습서와 동일하지 않습니다.

미리 노출된 `/validator` 리소스를 사용하여 `GET` 및 `POST` 메서드를 생성합니다. 두 메서드 모두 `http://petstore-demo-endpoint.execute-api.com/petstore/pets` HTTP 엔드포인트와 통합됩니다. 다음과 같은 두 가지 요청 검사를 구성합니다.

- `GET` 메서드에서 URL 쿼리 문자열 파라미터를 확인하는 `params-only` 검사기를 생성합니다.
- `POST` 메서드에서 요청 본문을 확인하는 `body-only` 검사기를 생성합니다.

이렇게 하면 특정 API 호출만 API로 전달될 수 있습니다.

AWS CloudFormation 스택을 만들려면

의 앱 생성 템플릿을 다운로드하고 압축을 풉니다. AWS CloudFormation

다음 자습서를 완료하려면 AWS Command Line Interface (AWS CLI) 버전 2가 필요합니다.

긴 명령의 경우 이스케이프 문자(\)를 사용하여 명령을 여러 행으로 분할합니다.

> **Note**
>
> Windows에서는 일반적으로 사용하는 일부 Bash CLI 명령(예: `zip`)은 운영 체제의 기본 제공 터미널에서 지원되지 않습니다. Ubuntu와 Bash의 Windows 통합 버전을 가져오려면 Linux용 Windows Subsystem을 설치합니다. 이 안내서의 예제 CLI 명령은 Linux 형식을 사용합니다. Windows CLI를 사용하는 경우 인라인 JSON 문서를 포함하는 명령의 형식을 다시 지정해야 합니다.

1. 다음 명령어를 사용하여 AWS CloudFormation 스택을 생성합니다.

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

2. AWS CloudFormation 템플릿에 지정된 리소스를 프로비저닝합니다. 리소스 프로비저닝을 완료하는 데 몇 분 정도 걸릴 수 있습니다. 다음 명령을 사용하여 AWS CloudFormation 스택 상태를 확인합니다.

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

3. AWS CloudFormation 스택 상태가 이면 `StackStatus: "CREATE_COMPLETE"` 다음 명령을 사용하여 향후 단계를 위한 관련 출력 값을 검색하십시오.

```
aws cloudformation describe-stacks --stack-name request-validation-tutorial-cli
--query "Stacks[*].Outputs[*].{OutputKey: OutputKey, OutputValue: OutputValue,
Description: Description}"
```

출력 값은 다음과 같습니다.

- ApiId, API의 ID입니다. 이 자습서에서 API ID는 `abc123`입니다.
- ResourceId, `GET` 및 `POST` 메서드가 노출되는 유효성 검사기 리소스의 ID입니다. 이 자습서에서 리소스 ID는 `efg456`입니다.

요청 검사기를 생성하고 모델을 가져오는 방법

1. AWS CLI로 요청 확인을 사용하려면 검사기가 필요합니다. 다음 명령을 사용하여 요청 파라미터만 확인하는 검사기를 만듭니다.

```
aws apigateway create-request-validator --rest-api-id abc123 \
      --no-validate-request-body \
      --validate-request-parameters \
      --name params-only
```

`params-only` 검사기의 ID를 기록해 둡니다.

2. 다음 명령을 사용하여 요청 본문만 확인하는 검사기를 만듭니다.

```
aws apigateway create-request-validator --rest-api-id abc123 \
      --validate-request-body \
      --no-validate-request-parameters \
      --name body-only
```

`body-only` 검사기의 ID를 기록해 둡니다.

3. 수신 요청의 본문에서 요청 확인을 사용하려면 모델이 필요합니다. 다음 명령을 사용하여 모델을 가져옵니다.

```
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}}}}'
```

일치하는 콘텐츠 유형이 없는 경우 요청 확인이 수행되지 않습니다. 콘텐츠 유형에 관계없이 동일한 모델을 사용하려면 `$default`을(를) 키로 지정합니다.

GET 및 **POST** 메서드를 생성하는 방법

1. 다음 명령을 사용하여 `/validate` 리소스에 GET HTTP 메서드를 추가합니다. 이 명령은 GET 메서드를 만들고 `params-only` 검사기를 추가하며 필요에 따라 `petType` 쿼리 문자열을 설정합니다.

```
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"
```

다음 명령을 사용하여 `/validate` 리소스에 POST HTTP 메서드를 추가합니다. 이 명령은 POST 메서드를 만들고 `body-only` 검사기를 추가하며 모델을 본문 전용 검사기에 연결합니다.

```
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. 다음 명령을 사용하여 `GET /validate` 메서드의 `200 OK` 응답을 설정합니다.

```
aws apigateway put-method-response --rest-api-id abc123  \
            --resource-id efg456 \
            --http-method GET \
            --status-code 200
```

다음 명령을 사용하여 POST /validate 메서드의 200 OK 응답을 설정합니다.

```
aws apigateway put-method-response --rest-api-id abc123  \
            --resource-id efg456 \
            --http-method POST \
            --status-code 200
```

3. 다음 명령을 사용하여 GET /validation 메서드에 지정된 HTTP 엔드포인트로 Integration을 설정합니다.

```
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'
```

다음 명령을 사용하여 POST /validation 메서드에 지정된 HTTP 엔드포인트로 Integration을 설정합니다.

```
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. 다음 명령을 사용하여 GET /validation 메서드 통합 응답을 설정합니다.

```
aws apigateway put-integration-response --rest-api-id abc123 \
              --resource-id efg456\
              --http-method GET \
              --status-code 200 \
              --selection-pattern ""
```

다음 명령을 사용하여 POST /validation 메서드 통합 응답을 설정합니다.

```
aws apigateway put-integration-response --rest-api-id abc123 \
```

```
            --resource-id efg456 \
            --http-method POST \
            --status-code 200 \
            --selection-pattern ""
```

API를 테스트하려면

1. 쿼리 문자열에 대한 요청 확인을 수행할 GET 메서드를 테스트하려면 다음 명령을 사용합니다.

```
aws apigateway test-invoke-method --rest-api-id abc123 \
            --resource-id efg456 \
            --http-method GET \
            --path-with-query-string '/validate?petType=dog'
```

   결과는 200 OK 및 반려견 목록을 반환합니다.

2. petType 쿼리 문자열을 포함하지 않고 테스트하려면 다음 명령을 사용합니다.

```
aws apigateway test-invoke-method --rest-api-id abc123 \
            --resource-id efg456 \
            --http-method GET
```

   결과는 400 오류를 반환합니다.

3. 요청 본문에 대한 요청 확인을 수행할 POST 메서드를 테스트하려면 다음 명령을 사용합니다.

```
 aws apigateway test-invoke-method --rest-api-id abc123 \
            --resource-id efg456 \
            --http-method POST \
            --body '{"id": 1, "name": "bella", "type": "dog", "price" : 400 }'
```

   결과는 200 OK 및 성공 메시지를 반환합니다.

4. 다음 명령을 사용하여 잘못된 본문을 사용하여 테스트합니다.

```
 aws apigateway test-invoke-method --rest-api-id abc123 \
              --resource-id efg456 \
              --http-method POST \
              --body '{"id": 1, "name": "bella", "type": "dog", "price" : 1000 }'
```

   반려견의 가격이 모델에서 정의한 최고 가격을 초과하므로 결과에 400 오류가 반환됩니다.

스택을 삭제하려면 AWS CloudFormation

- 다음 명령어를 사용하여 AWS CloudFormation 리소스를 삭제합니다.

```
aws cloudformation delete-stack  --stack-name request-validation-tutorial-cli
```

OpenAPI 정의를 사용하여 기본 요청 확인 설정

요청의 어느 부분을 확인할지 선택하기 위해 x-amazon-apigateway-request- 유효성 검사기. 요청 유효성 검사기 개체 객체 세트를 x-amazon-apigateway-request-유효성 검사기 개체 맵에 지정하여 API 수준에서 요청 검사기를 선언할 수 있습니다. 예시 OpenAPI 정의에는 두 가지 검사기가 있습니다.

- `RequestBodyModel` 데이터 모델을 사용하여 본문과 파라미터를 모두 확인하는 `all` 검사기
- 파라미터만 확인하는 `param-only` 검사기

API의 모든 메서드에 대해 요청 검사기를 활성화하려면 API OpenAPI 정의의 API 수준에서 x-amazon-apigateway-request-유효성 검사기 속성 속성을 지정합니다. 예시 OpenAPI 정의의 경우, 달리 재정의되지 않는 한 `all` 검사기가 모든 API 메서드에 사용됩니다. 모델을 사용하여 본문을 확인할 때 일치하는 콘텐츠 유형이 없으면 요청 확인이 수행되지 않습니다. 콘텐츠 유형에 관계없이 동일한 모델을 사용하려면 `$default`을(를) 키로 지정합니다.

개별 메서드에 대해 요청 검사기를 활성화하려면 메서드 수준에서 `x-amazon-apigateway-request-validator` 속성을 지정합니다. 예시의 OpenAPI 정의에서는 `param-only` 검사기가 `GET` 메서드의 `all` 검사기를 덮어씁니다.

OpenAPI 예시를 API Gateway로 가져오려면 리전 API를 API Gateway로 가져오기 또는 엣지 최적화된 API를 API Gateway로 가져오기에 대한 다음 지침을 참조하세요.

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
    }
  }
}
```

기본 요청 확인이 설정된 샘플 API의 OpenAPI 정의

다음 OpenAPI 정의는 요청 확인이 활성화된 샘플 API를 정의합니다. PetStore API는 API의 하위 집합입니다. POST 메서드를 노출시켜 pets 모음에 pet을 추가하고, GET 메서드를 노출시켜 지정된 유형으로 pet을 쿼리합니다.

x-amazon-apigateway-request-validators 맵에 API 수준의 두 가지 요청 검사기가 선언되어 있습니다. params-only 검사기는 해당 API에서 활성화되어 있고 GET 메서드에 의해 상속됩니다. API Gateway는 이 검사기를 사용하여 필수 쿼리 파라미터(q1)가 수신 요청에 포함되었으며 공백이 아님을 확인할 수 있습니다. all 검사기는 POST 메서드에서 활성화됩니다. 이 검사기는 필수 헤더 파라미터(h1)가 설정되었으며 공백이 아님을 확인합니다. 또한 이는 페이로드 형식이 지정

된 RequestBodyModel을 준수하는지 확인합니다. 일치하는 콘텐츠 유형이 없는 경우 요청 확인이 수행되지 않습니다. 모델을 사용하여 본문을 확인할 때 일치하는 콘텐츠 유형이 없으면 요청 확인이 수행되지 않습니다. 콘텐츠 유형에 관계없이 동일한 모델을 사용하려면 $default을(를) 키로 지정합니다.

이 모델에서는 입력 JSON 객체가 name, type 및 price 속성을 포함해야 합니다. name 속성은 임의의 문자열이 될 수 있으며, type은 지정된 열거 필드(["dog", "cat", "fish"]) 중 하나여야 하고, price는 25~500 사이여야 합니다. id 파라미터는 필요하지 않습니다.

OpenAPI 2.0

```
{
  "swagger": "2.0",
  "info": {
    "title": "ReqValidators Sample",
    "version": "1.0.0"
  },
  "schemes": [
    "https"
  ],
  "basePath": "/v1",
  "produces": [
    "application/json"
  ],
  "x-amazon-apigateway-request-validators" : {
    "all" : {
      "validateRequestBody" : true,
      "validateRequestParameters" : true
    },
    "params-only" : {
      "validateRequestBody" : false,
      "validateRequestParameters" : true
    }
  },
  "x-amazon-apigateway-request-validator" : "params-only",
  "paths": {
    "/validation": {
      "post": {
        "x-amazon-apigateway-request-validator" : "all",
        "parameters": [
          {
            "in": "header",
            "name": "h1",
            "required": true
```

```
      },
      {
        "in": "body",
        "name": "RequestBodyModel",
        "required": true,
        "schema": {
          "$ref": "#/definitions/RequestBodyModel"
        }
      }
    ],
    "responses": {
      "200": {
        "schema": {
          "type": "array",
          "items": {
            "$ref": "#/definitions/Error"
          }
        },
        "headers" : {
          "test-method-response-header" : {
            "type" : "string"
          }
        }
      }
    },
    "security" : [{
      "api_key" : []
    }],
    "x-amazon-apigateway-auth" : {
      "type" : "none"
    },
    "x-amazon-apigateway-integration" : {
      "type" : "http",
      "uri" : "http://petstore-demo-endpoint.execute-api.com/petstore/pets",
      "httpMethod" : "POST",
      "requestParameters": {
        "integration.request.header.custom_h1": "method.request.header.h1"
      },
      "responses" : {
        "2\\d{2}" : {
          "statusCode" : "200"
        },
        "default" : {
          "statusCode" : "400",
```

```
            "responseParameters" : {
              "method.response.header.test-method-response-header" : "'static
value'"
            },
            "responseTemplates" : {
              "application/json" : "json 400 response template",
              "application/xml" : "xml 400 response template"
            }
          }
        }
      }
    },
    "get": {
      "parameters": [
        {
          "name": "q1",
          "in": "query",
          "required": true
        }
      ],
      "responses": {
        "200": {
          "schema": {
            "type": "array",
            "items": {
              "$ref": "#/definitions/Error"
            }
          },
          "headers" : {
            "test-method-response-header" : {
              "type" : "string"
            }
          }
        }
      },
      "security" : [{
        "api_key" : []
      }],
      "x-amazon-apigateway-auth" : {
        "type" : "none"
      },
      "x-amazon-apigateway-integration" : {
        "type" : "http",
        "uri" : "http://petstore-demo-endpoint.execute-api.com/petstore/pets",
```

```
          "httpMethod" : "GET",
          "requestParameters": {
            "integration.request.querystring.type": "method.request.querystring.q1"
          },
          "responses" : {
            "2\\d{2}" : {
              "statusCode" : "200"
            },
            "default" : {
              "statusCode" : "400",
              "responseParameters" : {
                "method.response.header.test-method-response-header" : "'static
 value'"
              },
              "responseTemplates" : {
                "application/json" : "json 400 response template",
                "application/xml" : "xml 400 response template"
              }
            }
          }
        }
      }
    }
  },
  "definitions": {
    "RequestBodyModel": {
      "type": "object",
      "properties": {
        "id":    { "type": "integer" },
        "type":  { "type": "string", "enum": ["dog", "cat", "fish"] },
        "name":  { "type": "string" },
        "price": { "type": "number", "minimum": 25, "maximum": 500 }
      },
      "required": ["type", "name", "price"]
    },
    "Error": {
      "type": "object",
      "properties": {

      }
    }
  }
}
```

AWS CloudFormation 기본 요청 검증이 포함된 샘플 API의 템플릿

다음 AWS CloudFormation 예제 템플릿 정의는 요청 검증이 활성화된 샘플 API를 정의합니다. PetStore API는 API의 하위 집합입니다. `POST` 메서드를 노출시켜 `pets` 모음에 pet을 추가하고, `GET` 메서드를 노출시켜 지정된 유형으로 pet을 쿼리합니다.

두 개의 요청 검사기가 선언되어 있습니다.

`GETValidator`

이 검사기는 `GET` 메서드에서 활성화됩니다. API Gateway는 이 검사기를 사용하여 필수 쿼리 파라미터(q1)가 수신 요청에 포함되었으며 공백이 아님을 확인할 수 있습니다.

`POSTValidator`

이 검사기는 `POST` 메서드에서 활성화됩니다. API Gateway는 이 검사기를 사용하여 콘텐츠 유형이 `application/json`일 때 페이로드 요청 형식이 지정된 `RequestBodyModel`을 준수하는지 확인합니다. 일치하는 콘텐츠 유형이 없는 경우 요청 검증이 수행되지 않습니다. 콘텐츠 유형에 상관없이 동일한 모델을 사용하려면 `$default`를 지정합니다. `RequestBodyModel`에는 pet ID를 정의하는 추가 모델 `RequestBodyModelId`가 포함되어 있습니다.

```
AWSTemplateFormatVersion: 2010-09-09
Parameters:
  StageName:
    Type: String
    Default: v1
    Description: Name of API stage.
Resources:
  Api:
    Type: 'AWS::ApiGateway::RestApi'
    Properties:
      Name: ReqValidatorsSample
  RequestBodyModelId:
    Type: 'AWS::ApiGateway::Model'
    Properties:
      RestApiId: !Ref Api
      ContentType: application/json
      Description: Request body model for Pet ID.
      Schema:
        $schema: 'http://json-schema.org/draft-04/schema#'
        title: RequestBodyModelId
        properties:
```

```
            id:
              type: integer
  RequestBodyModel:
    Type: 'AWS::ApiGateway::Model'
    Properties:
      RestApiId: !Ref Api
      ContentType: application/json
      Description: Request body model for Pet type, name, price, and ID.
      Schema:
        $schema: 'http://json-schema.org/draft-04/schema#'
        title: RequestBodyModel
        required:
          - price
          - name
          - type
        type: object
        properties:
            id:
              "$ref": !Sub
                - 'https://apigateway.amazonaws.com/restapis/${Api}/models/
${RequestBodyModelId}'
                - Api: !Ref Api
                  RequestBodyModelId: !Ref RequestBodyModelId
            price:
              type: number
              minimum: 25
              maximum: 500
            name:
              type: string
            type:
              type: string
              enum:
                - "dog"
                - "cat"
                - "fish"
  GETValidator:
    Type: AWS::ApiGateway::RequestValidator
    Properties:
      Name: params-only
      RestApiId: !Ref Api
      ValidateRequestBody: False
      ValidateRequestParameters: True
  POSTValidator:
    Type: AWS::ApiGateway::RequestValidator
```

```
    Properties:
      Name: body-only
      RestApiId: !Ref Api
      ValidateRequestBody: True
      ValidateRequestParameters: False
  ValidationResource:
    Type: 'AWS::ApiGateway::Resource'
    Properties:
      RestApiId: !Ref Api
      ParentId: !GetAtt Api.RootResourceId
      PathPart: 'validation'
  ValidationMethodGet:
    Type: 'AWS::ApiGateway::Method'
    Properties:
      RestApiId: !Ref Api
      ResourceId: !Ref ValidationResource
      HttpMethod: GET
      AuthorizationType: NONE
      RequestValidatorId: !Ref GETValidator
      RequestParameters:
        method.request.querystring.q1: true
      Integration:
        Type: HTTP_PROXY
        IntegrationHttpMethod: GET
        Uri: http://petstore-demo-endpoint.execute-api.com/petstore/pets/
  ValidationMethodPost:
    Type: 'AWS::ApiGateway::Method'
    Properties:
      RestApiId: !Ref Api
      ResourceId: !Ref ValidationResource
      HttpMethod: POST
      AuthorizationType: NONE
      RequestValidatorId: !Ref POSTValidator
      RequestModels:
        application/json : !Ref RequestBodyModel
      Integration:
        Type: HTTP_PROXY
        IntegrationHttpMethod: POST
        Uri: http://petstore-demo-endpoint.execute-api.com/petstore/pets/
  ApiDeployment:
    Type: 'AWS::ApiGateway::Deployment'
    DependsOn:
      - ValidationMethodGet
      - RequestBodyModel
```

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

REST API에 대한 데이터 변환 설정

API Gateway에서 API 메서드 요청은 통합 요청 페이로드에서 여러 유형의 페이로드를 취할 수 있습니다. 마찬가지로 백엔드에서는 메서드 응답 페이로드가 아닌 통합 응답 페이로드를 반환할 수 있습니다. 매핑 템플릿을 사용하여 API Gateway에서 URL 경로 파라미터, URL 쿼리 문자열 파라미터, HTTP 헤더 및 요청 본문을 매핑할 수 있습니다.

매핑 템플릿이란 Velocity Template Language(VTL)로 표현된 스크립트로, JSONPath 표현식을 사용하여 페이로드에 적용됩니다.

페이로드는 JSON 스키마 draft 4에 따라 데이터 모델을 가질 수 있습니다. 모델에 대한 자세한 정보는 데이터 모델 이해 섹션을 참조하세요.

> **Note**
>
> 매핑 템플릿을 생성하기 위해 모델을 정의할 필요는 없지만, API Gateway에서 SDK를 생성하도록 하거나 API에 대한 요청 본문 확인을 활성화하려면 모델을 정의해야 합니다.

주제

- 매핑 템플릿 이해
- API Gateway에서 데이터 변환 설정
- 매핑 템플릿을 사용한 API의 요청 및 응답 파라미터와 상태 코드 재정의
- API Gateway 콘솔을 사용하여 요청 및 응답 데이터 매핑 설정
- 모델 및 매핑 템플릿 예제
- Amazon API Gateway API 요청 및 응답 데이터 매핑 참조
- API Gateway 매핑 템플릿과 액세스 로깅 변수 참조

매핑 템플릿 이해

API Gateway에서 API 메서드 요청 또는 응답은 통합 요청 페이로드 또는 응답에서 여러 유형의 페이로드를 취할 수 있습니다.

데이터를 다음과 같이 변환할 수 있습니다.

- 페이로드를 API 지정 형식에 일치시킵니다.
- API의 요청 및 응답 파라미터와 상태 코드를 재정의합니다.
- 클라이언트가 선택한 응답 헤더를 반환합니다.
- HTTP 프록시 또는 프록시의 메서드 요청에서 경로 파라미터, 쿼리 문자열 파라미터 또는 헤더 파라미터를 연결합니다. AWS 서비스
- 통합을 사용하여 전송할 데이터 (예: Amazon DynamoDB 또는 Lambda 함수 또는 HTTP 엔드포인트) 를 선택합니다. AWS 서비스

매핑 템플릿을 사용하여 데이터를 변환할 수 있습니다. 매핑 템플릿이란 Velocity Template Language(VTL)로 표현된 스크립트로, JSONPath를 사용하여 페이로드에 적용됩니다.

다음 예제는 데이터 변환을 위한 입력 데이터, 매핑 템플릿, 출력 데이터를 보여줍니다. PetStore

입력 데이터

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

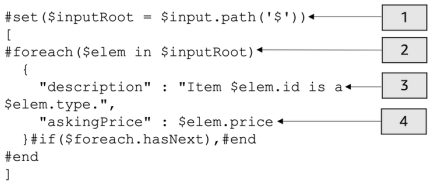
매핑 템플릿

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

#end
]
```

출력 데이터

```
[
  {
    "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
  }
]
```

다음 다이어그램은 이 매핑 템플릿의 세부 정보를 보여줍니다.



1. `$inputRoot` 변수는 이전 단원에서 다룬 원본 JSON 데이터의 루트 객체를 나타냅니다. 지시어는 # 기호로 시작됩니다.
2. `foreach` 루프는 원래 JSON 데이터의 각 객체를 반복합니다.
3. 설명은 Pet의 `id`와 원래 JSON 데이터의 `type`를 결합한 것입니다.
4. `askingPrice`는 원래 JSON 데이터의 `price`입니다.

PetStore 매핑 템플릿

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

이 매핑 템플릿은 다음과 같습니다.

1. 1번째 줄에서 `$inputRoot` 변수는 이전 섹션에서 다룬 원래 JSON 데이터의 루트 객체를 나타냅니다. 지시어는 # 기호로 시작됩니다.
2. 3번째 줄의 `foreach` 루프는 원래 JSON 데이터의 각 객체를 반복합니다.
3. 5번째 줄의 `description`은 Pet의 `id`와 원래 JSON 데이터의 `type`를 결합한 것입니다.
4. 6번째 줄의 `askingPrice`는 원래 JSON 데이터의 `price`입니다.

VTL(Velocity Template Language)에 대한 자세한 내용은 Apache Velocity - VTL 참조를 참조하십시오. JSONPath에 대한 자세한 내용은 JSONPath - JSON용 XPath를 참조하십시오.

매핑 템플릿은 JSON 객체를 기본 데이터로 간주합니다. 이 데이터를 위한 모델을 정의할 필요는 없습니다. 그러나 출력 데이터에 대한 모델을 사용하면 이전 데이터를 언어별 객체로 반환할 수 있습니다. 자세한 설명은 데이터 모델 이해 섹션을 참조하세요.

복잡한 매핑 템플릿

또한 더 복잡한 매핑 템플릿을 생성할 수 있습니다. 다음 예시에서는 반려동물을 살 여력이 있는지 확인하기 위해 참조 연결과 컷오프 100을 사용하는 것을 보여줍니다.

입력 데이터

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

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

매핑 템플릿

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

#end
]
```

출력 데이터

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

더 복잡한 모델은 예시 사진 앨범(예제 사진)을 참조하세요.

API Gateway에서 데이터 변환 설정

이 섹션에서는 콘솔과 AWS CLI를 사용하여 통합 요청 및 응답을 변환하도록 매핑 템플릿을 설정하는 방법을 보여줍니다.

주제

- API Gateway 콘솔을 사용하여 데이터 변환 설정
- AWS CLI를 사용하여 데이터 변환 설정
- 완성된 데이터 변환 AWS CloudFormation 템플릿
- 다음 단계

API Gateway 콘솔을 사용하여 데이터 변환 설정

이 자습서에서는 다음.zip 파일.zip을 사용하여 불완전한 API 및 DynamoDB 테이블을 생성합니다. data-transformation-tutorial-console 이 불완전한 API에는 `GET` 및 `POST` 메서드가 포함된 `/pets` 리소스가 있습니다.

- `GET` 메서드는 `http://petstore-demo-endpoint.execute-api.com/petstore/pets` HTTP 엔드포인트에서 데이터를 가져옵니다. 출력 데이터는 PetStore 매핑 템플릿의 매핑 템플릿에 따라 변환됩니다.
- `POST` 메서드를 사용하면 사용자가 매핑 템플릿을 사용하여 Amazon DynamoDB 테이블에 `POST` 반려동물 정보를 입력할 수 있습니다.

의 앱 생성 템플릿을 다운로드하고 압축을 풉니다. AWS CloudFormation 이 템플릿을 사용하여 반려동물 정보와 불완전한 API를 게시하기 위한 DynamoDB 테이블을 생성할 것입니다. API Gateway 콘솔에서 나머지 단계를 완료하게 됩니다.

스택을 AWS CloudFormation 만들려면

1. https://console.aws.amazon.com/cloudformation 에서 AWS CloudFormation 콘솔을 엽니다.
2. 스택 생성을 선택한 다음 새 리소스 사용(표준)을 선택합니다.
3. 템플릿 지정에서 템플릿 파일 업로드를 선택합니다.
4. 다운로드한 템플릿을 선택합니다.
5. 다음을 선택합니다.
6. 스택 이름에 **`data-transformation-tutorial-console`**을(를) 입력하고 다음을 선택합니다.
7. 스택 옵션 구성에서 다음을 선택합니다.
8. Capactivity의 경우 계정에서 IAM 리소스를 생성할 AWS CloudFormation 수 있다는 점을 확인하십시오.
9. 스택 생성을 선택합니다.

AWS CloudFormation 템플릿에 지정된 리소스를 프로비저닝합니다. 리소스 프로비저닝을 완료하는 데 몇 분 정도 걸릴 수 있습니다. AWS CloudFormation 스택 상태가 CREATE_COMPLETE이면 다음 단계로 넘어갈 준비가 된 것입니다.

`GET` 통합 응답을 테스트하는 방법

1. AWS CloudFormation 스택의 리소스 탭에서 API의 물리적 ID를 선택합니다. **`data-transformation-tutorial-console`**
2. 기본 탐색 창에서 리소스를 선택하고 GET 메서드를 선택합니다.
3. 테스트 탭을 선택합니다. 탭을 표시하려면 오른쪽 화살표 버튼을 선택해야 할 수도 있습니다.

   테스트 결과는 다음과 같습니다.

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

PetStore 매핑 템플릿의 매핑 테이블에 따라 이 출력 데이터를 변환합니다.

GET 통합 응답을 변환하는 방법

1. 통합 응답 탭을 선택합니다.

   현재 정의된 매핑 템플릿이 없으므로 통합 응답은 변환되지 않습니다.
2. 기본값 - 응답에서 편집을 선택합니다.
3. 매핑 템플릿을 선택한 후 다음을 수행합니다.

   a. 매핑 템플릿 추가(Add mapping template)를 선택합니다.

   b. 콘텐츠 유형에 **application/json**을 입력합니다.

   c. 템플릿 본문에 다음을 입력합니다.

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

```
#end
]
```

저장을 선택합니다.

GET 통합 응답을 테스트하는 방법

- 테스트 탭을 선택한 다음 테스트를 선택합니다.

  테스트 출력에는 변환된 응답이 표시됩니다.

```
[
  {
    "description" : "Item 1 is a dog.",
    "askingPrice" : 249.99
  },
  {
    "description" : "Item 2 is a cat.",
    "askingPrice" : 124.99
  },
  {
    "description" : "Item 3 is a fish.",
    "askingPrice" : 0.99
  }
]
```

POST 메서드의 입력 데이터를 변환하는 방법

1. POST 메서드를 선택합니다.
2. 통합 요청 탭을 선택한 다음 통합 요청 설정에서 편집을 선택합니다.

   AWS CloudFormation 템플릿이 일부 통합 요청 필드를 채웠습니다.

   - 통합 유형은 입니다. AWS 서비스
   - AWS 서비스 는 DynamoDB입니다.
   - HTTP 메서드는 POST입니다.
   - 작업은 PutItem입니다.

- API Gateway가 DynamoDB 테이블에 항목을 넣을 수 있도록 허용하는 실행 역할은 다음과 같습니다. `data-transformation-tutorial-console-APIGatewayRole` AWS CloudFormation API Gateway가 DynamoDB와 상호 작용하는 데 필요한 최소한의 권한을 갖도록 하기 위해 이 역할을 생성했습니다.

DynamoDB 테이블의 이름이 지정되지 않았습니다. 다음 단계에서 이름을 지정합니다.

3. 요청 본문 패스스루에서 없음을 선택합니다.

   이렇게 하면 API는 매핑 템플릿이 없는 Content-Types의 데이터를 거부합니다.

4. 매핑 템플릿을 선택합니다.
5. 콘텐츠 유형이 `application/json`으로 설정되어 있습니다. 이는 application/json이 아닌 콘텐츠 유형은 API에서 거부함을 의미합니다. 통합 패스스루 동작에 관한 자세한 정보는 통합 패스스루 동작 섹션을 참조하세요.
6. 텍스트 편집기에 다음 코드를 입력합니다.

```
{
    "TableName":"data-transformation-tutorial-console-ddb",
    "Item": {
        "id": {
            "N": $input.json("$.id")
        },
        "type": {
            "S": $input.json("$.type")
        },
        "price": {
            "N": $input.json("$.price")
        }
    }
}
```

   이 템플릿은 테이블을 `data-transformation-tutorial-console-ddb`로 지정하고 항목을 `id`, `type` 및 `price`로 설정합니다. 항목은 `POST` 메서드 본문에서 가져옵니다. 또한 매핑 템플릿을 만드는 데 데이터 모델을 사용할 수 있습니다. 자세한 설명은 API Gateway에서 요청 확인 사용 섹션을 참조하세요.

7. 저장을 선택하여 매핑 템플릿을 저장합니다.

메서드 및 **POST** 메서드의 통합 응답을 추가하는 방법

빈 메서드와 통합 응답을 AWS CloudFormation 생성했습니다. 이 응답을 편집하여 자세한 정보를 제공할 수 있습니다. 응답을 편집하는 방법에 대한 자세한 내용은 Amazon API Gateway API 요청 및 응답 데이터 매핑 참조 섹션을 참조하세요.

1. 통합 응답 탭의 기본값 - 응답에서 편집을 선택합니다.
2. 매핑 템플릿을 선택한 다음 매핑 템플릿 추가를 선택합니다.
3. Content-type에 **application/json**을 입력합니다.
4. 코드 편집기에서 다음 출력 매핑 템플릿을 입력하여 출력 메시지를 보냅니다.

   ```
   { "message" : "Your response was recorded at $context.requestTime" }
   ```

   컨텍스트 변수에 대한 자세한 내용은 $context데이터 모델, 권한 부여자, 매핑 템플릿, 액세스 로깅용 변수 CloudWatch 섹션을 참조하세요.
5. 저장을 선택하여 매핑 템플릿을 저장합니다.

POST 메서드 테스트

테스트 탭을 선택합니다. 탭을 표시하려면 오른쪽 화살표 버튼을 선택해야 할 수도 있습니다.

1. 요청 본문에 다음 예를 입력합니다.

   ```
   {
             "id": "4",
             "type" : "dog",
             "price": "321"
   }
   ```
2. 테스트를 선택합니다.

   출력에 성공 메시지가 표시되어야 합니다.

   https://console.aws.amazon.com/dynamodb/에서 DynamoDB 콘솔을 열어 예시 항목이 테이블에 있는지 확인할 수 있습니다.

AWS CloudFormation 스택을 삭제하려면

1. https://console.aws.amazon.com/cloudformation 에서 AWS CloudFormation 콘솔을 엽니다.

2. AWS CloudFormation 스택을 선택합니다.
3. 삭제를 선택한 다음 해당 선택을 확인합니다.

AWS CLI를 사용하여 데이터 변환 설정

이 자습서에서는 다음.zip 파일.zip을 사용하여 불완전한 API 및 DynamoDB 테이블을 생성합니다. data-transformation-tutorial-cli 이 불완전한 API에는 `http://petstore-demo-endpoint.execute-api.com/petstore/pets` HTTP 엔드포인트와 통합된 `GET` 메서드가 포함된 `/pets` 리소스가 있습니다. DynamoDB 테이블에 연결할 `POST` 메서드를 생성하고 매핑 템플릿을 사용하여 DynamoDB 테이블에 데이터를 입력합니다.

- PetStore 매핑 템플릿의 매핑 템플릿에 따라 출력 데이터를 변화합니다.
- 사용자가 매핑 템플릿을 사용하여 Amazon DynamoDB 테이블에 `POST` 반려동물 정보를 입력할 수 있도록 `POST` 메서드를 생성합니다.

스택을 생성하려면 AWS CloudFormation

의 앱 생성 템플릿을 다운로드하고 압축을 풉니다. AWS CloudFormation

다음 자습서를 완료하려면 AWS Command Line Interface (AWS CLI) 버전 2가 필요합니다.

긴 명령의 경우 이스케이프 문자(\)를 사용하여 명령을 여러 행으로 분할합니다.

> **Note**
>
> Windows에서는 일반적으로 사용하는 일부 Bash CLI 명령(예: `zip`)은 운영 체제의 기본 제공 터미널에서 지원되지 않습니다. Ubuntu와 Bash의 Windows 통합 버전을 가져오려면 Linux용 Windows Subsystem을 설치합니다. 이 안내서의 예제 CLI 명령은 Linux 형식을 사용합니다. Windows CLI를 사용하는 경우 인라인 JSON 문서를 포함하는 명령의 형식을 다시 지정해야 합니다.

1. 다음 명령어를 사용하여 AWS CloudFormation 스택을 생성합니다.

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

2. AWS CloudFormation 템플릿에 지정된 리소스를 프로비저닝합니다. 리소스 프로비저닝을 완료하는 데 몇 분 정도 걸릴 수 있습니다. 다음 명령을 사용하여 AWS CloudFormation 스택 상태를 확인합니다.

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

3. AWS CloudFormation 스택 상태가 이면 `StackStatus: "CREATE_COMPLETE"` 다음 명령을 사용하여 향후 단계를 위한 관련 출력 값을 검색하십시오.

```
aws cloudformation describe-stacks --stack-name data-transformation-tutorial-cli
--query "Stacks[*].Outputs[*].{OutputKey: OutputKey, OutputValue: OutputValue,
Description: Description}"
```

출력 값은 다음과 같습니다.

- ApiRole는 API Gateway가 DynamoDB 테이블에 항목을 넣을 수 있도록 허용하는 역할 이름입니다. 이 자습서에서 역할 이름은 `data-transformation-tutorial-cli-APIGatewayRole-ABCDEFG`입니다.
- TableNameDDB는 DynamoDB 테이블의 이름입니다. 이 자습서에서 테이블 이름은 `data-transformation-tutorial-cli-ddb`입니다.
- ResourceId, `GET` 및 `POST` 메서드가 노출되는 pets 리소스의 ID입니다. 이 자습서에서 리소스 ID는 `efg456`입니다.
- ApiId, API의 ID입니다. 이 자습서에서 API ID는 `abc123`입니다.

데이터 변환 전에 **`GET`** 메서드를 테스트하는 방법

- 다음 명령을 사용하여 `GET` 메서드를 테스트합니다.

```
aws apigateway test-invoke-method --rest-api-id abc123 \
          --resource-id efg456 \
          --http-method GET
```

테스트의 출력은 다음과 같습니다.

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

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

PetStore 매핑 템플릿의 매핑 테이블에 따라 이 출력 데이터를 변환합니다.

GET 통합 응답을 변환하는 방법

- 다음 명령을 사용하여 GET 메서드 통합 응답을 업데이트합니다. *rest-api-id*및 *### ID#* 원하는 값으로 바꾸십시오.

  다음 명령을 사용하여 통합 응답을 생성합니다.

```
aws apigateway put-integration-response --rest-api-id abc123 \
  --resource-id efg456 \
  --http-method GET \
  --status-code 200 \
  --selection-pattern "" \
  --response-templates '{"application/json": "#set($inputRoot = $input.path(\"$
\"))\n[\n#foreach($elem in $inputRoot)\n {\n  \"description\": \"Item $elem.id is a
 $elem.type\",\n  \"askingPrice\": \"$elem.price\"\n }#if($foreach.hasNext),#end\n
\n#end\n]"}'
```

GET 메서드를 테스트하는 방법

- 다음 명령을 사용하여 GET 메서드를 테스트합니다.

```
aws apigateway test-invoke-method --rest-api-id abc123 \
  --resource-id efg456 \
  --http-method GET \
```

테스트 출력에는 변환된 응답이 표시됩니다.

```
[
  {
    "description" : "Item 1 is a dog.",
    "askingPrice" : 249.99
  },
  {
    "description" : "Item 2 is a cat.",
    "askingPrice" : 124.99
  },
  {
    "description" : "Item 3 is a fish.",
    "askingPrice" : 0.99
  }
]
```

POST 메서드를 생성하는 방법

1. 다음 명령을 사용하여 /pets 리소스에 새로운 메서드를 생성합니다.

```
aws apigateway put-method --rest-api-id abc123 \
  --resource-id efg456 \
  --http-method POST \
  --authorization-type "NONE" \
```

이 메서드를 사용하면 스택에서 생성한 DynamoDB 테이블로 애완동물 정보를 보낼 수 있습니다. AWS CloudFormation

2. 다음 명령을 사용하여 메서드에 AWS 서비스 POST 통합을 생성합니다.

```
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. 다음 명령을 사용하여 성공적인 POST 메서드 호출에 대한 메서드 응답을 생성합니다.

```
aws apigateway put-method-response --rest-api-id abc123 \
  --resource-id efg456 \
  --http-method POST \
  --status-code 200
```

4. 다음 명령을 사용하여 성공적인 POST 메서드 호출에 대한 통합 응답을 생성합니다.

```
aws apigateway put-integration-response --rest-api-id abc123 \
  --resource-id efg456 \
  --http-method POST \
  --status-code 200 \
  --selection-pattern "" \
  --response-templates '{"application/json": "{\"message\": \"Your response was
 recorded at $context.requestTime\"}"}'
```

POST 메서드를 테스트하는 방법

- 다음 명령을 사용하여 POST 메서드를 테스트합니다.

```
aws apigateway test-invoke-method --rest-api-id abc123 \
  --resource-id efg456 \
  --http-method POST \
  --body '{\"id\": \"4\", \"type\": \"dog\", \"price\": \"321\"}'
```

출력에 성공 메시지가 표시됩니다.

AWS CloudFormation 스택을 삭제하려면

- 다음 명령어를 사용하여 AWS CloudFormation 리소스를 삭제합니다.

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

완성된 데이터 변환 AWS CloudFormation 템플릿

다음 예제는 GET 리소스와 메서드가 포함된 DynamoDB 테이블과 API를 생성하는 완성된 AWS CloudFormation /pets 템플릿입니다. POST

- GET 메서드는 http://petstore-demo-endpoint.execute-api.com/petstore/pets HTTP 엔드포인트에서 데이터를 가져옵니다. 출력 데이터는 PetStore 매핑 템플릿의 매핑 템플릿에 따라 변환됩니다.
- POST 메서드를 사용하면 사용자가 매핑 템플릿을 사용하여 DynamoDB 테이블에 POST 반려동물 정보를 입력할 수 있습니다.

```
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
```

다음 단계

더 복잡한 매핑 템플릿을 알아보려면 다음 예를 참조하세요.

- 예시 사진 앨범(예제 사진)을 통해 더 복잡한 모델과 매핑 템플릿을 볼 수 있습니다.
- 모델에 대한 자세한 내용은 데이터 모델 이해 섹션을 참조하세요.
- 다양한 응답 코드 출력을 매핑하는 방법에 대한 자세한 내용은 API Gateway 콘솔을 사용하여 요청 및 응답 데이터 매핑 설정 섹션을 참조하세요.
- API의 메서드 요청 데이터에서 데이터 매핑을 설정하는 방법에 대한 자세한 내용은 API Gateway 매핑 템플릿과 액세스 로깅 변수 참조 섹션을 참조하세요.

매핑 템플릿을 사용한 API의 요청 및 응답 파라미터와 상태 코드 재정의

표준 API Gateway 매개변수 및 응답 코드 매핑 템플릿을 사용하면 매개변수를 one-to-one 매핑하고 통합 응답 상태 코드 계열 (정규 표현식으로 일치) 을 단일 응답 상태 코드에 매핑할 수 있습니다. 매핑 템플릿 재정의를 사용하면 유연하게 many-to-one 매개변수 매핑을 수행하고, 표준 API Gateway 매핑이 적용된 후 매개변수를 재정의하고, 본문 내용이나 기타 매개변수 값을 기준으로 매개변수를 조건부로 매핑하고, 프로그래밍 방식으로 새 매개변수를 즉시 생성하고, 통합 엔드포인트에서 반환된 상태 코드를 재정의할 수 있습니다. 모든 유형의 요청 파라미터와 응답 헤더, 응답 상태 코드 등을 재정의할 수 있습니다.

다음은 매핑 템플릿 재정의의 사용 예시입니다.

- 새 헤더를 두 파라미터의 연속으로 생성(또는 기존 헤더 덮어쓰기)
- 응답 코드를 본문 콘텐츠에 따라 성공 또는 실패로 재정의
- 파라미터를 콘텐츠 또는 다른 파라미터의 콘텐츠를 바탕으로 조건부 리매핑
- JSON 본문의 콘텐츠에 반복 및 헤더 또는 쿼리 문자열로 키 값 페어 리매핑

매핑 템플릿 재정의를 생성하려면 매핑 템플릿에서 다음 `$context` 변수 중 하나 이상을 사용합니다.

요청 본문 매핑 템플릿	응답 본문 매핑 템플릿
`$context.requestOverride.header. header_name`	`$context.responseOverride.header. header_name`
`$context.requestOverride.path. path_name`	`$context.responseOverride.status`
`$context.requestOverride.querystring. querystring_name`	

> **Note**
>
> 매핑 템플릿 재정의는 프록시 통합 엔드포인트와 함께 사용할 수 없습니다. 데이터 매핑이 부족하기 때문입니다. 통합 유형에 대한 자세한 내용은 API Gateway API 통합 유형 선택 항목을 참조하십시오.

> **Important**
>
> 재정의는 최종입니다. 각 파라미터에 한 번만 재정의를 적용할 수 있습니다. 같은 파라미터에 여러 번 재정의를 시도하면 Amazon API Gateway에서 5XX 응답을 반환합니다. 템플릿에서 같은 파라미터를 여러 번 재정의해야 할 경우에는 변수를 하나 생성하여 템플릿 마지막에 재정의를 적용하는 것이 좋습니다. 전체 템플릿을 구문 분석한 후에만 템플릿이 적용된다는 점에 유의하세요. 자습서: API Gateway 콘솔에서 API의 요청 파라미터 및 헤더 재정의 섹션을 참조하십시오.

다음 자습서에서는 API Gateway 콘솔에서 매핑 템플릿 재정의를 생성하고 테스트하는 방법을 보여줍니다. 이 자습서에서는 샘플 API를 출발점으로 사용합니다. PetStore 두 자습서 모두 PetStore 샘플 API를 이미 만들었다고 가정합니다.

주제

- 자습서: API Gateway 콘솔에서 API의 응답 상태 코드 재정의
- 자습서: API Gateway 콘솔에서 API의 요청 파라미터 및 헤더 재정의
- 예: API Gateway CLI에서 API의 요청 파라미터 및 헤더 재정의

- 예: 다음에 대한 SDK를 사용하여 API의 요청 파라미터 및 헤더를 재정의합니다. JavaScript

자습서: API Gateway 콘솔에서 API의 응답 상태 코드 재정의

PetStore 샘플 API를 사용하여 애완동물을 가져오려면 의 `GET /pets/{petId}` API 메서드 요청을 사용합니다. 여기서 `{petId}` 는 런타임에 숫자를 받을 수 있는 경로 매개변수입니다.

이 자습서에서는 오류 조건을 감지했을 때 `GET`를 `$context.responseOverride.status`으로 매핑하는 매핑 템플릿을 생성하여 이 `400` 메서드의 응답 코드를 재정의합니다.

1. https://console.aws.amazon.com/apigateway에서 API Gateway 콘솔에 로그인합니다.
2. PetStore API에서 API를 선택한 다음 리소스를 선택합니다.
3. 리소스 트리에서, `/{petId}` 아래의 `GET` 메서드를 선택합니다.
4. 테스트 탭을 선택합니다. 탭을 표시하려면 오른쪽 화살표 버튼을 선택해야 할 수도 있습니다.
5. petId에 **`-1`**을 입력한 다음 테스트를 선택합니다.

   결과에 두 가지 사실이 보입니다.

   먼저 응답 본문에 out-of-range 오류가 표시됩니다.

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

   둘째, 로그 상자의 마지막 줄이 `Method completed with status: 200`으로 끝납니다.
6. 통합 응답 탭의 기본값 - 응답에서 편집을 선택합니다.
7. 매핑 템플릿을 선택합니다.
8. 매핑 템플릿 추가(Add mapping template)를 선택합니다.
9. 콘텐츠 유형에 **`application/json`**을 입력합니다.
10. 템플릿 본문에 다음을 입력합니다.

    ```
    #set($inputRoot = $input.path('$'))
    $input.json("$")
    ```

```
#if($inputRoot.toString().contains("error"))
#set($context.responseOverride.status = 400)
#end
```

11. 저장을 선택합니다.
12. 테스트 탭을 선택합니다.
13. petId에 **-1**을 입력합니다.
14. 결과에서 응답 본문은 out-of-range 오류를 나타냅니다.

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

그러나 로그 상자의 마지막 줄은 이제 `Method completed with status: 400`으로 끝납니다.

자습서: API Gateway 콘솔에서 API의 요청 파라미터 및 헤더 재정의

이 자습서에서는 다른 두 헤더를 결합하는 새 헤더 하나로 `GET`을 매핑하는 매핑 템플릿을 생성하여 `$context.requestOverride.header.`*`header_name`* 메서드의 요청 헤더 코드를 재정의합니다.

1. https://console.aws.amazon.com/apigateway에서 API Gateway 콘솔에 로그인합니다.
2. API에서 PetStore API를 선택합니다.
3. 리소스 트리에서, `/pet` 아래의 `GET` 메서드를 선택합니다.
4. 메서드 요청 탭의 메서드 요청 설정에서 편집을 선택합니다.
5. HTTP 요청 헤더를 선택한 다음, 헤더 추가를 선택합니다.
6. 이름에 **header1**를 입력합니다.
7. 헤더 추가를 선택한 다음 **header2**라는 두 번째 헤더를 생성합니다.
8. 저장을 선택합니다.
9. 통합 요청 탭의 통합 요청 설정에서 편집을 선택합니다.
10. 요청 본문 패스스루에서 정의된 템플릿이 없는 경우(권장)를 선택합니다.

11. 매핑 템플릿을 선택한 후 다음을 수행합니다.

    a. 매핑 템플릿 추가(Add mapping template)를 선택합니다.

    b. 콘텐츠 유형에 **application/json**을 입력합니다.

    c. 템플릿 본문에 다음을 입력합니다.

```
#set($header1Override = "foo")
#set($header3Value = "$input.params('header1')$input.params('header2')")
$input.json("$")
#set($context.requestOverride.header.header3 = $header3Value)
#set($context.requestOverride.header.header1 = $header1Override)
#set($context.requestOverride.header.multivalueheader=[$header1Override,
 $header3Value])
```

12. 저장을 선택합니다.
13. 테스트 탭을 선택합니다.
14. {pets}의 헤더(Headers)에 다음 코드를 복사합니다.

```
header1:header1Val
header2:header2Val
```

15. 테스트를 선택합니다.

    로그에 다음 텍스트가 포함된 항목이 보입니다.

```
Endpoint request headers: {header3=header1Valheader2Val,
header2=header2Val, header1=foo, x-amzn-apigateway-api-id=<api-id>,
Accept=application/json, multivalueheader=foo,header1Valheader2Val}
```

예: API Gateway CLI에서 API의 요청 파라미터 및 헤더 재정의

다음 CLI 예시는 `put-integration` 명령을 사용하여 응답 코드를 재정의하는 방법을 보여줍니다.

```
aws apigateway put-integration --rest-api-id <API_ID> --resource-
id <PATH_TO_RESOURCE_ID> --http-method <METHOD>
            --type <INTEGRATION_TYPE> --request-templates <REQUEST_TEMPLATE_MAP>
```

여기에서 *<REQUEST_TEMPLATE_MAP>*은(는) 콘텐츠 유형에서 템플릿 문자열로 적용하는 맵입니다. 그 맵의 구조는 다음과 같습니다.

```
Content_type1=template_string,Content_type2=template_string
```

또는 JSON 구문에서:

```
{"content_type1": "template_string"
...}
```

다음 예시는 `put-integration-response` 명령을 사용하여 API의 응답 코드를 재정의하는 방법을 보여줍니다.

```
aws apigateway put-integration-response --rest-api-id <API_ID> --resource-
id <PATH_TO_RESOURCE_ID> --http-method <METHOD>
            --status-code <STATUS_CODE> --response-templates <RESPONSE_TEMPLATE_MAP>
```

여기에서 *<RESPONSE_TEMPLATE_MAP>*의 형식은 *<REQUEST_TEMPLATE_MAP>*와(과) 동일합니다.

예: 다음에 대한 SDK를 사용하여 API의 요청 파라미터 및 헤더를 재정의합니다. JavaScript

다음 예시는 `put-integration` 명령을 사용하여 응답 코드를 재정의하는 방법을 보여줍니다.

요청:

```
var params = {
  httpMethod: 'STRING_VALUE', /* required */
  resourceId: 'STRING_VALUE', /* required */
  restApiId: 'STRING_VALUE', /* required */
  type: HTTP | AWS | MOCK | HTTP_PROXY | AWS_PROXY, /* required */
  requestTemplates: {
    '<Content_type>': 'TEMPLATE_STRING',
    /* '<String>': ... */
  },
};
apigateway.putIntegration(params, function(err, data) {
  if (err) console.log(err, err.stack); // an error occurred
  else     console.log(data);           // successful response
});
```

응답:

```
var params = {
  httpMethod: 'STRING_VALUE', /* required */
  resourceId: 'STRING_VALUE', /* required */
```

```
  restApiId: 'STRING_VALUE', /* required */
  statusCode: 'STRING_VALUE', /* required */
  responseTemplates: {
    '<Content_type>': 'TEMPLATE_STRING',
    /* '<String>': ... */
  },
};
apigateway.putIntegrationResponse(params, function(err, data) {
  if (err) console.log(err, err.stack); // an error occurred
  else     console.log(data);           // successful response
});
```

API Gateway 콘솔을 사용하여 요청 및 응답 데이터 매핑 설정

API Gateway 콘솔을 사용하여 API 통합 요청/응답을 정의하려면 다음 지침을 따르세요.

> **Note**
>
> 이 지침에서는 API Gateway 콘솔을 사용하여 API 통합 요청 설정 단원의 단계를 이미 완료한 것으로 가정합니다.

1. 리소스 창에서 메서드를 선택합니다.
2. 통합 요청 탭의 통합 요청 설정에서 편집을 선택합니다.
3. Lambda 함수, HTTP 프록시 또는 서비스 프록시로 변환하지 않고 매핑되지 않은 콘텐츠 유형의 메서드 요청 본문이 통합 요청을 통해 전달되는 방식을 구성하려면 요청 본문 통과 옵션을 선택합니다. AWS 여기에는 다음과 같은 세 가지 옵션이 있습니다.

   - 메서드 요청 콘텐츠 유형이 매핑 템플릿과 연결된 임의의 콘텐츠 유형과 일치하지 않은 경우 메서드 요청 본문이 통합 요청을 변환하지 않고 백엔드에 전달하도록 하려면 요청의 content-type 헤더와 일치하는 템플릿이 없는 경우를 선택합니다.

     > **Note**
     >
     > API Gateway API를 호출하는 경우 통합 리소스상에서 `WHEN_NO_MATCH`을 `passthroughBehavior` 속성 값으로 설정함으로써 이 옵션을 선택합니다.

   - 매핑 템플릿이 통합 요청에 정의되어 있지 않을 때 메서드 요청 본문이 통합 요청을 변환 없이 백엔드에 전달하도록 하려는 경우 When there are no templates defined (recommended)(정의된 템플릿이 없는 경우(권장))를 선택합니다. 이 옵션을 선택할 때 템플릿이 정의되어 있는 경우

매핑되지 않은 콘텐츠 유형의 메서드 요청은 HTTP 415 미지원 미디어 유형 응답과 함께 거부됩니다.

> **Note**
>
> API Gateway API를 호출하는 경우 통합 리소스상에서 `WHEN_NO_TEMPLATE`을 `passthroughBehavior` 속성 값으로 설정함으로써 이 옵션을 선택합니다.

- 메서드 요청 콘텐츠 유형이 통합 요청에 정의된 매핑 템플릿과 연결된 임의의 콘텐츠 유형과 일치하지 않거나 통합 요청에 매핑 템플릿이 정의되어 있지 않은 경우 메서드 요청이 전달하지 않도록 하려면 없음을 선택합니다. 매핑되지 않은 콘텐츠 유형의 메서드 요청은 HTTP 415 미지원 미디어 유형 응답과 함께 거부됩니다.

> **Note**
>
> API Gateway API를 호출하는 경우 통합 리소스상에서 `NEVER`을 `passthroughBehavior` 속성 값으로 설정함으로써 이 옵션을 선택합니다.

통합 패스스루 동작에 관한 자세한 정보는 통합 패스스루 동작을 참조하십시오.

4. HTTP 프록시 또는 AWS 서비스 프록시의 경우 통합 요청에 정의된 경로 파라미터, 쿼리 문자열 파라미터 또는 헤더 파라미터를 HTTP 프록시 또는 AWS 서비스 프록시의 메서드 요청의 해당 경로 파라미터, 쿼리 문자열 파라미터 또는 헤더 파라미터와 연결하려면 다음과 같이 하십시오.

   a. URL 경로 파라미터, URL 쿼리 문자열 파라미터 또는 HTTP 요청 헤더를 각각 선택한 다음 경로 추가, 쿼리 문자열 추가 또는 헤더 추가를 각각 선택합니다.

   b. [Name] 에는 HTTP 프록시 또는 AWS 서비스 프록시의 경로 파라미터, 쿼리 문자열 파라미터 또는 헤더 파라미터의 이름을 입력합니다.

   c. 다음에서 매핑됨에 경로 파라미터, 쿼리 문자열 파라미터 또는 헤더 파라미터의 매핑 값을 입력합니다. 다음 형식 중 하나를 사용합니다.

      - 메서드 요청 페이지에 정의된 *`parameter-name`*이라는 경로 파라미터에 대한 **`method.request.path.`*`parameter-name`***.
      - 메서드 요청 페이지에 정의된 *`parameter-name`*이라는 쿼리 문자열 파라미터에 대한 **`method.request.querystring.`*`parameter-name`***.
      - 메서드 요청 페이지에 정의된 *`parameter-name`*이라는 다중 값 쿼리 문자열 파라미터에 대한 **`method.request.multivaluequerystring.`*`parameter-name`***.

- 메서드 요청 페이지에 정의된 *parameter-name*이라는 헤더 파라미터에 대한 **method.request.header.*parameter-name***.

  아니면 리터럴 문자열 값(작은 따옴표 쌍으로 묶음)을 통합 헤더에 설정할 수 있습니다.
- 메서드 요청 페이지에 정의된 *parameter-name*이라는 다중 값 헤더 파라미터에 대한 **method.request.multivalueheader.*parameter-name***.

d. 다른 파라미터를 추가하려면 추가 버튼을 선택합니다.

5. 매핑 템플릿을 추가하려면 매핑 템플릿을 선택합니다.
6. 수신되는 요청에 대한 매핑 템플릿을 정의하려면 매핑 템플릿 추가를 선택합니다. 콘텐츠 유형에 콘텐츠 유형(예:**application/json**)을 입력합니다. 그런 다음, 매핑 템플릿을 직접 입력하거나 템플릿 생성을 선택하여 모델 템플릿에서 하나를 생성합니다. 자세한 설명은 매핑 템플릿 이해 섹션을 참조하세요.
7. 저장(Save)을 선택합니다.
8. 백엔드에서 받은 통합 응답을 호출 앱에 반환되는 API의 메서드 응답에 매핑할 수 있습니다. 백엔드에서 가용한 것 중 클라이언트가 선택한 응답 헤더에 반환하거나, 백엔드 응답 페이로드의 데이터 형식을 API 지정 형식으로 변환하는 것이 여기에 포함됩니다. 메서드 응답 및 통합 응답을 구성하여 그와 같은 매핑을 지정할 수 있습니다.

   메서드가 Lambda 함수, HTTP 프록시 AWS 또는 서비스 프록시에서 반환한 HTTP 상태 코드를 기반으로 사용자 지정 응답 데이터 형식을 수신하도록 하려면 다음을 수행하십시오.

   a. 통합 응답을 선택합니다. 기본값 - 응답에서 편집을 선택하여 메서드에서 200 HTTP 응답 코드에 대한 설정을 지정하거나, 응답 생성을 선택하여 메서드에서 임의의 다른 HTTP 응답 상태 코드에 대한 설정을 지정합니다.

   b. Lambda 오류 정규식 (Lambda 함수의 경우) 또는 HTTP 상태 정규식 (HTTP AWS 프록시 또는 서비스 프록시용) 의 경우 정규 표현식을 입력하여 이 출력 매핑에 매핑되는 Lambda 함수 오류 문자열 (Lambda 함수의 경우) 또는 HTTP 응답 상태 코드 (HTTP 프록시 또는 서비스 프록시용) 를 지정합니다. AWS 예를 들어, 모든 2xx HTTP 응답 상태 코드를 HTTP 프록시에서 이 출력 매핑으로 매핑하려면 HTTP status regex(HTTP 상태 regex)에서 '**2\d{2}**'를 입력합니다. Lambda 함수에서 “Invalid Request”를 포함하는 오류 메시지를 `400 Bad Request` 응답으로 반환하려면 “**.*Invalid request.***”를 Lambda 오류 regex 표현식으로 입력합니다. Lambda로부터 받은 매핑되지 않은 오류 메시지 모두에 대해 `400 Bad Request`를 반환하려면 Lambda 오류 regex에 “**(\n|.)+**”을 입력합니다. 이 마지막 표현식은 API의 기본 오류 응답에 사용할 수 있습니다.

> **Note**
>
> API Gateway는 응답 매핑을 위해서 Java 패턴 스타일 정규식을 사용합니다. 자세한 내용은 Oracle 설명서의 패턴 단원을 참조하십시오
> 오류 패턴은 Node.js의 `callback(errorMessage)`나 Java의 `throw new MyException(errorMessage)`에 의해 채워지는 Lambda 응답에 있는 `errorMessage` 속성의 전체 문자열과 대조됩니다. 또한 이스케이프된 문자는 표현식이 적용되기 전에 이스케이프가 해제됩니다.
> '+'를 선택 패턴으로 사용하여 응답을 필터링하는 경우 줄바꿈('\n') 문자를 포함하는 응답과 일치하지 않을 수 있다는 점에 유의하세요.

c. 활성화되어 있는 경우 메서드 응답 페이지에서 정의한 HTTP 응답 상태 코드를 메서드 응답 상태에서 선택합니다.

d. 메서드 응답 페이지에서 HTTP 응답 상태 코드에 대해 정의한 각 헤더에 대해 헤더 매핑에서 매핑 값을 지정합니다. Mapping value(매핑 값)에는 다음 형식 중 하나를 사용합니다.

- **`integration.response.multivalueheaders.`*`header-name`*** 여기에서 *`header-name`*은 백엔드로부터 받은 다중 값 응답 헤더의 이름입니다.

  예를 들어, 백엔드 응답의 `Date` 헤더를 API 메서드의 응답의 `Timestamp` 헤더로 반환하기 위해 Response header(응답 헤더) 열은 Timestamp 입력을 포함하게 되며 연결된 Mapping value(매핑 값)은 integration.response.multivalueheaders.Date로 설정되어야 합니다.

- **`integration.response.header.`*`header-name`*** 여기에서 *`header-name`*은 백엔드로부터 받은 단일 값 응답 헤더의 이름입니다.

  예를 들어, 백엔드 응답의 `Date` 헤더를 API 메서드의 응답의 `Timestamp` 헤더로 반환하기 위해 Response header(응답 헤더) 열은 Timestamp 입력을 포함하게 되며 연결된 Mapping value(매핑 값)은 integration.response.header.Date로 설정되어야 합니다.

e. 매핑 템플릿을 선택한 다음 매핑 템플릿 추가를 선택합니다. 콘텐츠 유형 상자에 Lambda 함수, HTTP 프록시 AWS 또는 서비스 프록시에서 메서드로 전달될 데이터의 콘텐츠 유형을 입력합니다. 그런 다음, 매핑 템플릿을 직접 입력하거나 템플릿 생성을 선택하여 모델 템플릿에서 하나를 생성합니다. 자세한 설명은 매핑 템플릿 이해 섹션을 참조하세요.

f. 저장(Save)을 선택합니다.

모델 및 매핑 템플릿 예제

다음 단원에서는 API Gateway에서 자체 API의 시작점으로 사용할 수 있는 모델 및 매핑 템플릿의 예제를 제공합니다. API Gateway의 모델 및 매핑 템플릿에 대한 자세한 내용은 PetStore 매핑 템플릿 단원을 참조하세요.

주제

- 사진 예제(API Gateway 모델 및 매핑 템플릿)
- 뉴스 기사 예제(API Gateway 모델 및 매핑 템플릿)
- 판매 송장 예(API Gateway 모델 및 매핑 템플릿)
- 직원 레코드 예(API Gateway 모델 및 매핑 템플릿)

사진 예제(API Gateway 모델 및 매핑 템플릿)

다음 예시는 API Gateway의 사진 앨범 API를 보여줍니다. 예시 데이터 변환, 추가 모델 및 매핑 템플릿을 제공합니다. 데이터 변환에 대한 자세한 내용은 매핑 템플릿 이해 섹션을 참조하세요. API Gateway의 모델 및 매핑 템플릿에 대한 자세한 내용은 PetStore 매핑 템플릿 섹션을 참조하세요.

주제

- 예시 데이터 변환
- 사진 데이터용 입력 모델
- 사진 데이터용 출력 모델
- 사진 데이터용 입력 매핑 템플릿

예시 데이터 변환

다음 예시에서는 Velocity Template Language(VTL) 매핑 템플릿을 사용하여 사진에 대한 입력 데이터를 변환하는 방법을 보여줍니다. Velocity Template Language에 대한 자세한 내용은 Apache Velocity - VTL 참조를 참조하세요.

입력 데이터

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

출력 매핑 템플릿

```
#set($inputRoot = $input.path('$'))
{
  "photos": [
#foreach($elem in $inputRoot.photos.photo)
    {
      "id": "$elem.id",
      "photographedBy": "$elem.photographer_first_name $elem.pho
tographer_last_name",
      "title": "$elem.title",
      "ispublic": $elem.ispublic,
      "isfriend": $elem.isfriend,
      "isfamily": $elem.isfamily
    }#if($foreach.hasNext),#end

#end
  ]
}
```

출력 데이터

```
{
  "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
    }
  ]
}
```

사진 데이터용 입력 모델

입력 데이터에 대한 모델을 정의할 수 있습니다. 이 입력 모델에는 한 장의 사진을 업로드해야 하며 페이지당 사진 수가 최소 10장으로 지정되어 있습니다. 이 입력 모델을 사용하여 SDK를 생성하거나 API에 대한 요청 확인을 활성화할 수 있습니다.

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

```
}
```

사진 데이터용 출력 모델

출력 데이터에 대한 모델을 정의할 수 있습니다. API에 유형이 엄격하게 지정된 SDK를 생성할 때 필수인 메서드 응답 모델에 이 모델을 사용할 수 있습니다. 이를 통해 출력은 Java 또는 Objective-C의 적절한 클래스에 캐스팅됩니다.

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

사진 데이터용 입력 매핑 템플릿

매핑 템플릿을 정의하여 입력 데이터를 수정할 수 있습니다. 추가 함수 통합 또는 통합 응답을 위해 입력 데이터를 수정할 수 있습니다.

```
#set($inputRoot = $input.path('$'))
{
  "photos": {
    "page": $inputRoot.photos.page,
    "pages": "$inputRoot.photos.pages",
    "perpage": $inputRoot.photos.perpage,
    "total": "$inputRoot.photos.total",
```

```
    "photo": [
#foreach($elem in $inputRoot.photos.photo)
      {
        "id": "$elem.id",
        "owner": "$elem.owner",
        "photographer_first_name" : "$elem.photographer_first_name",
        "photographer_last_name" : "$elem.photographer_last_name",
        "secret": "$elem.secret",
        "server": "$elem.server",
        "farm": $elem.farm,
        "title": "$elem.title",
        "ispublic": $elem.ispublic,
        "isfriend": $elem.isfriend,
        "isfamily": $elem.isfamily
      }#if($foreach.hasNext),#end

#end
    ]
  }
}
```

뉴스 기사 예제(API Gateway 모델 및 매핑 템플릿)

다음 단원에서는 API Gateway에서 샘플 뉴스 기사 API에 사용할 수 있는 모델 및 매핑 템플릿의 예를 제공합니다. API Gateway의 모델 및 매핑 템플릿에 대한 자세한 내용은 PetStore 매핑 템플릿 단원을 참조하세요.

주제

- 원본 데이터(뉴스 기사 예)
- 입력 모델(뉴스 기사 예)
- 입력 매핑 템플릿(뉴스 기사 예)
- 변환된 데이터(뉴스 기사 예)
- 출력 모델(뉴스 기사 예)
- 출력 매핑 템플릿(뉴스 기사 예)

원본 데이터(뉴스 기사 예)

다음은 뉴스 기사 예에 대한 원본 JSON 데이터입니다.

```
{
```

```
  "count": 1,
  "items": [
    {
      "last_updated_date": "2015-04-24",
      "expire_date": "2016-04-25",
      "author_first_name": "John",
      "description": "Sample Description",
      "creation_date": "2015-04-20",
      "title": "Sample Title",
      "allow_comment": "1",
      "author": {
        "last_name": "Doe",
        "email": "johndoe@example.com",
        "first_name": "John"
      },
      "body": "Sample Body",
      "publish_date": "2015-04-25",
      "version": "1",
      "author_last_name": "Doe",
      "parent_id": 2345678901,
      "article_url": "http://www.example.com/articles/3456789012"
    }
  ],
  "version": 1
}
```

입력 모델(뉴스 기사 예)

다음은 뉴스 기사 예에 대한 원본 JSON 데이터에 해당하는 입력 모델입니다.

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "title": "NewsArticleInputModel",
  "type": "object",
  "properties": {
    "count": { "type": "integer" },
    "items": {
      "type": "array",
      "items": {
        "type": "object",
        "properties": {
          "last_updated_date": { "type": "string" },
          "expire_date": { "type": "string" },
          "author_first_name": { "type": "string" },
```

```
          "description": { "type": "string" },
          "creation_date": { "type": "string" },
          "title": { "type": "string" },
          "allow_comment": { "type": "string" },
          "author": {
            "type": "object",
            "properties": {
              "last_name": { "type": "string" },
              "email": { "type": "string" },
              "first_name": { "type": "string" }
            }
          },
          "body": { "type": "string" },
          "publish_date": { "type": "string" },
          "version": { "type": "string" },
          "author_last_name": { "type": "string" },
          "parent_id": { "type": "integer" },
          "article_url": { "type": "string" }
        }
      }
    },
    "version": { "type": "integer" }
  }
}
```

입력 매핑 템플릿(뉴스 기사 예)

다음은 뉴스 기사 예에 대한 원본 JSON 데이터에 해당하는 입력 매핑 템플릿입니다.

```
#set($inputRoot = $input.path('$'))
{
  "count": $inputRoot.count,
  "items": [
#foreach($elem in $inputRoot.items)
    {
      "last_updated_date": "$elem.last_updated_date",
      "expire_date": "$elem.expire_date",
      "author_first_name": "$elem.author_first_name",
      "description": "$elem.description",
      "creation_date": "$elem.creation_date",
      "title": "$elem.title",
      "allow_comment": "$elem.allow_comment",
      "author": {
        "last_name": "$elem.author.last_name",
```

```
        "email": "$elem.author.email",
        "first_name": "$elem.author.first_name"
      },
      "body": "$elem.body",
      "publish_date": "$elem.publish_date",
      "version": "$elem.version",
      "author_last_name": "$elem.author_last_name",
      "parent_id": $elem.parent_id,
      "article_url": "$elem.article_url"
    }#if($foreach.hasNext),#end

#end
  ],
  "version": $inputRoot.version
}
```

변환된 데이터(뉴스 기사 예)

다음은 원본 뉴스 기사 예제의 JSON 데이터를 출력용으로 변환하는 방법의 예입니다.

```
{
  "count": 1,
  "items": [
    {
      "creation_date": "2015-04-20",
      "title": "Sample Title",
      "author": "John Doe",
      "body": "Sample Body",
      "publish_date": "2015-04-25",
      "article_url": "http://www.example.com/articles/3456789012"
    }
  ],
  "version": 1
}
```

출력 모델(뉴스 기사 예)

다음은 변환된 JSON 데이터 형식에 해당하는 출력 모델입니다.

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "title": "NewsArticleOutputModel",
  "type": "object",
```

```
  "properties": {
    "count": { "type": "integer" },
    "items": {
      "type": "array",
      "items": {
        "type": "object",
        "properties": {
          "creation_date": { "type": "string" },
          "title": { "type": "string" },
          "author": { "type": "string" },
          "body": { "type": "string" },
          "publish_date": { "type": "string" },
          "article_url": { "type": "string" }
        }
      }
    },
    "version": { "type": "integer" }
  }
}
```

출력 매핑 템플릿(뉴스 기사 예)

다음은 변환된 JSON 데이터 형식에 해당하는 출력 매핑 템플릿입니다. 여기서 템플릿 변수는 변환된 JSON 데이터 형식이 아니라 원본 형식을 기반으로 합니다.

```
#set($inputRoot = $input.path('$'))
{
  "count": $inputRoot.count,
  "items": [
#foreach($elem in $inputRoot.items)
    {
      "creation_date": "$elem.creation_date",
      "title": "$elem.title",
      "author": "$elem.author.first_name $elem.author.last_name",
      "body": "$elem.body",
      "publish_date": "$elem.publish_date",
      "article_url": "$elem.article_url"
    }#if($foreach.hasNext),#end

#end
  ],
  "version": $inputRoot.version
}
```

판매 송장 예(API Gateway 모델 및 매핑 템플릿)

다음 단원에서는 API Gateway에서 샘플 판매 송장 API에 사용할 수 있는 모델 및 매핑 템플릿의 예를 제공합니다. API Gateway의 모델 및 매핑 템플릿에 대한 자세한 내용은 PetStore 매핑 템플릿 단원을 참조하세요

주제

- 원본 데이터(판매 송장 예)
- 입력 모델(판매 송장 예)
- 입력 매핑 템플릿(판매 송장 예)
- 변환된 데이터(판매 송장 예)
- 출력 모델(판매 송장 예)
- 출력 매핑 템플릿(판매 송장 예)

원본 데이터(판매 송장 예)

다음은 판매 송장 예에 대한 원본 JSON 데이터입니다.

```
{
  "DueDate": "2013-02-15",
  "Balance": 1990.19,
  "DocNumber": "SAMP001",
  "Status": "Payable",
  "Line": [
    {
      "Description": "Sample Expense",
      "Amount": 500,
      "DetailType": "ExpenseDetail",
      "ExpenseDetail": {
        "Customer": {
          "value": "ABC123",
          "name": "Sample Customer"
        },
        "Ref": {
          "value": "DEF234",
          "name": "Sample Construction"
        },
        "Account": {
          "value": "EFG345",
          "name": "Fuel"
```

```
        },
        "LineStatus": "Billable"
      }
    }
  ],
  "Vendor": {
    "value": "GHI456",
    "name": "Sample Bank"
  },
  "APRef": {
    "value": "HIJ567",
    "name": "Accounts Payable"
  },
  "TotalAmt": 1990.19
}
```

입력 모델(판매 송장 예)

다음은 판매 송장 예에 대한 원본 JSON 데이터에 해당하는 입력 모델입니다.

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "title": "InvoiceInputModel",
  "type": "object",
  "properties": {
    "DueDate": { "type": "string" },
    "Balance": { "type": "number" },
    "DocNumber": { "type": "string" },
    "Status": { "type": "string" },
    "Line": {
      "type": "array",
      "items": {
        "type": "object",
        "properties": {
          "Description": { "type": "string" },
          "Amount": { "type": "integer" },
          "DetailType": { "type": "string" },
          "ExpenseDetail": {
            "type": "object",
            "properties": {
              "Customer": {
                "type": "object",
                "properties": {
                  "value": { "type": "string" },
```

```
                  "name": { "type": "string" }
                }
              },
              "Ref": {
                "type": "object",
                "properties": {
                  "value": { "type": "string" },
                  "name": { "type": "string" }
                }
              },
              "Account": {
                "type": "object",
                "properties": {
                  "value": { "type": "string" },
                  "name": { "type": "string" }
                }
              },
              "LineStatus": { "type": "string" }
            }
          }
        }
      }
    },
    "Vendor": {
      "type": "object",
      "properties": {
        "value": { "type": "string" },
        "name": { "type": "string" }
      }
    },
    "APRef": {
      "type": "object",
      "properties": {
        "value": { "type": "string" },
        "name": { "type": "string" }
      }
    },
    "TotalAmt": { "type": "number" }
  }
}
```

입력 매핑 템플릿(판매 송장 예)

다음은 판매 송장 예에 대한 원본 JSON 데이터에 해당하는 입력 매핑 템플릿입니다.

```
#set($inputRoot = $input.path('$'))
{
  "DueDate": "$inputRoot.DueDate",
  "Balance": $inputRoot.Balance,
  "DocNumber": "$inputRoot.DocNumber",
  "Status": "$inputRoot.Status",
  "Line": [
#foreach($elem in $inputRoot.Line)
    {
      "Description": "$elem.Description",
      "Amount": $elem.Amount,
      "DetailType": "$elem.DetailType",
      "ExpenseDetail": {
        "Customer": {
          "value": "$elem.ExpenseDetail.Customer.value",
          "name": "$elem.ExpenseDetail.Customer.name"
        },
        "Ref": {
          "value": "$elem.ExpenseDetail.Ref.value",
          "name": "$elem.ExpenseDetail.Ref.name"
        },
        "Account": {
          "value": "$elem.ExpenseDetail.Account.value",
          "name": "$elem.ExpenseDetail.Account.name"
        },
        "LineStatus": "$elem.ExpenseDetail.LineStatus"
      }
    }#if($foreach.hasNext),#end

#end
  ],
  "Vendor": {
    "value": "$inputRoot.Vendor.value",
    "name": "$inputRoot.Vendor.name"
  },
  "APRef": {
    "value": "$inputRoot.APRef.value",
    "name": "$inputRoot.APRef.name"
  },
  "TotalAmt": $inputRoot.TotalAmt
}
```

변환된 데이터(판매 송장 예)

다음은 원본 판매 송장 예의 JSON 데이터를 출력용으로 변환될 수 있는 방법에 대한 한 가지 예입니다.

```
{
  "DueDate": "2013-02-15",
  "Balance": 1990.19,
  "DocNumber": "SAMP001",
  "Status": "Payable",
  "Line": [
    {
      "Description": "Sample Expense",
      "Amount": 500,
      "DetailType": "ExpenseDetail",
      "Customer": "ABC123 (Sample Customer)",
      "Ref": "DEF234 (Sample Construction)",
      "Account": "EFG345 (Fuel)",
      "LineStatus": "Billable"
    }
  ],
  "TotalAmt": 1990.19
}
```

출력 모델(판매 송장 예)

다음은 변환된 JSON 데이터 형식에 해당하는 출력 모델입니다.

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "title": "InvoiceOutputModel",
  "type": "object",
  "properties": {
    "DueDate": { "type": "string" },
    "Balance": { "type": "number" },
    "DocNumber": { "type": "string" },
    "Status": { "type": "string" },
    "Line": {
      "type": "array",
      "items": {
        "type": "object",
        "properties": {
          "Description": { "type": "string" },
```

```
          "Amount": { "type": "integer" },
          "DetailType": { "type": "string" },
          "Customer": { "type": "string" },
          "Ref": { "type": "string" },
          "Account": { "type": "string" },
          "LineStatus": { "type": "string" }
        }
      }
    },
    "TotalAmt": { "type": "number" }
  }
}
```

출력 매핑 템플릿(판매 송장 예)

다음은 변환된 JSON 데이터 형식에 해당하는 출력 매핑 템플릿입니다. 여기서 템플릿 변수는 변환된 JSON 데이터 형식이 아니라 원본 형식을 기반으로 합니다.

```
#set($inputRoot = $input.path('$'))
{
  "DueDate": "$inputRoot.DueDate",
  "Balance": $inputRoot.Balance,
  "DocNumber": "$inputRoot.DocNumber",
  "Status": "$inputRoot.Status",
  "Line": [
#foreach($elem in $inputRoot.Line)
    {
      "Description": "$elem.Description",
      "Amount": $elem.Amount,
      "DetailType": "$elem.DetailType",
      "Customer": "$elem.ExpenseDetail.Customer.value
 ($elem.ExpenseDetail.Customer.name)",
      "Ref": "$elem.ExpenseDetail.Ref.value ($elem.ExpenseDetail.Ref.name)",
      "Account": "$elem.ExpenseDetail.Account.value
 ($elem.ExpenseDetail.Account.name)",
      "LineStatus": "$elem.ExpenseDetail.LineStatus"
    }#if($foreach.hasNext),#end

#end
  ],
  "TotalAmt": $inputRoot.TotalAmt
}
```

직원 레코드 예(API Gateway 모델 및 매핑 템플릿)

다음 단원에서는 API Gateway에서 샘플 직원 레코드 API에 사용할 수 있는 모델 및 매핑 템플릿의 예를 제공합니다. API Gateway의 모델 및 매핑 템플릿에 대한 자세한 내용은 PetStore 매핑 템플릿 단원을 참조하세요.

주제

- 원본 데이터(직원 레코드 예)
- 입력 모델(직원 레코드 예)
- 입력 매핑 템플릿(직원 레코드 예)
- 변환된 데이터(직원 레코드 예)
- 출력 모델(직원 레코드 예)
- 출력 매핑 템플릿(직원 레코드 예)

원본 데이터(직원 레코드 예)

다음은 직원 레코드 예에 대한 원본 JSON 데이터입니다.

```
{
  "QueryResponse": {
    "maxResults": "1",
    "startPosition": "1",
    "Employee": {
      "Organization": "false",
      "Title": "Mrs.",
      "GivenName": "Jane",
      "MiddleName": "Lane",
      "FamilyName": "Doe",
      "DisplayName": "Jane Lane Doe",
      "PrintOnCheckName": "Jane Lane Doe",
      "Active": "true",
      "PrimaryPhone": { "FreeFormNumber": "505.555.9999" },
      "PrimaryEmailAddr": { "Address": "janedoe@example.com" },
      "EmployeeType": "Regular",
      "status": "Synchronized",
      "Id": "ABC123",
      "SyncToken": "1",
      "MetaData": {
        "CreateTime": "2015-04-26T19:45:03Z",
        "LastUpdatedTime": "2015-04-27T21:48:23Z"
```

```
      },
      "PrimaryAddr": {
        "Line1": "123 Any Street",
        "City": "Any City",
        "CountrySubDivisionCode": "WA",
        "PostalCode": "01234"
      }
    }
  },
  "time": "2015-04-27T22:12:32.012Z"
}
```

입력 모델(직원 레코드 예)

다음은 직원 레코드 예에 대한 원본 JSON 데이터에 해당하는 입력 모델입니다.

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "title": "EmployeeInputModel",
  "type": "object",
  "properties": {
    "QueryResponse": {
      "type": "object",
      "properties": {
        "maxResults": { "type": "string" },
        "startPosition": { "type": "string" },
        "Employee": {
          "type": "object",
          "properties": {
            "Organization": { "type": "string" },
            "Title": { "type": "string" },
            "GivenName": { "type": "string" },
            "MiddleName": { "type": "string" },
            "FamilyName": { "type": "string" },
            "DisplayName": { "type": "string" },
            "PrintOnCheckName": { "type": "string" },
            "Active": { "type": "string" },
            "PrimaryPhone": {
              "type": "object",
              "properties": {
                "FreeFormNumber": { "type": "string" }
              }
            },
            "PrimaryEmailAddr": {
```

```
              "type": "object",
              "properties": {
                "Address": { "type": "string" }
              }
            },
            "EmployeeType": { "type": "string" },
            "status": { "type": "string" },
            "Id": { "type": "string" },
            "SyncToken": { "type": "string" },
            "MetaData": {
              "type": "object",
              "properties": {
                "CreateTime": { "type": "string" },
                "LastUpdatedTime": { "type": "string" }
              }
            },
            "PrimaryAddr": {
              "type": "object",
              "properties": {
                "Line1": { "type": "string" },
                "City": { "type": "string" },
                "CountrySubDivisionCode": { "type": "string" },
                "PostalCode": { "type": "string" }
              }
            }
          }
        }
      }
    },
    "time": { "type": "string" }
  }
}
```

입력 매핑 템플릿(직원 레코드 예)

다음은 직원 레코드 예에 대한 원본 JSON 데이터에 해당하는 입력 매핑 템플릿입니다.

```
#set($inputRoot = $input.path('$'))
{
  "QueryResponse": {
    "maxResults": "$inputRoot.QueryResponse.maxResults",
    "startPosition": "$inputRoot.QueryResponse.startPosition",
    "Employee": {
      "Organization": "$inputRoot.QueryResponse.Employee.Organization",
```

```
      "Title": "$inputRoot.QueryResponse.Employee.Title",
      "GivenName": "$inputRoot.QueryResponse.Employee.GivenName",
      "MiddleName": "$inputRoot.QueryResponse.Employee.MiddleName",
      "FamilyName": "$inputRoot.QueryResponse.Employee.FamilyName",
      "DisplayName": "$inputRoot.QueryResponse.Employee.DisplayName",
      "PrintOnCheckName": "$inputRoot.QueryResponse.Employee.PrintOnCheckName",
      "Active": "$inputRoot.QueryResponse.Employee.Active",
      "PrimaryPhone": { "FreeFormNumber":
 "$inputRoot.QueryResponse.Employee.PrimaryPhone.FreeFormNumber" },
      "PrimaryEmailAddr": { "Address":
 "$inputRoot.QueryResponse.Employee.PrimaryEmailAddr.Address" },
      "EmployeeType": "$inputRoot.QueryResponse.Employee.EmployeeType",
      "status": "$inputRoot.QueryResponse.Employee.status",
      "Id": "$inputRoot.QueryResponse.Employee.Id",
      "SyncToken": "$inputRoot.QueryResponse.Employee.SyncToken",
      "MetaData": {
        "CreateTime": "$inputRoot.QueryResponse.Employee.MetaData.CreateTime",
        "LastUpdatedTime": "$inputRoot.QueryResponse.Employee.MetaData.LastUpdatedTime"
      },
      "PrimaryAddr" : {
        "Line1": "$inputRoot.QueryResponse.Employee.PrimaryAddr.Line1",
        "City": "$inputRoot.QueryResponse.Employee.PrimaryAddr.City",
        "CountrySubDivisionCode":
 "$inputRoot.QueryResponse.Employee.PrimaryAddr.CountrySubDivisionCode",
        "PostalCode": "$inputRoot.QueryResponse.Employee.PrimaryAddr.PostalCode"
      }
    }
  },
  "time": "$inputRoot.time"
}
```

변환된 데이터(직원 레코드 예)

다음은 원본 직원 레코드의 JSON 데이터를 출력용으로 변환하는 한 가지 방법입니다.

```
{
  "QueryResponse": {
    "maxResults": "1",
    "startPosition": "1",
    "Employees": [
      {
        "Title": "Mrs.",
        "GivenName": "Jane",
        "MiddleName": "Lane",
```

```
        "FamilyName": "Doe",
        "DisplayName": "Jane Lane Doe",
        "PrintOnCheckName": "Jane Lane Doe",
        "Active": "true",
        "PrimaryPhone": "505.555.9999",
        "Email": [
          {
            "type": "primary",
            "Address": "janedoe@example.com"
          }
        ],
        "EmployeeType": "Regular",
        "PrimaryAddr": {
          "Line1": "123 Any Street",
          "City": "Any City",
          "CountrySubDivisionCode": "WA",
          "PostalCode": "01234"
        }
      }
    ]
  },
  "time": "2015-04-27T22:12:32.012Z"
}
```

출력 모델(직원 레코드 예)

다음은 변환된 JSON 데이터 형식에 해당하는 출력 모델입니다.

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "title": "EmployeeOutputModel",
  "type": "object",
  "properties": {
    "QueryResponse": {
      "type": "object",
      "properties": {
        "maxResults": { "type": "string" },
        "startPosition": { "type": "string" },
        "Employees": {
          "type": "array",
          "items": {
            "type": "object",
            "properties": {
              "Title": { "type": "string" },
```

```
              "GivenName": { "type": "string" },
              "MiddleName": { "type": "string" },
              "FamilyName": { "type": "string" },
              "DisplayName": { "type": "string" },
              "PrintOnCheckName": { "type": "string" },
              "Active": { "type": "string" },
              "PrimaryPhone": { "type": "string" },
              "Email": {
                "type": "array",
                "items": {
                  "type": "object",
                  "properties": {
                    "type": { "type": "string" },
                    "Address": { "type": "string" }
                  }
                }
              },
              "EmployeeType": { "type": "string" },
              "PrimaryAddr": {
                "type": "object",
                "properties": {
                  "Line1": {"type": "string" },
                  "City": { "type": "string" },
                  "CountrySubDivisionCode": { "type": "string" },
                  "PostalCode": { "type": "string" }
                }
              }
            }
          }
        }
      }
    },
    "time": { "type": "string" }
  }
}
```

출력 매핑 템플릿(직원 레코드 예)

다음은 변환된 JSON 데이터 형식에 해당하는 출력 매핑 템플릿입니다. 여기서 템플릿 변수는 변환된 JSON 데이터 형식이 아니라 원본 형식을 기반으로 합니다.

```
#set($inputRoot = $input.path('$'))
{
```

```
  "QueryResponse": {
    "maxResults": "$inputRoot.QueryResponse.maxResults",
    "startPosition": "$inputRoot.QueryResponse.startPosition",
    "Employees": [
      {
        "Title": "$inputRoot.QueryResponse.Employee.Title",
        "GivenName": "$inputRoot.QueryResponse.Employee.GivenName",
        "MiddleName": "$inputRoot.QueryResponse.Employee.MiddleName",
        "FamilyName": "$inputRoot.QueryResponse.Employee.FamilyName",
        "DisplayName": "$inputRoot.QueryResponse.Employee.DisplayName",
        "PrintOnCheckName": "$inputRoot.QueryResponse.Employee.PrintOnCheckName",
        "Active": "$inputRoot.QueryResponse.Employee.Active",
        "PrimaryPhone":
 "$inputRoot.QueryResponse.Employee.PrimaryPhone.FreeFormNumber",
        "Email" : [
          {
            "type": "primary",
            "Address": "$inputRoot.QueryResponse.Employee.PrimaryEmailAddr.Address"
          }
        ],
        "EmployeeType": "$inputRoot.QueryResponse.Employee.EmployeeType",
        "PrimaryAddr": {
          "Line1": "$inputRoot.QueryResponse.Employee.PrimaryAddr.Line1",
          "City": "$inputRoot.QueryResponse.Employee.PrimaryAddr.City",
          "CountrySubDivisionCode":
 "$inputRoot.QueryResponse.Employee.PrimaryAddr.CountrySubDivisionCode",
          "PostalCode": "$inputRoot.QueryResponse.Employee.PrimaryAddr.PostalCode"
        }
      }
    ]
  },
  "time": "$inputRoot.time"
}
```

Amazon API Gateway API 요청 및 응답 데이터 매핑 참조

이 섹션에서는 `context`, `stage` 또는 `util` 변수에 저장된 다른 데이터를 포함한 API의 메서드 요청 데이터에서 해당 통합 요청 파라미터로, 그리고 다른 데이터를 포함한 통합 응답 데이터에서 메서드 응답 파라미터로의 데이터 매핑을 설정하는 방법을 설명합니다. 메서드 요청 데이터는 요청 파라미터(경로, 쿼리 문자열, 헤더)와 본문을 포함합니다. 통합 응답 데이터는 응답 파라미터(헤더)와 본문을 포함합니다. 단계 변수 사용에 대한 자세한 내용은 Amazon API Gateway 단계 변수 참조 단원을 참조하십시오.

주제

- 통합 요청 파라미터에 메서드 요청 데이터 매핑
- 메서드 응답 헤더에 통합 응답 데이터 매핑
- 메서드와 통합 간의 요청 및 응답 페이로드 매핑
- 통합 패스스루 동작

통합 요청 파라미터에 메서드 요청 데이터 매핑

경로 변수 형식의 통합 요청 파라미터, 쿼리 문자열 또는 헤더를 정의된 메서드 요청 파라미터 및 페이로드에서 매핑할 수 있습니다.

다음 표에서 *PARAM_NAME*은 지정된 파라미터 유형의 메서드 요청 파라미터 이름입니다. 정규식 `'^[a-zA-Z0-9._$-]+$]'`와 일치해야 합니다. 이 파라미터가 이미 정의되어 있어야 참조할 수 있습니다. *JSONPath_EXPRESSION*은 요청 또는 응답 본문의 JSON 필드에 대한 JSONPath 표현식입니다.

> **Note**
>
> "$" 접두사는 이 구문에서 생략됩니다.

통합 요청 데이터 매핑 표현식

매핑된 데이터 원본	매핑 표현식
메서드 요청 경로	`method.request.path.` *PARAM_NAME*
메서드 요청 쿼리 문자열	`method.request.querystring.` *PARAM_NAME*
다중 값 메서드 요청 쿼리 문자열	`method.request.multivaluequerystring.` *PARAM_NAME*
메서드 요청 헤더	`method.request.header.` *PARAM_NAME*
다중 값 메서드 요청 헤더	`method.request.multivalueheader.` *PARAM_NAME*
메서드 요청 본문	`method.request.body`

매핑된 데이터 원본	매핑 표현식
메서드 요청 본문 () JsonPath	`method.request.body.` *JSONPath_EXPRESSION* .
단계 변수	`stageVariables.` *VARIABLE_NAME*
컨텍스트 변수	`context.`*VARIABLE_NAME* 은 지원되는 컨텍스트 변수 중 하나여야 합니다.
정적 값	*'STATIC_VALUE'* . *STATIC_VALUE* 는 문자열 리터럴이며 작은따옴표 쌍으로 묶여야 합니다.

Example OpenAPI의 메서드 요청 파라미터에서 매핑

다음 예제에서는 다음과 같이 매핑하는 OpenAPI 코드 조각을 보여줍니다.

- `methodRequestHeaderParam`라는 메서드 요청 헤더를 `integrationPathParam`라는 통합 요청 경로 파라미터로 매핑합니다.
- `methodRequestQueryParam`라는 다중 값 메서드 요청 쿼리 문자열은 `integrationQueryParam`라는 통합 요청 쿼리 문자열에 매핑됩니다.

```
...
"requestParameters" : {

    "integration.request.path.integrationPathParam" :
 "method.request.header.methodRequestHeaderParam",
    "integration.request.querystring.integrationQueryParam" :
 "method.request.multivaluequerystring.methodRequestQueryParam"

}
...
```

JSONPath 표현식을 사용하여 JSON 요청 본문의 필드에서 통합 요청 파라미터를 매핑할 수도 있습니다. 다음 표에서는 메서드 요청 본문 및 JSON 필드에 대한 매핑 표현식을 보여줍니다.

Example OpenAPI의 메서드 요청 본문에서 매핑

다음 예에서는 1) 메서드 요청 본문을 body-header라는 통합 요청 헤더에 매핑하고, 2) JSON 표현식(petstore.pets[0].name, $. 접두사 사용 안 함)으로 표시된 본문의 JSON 필드를 매핑하는 OpenAPI 코드 조각을 보여줍니다.

```
...
"requestParameters" : {

    "integration.request.header.body-header" : "method.request.body",
    "integration.request.path.pet-name" : "method.request.body.petstore.pets[0].name",

}
...
```

메서드 응답 헤더에 통합 응답 데이터 매핑

통합 응답 헤더, 통합 응답 본문, $context 변수 또는 정적 값에서 메서드 응답 헤더 파라미터를 매핑할 수 있습니다.

메서드 응답 헤더 매핑 표현식

매핑된 데이터 원본	매핑 표현식
통합 응답 헤더	integration.response.header.*PARAM_NAME*
통합 응답 헤더	integration.response.multivalueheader.*PARAM_NAME*
통합 응답 본문	integration.response.body
통합 응답 본문 (JsonPath)	integration.response.body.*JSONPath_EXPRESSION*
단계 변수	stageVariables.*VARIABLE_NAME*
컨텍스트 변수	context.*VARIABLE_NAME* 은 지원되는 컨텍스트 변수 중 하나여야 합니다.

매핑된 데이터 원본	매핑 표현식
정적 값	*'STATIC_VALUE'* . *STATIC_VALUE* 는 문자열 리터럴이며 작은따옴표 쌍으로 묶여야 합니다.

Example OpenAPI의 통합 응답에서 데이터 매핑

다음 예에서는 1) 통합 응답의 `redirect.url` JSONPath 필드에서 요청 응답의 `location` 헤더로 매핑하고, 2) 통합 응답의 `x-app-id` 헤더에서 메서드 응답의 `id` 헤더로 매핑하는 OpenAPI 코드 조각을 보여줍니다.

```
...
"responseParameters" : {

    "method.response.header.location" : "integration.response.body.redirect.url",
    "method.response.header.id" : "integration.response.header.x-app-id",
    "method.response.header.items" : "integration.response.multivalueheader.item",

}
...
```

메서드와 통합 간의 요청 및 응답 페이로드 매핑

API Gateway에서는 Velocity Template Language(VTL) 엔진을 사용하여 통합 요청 및 통합 응답에 대한 본문 매핑 템플릿을 처리합니다. 매핑 템플릿은 메서드 요청 페이로드를 해당 통합 요청 페이로드로 변환하고 통합 응답 본문을 메서드 응답 본문으로 변환합니다.

VTL 템플릿은 JSONPath 표현식, 기타 파라미터(예: 호출 컨텍스트, 단계 변수) 및 유틸리티 함수를 사용하여 JSON 데이터를 처리합니다.

페이로드의 데이터 구조를 설명하도록 모델을 정의한 경우 API Gateway에서는 모델을 사용하여 통합 요청 또는 통합 응답에 대한 스켈레톤 매핑 템플릿을 생성할 수 있습니다. 스켈레톤 템플릿을 보조 수단으로 사용하여 매핑 VTL 스크립트를 사용자 지정하고 확장할 수 있습니다. 페이로드의 데이터 구조에 대한 모델을 정의하지 않고 매핑 템플릿을 처음부터 생성할 수 있습니다.

VTL 매핑 템플릿 선택

API Gateway에서는 다음 로직을 사용하여 매핑 템플릿을 선택하고 VTL(Velocity Template Language)에서 메서드 요청에서 해당 통합 요청으로 또는 통합 응답에서 해당 메서드 응답으로 페이로드를 매핑합니다.

요청 페이로드의 경우 API Gateway에서는 요청의 `Content-Type` 헤더 값을 키로 사용하여 요청 페이로드에 대한 매핑 템플릿을 선택합니다. 응답 페이로드의 경우 API Gateway에서는 수신 요청의 `Accept` 헤더 값을 키로 사용하여 매핑 템플릿을 선택합니다.

`Content-Type` 헤더가 요청에 없는 경우 API Gateway에서는 기본값이 `application/json`인 것으로 가정합니다. 그런 요청의 경우 API Gateway에서는 `application/json`을 기본 키로 사용하여 매핑 템플릿을 선택합니다(정의되어 있는 경우). 이 키와 일치하는 템플릿이 없는 경우 API Gateway에서는 매핑되지 않은 템플릿을 통해 페이로드를 전달합니다(passthroughBehavior 속성이 `WHEN_NO_MATCH` 또는 `WHEN_NO_TEMPLATES`로 설정되어 있는 경우).

`Accept` 헤더가 요청에 지정되어 있지 않은 경우 API Gateway에서는 기본값이 `application/json`인 것으로 가정합니다. 이 경우 API Gateway에서는 `application/json`에 대한 기존 매핑 템플릿을 선택하여 응답 페이로드를 매핑합니다. `application/json`에 대한 템플릿이 정의되어 있지 않은 경우 API Gateway에서는 첫 번째 기존 템플릿을 선택한 후 이 템플릿을 기본값으로 사용하여 응답 페이로드를 매핑합니다. 마찬가지로 API Gateway에서는 지정된 `Accept` 헤더 값이 기존 템플릿 키와 일치하지 않을 경우 첫 번째 기존 템플릿을 사용합니다. 템플릿이 정의되어 있지 않은 경우 API Gateway에서는 매핑되지 않은 템플릿을 통해 응답 페이로드를 전달합니다.

예를 들어 API에 요청 페이로드에 대해 정의된 `application/json` 템플릿과 응답 페이로드에 대해 정의된 `application/xml` 템플릿이 있다고 가정합니다. 클라이언트가 요청에서 `"Content-Type : application/json"` 및 `"Accept : application/xml"` 헤더를 설정한 경우 요청 페이로드와 응답 페이로드는 모두 해당 매핑 템플릿에 따라 처리됩니다. `Accept:application/xml` 헤더가 없는 경우 `application/xml` 매핑 템플릿을 사용하여 응답 페이로드를 매핑합니다. 매핑되지 않은 응답 페이로드를 대신 반환하려면 `application/json`에 대해 빈 템플릿을 설정해야 합니다.

MIME 유형은 매핑 템플릿을 선택할 때 `Accept` 및 `Content-Type` 헤더에서만 사용됩니다. 예를 들어 `"Content-Type: application/json; charset=UTF-8"` 헤더에는 `application/json` 키가 선택된 요청 템플릿이 있습니다.

통합 패스스루 동작

비 프록시 통합의 경우, 메서드 요청에서 페이로드를 전달하고 `Content-Type` 헤더가 지정된 매핑 템플릿과 일치하지 않거나 매핑 템플릿이 정의되지 않은 경우 클라이언트가 제공한 요청 페이로드를 변

환 없이 통합 요청을 통해 백엔드로 전달되도록 선택할 수 있습니다. 이 프로세스를 통합 패스스루라고 합니다.

프록시 통합을 위해 API Gateway는 백엔드로 전체 요청을 패스스루하며, 패스스루 동작을 수정할 수 있는 옵션은 없습니다.

수신되는 요청의 실제 패스스루 동작은 통합 요청 설정 중에 지정된 매핑 템플릿에 대해 선택한 옵션과 수신되는 요청에 클라이언트가 설정한 Content Type 헤더에 따라 결정됩니다. 여기에는 다음과 같은 세 가지 옵션이 있습니다.

요청 Content-Type 헤더와 일치하는 템플릿이 없는 경우

메서드 요청 콘텐츠 유형이 매핑 템플릿과 연결된 콘텐츠 유형 중 일치하지 않는 경우 메서드 요청 본문이 변환 없이 통합 요청을 지나 백엔드에 전달되도록 하려면 이 옵션을 선택합니다.

API Gateway API를 호출하는 경우 이 옵션을 선택하려면 통합에서 `WHEN_NO_MATCH`를 `passthroughBehavior` 속성 값으로 설정하면 됩니다.

정의된 템플릿이 없는 경우(권장)

매핑 템플릿이 통합 요청에 정의되어 있지 않을 때 메서드 요청 본문이 변환 없이 통합 요청을 지나 벡엔드에 전달되도록 하려면 이 옵션을 선택합니다. 이 옵션을 선택할 때 템플릿이 정의되어 있는 경우 매핑되지 않은 콘텐츠 유형의 메서드 요청은 HTTP 415 미지원 미디어 유형 응답과 함께 거부됩니다.

API Gateway API를 호출하는 경우 이 옵션을 선택하려면 통합에서 `WHEN_NO_TEMPLATES`를 `passthroughBehavior` 속성 값으로 설정하면 됩니다.

없음

매핑 템플릿이 통합 요청에 정의되어 있지 않을 때 메서드 요청 본문이 변환 없이 통합 요청을 지나 벡엔드에 전달되지 않도록 하려면 이 옵션을 선택합니다. 이 옵션을 선택할 때 템플릿이 정의되어 있는 경우 매핑되지 않은 콘텐츠 유형의 메서드 요청은 HTTP 415 미지원 미디어 유형 응답과 함께 거부됩니다.

API Gateway API를 호출하는 경우 이 옵션을 선택하려면 통합에서 `NEVER`를 `passthroughBehavior` 속성 값으로 설정하면 됩니다.

다음 예제에서는 가능한 패스스루 동작을 설명합니다.

예제 1: 하나의 매핑 템플릿이 `application/json` 콘텐츠 유형에 대한 통합 요청에 정의되어 있습니다.

Content-type 헤더\선택된 패스스루 옵션	**WHEN_NO_MATCH**	**WHEN_NO_TEMPLATES**	**NEVER**
없음(기본값: application/json)	템플릿을 사용하여 요청 페이로드가 변환됩니다.	템플릿을 사용하여 요청 페이로드가 변환됩니다.	템플릿을 사용하여 요청 페이로드가 변환됩니다.
application/json	템플릿을 사용하여 요청 페이로드가 변환됩니다.	템플릿을 사용하여 요청 페이로드가 변환됩니다.	템플릿을 사용하여 요청 페이로드가 변환됩니다.
application/xml	요청 페이로드가 변환되지 않고 백엔드로 그대로 전송됩니다.	요청이 HTTP 415 Unsupported Media Type 응답과 함께 거부됩니다.	요청이 HTTP 415 Unsupported Media Type 응답과 함께 거부됩니다.

예제 2: 하나의 매핑 템플릿이 application/xml 콘텐츠 유형에 대한 통합 요청에 정의되어 있습니다.

Content-type 헤더\선택된 패스스루 옵션	**WHEN_NO_MATCH**	**WHEN_NO_TEMPLATES**	**NEVER**
없음(기본값: application/json)	요청 페이로드가 변환되지 않고 백엔드로 그대로 전송됩니다.	요청이 HTTP 415 Unsupported Media Type 응답과 함께 거부됩니다.	요청이 HTTP 415 Unsupported Media Type 응답과 함께 거부됩니다.
application/json	요청 페이로드가 변환되지 않고 백엔드로 그대로 전송됩니다.	요청이 HTTP 415 Unsupported Media Type 응답과 함께 거부됩니다.	요청이 HTTP 415 Unsupported Media Type 응답과 함께 거부됩니다.
application/xml	템플릿을 사용하여 요청 페이로드가 변환됩니다.	템플릿을 사용하여 요청 페이로드가 변환됩니다.	템플릿을 사용하여 요청 페이로드가 변환됩니다.

API Gateway 매핑 템플릿과 액세스 로깅 변수 참조

이 섹션에서는 Amazon API Gateway에서 데이터 모델, 권한 부여자, 매핑 템플릿 및 CloudWatch 액세스 로깅과 함께 사용하기 위해 정의하는 변수 및 함수에 대한 참조 정보를 제공합니다. 이러한 변수와 함수의 자세한 사용 방법은 매핑 템플릿 이해 단원을 참조하십시오. VTL(Velocity Template Language)에 대한 자세한 내용은 VTL 참조를 참조하세요.

주제

- $context데이터 모델, 권한 부여자, 매핑 템플릿, 액세스 로깅용 변수 CloudWatch
- $context 변수 템플릿 예제
- 액세스 로깅 전용의 $context 변수
- $input 변수
- $input 변수 템플릿 예제
- $stageVariables
- $util 변수

> **Note**
>
> `$method` 변수와 `$integration` 변수의 경우 the section called “요청 및 응답 데이터 매핑 참조” 단원을 참조하십시오.

`$context`데이터 모델, 권한 부여자, 매핑 템플릿, 액세스 로깅용 변수 CloudWatch

다음 `$context` 변수는 데이터 모델, 권한 부여자, 매핑 템플릿 및 액세스 로깅에 사용할 수 있습니다. CloudWatch

CloudWatch 액세스 로깅에만 사용할 수 있는 `$context` 변수에 대한 내용은 을 참조하십시오. the section called “액세스 로깅 전용의 `$context` 변수”

파라미터	설명
`$context.accountId`	API 소유자의 AWS 계정 ID.
`$context.apiId`	식별자 API Gateway가 API에 할당합니다.

파라미터	설명
`$context.authorizer.claims.`*`property`*	메서드 호출자가 성공적으로 인증된 후 Amazon Cognito 사용자 풀에서 반환된 클레임의 속성입니다. 자세한 내용은 the section called “REST API에 대한 권한 부여자로 Amazon Cognito 사용자 풀 사용” 단원을 참조하세요. **Note** `$context.authorizer.claims` 를 호출하면 null이 반환됩니다.
`$context.authorizer.principalId`	클라이언트가 전송한 토큰과 연결되고 API Gateway Lambda 권한 부여자(이전에는 사용자 지정 권한 부여자라고 함)에서 반환한 보안 주체 사용자 식별입니다. 자세한 설명은 the section called “Lambda 권한 부여자 사용” 섹션을 참조하세요.

파라미터	설명
`$context.authorizer.` *property*	API Gateway Lambda 권한 부여자 함수에서 반환된 `context` 맵의 지정된 키-값 페어의 문자열화된 값입니다. 예를 들어, 권한 부여자는 다음 `context` 맵을 반환합니다. `"context" : { "key": "value", "numKey": 1, "boolKey": true }` `$context.authorizer.key` 를 호출하면 `"value"` 문자열이 반환되고, `$context.authorizer.numKey` 를 호출하면 `"1"` 문자열이 반환되고, `$context.authorizer.boolKey` 를 호출하면 `"true"` 문자열이 반환됩니다. 자세한 설명은 the section called “Lambda 권한 부여자 사용” 섹션을 참조하세요.
`$context.awsEndpointRequestId`	AWS 엔드포인트의 요청 ID.
`$context.domainName`	API를 호출하는 데 사용되는 정식 도메인 이름입니다. 이 이름은 수신되는 `Host` 헤더와 동일해야 합니다.
`$context.domainPrefix`	`$context.domainName` 의 첫 번째 레이블입니다.

파라미터	설명
`$context.error.message`	API Gateway 오류 메시지가 포함된 문자열입니다. 이 변수는 Velocity Template Language 엔진에서 처리하지 않는 GatewayResponse바디 매핑 템플릿과 액세스 로깅에서 간단한 변수 대체에만 사용할 수 있습니다. 자세한 내용은 the section called “지표” 및 the section called “오류 응답을 사용자 지정하도록 게이트웨이 응답 설정” 섹션을 참조하세요.
`$context.error.messageString`	`$context.error.message` 의 따옴표 붙은 값, 즉 `"$context.error.message"` 입니다.
`$context.error.responseType`	한 유형입니다. GatewayResponse 이 변수는 Velocity Template Language 엔진에서 처리되지 않는 GatewayResponse바디 매핑 템플릿과 액세스 로깅에서 간단한 변수 대체에만 사용할 수 있습니다. 자세한 내용은 the section called “지표” 및 the section called “오류 응답을 사용자 지정하도록 게이트웨이 응답 설정” 섹션을 참조하세요.
`$context.error.validationErrorString`	세부적인 검증 오류 메시지를 포함하는 문자열입니다.
`$context.extendedRequestId`	API Gateway가 생성하여 API 요청에 할당하는 확장 ID입니다. 확장 요청 ID에는 디버깅 및 문제 해결에 유용한 정보가 포함되어 있습니다.
`$context.httpMethod`	사용된 HTTP 메서드입니다. 유효한 값에는 `DELETE`, `GET`, `HEAD`, `OPTIONS`, `PATCH`, `POST` 및 `PUT`이 있습니다.
`$context.identity.accountId`	요청과 관련된 AWS 계정 ID.

파라미터	설명
`$context.identity.apiKey`	API 키가 필요한 API 메서드의 경우, 이 변수는 메서드 요청과 연결된 API 키입니다. API 키가 필요 없는 메서드의 경우, 이 변수는 null입니다. 자세한 설명은 the section called “사용량 계획” 섹션을 참조하세요.
`$context.identity.apiKeyId`	API 키가 필요한 API 요청과 연결된 API 키 ID입니다.
`$context.identity.caller`	요청에 서명한 호출자의 보안 주체 ID입니다. IAM 권한 부여를 사용하는 라우팅에 대해 지원됩니다.
`$context.identity.cognitoAuthenticationProvider`	요청을 작성한 호출자가 사용한 Amazon Cognito 인증 공급자의 쉼표로 구분된 목록입니다. Amazon Cognito 자격 증명으로 요청을 서명한 경우에만 사용할 수 있습니다. 예를 들어, Amazon Cognito 사용자 풀의 자격 증명의 경우 `cognito-idp.`*`region`*`.amazonaws.com/`*`user_pool_id`*`,cognito-idp.`*`region`*`.amazonaws.com/`*`user_pool_id`*`:CognitoSignIn:`*`token subject claim`* 자세한 내용은 Amazon Cognito 개발자 안내서의 연동 자격 증명 사용을 참조하세요.
`$context.identity.cognitoAuthenticationType`	요청을 작성한 호출자의 Amazon Cognito 인증 유형입니다. Amazon Cognito 자격 증명으로 요청을 서명한 경우에만 사용할 수 있습니다. 가능한 값에는 인증 자격 증명에 대한 `authenticated` 및 미인증 자격 증명에 대한 `unauthenticated` 이(가) 포함됩니다.

파라미터	설명
`$context.identity.cognitoIdentityId`	요청을 작성한 호출자의 Amazon Cognito 자격 증명 ID입니다. Amazon Cognito 자격 증명으로 요청을 서명한 경우에만 사용할 수 있습니다.
`$context.identity.cognitoIdentityPoolId`	요청을 작성한 호출자의 Amazon Cognito 자격 증명 풀 ID입니다. Amazon Cognito 자격 증명으로 요청을 서명한 경우에만 사용할 수 있습니다.
`$context.identity.principalOrgId`	AWS 조직 ID입니다.
`$context.identity.sourceIp`	API Gateway 엔드포인트를 요청하는 즉시 TCP 연결의 소스 IP 주소입니다.
`$context.identity.clientCert.clientCertPem`	상호 TLS 인증 시 클라이언트가 제공한 PEM 인코딩된 클라이언트 인증서입니다. 클라이언트에서 상호 TLS가 활성화된 사용자 지정 도메인 이름을 사용하여 API에 액세스할 때 제공됩니다. 상호 TLS 인증이 실패할 경우 액세스 로그에만 제공됩니다.
`$context.identity.clientCert.subjectDN`	클라이언트가 제공하는 인증서의 주체가 갖는 고유 이름입니다. 클라이언트에서 상호 TLS가 활성화된 사용자 지정 도메인 이름을 사용하여 API에 액세스할 때 제공됩니다. 상호 TLS 인증이 실패할 경우 액세스 로그에만 제공됩니다.
`$context.identity.clientCert.issuerDN`	클라이언트가 제공하는 인증서의 발행자가 갖는 고유 이름입니다. 클라이언트에서 상호 TLS가 활성화된 사용자 지정 도메인 이름을 사용하여 API에 액세스할 때 제공됩니다. 상호 TLS 인증이 실패할 경우 액세스 로그에만 제공됩니다.

파라미터	설명
`$context.identity.clientCert.serialNumber`	인증서의 일련 번호입니다. 클라이언트에서 상호 TLS가 활성화된 사용자 지정 도메인 이름을 사용하여 API에 액세스할 때 제공됩니다. 상호 TLS 인증이 실패할 경우 액세스 로그에만 제공됩니다.
`$context.identity.clientCert.validity.notBefore`	인증서의 유효 기간이 시작되는 날짜입니다. 클라이언트에서 상호 TLS가 활성화된 사용자 지정 도메인 이름을 사용하여 API에 액세스할 때 제공됩니다. 상호 TLS 인증이 실패할 경우 액세스 로그에만 제공됩니다.
`$context.identity.clientCert.validity.notAfter`	인증서의 유효 기간이 끝나는 날짜입니다. 클라이언트에서 상호 TLS가 활성화된 사용자 지정 도메인 이름을 사용하여 API에 액세스할 때 제공됩니다. 상호 TLS 인증이 실패할 경우 액세스 로그에만 제공됩니다.
`$context.identity.user`	리소스 액세스에 대해 권한을 부여할 사용자의 보안 주체 ID입니다. IAM 권한 부여를 사용하는 라우팅에 대해 지원됩니다.
`$context.identity.userAgent`	API 호출자의 `User-Agent` 헤더입니다.
`$context.identity.userArn`	인증 후 식별된 실제 사용자의 ARN(Amazon Resource Name)입니다. 자세한 설명은 https://docs.aws.amazon.com/IAM/latest/UserGuide/id_users.html 섹션을 참조하세요.
`$context.path`	요청 경로입니다. 예를 들어 `https://{rest-api-id}.execute-api.{region}.amazonaws.com/{stage}/root/child` 의 비 프록시 요청 URL의 경우, `$context.path` 값은 `/{stage}/root/child` 입니다.

파라미터	설명
`$context.protocol`	요청 프로토콜입니다(예: `HTTP/1.1`). **Note** API Gateway API는 HTTP/2 요청을 수락할 수 있지만 API Gateway는 HTTP/1.1을 사용하여 백엔드 통합에 요청을 보냅니다. 따라서 클라이언트가 HTTP/2를 사용하는 요청을 보내더라도 요청 프로토콜은 HTTP/1.1로 기록됩니다.
`$context.requestId`	요청의 ID입니다. 클라이언트는 이 요청 ID를 재정의할 수 있습니다. API Gateway가 생성하는 고유한 요청 ID에 대한 `$context.extendedRequestId` 를 사용합니다.
`$context.requestOverride.header.` *`header_name`*	요청 헤더 재정의입니다. 이 파라미터를 정의하면 통합 요청(Integration Request) 창에서 정의한 HTTP 헤더(HTTP Headers) 대신에 사용할 헤더가 포함됩니다. 자세한 설명은 매핑 템플릿을 사용한 API의 요청 및 응답 파라미터와 상태 코드 재정의 섹션을 참조하세요.
`$context.requestOverride.path.` *`path_name`*	요청 경로 재정의입니다. 이 파라미터를 정의하면 통합 요청(Integration Request) 창에서 정의한 URL 경로 파라미터(URL Path Parameters) 대신에 사용할 요청 경로가 포함됩니다. 자세한 설명은 매핑 템플릿을 사용한 API의 요청 및 응답 파라미터와 상태 코드 재정의 섹션을 참조하세요.

파라미터	설명
`$context.requestOverride.querystring.` *querystring_name*	요청 쿼리 문자열 재정의입니다. 이 파라미터를 정의하면 통합 요청(Integration Request) 창에서 정의한 URL 쿼리 문자열 파라미터(URL Query String Parameters) 대신에 사용할 요청 쿼리 문자열이 포함됩니다. 자세한 설명은 매핑 템플릿을 사용한 API의 요청 및 응답 파라미터와 상태 코드 재정의 섹션을 참조하세요.
`$context.responseOverride.header.` *header_name*	응답 헤더 재정의입니다. 이 파라미터를 정의하면 통합 요청(Integration Response) 창에서 기본 매핑(Default mapping)으로 정의한 응답 헤더(Response header) 대신에 반환할 헤더가 포함됩니다. 자세한 설명은 매핑 템플릿을 사용한 API의 요청 및 응답 파라미터와 상태 코드 재정의 섹션을 참조하세요.
`$context.responseOverride.status`	응답 상태 코드 재정의입니다. 이 파라미터를 정의하면 통합 요청(Integration Response) 창에서 기본 매핑(Default mapping)으로 정의한 메서드 응답 상태(Method response status) 대신에 반환할 상태 코드가 포함됩니다. 자세한 설명은 매핑 템플릿을 사용한 API의 요청 및 응답 파라미터와 상태 코드 재정의 섹션을 참조하세요.
`$context.requestTime`	CLF 형식 요청 시간(`dd/MMM/yyyy:HH:mm:ss +-hhmm`)입니다.
`$context.requestTimeEpoch`	Epoch 형식 요청 시간(밀리초)입니다.
`$context.resourceId`	API Gateway가 리소스에 할당하는 식별자입니다.

파라미터	설명
$context.resourcePath	리소스에 대한 경로입니다. 예를 들어 https://{rest-api-id}.execute-api.{region}.amazonaws.com/{stage}/root/child 의 비 프록시 요청 URI의 경우, $context.resourcePath 값은 /root/child 입니다. 자세한 설명은 자습서: HTTP 비 프록시 통합을 사용하여 REST API 빌드 섹션을 참조하세요.
$context.stage	API 요청의 개발 단계입니다(예: Beta 또는 Prod).
$context.wafResponseCode	AWS WAF에서 받은 응답: WAF_ALLOW 또는 WAF_BLOCK . 스테이지가 웹 ACL과 연결되어 있지 않은 경우 설정되지 않습니다. 자세한 설명은 the section called “AWS WAF” 섹션을 참조하세요.
$context.webaclArn	요청을 허용할지 차단할지 결정하는 데 사용되는 웹 ACL의 전체 ARN입니다. 스테이지가 웹 ACL과 연결되어 있지 않은 경우 설정되지 않습니다. 자세한 설명은 the section called “AWS WAF” 섹션을 참조하세요.

$context 변수 템플릿 예제

API 메서드가 특정 형식의 데이터를 요구하는 백엔드로 구조화된 데이터를 전달하는 경우 매핑 템플릿에 $context 변수를 사용할 수 있습니다.

다음 예제는 받은 $context 변수를, 통합 요청 페이로드에서 약간 다른 이름을 가진 백엔드 변수에 매핑하는 매핑 템플릿을 보여 줍니다.

> **Note**
>
> 이러한 변수 중 하나가 API 키입니다. 이 예제는 메서드에서 "API 키 요구"가 활성화되었음을 전제로 합니다.

```
{
    "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"
}
```

액세스 로깅 전용의 **$context** 변수

다음 $context 변수는 액세스 로깅에만 사용할 수 있습니다. 자세한 설명은 the section called “CloudWatch 로그” 섹션을 참조하세요. (WebSocket API에 대해서는 을 참조하십시오the section called “지표”.)

파라미터	설명
$context.authorize.error	권한 부여 오류 메시지입니다.
$context.authorize.latency	권한 부여 지연 시간(ms)입니다.
$context.authorize.status	권한 부여 시도에서 반환된 상태 코드입니다.
$context.authorizer.error	권한 부여자로부터 반환된 오류 메시지입니다.
$context.authorizer.integrationLatency	권한 부여자 지연 시간(ms)입니다.

파라미터	설명
$context.authorizer.integrationStatus	Lambda 권한 부여자로부터 반환된 상태 코드입니다.
$context.authorizer.latency	권한 부여자 지연 시간(ms)입니다.
$context.authorizer.requestId	AWS 엔드포인트의 요청 ID.
$context.authorizer.status	권한 부여자로부터 반환된 상태 코드입니다.
$context.authenticate.error	인증 시도에서 반환된 오류 메시지입니다.
$context.authenticate.latency	인증 지연 시간(ms)입니다.
$context.authenticate.status	인증 시도에서 반환된 상태 코드입니다.
$context.customDomain.basePathMatched	들어오는 요청이 일치하는 API 매핑의 경로입니다. 클라이언트가 사용자 지정 도메인 이름을 사용하여 API에 액세스할 때 적용할 수 있습니다. 예를 들어 클라이언트가 https://api.example.com/v1/orders/1234 에 요청을 보내고 요청이 API 매핑을 경로 v1/orders 와(과) 일치시키는 경우 값은 v1/orders 입니다. 자세한 내용은 the section called “API 매핑” 단원을 참조하십시오.
$context.integration.error	통합에서 반환된 오류 메시지입니다.
$context.integration.integrationStatus	Lambda 프록시 통합의 경우, 백엔드 Lambda 함수 코드가 아닌 AWS Lambda에서 상태 코드가 반환되었습니다.
$context.integration.latency	통합 지연 시간(ms)입니다. $context.integrationLatency 와 동일합니다.
$context.integration.requestId	엔드포인트의 요청 ID. AWS $context.awsEndpointRequestId 와 동일합니다.

파라미터	설명
`$context.integration.status`	통합에서 반환된 상태 코드입니다. Lambda 프록시 통합의 경우 Lambda 함수 코드에서 반환된 상태 코드입니다.
`$context.integrationLatency`	통합 지연 시간(ms)입니다.
`$context.integrationStatus`	Lambda 프록시 통합의 경우 이 파라미터는 백엔드 Lambda 함수 코드가 아니라 반환된 AWS Lambda상태 코드를 나타냅니다.
`$context.responseLatency`	응답 지연 시간(ms)입니다.
`$context.responseLength`	바이트로 된 응답 페이로드 길이입니다.
`$context.status`	메서드 응답 상태입니다.
`$context.waf.error`	에서 반환된 오류 메시지. AWS WAF
`$context.waf.latency`	AWS WAF 지연 시간 (ms)
`$context.waf.status`	에서 반환된 상태 AWS WAF코드입니다.
`$context.xrayTraceId`	X-Ray 추적의 추적 ID입니다. 자세한 설명은 the section called “ 설정AWS X-Ray” 섹션을 참조하세요.

`$input` 변수

`$input` 변수는 매핑 템플릿에서 처리할 메서드 요청 페이로드와 파라미터를 나타냅니다. 이 변수는 네 가지 함수를 제공합니다.

변수 및 함수	설명
`$input.body`	원시 요청 페이로드를 문자열로 반환합니다.
`$input.json(x)`	이 함수는 JSONPath 표현식을 평가하고 결과를 JSON 문자열로 반환합니다.

변수 및 함수	설명
	예를 들어, `$input.json('$.pets')` 은 pets 구조를 나타내는 JSON 문자열을 반환합니다. JSONPath에 대한 자세한 내용은 JSONPath 또는 Java용 JSONPath를 참조하십시오.
`$input.params()`	모든 요청 파라미터의 맵을 반환합니다. 잠재적인 주입 공격을 방지하기 위해 `$util.escapeJavaScript` 을(를) 사용하여 결과를 삭제하는 것이 좋습니다. 요청 삭제를 완벽하게 제어하려면 템플릿 없이 프록시 통합을 사용하고 통합에서 요청 삭제를 처리합니다.
`$input.params(x)`	파라미터 이름 문자열 x가 지정된 경로, 쿼리 문자열 또는 헤더 값(순서대로 검색됨)에서 메서드 요청 파라미터 값을 반환합니다. 잠재적인 주입 공격을 방지하기 위해 `$util.escapeJavaScript` 을(를) 사용하여 파라미터를 삭제하는 것이 좋습니다. 파라미터 삭제를 완벽하게 제어하려면 템플릿 없이 프록시 통합을 사용하고 통합에서 요청 삭제를 처리합니다.

변수 및 함수	설명
`$input.path(x)`	JSONPath 표현식 문자열 (x)을 가져와서 결과의 JSON 객체로 반환합니다. 이를 통해 기본적으로 Apache Velocity Template Language(VTL)에서 페이로드 요소에 액세스하고 조작할 수 있습니다. 예를 들어, `$input.path('$.pets')` 표현식이 다음과 같은 객체를 반환할 경우: `[ { "id": 1, "type": "dog", "price": 249.99 }, { "id": 2, "type": "cat", "price": 124.99 }, { "id": 3, "type": "fish", "price": 0.99 } ]` `$input.path('$.pets').count()` 는 "3"을 반환합니다. JSONPath에 대한 자세한 내용은 JSONPath 또는 Java용 JSONPath를 참조하십시오.

$input 변수 템플릿 예제

파라미터 매핑 템플릿 예제

다음 파라미터 매핑 예제에서는 path, querystring 및 header를 포함한 모든 파라미터를 JSON 페이로드를 통해 통합 엔드포인트로 전달합니다.

```
#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
  }
}
```

사실상 이 매핑 템플릿은 다음과 같이 페이로드의 모든 요청 파라미터를 출력합니다.

```
{
  "params" : {
     "path" : {
       "path_name" : "path_value",
       ...
     }
     "header" : {
       "header_name" : "header_value",
       ...
     }
     "querystring" : {
       "querystring_name" : "querystring_value",
       ...
     }
   }
}
```

모델 변수를 사용하거나 사용하지 않고 쿼리 문자열과 요청 본문을 가져오기 위해 $input 변수를 사용할 수 있습니다. 또한 파라미터와 페이로드 또는 페이로드의 하위 섹션을 Lambda 함수로 가져올 수도 있습니다. 다음 예는 이 작업을 수행하는 방법을 보여줍니다.

$input을 사용하는 JSON 매핑 템플릿 예제

다음 예제에서는 매핑을 사용하여 쿼리 문자열에서 이름을 읽은 다음 전체 POST 본문을 요소에 포함하는 방법을 보여줍니다.

```
{
    "name" : "$input.params('name')",
    "body" : $input.json('$')
}
```

JSON 입력에 파싱할 수 없는 이스케이프 처리되지 않은 문자가 포함된 경우 400 응답이 JavaScript 반환될 수 있습니다. 위의 $util.escapeJavaScript($input.json('$'))을 적용하면 JSON 입력을 올바르게 구문 분석할 수 있습니다.

$input을 사용하는 매핑 템플릿 예제

다음 예제에서는 JSONPath 표현식을 json() 메서드에 전달하는 방법을 보여줍니다. 마침표 (.)와 속성 이름을 사용하여 요청 본문 객체의 특정 속성을 읽을 수도 있습니다.

```
{
    "name" : "$input.params('name')",
    "body" : $input.json('$.mykey')
}
```

메서드 요청 페이로드에 파싱할 수 없는 이스케이프 처리되지 않은 문자가 포함된 경우 응답을 받을 수 있습니다. JavaScript 400 이 경우 다음에 표시된 것처럼 매핑 템플릿에서 $util.escapeJavaScript() 함수를 호출해야 합니다.

```
{
    "name" : "$input.params('name')",
    "body" : "$util.escapeJavaScript($input.json('$.mykey'))"
}
```

$input을 사용하는 요청 및 응답 예제

다음은 세 가지 함수를 모두 사용하는 예입니다.

요청 템플릿:

```
Resource: /things/{id}

With input template:
{
    "id" : "$input.params('id')",
    "count" : "$input.path('$.things').size()",
    "things" : "$util.escapeJavaScript($input.json('$.things'))"
}

POST /things/abc
{
    "things" : {
        "1" : {},
        "2" : {},
        "3" : {}
    }
}
```

응답:

```
{
  "id": "abc",
  "count": "3",
  "things": {
    "1": {},
    "2": {},
    "3": {}
  }
}
```

더 많은 매핑 예는 매핑 템플릿 이해 단원을 참조하십시오.

$stageVariables

단계 변수는 파라미터 매핑과 매핑 템플릿 및 메서드 통합에 사용되는 ARN 및 URL의 자리 표시자로 사용할 수 있습니다. 자세한 설명은 the section called “스테이지 변수 설정” 섹션을 참조하세요.

구문	설명
$stageVariables. *<variable_name>*	*<variable_name>* 은 단계 변수 이름을 나타냅니다.

구문	설명
`$stageVariables['` *<variable_name>* `']`	*<variable_name>* 은 모든 단계 변수 이름을 나타냅니다.
`${stageVariables['` *<variable_name>* `']}`	*<variable_name>* 은 모든 단계 변수 이름을 나타냅니다.

`$util` 변수

`$util` 변수에는 매핑 템플릿에서 사용할 유틸리티 함수가 포함됩니다.

> **Note**
>
> 달리 지정하지 않는 한, 기본 문자 집합은 UTF-8입니다.

함수	설명
`$util.escapeJavaScript()`	문자열 규칙을 사용하여 문자열의 문자를 이스케이프합니다. JavaScript **Note** 이 함수는 일반 작은따옴표(')를 이스케이프된 작은따옴표(\')로 변환합니다. 그러나 이스케이프된 작은따옴표는 JSON에서 유효하지 않습니다. 따라서 이 함수의 결과를 JSON 속성에서 사용할 경우 이스케이프된 작은따옴표(\')를 다시 일반 작은따옴표(')로 변환해야 합니다. 방법은 다음 예제와 같습니다: `"input" : "$util.escapeJavaScript( data).replaceAll("\\'","'")"`

함수	설명

`$util.parseJson()`	"문자열화된" JSON을 가져와서 결과의 객체 표현을 반환합니다. 이 함수의 결과를 사용하여 기본적으로 Apache VTL(Velocity Template Language)에서 페이로드 요소에 액세스하고 조작할 수 있습니다. 예를 들어 다음과 같은 페이로드가 있고
	`{"errorMessage":"{\"key1\":\"var1\", \"key2\":{\"arr\":[1,2,3]}}"}`
	다음 매핑 템플릿을 사용하는 경우
	`#set ($errorMessageObj = $util.parseJson($input.path('$.errorMessage')))`
	`{`
	`"errorMessageObjKey2ArrVal" : $errorMessageObj.key2.arr[0]`
	`}`
	출력은 다음과 같습니다.
	`{`
	`"errorMessageObjKey2ArrVal" : 1`
	`}`

`$util.urlEncode()`	문자열을 “application/x-www-form-urlencoded” 형식으로 변환합니다.
`$util.urlDecode()`	“x-www-form-urlencoded애플리케이션/” 문자열을 디코딩합니다.
`$util.base64Encode()`	데이터를 base64 인코딩 문자열로 인코딩합니다.
`$util.base64Decode()`	데이터를 base64 인코딩 문자열에서 디코딩합니다.

API Gateway의 게이트웨이 응답

게이트웨이 응답은 API Gateway에서 정의한 응답 유형으로 식별됩니다. 응답은 HTTP 상태 코드, 파라미터 매핑에 지정되는 추가 헤더 집합, 비 VTL 매핑 템플릿에 의해 생성되는 페이로드 등으로 구성됩니다.

API Gateway REST API에서 게이트웨이 응답은 GatewayResponse로 표시됩니다. OpenAPI에서 `GatewayResponse` 인스턴스는 x-amazon-apigateway-gateway-responses.gatewayResponse 확장으로 설명됩니다.

게이트웨이 응답을 활성화하려면 API 수준에서 지원되는 응답 유형에 대한 게이트웨이 응답을 설정합니다. API Gateway에서 해당 유형의 응답을 반환할 때마다 게이트웨이 응답에 정의된 헤더 매핑 및 페이로드 매핑 템플릿을 적용하여 매핑된 결과가 API 호출자에게 반환됩니다.

다음 섹션에서는 API Gateway 콘솔 및 API Gateway REST API를 사용하여 게이트웨이 응답을 설정하는 방법을 보여줍니다.

오류 응답을 사용자 지정하도록 게이트웨이 응답 설정

API Gateway에서 수신 요청을 처리하지 못한 경우 요청을 통합 백엔드에 전달하지 않고 클라이언트에 오류 응답을 반환합니다. 기본적으로 오류 응답에는 오류를 설명하는 짧은 설명이 포함되어 있습니다. 예를 들어 정의되지 않은 API 리소스에 대한 작업을 호출할 경우 `{ "message": "Missing Authentication Token" }` 메시지를 포함하는 오류 응답이 수신됩니다. API Gateway를 처음 사용할 경우 실제로 무엇이 잘못되었는지 이해하기 어려울 수 있습니다.

일부 오류 응답에 대해 API 개발자는 API Gateway를 통해 다양한 형식으로 응답을 반환하도록 사용자 지정할 수 있습니다. `Missing Authentication Token` 예제에서 가능한 원인을 사용하여 원래 응답 페이로드에 힌트를 추가할 수 있습니다. 예를 들면 다음과 같습니다. `{"message":"Missing Authentication Token", "hint":"The HTTP method or resources may not be supported."}`.

API가 외부 Exchange와 AWS 클라우드 사이에서 중재할 경우, 통합 요청 또는 통합 응답에 대한 VTL 매핑 템플릿을 사용하여 페이로드를 다른 형식으로 매핑합니다. 하지만 VTL 매핑 템플릿은 성공적인 응답을 포함하는 유효한 요청에 대해서만 작동합니다.

요청이 유효하지 않은 경우 API Gateway에서는 통합을 완전히 무시하고 오류 응답을 반환합니다. 사용자 지정을 사용하여 오류 응답을 Exchange 호환 형식으로 렌더링해야 합니다. 여기서 사용자 지정은 간단한 변수 대체만 지원하는 비 VTL 매핑 템플릿으로 렌더링됩니다.

API Gateway에서 생성된 오류 응답을 API Gateway에서 생성되는 응답으로 일반화하며, 이를 게이트웨이 응답이라고 합니다. 이는 API Gateway에서 생성된 응답을 통합 응답과 구별합니다. 게이트웨이

응답 매핑 템플릿에서는 `$context` 변수 값, `$stageVariables` 속성 값 및 메서드 요청 파라미터에 `method.request.`*`param-position`*`.`*`param-name`* 형식으로 액세스할 수 있습니다.

`$context` 변수에 대한 자세한 내용은 $context데이터 모델, 권한 부여자, 매핑 템플릿, 액세스 로깅용 변수 CloudWatch 단원을 참조하십시오. `$stageVariables`에 대한 자세한 내용은 `$stageVariables` 단원을 참조하세요. 메서드 요청 파라미터에 대한 자세한 내용은 the section called “`$input` 변수” 단원을 참조하세요.

주제

- API Gateway 콘솔을 사용하여 REST API에 대한 게이트웨이 응답 설정
- API Gateway REST API를 사용하여 게이트웨이 응답 설정
- OpenAPI에서 게이트웨이 응답 사용자 지정 설정
- 게이트웨이 응답 유형

API Gateway 콘솔을 사용하여 REST API에 대한 게이트웨이 응답 설정

API Gateway 콘솔을 사용하여 게이트웨이 응답을 사용자 지정하려면

1. https://console.aws.amazon.com/apigateway에서 API Gateway 콘솔에 로그인합니다.
2. REST API를 선택합니다.
3. 기본 탐색 창에서 게이트웨이 응답을 선택합니다.
4. 응답 유형을 선택한 다음 편집을 선택합니다. 이 연습에서는 누락된 인증 토큰을 예로 들겠습니다.
5. API Gateway가 생성한 상태 코드를 변경하여, 사용자 API의 요구 사항에 부합하는 다른 상태 코드를 반환할 수 있습니다. 이 예에서 사용자 지정은 상태 코드를 기본값(`403`)에서 `404`로 변경합니다. 클라이언트가 지원되지 않거나 잘못되어 찾을 수 없는 리소스를 호출할 경우 이 오류 메시지가 발생하기 때문입니다.
6. 사용자 지정 헤더를 반환하려면 응답 헤더에서 헤더 추가를 선택합니다. 설명을 위해 다음 사용자 지정 헤더를 추가하겠습니다.

```
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
```

앞서 본 헤더 매핑에서 정적 도메인 이름(`'a.b.c'`)은 `Allow-Control-Allow-Origin` 헤더에 매핑되어 API에 대한 CORS 액세스를 허용합니다. `x-amzn-RequestId`의 입력 요청 헤더는 응

답의 request-id에 매핑됩니다. 수신되는 요청의 petId 경로 변수는 응답의 request-path 헤더에 매핑됩니다. 원래 요청의 q 쿼리 파라미터는 응답의 request-query 헤더에 매핑됩니다.

7. 응답 템플릿에서 콘텐츠 유형은 application/json를 유지하고 템플릿 본문 편집기에 다음과 같은 본문 매핑 템플릿을 입력합니다.

```
{
     "message":"$context.error.messageString",
     "type": "$context.error.responseType",
     "statusCode": "'404'",
     "stage": "$context.stage",
     "resourcePath": "$context.resourcePath",
     "stageVariables.a": "$stageVariables.a"
}
```

   이 예는 게이트웨이 응답 본문의 속성에 $context 및 $stageVariables 속성을 매핑하는 방법을 보여줍니다.

8. 변경 사항 저장을 선택합니다.
9. 새 단계 또는 기존 단계에 API를 배포합니다.

해당 API 메서드의 호출 URL이 https://*o81lxisefl*.execute-api.us-east-1.amazonaws.com/custErr/pets/{petId}라고 가정하고 다음 CURL 명령을 호출하여 게이트웨이 응답을 테스트합니다.

```
curl -v -H 'x-amzn-RequestId:123344566' https://o81lxisefl.execute-api.us-
east-1.amazonaws.com/custErr/pets/5/type?q=1
```

추가적인 쿼리 문자열 파라미터 q=1이 API와 호환되지 않기 때문에 오류가 반환되어 지정된 게이트웨이 응답이 트리거됩니다. 다음과 비슷한 게이트웨이 응답이 표시됩니다.

```
> 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
}
```

앞의 예에서는 API 백엔드가 Pet Store이고 API에 단계 변수, a가 정의된 것으로 가정했습니다.

API Gateway REST API를 사용하여 게이트웨이 응답 설정

API를 생성하고 식별자를 획득한 이후에 API Gateway REST API를 사용하여 게이트웨이 응답을 사용자 지정해야 합니다. API 식별자를 검색하려면 restapi:gateway-responses 링크 관계를 따르고 결과를 검사할 수 있습니다.

API Gateway REST API를 사용하여 게이트웨이 응답을 사용자 지정하려면

1. 전체 GatewayResponse 인스턴스를 덮어쓰려면 gatewayresponse:put 작업을 호출합니다. URL 경로 파라미터에 원하는 responseType을 지정하고 요청 페이로드에 statusCode, responseParameters 및 responseTemplates 매핑을 제공합니다.
2. `GatewayResponse` 인스턴스의 일부를 업데이트하려면 gatewayresponse:update 작업을 호출합니다. URL 경로 파라미터에 원하는 `responseType`을 지정하고 요청 페이로드에 원하는 개별 `GatewayResponse` 속성(예: `responseParameters` 또는 `responseTemplates` 매핑)을 제공합니다.

OpenAPI에서 게이트웨이 응답 사용자 지정 설정

API 루트 수준에서 `x-amazon-apigateway-gateway-responses` 확장을 사용하여 OpenAPI에서 게이트웨이 응답을 사용자 지정할 수 있습니다. 다음 OpenAPI 정의는

MISSING_AUTHENTICATION_TOKEN 유형의 GatewayResponse를 사용자 지정하는 예를 보여줍니다.

```
  "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}"
      }
    }
```

이 예제의 사용자 지정은 상태 코드를 기본값(403)에서 404로 변경합니다. 또한 application/json 미디어 유형에 대한 헤더 파라미터 네 개와 본문 매핑 템플릿 하나를 추가합니다.

게이트웨이 응답 유형

API Gateway에서는 API 개발자가 사용자 지정할 수 있도록 다음과 같은 게이트웨이 응답을 공개합니다.

게이트웨이 응답 유형	기본 상태 코드	설명
ACCESS_DENIED	403	권한 부여 실패(예: 사용자 지정 또는 Amazon Cognito 권한 부여자가 액세스를 거부한 경우)에 대한 게이트웨이 응답입니다. 응답 유형이 지정되지 않은 경우 이 응답의 기본 유형은 DEFAULT_4XX 입니다.
API_CONFIGURATION_ERROR	500	유효하지 않은 엔드포인트 주소가 제출된 경우, 이진 지원

게이트웨이 응답 유형	기본 상태 코드	설명
		이 수행될 때 이진 데이터에 대해 base64 디코딩이 실패한 경우, 통합 응답 매핑이 어떤 템플릿과도 일치할 수 없어서 기본 템플릿이 구성되어 있지 않은 경우를 포함하여 유효하지 않은 API 구성에 대한 게이트웨이 응답입니다. 응답 유형이 지정되지 않은 경우 이 응답의 기본 유형은 DEFAULT_5XX 입니다.
AUTHORIZER_CONFIGURATION_ERROR	500	사용자 지정 또는 Amazon Cognito 권한 부여자 연결 실패에 대한 게이트웨이 응답입니다. 응답 유형이 지정되지 않은 경우 이 응답의 기본 유형은 DEFAULT_5XX 입니다.
AUTHORIZER_FAILURE	500	사용자 지정 또는 Amazon Cognito 권한 부여자가 호출자를 인증하지 못한 경우의 게이트웨이 응답입니다. 응답 유형이 지정되지 않은 경우 이 응답의 기본 유형은 DEFAULT_5XX 입니다.
BAD_REQUEST_PARAMETERS	400	활성화된 요청 검사기에 따라 요청 파라미터를 확인할 수 없는 경우의 게이트웨이 응답입니다. 응답 유형이 지정되지 않은 경우 이 응답의 기본 유형은 DEFAULT_4XX 입니다.

게이트웨이 응답 유형	기본 상태 코드	설명
BAD_REQUEST_BODY	400	활성화된 요청 검사기에 따라 요청 본문을 확인할 수 없는 경우의 게이트웨이 응답입니다. 응답 유형이 지정되지 않은 경우 이 응답의 기본 유형은 DEFAULT_4XX 입니다.
DEFAULT_4XX	Null	상태 코드가 4XX인 지정되지 않은 응답 유형에 대한 기본 게이트웨이 응답입니다. 이 대체 게이트웨이 응답의 상태 코드를 변경하면 모든 다른 4XX 응답의 상태 코드가 새 값으로 변경됩니다. 이 상태 코드를 null로 재설정하면 모든 다른 4XX 응답의 상태 코드가 원래 값으로 되돌아갑니다. **Note** AWS WAF 사용자 지정 응답은 사용자 지정 게이트웨이 응답보다 우선합니다.

게이트웨이 응답 유형	기본 상태 코드	설명
DEFAULT_5XX	Null	상태 코드가 5XX인 지정되지 않은 응답 유형에 대한 기본 게이트웨이 응답입니다. 이 대체 게이트웨이 응답의 상태 코드를 변경하면 모든 다른 5XX 응답의 상태 코드가 새 값으로 변경됩니다. 이 상태 코드를 null로 재설정하면 모든 다른 5XX 응답의 상태 코드가 원래 값으로 되돌아갑니다.
EXPIRED_TOKEN	403	AWS 인증 토큰 만료 오류에 대한 게이트웨이 응답입니다. 응답 유형이 지정되지 않은 경우 이 응답의 기본 유형은 DEFAULT_4XX 입니다.
INTEGRATION_FAILURE	504	통합 실패 오류에 대한 게이트웨이 응답입니다. 응답 유형이 지정되지 않은 경우 이 응답의 기본 유형은 DEFAULT_5XX 입니다.
INTEGRATION_TIMEOUT	504	통합 시간 초과 오류에 대한 게이트웨이 응답입니다. 응답 유형이 지정되지 않은 경우 이 응답의 기본 유형은 DEFAULT_5XX 입니다.
INVALID_API_KEY	403	API 키를 요구하는 메서드에 대해 제출되는 잘못된 API 키에 대한 게이트웨이 응답입니다. 응답 유형이 지정되지 않은 경우 이 응답의 기본 유형은 DEFAULT_4XX 입니다.

게이트웨이 응답 유형	기본 상태 코드	설명
INVALID_SIGNATURE	403	잘못된 AWS 서명 오류에 대한 게이트웨이 응답입니다. 응답 유형이 지정되지 않은 경우 이 응답의 기본 유형은 DEFAULT_4XX 입니다.
MISSING_AUTHENTICATION_TOKEN	403	클라이언트가 지원되지 않는 API 메서드 또는 리소스를 호출하려고 시도하는 경우를 비롯한 누락된 인증 토큰 오류에 대한 게이트웨이 응답입니다. 응답 유형이 지정되지 않은 경우 이 응답의 기본 유형은 DEFAULT_4XX 입니다.
QUOTA_EXCEEDED	429	사용량 계획 할당량 초과 오류에 대한 게이트웨이 응답입니다. 응답 유형이 지정되지 않은 경우 이 응답의 기본 유형은 DEFAULT_4XX 입니다.
REQUEST_TOO_LARGE	413	요청이 너무 큼 오류에 대한 게이트웨이 응답입니다. 응답 유형이 지정되지 않은 경우 이 응답의 기본 유형은 DEFAULT_4XX 입니다.

게이트웨이 응답 유형	기본 상태 코드	설명
RESOURCE_NOT_FOUND	404	API 키 인증 및 권한 부여를 제외하고 API 요청에서 인증 및 권한 부여를 전달한 이후에 API Gateway에서 지정된 리소스를 찾을 수 없는 경우의 게이트웨이 응답입니다. 응답 유형이 지정되지 않은 경우 이 응답의 기본 유형은 DEFAULT_4XX 입니다.
THROTTLED	429	사용량 계획, 메서드, 단계 또는 계정 수준 조절 한도가 초과한 경우의 게이트웨이 응답입니다. 응답 유형이 지정되지 않은 경우 이 응답의 기본 유형은 DEFAULT_4XX 입니다.
UNAUTHORIZED	401	사용자 지정 또는 Amazon Cognito 권한 부여자가 호출자를 인증하지 못한 경우의 게이트웨이 응답입니다.
UNSUPPORTED_MEDIA_TYPE	415	엄격한 패스스루 동작을 활성화한 경우에 페이로드가 지원되지 않는 미디어 유형일 때의 게이트웨이 응답입니다. 응답 유형이 지정되지 않은 경우 이 응답의 기본 유형은 DEFAULT_4XX 입니다.

게이트웨이 응답 유형	기본 상태 코드	설명
WAF_FILTERED	403	요청이 AWS WAF에 의해 차단될 때의 게이트웨이 응답입니다. 응답 유형이 지정되지 않은 경우 이 응답의 기본 유형은 DEFAULT_4XX 입니다. **Note** AWS WAF 사용자 지정 응답은 사용자 지정 게이트웨이 응답보다 우선합니다.

REST API 리소스에 대한 CORS 활성화

CORS(Cross-origin 리소스 공유)는 브라우저에서 실행 중인 스크립트에서 시작되는 cross-origin HTTP 요청을 제한하는 브라우저 보안 기능입니다.

CORS 지원 활성화 여부 확인

cross-origin HTTP 요청은 다음에 대해 이루어지는 요청입니다.

- 다른 도메인(예: example.com에서 amazondomains.com으로)
- 다른 하위 도메인(예: example.com에서 petstore.example.com으로)
- 다른 포트(예: example.com에서 example.com:10777으로)
- 다른 프로토콜(예: https://example.com에서 http://example.com으로)

API에 액세스할 수 없고 Cross-Origin Request Blocked가 포함된 오류 메시지를 수신하는 경우 CORS를 활성화해야 할 수 있습니다.

Cross-origin HTTP 요청은 단순 요청과 비 단순 요청의 두 가지 유형으로 나눌 수 있습니다.

단순한 요청을 위한 CORS 사용 설정

다음과 같은 모든 조건을 충족하는 HTTP 요청은 단순 요청입니다.

- GET, HEAD, POST 요청만 허용하는 API 리소스에 대해 발행된 요청입니다.
- POST 메서드 요청인 경우 Origin 헤더를 포함해야 합니다.
- 요청 페이로드 콘텐츠 유형이 text/plain, multipart/form-data 또는 application/x-www-form-urlencoded입니다.
- 요청에 사용자 지정 헤더가 없습니다.
- 단순 요청에 대한 Mozilla CORS 문서에 나열된 추가 요청.

간단한 교차 오리진 POST 메서드 요청의 경우 리소스의 응답에 Access-Control-Allow-Origin: '*' 또는 Access-Control-Allow-Origin:*'origin'* 헤더가 포함되어야 합니다.

다른 모든 cross-origin HTTP 요청은 비 단순 요청입니다.

단순하지 않은 요청을 위한 CORS 사용 설정

API 리소스가 단순하지 않은 요청을 받는 경우 통합 유형에 따라 추가 CORS 지원을 사용 설정해야 합니다.

비프록시 통합을 위한 CORS 사용 설정

이러한 통합의 경우 CORS 프로토콜은 브라우저에 사전 요청을 서버로 보내고, 서버 승인(또는 보안 인증 정보 요청)을 기다린 후 실제 요청을 보내도록 요구합니다. 사전 요청에 적절한 응답을 보내도록 API를 구성해야 합니다.

사전 요청에 대한 응답을 만드는 방법

1. 임의 통합으로 OPTIONS 메서드를 생성합니다.
2. 다음 응답 헤더를 200 메서드 응답에 추가합니다.
   - Access-Control-Allow-Headers
   - Access-Control-Allow-Methods
   - Access-Control-Allow-Origin
3. 응답 헤더의 값을 입력합니다. 모든 오리진, 모든 메서드 및 일반적인 헤더를 허용하려면 다음 헤더 값을 사용합니다.
   - Access-Control-Allow-Headers: 'Content-Type,X-Amz-Date,Authorization,X-Api-Key,X-Amz-Security-Token'
   - Access-Control-Allow-Methods: '*'

- `Access-Control-Allow-Origin: '*'`

사전 요청을 생성한 후에는 CORS 사용이 설정된 모든 메서드에서 최소 200개 응답 모두에 대해 `Access-Control-Allow-Origin: '*'` 또는 `Access-Control-Allow-Origin:'origin'` 헤더를 반환해야 합니다.

를 사용하여 비프록시 통합을 위한 CORS 활성화 AWS Management Console

를 사용하여 CORS를 활성화할 수 있습니다 AWS Management Console . API Gateway에서 OPTIONS 메서드를 생성하고 기존 메서드 통합 응답에 `Access-Control-Allow-Origin` 헤더를 추가합니다. 이 방법이 항상 효과가 있는 것은 아니며, CORS 사용이 설정된 모든 메서드에서 최소 200개 응답 모두에 대해 `Access-Control-Allow-Origin` 헤더를 반환하도록 통합 응답을 수동으로 수정해야 하는 경우도 있습니다.

프록시 통합을 위한 CORS 지원 사용 설정

Lambda 프록시 통합 또는 HTTP 프록시 통합의 경우 프록시 통합은 통합 응답을 반환하지 않기 때문에 `Access-Control-Allow-Origin`, `Access-Control-Allow-Methods`, `Access-Control-Allow-Headers` 헤더를 반환할 책임이 백엔드에 있습니다.

다음 예제 Lambda 함수는 필요한 CORS 헤더를 반환합니다.

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!')
    }
```

주제

- API Gateway 콘솔을 사용하여 리소스에서 CORS 활성화
- API Gateway 가져오기 API를 사용하여 리소스에서 CORS 활성화
- CORS 테스트

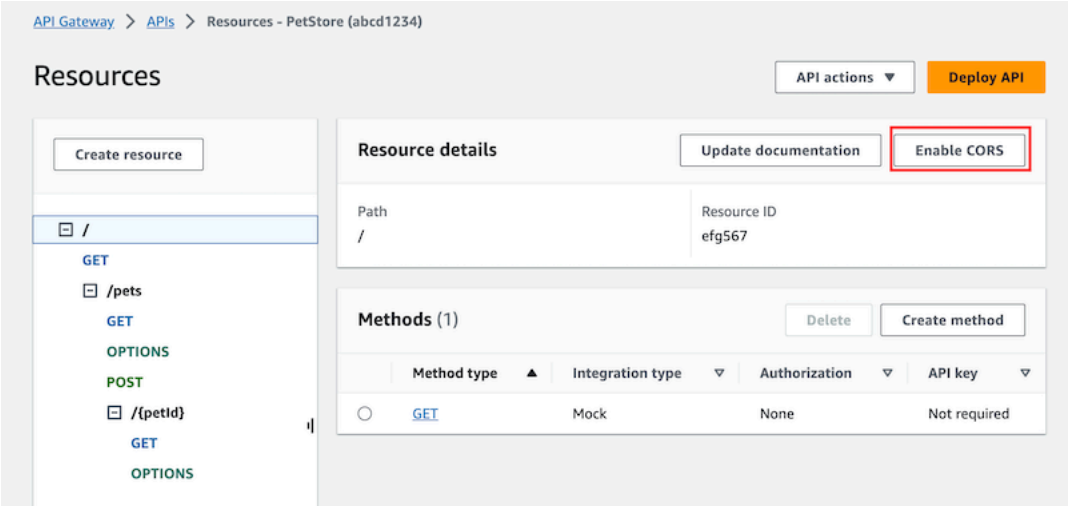
API Gateway 콘솔을 사용하여 리소스에서 CORS 활성화

생성한 REST API 리소스의 한 메서드 또는 모든 메서드에 대해 API Gateway 콘솔을 사용하여 CORS 지원을 활성화할 수 있습니다.

> **Important**
>
> 리소스는 하위 리소스를 포함할 수 있습니다. 리소스와 그 메서드에 대한 CORS 지원 활성화는 하위 리소스와 그 메서드에 대해 재귀적으로 활성화되지 않습니다.

REST API 리소스에 대해 CORS 지원을 활성화하려면

1. https://console.aws.amazon.com/apigateway에서 API Gateway 콘솔에 로그인합니다.
2. API를 선택합니다.
3. 리소스에서 리소스를 선택합니다.
4. 리소스 세부 정보 섹션에서 CORS 활성화를 선택합니다.



5. CORS 활성화 상자에서 다음을 수행합니다.

   a. (선택 사항) 사용자 지정 게이트웨이 응답을 생성한 후 응답에 대한 CORS 지원을 활성화하려면 게이트웨이 응답을 선택합니다.

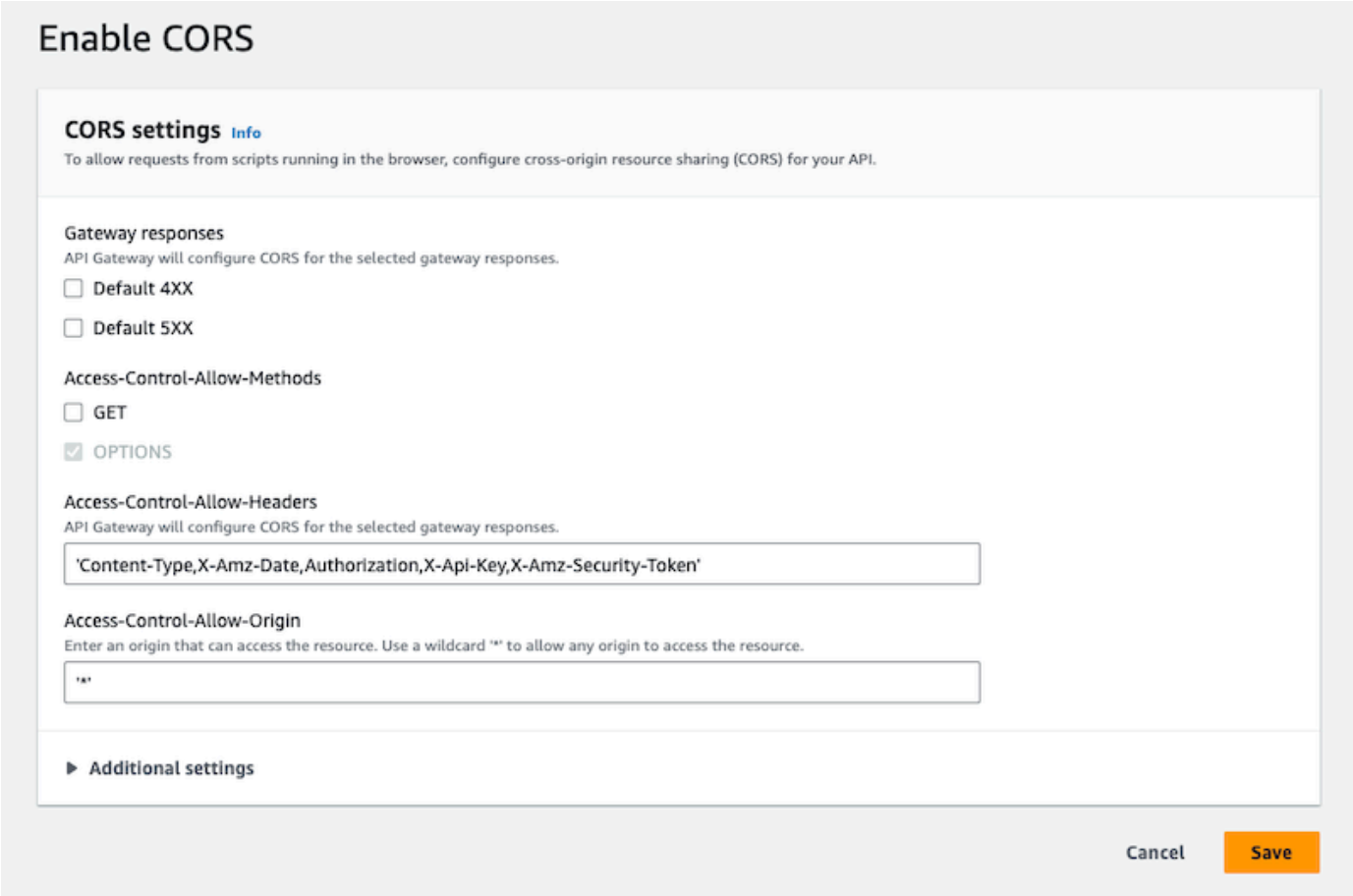
   b. 각 메서드를 선택하여 CORS 지원을 활성화합니다. `OPTION` 메서드에 CORS가 활성화되어 있어야 합니다.

      ANY 메서드에 대해 CORS 지원을 활성화하면 모든 메서드에 대해 CORS가 활성화됩니다.

   c. Access-Control-Allow-Headers 입력 필드에 클라이언트가 실제 리소스 요청 시에 제출해야 할 쉼표로 분리된 헤더 목록의 정적 문자열을 입력합니다. 콘솔에서 제공하는 `'Content-Type,X-Amz-Date,Authorization,X-Api-Key,X-Amz-Security-Token'`의 헤더 목록을 사용하거나 자체 헤더를 사용합니다.

   d. 콘솔에서 제공하는 `'*'` 값을 Access-Control-Allow-Origin 헤더 값으로 사용하여 모든 오리진으로부터 수신되는 액세스 요청을 허용하거나 리소스에 대한 액세스를 허용할 오리진을 지정합니다.

   e. 저장을 선택합니다.



> **Important**
>
> 상기 지침을 프록시 통합에서 ANY 메서드에 적용할 때 어떤 적용 가능한 CORS도 설정하지 않습니다. 대신 백엔드에서 해당 CORS 헤더를 반환해야 합니다(예: `Access-Control-Allow-Origin`).

`GET` 메서드에 대해 CORS가 활성화되면 `OPTIONS` 메서드가 리소스에 추가됩니다(아직 없는 경우). `OPTIONS` 메서드의 `200` 응답은 preflight 핸드셰이크를 수행하기 위해 `Access-Control-Allow-*` 헤더 세 개를 반환하도록 자동으로 구성됩니다. 또한 실제 (`GET`) 메서드는 기본적으로 200 응답에 `Access-Control-Allow-Origin` 헤더를 반환하도록 구성됩니다. 다른 응답 유형의 경우, `Cross-origin access` 오류를 반환하지 않으려면 '*' 또는 특정 오리진을 포함하는 `Access-Control-Allow-Origin'` 헤더를 반환하도록 수동으로 구성해야 합니다.

리소스에 대해 CORS 지원을 활성화한 후 API를 배포하거나 다시 배포해야 새로운 설정이 적용됩니다. 자세한 설명은 the section called “REST API 배포(콘솔)” 섹션을 참조하세요.

> **Note**
>
> 절차를 따른 후에도 리소스에서 CORS 지원을 활성화할 수 없는 경우 CORS 구성을 예제 API 리소스와 비교하는 것이 좋습니다. `/pets` 예제 API를 만드는 방법을 알아보려면 을 참조하십시오. the section called “자습서: 예제를 가져와 REST API 생성”

API Gateway 가져오기 API를 사용하여 리소스에서 CORS 활성화

API Gateway API 가져오기를 사용하는 경우 OpenAPI 파일을 사용하여 CORS 지원을 설정할 수 있습니다. 먼저, 필요한 헤더를 반환하는 리소스에 `OPTIONS` 메서드를 정의해야 합니다.

> **Note**
>
> 웹 브라우저는 CORS 요청을 허용하는 각 API 메서드에서 Access-Control-Allow-Headers 및 Access-Control-Allow-Origin 헤더를 설정할 것을 기대합니다. 또한 일부 브라우저는 먼저 동일한 리소스의 `OPTIONS` 메서드에 HTTP 요청을 한 후 동일한 헤더를 받을 것으로 예상합니다.

`Options` 메서드 예시

다음 예제에서는 모의 통합을 위해 `OPTIONS` 메서드를 생성합니다.

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}"
      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}"
        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"
```

리소스에 대해 OPTIONS 메서드를 구성한 후에는 CORS 요청을 수락해야 하는 동일한 리소스의 다른 메서드에 필요한 헤더를 추가할 수 있습니다.

1. Access-Control-Allow-Origin 및 헤더(Headers)를 응답 유형에 선언합니다.

   OpenAPI 3.0

   ```
     responses:
       200:
         description: Default response for CORS method
         headers:
           Access-Control-Allow-Origin:
             schema:
               type: "string"
           Access-Control-Allow-Methods:
             schema:
               type: "string"
           Access-Control-Allow-Headers:
             schema:
               type: "string"
         content: {}
   ```

   OpenAPI 2.0

   ```
       responses:
         200:
   ```

```
          description: Default response for CORS method
          headers:
            Access-Control-Allow-Headers:
              type: "string"
            Access-Control-Allow-Methods:
              type: "string"
            Access-Control-Allow-Origin:
              type: "string"
```

2. `x-amazon-apigateway-integration` 태그에서 이들 헤더의 정적 값에 대한 매핑을 설정합니다.

   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 : "'*'"
```

예시 API

다음 예시는 OPTIONS 메서드가 포함된 완전한 API 및 HTTP 통합이 있는 GET 메서드를 만듭니다.

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: "when_no_match"
        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: "when_no_match"
        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: "when_no_match"
        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: "when_no_match"
```

```
        type: "mock"
definitions:
  Empty:
    type: "object"
```

CORS 테스트

API를 호출하고 응답에서 CORS 헤더를 확인하여 API의 CORS 구성을 테스트할 수 있습니다. 다음 `curl` 명령은 배포된 API에 OPTIONS 요청을 보냅니다.

```
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
```

응답의 `Access-Control-Allow-Origin`, `Access-Control-Allow-Headers` 및 `Access-Control-Allow-Methods` 헤더는 API가 CORS를 지원함을 나타냅니다. 자세한 내용은 REST API 리소스에 대한 CORS 활성화을(를) 참조하세요.

REST API에 대한 이진 미디어 유형 작업

API Gateway에서 API 요청과 응답은 텍스트 또는 이진 페이로드를 포함합니다. 텍스트 페이로드는 UTF-8 인코딩 형식의 JSON 문자열입니다. 이진 페이로드는 텍스트 페이로드를 제외한 페이로드입니다. 예를 들어 JPEG 파일, GZip 파일, XML 파일 등이 이진 페이로드가 될 수 있습니다. 이진 미디어를 지원하는 데 필요한 API 구성은 API가 프록시 통합을 사용하는지 아니면 비 프록시 통합을 사용하는지에 따라 달라집니다.

AWS Lambda 프록시 통합

AWS Lambda 프록시 통합을 위한 이진 페이로드를 처리하려면 함수의 응답을 base64로 인코딩해야 합니다. 또한 API에 대한 binaryMediaTypes를 구성해야 합니다. API의 `binaryMediaTypes` 구성은

API가 이진 데이터로 처리하는 콘텐츠 유형 목록입니다. 이진 미디어 유형의 예로는 `image/png` 또는 `application/octet-stream` 등이 있습니다. 와일드카드 문자(*)를 사용하여 여러 미디어 유형을 처리할 수 있습니다. 예를 들어 `*/*`에는 모든 콘텐츠 유형이 포함됩니다.

예제 코드는 the section called “Lambda 프록시 통합에서 이진 미디어 반환” 항목을 참조하세요.

비 프록시 통합

비 프록시 통합을 위한 이진 페이로드를 처리하려면 `RestApi` 리소스의 binaryMediaTypes 목록에 미디어 유형을 추가합니다. API의 `binaryMediaTypes` 구성은 API가 이진 데이터로 처리하는 콘텐츠 유형 목록입니다. 또는 통합(Integration) 및 IntegrationResponse 리소스에서 contentHandling 속성을 설정할 수 있습니다. `contentHandling` 값은 `CONVERT_TO_BINARY` 또는 `CONVERT_TO_TEXT`이거나 정의되지 않을 수 있습니다.

`contentHandling` 값과 응답의 `Content-Type` 헤더 또는 수신 요청의 `Accept` 헤더가 `binaryMediaTypes` 목록의 항목과 일치하는지 여부에 따라 API Gateway에서 원시 이진 유형을 base64로 인코딩된 문자열로 인코딩하거나, base64로 인코딩된 문자열을 원시 바이트로 다시 디코딩하거나, 본문을 수정하지 않고 전달할 수 있습니다.

API Gateway에서 API에 대해 이진 페이로드를 지원하려면 API를 다음과 같이 구성해야 합니다.

- RestApi 리소스의 `binaryMediaTypes` 목록에 원하는 이진 미디어 형식을 추가합니다. 이 속성과 `contentHandling` 속성을 정의하지 않은 경우 페이로드는 UTF-8로 인코딩된 JSON 문자열로 처리됩니다.
- Integration 리소스의 `contentHandling` 속성을 설정합니다.
  - 요청 페이로드를 base64로 인코딩된 문자열에서 이진 Blob로 변환하려면 속성을 `CONVERT_TO_BINARY`로 설정합니다.
  - 요청 페이로드를 이진 Blob에서 base64로 인코딩된 문자열로 변환하려면 속성을 `CONVERT_TO_TEXT`로 설정합니다.
  - 페이로드를 수정하지 않고 전달하려면 속성을 정의되지 않은 상태로 둡니다. 이진 페이로드를 수정하지 않고 전달하려면 `Content-Type`이 `binaryMediaTypes` 항목 중 하나와 일치하고 API에 대해 패스스루 동작이 활성화되어 있어야 합니다.
- IntegrationResponse 리소스의 `contentHandling` 속성을 설정합니다. `contentHandling` 속성, 클라이언트 요청의 `Accept` 헤더 및 API `binaryMediaTypes`의 조합에 따라 API Gateway에서 콘텐츠 유형 변환을 처리하는 방법이 결정됩니다. 자세한 내용은 the section called “API Gateway의 콘텐츠 유형 변환” 단원을 참조하세요.

> **Important**
>
> 요청의 `Accept` 헤더에 여러 미디어 유형이 포함되어 있는 경우 API Gateway에서는 첫 번째 `Accept` 미디어 유형만 인식합니다. `Accept` 미디어 유형의 순서를 제어할 수 없고 이진 콘텐츠의 미디어 유형이 목록의 맨 앞에 있지 않은 경우, API의 `binaryMediaTypes` 목록에서 첫 번째 `Accept` 미디어 유형을 추가합니다. API Gateway는 이 목록의 모든 콘텐츠 유형을 이진 수로 처리합니다.
>
> 예를 들어 브라우저에서 `<img>` 요소를 사용하여 JPEG 파일을 보내려는 경우 브라우저에서 요청에 `Accept:image/webp,image/*,*/*;q=0.8`을 전송할 수 있습니다. `image/webp`를 `binaryMediaTypes` 목록에 추가하여 엔드포인트에서 JPEG 파일을 이진 유형으로 수신합니다.

API Gateway에서 텍스트 및 이진 페이로드를 처리하는 방법에 대한 자세한 내용은 API Gateway의 콘텐츠 유형 변환 단원을 참조하세요.

API Gateway의 콘텐츠 유형 변환

API `binaryMediaTypes`, 클라이언트 요청의 헤더 및 통합 `contentHandling` 속성의 조합에 따라 API Gateway에서 페이로드를 인코딩하는 방법이 결정됩니다.

다음 표는 API Gateway에서 요청 `Content-Type` 헤더의 특정 구성, RestApi 리소스의 `binaryMediaTypes` 목록, 통합(Integration) 리소스의 `contentHandling` 속성 값에 대한 요청 페이로드를 변환하는 방법을 보여줍니다.

API Gateway의 API 요청 콘텐츠 유형 변환

메서드 요청 페이로드	요청 **Content-Type** 헤더	**binaryMediaTypes**	**contentHandling**	통합 요청 페이로드
텍스트 데이터	모든 데이터 형식	정의되지 않음	정의되지 않음	UTF8로 인코딩된 문자열
텍스트 데이터	모든 데이터 형식	정의되지 않음	`CONVERT_TO_BINARY`	Base64로 디코딩된 이진 BLOB
텍스트 데이터	모든 데이터 형식	정의되지 않음	`CONVERT_TO_TEXT`	UTF8로 인코딩된 문자열

메서드 요청 페이로드	요청 **Content-Type** 헤더	**binaryMediaTypes**	**contentHandling**	통합 요청 페이로드
텍스트 데이터	텍스트 데이터 형식	일치하는 미디어 유형이 있는 세트	정의되지 않음	텍스트 데이터
텍스트 데이터	텍스트 데이터 형식	일치하는 미디어 유형이 있는 세트	CONVERT_TO_BINARY	Base64로 디코딩된 이진 BLOB
텍스트 데이터	텍스트 데이터 형식	일치하는 미디어 유형이 있는 세트	CONVERT_TO_TEXT	텍스트 데이터
이진 데이터	이진 데이터 형식	일치하는 미디어 유형이 있는 세트	정의되지 않음	이진 데이터
이진 데이터	이진 데이터 형식	일치하는 미디어 유형이 있는 세트	CONVERT_TO_BINARY	이진 데이터
이진 데이터	이진 데이터 형식	일치하는 미디어 유형이 있는 세트	CONVERT_TO_TEXT	Base64로 인코딩된 문자열

다음 표는 API Gateway에서 요청 `Accept` 헤더의 특정 구성, RestApi 리소스의 `binaryMediaTypes` 목록, IntegrationResponse 리소스의 `contentHandling` 속성 값에 대한 응답 페이로드를 변환하는 방법을 보여줍니다.

> **Important**
>
> 요청의 `Accept` 헤더에 여러 미디어 유형이 포함되어 있는 경우 API Gateway에서는 첫 번째 `Accept` 미디어 유형만 인식합니다. `Accept` 미디어 유형의 순서를 제어할 수 없고 이진 콘텐츠의 미디어 유형이 목록의 맨 앞에 있지 않은 경우, API의 `binaryMediaTypes` 목록에서 첫 번째 `Accept` 미디어 유형을 추가합니다. API Gateway는 이 목록의 모든 콘텐츠 유형을 이진 수로 처리합니다.
>
> 예를 들어 브라우저에서 `<img>` 요소를 사용하여 JPEG 파일을 보내려는 경우 브라우저에서 요청에 `Accept:image/webp,image/*,*/*;q=0.8`을 전송할 수 있습니다. `image/webp`를 `binaryMediaTypes` 목록에 추가하여 엔드포인트에서 JPEG 파일을 이진 유형으로 수신합니다.

API Gateway 응답 콘텐츠 유형 변환

통합 응답 페이로드	요청 **Accept** 헤더	**binaryMediaTypes**	**contentHandling**	메서드 응답 페이로드
텍스트 또는 이진 데이터	텍스트 유형	정의되지 않음	정의되지 않음	UTF8로 인코딩된 문자열
텍스트 또는 이진 데이터	텍스트 유형	정의되지 않음	CONVERT_TO_BINARY	Base64로 디코딩된 BLOB
텍스트 또는 이진 데이터	텍스트 유형	정의되지 않음	CONVERT_TO_TEXT	UTF8로 인코딩된 문자열
텍스트 데이터	텍스트 유형	일치하는 미디어 유형이 있는 세트	정의되지 않음	텍스트 데이터
텍스트 데이터	텍스트 유형	일치하는 미디어 유형이 있는 세트	CONVERT_TO_BINARY	Base64로 디코딩된 BLOB
텍스트 데이터	텍스트 유형	일치하는 미디어 유형이 있는 세트	CONVERT_TO_TEXT	UTF8로 인코딩된 문자열
텍스트 데이터	이진수 형식	일치하는 미디어 유형이 있는 세트	정의되지 않음	Base64로 디코딩된 BLOB
텍스트 데이터	이진수 형식	일치하는 미디어 유형이 있는 세트	CONVERT_TO_BINARY	Base64로 디코딩된 BLOB
텍스트 데이터	이진수 형식	일치하는 미디어 유형이 있는 세트	CONVERT_TO_TEXT	UTF8로 인코딩된 문자열
이진 데이터	텍스트 유형	일치하는 미디어 유형이 있는 세트	정의되지 않음	Base64로 인코딩된 문자열
이진 데이터	텍스트 유형	일치하는 미디어 유형이 있는 세트	CONVERT_TO_BINARY	이진 데이터
이진 데이터	텍스트 유형	일치하는 미디어 유형이 있는 세트	CONVERT_TO_TEXT	Base64로 인코딩된 문자열

통합 응답 페이로드	요청 **Accept** 헤더	**binaryMediaTypes**	**contentHandling**	메서드 응답 페이로드
이진 데이터	이진수 형식	일치하는 미디어 유형이 있는 세트	정의되지 않음	이진 데이터
이진 데이터	이진수 형식	일치하는 미디어 유형이 있는 세트	CONVERT_TO_BINARY	이진 데이터
이진 데이터	이진수 형식	일치하는 미디어 유형이 있는 세트	CONVERT_TO_TEXT	Base64로 인코딩된 문자열

텍스트 페이로드를 이진 BLOB으로 변환할 때 API Gateway에서는 텍스트 데이터가 Base64로 인코딩된 문자열인 것으로 간주하고 이진 데이터를 Base64로 디코딩된 BLOB으로 출력합니다. 변환에 실패하면 API 구성 오류를 나타내는 `500` 응답이 반환됩니다. API에 대해 패스스루 동작을 활성화해야 하되, 이러한 변환에 대해 매핑 템플릿을 제공하지 않습니다.

이진 페이로드를 텍스트 문자열로 변환할 경우 API Gateway에서는 항상 이진 데이터에 Base64 인코딩을 적용합니다. 다음과 같은 예제 매핑 템플릿에 표시된 것처럼 이러한 페이로드에 대해 매핑 템플릿을 정의할 수 있지만, `$input.body`를 통해서만 매핑 템플릿의 Base64로 인코딩된 문자열에 액세스할 수 있습니다.

```
{
    "data": "$input.body"
}
```

이진 페이로드를 수정하지 않고 전달하려면 API에서 패스스루 동작을 활성화해야 합니다.

API Gateway 콘솔을 사용하여 이진 지원 활성화

이 단원에서는 API Gateway 콘솔을 사용하여 이진 지원을 활성화하는 방법을 설명합니다. 예를 들어 Amazon S3에 통합된 API를 사용합니다. 지원되는 미디어 유형을 설정하고 페이로드를 처리하는 방법을 지정하는 작업을 중점적으로 다룹니다. Amazon S3에 통합된 API를 생성하는 방법에 대한 자세한 내용은 자습서: API Gateway에서 REST API를 Amazon S3 프록시로 생성 단원을 참조하세요.

API Gateway 콘솔을 사용하여 이진 지원을 활성화하려면

1. API에 이진 미디어 형식 설정:

a. 새 API를 생성하거나 기존 API를 선택합니다. 이 예에서는 API 이름을 `FileMan`으로 지정합니다.

b. 기본 탐색 창에서 선택한 API 아래에서 API 설정을 선택합니다.

c. API 설정 창의 이진 미디어 형식 섹션에서 미디어 형식 관리를 선택합니다.

d. 이진 미디어 형식 추가를 선택합니다.

e. 입력 텍스트 필드에 필요한 미디어 형식(예: **`image/png`**)을 입력합니다. 필요한 경우 이 단계를 반복하여 미디어 유형을 더 추가합니다. 모든 이진수 미디어 유형을 지원하려면 `*/*`을(를) 지정합니다.

f. 변경 사항 저장을 선택합니다.

2. API 메서드에 메시지 페이로드가 처리되는 방식을 설정합니다.

a. 새 리소스를 생성하거나 API에서 기존 리소스를 선택합니다. 이 예에서는 `/{folder}/{item}` 리소스를 사용합니다.

b. 새 메서드를 생성하거나 리소스에서 기존 메서드를 선택합니다. 예를 들어 Amazon S3에서 `GET /{folder}/{item}` 작업과 통합된 `Object GET` 메서드를 사용합니다.

c. 콘텐츠 처리에서 옵션을 선택합니다.

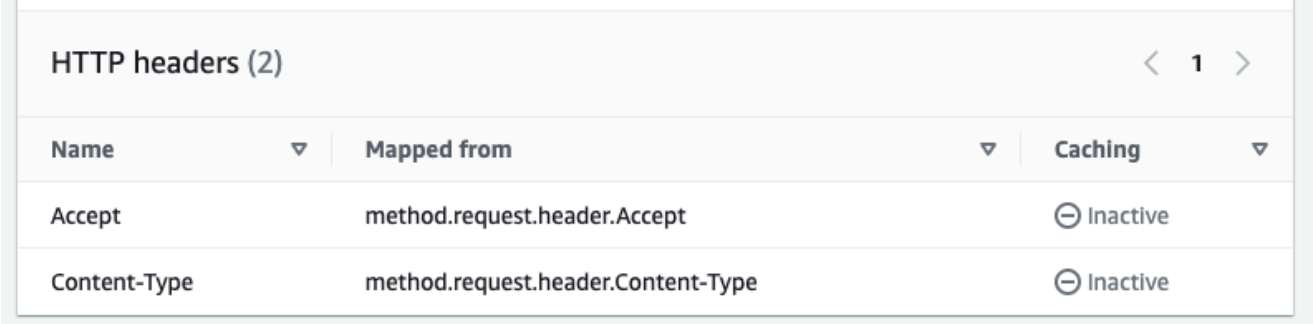
클라이언트와 백엔드가 동일한 이진 형식을 수락하는 경우 본문을 변환하고 싶지 않다면 패스스루(Passthrough)를 선택합니다. 백엔드에서 이진 요청 페이로드를 JSON 속성으로 전달

받기를 요구하는 등의 경우, 텍스트로 변환을 선택하여 이진 본문을 Base64로 인코딩된 문자열로 변환합니다. 클라이언트가 Base64로 인코딩된 문자열을 제출하고 백엔드에서 원래 이진 형식을 요구하는 경우나 엔드포인트가 Base64로 인코딩된 문자열을 반환하고 클라이언트가 이진 출력만 수락하는 경우에는 이진으로 변환을 선택합니다.

d. 요청 본문 패스스루에서 정의된 템플릿이 없는 경우(권장)를 선택하여 요청 본문에 패스스루 동작을 활성화합니다.

   안 함을 선택할 수도 있습니다. 이렇게 하면 API가 매핑 템플릿이 없는 content-types의 데이터를 거부합니다.

e. 수신되는 요청의 `Accept` 헤더를 통합 요청에 보존합니다. `contentHandling`을 `passthrough`로 설정하고 실행 시간에 이 설정을 재정의하고 싶은 경우 이렇게 해야 합니다.



f. 텍스트로 전환하려면 필요한 형식에 Base64로 인코딩된 이진 데이터를 입력하도록 매핑 템플릿을 정의합니다.

   텍스트로 변환할 매핑 템플릿의 예는 다음과 같습니다.

```
{
  "operation": "thumbnail",
  "base64Image": "$input.body"
}
```

   이 매핑 템플릿의 형식은 입력 항목의 엔드포인트 요구 사항에 따라 결정됩니다.

g. 저장을 선택합니다.

API Gateway REST API를 사용하여 이진 지원 활성화

다음 작업에서는 API Gateway REST API 호출을 사용하여 이진 지원을 활성화하는 방법을 보여 줍니다.

주제

- 지원되는 이진 미디어 지원을 API에 추가 및 업데이트
- 요청 페이로드 변환 구성
- 응답 페이로드 변환 구성
- 이진 데이터를 텍스트 데이터로 변환
- 텍스트 데이터를 이진 페이로드로 변환
- 이진 페이로드 패스스루

지원되는 이진 미디어 지원을 API에 추가 및 업데이트

API Gateway에서 새로운 이진 미디어 유형을 지원하도록 활성화하려면 이진 미디어 유형을 `RestApi` 리소스의 `binaryMediaTypes` 목록에 추가해야 합니다. 예를 들어 API Gateway에서 JPEG 이미지를 처리하도록 하려면 `PATCH` 요청을 `RestApi` 리소스에 제출합니다.

```
PATCH /restapis/<restapi_id>

{
  "patchOperations" : [ {
    "op" : "add",
    "path" : "/binaryMediaTypes/image~1jpeg"
  }
 ]
}
```

`image/jpeg` 속성 값의 일부인 `path`의 MIME 유형 사양은 `image~1jpeg`로 이스케이프됩니다.

지원되는 이진 미디어 유형을 업데이트하려면 `binaryMediaTypes` 리소스의 `RestApi` 목록에서 미디어 유형을 교체하거나 제거합니다. 예를 들어 이진 지원을 JPEG 파일에서 원시 바이트로 변경하려면 다음과 같이 `PATCH` 요청을 `RestApi` 리소스에 제출합니다.

```
PATCH /restapis/<restapi_id>

{
  "patchOperations" : [{
    "op" : "replace",
    "path" : "/binaryMediaTypes/image~1jpeg",
    "value" : "application/octet-stream"
  },
  {
```

```
    "op" : "remove",
    "path" : "/binaryMediaTypes/image~1jpeg"
  }]
}
```

요청 페이로드 변환 구성

엔드포인트에 이진 입력이 필요한 경우 contentHandling 리소스의 Integration 속성을 CONVERT_TO_BINARY로 설정합니다. 이렇게 하려면 다음과 같이 PATCH 요청을 제출합니다.

```
PATCH /restapis/<restapi_id>/resources/<resource_id>/methods/<http_method>/integration

{
  "patchOperations" : [ {
    "op" : "replace",
    "path" : "/contentHandling",
    "value" : "CONVERT_TO_BINARY"
  }]
}
```

응답 페이로드 변환 구성

클라이언트에서 엔드포인트에서 반환되는 Base64로 인코딩된 페이로드가 아닌 이진 BLOB 형태의 결과를 수락하는 경우, IntegrationResponse 리소스의 contentHandling 속성을 CONVERT_TO_BINARY로 설정합니다. 이렇게 하려면 다음과 같이 PATCH 요청을 제출합니다.

```
PATCH /restapis/<restapi_id>/resources/<resource_id>/methods/<http_method>/integration/
responses/<status_code>

{
  "patchOperations" : [ {
    "op" : "replace",
    "path" : "/contentHandling",
    "value" : "CONVERT_TO_BINARY"
  }]
}
```

이진 데이터를 텍스트 데이터로 변환

API Gateway를 통해 이진 데이터를 AWS Lambda 또는 Kinesis에 대한 입력의 JSON 속성으로 전송하려면 다음을 수행합니다.

1. application/octet-stream의 새 이진 미디어 유형을 API의 binaryMediaTypes 목록에 추가하여 API의 이진 페이로드 지원을 활성화합니다.

```
PATCH /restapis/<restapi_id>

{
  "patchOperations" : [ {
    "op" : "add",
    "path" : "/binaryMediaTypes/application~1octet-stream"
  }
 ]
}
```

2. Integration 리소스의 contentHandling 속성에서 CONVERT_TO_TEXT를 설정하고 매핑 템플릿을 제공하여 이진 데이터의 Base64로 인코딩된 문자열을 JSON 속성에 할당합니다. 다음 예제에서 JSON 속성은 body이고 $input.body는 Base64로 인코딩된 문자열을 보유합니다.

```
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\"}"
    }
  ]
}
```

텍스트 데이터를 이진 페이로드로 변환

Lambda 함수가 이미지 파일을 Base64로 인코딩된 문자열로 반환한다고 가정합니다. API Gateway를 통해 이 이진 출력을 클라이언트에 전달하려면 다음을 수행합니다.

1. binaryMediaTypes의 이진 미디어 유형을 추가하여(목록에 아직 없는 경우) API의 application/octet-stream 목록을 업데이트합니다.

```
PATCH /restapis/<restapi_id>

{
  "patchOperations" : [ {
    "op" : "add",
    "path" : "/binaryMediaTypes/application~1octet-stream",
  }]
}
```

2. contentHandling 리소스의 Integration 속성을 CONVERT_TO_BINARY로 설정합니다. 매핑 템플릿을 정의하지 마세요. 매핑 템플릿을 정의하지 않을 경우, API Gateway에서 패스스루 템플릿을 호출하여 Base64로 디코딩된 이진 BLOB을 클라이언트에 이미지 파일로 반환합니다.

```
PATCH /restapis/<restapi_id>/resources/<resource_id>/methods/<http_method>/
integration/responses/<status_code>

{
  "patchOperations" : [
    {
      "op" : "replace",
      "path" : "/contentHandling",
      "value" : "CONVERT_TO_BINARY"
    }
  ]
}
```

이진 페이로드 패스스루

API Gateway를 사용하여 Amazon S3 버킷에 이미지를 저장하려면 다음을 수행합니다.

1. binaryMediaTypes의 이진 미디어 유형을 추가하여(목록에 아직 없는 경우) API의 application/octet-stream 목록을 업데이트합니다.

```
PATCH /restapis/<restapi_id>

{
  "patchOperations" : [ {
    "op" : "add",
```

```
    "path" : "/binaryMediaTypes/application~1octet-stream"
  }
 ]
}
```

2. contentHandling 리소스의 Integration 속성을 CONVERT_TO_BINARY로 설정합니다. 매핑 템플릿을 정의하지 않고 WHEN_NO_MATCH를 passthroughBehavior 속성 값으로 설정합니다. 그러면 API Gateway에서 패스스루 템플릿을 호출할 수 있습니다.

```
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"
    }
  ]
}
```

콘텐츠 인코딩 가져오기 및 내보내기

RestApi에서 binaryMediaTypes 목록을 가져오려면 API의 OpenAPI 정의 파일에 대해 다음 API Gateway 확장을 사용합니다. 이 확장은 API 설정을 내보내는 데에도 사용됩니다.

- x-amazon-apigateway-binary-미디어 유형 속성

Integration 또는 IntegrationResponse 리소스에서 contentHandling 속성 값을 가져오거나 내보내려면 OpenAPI 정의에 대해 다음 API Gateway 확장을 사용합니다.

- x-amazon-apigateway-integration 객체
- x-amazon-apigateway-integration.response 객체

이진 지원 예제

다음 예에서는 API Gateway API를 통해 Amazon S3 또는 AWS Lambda에서 이진 파일에 액세스하는 방법을 보여줍니다.

주제

- Lambda 프록시 통합에서 이진 미디어 반환
- API Gateway API를 통해 Amazon S3에서 이진 파일에 액세스
- API Gateway API를 사용하여 Lambda의 이진 파일에 액세스

Lambda 프록시 통합에서 이진 미디어 반환

AWS Lambda 프록시 통합에서 이진 미디어를 반환하려면 Lambda 함수의 응답을 base64로 인코딩합니다. 또한 API의 이진 미디어 유형을 구성해야 합니다. 페이로드 크기 제한은 10MB입니다.

> **Note**
>
> 웹 브라우저를 사용하여 이 예제 통합이 포함된 API를 호출하려면 API의 이진 미디어 유형을 `*/*`로 설정합니다. API Gateway는 클라이언트의 첫 번째 `Accept` 헤더를 사용하여 응답이 이진 미디어를 반환해야 하는지 결정합니다. 브라우저의 요청과 같이 `Accept` 헤더 값의 순서를 제어할 수 없을 때 이진 미디어를 반환하려면 API의 이진 미디어 유형을 `*/*`(모든 콘텐츠 유형에 대해)로 설정합니다.

다음 예시 Lambda 함수는 Amazon S3의 이진 이미지나 텍스트를 클라이언트에 반환할 수 있습니다. 함수의 응답에는 클라이언트에 반환하는 데이터 유형을 나타내는 `Content-Type` 헤더가 포함됩니다. 이 함수는 반환하는 데이터 유형에 따라 응답에 `isBase64Encoded` 속성을 조건부로 설정합니다.

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>",
        }
```

이진 미디어 유형에 대한 자세한 내용은 REST API에 대한 이진 미디어 유형 작업 단원을 참조하세요.

API Gateway API를 통해 Amazon S3에서 이진 파일에 액세스

다음 예에서는 Amazon S3에서 이미지에 액세스하는 데 사용되는 OpenAPI 파일, Amazon S3에서 이미지를 다운로드하는 방법, Amazon S3에 이미지를 업로드하는 방법을 보여줍니다.

주제

- Amazon S3에서 이미지에 액세스하는 샘플 API의 OpenAPI 파일
- Amazon S3에서 이미지 다운로드
- Amazon S3에 이미지 업로드

Amazon S3에서 이미지에 액세스하는 샘플 API의 OpenAPI 파일

다음 OpenAPI 파일은 Amazon S3에서 이미지 파일을 다운로드하거나 Amazon S3에 업로드하는 방법을 설명하는 샘플 API를 보여줍니다. 이 API는 지정된 이미지 파일을 다운로드하거나 업로드하기 위한 `GET /s3?key={file-name}` 및 `PUT /s3?key={file-name}` 메서드를 노출합니다. `GET` 메서드는 200 OK 응답에서 제공된 매핑 템플릿에 따라 JSON 출력의 일부로서 이미지 파일을 Base64로 인코딩된 문자열 형태로 반환합니다. `PUT` 메서드는 원시 이진 BLOB을 입력으로 사용하여 빈 페이로드를 포함하는 200 OK 응답을 반환합니다.

OpenAPI 3.0

```
{
```

```
"openapi": "3.0.0",
"info": {
   "version": "2016-10-21T17:26:28Z",
   "title": "ApiName"
},
"paths": {
   "/s3": {
      "get": {
         "parameters": [
            {
               "name": "key",
               "in": "query",
               "required": false,
               "schema": {
                  "type": "string"
               }
            }
         ],
         "responses": {
            "200": {
               "description": "200 response",
               "content": {
                  "application/json": {
                     "schema": {
                        "$ref": "#/components/schemas/Empty"
                     }
                  }
               }
            },
            "500": {
               "description": "500 response"
            }
         },
         "x-amazon-apigateway-integration": {
            "credentials": "arn:aws:iam::123456789012:role/binarySupportRole",
            "responses": {
               "default": {
                  "statusCode": "500"
               },
               "2\\d{2}": {
                  "statusCode": "200"
               }
            },
            "requestParameters": {
```

```
                "integration.request.path.key": "method.request.querystring.key"
            },
            "uri": "arn:aws:apigateway:us-west-2:s3:path/{key}",
            "passthroughBehavior": "when_no_match",
            "httpMethod": "GET",
            "type": "aws"
        }
    },
    "put": {
        "parameters": [
            {
                "name": "key",
                "in": "query",
                "required": false,
                "schema": {
                    "type": "string"
                }
            }
        ],
        "responses": {
            "200": {
                "description": "200 response",
                "content": {
                    "application/json": {
                        "schema": {
                            "$ref": "#/components/schemas/Empty"
                        }
                    },
                    "application/octet-stream": {
                        "schema": {
                            "$ref": "#/components/schemas/Empty"
                        }
                    }
                }
            },
            "500": {
                "description": "500 response"
            }
        },
        "x-amazon-apigateway-integration": {
            "credentials": "arn:aws:iam::123456789012:role/binarySupportRole",
            "responses": {
                "default": {
                    "statusCode": "500"
```

```
                    },
                    "2\\d{2}": {
                       "statusCode": "200"
                    }
                 },
                 "requestParameters": {
                    "integration.request.path.key": "method.request.querystring.key"
                 },
                 "uri": "arn:aws:apigateway:us-west-2:s3:path/{key}",
                 "passthroughBehavior": "when_no_match",
                 "httpMethod": "PUT",
                 "type": "aws",
                 "contentHandling": "CONVERT_TO_BINARY"
              }
           }
        }
     },
     "x-amazon-apigateway-binary-media-types": [
        "application/octet-stream",
        "image/jpeg"
     ],
     "servers": [
        {
           "url": "https://abcdefghi.execute-api.us-east-1.amazonaws.com/{basePath}",
           "variables": {
              "basePath": {
                "default": "/v1"
              }
           }
        }
     ],
     "components": {
        "schemas": {
           "Empty": {
              "type": "object",
              "title": "Empty Schema"
           }
        }
     }
}
```

OpenAPI 2.0

```
{
  "swagger": "2.0",
  "info": {
    "version": "2016-10-21T17:26:28Z",
    "title": "ApiName"
  },
  "host": "abcdefghi.execute-api.us-east-1.amazonaws.com",
  "basePath": "/v1",
  "schemes": [
    "https"
  ],
  "paths": {
    "/s3": {
      "get": {
        "produces": [
          "application/json"
        ],
        "parameters": [
          {
            "name": "key",
            "in": "query",
            "required": false,
            "type": "string"
          }
        ],
        "responses": {
          "200": {
            "description": "200 response",
            "schema": {
              "$ref": "#/definitions/Empty"
            }
          },
          "500": {
            "description": "500 response"
          }
        },
        "x-amazon-apigateway-integration": {
          "credentials": "arn:aws:iam::123456789012:role/binarySupportRole",
          "responses": {
            "default": {
              "statusCode": "500"
            },
```

```
            "2\\d{2}": {
              "statusCode": "200"            }
          },
          "requestParameters": {
            "integration.request.path.key": "method.request.querystring.key"
          },
          "uri": "arn:aws:apigateway:us-west-2:s3:path/{key}",
          "passthroughBehavior": "when_no_match",
          "httpMethod": "GET",
          "type": "aws"
        }
      },
      "put": {
        "produces": [
          "application/json", "application/octet-stream"
        ],
        "parameters": [
          {
            "name": "key",
            "in": "query",
            "required": false,
            "type": "string"
          }
        ],
        "responses": {
          "200": {
            "description": "200 response",
            "schema": {
              "$ref": "#/definitions/Empty"
            }
          },
          "500": {
            "description": "500 response"
          }
        },
        "x-amazon-apigateway-integration": {
          "credentials": "arn:aws:iam::123456789012:role/binarySupportRole",
          "responses": {
            "default": {
              "statusCode": "500"
            },
            "2\\d{2}": {
              "statusCode": "200"
            }
```

```
          },
          "requestParameters": {
            "integration.request.path.key": "method.request.querystring.key"
          },
          "uri": "arn:aws:apigateway:us-west-2:s3:path/{key}",
          "passthroughBehavior": "when_no_match",
          "httpMethod": "PUT",
          "type": "aws",
          "contentHandling" : "CONVERT_TO_BINARY"
        }
      }
    }
  },
  "x-amazon-apigateway-binary-media-types" : ["application/octet-stream", "image/
jpeg"],
  "definitions": {
    "Empty": {
      "type": "object",
      "title": "Empty Schema"
    }
  }
}
```

Amazon S3에서 이미지 다운로드

Amazon S3에서 이미지 파일(`image.jpg`)을 이진 BLOB으로 다운로드하려면

```
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
```

성공적인 응답은 다음과 같습니다.

```
200 OK HTTP/1.1

[raw bytes]
```

`Accept` 헤더가 `application/octet-stream`의 이진 미디어 유형으로 설정되고 API에 대해 이진 지원이 활성화되어 있으므로 원시 바이트가 반환됩니다.

또는 Amazon S3에서 이미지 파일(`image.jpg`)을 JSON 속성으로 형식 지정되고 Base64로 인코딩된 문자열로 다운로드하려면, 아래 굵은 글꼴의 OpenAPI 정의 블록에서와 같이 200 통합 응답에 응답 템플릿을 추가합니다.

```
        "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}"
              }
            }
          },
```

이미지 파일 다운로드 요청은 다음과 같습니다.

```
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
```

성공적인 응답은 다음과 같습니다.

```
200 OK HTTP/1.1

{
  "image": "W3JhdyBieXRlc10="
}
```

Amazon S3에 이미지 업로드

이미지 파일(`image.jpg`)을 Amazon S3에 이진 BLOB으로 업로드하려면

```
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]
```

성공적인 응답은 다음과 같습니다.

```
200 OK HTTP/1.1
```

이미지 파일(`image.jpg`)을 Amazon S3에 Base64로 인코딩된 문자열로 업로드하려면

```
PUT /v1/s3?key=image.jpg HTTP/1.1
Host: abcdefghi.execute-api.us-east-1.amazonaws.com
Content-Type: application/json
Accept: application/json

W3JhdyBieXRlc10=
```

`Content-Type` 헤더 값이 `application/json`으로 설정되어 있으므로 입력 페이로드는 Base64로 인코딩된 문자열이어야 합니다. 성공적인 응답은 다음과 같습니다.

```
200 OK HTTP/1.1
```

API Gateway API를 사용하여 Lambda의 이진 파일에 액세스

다음 예에서는 API Gateway API를 통해 AWS Lambda에서 이진 파일에 액세스하는 방법을 보여줍니다. 샘플 API는 OpenAPI 파일에 제공됩니다. 이 코드 예제에서는 API Gateway REST API 호출을 사용합니다.

주제

- Lambda에서 이미지에 액세스하는 샘플 API의 OpenAPI 파일
- Lambda에서 이미지 다운로드
- Lambda에 이미지 업로드

Lambda에서 이미지에 액세스하는 샘플 API의 OpenAPI 파일

다음 OpenAPI 파일은 Lambda에서 이미지 파일을 다운로드하거나 Lambda에 업로드하는 방법을 설명하는 예제 API를 보여줍니다.

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"
    }
  }
}
```

Lambda에서 이미지 다운로드

Lambda에서 이미지 파일(image.jpg)을 이진 BLOB으로 다운로드하려면

```
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
```

성공적인 응답은 다음과 같습니다.

```
200 OK HTTP/1.1

[raw bytes]
```

Lambda에서 이미지 파일(image.jpg)을 JSON 속성으로 형식 지정되고 Base64로 인코딩된 문자열로 다운로드하려면

```
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
```

성공적인 응답은 다음과 같습니다.

```
200 OK HTTP/1.1

{
  "image": "W3JhdyBieXRlc10="
}
```

Lambda에 이미지 업로드

이미지 파일(image.jpg)을 Lambda에 이진 BLOB으로 업로드하려면

```
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]
```

성공적인 응답은 다음과 같습니다.

```
200 OK
```

이미지 파일(image.jpg)을 Lambda에 base64로 인코딩된 문자열로 업로드하려면

```
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=
```

성공적인 응답은 다음과 같습니다.

```
200 OK
```

Amazon API Gateway에서 REST API 호출

배포된 API를 호출하기 위해 클라이언트는 API 실행을 위한 API Gateway 구성 요소 서비스(예: `execute-api`)에 대한 URL 요청을 제출합니다.

기본 REST API URL의 형식은 다음과 같습니다.

```
https://restapi_id.execute-api.region.amazonaws.com/stage_name/
```

여기서 *restapi_id#* API 식별자, *### ##*, *stage_name# API ### ## #####*. AWS

> **Important**
>
> API를 호출하기 전에 API Gateway에서 배포해야 합니다. API 배포에 대한 지침은 을 참조하십시오. Amazon API Gateway에서 REST API 배포

주제

- API의 호출 URL 가져오기
- API 호출
- API Gateway 콘솔을 사용하여 REST API 메서드 테스트
- 생성된 SDK를 통해 REST API 호출
- 프라이빗 API를 호출하는 방법

API의 호출 URL 가져오기

콘솔 AWS CLI, 또는 내보낸 OpenAPI 정의를 사용하여 API의 호출 URL을 가져올 수 있습니다.

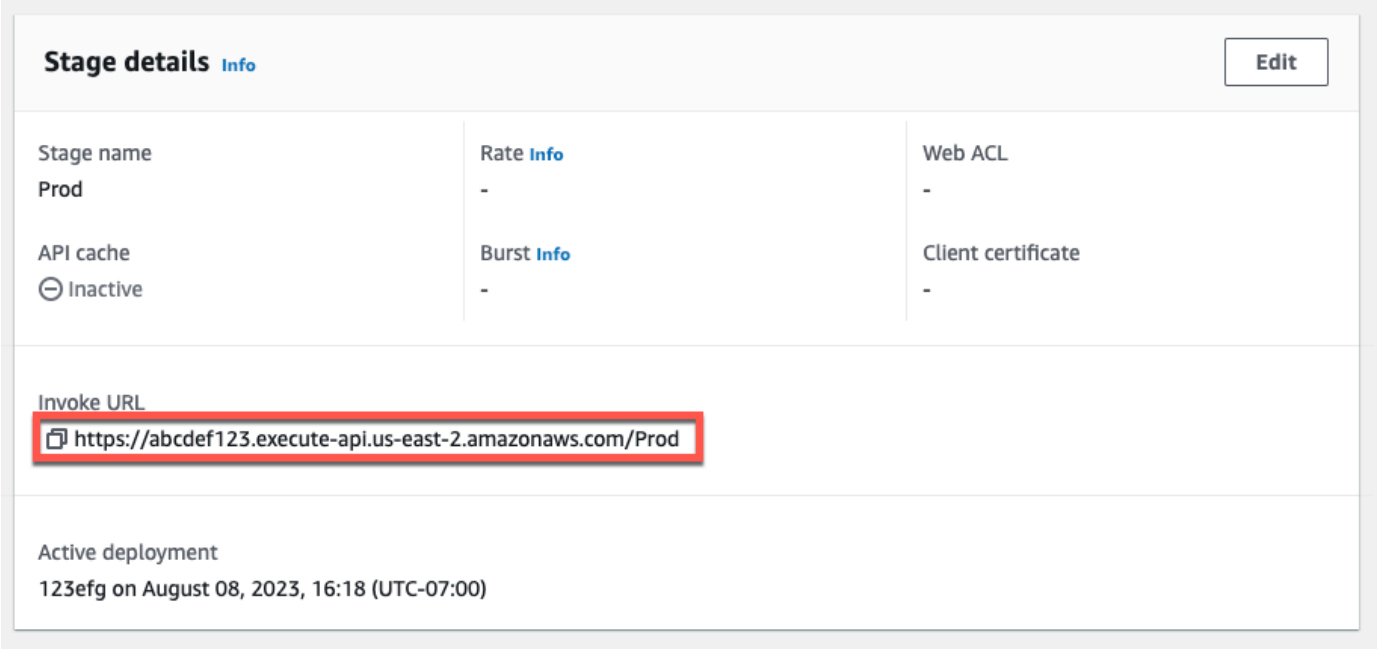
콘솔을 사용하여 API의 호출 URL 가져오기

다음 절차는 REST API 콘솔에서 API의 호출 URL을 가져오는 방법을 보여줍니다.

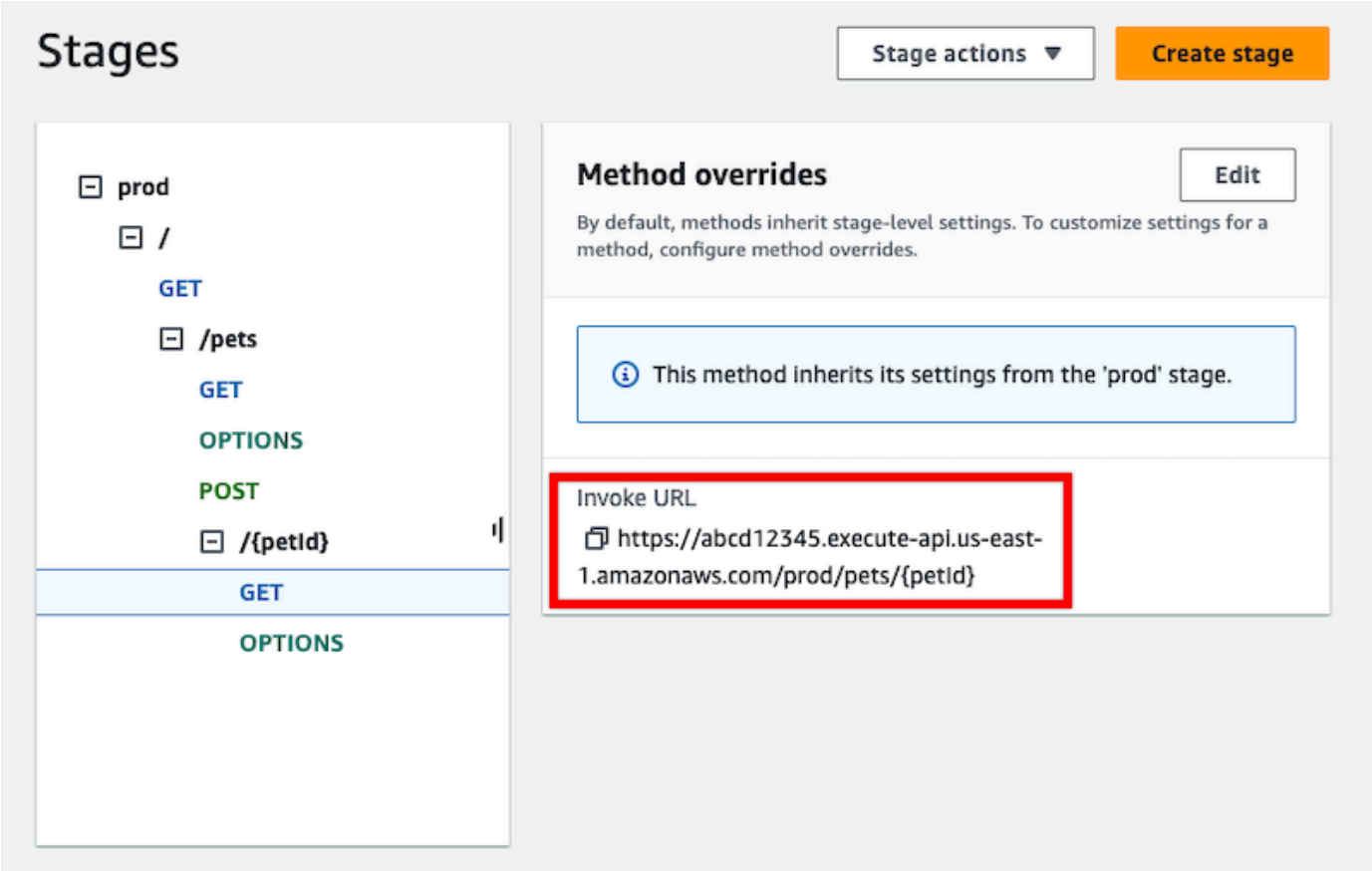
REST API 콘솔을 사용하여 API의 호출 URL을 가져오려면

1. https://console.aws.amazon.com/apigateway에서 API Gateway 콘솔에 로그인합니다.
2. 배포된 API를 선택합니다.
3. 기본 탐색 창에서 스테이지를 선택합니다.
4. 스테이지 세부 정보에서 복사 아이콘을 선택하여 API의 호출 URL을 복사합니다.

   이 URL은 API의 루트 리소스용입니다.



5. API의 다른 리소스에 대한 API의 호출 URL을 가져오려면 보조 탐색 창 아래의 단계를 확장한 다음 메서드를 선택합니다.
6. 복사 아이콘을 선택하여 API의 리소스 수준 호출 URL을 복사합니다.



를 사용하여 API의 호출 URL 가져오기 AWS CLI

다음 절차는 를 사용하여 API의 호출 URL을 얻는 방법을 보여줍니다. AWS CLI

를 사용하여 API의 호출 URL을 얻으려면 AWS CLI

1. 다음 명령을 사용하여 을 (를) 가져올 수 있습니다`rest-api-id`. 이 명령은 해당 지역의 모든 `rest-api-id` 값을 반환합니다. 자세한 내용은 을 참조하십시오 get-rest-apis.

```
aws apigateway get-rest-apis
```

2. 예제를 `rest-api-id` y로 바꾸고`rest-api-id`, 예제 *`{stage-name}`* `#` *`{stage-name}`* `##` *`###`*`,` *`{region}`* `#` `##` 지역으로 바꾸십시오.

```
https://{restapi_id}.execute-api.{region}.amazonaws.com/{stage_name}/
```

내보낸 API의 OpenAPI 정의 파일을 사용하여 API의 호출 URL 가져오기

내보낸 API의 OpenAPI 정의 파일의 host 및 basePath 필드를 결합하여 루트 URL을 생성할 수도 있습니다. API를 내보내는 방법에 대한 지침은 을 참조하십시오the section called “REST API 내보내기”.

API 호출

브라우저, curl 또는 Postman과 같은 기타 애플리케이션을 사용하여 배포된 API를 호출할 수 있습니다.

또한 API Gateway 콘솔을 사용하여 API 호출을 테스트할 수 있습니다. 테스트에서는 API Gateway의 TestInvoke 기능을 사용하므로 API를 배포하기 전에 API를 테스트할 수 있습니다. 자세한 설명은 the section called “콘솔을 사용하여 REST API 메서드 테스트” 섹션을 참조하세요.

> **Note**
>
> 호출 URL의 쿼리 문자열 파라미터 값은 %%를 포함할 수 없습니다.

웹 브라우저를 사용하여 API 호출

API에서 익명 액세스를 허용하는 경우 모든 웹 브라우저를 사용하여 모든 메서드를 호출할 수 있습니다. GET 브라우저의 주소 표시줄에 전체 호출 URL을 입력합니다.

다른 방법이나 인증이 필요한 호출의 경우 페이로드를 지정하거나 요청에 서명해야 합니다. HTML 페이지 뒤의 스크립트나 SDK 중 하나를 사용하는 클라이언트 애플리케이션에서 이러한 작업을 처리할 수 있습니다. AWS

curl을 사용하여 API 호출

터미널에서 curl과 같은 도구를 사용하여 API를 호출할 수 있습니다. 다음 예제 curl 명령은 API prod 스테이지의 getUsers 리소스에서 GET 메서드를 호출합니다.

Linux or Macintosh

```
curl -X GET 'https://b123abcde4.execute-api.us-west-2.amazonaws.com/prod/getUsers'
```

Windows

```
curl -X GET "https://b123abcde4.execute-api.us-west-2.amazonaws.com/prod/getUsers"
```

API Gateway 콘솔을 사용하여 REST API 메서드 테스트

API Gateway 콘솔을 사용하여 REST API 메서드를 테스트합니다.

주제

- 사전 조건
- API Gateway 콘솔을 사용하여 메서드 테스트

사전 조건

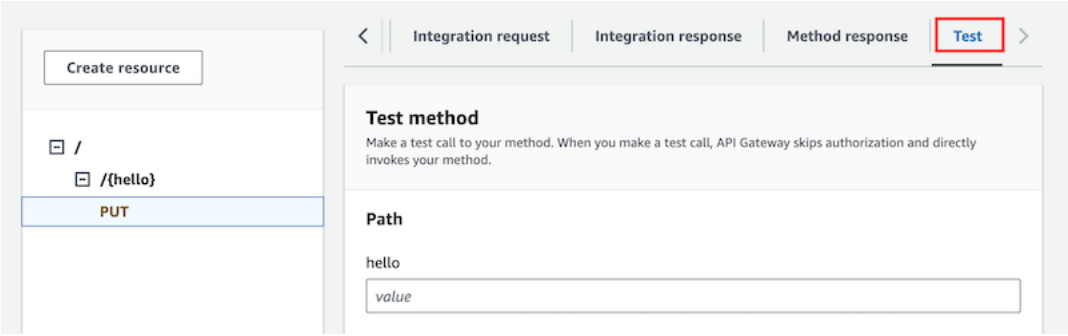
- 테스트하려는 메서드의 설정을 지정해야 합니다. API Gateway에서 REST API 메서드 설정의 지침을 따르세요.

API Gateway 콘솔을 사용하여 메서드 테스트

> **Important**
>
> API Gateway 콘솔을 사용하여 메서드를 테스트하면 리소스가 변경될 수 있으며 이 변경은 취소할 수 없습니다. API Gateway 콘솔을 사용하여 메서드를 테스트하는 것은 API Gateway 콘솔 외부에서 메서드를 호출하는 것과 같습니다. 예를 들어 API Gateway 콘솔을 사용하여 API 리소스를 삭제하는 메서드를 호출할 경우 메서드 호출이 성공하면 API 리소스가 삭제됩니다.

메서드를 테스트하려면

1. https://console.aws.amazon.com/apigateway에서 API Gateway 콘솔에 로그인합니다.
2. REST API를 선택합니다.
3. 리소스 창에서 테스트하려는 메서드를 선택합니다.
4. 테스트(Test) 탭을 선택합니다. 탭을 표시하려면 오른쪽 화살표 버튼을 선택해야 할 수도 있습니다.



표시된 임의의 상자(예: 쿼리 문자열, 헤더 및 요청 본문)에 값을 입력합니다. 콘솔에는 기본 애플리케이션/json 양식으로 메서드 요청에 이러한 값이 포함됩니다.

지정할 필요가 있는 추가 옵션에 대해서는 API 소유자에게 문의하세요.

5. 테스트를 선택합니다. 다음 정보가 표시됩니다.

   - 요청은 메서드에 대해 호출된 리소스의 경로입니다.
   - 상태는 응답의 HTTP 상태 코드입니다.
   - 지연 시간은 호출자로부터 요청을 수신한 시간과 응답 반환 시간의 시간차입니다.
   - 응답 본문은 HTTP 응답 본문입니다.
   - 응답 헤더는 HTTP 응답 헤더입니다.

   > **Tip**
   >
   > 매핑에 따라 HTTP 상태 코드, 응답 본문 및 응답 헤더가 Lambda 함수, HTTP 프록시 또는 AWS 서비스 프록시에서 전송한 것과 다를 수 있습니다.

   - 로그는 이 메서드가 API Gateway 콘솔 밖에서 호출되었을 경우 작성되었을 시뮬레이션된 Amazon CloudWatch Logs 입력입니다.

   > **Note**
   >
   > CloudWatch Logs 입력은 시뮬레이션된 것이지만 메서드 호출의 결과는 실제입니다.

API Gateway 콘솔을 사용하는 것 이외에 AWS CLI 또는 API Gateway용 AWS SDK를 사용해 메서드 호출을 테스트할 수 있습니다. AWS CLI를 사용해 테스트하려면 test-invoke-method 단원을 참조하십시오.

생성된 SDK를 통해 REST API 호출

이 섹션에서는 자바, 안드로이드용 자바, 루비, 오브젝티브-C, Swift로 작성된 클라이언트 앱에서 생성된 SDK를 통해 API를 호출하는 방법을 보여줍니다. JavaScript

주제

- API Gateway에서 생성한 REST API용 Java SDK 사용
- API Gateway에서 생성한 REST API용 Android SDK 사용
- API Gateway에서 생성한 REST API용 JavaScript SDK 사용
- API Gateway에서 생성한 REST API용 Ruby SDK 사용
- Objective-C 또는 Swift에서 API Gateway에 의해 생성되는 REST API용 iOS SDK 사용

API Gateway에서 생성한 REST API용 Java SDK 사용

이 단원에서는 API Gateway에서 생성된 REST API용 Java SDK를 사용하는 단계를 간편 계산기 API를 예로 들어 설명합니다. 계속하려면 먼저 API Gateway 콘솔을 사용하여 API용 SDK 생성의 단계를 완료해야 합니다.

API Gateway가 생성한 Java SDK를 설치하고 사용하려면

1. 앞서 다운로드한 API Gateway가 생성한 .zip 파일의 압축을 풉니다.
2. Apache Maven(버전 3.5 이상)을 다운로드하여 설치합니다.
3. JDK 8을 다운로드하여 설치합니다.
4. `JAVA_HOME` 환경 변수를 설정합니다.
5. pom.xml 파일이 있는, 압축을 해제한 SDK 폴더로 이동합니다. 기본적으로 이 폴더는 `generated-code`입니다. mvn install 명령을 실행해 컴파일된 아티팩트 파일을 로컬 Maven 리포지토리에 설치합니다. 그러면 컴파일된 SDK 라이브러리를 포함하는 `target` 폴더가 생성됩니다.
6. 빈 디렉터리에 다음 명령을 입력해 클라이언트 프로젝트 스텁을 생성하여 설치된 SDK 라이브러리를 사용해 API를 호출합니다.

```
mvn -B archetype:generate \
    -DarchetypeGroupdId=org.apache.maven.archetypes \
    -DgroupId=examples.aws.apig.simpleCalc.sdk.app \
    -DartifactId=SimpleCalc-sdkClient
```

> **Note**
>
> 앞의 명령에서 가독성을 높이기 위해 \ 구분자가 포함됩니다. 전체 명령은 구분자 없이 한 줄로 표시되어야 합니다.

이 명령은 애플리케이션 스텁을 생성합니다. 이 애플리케이션 스텁은 프로젝트의 루트 디렉터리에 pom.xml 파일과 src 폴더를 포함합니다(앞의 명령에서 *SimpleCalc-sdkClient*). 처음에는 두 개의 소스 파일, src/main/java/*{package-path}*/App.java와 src/test/java/*{package-path}*/AppTest.java가 있습니다. 이 예제에서 *{package-path}*는 examples/aws/apig/simpleCalc/sdk/app입니다. 이 패키지 경로는 DarchetypeGroupdId 값에서 파생됩니다. 클라이언트 애플리케이션에 App.java 파일을 템플릿으로 사용하고, 필요에 따라 같은 폴더에 다른 템플릿을 추가할 수 있습니다. 애플리케이션에 AppTest.java 파일을 단위 테스트 템플릿으로 사용하고, 필요에 따라 같은 테스트 폴더에 다른 테스트 코드 파일을 추가할 수 있습니다.

7. 생성된 pom.xml 파일의 패키지 종속성을 다음과 같이 업데이트하여, 프로젝트의 groupId, artifactId, version, name 속성을 대체합니다.

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/
POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>examples.aws.apig.simpleCalc.sdk.app</groupId>
  <artifactId>SimpleCalc-sdkClient</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>SimpleCalc-sdkClient</name>
  <url>http://maven.apache.org</url>

   <dependencies>
      <dependency>
          <groupId>com.amazonaws</groupId>
          <artifactId>aws-java-sdk-core</artifactId>
          <version>1.11.94</version>
      </dependency>
      <dependency>
          <groupId>my-apig-api-examples</groupId>
          <artifactId>simple-calc-sdk</artifactId>
          <version>1.0.0</version>
```

```
      </dependency>

    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.12</version>
      <scope>test</scope>
    </dependency>

    <dependency>
        <groupId>commons-io</groupId>
        <artifactId>commons-io</artifactId>
        <version>2.5</version>
    </dependency>
  </dependencies>

  <build>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>3.5.1</version>
        <configuration>
          <source>1.8</source>
          <target>1.8</target>
        </configuration>
      </plugin>
    </plugins>
  </build>
</project>
```

> **Note**
>
> `aws-java-sdk-core`의 종속 아티팩트 새 버전이 위에서 지정한 버전(`1.11.94`)과 호환되지 않는 경우, `<version>` 태그를 새 버전으로 업데이트해야 합니다.

8. 다음으로, SDK의 `getABOp(GetABOpRequest req)`, `getApiRoot(GetApiRootRequest req)`, `postApiRoot(PostApiRootRequest req)` 메서드를 호출하여, SDK를 사용해 API를 호출하는 방법을 살펴봅니다. 이 메서드는 각각 `GET /{a}/{b}/{op}` API 요청의 페이로드가 포함된 `GET /?a={x}&b={y}&op={operator}`, `POST /`, `{"a": x, "b": y, "op": "operator"}` 메서드에 해당합니다.

다음과 같이 App.java 파일을 업데이트합니다.

```
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());


   }
```

```
}
```

앞의 예에서 SDK 클라이언트를 인스턴스화하는 데 사용된 구성 설정은 설명용이며, 권장되는 모범 사례라고 볼 수는 없습니다. `sdkClient.shutdown()` 호출도 마찬가지로, 리소스 확보 시점에 대한 정밀한 제어가 필요한 경우 등에 선택하는 옵션입니다.

Java SDK를 사용하여 API를 호출하는 기본 패턴은 이미 살펴보았습니다. 다른 API 메서드 호출에도 이 지침을 적용할 수 있습니다.

API Gateway에서 생성한 REST API용 Android SDK 사용

이 단원에서는 API Gateway에서 생성된 REST API용 Android SDK의 사용 단계를 살펴봅니다. API Gateway 콘솔을 사용하여 API용 SDK 생성의 단계를 이미 완료한 경우에만 계속 진행할 수 있습니다.

> **Note**
>
> 생성된 SDK는 Android 4.4 이하 버전에서는 호환되지 않습니다. 자세한 내용은 the section called “중요 정보” 단원을 참조하세요.

API Gateway가 생성한 Android SDK 설치 및 사용 방법

1. 앞서 다운로드한 API Gateway가 생성한 .zip 파일의 압축을 풉니다.
2. Apache Maven(버전 3.x 권장)을 다운로드하여 설치합니다.
3. JDK 8을 다운로드하여 설치합니다.
4. `JAVA_HOME` 환경 변수를 설정합니다.
5. mvn install 명령을 실행해 컴파일된 아티팩트 파일을 로컬 Maven 리포지토리에 설치합니다. 그러면 컴파일된 SDK 라이브러리를 포함하는 `target` 폴더가 생성됩니다.
6. `target` 폴더에서 SDK 파일(이름은 SDK를 생성할 때 지정한 아티팩트 ID(Artifact Id) 및 아티팩트 버전(Artifact Version)에서 파생됨(예: `simple-calcsdk-1.0.0.jar`))을 `target/lib` 폴더의 다른 모든 라이브러리와 함께 프로젝트의 `lib` 폴더로 복사합니다.

   Android Studio를 사용하는 경우, 클라이언트 앱 모듈에 `libs` 폴더를 생성하고 필수 .jar 파일을 복사하여 이 폴더에 붙여 넣습니다. 모듈의 gradle 파일에 있는 종속성 섹션에 다음이 포함되어 있는지 확인합니다.

```
    compile fileTree(include: ['*.jar'], dir: 'libs')
```

```
    compile fileTree(include: ['*.jar'], dir: 'app/libs')
```

복제된 .jar 파일이 선언되지 않도록 합니다.

7. `ApiClientFactory` 클래스를 사용하여 API Gateway에서 생성한 SDK를 초기화합니다. 예:

```
ApiClientFactory factory = new ApiClientFactory();

// Create an instance of your SDK. Here, 'SimpleCalcClient.java' is the compiled
 java class for the SDK generated by API Gateway.
final SimpleCalcClient client = factory.build(SimpleCalcClient.class);

// Invoke a method:
//   For the 'GET /?a=1&b=2&op=+' method exposed by the API, you can invoke it by
 calling the following SDK method:

Result output = client.rootGet("1", "2", "+");

//     where the Result class of the SDK corresponds to the Result model of the
 API.
//

//   For the 'GET /{a}/{b}/{op}'  method exposed by the API, you can call the
 following SDK method to invoke the request,

Result output = client.aBOpGet(a, b, c);

//     where a, b, c can be "1", "2", "add", respectively.

//   For the following API method:
//        POST /
//        host: ...
//        Content-Type: application/json
//
//        { "a": 1, "b": 2, "op": "+" }
// you can call invoke it by calling the rootPost method of the SDK as follows:
Input body = new Input();
input.a=1;
input.b=2;
input.op="+";
Result output = client.rootPost(body);

//      where the Input class of the SDK corresponds to the Input model of the API.
```

```
// Parse the result:
//     If the 'Result' object is { "a": 1, "b": 2, "op": "add", "c":3"}, you
 retrieve the result 'c') as

String result=output.c;
```

8. Amazon Cognito 자격 증명 제공자를 이용하여 API에 대한 호출에 권한을 부여하려면, 다음 예제와 같이 `ApiClientFactory` 클래스를 사용하여 API Gateway에 의해 생성된 SDK를 통해 일단의 AWS 자격 증명을 전달합니다.

```
// Use CognitoCachingCredentialsProvider to provide AWS credentials
// for the ApiClientFactory
AWSCredentialsProvider credentialsProvider = new CognitoCachingCredentialsProvider(
        context,          // activity context
        "identityPoolId", // Cognito identity pool id
        Regions.US_EAST_1 // region of Cognito identity pool
);

ApiClientFactory factory = new ApiClientFactory()
  .credentialsProvider(credentialsProvider);
```

9. API Gateway에 의해 생성된 SDK를 사용해 API 키를 설정하려면 다음과 비슷한 코드를 사용합니다.

```
ApiClientFactory factory = new ApiClientFactory()
  .apiKey("YOUR_API_KEY");
```

API Gateway에서 생성한 REST API용 JavaScript SDK 사용

> **Note**
>
> 이러한 지침에서는 API Gateway 콘솔을 사용하여 API용 SDK 생성의 지침을 이미 완료한 것으로 가정합니다.

> **Important**
>
> API에 ANY 메서드만 정의된 경우 생성된 SDK 패키지에는 `apigClient.js` 파일이 포함되지 않으므로 ANY 메서드를 직접 정의할 필요가 없습니다.

API Gateway가 생성한 REST API용 JavaScript SDK를 설치, 시작, 호출하려면

1. 앞서 다운로드한 API Gateway가 생성한 .zip 파일의 압축을 풉니다.
2. API Gateway에서 생성한 SDK에 의해 생성된 모든 메서드에 대한 CORS(cross-origin 리소스 공유)를 활성화합니다. 지침은 REST API 리소스에 대한 CORS 활성화 단원을 참조하세요.
3. 웹 페이지에서 다음 스크립트에 참조를 포함시킵니다.

```
<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. 코드에서 다음과 비슷한 코드를 사용하여 API Gateway에 의해 생성된 SDK를 초기화합니다.

```
var apigClient = apigClientFactory.newClient();
```

AWS 자격 증명과 함께 API Gateway에 의해 생성된 SDK를 초기화하려면 다음과 비슷한 코드를 사용합니다. AWS 자격 증명을 사용하는 경우 API에 대한 모든 요청이 서명되어야 합니다.

```
var apigClient = apigClientFactory.newClient({
  accessKey: 'ACCESS_KEY',
  secretKey: 'SECRET_KEY',
});
```

API 키를 API Gateway에 의해 생성된 SDK와 함께 사용하려면 다음과 비슷한 코드를 사용하여 API 키를 `Factory` 객체에 파라미터로 전달합니다. API 키를 사용하는 경우 키는 `x-api-key` 헤더의 일부로 지정되며 API에 대한 모든 요청이 서명됩니다. 이는 각 요청에 대해 적절한 CORS 수락 헤더를 설정해야 한다는 의미입니다.

```
var apigClient = apigClientFactory.newClient({
  apiKey: 'API_KEY'
});
```

5. 다음과 비슷한 코드를 사용하여 API 메서드를 API Gateway에 호출합니다. 각 호출은 성공 및 실패 콜백과 함께 프라미스를 반환합니다.

```
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.
    });
```

여기에서는 *methodName*이 메서드 요청의 리소스 경로 및 HTTP 동사로부터 구성됩니다. SimpleCalc API의 경우, 해당하는 SDK 메서드의 API 메서드에 대한

```
1. GET /?a=...&b=...&op=...
2. POST /

   { "a": ..., "b": ..., "op": ...}
3. GET /{a}/{b}/{op}
```

SDK 메서드는 다음과 같습니다.

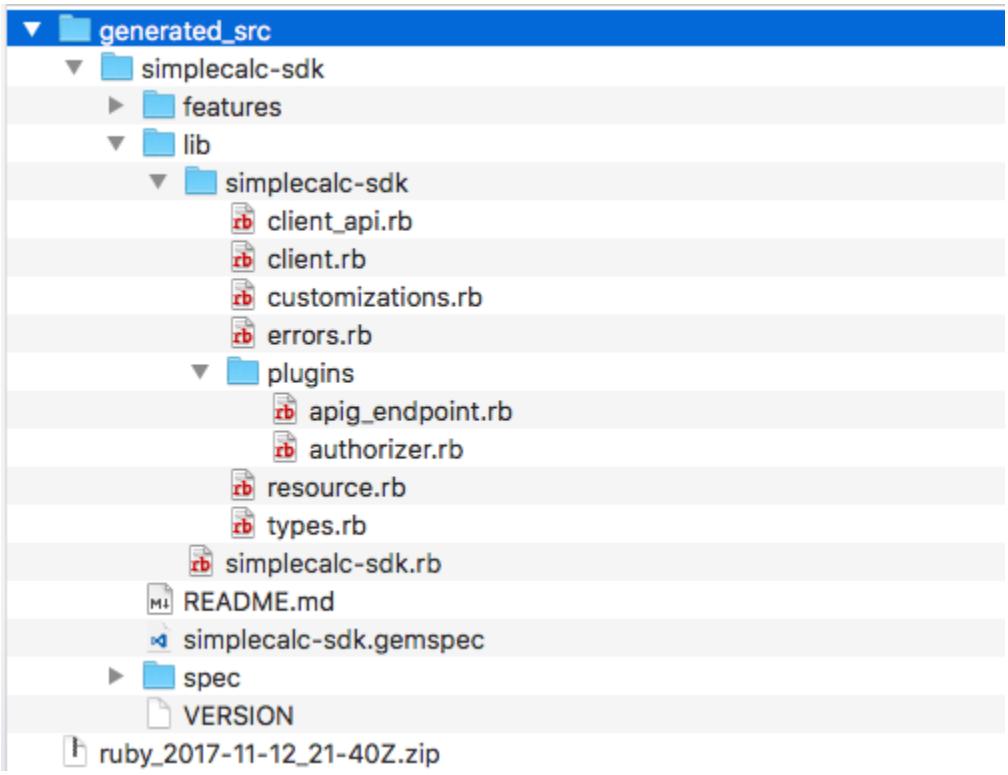
```
1. rootGet(params);      // where params={"a": ..., "b": ..., "op": ...} is
 resolved to the query parameters
2. rootPost(null, body); // where body={"a": ..., "b": ..., "op": ...}
3. aBOpGet(params);      // where params={"a": ..., "b": ..., "op": ...} is
 resolved to the path parameters
```

API Gateway에서 생성한 REST API용 Ruby SDK 사용

> **Note**
>
> 이 지침에서는 API Gateway 콘솔을 사용하여 API용 SDK 생성의 지침을 이미 완료한 것으로 가정합니다.

API Gateway가 생성한 REST API용 Ruby SDK를 설치, 인스턴스화, 호출하려면

1. 다운로드한 Ruby SDK 파일 압축을 풉니다. 생성된 SDK 원본은 다음과 같이 표시됩니다.



2. 터미널 창에서 다음 셸 명령을 사용하여 생성된 SDK 원본에서 Ruby Gem을 구축합니다.

```
# change to /simplecalc-sdk directory
cd simplecalc-sdk

# build the generated gem
gem build simplecalc-sdk.gemspec
```

   이 작업 후 simplecalc-sdk-1.0.0.gem을 사용할 수 있게 됩니다.

3. Gem 설치:

```
gem install simplecalc-sdk-1.0.0.gem
```

4. 클라이언트 애플리케이션을 생성합니다. 앱에서 Ruby SDK 클라이언트를 인스턴스화하고 초기화합니다.

```
require 'simplecalc-sdk'
client = SimpleCalc::Client.new(
```

```
    http_wire_trace: true,
    retry_limit: 5,
    http_read_timeout: 50
)
```

API에 AWS_IAM 유형의 권한 부여가 구성되어 있는 경우, 초기화 도중 accessKey 및 secretKey를 제공하여 호출자의 AWS 자격 증명을 포함할 수 있습니다.

```
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. 앱에서 SDK를 통해 API를 호출합니다.

> **Tip**
>
> SDK 메서드 호출 규칙에 익숙하지 않다면 생성된 SDK client.rb 폴더에 있는 lib 파일을 검토할 수 있습니다. 이 폴더에는 지원되는 각각의 API 메서드 호출 설명서가 포함되어 있습니다.

지원되는 연산을 검색하려면

```
# to show supported operations:
puts client.operation_names
```

이로 인해 각각 GET /?a={.}&b={.}&op={.}, GET /{a}/{b}/{op} 및 POST / API 메서드와 {a:"…", b:"…", op:"…"} 형식의 페이로드에 해당하는 다음의 표시가 나타납니다.

```
[:get_api_root, :get_ab_op, :post_api_root]
```

GET /?a=1&b=2&op=+ API 메서드를 호출하려면 다음 Ruby SDK 메서드를 호출하세요.

```
resp = client.get_api_root({a:"1", b:"2", op:"+"})
```

`POST /` 페이로드가 포함된 `{a: "1", b: "2", "op": "+"}` API 메서드를 호출하려면 다음 Ruby SDK 메서드를 호출하세요.

```
resp = client.post_api_root(input: {a:"1", b:"2", op:"+"})
```

`GET /1/2/+` API 메서드를 호출하려면 다음 Ruby SDK 메서드를 호출하세요.

```
resp = client.get_ab_op({a:"1", b:"2", op:"+"})
```

SDK 메서드 호출이 성공하면 다음과 같은 응답을 반환합니다.

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

Objective-C 또는 Swift에서 API Gateway에 의해 생성되는 REST API용 iOS SDK 사용

이 자습서에서는 Objective-C 또는 Swift 앱에서 API Gateway에 의해 생성되는 REST API용 iOS SDK를 사용하여 기본 API를 호출하는 방법을 보여줍니다. SimpleCalc API를 예제로 사용하여 다음 주제를 설명합니다.

- 필요한 AWS Mobile SDK 구성 요소를 Xcode 프로젝트에 설치하는 방법
- API 메서드를 호출하기 전에 API 클라이언트 객체를 생성하는 방법
- API 클라이언트 객체에서 해당 SDK 메서드를 통해 API 메서드를 호출하는 방법
- 메서드 입력을 준비하고 SDK의 해당 모델 클래스를 사용하여 결과를 구문 분석하는 방법

주제

- 생성된 iOS SDK(Objective-C)를 사용하여 API 호출

- API를 호출하기 위해 생성된 iOS SDK(Swift) 사용

생성된 iOS SDK(Objective-C)를 사용하여 API 호출

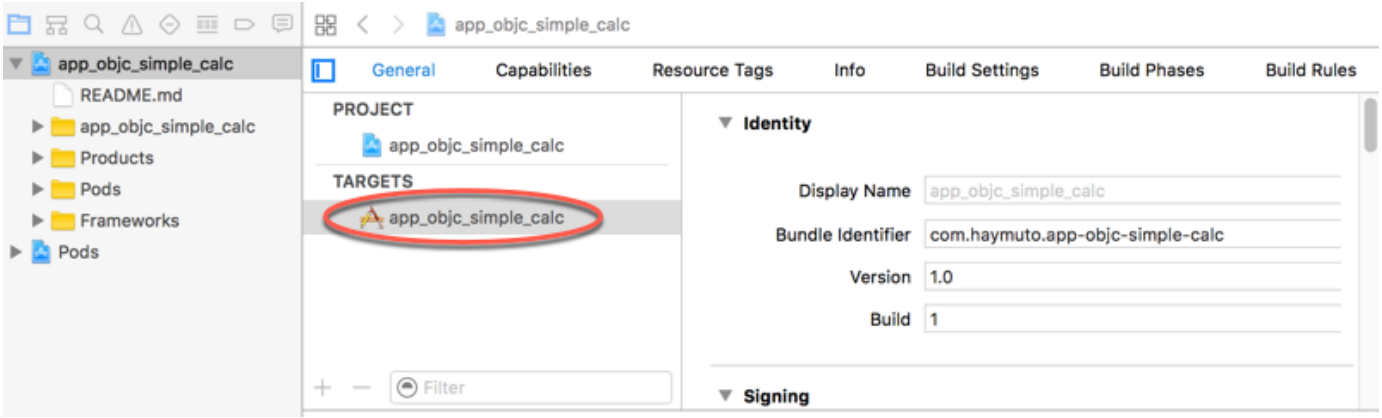
다음 절차를 시작하기 전에 Objective-C에서 iOS에 대한 API Gateway 콘솔을 사용하여 API용 SDK 생성의 단계를 완료하고 생성된 SDK에 대한 .zip 파일을 다운로드해야 합니다.

Objective-C 프로젝트에서 API Gateway에 의해 생성되는 iOS SDK 및 AWS Mobile SDK 설치

다음 절차에서는 SDK를 설치하는 방법을 설명합니다.

Objective-C에서 API Gateway가 생성한 iOS SDK를 설치하고 사용하려면

1. 앞서 다운로드한 API Gateway가 생성한 .zip 파일의 압축을 풉니다. SimpleCalc API를 사용하여 압축 해제된 SDK 폴더의 이름을 **`sdk_objc_simple_calc`** 등으로 바꿉니다. 이 SDK 폴더에는 `README.md` 파일과 `Podfile` 파일이 있습니다. `README.md` 파일에는 SDK를 설치하고 사용하기 위한 지침이 포함되어 있습니다. 이 자습서는 이러한 지침에 대한 세부 정보를 제공합니다. 설치 시 CocoaPods를 활용해 필요한 API Gateway 라이브러리와 기타 종속 AWS Mobile SDK 구성 요소를 가져옵니다. `Podfile`을 업데이트하여 앱의 Xcode 프로젝트로 SDK를 가져와야 합니다. 보관되지 않은 SDK 폴더에는 생성된 API SDK의 소스 코드가 포함된 `generated-src` 폴더도 있습니다.
2. Xcode를 시작하고 새로운 iOS Objective-C 프로젝트를 생성합니다. 프로젝트의 대상을 기록해 둡니다. `Podfile`에 이를 설정해야 합니다.



3. CocoaPods를 이용해 AWS Mobile SDK for iOS를 Xcode 프로젝트로 가져오려면 다음과 같이 하세요.

   a. 터미널 창에 다음 명령을 실행하여 CocoaPods를 설치합니다.

```
sudo gem install cocoapods
pod setup
```

b. 압축을 푼 SDK 폴더에서 `Podfile` 파일을 복사하여 Xcode 프로젝트 파일을 포함하는 동일 디렉터리에 추가합니다. 다음 블록:

```
target '<YourXcodeTarget>' do
    pod 'AWSAPIGateway', '~> 2.4.7'
end
```

을 프로젝트 대상 이름: 으로 바꿉니다.

```
target 'app_objc_simple_calc' do
    pod 'AWSAPIGateway', '~> 2.4.7'
end
```

Xcode 프로젝트에 이미 `Podfile`이라는 파일이 있는 경우, 다음 코드 줄을 추가합니다.

```
pod 'AWSAPIGateway', '~> 2.4.7'
```

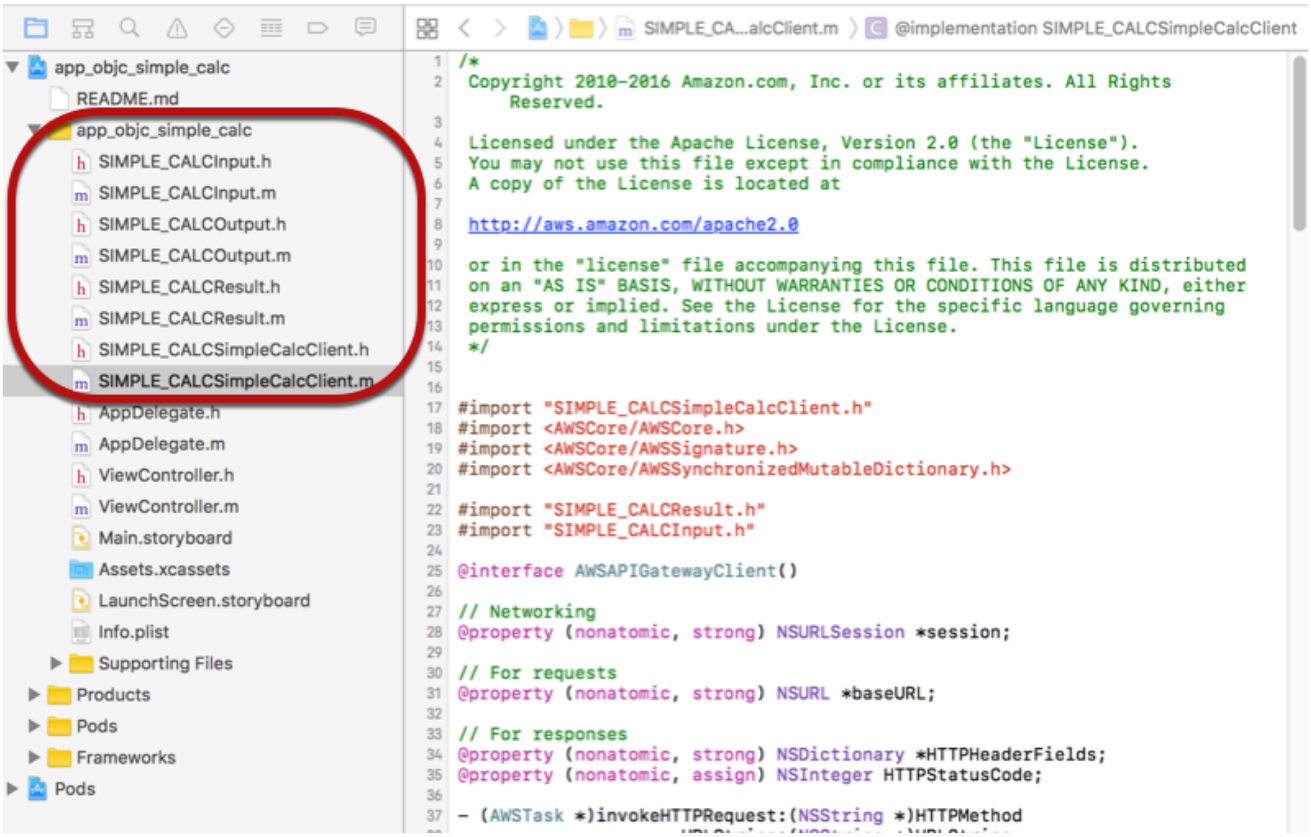
c. 터미널 창을 열고 다음 명령을 실행합니다.

```
pod install
```

그러면 API Gateway 구성 요소와 기타 종속 AWS Mobile SDK 구성 요소가 설치됩니다.

d. Xcode 프로젝트를 닫은 후 `.xcworkspace` 파일을 열어 Xcode를 다시 시작합니다.

e. 압축을 푼 SDK의 `.h` 디렉터리에 포함된 모든 `.m` 및 `generated-src` 파일을 Xcode 프로젝트에 추가합니다.



AWS Mobile SDK를 명시적으로 다운로드하거나 Carthage를 이용해 프로젝트로 AWS Mobile SDK for iOS Objective-C를 가져오려면 README.md 파일의 지침을 따르세요. 이러한 옵션 중 하나만 사용하여 AWS Mobile SDK를 가져와야 합니다.

Objective-C 프로젝트에서 API Gateway에 의해 생성되는 iOS SDK를 사용하여 API 메서드 호출

이 SimpleCalc API에 대한 `SIMPLE_CALC` 접두사와 메서드의 입력(`Input`) 및 출력(`Result`)에 대한 두 모델을 사용하여 SDK를 생성한 경우 SDK에서 결과 API 클라이언트 클래스는 `SIMPLE_CALCSimpleCalcClient`이고 해당 데이터 클래스는 각각 `SIMPLE_CALCInput` 및 `SIMPLE_CALCResult`입니다. API 요청 및 응답은 SDK 메서드에 다음과 같이 매핑됩니다.

- 다음의 API 요청이

```
GET /?a=...&b=...&op=...
```

다음의 SDK 메서드가 됩니다.

```
(AWSTask *)rootGet:(NSString *)op a:(NSString *)a b:(NSString *)b
```

`AWSTask.result` 모델이 메서드 응답에 추가된 경우 `SIMPLE_CALCResult` 속성은 `Result` 유형입니다. 그렇지 않은 경우 속성은 `NSDictionary` 유형입니다.

- 다음의 이 API 요청이

```
POST /

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

다음의 SDK 메서드가 됩니다.

```
(AWSTask *)rootPost:(SIMPLE_CALCInput *)body
```

- 다음의 API 요청이

```
GET /{a}/{b}/{op}
```

다음의 SDK 메서드가 됩니다.

```
(AWSTask *)aBOpGet:(NSString *)a b:(NSString *)b op:(NSString *)op
```

다음 절차에서는 Objective-C 앱 소스 코드에서 API 메서드를 호출하는 방법(예: `viewDidLoad` 파일에서 `ViewController.m` 위임자의 일부로 호출)을 설명합니다.

API Gateway가 생성한 iOS SDK를 통해 API를 호출하려면

1. API 클라이언트 클래스 헤더 파일을 가져와 앱에서 API 클라이언트 클래스를 호출할 수 있도록 합니다.

   ```
   #import "SIMPLE_CALCSimpleCalc.h"
   ```

   `#import` 명령문은 두 모델 클래스에 대한 `SIMPLE_CALCInput.h` 및 `SIMPLE_CALCResult.h`도 가져옵니다.

2. API 클라이언트 클래스 인스턴스화:

```
SIMPLE_CALCSimpleCalcClient *apiInstance = [SIMPLE_CALCSimpleCalcClient
 defaultClient];
```

API를 통해 Amazon Cognito를 사용하려면 다음과 같이 기본 `defaultServiceConfiguration` 객체에 `AWSServiceManager` 속성을 설정한 후 `defaultClient` 메서드를 호출해 API 클라이언트 객체를 생성합니다(앞의 예와 같음).

```
AWSCognitoCredentialsProvider *creds = [[AWSCognitoCredentialsProvider alloc]
 initWithRegionType:AWSRegionUSEast1 identityPoolId:your_cognito_pool_id];
AWSServiceConfiguration *configuration = [[AWSServiceConfiguration alloc]
 initWithRegion:AWSRegionUSEast1 credentialsProvider:creds];
AWSServiceManager.defaultServiceManager.defaultServiceConfiguration =
 configuration;
```

3. `GET /?a=1&b=2&op=+` 메서드를 호출하여 1+2를 수행합니다.

```
[[apiInstance rootGet: @"+" a:@"1" b:@"2"] continueWithBlock:^id _Nullable(AWSTask
 * _Nonnull task) {
    _textField1.text = [self handleApiResponse:task];
    return nil;
}];
```

여기에서 헬퍼 함수 `handleApiResponse:task`는 결과 서식을 텍스트 필드 (`_textField1`)에 표시할 문자열로 지정합니다.

```
- (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;
}
```

표시되는 결과는 `1 + 2 = 3`입니다.

4. 페이로드와 함께 `POST /` 메서드를 호출하여 1-2를 수행합니다.

```
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;
    }];
```

표시되는 결과는 `1 - 2 = -1`입니다.

5. `GET /{a}/{b}/{op}`를 호출하여 `1/2`를 수행합니다.

```
[[apiInstance aBOpGet:@"1" b:@"2" op:@"div"] continueWithBlock:^id
 _Nullable(AWSTask * _Nonnull task) {
    _textField3.text = [self handleApiResponse:task];
    return nil;
}];
```

표시되는 결과는 `1 div 2 = 0.5`입니다. 여기서 `div` 자리에 `/`가 사용되는데, 이유는 백엔드의 간단한 Lambda 함수가 URL로 인코딩된 경로 변수를 처리하지 않기 때문입니다.

API를 호출하기 위해 생성된 iOS SDK(Swift) 사용

다음 절차를 시작하기 전에 Swift에서 iOS에 대한 API Gateway 콘솔을 사용하여 API용 SDK 생성의 단계를 완료하고 생성된 SDK에 대한 .zip 파일을 다운로드해야 합니다.

주제

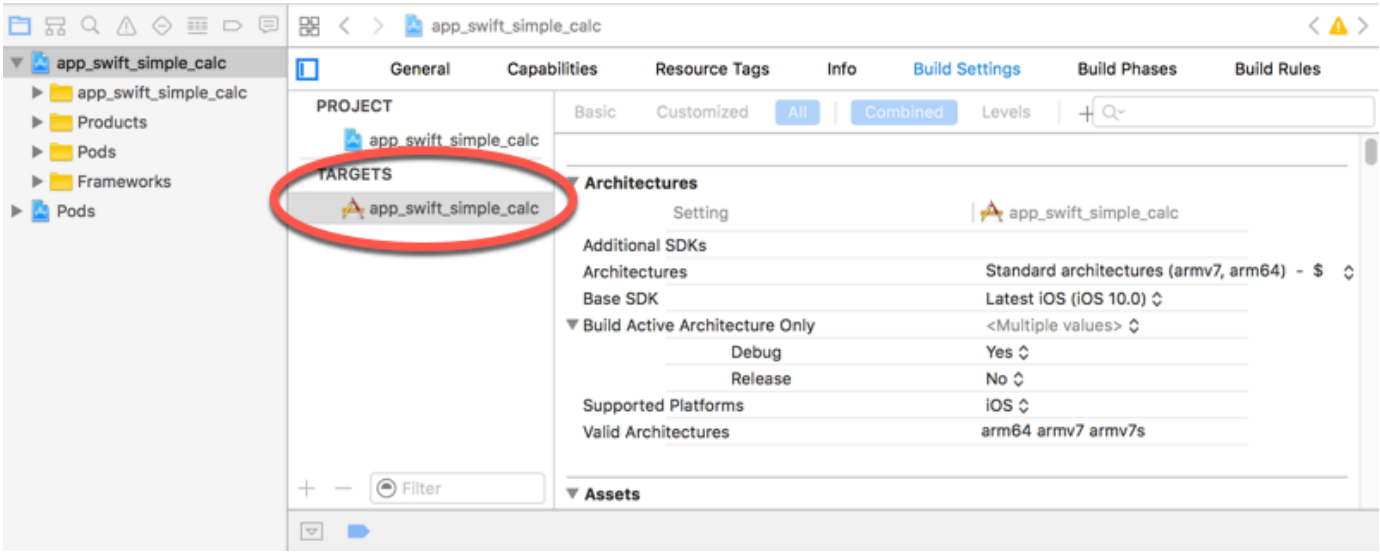
- AWS Mobile SDK 및 API Gateway 생성 SDK를 Swift 프로젝트에 설치
- Swift 프로젝트에서 API Gateway가 생성한 iOS SDK를 통해 API 메서드 호출

AWS Mobile SDK 및 API Gateway 생성 SDK를 Swift 프로젝트에 설치

다음 절차에서는 SDK를 설치하는 방법을 설명합니다.

Swift에서 API Gateway가 생성한 iOS SDK를 설치하고 사용하려면

1. 앞서 다운로드한 API Gateway가 생성한 .zip 파일의 압축을 풉니다. SimpleCalc API를 사용하여 압축 해제된 SDK 폴더의 이름을 **sdk_swift_simple_calc** 등으로 바꿉니다. 이 SDK 폴더에는 `README.md` 파일과 `Podfile` 파일이 있습니다. `README.md` 파일에는 SDK를 설치하고 사용하기 위한 지침이 포함되어 있습니다. 이 자습서는 이러한 지침에 대한 세부 정보를 제공합니다. 설치 시 CocoaPods를 활용해 필요한 AWS Mobile SDK 구성 요소를 가져옵니다. `Podfile`을 업데이트하여 Swift 앱의 Xcode 프로젝트로 SDK를 가져와야 합니다. 아카이빙을 해제한 SDK 폴더에는 사용자 API에서 생성한 SDK의 소스 코드를 포함하는 `generated-src` 폴더도 포함되어 있습니다.
2. Xcode를 시작하고 새로운 iOS Swift 프로젝트를 생성합니다. 프로젝트의 대상을 기록해 둡니다. `Podfile`에 이를 설정해야 합니다.



3. CocoaPods를 이용해 필요한 AWS Mobile SDK 구성 요소를 Xcode 프로젝트로 가져오려면 다음과 같이 합니다.

   a. CocoaPods가 아직 설치되어 있지 않은 경우 터미널 창에 다음 명령을 실행하여 설치합니다.

   ```
   sudo gem install cocoapods
   pod setup
   ```

   b. 압축을 푼 SDK 폴더에서 `Podfile` 파일을 복사하여 Xcode 프로젝트 파일을 포함하는 동일 디렉터리에 추가합니다. 다음 블록:

   ```
   target '<YourXcodeTarget>' do
       pod 'AWSAPIGateway', '~> 2.4.7'
   end
   ```

프로젝트 대상 이름은 다음과 같습니다.

```
target 'app_swift_simple_calc' do
    pod 'AWSAPIGateway', '~> 2.4.7'
end
```

Xcode 프로젝트에 이미 올바른 대상의 Podfile이 있는 경우, 다음 코드 줄을 do ... end 루프에 추가하기만 하면 됩니다.

```
pod 'AWSAPIGateway', '~> 2.4.7'
```

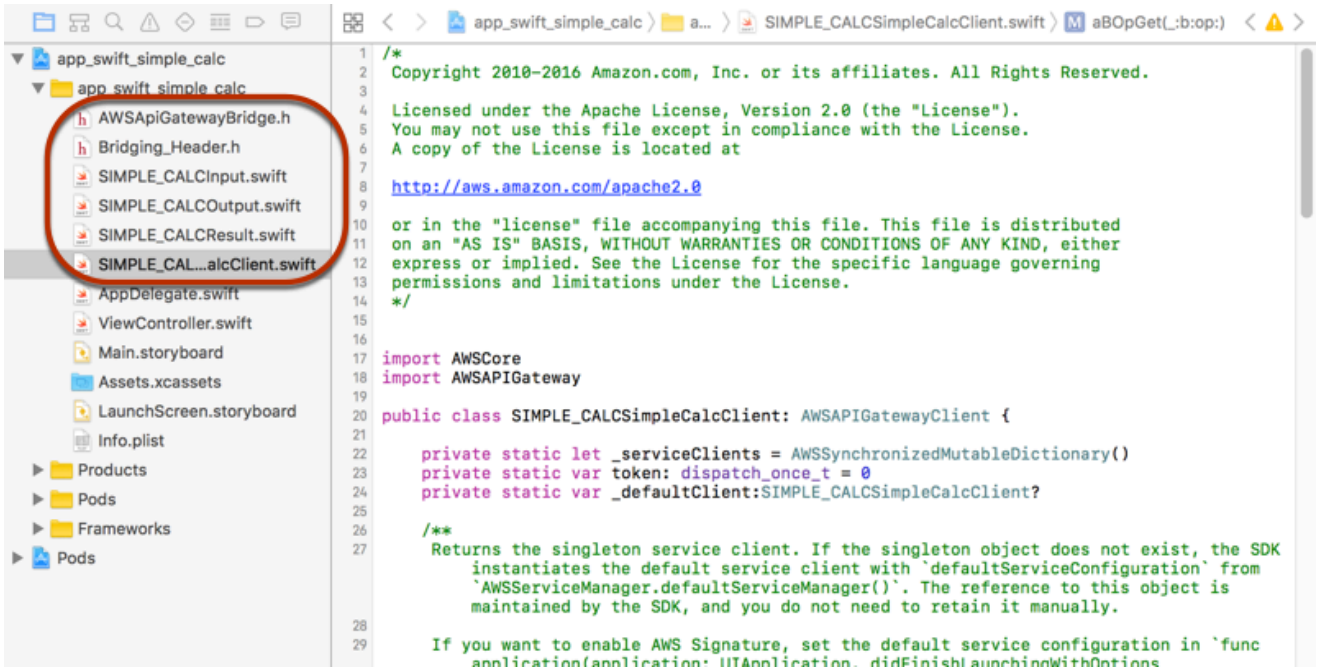
c. 터미널 창을 열고 앱 디렉터리에서 다음 명령을 실행합니다.

```
pod install
```

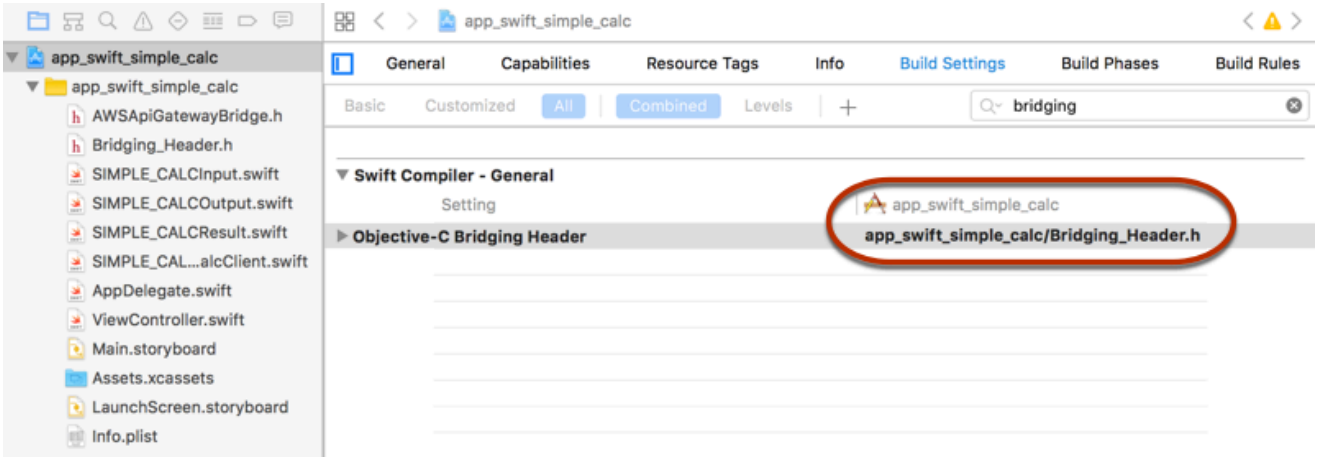
그러면 앱의 프로젝트에 API Gateway 구성 요소와 종속 AWS Mobile SDK 구성 요소가 설치됩니다.

d. Xcode 프로젝트를 닫은 후 *.xcworkspace 파일을 열어 Xcode를 다시 시작합니다.

e. 추출한 .h 디렉터리에서 SDK의 모든 헤더 파일(.swift) 및 Swift 소스 코드 파일(generated-src)을 Xcode 프로젝트에 추가합니다.

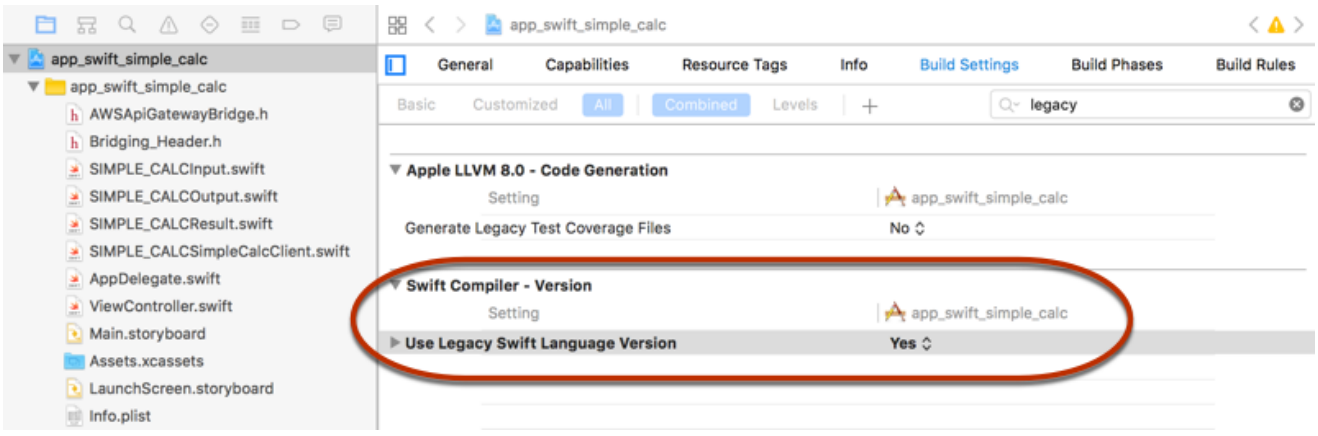


f. Swift 코드 프로젝트에서 AWS Mobile SDK의 Objective-C 라이브러리 호출을 활성화하려면 Xcode 프로젝트 구성의 Swift Compiler - General 설정에서 Objective-C Bridging Header 속성에 `Bridging_Header.h` 파일 경로를 설정합니다.



> **Tip**
>
> Xcode의 검색 상자에 **`bridging`**을 입력하여 Objective-C Bridging Header 속성을 찾을 수 있습니다.

g. Xcode 프로젝트를 구축하여 적절하게 구성되어 있는지 확인한 후 다음 단계로 진행합니다. Xcode가 AWS Mobile SDK에 지원되는 Swift보다 최신 버전을 사용하는 경우, Swift 컴파일러 오류를 받게 됩니다. 이 경우 Swift Compiler - Version 설정에서 Use Legacy Swift Language Version 속성을 예로 설정합니다.



AWS Mobile SDK를 명시적으로 다운로드하거나 Carthage를 이용해 Swift에서 AWS Mobile SDK for iOS를 프로젝트로 가져오려면 SDK 패키지와 함께 제공된 `README.md` 파일의 지침을 따릅니다. 이러한 옵션 중 하나만 사용하여 AWS Mobile SDK를 가져와야 합니다.

Swift 프로젝트에서 API Gateway가 생성한 iOS SDK를 통해 API 메서드 호출

API 요청 및 응답의 입력(`Input`) 및 출력(`Result`)을 설명하는 두 가지 모델을 사용하여 이 SimpleCalc API에 대한 `SIMPLE_CALC` 접두사를 사용하여 SDK를 생성한 경우 SDK에서 결과 API 클라이언트 클래스는 `SIMPLE_CALCSimpleCalcClient`가 되고 해당 데이터 클래스는 각각 `SIMPLE_CALCInput` 및 `SIMPLE_CALCResult`입니다. API 요청 및 응답은 SDK 메서드에 다음과 같이 매핑됩니다.

- 다음의 API 요청이

  ```
  GET /?a=...&b=...&op=...
  ```

  다음의 SDK 메서드가 됩니다.

  ```
  public func rootGet(op: String?, a: String?, b: String?) -> AWSTask
  ```

  `AWSTask.result` 모델이 메서드 응답에 추가된 경우 `SIMPLE_CALCResult` 속성은 `Result` 유형입니다. 그렇지 않으면 `NSDictionary` 유형입니다.

- 다음의 이 API 요청이

  ```
  POST /

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

  다음의 SDK 메서드가 됩니다.

  ```
  public func rootPost(body: SIMPLE_CALCInput) -> AWSTask
  ```

- 다음의 API 요청이

```
GET /{a}/{b}/{op}
```

다음의 SDK 메서드가 됩니다.

```
public func aBOpGet(a: String, b: String, op: String) -> AWSTask
```

다음 절차에서는 Swift 앱 소스 코드에서 API 메서드를 호출하는 방법(예: `viewDidLoad()` 파일에서 `ViewController.m` 위임자의 일부로 호출)을 설명합니다.

API Gateway가 생성한 iOS SDK를 통해 API를 호출하려면

1. API 클라이언트 클래스 인스턴스화:

   ```
   let client = SIMPLE_CALCSimpleCalcClient.default()
   ```

   API를 통해 Amazon Cognito를 사용하려면 (다음과 같이) 기본 AWS 서비스 구성을 설정한 후, (이전과 같이) `default` 메서드를 가져옵니다.

   ```
   let credentialsProvider =
    AWSCognitoCredentialsProvider(regionType: AWSRegionType.USEast1, identityPoolId:
    "my_pool_id")
   let configuration = AWSServiceConfiguration(region: AWSRegionType.USEast1,
    credentialsProvider: credentialsProvider)
   AWSServiceManager.defaultServiceManager().defaultServiceConfiguration =
    configuration
   ```

2. `GET /?a=1&b=2&op=+` 메서드를 호출하여 1+2를 수행합니다.

   ```
   client.rootGet("+", a: "1", b:"2").continueWithBlock {(task: AWSTask) -> AnyObject?
    in
       self.showResult(task)
       return nil
   }
   ```

   헬퍼 함수 `self.showResult(task)`가 콘솔에 결과나 오류를 인쇄합니다. 예:

   ```
   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)")
        }
    }
}
```

생산 앱에서 텍스트 필드에 결과나 오류를 표시할 수 있습니다. 표시되는 결과는 `1 + 2 = 3`입니다.

3. 페이로드와 함께 `POST /` 메서드를 호출하여 `1-2`를 수행합니다.

```
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
}
```

표시되는 결과는 `1 - 2 = -1`입니다.

4. `GET /{a}/{b}/{op}`를 호출하여 `1/2`를 수행합니다.

```
client.aBOpGet("1", b:"2", op:"div").continueWithBlock {(task: AWSTask) ->
 AnyObject? in
    self.showResult(task)
    return nil
}
```

표시되는 결과는 `1 div 2 = 0.5`입니다. 여기서 `div` 자리에 `/`가 사용되는데, 이유는 백엔드의 간단한 Lambda 함수가 URL로 인코딩된 경로 변수를 처리하지 않기 때문입니다.

프라이빗 API를 호출하는 방법

프라이빗 API는 VPC 내에서만 액세스할 수 있으며, 리소스 정책은 구성한 VPC 및 VPC 엔드포인트의 액세스를 허용해야 합니다. 프라이빗 API에 액세스하는 방식은 VPC 엔드포인트에 프라이빗 DNS를 활성화했는지 여부에 따라 다릅니다. 예를 들어 AWS Direct Connect를 통해 온프레미스 네트워크에서 프라이빗 API에 액세스하는 동안에는 VPC 엔드포인트에서 프라이빗 DNS가 활성화됩니다. 이 경우 엔드포인트 특정한 퍼블릭 DNS 호스트 이름을 사용하여 프라이빗 API 호출에 설명된 단계를 따르십시오.

프라이빗 API를 배포한 후, 프라이빗 DNS(프라이빗 DNS 이름 지정을 활성화한 경우) 및 퍼블릭 DNS를 통해 그 API에 액세스할 수 있습니다.

프라이빗 API의 DNS 이름을 얻으려면 다음과 같이 하세요.

1. AWS Management Console 로그인하고 https://console.aws.amazon.com/vpc/ 에서 Amazon VPC 콘솔을 엽니다.
2. 왼쪽의 탐색 창에서 엔드포인트를 선택한 후 API Gateway용 인터페이스 VPC 종단점을 선택합니다.
3. 세부 정보 창의 DNS 이름 필드에 값이 5개 표시됩니다. 처음 3개는 해당 API의 퍼블릭 DNS 이름입니다. 나머지 2개는 프라이빗 DNS 이름입니다.

프라이빗 DNS 이름으로 프라이빗 API 호출

> **Warning**
>
> API Gateway의 인터페이스 VPC 엔드포인트를 생성할 때 프라이빗 DNS 이름 활성화 옵션을 선택하면 VPC 엔드포인트가 있는 VPC는 퍼블릭(엣지 최적화 및 리전) API에 액세스할 수 없습니다. 자세한 내용은 API Gateway VPC 종단점에서 퍼블릭 API에 연결할 수 없는 이유는 무엇입니까?를 참조하세요.

프라이빗 DNS를 활성화했다면 다음과 같이 그 프라이빗 DNS 이름으로 프라이빗 API에 액세스할 수 있습니다.

```
{restapi-id}.execute-api.{region}.amazonaws.com
```

API를 호출하는 기본 URL의 형식은 다음과 같습니다.

```
https://{restapi-id}.execute-api.{region}.amazonaws.com/{stage}
```

예를 들어 이 예제에서 `GET /pets` 및 `GET /pets/{petId}` 메서드를 설정했고 REST API ID가 `01234567ab`이며 해당 리전은 `us-west-2`라고 가정하면, 브라우저에 다음 URL을 입력하여 API를 테스트할 수 있습니다.

```
https://01234567ab.execute-api.us-west-2.amazonaws.com/test/pets
```

및

```
https://01234567ab.execute-api.us-west-2.amazonaws.com/test/pets/1
```

또는 다음 cURL 명령을 사용할 수 있습니다.

```
curl -X GET https://01234567ab.execute-api.us-west-2.amazonaws.com/test/pets
```

그리고

```
curl -X GET https://01234567ab.execute-api.us-west-2.amazonaws.com/test/pets/2
```

를 사용하여 프라이빗 API에 액세스할 수 있습니다. AWS Direct Connect

또한 를 AWS Direct Connect 사용하여 온프레미스 네트워크에서 Amazon VPC로 전용 프라이빗 연결을 설정하고 퍼블릭 DNS 이름을 사용하여 해당 연결을 통해 프라이빗 API 엔드포인트에 액세스할 수 있습니다.

또한 Amazon Route 53 Resolver 인바운드 엔드포인트를 설정하고 원격 네트워크에서 프라이빗 DNS의 모든 DNS 쿼리를 전달함으로써 프라이빗 DNS 이름을 사용하여 온프레미스 네트워크에서 프라이빗 API에 액세스할 수 있습니다. 자세한 내용은 Amazon Route 53 개발자 안내서의 VPC로 인바운드 DNS 쿼리 전달을 참조하세요.

Route53 별칭을 사용하여 프라이빗 API 액세스

VPC 엔드포인트를 프라이빗 REST API와 연결 또는 연결 해제에 설명된 절차를 사용하면 VPC 엔드포인트를 프라이빗 API와 연결하거나 연결을 해제할 수 있습니다.

REST API를 호출할 VPC 엔드포인트와 프라이빗 API의 REST API ID를 연결하면, 다음 형식의 기본 URL을 사용하여 Route53 별칭을 통해 API를 호출할 수 있습니다.

생성된 기본 URL의 형식은 다음과 같습니다.

```
https://{rest-api-id}-{vpce-id}.execute-api.{region}.amazonaws.com/{stage}
```

예를 들어 이 예에서 GET /pets 및 GET /pets/*{petId}* 메소드를 설정했다고 가정하고 API의 API ID가 01234567ab였고 VPC 엔드포인트 ID는 vpce-01234567abcdef012였으며 리전은 us-west-2였다고 가정한다면, 다음과 같이 API를 호출할 수 있습니다.

```
curl -v https://01234567ab-vpce-01234567abcdef012.execute-api.us-west-2.amazonaws.com/
test/pets/
```

엔드포인트별 퍼블릭 DNS 호스트 이름을 사용하여 프라이빗 API 호출

엔드포인트 특정한 DNS 호스트 이름을 사용하여 프라이빗 API에 액세스할 수 있습니다. 이들 이름은 프라이빗 API에 대한 VPC 엔드포인트 ID 또는 API ID가 포함된 퍼블릭 DNS 호스트 이름입니다.

생성된 기본 URL의 형식은 다음과 같습니다.

```
https://{public-dns-hostname}.execute-api.{region}.vpce.amazonaws.com/{stage}
```

예를 들어, 이 예제에서 `GET /pets` 및 `GET /pets/{petId}` 메서드를 설정했고 API의 API ID는 `01234567ab`이고 퍼블릭 DNS 호스트 이름은 `vpce-01234567abcdef012-01234567`이며 리전은 `us-west-2`라고 가정하면 다음 예와 같이 cURL 명령으로 `Host` 헤더를 사용하여 VPCE ID를 통해 API를 테스트할 수 있습니다.

```
curl -v https://vpce-01234567abcdef012-01234567.execute-api.us-
west-2.vpce.amazonaws.com/test/pets -H 'Host: 01234567ab.execute-api.us-
west-2.amazonaws.com'
```

또는 다음 형식의 cURL 명령으로 `x-apigw-api-id` 헤더를 사용하여 API ID를 통해 프라이빗 API에 액세스할 수 있습니다.

```
curl -v https://{public-dns-hostname}.execute-api.{region}.vpce.amazonaws.com/test -
H'x-apigw-api-id:{api-id}'
```

OpenAPI를 사용하여 REST API 구성

API Gateway를 사용하여 외부 정의 파일에서 API Gateway로 REST API를 가져올 수 있습니다. 현재 API Gateway는 Amazon API Gateway의 REST API 중요 정보에 나열된 파일을 제외하고 OpenAPI

v2.0 및 OpenAPI v3.0 정의 파일을 지원합니다. 새 정의로 덮어쓰기를 해서API를 업데이트하거나, 정의를 기존 API에 병합할 수 있습니다. 요청 URL에서 mode 쿼리 파라미터를 사용하여 옵션을 지정합니다.

API Gateway 콘솔에서 API 가져오기 기능을 사용하려면 자습서: 예제를 가져와 REST API 생성을 참조하세요.

주제

- 엣지 최적화된 API를 API Gateway로 가져오기
- 리전 API를 API Gateway로 가져오기
- OpenAPI 파일을 가져와 기존 API 정의 업데이트
- OpenAPI basePath 속성 설정
- AWSOpenAPI 가져오기를 위한 변수
- 가져오는 중 발생하는 오류 및 경고
- API Gateway에서 REST API 내보내기

엣지 최적화된 API를 API Gateway로 가져오기

API OpenAPI 정의 파일을 가져와 EDGE 엔드포인트 유형을 OpenAPI 파일 외에 추가 입력으로 가져오기 작업에 지정함으로써 새로운 엣지 최적화 API를 생성할 수 있습니다. API Gateway 콘솔, AWS CLI 또는 AWS SDK를 사용하여 이를 수행할 수 있습니다.

API Gateway 콘솔에서 API 가져오기 기능을 사용하려면 자습서: 예제를 가져와 REST API 생성을 참조하세요.

주제

- API Gateway 콘솔을 사용하여 엣지 최적화 API 가져오기
- AWS CLI를 사용해 엣지 최적화 API 가져오기

API Gateway 콘솔을 사용하여 엣지 최적화 API 가져오기

API Gateway 콘솔을 사용해 엣지 최적화 API 가져오려면 다음을 따릅니다.

1. https://console.aws.amazon.com/apigateway에서 API Gateway 콘솔에 로그인합니다.
2. API 생성을 선택합니다.

3. REST API 아래에서 가져오기를 선택합니다.
4. API의 OpenAPI 정의를 복사하여 코드 편집기에 붙여넣거나 파일 선택을 선택하여 로컬 드라이브로부터 OpenAPI 파일을 로드합니다.
5. API 엔드포인트 유형에서 엣지 최적화를 선택합니다.
6. OpenAPI 정의를 가져오려면 API 생성을 선택합니다.

AWS CLI를 사용해 엣지 최적화 API 가져오기

OpenAPI 정의 파일에서 API를 가져와 AWS CLI를 사용해 새로운 엣지 최적화 API를 생성하려면 다음과 같이 `import-rest-api` 명령을 사용하십시오.

```
aws apigateway import-rest-api \
    --fail-on-warnings \
    --body 'file://path/to/API_OpenAPI_template.json'
```

또는 다음과 같이 `endpointConfigurationTypes`에 대한 `EDGE` 쿼리 문자열 파라미터의 명시적인 사양을 사용합니다.

```
aws apigateway import-rest-api \
    --parameters endpointConfigurationTypes=EDGE \
    --fail-on-warnings \
    --body 'file://path/to/API_OpenAPI_template.json'
```

리전 API를 API Gateway로 가져오기

API를 가져올 때 API에 대해 리전 엔드포인트 구성을 선택할 수 있습니다. API Gateway 콘솔, AWS CLI 또는 AWS SDK를 사용할 수 있습니다.

API를 내보낼 때 내보낸 API 정의에 API 엔드포인트 구성은 포함되지 않습니다.

API Gateway 콘솔에서 API 가져오기 기능을 사용하려면 자습서: 예제를 가져와 REST API 생성을 참조하세요.

주제

- API Gateway 콘솔을 사용하여 리전 API 가져오기
- AWS CLI를 사용하여 리전 API 가져오기

API Gateway 콘솔을 사용하여 리전 API 가져오기

API Gateway 콘솔을 사용하여 리전 엔드포인트의 API를 가져오려면 다음과 같이 합니다.

1. https://console.aws.amazon.com/apigateway에서 API Gateway 콘솔에 로그인합니다.
2. API 생성을 선택합니다.
3. REST API 아래에서 가져오기를 선택합니다.
4. API의 OpenAPI 정의를 복사하여 코드 편집기에 붙여넣거나 파일 선택을 선택하여 로컬 드라이브로부터 OpenAPI 파일을 로드합니다.
5. API 엔드포인트 유형에서 지역을 선택합니다.
6. OpenAPI 정의를 가져오려면 API 생성을 선택합니다.

AWS CLI를 사용하여 리전 API 가져오기

AWS CLI를 사용하여 OpenAPI 정의 파일에서 API를 가져오려면 다음과 같이 `import-rest-api` 명령을 사용하십시오.

```
aws apigateway import-rest-api \
    --parameters endpointConfigurationTypes=REGIONAL \
    --fail-on-warnings \
    --body 'file://path/to/API_OpenAPI_template.json'
```

OpenAPI 파일을 가져와 기존 API 정의 업데이트

단계와 단계 변수 또는 API 키에 대한 참조뿐 아니라 엔드포인트 구성을 변경하지 않은 상태에서 기존 API를 업데이트만할 목적으로 API 정의를 가져올 수 있습니다.

가져와 업데이트하는 작업은 병합 또는 덮어쓰기의 두 가지 모드에서 발생할 수 있습니다.

API(A)를 다른 API(B)로 병합할 때 두 API가 충돌하는 정의를 공유하지 않으면 그 결과로 얻는 API는 A와 B의 정의를 모두 유지합니다. 충돌이 발생하는 경우 병합하는 API(A)의 메서드 정의가 병합되는 API(B)의 해당하는 메서드 정의를 재정의합니다. 예를 들어 B가 다음 메서드를 선언하여 `200` 및 `206` 응답을 반환했다고 가정하겠습니다.

```
GET /a
POST /a
```

그리고 A는 다음 메서드를 선언하여 `200` 및 `400` 응답을 반환합니다.

```
GET /a
```

A를 B에 병합하면 그 결과로 얻는 API는 다음 메서드를 출력합니다.

```
GET /a
```

200 및 400 응답을 반환하고

```
POST /a
```

200 및 206 응답을 반환합니다.

외부 API 정의를 여러 개의 더 작은 파트로 분해하고 한 번에 한 파트의 변경사항만 적용할 경우 API 병합이 유용합니다. 예를 들어, 여러 팀이 API의 서로 다른 파트를 담당하고 있고 다른 비율로 변경사항을 사용할 수 있는 경우 병합이 발생할 수 있습니다. 이 모드에서는 가져온 정의에 특별히 정의되지 않은 기존 API의 항목이 그대로 유지됩니다.

API(A)가 다른 API(B)를 덮어쓰면 그 결과로 얻는 API는 덮어쓴 API(A)의 정의를 취합니다. API 덮어쓰기는 외부 API 정의에 API의 전체 정의가 포함된 경우 유용합니다. 이 모드에서는 가져온 정의에 특별히 정의되지 않은 기존 API의 항목이 삭제됩니다.

API를 병합하려면 PUT에 `https://apigateway.<region>.amazonaws.com/restapis/<restapi_id>?mode=merge` 요청을 제출하세요. `restapi_id` 경로 파라미터 값이 제공된 API 정의가 병합되는 API를 지정합니다.

다음 코드 조각에서는 JSON의 OpenAPI API 정의를 페이로드로 API Gateway의 이미 지정된 API와 병합하는 PUT 요청 예를 보여줍니다.

```
PUT /restapis/<restapi_id>?mode=merge
Host:apigateway.<region>.amazonaws.com
Content-Type: application/json
Content-Length: ...

An OpenAPI API definition in JSON
```

병합 업데이트 작업에서는 2개의 전체 API 정의를 가져와서 함께 병합합니다. 소규모 증분 변경사항의 경우 리소스 업데이트 작업을 사용할 수 있습니다.

API를 덮어쓰려면 PUT 요청을 `https://apigateway.<region>.amazonaws.com/restapis/<restapi_id>?mode=overwrite`에 제출하세요. `restapi_id` 경로 파라미터는 제공된 API 정의로 덮어쓰는 API를 지정합니다.

다음 코드 조각에서는 JSON 형식의 OpenAPI 정의의 페이로드가 포함된 덮어쓰기 요청의 예를 보여줍니다.

```
PUT /restapis/<restapi_id>?mode=overwrite
Host:apigateway.<region>.amazonaws.com
Content-Type: application/json
Content-Length: ...

An OpenAPI API definition in JSON
```

`mode` 쿼리 파라미터가 지정되지 않은 경우 병합이 사용됩니다.

> **Note**
>
> PUT 작업은 idempotent 방식이지만 원자성 작업은 아닙니다. 즉, 처리 과정에서 시스템 오류가 발생하면 API의 상태가 좋지 않을 수 있습니다. 그러나 작업을 성공적으로 반복하면 API가 첫 번째 작업이 성공한 것과 동일한 최종 상태가 됩니다.

OpenAPI **basePath** 속성 설정

OpenAPI 2.0에서 `basePath` 속성을 사용하여 `paths` 속성에 정의된 각 경로 앞에 하나 이상의 경로 파트를 제공할 수 있습니다. API Gateway에 리소스 경로를 표시할 여러 방법이 있기 때문에 Import API 기능은 가져오기 과정에서 `basePath` 속성을 해석하기 위한 옵션(무시, 앞에 추가, 분할)을 제공합니다.

OpenAPI 3.0에서, `basePath`는 더 이상 상위 수준 속성이 아닙니다. 대신 API Gateway는 서버 변수를 규칙으로 사용합니다. Import API 기능은 가져오기 중에 기본 경로를 해석하기 위한 동일한 옵션을 제공합니다. 기본 경로는 다음과 같이 식별됩니다.

- API에 `basePath` 변수가 없을 경우 Import API 기능은 `server.url` 문자열을 확인하여 "/" 위의 경로를 포함하는지 확인합니다. 포함할 경우 이 경로가 기본 경로로 사용됩니다.

- API에 basePath 변수가 한 개만 있을 경우 Import API 기능은 server.url에 참조되지 않았더라도 이 변수를 기본 경로로 사용합니다.
- API에 basePath 변수가 여러 개 있을 경우 Import API 기능은 첫 번째 변수만 기본 경로로 사용합니다.

Ignore

OpenAPI 파일에서 basePath 값이 /a/b/c이고, paths 속성에 /e 및 /f가 포함된 경우 다음 POST 또는 PUT 요청은

```
POST /restapis?mode=import&basepath=ignore
```

```
PUT /restapis/api_id?basepath=ignore
```

API에 다음 리소스를 생성합니다.

- /
- /e
- /f

그 결과 basePath는 존재하지 않는 것처럼 처리되고, 선언된 모든 API 리소스가 호스트를 기준으로 제공됩니다. 예를 들어, 프로덕션 단계를 참조하는 Base Path 및 Stage 값을 포함하지 않는 API 매핑이 포함된 사용자 지정 도메인 이름이 있는 경우 이 옵션을 사용할 수 있습니다.

> **Note**
>
> API Gateway는 정의 파일에 명시적으로 선언되지 않은 경우에도 자동으로 루트 리소스를 생성합니다.

지정하지 않을 경우 basePath는 기본적으로 ignore를 사용합니다.

Prepend

OpenAPI 파일에서 basePath 값이 /a/b/c이고, paths 속성에 /e 및 /f가 포함된 경우 다음 POST 또는 PUT 요청은

```
POST /restapis?mode=import&basepath=prepend
```

```
PUT /restapis/api_id?basepath=prepend
```

API에 다음 리소스를 생성합니다.

- /
- /a
- /a/b
- /a/b/c
- /a/b/c/e
- /a/b/c/f

그 결과 basePath는 (메서드 없는) 추가 리소스 지정으로 처리되고 이 리소스를 선언된 리소스 세트에 추가합니다. 예를 들어, 다른 팀이 API의 다른 파트를 담당하고 있고 basePath가 각 팀의 API 파트에 대한 경로 위치를 참조할 수 있는 경우 이 옵션을 사용할 수 있습니다.

> **Note**
>
> API Gateway는 정의에 명시적으로 선언되지 않은 경우에도 자동으로 중간 리소스를 생성합니다.

분할

OpenAPI 파일에서 basePath 값이 /a/b/c이고, paths 속성에 /e 및 /f가 포함된 경우 다음 POST 또는 PUT 요청은

```
POST /restapis?mode=import&basepath=split
```

```
PUT /restapis/api_id?basepath=split
```

API에 다음 리소스를 생성합니다.

- /

- /b
- /b/c
- /b/c/e
- /b/c/f

그 결과 가장 위의 경로 파트 /a를 리소스 경로의 시작으로 처리하고 API내에 추가 리소스(메서드 없음)를 생성합니다. 예를 들어, a가 API 일부로 공개할 단계 이름인 경우 이 옵션을 사용할 수 있습니다.

AWSOpenAPI 가져오기를 위한 변수

OpenAPI 정의에서 다음 AWS 변수를 사용할 수 있습니다. API Gateway는 API를 가져올 때 변수를 확인합니다. 변수를 지정하려면 ${*variable-name*}을 사용합니다.

AWS 변수

변수 이름	설명
AWS::AccountId	API를 가져오는 AWS 계정 ID입니다(예: 123456789012).
AWS::Partition	API를 가져올 AWS 파티션입니다. 표준 AWS 리전에서 파티션은 aws입니다.
AWS::Region	API를 가져오는 AWS 리전(예: us-east-2)입니다.

AWS 변수 예제

다음 예제에서는 AWS 변수를 사용하여 통합에 대한 AWS Lambda 함수를 지정합니다.

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
```

가져오는 중 발생하는 오류 및 경고

가져오기 중 오류 발생

가져오기 중에 잘못된 OpenAPI 문서 같은 주요 문제에 대한 오류가 생성될 수 있습니다. 오류가 실패한 응답에서 예외(예: `BadRequestException`)로 반환됩니다. 오류가 발생하면 새 API 정의가 삭제되고 기존 API가 변경되지 않습니다.

가져오기 중 경고 발생

가져오는 동안 누락된 모델 참조와 같은 사소한 문제에 대한 경고가 생성될 수 있습니다. 경고가 발생한 경우 `failonwarnings=false` 쿼리 표현식이 요청 URL에 추가되면 작업이 계속됩니다. 그렇지 않으면 업데이트가 롤백됩니다. 기본적으로 `failonwarnings`는 `false`로 설정됩니다. 이 경우 경고가 결과 RestApi 리소스에서 필드로 반환됩니다. 그렇지 않으면 경고가 예외의 메시지로 반환됩니다.

API Gateway에서 REST API 내보내기

API Gateway 콘솔을 사용하거나 달리 API Gateway에서 REST API를 생성 및 구성한 경우, Amazon API Gateway Control Service의 일부인 API Gateway API 내보내기를 사용하여 이를 OpenAPI 파일로 내보낼 수 있습니다. API Gateway 내보내기 API를 사용하려면 API 요청에 서명해야 합니다. 요청 서명에 대한 자세한 내용은 IAM 사용 설명서의 AWS API 요청에 서명을 참조하세요. Postman 확장뿐만 아니라 API Gateway 통합 확장도 내보낸 OpenAPI 정의 파일에 포함시킬 수 있습니다.

> **Note**
>
> AWS CLI를 사용하여 API를 내보낼 경우 다음 예와 같이 확장 파라미터를 포함시켜서 `x-amazon-apigateway-request-validator` 확장이 포함되도록 하세요.
>
> ```
> aws apigateway get-export --parameters extensions='apigateway' --rest-api-id
>  abcdefg123 --stage-name dev --export-type swagger latestswagger2.json
> ```

페이로드가 `application/json` 유형이 아닐 경우 API를 내보낼 수 없습니다. 내보내기를 시도하면 JSON 본문 모델이 없다는 오류 응답을 받게 됩니다.

REST API 내보내기 요청

내보내기 API를 사용하여 GET 요청을 제출하고 내보낼 API를 URL 경로의 일부로 지정하여 기존 REST API를 내보내세요. 요청 URL의 형식은 다음과 같습니다.

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
```

API Gateway 확장(`integration` 값 사용) 또는 Postman 확장을 포함할지(`postman` 값 사용) 지정하는 `extensions` 쿼리 문자열을 추가할 수 있습니다.

그리고 `Accept` 헤더를 `application/json` 또는 `application/yaml`로 설정하여 API 정의 출력을 각각 JSON 또는 YAML 형식으로 받을 수 있습니다.

API Gateway 내보내기 API를 사용하여 GET 요청을 제출하는 자세한 내용은 GetExport를 참조하세요.

> **Note**
>
> API에서 모델을 정의한 경우 API Gateway가 모델을 내보내려면 "application/json" 콘텐츠 유형이어야 합니다. 그렇지 않으면 API Gateway에서 "Only found non-JSON body models for ..." 오류 메시지와 함께 예외가 발생합니다.
> 모델은 속성을 포함하거나 특정 JSONSchema 형식으로 정의해야 합니다.

REST API OpenAPI 정의를 JSON으로 다운로드

OpenAPI 정의의 REST API를 JSON 형식으로 내보내고 다운로드하려면

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
```

예를 들어, 여기서 *<region>*은 us-east-1일 수 있습니다. API Gateway를 사용할 수 있는 전체 리전은 리전 및 엔드포인트를 참조하세요.

REST API OpenAPI 정의를 YAML로 다운로드

OpenAPI 정의의 REST API를 YAML 형식으로 내보내고 다운로드하려면

OpenAPI 3.0

```
GET /restapis/<restapi_id>/stages/<stage_name>/exports/oas30
Host: apigateway.<region>.amazonaws.com
Accept: application/yaml
```

OpenAPI 2.0

```
GET /restapis/<restapi_id>/stages/<stage_name>/exports/swagger
Host: apigateway.<region>.amazonaws.com
Accept: application/yaml
```

Postman 확장이 포함된 REST API OpenAPI 정의를 JSON으로 다운로드

OpenAPI 정의의 REST API를 Postman을 사용해 JSON 형식으로 내보내고 다운로드하려면

OpenAPI 3.0

```
GET /restapis/<restapi_id>/stages/<stage_name>/exports/oas30?extensions=postman
Host: apigateway.<region>.amazonaws.com
Accept: application/json
```

OpenAPI 2.0

```
GET /restapis/<restapi_id>/stages/<stage_name>/exports/swagger?extensions=postman
Host: apigateway.<region>.amazonaws.com
Accept: application/json
```

API Gateway 통합이 포함된 REST API OpenAPI 정의를 YAML로 다운로드

OpenAPI 정의의 REST API를 API Gateway 통합을 사용해 YAML 형식으로 내보내고 다운로드하려면

OpenAPI 3.0

```
GET /restapis/<restapi_id>/stages/<stage_name>/exports/oas30?extensions=integrations
Host: apigateway.<region>.amazonaws.com
Accept: application/yaml
```

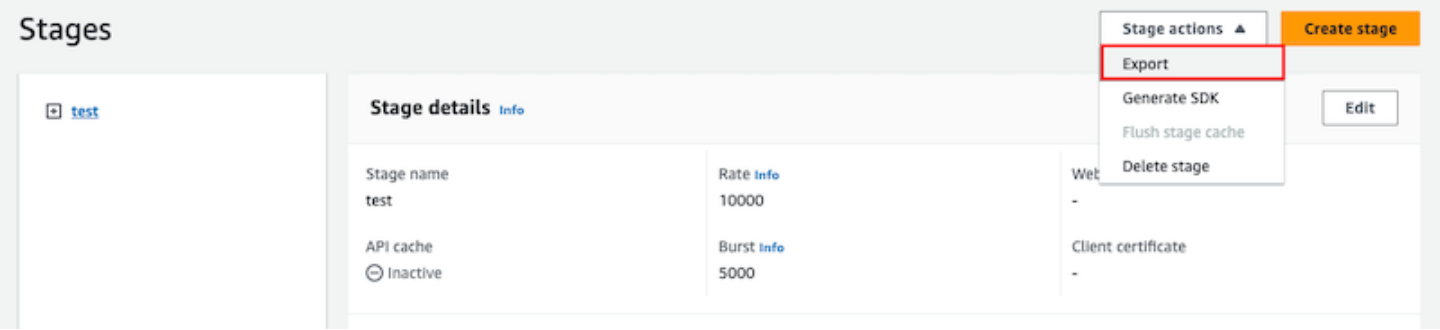
OpenAPI 2.0

```
GET /restapis/<restapi_id>/stages/<stage_name>/exports/swagger?
extensions=integrations
Host: apigateway.<region>.amazonaws.com
Accept: application/yaml
```

API Gateway 콘솔을 사용하여 REST API 내보내기

REST API를 단계로 배포한 후 API Gateway 콘솔을 사용하여 단계의 API를 OpenAPI 파일로 계속 내보낼 수 있습니다.

API Gateway 콘솔의 스테이지 창에서 스테이지 작업, 내보내기를 선택합니다.



API 사양 유형, 형식 및 확장을 지정하여 API의 OpenAPI 정의를 다운로드합니다.

고객이 호출할 수 있도록 REST API 게시

단순히 API Gateway API를 생성하고 개발하면 자동으로 사용자가 API를 호출할 수 있게 되는 것은 아닙니다. 호출이 가능하려면 API를 스테이지에 배포해야 합니다. 또한 사용자가 API에 액세스하는 데 사용할 URL을 사용자 지정할 수 있습니다. 브랜드와 일치하는 도메인이나 API의 기본 URL보다 기억에 남는 도메인을 지정할 수 있습니다.

이 섹션에서는 API를 배포하고 액세스를 위해 사용자에게 제공하는 URL을 사용자 지정하는 방법에 대해 알아볼 수 있습니다.

> **Note**
>
> API Gateway API의 보안을 강화하기 위해 `execute-api.{region}.amazonaws.com` 도메인은 PSL(Public Suffix List)에 등록됩니다. 보안 강화를 위해 API Gateway API 기본 도메인 이름에 민감한 쿠키를 설정해야 하는 경우 `__Host-` 접두사가 있는 쿠키를 사용하는 것이 좋습니다. 이렇게 쿠키를 설정하면 교차 사이트 요청 위조 시도(CSRF)로부터 도메인을 보호하는 데 도움이 됩니다. 자세한 내용은 Mozilla 개발자 네트워크의 Set-Cookie 페이지를 참조하세요.

주제

- Amazon API Gateway에서 REST API 배포

- REST API에 대한 사용자 지정 도메인 이름 설정

Amazon API Gateway에서 REST API 배포

API를 생성한 후, 사용자가 호출할 수 있도록 배포해야 합니다.

API를 배포하려면 API 배포를 생성해 단계에 연결합니다. 스테이지는 API의 수명 주기 상태에 대한 논리적 참조(예: dev, prod, beta, v2)입니다. API 스테이지는 API ID 및 스테이지 이름으로 식별됩니다. 이러한 스테이지는 API를 호출하는 데 사용하는 URL에 포함됩니다. 각 스테이지는 API 배포에 대한 명명된 참조이며, 클라이언트 애플리케이션 프로그램에서 호출을 하는 데 사용할 수 있습니다.

> **Important**
>
> API를 업데이트할 때마다 API를 기존 스테이지 또는 새 스테이지에 다시 배포해야 합니다. API를 업데이트하면 라우팅, 메서드, 통합, 권한 부여자 및 스테이지 설정 이외의 다른 항목이 수정됩니다.

API가 개선됨에 따라 다른 단계에 다른 API 버전으로 계속 API를 배포할 수 있습니다. API 업데이트를 Canary 릴리스 배포 형태로 배포할 수도 있습니다. 이렇게 하면 API 클라이언트가 동일한 스테이지에서 프로덕션 릴리스를 통해 프로덕션 버전에 액세스하고, Canary 릴리스를 통해 업데이트된 버전에 액세스할 수 있습니다.

배포된 API를 호출하기 위해 클라이언트는 API의 URL에 대해 요청을 제출합니다. 이 URL은 API의 프로토콜(HTTP(S) 또는 (WSS)), 호스트 이름, 단계 이름, 리소스 경로(REST API에 대한)에 의해 결정됩니다. 호스트 이름과 스테이지 이름에 따라 API의 기본 URL이 결정됩니다.

API의 기본 도메인 이름을 사용하면 지정된 스테이지(*{stageName}*)의 REST API 기본 URL은 다음 형식이 됩니다.

```
https://{restapi-id}.execute-api.{region}.amazonaws.com/{stageName}
```

API의 기본 URL을 사용자에게 더욱 친숙하게 만들기 위해 사용자 지정 도메인 이름(예: api.example.com)을 생성하여 API의 기본 호스트 이름을 대체할 수 있습니다. 사용자 지정 도메인 이름 아래서 여러 API를 지원하려면 API 단계를 기본 경로에 매핑해야 합니다.

*{api.example.com}*의 사용자 지정 도메인 이름과 사용자 지정 도메인 이름 아래의 기본 경로(*{basePath}*)에 매핑된 API 단계를 사용하면 기본 REST API URL은 다음과 같이 됩니다.

```
https://{api.example.com}/{basePath}
```

각 단계에 대해 기본 계정 수준 요청 조절 한도를 조정하고 API 캐싱을 활성화하여 API 성능을 최적화할 수 있습니다. 또한 CloudTrail 또는 CloudWatch에 대한 API 호출 로깅을 활성화할 수 있으며, 백엔드에서 API 요청을 인증하기 위한 클라이언트 인증서를 선택할 수 있습니다. 뿐만 아니라, 개별 메서드에 대해 스테이지 수준의 설정을 재정의하고, 실행 시간에 스테이지별 환경 컨텍스트를 API 통합에 전달하도록 스테이지 변수를 정의할 수 있습니다.

단계를 통해 API의 강력한 버전 관리가 가능합니다. 예를 들어 API를 `test` 단계와 `prod` 단계에 배포하고, `test` 단계를 테스트 빌드로, `prod` 단계를 안정적 빌드로 사용할 수 있습니다. 업데이트가 테스트를 통과한 후 `test` 단계를 `prod` 단계로 승격할 수 있습니다. 승격은 API를 `prod` 단계에 재배포하거나 단계 변수 값을 `test`의 단계 이름에서 `prod`의 단계 이름으로 업데이트하면 가능합니다.

이 단원에서는 API Gateway 콘솔을 사용하거나 API Gateway REST API를 호출하여 API를 배포하는 방법에 대해 설명합니다. 다른 도구를 사용하려면 AWS CLI 또는 AWS SDK 설명서를 참조하세요.

주제

- API Gateway에서 REST API를 배포합니다.
- REST API에 대한 단계 설정
- API Gateway Canary 릴리스 배포 설정
- 재배포가 필요한 REST API 업데이트

API Gateway에서 REST API를 배포합니다.

API Gateway에서 REST API 배포는 Deployment 리소스로 표현됩니다. RestApi 리소스로 표현되는 API의 실행 파일과 유사합니다.

클라이언트가 API를 호출하기 위해서는 배포를 생성하여 단계를 배포에 연결해야 합니다. 단계는 Stage 리소스로 표시됩니다. 이 리소스는 메서드, 통합, 모델, 매핑 템플릿 및 Lambda 권한 부여자(이전에는 사용자 지정 권한 부여자라고 함)를 포함한 API의 스냅샷을 나타냅니다. API를 업데이트하면 새 단계를 기존 배포에 연결하여 API를 다시 배포할 수 있습니다. 단계 생성은 the section called “단계 설정”에서 다루고 있습니다.

주제

- AWS CLI를 사용하여 배포 생성
- API Gateway 콘솔에서 REST API 배포

AWS CLI를 사용하여 배포 생성

배포를 생성할 때 Deployment 리소스를 인스턴스화합니다. API Gateway 콘솔, AWS CLI, AWS SDK 또는 API Gateway REST API를 사용하여 배포를 생성할 수 있습니다.

CLI를 사용하여 배포를 생성하려면 create-deployment 명령을 사용하세요.

```
aws apigateway create-deployment --rest-api-id <rest-api-id> --region <region>
```

이 배포를 단계에 연결할 때까지는 API를 호출할 수 없습니다. 기존 단계의 경우, 단계의 `deploymentId` 속성을 새로 생성된 배포 ID(<deployment-id>)로 업데이트하여 이 작업을 수행할 수 있습니다.

```
aws apigateway update-stage --region <region> \
   --rest-api-id <rest-api-id> \
   --stage-name <stage-name> \
   --patch-operations op='replace',path='/deploymentId',value='<deployment-id>'
```

API를 처음 배포할 때 단계 생성과 배포 생성을 동시에 조합할 수 있습니다.

```
aws apigateway create-deployment --region <region> \
   --rest-api-id <rest-api-id> \
   --stage-name <stage-name>
```

이 작업은 API를 처음 배포하거나 API를 새 단계에 다시 배포할 때 API Gateway 콘솔의 백그라운드에서 수행됩니다.

API Gateway 콘솔에서 REST API 배포

REST API를 처음 배포하기 이전에 생성해야 합니다. 자세한 내용은 Amazon API Gateway에서 REST API 생성 단원을 참조하세요.

주제

- REST API를 단계에 배포
- REST API를 단계에 재배포
- REST API 배포의 단계 구성 업데이트
- REST API 배포에 대한 단계 변수 설정

- 다른 REST API 배포에 단계 연결

REST API를 단계에 배포

API Gateway 콘솔에서 배포를 생성하고 새 단계 또는 기존 단계에 연결하여 API를 배포할 수 있습니다.

> **Note**
>
> API Gateway의 단계를 다른 배포와 연결하려면 그 대신 다른 REST API 배포에 단계 연결 단원을 참조하세요.

1. https://console.aws.amazon.com/apigateway에서 API Gateway 콘솔에 로그인합니다.
2. 배포하고자 하는 API를 API(APIs) 탐색 창에서 선택합니다.
3. 리소스 창에서 API 배포를 선택합니다.
4. 스테이지의 경우 다음 중 하나를 선택합니다.
   a. 새 스테이지를 생성하려면 새 스테이지를 선택한 다음 스테이지 이름에 이름을 입력합니다. 선택적으로 배포 설명에 배포에 대한 설명을 제공할 수 있습니다.
   b. 기존 스테이지를 선택하려면 드롭다운 메뉴에서 스테이지 이름을 선택합니다. 배포 설명에 새 배포에 대한 설명을 입력할 수 있습니다.
   c. 스테이지와 연결되지 않은 배포를 생성하려면 스테이지 없음을 선택합니다. 나중에 이 배포를 스테이지와 연결할 수 있습니다.
5. [배포]를 선택합니다.

REST API를 단계에 재배포

API를 재배포하려면 the section called “REST API를 단계에 배포”의 단계와 같은 단계를 수행합니다. 원하는 만큼 같은 단계를 재사용할 수 있습니다.

REST API 배포의 단계 구성 업데이트

API를 배포한 후 스테이지 설정을 수정하여 API 캐시, 로깅 또는 요청 스로틀을 활성화 또는 비활성화할 수 있습니다. 또한 백엔드에서 API Gateway를 인증하기 위한 클라이언트 인증서를 선택하고, 실행 시간에 배포 컨텍스트를 API 통합에 전달하도록 단계 변수를 설정할 수 있습니다. 자세한 내용은 단계 설정 업데이트 단원을 참조하세요.

> **Important**
>
> 단계 설정을 수정한 후에는 API를 다시 배포해야 변경 사항이 적용됩니다.

> **Note**
>
> 업데이트된 설정(예: 로깅 활성화)에 새 IAM 역할이 필요한 경우 API를 다시 배포하지 않고 필요한 IAM 역할을 추가할 수 있습니다. 하지만 새 IAM 역할을 적용하려면 몇 분 정도 걸릴 수 있습니다. 적용이 되기 전에는 로깅 옵션을 활성화한 경우에도 API 호출에 대한 트레이스가 기록되지 않습니다.

REST API 배포에 대한 단계 변수 설정

배포를 위해 실행 시간에 배포 관련 데이터를 API 통합에 전달하도록 스테이지 변수를 설정하거나 수정할 수 있습니다. 이 작업은 단계 편집기(Stage Editor)의 단계 변수(Stage Variables) 탭에서 수행할 수 있습니다. 자세한 내용은 REST API 배포에 대한 스테이지 변수 설정 단원의 지침을 참조하세요.

다른 REST API 배포에 단계 연결

배포는 API 스냅샷을 나타내고 단계는 스냅샷에 대한 경로를 정의하므로 다른 배포-단계 조합을 선택하여 사용자가 API의 다른 버전을 호출하는 방법을 제어할 수 있습니다. 이 기능은 API 상태를 이전 배포로 롤백하거나 API의 '프라이빗 분기'를 퍼블릭 분기로 병합하려는 경우 등에 유용합니다.

다음 절차에서는 API Gateway 콘솔에서 Stage Editor(단계 편집기)를 사용하여 이 작업을 수행하는 방법을 보여 줍니다. 여기서는 API를 두 번 이상 배포했다고 가정합니다.

1. 아직 스테이지 창에 있지 않은 경우 기본 탐색 창에서 스테이지를 선택합니다.
2. 업데이트할 스테이지를 선택합니다.
3. 배포 기록 탭에서 스테이지를 사용하고자 하는 배포를 선택합니다.
4. 활성 배포 변경을 선택합니다.
5. 활성 배포를 변경할지 확인하고 활성 배포로 설정 대화 상자에서 활성 배포 변경을 선택합니다.

REST API에 대한 단계 설정

단계란 API의 스냅샷인 배포에 대한 명명된 참조입니다. 단계를 사용하여 특정 배포를 관리하고 최적화합니다. 예를 들어 캐싱을 활성화하거나, 요청 조절을 사용자 지정하거나, 로깅을 구성하거나, 스테이지 변수를 정의하거나, 테스트용 Canary 릴리스를 연결하도록 스테이지 설정을 구성할 수 있습니다.

주제

- API Gateway 콘솔을 사용하여 단계 설정
- API Gateway에서 API 단계에 대한 태그 설정
- REST API 배포에 대한 스테이지 변수 설정

API Gateway 콘솔을 사용하여 단계 설정

주제

- 새 단계 생성
- 단계 설정 업데이트
- 스테이지 수준 설정 재정의
- 단계 삭제

새 단계 생성

처음 배포한 후 단계를 더 추가하고 이를 기존 배포에 연결할 수 있습니다. API Gateway 콘솔을 사용하여 새 단계를 생성하거나, API를 배포할 때 기존 단계를 선택할 수 있습니다. 일반적으로, API를 재배포하기 전에 API 배포에 새 단계를 추가할 수 있습니다. API Gateway 콘솔을 사용하여 새 스테이지를 생성하려면 다음 단계를 따릅니다.

1. https://console.aws.amazon.com/apigateway에서 API Gateway 콘솔에 로그인합니다.
2. REST API를 선택합니다.
3. 기본 탐색 창에서 API에 대해 스테이지를 선택합니다.
4. 스테이지 탐색 창에서 스테이지 생성을 선택합니다.
5. 스테이지 이름에 이름(예: **`prod`**)을 입력합니다.

> **Note**
>
> 단계 이름에는 영숫자, 하이픈, 밑줄만 사용할 수 있습니다. 최대 길이는 128자입니다.

6. (선택 사항). 설명에 간략한 설명을 입력합니다.
7. 배포에서 이 스테이지와 연결하고자 하는 기존 API 배포의 날짜와 시간을 선택합니다.
8. 추가 설정에서 스테이지에 대한 추가 설정을 지정할 수 있습니다.
9. 스테이지 생성을 선택합니다.

단계 설정 업데이트

API 배포에 성공하면 해당 단계가 기본 설정으로 채워집니다. 콘솔 또는 API Gateway REST API를 사용하여 API 캐싱 및 로깅을 포함한 단계 설정을 변경할 수 있습니다. 다음 단계에서는 API Gateway 콘솔의 스테이지 편집기를 사용하여 이 작업을 수행하는 방법을 보여줍니다.

API Gateway 콘솔을 사용하여 단계 설정 업데이트

이러한 단계에서는 단계에 API를 이미 배포한 것으로 가정합니다.

1. https://console.aws.amazon.com/apigateway에서 API Gateway 콘솔에 로그인합니다.
2. REST API를 선택합니다.
3. 기본 탐색 창에서 API에 대해 스테이지를 선택합니다.
4. 단계(Stages) 창에서 단계 명칭을 선택합니다.
5. 스테이지 세부 정보 섹션에서 편집을 선택합니다.
6. (선택 사항) 스테이지 설명에서 설명을 편집합니다.
7. 추가 설정에서 다음 설정을 수정합니다.

캐시 설정

스테이지에 API 캐싱을 활성화하려면 프로비저닝 API 캐시를 켜십시오. 그런 다음 기본 메서드 수준 캐싱, 캐시 용량, 캐시 데이터 암호화, 캐시 time-to-live (TTL) 및 키별 캐시 무효화에 대한 요구 사항을 구성합니다.

기본 메서드 수준 캐싱을 켜거나 특정 방법에 대해 메서드 수준 캐시를 켜기 전까지는 캐싱이 활성화되지 않습니다.

캐시 설정에 관한 자세한 내용은 응답성 향상을 위한 API 캐싱 활성화 섹션을 참조하세요.

> **Note**
>
> API 단계에서 API 캐싱을 활성화하면 AWS 계정에 API 캐싱 요금이 부과될 수 있습니다. 캐싱은 프리 티어에 AWS 적합하지 않습니다.

제한 설정

이 API에 연결된 모든 메서드에 스테이지 수준 제한 대상을 설정하려면 제한을 켭니다.

요율에서 대상 요율을 입력합니다. 토큰이 토큰 버킷에 추가되는 속도(초당 요청 수)입니다. 스테이지 수준 속도는 REST API 구성 및 실행에 대한 API Gateway 할당량에 지정된 계정 수준 속도보다 클 수 없습니다.

버스트에 대상 버스트율을 입력합니다. 버스트율은 토큰 버킷의 용량입니다. 이렇게 하면 일정 기간 동안 목표 요율보다 더 많은 요청을 처리할 수 있습니다. 이 단계 수준 버스트율은 REST API 구성 및 실행에 대한 API Gateway 할당량에 지정된 계정 수준 버스트율보다 클 수 없습니다.

> **Note**
>
> 제한 속도는 엄격한 제한이 아니며 최선으로 적용됩니다. 경우에 따라 클라이언트가 설정한 대상을 초과할 수 있습니다. 비용을 제어하거나 API에 대한 액세스를 차단하기 위해 제한에 의존하지 마십시오. 비용을 모니터링하는 데 AWS Budgets를 사용하고 API 요청을 관리하는 데 AWS WAF를 사용하는 것이 좋습니다.

방화벽 및 인증서 설정

AWS WAF 웹 ACL을 스테이지와 연결하려면 웹 ACL 드롭다운 목록에서 웹 ACL을 선택합니다. 원하는 경우 Block API Request if WebACL cannot be evaluated (Fail- Close)(WebACL을 평가할 수 없는 경우 API 요청 차단(Fail-Close))를 선택합니다.

스테이지의 클라이언트 인증서를 선택하려면 클라이언트 인증서 드롭다운 메뉴에서 인증서를 선택합니다.

8. 저장을 선택합니다.

9. 이 API Gateway API의 이 단계와 관련된 모든 메서드에 대해 Amazon CloudWatch Logs를 활성화하려면 로그 및 추적 섹션에서 편집을 선택합니다.

> **Note**
>
> CloudWatch 로그를 활성화하려면 API Gateway가 사용자를 대신하여 로그에 정보를 기록할 수 있도록 하는 IAM 역할의 ARN도 지정해야 합니다. CloudWatch 이를 위해 APIs(API) 기본 탐색 창에서 설정을 선택합니다. 그런 다음 CloudWatch 로그 역할에 IAM 역할의 ARN을 입력합니다.
> 일반적인 애플리케이션 시나리오에서 IAM 역할은 다음과 같은 액세스 정책 설명이 포함된 `AmazonAPIGatewayPushToCloudWatchLogs`의 관리형 정책을 연결할 수 있습니다.
>
> ```
> {
>     "Version": "2012-10-17",
>     "Statement": [
>         {
>             "Effect": "Allow",
>             "Action": [
>                 "logs:CreateLogGroup",
>                 "logs:CreateLogStream",
>                 "logs:DescribeLogGroups",
>                 "logs:DescribeLogStreams",
>                 "logs:PutLogEvents",
>                 "logs:GetLogEvents",
>                 "logs:FilterLogEvents"
>             ],
>             "Resource": "*"
>         }
>     ]
> }
> ```
>
> IAM 역할에는 다음과 같은 신뢰 관계 설명도 포함되어야 합니다.
>
> ```
> {
>   "Version": "2012-10-17",
>   "Statement": [
>     {
>       "Sid": "",
>       "Effect": "Allow",
>       "Principal": {
> ```

```
        "Service": "apigateway.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

에 대한 CloudWatch 자세한 내용은 Amazon CloudWatch 사용 설명서를 참조하십시오.

10. 로그 수준에 대해 다음 중에서 선택합니다.
    - 오류 수준의 항목만 CloudWatch 로그에 기록하려면 오류만을 선택합니다.
    - 오류 수준의 항목만 CloudWatch 로그에 기록하거나 오류 및 정보 로그를 선택하여 모든 오류 이벤트와 추가 정보 이벤트를 포함하십시오.
    - 전체 API 호출 요청과 응답 정보를 로깅하려면 전체 요청 및 응답 로그를 선택합니다. 전체 요청 및 응답 로그 옵션을 선택하지 않는 한, 민감한 데이터는 로깅되지 않습니다.
11. API CloudWatch Gateway가`API calls`,, `Latency` `Integration latency` `400 errors` `500 errors`, 및 의 API 측정치를 보고하도록 하려면 세부 지표를 선택합니다. 에 대한 CloudWatch 자세한 내용은 Amazon CloudWatch 사용 설명서를 참조하십시오.

> **Important**
>
> 메서드 수준 CloudWatch 지표 액세스 요금은 계정에 청구되지만 API 수준 또는 단계 수준 지표에는 요금이 부과되지 않습니다.

12. 대상에 대한 액세스 로깅을 활성화하려면 사용자 지정 액세스 로깅을 켭니다.
13. 액세스 로그 대상 ARN에 로그 그룹 또는 Firehose 스트림의 ARN을 입력합니다.

    Firehose의 ARN 형식은 입니다. `arn:aws:firehose:`*`{region}`*`:`*`{account-id}`*`:deliverystream/amazon-apigateway-`*`{your-stream-name}`* Firehose 스트림의 이름은 다음과 같아야 합니다. `amazon-apigateway-`*`{your-stream-name}`*
14. 로그 형식에 로그 형식을 입력합니다. 예제 로그 형식에 대한 자세한 내용은 the section called “API Gateway에 대한 CloudWatch 로그 형식” 섹션을 참조하세요.
15. API 스테이지에 대해 AWS X-Ray 추적을 활성화하려면 X-Ray 추적을 선택합니다. 자세한 내용은 X-Ray를 사용하여 REST API에 대한 사용자 요청 추적 섹션을 참조하세요.
16. 변경 사항 저장(Save changes)을 선택합니다. API를 다시 배포하여 새 설정을 적용합니다.

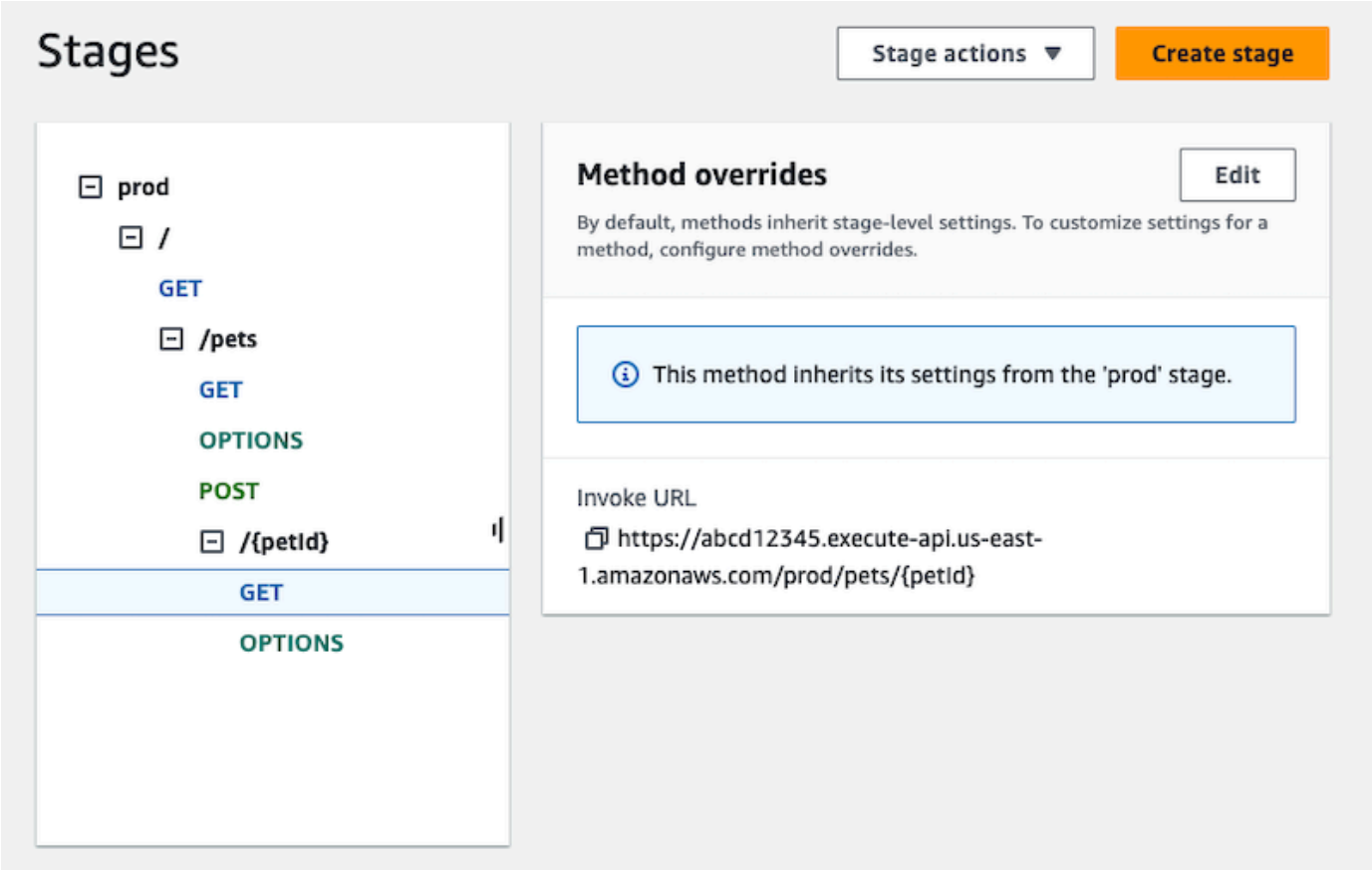
스테이지 수준 설정 재정의

다음과 같이 활성화된 스테이지 수준 설정을 재정의할 수 있습니다. 이러한 옵션 중 일부는 추가 요금이 부과될 수 있습니다. AWS 계정

API Gateway 콘솔을 사용하여 스테이지 수준 설정 재정의

API Gateway 콘솔을 사용하여 스테이지 수준 설정을 재정의하려면

1. 메서드 재정의를 구성하려면 보조 탐색 창 아래에서 스테이지를 확장한 다음 메서드를 선택합니다.



2. 메서드 재정의에서 편집을 선택합니다.
3. 메서드 수준 CloudWatch 설정을 켜려면 CloudWatch 로그에서 로깅 수준을 선택합니다.
4. 메서드 수준의 세부 지표를 켜려면 세부 지표를 선택합니다. 메서드 수준 CloudWatch 메트릭에 액세스하는 경우 계정에 요금이 부과되지만 API 수준 또는 단계 수준 메트릭에 대한 액세스는 청구되지 않습니다.
5. 메서드 수준 제한을 켜려면 제한을 선택합니다. 적절한 메서드 수준 옵션을 입력합니다. 제한에 대한 자세한 내용은 the section called “Throttling”을 참조하십시오.

6. 메서드 수준 캐시를 구성하려면 메서드 캐시 활성화를 선택합니다. 스테이지 세부 정보에서 기본 메서드 수준 캐싱 설정을 변경해도 이 설정에는 영향을 주지 않습니다.
7. 저장을 선택합니다.

단계 삭제

단계가 더 이상 필요하지 않은 경우 해당 단계를 삭제하여 미사용 리소스에 대해 요금이 발생하는 것을 방지할 수 있습니다. 다음 단계에서는 API Gateway 콘솔을 사용하여 단계를 삭제하는 방법을 보여 줍니다.

> **Warning**
>
> 스테이지를 삭제하면 API 호출자가 해당 API의 일부 또는 전체를 사용하지 못할 수도 있습니다. 단계 삭제는 취소할 수 없지만, 단계를 다시 생성한 다음 동일한 배포에 연결할 수 있습니다.

API Gateway 콘솔을 사용하여 단계 삭제

1. https://console.aws.amazon.com/apigateway에서 API Gateway 콘솔에 로그인합니다.
2. REST API를 선택합니다.
3. 기본 탐색 창에서 스테이지를 선택합니다.
4. 스테이지 창에서 삭제하고자 하는 스테이지를 선택한 다음, 스테이지 작업, 스테이지 삭제를 선택합니다.
5. 메시지가 표시되면 **`confirm`**을 입력한 후 삭제를 선택합니다.

API Gateway에서 API 단계에 대한 태그 설정

API Gateway에서 API 단계에 태그를 추가하거나 단계에서 태그를 제거하거나 태그를 볼 수 있습니다. 이렇게 하려면 API Gateway 콘솔, AWS CLI/SDK 또는 API Gateway REST API를 사용할 수 있습니다.

단계는 해당 상위 REST API에서 태그를 상속할 수도 있습니다. 자세한 내용은 the section called “Amazon API Gateway V1 API의 태그 상속” 단원을 참조하세요.

API Gateway 리소스에 태그를 지정하는 방법에 대한 자세한 내용은 태그 지정 단원을 참조하세요.

주제

- API Gateway 콘솔을 사용하여 API 단계에 대한 태그 설정

- 를 사용하여 API 단계에 태그를 설정합니다. AWS CLI
- API Gateway REST API를 사용하여 API 단계의 태그 설정

API Gateway 콘솔을 사용하여 API 단계에 대한 태그 설정

다음 절차에서는 API 단계의 태그를 설정하는 방법을 설명합니다.

API Gateway 콘솔을 사용하여 API 단계의 태그를 설정하려면

1. API Gateway 콘솔에 로그인합니다.
2. 기존 API를 선택하거나 리소스, 메서드, 해당 통합이 포함된 새 API를 만듭니다.
3. 단계를 선택하거나 API를 새 단계에 배포합니다.
4. 기본 탐색 창에서 스테이지를 선택합니다.
5. 태그 탭을 선택합니다. 탭을 표시하려면 오른쪽 화살표 버튼을 선택해야 할 수도 있습니다.
6. 태그 관리를 선택합니다.
7. 태그 편집기에서 태그 추가를 선택합니다. 키 필드에 태그 키(예: `Department`)를 입력하고, 값 필드에 태그 값(예: `Sales`)을 입력합니다. 태그를 저장하려면 저장을 선택합니다.
8. 필요에 따라 단계 5를 반복하여 API 스테이지에 더 많은 태그를 추가합니다. 단계당 최대 태그 수는 50개입니다.
9. 스테이지에서 기존 태그를 제거하려면 제거를 선택합니다.
10. API가 이전에 API Gateway 콘솔에 배포된 경우 변경 사항을 적용하려면 다시 배포해야 합니다.

를 사용하여 API 단계에 태그를 설정합니다. AWS CLI

create-stage 명령 또는 tag-resource 명령을 AWS CLI 사용하여 API 단계에 태그를 설정할 수 있습니다. untag-resource 명령을 사용하여 API 단계에서 하나 이상의 태그를 삭제할 수 있습니다.

다음 예제에서는 단계를 생성할 때 태그를 추가합니다. `test`

```
aws apigateway create-stage --rest-api-id abc1234 --stage-name test --description
 'Testing stage' --deployment-id efg456 --tag Department=Sales
```

다음 예제에서는 `prod` 스테이지에 태그를 추가합니다.

```
aws apigateway tag-resource --resource-arn arn:aws:apigateway:us-east-2::/
restapis/abc123/stages/prod --tags Department=Sales
```

다음 예제에서는 test 스테이지에서 Department=Sales 태그를 제거합니다.

```
aws apigateway untag-resource --resource-arn arn:aws:apigateway:us-east-2::/
restapis/abc123/stages/test --tag-keys Department
```

API Gateway REST API를 사용하여 API 단계의 태그 설정

다음 중 하나를 수행하면 API Gateway REST API를 사용하여 API 단계의 태그를 설정할 수 있습니다.

- API 단계에 태그를 지정하려면 tags:tag를 호출합니다.
- API 단계에서 하나 이상의 태그를 삭제하려면 tags:untag를 호출합니다.
- 생성 중인 API 단계에 하나 이상의 태그를 추가하려면 stage:create를 호출합니다.

tags:get을 호출하여 API 단계에서 태그를 설명할 수도 있습니다.

API 단계에 태그 지정

API(m5zr3vnks7)를 단계(test)에 배포한 후 tags:tag를 호출하여 단계에 태그를 지정합니다. 필요한 단계 Amazon 리소스 이름(ARN)(arn:aws:apigateway:us-east-1::/restapis/m5zr3vnks7/stages/test)은 URL로 인코딩되어야 합니다(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"
  }
}
```

이전 요청을 사용하여 기존 태그를 새 값으로 업데이트할 수도 있습니다.

stage:create를 호출하여 단계를 생성할 때 단계에 태그를 추가할 수 있습니다.

```
POST /restapis/<restapi_id>/stages

{
  "stageName" : "test",
  "deploymentId" : "adr134",
  "description" : "test deployment",
```

```
  "cacheClusterEnabled" : "true",
  "cacheClusterSize" : "500",
  "variables" : {
    "sv1" : "val1"
  },
  "documentationVersion" : "test",

  "tags" : {
    "Department" : "Sales",
    "Division" : "Retail"
  }
}
```

API 단계에서 태그 해제

단계에서 Department 태그를 제거하려면 tags:untag를 호출합니다.

```
DELETE /tags/arn%3Aaws%3Aapigateway%3Aus-east-1%3A%3A%2Frestapis%2Fm5zr3vnks7%2Fstages
%2Ftest?tagKeys=Department
Host: apigateway.us-east-1.amazonaws.com
Authorization: ...
```

둘 이상의 태그를 제거하려면 쿼리 표현식에서 쉼표로 구분된 태그 키 목록을 사용합니다. 예를 들면 다음과 같습니다. ?tagKeys=Department,Division,…

API 단계의 태그 설명

지정된 단계에서 기존 태그를 설명하려면 tags:get을 호출합니다.

```
GET /tags/arn%3Aaws%3Aapigateway%3Aus-east-1%3A%3A%2Frestapis%2Fm5zr3vnks7%2Fstages
%2Ftags
Host: apigateway.us-east-1.amazonaws.com
Authorization: ...
```

성공적인 응답은 다음과 같습니다.

```
200 OK

{
    "_links": {
        "curies": {
            "href": "http://docs.aws.amazon.com/apigateway/latest/developerguide/
restapi-tags-{rel}.html",
```

```
            "name": "tags",
            "templated": true
        },
        "tags:tag": {
            "href": "/tags/arn%3Aaws%3Aapigateway%3Aus-east-1%3A%3A%2Frestapis
%2Fm5zr3vnks7%2Fstages%2Ftags"
        },
        "tags:untag": {
            "href": "/tags/arn%3Aaws%3Aapigateway%3Aus-east-1%3A%3A%2Frestapis
%2Fm5zr3vnks7%2Fstages%2Ftags{?tagKeys}",
            "templated": true
        }
    },
    "tags": {
        "Department": "Sales"
    }
}
```

REST API 배포에 대한 스테이지 변수 설정

단계 변수는 REST API의 배포 단계와 연결된 구성 속성으로 정의되는 이름-값 페어입니다. 환경 변수와 비슷한 역할을 하며 API 설정 및 매핑 템플릿에 사용할 수 있습니다.

예를 들어 단계 구성에서 단계 변수를 정의한 다음, 변수 값을 REST API의 메서드에 대한 HTTP 통합의 URL 문자열로 설정할 수 있습니다. 나중에 API 설정에 있는 연결된 스테이지 변수 이름을 사용하여 URL 문자열을 참조할 수 있습니다. 이렇게 하면 스테이지 변수 값을 해당 URL로 재설정함으로써 각 스테이지의 서로 다른 엔드포인트에서 동일한 API 설정을 사용할 수 있습니다.

매핑 템플릿에서 단계 변수를 액세스하거나 구성 파라미터를 AWS Lambda 또는 HTTP 백엔드에 전달할 수도 있습니다.

매핑 템플릿에 대한 자세한 내용은 API Gateway 매핑 템플릿과 액세스 로깅 변수 참조 단원을 참조하세요.

> **Note**
>
> 단계 변수는 자격 증명과 같은 중요한 데이터에 사용할 수 없습니다. 민감한 데이터를 통합에 전달하려면 AWS Lambda 권한 부여자를 사용하세요. Lambda 권한 부여자의 출력에서 중요한 데이터를 통합에 전달할 수 있습니다. 자세한 내용은 the section called “Amazon API Gateway Lambda 권한 부여자의 출력” 단원을 참조하세요.

사용 사례

API Gateway의 배포 단계에서 각 API에 대한 여러 릴리스 단계(예: 알파, 베타 및 프로덕션)를 관리할 수 있습니다. 단계 변수를 사용하여 API 배포 단계가 다양한 백엔드 엔드포인트와 상호 작용하도록 구성할 수 있습니다.

예를 들어, API에서 백엔드 웹 호스트(예: `http://example.com`)에 GET 요청을 HTTP 프록시로 전달할 수 있습니다. 이 경우 개발자가 프로덕션 엔드포인트를 호출하면 API Gateway가 example.com을 호출하도록 단계 변수에서 백엔드 웹 호스트가 구성됩니다. 베타 엔드포인트를 호출하면, API Gateway는 베타 단계의 단계 변수에서 구성된 값을 사용하고 다른 웹 호스트를 호출합니다(예: `beta.example.com`). 마찬가지로 단계 변수를 사용하여 API의 각 단계에 대해 서로 다른 AWS Lambda 함수 이름을 지정할 수 있습니다.

또한 단계 변수를 사용하여 매핑 템플릿을 통해 구성 파라미터를 Lambda 함수에 전달할 수도 있습니다. 예를 들어 API의 여러 단계에 동일한 Lambda 함수를 다시 사용하려고 할 수 있지만, 이 함수는 호출되는 단계에 따라 다른 Amazon DynamoDB 테이블에서 데이터를 읽어야 합니다. 이때 Lambda 함수에 대한 요청을 생성하는 매핑 템플릿에서 단계 변수를 사용하여 테이블 이름을 Lambda에 전달할 수 있습니다.

예

스테이지 변수를 사용하여 HTTP 통합 엔드포인트를 사용자 지정하려면, 먼저 지정된 이름(예: **`url`**)의 스테이지 변수를 구성하고 여기에 값(예: **`example.com`**)을 할당합니다. 그런 다음 메소드 구성에서 HTTP 프록시 통합을 설정합니다. 엔드포인트의 URL을 입력하는 대신, 단계 변수 값인 **`http://${stageVariables.url}`**을 사용하도록 API Gateway에 지시할 수 있습니다. 이 값은 API가 실행되는 단계에 따라 실행 시간에 단계 변수 `${}`를 대체하도록 API Gateway에 지시합니다.

비슷한 방식으로 단계 변수를 참조하여 자격 증명 필드에 Lambda 함수 이름, 서비스 프록시 경로 AWS 또는 AWS 역할 ARN을 지정할 수 있습니다.

Lambda 함수 이름을 단계 변수 값으로 지정할 때는 Lambda 함수에 대한 권한을 수동으로 구성해야 합니다. API Gateway 콘솔에서 Lambda 함수를 지정하면 적절한 권한을 구성하기 위한 명령이 팝업됩니다. AWS CLI AWS Command Line Interface (AWS CLI) 를 사용하여 이 작업을 수행할 수도 있습니다.

```
aws lambda add-permission --function-name "arn:aws:lambda:us-
east-2:123456789012:function:my-function" --source-arn "arn:aws:execute-api:us-
east-2:123456789012:api_id/*/HTTP_METHOD/resource" --principal apigateway.amazonaws.com
 --statement-id apigateway-access --action lambda:InvokeFunction
```

Amazon API Gateway 콘솔을 사용하여 단계 변수 설정

이 자습서에서는 Amazon API Gateway 콘솔을 사용하여 샘플 API의 두 가지 배포 단계에 대해 단계 변수를 설정하는 방법을 알아봅니다. 시작하기 전에 다음 사전 요구 사항을 충족하는지 확인하세요.

- API Gateway에 사용 가능한 API가 있어야 합니다. Amazon API Gateway에서 REST API 생성의 지침을 따르세요.
- API는 1회 이상 배포되어야 합니다. Amazon API Gateway에서 REST API 배포의 지침을 따르세요.
- 배포된 API에 대한 첫 단계를 생성해야 합니다. 새 단계 생성의 지침을 따르세요.

API Gateway 콘솔을 사용하여 단계 변수를 선언하려면

1. https://console.aws.amazon.com/apigateway에서 API Gateway 콘솔에 로그인합니다.
2. API를 생성한 다음 API의 루트 리소스에 GET 메서드를 생성합니다. 통합 유형을 HTTP로 설정하고 엔드포인트 URL을 **`http://${stageVariables.url}`**로 설정합니다.
3. API를 **`beta`**라는 스테이지에 배포합니다.
4. 기본 탐색 창에서 스테이지를 선택한 후 베타 스테이지를 선택합니다.
5. 스테이지 변수 탭에서 편집을 선택합니다.
6. 스테이지 변수 추가를 선택합니다.
7. 이름에 **`url`**를 입력합니다. 값에 **`httpbin.org/get`**을 입력합니다.
8. 스테이지 변수 추가를 선택하고 다음을 수행합니다.

   이름에 **`stageName`**를 입력합니다. 값에 **`beta`**을 입력합니다.
9. 스테이지 변수 추가를 선택하고 다음을 수행합니다.

   이름에 **`function`**를 입력합니다. 값에 **`HelloWorld`**을 입력합니다.

> **Note**
>
> Lambda 함수를 단계 변수의 값으로 설정할 때는 **`HelloWorld`**, **`HelloWorld:1`** 또는 **`HelloWorld:alpha`**와 같이 별칭이나 버전 사양이 포함될 수 있는 함수의 로컬 이름을 사용합니다. 해당 함수의 ARN을 사용하면 안 됩니다(예: **`arn:aws:lambda:us-east-1:123456789012:function:HelloWorld`**). API Gateway 콘솔은 Lambda 함수에 대한 스테이지 변수 값을 정규화되지 않은 함수 이름으로 가정하고 주어진 스테이지 변수를 ARN으로 확장합니다.

10. 저장을 선택합니다.
11. 이제 두 번째 스테이지를 생성합니다. 스테이지 탐색 창에서 스테이지 생성을 선택합니다. 단계 이름에 **prod**를 입력합니다. 배포에서 최신 배포를 선택한 후 스테이지 생성을 선택합니다.
12. 베타 스테이지와 마찬가지로, 동일한 세 스테이지 변수(url, stageName, function)를 각각 다른 값(**petstore-demo-endpoint.execute-api.com/petstore/pets**, **prod**, **HelloEveryone**)으로 설정합니다.

    스테이지 변수를 사용하는 방법은 the section called “스테이지 변수 사용 ” 섹션을 참조하세요.

Amazon API Gateway 단계 변수 사용

API Gateway 단계 변수를 사용하여 다양한 API 배포 단계의 HTTP 및 Lambda 백엔드에 액세스할 수 있습니다. 또한 단계 변수를 사용하여 스테이지별 구성 메타데이터를 HTTP 백엔드에는 쿼리 파라미터로 전달하고 Lambda 함수에는 입력 매핑 템플릿에서 생성되는 페이로드로 전달할 수도 있습니다.

필수 조건

url 스테이지 변수가 두 가지 HTTP 엔드포인트로 설정된 두 스테이지를 생성해야 합니다. 즉, 두 가지 Lambda 함수에 할당된 function 스테이지 변수 및 스테이지별 메타데이터가 포함된 stageName 스테이지 변수를 말합니다.

단계 변수를 사용하여 API를 통해 HTTP 엔드포인트 액세스

1. 단계 탐색 창에서 베타를 선택합니다. 스테이지 세부 정보에서 복사 아이콘을 선택해 API의 호출 URL을 복사하여 웹 브라우저에 입력합니다. 그러면 API의 루트 리소스에 대한 [beta] 단계 `GET` 요청이 시작됩니다.

   > **Note**
   >
   > [Invoke URL] 링크는 [beta] 단계에서 API의 루트 리소스를 가리킵니다. 웹 브라우저에 URL을 입력하면 루트 리소스에서 링크 베타 스테이지 `GET` 메서드를 호출합니다. 루트 리소스 자체가 아닌 하위 리소스에 메서드가 정의된 경우, 웹 브라우저에 URL을 입력하면 `{"message":"Missing Authentication Token"}` 오류 응답이 반환됩니다. 이 경우 특정 하위 리소스의 이름을 URL 호출(Invoke URL) 링크에 추가해야 합니다.

2. [beta] 단계 `GET` 요청에서 받은 응답은 다음에 표시됩니다. 브라우저로 [http://httpbin.org/get] 페이지를 탐색해 결과를 확인할 수도 있습니다. 이 값은 [beta] 단계의 `url` 변수에 할당되었습니다. 이 두 응답은 동일합니다.

3. 단계(Stages) 탐색 창에서 prod 단계를 선택합니다. 스테이지 세부 정보에서 복사 아이콘을 선택해 API의 호출 URL을 복사하여 웹 브라우저에 입력합니다. 그러면 API의 루트 리소스에 대한 prod 단계 GET 요청이 시작됩니다.
4. [prod] 단계 GET 요청에서 받은 응답은 다음에 표시됩니다. 브라우저를 사용하여 petstore-demo-endpointhttp://.execute-api.com/petstore/pets로 이동하여 결과를 확인할 수 있습니다. 이 값은 [prod] 단계의 `url` 변수에 할당되었습니다. 이 두 응답은 동일합니다.

쿼리 파라미터 표현식에서 스테이지 변수를 통해 HTTP 백엔드에 스테이지별 메타데이터 전달

이 절차에서는 쿼리 파라미터 표현식에서 스테이지 변수 값을 사용하여 스테이지별 메타데이터를 HTTP 백엔드로 전달하는 방법에 대해 설명합니다. `stageName`에 설명된 Amazon API Gateway 콘솔을 사용하여 단계 변수 설정 단계 변수를 사용하겠습니다.

1. 리소스 탐색 창에서 GET 메서드를 선택합니다.

   쿼리 문자열 파라미터를 메서드의 URL에 추가하려면 메서드 요청 탭을 선택한 다음 메서드 요청 설정 섹션에서 편집을 선택합니다.
2. URL 쿼리 문자열 파라미터를 선택하고 다음을 수행합니다.

   a. 쿼리 문자열 추가(Add query string)를 선택합니다.

   b. 이름에 **`stageName`**를 입력합니다.

   c. 필수 상태로 유지하고 캐싱을 해제합니다.
3. 저장을 선택합니다.
4. 통합 요청 탭을 선택한 다음 통합 요청 설정 섹션에서 편집을 선택합니다.
5. 엔드포인트 URL의 경우 이전에 정의한 URL 값에 **`?stageName=${stageVariables.stageName}`**을 추가하여 전체 엔드포인트 URL이 **`http://${stageVariables.url}?stageName=${stageVariables.stageName}`**이 되도록 합니다.
6. 배포 API를 선택하고 베타 스테이지를 선택합니다.
7. 기본 탐색 창에서 스테이지를 선택합니다. 단계 탐색 창에서 베타를 선택합니다. 스테이지 세부 정보에서 복사 아이콘을 선택해 API의 호출 URL을 복사하여 웹 브라우저에 입력합니다.

> **Note**
>
> `url` 변수 "http://httpbin.org/get"에서 지정한 바와 같이 HTTP 엔드포인트가 쿼리 파라미터 표현식을 수락하고 응답으로 `args` 객체로 반환하기 때문에 베타 스테이지를 사용합니다.

8. 응답은 다음과 같습니다. `beta` 단계 변수에 할당된 `stageName`는 백엔드에 `stageName` 인수로 전달됩니다.


```
{
  "args": {
    "stageName": "beta"
  },
  "headers": {
    "Accept": "application/json",
    "Host": "httpbin.org",
    "User-Agent": "AmazonAPIGateway_abc1234",
    "X-Amzn-Apigateway-Api-Id": "abc1234",
    "X-Amzn-Trace-Id": "Self=1234567890abcdef0;Root=021345abcdef6789"
  },
  "origin": "192.0.2.9",
  "url": "http://httpbin.org/get?stageName=beta"
}
```


단계 변수를 사용하여 API를 통해 Lambda 함수 호출

이 절차에서는 스테이지 변수를 사용하여 Lambda 함수를 API 백엔드로 호출하는 방법을 설명합니다. 앞서 설명한 `function` 단계 변수를 사용하겠습니다. 자세한 설명은 Amazon API Gateway 콘솔을 사용하여 단계 변수 설정 섹션을 참조하세요.

1. 기본 Node.js 런타임을 사용하여 **`HelloWorld`**라는 Lambda 함수를 생성합니다. 코드에는 다음이 포함되어 있습니다.

```
export const handler = function(event, context, callback) {
    if (event.stageName)
        callback(null, 'Hello, World! I\'m calling from the ' + event.stageName + '
 stage.');
    else
        callback(null, 'Hello, World! I\'m not sure where I\'m calling from...');
};
```

Lambda 함수를 생성하는 방법에 대한 자세한 내용은 REST API 콘솔 시작하기를 참조하세요.

2. 리소스 창에서 리소스 생성을 선택하고 다음을 수행합니다.

   a. 리소스 경로에서 /를 선택합니다.

   b. 리소스 이름에 **lambdav1**을 입력합니다.

   c. 리소스 생성을 선택합니다.

3. /lambdav1 리소스를 선택한 다음 메서드 생성을 선택합니다.

   이어서 다음을 수행합니다.

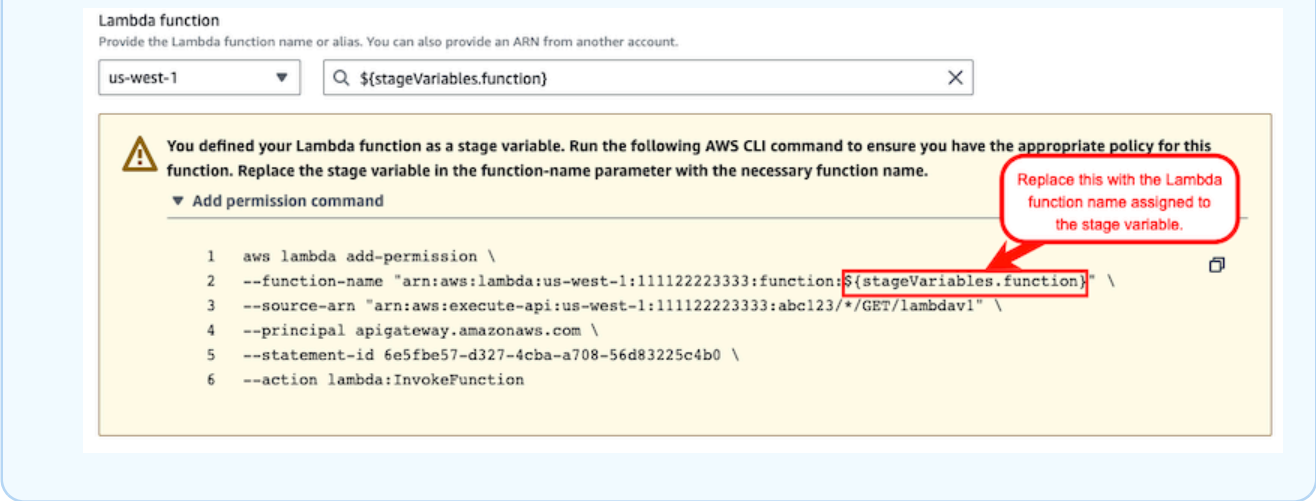
   a. 메서드 유형에서 GET을 선택합니다.

   b. 통합 유형에서 Lambda 함수를 선택합니다.

   c. Lambda 프록시 통합을 비활성화된 상태로 유지합니다.

   d. Lambda 함수에서 `${stageVariables.function}`을(를) 입력합니다.

> **Tip**
>
> 권한 명령 추가라는 메시지가 표시되면 AWS CLI 명령을 복사합니다. `function` 스테이지 변수에 할당될 각 Lambda 함수에서 명령을 실행합니다. 예를 들어, `$stageVariables.function` 값이 이면 `HelloWorld` 다음 명령을 실행합니다. AWS CLI
>
> ```
> aws lambda add-permission --function-name arn:aws:lambda:us-
> east-1:account-id:function:HelloWorld --source-arn arn:aws:execute-
> api:us-east-1:account-id:api-id/*/GET/lambdav1 --principal
>  apigateway.amazonaws.com --statement-id statement-id-guid --action
>  lambda:InvokeFunction
> ```
>
> 그렇지 않으면 메서드 호출 시 `500 Internal Server Error` 응답이 반환됩니다. `${stageVariables.function}`을 스테이지 변수에 할당된 Lambda 함수 이름으로 바꾸어야 합니다.



e. 메서드 생성을 선택합니다.

4. **prod** 및 **beta** 스테이지 모두에 API를 배포합니다.
5. 기본 탐색 창에서 스테이지를 선택합니다. 단계 탐색 창에서 베타를 선택합니다. 스테이지 세부 정보에서 복사 아이콘을 선택해 API의 호출 URL을 복사하여 웹 브라우저에 입력합니다. URL에 **/lambdav1**을 추가한 다음 Enter 키를 누릅니다.

   응답은 다음과 같습니다.

```
"Hello, World! I'm not sure where I'm calling from..."
```

단계 변수를 통해 Lambda 함수에 단계별 메타데이터 전달

이 절차에서는 단계 변수를 사용하여 Lambda 함수로 단계별 구성 메타데이터를 전달하는 방법을 설명합니다. POST 메서드 및 입력 매핑 템플릿을 만들어 앞서 설명한 `stageName` 스테이지 변수를 사용해 페이로드를 생성합니다.

1. /lambdav1 리소스를 선택한 다음 Create 메서드를 선택합니다.

   이어서 다음을 수행합니다.

   a. 메서드 유형에서 POST를 선택합니다.
   b. 통합 유형에서 Lambda 함수를 선택합니다.
   c. Lambda 프록시 통합을 비활성화된 상태로 유지합니다.
   d. Lambda 함수에서 `${stageVariables.function}`을(를) 입력합니다.
   e. 권한 명령 추가라는 메시지가 표시되면 AWS CLI 명령을 복사합니다. `function` 스테이지 변수에 할당될 각 Lambda 함수에서 명령을 실행합니다.

f. 메서드 생성을 선택합니다.

2. 통합 요청 탭을 선택한 다음 통합 요청 설정 섹션에서 편집을 선택합니다.
3. 매핑 템플릿을 선택한 다음 매핑 템플릿 추가를 선택합니다.
4. 콘텐츠 유형에 **application/json**을 입력합니다.
5. 템플릿 본문에 다음 템플릿을 입력합니다.

```
#set($inputRoot = $input.path('$'))
{
    "stageName" : "$stageVariables.stageName"
}
```

> **Note**
>
> 매핑 템플릿에서 스테이지 변수는 따옴표 안에 참조되어야 합니다("$stageVariables.stageName" 또는 "${stageVariables.stageName}"에서와 같이). 다른 곳에서는 따옴표 없이 참조해야 합니다(${stageVariables.function}에서와 같이).

6. 저장을 선택합니다.
7. **beta** 및 **prod** 스테이지 모두에 API를 배포합니다.
8. REST API 클라이언트를 사용하여 스테이지별 메타데이터를 전달하려면 다음을 수행합니다.

a. 단계 탐색 창에서 베타를 선택합니다. 스테이지 세부 정보에서 복사 아이콘을 선택해 API의 호출 URL을 복사하여 REST API 클라이언트의 입력 필드에 입력합니다. **/lambdav1**을 추가한 후 요청을 제출합니다.

응답은 다음과 같습니다.

```
"Hello, World! I'm calling from the beta stage."
```

b. 스테이지 탐색 창에서 프로덕션을 선택합니다. 스테이지 세부 정보에서 복사 아이콘을 선택해 API의 호출 URL을 복사하여 REST API 클라이언트의 입력 필드에 입력합니다. **/lambdav1**을 추가한 후 요청을 제출합니다.

응답은 다음과 같습니다.

```
"Hello, World! I'm calling from the prod stage."
```

9. 테스트 기능을 사용하여 스테이지별 메타데이터를 전달하려면 다음을 수행합니다.

   a. 리소스 탐색 창에서 테스트 탭을 선택합니다. 탭을 표시하려면 오른쪽 화살표 버튼을 선택해야 할 수도 있습니다.

   b. function에 **HelloWorld**를 입력합니다.

   c. stageName에 **beta**를 입력합니다.

   d. 테스트를 선택합니다. POST 요청에 본문을 추가할 필요는 없습니다.

      응답은 다음과 같습니다.

      ```
      "Hello, World! I'm calling from the beta stage."
      ```

   e. 이전 단계를 반복하여 프로덕션 스테이지를 테스트할 수 있습니다. stageName에 **Prod**를 입력합니다.

      응답은 다음과 같습니다.

      ```
      "Hello, World! I'm calling from the prod stage."
      ```

Amazon API Gateway 단계 변수 참조

다음과 같은 경우에 API Gateway 단계 변수를 사용할 수 있습니다.

파라미터 매핑 표현식

API 메서드의 요청 또는 응답 헤더 파라미터를 부분적으로 바꾸지 않고도 파라미터 매핑 표현식에 단계 변수를 사용할 수 있습니다. 다음 예에서는 $ 및 {...}로 묶지 않고 단계 변수를 참조합니다.

- `stageVariables.<variable_name>`

매핑 템플릿

다음 예에서 보듯 단계 변수는 매핑 템플릿 어디에서나 사용할 수 있습니다.

- `{ "name" : "$stageVariables.<variable_name>"}`
- `{ "name" : "${stageVariables.<variable_name>}"}`

HTTP 통합 URI

다음 예제에서 보듯 스테이지 변수를 HTTP 통합 URL의 일부로 사용할 수 있습니다.

- 프로토콜이 없는 전체 URI - `http://${stageVariables.<variable_name>}`
- 전체 도메인 - `http://${stageVariables.<variable_name>}/resource/operation`
- 하위 도메인 - `http://${stageVariables.<variable_name>}.example.com/resource/operation`
- 경로 - `http://example.com/${stageVariables.<variable_name>}/bar`
- 쿼리 문자열 - `http://example.com/foo?q=${stageVariables.<variable_name>}`

AWS 통합 URI

다음 예제와 같이 단계 변수를 AWS URI 작업 또는 경로 구성 요소의 일부로 사용할 수 있습니다.

- `arn:aws:apigateway:<region>:<service>:${stageVariables.<variable_name>}`

AWS 통합 URI (람다 함수)

다음 예제와 같이 Lambda 함수 이름 또는 버전/별칭 대신에 단계 변수를 사용할 수 있습니다.

- `arn:aws:apigateway:<region>:lambda:path/2015-03-31/functions/arn:aws:lambda:<region>:<account_id>:function:${stageVariables.<function_variable_name>}/invocations`
- `arn:aws:apigateway:<region>:lambda:path/2015-03-31/functions/arn:aws:lambda:<region>:<account_id>:function:<function_name>:${stageVariables.<version_variable_name>}/invocations`

> **Note**
>
> Lambda 함수에 대해 단계 변수를 사용하려면 함수가 API와 동일한 계정에 있어야 합니다. 단계 변수는 교차 계정 Lambda 함수를 지원하지 않습니다.

AWS 통합 자격 증명

다음 예와 같이 단계 변수를 AWS 사용자/역할 자격 증명 ARN의 일부로 사용할 수 있습니다.

- `arn:aws:iam::<account_id>:${stageVariables.<variable_name>}`

API Gateway Canary 릴리스 배포 설정

Canary 릴리스란 (다른 소프트웨어뿐 아니라) API의 새 버전을 테스트 목적으로 배포하고 같은 단계에서의 정상 작동을 위해 기본 버전은 프로덕션 릴리스로 배포하는 소프트웨어 개발 전략입니다. 논의를 위해 이 설명서에서는 기본 버전을 프로덕션 릴리스라고 합니다. 이 방법이 합리적이기는 하지만 테스트를 위해 프로덕션 버전이 아닌 어떤 버전에서도 자유롭게 Canary 릴리스를 적용할 수 있습니다.

Canary 릴리스 배포에서 전체 API 트래픽은 사전 구성된 비율로 프로덕션 릴리스와 Canary 릴리스로 임의로 분할됩니다. 일반적으로 Canary 릴리스에는 API 트래픽 중 작은 백분율이 주어지며, 프로덕션 릴리스가 나머지를 차지합니다. 업데이트된 API 기능은 Canary를 통하는 API 트래픽에만 보입니다. 테스트 범위 또는 성능을 최적화하기 위해 Canary 트래픽 백분율을 조정할 수 있습니다.

Canary 트래픽을 적은 수준으로 유지하고 선택이 임의로 이루어지면 대부분의 사용자는 새 버전의 잠재적 버그로 인한 부정적 영향을 받지 않으며, 단일 사용자가 부정적 영향을 계속 받는 경우는 없습니다.

테스트 지표가 요구 사항을 통과한 후에는 Canary 릴리스를 프로덕션 릴리스로 승격하고 배포에서 Canary를 비활성화할 수 있습니다. 이렇게 하면 새 기능을 프로덕션 단계에서 사용할 수 있습니다.

주제

- API Gateway에서 Canary 릴리스 배포
- Canary 릴리스 배포 생성
- Canary 릴리스 업데이트
- Canary 릴리스 승격
- Canary 릴리스 비활성화

API Gateway에서 Canary 릴리스 배포

API Gateway에서 Canary 릴리스 배포는 API 기본 버전의 프로덕션 릴리스를 위한 배포 단계를 사용하고, API의 기본 버전과 관련하여 새 버전에 대한 Canary 릴리스를 단계에 연결합니다. 이 단계는 초기 배포에 연결되고, Canary는 후속 배포에 연결됩니다. 처음에는 단계와 Canary 모두 동일한 API 버전을 가리킵니다. 이 단원에서는 단계와 프로덕션 릴리스를 구분하지 않고 사용하며, Canary와 Canary 릴리스를 구분하지 않고 사용합니다.

Canary 릴리스가 포함된 API를 배포하려면 일반 배포의 단계에 Canary 설정을 추가하여 Canary 릴리스 배포를 생성합니다. Canary 설정은 기본 Canary 릴리스를 설명하고, 단계는 이 배포 내의 API 프로

덕션 릴리스를 나타냅니다. Canary 설정을 추가하려면 배포 단계에서 `canarySettings`를 설정하고 다음을 지정하세요.

- 처음에는 단계에서 설정된 기본 버전 배포의 ID와 동일한 배포 ID.
- Canary 릴리스의 API 트래픽 백분율(0.0~100.0%, 0.0%와 100.0% 포함).
- 프로덕션 릴리스 단계 변수를 재정의할 수 있는 Canary 릴리스의 단계 변수.
- useStageCache가 설정되어 있고 API 캐싱이 단계에서 활성화된 경우, Canary 요청에 단계 캐시 사용.

Canary 릴리스가 활성화된 후에는 Canary 릴리스가 비활성화되고 단계에서 Canary 설정이 제거될 때까지 배포 단계를 다른 비Canary 릴리스 배포에 연결할 수 없습니다.

API 실행 로깅을 활성화하면 Canary 릴리스는 모든 Canary 요청에 대해 자체 로그와 지표를 생성합니다. 이 로그와 지표는 프로덕션 단계 CloudWatch Logs 로그 그룹뿐 아니라 Canary별 CloudWatch Logs 로그 그룹에도 보고됩니다. 액세스 로깅도 마찬가지입니다. 별도의 Canary별 로그는 새 API 변경의 유효성을 검사하여 변경을 수락해 Canary 릴리스를 프로덕션 단계로 승격하거나 변경을 취소하고 Canary 릴리스를 프로덕션 단계에서 되돌릴지 여부를 결정하는 데 유용합니다.

프로덕션 단계 실행 로그 그룹은 `API-Gateway-Execution-Logs/{rest-api-id}/{stage-name}`으로 명명되며, Canary 릴리스 실행 로그 그룹은 `API-Gateway-Execution-Logs/{rest-api-id}/{stage-name}/Canary`로 명명됩니다. 액세스 로깅의 경우, 새 로그 그룹을 만들거나 기존 로그 그룹을 선택해야 합니다. Canary 릴리스 액세스 로그 그룹 이름은 선택한 로그 그룹 이름에 `/Canary` 접미사가 추가됩니다.

Canary 릴리스는 단계 캐시(활성화한 경우)를 사용하여 응답을 저장하고, 사전 구성된 TTL(time-to-live) 기간 내에 캐시된 항목을 사용하여 다음 번 Canary 요청에 결과를 반환합니다.

Canary 릴리스 배포에서 API 트래픽의 프로덕션 릴리스와 Canary 릴리스는 같은 버전이나 서로 다른 버전에 연결할 수 있습니다. 프로덕션 릴리스와 Canary 릴리스가 서로 다른 버전에 연결된 경우, 프로덕션 요청의 응답과 Canary 요청의 응답은 별도로 캐시되며, 단계 캐시는 프로덕션 요청과 Canary 요청에 해당하는 결과를 반환합니다. 프로덕션 릴리스와 Canary 릴리스가 같은 배포에 연결된 경우, 단계 캐시는 두 가지 요청 유형에 단일 캐시 키를 사용하여 프로덕션 릴리스와 Canary 릴리스의 동일한 요청에 대해 동일한 응답을 반환합니다.

Canary 릴리스 배포 생성

API를 배포할 때 Canary 설정을 배포 생성 작업에 대한 추가 입력으로 사용하여 Canary 릴리스 배포를 생성합니다.

`stage:update`를 요청하여 단계에서 Canary 설정을 추가하는 방법으로 기존의 비 Canary 배포에서 Canary 릴리스 배포를 생성할 수도 있습니다.

비Canary 릴리스 배포를 생성할 때 기존 단계 이름이 아닌 이름을 지정할 수 있습니다. 지정한 단계가 없는 경우, API Gateway가 단계를 생성합니다. 하지만 Canary 릴리스 배포를 생성할 때는 기존 단계 이름이 아닌 이름을 지정할 수 없습니다. 그러면 오류가 발생하고 API Gateway가 Canary 릴리스 배포를 생성하지 않습니다.

API Gateway 콘솔, AWS CLI 또는 AWS SDK를 사용하여 API Gateway에서 Canary 릴리스 배포를 생성할 수 있습니다.

주제

- API Gateway 콘솔을 사용하여 Canary 배포 생성
- AWS CLI를 사용하여 Canary 배포 생성

API Gateway 콘솔을 사용하여 Canary 배포 생성

API Gateway 콘솔을 사용하여 Canary 릴리스 배포를 생성하려면 아래 지침에 따릅니다.

초기 Canary 릴리스 배포를 생성하려면

1. API Gateway 콘솔에 로그인합니다.
2. 기존 REST API를 선택하거나 새 REST API를 생성합니다.
3. 기본 탐색 창에서 리소스를 선택한 다음 API 배포를 선택합니다. API 배포I에서 화면 지침에 따라 새 단계에 API를 배포합니다.

   지금까지 프로덕션 릴리스 단계에 API를 배포했습니다. 다음으로 단계에서 Canary 설정을 구성하고, 필요하다면 캐싱을 활성화하거나 단계 변수를 설정하거나 API 실행 로그 또는 액세스 로그를 구성합니다.
4. API 캐싱을 활성화하거나 AWS WAF 웹 ACL을 스테이지에 연결하려면 스테이지 세부 정보 섹션에서 편집을 선택합니다. 자세한 내용은 the section called “캐시 설정” 또는 the section called “API Gateway 콘솔을 사용하여 API Gateway API 스테이지와 AWS WAF 리전 웹 ACL을 연결하는 방법” 단원을 참조하세요.
5. 실행 로깅 또는 액세스 로깅을 구성하려면 로그 및 추적 섹션에서 편집을 선택하고 화면 지침에 따릅니다. 자세한 내용은 API Gateway에서 REST API에 대한 CloudWatch 로깅 설정 섹션을 참조하세요.

6. 스테이지 변수를 설정하려면 스테이지 변수 탭을 선택하고 화면 지침에 따라 스테이지 변수를 추가하거나 수정합니다. 자세한 내용은 the section called “스테이지 변수 설정” 섹션을 참조하세요.
7. Canary 탭을 선택한 후 Canary 생성을 선택합니다. Canary 탭을 표시하려면 오른쪽 화살표 버튼을 선택해야 할 수도 있습니다.
8. Canary 설정에서 Canary에 Canary로 전환할 요청의 비율을 입력합니다.
9. 필요한 경우 스테이지 캐시를 선택하여 Canary 릴리스에 대한 캐싱을 활성화합니다. API 캐싱이 활성화될 때까지는 Canary 릴리스에 캐시를 사용할 수 없습니다.
10. 기존 스테이지 변수를 재정의하려면 Canary 재정의에 새 스테이지 변수 값을 입력합니다.

배포 단계에서 Canary 릴리스가 초기화된 후 API를 변경하고 변경을 테스트하는 것이 좋습니다. 업데이트된 버전과 기본 버전에 동일한 단계를 통해 액세스할 수 있도록 API를 같은 단계에 다시 배포할 수 있습니다. 다음 단계에서 그 방법을 설명합니다.

Canary에 최신 API 버전을 배포하려면

1. API를 업데이트할 때마다 API 배포를 선택합니다.
2. API 배포에서 Canary가 포함된 스테이지를 배포 스테이지 드롭다운 목록에서 선택합니다.
3. (선택 사항) 배포 설명에 설명을 입력합니다.
4. 최신 API 버전을 Canary 릴리스에 푸시하려면 배포(Deploy)를 선택합니다.
5. 원한다면 초기 Canary 릴리스 배포를 생성하려면의 설명대로 단계 설정, 로그 또는 Canary 설정을 다시 구성하세요.

결과적으로 Canary 릴리스는 최신 버전을 가리키고, 프로덕션 릴리스는 여전히 API 초기 버전을 가리키게 됩니다. canarySettings에는 이제 새 deploymentId 값이 있는 반면, 단계에는 여전히 초기 deploymentId 값이 있습니다. 백그라운드에서 콘솔이 stage:update를 호출합니다.

AWS CLI를 사용하여 Canary 배포 생성

먼저 Canary 없이 2개의 단계 변수로 기준 배포를 생성합니다.

```
aws apigateway create-deployment \
    --variables sv0=val0,sv1=val1 \
    --rest-api-id abcd1234 \
    --stage-name 'prod' \
```

명령은 결과로 얻은 다음과 비슷한 `Deployment` 표현을 반환합니다.

```
{
    "id": "du4ot1",
    "createdDate": 1511379050
}
```

이렇게 얻은 배포 id는 API의 스냅샷(또는 버전)을 식별합니다.

이제 prod 단계에서 Canary 배포를 생성합니다.

```
aws apigateway create-deployment  --rest-api-id abcd1234 \
    --canary-settings \
    '{
        "percentTraffic":10.5,
        "useStageCache":false,
        "stageVariableOverrides":{
            "sv1":"val2",
            "sv2":"val3"
        }
    }' \
    --stage-name 'prod'
```

지정된 단계(prod)가 존재하지 않으면 앞의 명령은 오류를 반환합니다. 그렇지 않으면 새로 생성된 다음과 비슷한 배포 리소스 표현을 반환합니다.

```
{
    "id": "a6rox0",
    "createdDate": 1511379433
}
```

이렇게 얻은 배포 id는 Canary 릴리스의 API 테스트 버전을 식별합니다. 그 결과, 연결된 단계는 Canary가 활성화됩니다. 다음과 비슷한 get-stage 명령을 호출하여 이 단계 표현을 볼 수 있습니다.

```
aws apigateway get-stage --rest-api-id acbd1234 --stage-name prod
```

다음은 Stage의 표현을 명령의 출력으로 보여 줍니다.

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

이 예에서 API 기본 버전은 단계 변수 `{"sv0":val0", "sv1":val1"}`을 사용하며, 테스트 버전은 단계 변수 `{"sv1":val2", "sv2":val3"}`를 사용합니다. 프로덕션 릴리스와 Canary 릴리스는 모두 동일한 단계 변수 `sv1`을 사용하지만 값은 서로 달라서 각각 `val1`과 `val2`입니다. 단계 변수 `sv0`은 프로덕션 릴리스에서만 사용되며, 단계 변수 `sv2`는 Canary 릴리스에서만 사용됩니다.

Canary를 활성화하도록 단계를 업데이트하여 기존 일반 배포에서 Canary 릴리스 배포를 생성할 수 있습니다. 이를 보여 주려면 먼저 일반 배포를 생성하세요.

```
aws apigateway create-deployment \
    --variables sv0=val0,sv1=val1 \
    --rest-api-id abcd1234 \
    --stage-name 'beta'
```

이 명령은 기본 버전 배포의 표현을 반환합니다.

```
{
    "id": "cifeiw",
    "createdDate": 1511380879
}
```

연결된 베타 단계에는 Canary 설정이 없습니다.

```
{
```

```
    "stageName": "beta",
    "variables": {
        "sv0": "val0",
        "sv1": "val1"
    },
    "cacheClusterEnabled": false,
    "cacheClusterStatus": "NOT_AVAILABLE",
    "deploymentId": "cifeiw",
    "lastUpdatedDate": 1511380879,
    "createdDate": 1511380879,
    "methodSettings": {}
}
```

이제 단계에서 Canary를 연결하여 새 Canary 릴리스 배포를 생성합니다.

```
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"
        }]'
```

업데이트된 단계의 표현은 다음과 같습니다.

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

기존 API 버전에서 방금 Canary를 활성화했기 때문에 프로덕션 릴리스(`Stage`)와 Canary 릴리스(`canarySettings`)는 모두 동일한 배포, 즉 API의 동일 버전(`deploymentId`)을 가리킵니다. API를 변경하여 이 단계에 다시 배포한 후 새 버전은 Canary 릴리스에 있게 되고, 기본 버전은 프로덕션 릴리스에 남게 됩니다. 이것은 Canary 릴리스의 `deploymentId`가 새 배포 `id`로 업데이트되고 프로덕션 릴리스의 `deploymentId`가 변경되지 않고 남아 있는 단계 변화에서 확인할 수 있습니다.

Canary 릴리스 업데이트

Canary 릴리스가 배포된 후 테스트 성능을 최적화하기 위해 Canary 트래픽 백분율을 조정하거나 단계 캐시 사용을 활성화 또는 비활성화할 수 있습니다. 실행 컨텍스트가 업데이트될 때 Canary 릴리스에서 사용되는 단계 변수를 수정할 수도 있습니다. 이러한 업데이트를 수행하려면 CanarySettings에서 새 값을 사용하여 stage:update 작업을 호출합니다.

API Gateway 콘솔, AWS CLI update-stage 명령 또는 AWS SDK를 사용하여 Canary 릴리스를 업데이트할 수 있습니다.

주제

- API Gateway 콘솔을 사용하여 Canary 릴리스 업데이트
- AWS CLI를 사용하여 Canary 릴리스 업데이트

API Gateway 콘솔을 사용하여 Canary 릴리스 업데이트

API Gateway 콘솔을 사용하여 단계에서 기존의 Canary 설정을 업데이트하려면 다음을 수행합니다.

기존 Canary 설정을 업데이트하려면

1. API Gateway 콘솔에 로그인하고 기존 REST API를 선택합니다.

2. 기본 탐색 창에서 스테이지를 선택한 후 기존 스테이지를 선택합니다.
3. Canary 탭을 선택한 후 편집을 선택합니다. Canary 탭을 표시하려면 오른쪽 화살표 버튼을 선택해야 할 수도 있습니다.
4. 0.0부터 100.0 사이의 백분율 숫자(0.0과 100.0 포함)를 늘리거나 줄여 요청 배포를 업데이트합니다.
5. 스테이지 캐시 확인란을 선택하거나 선택 취소합니다.
6. Canary 스테이지 변수를 추가, 제거 또는 수정합니다.
7. 저장을 선택합니다.

AWS CLI를 사용하여 Canary 릴리스 업데이트

AWS CLI를 사용하여 Canary를 업데이트하려면 `update-stage` 명령을 호출합니다.

Canary에 대한 단계 캐시 사용을 활성화하거나 비활성화하려면 다음과 같이 `update-stage` 명령을 호출합니다.

```
aws apigateway update-stage \
    --rest-api-id {rest-api-id} \
    --stage-name '{stage-name}'  \
    --patch-operations op=replace,path=/canarySettings/useStageCache,value=true
```

Canary 트래픽 백분율을 조정하려면 `update-stage`를 호출하여 단계에서 `/canarySettings/percentTraffic` 값을 바꿉니다.

```
aws apigateway update-stage \
    --rest-api-id {rest-api-id} \
    --stage-name '{stage-name}' \
    --patch-operations op=replace,path=/canarySettings/percentTraffic,value=25.0
```

Canary 단계 변수 추가, 교체 또는 제거를 포함하여 Canary 단계 변수를 업데이트하려면

```
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"
      }]'
```

작업들을 단일 patch-operations 값으로 결합하면 위의 모두를 업데이트할 수 있습니다.

```
aws apigateway update-stage  \
    --rest-api-id {rest-api-id} \
    --stage-name '{stage-name}' \
    --patch-operations '[{
        "op": "replace",
        "path": "/canarySettings/percentTraffic",
        "value": "20.0"
    }, {
        "op": "replace",
        "path": "/canarySettings/useStageCache",
        "value": "true"
    }, {
        "op": "remove",
        "path": "/canarySettings/stageVariableOverrides/var1"
    }, {
        "op": "replace",
        "path": "/canarySettings/stageVariableOverrides/newVar",
        "value": "newVal"
    }, {
        "op": "replace",
        "path": "/canarySettings/stageVariableOverrides/val2",
        "value": "val4"
      }]'
```

Canary 릴리스 승격

Canary 릴리스를 승격하면 테스트 중인 API 버전을 프로덕션 단계에서 사용할 수 있습니다. 이 작업에는 다음이 포함됩니다.

- Canary의 배포 ID 설정으로 단계의 배포 ID를 재설정합니다. 이렇게 하면 Canary 스냅샷으로 단계의 API 스냅샷이 업데이트되어 테스트 버전도 프로덕션 릴리스가 됩니다.

- Canary 단계 변수가 있다면 사용하여 단계 변수를 업데이트합니다. 이렇게 하면 Canary의 API 실행 컨텍스트로 단계의 API 실행 컨텍스트가 업데이트됩니다. 이 업데이트가 없으면 테스트 버전이 다른 단계 변수를 사용하거나 기존 단계 변수의 다른 값을 사용하는 경우, 새 API 버전이 예기치 못한 결과를 산출할 수 있습니다.
- Canary 트래픽 백분율을 0.0%로 설정합니다.

Canary 릴리스 승격은 단계에서 Canary를 비활성화하지 않습니다. Canary를 비활성화하려면 단계에서 Canary 설정을 제거해야 합니다.

주제

- API Gateway 콘솔을 사용하여 Canary 릴리스 승격
- AWS CLI를 사용하여 Canary 릴리스 승격

API Gateway 콘솔을 사용하여 Canary 릴리스 승격

API Gateway 콘솔을 사용하여 Canary 릴리스 배포를 승격하려면 다음을 수행합니다.

Canary 릴리스 배포를 승격하려면

1. API Gateway 콘솔에 로그인하고 기본 탐색 창에서 기존 API를 선택합니다.
2. 기본 탐색 창에서 스테이지를 선택한 후 기존 스테이지를 선택합니다.
3. Canary 탭을 선택합니다.
4. Canary 승격을 선택합니다.
5. 수행할 변경을 확인하고 Canary 승격을 선택합니다.

승격 후 프로덕션 릴리스는 Canary 릴리스와 동일한 API 버전(deploymentId)을 참조합니다. 이것은 AWS CLI를 사용하여 확인할 수 있습니다. 예제는 the section called “AWS CLI를 사용하여 Canary 릴리스 승격” 단원을 참조하세요.

AWS CLI를 사용하여 Canary 릴리스 승격

AWS CLI 명령을 사용하여 Canary 릴리스를 프로덕션 릴리스로 승격하려면 `update-stage` 명령을 호출하여 Canary에 연결된 `deploymentId`를 단계에 연결된 `deploymentId`로 복사하고, Canary 트래픽 백분율을 0(`0.0`)으로 재설정하고, Canary에 바인딩된 모든 단계 변수를 해당 단계에 바인딩된 단계 변수로 복사합니다.

다음과 비슷한 단계로 설명되는 Canary 릴리스 배포가 있다고 가정합니다.

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

다음 update-stage 요청을 호출하여 승격합니다.

```
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"
      }]'
```

승격이 끝난 후의 단계는 다음과 같이 됩니다.

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

위와 같이, Canary 릴리스를 단계로 승격해도 Canary는 비활성화되지 않으며 배포는 Canary 릴리스 배포로 남아 있게 됩니다. 일반 프로덕션 릴리스 배포를 수행하려면 Canary 설정을 비활성화해야 합니

다. Canary 릴리스 배포를 비활성화하는 방법에 대한 자세한 내용은 the section called “Canary 릴리스 비활성화” 단원을 참조하세요.

Canary 릴리스 비활성화

Canary 릴리스 배포를 비활성화하려면 `canarySettings`를 null로 설정하여 스테이지에서 제거합니다.

API Gateway 콘솔, AWS CLI 또는 AWS SDK를 사용하여 Canary 릴리스 배포를 비활성화할 수 있습니다.

주제

- API Gateway 콘솔을 사용하여 Canary 릴리스 비활성화
- AWS CLI를 사용하여 Canary 릴리스 비활성화

API Gateway 콘솔을 사용하여 Canary 릴리스 비활성화

API Gateway 콘솔을 사용하여 Canary 릴리스 배포를 비활성화하려면 다음 단계를 사용합니다.

Canary 릴리스 배포를 비활성화하려면

1. API Gateway 콘솔에 로그인하고 기본 탐색 창에서 기존 API를 선택합니다.
2. 기본 탐색 창에서 스테이지를 선택한 후 기존 스테이지를 선택합니다.
3. Canary 탭을 선택합니다.
4. Delete을 선택합니다.
5. 삭제를 선택하여 Canary 삭제를 확인합니다.

그 결과, `canarySettings` 속성이 `null`이 되고 배포 단계에서 제거됩니다. 이것은 AWS CLI를 사용하여 확인할 수 있습니다. 예제는 the section called “AWS CLI를 사용하여 Canary 릴리스 비활성화” 단원을 참조하세요.

AWS CLI를 사용하여 Canary 릴리스 비활성화

AWS CLI를 사용하여 Canary 릴리스 배포를 비활성화하려면 다음과 같이 `update-stage` 명령을 호출합니다.

```
aws apigateway update-stage \
    --rest-api-id abcd1234 \
```

```
    --stage-name canary \
    --patch-operations '[{"op":"remove", "path":"/canarySettings"}]'
```

성공적인 응답은 다음과 비슷한 페이로드를 반환합니다.

```
{
    "stageName": "prod",
    "accessLogSettings": {
        ...
    },
    "cacheClusterEnabled": false,
    "cacheClusterStatus": "NOT_AVAILABLE",
    "deploymentId": "nfcn0x",
    "lastUpdatedDate": 1511309280,
    "createdDate": 1511152939,
    "methodSettings": {
        ...
    }
}
```

출력에 표시된 것과 같이, `canarySettings` 속성은 Canary가 비활성화된 배포의 단계에 더 이상 나타나지 않습니다.

재배포가 필요한 REST API 업데이트

API를 유지하는 것은 기존 API 설정을 확인, 업데이트 및 삭제하는 것에 해당합니다. API Gateway 콘솔, AWS CLI, SDK 또는 API Gateway REST API를 사용하여 API를 유지 관리할 수 있습니다. API 업데이트에는 API의 일부 리소스 속성 또는 구성 설정의 수정이 포함됩니다. 리소스 업데이트에서는 API를 재배포해야 하지만, 구성 업데이트에서는 그렇지 않습니다.

업데이트할 수 있는 API 리소스는 다음 표에 자세히 나와 있습니다.

API 재배포가 필요한 API 리소스 업데이트

리소스	설명
ApiKey	적용 가능한 속성 및 지원되는 작업은 apikey:update를 참조하세요. 업데이트 시 API를 재배포해야 합니다.
권한 부여자	적용 가능한 속성 및 지원되는 작업은 authorizer:update를 참조하세요. 업데이트 시 API를 재배포해야 합니다.

리소스	설명
DocumentationPart	적용 가능한 속성 및 지원되는 작업은 documentationpart:update를 참조하세요. 업데이트 시 API를 재배포해야 합니다.
DocumentationVersion	적용 가능한 속성 및 지원되는 작업은 documentationversion:update를 참조하세요. 업데이트 시 API를 재배포해야 합니다.
GatewayResponse	적용 가능한 속성 및 지원되는 작업은 gatewayresponse:update를 참조하세요. 업데이트 시 API를 재배포해야 합니다.
통합	적용 가능한 속성 및 지원되는 작업은 integration:update를 참조하세요. 업데이트 시 API를 재배포해야 합니다.
IntegrationResponse	적용 가능한 속성 및 지원되는 작업은 integrationresponse:update를 참조하세요. 업데이트 시 API를 재배포해야 합니다.
방법	적용 가능한 속성 및 지원되는 작업은 method:update를 참조하세요. 업데이트 시 API를 재배포해야 합니다.
MethodResponse	적용 가능한 속성 및 지원되는 작업은 methodresponse:update를 참조하세요. 업데이트 시 API를 재배포해야 합니다.
모델	적용 가능한 속성 및 지원되는 작업은 model:update를 참조하세요. 업데이트 시 API를 재배포해야 합니다.
RequestValidator	적용 가능한 속성 및 지원되는 작업은 requestvalidator:update를 참조하세요. 업데이트 시 API를 재배포해야 합니다.
리소스	적용 가능한 속성 및 지원되는 작업은 resource:update를 참조하세요. 업데이트 시 API를 재배포해야 합니다.
RestApi	적용 가능한 속성 및 지원되는 작업은 restapi:update를 참조하세요. 업데이트 시 API를 재배포해야 합니다.
VpcLink	적용 가능한 속성 및 지원되는 작업은 vpclink:update를 참조하세요. 업데이트 시 API를 재배포해야 합니다.

업데이트할 수 있는 API 구성은 다음 표에 자세히 나와 있습니다.

API 재배포가 불필요한 API 구성 업데이트

Configuration	설명
계정	적용 가능한 속성 및 지원되는 작업은 account:update를 참조하세요. 업데이트 시 API를 재배포할 필요가 없습니다.
배포	적용 가능한 속성 및 지원되는 작업은 deployment:update를 참조하세요.
DomainName	적용 가능한 속성 및 지원되는 작업은 domainname:update를 참조하세요. 업데이트 시 API를 재배포할 필요가 없습니다.
BasePathMapping	적용 가능한 속성 및 지원되는 작업은 basepathmapping:update를 참조하세요. 업데이트 시 API를 재배포할 필요가 없습니다.
단계	적용 가능한 속성 및 지원되는 작업은 stage:update를 참조하세요. 업데이트 시 API를 재배포할 필요가 없습니다.
사용량	적용 가능한 속성 및 지원되는 작업은 usage:update를 참조하세요. 업데이트 시 API를 재배포할 필요가 없습니다.
UsagePlan	적용 가능한 속성 및 지원되는 작업은 usageplan:update를 참조하세요. 업데이트 시 API를 재배포할 필요가 없습니다.

REST API에 대한 사용자 지정 도메인 이름 설정

사용자 지정 도메인 이름은 API 사용자에게 제공할 수 있는 더 간단하고 직관적인 URL입니다.

API를 배포한 후 사용자 및 사용자 고객은 다음 형식의 기본 URL을 사용하여 API를 호출할 수 있습니다.

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

여기서 `api-id`는 API Gateway에서 생성되고, `region`(AWS 리전)은 API를 생성할 때 사용자가 지정하며, `stage`는 API를 배포할 때 사용자가 지정합니다.

URL의 호스트 이름 부분(즉, *`api-id`*`.execute-api.`*`region`*`.amazonaws.com`)은 API 엔드포인트를 가리킵니다. 기본 API 엔드포인트는 기억하기가 어려우며 사용자에게 친숙하지 않을 수 있습니다.

사용자 지정 도메인 이름을 사용하면 API의 호스트 이름을 설정하고 기본 경로(예: `myservice`)를 선택하여 대체 URL을 API에 매핑할 수 있습니다. 예를 들어, 더 사용자 친화적인 API 기본 URL은 다음과 같습니다.

```
https://api.example.com/myservice
```

> **Note**
>
> 리전 사용자 지정 도메인은 REST API 및 HTTP API와 연결될 수 있습니다. API Gateway 버전 2 API를 사용하여 REST API에 대한 리전 사용자 지정 도메인 이름을 생성하고 관리할 수 있습니다.
> 사용자 지정 도메인 이름은 프라이빗 API에 사용할 수 없습니다.
> 해당 REST API에서 지원하는 최소 TLS 버전을 선택할 수 있습니다. REST API의 경우 TLS 1.2 또는 TLS 1.0을 선택할 수 있습니다.

도메인 이름 등록

API에 대한 사용자 지정 도메인 이름을 설정하려면 등록된 인터넷 도메인 이름이 있어야 합니다. 필요한 경우 Amazon Route 53를 사용하거나 사용자가 선택한 서드 파티 도메인 등록자를 사용하여 인터넷 도메인을 등록할 수 있습니다. API의 사용자 지정 도메인 이름은 하위 도메인의 이름이거나 등록된 인터넷 도메인의 루트 도메인("zone apex"라고도 함) 이름일 수 있습니다.

API Gateway에서 사용자 지정 도메인 이름을 생성한 후에는 DNS 공급자의 리소스 레코드를 생성하거나 업데이트하여 API 엔드포인트에 매핑해야 합니다. 이렇게 매핑하지 않으면 사용자 지정 도메인 이름이 목적지인 API 요청은 API Gateway에 도달할 수 없습니다.

> **Note**
>
> 엣지에 최적화된 사용자 지정 도메인 이름은 특정 리전에서 생성되고 특정 AWS 계정이 소유합니다. 리전 또는 AWS 계정 간에 이러한 사용자 지정 도메인 이름을 이동하려면 기존 CloudFront 배포를 삭제하고 새 배포를 생성해야 합니다. 새 사용자 지정 도메인 이름을 사용할 수 있게 될 때까지 프로세스는 약 30분 정도 걸릴 수 있습니다. 자세한 내용은 CloudFront 배포 업데이트를 참조하세요.

에지 최적화 사용자 지정 도메인 이름

엣지 최적화 API를 배포하면 API Gateway는 Amazon CloudFront 배포와 DNS 레코드를 설정하여 API 도메인 이름을 CloudFront 배포 도메인 이름에 매핑합니다. 그러면 API에 대한 요청은 매핑된 CloudFront 배포를 통해 API Gateway로 라우팅됩니다.

엣지 최적화된 API에 대한 사용자 지정 도메인 이름을 생성하면 API Gateway는 CloudFront 배포를 설정합니다. 하지만 사용자 지정 도메인 이름을 CloudFront 배포 도메인 이름에 매핑하려면 DNS 레코드를 설정해야 합니다. 이 매핑은 매핑된 CloudFront 배포를 통해 API Gateway에 라우팅될 사용자 지정 도메인 이름에 바인딩된 API 요청에 사용됩니다. 또한 사용자 지정 도메인 이름에 대한 인증서를 제출해야 합니다.

> **Note**
>
> API Gateway가 생성한 CloudFront 배포는 API Gateway와 연계된 리전별 계정이 소유합니다. CloudWatch Logs에서 이러한 CloudFront 배포를 생성하고 업데이트하는 작업을 추적할 때는 이 API Gateway 계정 ID를 사용해야 합니다. 자세한 내용은 CloudTrail에서 사용자 지정 도메인 이름 생성 로그 단원을 참조하세요.

엣지 최적화 사용자 지정 도메인 이름을 설정하거나 해당 인증서를 업데이트하려면 CloudFront 배포를 업데이트할 수 있는 권한이 있어야 합니다.

액세스 권한을 제공하려면 사용자, 그룹 또는 역할에 권한을 추가합니다.

- AWS IAM Identity Center의 사용자 및 그룹:

  권한 세트를 생성합니다. AWS IAM Identity Center 사용 설명서의 권한 세트 생성의 지침을 따르세요.
- 자격 증명 공급자를 통해 IAM에서 관리되는 사용자:

  아이덴티티 페더레이션을 위한 역할을 생성합니다. IAM 사용 설명서의 서드 파티 자격 증명 공급자의 역할 만들기(연합)의 지침을 따르세요.
- IAM 사용자:
  - 사용자가 맡을 수 있는 역할을 생성합니다. IAM 사용 설명서에서 IAM 사용자의 역할 생성의 지침을 따르세요.
  - (권장되지 않음) 정책을 사용자에게 직접 연결하거나 사용자를 사용자 그룹에 추가합니다. IAM 사용 설명서에서 사용자(콘솔)에 권한 추가의 지침을 따르세요.

CloudFront 배포를 업데이트하려면 다음 권한이 필요합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
         {
            "Sid": "AllowCloudFrontUpdateDistribution",
            "Effect": "Allow",
            "Action": [
                "cloudfront:updateDistribution"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

API Gateway는 CloudFront 배포에서 SNI(Server Name Indication)를 활용하여 엣지 최적화 사용자 지정 도메인 이름을 지원합니다. 필요한 인증서 형식 및 인증서 키 길이의 최대 크기를 포함하여 CloudFront 배포에서 사용자 지정 도메인 이름을 사용하는 방법에 대한 자세한 내용은 Amazon CloudFront 개발자 안내서의 대체 도메인 이름 및 HTTPS 사용을 참조하세요.

사용자 지정 도메인 이름을 API의 호스트 이름으로 설정하려면 API 소유자인 사용자가 사용자 지정 도메인 이름에 대한 SSL/TLS 인증서를 제공해야 합니다.

엣지에 최적화된 사용자 지정 도메인 이름에 대한 인증서를 제공하기 위해 ACM에서 새 인증서를 생성하거나 `us-east-1` 리전(미국 동부(북부 버지니아))의 타사 인증 기관에서 발행한 인증서를 ACM으로 가져오도록 AWS Certificate Manager(ACM)에 요청할 수 있습니다.

리전 사용자 지정 도메인 이름

리전 API에 대해 사용자 지정 도메인 이름을 생성하면 API Gateway는 해당 API에 대해 리전 도메인 이름을 생성합니다. DNS 레코드를 설정하여 사용자 지정 도메인 이름이 리전 도메인 이름을 가리키도록 해야 합니다. 또한 사용자 지정 도메인 이름에 대한 인증서를 제출해야 합니다.

와일드카드 사용자 정의 도메인 이름

와일드카드 사용자 지정 도메인 이름을 사용하면 기본 할당량을 초과하지 않고 거의 무제한 수의 도메인 이름을 지원할 수 있습니다. 예를 들어 각 고객에게 고유한 도메인 이름인 *customername*`.api.example.com`을(를) 제공할 수 있습니다.

와일드카드 사용자 지정 도메인 이름을 생성하려면, 와일드카드(*)를 루트 도메인의 가능한 모든 하위 도메인을 나타내는 사용자 지정 도메인의 첫 번째 하위 도메인으로 지정할 수 있습니다.

예를 들어 와일드카드 사용자 정의 도메인 이름 `*.example.com`을 사용하면 `a.example.com`, `b.example.com` 및 `c.example.com` 같은 하위 도메인이 모두 동일한 도메인으로 라우팅됩니다.

와일드카드 사용자 지정 도메인 이름은 API Gateway의 표준 사용자 지정 도메인 이름과 별개의 구성을 지원합니다. 예를 들어, 단일 AWS 계정에서 `*.example.com`과 `a.example.com`가 다르게 동작하도록 구성할 수 있습니다.

`$context.domainName` 및 `$context.domainPrefix` 컨텍스트 변수를 사용하여 클라이언트가 API를 호출하는 데 사용한 도메인 이름을 확인할 수 있습니다. 컨텍스트 변수에 대해 자세히 알아보려면 API Gateway 매핑 템플릿과 액세스 로깅 변수 참조 단원을 참조하십시오.

와일드카드 사용자 지정 도메인 이름을 생성하려면 DNS 또는 이메일 검증 방법을 사용하여 검증한 ACM에서 발급한 인증서를 제공해야 합니다.

> **Note**
>
> 다른 AWS 계정에서 와일드카드 사용자 지정 도메인 이름과 충돌하는 사용자 지정 도메인 이름을 만든 경우에는 와일드카드 사용자 지정 도메인 이름을 생성할 수 없습니다. 예를 들어 계정 A에서 `a.example.com`가 생성된 경우, 계정 B는 와일드카드 사용자 지정 도메인 이름 `*.example.com`을 생성할 수 없습니다.
> 계정 A와 계정 B가 한 소유자를 공유하는 경우 AWS 지원 센터에 문의하여 예외를 요청할 수 있습니다.

사용자 지정 도메인 이름에 대한 인증서

> **Important**
>
> 사용자 지정 도메인 이름에 대한 인증서를 지정합니다. 애플리케이션에서 ACM 인증서를 고정하기 위해 인증서 고정(SSL 고정)을 사용하는 경우, AWS가 인증서를 갱신한 후 애플리케이션이 도메인에 연결하지 못하게 될 수 있습니다. 자세한 내용은 AWS Certificate Manager 사용 설명서의 인증서 고정 문제를 참조하세요.

ACM을 지원하는 리전에서 사용자 지정 도메인 이름에 대한 인증서를 제공하려면 ACM에 인증서를 요청해야 합니다. ACM을 지원하지 않는 리전에서 리전 사용자 지정 도메인 이름에 대한 인증서를 제공하려면 해당 리전의 API Gateway로 인증서를 가져와야 합니다.

SSL/TLS 인증서를 가져오려면 PEM 형식의 SSL/TLS 인증서 본문, 해당하는 프라이빗 키 및 사용자 지정 도메인 이름에 대한 인증서 체인을 제공해야 합니다. ACM에 저장된 각 인증서는 ARN으로 식별됩니다. 해당 ARN을 참조하기만 하면 도메인 이름에 대해 AWS에서 관리하는 인증서를 사용할 수 있습니다.

ACM을 사용하면 API에 대한 사용자 지정 도메인 이름을 간편하게 설정하고 사용할 수 있습니다. 지정된 도메인 이름에 대한 인증서를 생성하거나 인증서를 가져오고, API Gateway에서 ACM이 제공한 인증서의 ARN을 사용하여 도메인 이름을 설정한 다음, 사용자 지정 도메인 이름 아래의 기본 경로를 API의 배포된 단계에 매핑합니다. ACM에서 발행한 인증서를 사용하면 프라이빗 키 같은 민감한 인증서 세부 정보의 공개를 걱정할 필요가 없습니다.

주제

- AWS Certificate Manager에서 인증서 준비하기
- API Gateway에서 사용자 지정 도메인에 대한 보안 정책 선택
- 에지 최적화 사용자 지정 도메인 이름 생성
- API Gateway에서 리전 사용자 지정 도메인 이름 설정
- 사용자 지정 도메인 이름을 다른 API 엔드포인트로 마이그레이션
- REST API에 대한 API 매핑 작업
- REST API에 대한 기본 엔드포인트 비활성화
- DNS 장애 조치에 대한 사용자 지정 상태 확인 구성

AWS Certificate Manager에서 인증서 준비하기

API에 대한 사용자 지정 도메인 이름을 설정하기 전에 에서 SSL/TLS 인증서를 준비해야 합니다AWS Certificate Manager 다음 단계에서 이 방법을 설명합니다. 자세한 내용은 AWS Certificate Manager 사용 설명서를 참조하세요.

> **Note**
>
> API Gateway 엣지 최적화 사용자 지정 도메인 이름이 포함된 ACM 인증서를 사용하려면 미국 동부(버지니아 북부)(`us-east-1`) 리전에서 인증서를 요청하거나 가져와야 합니다. API Gateway 리전 사용자 지정 도메인 이름의 경우, API와 동일한 리전에서 인증서를 요청하거나

가져와야 합니다. 인증서는 공개적으로 신뢰할 수 있는 인증 기관에서 서명해야 하며 사용자 지정 도메인 이름을 포함해야 합니다.

먼저 인터넷 도메인(예: *example*.com)을 등록합니다. Amazon Route 53 또는 서드 파티 인증 도메인 등록자를 사용할 수 있습니다. 그러한 등록 대행자의 목록은 ICANN 웹 사이트에서 Accredited Registrar Directory를 참조하세요.

도메인 이름에 대한 SSL/TLS 인증서를 ACM에서 생성하거나 ACM으로 가져오려면 다음 중 하나를 수행합니다.

도메인 이름에 대해 ACM이 제공한 인증서를 요청하려면

1. AWS Certificate Manager 콘솔에 로그인합니다.
2. 인증서 요청을 선택합니다.
3. API에 대한 사용자 지정 도메인 이름(예: 도메인 이름에 api.example.com)을 입력합니다.
4. 필요에 따라 이 인증서에 다른 이름 추가를 선택할 수도 있습니다.
5. [Review and request]를 선택합니다.
6. [Confirm and request]를 선택합니다.
7. 요청이 유효하려면 ACM이 인증서를 발급하기 전에 인터넷 도메인의 등록 소유자가 요청에 동의해야 합니다.

도메인 이름에 대한 인증서를 ACM으로 가져오려면

1. 인증 기관으로부터 여러분의 사용자 지정 도메인 이름에 대한 PEM 인코딩 SSL/TLS 인증서를 취득합니다. CA 등 일부 목록은 Mozilla Included CA List를 참조하세요.

   a. OpenSSL 웹 사이트에 있는 OpenSSL 도구 키트를 사용하여 인증서의 프라이빗 키를 생성하고 파일에 생성된 결과물을 저장합니다.

   ```
   openssl genrsa -out private-key-file 2048
   ```

   > **Note**
   >
   > Amazon API Gateway는 Amazon CloudFront를 활용하여 사용자 지정 도메인 이름에 대한 인증서를 지원합니다. 그렇기 때문에 사용자 지정 도메인 이름의 SSL/TLS 인증서에 대한 요구 사항과 제약 조건이 CloudFront에서 지정됩니다. 예를 들어, 퍼블릭

키의 최대 크기는 2048이고 프라이빗 키 크기는 1024, 2048 및 4096입니다. 퍼블릭 키 크기는 이용하는 인증 기관에서 정합니다. 이용하는 인증 기관에 기본 설정 길이와 다른 크기의 키를 반환하도록 요청합니다. 자세한 내용은 객체에 대한 액세스 보안 및 서명된 URL 및 서명된 쿠키 생성을 참조하세요.

b. OpenSSL을 사용하여 이전에 생성된 프라이빗 키로 인증서 서명 요청(CSR) 생성:

```
openssl req -new -sha256 -key private-key-file -out CSR-file
```

c. CSR을 인증 기관에 제출하고 그 결과 얻은 인증서를 저장합니다.

d. 인증 기관으로부터 인증서 체인을 다운로드합니다.

> **Note**
>
> 다른 방법으로 프라이빗 키를 얻고 키가 암호화된 경우, 사용자 지정 도메인 이름을 설정하기 위해 키를 API Gateway에 제출하기 전에 다음 명령을 사용하여 키를 암호 해독할 수 있습니다.
>
> ```
> openssl pkcs8 -topk8 -inform pem -in MyEncryptedKey.pem -outform pem -
> nocrypt -out MyDecryptedKey.pem
> ```

2. 인증서를 AWS Certificate Manager로 업로드합니다.

a. AWS Certificate Manager 콘솔에 로그인합니다.

b. [Import a certificate]을 선택합니다.

c. 인증서 본문에서 인증 기관으로부터 받은 PEM 형식의 서버 인증서 본문을 입력하거나 복사하여 붙여 넣습니다. 다음은 그와 같은 인증서의 축약된 예입니다.

```
-----BEGIN CERTIFICATE-----
EXAMPLECA+KgAwIBAgIQJ1XxJ8Pl++gOfQtj0IBoqDANBgkqhkiG9w0BAQUFADBB
...
az8Cg1aicxLBQ7EaWIhhgEXAMPLE
-----END CERTIFICATE-----
```

d. 인증서 프라이빗 키에서 사용자가 사용하는 PEM 형식 인증서의 프라이빗 키를 입력하거나 복사하여 붙여 넣습니다. 다음은 그와 같은 키의 축약된 예입니다.

```
-----BEGIN RSA PRIVATE KEY-----
```

```
EXAMPLEBAAKCAQEA2Qb3LDHD7StY7Wj6U2/opV6Xu37qUCCkeDWhwpZMYJ9/nETO
...
1qGvJ3u04vdnzaYN5WoyN5LFckrlA71+CszD1CGSqbVDWEXAMPLE
-----END RSA PRIVATE KEY-----
```

e. 인증서 체인에서 PEM 형식의 중간 인증서와 루트 인증서(선택 사항)를 빈 줄 없이 서로 이어서 입력하거나 복사하여 붙여 넣습니다. 루트 인증서를 포함하는 경우 인증서 체인은 중간 인증서로 시작하고 루트 인증서를 끝나야 한다. 인증 기관에서 제공한 중간 인증서를 사용합니다. 신뢰 경로 체인에 없는 중간 인증서를 포함시키지 마세요. 다음은 축약된 예입니다.

```
-----BEGIN CERTIFICATE-----
EXAMPLECA4ugAwIBAgIQWrYdrB5NogYUx1U9Pamy3DANBgkqhkiG9w0BAQUFADCB
...
8/ifBlIK3se2e4/hEfcEejX/arxbx1BJCHBvlEPNnsdw8EXAMPLE
-----END CERTIFICATE-----
```

다음은 또 다른 예입니다.

```
-----BEGIN CERTIFICATE-----
Intermediate certificate 2
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
Intermediate certificate 1
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
Optional: Root certificate
-----END CERTIFICATE-----
```

f. [Review and import]를 선택합니다.

인증서를 성공적으로 생성하거나 가져온 후에 인증서 ARN을 기록해 둡니다. 사용자 지정 도메인 이름을 설정할 때 필요합니다.

API Gateway에서 사용자 지정 도메인에 대한 보안 정책 선택

Amazon API Gateway 사용자 지정 도메인의 보안을 강화하기 위해 API Gateway 콘솔 AWS CLI, 또는 AWS SDK에서 보안 정책을 선택할 수 있습니다.

보안 정책은 API Gateway에서 제공하는 최소 TLS 버전과 암호 제품군의 사전 정의된 조합입니다. TLS 버전 1.2 또는 TLS 버전 1.0 보안 정책 중 하나를 선택할 수 있습니다. TLS 프로토콜은 클라이언트와 서버 간의 변조 및 도청과 같은 네트워크 보안 문제를 해결합니다. 클라이언트가 사용자 지정의 도

메인을 통해 API에 TLS 핸드셰이크를 설정하면 보안 정책은 클라이언트가 선택할 수 있는 TLS 버전 및 암호 제품군 옵션을 적용합니다.

사용자 지정 도메인 설정에서, 보안 정책은 다음 두 가지 설정을 결정합니다.

- API Gateway가 API 클라이언트와 통신하는 데 사용하는 최소 TLS 버전
- API Gateway가 API 클라이언트에 반환하는 콘텐츠를 암호화하는 데 사용하는 암호

TLS 1.0 보안 정책을 선택하면 보안 정책은 TLS 1.0, TLS 1.2 및 TLS 1.3 트래픽을 허용합니다. TLS 1.2 보안 정책을 선택하면 보안 정책은 TLS 1.2 및 TLS 1.3 트래픽을 허용하고 TLS 1.0 트래픽은 거부합니다.

> **Note**
>
> 사용자 지정 도메인에 대한 보안 정책만 지정할 수 있습니다. 기본 엔드포인트를 사용하는 API의 경우 API Gateway는 다음 보안 정책을 사용합니다.
>
> - 엣지 최적화 API의 경우: `TLS-1-0`
> - 리전 API의 경우: `TLS-1-0`
> - 프라이빗 API의 경우: `TLS-1-2`

주제

- 커스텀 도메인의 보안 정책을 지정하는 방법
- 엣지에 최적화된 사용자 지정 도메인에 지원되는 보안 정책, TLS 프로토콜 버전 및 암호
- 지원되는 보안 정책, TLS 프로토콜 버전, 지역 사용자 지정 도메인에 대한 암호
- 프라이빗 API에 지원되는 TLS 프로토콜 버전 및 암호
- OpenSSL 및 RFC 암호 이름
- HTTP API 및 WebSocket API에 대한 정보

커스텀 도메인의 보안 정책을 지정하는 방법

사용자 지정 도메인 이름을 생성할 때 해당 도메인 이름에 대한 보안 정책을 지정합니다. 사용자 지정 도메인을 만드는 방법을 알아보려면 the section called “에지 최적화 사용자 지정 도메인 이름 생성” 또는 을 참조하십시오the section called “리전 사용자 지정 도메인 이름 설정”.

커스텀 도메인 이름의 보안 정책을 변경하려면 커스텀 도메인 설정을 업데이트하세요. AWS Management Console AWS CLI, 또는 AWS SDK를 사용하여 사용자 지정 도메인 이름 설정을 업데이트할 수 있습니다.

API Gateway REST API를 사용하는 AWS CLI경우 또는 `securityPolicy` 파라미터에 새 TLS 버전을 지정하십시오. `TLS_1_0` `TLS_1_2` 자세한 내용은 Amazon API Gateway REST API 레퍼런스 또는 update-domain-name레퍼런스의 도메인 이름 업데이트를 참조하십시오.AWS CLI

업데이트 작업을 완료하는 데 몇 분 정도 걸릴 수 있습니다.

엣지에 최적화된 사용자 지정 도메인에 지원되는 보안 정책, TLS 프로토콜 버전 및 암호

다음 표에는 엣지 최적화 사용자 지정 도메인 이름에 지정할 수 있는 보안 정책이 설명되어 있습니다.

보안 정책	TLS_1_0	TLS_1_2
TLS 프로토콜		
TLSv1.3	♦	♦
TLSv1.2	♦	♦
TLSv1.1	♦	
TLSv1	♦	
TLS 암호		
TLS_AES_128_GCM_SHA256	♦	♦
TLS_AES_256_GCM_SHA384	♦	♦
TLS_CHACHA20_POLY1305_SHA256	♦	♦
ECDHE-ECDSA-AES128-GCM-SHA256	♦	♦
ECDHE-ECDSA-AES128-SHA256	♦	♦
ECDHE-ECDSA-AES128-SHA	♦	

보안 정책	TLS_1_0	TLS_1_2
ECDHE-ECDSA-AES256-GCM-SHA384	♦	♦
ECDHE-ECDSA-CHACHA20-POLY1305	♦	♦
ECDHE-ECDSA-AES256-SHA384	♦	♦
ECDHE-ECDSA-AES256-SHA	♦	
ECDHE-RSA-AES128-GCM-SHA256	♦	♦
ECDHE-RSA-AES128-SHA256	♦	♦
ECDHE-RSA-AES128-SHA	♦	
ECDHE-RSA-AES256-GCM-SHA384	♦	♦
ECDHE-RSA-CHACHA20-POLY1305	♦	♦
ECDHE-RSA-AES256-SHA384	♦	♦
ECDHE-RSA-AES256-SHA	♦	
AES128-GCM-SHA256	♦	
AES256-GCM-SHA384	♦	♦
AES128-SHA256	♦	♦
AES256-SHA	♦	
AES128-SHA	♦	

보안 정책	TLS_1_0	TLS_1_2
DES-CBC3-SHA	♦	

지원되는 보안 정책, TLS 프로토콜 버전, 지역 사용자 지정 도메인에 대한 암호

다음 표에는 지역 사용자 지정 도메인 이름에 지정할 수 있는 보안 정책이 설명되어 있습니다.

보안 정책	TLS_1_0	TLS_1_2
TLS 프로토콜		
TLSv1.3	♦	♦
TLSv1.2	♦	♦
TLSv1.1	♦	
TLSv1	♦	
TLS 암호		
TLS_AES_128_GCM_SHA256	♦	♦
TLS_AES_256_GCM_SHA384	♦	♦
TLS_CHACHA20_POLY1305_SHA256	♦	♦
ECDHE-ECDSA-AES128-GCM-SHA256	♦	♦
ECDHE-RSA-AES128-GCM-SHA256	♦	♦
ECDHE-ECDSA-AES128-SHA256	♦	♦
ECDHE-RSA-AES128-SHA256	♦	♦

보안 정책	TLS_1_0	TLS_1_2
ECDHE-ECDSA-AES128-SHA	♦	
ECDHE-RSA-AES128-SHA	♦	
ECDHE-ECDSA-AES256-GCM-SHA384	♦	♦
ECDHE-RSA-AES256-GCM-SHA384	♦	♦
ECDHE-ECDSA-AES256-SHA384	♦	♦
ECDHE-RSA-AES256-SHA384	♦	♦
ECDHE-RSA-AES256-SHA	♦	
ECDHE-ECDSA-AES256-SHA	♦	
AES128-GCM-SHA256	♦	♦
AES128-SHA256	♦	♦
AES128-SHA	♦	
AES256-GCM-SHA384	♦	♦
AES256-SHA256	♦	♦
AES256-SHA	♦	

프라이빗 API에 지원되는 TLS 프로토콜 버전 및 암호

다음 표에는 프라이빗 API에 지원되는 TLS 프로토콜 및 암호가 설명되어 있습니다. 프라이빗 API에 대한 보안 정책 지정은 지원되지 않습니다.

보안 정책	TLS_1_2
TLS 프로토콜	
TLSv1.2	♦
TLS 암호	
ECDHE-ECDSA-AES128-GCM-SHA256	♦
ECDHE-RSA-AES128-GCM-SHA256	♦
ECDHE-ECDSA-AES128-SHA256	♦
ECDHE-RSA-AES128-SHA256	♦
ECDHE-ECDSA-AES256-GCM-SHA384	♦
ECDHE-RSA-AES256-GCM-SHA384	♦
ECDHE-ECDSA-AES256-SHA384	♦
ECDHE-RSA-AES256-SHA384	♦
AES128-GCM-SHA256	♦
AES128-SHA256	♦
AES256-GCM-SHA384	♦
AES256-SHA256	♦

OpenSSL 및 RFC 암호 이름

OpenSSL과 IETF RFC 5246은 동일한 암호에 대해 서로 다른 이름을 사용합니다. 다음 표에는 각 암호의 RFC 이름에 OpenSSL 이름이 매핑되어 있습니다.

OpenSSL 암호화 이름	RFC 암호화 이름
TLS_AES_128_GCM_SHA256	TLS_AES_128_GCM_SHA256
TLS_AES_256_GCM_SHA384	TLS_AES_256_GCM_SHA384
TLS_CHACHA20_POLY1305_SHA256	TLS_CHACHA20_POLY1305_SHA256
ECDHE-RSA-AES128-GCM-SHA256	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
ECDHE-RSA-AES128-SHA256	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
ECDHE-RSA-AES128-SHA	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
ECDHE-RSA-AES256-GCM-SHA384	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
ECDHE-RSA-AES256-SHA384	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384
ECDHE-RSA-AES256-SHA	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
AES128-GCM-SHA256	TLS_RSA_WITH_AES_128_GCM_SHA256

OpenSSL 암호화 이름	RFC 암호화 이름
AES256-GCM-SHA384	TLS_RSA_WITH_AES_256_GCM_SHA384
AES128-SHA256	TLS_RSA_WITH_AES_128_CBC_SHA256
AES256-SHA	TLS_RSA_WITH_AES_256_CBC_SHA
AES128-SHA	TLS_RSA_WITH_AES_128_CBC_SHA
DES-CBC3-SHA	TLS_RSA_WITH_3DES_EDE_CBC_SHA

HTTP API 및 WebSocket API에 대한 정보

HTTP API 및 WebSocket API에 대한 자세한 내용은 및 을 참조하십시오. the section called “HTTP API에 대한 보안 정책” the section called “ WebSocket API에 대한 보안 정책”

에지 최적화 사용자 지정 도메인 이름 생성

주제

- API Gateway API에 대한 엣지 최적화 사용자 지정 도메인 이름 설정
- CloudTrail에서 사용자 지정 도메인 이름 생성 로그
- 사용자 지정 도메인 이름을 호스트 이름으로 사용하여 API의 기본 경로 매핑 구성
- ACM에 가져온 인증서 교체
- 사용자 지정 도메인 이름을 사용하여 API 호출

API Gateway API에 대한 엣지 최적화 사용자 지정 도메인 이름 설정

다음 절차에서는 API Gateway 콘솔을 사용하여 API에 대한 사용자 지정 도메인 이름을 생성하는 방법을 설명합니다.

API Gateway 콘솔을 사용하여 사용자 지정 도메인 이름을 생성하려면

1. https://console.aws.amazon.com/apigateway에서 API Gateway 콘솔에 로그인합니다.
2. 기본 탐색 창에서 사용자 지정 도메인 이름을 선택합니다.

3. 생성을 선택합니다.
4. 도메인 이름에 도메인 이름을 입력합니다.
5. 구성에서 Edge-optimized(엣지 최적화)를 선택합니다.
6. 최소 TLS 버전을 선택합니다.
7. ACM 인증서를 선택합니다.

> **Note**
>
> API Gateway 엣지 최적화 사용자 지정 도메인 이름이 포함된 ACM 인증서를 사용하려면 `us-east-1` 리전(미국 동부(버지니아 북부))에서 인증서를 요청하거나 가져와야 합니다.

8. 도메인 이름 생성을 선택합니다.
9. 사용자 지정 도메인 이름을 생성한 후에는 *`distribution-id`*`.cloudfront.net` 형태의 연결된 CloudFront 배포 도메인 이름이 인증서 ARN과 함께 콘솔에 표시됩니다. 출력에 표시된 CloudFront 배포 도메인 이름을 적어 둡니다. 다음 단계인 DNS에서 사용자 지정 도메인의 CNAME 값 또는 A 레코드 별칭 대상을 설정할 때 필요합니다.

> **Note**
>
> 새로 생성된 사용자 지정 도메인 이름은 사용할 수 있을 때까지 약 40분이 소요됩니다. 그동안 DNS 레코드 별칭을 구성하여 사용자 지정 도메인 이름을 연결된 CloudFront 배포 도메인 이름에 매핑하고, 사용자 지정 도메인 이름이 초기화되는 동안 사용자 지정 도메인 이름의 기본 경로 매핑을 설정할 수 있습니다.

10. 그런 다음 사용자 지정 도메인 이름을 연결된 CloudFront 배포에 매핑하도록 DNS 공급자를 사용하여 DNS 레코드를 구성합니다. Amazon Route 53에 대한 지침은 Amazon Route 53 개발자 안내서에서 도메인 이름을 사용하여 Amazon API Gateway API로 트래픽 라우팅을 참조하세요.

    DNS 공급자 대부분의 경우, 사용자 지정 도메인 이름은 CNAME 리소스 레코드 세트로 호스팅 영역에 추가됩니다. CNAME 레코드 이름은 앞서 도메인 이름에 입력한 사용자 지정 도메인 이름(예: `api.example.com`)을 지정합니다. CNAME 레코드 값은 CloudFront 배포의 도메인 이름을 지정합니다. 그러나 사용자 지정 도메인이 zone apex인 경우(즉, `example.com`이 아닌 `api.example.com`인 경우) CNAME 레코드는 작동하지 않습니다. zone apex는 일반적으로 조직의 루트 도메인이라고도 합니다. zone apex의 경우 A 레코드 별칭을 사용해야 하는데, 다만 DNS 공급자가 그것을 지원해야 합니다.

Route 53을 사용하면 사용자 지정 도메인 이름의 A 레코드 별칭을 생성하고 CloudFront 배포 도메인 이름을 별칭 대상으로 지정할 수 있습니다. 다시 말해서 사용자 지정 도메인이 zone apex라고 하더라도 Route 53은 사용자 지정 도메인 이름을 라우팅할 수 있습니다. 자세한 내용은 Amazon Route 53 개발자 안내서의 별칭 및 비별칭 리소스 레코드 세트 간의 선택에서 확인할 수 있습니다.

도메인 이름 매핑이 전적으로 Route 53 내에서 수행되기 때문에 A 레코드 별칭을 사용하면 기본 CloudFront 배포 도메인 이름의 노출도 방지할 수 있습니다. 이러한 이유로 가능하면 항상 Route 53 A 레코드 별칭을 사용하는 것이 좋습니다.

API Gateway 콘솔 사용 외에도 API Gateway REST API, AWS CLI 또는 AWS SDK 중 하나를 사용하여 API에 대한 사용자 지정 도메인 이름을 설정할 수 있습니다. 다음 절차에서는 REST API 호출을 사용하여 이를 수행하는 단계를 간략하게 설명합니다.

API Gateway REST API를 사용하여 사용자 지정 도메인 이름을 설정하려면

1. domainname:create를 호출하여 사용자 지정 도메인 이름과 AWS Certificate Manager에 저장되는 인증서의 ARN을 지정합니다.

   성공적인 API 호출은 페이로드에 인증서 ARN뿐 아니라 연결된 CloudFront 배포 이름을 포함하는 `201 Created` 응답을 반환합니다.
2. 출력에 표시된 CloudFront 배포 도메인 이름을 적어 둡니다. 다음 단계인 DNS에서 사용자 지정 도메인의 CNAME 값 또는 A 레코드 별칭 대상을 설정할 때 필요합니다.
3. 이전 절차에 따라 사용자 지정 도메인 이름을 CloudFront 배포 이름에 매핑하도록 A 레코드 별칭을 설정합니다.

이 REST API 호출의 코드 예는 domainname:create를 참조하세요.

CloudTrail에서 사용자 지정 도메인 이름 생성 로그

계정에서 수행한 API Gateway 호출을 로깅하기 위해 CloudTrail을 활성화한 경우, API에 대한 사용자 지정 도메인 이름을 생성하거나 업데이트할 때 API Gateway는 연결된 CloudFront 배포 업데이트를 기록합니다. 이러한 CloudFront 배포는 API Gateway가 소유하므로, 이러한 각 보고된 CloudFront 배포는 API 소유자의 계정 ID 대신 다음과 같은 리전별 API Gateway 계정 ID 중 하나로 식별됩니다.

Region	계정 ID
us-east-1	392220576650

Region	계정 ID
us-east-2	718770453195
us-west-1	968246515281
us-west-2	109351309407
ca-central-1	796887884028
eu-west-1	631144002099
eu-west-2	544388816663
eu-west-3	061510835048
eu-central-1	474240146802
eu-central-2	166639821150
eu-north-1	394634713161
eu-south-1	753362059629
eu-south-2	359345898052
ap-northeast-1	969236854626
ap-northeast-2	020402002396
ap-northeast-3	360671645888
ap-southeast-1	195145609632
ap-southeast-2	798376113853
ap-southeast-3	652364314486
ap-southeast-4	849137399833
ap-south-1	507069717855

Region	계정 ID
ap-south-2	644042651268
ap-east-1	174803364771
sa-east-1	287228555773
me-south-1	855739686837
me-central-1	614065512851

사용자 지정 도메인 이름을 호스트 이름으로 사용하여 API의 기본 경로 매핑 구성

단일 사용자 지정 도메인 이름을 여러 API의 호스트 이름으로 사용할 수 있습니다. 사용자 지정 도메인 이름에 대한 기본 경로 매핑을 구성하여 이를 달성합니다. 기본 경로 매핑을 사용하면 사용자 지정 도메인의 API를 사용자 지정 도메인 이름 및 연관된 기본 경로의 결합을 통해 액세스할 수 있습니다.

예를 들어, `PetStore`라는 API 와 `PetShop`이라는 다른 API를 생성하고 API Gateway에서 `api.example.com`의 사용자 지정 도메인 이름을 설정한 경우 `PetStore` API의 URL을 `https://api.example.com` 또는 `https://api.example.com/myPetStore`로 설정할 수 있습니다. `PetStore` API는 빈 문자열의 기본 경로 또는 `myPetStore`의 사용자 지정 도메인 이름 아래의 `api.example.com`와 연관됩니다. 마찬가지로 `yourPetShop` API에 대해 `PetShop`의 기본 경로를 지정할 수 있습니다. `https://api.example.com/yourPetShop`의 URL은 `PetShop` API의 루트 URL입니다.

API에 대한 기본 경로를 설정하기 전에 의 단계를 완료합니다[API Gateway API에 대한 엣지 최적화 사용자 지정 도메인 이름 설정]

다음 절차에서는 사용자 지정 도메인 이름에서 API 스테이지로 경로를 매핑하도록 API 매핑을 설정합니다.

API Gateway 콘솔을 사용하여 API 매핑 이름을 생성하려면

1. https://console.aws.amazon.com/apigateway에서 API Gateway 콘솔에 로그인합니다.
2. 사용자 지정 도메인 이름을 선택합니다.
3. API 매핑 구성을 선택합니다.
4. Add new mapping(새 매핑 추가)을 선택합니다.
5. 매핑에 대한 API, 스테이지, 및 경로(선택 사항)를 지정합니다.

6. 저장을 선택합니다.

또한 API Gateway REST API, AWS CLI 또는 AWS SDK 중 하나를 호출하여 사용자 지정 도메인 이름을 호스트 이름으로 사용하는 API의 기본 경로 매핑을 설정할 수 있습니다. 다음 절차에서는 REST API 호출을 사용하여 이를 수행하는 단계를 간략하게 설명합니다.

API Gateway REST API를 사용하여 API의 기본 경로 매핑을 설정하려면

- 요청 페이로드에서 `basePath`, `restApiId` 및 배포 `stage` 속성을 지정하여 특정 사용자 지정 도메인 이름에 대해 basepathmapping:create를 호출합니다.

  성공적인 API 응답은 `201 Created` 응답을 반환합니다.

REST API 호출의 코드 예는 basepathmapping:create를 참조하세요.

ACM에 가져온 인증서 교체

ACM은 발급한 인증서의 갱신을 자동으로 처리합니다. 사용자 지정 도메인 이름에 대해 ACM이 발급한 인증서는 교체할 필요가 없습니다. CloudFront가 사용자를 대신하여 처리합니다.

하지만 인증서를 ACM에 가져와서 사용자 지정 도메인 이름에 사용할 경우 만료되기 전에 인증서를 교체해야 합니다. 여기에는 도메인 이름에 대한 새 타사 인증서를 가져오고 기존 인증서를 새 인증서로 교체하는 작업이 포함됩니다. 새로 가져온 인증서가 만료되면 이 프로세스를 반복해야 합니다. 또는 도메인 이름에 대한 새 인증서를 발급하도록 ACM에 요청하고 기존 인증서를 ACM에서 발급한 새 인증서로 교체할 수 있습니다. 이후에는 ACM과 CloudFront에서 인증서 교체를 자동으로 처리하도록 둘 수 있습니다. 새 ACM 인증서를 생성하거나 가져오려면 지정된 도메인 이름에 대해 새 ACM 인증서를 요청하거나 가져오는 단계를 수행합니다.

도메인 이름에 대한 인증서를 교체하려면 API Gateway 콘솔, API Gateway REST API, AWS CLI 또는 AWS SDK 중 하나를 사용할 수 있습니다.

API Gateway 콘솔을 사용하여 ACM에 가져온 만료될 인증서를 교체하려면

1. ACM에서 인증서를 요청하거나 가져옵니다.
2. API Gateway 콘솔로 돌아갑니다.
3. API Gateway 콘솔 기본 탐색 창에서 Custom Domain Names(사용자 지정 도메인 이름)를 선택합니다.
4. 사용자 지정 도메인 이름을 선택합니다.

5. [Edit]를 선택합니다.
6. ACM certificate(ACM 인증서) 드롭다운 목록에서 원하는 인증서를 선택합니다.
7. 저장을 선택하여 사용자 지정 도메인 이름에 대한 인증서 교체를 시작합니다.

> **Note**
>
> 프로세스를 완료하는 데는 약 40분이 소요됩니다. 교체가 완료된 후에 ACM 인증서(ACM Certificate) 옆에 있는 양방향 화살표를 선택하여 원래 인증서로 롤백할 수 있습니다.

사용자 지정 도메인 이름에 대해 가져온 인증서를 프로그래밍 방식으로 교체하는 방법을 설명하기 위해 여기서는 API Gateway REST API를 사용하여 단계를 설명합니다.

API Gateway REST API를 사용하여 가져온 인증서 교체

- 지정된 도메인 이름에 대한 새 ACM 인증서의 ARN을 지정하여 domainname:update 작업을 호출합니다.

사용자 지정 도메인 이름을 사용하여 API 호출

사용자 지정 도메인 이름을 사용하여 API를 호출하는 것은 올바른 URL을 사용한 경우 기본 도메인 이름을 사용하여 API를 호출하는 것과 동일합니다.

다음 예제에서는 기본 URL 세트와 지정된 리전(`udxjef`) 및 해당 사용자 지정 도메인 이름(`qf3duz`)의 두 API(`us-east-1` 및 `api.example.com`)에 해당되는 사용자 지정 URL을 비교 및 대비합니다.

기본 및 사용자 지정 도메인 이름이 포함된 API의 루트 URL

API ID	Stage	기본 URL	기본 경로	사용자 지정 URL
udxjef	pro	https://udxjef.execute-api.us-east-1.amazonaws.com/pro	/petstore	https://api.example.com/petstore
udxjef	tst	https://udxjef.execute-api.us-east-1	/petdepot	https://api.example.com/petdepot

API ID	Stage	기본 URL	기본 경로	사용자 지정 URL
		.amazonaws.com/tst		
qf3duz	dev	https://qf3duz.execute-api.us-east-1.amazonaws.com/dev	/bookstore	https://api.example.com/bookstore
qf3duz	tst	https://qf3duz.execute-api.us-east-1.amazonaws.com/tst	/bookstand	https://api.example.com/bookstand

API Gateway는 SNI(Server Name Indication)를 사용하여 API에 대한 사용자 지정 도메인 이름을 지원합니다. 브라우저 또는 SNI를 지원하는 클라이언트 라이브러리를 사용하여 사용자 지정 도메인 이름이 포함된 API를 호출할 수 있습니다.

API Gateway는 CloudFront 배포에 SNI를 적용합니다. CloudFront에서 사용자 지정 도메인 이름을 사용하는 방법에 대한 자세한 내용은 Amazon CloudFront 사용자 지정 SSL을 참조하세요.

API Gateway에서 리전 사용자 지정 도메인 이름 설정

AWS 리전의 경우 리전 API 엔드포인트에 대한 사용자 지정 도메인 이름을 생성할 수 있습니다. 사용자 지정 도메인 이름을 생성하려면 리전별 ACM 인증서를 제공해야 합니다. 사용자 지정 도메인 이름 인증서 생성 또는 업로드에 대한 자세한 내용은 AWS Certificate Manager에서 인증서 준비하기를 참조하세요.

> **Important**
>
> API Gateway 리전 사용자 지정 도메인 이름의 경우, API와 동일한 리전에서 인증서를 요청하거나 가져와야 합니다.

ACM 인증서로 리전 사용자 지정 도메인 이름을 생성(또는 마이그레이션)하면 API Gateway는 해당 계정에 서비스 연결 역할을 생성합니다(이 역할이 아직 없는 경우). 서비스 연결 역

할은 ACM 인증서를 해당 리전 엔드포인트에 연결하는 데 필요합니다. 그 역할의 이름은 AWSServiceRoleForAPIGateway이고 이 역할에는 APIGatewayServiceRolePolicy 관리형 정책이 연결됩니다. 서비스 연결 역할을 사용하는 방법에 대한 자세한 내용은 서비스 연결 역할 사용을 참조하세요.

> **Important**
>
> DNS 레코드를 생성하여 사용자 지정 도메인 이름이 리전 도메인 이름을 가리키도록 해야 합니다. 이렇게 하면 목적지가 사용자 지정 도메인 이름인 트래픽이 API의 리전 호스트 이름으로 라우팅됩니다. DNS 레코드는 CNAME 또는 “A” 유형일 수 있습니다.

주제

- API Gateway 콘솔을 사용하여 ACM 인증서로 리전 사용자 지정 도메인 이름 설정
- AWS CLI를 사용하여 ACM 인증서로 리전 사용자 지정 도메인 이름 설정

API Gateway 콘솔을 사용하여 ACM 인증서로 리전 사용자 지정 도메인 이름 설정

API Gateway 콘솔을 사용하여 리전 사용자 지정 도메인 이름을 설정하려면 다음 절차를 따르세요.

API Gateway 콘솔을 사용하여 리전 사용자 지정 도메인 이름을 설정하려면

1. https://console.aws.amazon.com/apigateway에서 API Gateway 콘솔에 로그인합니다.
2. 기본 탐색 창에서 사용자 지정 도메인 이름을 선택합니다.
3. Create를 선택합니다.
4. 도메인 이름에 도메인 이름을 입력합니다.
5. 구성에서 Regional(지역별)을 선택합니다.
6. 최소 TLS 버전을 선택합니다.
7. ACM 인증서를 선택합니다. 인증서는 API와 동일한 리전에 있어야 합니다.
8. Create를 선택합니다.
9. Route 53 설명서에 따라 트래픽을 API Gateway로 라우팅하도록 Route 53을 구성합니다.

다음 절차에서는 사용자 지정 도메인 이름에서 API 스테이지로 경로를 매핑하도록 API 매핑을 설정합니다.

API Gateway 콘솔을 사용하여 API 매핑 이름을 생성하려면

1. https://console.aws.amazon.com/apigateway에서 API Gateway 콘솔에 로그인합니다.
2. 사용자 지정 도메인 이름을 선택합니다.
3. API 매핑 구성을 선택합니다.
4. Add new mapping(새 매핑 추가)을 선택합니다.
5. 매핑에 대한 API, 스테이지 및 경로를 지정합니다.
6. 저장(Save)을 선택합니다.

사용자 지정 도메인에 대한 기본 경로 매핑 설정에 대해 자세히 알아보려면 사용자 지정 도메인 이름을 호스트 이름으로 사용하여 API의 기본 경로 매핑 구성 단원을 참조하세요.

AWS CLI를 사용하여 ACM 인증서로 리전 사용자 지정 도메인 이름 설정

AWS CLI를 사용하여 리전 API에 대한 사용자 지정 도메인 이름을 설정하려면 다음 절차를 따르세요.

1. `create-domain-name`을 호출하여 사용자 지정 도메인 이름과 리전 인증서의 ARN을 지정합니다.

```
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
```

   지정된 인증서의 출처는 `us-west-2` 리전이고, 이 예제에서는 기본 API의 출처도 같은 리전이라고 가정합니다.

   호출이 성공하면 다음과 비슷한 결과가 반환됩니다.

```
{
    "ApiMappingSelectionExpression": "$request.basepath",
    "DomainName": "regional.example.com",
    "DomainNameConfigurations": [
        {
            "ApiGatewayDomainName": "d-id.execute-api.us-west-2.amazonaws.com",
            "CertificateArn": "arn:aws:acm:us-west-2:123456789012:certificate/id",
            "DomainNameStatus": "AVAILABLE",
            "EndpointType": "REGIONAL",
            "HostedZoneId": "id",
```

```
            "SecurityPolicy": "TLS_1_2"
        }
    ]
}
```

DomainNameConfigurations 속성 값은 리전 API의 호스트 이름을 반환합니다. DNS 레코드를 생성하여 사용자 지정 도메인 이름이 이 리전 도메인 이름을 가리키도록 해야 합니다. 이렇게 하면 목적지가 사용자 지정 도메인 이름인 트래픽이 이 리전 API의 호스트 이름으로 라우팅됩니다.

2. DNS 레코드를 생성하여 사용자 지정 도메인 이름 및 리전 도메인 이름을 연결합니다. 이렇게 하면 목적지가 사용자 지정 도메인 이름인 요청이 API의 리전 호스트 이름으로 라우팅됩니다.
3. 기본 경로 매핑을 추가하여 지정된 사용자 지정 도메인 이름(예: 0qzs2sy7bh)으로 배포 단계(예: test)에서 지정 API(예: regional.example.com)를 노출합니다.

```
aws apigatewayv2 create-api-mapping \
    --domain-name 'regional.example.com' \
    --api-mapping-key 'myApi' \
    --api-id 0qzs2sy7bh \
    --stage 'test'
```

결과적으로 그 단계에서 배포되는 API에 대해 사용자 지정 도메인 이름을 사용하는 기본 URL은 https://regional.example.com/myAPI이 됩니다.

4. 리전 사용자 지정 도메인 이름을 지정된 호스팅 영역 ID의 호스트 이름으로 매핑하도록 DNS 레코드를 구성합니다. 먼저, 리전 도메인 이름에 대해 DNS 레코드를 설정하기 위한 구성을 포함하는 JSON 파일을 생성합니다. 다음 예제에서는 DNS A 레코드를 생성하여 사용자 지정 도메인 이름을 생성하는 과정에서 프로비저닝된 리전 호스트 이름(regional.example.com)에 리전 사용자 지정 도메인 이름(d-numh1z56v6.execute-api.us-west-2.amazonaws.com)을 매핑하는 방법을 보여줍니다. DNSName의 HostedZoneId 및 AliasTarget 속성은 사용자 지정 도메인 이름의 regionalDomainName 및 regionalHostedZoneId 값을 가져올 수 있습니다. 또한 Amazon API Gateway 엔드포인트 및 할당량에서 리전 Route 53 호스팅 영역 ID를 가져올 수 있습니다.

```
{
  "Changes": [
    {
      "Action": "CREATE",
      "ResourceRecordSet": {
        "Name": "regional.example.com",
        "Type": "A",
```

```
        "AliasTarget": {
          "DNSName": "d-numh1z56v6.execute-api.us-west-2.amazonaws.com",
          "HostedZoneId": "Z2OJLYMUO9EFXC",
          "EvaluateTargetHealth": false
        }
      }
    }
  ]
}
```

5. 다음 CLI 명령을 실행합니다.

```
aws route53 change-resource-record-sets \
    --hosted-zone-id {your-hosted-zone-id} \
    --change-batch file://path/to/your/setup-dns-record.json
```

*{your-hosted-zone-id}*는 계정에 설정된 DNS 레코드의 Route 53 호스팅 영역 ID입니다. change-batch 파라미터 값은 폴더(*path/to/your*)의 JSON 파일(*setup-dns-record.json*)을 가리킵니다.

사용자 지정 도메인 이름을 다른 API 엔드포인트로 마이그레이션

엣지 최적화 및 리전 엔드포인트 간에 사용자 지정 도메인 이름을 마이그레이션할 수 있습니다. 먼저 사용자 지정 도메인 이름에 대한 기존 endpointConfiguration.types 목록에 새로운 엔드포인트 구성 유형을 추가합니다. 그 다음에 DNS 레코드를 설정하여 사용자 지정 도메인 이름이 새로 프로비저닝된 엔드포인트를 가리키도록 합니다. 선택 사항인 마지막 단계는 폐기된 사용자 지정 도메인 이름 구성 데이터를 제거하는 것입니다.

마이그레이션을 계획할 때는 엣지 최적화 API의 사용자 지정 도메인 이름의 경우 ACM이 제공하는 필수 인증서가 미국 동부(버지니아 북부) 리전(us-east-1)의 인증서여야 한다는 점을 기억해야 합니다. 이 인증서는 모든 지리적 위치에 배포됩니다. 하지만 리전 API의 경우 리전 도메인 이름에 대한 ACM 인증서는 API를 호스팅하는 동일한 리전의 인증서여야 합니다. 먼저 API에 대해 로컬인 리전에 새로운 ACM 인증서를 요청하여 us-east-1 리전에 있지 않은 엣지 최적화 사용자 지정 도메인 이름을 리전 사용자 지정 도메인 이름으로 마이그레이션할 수 있습니다.

API Gateway에서 엣지 최적화 사용자 지정 도메인 이름과 리전 사용자 지정 도메인 이름 간에 마이그레이션을 완료하려면 최대 60초까지 걸릴 수 있습니다. 새로 생성된 엔드포인트가 트래픽을 수신할 준비가 되도록 하는 경우 DNS 레코드를 업데이트하는 시점에 따라 마이그레이션 시간이 달라집니다.

주제

- AWS CLI를 사용한 사용자 지정 도메인 이름 마이그레이션

AWS CLI를 사용한 사용자 지정 도메인 이름 마이그레이션

AWS CLI를 사용하여 엣지 최적화 엔드포인트에서 리전 엔드포인트로 또는 그 반대로 사용자 지정 도메인 이름을 마이그레이션하려면 `update-domain-name` 명령을 호출하여 새 엔드포인트 유형을 추가하고 선택 사항으로 `update-domain-name` 명령을 호출하여 이전 엔드포인트 유형을 제거합니다.

주제

- 엣지 최적화 사용자 지정 도메인 이름을 리전으로 마이그레이션
- 리전 사용자 지정 도메인 이름을 엣지 최적화 사용자 지정 도메인 이름으로 마이그레이션

엣지 최적화 사용자 지정 도메인 이름을 리전으로 마이그레이션

엣지 최적화 사용자 지정 도메인 이름을 리전 사용자 지정 도메인 이름으로 마이그레이션하려면 다음과 같이 update-domain-name CLI 명령을 호출하세요.

```
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' } \
      ]
```

리전 인증서는 리전 API와 동일한 리전의 것이어야 합니다.

성공적인 응답에는 다음과 유사한 `200 OK` 상태 코드와 본문이 있습니다.

```
{
    "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"
}
```

그 결과 얻는 `regionalDomainName` 속성은 마이그레이션된 리전 사용자 지정 도메인 이름에 대해 리전 API 호스트 이름을 반환합니다. 리전 사용자 지정 도메인 이름이 이 리전 호스트 이름을 가리키도록 DNS 레코드를 설정해야 합니다. 이렇게 하면 목적지가 사용자 지정 도메인 이름인 트래픽이 리전 호스트로 라우팅됩니다.

DNS 레코드가 설정된 후에는 [update-domain-name](#)의 AWS CLI 명령을 호출하여 엣지 최적화 사용자 지정 도메인 이름을 제거할 수 있습니다.

```
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'} \
        ]
```

리전 사용자 지정 도메인 이름을 엣지 최적화 사용자 지정 도메인 이름으로 마이그레이션

리전 사용자 지정 도메인 이름을 엣지 최적화 사용자 지정 도메인 이름으로 마이그레이션하려면 다음과 같이 `update-domain-name`의 AWS CLI 명령을 호출하세요.

```
aws apigateway update-domain-name \
    --domain-name 'api.example.com' \
    --patch-operations  [ \
            { op:'add', path:'/endpointConfiguration/types',value: 'EDGE' }, \
            { op:'add', path:'/certificateName', value:'edge-cert'}, \
            { op:'add', path:'/certificateArn', value: 'arn:aws:acm:us-
east-1:123456789012:certificate/34a95aa1-77fa-427c-aa07-3a88bd9f3c0a' } \
        ]
```

엣지 최적화 도메인 인증서는 `us-east-1` 리전에 생성되어야 합니다.

성공적인 응답에는 다음과 유사한 `200 OK` 상태 코드와 본문이 있습니다.

```
{
```

```
    "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"
}
```

API Gateway는 지정된 사용자 지정 도메인 이름에 대해 distributionDomainName 속성 값으로 엣지 최적화 API 호스트 이름을 반환합니다. DNS 레코드를 설정하여 엣지 최적화 사용자 지정 도메인 이름이 이 배포 도메인 이름을 가리키도록 해야 합니다. 이렇게 하면 목적지가 엣지 최적화 사용자 지정 도메인 이름인 트래픽이 엣지 최적화 API 호스트 이름으로 라우팅됩니다.

DNS 레코드가 설정된 후에는 다음과 같이 사용자 지정 도메인 이름의 REGION 엔드포인트 유형을 제거할 수 있습니다.

```
aws apigateway update-domain-name \
    --domain-name api.example.com \
    --patch-operations [ \
        {op:'remove', path:'/endpointConfiguration/types', value:'REGIONAL'}, \
        {op:'remove', path:'regionalCertificateArn'} \
      ]
```

이 명령의 결과는 엣지 최적화 도메인 이름 구성 데이터만 있는 다음 출력과 유사합니다.

```
{
    "certificateArn": "arn:aws:acm:us-
east-1:738575810317:certificate/34a95aa1-77fa-427c-aa07-3a88bd9f3c0a",
    "certificateName": "edge-cert",
    "certificateUploadDate": "2017-10-16T23:22:57Z",
    "distributionDomainName": "d1frvgze7vy1bf.cloudfront.net",
    "domainName": "regional.haymuto.com",
    "endpointConfiguration": {
```

```
        "types": "EDGE"
    }
}
```

REST API에 대한 API 매핑 작업

API 매핑을 사용하여 API 스테이지를 사용자 지정 도메인 이름에 연결합니다. 도메인 이름을 생성하고 DNS 레코드를 구성한 후에는 API 매핑을 사용하여 사용자 지정 도메인 이름을 통해 API로 트래픽을 보냅니다.

API 매핑은 API, 스테이지 및 매핑에 사용할 경로(선택 사항)를 지정합니다. 예를 들어 API의 `production` 스테이지를 `https://api.example.com/orders`에 매핑할 수 있습니다.

HTTP 및 REST API 스테이지를 동일한 사용자 지정 도메인 이름에 매핑할 수 있습니다.

API 매핑을 생성하기 전에 API, 스테이지 및 사용자 지정 도메인 이름이 있어야 합니다. 사용자 지정 도메인 이름 생성에 대한 자세한 내용은 the section called “리전 사용자 지정 도메인 이름 설정” 단원을 참조하세요.

API 요청 라우팅

여러 수준(예: `orders/v1/items` 및 `orders/v2/items`)으로 API 매핑을 구성할 수 있습니다.

> **Note**
>
> 여러 수준으로 API 매핑을 구성하려면 사용자 지정 도메인 이름이 리전별로 고유해야 하며 TLS 1.2 보안 정책을 사용해야 합니다.

여러 수준 API 매핑의 경우 API Gateway는 일치하는 경로가 가장 긴 API 매핑으로 요청을 라우팅합니다. API Gateway는 호출할 API를 선택하기 위해 API 경로가 아닌 API 매핑에 대해 구성된 경로만 고려합니다. 요청과 일치하는 경로가 없으면 API Gateway는 빈 경로 (`none`)에 매핑한 API로 요청을 보냅니다.

여러 수준 API 매핑을 사용하는 사용자 지정 도메인 이름의 경우 API Gateway는 일치하는 경로가 가장 긴 API 매핑으로 요청을 라우팅합니다.

예를 들어 다음 API 매핑이 있는 사용자 지정 도메인 이름 `https://api.example.com`을(를) 살펴보겠습니다.

1. (none)이(가) API 1에 매핑됩니다.
2. orders이(가) API 2에 매핑됩니다.
3. orders/v1/items이(가) API 3에 매핑됩니다.
4. orders/v2/items이(가) API 4에 매핑됩니다.
5. orders/v2/items/categories이(가) API 5에 매핑됩니다.

요청	선택한 API	설명
https://api.example.com/orders	API 2	요청은 이 API 매핑과 정확히 일치합니다.
https://api.example.com/orders/v1/items	API 3	요청은 이 API 매핑과 정확히 일치합니다.
https://api.example.com/orders/v2/items	API 4	요청은 이 API 매핑과 정확히 일치합니다.
https://api.example.com/orders/v1/items/123	API 3	API Gateway는 일치하는 경로가 가장 긴 매핑을 선택합니다. 요청 끝에 있는 123은(는) 선택 항목에 영향을 주지 않습니다.
https://api.example.com/orders/v2/items/categories/5	API 5	API Gateway는 일치하는 경로가 가장 긴 매핑을 선택합니다.
https://api.example.com/customers	API 1	API Gateway는 빈 매핑을 캐치올(catch-all)로 사용합니다.
https://api.example.com/ordersandmore	API 2	API Gateway는 일치하는 접두사가 가장 긴 매핑을 선택합니다. 단일 수준 매핑으로 구성된 사용자 지정 도메인 이름(예: 단지 https://api.example.com/

요청	선택한 API	설명
		`orders` 및 `https://api.example.com/`)의 경우 `ordersandmore` 와 일치하는 경로가 없으므로 API Gateway는 `API 1`을 선택합니다.

제한 사항

- API 매핑에서 사용자 지정 도메인 이름과 매핑된 API는 동일한 AWS 계정에 있어야 합니다.
- API 매핑에는 문자, 숫자 및 문자 `$-_.+!*'()/`만 포함해야 합니다.
- API 매핑에서 경로의 최대 길이는 300자입니다.
- 각 도메인 이름에 대해 여러 수준의 API 매핑이 200개 있을 수 있습니다.
- TLS 1.2 보안 정책을 사용하여 HTTP API를 리전별 사용자 지정 도메인 이름에만 매핑할 수 있습니다.
- WebSocket API를 HTTP API 또는 REST API와 동일한 사용자 지정 도메인 이름에 매핑할 수 없습니다.

API 매핑 생성

API 매핑을 생성하려면 먼저 사용자 지정 도메인 이름, API 및 스테이지를 생성해야 합니다. 사용자 지정 도메인 이름 생성에 대한 자세한 내용은 the section called “리전 사용자 지정 도메인 이름 설정” 단원을 참조하세요.

예를 들어 모든 리소스를 생성하는 AWS Serverless Application Model 템플릿에 대해서는 GitHub의 SAM 세션을 참조하세요.

AWS Management Console

API 매핑 생성

1. https://console.aws.amazon.com/apigateway에서 API Gateway 콘솔에 로그인합니다.
2. 사용자 지정 도메인 이름을 선택합니다.
3. 이미 생성한 사용자 지정 도메인 이름을 선택합니다.

4. API 매핑을 선택합니다.
5. API 매핑 구성을 선택합니다.
6. 새 매핑 추가를 선택합니다.
7. API, 스테이지 및 경로(선택 사항)를 입력합니다.
8. 저장을 선택합니다.

AWS CLI

다음 AWS CLI 명령은 API 매핑을 생성합니다. 이 예에서 API Gateway는 지정된 API 및 스테이지로 `api.example.com/v1/orders` 요청을 보냅니다.

> **Note**
>
> 여러 수준으로 API 매핑을 생성하려면 `apigatewayv2`을(를) 사용해야 합니다.

```
aws apigatewayv2 create-api-mapping \
    --domain-name api.example.com \
    --api-mapping-key v1/orders \
    --api-id a1b2c3d4 \
    --stage test
```

AWS CloudFormation

다음 AWS CloudFormation 예에서는 API 매핑을 생성합니다.

> **Note**
>
> 여러 수준으로 API 매핑을 생성하려면 `AWS::ApiGatewayV2`을(를) 사용해야 합니다.

```
MyApiMapping:
  Type: 'AWS::ApiGatewayV2::ApiMapping'
  Properties:
    DomainName: api.example.com
    ApiMappingKey: 'orders/v2/items'
    ApiId: !Ref MyApi
    Stage: !Ref MyStage
```

REST API에 대한 기본 엔드포인트 비활성화

기본적으로 클라이언트는 API에 대해 API Gateway가 생성하는 `execute-api` 엔드포인트를 사용하여 API를 호출할 수 있습니다. 클라이언트가 사용자 지정 도메인 이름을 사용해야만 API에 액세스할 수 있도록 하려면 기본 `execute-api` 엔드포인트를 비활성화합니다. 클라이언트는 여전히 기본 엔드포인트에 연결할 수 있지만 `403 Forbidden` 상태 코드를 받게 됩니다.

> **Note**
>
> 기본 엔드포인트를 비활성화하면 API의 모든 스테이지에 영향이 미칩니다.

다음 AWS CLI 명령은 REST API에 대한 기본 엔드포인트를 비활성화합니다.

```
aws apigateway update-rest-api \
    --rest-api-id abcdef123 \
    --patch-operations op=replace,path=/disableExecuteApiEndpoint,value='True'
```

기본 엔드포인트를 비활성화한 후 변경 사항을 적용하려면 API를 배포해야 합니다.

다음 AWS CLI 명령은 배포를 생성합니다.

```
aws apigateway create-deployment \
    --rest-api-id abcdef123 \
    --stage-name dev
```

DNS 장애 조치에 대한 사용자 지정 상태 확인 구성

Amazon Route 53 상태 확인을 사용하여 기본 AWS 리전의 API Gateway API에서 보조 리전에 있는 API Gateway API로의 DNS 장애 조치를 제어할 수 있습니다. 이는 리전 문제 발생 시 영향을 완화하는 데 도움이 될 수 있습니다. 사용자 지정 도메인을 사용하는 경우 클라이언트가 API 엔드포인트를 변경하지 않고도 장애 조치를 수행할 수 있습니다.

별칭 레코드로 Evaluate Target Health(대상 상태 평가)를 선택하면 해당 레코드는 해당 리전에서 API Gateway 서비스를 사용할 수 없는 경우에만 실패합니다. 경우에 따라 자체 API Gateway API가 그 이전에 중단될 수 있습니다. DNS 장애 조치를 직접 제어하려면 API Gateway API에 대한 사용자 지정

Route 53 상태 확인을 구성하세요. 이 예시에서는 운영자가 DNS 장애 조치를 제어하는 데 도움이 되는 CloudWatch 경보를 사용합니다. 장애 조치 구성을 위한 더 많은 예제 및 기타 고려 사항은 Route 53을 사용한 재해 복구 메커니즘 생성과 AWS Lambda 및 CloudWatch를 사용하여 VPC의 프라이빗 리소스에 대한 Route 53 상태 확인 수행을 참조하세요.

주제

- 필수 조건
- 1단계: 리소스 설정
- 2단계: 보조 리전으로의 장애 조치 시작
- 3단계: 장애 조치 테스트
- 4단계: 기본 리전으로 돌아가기
- 다음 단계: 사용자 지정 및 정기적인 테스트

필수 조건

이 절차를 완료하려면 다음 리소스를 만들고 구성해야 합니다.

- 자신이 소유한 웹 도메인
- 두 AWS 리전에 있는 해당 도메인 이름에 대한 ACM 인증서. 자세한 내용은 the section called “AWS Certificate Manager에서 인증서 준비하기” 섹션을 참조하세요.
- 도메인 이름을 위한 Route 53 호스팅 영역. 자세한 내용은 Amazon Route 53 개발자 안내서의 호스팅 영역 작업 섹션을 참조하세요.

도메인 이름에 대한 Route 53 장애 조치 DNS 레코드를 생성하는 방법에 대한 자세한 내용은 Amazon Route 53 개발자 안내서의 라우팅 정책 선택을 참조하세요. CloudWatch 경보를 모니터링하는 방법에 대한 자세한 내용은 Amazon Route 53 개발자 안내서의 CloudWatch 경보 모니터링을 참조하세요.

1단계: 리소스 설정

이 예에서는 다음 리소스를 생성하여 도메인 이름에 대한 DNS 장애 조치를 구성합니다.

- 두 AWS 리전에 있는 API Gateway API
- 두 AWS 리전에 있는 이름이 같은 API Gateway 사용자 지정 도메인 이름
- API Gateway API를 사용자 지정 도메인 이름에 연결하는 API Gateway API 매핑
- 도메인 이름에 대한 Route 53 장애 조치 DNS 레코드
- 보조 리전의 CloudWatch 경보

- 보조 리전의 CloudWatch 경보를 기반으로 하는 Route 53 상태 확인

먼저 기본 및 보조 리전에 필요한 모든 리소스가 있는지 확인하세요. 보조 리전에는 알람 및 상태 확인이 포함되어야 합니다. 이렇게 하면 장애 조치를 수행하기 위해 기본 리전에 의존하지 않아도 됩니다. 이러한 리소스를 만드는 예제 AWS CloudFormation 템플릿은 primary.yaml 및 secondary.yaml 섹션을 참조하세요.

> **Important**
>
> 보조 리전으로 장애 조치하기 전에 필요한 리소스를 모두 사용할 수 있는지 확인하세요. 그렇지 않으면 API가 보조 리전의 트래픽을 수용할 준비가 되지 않습니다.

2단계: 보조 리전으로의 장애 조치 시작

다음 예에서 대기 리전은 CloudWatch 지표를 수신하고 장애 조치를 시작합니다. 장애 조치를 시작하기 위해 운영자의 개입이 필요한 사용자 지정 메트릭을 사용합니다.

```
aws cloudwatch put-metric-data \
    --metric-name Failover \
    --namespace HealthCheck \
    --unit Count \
    --value 1 \
    --region us-west-1
```

구성한 CloudWatch 경보에 해당하는 데이터로 지표 데이터를 바꿉니다.

3단계: 장애 조치 테스트

API를 호출하고 보조 리전으로부터 응답이 수신되는지 확인하세요. 1단계에서 예제 템플릿을 사용한 경우 장애 조치 후 응답이 `{"message": "Hello from the secondary Region!"}`에서 `{"message": "Hello from the primary Region!"}`으로 변경됩니다.

```
curl https://my-api.example.com

{"message": "Hello from the secondary Region!"}
```

4단계: 기본 리전으로 돌아가기

기본 리전으로 돌아가려면 상태 확인을 통과하도록 하는 CloudWatch 지표를 전송하세요.

```
aws cloudwatch put-metric-data \
    --metric-name Failover \
    --namespace HealthCheck \
    --unit Count \
    --value 0 \
    --region us-west-1
```

구성한 CloudWatch 경보에 해당하는 데이터로 지표 데이터를 바꿉니다.

API를 호출하고 기본 리전으로부터 응답이 수신되는지 확인하세요. 1단계에서 예제 템플릿을 사용한 경우 응답이 `{"message": "Hello from the primary Region!"}`에서 `{"message": "Hello from the secondary Region!"}`으로 변경됩니다.

```
curl https://my-api.example.com

{"message": "Hello from the primary Region!"}
```

다음 단계: 사용자 지정 및 정기적인 테스트

이 예제에서는 DNS 장애 조치를 구성하는 한 가지 방법을 보여줍니다. 장애 조치를 관리하는 상태 확인에 다양한 CloudWatch 지표 또는 HTTP 엔드포인트를 사용할 수 있습니다. 장애 조치 메커니즘을 정기적으로 테스트하여 예상대로 작동하는지, 운영자가 장애 조치 절차를 잘 알고 있는지 확인하세요.

REST API의 성능 최적화

API를 호출할 수 있도록 한 후 응답성을 향상하기 위해 API를 최적화해야 할 수 있습니다. API Gateway는 응답 캐싱 및 페이로드 압축과 같은 API를 최적화하기 위한 몇 가지 전략을 제공합니다. 이 섹션에서는 이러한 기능을 활성화하는 방법을 배울 수 있습니다.

주제

- 응답성 향상을 위한 API 캐싱 활성화
- API에 대한 페이로드 압축 활성화

응답성 향상을 위한 API 캐싱 활성화

Amazon API Gateway에서 API 캐싱을 활성화하여 엔드포인트의 응답을 캐싱할 수 있습니다. 캐싱을 사용하면 엔드포인트에 대한 호출 수를 줄이고 API에 대한 요청 지연 시간을 개선할 수 있습니다.

단계에 대한 캐싱을 활성화하면 API Gateway는 지정된 time-to-live (TTL) 기간 (초) 동안 엔드포인트의 응답을 캐싱합니다. 그런 다음 API Gateway는 엔드포인트에 요청하는 대신 캐시에서 엔드포인트 응답을 조회하여 요청에 응답합니다. API 캐싱에 대한 기본 TTL 값은 300초입니다. 최대 TTL 값은 3600초입니다. TTL=0이면 캐싱이 비활성화됩니다.

> **Note**
>
> 캐싱은 최선의 작업을 기반으로 이루어집니다. CloudWatch Amazon의 `CacheHitCount` 및 `CacheMissCount` 지표를 사용하여 API Gateway가 API 캐시에서 제공하는 요청을 모니터링할 수 있습니다.

캐싱할 수 있는 응답의 최대 크기는 1048576바이트입니다. 캐시 데이터 암호화는 캐싱할 경우 응답의 크기를 증가시킬 수 있습니다.

이것은 HIPAA 적격 서비스입니다. 1996년 미국 건강 보험 양도 및 책임법 (HIPAA) 및 보호 대상 건강 정보 (PHI) 를 처리, 저장 및 전송하기 위한 AWS 서비스 사용에 대한 AWS자세한 내용은 HIPAA 개요를 참조하십시오.

> **Important**
>
> 단계에 대한 캐싱을 활성화할 경우, `GET` 메서드만 기본적으로 캐싱을 활성화합니다. 이렇게 하면 API의 안전성 및 가용성이 보장됩니다. 메서드 설정을 재정의함으로써 다른 메서드에 대해 캐싱을 활성화할 수 있습니다.

> **Important**
>
> 캐싱의 경우 선택한 캐시 크기에 따라 시간 단위로 요금이 청구됩니다. 캐싱은 프리 티어 적용 대상이 아닙니다. AWS 자세한 내용은 API Gateway 요금을 참조하세요.

Amazon API Gateway 캐싱 활성화

API Gateway에서는 특정 단계에 대한 캐싱을 활성화할 수 있습니다.

캐싱을 활성화할 경우 캐시 용량을 선택해야 합니다. 일반적으로 용량이 클수록 성능이 우수하지만 비용이 더 높습니다. 지원되는 캐시 크기는 API Gateway API 참조를 참조하십시오 cacheClusterSize.

API Gateway에서 전용 캐시 인스턴스를 생성하여 캐싱을 활성화합니다. 이 프로세스에는 최대 4분이 걸릴 수 있습니다.

API Gateway에서 기존 캐시 인스턴스를 제거하고 수정된 용량으로 새 캐시 인스턴스를 생성하여 캐싱 용량을 변경합니다. 캐시된 모든 기존 데이터가 삭제됩니다.

> **Note**
>
> 캐시 용량은 캐시 인스턴스의 CPU, 메모리 및 네트워크 대역폭에 영향을 미칩니다. 따라서 캐시 용량이 캐시 성능에 영향을 줄 수 있습니다.
> API Gateway에서는 10분 로드 테스트를 실행하여 캐시 용량이 워크로드에 적합한지 확인하는 것이 좋습니다. 로드 테스트 중 트래픽이 프로덕션 트래픽을 미러링하는지 확인합니다. 예를 들어, 램프 업, 상수 트래픽 및 트래픽 급증을 포함합니다. 로드 테스트에는 캐시에서 제공할 수 있는 응답과 캐시에 항목을 추가하는 고유 응답이 포함되어야 합니다. 로드 테스트 중에 지연 시간, 4xx, 5xx, 캐시 적중 및 캐시 누락 지표를 모니터링합니다. 이러한 지표를 기반으로 필요에 따라 캐시 용량을 조정합니다. 부하 테스트에 대한 자세한 내용은 속도 제한에 도달하지 않도록 최상의 API Gateway 캐시 용량을 선택하려면 어떻게 하나요?를 참조하세요.

API Gateway 콘솔의 스테이지 페이지에서 캐싱을 구성합니다. 스테이지 캐시를 프로비저닝하고 기본 메서드 수준 캐시 설정을 지정합니다. 기본 메서드 수준 캐시를 켜면 해당 메서드에 메서드 재정의가 없는 한 스테이지의 모든 메서드에 대해 메서드 수준 캐싱이 설정됩니다. 스테이지에 배포하는 추가 `GET` 메서드에는 모두 메서드 수준 캐시가 있습니다. 스테이지의 특정 메서드에 대한 메서드 수준 캐싱 설정을 구성하려면 메서드 재정의를 사용할 수 있습니다. 메서드 재정의에 대한 자세한 내용은 을 참조하십시오. the section called “메서드 캐싱에 대한 단계 캐싱 재정의”

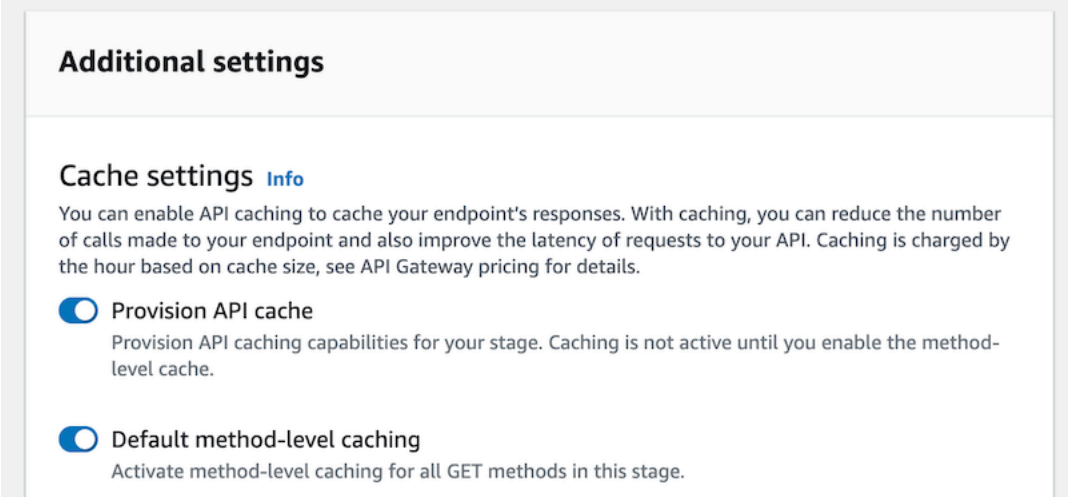
지정된 단계에 API 캐싱을 설정하려면 다음을 수행합니다.

1. https://console.aws.amazon.com/apigateway에서 API Gateway 콘솔에 로그인합니다.
2. 단계를 선택합니다.
3. API에 대한 단계 목록에서 단계를 선택합니다.
4. 스테이지 세부 정보 섹션에서 편집을 선택합니다.
5. 추가 설정에서 캐시 설정에 대해 API 캐시 프로비저닝을 켜십시오.

   이렇게 하면 스테이지에 캐시 클러스터를 프로비저닝할 수 있습니다.
6. 스테이지에 대한 캐싱을 활성화하려면 기본 메서드 수준 캐싱을 켜십시오.

이렇게 하면 스테이지의 모든 메서드에 대해 메서드 수준 캐싱이 활성화됩니다. 이 단계에 배포하는 추가 GET 메서드에는 모두 메서드 수준 캐시가 있습니다.

> **Note**
>
> 메서드 수준 캐시에 대한 기존 설정이 있는 경우 기본 메서드 수준 캐싱 설정을 변경해도 기존 설정에는 영향을 주지 않습니다.



7. 변경 사항 저장를 선택합니다.

> **Note**
>
> API Gateway에서 캐시 생성 또는 삭제를 완료하는 데 4분 정도 걸립니다.
> 캐시가 생성되면 캐시 클러스터 값이 에서 로 변경됩니다. `Create in progress` `Active`
> 캐시 삭제가 완료되면 캐시 클러스터 값이 에서 `Delete in progress` 로 변경됩니다`Inactive`.
> 스테이지의 모든 메서드에 대해 메서드 수준 캐싱을 설정하면 기본 메서드 수준 캐싱 값이 로 변경됩니다. `Active` 스테이지의 모든 메서드에 대해 메서드 수준 캐싱을 끄면 기본 메서드 수준 캐싱 값이 로 변경됩니다. `Inactive` 메서드 수준 캐시에 대한 기존 설정이 있는 경우 캐시 상태를 변경해도 해당 설정에는 영향을 주지 않습니다.

스테이지의 캐시 설정에서 캐싱을 활성화할 때 GET 메서드만 캐시됩니다. API의 안전성 및 가용성을 확보하려면 이 설정을 변경하지 않는 것이 좋습니다. 그러나 메서드 설정을 재정의함으로써 다른 메서드에 대해 캐싱을 활성화할 수 있습니다.

캐싱이 예상한 대로 작동하는지 확인하려면 두 가지 일반적인 옵션이 있습니다.

- API 및 스테이지의 CloudWatch CacheHitCount메트릭과 CacheMissCount스테이지를 검사하세요.
- 타임스탬프를 응답에 넣습니다.

> **Note**
>
> CloudFront 응답의 X-Cache 헤더를 사용하여 API가 API Gateway 캐시 인스턴스에서 제공되고 있는지 판단해서는 안 됩니다.

메서드 수준 캐싱을 위한 API Gateway 단계 수준 캐싱을 재정의합니다.

특정 메서드의 캐싱을 켜거나 끄면 단계 수준 캐시 설정을 재정의할 수 있습니다. 또한 TTL 기간을 수정하거나 캐시된 응답의 암호화를 켜거나 끌 수 있습니다.

스테이지 세부 정보에서 기본 메서드 수준 캐싱 설정을 변경해도 재정의가 있는 메서드 수준 캐시 설정에는 영향을 주지 않습니다.

캐싱하는 메서드가 중요 데이터를 응답으로 수신할 경우 Cache Settings(캐시 설정)에서 Encrypt cache data(캐시 데이터 암호화)를 선택합니다.

콘솔을 사용하여 개별 메서드에 대한 API 캐싱을 구성하려면 다음을 수행합니다.

1. https://console.aws.amazon.com/apigateway에서 API Gateway 콘솔에 로그인합니다.
2. API를 선택합니다.
3. 단계를 선택합니다.
4. 이를 위해 단계 보조 탐색 창에서 단계를 확장하고 메서드를 선택합니다.
5. 메서드 재정의 섹션에서 편집을 선택합니다.
6. 메서드 설정 섹션에서 메서드 캐시 활성화를 설정 또는 해제하거나 기타 원하는 옵션을 사용자 지정합니다.

> **Note**
>
> 스테이지에 캐시 클러스터를 프로비전할 때까지는 캐싱이 활성화되지 않습니다.

7. 저장을 선택합니다.

메서드/통합 파라미터를 캐시 키로 사용하여 캐시된 응답 인덱싱

캐시된 메서드 또는 통합에 사용자 지정 헤더, URL 경로 또는 쿼리 문자열 형식을 취할 수 있는 파라미터가 있는 경우 파라미터 중 일부 또는 모두를 사용하여 캐시 키를 생성할 수 있습니다. API Gateway에서는 사용된 파라미터 값에 따라 메서드의 응답을 캐싱할 수 있습니다.

> **Note**
>
> 리소스의 캐싱을 설정하려면 캐시 키가 필요합니다.

예를 들어 다음과 같은 형식의 요청이 있다고 가정하겠습니다.

```
GET /users?type=... HTTP/1.1
host: example.com
...
```

이 요청에서 `type`은 `admin` 또는 `regular` 값을 가질 수 있습니다. `type` 파라미터가 캐시 키의 일부로 포함되어 있는 경우, `GET /users?type=admin`의 응답은 `GET /users?type=regular`의 응답과 별도로 캐싱됩니다.

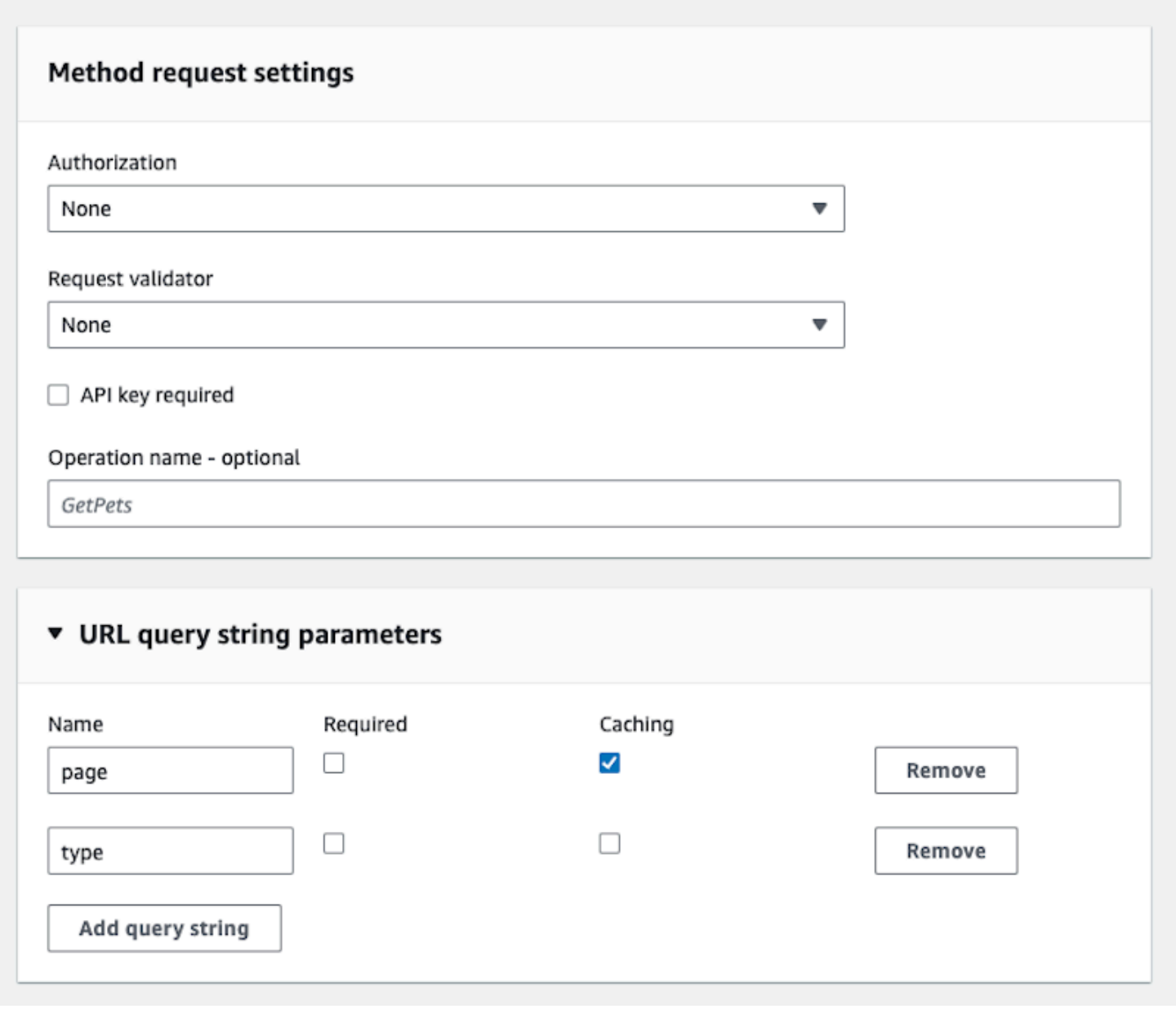
메서드 또는 통합 요청에서 두 개 이상의 파라미터를 취하는 경우 파라미터의 일부 또는 모두를 포함하여 캐시 키를 생성하도록 선택할 수 있습니다. 예를 들어 TTL 기간 내에 나열된 순서로 생성되는 다음 요청에 대해 캐시 키에 `type` 파라미터만 포함할 수 있습니다.

```
GET /users?type=admin&department=A HTTP/1.1
host: example.com
...
```

이 요청의 응답은 캐싱되어 다음 요청을 처리하는 데 사용됩니다.

```
GET /users?type=admin&department=B HTTP/1.1
host: example.com
...
```

메서드 또는 통합 요청 파라미터를 API Gateway 콘솔에서 캐시 키의 일부로 포함하려면 파라미터를 추가한 후 캐싱을 선택합니다.



API Gateway에서 API 단계 캐시 비우기

API 캐싱을 활성화한 경우 API 클라이언트가 통합 엔드포인트에서 최신 응답을 가져오도록 API 단계의 캐시를 비울 수 있습니다.

API 스테이지 캐시를 비우려면 스테이지 작업 메뉴를 선택한 다음 스테이지 캐시 비우기를 선택합니다.

> **Note**
>
> 캐시가 플러시되면 캐시가 다시 빌드될 때까지 통합 엔드포인트에서 응답이 처리됩니다. 이 기간 동안 통합 엔드포인트에 전송되는 요청 수가 증가할 수 있습니다. 이로 인해 API의 전체 지연 시간이 일시적으로 증가할 수 있습니다.

API Gateway 캐시 항목 무효화

API의 클라이언트는 기존 캐시 엔트리를 무효화하고 개별 요청에 대해 통합 엔드포인트에서 캐시 엔트리를 다시 로드할 수 있습니다. 클라이언트는 `Cache-Control: max-age=0` 헤더를 포함하는 요청을 전송해야 합니다. 클라이언트는 캐시 대신 통합 엔드포인트에서 직접 응답을 수신합니다(클라이언트에게 해당 권한이 부여된 경우). 이 경우 기존 캐시 엔트리를 통합 엔드포인트에서 가져오는 새 응답으로 대체합니다.

클라이언트에게 권한을 부여하려면 다음과 같은 형식의 정책을 사용자에 대한 IAM 실행 역할에 추가합니다.

> **Note**
>
> 교차 계정 캐시 무효화는 지원되지 않습니다.

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

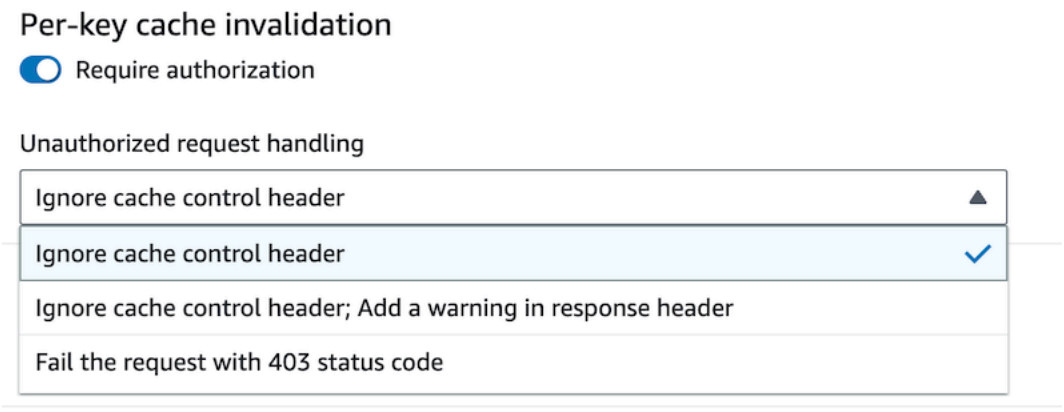
```
      ],
      "Resource": [
        "arn:aws:execute-api:region:account-id:api-id/stage-name/GET/resource-path-specifier"
      ]
    }
  ]
}
```

이 정책은 API Gateway 실행 서비스에서 지정된 리소스 요청에 대한 캐시를 무효화하도록 허용합니다. 타겟 리소스 그룹을 지정하려면 `account-id`, `api-id` 및 `Resource` ARN 값의 기타 엔트리에 대해 와일드카드(*) 문자를 사용합니다. API Gateway 실행 서비스에 대한 권한을 설정하는 방법에 대한 자세한 내용은 IAM 권한을 사용하여 API에 대한 액세스 제어 단원을 참조하세요.

`InvalidateCache` 정책을 적용하지 않은 경우(또는 콘솔에서 Require authorization(권한 부여 필요) 확인란을 선택한 경우), 모든 클라이언트가 API 캐시를 무효화할 수 있습니다. 대부분의 클라이언트가 API 캐시를 무효화할 경우 API의 지연 시간이 일시적으로 증가할 수 있습니다.

정책이 적용되면 캐싱이 활성화되어 권한 부여가 필요합니다.

API Gateway 콘솔의 무단 요청 처리에서 옵션을 선택하여 무단 요청이 처리되는 방식을 제어할 수 있습니다.



세 옵션의 동작은 다음과 같습니다.

- 403 상태 코드와 함께 요청 실패: 403 권한 없음 응답을 반환합니다.

  API를 사용하여 이 옵션을 설정하려면 `FAIL_WITH_403`을 사용합니다.
- 캐시 컨트롤 헤더 무시, 응답 헤더에 경고 추가: 요청을 처리하고 응답에 경고 헤더를 추가합니다.

API를 사용하여 이 옵션을 설정하려면 SUCCEED_WITH_RESPONSE_HEADER을 사용합니다.

- 캐시 컨트롤 헤더 무시: 요청을 처리하고 응답에 경고 헤더를 추가하지 않습니다.

  API를 사용하여 이 옵션을 설정하려면 SUCCEED_WITHOUT_RESPONSE_HEADER을 사용합니다.

API에 대한 페이로드 압축 활성화

API Gateway를 통해 클라이언트는 지원되는 콘텐츠 코딩 중 하나를 사용하여 압축된 페이로드로 API를 호출할 수 있습니다. 기본적으로 API Gateway는 메서드 요청 페이로드의 압축 해제를 지원합니다. 하지만 메서드 응답 페이로드의 압축을 활성화하도록 API를 구성해야 합니다.

API에서 압축을 활성화하려면 API를 생성할 때나 API를 생성한 이후에 minimumCompressionsSize 속성을 0에서 10485760(1천만 바이트) 사이의 음이 아닌 정수로 설정하세요. API에서 압축을 비활성화하려면 minimumCompressionSize를 null로 설정하거나 제거하십시오. API Gateway 콘솔, AWS CLI 또는 API Gateway REST API를 사용하여 API에서 압축을 활성화하거나 비활성화할 수 있습니다.

크기에 상관없이 페이로드에서 압축을 적용하려면 minimumCompressionSize 값을 0으로 설정하십시오. 하지만 크기가 작은 데이터를 압축하면 실제로 최종 데이터 크기가 늘어날 수도 있습니다. 뿐만 아니라 API Gateway에서의 압축과 클라이언트에서의 압축 해제로 전체 지연 시간과 필요한 컴퓨팅 시간이 늘어날 수 있습니다. API에 대해 테스트 케이스를 실행하여 최적의 값을 결정해야 합니다.

클라이언트가 압축된 페이로드와 적절한 Content-Encoding 헤더와 함께 API 요청을 제출하면 API Gateway는 해당 요청을 통합 엔드포인트에 전달하기 전에 관련 매핑 템플릿을 압축 해제하고 적용할 수 있습니다. 압축이 활성화되고 API가 배포된 후 메서드 요청에서 적절한 Accept-Encoding 헤더가 지정된 경우, 클라이언트는 압축된 페이로드가 포함된 API 응답을 수신할 수 있습니다.

통합 엔드포인트가 압축되지 않은 JSON 페이로드를 예상하고 반환하면 압축되지 않은 JSON 페이로드를 위해 구성된 모든 매핑 템플릿을 압축된 페이로드에 적용할 수 있습니다. 압축된 메서드 요청 페이로드의 경우, API Gateway는 페이로드 압축을 해제하고 매핑 템플릿을 적용하며 매핑된 요청을 통합 엔드포인트에 전달합니다. 압축되지 않은 통합 응답 페이로드의 경우, API Gateway는 매핑 템플릿을 적용하고 매핑된 페이로드를 압축해 압축된 페이로드를 클라이언트에게 반환합니다.

주제

- API에서 페이로드 압축 활성화
- 압축된 페이로드가 포함된 API 메서드 호출
- 압축된 페이로드가 포함된 API 응답 수신

API에서 페이로드 압축 활성화

API Gateway 콘솔, AWS CLI 또는 AWS SDK를 사용하여 API에서 압축을 활성화할 수 있습니다.

기존 API에서는 변경 내용이 효과를 발휘하려면 압축을 활성화한 후 API를 배포해야 합니다. 새 API의 경우, API 설정이 완료된 후 API를 배포할 수 있습니다.

> **Note**
>
> 우선 순위가 가장 높은 콘텐츠 인코딩은 API Gateway에 의해 지원되는 인코딩이어야 합니다. 그렇지 않은 경우 압축은 응답 페이로드에 적용되지 않습니다.

주제

- API Gateway 콘솔을 사용하여 API에서 페이로드 압축 활성화
- AWS CLI을 사용해 API에서 페이로드 압축 활성화
- API Gateway가 지원하는 콘텐츠 코딩

API Gateway 콘솔을 사용하여 API에서 페이로드 압축 활성화

다음 절차는 API에서 페이로드 압축을 활성화하는 방법을 설명합니다.

API Gateway 콘솔을 사용하여 페이로드 압축을 활성화하려면

1. https://console.aws.amazon.com/apigateway에서 API Gateway 콘솔에 로그인합니다.
2. 기존 API를 선택하거나 새 API를 생성합니다.
3. 기본 탐색 창에서 API 설정을 선택합니다.
4. API 세부 정보 섹션에서 편집을 선택합니다.
5. 콘텐츠 인코딩을 켜서 페이로드 압축을 활성화합니다. 최소 본문 크기 옆에 최소 압축 크기(바이트)에 해당하는 숫자를 입력합니다. 압축을 해제하려면 콘텐츠 인코딩 옵션을 끕니다.
6. 변경 사항 저장을 선택합니다.

AWS CLI을 사용해 API에서 페이로드 압축 활성화

AWS CLI를 사용하여 새 API를 생성하고 압축을 활성화하려면 다음과 같이 `create-rest-api` 명령을 호출하십시오.

```
aws apigateway create-rest-api \
    --name "My test API" \
    --minimum-compression-size 0
```

AWS CLI를 사용하여 기존 API에서 압축을 활성화하려면 다음과 같이 update-rest-api 명령을 호출하십시오.

```
aws apigateway update-rest-api \
    --rest-api-id 1234567890 \
    --patch-operations op=replace,path=/minimumCompressionSize,value=0
```

minimumCompressionSize 속성의 음수가 아닌 정수 값은 0에서 10485760(10M바이트) 사이입니다. 압축 임계값을 측정합니다. 페이로드 크기가 이 값보다 작으면 페이로드에 압축 또는 압축 해제가 적용되지 않습니다. 이 값을 0으로 설정하면 모든 페이로드 크기에 압축이 허용됩니다.

AWS CLI를 사용하여 압축을 비활성화하려면 다음과 같이 update-rest-api 명령을 호출하십시오.

```
aws apigateway update-rest-api \
    --rest-api-id 1234567890 \
    --patch-operations op=replace,path=/minimumCompressionSize,value=
```

이전 호출에서 value를 빈 문자열 ""로 설정하거나 value 속성을 모두 생략할 수도 있습니다.

API Gateway가 지원하는 콘텐츠 코딩

API Gateway가 지원하는 콘텐츠 코딩은 다음과 같습니다.

- deflate
- gzip
- identity

RFC 7231 사양에 따르면 API Gateway는 다음의 Accept-Encoding 헤더 형식도 지원합니다.

- 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`

압축된 페이로드가 포함된 API 메서드 호출

압축된 페이로드가 포함된 API 요청을 하려면 클라이언트는 지원되는 콘텐츠 코딩 중 하나를 사용하여 `Content-Encoding` 헤더를 설정해야 합니다.

API 클라이언트이고 PetStore API 메서드(`POST /pets`)를 호출하려 한다고 가정해 보십시오. 다음의 JSON 출력을 사용하여 메서드를 호출하지 마십시오.

```
POST /pets
Host: {petstore-api-id}.execute-api.{region}.amazonaws.com
Content-Length: ...

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

그 대신 GZIP 코딩을 사용하여 압축된 동일한 페이로드가 포함된 메서드를 호출할 수 있습니다.

```
POST /pets
Host: {petstore-api-id}.execute-api.{region}.amazonaws.com
Content-Encoding:gzip
Content-Length: ...

[binary]
```

API Gateway는 요청을 수신하면 지정된 콘텐츠 코딩이 지원되는지 확인합니다. 그런 다음 지정된 콘텐츠 코딩으로 페이로드 압축을 해제하려 시도합니다. 압축 해제에 성공하면 통합 엔드포인트에 요청을 발송합니다. 지정된 코딩이 지원되지 않거나 제공된 페이로드가 지정된 코딩으로 압축되지 않은 경우, API Gateway는 `415 Unsupported Media Type` 오류 응답을 반환합니다. API 및 스테이지가 식별되기 전에 압축 해제의 초기 단계에서 오류가 발생한 경우, 오류가 CloudWatch Logs에 기록되지 않습니다.

압축된 페이로드가 포함된 API 응답 수신

압축이 활성화된 API에서 요청을 할 때 클라이언트는 지원되는 콘텐츠 코딩으로 `Accept-Encoding` 헤더를 지정하여 특정 형식의 압축된 응답 페이로드를 수신할 수 있습니다.

API Gateway는 다음 조건이 충족되는 경우에만 응답 페이로드를 압축합니다.

- 수신되는 요청에는 지원되는 코딩 및 형식의 `Accept-Encoding` 헤더가 있습니다.

> **Note**
>
> 헤더가 설정되지 않은 경우, 기본값은 RFC 7231에 정의된 대로 *입니다. 이러한 경우 API Gateway는 페이로드를 압축하지 않습니다. 일부 브라우저 또는 클라이언트는 압축이 활성화된 요청에 `Accept-Encoding`(예: `Accept-Encoding:gzip, deflate, br`)을 자동으로 추가할 수 있습니다. 그러면 API Gateway에서 페이로드 압축이 트리거될 수 있습니다. 지원되는 `Accept-Encoding` 헤더 값을 명시적으로 지정하지 않으면 API Gateway가 페이로드를 압축하지 않습니다.

- API에서 `minimumCompressionSize`가 설정되어 압축을 활성화합니다.
- 통합 응답에는 `Content-Encoding` 헤더가 없습니다.
- 관련 매핑 템플릿이 적용된 후의 통합 응답 페이로드의 크기는 지정된 `minimumCompressionSize` 값보다 크거나 같습니다.

API Gateway는 페이로드를 압축하기 전에 통합 응답을 위해 구성된 모든 매핑 템플릿을 적용합니다. 통합 응답에 `Content-Encoding` 헤더가 포함된 경우, API Gateway는 합 응답 페이로드가 이미 압축되어 있다고 가정하고 압축 처리를 건너뜁니다.

한 가지 예는 PetStore API 예와 다음의 요청입니다.

```
GET /pets
Host: {petstore-api-id}.execute-api.{region}.amazonaws.com
Accept: application/json
```

백엔드는 다음과 비슷한 압축되지 않은 JSON 페이로드가 포함된 요청에 응답합니다.

```
200 OK

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

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

API 클라이언트는 이 출력을 압축된 페이로드로 수신하기 위해 다음과 같이 요청을 제출할 수 있습니다.

```
GET /pets
Host: {petstore-api-id}.execute-api.{region}.amazonaws.com
Accept-Encoding:gzip
```

클라이언트는 다음과 비슷한 `Content-Encoding` 헤더 및 GZIP 인코딩된 페이로드가 포함된 응답을 수신합니다.

```
200 OK
Content-Encoding:gzip
...

◆◆◆RP◆

J◆)JV
◆:P^IeA*◆◆◆◆◆◆+(◆L ◆X◆YZ◆ku0L0B7!9◆◆C#◆&◆◆◆◆Y◆◆a◆◆◆^◆X
```

응답 페이로드가 압축되면 데이터 전송 요금은 압축된 데이터 크기에 대해서만 청구됩니다.

클라이언트에 REST API 배포

이 섹션에서는 고객에게 API Gateway API를 배포하는 방법에 대해 자세히 설명합니다. API 배포에는 고객이 클라이언트 애플리케이션을 다운로드 및 통합하도록 SDK를 생성하고, 고객이 클라이언트 애플리케이션에서 호출하는 방법을 알 수 있도록 API를 문서화하며, API를 제품 및 서비스의 일부로 사용할 수 있게 만드는 작업이 포함됩니다.

주제

- API 키를 사용하여 사용량 계획 생성 및 사용
- REST API 문서화
- API Gateway에서 REST API용 SDK 생성
- 를 통해 API Gateway API 판매AWS Marketplace

API 키를 사용하여 사용량 계획 생성 및 사용

API를 생성, 테스트 및 배포한 후 API Gateway 사용량 계획을 사용하여 API를 고객을 위한 제품 혜택으로 제공할 수 있습니다. 고객이 선택한 API에 액세스할 수 있도록 사용 계획 및 API 키를 구성하고 정의된 제한 및 할당량에 따라 해당 API에 대한 요청 제한을 시작할 수 있습니다. API 또는 API 메서드 수준에서 설정할 수 있습니다.

사용량 계획과 API 키란 무엇인가?

사용 계획은 배포된 하나 이상의 API 단계 및 메서드에 액세스할 수 있는 사용자를 지정하고 선택적으로 요청 제한을 시작하도록 대상 요청 속도를 설정합니다. 계획은 API 키를 사용하여 API 클라이언트와 각 키에 대해 연결된 API 단계에 액세스할 수 있는 사용자를 식별합니다.

API 키는 앱 개발자 고객에게 API 액세스 권한을 부여하기 위해 배포하는 영숫자 문자열 값입니다. API 키를 Lambda 권한 부여자, IAM 역할 또는 Amazon Cognito와 함께 사용하여 API에 대한 액세스를 제어할 수 있습니다. API Gateway가 사용자 대신 API 키를 생성하거나, 사용자가 CSV 파일에서 API 키를 가져올 수 있습니다. API Gateway에서 API 키를 생성하거나, 외부 소스에서 API Gateway로 API 키를 가져올 수 있습니다. 자세한 내용은 the section called “API Gateway 콘솔을 사용하여 API 키 설정” 단원을 참조하세요.

API 키는 이름과 값을 갖습니다. ("API 키"와 "API 키 값"은 종종 서로 바꿔 사용됩니다.) 이름은 1,024자를 초과할 수 없습니다. 값은 20~128자의 영숫자 문자열입니다(예: `apikey1234abcdefghij0123456789`).

> **Important**
>
> API 키 값은 고유해야 합니다. 이름은 다르고 값은 동일한 두 개의 API 키를 생성하려고 하면 API Gateway는 이 두 키를 동일한 API 키로 인식합니다.
> API 키는 하나 이상의 사용량 계획과 연결할 수 있습니다. 사용량 계획은 하나 이상의 단계와 연결할 수 있습니다. 그러나 지정된 API 키는 API의 각 단계에 대하여 단 하나의 사용량 계획에만 연결할 수 있습니다.

제한 한도는 요청 제한이 시작되어야 하는 대상 지점을 설정합니다. 이는 API 또는 API 메서드 수준에서 설정할 수 있습니다.

할당량 한도는 지정된 시간 간격 내에 제출할 수 있는 지정된 API 키로 최대 요청 수를 설정합니다. 사용량 계획 구성에 따라 API 키 권한 부여를 요구하도록 개별 API 메서드를 구성할 수 있습니다.

조절 및 할당량 한도는 한 가지 사용량 계획의 모든 API 단계에서 집계된 요청의 개별 API 키에 적용됩니다.

> **Note**
>
> 사용량 계획 조절 및 할당량은 엄격한 제한이 아니며 최선으로 적용됩니다. 경우에 따라 클라이언트가 설정한 할당량을 초과할 수 있습니다. 비용을 제어하거나 API에 대한 액세스를 차단하기 위해 사용량 계획 할당량 또는 제한에 의존하지 마십시오. 비용을 모니터링하는 데 AWS Budgets를 사용하고 API 요청을 관리하는 데 AWS WAF 를 사용하는 것이 좋습니다.

API 키 및 사용량 계획의 위한 모범 사례

다음은 API 키 및 사용량 계획을 사용할 때 따를 수 있는 모범 사례입니다.

> **Important**
>
> - API에 대한 액세스를 제어하기 위해 인증 또는 권한 부여에 API 키를 사용하지 않도록 합니다. 사용 계획에 API가 여러 개 있는 경우, 해당 사용 계획의 한 API에 대해 유효한 API 키가 있는 사용자는 해당 사용 계획의 모든 API에 액세스할 수 있습니다. 대신, API에 대한 액세스를 제어하려면 IAM 역할, Lambda 권한 부여자 또는 Amazon Cognito 사용자 풀을 사용합니다.
> - API 게이트웨이에서 생성되는 API 키를 사용합니다. API 키에는 기밀 정보가 포함되어서는 안 됩니다. 클라이언트에서는 일반적으로 로깅될 수 있는 헤더에 이를 전송합니다.

- 개발자 포털을 사용하여 API를 게시할 경우 다른 API를 고객에게 공개하지 않았더라도 해당 사용량 계획의 모든 API가 구독 가능해집니다.
- 경우에 따라 클라이언트가 설정한 할당량을 초과할 수 있습니다. 비용의 제어를 위해 사용 계획에 의존하지 마십시오. 비용을 모니터링하는 데 AWS Budgets를 사용하고 API 요청을 관리하는 데 AWS WAF를 사용하는 것이 좋습니다.
- 사용량 계획에 API 키를 추가한 후 업데이트 작업을 완료하는 데 몇 분 정도 걸릴 수 있습니다.

사용량 계획의 구성 단계

다음 단계에서는 API 소유자가 고객을 위해 사용량 계획을 어떻게 생성하고 구성하는지 설명합니다.

사용량 계획을 구성하려면

1. 하나 이상의 API를 생성하고 API 키가 필요한 메서드를 구성한 후 단계에 API를 배포합니다.
2. API를 사용할 앱 개발자(고객)에게 배포할 API 키를 생성하거나 가져옵니다.
3. 원하는 스로틀과 할당량 제한을 이용해 사용량 계획을 생성합니다.
4. API 단계와 API 키를 사용량 계획에 연결합니다.

API 호출자는 API에 대한 요청에서 `x-api-key` 헤더에 할당된 API 키를 제공해야 합니다.

> **Note**
>
> 사용량 계획에 API 메서드를 포함하려면 개별 API 메서드에 API 키가 필요하도록 구성해야 합니다. 고려해야 할 모범 사례는 the section called “API 키 및 사용량 계획의 위한 모범 사례” 섹션을 참조하세요.

API 키 소스 선택

사용량 계획을 API와 연결하고 API 메서드에서 API 키를 활성화할 때 API에 수신되는 모든 요청에는 API 키가 포함되어야 합니다. API Gateway는 이 키를 읽고 사용량 계획에 있는 키와 비교합니다. 일치하는 항목이 있으면 API Gateway는 계획의 요청 한도와 할당량에 따라 요청을 제한합니다. 그렇지 않으면 `InvalidKeyParameter` 예외가 발생합니다. 그 결과, 호출자는 `403 Forbidden` 응답을 수신합니다.

API Gateway API는 다음 둘 중 하나의 소스에서 API 키를 수신할 수 있습니다.

`HEADER`

고객에게 API 키를 배포하고 수신되는 각 요청의 `X-API-Key` 헤더로서 이 API 키를 전달하도록 요청합니다.

`AUTHORIZER`

Lambda 권한 부여자가 권한 부여 응답의 일부로 API 키를 반환하도록 할 수 있습니다. 권한 부여 응답에 대한 자세한 내용은 the section called “Amazon API Gateway Lambda 권한 부여자의 출력” 단원을 참조하세요.

> **Note**
>
> 고려해야 할 모범 사례는 the section called “API 키 및 사용량 계획의 위한 모범 사례” 섹션을 참조하세요.

API Gateway 콘솔을 사용하여 API의 API 키 소스를 선택하려면

1. API Gateway 콘솔에 로그인합니다.
2. 기존 API를 선택하거나 새 API를 생성합니다.
3. 기본 탐색 창에서 API 설정을 선택합니다.
4. API 세부 정보 섹션에서 편집을 선택합니다.
5. API 키 소스의 드롭다운 목록에서 `Header` 또는 `Authorizer`를 선택합니다.
6. 변경 사항 저장를 선택합니다.

를 사용하여 API의 API 키 소스를 선택하려면 다음과 같이 `update-rest-api`명령을 호출합니다. AWS CLI

```
aws apigateway update-rest-api --rest-api-id 1234123412 --patch-operations
 op=replace,path=/apiKeySource,value=AUTHORIZER
```

클라이언트가 API 키를 제출하도록 하려면 이전의 CLI 명령에서 `value`를 `HEADER`로 설정합니다.

API Gateway REST API를 사용하여 API의 API 키 소스를 선택하려면 다음과 같이 `restapi:update`를 호출합니다.

```
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"
    }
  ]
}
```

권한 부여자가 API 키를 반환하도록 하려면 앞의 `value` 입력에서 `AUTHORIZER`를 `patchOperations`로 설정합니다.

선택하는 API 키 원본 유형에 따라 다음 절차 중 하나를 사용하여 메서드 호출에서 헤더로 제공된 API 키 또는 권한 부여자가 반환한 API 키를 사용합니다.

헤더로 제공된 API 키를 사용하려면

1. 원하는 API 메서드를 사용하여 API를 생성한 다음, API를 스테이지에 배포합니다.
2. 새 사용량 계획을 만들거나 기존 사용량 계획을 선택합니다. 사용량 계획에 배포한 API 단계를 추가합니다. 사용량 계획에 API 키를 연결하거나 계획에서 기존 API 키를 선택합니다. 선택한 API 키 값을 적어 둡니다.
3. API 키를 요구하도록 API 메서드를 설정합니다.
4. 동일한 단계에 API를 다시 배포합니다. 새 단계에 API를 배포하는 경우 새 API 단계를 연결하도록 사용량 계획을 업데이트해야 합니다.

이제 클라이언트가 `x-api-key` 헤더에 선택한 API 키를 헤더 값으로 제공하여 API 메서드를 호출할 수 있습니다.

권한 부여자가 제공한 API 키를 사용하려면

1. 원하는 API 메서드를 사용하여 API를 생성한 다음, API를 스테이지에 배포합니다.
2. 새 사용량 계획을 만들거나 기존 사용량 계획을 선택합니다. 사용량 계획에 배포한 API 단계를 추가합니다. 사용량 계획에 API 키를 연결하거나 계획에서 기존 API 키를 선택합니다. 선택한 API 키 값을 적어 둡니다.
3. 토큰 기반 Lambda 권한 부여자를 생성합니다. 권한 부여 응답의 루트 수준 속성으로 `usageIdentifierKey:`*`{api-key}`*를 포함시킵니다.
4. API 키를 요구하도록 API 메서드를 설정하고, 메서드에서 Lambda 권한 부여자를 활성화합니다.
5. 동일한 단계에 API를 다시 배포합니다. 새 단계에 API를 배포하는 경우 새 API 단계를 연결하도록 사용량 계획을 업데이트해야 합니다.

이제 클라이언트가 API 키를 명시적으로 제공하지 않고도 API 키 요구 메서드를 호출할 수 있습니다. 권한 부여자가 반환한 API 키가 자동으로 사용됩니다.

API Gateway 콘솔을 사용하여 API 키 설정

API 키를 설정하려면 다음 작업을 수행합니다.

- API 메서드를 구성해 API 키를 요구합니다.
- 리전의 API에 대해 API 키를 생성하거나 가져옵니다.

API 키를 설정하기 전에 API를 생성하여 단계에 배포해야 합니다. API 키 값을 생성한 후에는 변경할 수 없습니다.

API Gateway 콘솔을 사용하여 API를 생성하고 배포하는 방법에 대한 지침은 각각 Amazon API Gateway에서 REST API 생성 및 Amazon API Gateway에서 REST API 배포 단원을 참조하세요.

API 키를 생성한 후에는 이를 사용량 계획에 연결해야 합니다. 자세한 설명은 API Gateway 콘솔을 사용하여 사용량 계획 생성, 구성 및 테스트 섹션을 참조하세요.

> **Note**
>
> 고려해야 할 모범 사례는 the section called “API 키 및 사용량 계획의 위한 모범 사례” 섹션을 참조하세요.

주제

- 메서드에서 API 키 요구
- API 키 생성
- API 키 가져오기

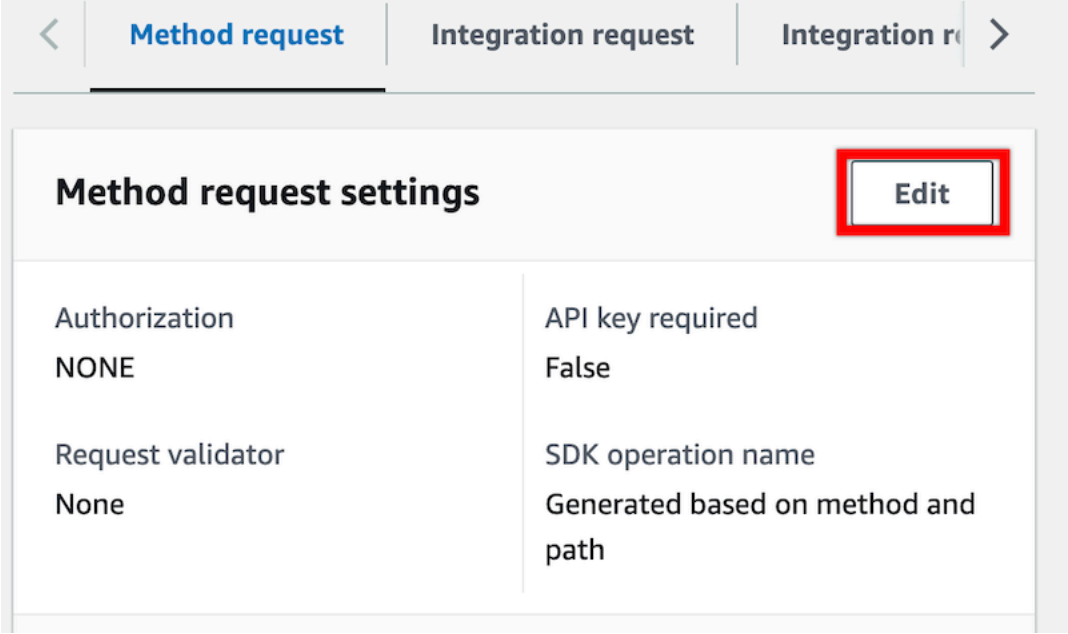
메서드에서 API 키 요구

다음 절차에서는 API 키를 요구하도록 API 메서드를 구성하는 방법에 대해 설명합니다.

API 메서드를 구성해 API 키를 요구하려면

1. https://console.aws.amazon.com/apigateway에서 API Gateway 콘솔에 로그인합니다.

2. REST API를 선택합니다.
3. API Gateway 기본 탐색 창에서 리소스를 선택합니다.
4. 리소스에서 새 메서드를 생성하거나 기존 메서드를 선택합니다.
5. 메서드 요청 탭의 메서드 요청 설정에서 편집을 선택합니다.



6. API 키가 필요함을 선택합니다.
7. 저장을 선택합니다.
8. API를 배포하거나 다시 배포하여 요구 사항을 적용합니다.

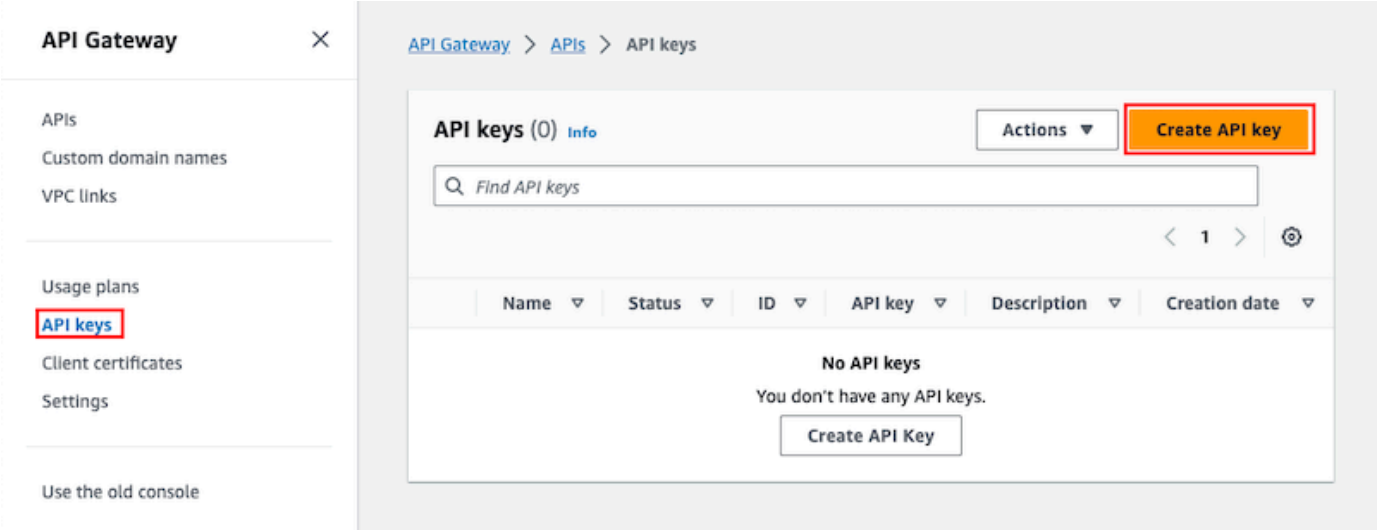
API 키가 필요함 옵션이 `false`로 설정되어 있고 이전 단계를 실행하지 않는 경우, API 단계와 연결된 어떤 API 키도 해당 메서드에 사용되지 않습니다.

API 키 생성

사용량 계획에 사용하기 위해 이미 API키를 생성했거나 가져온 경우, 이 절차와 다음 절차를 건너뛰십시오.

API 키를 생성하려면

1. https://console.aws.amazon.com/apigateway에서 API Gateway 콘솔에 로그인합니다.
2. REST API를 선택합니다.
3. API Gateway 기본 탐색 창에서 API 키를 선택합니다.
4. API 키 생성을 선택합니다.



5. 이름에 이름을 입력합니다.
6. (선택 사항) 설명에 설명을 입력합니다.
7. API 키의 경우 자동 생성을 선택하여 API Gateway가 키 값을 생성하도록 하거나 사용자 지정을 선택하여 자체 키 값을 생성합니다.
8. 저장을 선택합니다.

API 키 가져오기

다음 절차에서는 사용량 계획에 사용할 API 키를 가져오는 방법을 설명합니다.

API 키를 가져오려면

1. https://console.aws.amazon.com/apigateway에서 API Gateway 콘솔에 로그인합니다.
2. REST API를 선택합니다.
3. 기본 탐색 창에서 API 키를 선택합니다.
4. 작업 드롭다운 메뉴를 선택한 다음 API 키 가져오기를 선택합니다.
5. 쉼표로 구분된 키 파일을 로드하려면 파일 선택을 선택합니다. 텍스트 편집기에 키를 입력할 수도 있습니다. 파일 형식에 대한 자세한 내용은 the section called “API Gateway API 키 파일 형식”을 참조하세요.
6. 경고 실패를 선택해 오류 발생 시 가져오기를 중단하거나 경고 무시를 선택해 오류 발생 시에도 유효한 키 항목을 계속 가져옵니다.
7. API 키를 가져오려면 가져오기를 선택합니다.

API Gateway 콘솔을 사용하여 사용량 계획 생성, 구성 및 테스트

사용량 계획을 생성하기 전에 원하는 API 키를 설정했는지 확인합니다. 자세한 내용은 API Gateway 콘솔을 사용하여 API 키 설정 단원을 참조하세요.

이 단원에서는 API Gateway 콘솔을 사용하여 사용량 계획을 생성하고 사용하는 방법을 설명합니다.

주제

- API를 기본 사용량 계획으로 마이그레이션(필요한 경우)
- 사용량 계획 생성
- 사용량 계획 테스트
- 사용량 계획 유지 관리

API를 기본 사용량 계획으로 마이그레이션(필요한 경우)

사용량 계획 기능이 출시된 2016년 8월 11일 이후에 API Gateway를 사용하기 시작한 경우, 지원되는 모든 리전에서 사용량 계획이 자동으로 활성화됩니다.

이 날짜 이전에 API Gateway를 사용하기 시작한 경우 기본 사용량 계획으로 마이그레이션해야 할 수 있습니다. 선택한 리전에서 처음으로 사용량 계획을 사용하기 전에 Enable Usage Plans(사용량 계획 활성화) 옵션이 메시지로 표시됩니다. 이 옵션을 활성화하면 기존 API 키에 연결된 모든 고유 API 단계에 대해 기본 사용량 계획이 생성됩니다. 조절 및 할당량 한도가 초기에 설정되어 있지 않은 기본 사용량 계획에서는 API 키와 API 단계 간의 연결이 사용량 계획에 복사됩니다. API는 전과 동일하게 동작합니다. 하지만 속성을 사용하는 대신 이 `UsagePlan`apiStages속성을 사용하여 지정된 API 단계 값(`apiId`및`stage`) 을 포함된 API 키와 연결 (`UsagePlanKey`) 해야 합니다. ApiKey`stageKeys`

기본 사용량 계획으로 이미 마이그레이션했는지 여부를 점검하려면 `get-account` CLI 명령을 사용하세요. 사용량 계획이 활성화되면 명령 출력에서 `features` 목록에 `"UsagePlans"` 항목이 포함됩니다.

다음과 AWS CLI 같이 API를 기본 사용량 계획으로 마이그레이션할 수도 있습니다.

를 사용하여 기본 사용량 요금제로 마이그레이션하려면 AWS CLI

1. CLI 명령 `update-account`를 호출합니다.
2. `cli-input-json` 파라미터에 대해 다음 JSON을 사용합니다.

```
[
```

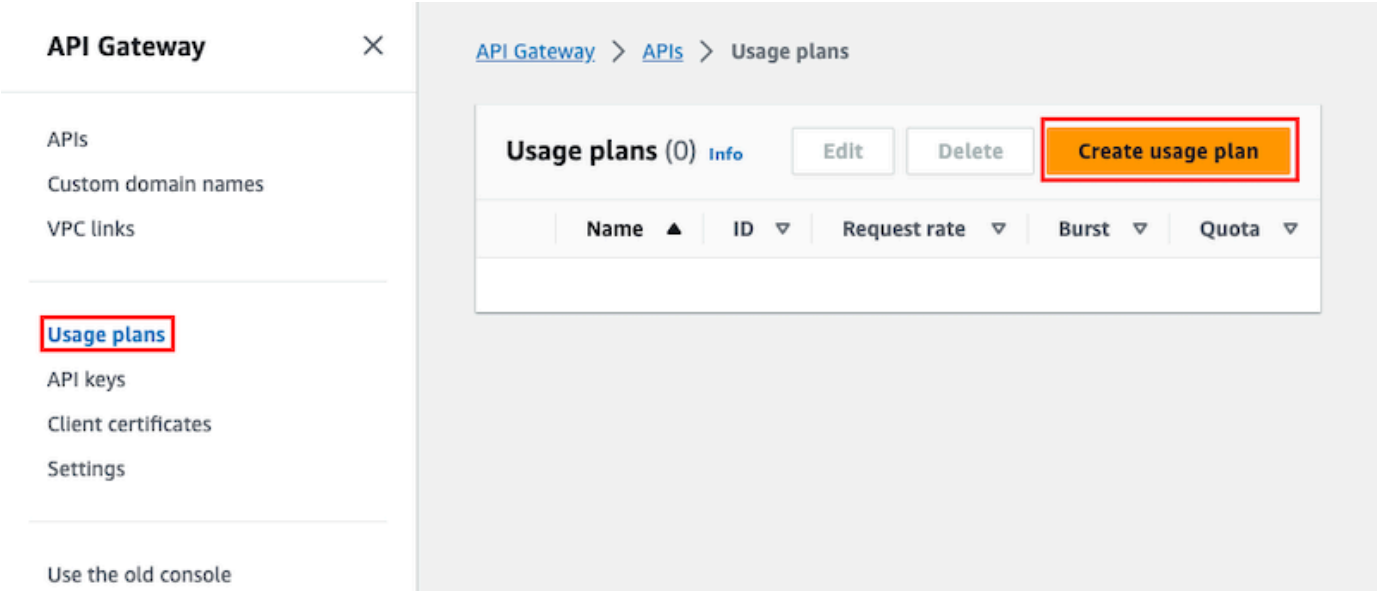
```
    {
        "op": "add",
        "path": "/features",
        "value": "UsagePlans"
    }
]
```

사용량 계획 생성

다음 절차에서는 사용량 계획을 생성하는 방법에 대해 설명합니다.

사용량 계획을 생성하려면

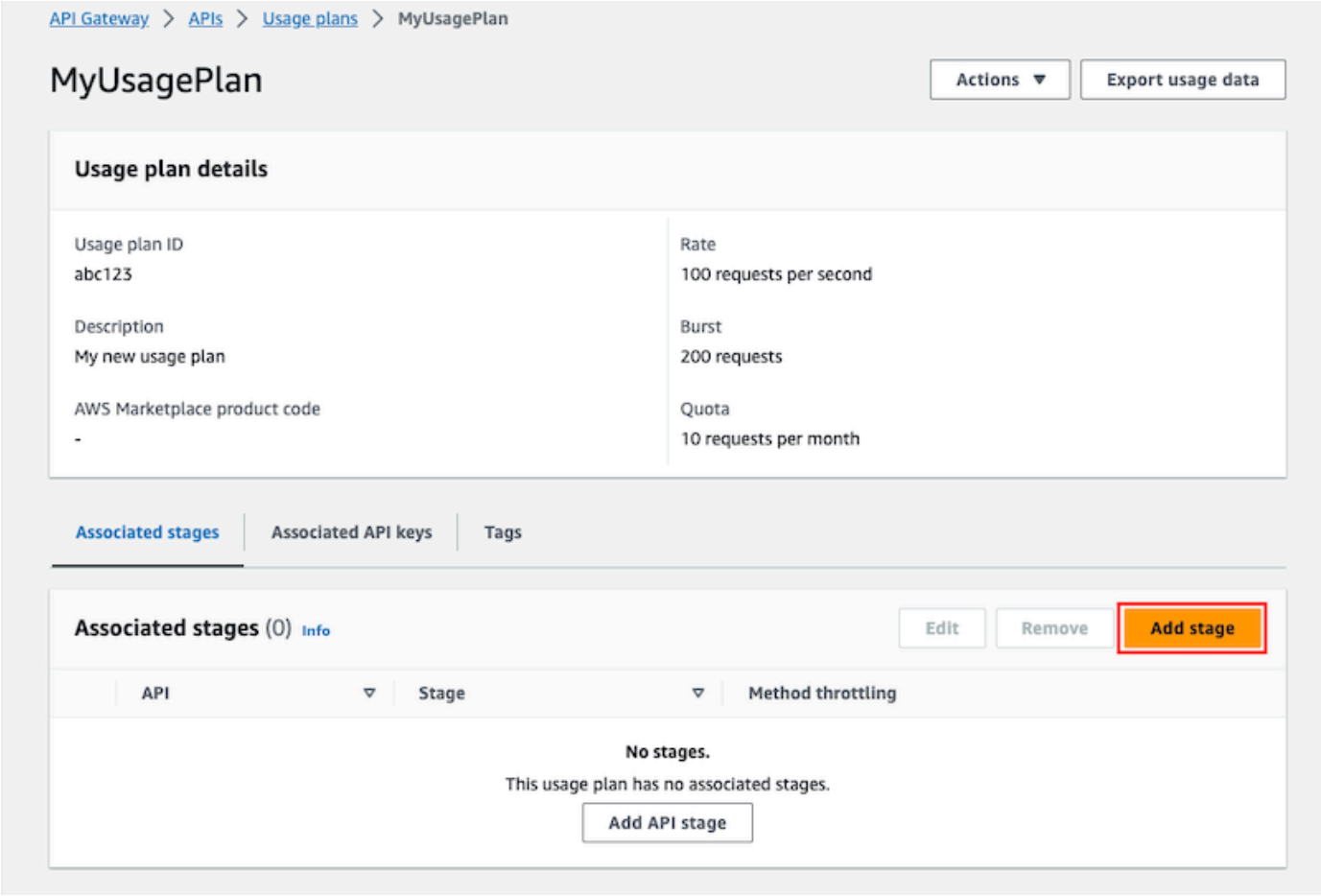
1. https://console.aws.amazon.com/apigateway에서 API Gateway 콘솔에 로그인합니다.
2. API Gateway 기본 탐색 창에서 사용량 계획을 선택한 다음 사용량 계획 생성을 선택합니다.



3. 이름에 이름을 입력합니다.
4. (선택 사항) 설명에 설명을 입력합니다.
5. 기본적으로 사용량 계획에는 제한이 활성화됩니다. 사용량 계획의 요율 및 버스트를 입력합니다. 제한을 해제하려면 조절을 선택합니다.
6. 기본적으로 사용량 계획에서는 일정 기간 할당량이 활성화됩니다. 요청에 사용량 계획 기간 동안 사용자가 할 수 있는 총 요청 수를 입력합니다. 할당량을 해제하려면 할당량을 선택합니다.
7. 사용량 계획 생성을 선택합니다.

사용량 계획에 스테이지를 추가하려면

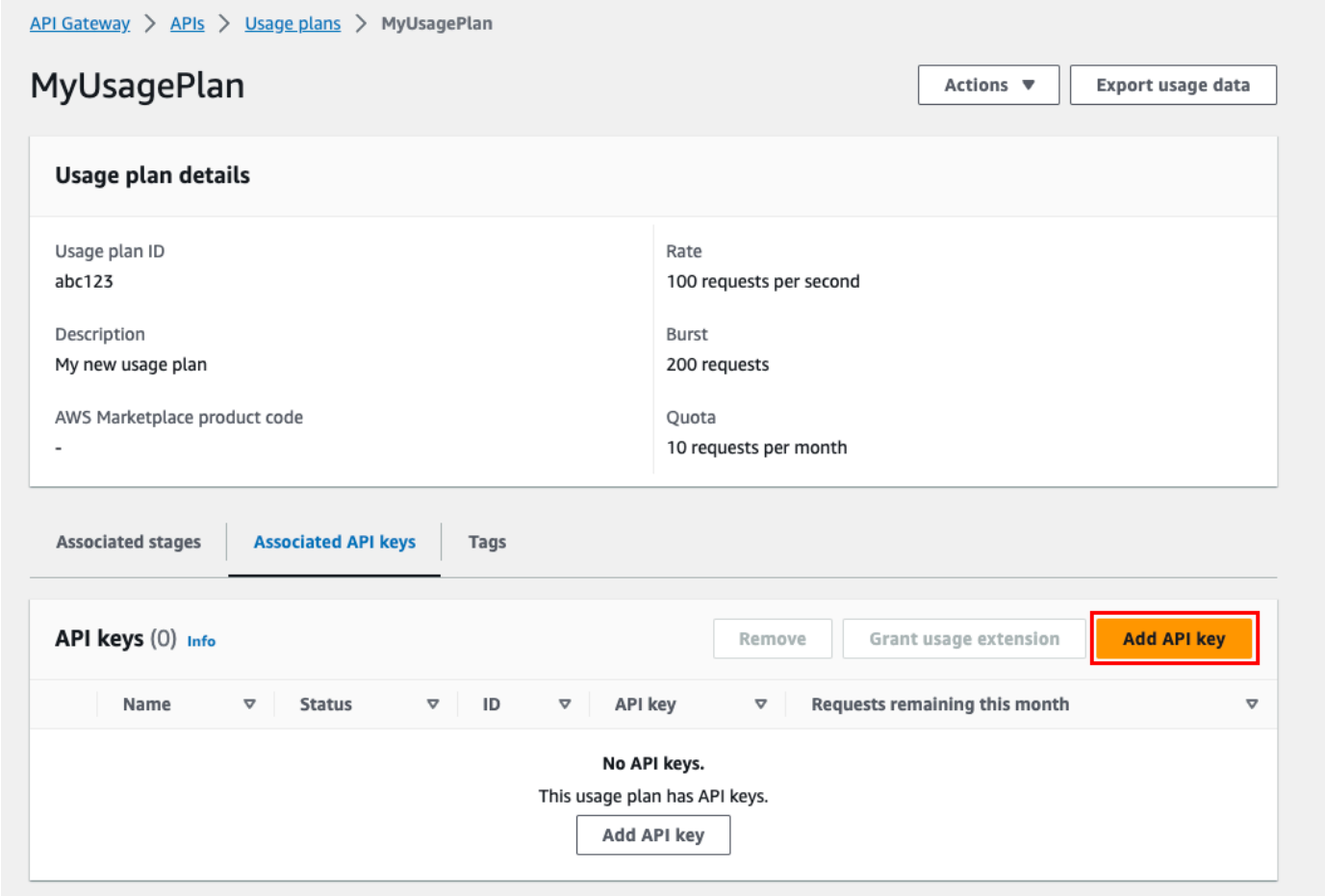
1. 사용량 계획을 선택합니다.
2. 연결된 스테이지 탭에서 스테이지 추가를 선택합니다.



3. API에서 API를 선택합니다.
4. 스테이지에서 스테이지를 선택합니다.
5. (선택 사항) 메서드 수준 제한을 설정하려면 다음을 수행합니다.
   a. 메서드 수준 제한을 선택한 다음 메서즈 추가를 선택합니다.
   b. 리소스에서 API의 리소스를 선택합니다.
   c. 메서드에서 API의 메서드를 선택합니다.
   d. 사용량 계획의 요율 및 버스트를 입력합니다.
6. 사용량 계획 추가를 선택합니다.

사용량 계획에 키를 추가하려면

1. 연결된 API 키 탭에서 API 키 추가를 선택합니다.



2. a. 기존 키를 사용량 계획에 연결하려면 기존 키 추가를 선택한 다음 드롭다운 메뉴에서 기존 키를 선택합니다.

   b. 새 API 키를 생성하려면 새 키 생성 및 추가를 선택한 다음 새 키를 생성합니다. 새 키를 생성하는 방법에 대한 자세한 정보는 API 키 생성 섹션을 참조하세요.

3. API 키 추가를 선택합니다.

사용량 계획 테스트

사용량 계획을 테스트하려면 AWS SDK 또는 Postman과 같은 REST API 클라이언트를 사용할 수 있습니다. AWS CLIPostman을 사용하여 사용량 계획을 테스트하는 예는 사용량 계획 테스트 단원을 참조하세요.

사용량 계획 유지 관리

사용량 계획을 유지 관리하려면 지정된 기간 동안 사용한 할당량과 남은 할당량을 모니터링하고, 필요하면 남은 할당량을 지정된 크기만큼 확장해야 합니다. 다음 절차에서는 할당량을 모니터링하는 방법을 설명합니다.

사용한 할당량과 남은 할당량을 모니터링하려면

1. https://console.aws.amazon.com/apigateway에서 API Gateway 콘솔에 로그인합니다.
2. API Gateway 기본 탐색 창에서 사용량 계획을 선택합니다.
3. 사용량 계획을 선택합니다.
4. 연결된 API 키 탭을 선택하면 각 키에 대해 해당 기간 동안 남아 있는 요청 수를 확인할 수 있습니다.
5. (선택 사항) 사용량 데이터 내보내기를 선택한 다음 시작 날짜 및 종료 날짜를 선택합니다. 그런 다음 내보내는 데이터 형식으로 JSON 또는 CSV를 선택하고 내보내기를 선택합니다.

   다음은 내보낸 파일의 예입니다.

```
{
  "thisPeriod": {
    "px1KW6...qBazOJH": [
      [
        0,
        5000
      ],
      [
        0,
        5000
      ],
      [
        0,
        10
      ]
    ]
  },
  "startDate": "2016-08-01",
  "endDate": "2016-08-03"
}
```

이 예에서 사용량 데이터는 API 키(`px1KW6...qBaz0JH`)로 식별되듯이 2016년 8월 1일부터 2016년 8월 3일 사이의 API 클라이언트의 일일 사용량 데이터를 보여 줍니다. 일일 사용량 데이터는 사용한 할당량과 남은 할당량을 보여줍니다. 이 예에서 구독자는 할당량을 아직 사용하지 않았고, API 소유자 또는 관리자가 셋째 날에 남은 할당량을 5000에서 10으로 줄였습니다.

다음 절차에서는 할당량을 수정하는 방법을 설명합니다.

남은 할당량을 확장하려면

1. https://console.aws.amazon.com/apigateway에서 API Gateway 콘솔에 로그인합니다.
2. API Gateway 기본 탐색 창에서 사용량 계획을 선택합니다.
3. 사용량 계획을 선택합니다.
4. 연결된 API 키 탭을 선택하면 각 키에 대해 해당 기간 동안 남아 있는 요청 수를 확인할 수 있습니다.
5. API 키를 선택한 다음 사용량 확장 부여를 선택합니다.
6. 남은 요청 할당량에 숫자를 입력합니다. 사용량 계획 기간 동안 남은 요청을 늘리거나 줄일 수 있습니다.
7. 업데이트 할당량을 선택합니다.

API Gateway REST API를 사용하여 API 키 설정

API 키를 설정하려면 다음 작업을 수행합니다.

- API 메서드를 구성해 API 키를 요구합니다.
- 리전의 API에 대해 API 키를 생성하거나 가져옵니다.

API 키를 설정하기 전에 API를 생성하여 단계에 배포해야 합니다. API 키 값을 일단 생성하면 이후에 변경할 수 없습니다.

API를 생성하고 배포하기 위한 REST API 호출의 경우 각각 `restapi:create` 및 `deployment:create`를 참조하세요.

> **Note**
>
> 고려해야 할 모범 사례는 the section called “API 키 및 사용량 계획의 위한 모범 사례” 섹션을 참조하세요.

주제

- 메서드에서 API 키 한 개 요구
- API 키 생성 또는 가져오기

메서드에서 API 키 한 개 요구

메서드에서 API 키를 요구하려면 다음 중 하나를 실시합니다.

- `method:put`을 호출하여 메서드를 생성합니다. 요청 페이로드에서 `apiKeyRequired`를 `true`로 설정합니다.
- `method:update`를 호출하여 `apiKeyRequired`를 `true`로 설정합니다.

API 키 생성 또는 가져오기

API 키를 생성하거나 가져오려면 다음 중 하나를 실시합니다.

- `apikey:create`를 호출하여 API 키를 생성합니다.
- `apikey:import`를 호출하여 파일에서 API 키를 가져옵니다. 파일 형식은 API Gateway API 키 파일 형식 단원을 참조하세요.

새 API 키의 값은 변경할 수 없습니다. 사용량 계획을 구성하는 방법에 대해 알아보려면 API Gateway CLI 및 REST API를 사용하여 사용량 계획을 생성, 구성 및 테스트 단원을 참조하세요.

API Gateway CLI 및 REST API를 사용하여 사용량 계획을 생성, 구성 및 테스트

사용량 계획을 구성하기 전에 선택한 API의 메서드가 API 키를 요구하도록 설정하고, 해당 API를 단계에 배포하거나 재배포하고, API 키를 하나 이상 생성하거나 가져와야 합니다. 자세한 내용은 API Gateway REST API를 사용하여 API 키 설정 단원을 참조하세요.

사용량 계획에 추가할 API를 이미 생성한 경우, API Gateway REST API를 사용하여 사용량 계획을 구성하려면 다음 지침을 따르십시오.

주제

- 기본 사용량 계획으로 마이그레이션
- 사용량 계획 생성
- AWS CLI를 사용하여 사용량 계획 관리
- 사용량 계획 테스트

기본 사용량 계획으로 마이그레이션

사용량 계획을 처음 생성하는 경우 다음 본문과 함께 `account:update`를 호출하여 선택한 API 키와 연결된 기존 API 단계를 사용량 계획으로 마이그레이션할 수 있습니다.

```
{
  "patchOperations" : [ {
    "op" : "add",
    "path" : "/features",
    "value" : "UsagePlans"
  } ]
}
```

API 키와 연결된 API 단계를 마이그레이션하는 방법에 대한 자세한 내용은 API Gateway 콘솔에서 기본 사용량 계획으로 마이그레이션을 참조하세요.

사용량 계획 생성

다음 절차에서는 사용량 계획을 생성하는 방법에 대해 설명합니다.

REST API로 사용량 계획을 생성하려면

1. `usageplan:create`를 호출하여 사용량 계획을 생성합니다. 페이로드에서 계획의 이름과 설명, 연결된 API 단계, 속도 제한, 할당량을 지정합니다.

   결과적 사용량 계획 식별자를 기록해 둡니다. 이 정보는 다음 단계에서 필요합니다.
2. 다음 중 하나를 수행하세요.

   a. `usageplankey:create`를 호출하여 API 키를 사용량 계획에 추가합니다. 페이로드에서 `keyId`와 `keyType`을 지정합니다.

      사용량 계획에 API 키를 더 추가하려면 한 번에 한 API 키씩 앞의 호출을 반복합니다.

b. apikey:import를 호출하여 하나 이상의 API 키를 지정된 사용량 계획에 직접 추가합니다. 요청 페이로드는 API 키 값, 연결된 사용량 계획 식별자, 그리고 사용량 계획에 대해 활성화된 키를 보여 주는 부울 플래그를 포함해야 하며, API 키 이름 및 설명도 포함할 수 있습니다.

다음 apikey:import 요청의 예는 3개의 API 키(key, name, description으로 식별됨)를 하나의 사용량 계획(usageplanIds로 식별됨)에 추가합니다.

```
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
```

따라서 3개의 UsagePlanKey 리소스가 생성되고 UsagePlan에 추가됩니다.

이런 식으로 두 개 이상의 사용량 계획에 API 키를 추가할 수도 있습니다. 이를 위해 각 usageplanIds 열 값을 사용자가 선택한 사용량 계획 식별자를 포함하고 따옴표로 묶인, 쉼표로 구분된 문자열로 변경합니다("n371pt,m282qs" 또는 'n371pt,m282qs').

> **Note**
>
> API 키는 하나 이상의 사용량 계획과 연결할 수 있습니다. 사용량 계획은 하나 이상의 단계와 연결할 수 있습니다. 그러나 지정된 API 키는 API의 각 단계에 대하여 단 하나의 사용량 계획에만 연결할 수 있습니다.

AWS CLI를 사용하여 사용량 계획 관리

다음 코드 예제에서는 update-usage-plan 명령을 호출하여 사용량 계획에서 메서드 수준 조절 설정을 추가, 삭제 또는 수정하는 방법을 보여 줍니다.

> **Note**
>
> us-east-1을 API에 해당하는 리전 값으로 변경해야 합니다.

개별 리소스와 메서드 조절의 속도 제한을 추가하거나 바꾸는 방법:

```
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"
```

개별 리소스와 메서드 조절의 버스트 제한을 추가하거나 바꾸는 방법:

```
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"
```

개별 리소스와 메서드에 대한 메서드 수준 조절 설정을 제거하는 방법:

```
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=""
```

API에 대한 모든 메서드 수준 조절 설정을 제거하는 방법:

```
aws apigateway --region us-east-1 update-usage-plan --usage-plan-id <planId> --patch-
operations op="remove",path="/apiStages/<apiId>:<stage>/throttle ",value=""
```

다음은 Pet Store 샘플 API를 사용하는 예입니다.

```
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}}"'
```

사용량 계획 테스트

에서 자습서: 예제를 가져와 REST API 생성 만든 PetStore API를 예로 들어 보겠습니다. 이 API는 API 키 `Hiorr45VR...c4GJc`를 사용하도록 구성되었다고 가정합니다. 다음 단계에서는 사용량 단계를 테스트하는 방법에 대해 설명합니다.

사용량 계획을 테스트하려면

- 사용량 계획에 API의 GET 쿼리 파라미터(예: /pets)가 포함된 Pets 리소스(?type=...&page=...)에서 xbvxlpijch 요청을 합니다.

```
GET /testStage/pets?type=dog&page=1 HTTP/1.1
x-api-key: Hiorr45VR...c4GJc
Content-Type: application/x-www-form-urlencoded
Host: xbvxlpijch.execute-api.ap-southeast-1.amazonaws.com
X-Amz-Date: 20160803T001845Z
Authorization: AWS4-HMAC-SHA256 Credential={access_key_ID}/20160803/ap-southeast-1/
execute-api/aws4_request, SignedHeaders=content-type;host;x-amz-date;x-api-key,
 Signature={sigv4_hash}
```

> **Note**
>
> API Gateway의 execute-api 구성 요소에 이 요청을 제출하고 필수 Hiorr45VR...c4GJc 헤더에 필수 API 키(예: x-api-key)를 제공해야 합니다.

응답이 성공하면 백엔드에서 200 OK 상태 코드와 요청한 결과가 포함된 페이로드가 반환됩니다. x-api-key 헤더 설정을 잊었거나 잘못된 키로 설정한 경우, 403 Forbidden 응답을 받게 됩니다. 하지만 API 키를 요구하도록 메서드를 구성하지 않은 경우, 200 OK 헤더를 올바르게 설정하든 그렇지 않든 x-api-key 응답을 받을 가능성이 높으며, 사용량 계획의 조절 및 할당량 제한은 우회됩니다.

경우에 따라 API Gateway가 요청에 대해 사용량 계획 조절 제한 또는 할당량을 적용할 수 없는 내부 오류가 발생할 경우, API Gateway는 조절 제한 또는 할당량을 사용량 계획에 지정된 대로 적용하지 않고 요청을 제공합니다. 하지만 CloudWatch에 Usage Plan check failed due to an internal error 오류 메시지가 기록됩니다. 가끔 발생하는 이러한 오류는 무시해도 좋습니다.

를 사용하여 API 키와 사용량 계획을 만들고 구성하십시오. AWS CloudFormation

를 AWS CloudFormation 사용하여 API 메서드에서 API 키를 요구하고 API에 대한 사용량 계획을 생성할 수 있습니다. 예제 AWS CloudFormation 템플릿은 다음과 같은 작업을 수행합니다.

- GET 및 POST 메서드를 사용하여 API Gateway API를 생성합니다.

- GET 및 POST 메서드에 대한 API 키가 필요합니다. 이 API는 들어오는 각 요청의 X-API-KEY 헤더에서 키를 받습니다.
- API 키 생성
- 매월 1,000개 요청의 월별 할당량, 초당 100개 요청의 스로틀링 속도 제한, 초당 200개 요청의 스로틀링 버스트 제한을 지정하는 사용 계획을 생성합니다.
- GET 메서드에 대해 초당 50개 요청의 메서드 수준 스로틀링 속도 제한과 초당 100개 요청의 메서드 수준 스로틀링 버스트 제한을 지정합니다.
- API 단계와 API 키를 사용량 계획에 연결합니다.

```
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}'
```

API Gateway API 키 파일 형식

API Gateway는 쉼표로 분리된 값(CSV) 형식의 외부 파일에서 API 키를 가져온 다음, 가져온 키를 하나 이상의 사용량 계획과 연결할 수 있습니다. 가져온 파일에는 Name 및 Key열이 포함되어 있어야 합니다. 다음 예에서 보듯 열 헤더 이름은 대/소문자를 구분하지 않으며, 열 순서는 상관없습니다.

```
Key,name
apikey1234abcdefghij0123456789,MyFirstApiKey
```

Key값은 20~128자 사이여야 합니다. Name 값은 1,204자를 초과할 수 없습니다.

또한 API 키 파일은 다음과 같이 Description열, Enabled 열 또는 UsagePlanIds열을 포함할 수 있습니다.

```
Name,key,description,Enabled,usageplanIds
MyFirstApiKey,apikey1234abcdefghij0123456789,An imported key,TRUE,c7y23b
```

키 1개가 둘 이상의 사용량 계획과 연결된 경우, 다음 예에서 보듯 UsagePlanIds 값은 큰따옴표나 작은따옴표로 묶인 사용량 계획 ID를 쉼표로 구분한 문자열입니다.

```
Enabled,Name,key,UsageplanIds
true,MyFirstApiKey,apikey1234abcdefghij0123456789,"c7y23b,glvrsr"
```

인식할 수 없는 열도 사용할 수 있지만 무시됩니다. 기본값은 빈 문자열 또는 true 부울 값입니다.

최신 버전으로 이전 버전을 덮어쓰면 동일한 API 키를 여러 번 가져올 수 있습니다. key 값이 동일한 경우 두 API 키는 동일합니다.

> **Note**
>
> 고려해야 할 모범 사례는 the section called “API 키 및 사용량 계획의 위한 모범 사례” 섹션을 참조하세요.

REST API 문서화

고객이 API를 이해하고 사용할 수 있도록 API를 문서화해야 합니다. API를 문서화하는 데 도움이 되도록 API Gateway에서는 개별 API 엔터티에 대한 도움말 콘텐츠를 API 개발 프로세스의 핵심 부분으로 추가하고 업데이트할 수 있습니다. API Gateway에서는 소스 콘텐츠가 저장되며 다양한 버전의 설명서를 보관할 수 있습니다. 설명서 버전을 API 단계와 연결하고, 단계별 설명서 스냅샷을 외부 OpenAPI 파일로 내보내고, 이 파일을 설명서 간행물로 배포할 수 있습니다.

API를 문서화하려면 API Gateway REST API를 호출하거나, API Gateway용 AWS SDK 또는 AWS CLI 중 하나를 사용하거나, API Gateway 콘솔을 사용할 수 있습니다. 또한 외부 OpenAPI 파일에 정의된 설명서 부분을 가져오거나 내보낼 수 있습니다.

개발자 포털을 사용하면 API 설명서를 개발자와 공유할 수 있습니다. 예제를 보려면 AWS 파트너 네트워크(APN) 블로그에서 Integrating ReadMe with API Gateway to Keep Your Developer Hub Up to Date(ReadMe를 API Gateway와 통합하여 개발자 허브를 최신 상태로 유지)를 참조하세요.

주제

- API Gateway에 API 설명서 표시
- API Gateway 콘솔을 사용하여 API 문서화
- API Gateway 콘솔을 사용하여 API 설명서 게시
- API Gateway REST API를 사용하여 API 문서화
- API Gateway REST API를 사용하여 API 설명서 게시
- API 설명서 가져오기
- API 설명서에 대한 액세스 제어

API Gateway에 API 설명서 표시

API Gateway API 설명서는 API, 리소스, 메서드, 요청, 응답, 메시지 파라미터(즉 경로, 쿼리, 헤더)뿐 아니라 권한 부여자 및 모델을 포함하는 특정 API 엔터티와 연결된 개별 설명서 부분으로 구성됩니다.

API Gateway에서 설명서 부분은 DocumentationPart 리소스로 표시됩니다. API 설명서 전체는 DocumentationParts 컬렉션으로 표시됩니다.

API 문서화에는 API가 개선됨에 따라 `DocumentationPart` 인스턴스 생성, 인스턴스를 `DocumentationParts` 모음에 추가, 문서 부분의 버전을 관리하는 작업이 포함됩니다.

주제

- 설명서 부분
- 설명서 버전

설명서 부분

DocumentationPart 리소스는 개별 API 엔터티에 적용 가능한 설명서 콘텐츠를 저장하는 JSON 객체입니다. `properties` 필드에는 설명서 콘텐츠가 키-값 페어의 맵으로 포함됩니다. `location` 속성은 연관된 API 엔터티를 식별합니다.

콘텐츠 맵의 형태는 API 개발자가 결정합니다. 키-값 페어의 값은 문자열, 숫자, 부울, 객체 또는 어레이일 수 있습니다. `location` 객체의 형태는 대상 엔터티 유형에 따라 결정됩니다.

`DocumentationPart` 리소스는 콘텐츠 상속을 지원합니다. API 엔터티의 설명서 콘텐츠는 해당 API 엔터티의 하위에 적용됩니다. 하위 엔터티 및 콘텐츠 상속의 정의에 대한 자세한 내용은 더 일반적인 사양의 API 엔터티에서 콘텐츠 상속 단원을 참조하십시오.

설명서 부분의 위치

DocumentationPart 인스턴스의 location 속성은 연결된 콘텐츠가 적용되는 API 엔터티를 식별합니다. API 엔터티는 API Gateway REST API 리소스(예: RestApi, Resource, Method, MethodResponse, Authorizer 또는 Model)일 수 있습니다. 엔터티는 URL 경로 파라미터, 쿼리 문자열 파라미터, 요청 또는 응답 헤더 파라미터, 요청 또는 응답 본문 또는 응답 상태 코드와 같은 메시지 파라미터일 수도 있습니다.

API 엔터티를 지정하려면 `location` 객체의 type 속성을 `API`, `AUTHORIZER`, `MODEL`, `RESOURCE`, `METHOD`, `PATH_PARAMETER`, `QUERY_PARAMETER`, `REQUEST_HEADER`, `REQUEST_BODY`, `RESPONSE`, `RESPONSE_HEADER` 또는 `RESPONSE_BODY` 중 하나로 설정합니다.

API 엔터티의 type에 따라 method, name, path 및 statusCode를 포함한 기타 location 속성을 지정할 수 있습니다. 이러한 속성이 모두 특정 API 엔터티에 유효하지는 않습니다. 예를 들어 type, path, name 및 statusCode는 RESPONSE 엔터티의 유효한 속성이고, type 및 path만 RESOURCE 엔터티의 유효한 위치 속성입니다. 주어진 API 엔터티에 대해 location의 DocumentationPart에 잘못된 필드를 포함시키는 것은 오류입니다.

모든 유효한 location 필드가 필요하지는 않습니다. 예를 들어, type은 모든 API 엔터티의 유효한 필수 location 필드입니다. 그러나 method, path 및 statusCode는 유효하지만 RESPONSE 엔터티의 필수 속성은 아닙니다. 명시적으로 지정되지 않으면 유효한 location 필드는 기본값을 사용합니다. 기본 path 값은 /, 즉 API의 루트 리소스입니다. method의 기본값 또는 statusCode는 *입니다. 이는 각각 모든 메서드 또는 상태 코드 값을 의미합니다.

설명서 부분의 콘텐츠

properties 값은 JSON 문자열로 인코딩됩니다. properties 값에는 설명서 요구사항을 충족하기 위해 선택한 정보가 포함되어 있습니다. 예를 들어, 다음은 유효한 콘텐츠 맵입니다.

```
{
  "info": {
    "description": "My first API with Amazon API Gateway."
  },
  "x-custom-info" : "My custom info, recognized by OpenAPI.",
  "my-info" : "My custom info not recognized by OpenAPI."
}
```

API Gateway는 유효한 JSON 문자열을 콘텐츠 맵으로 허용하지만, 콘텐츠 속성은 OpenAPI에서 인식할 수 있는 범주와 인식할 수 없는 범주라는 두 가지 범주로 처리됩니다. 이전 예에서 info, description 및 x-custom-info는 OpenAPI에서 표준 OpenAPI 객체, 속성 또는 확장으로 인식합니다. 반대로 my-info는 OpenAPI 사양을 준수하지 않습니다. API Gateway는 OpenAPI 호환 콘텐츠 속성을 연결된 DocumentationPart 인스턴스의 API 엔터티 정의로 전파합니다. API Gateway는 비호환 콘텐츠 속성을 API 엔터티 정의로 전파하지 않습니다.

또 다른 예로, 여기에 DocumentationPart 엔터티를 대상으로 하는 Resource가 있습니다.

```
{
    "location" : {
        "type" : "RESOURCE",
        "path": "/pets"
    },
    "properties" : {
```

```
        "summary" : "The /pets resource represents a collection of pets in PetStore.",
        "description": "... a child resource under the root...",
    }
}
```

여기서 type 및 path는 모두 RESOURCE 유형의 대상을 식별하는 데 유효한 필드입니다. 루트 리소스(/)의 경우, path 필드를 생략할 수 있습니다.

```
{
    "location" : {
        "type" : "RESOURCE"
    },
    "properties" : {
        "description" : "The root resource with the default path specification."
    }
}
```

이는 다음 DocumentationPart 인스턴스와 동일합니다.

```
{
    "location" : {
        "type" : "RESOURCE",
        "path": "/"
    },
    "properties" : {
        "description" : "The root resource with an explicit path specification"
    }
}
```

더 일반적인 사양의 API 엔터티에서 콘텐츠 상속

선택적 location 필드의 기본값은 API 엔터티의 패턴이 있는 설명을 제공합니다. location 객체의 기본값을 사용하면 이 properties 패턴 유형의 DocumentationPart 인스턴스에 location 맵의 일반적인 설명을 추가할 수 있습니다. 특정 엔터티에 연결된 DocumentationPart 인스턴스가 아직 없으면, API Gateway는 일반 API 엔터티의 location에서 적용 가능한 OpenAPI 설명서 속성을 추출하고 일반 location 패턴과 일치하거나 정확한 값과 일치하는 DocumentationPart 필드를 사용하여 특정 API 엔터티에 해당 속성을 주입합니다. 이 동작은 더 일반적인 사양의 API 엔터티에서 콘텐츠 상속이라고도 합니다.

콘텐츠 상속은 특정 API 개체 유형에는 적용되지 않습니다. 자세한 내용은 아래 표를 참조하십시오.

API 엔터티가 둘 이상의 DocumentationPart 위치 패턴과 일치할 때 엔터티는 설명서 부분을 우선 순위가 가장 높은 위치 필드로 상속합니다. 우선 순위는 path > statusCode입니다. path 필드와 일치시키기 위해 API Gateway는 가장 구체적인 경로 값이 있는 엔터티를 선택합니다. 다음 표에는 몇 가지 예가 나와 있습니다.

케이스	path	statusCode	name	설명
1	/pets	*	id	이 위치 패턴과 관련된 설명서는 위치 패턴과 일치하는 엔터티에 상속됩니다.
2	/pets	200	id	케이스 2가 케이스 1보다 더 구체적이

케이스	path	statusCode	name	설명	
				기 때문에 이 위치 패턴과 관련된 문서는 케이스 1과 케이스 2가 일치할 때 위치 패턴과 일치하는 엔터티에 상속됩니다.	

케이스	path	statusCode	name	설명
3	/pets/petId	*	id	케이스 3이 케이스 2보다 우선 순위가 높고 케이스 1보다 구체적이기 때문에 케이스 1, 2 및 3이 일치할 때 이 위치 패턴과 관련된 문서

케이스	path	statusCode	name	설명	
				가 위치 패턴과 일치하는 엔터티에 상속됩니다.	

보다 일반적인 DocumentationPart 인스턴스와 보다 구체적인 인스턴스를 대조하는 또 다른 예가 있습니다. 재정의되지 않는 한 다음과 같은 "Invalid request error"의 일반적인 오류 메시지가 400 오류 응답의 OpenAPI 정의에 삽입됩니다.

```
{
    "location" : {
        "type" : "RESPONSE",
        "statusCode": "400"
    },
    "properties" : {
        "description" : "Invalid request error."
    }"
}
```

다음 덮어쓰기를 사용하면 400 리소스의 모든 메서드에 대한 /pets 응답에 "Invalid petId specified"가 대신 설명됩니다.

```
{
    "location" : {
        "type" : "RESPONSE",
        "path": "/pets",
        "statusCode": "400"
    },
```

```
    "properties" : "{
        "description" : "Invalid petId specified."
    }"
}
```

DocumentationPart의 유효한 위치 필드

다음 표는 지정된 유형의 API 엔터티와 연결된 DocumentationPart 리소스의 적용 가능한 기본값과 유효 및 필수 필드를 보여줍니다.

API 엔터티	유효한 위치 필드	필수 위치 필드	기본 필드 값	상속 가능한 콘텐츠
API	`{ "location": { "type": "API" }, ... }`	type	해당 사항 없음	아니요
리소스	`{ "location": { "type": "RESOURCE", "path": "resource_path " }, ... }`	type	path의 기본값은 /입니다.	아니요
방법	`{ "location": { "type": "METHOD", "path": "resource_path ", "method": "http_verb " }, ...`	type	path 및 method의 기본값은 각각 / 및 *입니다.	예, 접두사로 path를 매칭하고 모든 값의 method를 매칭합니다.

API 엔터티	유효한 위치 필드	필수 위치 필드	기본 필드 값	상속 가능한 콘텐츠
	<pre>}</pre>			

쿼리 파라미터	<pre>{
	    "location": {
	        "type": "QUERY_PARAMETER",
	        "path": "*resource_path* ",
	        "method": "*HTTP_verb* ",
	        "name": "*query_parameter_name* "
	    },
	    ...
	}</pre>
		type	path 및 method의 기본값은 각각 / 및 *입니다.	예, 접두사로 path를 매칭하고 정확한 값으로 method를 매칭합니다.

요청 본문	<pre>{
	    "location": {
	        "type": "REQUEST_BODY",
	        "path": "*resource_path* ",
	        "method": "*http_verb* "
	    },
	    ...
	}</pre>
		type	path 및 method의 기본값은 각각 / 및 *입니다.	예, 접두사로 path를 매칭하고 정확한 값으로 method를 매칭합니다.

API 엔터티	유효한 위치 필드	필수 위치 필드	기본 필드 값	상속 가능한 콘텐츠

요청 헤더 파라미터	{
	"location": {
	"type": "REQUEST_HEADER",
	"path": "*resource_path* ",
	"method": "*HTTP_verb* ",
	"name": "*header_name* "
	},
	...
	}
		type, name	path 및 method의 기본값은 각각 / 및 *입니다.	예, 접두사로 path를 매칭하고 정확한 값으로 method를 매칭합니다.

요청 경로 파라미터	{
	"location": {
	"type": "PATH_PARAMETER",
	"path": "*resource*/{*path_parameter_name* }",
	"method": "*HTTP_verb* ",
	"name": "*path_parameter_name* "
	},
	...
	}
		type, name	path 및 method의 기본값은 각각 / 및 *입니다.	예, 접두사로 path를 매칭하고 정확한 값으로 method를 매칭합니다.

API 엔터티	유효한 위치 필드	필수 위치 필드	기본 필드 값	상속 가능한 콘텐츠

응답	{
	"location": {
	"type": "RESPONSE",
	"path": "*resource_path* ",
	"method": "*http_verb* ",
	"statusCode": "*status_code* "
	},
	...
	}
		type	path, method 및 statusCode 의 기본값은 각각 /, * 및 *입니다.	예, 접두사로 path를 매칭하고 정확한 값으로 method 및 statusCode 를 매칭합니다.

응답 헤더	{
	"location": {
	"type": "RESPONSE_HEADER",
	"path": "*resource_path* ",
	"method": "*http_verb* ",
	"statusCode": "*status_code* ",
	"name": "*header_name* "
	},
	...
	}
		type, name	path, method 및 statusCode 의 기본값은 각각 /, * 및 *입니다.	예, 접두사로 path를 매칭하고 정확한 값으로 method 및 statusCode 를 매칭합니다.

API 엔터티	유효한 위치 필드	필수 위치 필드	기본 필드 값	상속 가능한 콘텐츠

응답 본문	{
	"location": {
	"type": "RESPONSE_BODY",
	"path": "*resource_path* ",
	"method": "*http_verb* ",
	"statusCode": "*status_code* "
	},
	...
	}
		type	path, method 및 statusCode 의 기본값은 각각 /, * 및 *입니다.	예, 접두사로 path를 매칭하고 정확한 값으로 method 및 statusCode 를 매칭합니다.

권한 부여자	{
	"location": {
	"type": "AUTHORIZER",
	"name": "*authorizer_name* "
	},
	...
	}
		type	해당 사항 없음	아니요

모델	{
	"location": {
	"type": "MODEL",
	"name": "*model_name* "
	},
	...
	}
		type	해당 사항 없음	아니요

설명서 버전

설명서 버전은 API의 DocumentationParts 컬렉션에 대한 스냅샷이며 버전 식별자로 태그 지정됩니다. API의 설명서를 게시하려면 설명서 버전을 만들고 이를 API 단계와 연결하고 API 설명서의 해당 단계별 버전을 외부 OpenAPI 파일로 내보내는 작업이 필요합니다. API Gateway에서 설명서 스냅샷은 DocumentationVersion 리소스로 표시됩니다.

API를 업데이트하면 새 버전의 API가 만들어집니다. API Gateway에서는 DocumentationVersions 컬렉션을 사용하여 모든 설명서 버전을 유지 관리합니다.

API Gateway 콘솔을 사용하여 API 문서화

이 단원에서는 API Gateway 콘솔을 사용하여 API의 설명서 부분을 생성하고 유지 관리하는 방법에 대해 설명합니다.

API의 설명서를 작성하고 편집하기 위한 전제 조건으로 API를 이미 작성했어야 합니다. 이 단원에서는 PetStore API를 예로 사용합니다. API Gateway 콘솔을 사용하여 API를 생성하려면 자습서: 예제를 가져와 REST API 생성의 지침을 따릅니다.

주제

- API 엔터티 문서화
- RESOURCE 엔터티 문서화
- METHOD 엔터티 문서화
- QUERY_PARAMETER 엔터티 문서화
- PATH_PARAMETER 엔터티 문서화
- REQUEST_HEADER 엔터티 문서화
- REQUEST_BODY 엔터티 문서화
- RESPONSE 엔터티 문서화
- RESPONSE_HEADER 엔터티 문서화
- RESPONSE_BODY 엔터티 문서화
- MODEL 엔터티 문서화
- AUTHORIZER 엔터티 문서화

API 엔터티 문서화

API 엔터티에 새 설명서 부분을 추가하려면 다음을 수행합니다.

1. 기본 탐색 창에서 설명서를 선택한 다음 설명서 부분 생성을 선택합니다.
2. 설명서 유형에서 API를 선택합니다.

   설명서 부분이 API용으로 생성되지 않은 경우, 설명서 부분의 properties 맵 편집기를 가져옵니다. 텍스트 편집기에 다음 properties 맵을 입력합니다.

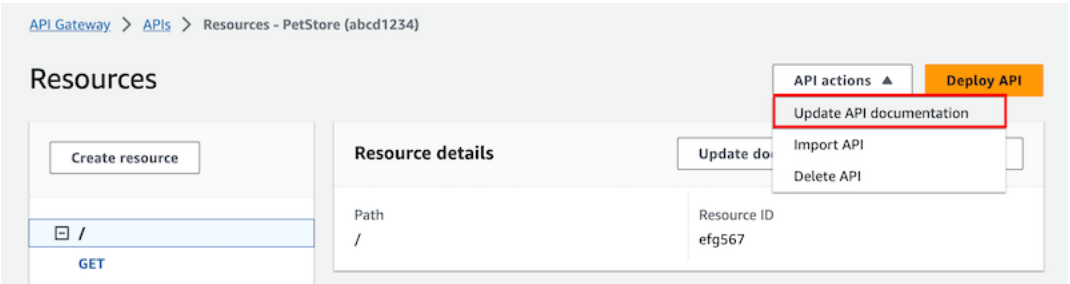
```
{
  "info": {
    "description": "Your first API Gateway API.",
    "contact": {
        "name": "John Doe",
        "email": "john.doe@api.com"
    }
  }
}
```

> **Note**
>
> properties 맵을 JSON 문자열에 인코딩할 필요가 없습니다. API Gateway 콘솔은 JSON 객체를 자동으로 문자열화합니다.

3. 설명서 부분 생성을 선택합니다.

리소스 창에서 API 엔터티에 새 설명서 부분을 추가하려면 다음을 수행합니다.

1. 기본 탐색 창에서 리소스를 선택합니다.
2. API 작업 메뉴를 선택한 다음 API 설명서 업데이트를 선택합니다.



기존 설명서 부분을 편집하려면 다음을 수행합니다.

1. 설명서 창에서 리소스 및 메서드 탭을 선택합니다.

2. API 이름을 선택한 다음 API 카드에서 편집을 선택합니다.

RESOURCE 엔터티 문서화

RESOURCE 엔터티에 새 설명서 부분을 추가하려면 다음을 수행합니다.

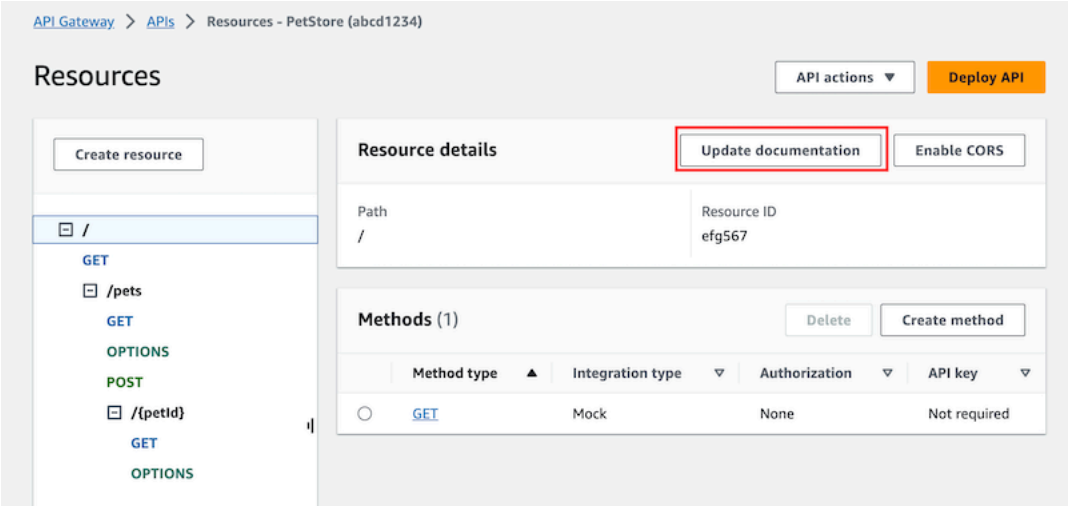
1. 기본 탐색 창에서 설명서를 선택한 다음 설명서 부분 생성을 선택합니다.
2. 설명서 유형에서 리소스를 선택합니다.
3. 경로에 경로를 입력합니다.
4. 텍스트 편집기에서 설명을 입력합니다. 예를 들면 다음과 같습니다.

```
{
    "description": "The PetStore's root resource."
}
```

5. 설명서 부분 생성을 선택합니다. 목록에 없는 리소스에 대한 설명서를 생성할 수 있습니다.
6. 필요한 경우 이 단계를 반복하여 다른 설명서 부분을 추가하거나 편집합니다.

리소스 창에서 RESOURCE 엔터티에 새 설명서 부분을 추가하려면 다음을 수행합니다.

1. 기본 탐색 창에서 리소스를 선택합니다.
2. 리소스를 선택한 다음 설명서 업데이트를 선택합니다.



기존 설명서 부분을 편집하려면 다음을 수행합니다.

1. 설명서 창에서 리소스 및 메서드 탭을 선택합니다.
2. 설명서 부분이 포함된 리소스를 선택한 다음 편집을 선택합니다.

METHOD 엔터티 문서화

METHOD 엔터티에 새 설명서 부분을 추가하려면 다음을 수행합니다.

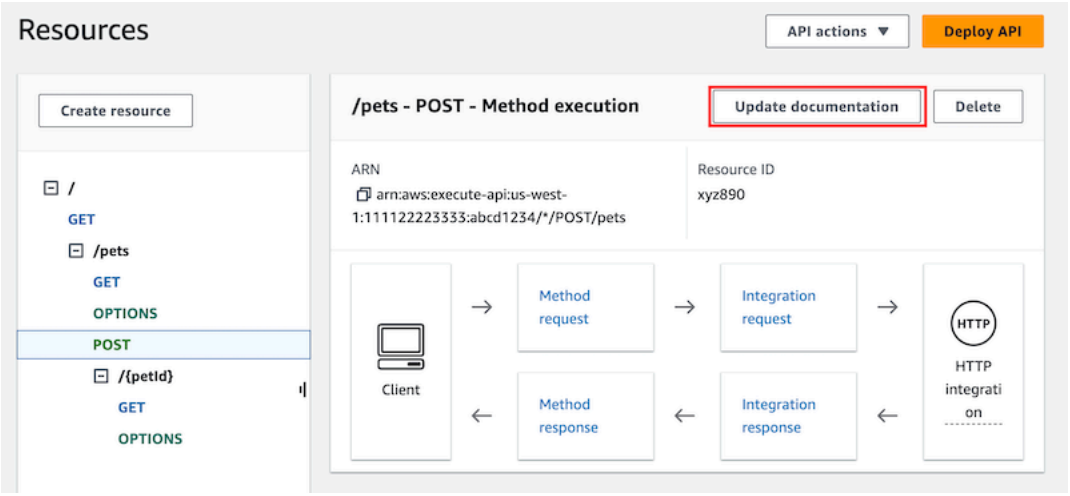
1. 기본 탐색 창에서 설명서를 선택한 다음 설명서 부분 생성을 선택합니다.
2. 설명서 유형에서 메서드를 선택합니다.
3. 경로에 경로를 입력합니다.
4. 메서드에서 HTTP 동사를 선택합니다.
5. 텍스트 편집기에서 설명을 입력합니다. 예를 들면 다음과 같습니다.

```
{
  "tags" : [ "pets" ],
  "summary" : "List all pets"
}
```

6. 설명서 부분 생성을 선택합니다. 목록에 없는 메서드에 대한 설명서를 생성할 수 있습니다.
7. 필요한 경우 이 단계를 반복하여 다른 설명서 부분을 추가하거나 편집합니다.

리소스 창에서 METHOD 엔터티에 새 설명서 부분을 추가하려면 다음을 수행합니다.

1. 기본 탐색 창에서 리소스를 선택합니다.
2. 메서드를 선택한 다음 설명서 업데이트를 선택합니다.



기존 설명서 부분을 편집하려면 다음을 수행합니다.

1. 설명서 창에서 리소스 및 메서드 탭을 선택합니다.
2. 메서드를 선택하거나 메서드가 포함된 리소스를 선택한 다음 검색 창을 사용하여 설명서 부분을 찾고 선택할 수 있습니다.
3. 편집을 선택합니다.

QUERY_PARAMETER 엔터티 문서화

QUERY_PARAMETER 엔터티에 새 설명서 부분을 추가하려면 다음을 수행합니다.

1. 기본 탐색 창에서 설명서를 선택한 다음 설명서 부분 생성을 선택합니다.
2. 설명서 유형에서 쿼리 파라미터를 선택합니다.
3. 경로에 경로를 입력합니다.
4. 메서드에서 HTTP 동사를 선택합니다.
5. 이름에 이름을 입력합니다.
6. 텍스트 편집기에서 설명을 입력합니다.
7. 설명서 부분 생성을 선택합니다. 목록에 없는 쿼리 파라미터에 대한 설명서를 생성할 수 있습니다.
8. 필요한 경우 이 단계를 반복하여 다른 설명서 부분을 추가하거나 편집합니다.

기존 설명서 부분을 편집하려면 다음을 수행합니다.

1. 설명서 창에서 리소스 및 메서드 탭을 선택합니다.
2. 쿼리 파라미터를 선택하거나 쿼리 파라미터가 포함된 리소스를 선택한 다음 검색 창을 사용하여 설명서 부분을 찾고 선택할 수 있습니다.
3. 편집을 선택합니다.

PATH_PARAMETER 엔터티 문서화

PATH_PARAMETER 엔터티에 새 설명서 부분을 추가하려면 다음을 수행합니다.

1. 기본 탐색 창에서 설명서를 선택한 다음 설명서 부분 생성을 선택합니다.
2. 설명서 유형에서 경로 파라미터를 선택합니다.
3. 경로에 경로를 입력합니다.
4. 메서드에서 HTTP 동사를 선택합니다.

5. 이름에 이름을 입력합니다.
6. 텍스트 편집기에서 설명을 입력합니다.
7. 설명서 부분 생성을 선택합니다. 목록에 없는 경로 파라미터에 대한 설명서를 생성할 수 있습니다.
8. 필요한 경우 이 단계를 반복하여 다른 설명서 부분을 추가하거나 편집합니다.

기존 설명서 부분을 편집하려면 다음을 수행합니다.

1. 설명서 창에서 리소스 및 메서드 탭을 선택합니다.
2. 경로 파라미터를 선택하거나 경로 파라미터가 포함된 리소스를 선택한 다음 검색 창을 사용하여 설명서 부분을 찾고 선택할 수 있습니다.
3. 편집을 선택합니다.

REQUEST_HEADER 엔터티 문서화

REQUEST_HEADER 엔터티에 새 설명서 부분을 추가하려면 다음을 수행합니다.

1. 기본 탐색 창에서 설명서를 선택한 다음 설명서 부분 생성을 선택합니다.
2. 설명서 유형에서 요청 헤더를 선택합니다.
3. 경로에 요청 헤더의 경로를 입력합니다.
4. 메서드에서 HTTP 동사를 선택합니다.
5. 이름에 이름을 입력합니다.
6. 텍스트 편집기에서 설명을 입력합니다.
7. 설명서 부분 생성을 선택합니다. 목록에 없는 요청 헤더에 대한 설명서를 생성할 수 있습니다.
8. 필요한 경우 이 단계를 반복하여 다른 설명서 부분을 추가하거나 편집합니다.

기존 설명서 부분을 편집하려면 다음을 수행합니다.

1. 설명서 창에서 리소스 및 메서드 탭을 선택합니다.
2. 요청 헤더를 선택하거나 요청 헤더가 포함된 리소스를 선택한 다음 검색 창을 사용하여 설명서 부분을 찾고 선택할 수 있습니다.
3. 편집을 선택합니다.

REQUEST_BODY 엔터티 문서화

REQUEST_BODY 엔터티에 새 설명서 부분을 추가하려면 다음을 수행합니다.

1. 기본 탐색 창에서 설명서를 선택한 다음 설명서 부분 생성을 선택합니다.
2. 설명서 유형에서 요청 본문을 선택합니다.
3. 경로에 요청 본문의 경로를 입력합니다.
4. 메서드에서 HTTP 동사를 선택합니다.
5. 텍스트 편집기에서 설명을 입력합니다.
6. 설명서 부분 생성을 선택합니다. 목록에 없는 요청 본문에 대한 설명서를 생성할 수 있습니다.
7. 필요한 경우 이 단계를 반복하여 다른 설명서 부분을 추가하거나 편집합니다.

기존 설명서 부분을 편집하려면 다음을 수행합니다.

1. 설명서 창에서 리소스 및 메서드 탭을 선택합니다.
2. 요청 본문을 선택하거나 요청 본문이 포함된 리소스를 선택한 다음 검색 창을 사용하여 설명서 부분을 찾고 선택할 수 있습니다.
3. 편집을 선택합니다.

RESPONSE 엔터티 문서화

RESPONSE 엔터티에 새 설명서 부분을 추가하려면 다음을 수행합니다.

1. 기본 탐색 창에서 설명서를 선택한 다음 설명서 부분 생성을 선택합니다.
2. 설명서 유형에서 응답(상태 코드)을 선택합니다.
3. 경로에 응답의 경로를 입력합니다.
4. 메서드에서 HTTP 동사를 선택합니다.
5. 상태 코드에 HTTP 상태 코드를 입력합니다.
6. 텍스트 편집기에서 설명을 입력합니다.
7. 설명서 부분 생성을 선택합니다. 목록에 없는 응답 상태 코드에 대한 설명서를 생성할 수 있습니다.
8. 필요한 경우 이 단계를 반복하여 다른 설명서 부분을 추가하거나 편집합니다.

기존 설명서 부분을 편집하려면 다음을 수행합니다.

1. 설명서 창에서 리소스 및 메서드 탭을 선택합니다.
2. 응답 상태 코드를 선택하거나 응답 상태 코드가 포함된 리소스를 선택한 다음 검색 창을 사용하여 설명서 부분을 찾고 선택할 수 있습니다.
3. 편집을 선택합니다.

RESPONSE_HEADER 엔터티 문서화

RESPONSE_HEADER 엔터티에 새 설명서 부분을 추가하려면 다음을 수행합니다.

1. 기본 탐색 창에서 설명서를 선택한 다음 설명서 부분 생성을 선택합니다.
2. 설명서 유형에서 응답 헤더를 선택합니다.
3. 경로에 응답 헤더의 경로를 입력합니다.
4. 메서드에서 HTTP 동사를 선택합니다.
5. 상태 코드에 HTTP 상태 코드를 입력합니다.
6. 텍스트 편집기에서 설명을 입력합니다.
7. 설명서 부분 생성을 선택합니다. 목록에 없는 응답 헤더에 대한 설명서를 생성할 수 있습니다.
8. 필요한 경우 이 단계를 반복하여 다른 설명서 부분을 추가하거나 편집합니다.

기존 설명서 부분을 편집하려면 다음을 수행합니다.

1. 설명서 창에서 리소스 및 메서드 탭을 선택합니다.
2. 응답 헤더를 선택하거나 응답 헤더가 포함된 리소스를 선택한 다음 검색 창을 사용하여 설명서 부분을 찾고 선택할 수 있습니다.
3. 편집을 선택합니다.

RESPONSE_BODY 엔터티 문서화

RESPONSE_BODY 엔터티에 새 설명서 부분을 추가하려면 다음을 수행합니다.

1. 기본 탐색 창에서 설명서를 선택한 다음 설명서 부분 생성을 선택합니다.
2. 설명서 유형에서 응답 본문을 선택합니다.
3. 경로에 응답 본문의 경로를 입력합니다.
4. 메서드에서 HTTP 동사를 선택합니다.
5. 상태 코드에 HTTP 상태 코드를 입력합니다.

6. 텍스트 편집기에서 설명을 입력합니다.
7. 설명서 부분 생성을 선택합니다. 목록에 없는 응답 본문에 대한 설명서를 생성할 수 있습니다.
8. 필요한 경우 이 단계를 반복하여 다른 설명서 부분을 추가하거나 편집합니다.

기존 설명서 부분을 편집하려면 다음을 수행합니다.

1. 설명서 창에서 리소스 및 메서드 탭을 선택합니다.
2. 응답 본문을 선택하거나 응답 본문이 포함된 리소스를 선택한 다음 검색 창을 사용하여 설명서 부분을 찾고 선택할 수 있습니다.
3. 편집을 선택합니다.

MODEL 엔터티 문서화

MODEL 엔터티를 문서화하려면 해당 모델의 DocumentPart 인스턴스 및 각각의 properties를 생성하고 관리해야 합니다. 예를 들어, 기본적으로 모든 API와 함께 제공되는 Error 모델에는 다음과 같은 스키마 정의가 있습니다.

```
{
  "$schema" : "http://json-schema.org/draft-04/schema#",
  "title" : "Error Schema",
  "type" : "object",
  "properties" : {
    "message" : { "type" : "string" }
  }
}
```

그리고 두 개의 DocumentationPart 인스턴스가 필요합니다. 하나는 Model용이고 다른 하나는 message 속성용입니다.

```
{
  "location": {
    "type": "MODEL",
    "name": "Error"
  },
  "properties": {
    "title": "Error Schema",
    "description": "A description of the Error model"
  }
}
```

및

```
{
  "location": {
    "type": "MODEL",
    "name": "Error.message"
  },
  "properties": {
    "description": "An error message."
  }
}
```

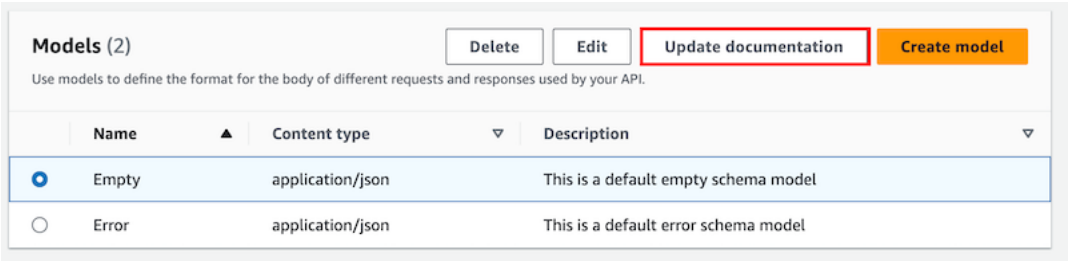
API를 내보낼 경우 `DocumentationPart`'의 속성이 원본 스키마의 값을 재정의합니다.

`MODEL` 엔터티에 새 설명서 부분을 추가하려면 다음을 수행합니다.

1. 기본 탐색 창에서 설명서를 선택한 다음 설명서 부분 생성을 선택합니다.
2. 설명서 유형에서 모델을 선택합니다.
3. 이름에 모델의 이름을 입력합니다.
4. 텍스트 편집기에서 설명을 입력합니다.
5. 설명서 부분 생성을 선택합니다. 목록에 없는 모델에 대한 설명서를 생성할 수 있습니다.
6. 필요한 경우 이 단계를 반복하여 설명서 부분을 다른 모델에 추가하거나 편집합니다.

모델 창에서 `MODEL` 엔터티에 새 설명서 부분을 추가하려면 다음을 수행합니다.

1. 기본 탐색 창에서 모델을 선택합니다.
2. 모델을 선택한 다음 설명서 업데이트를 선택합니다.



기존 설명서 부분을 편집하려면 다음을 수행합니다.

1. 설명서 창에서 모델 탭을 선택합니다.

2. 검색 창을 사용하거나 모델을 선택한 다음 편집을 선택합니다.

AUTHORIZER 엔터티 문서화

AUTHORIZER 엔터티에 새 설명서 부분을 추가하려면 다음을 수행합니다.

1. 기본 탐색 창에서 설명서를 선택한 다음 설명서 부분 생성을 선택합니다.
2. 설명서 유형에서 권한 부여자를 선택합니다.
3. 이름에 권한 부여자 이름을 입력합니다.
4. 텍스트 편집기에서 설명을 입력합니다. 권한 부여자에 대한 유효한 `location` 필드의 값을 지정합니다.
5. 설명서 부분 생성을 선택합니다. 목록에 없는 권한 부여자에 대한 설명서를 생성할 수 있습니다.
6. 필요한 경우 이 단계를 반복하여 설명서 부분을 다른 권한 부여자에 추가하거나 편집합니다.

기존 설명서 부분을 편집하려면 다음을 수행합니다.

1. 설명서 창에서 권한 부여자 탭을 선택합니다.
2. 검색 창을 사용하거나 권한 부여자를 선택한 다음 편집을 선택합니다.

API Gateway 콘솔을 사용하여 API 설명서 게시

다음 절차에서는 설명서 버전을 게시하는 방법을 설명합니다.

API Gateway 콘솔을 사용하여 설명서 버전을 게시하려면

1. 기본 탐색 창에서 설명서를 선택합니다.
2. 설명서 게시를 선택합니다.
3. 출판 설정:
    a. 스테이지에서 스테이지를 선택합니다.
    b. 버전에 버전 식별자(예: `1.0.0`)를 입력합니다.
    c. (선택 사항) 설명에 설명을 입력합니다.
4. [Publish]를 선택합니다.

이제 설명서를 외부 OpenAPI 파일로 내보내서 게시된 설명서를 다운로드할 수 있습니다. 자세한 내용은 the section called “REST API 내보내기” 단원을 참조하십시오.

API Gateway REST API를 사용하여 API 문서화

이 단원에서는 API Gateway REST API를 사용하여 API의 설명서 부분을 생성하고 유지 관리하는 방법에 대해 설명합니다.

API 설명서를 생성 및 편집하기 전에 먼저 API를 만드십시오. 이 단원에서는 PetStore API를 예로 사용합니다. API Gateway 콘솔을 사용하여 API를 생성하려면 자습서: 예제를 가져와 REST API 생성의 지침을 따릅니다.

주제

- API 엔터티 문서화
- RESOURCE 엔터티 문서화
- METHOD 엔터티 문서화
- QUERY_PARAMETER 엔터티 문서화
- PATH_PARAMETER 엔터티 문서화
- REQUEST_BODY 엔터티 문서화
- REQUEST_HEADER 엔터티 문서화
- RESPONSE 엔터티 문서화
- RESPONSE_HEADER 엔터티 문서화
- AUTHORIZER 엔터티 문서화
- MODEL 엔터티 문서화
- 설명서 부분 업데이트
- 설명서 부분 나열

API 엔터티 문서화

API에 대한 설명서를 추가하려면 API 엔터티에 대한 DocumentationPart 리소스를 추가합니다.

```
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}"
}
```

성공하면 해당 작업은 페이로드에서 새로 생성된 `201 Created` 인스턴스를 포함하는 `DocumentationPart` 응답을 반환합니다. 예:

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

설명서 부분이 이미 추가된 경우 `409 Conflict`의 오류 메시지를 포함하여 `Documentation part already exists for the specified location: type 'API'."` 응답이 반환됩니다. 이 경우 documentationpart:update 작업을 호출해야 합니다.

```
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}"
  } ]
}
```

응답이 성공하면 업데이트된 200 OK 인스턴스를 포함한 페이로드와 함께 DocumentationPart 상태 코드를 반환합니다.

RESOURCE 엔터티 문서화

API의 루트 리소스에 대한 설명서를 추가하려면 해당 Resource 리소스를 대상으로 하는 DocumentationPart 리소스를 추가합니다.

```
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}"
}
```

성공하면 해당 작업은 페이로드에서 새로 생성된 201 Created 인스턴스를 포함하는 DocumentationPart 응답을 반환합니다. 예:

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

리소스 경로가 지정되지 않은 경우 리소스가 루트 리소스로 간주됩니다. `"path": "/"`를 `properties`에 추가하여 명시적으로 지정할 수 있습니다.

API의 하위 리소스에 대한 설명서를 추가하려면 해당 Resource 리소스를 대상으로 하는 DocumentationPart 리소스를 추가합니다.

```
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}"
}
```

성공하면 해당 작업은 페이로드에서 새로 생성된 `201 Created` 인스턴스를 포함하는 `DocumentationPart` 응답을 반환합니다. 예:

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

경로 파라미터로 지정된 하위 리소스에 대한 설명서를 추가하려면 Resource 리소스를 대상으로 하는 DocumentationPart 리소스를 추가합니다.

```
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}"
}
```

성공하면 해당 작업은 페이로드에서 새로 생성된 `201 Created` 인스턴스를 포함하는 `DocumentationPart` 응답을 반환합니다. 예:

```
{
  "_links": {
    "curies": {
      "href": "http://docs.aws.amazon.com/apigateway/latest/developerguide/restapi-
documentationpart-{rel}.html",
      "name": "documentationpart",
      "templated": true
    },
    "self": {
      "href": "/restapis/4wk1k4onj3/documentation/parts/k6fpwb"
    },
    "documentationpart:delete": {
      "href": "/restapis/4wk1k4onj3/documentation/parts/k6fpwb"
    },
    "documentationpart:update": {
      "href": "/restapis/4wk1k4onj3/documentation/parts/k6fpwb"
    }
  },
  "id": "k6fpwb",
  "location": {
    "path": "/pets/{petId}",
    "method": null,
    "name": null,
    "statusCode": null,
    "type": "RESOURCE"
  },
  "properties": "{\n\t\"description\" : \"A child resource specified by the petId path
 parameter.\"\n}"
}
```

> **Note**
>
> RESOURCE 엔터티의 DocumentationPart 인스턴스는 하위 리소스가 상속할 수 없습니다.

METHOD 엔터티 문서화

API의 메서드에 대한 설명서를 추가하려면 해당 Method 리소스를 대상으로 하는 DocumentationPart 리소스를 추가합니다.

```
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}"
}
```

성공하면 해당 작업은 페이로드에서 새로 생성된 `201 Created` 인스턴스를 포함하는 `DocumentationPart` 응답을 반환합니다. 예:

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

성공하면 해당 작업은 페이로드에서 새로 생성된 `201 Created` 인스턴스를 포함하는 `DocumentationPart` 응답을 반환합니다. 예:

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

이전 요청에서 location.method 필드가 지정되지 않은 경우 와일드카드 ANY 문자로 표현되는 * 메서드로 간주됩니다.

METHOD 엔터티의 설명서 콘텐츠를 업데이트하려면 documentationpart:update 작업을 호출하여 새 properties 맵을 제공합니다.

```
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}"
  } ]
}
```

응답이 성공하면 업데이트된 200 OK 인스턴스를 포함한 페이로드와 함께 DocumentationPart 상태 코드를 반환합니다. 예:

```
{
  "_links": {
    "curies": {
      "href": "http://docs.aws.amazon.com/apigateway/latest/developerguide/restapi-
documentationpart-{rel}.html",
      "name": "documentationpart",
      "templated": true
    },
    "self": {
      "href": "/restapis/4wk1k4onj3/documentation/parts/o64jbj"
    },
```

```
    "documentationpart:delete": {
      "href": "/restapis/4wk1k4onj3/documentation/parts/o64jbj"
    },
    "documentationpart:update": {
      "href": "/restapis/4wk1k4onj3/documentation/parts/o64jbj"
    }
  },
  "id": "o64jbj",
  "location": {
    "path": "/pets",
    "method": "GET",
    "name": null,
    "statusCode": null,
    "type": "METHOD"
  },
  "properties": "{\n\t\"tags\" : [ \"pets\" ], \n\t\"summary\" : \"List all pets.\"\n}"
}
```

QUERY_PARAMETER 엔터티 문서화

요청 쿼리 파라미터에 대한 설명서를 추가하려면 QUERY_PARAMETER 및 path의 유효한 필드를 사용하여 name 유형을 대상으로 하는 DocumentationPart 리소스를 추가합니다.

```
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}"
}
```

성공하면 해당 작업은 페이로드에서 새로 생성된 201 Created 인스턴스를 포함하는 DocumentationPart 응답을 반환합니다. 예:

```
{
  "_links": {
    "curies": {
      "href": "http://docs.aws.amazon.com/apigateway/latest/developerguide/restapi-
documentationpart-{rel}.html",
      "name": "documentationpart",
      "templated": true
    },
    "self": {
      "href": "/restapis/4wk1k4onj3/documentation/parts/h9ht5w"
    },
    "documentationpart:delete": {
      "href": "/restapis/4wk1k4onj3/documentation/parts/h9ht5w"
    },
    "documentationpart:update": {
      "href": "/restapis/4wk1k4onj3/documentation/parts/h9ht5w"
    }
  },
  "id": "h9ht5w",
  "location": {
    "path": "/pets",
    "method": "GET",
    "name": "page",
    "statusCode": null,
    "type": "QUERY_PARAMETER"
  },
  "properties": "{\n\t\"description\" : \"Page number of results to return.\"\n}"
}
```

properties 엔터티의 설명서 부분의 QUERY_PARAMETER 맵은 QUERY_PARAMETER 하위 엔터티 중 하나만 상속할 수 있습니다. 예를 들어, treats 다음에 /pets/{petId} 리소스를 추가하고, GET에서 /pets/{petId}/treats 메서드를 활성화하고, page 쿼리 파라미터를 표시한 경우, 사용자가 DocumentationPart 리소스를 properties 메서드의 GET /pets 쿼리 파라미터에 명시적으로 추가하지 않으면 이 하위 쿼리 파라미터는 DocumentationPart 메서드의 이름이 같은 쿼리 파라미터에서 page의 GET /pets/{petId}/treats 맵을 상속합니다.

PATH_PARAMETER 엔터티 문서화

경로 파라미터에 대한 설명서를 추가하려면 PATH_PARAMETER 엔터티에 대한 DocumentationPart 리소스를 추가합니다.

```
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}"
}
```

성공하면 해당 작업은 페이로드에서 새로 생성된 `201 Created` 인스턴스를 포함하는 `DocumentationPart` 응답을 반환합니다. 예:

```
{
  "_links": {
    "curies": {
      "href": "http://docs.aws.amazon.com/apigateway/latest/developerguide/restapi-
documentationpart-{rel}.html",
      "name": "documentationpart",
      "templated": true
    },
    "self": {
      "href": "/restapis/4wk1k4onj3/documentation/parts/ckpgog"
    },
    "documentationpart:delete": {
      "href": "/restapis/4wk1k4onj3/documentation/parts/ckpgog"
    },
    "documentationpart:update": {
      "href": "/restapis/4wk1k4onj3/documentation/parts/ckpgog"
    }
  },
  "id": "ckpgog",
  "location": {
    "path": "/pets/{petId}",
    "method": "*",
    "name": "petId",
```

```
    "statusCode": null,
    "type": "PATH_PARAMETER"
  },
  "properties": "{\n  \"description\" : \"The id of the pet to retrieve\"\n}"
}
```

REQUEST_BODY 엔터티 문서화

요청 본문에 대한 설명서를 추가하려면 요청 본문에 대한 DocumentationPart 리소스를 추가합니다.

```
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}"
}
```

성공하면 해당 작업은 페이로드에서 새로 생성된 `201 Created` 인스턴스를 포함하는 `DocumentationPart` 응답을 반환합니다. 예:

```
{
  "_links": {
    "curies": {
      "href": "http://docs.aws.amazon.com/apigateway/latest/developerguide/restapi-
documentationpart-{rel}.html",
      "name": "documentationpart",
      "templated": true
    },
    "self": {
      "href": "/restapis/4wk1k4onj3/documentation/parts/kgmfr1"
    },
    "documentationpart:delete": {
      "href": "/restapis/4wk1k4onj3/documentation/parts/kgmfr1"
```

```
    },
    "documentationpart:update": {
      "href": "/restapis/4wk1k4onj3/documentation/parts/kgmfr1"
    }
  },
  "id": "kgmfr1",
  "location": {
    "path": "/pets",
    "method": "POST",
    "name": null,
    "statusCode": null,
    "type": "REQUEST_BODY"
  },
  "properties": "{\n\t\"description\" : \"A Pet object to be added to PetStore.\"\n}"
}
```

REQUEST_HEADER 엔터티 문서화

요청 헤더에 대한 설명서를 추가하려면 요청 헤더에 대한 DocumentationPart 리소스를 추가합니다.

```
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}"
}
```

성공하면 해당 작업은 페이로드에서 새로 생성된 `201 Created` 인스턴스를 포함하는 `DocumentationPart` 응답을 반환합니다. 예:

```
{
```

```
  "_links": {
    "curies": {
      "href": "http://docs.aws.amazon.com/apigateway/latest/developerguide/restapi-
documentationpart-{rel}.html",
      "name": "documentationpart",
      "templated": true
    },
    "self": {
      "href": "/restapis/4wk1k4onj3/documentation/parts/h0m3uf"
    },
    "documentationpart:delete": {
      "href": "/restapis/4wk1k4onj3/documentation/parts/h0m3uf"
    },
    "documentationpart:update": {
      "href": "/restapis/4wk1k4onj3/documentation/parts/h0m3uf"
    }
  },
  "id": "h0m3uf",
  "location": {
    "path": "/pets",
    "method": "GET",
    "name": "x-my-token",
    "statusCode": null,
    "type": "REQUEST_HEADER"
  },
  "properties": "{\n\t\"description\" : \"A custom token used to authorization the
 method invocation.\"\n}"
}
```

RESPONSE 엔터티 문서화

상태 코드의 응답에 대한 설명서를 추가하려면 해당 MethodResponse 리소스를 대상으로 하는 DocumentationPart 리소스를 추가합니다.

```
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}"
}
```

성공하면 해당 작업은 페이로드에서 새로 생성된 `201 Created` 인스턴스를 포함하는 `DocumentationPart` 응답을 반환합니다. 예:

```
{
    "_links": {
        "self": {
            "href": "/restapis/4wk1k4onj3/documentation/parts/lattew"
        },
        "documentationpart:delete": {
            "href": "/restapis/4wk1k4onj3/documentation/parts/lattew"
        },
        "documentationpart:update": {
            "href": "/restapis/4wk1k4onj3/documentation/parts/lattew"
        }
    },
    "id": "lattew",
    "location": {
        "path": "/",
        "method": "*",
        "name": null,
        "statusCode": "200",
        "type": "RESPONSE"
    },
    "properties": "{\n  \"description\" : \"Successful operation.\"\n}"
}
```

`RESPONSE_HEADER` 엔터티 문서화

응답 헤더에 대한 설명서를 추가하려면 응답 헤더에 대한 DocumentationPart 리소스를 추가합니다.

```
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}"
```

성공하면 해당 작업은 페이로드에서 새로 생성된 `201 Created` 인스턴스를 포함하는 `DocumentationPart` 응답을 반환합니다. 예:

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

이 Content-Type 응답 헤더의 설명서는 모든 API 응답의 Content-Type 헤더에 대한 기본 설명서입니다.

AUTHORIZER 엔터티 문서화

API 권한 부여자에 대한 설명서를 추가하려면 지정된 권한 부여자를 대상으로 하는 DocumentationPart 리소스를 추가합니다.

```
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}"
}
```

성공하면 해당 작업은 페이로드에서 새로 생성된 201 Created 인스턴스를 포함하는 DocumentationPart 응답을 반환합니다. 예:

```
{
  "_links": {
    "curies": {
      "href": "http://docs.aws.amazon.com/apigateway/latest/developerguide/restapi-
documentationpart-{rel}.html",
      "name": "documentationpart",
      "templated": true
    },
    "self": {
      "href": "/restapis/4wk1k4onj3/documentation/parts/pw3qw3"
    },
    "documentationpart:delete": {
```

```
      "href": "/restapis/4wk1k4onj3/documentation/parts/pw3qw3"
    },
    "documentationpart:update": {
      "href": "/restapis/4wk1k4onj3/documentation/parts/pw3qw3"
    }
  },
  "id": "pw3qw3",
  "location": {
    "path": null,
    "method": null,
    "name": "myAuthorizer",
    "statusCode": null,
    "type": "AUTHORIZER"
  },
  "properties": "{\n\t\"description\" : \"Authorizes invocations of configured methods.
\"\n}"
}
```

> **Note**
>
> AUTHORIZER 엔터티의 DocumentationPart 인스턴스는 하위 리소스가 상속할 수 없습니다.

MODEL 엔터티 문서화

MODEL 엔터티를 문서화하려면 해당 모델의 DocumentPart 인스턴스 및 각각의 properties를 생성하고 관리해야 합니다. 예를 들어, 기본적으로 모든 API와 함께 제공되는 Error 모델에는 다음과 같은 스키마 정의가 있습니다.

```
{
  "$schema" : "http://json-schema.org/draft-04/schema#",
  "title" : "Error Schema",
  "type" : "object",
  "properties" : {
    "message" : { "type" : "string" }
  }
}
```

그리고 두 개의 DocumentationPart 인스턴스가 필요합니다. 하나는 Model용이고 다른 하나는 message 속성용입니다.

```
{
```

```
  "location": {
    "type": "MODEL",
    "name": "Error"
  },
  "properties": {
    "title": "Error Schema",
    "description": "A description of the Error model"
  }
}
```

및

```
{
  "location": {
    "type": "MODEL",
    "name": "Error.message"
  },
  "properties": {
    "description": "An error message."
  }
}
```

API를 내보낼 경우 DocumentationPart'의 속성이 원본 스키마의 값을 재정의합니다.

API 모델에 대한 설명서를 추가하려면 지정된 모델을 대상으로 하는 DocumentationPart 리소스를 추가합니다.

```
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}"
}
```

성공하면 해당 작업은 페이로드에서 새로 생성된 `201 Created` 인스턴스를 포함하는 `DocumentationPart` 응답을 반환합니다. 예:

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

동일한 단계를 반복하여 모델의 속성에 대한 DocumentationPart 인스턴스를 생성합니다.

> **Note**
>
> `MODEL` 엔터티의 DocumentationPart 인스턴스는 하위 리소스가 상속할 수 없습니다.

설명서 부분 업데이트

모든 API 엔터티 유형의 설명서 부분을 업데이트하려면 지정된 부분 식별자의 DocumentationPart 인스턴스에서 PATCH 요청을 제출하여 기존 properties 맵을 새로운 맵으로 교체합니다.

```
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"
  } ]
}
```

응답이 성공하면 업데이트된 200 OK 인스턴스를 포함한 페이로드와 함께 DocumentationPart 상태 코드를 반환합니다.

단일 PATCH 요청으로 여러 설명서 부분을 업데이트할 수 있습니다.

설명서 부분 나열

모든 API 엔터티 유형의 설명서 부분을 나열하려면 DocumentationParts 컬렉션에서 GET 요청을 제출합니다.

```
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
```

응답이 성공하면 사용 가능한 200 OK 인스턴스를 포함한 페이로드와 함께 DocumentationPart 상태 코드를 반환합니다.

API Gateway REST API를 사용하여 API 설명서 게시

API에 대한 설명서를 게시하려면 설명서 스냅샷을 생성 또는 업데이트하거나 가져온 다음, 설명서 스냅샷을 API 단계와 연결합니다. 설명서 스냅샷을 생성할 때 이를 동시에 API 단계와 연결할 수도 있습니다.

주제

- 설명서 스냅샷을 생성하여 API 단계와 연결
- 설명서 스냅샷 만들기
- 설명서 스냅샷 업데이트
- 설명서 스냅샷 가져오기
- 설명서 스냅샷을 API 단계와 연결
- 단계와 연결된 설명서 스냅샷 다운로드

설명서 스냅샷을 생성하여 API 단계와 연결

API의 설명서 부분의 스냅샷을 생성하여 동시에 API 단계와 연결하려면 다음 POST 요청을 제출하십시오.

```
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"
}
```

성공하면 해당 작업은 페이로드로 새로 생성된 `200 OK` 인스턴스를 포함하는 `DocumentationVersion` 응답을 반환합니다.

또는 API 단계와 먼저 연결하지 않고 설명서 스냅샷을 생성한 다음, restapi:update를 호출하여 스냅샷을 지정된 API 단계와 연결할 수 있습니다. 기존 설명서 스냅샷을 업데이트 또는 쿼리한 다음, 단계 연결을 업데이트할 수도 있습니다. 다음 4개 단원에서 단계를 보여드립니다.

설명서 스냅샷 만들기

API 설명서 부분의 스냅샷을 생성하려면 새 DocumentationVersion 리소스를 생성하여 API의 DocumentationVersions 컬렉션에 추가합니다.

```
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"
}
```

성공하면 해당 작업은 페이로드로 새로 생성된 200 OK 인스턴스를 포함하는 DocumentationVersion 응답을 반환합니다.

설명서 스냅샷 업데이트

해당 DocumentationVersion 리소스의 description 속성을 수정하는 방법으로만 설명서 스냅샷을 업데이트할 수 있습니다. 다음 예에서는 *version* 버전 ID(예: 1.0.0)로 식별된 설명서 스냅샷의 설명을 업데이트하는 방법을 보여줍니다.

```
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."
    }]
}
```

성공하면 해당 작업은 페이로드로 업데이트된 `200 OK` 인스턴스를 포함하는 `DocumentationVersion` 응답을 반환합니다.

설명서 스냅샷 가져오기

설명서 스냅샷을 가져오려면 지정된 DocumentationVersion 리소스에 대해 `GET` 요청을 제출합니다. 다음 예에서는 지정된 버전 ID 1.0.0의 설명서 스냅샷을 가져오는 방법을 보여줍니다.

```
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
```

설명서 스냅샷을 API 단계와 연결

API 설명서를 게시하려면 설명서 스냅샷을 API 단계와 연결합니다. 설명서 버전을 단계와 연결하기 전에 API 단계를 이미 생성해야 합니다.

API Gateway REST API를 사용하여 설명서 스냅샷을 API 단계와 연결하려면 stage:update 작업을 호출하여 `stage.documentationVersion` 속성에서 원하는 설명서 버전을 설정합니다.

```
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"
    }]
}
```

단계와 연결된 설명서 스냅샷 다운로드

설명서 부분의 한 버전이 단계와 연결된 후, API Gateway 콘솔, API Gateway REST API, SDK 중 하나 또는 API Gateway용 AWS CLI를 사용하여 API 엔터티 정의와 함께 설명서 부분을 외부 파일로 내보낼 수 있습니다. 이 프로세스는 API를 내보내는 프로세스와 동일합니다. 내보낸 파일 형식은 JSON 또는 YAML일 수 있습니다.

또한 API Gateway REST API를 사용하여 API 설명서 부분, API 통합 및 권한 부여자를 API 내보내기에 포함하도록 `extension=documentation,integrations,authorizers` 쿼리 파라미터를 명시적으로 설정할 수도 있습니다. 기본적으로 API를 내보낼 때 설명서 부분은 포함되지만 통합 및 권한 부여자는 제외됩니다. API 내보내기의 기본 출력은 설명서 배포에 적합합니다.

API Gateway REST API를 사용하여 API 설명서를 외부 JSON OpenAPI 파일로 내보내려면 다음 `GET` 요청을 제출합니다.

```
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
```

여기서 `x-amazon-apigateway-documentation` 객체에는 설명서 부분이 포함되고 API 엔터티 정의에는 OpenAPI에서 지원하는 설명서 속성이 포함됩니다. 출력에는 통합 또는 Lambda 권한 부여자(이전에는 사용자 지정 권한 부여자라고 함)의 세부 정보가 포함되지 않습니다. 두 가지 세부 정보를 모두 포함하려면 `extensions=integrations,authorizers,documentation`을 설정하십시오. 통합 세부 정보는 포함하지만 권한 부여자 세부 정보는 포함하지 않으려면 `extensions=integrations,documentation`을 설정하십시오.

결과를 JSON 파일로 출력하려면 요청에 `Accept:application/json` 헤더를 설정해야 합니다. YAML 출력을 생성하려면 요청 헤더를 `Accept:application/yaml`로 변경하십시오.

예를 들어, 루트 리소스(`GET`)에서 간단한 `/` 메서드를 표시하는 API를 볼 수 있습니다. 이 API에는 OpenAPI 정의 파일에 정의된 API 엔터티가 네 개 있습니다. 각각 `API`, `MODEL`, `METHOD` 및 `RESPONSE` 유형에 해당하는 엔터티입니다. 설명서 부분이 각 `API`, `METHOD` 및 `RESPONSE` 엔터티에 추가되었습니다. 이전 설명서 내보내기 명령을 호출하면 설명서 부분이 `x-amazon-apigateway-`

documentation 객체에 표준 OpenAPI 파일에 대한 확장으로 나열되는 다음과 같은 출력이 표시됩니다.

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"
}
```

설명서 부분의 `properties` 맵에 정의된 OpenAPI 호환 속성의 경우 API Gateway는 연결된 API 엔터티 정의에 속성을 삽입합니다. `x-`*`something`*의 속성은 표준 OpenAPI 확장입니다. 이 확장은 API 엔터티 정의에 전파됩니다. 예를 들어, `x-example` 메서드에 대한 `GET` 속성을 참조하십시오. `foo` 같은 속성은 OpenAPI 사양의 일부가 아니고 연결된 API 엔터티 정의에 삽입되지 않습니다.

설명서 렌더링 도구(예: OpenAPI UI)가 API 엔터티 정의를 구문 분석하여 설명서 속성을 추출할 경우, `properties` 인스턴스의 Swagger 비호환 `DocumentationPart` 속성은 도구에서 사용할 수 없습니다. 그러나 설명서 렌더링 도구에서 `x-amazon-apigateway-documentation` 객체를 구문 분석하여 콘텐츠를 가져오거나 도구에서 restapi:documentation-parts 및 documenationpart:by-id를 호출하여 API Gateway에서 설명서 부분을 가져올 경우, 모든 설명서 속성을 도구에 사용하여 표시할 수 있습니다.

통합 세부 정보를 포함하는 API 엔터티가 있는 설명서를 JSON OpenAPI 파일로 내보내려면 다음 `GET` 요청을 제출하십시오.

```
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
```

통합 및 권한 부여자 세부 정보를 포함하는 API 엔터티가 있는 설명서를 YAML OpenAPI 파일로 내보내려면 다음 GET 요청을 제출하십시오.

```
GET /restapis/restapi_id/stages/stage_name/exports/swagger?
extensions=integrations,authorizers,documentation HTTP/1.1
Accept: application/yaml
Host: apigateway.region.amazonaws.com
Content-Type: application/json
X-Amz-Date: YYYYMMDDTttttttZ
Authorization: AWS4-HMAC-SHA256 Credential=access_key_id/YYYYMMDD/region/
apigateway/aws4_request, SignedHeaders=content-length;content-type;host;x-amz-date,
 Signature=sigv4_secret
```

API Gateway 콘솔을 사용하여 API의 게시된 설명서를 내보내거나 다운로드하려면 API Gateway 콘솔을 사용하여 REST API 내보내기의 지침을 따릅니다.

API 설명서 가져오기

API 엔터티 정의 가져오기와 마찬가지로 외부 OpenAPI 파일에서 API Gateway의 API로 설명서 부분을 가져올 수 있습니다. 유효한 OpenAPI 정의 파일의 x-amazon-apigateway-documentation 객체 확장 내에서 가져올 설명서 부분을 지정합니다. 설명서를 가져와도 기존 API 엔터티 정의는 변경되지 않습니다.

새로 지정된 설명서 부분을 API Gateway의 기존 설명서 부분에 병합하거나 기존 설명서 부분을 덮어쓸 수 있습니다. MERGE 모드에서는 OpenAPI 파일에 정의된 새 설명서 부분이 API의 DocumentationParts 컬렉션에 추가됩니다. 가져온 DocumentationPart가 이미 있는 경우 두 개의 속성이 다르면 가져온 속성이 기존 속성을 대체합니다. 다른 기존 설명서 속성은 그대로 남아 있습니다. OVERWRITE 모드에서는 전체 DocumentationParts 컬렉션이 가져온 OpenAPI 정의 파일에 따라 대체됩니다.

API Gateway REST API를 사용하여 설명서 부분 가져오기

API Gateway REST API를 사용하여 API 설명서를 가져오려면 documentationpart:import 작업을 호출합니다. 다음 예에서는 API의 기존 설명서 부분을 단일 `GET /` 메서드로 덮어쓰고 성공하면 `200 OK` 응답을 반환하는 방법을 보여줍니다.

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"
        }
      }
    ]
  }
}
```

성공하면 이 요청은 페이로드에 가져온 `DocumentationPartId`를 포함하는 200 OK 응답을 반환합니다.

```
{
  "ids": [
    "kg3mth",
    "796rtf",
    "zhek4p",
    "5ukm9s"
  ]
}
```

또한 restapi:import 또는 restapi:put을 호출하고 `x-amazon-apigateway-documentation` 객체의 설명서 부분을 API 정의의 입력 OpenAPI 파일의 부분으로 제공할 수도 있습니다. API 가져오기에서 설명서 부분을 제외하려면 요청 쿼리 파라미터에서 `ignore=documentation`을 설정하십시오.

API Gateway 콘솔을 사용하여 문서 부분 가져오기

다음 지침에서는 설명서 부분을 가져오는 방법을 설명합니다.

콘솔을 사용하여 외부 파일로부터 API의 설명서 부분 가져오는 방법

1. 기본 탐색 창에서 설명서를 선택합니다.
2. 가져오기를 선택합니다.
3. 기존 설명서가 있는 경우 새 문서에 대한 덮어쓰기 또는 병합 선택합니다.
4. 파일 선택을 선택하여 드라이브로부터 파일을 로드하거나 파일 내용을 파일 보기에 입력합니다. 예를 들어, API Gateway REST API를 사용하여 설명서 부분 가져오기에 포함된 예제 요청의 페이로드를 참조합니다.
5. 가져올 때 경고를 처리하는 방법을 선택합니다. 경고 실패 또는 경고 무시를 선택합니다. 자세한 내용은 the section called “가져오는 중 발생하는 오류 및 경고” 섹션을 참조하세요.
6. [Import]를 선택합니다.

API 설명서에 대한 액세스 제어

API 설명서를 작성 및 편집하는 전담 설명서 팀이 있는 경우 개발자(API 개발) 및 작성자 또는 편집자(콘텐츠 개발)에 대한 별도의 액세스 권한을 구성할 수 있습니다. 이는 설명서 작성에 타사 공급업체가 관련된 경우 특히 해당합니다.

설명서 팀에게 API 설명서를 생성, 업데이트 및 게시할 수 있는 액세스 권한을 부여하려면 다음 IAM 정책을 사용하여 설명서 팀에게 IAM 역할을 할당할 수 있습니다. 여기서 *account_id*는 설명서 팀의 AWS 계정 ID입니다.

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

    {
      "Sid": "StmtDocPartsAddEditViewDelete",
      "Effect": "Allow",
      "Action": [
        "apigateway:GET",
        "apigateway:PUT",
        "apigateway:POST",
        "apigateway:PATCH",
        "apigateway:DELETE"
      ],
      "Resource": [
        "arn:aws:apigateway::account_id:/restapis/*/documentation/*"
      ]
```

```
    }
  ]
}
```

API Gateway 리소스에 액세스할 수 있는 권한을 설정하는 방법에 대한 자세한 내용은 the section called “Amazon API Gateway에서 IAM을 사용하는 방식” 단원을 참조하세요.

API Gateway에서 REST API용 SDK 생성

REST API를 플랫폼별 또는 언어별 방식으로 호출하려면, API의 플랫폼별 또는 언어별 SDK를 생성해야 합니다. 현재 API Gateway에서는 Java, JavaScript, Java for Android, iOS용 Objective-C 또는 Swift, Ruby로 API용 SDK를 생성할 수 있습니다.

이 단원에서는 API Gateway API의 SDK를 생성하는 방법에 대해 설명합니다. 또한 Java 앱, Android용 Java 앱, iOS용 Objective-C 및 Swift 앱 및 JavaScript 앱에서 생성된 SDK를 사용하는 방법을 보여줍니다.

원활한 설명을 위해 이 간편 계산기 Lambda 함수를 표시하는 이 API Gateway API를 사용합니다.

계속하기 전에 API Gateway에서 API를 생성하거나 가져와서 최소 한 번 이상 배포하십시오. 지침은 Amazon API Gateway에서 REST API 배포 단원을 참조하십시오.

주제

- API Gateway 콘솔을 사용하여 API용 SDK 생성
- AWS CLI 명령을 사용하여 API용 SDK 생성
- 간편 계산기 Lambda 함수
- API Gateway의 간편 계산기 API
- 간편 계산기 API OpenAPI 정의

API Gateway 콘솔을 사용하여 API용 SDK 생성

API Gateway에서 플랫폼별 또는 언어별로 API용 SDK를 생성하려면, 먼저 API를 생성하고 테스트하여 단계에 배포해야 합니다. 설명을 돕기 위해 이 단원에서는 언어별 SDK 또는 플랫폼별 SDK를 생성하는 간편 계산기 API를 예로 들어 보겠습니다. 이 API를 생성, 테스트, 배포하는 방법에 대한 지침은 간편 계산기 API 생성 단원을 참조하십시오.

주제

- API의 Java SDK 생성
- API의 Android SDK 생성
- API의 iOS SDK 생성
- REST API의 JavaScript SDK 생성
- API의 Ruby SDK 생성

API의 Java SDK 생성

API Gateway에서 API의 Java SDK를 생성하려면

1. https://console.aws.amazon.com/apigateway에서 API Gateway 콘솔에 로그인합니다.
2. REST API를 선택합니다.
3. 단계를 선택합니다.
4. 스테이지 창에서 스테이지의 이름을 선택합니다.
5. 스테이지 작업 메뉴를 연 다음 SDK 생성을 선택합니다.
6. 플랫폼에서 Java 플랫폼을 선택하고 다음을 수행합니다.

    a. 서비스 이름에 SDK 이름을 지정합니다. 예: **`SimpleCalcSdk`**. 이 이름이 SDK 클라이언트 클래스 이름이 됩니다. 이 이름은 SDK 프로젝트 폴더에 있는 pom.xml 파일에서 <name> 아래에 있는 <project> 태그에 해당합니다. 하이픈이 포함되면 안 됩니다.

    b. Java 패키지 이름(Java Package Name)에 SDK의 패키지 이름을 지정합니다. 예: **`examples.aws.apig.simpleCalc.sdk`**. 이 패키지 이름은 SDK 라이브러리의 네임스페이스로 사용됩니다. 하이픈이 포함되면 안 됩니다.

    c. Java Build System(Java 빌드 시스템)에 **`maven`** 또는 **`gradle`**을 입력하여 빌드 시스템을 지정합니다.

    d. Java Group Id(Java 그룹 ID)에 SDK 프로젝트의 그룹 식별자를 입력합니다. 예를 들면 **`my-apig-api-examples`**를 입력합니다. 이 식별자는 SDK 프로젝트 폴더에 있는 <groupId> 파일의 <project> 아래에 있는 `pom.xml` 태그에 해당합니다.

    e. Java Artifact Id(Java 아티팩트 ID)에 SDK 프로젝트의 아티팩트 식별자를 입력합니다. 예를 들면 **`simple-calc-sdk`**를 입력합니다. 이 식별자는 SDK 프로젝트 폴더에 있는 <artifactId> 파일의 <project> 아래에 있는 `pom.xml` 태그에 해당합니다.

    f. Java Artifact Version(Java 아티팩트 버전)에 버전 식별자 문자열을 입력합니다. 예: **`1.0.0`**. 이 버전 식별자는 SDK 프로젝트 폴더에 있는 <version> 파일의 <project> 아래에 있는 `pom.xml` 태그에 해당합니다.

g. Source Code License Text(소스 코드 라이선스 텍스트)에 소스 코드의 라이선스 텍스트를 입력합니다(해당하는 경우).

7. SDK 생성를 선택한 다음, 화면 지침에 따라 API Gateway에서 생성된 SDK를 다운로드합니다.

생성된 SDK를 사용하려면 API Gateway에서 생성한 REST API용 Java SDK 사용의 지침을 따르십시오.

API를 업데이트할 때마다 해당 API를 재배포하고 SDK를 재생성하여 업데이트 내용이 포함되도록 해야 합니다.

API의 Android SDK 생성

API Gateway에서 API의 Android SDK를 생성하려면

1. https://console.aws.amazon.com/apigateway에서 API Gateway 콘솔에 로그인합니다.
2. REST API를 선택합니다.
3. 단계를 선택합니다.
4. 스테이지 창에서 스테이지의 이름을 선택합니다.
5. 스테이지 작업 메뉴를 연 다음 SDK 생성을 선택합니다.
6. 플랫폼에서 Android 플랫폼을 선택하고 다음을 수행합니다.

   a. 그룹 ID에 해당 프로젝트의 고유 식별자를 입력합니다. 이는 `pom.xml` 파일에 사용됩니다(예: **`com.mycompany`**).

   b. Invoker package(호출자 패키지)에 생성된 클라이언트 클래스의 네임스페이스를 입력합니다(예: **`com.mycompany.clientsdk`**).

   c. Artifact ID(아티팩트 ID)에 버전 없이 컴파일된 .jar 파일의 이름을 입력합니다. 이는 `pom.xml` 파일에 사용됩니다(예: **`aws-apigateway-api-sdk`**).

   d. Artifact version(아티팩트 버전)에 생성된 클라이언트의 아티팩트 버전 번호를 입력합니다. 이는 `pom.xml` 파일에 사용되며 *`major.minor.patch`* 패턴을 따라야 합니다(예: **`1.0.0`**)

7. SDK 생성를 선택한 다음, 화면 지침에 따라 API Gateway에서 생성된 SDK를 다운로드합니다.

생성된 SDK를 사용하려면 API Gateway에서 생성한 REST API용 Android SDK 사용의 지침을 따르십시오.

API를 업데이트할 때마다 해당 API를 재배포하고 SDK를 재생성하여 업데이트 내용이 포함되도록 해야 합니다.

API의 iOS SDK 생성

API Gateway에서 API의 iOS SDK를 생성하려면

1. https://console.aws.amazon.com/apigateway에서 API Gateway 콘솔에 로그인합니다.
2. REST API를 선택합니다.
3. 단계를 선택합니다.
4. 스테이지 창에서 스테이지의 이름을 선택합니다.
5. 스테이지 작업 메뉴를 연 다음 SDK 생성을 선택합니다.
6. 플랫폼에서 iOS(Objective-C) 또는 iOS(Swift) 플랫폼을 선택하고 다음을 수행합니다.
   - 접두사 상자에 고유한 접두사를 입력합니다.

     접두사의 효과는 다음과 같습니다. 예를 들어 `input`, `output`, 및 `result` 모델을 이용해 **`SIMPLE_CALC`**를 SimpleCalc API의 SDK에 대한 접두사로 할당하는 경우, 생성된 SDK에는 메서드 요청/응답을 포함해 API를 캡슐화하는 `SIMPLE_CALCSimpleCalcClient` 클래스가 포함됩니다. 또한 생성된 SDK에는 각각 입력, 출력, 결과를 나타내는 `SIMPLE_CALCinput`, `SIMPLE_CALCoutput`, `SIMPLE_CALCresult` 클래스가 포함되어, 요청 입력과 응답 출력을 나타냅니다. 자세한 내용은 Objective-C 또는 Swift에서 API Gateway에 의해 생성되는 REST API용 iOS SDK 사용 단원을 참조하세요.
7. SDK 생성를 선택한 다음, 화면 지침에 따라 API Gateway에서 생성된 SDK를 다운로드합니다.

생성된 SDK를 사용하려면 Objective-C 또는 Swift에서 API Gateway에 의해 생성되는 REST API용 iOS SDK 사용의 지침을 따르십시오.

API를 업데이트할 때마다 해당 API를 재배포하고 SDK를 재생성하여 업데이트 내용이 포함되도록 해야 합니다.

REST API의 JavaScript SDK 생성

API Gateway에서 API의 JavaScript SDK를 생성하려면

1. https://console.aws.amazon.com/apigateway에서 API Gateway 콘솔에 로그인합니다.
2. REST API를 선택합니다.
3. 단계를 선택합니다.
4. 스테이지 창에서 스테이지의 이름을 선택합니다.

5. 스테이지 작업 메뉴를 연 다음 SDK 생성을 선택합니다.
6. 플랫폼에서 JavaScript 플랫폼을 선택합니다.
7. SDK 생성를 선택한 다음, 화면 지침에 따라 API Gateway에서 생성된 SDK를 다운로드합니다.

생성된 SDK를 사용하려면 API Gateway에서 생성한 REST API용 JavaScript SDK 사용의 지침을 따르십시오.

API를 업데이트할 때마다 해당 API를 재배포하고 SDK를 재생성하여 업데이트 내용이 포함되도록 해야 합니다.

API의 Ruby SDK 생성

API Gateway에서 API의 Ruby SDK를 생성하려면

1. https://console.aws.amazon.com/apigateway에서 API Gateway 콘솔에 로그인합니다.
2. REST API를 선택합니다.
3. 단계를 선택합니다.
4. 스테이지 창에서 스테이지의 이름을 선택합니다.
5. 스테이지 작업 메뉴를 연 다음 SDK 생성을 선택합니다.
6. 플랫폼에서 Ruby 플랫폼을 선택하고 다음을 수행합니다.

    a. 서비스 이름에 SDK 이름을 지정합니다. 예: **`SimpleCalc`**. 이것은 API의 Ruby Gem 네임스페이스를 생성하는 데 사용됩니다. 이름은 다른 특수 문자나 숫자 없이 모두 문자(`a-zA-Z`)여야 합니다.

    b. Ruby Gem 이름(Ruby Gem Name)에서 API를 위해 생성된 SDK 소스 코드를 포함하도록 Ruby Gem 이름을 지정합니다. 기본적으로 이름은 소문자 서비스 이름에 `-sdk` 접미사가 붙습니다(예: **`simplecalc-sdk`**).

    c. Ruby Gem 버전(Ruby Gem Version)에서 생성된 Ruby Gem의 버전 번호를 지정합니다. 기본적으로 `1.0.0`로 설정됩니다.
7. SDK 생성를 선택한 다음, 화면 지침에 따라 API Gateway에서 생성된 SDK를 다운로드합니다.

생성된 SDK를 사용하려면 API Gateway에서 생성한 REST API용 Ruby SDK 사용의 지침을 따르십시오.

API를 업데이트할 때마다 해당 API를 재배포하고 SDK를 재생성하여 업데이트 내용이 포함되도록 해야 합니다.

AWS CLI 명령을 사용하여 API용 SDK 생성

AWS CLI를 사용하면 get-sdk 명령을 호출하여 지원되는 플랫폼의 API용 SDK를 생성하고 다운로드할 수 있습니다. 지원되는 일부 플랫폼에서 이렇게 하는 방법이 아래에 나와 있습니다.

주제

- AWS CLI를 사용하여 Android SDK용 Java를 생성 및 다운로드
- AWS CLI를 사용하여 JavaScript SDK를 생성 및 다운로드
- AWS CLI를 사용하여 Ruby SDK를 생성 및 다운로드

AWS CLI를 사용하여 Android SDK용 Java를 생성 및 다운로드

지정된 단계(udpuvvzbkc)에서 API(test)의 API Gateway가 생성하는 Java for Android SDK를 생성하고 다운로드하려면 다음과 같이 명령을 호출합니다.

```
aws apigateway get-sdk \
            --rest-api-id udpuvvzbkc \
            --stage-name test \
            --sdk-type android \
            --parameters groupId='com.mycompany',\
                invokerPackage='com.mycompany.myApiSdk',\
                artifactId='myApiSdk',\
                artifactVersion='0.0.1' \
            ~/apps/myApi/myApi-android-sdk.zip
```

~/apps/myApi/myApi-android-sdk.zip의 마지막 입력은 myApi-android-sdk.zip이라는 다운로드한 SDK 파일의 경로입니다.

AWS CLI를 사용하여 JavaScript SDK를 생성 및 다운로드

지정된 단계(udpuvvzbkc)에서 API(test)의 API Gateway가 생성하는 JavaScript SDK를 생성하고 다운로드하려면 다음과 같이 명령을 호출합니다.

```
aws apigateway get-sdk \
            --rest-api-id udpuvvzbkc \
            --stage-name test \
            --sdk-type javascript \
            ~/apps/myApi/myApi-js-sdk.zip
```

~/apps/myApi/myApi-js-sdk.zip의 마지막 입력은 myApi-js-sdk.zip이라는 다운로드한 SDK 파일의 경로입니다.

AWS CLI를 사용하여 Ruby SDK를 생성 및 다운로드

특정 단계(udpuvvzbkc)에 있는 API(test)의 Ruby SDK를 생성하고 다운로드하려면 다음과 같이 명령을 호출합니다.

```
aws apigateway get-sdk \
            --rest-api-id udpuvvzbkc \
            --stage-name test  \
            --sdk-type ruby \
            --parameters service.name=myApiRubySdk,ruby.gem-name=myApi,ruby.gem-
version=0.01 \
            ~/apps/myApi/myApi-ruby-sdk.zip
```

~/apps/myApi/myApi-ruby-sdk.zip의 마지막 입력은 myApi-ruby-sdk.zip이라는 다운로드한 SDK 파일의 경로입니다.

그런 다음 생성된 SDK를 사용하여 기본 API를 호출하는 방법을 살펴보겠습니다. 자세한 내용은 생성된 SDK를 통해 REST API 호출 섹션을 참조하세요.

간편 계산기 Lambda 함수

예를 들어 더하기, 빼기, 곱하기, 나누기의 이진 연산을 수행하는 Node.js Lambda 함수를 사용해 보겠습니다.

주제

- 간편 계산기 Lambda 함수 입력 형식
- 간편 계산기 Lambda 함수 출력 형식
- 간편 계산기 Lambda 함수 구현

간편 계산기 Lambda 함수 입력 형식

이 함수의 입력 형식은 다음과 같습니다.

```
{ "a": "Number", "b": "Number", "op": "string"}
```

여기서 op는 (+, -, *, /, add, sub, mul, div) 중 하나일 수 있습니다.

간편 계산기 Lambda 함수 출력 형식

작업이 성공할 경우 반환되는 결과 형식은 다음과 같습니다.

```
{ "a": "Number", "b": "Number", "op": "string", "c": "Number"}
```

여기서 c는 계산 결과를 유지합니다.

간편 계산기 Lambda 함수 구현

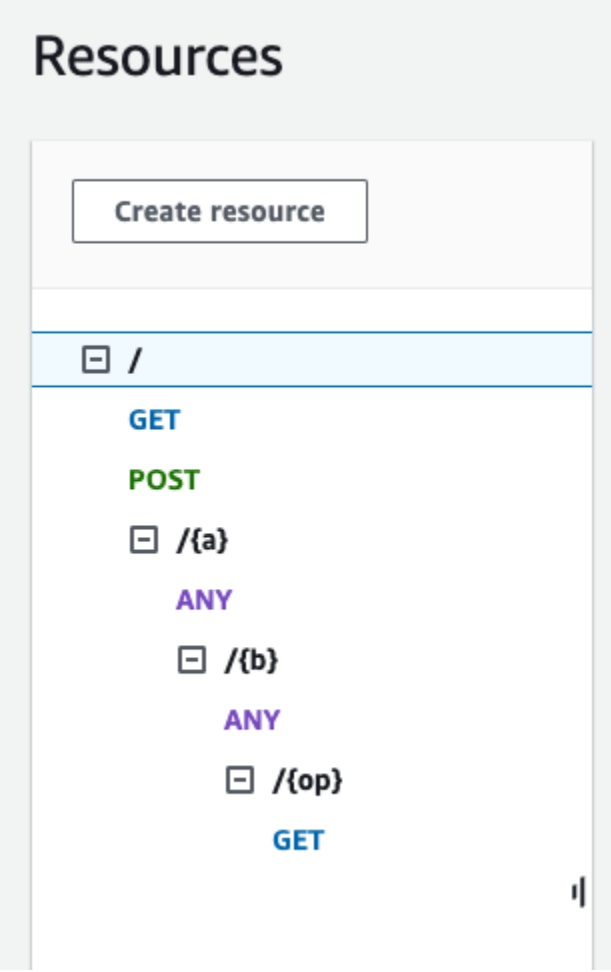
Lambda 함수의 구현은 다음과 같습니다.

```
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 Gateway의 간편 계산기 API

간편 계산기 API는 the section called “간편 계산기 Lambda 함수”을 호출하는 세 가지 메서드(GET, POST, GET)를 제공합니다. 이 API를 그래픽으로 나타내면 다음과 같습니다.



다음 세 가지 메서드는 다양한 방식으로 백엔드 Lambda 함수에 대한 입력을 제공하여 동일한 작업을 수행할 수 있음을 보여 줍니다.

- `GET /?a=...&b=...&op=...` 메서드는 쿼리 파라미터를 사용하여 입력값을 지정합니다.
- `POST /` 메서드는 `{"a":"Number", "b":"Number", "op":"string"}`의 JSON 페이로드를 사용하여 입력값을 지정합니다.
- `GET /{a}/{b}/{op}` 메서드는 경로 파라미터를 사용하여 입력값을 지정합니다.

정의되지 않은 경우, API Gateway는 HTTP 메서드 부분과 경로 부분을 조합하여 해당 SDK 메서드 이름을 생성합니다. 루트 경로 부분(/)은 `Api Root`라고 합니다. 예를 들어 `GET /?a=...&b=...&op=...` API 메서드의 기본 Java SDK 메서드 이름은 `getABOp`이고, `POST /`의 기본 SDK 메서드 이름은 `postApiRoot`이며, `GET /{a}/{b}/{op}`의 기본 SDK 메서드 이름은 `getABOp`입니다. 개별 SDK는 명명 규칙을 사용자 지정할 수 있습니다. SDK별 메서드 이름에 대해서는 생성된 SDK 소스의 설명서를 참고하십시오.

각 API 메서드에서 operationName 속성을 지정하여 기본 SDK 메서드 이름을 재정의할 수 있으며, 이렇게 재정의해야 합니다. API Gateway REST API를 사용하여 API 메서드를 생성하거나 API 메서드를 업데이트할 때 이렇게 재정의할 수 있습니다. API Swagger 정의에서 `operationId`를 설정하여 같은 결과를 달성할 수 있습니다.

이 API에 대해 API Gateway에서 생성된 SDK를 사용하여 이러한 메서드를 호출하는 방법을 보여드리기 전에, 먼저 메서드를 설정하는 방법을 간략하게 복습해 보겠습니다. 자세한 지침은 단원을 참조하십시오Amazon API Gateway에서 REST API 생성 API Gateway를 처음 사용하는 경우 먼저 Lambda 통합을 사용하여 API Gateway REST API 구축 단원을 참조하십시오.

입력 및 출력 모델 생성

강력한 입력 형식을 SDK에 지정하기 위해, 먼저 API의 `Input` 모델을 생성합니다. 응답 본문 데이터 유형을 설명하기 위해 `Output` 모델과 `Result` 모델을 생성합니다.

입력, 출력 및 결과에 대한 모델을 생성하려면

1. 기본 탐색 창에서 모델을 선택합니다.
2. 모델 생성을 선택합니다.
3. 이름에 **`input`**를 입력합니다.
4. 콘텐츠 유형에 **`application/json`**을 입력합니다.

일치하는 콘텐츠 유형이 없는 경우 요청 확인이 수행되지 않습니다. 콘텐츠 유형에 관계없이 동일한 모델을 사용하려면 **$default**를 입력합니다.

5. 모델 스키마에서 다음 모델을 입력합니다.

```
{
    "$schema" : "$schema": "http://json-schema.org/draft-04/schema#",
    "type":"object",
    "properties":{
        "a":{"type":"number"},
        "b":{"type":"number"},
        "op":{"type":"string"}
    },
    "title":"Input"
}
```

6. 모델 생성을 선택합니다.
7. 다음 단계를 반복하여 `Output` 모델과 `Result` 모델을 생성합니다.

   `Output` 모델의 경우 모델 스키마에 다음을 입력합니다.

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "type": "object",
    "properties": {
        "c": {"type":"number"}
    },
    "title": "Output"
}
```

   `Result` 모델의 경우 모델 스키마에 다음을 입력합니다. API ID `abc123`을 해당 API ID로 바꿉니다.

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "type":"object",
    "properties":{
        "input":{
            "$ref":"https://apigateway.amazonaws.com/restapis/abc123/models/Input"
        },
        "output":{
```

```
            "$ref":"https://apigateway.amazonaws.com/restapis/abc123/models/Output"
        }
    },
    "title":"Result"
}
```

GET/메서드 쿼리 파라미터 설정

`GET /?a=..&b=..&op=..` 메서드의 경우, Method Request(메서드 요청)에서 쿼리 파라미터를 선언합니다.

GEL/URL 쿼리 문자열 파라미터를 설정하려면

1. 루트(/) 리소스의 GET 메서드에 대한 메서드 요청 섹션에서 편집을 선택합니다.
2. URL 쿼리 문자열 파라미터를 선택하고 다음을 수행합니다.

   a. 쿼리 문자열 추가(Add query string)를 선택합니다.

   b. 이름에 **a**를 입력합니다.

   c. 필수 상태로 유지하고 캐싱을 해제합니다.

   d. 캐싱을 꺼진 상태로 둡니다.

   같은 단계를 반복하여 **b**라는 쿼리 문자열과 **op**라는 쿼리 문자열을 생성합니다.
3. 저장을 선택합니다.

페이로드의 데이터 모델을 백엔드 입력값으로 설정

`POST /` 메서드의 경우, `Input` 모델을 생성하고 이를 메서드 요청에 추가하여 입력 데이터의 형태를 정의합니다.

페이로드의 데이터 모델을 백엔드 입력으로 설정하려면

1. 루트(/) 리소스의 POST 메서드에 대한 메서드 요청 섹션에서 편집을 선택합니다.
2. 요청 본문을 선택합니다.
3. 모델 추가를 선택합니다.
4. 콘텐츠 유형에 **application/json**을 입력합니다.
5. 모델에서 입력을 선택합니다.

6. 저장을 선택합니다.

이 모델을 사용하는 API 고객은 SDK를 호출하여 `Input` 객체를 인스턴스화하고 입력값을 지정할 수 있습니다. 이 모델을 사용하지 않는 고객은 딕셔너리 객체를 생성하여 Lambda 함수에 대한 JSON 입력을 표시해야 합니다.

백엔드 결과 출력용 데이터 모델 설정

세 가지 메서드 모두에 대해 `Result` 모델을 생성하고 메서드의 `Method Response`에 추가하여 Lambda 함수에서 반환되는 출력 형태를 정의합니다.

백엔드 결과 출력용 데이터 모델을 설정하려면

1. /{a}/{b}/{op} 리소스를 선택한 다음 GET 메서드를 선택합니다.
2. 메서드 응답 탭의 응답 200에서 편집을 선택합니다.
3. 요청 본문에서 모델 추가를 선택합니다.
4. 콘텐츠 유형에 **`application/json`**을 입력합니다.
5. 모델에서 결과를 선택합니다.
6. 저장을 선택합니다.

API 고객은 `Result` 객체의 속성을 읽고 이 모델로 출력값을 분석할 수 있습니다. 이 모델을 사용하지 않는 경우, 고객이 딕셔너리 객체를 생성해서 JSON 출력을 표시해야 합니다.

간편 계산기 API OpenAPI 정의

다음은 간편 계산기 API의 OpenAPI 정의입니다. 이 정의를 계정으로 가져올 수 있습니다. 하지만 가져온 후에는 Lambda 함수에서 리소스 기반 권한을 다시 설정해야 합니다. 이렇게 하려면 사용자의 계정에서 생성한 Lambda 함수를 API Gateway 콘솔의 Integration Request(통합 요청)에서 다시 선택합니다. 그러면 API Gateway 콘솔에서 필요한 권한이 재설정됩니다. 또는 AWS Command Line Interface에서 Lambda 명령 add-permission을 사용할 수 있습니다.

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"
          }
        }
      }
    }
  }
}
```

를 통해 API Gateway API 판매AWS Marketplace

API를 빌드, 테스트 및 배포한 후에는 API를 API Gateway 사용량 계획에 패키징하고 AWS Marketplace를 통해 해당 계획을 Software as a Service(SaaS) 제품으로 판매할 수 있습니다. AWS Marketplace에서는 사용량 계획에 대한 요청 수를 기반으로 제품을 구독하는 API 구매자에게 요금을 청구합니다.

AWS Marketplace에서 API를 판매하려면 AWS Marketplace를 API Gateway와 통합하도록 유통 경로를 설정해야 합니다. 일반적으로 AWS Marketplace에 제품을 나열하고, API Gateway에서 AWS Marketplace에 사용 지표를 전송하도록 허용하는 적절한 정책을 사용하여 IAM 역할을 설정하고, AWS Marketplace 제품을 API Gateway 사용 계획과 연결하고, AWS Marketplace 구매자를 API Gateway API 키와 연결합니다. 세부 정보는 다음 단원에 나와 있습니다.

AWS Marketplace에서 API를 SaaS 제품으로 판매하는 방법에 대한 자세한 내용은 AWS Marketplace 사용 설명서를 참조하십시오.

주제

- API Gateway와 AWS Marketplace의 통합 초기화
- 사용량 계획에 대한 고객 구독 처리

API Gateway와 AWS Marketplace의 통합 초기화

다음은 API를 SaaS 제품으로 판매할 수 있도록 API Gateway와 AWS Marketplace의 일회성 통합을 초기화하는 데 필요한 작업입니다.

에 제품 나열AWS Marketplace

사용량 계획을 SaaS 제품으로 나열하려면 AWS Marketplace를 통해 제품 로드 양식을 제출합니다. 제품에는 `apigateway` 유형의 `requests` 차원이 포함되어 있어야 합니다. 이 차원은 요청당 가격을 정의하며 API Gateway에서 API에 대한 요청을 측정하는 데 사용됩니다.

측정 역할 생성

다음과 같은 실행 정책 및 신뢰 정책을 사용하여 `ApiGatewayMarketplaceMeteringRole`이라는 IAM 역할을 생성합니다. 이 역할을 사용하여 API Gateway에서 AWS Marketplace로 사용 지표를 자동으로 보낼 수 있습니다.

측정 역할의 실행 정책

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "aws-marketplace:BatchMeterUsage",
        "aws-marketplace:ResolveCustomer"
      ],
      "Resource": "*",
      "Effect": "Allow"
    }
  ]
}
```

측정 역할의 신뢰 관계 정책

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

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

사용량 계획과 AWS Marketplace 제품 연결

AWS Marketplace에 제품을 나열하면 AWS Marketplace 제품 코드를 받게 됩니다. API Gateway를 AWS Marketplace와 통합하려면 사용량 계획을 AWS Marketplace 제품 코드와 연결합니다. API Gateway 콘솔, API Gateway REST API, API Gateway용 AWS CLI 또는 API Gateway용 AWS SDK를 사용하여 API Gateway `UsagePlan`의 `productCode` 필드를 AWS Marketplace 제품 코드로 설정하여 연결을 활성화합니다. 다음 코드 예제에서는 API Gateway REST API를 사용합니다.

```
PATCH /usageplans/USAGE_PLAN_ID
Host: apigateway.region.amazonaws.com
Authorization: ...

{
    "patchOperations" : [{
        "path" : "/productCode",
        "value" : "MARKETPLACE_PRODUCT_CODE",
        "op" : "replace"
    }]
}
```

사용량 계획에 대한 고객 구독 처리

다음은 개발자 포털 애플리케이션에서 처리되는 작업입니다.

고객이 AWS Marketplace를 통해 제품을 구독하면 AWS Marketplace에서는 AWS Marketplace에 제품을 나열할 때 등록한 SaaS 구독 URL로 `POST` 요청을 전달합니다. `POST` 요청은 구매자 정보가 들어 있는 `x-amzn-marketplace-token` 파라미터와 함께 제공됩니다. SaaS 고객 온보딩의 지침에 따라 개발자 포털 애플리케이션에서 이 리디렉션을 처리하십시오.

고객의 구독 요청에 응답하여 AWS Marketplace는 구독할 수 있는 Amazon SNS 주제에 `subscribe-success` 알림을 보냅니다. (SaaS 고객 온보딩을 참조하십시오.) 고객 구독 요청을 수락하려면 고객에 대한 API Gateway API 키를 생성하거나 검색하고, AWS Marketplace에서 프로비저닝된 고객의 `customerId`를 API 키와 연결한 다음 사용량 계획에 API 키를 추가하여 `subscribe-success` 알림을 처리합니다.

고객의 구독 요청이 완료되면 개발자 포털 애플리케이션에서 고객에게 연결된 API 키를 제공하고, 고객에게 API를 요청할 때 `x-api-key` 헤더에 이 API 키를 포함시키도록 알려야 합니다.

고객이 사용량 계획에 대한 구독을 취소할 경우 AWS Marketplace에서는 `unsubscribe-success` 알림을 SNS 주제에 보냅니다. 고객의 구독 해지 프로세스를 완료하려면 고객의 API 키를 사용량 계획에서 제거하여 `unsubscribe-success` 알림을 처리합니다.

고객에게 사용량 계획 액세스 권한 부여

지정된 고객에게 사용량 계획에 대한 액세스 권한을 부여하려면 API Gateway API를 사용하여 고객에 대한 API 키를 가져오거나 생성한 다음 API 키를 사용량 계획에 추가합니다.

다음 예에서는 API Gateway REST API를 호출하여 특정 AWS Marketplace `customerId` 값(*MARKETPLACE_CUSTOMER_ID*)을 가진 새로운 API 키를 생성하는 방법을 보여줍니다.

```
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"
}
```

다음 예에서는 특정 AWS Marketplace `customerId` 값(*MARKETPLACE_CUSTOMER_ID*)을 API 키를 가져오는 방법을 보여줍니다.

```
GET apikeys?customerId=MARKETPLACE_CUSTOMER_ID HTTP/1.1
Host: apigateway.region.amazonaws.com
Authorization: ...
```

API 키를 사용량 계획에 추가하려면 관련 사용량 계획에 대한 API 키를 사용하여 UsagePlanKey를 생성하십시오. 다음 예에서는 API Gateway REST API를 사용하여 이 작업을 수행하는 방법을 보여 줍니다. 여기서 `n371pt`는 사용량 계획 ID이고 `q5ugs7qjjh`는 이전 예에서 반환된 예제 API `keyId`입니다.

```
POST /usageplans/n371pt/keys HTTP/1.1
Host: apigateway.region.amazonaws.com
Authorization: ...

{
    "keyId": "q5ugs7qjjh",
    "keyType": "API_KEY"
}
```

고객과 API 키 연결

ApiKey의 customerId 필드를 고객의 AWS Marketplace 고객 ID로 업데이트해야 합니다. 그러면 API 키가 AWS Marketplace 고객과 연결되고, 구매자에 대한 측정 및 청구가 활성화됩니다. 다음 코드 예제에서는 API Gateway REST API를 호출하여 이 작업을 수행합니다.

```
PATCH /apikeys/q5ugs7qjjh
Host: apigateway.region.amazonaws.com
Authorization: ...

{
    "patchOperations" : [{
        "path" : "/customerId",
        "value" : "MARKETPLACE_CUSTOMER_ID",
        "op" : "replace"
    }]
}
```

REST API 보호

API Gateway는 악의적인 사용자나 트래픽 스파이크 같은 특정 위협으로부터 API를 보호할 수 있는 여러 가지 방법을 제공합니다. SSL 인증서 생성, 웹 애플리케이션 방화벽 구성, 제한 대상 설정, Virtual Private Cloud(VPC)에서만 API에 대한 액세스 허용과 같은 전략을 사용하여 API를 보호할 수 있습니다. 이 섹션에서는 API Gateway를 사용하여 이러한 기능을 활성화하는 방법을 배울 수 있습니다.

주제

- REST API에 대한 상호 TLS 인증 구성
- 백엔드 인증을 위한 SSL 인증서 생성 및 구성
- AWS WAF을 사용하여 API 보호
- 처리량 향상을 위해 API 요청 조절

- Amazon API Gateway에서 프라이빗 API 생성

REST API에 대한 상호 TLS 인증 구성

상호 TLS 인증에는 클라이언트와 서버 간의 양방향 인증이 필요합니다. 상호 TLS를 사용하는 경우 클라이언트가 API에 액세스하려면 자격 증명을 확인하기 위해 X.509 인증서를 제공해야 합니다. 상호 TLS는 사물 인터넷 (IoT) 및 business-to-business 애플리케이션의 일반적인 요구 사항입니다.

API Gateway가 지원하는 다른 권한 부여 및 인증 작업과 함께 상호 TLS를 사용할 수 있습니다. API Gateway는 클라이언트가 제공하는 인증서를 Lambda 권한 부여자와 백엔드 통합에 전달합니다.

> **Important**
>
> 기본적으로 클라이언트는 API에 대해 API Gateway가 생성하는 `execute-api` 엔드포인트를 사용하여 API를 호출할 수 있습니다. 클라이언트가 상호 TLS가 활성화된 사용자 지정 도메인 이름을 사용해야만 API에 액세스할 수 있도록 하려면 기본 `execute-api` 엔드포인트를 비활성화합니다. 자세한 내용은 the section called “기본 엔드포인트 비활성화” 단원을 참조하세요.

주제

- 상호 TLS의 사전 조건
- 사용자 지정 도메인 이름에 대한 상호 TLS 구성
- 상호 TLS가 요구되는 사용자 지정 도메인 이름을 사용하여 API 호출
- 트러스트 스토어 업데이트
- 상호 TLS 비활성화
- 인증서 경고 문제 해결
- 도메인 이름 충돌 문제 해결
- 도메인 이름 상태 메시지 문제 해결

상호 TLS의 사전 조건

상호 TLS를 구성하려면 다음이 필요합니다.

- 사용자 지정 도메인 이름
- 사용자 지정 도메인 이름에 AWS Certificate Manager 대해 하나 이상의 인증서가 구성되어 있어야 합니다.

- Amazon S3에 구성 및 업로드된 트러스트 스토어

사용자 지정 도메인 이름

REST API에 상호 TLS를 활성화하려면 API에 대한 사용자 지정 도메인 이름을 구성해야 합니다. 사용자 지정 도메인 이름에 상호 TLS를 활성화한 다음 클라이언트에 사용자 지정 도메인 이름을 제공할 수 있습니다. 상호 TLS가 활성화된 사용자 지정 도메인 이름을 사용하여 API에 액세스하려면 클라이언트가 API 요청에서 신뢰할 수 있는 인증서를 제공해야 합니다. 추가 정보는 the section called “사용자 지정 도메인 이름” 단원에서 확인할 수 있습니다.

AWS Certificate Manager 발급한 인증서 사용

ACM에서 직접 공인 인증서를 요청하거나 공용 인증서 또는 자체 서명된 인증서를 가져올 수 있습니다. ACM에 인증서를 설정하려면 ACM으로 이동하세요. 인증서를 가져오려면 다음 단원을 계속 읽으세요.

가져온 AWS Private Certificate Authority 인증서 또는 인증서 사용

ACM으로 가져온 인증서 또는 상호 TLS를 AWS Private Certificate Authority 통해 가져온 인증서를 사용하려면 API Gateway에서 ACM에서 `ownershipVerificationCertificate` 발급한 인증서가 필요합니다. 이 소유권 인증서는 도메인 이름을 사용할 수 있는 권한이 있는지 확인하는 데에만 사용됩니다. TLS 핸드셰이크에는 사용되지 않습니다. 아직 `ownershipVerificationCertificate`가 없으면 https://console.aws.amazon.com/acm/으로 이동하여 하나를 설정합니다.

도메인 이름의 수명 동안 이 인증서를 유효하게 유지해야 합니다. 인증서가 만료되고 자동 갱신이 실패하면 도메인 이름에 대한 모든 업데이트가 잠깁니다. 다른 변경 사항을 적용하기 전에 유효한 `ownershipVerificationCertificate`를 사용해 `ownershipVerificationCertificateArn`을 업데이트해야 합니다. `ownershipVerificationCertificate`는 API Gateway의 다른 상호 TLS 도메인에 대한 서버 인증서로 사용할 수 없습니다. 인증서를 ACM으로 직접 다시 가져오는 경우 발급자는 동일하게 유지해야 합니다.

트러스트 스토어 구성

트러스트 스토어는 파일 확장자가 `.pem`인 텍스트 파일입니다. 인증 기관으로부터의 신뢰할 수 있는 인증서 목록입니다. 상호 TLS를 사용하려면 API에 액세스하도록 신뢰할 수 있는 X.509 인증서의 트러스트 스토어를 생성합니다.

발급 CA 인증서부터 루트 CA 인증서까지의 전체 신뢰 체인을 트러스트 스토어에 포함시켜야 합니다. API Gateway는 현재 신뢰 체인에 있는 모든 CA에서 발급한 클라이언트 인증서를 수락합니다. 공인 또

는 사설 인증 기관의 인증서가 포함될 수 있습니다. 인증서의 체인 길이는 최대 4개가 될 수 있습니다. 자체 서명된 인증서를 제공할 수도 있습니다. 트러스트 스토어에서 지원되는 해싱 알고리즘은 다음과 같습니다.

- SHA-256 이상
- RSA-2048 이상
- ECDSA-256 이상

API Gateway에서는 여러 인증서 속성의 유효성을 검사합니다. Lambda 권한 부여자를 사용하여 클라이언트가 API를 호출할 때 인증서가 해지되었는지 확인하는 것과 같은 추가 검사를 수행할 수 있습니다. API Gateway는 다음 속성의 유효성을 검사합니다.

검증	설명
X.509 구문	인증서는 X.509 구문 요구 사항을 충족해야 합니다.
무결성	인증서 내용이 트러스트 스토어의 인증 기관에서 서명한 내용에서 변경되어서는 안 됩니다.
유효성	인증서의 유효 기간이 최신이어야 합니다.
이름 체인/키 체인	인증서의 이름과 주체는 끊어지지 않는 체인을 형성해야 합니다. 인증서의 체인 길이는 최대 4개가 될 수 있습니다.

단일 파일의 Amazon S3 버킷에 트러스트 스토어를 업로드합니다.

다음은 .pem 파일의 예입니다.

Example certificates.pem

```
-----BEGIN CERTIFICATE-----
<Certificate contents>
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
<Certificate contents>
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
```

```
<Certificate contents>
-----END CERTIFICATE-----
...
```

다음 AWS CLI 명령은 Amazon S3 `certificates.pem` 버킷에 업로드됩니다.

```
aws s3 cp certificates.pem s3://bucket-name
```

API Gateway가 신뢰 저장소에 액세스할 수 있도록 하려면 Amazon S3 버킷에 API Gateway에 대한 읽기 권한이 있어야 합니다.

사용자 지정 도메인 이름에 대한 상호 TLS 구성

REST API에 대한 상호 TLS를 구성하려면 API에 대해 최소 TLS 버전이 1.2인 리전 사용자 지정 도메인 이름을 사용해야 합니다. 사용자 지정 도메인 이름 생성 및 구성에 대한 자세한 내용은 the section called “리전 사용자 지정 도메인 이름 설정” 단원을 참조하세요.

> **Note**
>
> 프라이빗 API에는 상호 TLS가 지원되지 않습니다.

트러스트 스토어를 Amazon S3에 업로드한 후 상호 TLS를 사용하도록 사용자 지정 도메인 이름을 구성할 수 있습니다. 다음 내용(슬래시 포함)을 터미널에 붙여 넣습니다.

```
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
```

도메인 이름을 생성한 후에는 API 작업에 대한 DNS 레코드와 기본 경로 매핑을 구성해야 합니다. 자세한 내용은 API Gateway에서 리전 사용자 지정 도메인 이름 설정 단원을 참조하십시오.

상호 TLS가 요구되는 사용자 지정 도메인 이름을 사용하여 API 호출

상호 TLS가 활성화된 API를 호출하려면 클라이언트가 API 요청에서 신뢰할 수 있는 인증서를 제공해야 합니다. 클라이언트가 API를 호출하려고 하면 API Gateway는 트러스트 스토어에서 클라이언트 인

증서의 발급자를 찾습니다. API Gateway가 요청을 진행하려면 인증서의 발급자 및 루트 CA 인증서까지의 전체 신뢰 체인이 트러스트 스토어에 있어야 합니다.

다음 예제 `curl` 명령은 요청에 `api.example.com`,을 포함하여 `my-cert.pem`에 요청을 보냅니다. `my-key.key`는 인증서의 프라이빗 키입니다.

```
curl -v --key ./my-key.key --cert ./my-cert.pem api.example.com
```

트러스트 스토어가 인증서를 신뢰하는 경우에만 API가 호출됩니다. 다음 조건에서는 API Gateway가 TLS 핸드셰이크에 실패하고 `403` 상태 코드와 함께 요청을 거부합니다. 인증서가 다음 상태인 경우:

- 신뢰할 수 없음
- 만료됨
- 지원되는 알고리즘을 사용하지 않음

> **Note**
>
> API Gateway에서는 인증서가 해지되었는지 여부를 확인하지 않습니다.

트러스트 스토어 업데이트

트러스트 스토어의 인증서를 업데이트하려면 새 인증서 번들을 Amazon S3에 업로드합니다. 그런 다음 업데이트된 인증서를 사용하도록 사용자 지정 도메인 이름을 업데이트할 수 있습니다.

Amazon S3 버전 관리를 사용하여 트러스트 스토어의 여러 버전을 유지 관리합니다. 새 트러스트 스토어 버전을 사용하도록 사용자 지정 도메인 이름을 업데이트하면 인증서가 유효하지 않을 경우 API Gateway에서 경고가 반환됩니다.

API Gateway는 도메인 이름을 업데이트할 때만 인증서 경고를 생성합니다. API Gateway는 이전에 업로드한 인증서 만료를 알려주지 않습니다.

다음 AWS CLI 명령은 새 신뢰 저장소 버전을 사용하도록 사용자 지정 도메인 이름을 업데이트합니다.

```
aws apigateway update-domain-name \
    --domain-name api.example.com \
    --patch-operations op='replace',path='/mutualTlsAuthentication/
truststoreVersion',value='abcdef123'
```

상호 TLS 비활성화

사용자 지정 도메인 이름에 상호 TLS를 사용하지 않도록 설정하려면 다음 명령과 같이 사용자 지정 도메인 이름에서 해당 트러스트 스토어를 제거합니다.

```
aws apigateway update-domain-name \
    --domain-name api.example.com \
    --patch-operations op='replace',path='/mutualTlsAuthentication/
truststoreUri',value=''
```

인증서 경고 문제 해결

상호 TLS가 포함된 사용자 지정 도메인 이름을 만들 때 트러스트 스토어의 인증서가 유효하지 않을 경우 API Gateway 에서 경고가 반환됩니다. 이는 새 트러스트 스토어를 사용하도록 사용자 지정 도메인 이름을 업데이트할 때에도 발생할 수 있습니다. 이러한 경고는 경고를 유발한 인증서와 인증서 주체에 문제가 있음을 나타냅니다. API에 대해 상호 TLS가 계속 활성화되어 있지만 일부 클라이언트는 API에 액세스하지 못할 수 있습니다.

경고를 유발한 인증서를 식별하려면 트러스트 스토어에서 인증서를 디코딩해야 합니다. `openssl` 같은 도구를 사용하여 인증서를 디코딩하고 해당 주체를 식별할 수 있습니다.

다음 명령은 주체를 포함하여 인증서의 내용을 표시합니다.

```
openssl x509 -in certificate.crt -text -noout
```

경고를 유발한 인증서를 업데이트하거나 제거한 다음 새 트러스트 스토어를 Amazon S3에 업로드합니다. 새 트러스트 스토어를 업로드한 후 새 트러스트 스토어를 사용하도록 사용자 지정 도메인 이름을 업데이트합니다.

도메인 이름 충돌 문제 해결

오류 `"The certificate subject <certSubject> conflicts with an existing certificate from a different issuer."`은(는) 여러 인증 기관이 이 도메인에 대한 인증서를 발급했음을 의미합니다. 인증서의 각 주체에 대해 API Gateway에는 상호 TLS 도메인에 대한 발급자가 하나만 있을 수 있습니다. 해당 주제에 대한 모든 인증서는 단일 발급자를 통해서만 받아야 합니다. 본인이 제어할 수 없지만 도메인 이름의 소유권을 증명할 수는 있는 인증서에 문제가 있는 경우 AWS Support에 문의를 눌러 티켓을 엽니다.

도메인 이름 상태 메시지 문제 해결

`PENDING_CERTIFICATE_REIMPORT`: 이것은 인증서를 ACM으로 다시 가져왔으며, 새 인증서에 SAN(주체 대체 이름)이 `ownershipVerificationCertificate`에 포함되지 않거나 인증서의 주체 또는 SAN이 도메인 이름을 포함하지 않기 때문에 유효성 검사에 실패했다는 의미입니다. 잘못 구성되었거나 잘못된 인증서를 가져왔을 수 있습니다. 유효한 인증서를 ACM으로 다시 가져와야 합니다. 검증에 대한 자세한 내용은 도메인 소유권 검증을 참조하세요.

`PENDING_OWNERSHIP_VERIFICATION`: 이것은 이전에 확인된 인증서가 만료되어 ACM이 인증서를 자동 갱신할 수 없다는 의미입니다. 인증서를 갱신하거나 새 인증서를 요청해야 합니다. 인증서 갱신에 대한 자세한 내용은 ACM의 관리형 인증서 갱신 문제 해결 가이드를 참조하세요.

백엔드 인증을 위한 SSL 인증서 생성 및 구성

API Gateway를 사용하면 SSL 인증서를 생성한 다음, 백엔드에서 퍼블릭 키를 사용하여 백엔드 시스템에 대한 HTTP 요청의 출처가 API Gateway인지 확인할 수 있습니다. 이렇게 하면 백엔드에 공개적으로 액세스할 수 있는 경우에도 HTTP 백엔드가 Amazon API Gateway에서 시작된 요청만 제어 및 수락하도록 할 수 있습니다.

> **Note**
>
> 일부 백엔드 서버는 API Gateway처럼 SSL 클라이언트 인증을 지원하지 않으며, SSL 인증서 오류를 반환할 수 있습니다. 호환되지 않는 백엔드 서버의 목록은 the section called “중요 정보”를 참조하십시오.

API Gateway에서 생성된 SSL 인증서는 자체 서명되며 인증서의 퍼블릭 키만 API Gateway 콘솔에 표시되거나 API를 통해서 볼 수 있습니다.

주제

- API Gateway 콘솔을 사용하여 클라이언트 인증서 생성
- SSL 인증서를 사용하도록 API 구성
- 클라이언트 인증서 구성을 확인하기 위한 호출 테스트
- 클라이언트 인증서의 유효성 검사를 위해 백엔드 HTTPS 서버 구성
- 기존 클라이언트 인증서 교체
- HTTP 및 HTTP 프록시 통합에 대한 API Gateway 지원 인증 기관

API Gateway 콘솔을 사용하여 클라이언트 인증서 생성

1. https://console.aws.amazon.com/apigateway/에서 Amazon API Gateway 콘솔을 엽니다.
2. REST API를 선택합니다.
3. 기본 탐색 창에서 클라이언트 인증서를 선택합니다.
4. 클라이언트 인증서 페이지에서 인증서 생성을 선택합니다.
5. (선택 사항) 설명에 설명을 입력합니다.
6. 인증서 생성을 선택하여 인증서를 생성합니다. API Gateway는 새로운 인증서를 생성하고 새 인증서 GUID를 PEM 인코딩된 퍼블릭 키와 함께 반환합니다.

이제 인증서를 사용하도록 API를 구성할 준비가 되었습니다.

SSL 인증서를 사용하도록 API 구성

이 지침에서는 API Gateway 콘솔을 사용하여 클라이언트 인증서 생성을 이미 완료한 것으로 가정합니다.

1. 클라이언트 인증서를 사용하고자 하는 API를 API Gateway 콘솔에서 생성하거나 엽니다. API가 스테이지에 배포되었는지 확인합니다.
2. 기본 탐색 창에서 스테이지를 선택합니다.
3. 스테이지 세부 정보 섹션에서 편집을 선택합니다.
4. 클라이언트 인증서에서 인증서를 선택합니다.
5. 변경 사항 저장을 선택합니다.

   API Gateway 콘솔에서 이전에 API를 배포한 경우 변경 사항을 적용하려면 API를 다시 배포해야 합니다. 자세한 내용은 the section called “REST API를 단계에 재배포” 단원을 참조하세요.

API에 대한 인증서를 선택하고 저장하면 API Gateway는 API의 HTTP 통합에 대한 모든 호출에 인증서를 사용합니다.

클라이언트 인증서 구성을 확인하기 위한 호출 테스트

1. API 메서드를 선택합니다. 테스트(Test) 탭을 선택합니다. 테스트 탭을 표시하려면 오른쪽 화살표 버튼을 선택해야 할 수도 있습니다.
2. 클라이언트 인증서에서 인증서를 선택합니다.

3. 테스트를 선택합니다.

API Gateway는 선택된 SSL 인증서를 제공하여 HTTP 백엔드에서 API를 인증할 수 있도록 합니다.

클라이언트 인증서의 유효성 검사를 위해 백엔드 HTTPS 서버 구성

이 지침에서는 이미 API Gateway 콘솔을 사용하여 클라이언트 인증서 생성을 완료하고 클라이언트 인증서 사본을 다운로드한 것으로 가정합니다. API Gateway REST API의 `clientcertificate:by-id` 또는 AWS CLI의 `get-client-certificate`를 호출하여 클라이언트 인증서를 다운로드할 수 있습니다.

API Gateway의 클라이언트 SSL 인증서를 확인하기 위해 백엔드 HTTPS 서버를 구성하기 전에 신뢰할 수 있는 인증 기관에서 제공한 PEM 인코딩된 프라이빗 키와 서버 측 인증서를 입수해야 합니다.

서버 도메인 이름이 `myserver.mydomain.com`일 경우, 서버 인증서의 CNAME 값이 `myserver.mydomain.com` 또는 `*.mydomain.com`이어야 합니다.

지원되는 인증 기관에는 Let's Encrypt 또는 the section called “HTTP 및 HTTP 프록시 통합에 대해 지원되는 인증 기관” 중 하나가 포함됩니다.

예를 들어 클라이언트 인증서 파일이 `apig-cert.pem`이고 서버 프라이빗 키와 인증서 파일이 각각 `server-key.pem` 및 `server-cert.pem`이라고 가정합니다. 백엔드에 있는 Node.js 서버의 경우 다음과 유사하게 서버를 구성할 수 있습니다.

```
var fs = require('fs');
var https = require('https');
var options = {
    key: fs.readFileSync('server-key.pem'),
    cert: fs.readFileSync('server-cert.pem'),
    ca: fs.readFileSync('apig-cert.pem'),
    requestCert: true,
    rejectUnauthorized: true
};
https.createServer(options, function (req, res) {
    res.writeHead(200);
    res.end("hello world\n");
}).listen(443);
```

node-express 앱의 경우, client-certificate-auth 모듈을 사용하여 PEM 인코딩된 인증서로 클라이언트 요청을 인증할 수 있습니다.

다른 HTTPS 서버는 해당 서버에 대한 설명서를 참조하세요.

기존 클라이언트 인증서 교체

API Gateway에 의해 생성된 클라이언트 인증서는 365일간 유효합니다. API 가동 중지를 방지하려면 API 단계의 클라이언트 인증서가 만료되기 전에 인증서를 교체해야 합니다. API Gateway REST API의 clientCertificate:by-id 또는 get-client-certificate의 AWS CLI 명령을 호출하여 반환된 expirationDate 속성에서 인증서 만료 날짜를 확인할 수 있습니다.

클라이언트 인증서를 교체하려면 다음을 수행합니다.

1. API Gateway REST API의 clientcertificate:generate 또는 AWS CLI generate-client-certificate 명령을 호출하여 새 클라이언트 인증서를 생성합니다. 자습서에서는 새 클라이언트 인증서 ID가 `ndiqef`라고 가정합니다.
2. 백엔드 서버를 업데이트하여 새 클라이언트 인증서를 포함시킵니다. 아직 기존 클라이언트 인증서를 제거하지 마세요.

   업데이트를 완료하려면 일부 서버를 재시작해야 할 수 있습니다. 서버 설명서를 참조하여 업데이트 도중 서버를 재시작해야 하는지 확인하세요.
3. API Gateway REST API의 stage:update를 호출하여 새 클라이언트 인증서 ID(`ndiqef`)의 새 클라이언트 인증서를 사용하도록 API 단계를 업데이트합니다.

   ```
   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"
       }
     ]
   }
   ```

   또는 CLI update-stage 명령을 호출할 수 있습니다.
4. 백엔드 서버를 업데이트하여 기존 인증서를 제거합니다.

5. 기존 인증서의 clientCertificateId(a1b2c3)를 지정해 API Gateway REST API의 clientcertificate:delete를 호출하여 API Gateway에서 기존 인증서를 삭제합니다.

```
DELETE /clientcertificates/a1b2c3
```

또는 CLI delete-client-certificate 명령을 호출할 수 있습니다.

```
aws apigateway delete-client-certificate --client-certificate-id a1b2c3
```

이전에 배포한 API에 대해 콘솔에서 클라이언트 인증서를 교체하려면 다음과 같이 합니다.

1. 기본 탐색 창에서 클라이언트 인증서를 선택합니다.
2. 클라이언트 인증서 창에서 인증서 생성을 선택합니다.
3. 클라이언트 인증서를 사용하려는 API를 엽니다.
4. 선택한 API에 있는 단계(Stages)를 선택한 다음 단계를 선택합니다.
5. 스테이지 세부 정보 섹션에서 편집을 선택합니다.
6. 클라이언트 인증서에서 새 인증서를 선택합니다.
7. 설정을 저장하려면 변경 사항 저장을 선택합니다.

변경을 적용하려면 API를 재배포해야 합니다. 자세한 내용은 the section called “REST API를 단계에 재배포” 섹션을 참조하세요.

HTTP 및 HTTP 프록시 통합에 대한 API Gateway 지원 인증 기관

다음 목록은 HTTP 및 HTTP 프록시 통합에 대해 API Gateway에서 지원되는 인증 기관을 보여줍니다.

```
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 WAF을 사용하여 API 보호

AWS WAF는 웹 애플리케이션 및 API를 공격으로부터 보호하는 데 도움이 되는 웹 애플리케이션 방화벽입니다. 사용자가 정의한 맞춤형 웹 보안 규칙 및 조건에 따라 웹 요청을 허용, 차단 또는 계산하는 규칙 집합(웹 액세스 제어 목록 또는 웹 ACL)을 구성할 수 있습니다. 자세한 내용은 AWS WAF 작동 방식을 참조하세요.

AWS WAF를 사용하여 SQL 명령어 주입 및 교차 사이트 스크립팅(XSS) 공격과 같은 일반적인 웹 익스플로잇으로부터 API Gateway REST API를 보호할 수 있습니다. 이는 API 가용성 및 성능에 영향을 미치거나, 보안을 손상시키거나, 리소스를 과도하게 소비할 수 있습니다. 예를 들어 지정된 IP 주소 범위의 요청, CIDR 블록의 요청, 특정 국가 또는 리전에서 시작된 요청, 악성 SQL 코드가 포함된 요청, 악성 스크립트가 포함된 요청을 허용하거나 차단하는 규칙을 만들 수 있습니다.

HTTP 헤더, 메서드, 쿼리 문자열, URI 및 요청 본문(처음 8KB로 제한)에 지정된 문자열이나 정규식 패턴과 일치하는 규칙을 만들 수도 있습니다. 또한 특정 사용자 에이전트, 악성 봇, 콘텐츠 스크래퍼의 공격을 차단하는 규칙을 생성할 수 있습니다. 예를 들어, 비율 기반 규칙을 사용하여 연속적으로 업데이트되는 이후 5분 동안 각 클라이언트 IP에서 허용하는 웹 요청 수를 지정할 수 있습니다.

> **Important**
>
> AWS WAF는 웹 익스플로잇에 대한 방어벽 제1선입니다. API에서 AWS WAF가 활성화된 경우, AWS WAF 규칙은 다른 액세스 제어 기능(리소스 정책, IAM 정책, Lambda 권한 부여자, Amazon Cognito 권한 부여자)보다 먼저 평가됩니다. 예를 들어 AWS WAF가 리소스 정책이 허용하는 CIDR 블록으로부터의 액세스를 차단할 경우, AWS WAF가 우선권을 가지므로 리소스 정책은 평가되지 않습니다.

API에서 AWS WAF을 활성화하려면 다음을 수행해야 합니다.

1. AWS WAF 콘솔, AWS SDK 또는 CLI를 사용하여 AWS WAF 관리형 규칙과 사용자 지정 규칙을 원하는 대로 조합한 리전 웹 ACL을 생성합니다. 자세한 내용은 AWS WAF 시작 및 웹 ACL(웹 액세스 제어 목록) 생성 및 구성을 참조하세요.

   > **Important**
   >
   > API Gateway에는 리전 웹 ACL이 필요합니다.

2. AWS WAF 리전 웹 ACL과 API 스테이지를 연결합니다. AWS WAF 콘솔, AWS SDK, CLI 또는 API Gateway 콘솔을 사용하여 이 작업을 수행할 수 있습니다.

API Gateway 콘솔을 사용하여 API Gateway API 스테이지와 AWS WAF 리전 웹 ACL을 연결하는 방법

API Gateway 콘솔을 사용하여 AWS WAF 리전 웹 ACL을 기존 API Gateway API 스테이지와 연결하려면 다음 단계를 수행합니다.

1. https://console.aws.amazon.com/apigateway에서 API Gateway 콘솔에 로그인합니다.
2. 기존 API를 선택하거나 새 API를 생성합니다.
3. 기본 탐색 창에서 스테이지를 선택한 후 원하는 스테이지를 선택합니다.
4. 스테이지 세부 정보 섹션에서 편집을 선택합니다.
5. 리전 웹 ACL과 API 스테이지를 연결하려면
   - 웹 애플리케이션 방화벽(AWS WAF)에서 이 스테이지와 연결할 리전 웹 ACL을 선택합니다.
6. 변경 사항 저장을 선택합니다.

AWS CLI를 사용하여 API Gateway API 스테이지와 AWS WAF 리전 웹 ACL을 연결하는 방법

AWS CLI를 사용하여 AWS WAF 리전 웹 ACL을 기존 API Gateway API 스테이지와 연결하려면 다음 예제에서와 같이 `associate-web-acl` 명령을 호출합니다.

```
aws waf-regional associate-web-acl \
--web-acl-id 'aabc123a-fb4f-4fc6-becb-2b00831cadcf' \
--resource-arn 'arn:aws:apigateway:{region}::/restapis/4wk1k4onj3/stages/prod'
```

AWS WAF REST API를 사용하여 API 단계와 AWS WAF 리전 웹 ACL 연결

AWS WAF REST API를 사용하여 AWS WAF 리전 웹 ACL을 기존 API Gateway 스테이지와 연결하려면 다음 예제에서와 같이 `AssociateWebACL` 명령을 호출합니다.

```
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'
```

```
)
```

처리량 향상을 위해 API 요청 조절

API에 대한 제한 및 할당량을 구성하여 너무 많은 요청으로 인해 과부하되지 않도록 보호할 수 있습니다. 제한과 할당량은 모두 최선의 방식으로 적용되며 보장된 요청 한도가 아닌 대상으로 간주해야 합니다.

API Gateway는 요청에 대해 토큰이 계산되는 토큰 버킷 알고리즘을 사용하여 API에 대한 요청을 제한합니다. 특히 API Gateway는 리전별로 계정의 모든 API에 대한 요청 제출 속도와 버스트를 검사합니다. 토큰 버킷 알고리즘에서 버스트는 이러한 제한의 사전 정의된 초과 실행을 허용할 수 있지만, 다른 요인으로도 제한을 초과할 수 있습니다.

요청 제출이 정상 상태의 요청 속도 및 버스트 제한을 초과할 경우 API Gateway에서 요청을 제한하기 시작합니다. 이 시점에서 클라이언트는 `429 Too Many Requests` 오류 응답을 받을 수 있습니다. 이러한 예외를 포착하면 클라이언트는 속도 제한 방식으로 실패한 요청을 다시 제출할 수 있습니다.

API 개발자는 개별 API 단계 또는 메서드에 대한 대상 제한을 설정하여 계정의 모든 API에서 전반적인 성능을 향상시킬 수 있습니다. 또는 지정된 요청 비율 및 할당량을 기반으로 클라이언트 요청 제출에 대한 제한을 설정하도록 사용량 계획을 활성화할 수 있습니다.

주제

- API Gateway에 조절 한도 설정을 적용하는 방법
- 리전별 계정 수준 조절
- 사용 계획에서 API 수준 및 단계 수준의 제한 대상 구성
- 스테이지 수준 제한 대상 구성
- 사용 계획에서 메서드 수준 제한 대상 구성

API Gateway에 조절 한도 설정을 적용하는 방법

API에 대한 제한 및 할당량 설정을 구성하기 전에 Amazon API Gateway에서 어떻게 적용되는지 이해하면 유용합니다.

Amazon API Gateway는 네 가지 기본 유형의 제한 관련 설정을 제공합니다.

- AWS 제한 한도는 리전의 모든 계정과 클라이언트에 적용됩니다. 이 한도 설정은 API 및 사용자의 계정이 너무 많은 요청 때문에 가득 차지 않도록 방지하기 위해 존재합니다. 이러한 제한은 AWS에서 설정하며 고객이 변경할 수 없습니다.

- 계정당 한도는 지정된 리전에 있는 계정의 모든 API에 적용됩니다. 계정 수준 속도 제한은 요청 시 늘릴 수 있습니다. 제한 시간이 더 짧고 페이로드가 더 작은 API를 사용하면 더 높은 제한이 가능합니다. 리전별로 계정 수준 조절 한도 증가를 요청하려면 AWS 지원 센터에 문의하시기 바랍니다. 자세한 내용은 할당량 및 중요 정보 섹션을 참조하세요. 이러한 제한은 AWS 제한 한도보다 높을 수 없습니다.
- API별, 단계별 제한 한도는 단계의 API 메서드 수준에서 적용됩니다. 모든 메서드에 대해 동일한 설정을 구성하거나 각 메서드에 대해 다른 제한 설정을 구성할 수 있습니다. 이러한 제한은 AWS 제한 한도보다 높을 수 없습니다.
- 클라이언트별 제한 한도는 사용량 계획과 연결된 API 키를 클라이언트 식별자로 사용하는 클라이언트에게 적용됩니다. 이러한 한도는 계정당 한도보다 높을 수 없습니다.

API Gateway 조절 관련 설정은 다음 순서로 적용됩니다.

1. 사용량 계획의 API 단계에 대해 설정한 클라이언트별 또는 메서드별 제한 한도
2. API 스테이지에 대해 설정한 메서드별 제한 한도
3. 리전별 계정 수준 조절
4. AWS 리전별 제한

리전별 계정 수준 조절

기본적으로 API Gateway는 리전별로 AWS 계정 내의 모든 API에서 안정적인 상태의 초당 요청(RPS)을 제한합니다. 또한 리전별로 AWS 계정 내 모든 API에 대해 버스트(즉, 최대 버킷 크기)를 제한합니다. API Gateway에서 버스트 제한은 API Gateway가 `429 Too Many Requests` 오류 응답을 반환하기 전에 수행할 동시 요청 제출의 최대 목표 수를 나타냅니다. 조절 할당량에 대한 자세한 내용은 할당량 및 중요 정보 단원을 참조하세요.

사용 계획에서 API 수준 및 단계 수준의 제한 대상 구성

사용 계획에서 API 또는 스테이지 수준의 모든 메서드에 대한 메서드별 제한 대상을 설정할 수 있습니다. 토큰이 토큰 버킷에 추가되는 속도(초당 요청 수)인 제한율을 지정할 수 있습니다. 토큰 버킷의 용량인 제한 버스트를 지정할 수도 있습니다.

스테이지 수준 제한 대상 구성

AWS CLI, SDK 및 AWS Management Console을 사용하여 스테이지 수준 제한 대상을 생성할 수 있습니다. 자세한 내용은 ??? 섹션을 참조하세요.

사용 계획에서 메서드 수준 제한 대상 구성

사용량 계획 생성에 표시된 대로 사용 계획의 메서드 수준에서 추가 제한 대상을 설정할 수 있습니다. API Gateway 콘솔에서는 메서드 조절 구성(Configure Method Throttling) 설정에서 `Resource=<resource>`, `Method=<method>`를 지정하여 설정합니다. 예를 들어, PetStore 예시에서 `Resource=/pets`, `Method=GET`을 지정할 수 있습니다.

Amazon API Gateway에서 프라이빗 API 생성

Amazon API Gateway를 사용하면 인터페이스 VPC 종단점을 사용하여 Amazon VPC의 Virtual Private Cloud에서만 액세스할 수 있는 프라이빗 REST API를 생성할 수 있습니다. VPC에서 생성하는 엔드포인트 네트워크 인터페이스입니다.

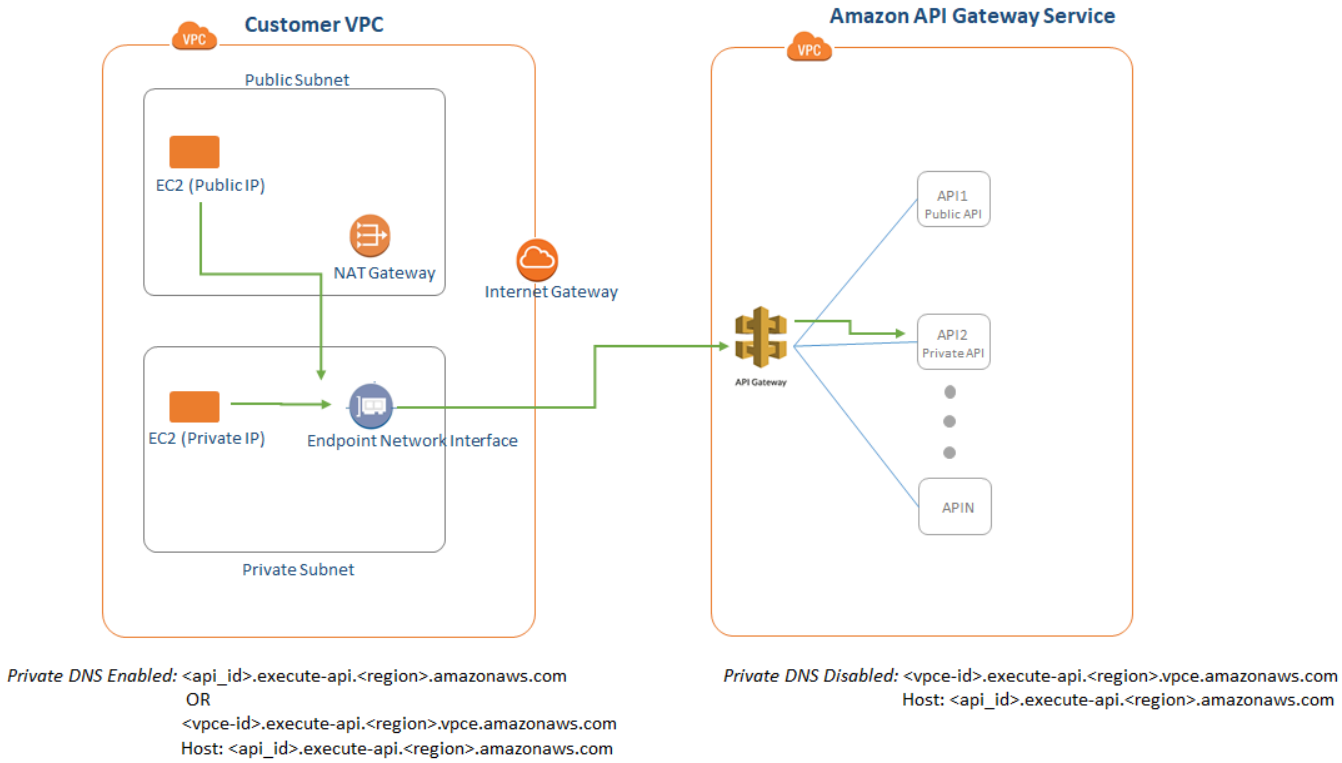
리소스 정책을 사용하면 계정 전체를 포함하여 선택한 VPC 및 VPC 엔드포인트에서 API에 대한 액세스를 허용하거나 거부할 수 있습니다. AWS 각 엔드포인트를 사용하여 여러 개의 프라이빗 API에 액세스할 수 있습니다. 를 AWS Direct Connect 사용하여 온프레미스 네트워크에서 Amazon VPC로 연결을 설정하고 해당 연결을 통해 프라이빗 API에 액세스할 수도 있습니다.

> **Important**
>
> 프라이빗 API에 대한 액세스를 특정 VPC 또는 VPC 엔드포인트로 제한하려면 API의 리소스 정책에 `aws:SourceVpc` 및 `aws:SourceVpce` 조건을 추가합니다. 예시 정책은 the section called “예제: 소스 VPC 또는 VPC 종단점에 따라 프라이빗 API 트래픽 허용” 섹션을 참조하세요.

어떤 경우에든 프라이빗 API로 가는 트래픽은 안전한 연결을 사용하고, Amazon 네트워크를 벗어나지 않으며, 퍼블릭 인터넷과 격리됩니다.

아래 다이어그램과 같이 API Gateway용 인터페이스 VPC 종단점을 통해 프라이빗 API에 액세스할 수 있습니다. 프라이빗 DNS를 활성화한 경우, 프라이빗 또는 퍼블릭 DNS 이름으로 API에 액세스할 수 있습니다. 프라이빗 DNS가 비활성인 경우, 퍼블릭 DNS 이름만 사용할 수 있습니다.

> **Note**
>
> API Gateway 프라이빗 API는 TLS 1.2만 지원합니다. 이전 버전의 TLS는 지원되지 않습니다.



프라이빗 API를 만드는 단계는 대략 다음과 같습니다.

1. 먼저 API 실행에 사용할 API Gateway 구성 요소 서비스에 대해 인터페이스 VPC 종단점을 생성합니다. VPC에서는 이를 `execute-api`라고 부릅니다.
2. 프라이빗 API를 생성 후 테스트합니다.

   a. 다음 절차 중 하나를 사용하여 API를 만듭니다.

      - API Gateway 콘솔
      - API Gateway CLI
      - AWS SDK는 다음과 같습니다. JavaScript

   b. VPC 엔드포인트에 대한 액세스 권한을 부여하려면 리소스 정책을 만들어 API에 연결합니다.

   c. API를 테스트합니다.

> **Note**
>
> 아래 절차에서는 완전히 구성된 VPC가 이미 있는 것으로 가정합니다. 자세한 내용과 VPC를 만들기 시작하는 방법은 Amazon VPC 사용 설명서의 Amazon VPC 시작하기를 참조하세요.

프라이빗 API 배포 시 고려 사항

- 기존의 퍼블릭 API(리전 또는 엣지 최적화된 API)를 프라이빗 API로 전환할 수 있고, 프라이빗 API를 리전 API로 전환할 수 있습니다. 그러나 프라이빗 API를 엣지 최적화된 API로 전환할 수는 없습니다. 자세한 내용은 ??? 단원을 참조하세요.
- VPC 및 VPC 종단점에 프라이빗 API 액세스 권한을 부여하려면, 리소스 정책을 생성하여 새로 생성된(또는 전환된) API에 연결해야 합니다. 정책을 연결할 때까지는 그 API에 대한 모든 호출이 실패합니다. 자세한 내용은 ??? 단원을 참조하세요.
- 사용자 지정 도메인 이름은 프라이빗 API에 사용할 수 없습니다.
- VPC 엔드포인트 하나를 사용하여 여러 개의 프라이빗 API에 액세스할 수 있습니다.
- VPC 종단점을 REST API에 연결하거나 연결을 해제하여 Route 53 별칭에 DNS 레코드를 제공하고 프라이빗 API 호출을 간소화할 수 있습니다. 자세한 내용은 VPC 엔드포인트를 프라이빗 REST API와 연결 또는 연결 해제를 참조하십시오.

> **Note**
>
> 프라이빗 API를 위한 VPC 엔드포인트에도 다른 인터페이스 VPC 엔드포인트와 동일한 제한이 적용됩니다. 자세한 내용은 AWS PrivateLink 가이드의 인터페이스 엔드포인트 속성 및 제한을 참조하세요. 공유 VPC 및 공유 서브넷에서 API Gateway를 사용하는 방법에 대한 자세한 내용은 AWS PrivateLink 가이드의 공유 서브넷을 참조하세요.

주제

- API Gateway execute-api용 인터페이스 VPC 종단점 생성
- API Gateway 콘솔을 사용하여 프라이빗 API 생성
- 를 사용하여 프라이빗 API를 생성합니다. AWS CLI
- AWS SDK를 사용하여 프라이빗 API를 생성하십시오. JavaScript
- 프라이빗 API에 대한 리소스 정책 설정
- API Gateway 콘솔을 사용하여 프라이빗 API 배포

- VPC 엔드포인트를 프라이빗 REST API와 연결 또는 연결 해제

API Gateway **execute-api**용 인터페이스 VPC 종단점 생성

API 실행을 위한 API Gateway 구성 요소 서비스를 execute-api라고 합니다. 배포된 프라이빗 API에 액세스하려면 VPC에서 해당하는 인터페이스 VPC 종단점을 생성해야 합니다.

VPC 종단점을 생성한 뒤 이를 사용하여 여러 개의 프라이빗 API에 액세스할 수 있습니다.

API Gateway **execute-api**용 인터페이스 VPC 종단점을 만들려면

1. AWS Management Console 로그인하고 https://console.aws.amazon.com/vpc/ 에서 Amazon VPC 콘솔을 엽니다.
2. 탐색 창에서 [Endpoints]와 [Create Endpoint]를 선택합니다.
3. 서비스 범주에서 AWS 서비스를 선택해야 합니다.
4. 서비스 이름에서 연결하려는 AWS 지역을 포함하여 API Gateway 서비스 엔드포인트를 선택합니다. 형식은 `com.amazonaws.region.execute-api`입니다(예: `com.amazonaws.us-east-1.execute-api`).

   [Type]에서 [Interface]를 나타내는지 확인합니다.
5. 다음 정보를 완성합니다.

   - VPC에서 엔드포인트를 생성할 VPC를 선택합니다.
   - 서브넷에서 엔드포인트 네트워크 인터페이스를 만들 서브넷(가용 영역)을 선택합니다. API의 가용성을 높이려면 여러 서브넷을 선택합니다.

     > **Note**
     >
     > 일부 AWS 서비스에서 지원되지 않는 가용 영역도 있습니다.

   - 프라이빗 DNS 이름 활성화를 위해 확인란을 선택된 상태로 둡니다. 프라이빗 DNS는 기본적으로 활성화되어 있습니다.

     프라이빗 DNS를 활성화하면 프라이빗 DNS 또는 퍼블릭 DNS를 통해 API에 액세스할 수 있게 됩니다. (이 설정은 API에 액세스할 수 있는 사용자가 아니라, 사용할 수 있는 DNS 주소 유형에만 영향을 미칩니다.) 그러나 프라이빗 DNS가 활성화된 API Gateway VPC 종단점을 사용하면 VPC에서 퍼블릭 API에 액세스할 수 없습니다 이러한 DNS 설정은 엣지 최적화된 사용자 지정 도메인 이름을 사용하여 퍼블릭 API에 액세스하는 경우 VPC에서 이러한 퍼블릭 API를 호출하

는 기능에는 영향을 미치지 않습니다. (프라이빗 DNS를 사용하여 프라이빗 API에 액세스하면서) 엣지 최적화된 사용자 지정 도메인 이름을 사용하여 퍼블릭 API에 액세스하는 것은 프라이빗 DNS가 활성화된 엔드포인트를 생성한 VPC에서 퍼블릭 API와 프라이빗 API에 모두 액세스할 수 있는 한 가지 방법입니다.

> **Note**
>
> 프라이빗 DNS를 활성화된 상태로 두는 것이 바람직합니다. 프라이빗 DNS를 활성화하지 않으면 퍼블릭 DNS를 통해서만 API에 액세스할 수 있습니다. 자세한 내용은 프라이빗 API를 호출하는 방법를 참조하세요.

프라이빗 DNS 옵션을 사용하려면 VPC 속성 `enableDnsSupport` 및 `enableDnsHostnames`를 `true`로 설정해야 합니다. 자세한 내용은 Amazon VPC 사용 설명서의 VPC에서 DNS 지원 및 VPC에 대한 DNS 지원 업데이트를 참조하세요.

- 보안 그룹에서 VPC 엔드포인트 네트워크 인터페이스와 연결할 보안 그룹을 선택합니다.

  선택한 보안 그룹은 VPC의 IP 범위 또는 VPC의 다른 보안 그룹으로부터 오는 TCP 포트 443의 인바운드 HTTPS 트래픽을 허용하도록 설정되어 있어야 합니다.

6. [Create endpoint]를 선택합니다.

API Gateway 콘솔을 사용하여 프라이빗 API 생성

API Gateway 콘솔을 사용하여 프라이빗 API를 생성하려면

1. https://console.aws.amazon.com/apigateway에서 API Gateway 콘솔에 로그인합니다.
2. API 생성을 선택합니다.
3. REST API에서 빌드를 선택합니다.
4. 이름에 이름을 입력합니다.
5. (선택 사항) 설명에 설명을 입력합니다.
6. API 엔드포인트 유형에서 프라이빗을 선택합니다.
7. API 생성(Create API)을 선택합니다.

여기서부터는???의 1 - 6단계에 설명된 대로 API 메서드와 이에 연결된 통합을 설정하는 단계로 나아갈 수 있습니다.

> **Note**
>
> API에 VPC 또는 VPC 엔드포인트에 대한 액세스 권한을 부여하는 리소스 정책이 생길 때까지는 모든 API 호출이 실패합니다. API를 테스트하고 배포하기 전에 에 설명된 것과 같이 리소스 정책을 생성하고 이 정책을 API에 연결해야 합니다???

를 사용하여 프라이빗 API를 생성합니다. AWS CLI

를 사용하여 프라이빗 API를 생성하려면 다음 `create-rest-api` 명령을 호출합니다. AWS CLI

```
aws apigateway create-rest-api \
        --name 'Simple PetStore (AWS CLI, Private)' \
        --description 'Simple private PetStore API' \
        --region us-west-2 \
        --endpoint-configuration '{ "types": ["PRIVATE"] }'
```

성공적으로 호출되면 다음과 비슷한 출력 결과를 반환합니다.

```
{
    "createdDate": "2017-10-13T18:41:39Z",
    "description": "Simple private PetStore API",
    "endpointConfiguration": {
        "types": "PRIVATE"
    },
    "id": "0qzs2sy7bh",
    "name": "Simple PetStore (AWS CLI, Private)"
}
```

여기서부터는 the section called “AWS CLI 명령을 사용하여 엣지 최적화 API 설정”에 제공된 지침에 따라 이 API에 대해 메서드와 통합을 설정할 수 있습니다.

API를 테스트할 준비가 되면, 에 설명된 것과 같이 리소스 정책을 만들고 이 정책을 API에 연결합니다???.

AWS SDK를 사용하여 프라이빗 API를 생성하십시오. JavaScript

AWS SDK를 사용하여 프라이빗 API를 만들려면: JavaScript

```
apig.createRestApi({
```

```
 name: "Simple PetStore (node.js SDK, private)",
 endpointConfiguration: {
  types: ['PRIVATE']
 },
 description: "Demo private API created using the AWS SDK for node.js",
 version: "0.00.001"
}, function(err, data){
 if (!err) {
  console.log('Create API succeeded:\n', data);
  restApiId = data.id;
 } else {
  console.log('Create API failed:\n', err);
 }
});
```

성공적으로 호출되면 다음과 비슷한 출력 결과를 반환합니다.

```
{
    "createdDate": "2017-10-13T18:41:39Z",
    "description": "Demo private API created using the AWS SDK for node.js",
    "endpointConfiguration": {
        "types": "PRIVATE"
    },
    "id": "0qzs2sy7bh",
    "name": "Simple PetStore (node.js SDK, private)"
}
```

이전 단계를 완료한 후에 the section called “Node.js용 AWS SDK를 사용한 엣지 최적화 API 설정”의 지침에 따라 이 API에 대해 메서드와 통합을 설정할 수 있습니다.

API를 테스트할 준비가 되면, 에 설명된 것과 같이 리소스 정책을 만들고 이 정책을 API에 연결합니다???.

프라이빗 API에 대한 리소스 정책 설정

프라이빗 API에 액세스하려면 먼저 리소스 정책을 생성하여 API에 연결해야 합니다. 이렇게 하면 VPC와 VPC 엔드포인트 또는 액세스 권한을 명시적으로 부여한 다른 계정의 VPC와 VPC 엔드포인트에서 AWS API에 액세스할 수 있습니다.

그러려면 the section called “API Gateway 리소스 정책을 생성하여 API에 연결”의 지침을 따르십시오. 5단계에서 소스 VPC 예제를 선택합니다. {{vpceID}}(중괄호 포함)를 사용자의 VPC 종단점 ID로 바꾼 후 저장을 선택하여 리소스 정책을 저장합니다.

또한 부여되는 액세스를 지정하기 위해 VPC 종단점에 엔드포인트 정책을 연결하는 것이 좋습니다. 자세한 내용은 the section called “프라이빗 API에 대한 VPC 종단점 정책 사용” 단원을 참조하세요.

API Gateway 콘솔을 사용하여 프라이빗 API 배포

프라이빗 API를 배포하려면 API Gateway 콘솔에서 다음을 수행하세요.

1. API를 선택합니다.
2. Deploy API(API 배포)를 선택합니다.
3. 스테이지에서 새 스테이지를 선택합니다.
4. 스테이지 이름에 스테이지 이름을 입력합니다.
5. (선택 사항) 설명에 설명을 입력합니다.
6. 배포를 선택합니다.

VPC 엔드포인트를 프라이빗 REST API와 연결 또는 연결 해제

VPC 종단점을 프라이빗 API와 연결하면 API Gateway에서 새로운 Route 53 ALIAS DNS 레코드를 생성합니다. `Host` 헤더를 재정의하거나 `x-apigw-api-id` 헤더를 전달하지 않고 에지 최적화 또는 리전 API를 호출할 때처럼 이 레코드를 사용하여 비공개 API를 호출할 수 있습니다.

생성된 기본 URL의 형식은 다음과 같습니다.

```
https://{rest-api-id}-{vpce-id}.execute-api.{region}.amazonaws.com/{stage}
```

VPC 엔드포인트를 프라이빗 REST API와 연결하거나 연결을 해제하려면 API 구성을 업데이트해야 합니다. API Gateway 콘솔, AWS CLI 또는 API Gateway용 AWS SDK를 사용하여 이 변경을 수행할 수 있습니다. DNS 전파로 인해 업데이트 작업을 완료하는 데 몇 분이 걸릴 수 있습니다. 그 동안에도 API를 사용할 수 있지만, 새로 생성된 DNS URL의 DNS 전파는 계속 진행 중일 수 있습니다. 몇 분 후에도 DNS에서 새 URL이 확인되지 않으면 API의 새로운 배포 생성을 시도할 수 있습니다.

API Gateway 콘솔을 사용하여 VPC 엔드포인트를 프라이빗 REST API와 연결

추가 VPC 엔드포인트를 프라이빗 API와 연결하려면

1. https://console.aws.amazon.com/apigateway에서 API Gateway 콘솔에 로그인합니다.
2. 프라이빗 API를 선택합니다.
3. 기본 탐색 창에서 리소스 정책을 선택합니다.

4. 추가 VPC 엔드포인트에서 직접적으로 호출할 수 있도록 리소스 정책을 편집합니다.
5. 기본 탐색 창에서 API 설정을 선택합니다.
6. API 세부 정보 섹션에서 편집을 선택합니다.
7. VPC 엔드포인트 ID에서 추가 VPC 엔드포인트 ID를 선택합니다.
8. 저장을 선택합니다.
9. 변경 사항을 적용하려면 API를 재배포합니다.

AWS CLI를 사용하여 VPC 엔드포인트를 프라이빗 REST API와 연결합니다.

API 생성 시 VPC 엔드포인트를 연결하려면 다음 명령을 사용하세요.

```
aws apigateway create-rest-api \
    --name Petstore \
    --endpoint-configuration '{ "types": ["PRIVATE"], "vpcEndpointIds" :
 ["vpce-0212a4ababd5b8c3e", "vpce-0393a628149c867ee"] }' \
    --region us-west-2
```

출력은 다음과 같습니다.

```
{
    "apiKeySource": "HEADER",
    "endpointConfiguration": {
        "types": [
            "PRIVATE"
        ],
        "vpcEndpointIds": [
            "vpce-0212a4ababd5b8c3e",
            "vpce-0393a628149c867ee"
        ]
    },
    "id": "u67n3ov968",
    "createdDate": 1565718256,
    "name": "Petstore"
}
```

VPC 엔드포인트를 이미 생성된 프라이빗 API에 연결하려면 다음 CLI 명령을 사용하세요.

```
aws apigateway update-rest-api \
```

```
    --rest-api-id u67n3ov968 \
    --patch-operations "op='add',path='/endpointConfiguration/
vpcEndpointIds',value='vpce-01d622316a7df47f9'" \
    --region us-west-2
```

출력은 다음과 같습니다.

```
{
    "name": "Petstore",
    "apiKeySource": "1565718256",
    "tags": {},
    "createdDate": 1565718256,
    "endpointConfiguration": {
        "vpcEndpointIds": [
            "vpce-0212a4ababd5b8c3e",
            "vpce-0393a628149c867ee",
            "vpce-01d622316a7df47f9"
        ],
        "types": [
            "PRIVATE"
        ]
    },
    "id": "u67n3ov968"
}
```

API Gateway 콘솔을 사용하여 프라이빗 REST API에서 VPC 엔드포인트 연결 해제

프라이빗 REST API에서 VPC 엔드포인트 연결을 해제하려면

1. https://console.aws.amazon.com/apigateway에서 API Gateway 콘솔에 로그인합니다.
2. 프라이빗 API를 선택합니다.
3. 기본 탐색 창에서 리소스 정책을 선택합니다.
4. 리소스 정책을 편집하여 프라이빗 API에서 연결 해제할 VPC 엔드포인트에 대한 언급을 제거합니다.
5. 기본 탐색 창에서 API 설정을 선택합니다.
6. API 세부 정보 섹션에서 편집을 선택합니다.
7. VPC 엔드포인트 ID에서 X를 선택하여 VPC 엔드포인트를 연결 해제합니다.
8. 저장을 선택합니다.
9. 변경 사항을 적용하려면 API를 재배포합니다.

AWS CLI를 사용하여 프라이빗 REST API에서 VPC 엔드포인트 연결 해제

프라이빗 API에서 VPC 엔드포인트 연결을 해제하려면 다음 CLI 명령을 사용하세요.

```
aws apigateway update-rest-api \
    --rest-api-id u67n3ov968 \
    --patch-operations "op='remove',path='/endpointConfiguration/
vpcEndpointIds',value='vpce-0393a628149c867ee'" \
    --region us-west-2
```

출력은 다음과 같습니다.

```
{
    "name": "Petstore",
    "apiKeySource": "1565718256",
    "tags": {},
    "createdDate": 1565718256,
    "endpointConfiguration": {
        "vpcEndpointIds": [
            "vpce-0212a4ababd5b8c3e",
            "vpce-01d622316a7df47f9"
        ],
        "types": [
            "PRIVATE"
        ]
    },
    "id": "u67n3ov968"
}
```

REST API 모니터링

이 섹션에서는 CloudWatch 측정항목, CloudWatch 로그, Firehose 등을 사용하여 API를 모니터링하는 방법을 알아볼 수 있습니다. AWS X-Ray CloudWatch 실행 로그와 CloudWatch 지표를 결합하여 오류 및 실행 추적을 기록하고 API 성능을 모니터링할 수 있습니다. Firehose에 API 호출을 기록하고 싶을 수도 있습니다. API를 구성하는 다운스트림 서비스를 통해 호출을 추적하는 AWS X-Ray 데 사용할 수도 있습니다.

> **Note**
>
> API Gateway는 다음과 같은 경우 로그 및 지표를 생성하지 않을 수 있습니다.

- 413 요청 엔터티가 너무 큼 오류가 발생한 경우
- 429개 초과로 요청이 너무 많음 오류가 발생한 경우
- API 매핑이 없는 사용자 지정 도메인으로 보낸 요청에서 400개의 연속 오류가 발생한 경우
- 내부 오류로 인해 500개의 연속 오류가 발생한 경우

API Gateway는 REST API 메서드를 테스트할 때 로그와 지표를 생성하지 않습니다. CloudWatch 항목이 시뮬레이션됩니다. 자세한 내용은 the section called “콘솔을 사용하여 REST API 메서드 테스트”을(를) 참조하세요.

주제

- Amazon CloudWatch 지표를 사용한 REST API 실행 모니터링
- API Gateway에서 REST API에 대한 CloudWatch 로깅 설정
- Amazon 데이터 파이어호스에 대한 API 호출 로깅
- X-Ray를 사용하여 REST API에 대한 사용자 요청 추적

Amazon CloudWatch 지표를 사용한 REST API 실행 모니터링

API Gateway에서 원시 데이터를 수집하고 읽기 가능한 실시간에 가까운 지표로 처리하는 CloudWatch를 사용하여 API 실행을 모니터링할 수 있습니다. 이러한 통계는 15개월 간 기록되므로 기록 정보에 액세스하고 웹 애플리케이션이나 서비스가 어떻게 수행되고 있는지 전체적으로 더 잘 파악할 수 있습니다. 기본적으로 API Gateway 지표 데이터는 1분 간격으로 CloudWatch로 자동 전송됩니다. 자세한 내용은 Amazon CloudWatch 사용 설명서의 Amazon CloudWatch란 무엇입니까?를 참조하십시오.

API Gateway에서 보고하는 지표는 다양한 방법으로 분석할 수 있는 정보를 제공합니다. 다음 목록은 시작을 위한 권장 사항인 지표의 몇 가지 일반적인 용도를 보여줍니다.

- IntegrationLatency 측정치를 모니터링하여 백엔드의 응답 속도를 측정합니다.
- Latency 측정치를 모니터링하여 API 호출의 전반적인 응답 속도를 측정합니다.
- CacheHitCount 및 CacheMissCount 측정치를 모니터링하여 캐시 용량을 최적화하고 원하는 성능을 달성합니다.

주제

- Amazon API Gateway 차원 및 지표
- API Gateway의 API 대시보드를 사용하여 CloudWatch 지표 보기
- CloudWatch 콘솔에서 API Gateway 지표 보기
- CloudWatch 콘솔에서 API Gateway 로그 이벤트 보기
- 의 모니터링 도구AWS

Amazon API Gateway 차원 및 지표

API Gateway가 Amazon에 전송하는 지표 및 측정기준은 CloudWatch 다음과 같습니다. 자세한 설명은 Amazon CloudWatch 지표를 사용한 REST API 실행 모니터링 섹션을 참조하세요.

API Gateway 지표

Amazon API Gateway는 CloudWatch 매분마다 지표 데이터를 전송합니다.

`AWS/ApiGateway` 네임스페이스에 포함된 지표는 다음과 같습니다.

지표	설명
4XXError	지정한 기간 내에 캡처된 클라이언트 측 오류 수 API Gateway는 수정된 게이트웨이 응답 상태 코드를 4xxError 오류로 계산합니다. `Sum` 통계는 이 지표 즉, 지정된 기간 내 총 4XXError 오류 개수를 나타냅니다. `Average` 통계는 4XXError 오류율 즉, 총 4XXError 오류 개수를 해당 기간 동안의 총 요청 수로 나눈 것입니다. 분모는 Count 지표에 해당합니다(아래). Unit: Count
5XXError	지정한 기간 내에 캡처된 서버 측 오류 수. `Sum` 통계는 이 지표 즉, 지정된 기간 내 총 5XXError 오류 개수를 나타냅니다. `Average` 통계는 5XXError 오류율 즉, 총 5XXError 오류 개수를 해당 기간 동안의 총 요청 수로 나눈 것입니다. 분모는 Count 지표에 해당합니다(아래).

지표	설명
	Unit: Count
CacheHitCount	지정된 기간 내 API 캐시에서 처리한 요청 수 Sum 통계는 이 지표 즉, 지정된 기간 내 총 캐시 적중 수를 나타냅니다. Average 통계는 캐시 적중률 즉, 총 캐시 적중 수를 해당 기간 동안의 총 요청 수로 나눈 것입니다. 분모는 Count 지표에 해당합니다(아래). Unit: Count
CacheMissCount	API 캐싱이 활성화되어 있을 때 지정된 기간 동안 백엔드에서 처리된 요청 수 Sum 통계는 이 지표 즉, 지정된 기간 내 총 캐시 누락 수를 나타냅니다. Average 통계는 캐시 누락률 즉, 총 캐시 누락 수를 해당 기간 동안의 총 요청 수로 나눈 것입니다. 분모는 Count 지표에 해당합니다(아래). Unit: Count
Count	지정된 기간 동안의 총 API 요청 수. SampleCount 통계는 이 지표를 나타냅니다. Unit: Count
IntegrationLatency	API Gateway가 요청을 백엔드로 릴레이할 때부터 백엔드에서 응답을 수신할 때까지의 시간입니다. Unit: Millisecond
Latency	API Gateway가 클라이언트에서 요청을 수신할 때부터 클라이언트에게 응답을 반환할 때까지의 시간입니다. 지연 시간에는 통합 지연 시간과 기타 API Gateway 오버헤드가 포함됩니다. Unit: Millisecond

지표 차원

다음 표의 차원을 사용하여 API Gateway 지표를 필터링할 수 있습니다.

> **Note**
>
> API Gateway는 측정치를 로 전송하기 전에 ApiName 차원에서 ASCII가 아닌 문자를 제거합니다. CloudWatch APIName에 ASCII 문자가 없는 경우 API ID가 ApiName으로 사용됩니다.

차원	설명
ApiName	지정한 API 이름을 사용하여 REST API에 대한 API Gateway 지표를 필터링합니다.
ApiName, Method, Resource, Stage	지정한 API 이름, 단계, 리소스 및 메서드를 사용하여 API 메서드에 대한 API Gateway 지표를 필터링합니다. API Gateway는 세부 CloudWatch 지표를 명시적으로 활성화하지 않는 한 이러한 지표를 전송하지 않습니다. 콘솔에서 스테이지를 선택한 다음 로그 및 추적에서 편집을 선택합니다. 세부 지표를 선택한 다음 변경 사항 저장을 선택합니다. 또는 update-stage AWS CLI 명령을 호출하여 속성을 업데이트하는 방법도 있습니다. `metricsEnabled true` 이러한 지표를 활성화하면 계정에 추가 비용이 발생합니다. 가격 정보는 Amazon CloudWatch 요금을 참조하십시오.
ApiName, Stage	지정한 API 이름과 단계를 사용하여 API 단계 리소스에 대한 API Gateway 지표를 필터링합니다.

API Gateway의 API 대시보드를 사용하여 CloudWatch 지표 보기

API Gateway 콘솔에서 API 대시보드를 사용하여 API Gateway에 배포된 API의 CloudWatch 지표를 표시할 수 있습니다. 시간의 흐름에 따른 API 활동을 요약하여 표시합니다.

주제

- 사전 조건
- 대시보드에서 API 활동 검사

사전 조건

1. API Gateway에서 생성된 API가 있어야 합니다. Amazon API Gateway에서 REST API 생성의 지침을 따르십시오.
2. API는 1회 이상 배포되어야 합니다. Amazon API Gateway에서 REST API 배포의 지침을 따르십시오.

대시보드에서 API 활동 검사

1. https://console.aws.amazon.com/apigateway에서 API Gateway 콘솔에 로그인합니다.
2. API를 선택합니다.
3. 기본 탐색 창에서 대시보드를 선택합니다.
4. 스테이지에서 원하는 스테이지를 선택합니다.
5. 날짜 범위를 지정하려면 날짜 범위를 선택합니다.
6. 필요할 경우 새로 고침을 수행하고 개별 그래프(API 호출, 지연 시간, 통합 지연 시간, 지연 시간, 4xx 오류 및 5xx 오류)에 표시된 개별 측정치를 확인합니다.

> **Tip**
>
> 메서드 수준 CloudWatch 지표를 검사하려면 메서드 수준에서 CloudWatch Logs를 활성화했는지 확인합니다. 메서드 수준 로깅을 설정하는 방법에 대한 자세한 내용은 단원을 참조하십시오API Gateway 콘솔을 사용하여 단계 설정 업데이트

CloudWatch 콘솔에서 API Gateway 지표 보기

측정치는 먼저 서비스 네임스페이스별로 그룹화된 다음, 각 네임스페이스 내에서 다양한 차원 조합별로 그룹화됩니다.

CloudWatch 콘솔을 사용하여 API Gateway 지표를 보려면

1. https://console.aws.amazon.com/cloudwatch/에서 CloudWatch 콘솔을 엽니다.

2. 필요한 경우 AWS 리전을 변경합니다. 탐색 모음에서 AWS 리소스가 상주하는 리전을 선택합니다. 자세한 내용은 리전 및 엔드포인트를 참조하세요.
3. 탐색 창에서 [Metrics]를 선택합니다.
4. 모든 측정치(All metrics) 탭에서 API 게이트웨이(API Gateway)를 선택합니다.
5. 측정치를 단계별로 확인하려면 스테이지별 패널을 선택합니다. 그 다음에 원하는 API와 측정치 이름을 선택합니다.
6. 측정치를 특정 API별로 확인하려면 Api 이름별 패널을 선택합니다. 그 다음에 원하는 API와 측정치 이름을 선택합니다.

AWS CLI를 사용하여 지표를 보려면

1. 명령 프롬프트에서 다음 명령을 사용하여 측정치를 나열합니다.

```
aws cloudwatch list-metrics --namespace "AWS/ApiGateway"
```

2. 5분 간격의 시간 주기에 대해 특정 통계(예: `Average`)를 보려면 다음 명령을 호출합니다.

```
aws cloudwatch get-metric-statistics --namespace AWS/ApiGateway --metric-name Count
 --start-time 2011-10-03T23:00:00Z --end-time 2017-10-05T23:00:00Z --period 300 --
statistics Average
```

CloudWatch 콘솔에서 API Gateway 로그 이벤트 보기

필수 조건

1. API Gateway에서 생성된 API가 있어야 합니다. Amazon API Gateway에서 REST API 생성의 지침을 따르십시오.
2. API를 1회 이상 배포 및 호출한 상태여야 합니다. Amazon API Gateway에서 REST API 배포 및 Amazon API Gateway에서 REST API 호출의 지침을 따르십시오.
3. 스테이지에 대해 CloudWatch Logs를 활성화한 상태여야 합니다. API Gateway에서 REST API에 대한 CloudWatch 로깅 설정의 지침을 따르세요.

CloudWatch 콘솔을 사용하여 로깅된 API 요청 및 응답을 보려면

1. https://console.aws.amazon.com/cloudwatch/에서 CloudWatch 콘솔을 엽니다.

2. 필요한 경우 AWS 리전을 변경합니다. 탐색 모음에서 AWS 리소스가 상주하는 리전을 선택합니다. 자세한 내용은 리전 및 엔드포인트를 참조하세요.
3. 왼쪽 탐색 창에서 로그(Logs), 로그 그룹(Log groups)을 선택합니다.
4. 로그 그룹 테이블에서 API-Gateway-Execution-Logs_{rest-api-id}/{stage-name} 이름의 로그 그룹을 선택합니다.
5. 로그 스트림 테이블에서 로그 스트림을 선택합니다. 타임스탬프를 사용하면 관심 있는 로그 스트림을 찾는 데 도움이 됩니다.
6. 텍스트를 선택하여 원시 텍스트를 보거나 열을 선택하여 각 행의 이벤트를 확인합니다.

> **Important**
>
> CloudWatch를 사용하여 로그 그룹 또는 스트림을 삭제할 수 있습니다. API Gateway API 로그 그룹 또는 스트림을 수동으로 삭제하지 마십시오. 그 대신 API Gateway가 이러한 리소스를 관리하도록 하십시오. 로그 그룹 또는 스트림을 수동으로 삭제하면 API 요청 및 응답이 기록되지 않을 수 있습니다. 이런 일이 발생하는 경우, API에 대한 전체 로그 그룹을 삭제하고 API를 다시 배포할 수 있습니다. 이렇게 하는 이유는 API Gateway는 API가 배포되는 시점에 API 단계에 대해 로그 그룹 또는 로그 스트림을 생성하기 때문입니다.

의 모니터링 도구AWS

AWS는 API Gateway를 모니터링하는 데 사용할 수 있는 다양한 도구를 제공합니다. 모니터링을 자동으로 수행하도록 구성할 수 있는 도구도 있지만, 수동 작업이 필요한 도구도 있습니다. 모니터링 작업을 최대한 자동화하는 것이 좋습니다.

의 자동 모니터링 도구AWS

다음과 같은 자동 모니터링 도구를 사용하여 API Gateway를 감시하고 문제가 발생할 경우 보고할 수 있습니다.

- Amazon CloudWatch 경보 – 지정한 기간 동안 단일 지표를 감시하고, 여러 기간에 대해 지정된 임계값과 관련하여 지표 값을 기준으로 하나 이상의 작업을 수행합니다. 이 태스크는 Amazon Simple Notification Service(Amazon SNS) 주제 또는 Amazon EC2 Auto Scaling 정책에 전송되는 알림입니다. CloudWatch 경보는 특정 상태에 있다는 이유만으로는 태스크를 호출하지 않습니다. 상태가 변경되고 지정한 기간 동안 유지되어야 합니다. 자세한 내용은 Amazon CloudWatch 지표를 사용한 REST API 실행 모니터링 단원을 참조하세요.

- Amazon CloudWatch Logs – AWS CloudTrail 또는 기타 소스의 로그 파일을 모니터링, 저장 및 액세스합니다. 자세한 내용은 Amazon CloudWatch 사용 설명서의 로그 파일 모니터링을 참조하세요.
- Amazon CloudWatch Events – 이벤트를 일치시키고 하나 이상의 대상 함수 또는 스트림으로 라우팅하여 값을 변경하거나 상태 정보를 캡처하거나 수정 작업을 수행합니다. 자세한 내용은 Amazon CloudWatch 사용 설명서의 Amazon CloudWatch Events란 무엇입니까?를 참조하세요.
- AWS CloudTrail Log Monitoring – 계정 간에 로그 파일을 공유하고, CloudTrail 로그 파일을 CloudWatch Logs에 전송하여 실시간으로 모니터링하며, Java에서 로그 처리 애플리케이션을 작성하고, CloudTrail에서 전송한 후 로그 파일이 변경되지 않았는지 확인합니다. 자세한 내용은 AWS CloudTrail 사용 설명서의 CloudTrail 로그 파일 작업을 참조하세요.

수동 모니터링 도구

API Gateway 모니터링의 또 한 가지 중요한 부분은 CloudWatch 경보에 포함되지 않는 항목을 수동으로 모니터링해야 한다는 점입니다. API Gateway, CloudWatch 및 기타 AWS 콘솔 대시보드에서 AWS 환경의 상태를 한눈에 볼 수 있습니다. API 실행에 대한 로그 파일도 확인하는 것이 좋습니다.

- API Gateway 대시보드에는 지정된 기간 동안 지정된 API 단계에 대해 다음과 같은 통계가 표시됩니다.
  - API 호출
  - 캐시 적중(Cache Hit)은 API 캐싱이 활성화될 때만 해당됩니다.
  - 캐시 누락(Cache Miss)은 API 캐싱이 활성화될 때만 해당됩니다.
  - Latency
  - 통합 지연 시간
  - 4XX 오류
  - 5XX 오류
- CloudWatch 홈 페이지에는 다음 내용이 표시됩니다.
  - 현재 경보 및 상태
  - 경보 및 리소스 그래프
  - 서비스 상태

  또한 CloudWatch를 사용하여 다음을 수행할 수 있습니다.
  - 사용자 지정 대시보드를 생성하여 관심 있는 서비스 모니터링
  - 지표 데이터를 그래프로 작성하여 문제를 해결하고 추세 파악
  - 모든 AWS 리소스 지표 검색 및 찾아보기

- 문제에 대해 알려주는 경보 생성 및 편집

API Gateway를 모니터링하기 위한 CloudWatch 경보 생성

경보 때문에 상태가 변경되면 Amazon SNS 메시지를 보내는 CloudWatch 경보를 생성할 수 있습니다. 경보는 지정한 기간에 단일 지표를 감시하고 여러 기간에 지정된 임계값에 대한 지표 값을 기준으로 작업을 하나 이상 수행합니다. 이 작업은 Amazon SNS 주제 또는 Auto Scaling 정책으로 전송되는 알림입니다. 경보는 지속적인 상태 변경에 대해서만 작업을 호출합니다. CloudWatch 경보는 특정 상태에 있다는 이유만으로는 태스크를 호출하지 않습니다. 상태가 변경되고 지정한 기간 동안 유지되어야 합니다.

API Gateway에서 REST API에 대한 CloudWatch 로깅 설정

요청 실행 또는 API에 대한 클라이언트 액세스와 관련된 문제를 디버깅하기 위해 Amazon CloudWatch Logs를 활성화하여 API 호출을 로깅할 수 있습니다. CloudWatch에 대한 자세한 내용은 the section called “CloudWatch 지표” 단원을 참조하십시오.

API Gateway에 대한 CloudWatch 로그 형식

CloudWatch의 API 로깅에는 실행 로깅과 액세스 로깅이라는 두 가지 유형이 있습니다. 실행 로깅에서 API Gateway는 CloudWatch Logs를 관리합니다. 이 프로세스에는 로그 그룹 및 로그 스트림을 생성하는 작업과 호출자의 요청 및 응답을 로그 스트림에 보고하는 작업이 포함됩니다.

로깅된 데이터에는 오류 또는 실행 추적(예: 요청 또는 응답 파라미터 값 또는 페이로드), Lambda 권한 부여자(이전에는 사용자 지정 권한 부여자라고 함)가 사용하는 데이터, API 키가 필요한지 여부, 사용량 계획이 활성화되는지 여부 등이 포함됩니다.

API를 배포하면 API Gateway는 로그 그룹을 생성하고 로그 그룹 아래에 로그 스트림을 생성합니다. 로그 그룹은 `API-Gateway-Execution-Logs_{rest-api-id}/{stage_name}` 형식에 따라 명명됩니다. 로그는 각 로그 그룹 내에서 로그 스트림으로 다시 나뉘며, 로그 스트림은 로깅된 데이터가 보고될 때 마지막 이벤트 시간 기준으로 정렬됩니다.

액세스 로깅에서 API 개발자가 원하는 것은 누가 API에 액세스했고 호출자가 API에 어떻게 액세스했는지 로깅하는 것입니다. 자체 로그 그룹을 생성하거나 API Gateway에서 관리할 수 있는 기존 로그 그룹을 선택할 수 있습니다. 액세스 세부 정보를 지정하려면 `$context` 변수(선택한 형식으로 표현)를 선택하고 로그 그룹을 대상으로 선택합니다. 액세스 로그 형식에는 최소한 `$context.requestId` 또는 `$context.extendedRequestId`가 포함되어야 합니다. 모범 사례로 로그 형식에 `$context.requestId`과 `$context.extendedRequestId`를 포함합니다. `$context.requestId`은 `x-amzn-RequestId` 헤더에 값을 로그합니다. 클라이언트는 `x-amzn-`

RequestId 헤더의 값을 범용 고유 식별자(UUID) 형식의 값으로 재정의할 수 있습니다. API Gateway는 x-amzn-RequestId 응답 헤더에서 이 요청 ID를 반환합니다. API Gateway는 UUID 형식이 아닌 재정의된 요청 ID를 액세스 로그에 있는 *UUID*_REPLACED_INVALID_REQUEST_ID로 대체합니다. $context.extendedRequestId는 API Gateway가 생성하는 고유한 ID입니다. API Gateway는 x-amz-apigw-id 응답 헤더에서 이 요청 ID를 반환합니다. API 호출자는 이 요청 ID를 제공하거나 재정의할 수 없습니다. 자세한 내용은 the section called “$context데이터 모델, 권한 부여자, 매핑 템플릿, 액세스 로깅용 변수 CloudWatch ” 섹션을 참조하세요.

> **Note**
>
> $context 변수만 지원되며 $input 등은 지원되지 않습니다.

Common Log Format(CLF), JSON, XML 또는 CSV 같은 분석 백엔드도 채택하는 로그 형식을 선택하십시오. 그러면 액세스 로그를 분석 백엔드에 직접 피드하여 지표를 계산하고 렌더링할 수 있습니다. 로그 형식을 정의하려면 단계의 accessLogSettings/destinationArn 속성에서 로그 그룹 ARN을 설정합니다. ARN 열을 표시하도록 선택한 경우, CloudWatch 콘솔에서 로그 그룹 ARN을 얻을 수 있습니다. 액세스 로그 형식을 정의하려면 단계의 accessLogSetting/format 속성에서 선택한 형식을 설정합니다.

일반적으로 사용되는 몇 가지 액세스 로그 형식의 예가 API Gateway 콘솔에 표시되며 다음과 같이 나열됩니다.

- CLF(Common Log Format):

```
$context.identity.sourceIp $context.identity.caller  \
$context.identity.user [$context.requestTime] \
"$context.httpMethod $context.resourcePath $context.protocol" \
$context.status $context.responseLength $context.requestId $context.extendedRequestId
```

  연속 문자(\)는 시각적 보조 수단으로 사용됩니다. 로그 형식은 한 줄이어야 합니다. 로그 형식의 끝에 줄바꿈 문자(\n)를 추가하여 각 로그 항목의 끝에 줄바꿈 문자를 포함할 수 있습니다.

- 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" \
}
```

연속 문자(\)는 시각적 보조 수단으로 사용됩니다. 로그 형식은 한 줄이어야 합니다. 로그 형식의 끝에 줄바꿈 문자(\n)를 추가하여 각 로그 항목의 끝에 줄바꿈 문자를 포함할 수 있습니다.

- 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>
```

연속 문자(\)는 시각적 보조 수단으로 사용됩니다. 로그 형식은 한 줄이어야 합니다. 로그 형식의 끝에 줄바꿈 문자(\n)를 추가하여 각 로그 항목의 끝에 줄바꿈 문자를 포함할 수 있습니다.

- CSV(쉼표로 분리된 값):

```
$context.identity.sourceIp,$context.identity.caller,\
$context.identity.user,$context.requestTime,$context.httpMethod,\
$context.resourcePath,$context.protocol,$context.status,\
$context.responseLength,$context.requestId,$context.extendedRequestId
```

연속 문자(\)는 시각적 보조 수단으로 사용됩니다. 로그 형식은 한 줄이어야 합니다. 로그 형식의 끝에 줄바꿈 문자(\n)를 추가하여 각 로그 항목의 끝에 줄바꿈 문자를 포함할 수 있습니다.

CloudWatch 로깅에 대한 권한

CloudWatch Logs를 활성화하려면 계정의 CloudWatch에 로그를 읽고 쓸 수 있는 권한을 API Gateway에 부여해야 합니다. `AmazonAPIGatewayPushToCloudWatchLogs` 관리형 정책(ARN이 `arn:aws:iam::aws:policy/service-role/AmazonAPIGatewayPushToCloudWatchLogs`)에는 필요한 모든 권한이 있습니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:DescribeLogGroups",
                "logs:DescribeLogStreams",
                "logs:PutLogEvents",
                "logs:GetLogEvents",
                "logs:FilterLogEvents"
            ],
            "Resource": "*"
        }
    ]
}
```

> **Note**
>
> API Gateway는 IAM 역할을 수임하기 위해 AWS Security Token Service를 호출하므로 AWS STS가 해당 리전에 대해 활성화되어 있는지 확인하십시오. 자세한 내용은 AWS 리전에서 AWS STS 관리를 참조하세요.

이러한 권한을 계정에 부여하려면 신뢰할 수 있는 엔터티로 `apigateway.amazonaws.com`이 포함된 IAM 역할을 생성하고, 이전 정책을 이 IAM 역할에 연결한 다음, 계정의 cloudWatchRoleArn 속성에서 IAM 역할 ARN을 설정합니다. CloudWatch Logs를 활성화하려는 각 AWS 리전에 대해 개별적으로 cloudWatchRoleArn 속성을 설정해야 합니다.

IAM 역할 ARN을 설정할 때 오류가 발생하는 경우, AWS Security Token Service 계정 설정을 확인하여 사용 중인 리전에 AWS STS가 활성화되어 있는지 확인합니다. AWS STS 활성화에 대한 자세한 내용은 IAM 사용 설명서의 AWS 리전에서 AWS STS 관리를 참조하세요.

API Gateway 콘솔을 사용하여 CloudWatch API 로깅 설정

CloudWatch API 로깅을 설정하려면 API를 단계에 배포해야 합니다. 또한 계정에 대해 적절한 CloudWatch Logs 역할 ARN도 구성해야 합니다.

1. https://console.aws.amazon.com/apigateway에서 API Gateway 콘솔에 로그인합니다.
2. 기본 탐색 창에서 설정을 선택한 다음 로깅에서 편집을 선택합니다.
3. CloudWatch 로그 역할 ARN에서 적절한 권한이 있는 IAM 역할의 ARN을 입력합니다. API Gateway를 사용하여 API를 생성하는 AWS 계정마다 이 작업을 한 번씩 수행해야 합니다.
4. 기본 탐색 창에서 API를 선택하고 다음 중 하나를 수행합니다.
   a. 기존 API를 선택한 다음 스테이지를 선택합니다.
   b. API를 생성하여 스테이지에 배포합니다.
5. 기본 탐색 창에서 스테이지를 선택합니다.
6. 로그 및 추적 섹션에서 편집을 선택합니다.
7. 실행 로깅을 활성화하려면 다음과 같이 합니다.
   a. CloudWatch Logs 드롭다운 메뉴에서 로깅 수준을 선택합니다.

      > **Warning**
      > 전체 요청 및 응답 로그는 API 문제를 해결하는 데 유용하지만 민감한 데이터를 로깅할 수 있습니다. 프로덕션 API에는 되도록 전체 요청 및 응답 로그를 사용하지 않는 것이 좋습니다.

   b. 원하는 경우 세부 지표를 선택하여 CloudWatch 세부 지표를 활성화합니다.

   CloudWatch 지표에 대한 자세한 내용은 the section called “CloudWatch 지표” 단원을 참조하십시오.
8. 액세스 로깅을 활성화하려면 다음과 같이 합니다.
   a. 사용자 지정 액세스 로깅을 활성화합니다.

b. 액세스 로그 대상 ARN에 로그 그룹의 ARN을 입력합니다. ARN 형식은 `arn:aws:logs:{region}:{account-id}:log-group:log-group-name`입니다.

c. 로그 형식에 로그 형식을 입력합니다. CLF, JSON, XML 또는 CSV를 선택할 수 있습니다. 예제 로그 형식에 대한 자세한 내용은 the section called “API Gateway에 대한 CloudWatch 로그 형식” 섹션을 참조하세요.

9. 변경 사항 저장을 선택합니다.

> **Note**
>
> 실행 로깅과 액세스 로깅을 서로 독립적으로 활성화할 수 있습니다.

이제 API Gateway가 API에 대한 요청을 로깅할 준비가 되었습니다. 단계 설정, 로그 또는 단계 변수를 업데이트할 때 API를 다시 배포할 필요가 없습니다.

AWS CloudFormation을 사용하여 CloudWatch API 로깅 설정

다음 예제 AWS CloudFormation 템플릿을 사용하여 Amazon CloudWatch Logs 로그 그룹을 생성하고 단계에 대한 실행 및 액세스 로깅을 구성합니다. CloudWatch Logs를 활성화하려면 계정의 CloudWatch에 로그를 읽고 쓸 수 있는 권한을 API Gateway에 부여해야 합니다. 자세한 내용은 AWS CloudFormation 사용 설명서의 계정과 IAM 역할 연결을 참조하세요.

```
  TestStage:
    Type: AWS::ApiGateway::Stage
    Properties:
      StageName: test
      RestApiId: !Ref MyAPI
      DeploymentId: !Ref Deployment
      Description: "test stage description"
      MethodSettings:
        - ResourcePath: "/*"
          HttpMethod: "*"
          LoggingLevel: INFO
      AccessLogSetting:
        DestinationArn: !GetAtt MyLogGroup.Arn
        Format: $context.extendedRequestId $context.identity.sourceIp
$context.identity.caller $context.identity.user [$context.requestTime]
"$context.httpMethod $context.resourcePath $context.protocol" $context.status
$context.responseLength $context.requestId
```

```
  MyLogGroup:
    Type: AWS::Logs::LogGroup
    Properties:
      LogGroupName: !Join
        - '-'
        - - !Ref MyAPI
          - access-logs
```

Amazon 데이터 파이어호스에 대한 API 호출 로깅

API에 대한 클라이언트 액세스와 관련된 문제를 디버깅하는 데 도움이 되도록 Amazon Data Firehose에 API 호출을 기록할 수 있습니다. Firehose에 대한 자세한 내용은 Amazon 데이터 Firehose란 무엇입니까? 를 참조하십시오. .

액세스 로깅의 경우 Firehose만 활성화할 수 있으며 CloudWatch 둘 다 활성화할 수는 없습니다. 하지만 실행 로깅은 CloudWatch 활성화하고 액세스 로깅에는 Firehose를 활성화할 수 있습니다.

주제

- API Gateway의 Firehose 로그 형식
- Firehose 로깅을 위한 권한
- API Gateway 콘솔을 사용하여 Firehose 액세스 로깅을 설정합니다.

API Gateway의 Firehose 로그 형식

Firehose 로깅은 로깅과 동일한 CloudWatch 형식을 사용합니다.

Firehose 로깅을 위한 권한

Firehose 액세스 로깅이 스테이지에 활성화되면 API Gateway는 계정에 서비스 연결 역할을 생성합니다. 해당 역할이 아직 존재하지 않는 경우 API Gateway는 계정에 서비스 연결 역할을 생성합니다. 그 역할의 이름은 `AWSServiceRoleForAPIGateway`이고 이 역할에는 `APIGatewayServiceRolePolicy` 관리형 정책이 연결됩니다. 서비스 연결 역할에 대한 자세한 내용은 서비스 연결 역할 사용 단원을 참조하십시오.

> **Note**
>
> Firehose 스트림의 이름은 다음과 같아야 합니다. `amazon-apigateway-{your-stream-name}`

API Gateway 콘솔을 사용하여 Firehose 액세스 로깅을 설정합니다.

API 로깅을 설정하려면 API를 단계에 배포해야 합니다. 또한 Firehose 스트림도 생성해야 합니다.

1. https://console.aws.amazon.com/apigateway에서 API Gateway 콘솔에 로그인합니다.
2. 다음 중 하나를 수행합니다.
   a. 기존 API를 선택한 다음 스테이지를 선택합니다.
   b. API를 생성하여 단계에 배포합니다.
3. 기본 탐색 창에서 스테이지를 선택합니다.
4. 로그 및 추적 섹션에서 편집을 선택합니다.
5. Firehose 스트림에 대한 액세스 로깅을 활성화하려면:
   a. 사용자 지정 액세스 로깅을 활성화합니다.
   b. 액세스 로그 대상 ARN에 Firehose 스트림의 ARN을 입력합니다. ARN 형식은 `arn:aws:firehose:{region}:{account-id}:deliverystream/amazon-apigateway-{your-stream-name}`입니다.

      > **Note**
      >
      > Firehose 스트림의 이름은 다음과 같아야 합니다. `amazon-apigateway-{your-stream-name}`

   c. 로그 형식에 로그 형식을 입력합니다. CLF, JSON, XML 또는 CSV를 선택할 수 있습니다. 예제 로그 형식에 대한 자세한 내용은 the section called “API Gateway에 대한 CloudWatch 로그 형식” 섹션을 참조하세요.
6. 변경 사항 저장를 선택합니다.

이제 API Gateway에서 API에 대한 요청을 Firehose에 기록할 준비가 되었습니다. 단계 설정, 로그 또는 단계 변수를 업데이트할 때 API를 다시 배포할 필요가 없습니다.

X-Ray를 사용하여 REST API에 대한 사용자 요청 추적

AWS X-Ray를 사용하여 Amazon API Gateway REST API를 통해 기본 서비스로 이동하는 사용자 요청을 추적하고 분석할 수 있습니다. API Gateway는 모든 API Gateway REST API 엔드포인트 유형(리전, 엣지 최적화, 프라이빗)에 대해 X-Ray 추적을 지원합니다. X-Ray를 사용할 수 있는 모든 AWS 리전에서 Amazon API Gateway와 함께 X-Ray를 사용할 수 있습니다.

X-Ray는 전체 요청에 대한 엔드 투 엔드 보기를 제공하므로 API와 백엔드 서비스에서 지연 시간을 분석할 수 있습니다. X-Ray 서비스 맵을 사용하여 전체 요청의 지연 시간과 X-Ray와 통합되는 다운스트림 서비스의 지연 시간을 볼 수 있습니다. 또한 샘플링 규칙을 구성하여 지정한 기준에 따라 어떤 요청을 어떤 샘플링 속도로 기록할지를 X-Ray에 지정할 수도 있습니다.

이미 추적 중인 서비스에서 API Gateway API를 호출하면, API에서 X-Ray 추적을 활성화하지 않더라도 API Gateway가 추적을 전달합니다.

API Gateway 콘솔을 사용하거나 API Gateway API 또는 CLI를 사용하여 API 단계에 대해 X-Ray를 활성화할 수 있습니다.

주제

- API Gateway REST API를 사용하여 AWS X-Ray 설정
- API Gateway에서 AWS X-Ray 서비스 맵 및 트레이스 보기 사용
- API Gateway API에 대한 AWS X-Ray 샘플링 규칙 구성
- Amazon API Gateway API에 대한 AWS X-Ray 추적 이해

API Gateway REST API를 사용하여 AWS X-Ray 설정

이 단원에서는 API Gateway REST API를 사용하여 AWS X-Ray를 설정하는 방법에 대한 자세한 내용을 알아볼 수 있습니다.

주제

- API Gateway에 대한 X-Ray 추적 모드
- X-Ray 추적에 대한 권한
- API Gateway 콘솔에서 X-Ray 추적 활성화
- API Gateway CLI를 사용하여 AWS X-Ray 추적 활성화

API Gateway에 대한 X-Ray 추적 모드

애플리케이션을 경유하는 요청 경로는 트레이스 ID로 트레이스됩니다. 트레이스는 단일 요청(보통 HTTP `GET` 또는 `POST` 요청)에 의해 생성된 모든 세그먼트를 수집합니다.

API Gateway API에 대한 두 가지 추적 모드가 있습니다.

- 수동: API 단계에서 X-Ray 추적을 활성화하지 않은 경우 이 모드가 기본 설정입니다. 이 접근 방식에서는 업스트림 서비스에서 X-Ray를 활성화한 경우에만 API Gateway API를 추적합니다.

- 활성: API Gateway API 단계에 이 설정이 있으면 API Gateway는 X-Ray에서 지정한 샘플링 알고리즘을 기반으로 API 호출 요청을 자동으로 샘플링합니다.

  단계에서 활성 추적을 활성화할 때 역할이 이미 존재하지 않으면, API Gateway가 사용자의 계정에 서비스 연결 역할을 생성합니다. 그 역할의 이름은 `AWSServiceRoleForAPIGateway`이고 이 역할에는 `APIGatewayServiceRolePolicy` 관리형 정책이 연결됩니다. 서비스 연결 역할에 대한 자세한 내용은 서비스 연결 역할 사용 단원을 참조하십시오.

  > **Note**
  >
  > X-Ray는 API가 수신하는 요청의 대표 샘플을 여전히 제공하면서 추적이 효율적으로 수행되도록 샘플링 알고리즘을 적용합니다. 기본 샘플링 알고리즘은 초당 1회 요청인데, 요청의 5%가 이 제한을 넘어서 샘플링됩니다.

API Gateway 관리 콘솔, API Gateway CLI 또는 AWS SDK를 사용하여 API에 대한 추적 모드를 변경할 수 있습니다.

X-Ray 추적에 대한 권한

단계에서 활성 추적을 활성화할 때 역할이 이미 존재하지 않으면, API Gateway가 사용자의 계정에 서비스 연결 역할을 생성합니다. 그 역할의 이름은 `AWSServiceRoleForAPIGateway`이고 이 역할에는 `APIGatewayServiceRolePolicy` 관리형 정책이 연결됩니다. 서비스 연결 역할에 대한 자세한 내용은 서비스 연결 역할 사용 단원을 참조하십시오.

API Gateway 콘솔에서 X-Ray 추적 활성화

Amazon API Gateway 콘솔을 사용하여 API 단계에서 활성 추적을 활성화할 수 있습니다.

이러한 단계에서는 단계에 API를 이미 배포한 것으로 가정합니다.

1. https://console.aws.amazon.com/apigateway에서 API Gateway 콘솔에 로그인합니다.
2. API를 선택하고 기본 탐색 창에서 스테이지를 선택합니다.
3. 스테이지 창에서 스테이지를 선택합니다.
4. 로그 및 추적 섹션에서 편집을 선택합니다.
5. 활성 X-Ray 추적을 활성화하려면 X-Ray 추적에서 X-Ray 추적을 활성화합니다.
6. 변경 사항 저장을 선택합니다.

API 단계에 대해 X-Ray를 활성화한 후에는 X-Ray 관리 콘솔을 사용하여 추적 및 서비스 맵을 볼 수 있습니다.

API Gateway CLI를 사용하여 AWS X-Ray 추적 활성화

스테이지를 생성할 때 AWS CLI를 사용하여 API 스테이지에 대해 활성 X-Ray 추적을 활성화하려면 다음 예제와 같이 create-stage 명령을 호출합니다.

```
aws apigateway create-stage \
    --rest-api-id {rest-api-id} \
    --stage-name {stage-name} \
    --deployment-id {deployment-id} \
    --region {region} \
    --tracing-enabled=true
```

다음은 호출이 제대로 이루어진 경우의 예제 출력입니다.

```
{
    "tracingEnabled": true,
    "stageName": {stage-name},
    "cacheClusterEnabled": false,
    "cacheClusterStatus": "NOT_AVAILABLE",
    "deploymentId": {deployment-id},
    "lastUpdatedDate": 1533849811,
    "createdDate": 1533849811,
    "methodSettings": {}
}
```

스테이지를 생성할 때 AWS CLI를 사용하여 API 스테이지에 대해 활성 X-Ray 추적을 비활성화하려면 다음 예제와 같이 create-stage 명령을 호출합니다.

```
aws apigateway create-stage \
    --rest-api-id {rest-api-id} \
    --stage-name {stage-name} \
    --deployment-id {deployment-id} \
    --region {region} \
    --tracing-enabled=false
```

다음은 호출이 제대로 이루어진 경우의 예제 출력입니다.

```
{
    "tracingEnabled": false,
```

```
    "stageName": {stage-name},
    "cacheClusterEnabled": false,
    "cacheClusterStatus": "NOT_AVAILABLE",
    "deploymentId": {deployment-id},
    "lastUpdatedDate": 1533849811,
    "createdDate": 1533849811,
    "methodSettings": {}
}
```

AWS CLI를 사용하여 이미 배포된 API에 대해 활성 X-Ray 추적을 활성화하려면 다음과 같이 update-stage 명령을 호출합니다.

```
aws apigateway update-stage \
    --rest-api-id {rest-api-id} \
    --stage-name {stage-name} \
    --patch-operations op=replace,path=/tracingEnabled,value=true
```

AWS CLI를 사용하여 이미 배포된 API에 대해 활성 X-Ray 추적을 비활성화하려면 다음 예제와 같이 update-stage 명령을 호출합니다.

```
aws apigateway update-stage \
    --rest-api-id {rest-api-id} \
    --stage-name {stage-name} \
    --region {region} \
    --patch-operations op=replace,path=/tracingEnabled,value=false
```

다음은 호출이 제대로 이루어진 경우의 예제 출력입니다.

```
{
    "tracingEnabled": false,
    "stageName": {stage-name},
    "cacheClusterEnabled": false,
    "cacheClusterStatus": "NOT_AVAILABLE",
    "deploymentId": {deployment-id},
    "lastUpdatedDate": 1533850033,
    "createdDate": 1533849811,
    "methodSettings": {}
}
```

API 단계에 대해 X-Ray를 활성화한 후에는 X-Ray CLI를 사용하여 추적 정보를 가져옵니다. 자세한 내용은 AWS CLI를 통한 AWS X-Ray API 사용 단원을 참조하세요.

API Gateway에서 AWS X-Ray 서비스 맵 및 트레이스 보기 사용

이 단원에는 API Gateway로 AWS X-Ray 서비스 맵 및 트레이스 보기를 사용하는 방법에 대한 자세한 정보가 나와 있습니다.

서비스 맵과 트레이스 보기 및 그 해석 방법에 대한 자세한 정보는 AWS X-Ray콘솔 단원을 참조하십시오.

주제

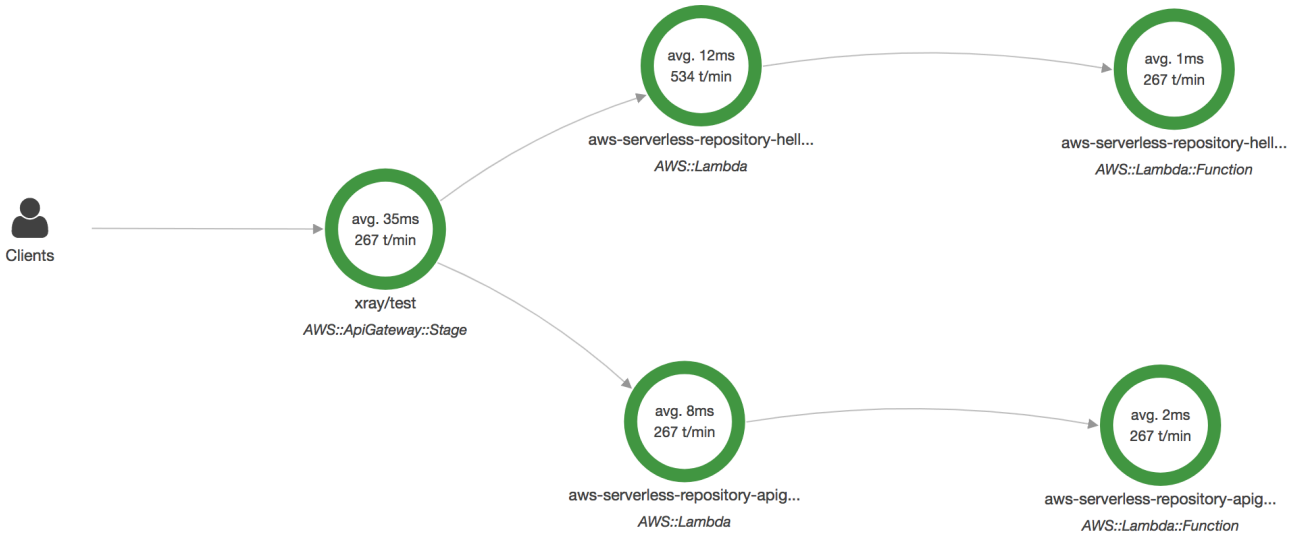
- X-Ray 서비스 맵 예제
- X-Ray 추적 보기 예

X-Ray 서비스 맵 예제

AWS X-Ray 서비스 맵은 API 및 그 다운스트림 서비스에 대한 정보를 보여줍니다. API Gateway에서 API 단계에 대해 X-Ray를 활성화하면 API Gateway 서비스에 사용된 전체 시간에 대한 정보를 포함하여 서비스 맵에 노드가 표시됩니다. 선택된 일정에 대한 API 응답 시간의 응답 상태 및 히스토그램에 대한 자세한 정보를 가져올 수 있습니다. AWS Lambda 및 Amazon DynamoDB와 같은 AWS 서비스와 통합하는 API의 경우, 이러한 서비스와 관련된 성능 지표를 제공하는 추가 노드가 표시됩니다. 각 API 단계에 대한 서비스 맵이 부여됩니다.

다음 예제는 `test`라는 API의 `xray` 단계에 대한 서비스 맵을 보여 줍니다. 이 API에는 Lambda 권한 부여자 함수 및 Lambda 백엔드 함수를 포함한 Lambda 통합이 있습니다. 노드는 API Gateway 서비스, Lambda 서비스 및 두 개의 Lambda 함수를 나타냅니다.

서비스 맵 구조에 대한 자세한 설명은 서비스 맵 보기를 참조하십시오.



서비스 맵에서 API 단계의 트레이스 보기를 확대하여 볼 수 있습니다. 트레이스는 세그먼트 및 하위 세그먼트로 표현된 API와 관련된 세부 정보를 표시합니다. 예를 들어, 위에 표시된 서비스 맵에 대한 추적에는 Lambda 서비스 및 Lambda 함수에 대한 세그먼트가 포함됩니다. 자세한 내용은 AWS X-Ray 추적으로서의 Lambda를 참조하세요.

X-Ray 서비스 맵에서 노드 또는 엣지를 선택하면 X-Ray 콘솔에 지연 시간 분포 히스토그램이 표시됩니다. 지연 시간 히스토그램을 사용하여 서비스에서 요청을 완료하는 데 소요되는 시간을 확인할 수 있습니다. 다음은 이전 서비스 맵에 있는 `xray/test`라는 API Gateway 단계의 히스토그램입니다. 지연 시간 분포 히스토그램에 대한 자세한 설명은 AWS X-Ray 콘솔에서 지연 시간 히스토그램 사용 단원을 참조하십시오.

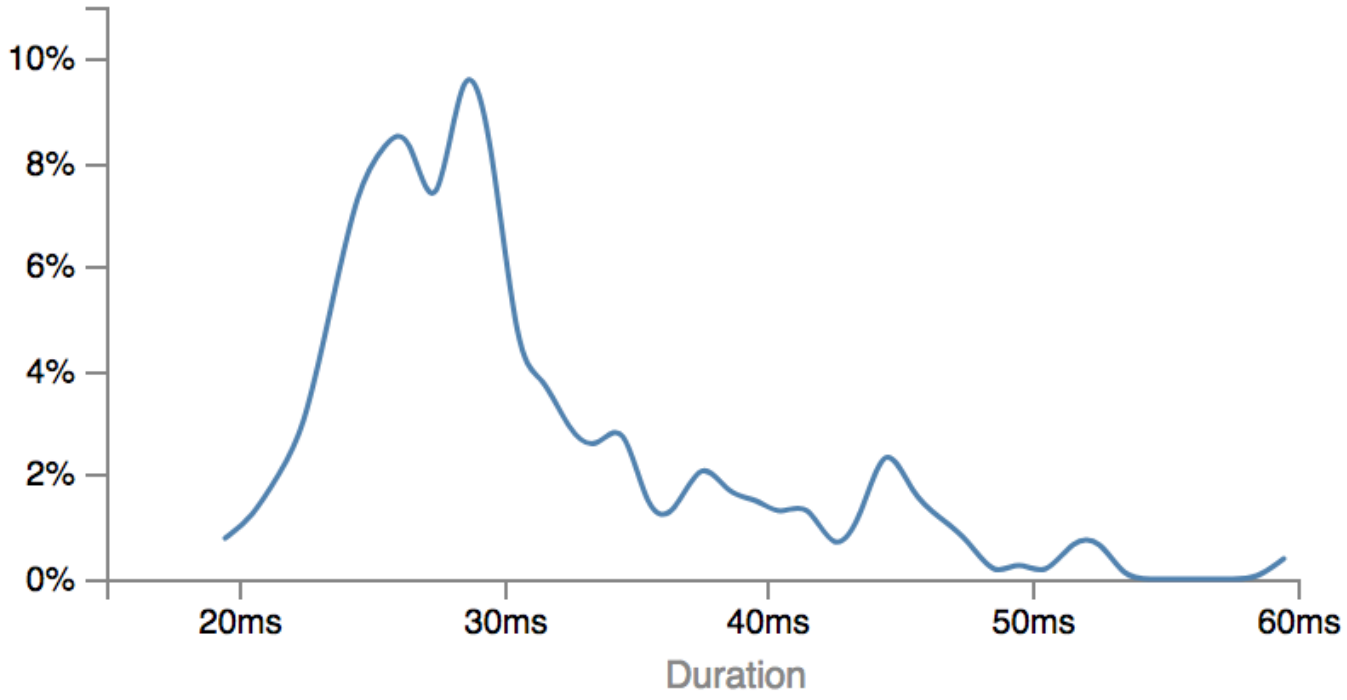
Service details

Name: xray/test

Type: AWS::ApiGateway::Stage

Response distribution

Click and drag to select an area to zoom in on or use as a latency filter when viewing traces.



Response status

Choose response statuses to add to the filter when viewing traces.

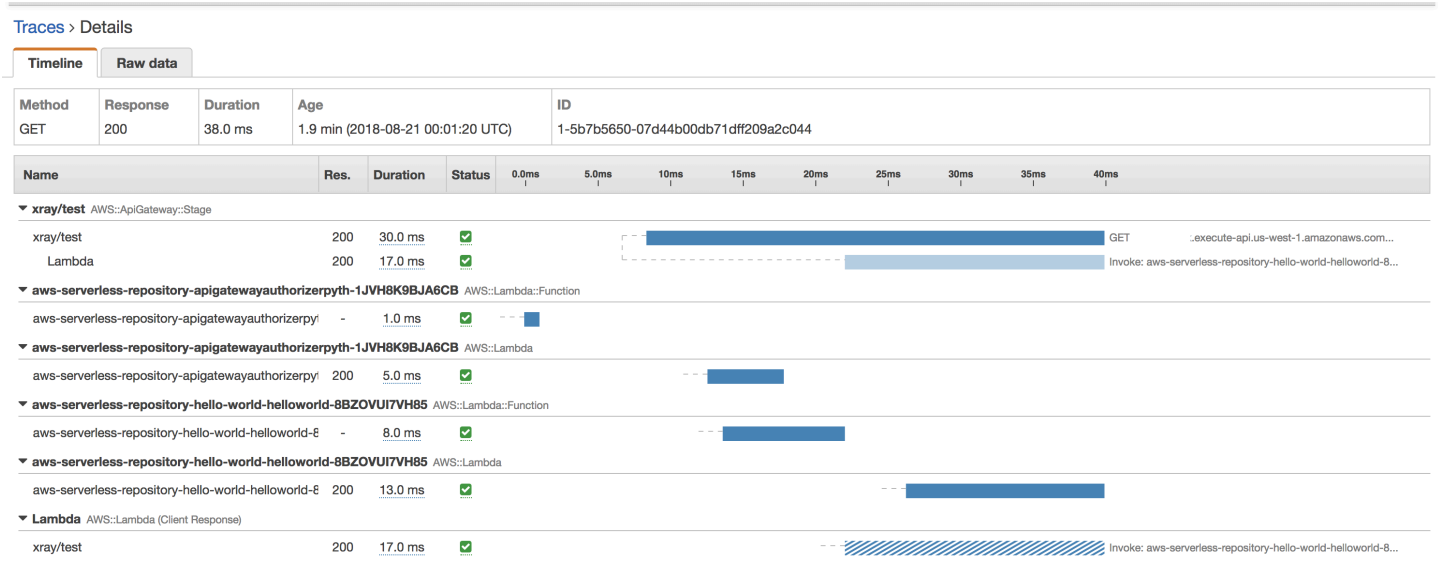
OK: 100% Error: 0%

Fault: 0% Throttle: 0%

X-Ray 추적 보기 예

다음 다이어그램에서는 Lambda 백엔드 함수 및 Lambda 권한 부여자 함수와 함께 위에서 설명한 예제 API에 대해 생성된 추적 보기를 보여 줍니다. 성공적인 API 메서드 요청이 200 응답 코드로 표시됩니다.

추적 보기에 대한 자세한 설명은 추적 보기를 참조하십시오.



API Gateway API에 대한 AWS X-Ray 샘플링 규칙 구성

AWS X-Ray 콘솔 또는 SDK를 사용하여 Amazon API Gateway API에 대한 샘플링 규칙을 구성할 수 있습니다. 샘플링 규칙은 X-Ray가 API에 대해 기록해야 하는 요청을 지정합니다. 샘플링 규칙을 사용자 지정하여 기록할 데이터의 양을 제어하고 코드를 수정하거나 재배포하는 대신 즉석에서 샘플링 동작을 수정할 수 있습니다.

X-Ray 샘플링 규칙을 지정하기 전에 X-Ray 개발자 안내서의 다음 주제를 읽으십시오.

- AWS X-Ray 콘솔에서 샘플링 규칙 구성
- X-Ray API에 샘플링 규칙 사용

주제

- API Gateway API에 대한 X-Ray 샘플링 규칙 옵션 값
- X-Ray 샘플링 규칙 예제

API Gateway API에 대한 X-Ray 샘플링 규칙 옵션 값

다음 X-Ray 샘플링 옵션은 API Gateway와 관련됩니다. 문자열 값은 와일드카드를 사용하여 단일 문자(?) 또는 0개 이상의 문자(*)와 일치시킬 수 있습니다. 리저버 및 속도 설정의 사용 방법에 대한 상세 설명을 포함한 자세한 내용은 AWS X-Ray 콘솔에서 샘플링 규칙 구성 단원을 참조하십시오.

- 규칙 이름(문자열) - 규칙의 고유한 이름입니다.
- 우선 순위(1~9999 사이의 정수) - 샘플링 규칙의 우선 순위입니다. 서비스에서 규칙의 우선 순위를 오름차순으로 평가하며 일치하는 첫 번째 규칙으로 샘플링을 결정합니다.
- 리저버(음수가 아닌 정수) - 고정 속도를 적용하기 전, 초당 계측과 일치하는 요청의 고정 수입니다. 리저버는 서비스에서 직접 사용하지 않지만 규칙을 총체적으로 사용하여 모든 서비스에 적용됩니다.
- 속도(0~100 사이의 숫자) - 리저버가 소진된 후, 계측과 일치하는 요청의 백분율입니다.
- 서비스 이름(문자열) - ***{api-name}/{stage-name}*** 형식으로 표시된 API 단계 이름입니다. 예를 들어, PetStore 샘플 API를 `test`라는 단계로 배포하려는 경우, 샘플링 규칙에서 지정할 서비스 이름 값은 **`pets/test`**일 수 있습니다.
- 서비스 유형(문자열) - API Gateway API의 경우 **`AWS::ApiGateway::Stage`** 또는 **`AWS::ApiGateway::*`**를 지정할 수 있습니다.
- 호스트(문자열) - HTTP 호스트 헤더에 있는 호스트 이름입니다. 이 호스트 이름을 **`*`**으로 설정하여 모든 호스트 이름에 대해 일치시킵니다. 또는 일치시킬 전체 또는 부분적 호스트 이름(예: **`api.example.com`** 또는 **`*.example.com`**)을 지정할 수 있습니다.
- 리소스 ARN(문자열) - API 단계의 ARN입니다(예: **`arn:aws:apigateway:`*`region`*`::/restapis/`*`api-id`*`/stages/`*`stage-name`***).

  단계 이름은 콘솔이나 API Gateway CLI 또는 API에서 가져올 수 있습니다. ARN 형식에 대한 자세한 내용은 Amazon Web Services 일반 참조 단원을 참조하십시오.
- HTTP 메서드(문자열) - 샘플링할 메서드입니다(예: **`GET`**).
- URL 경로(문자열) - 요청의 URL 경로입니다.
- (선택 사항) 속성(키 및 값) - 원본 HTTP 요청의 헤더입니다(예: **`Connection`**, **`Content-Length`** 또는 **`Content-Type`**). 각 속성 값은 최대 32자일 수 있습니다.

X-Ray 샘플링 규칙 예제

샘플링 규칙 예제 #1

이 규칙은 `GET` 단계에서 `testxray` API에 대한 모든 `test` 요청을 샘플링합니다.

- 규칙 이름 - **test-sampling**
- 우선 순위 - **17**
- 리저버 크기 - **10**
- 고정 비율 - **10**
- 서비스 이름 - **testxray/test**
- 서비스 유형 - **AWS::ApiGateway::Stage**
- HTTP 메서드 - **GET**
- 리소스 ARN - *****
- 호스트 - *****

샘플링 규칙 예제 #2

이 규칙은 `testxray` 단계에서 `prod` API에 대한 모든 요청을 샘플링합니다.

- 규칙 이름 - **prod-sampling**
- 우선 순위 - **478**
- 리저버 크기 - **1**
- 고정 비율 - **60**
- 서비스 이름 - **testxray/prod**
- 서비스 유형 - **AWS::ApiGateway::Stage**
- HTTP 메서드 - *****
- 리소스 ARN - *****
- 호스트 - *****
- 속성 - **{}**

Amazon API Gateway API에 대한 AWS X-Ray 추적 이해

이 단원에서는 Amazon API Gateway API에 대한 AWS X-Ray 추적 세그먼트, 하위 세그먼트 및 기타 추적 필드에 대해 설명합니다.

이 단원을 읽기 전에 X-Ray 개발자 안내서의 다음 주제를 검토하십시오.

- AWS X-Ray 콘솔

- AWS X-Ray 세그먼트 문서
- X-Ray 개념

주제

- API Gateway API에 대한 추적 객체의 예
- 트레이스 이해

API Gateway API에 대한 추적 객체의 예

이 단원에서는 API Gateway API에 대한 추적에서 볼 수 있는 몇 가지 객체에 대해 설명합니다.

주석

주석이 세그먼트와 하위 세그먼트에 표시될 수 있습니다. 주석은 트레이스를 필터링하는 샘플링 규칙에서 필터링 표현식으로 사용됩니다. 자세한 내용은 AWS X-Ray 콘솔의 샘플링 규칙 구성 단원을 참조하십시오.

다음은 `annotations` 객체의 예입니다. 여기서 API 단계는 API ID와 API 단계 이름으로 식별됩니다.

```
"annotations": {
    "aws:api_id": "a1b2c3d4e5",
    "aws:api_stage": "dev"
}
```

AWS 리소스 데이터

`aws` 객체는 세그먼트에만 나타납니다. 다음은 기본 샘플링 규칙과 일치하는 aws 객체의 예입니다. 샘플링 규칙에 대한 심층적 설명은 AWS X-Ray 콘솔의 샘플링 규칙 구성 단원을 참조하십시오.

```
"aws": {
    "xray": {
        "sampling_rule_name": "Default"
    },
    "api_gateway": {
        "account_id": "123412341234",
        "rest_api_id": "a1b2c3d4e5",
        "stage": "dev",
        "request_id": "a1b2c3d4-a1b2-a1b2-a1b2-a1b2c3d4e5f6"
    }
```

```
}
```

트레이스 이해

다음은 API Gateway 단계에 대한 추적 세그먼트입니다. 추적 세그먼트를 구성하는 필드에 대한 자세한 설명은 AWS X-Ray 개발자 안내서의 AWS X-Ray 세그먼트 문서 단원을 참조하세요.

```
        {
            "Document": {
                "id": "a1b2c3d4a1b2c3d4",
                "name": "testxray/dev",
                "start_time": 1533928226.229,
                "end_time": 1533928226.614,
                "metadata": {
                    "default": {
                        "extended_request_id": "abcde12345abcde=",
                        "request_id": "a1b2c3d4-a1b2-a1b2-a1b2-a1b2c3d4e5f6"
                    }
                },
                "http": {
                    "request": {
                        "url": "https://example.com/dev?
username=demo&message=hellofromdemo/",
                        "method": "GET",
                        "client_ip": "192.0.2.0",
                        "x_forwarded_for": true
                    },
                    "response": {
                        "status": 200,
                        "content_length": 0
                    }
                },
                "aws": {
                    "xray": {
                        "sampling_rule_name": "Default"
                    },
                    "api_gateway": {
                        "account_id": "123412341234",
                        "rest_api_id": "a1b2c3d4e5",
                        "stage": "dev",
                        "request_id": "a1b2c3d4-a1b2-a1b2-a1b2-a1b2c3d4e5f6"
                    }
                },
```

```
                "annotations": {
                    "aws:api_id": "a1b2c3d4e5",
                    "aws:api_stage": "dev"
                },
                "trace_id": "1-a1b2c3d4-a1b2c3d4a1b2c3d4a1b2c3d4",
                "origin": "AWS::ApiGateway::Stage",
                "resource_arn": "arn:aws:apigateway:us-east-1::/restapis/a1b2c3d4e5/
stages/dev",
                "subsegments": [
                    {
                        "id": "abcdefgh12345678",
                        "name": "Lambda",
                        "start_time": 1533928226.233,
                        "end_time": 1533928226.6130002,
                        "http": {
                            "request": {
                                "url": "https://example.com/2015-03-31/functions/
arn:aws:lambda:us-east-1:123412341234:function:xray123/invocations",
                                "method": "GET"
                            },
                            "response": {
                                "status": 200,
                                "content_length": 62
                            }
                        },
                        "aws": {
                            "function_name": "xray123",
                            "region": "us-east-1",
                            "operation": "Invoke",
                            "resource_names": [
                                "xray123"
                            ]
                        },
                        "namespace": "aws"
                    }
                ]
            },
            "Id": "a1b2c3d4a1b2c3d4"
        }
```

HTTP API 작업

REST API와 HTTP API는 모두 RESTful API 제품입니다. REST API는 HTTP API보다 더 많은 기능을 지원하지만 HTTP API는 최소한의 기능으로 설계되어 더 낮은 가격으로 제공될 수 있습니다. 자세한 내용은 the section called “REST API와 HTTP API 중에서 선택 ” 섹션을 참조하세요.

HTTP API를 사용하여 AWS Lambda 함수 또는 라우팅 가능한 HTTP 엔드포인트에 요청을 전송할 수 있습니다. 예를 들어 백엔드의 Lambda 함수와 통합되는 HTTP API를 생성할 수 있습니다. 클라이언트가 API를 호출하면 API Gateway는 Lambda 함수에 요청을 전송하고 함수의 응답을 클라이언트에 반환합니다.

HTTP API는 OpenID Connect 및 OAuth 2.0 권한 부여를 지원합니다. 여기에서는 CORS(Cross-Origin Resource Sharing) 및 자동 배포가 기본적으로 지원됩니다.

AWS 관리 콘솔, AWS CLI, API, AWS CloudFormation 또는 SDK를 사용하여 HTTP API를 생성할 수 있습니다.

주제

- API Gateway에서 HTTP API 개발
- 고객이 호출할 수 있도록 HTTP API 게시
- HTTP API 보호
- HTTP API 모니터링
- HTTP API 관련 문제 해결

API Gateway에서 HTTP API 개발

이 단원에서는 API Gateway API를 개발하는 동안 필요한 API Gateway 기능에 대해 자세히 설명합니다.

API Gateway API를 개발할 때 API의 여러 특성을 결정합니다. 이러한 특성은 API의 사용 사례에 따라 달라집니다. 예를 들어, 특정 클라이언트만 API를 호출하도록 허용하거나 모든 사용자가 API를 사용할 수 있도록 할 수 있습니다. API 호출로 Lambda 함수를 실행하거나, 데이터베이스 쿼리를 작성하거나, 애플리케이션을 호출할 수 있습니다.

주제

- HTTP API 만들기
- HTTP API에 대한 라우팅 작업

- API Gateway의 HTTP API에 대한 액세스 제어 및 관리
- HTTP API에 대한 통합 구성
- HTTP API에 대한 CORS 구성
- API 요청 및 응답 변환
- HTTP API에 대한 OpenAPI 정의 작업

HTTP API 만들기

함수 API를 만들려면 최소 하나 이상의 경로, 통합, 단계 및 배포가 있어야 합니다.

다음 예제에서는 AWS Lambda 또는 HTTP 통합, 경로, 변경 사항을 자동으로 배포하도록 구성된 기본 단계를 사용하여 API를 생성하는 방법을 보여 줍니다.

이 가이드에서는 API Gateway와 Lambda에 대해 잘 알고 있다고 가정합니다. 자세한 가이드는 시작하기 단원을 참조하세요.

주제

- 를 사용하여 HTTP API 생성AWS Management Console
- AWS CLI를 사용하여 HTTP API 생성

를 사용하여 HTTP API 생성AWS Management Console

1. API Gateway 콘솔을 엽니다.
2. API 생성(Create API)을 선택합니다.
3. HTTP API에서 빌드를 선택합니다.
4. 통합 추가를 선택한 다음, AWS Lambda 함수를 선택하거나 HTTP 엔드포인트를 입력합니다.
5. 이름에 API의 이름을 입력합니다.
6. [Review and create]를 선택합니다.
7. Create를 선택합니다.

이제 API를 호출할 준비가 되었습니다. 브라우저에서 호출 URL을 입력하거나 Curl을 사용하여 API를 테스트할 수 있습니다.

```
curl https://api-id.execute-api.us-east-2.amazonaws.com
```

AWS CLI를 사용하여 HTTP API 생성

빠른 생성을 사용하여 Lambda 또는 HTTP 통합, 기본 catch-all 경로, 변경 사항을 자동으로 배포하도록 구성된 기본 스테이지를 통해 API를 생성할 수 있습니다. 다음 명령은 빠른 생성을 사용하여 백엔드에서 Lambda 함수와 통합되는 API를 생성합니다.

> **Note**
>
> Lambda 통합을 호출하려면 API Gateway에 필요한 권한이 있어야 합니다. 리소스 기반 정책이나 IAM 역할을 사용하여 API Gateway 권한을 부여하여 Lambda 함수를 호출할 수 있습니다. 자세한 내용은 AWS Lambda 개발자 안내서의 AWS Lambda 권한을 참조하세요.

Example

```
aws apigatewayv2 create-api --name my-api --protocol-type HTTP --target
 arn:aws:lambda:us-east-2:123456789012:function:function-name
```

이제 API를 호출할 준비가 되었습니다. 브라우저에서 호출 URL을 입력하거나 Curl을 사용하여 API를 테스트할 수 있습니다.

```
curl https://api-id.execute-api.us-east-2.amazonaws.com
```

HTTP API에 대한 라우팅 작업

수신 API 요청을 백엔드 리소스로 직접 라우팅합니다. 라우팅은 HTTP 메서드와 리소스 경로(예: `GET /pets`) 등 두 부분으로 구성됩니다. 라우팅에 대한 특정 HTTP 메서드를 정의할 수 있습니다. 또는 `ANY` 메서드를 사용하여 리소스에 대해 정의하지 않은 모든 메서드와 일치시킬 수 있습니다. 다른 경로와 일치하지 않는 요청에 대해 catch-all 역할을 하는 `$default` 경로를 생성할 수 있습니다.

> **Note**
>
> API Gateway는 URL 인코딩 요청 파라미터를 백엔드 통합에 전달하기 전에 디코딩합니다.

경로 변수 작업

HTTP API 라우팅에서 경로 변수를 사용할 수 있습니다.

예를 들어, GET /pets/{petID} 라우팅은 클라이언트가 GET에 제출하는 https://*api-id*.execute-api.*us-east-2*.amazonaws.com/pets/6 요청을 포착합니다.

복잡한 경로 변수는 라우팅의 모든 하위 리소스를 포착합니다. 복잡한 경로 변수를 생성하려면 변수 이름에 +를 추가합니다(예: {proxy+}). 복잡한 경로 변수는 리소스 경로의 끝에 있어야 합니다.

쿼리 문자열 파라미터 작업

HTTP API에 대한 요청에 포함된 경우 API Gateway에서는 기본적으로 쿼리 문자열 파라미터를 백엔드 통합으로 보냅니다.

예를 들어, 클라이언트가 요청을 https://*api-id*.execute-api.*us-east-2*.amazonaws.com/pets*?id=4&type=dog*에 보내는 경우 쿼리 문자열 파라미터 ?id=4&type=dog가 통합에 전송됩니다.

$default 라우팅 작업

$default 라우팅은 API의 다른 라우팅과 명시적으로 일치하지 않는 요청을 포착합니다.

$default 라우팅이 요청을 받는 경우 API Gateway에서 전체 요청 경로를 통합에 보냅니다. 예를 들어 $default 라우팅만 있는 API를 생성하여 ANY HTTP 엔드포인트가 있는 https://petstore-demo-endpoint.execute-api.com 메서드에 통합할 수 있습니다. 요청을 https://*api-id*.execute-api.*us-east-2*.amazonaws.com/store/checkout에 보내는 경우 API Gateway에서 요청을 https://petstore-demo-endpoint.execute-api.com/store/checkout에 보냅니다.

HTTP 통합에 대한 자세한 내용은 HTTP API에 대한 HTTP 프록시 통합 작업 단원을 참조하세요.

API 요청 라우팅

클라이언트가 API 요청을 보내는 경우 API Gateway에서 먼저 요청을 라우팅할 스테이지를 결정합니다. 요청이 스테이지와 명시적으로 일치하는 경우 API Gateway는 요청을 해당 스테이지로 보냅니다. 요청과 완전히 일치하는 스테이지가 없는 경우 API Gateway는 요청을 $default 스테이지로 보냅니다. $default 스테이지가 없는 경우 API에서 {"message":"Not Found"}가 반환되며 CloudWatch 로그가 생성되지 않습니다.

스테이지를 선택한 후 API Gateway는 라우팅을 선택합니다. API Gateway는 다음 우선 순위를 사용하여 가장 일치하는 라우팅을 선택합니다.

1. 라우팅 및 메서드에 완전히 일치시킵니다.
2. 복잡한 경로 변수({proxy+})가 있는 라우팅 및 메서드에 일치시킵니다.

3. $default 라우팅

요청과 일치하는 라우팅이 없으면 API Gateway에서 {"message":"Not Found"}를 클라이언트로 반환합니다.

예를 들어 $default 단계가 있는 API와 다음 예제 라우팅을 고려합니다.

1. GET /pets/dog/1
2. GET /pets/dog/{id}
3. GET /pets/{proxy+}
4. ANY /{proxy+}
5. $default

다음 표에는 API Gateway에서 요청을 예제 라우팅으로 라우팅하는 방법이 요약되어 있습니다.

요청	선택한 라우팅	설명
GET https://*api-id*.execute-api.*region*.amazonaws.com/pets/dog/1	GET /pets/dog/1	요청이 이 정적 라우팅과 완전히 일치합니다.
GET https://*api-id*.execute-api.*region*.amazonaws.com/pets/dog/2	GET /pets/dog/{id}	요청이 이 라우팅과 완전히 일치합니다.
GET https://*api-id*.execute-api.*region*.amazonaws.com/pets/cat/1	GET /pets/{proxy+}	요청이 라우팅과 완전히 일치하지 않습니다. GET 메서드와 복잡한 경로 변수가 있는 라우팅은 이 요청을 포착합니다.
POST https://*api-id*.execute-api.*region*.amazonaws.com/test/5	ANY /{proxy+}	ANY 메서드는 라우팅에 대해 정의하지 않은 모든 메서드와 일치합니다. 복잡한 경로 변수가 있는 라우팅은 $default

요청	선택한 라우팅	설명
		라우팅보다 우선 순위가 높습니다.

API Gateway의 HTTP API에 대한 액세스 제어 및 관리

API Gateway는 HTTP API 액세스 제어 및 관리에 다중 메커니즘을 지원합니다.

- Lambda 권한 부여자는 Lambda 함수를 사용하여 API에 대한 액세스를 제어합니다. 자세한 내용은 HTTP API에 대한 AWS Lambda 권한 부여자와 협력하기 단원을 참조하세요.
- JWT 권한 부여자는 JSON 웹 토큰을 사용하여 API에 대한 액세스를 제어합니다. 자세한 내용은 JWT 권한 부여자를 사용하여 HTTP API에 대한 액세스 제어 단원을 참조하세요.
- 표준 AWS IAM 역할 및 정책은 유연하고 강력한 액세스 제어를 제공합니다. IAM 역할 및 정책을 사용하여 API를 생성하고 관리할 수 있는 사용자와 API를 호출할 수 있는 사용자를 제어할 수 있습니다. 자세한 내용은 섹션을 참조하세요IAM 권한 부여 사용

HTTP API에 대한 AWS Lambda 권한 부여자와 협력하기

Lambda 권한 부여자를 사용하여 HTTP API에 대한 액세스를 제어하는 Lambda 함수를 사용합니다. 그런 다음 클라이언트가 API를 호출하면 API Gateway에서 Lambda 함수를 호출합니다. API Gateway는 Lambda 함수의 응답을 사용하여 클라이언트가 API에 액세스할 수 있는지 여부를 결정합니다.

페이로드 형식 버전

권한 부여자 페이로드 형식 버전은 API Gateway에서 Lambda 권한 부여자로 전송하는 데이터의 형식과 API Gateway에서 Lambda의 응답을 해석하는 방법을 지정합니다. 페이로드 형식 버전을 지정하지 않는 경우 는 기본적으로 최신 버전을 AWS Management Console 사용합니다. AWS CloudFormation, 또는 SDK를 사용하여 Lambda 권한 AWS CLI부여자를 생성하는 경우 를 지정해야 합니다. `authorizerPayloadFormatVersion` 지원되는 값은 `1.0` 및 `2.0`입니다.

REST API와의 호환성이 필요한 경우 `1.0` 버전을 사용합니다.

다음 예제에서는 각 페이로드 형식 버전의 구조를 보여 줍니다.

2.0

```
{
  "version": "2.0",
```

```
  "type": "REQUEST",
  "routeArn": "arn:aws:execute-api:us-east-1:123456789012:abcdef123/test/GET/
request",
  "identitySource": ["user1", "123"],
  "routeKey": "$default",
  "rawPath": "/my/path",
  "rawQueryString": "parameter1=value1&parameter1=value2&parameter2=value",
  "cookies": ["cookie1", "cookie2"],
  "headers": {
    "header1": "value1",
    "header2": "value2"
  },
  "queryStringParameters": {
    "parameter1": "value1,value2",
    "parameter2": "value"
  },
  "requestContext": {
    "accountId": "123456789012",
    "apiId": "api-id",
    "authentication": {
      "clientCert": {
        "clientCertPem": "CERT_CONTENT",
        "subjectDN": "www.example.com",
        "issuerDN": "Example issuer",
        "serialNumber": "1",
        "validity": {
          "notBefore": "May 28 12:30:02 2019 GMT",
          "notAfter": "Aug  5 09:36:04 2021 GMT"
        }
      }
    },
    "domainName": "id.execute-api.us-east-1.amazonaws.com",
    "domainPrefix": "id",
    "http": {
      "method": "POST",
      "path": "/my/path",
      "protocol": "HTTP/1.1",
      "sourceIp": "IP",
      "userAgent": "agent"
    },
    "requestId": "id",
    "routeKey": "$default",
    "stage": "$default",
    "time": "12/Mar/2020:19:03:58 +0000",
```

```
    "timeEpoch": 1583348638390
  },
  "pathParameters": { "parameter1": "value1" },
  "stageVariables": { "stageVariable1": "value1", "stageVariable2": "value2" }
}
```

1.0

```
{
  "version": "1.0",
  "type": "REQUEST",
  "methodArn": "arn:aws:execute-api:us-east-1:123456789012:abcdef123/test/GET/
request",
  "identitySource": "user1,123",
  "authorizationToken": "user1,123",
  "resource": "/request",
  "path": "/request",
  "httpMethod": "GET",
  "headers": {
    "X-AMZ-Date": "20170718T062915Z",
    "Accept": "*/*",
    "HeaderAuth1": "headerValue1",
    "CloudFront-Viewer-Country": "US",
    "CloudFront-Forwarded-Proto": "https",
    "CloudFront-Is-Tablet-Viewer": "false",
    "CloudFront-Is-Mobile-Viewer": "false",
    "User-Agent": "..."
  },
  "queryStringParameters": {
    "QueryString1": "queryValue1"
  },
  "pathParameters": {},
  "stageVariables": {
    "StageVar1": "stageValue1"
  },
  "requestContext": {
    "path": "/request",
    "accountId": "123456789012",
    "resourceId": "05c7jb",
    "stage": "test",
    "requestId": "...",
    "identity": {
      "apiKey": "...",
```

```
      "sourceIp": "...",
      "clientCert": {
        "clientCertPem": "CERT_CONTENT",
        "subjectDN": "www.example.com",
        "issuerDN": "Example issuer",
        "serialNumber": "a1:a1:a1:a1:a1:a1:a1:a1:a1:a1:a1:a1:a1:a1:a1:a1",
        "validity": {
          "notBefore": "May 28 12:30:02 2019 GMT",
          "notAfter": "Aug  5 09:36:04 2021 GMT"
        }
      }
    },
    "resourcePath": "/request",
    "httpMethod": "GET",
    "apiId": "abcdef123"
  }
}
```

Lambda 권한 부여자 응답 형식

페이로드 형식 버전은 Lambda 함수에서 반환되어야 하는 응답 구조도 결정합니다.

형식 1.0에 대한 Lambda 함수 응답

`1.0` 형식 버전을 선택하는 경우 Lambda 권한 부여자는 API 라우팅에 대한 액세스를 허용하거나 거부하는 IAM 정책을 반환해야 합니다. 정책에서 표준 IAM 정책 구문을 사용할 수 있습니다. IAM 정책에 대한 예시는 the section called “ API 호출을 위한 액세스 제어” 단원을 참조하세요. `$context.authorizer.`*`property`*를 사용하여 컨텍스트 속성을 Lambda 통합 또는 액세스 로그에 전달할 수 있습니다. `context` 객체는 선택 사항이며 `claims`는 예약된 자리 표시자이므로 컨텍스트 객체로 사용할 수 없습니다. 자세한 내용은 the section called “로깅 변수” 단원을 참조하세요.

Example

```
{
  "principalId": "abcdef", // The principal user identification associated with the
 token sent by the client.
  "policyDocument": {
    "Version": "2012-10-17",
    "Statement": [
      {
        "Action": "execute-api:Invoke",
        "Effect": "Allow|Deny",
```

```
        "Resource": "arn:aws:execute-api:{regionId}:{accountId}:{apiId}/{stage}/
{httpVerb}/[{resource}/[{child-resources}]]"
      }
    ]
  },
  "context": {
    "exampleKey": "exampleValue"
  }
}
```

형식 2.0에 대한 Lambda 함수 응답

`2.0` 형식 버전을 선택하는 경우 Lambda 함수에서 부울 값 또는 표준 IAM 정책 구문을 사용하는 IAM 정책을 반환할 수 있습니다. 부울 값을 반환하려면 권한 부여자에 대한 간단한 응답을 활성화합니다. 다음 예제에서는 Lambda 함수에서 반환하도록 코딩해야 하는 형식을 보여줍니다. `context` 객체는 선택 사항입니다. `$context.authorizer.`*`property`*를 사용하여 컨텍스트 속성을 Lambda 통합 또는 액세스 로그에 전달할 수 있습니다. 자세한 내용은 the section called “로깅 변수” 단원을 참조하세요.

Simple response

```
{
  "isAuthorized": true/false,
  "context": {
    "exampleKey": "exampleValue"
  }
}
```

IAM policy

```
{
  "principalId": "abcdef", // The principal user identification associated with the
 token sent by the client.
  "policyDocument": {
    "Version": "2012-10-17",
    "Statement": [
      {
        "Action": "execute-api:Invoke",
        "Effect": "Allow|Deny",
        "Resource": "arn:aws:execute-api:{regionId}:{accountId}:{apiId}/{stage}/
{httpVerb}/[{resource}/[{child-resources}]]"
      }
```

```
    ]
  },
  "context": {
    "exampleKey": "exampleValue"
  }
}
```

Lambda 권한 부여자 함수의 예

다음 예제 Node.js Lamdba 함수는 `2.0` 페이로드 형식 버전의 경우 Lambda 함수에서 반환해야 하는 응답 형식을 보여줍니다.

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
```

자격 증명 원본

선택적으로 Lambda 권한 부여자에 대한 자격 증명 원본을 지정할 수 있습니다. 자격 증명 원본은 요청에 권한을 부여하는 데 필요한 데이터의 위치를 지정합니다. 예를 들어 헤더 또는 쿼리 문자열 값을 자격 증명 원본으로 지정할 수 있습니다. 자격 증명 원본을 지정하는 경우 클라이언트에서 해당 소스를 요청에 포함해야 합니다. 클라이언트의 요청에 자격 증명 원본이 포함되어 있지 않은 경우 API Gateway는 Lambda 권한 부여자를 호출하지 않으며 클라이언트가 `401` 오류를 수신합니다. 지원되는 자격 증명 원본은 다음과 같습니다.

선택 표현식

유형	예	참고
헤더 값	$request.header.*name*	헤더 이름은 대/소문자를 구분하지 않습니다.
쿼리 문자열 값	$request.querystring.*name*	쿼리 문자열 이름은 대/소문자를 구분합니다.
컨텍스트 변수	$context.*variableName*	지원되는 컨텍스트 변수의 값입니다.
단계 변수	$stageVariables.*variableName*	스테이지 변수의 값입니다.

권한 부여자 응답 캐싱

를 지정하여 Lambda 권한 부여자에 대한 캐싱을 활성화할 수 있습니다. authorizerResultTtlInSeconds 권한 부여자에 대해 캐싱이 활성화되면 API Gateway에서 권한 부여자의 자격 증명 원본을 캐시 키로 사용합니다. 클라이언트가 구성된 TTL 내의 자격 증명 원본에서 동일한 파라미터를 지정하는 경우 API Gateway에서 Lambda 함수를 호출하는 대신 캐싱된 권한 부여자 결과를 사용합니다.

캐싱을 활성화하려면 권한 부여자에게 자격 증명 원본이 하나 이상 있어야 합니다.

권한 부여자에 대해 간단한 응답을 활성화하면 권한 부여자의 응답이 캐싱된 자격 증명 원본 값과 일치하는 모든 API 요청을 완전히 허용하거나 거부합니다. 보다 세분화된 권한이 필요한 경우 간단한 응답을 비활성화하고 IAM 정책을 반환합니다.

기본적으로 API Gateway에서는 권한 부여자를 사용하는 API의 모든 라우팅에 캐싱된 권한 부여자 응답을 사용합니다. 라우팅별로 응답을 캐싱하려면 권한 부여자의 자격 증명 원본에 `$context.routeKey`를 추가합니다.

Lambda 권한 부여자 만들기

Lambda 권한 부여자를 만들 때 API Gateway에 사용할 Lambda 함수를 지정합니다. 함수의 리소스 정책 또는 IAM 역할을 사용하여 Lambda 함수를 호출할 수 있는 권한을 API Gateway에 부여해야 합니다. 이 예제에서는 함수에 대한 리소스 정책을 업데이트하여 Lambda 함수를 호출할 수 있는 권한을 API Gateway에 부여합니다.

```
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
```

다음 명령은 Lambda 함수를 호출할 수 있는 권한을 API Gateway에 부여합니다. API Gateway에 함수를 호출할 수 있는 권한이 없으면 클라이언트가 `500 Internal Server Error`를 수신합니다.

```
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"
```

권한 부여자를 생성하고 이를 호출할 수 있는 권한을 API Gateway에 부여한 후에는 권한 부여자를 사용하도록 라우팅을 업데이트합니다.

```
aws apigatewayv2 update-route \
    --api-id abcdef123 \
    --route-id acd123 \
    --authorization-type CUSTOM \
    --authorizer-id def123
```

Lambda 권한 부여자 문제 해결

API Gateway에서 Lambda 권한 부여자를 호출할 수 없거나 Lambda 권한 부여자가 잘못된 형식으로 응답을 반환하는 경우 클라이언트가 `500 Internal Server Error`를 수신합니다.

오류를 해결하려면 API 스테이지에 대한 액세스 로깅을 활성화합니다. 로그 형식에 `$context.authorizer.error` 로깅 변수를 포함합니다.

로그에 API Gateway가 함수를 호출할 권한이 없다고 표시되면 함수의 리소스 정책을 업데이트하거나 권한 부여자를 호출할 수 있는 권한을 API Gateway에 부여하는 IAM 역할을 제공합니다.

로그에 Lambda 함수가 잘못된 응답을 반환한다고 표시되면 Lambda 함수가 필요한 형식으로 응답을 반환하는지 확인합니다.

JWT 권한 부여자를 사용하여 HTTP API에 대한 액세스 제어

JWT(JSON Web Token)를 OIDC(OpenID Connect) 및 OAuth 2.0 프레임워크의 일부로 사용하여 API에 대한 클라이언트 액세스를 제한할 수 있습니다.

API의 경로에 대해 JWT 권한 부여자를 구성하면 API Gateway는 클라이언트가 API 요청과 함께 제출하는 JWT를 검증합니다. API Gateway는 토큰 검증 및 선택적으로 토큰의 범위를 기반으로 요청을 허용하거나 거부합니다. 경로의 범위를 구성하는 경우 토큰에 경로의 범위 중 하나 이상이 포함되어야 합니다.

API의 각 경로에 대해 고유한 권한 부여자를 구성하거나, 여러 경로에 대해 동일한 권한 부여자를 사용할 수 있습니다.

> **Note**
>
> OpenID Connect ID 토큰과 같은 기타 유형의 JWT와 JWT 액세스 토큰을 구별하는 표준 메커니즘은 없습니다. API 인증에 ID 토큰이 필요하지 않은 경우 인증 범위가 필요하도록 경로를 구성하는 것이 좋습니다. 자격 증명 공급자가 JWT 액세스 토큰을 발급할 때만 사용하는 발급자 또는 대상 그룹이 필요하도록 JWT 권한 부여자를 구성할 수도 있습니다.

JWT 권한 부여자를 통한 API 요청 승인

API Gateway는 다음과 같은 일반 워크플로우를 사용하여 JWT 권한 부여자를 사용하도록 구성된 라우팅에 대한 요청을 승인합니다.

1. 토큰에 대한 `identitySource`를 확인합니다. `identitySource`에는 토큰만 포함하거나 `Bearer` 접두사가 붙은 토큰만 포함될 수 있습니다.
2. 토큰을 디코딩합니다.
3. 발행자의 `jwks_uri`에서 가져온 퍼블릭 키를 사용하여 토큰의 알고리즘과 서명을 확인합니다. 현재 RSA 기반 알고리즘만 지원됩니다. API Gateway는 퍼블릭 키를 2시간 동안 캐시할 수 있습니다. 가장 좋은 방법은 키를 교체할 때 이전 키와 새 키가 모두 유효한 유예 기간을 두는 것입니다.
4. 클레임을 검증합니다. API Gateway는 다음 토큰 클레임을 평가합니다.
   - `kid` – 토큰에는 토큰에 서명한 `jwks_uri`의 키와 일치하는 헤더 클레임이 있어야 합니다.
   - `iss` – 권한 부여자에 대해 구성된 `issuer`와 일치해야 합니다.
   - `aud` 또는 `client_id` – 권한 부여자에 대해 구성된 `audience` 항목 중 하나와 일치해야 합니다. API `aud` Gateway는 존재하지 않는 `client_id` 경우에만 유효성을 검사합니다. `aud`과 `client_id` 가 모두 있는 경우 API Gateway는 평가합니다`aud`.
   - `exp` – 현재 시간(UTC 기준) 이후여야 합니다.
   - `nbf` – 현재 시간(UTC 기준) 이전이어야 합니다.
   - `iat` – 현재 시간(UTC 기준) 이전이어야 합니다.
   - `scope` 또는 `scp` – 토큰에 라우팅의 `authorizationScopes` 범위 중 하나 이상이 포함되어야 합니다.

이러한 단계 중 하나가 실패하면 API Gateway는 API 요청을 거부합니다.

JWT를 검증한 후 API Gateway는 토큰의 클레임을 API 경로의 통합에 전달합니다. Lambda 함수와 같은 백엔드 리소스는 JWT 클레임에 액세스할 수 있습니다. 예를 들어, JWT에 자격 증명 클레임

emailID가 포함된 경우 $event.requestContext.authorizer.jwt.claims.emailID의 Lambda 통합에 해당 클레임을 사용할 수 있습니다. API Gateway가 Lambda 통합으로 보내는 페이로드에 대한 자세한 내용은 the section called “AWS Lambda 통합” 단원을 참조하세요.

JWT 권한 부여자 생성

JWT 권한 부여자를 생성하기 전에 클라이언트 애플리케이션을 자격 증명 공급자에 등록해야 합니다. 또한 HTTP API를 생성해야 합니다. HTTP API 생성 예제는 HTTP API 만들기 단원을 참조하세요.

콘솔을 사용하여 JWT 권한 부여자를 생성합니다.

다음 단계는 콘솔을 사용하여 JWT 권한 부여자를 만드는 방법을 보여줍니다.

콘솔을 사용하여 JWT 권한 부여자를 만들려면

1. https://console.aws.amazon.com/apigateway에서 API Gateway 콘솔에 로그인합니다.
2. HTTP API를 선택합니다.
3. 기본 탐색 창에서 [권한 부여] 를 선택합니다.
4. 권한 부여자 관리 탭을 선택합니다.
5. 생성을 선택하세요.
6. 권한 부여자 유형으로는 JWT를 선택합니다.
7. JWT 권한 부여자를 구성하고 토큰 소스를 정의하는 ID 소스를 지정합니다.
8. 생성을 선택하세요.

를 사용하여 JWT 권한 부여자를 생성합니다. AWS CLI

다음 AWS CLI 명령은 JWT 권한 부여자를 만듭니다. jwt-configuration의 경우 ID 공급자의 Audience 및 Issuer를 지정합니다. Amazon Cognito를 자격 증명 공급자로 사용하는 경우에는 그렇습니다. IssuerUrl https://cognito-idp.*us-east-2*.amazonaws.com/*userPoolID*

```
aws apigatewayv2 create-authorizer \
    --name authorizer-name \
    --api-id api-id \
    --authorizer-type JWT \
    --identity-source '$request.header.Authorization' \
    --jwt-configuration Audience=audience,Issuer=IssuerUrl
```

다음을 사용하여 JWT 권한 부여자를 생성합니다. AWS CloudFormation

다음 AWS CloudFormation 템플릿은 Amazon Cognito를 자격 증명 공급자로 사용하는 JWT 권한 부여자를 사용하여 HTTP API를 생성합니다.

AWS CloudFormation 템플릿의 출력은 Amazon Cognito 호스팅 UI의 URL로, 클라이언트가 가입하고 로그인하여 JWT를 받을 수 있습니다. 클라이언트가 로그인하면 URL에 액세스 토큰이 있는 HTTP API로 클라이언트가 리디렉션됩니다. 액세스 토큰으로 API를 호출하려면 토큰을 쿼리 문자열 파라미터로 사용하도록 URL의 ? 를 a로 변경하십시오. #

예제 템플릿 AWS CloudFormation

```
AWSTemplateFormatVersion: '2010-09-09'
Description: |
  Example HTTP API with a JWT authorizer. This template includes an Amazon Cognito user
 pool as the issuer for the JWT authorizer
  and an Amazon Cognito app client as the audience for the authorizer. The outputs
 include a URL for an Amazon Cognito hosted UI where clients can
  sign up and sign in to receive a JWT. After a client signs in, the client is
 redirected to your HTTP API with an access token
  in the URL. To invoke the API with the access token, change the '#' in the URL to a
 '?' to use the token as a query string parameter.

Resources:
  MyAPI:
    Type: AWS::ApiGatewayV2::Api
    Properties:
      Description: Example HTTP API
      Name: api-with-auth
      ProtocolType: HTTP
      Target: !GetAtt MyLambdaFunction.Arn
  DefaultRouteOverrides:
    Type: AWS::ApiGatewayV2::ApiGatewayManagedOverrides
    Properties:
      ApiId: !Ref MyAPI
      Route:
        AuthorizationType: JWT
        AuthorizerId: !Ref JWTAuthorizer
  JWTAuthorizer:
    Type: AWS::ApiGatewayV2::Authorizer
    Properties:
      ApiId: !Ref MyAPI
      AuthorizerType: JWT
```

```
      IdentitySource:
        - '$request.querystring.access_token'
      JwtConfiguration:
        Audience:
        - !Ref AppClient
        Issuer: !Sub https://cognito-idp.${AWS::Region}.amazonaws.com/${UserPool}
      Name: test-jwt-authorizer
  MyLambdaFunction:
    Type: AWS::Lambda::Function
    Properties:
      Runtime: nodejs18.x
      Role: !GetAtt FunctionExecutionRole.Arn
      Handler: index.handler
      Code:
        ZipFile: |
          exports.handler = async (event) => {
              const response = {
                  statusCode: 200,
                  body: JSON.stringify('Hello from the ' + event.routeKey + ' route!'),
              };
              return response;
          };
  APIInvokeLambdaPermission:
    Type: AWS::Lambda::Permission
    Properties:
      FunctionName: !Ref MyLambdaFunction
      Action: lambda:InvokeFunction
      Principal: apigateway.amazonaws.com
      SourceArn: !Sub arn:${AWS::Partition}:execute-api:${AWS::Region}:
${AWS::AccountId}:${MyAPI}/$default/$default
  FunctionExecutionRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Version: '2012-10-17'
        Statement:
          - Effect: Allow
            Principal:
              Service:
              - lambda.amazonaws.com
            Action:
              - 'sts:AssumeRole'
      ManagedPolicyArns:
        - arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole
```

```
  UserPool:
    Type: AWS::Cognito::UserPool
    Properties:
      UserPoolName: http-api-user-pool
      AutoVerifiedAttributes:
        - email
      Schema:
        - Name: name
          AttributeDataType: String
          Mutable: true
          Required: true
        - Name: email
          AttributeDataType: String
          Mutable: false
          Required: true
  AppClient:
    Type: AWS::Cognito::UserPoolClient
    Properties:
      AllowedOAuthFlows:
        - implicit
      AllowedOAuthScopes:
        - aws.cognito.signin.user.admin
        - email
        - openid
        - profile
      AllowedOAuthFlowsUserPoolClient: true
      ClientName: api-app-client
      CallbackURLs:
        - !Sub https://${MyAPI}.execute-api.${AWS::Region}.amazonaws.com
      ExplicitAuthFlows:
        - ALLOW_USER_PASSWORD_AUTH
        - ALLOW_REFRESH_TOKEN_AUTH
      UserPoolId: !Ref UserPool
      SupportedIdentityProviders:
        - COGNITO
  HostedUI:
    Type: AWS::Cognito::UserPoolDomain
    Properties:
      Domain: !Join
        - '-'
        - - !Ref MyAPI
          - !Ref AppClient
      UserPoolId: !Ref UserPool
Outputs:
```

```
  SignupURL:
    Value: !Sub https://${HostedUI}.auth.${AWS::Region}.amazoncognito.com/login?
client_id=${AppClient}&response_type=token&scope=email+profile&redirect_uri=https://
${MyAPI}.execute-api.${AWS::Region}.amazonaws.com
```

JWT 권한 부여자를 사용하도록 경로 업데이트

콘솔 AWS CLI, 또는 AWS SDK를 사용하여 JWT 권한 부여자를 사용하도록 경로를 업데이트할 수 있습니다.

콘솔을 사용하여 JWT 권한 부여자를 사용하도록 경로를 업데이트하십시오.

다음 단계는 콘솔을 사용하여 JWT 권한 부여자를 사용하도록 경로를 업데이트하는 방법을 보여줍니다.

콘솔을 사용하여 JWT 권한 부여자를 만들려면

1. https://console.aws.amazon.com/apigateway에서 API Gateway 콘솔에 로그인합니다.
2. HTTP API를 선택합니다.
3. 기본 탐색 창에서 [권한 부여] 를 선택합니다.
4. 방법을 선택한 다음 드롭다운 메뉴에서 승인자를 선택하고 승인자 연결을 선택합니다.

를 사용하여 JWT 권한 부여자를 사용하도록 경로를 업데이트하십시오. AWS CLI

다음 명령은 를 사용하여 JWT 권한 부여자를 사용하도록 경로를 업데이트합니다. AWS CLI

```
aws apigatewayv2 update-route \
   --api-id api-id  \
   --route-id route-id  \
   --authorization-type JWT \
   --authorizer-id authorizer-id \
   --authorization-scopes user.email
```

IAM 권한 부여 사용

HTTP API 라우팅에 대해 IAM 권한 부여를 활성화할 수 있습니다. IAM 권한 부여가 활성화된 경우 클라이언트는 서명 버전 4를 사용하여 AWS 자격 증명으로 요청에 서명해야 합니다. API Gateway는 클라이언트가 라우팅에 대한 `execute-api` 권한을 가진 경우에만 API 라우팅을 호출합니다.

HTTP API에 대한 IAM 권한 부여는 REST API에 대한 권한 부여와 유사합니다.

> **Note**
>
> 리소스 정책은 현재 HTTP API에 대해 지원되지 않습니다.

클라이언트에 API 호출 권한을 부여하는 IAM 정책의 예는 the section called “ API 호출을 위한 액세스 제어” 단원을 참조하세요.

라우팅에 대한 IAM 권한 부여 활성화

다음 AWS CLI 명령은 HTTP API 경로에 대한 IAM 권한 부여를 활성화합니다.

```
aws apigatewayv2 update-route \
    --api-id abc123 \
    --route-id abcdef \
    --authorization-type AWS_IAM
```

HTTP API에 대한 통합 구성

통합은 라우팅을 백엔드 리소스에 연결합니다. HTTP API는 Lambda 프록시, AWS 서비스 및 HTTP 프록시 통합을 지원합니다. 예를 들어, 가입 고객을 처리하는 Lambda 함수와 통합하도록 API의 POST 경로에 대한 `/signup` 요청을 구성할 수 있습니다.

주제

- HTTP API에 대한 AWS Lambda 프록시 통합 작업
- HTTP API에 대한 HTTP 프록시 통합 작업
- HTTP API에 대한 AWS 서비스 통합 작업
- HTTP API에 대한 프라이빗 통합 작업

HTTP API에 대한 AWS Lambda 프록시 통합 작업

Lambda 프록시 통합을 사용하면 API 라우팅을 Lambda 함수와 통합할 수 있습니다. 클라이언트가 API를 호출하면 API Gateway는 Lambda 함수에 요청을 전송하고 함수의 응답을 클라이언트에 반환합니다. HTTP API 생성 예제는 HTTP API 만들기 단원을 참조하세요.

페이로드 형식 버전

페이로드 형식 버전은 API Gateway에서 Lambda 통합으로 전송하는 데이터의 형식과 API Gateway에서 Lambda의 응답을 해석하는 방법을 지정합니다. 페이로드 형식 버전을 지정하지 않으면 AWS

Management Console에서는 기본적으로 최신 버전을 사용합니다. AWS CLI, AWS CloudFormation 또는 SDK를 사용하여 Lambda 통합을 생성하는 경우 `payloadFormatVersion`을 지정해야 합니다. 지원되는 값은 `1.0` 및 `2.0`입니다.

다음 예제에서는 각 페이로드 형식 버전의 구조를 보여 줍니다.

> **Note**
>
> 헤더 이름은 소문자입니다.
> 형식 `2.0`에는 `multiValueHeaders` 또는 `multiValueQueryStringParameters` 필드가 없습니다. 중복 헤더는 쉼표로 결합되어 `headers` 필드에 포함됩니다. 중복 쿼리 문자열은 쉼표로 결합되어 `queryStringParameters` 필드에 포함됩니다.
> 형식 `2.0`에는 새 `cookies` 필드가 포함됩니다. 요청의 모든 쿠키 헤더는 쉼표로 결합되어 `cookies` 필드에 추가됩니다. 클라이언트에 대한 응답에서 각 쿠키는 `set-cookie` 헤더가 됩니다.

2.0

```
{
  "version": "2.0",
  "routeKey": "$default",
  "rawPath": "/my/path",
  "rawQueryString": "parameter1=value1&parameter1=value2&parameter2=value",
  "cookies": [
    "cookie1",
    "cookie2"
  ],
  "headers": {
    "header1": "value1",
    "header2": "value1,value2"
  },
  "queryStringParameters": {
    "parameter1": "value1,value2",
    "parameter2": "value"
  },
  "requestContext": {
    "accountId": "123456789012",
    "apiId": "api-id",
    "authentication": {
      "clientCert": {
```

```
        "clientCertPem": "CERT_CONTENT",
        "subjectDN": "www.example.com",
        "issuerDN": "Example issuer",
        "serialNumber": "a1:a1:a1:a1:a1:a1:a1:a1:a1:a1:a1:a1:a1:a1:a1:a1",
        "validity": {
          "notBefore": "May 28 12:30:02 2019 GMT",
          "notAfter": "Aug  5 09:36:04 2021 GMT"
        }
      }
    },
    "authorizer": {
      "jwt": {
        "claims": {
          "claim1": "value1",
          "claim2": "value2"
        },
        "scopes": [
          "scope1",
          "scope2"
        ]
      }
    },
    "domainName": "id.execute-api.us-east-1.amazonaws.com",
    "domainPrefix": "id",
    "http": {
      "method": "POST",
      "path": "/my/path",
      "protocol": "HTTP/1.1",
      "sourceIp": "192.0.2.1",
      "userAgent": "agent"
    },
    "requestId": "id",
    "routeKey": "$default",
    "stage": "$default",
    "time": "12/Mar/2020:19:03:58 +0000",
    "timeEpoch": 1583348638390
  },
  "body": "Hello from Lambda",
  "pathParameters": {
    "parameter1": "value1"
  },
  "isBase64Encoded": false,
  "stageVariables": {
    "stageVariable1": "value1",
```

```
    "stageVariable2": "value2"
  }
}
```

1.0

```
{
  "version": "1.0",
  "resource": "/my/path",
  "path": "/my/path",
  "httpMethod": "GET",
  "headers": {
    "header1": "value1",
    "header2": "value2"
  },
  "multiValueHeaders": {
    "header1": [
      "value1"
    ],
    "header2": [
      "value1",
      "value2"
    ]
  },
  "queryStringParameters": {
    "parameter1": "value1",
    "parameter2": "value"
  },
  "multiValueQueryStringParameters": {
    "parameter1": [
      "value1",
      "value2"
    ],
    "parameter2": [
      "value"
    ]
  },
  "requestContext": {
    "accountId": "123456789012",
    "apiId": "id",
    "authorizer": {
      "claims": null,
      "scopes": null
```

```
    },
    "domainName": "id.execute-api.us-east-1.amazonaws.com",
    "domainPrefix": "id",
    "extendedRequestId": "request-id",
    "httpMethod": "GET",
    "identity": {
      "accessKey": null,
      "accountId": null,
      "caller": null,
      "cognitoAuthenticationProvider": null,
      "cognitoAuthenticationType": null,
      "cognitoIdentityId": null,
      "cognitoIdentityPoolId": null,
      "principalOrgId": null,
      "sourceIp": "192.0.2.1",
      "user": null,
      "userAgent": "user-agent",
      "userArn": null,
      "clientCert": {
        "clientCertPem": "CERT_CONTENT",
        "subjectDN": "www.example.com",
        "issuerDN": "Example issuer",
        "serialNumber": "a1:a1:a1:a1:a1:a1:a1:a1:a1:a1:a1:a1:a1:a1:a1:a1",
        "validity": {
          "notBefore": "May 28 12:30:02 2019 GMT",
          "notAfter": "Aug  5 09:36:04 2021 GMT"
        }
      }
    },
    "path": "/my/path",
    "protocol": "HTTP/1.1",
    "requestId": "id=",
    "requestTime": "04/Mar/2020:19:15:17 +0000",
    "requestTimeEpoch": 1583349317135,
    "resourceId": null,
    "resourcePath": "/my/path",
    "stage": "$default"
  },
  "pathParameters": null,
  "stageVariables": null,
  "body": "Hello from Lambda!",
  "isBase64Encoded": false
}
```

Lambda 함수 응답 형식

페이로드 형식 버전은 Lambda 함수가 반환해야 하는 응답 구조를 결정합니다.

형식 1.0에 대한 Lambda 함수 응답

`1.0` 형식 버전에서는 Lambda 통합이 다음 형식으로 응답을 반환해야 합니다.

Example

```
{
    "isBase64Encoded": true|false,
    "statusCode": httpStatusCode,
    "headers": { "headername": "headervalue", ... },
    "multiValueHeaders": { "headername": ["headervalue", "headervalue2", ...], ... },
    "body": "..."
}
```

형식 2.0에 대한 Lambda 함수 응답

`2.0` 형식 버전을 사용하면 API Gateway가 응답 형식을 추론할 수 있습니다. API Gateway는 Lambda 함수가 유효한 JSON을 반환하고 `statusCode`을 반환하지 않는 경우 다음과 같은 가정을 합니다.

- `isBase64Encoded`은 `false`입니다.
- `statusCode`은 `200`입니다.
- `content-type`은 `application/json`입니다.
- `body`은 함수의 응답입니다.

다음 예제는 Lambda 함수의 출력과 API Gateway의 해석을 보여 줍니다.

Lambda 함수 출력	API Gateway 해석
`"Hello from Lambda!"`	`{ "isBase64Encoded": false, "statusCode": 200, "body": "Hello from Lambda!", "headers": { "content-type": "application/json" } }`

Lambda 함수 출력	API Gateway 해석
	}

{ "message": "Hello from Lambda!" }	{
	  "isBase64Encoded": false,
	  "statusCode": 200,
	  "body": "{ \"message\": \"Hello from Lambda!\" }",
	  "headers": {
	  "content-type": "application/json"
	  }
	}

응답을 사용자 지정하려면 Lambda 함수가 다음 형식의 응답을 반환해야 합니다.

```
{
    "cookies" : ["cookie1", "cookie2"],
    "isBase64Encoded": true|false,
    "statusCode": httpStatusCode,
    "headers": { "headername": "headervalue", ... },
    "body": "Hello from Lambda!"
}
```

HTTP API에 대한 HTTP 프록시 통합 작업

HTTP 프록시 통합을 사용하면 API 라우팅을 공개적으로 라우팅할 수 있는 HTTP 엔드포인트에 연결할 수 있습니다. 이 통합 유형에서는 API Gateway가 프런트 엔드와 백엔드 사이에 전체 요청 및 응답을 전달합니다.

HTTP 통합을 생성하려면 공개적으로 라우팅할 수 있는 HTTP 엔드포인트의 URL을 제공합니다.

경로 변수와의 HTTP 프록시 통합

HTTP API 라우팅에서 경로 변수를 사용할 수 있습니다.

예를 들어 라우팅 `/pets/{petID}`은 `/pets/6`에 대한 요청을 포착합니다. 통합 URI의 경로 변수를 참조하여 변수의 내용을 통합에 전송할 수 있습니다. 예를 들면, `/pets/extendedpath/{petID}`입니다.

복잡한 경로 변수를 사용하여 라우팅의 모든 자식 리소스를 포착할 수 있습니다. 복잡한 경로 변수를 생성하려면 변수 이름에 +를 추가합니다(예: `{proxy+}`).

모든 요청을 포착하는 HTTP 프록시 통합으로 라우팅을 설정하려면 복잡한 경로 변수(예: `/parent/{proxy+}`)를 사용하여 API 라우팅을 생성합니다. `https://petstore-demo-endpoint.execute-api.com/petstore/{proxy}` 메서드에서 HTTP 엔드포인트(예: ANY)에 라우팅을 통합합니다. 복잡한 경로 변수는 리소스 경로의 끝에 있어야 합니다.

HTTP API에 대한 AWS 서비스 통합 작업

1급 통합을 사용하여 AWS 서비스와 HTTP API를 통합할 수 있습니다. 1급 통합은 HTTP API 경로를 AWS 서비스 API에 연결합니다. 클라이언트가 1급 통합이 지원되는 경로를 호출하면 API Gateway에서 AWS 서비스 API를 자동으로 호출합니다. 예를 들어 1급 통합을 사용하여 Amazon Simple Queue Service 대기열에 메시지를 보내거나 AWS Step Functions 상태 시스템을 시작할 수 있습니다. 지원되는 서비스 작업은 the section called “AWS 서비스 통합 참조” 단원을 참조하세요.

매핑 요청 파라미터

1급 통합에는 필수 및 선택적 파라미터가 사용됩니다. 통합을 생성하려면 필요한 모든 파라미터를 구성해야 합니다. 정적 값을 사용하거나 실행 시간에 동적으로 평가되는 파라미터를 매핑할 수 있습니다. 지원되는 통합 및 파라미터의 전체 목록은 the section called “AWS 서비스 통합 참조” 단원을 참조하세요.

파라미터 매핑

유형	예	참고
헤더 값	$request.header.*name*	헤더 이름은 대/소문자를 구분하지 않습니다. API Gateway는 쉼표를 사용하여 여러 헤더 값을 결합합니다(예: `"header1" : "value1,value2"`).
쿼리 문자열 값	$request.querystring.*name*	쿼리 문자열 이름은 대/소문자를 구분합니다. API Gateway는 쉼표를 사용하여 여러 값을 결합합니다(예: `"querystring1": "Value1,Value2"`).

유형	예	참고
경로 파라미터	$request.path.*name*	요청에 포함된 경로 파라미터의 값입니다. 예를 들어 라우팅이 `/pets/{petId}` 인 경우 요청의 `petId` 파라미터를 *`$request.path.petId`* 로 매핑할 수 있습니다.
요청 본문 패스스루	$request.body	API Gateway는 전체 요청 본문을 전달합니다.
요청 본문	$request.body.*name*	JSON 경로 표현식. 재귀적 하강(`$request.body..` *name*) 및 필터 표현식(`?(`*expression*`)`)은 지원되지 않습니다. **Note** JSON 경로를 지정하면 API Gateway가 요청 본문을 100KB로 잘라낸 다음 선택 표현식을 적용합니다. 100KB보다 큰 페이로드를 보내려면 `$request.body` 을(를) 지정합니다.
컨텍스트 변수	$context.*variableName*	지원되는 컨텍스트 변수의 값입니다.
단계 변수	$stageVariables.*variableName*	스테이지 변수의 값입니다.
정적 값	*`string`*	상수 값입니다.

1급 통합 생성

1급 통합을 생성하기 전에 통합 중인 AWS 서비스 작업을 호출할 수 있는 권한을 API Gateway에 부여하는 IAM 역할을 생성해야 합니다. 자세한 내용은 AWS 서비스에 대한 역할 생성을 참조하세요.

1급 통합을 생성하려면 지원되는 AWS 서비스 작업(예: SQS-SendMessage)을 선택하고, 요청 파라미터를 구성하고, 통합된 AWS 서비스 API를 호출할 수 있는 권한을 API Gateway에 부여하는 역할을 제공합니다. 통합 하위 유형에 따라 다른 요청 파라미터가 필요합니다. 자세한 내용은 the section called "AWS 서비스 통합 참조" 단원을 참조하세요.

다음 AWS CLI 명령은 Amazon SQS 메시지를 보내는 통합을 생성합니다.

```
aws apigatewayv2 create-integration \
    --api-id abcdef123 \
    --integration-subtype SQS-SendMessage \
    --integration-type AWS_PROXY \
    --payload-format-version 1.0 \
    --credentials-arn arn:aws:iam::123456789012:role/apigateway-sqs \
    --request-parameters '{"QueueUrl": "$request.header.queueUrl", "MessageBody":
 "$request.body.message"}'
```

AWS CloudFormation를 사용하여 1급 통합 생성

다음 예시는 Amazon EventBridge과의 최고 수준의 통합을 통해 /{source}/{detailType} 경로를 생성하는 AWS CloudFormation 스니펫을 보여줍니다.

Source 파라미터는 {source} 경로 파라미터에 매핑되고, DetailType는 {DetailType} 경로 파라미터에 매핑되며, Detail 파라미터는 요청 본문에 매핑됩니다.

이 스니펫에는 API Gateway에 PutEvents 작업을 간접 호출할 권한을 부여하는 이벤트 버스 또는 IAM 역할이 표시되지 않습니다.

```
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"
```

통합 하위 유형 참조

HTTP API에 지원되는 통합 하위 유형은 다음과 같습니다.

통합 하위 유형

- EventBridge-PutEvents
- SQS-SendMessage
- SQS-ReceiveMessage
- SQS-DeleteMessage
- SQS-PurgeQueue
- AppConfig-GetConfiguration
- Kinesis-PutRecord
- StepFunctions-StartExecution
- StepFunctions-StartSyncExecution
- StepFunctions-StopExecution

EventBridge-PutEvents

규칙에 일치시킬 수 있도록 사용자 지정 이벤트를 Amazon EventBridge로 보냅니다.

EventBridge-PutEvents 1.0

파라미터	필수
세부 정보	True

파라미터	필수
DetailType	True
소스	True
시간	False
EventBusName	False
리소스	False
Region	False
TraceHeader	False

자세한 내용은 Amazon EventBridge API 참조의 PutEvents를 참조하세요.

SQS-SendMessage

메시지를 지정된 대기열에 전달합니다.

SQS-SendMessage 1.0

파라미터	필수
QueueUrl	True
MessageBody	True
DelaySeconds	False
MessageAttributes	False
MessageDeduplicationId	False
MessageGroupId	False
MessageSystemAttributes	False
Region	False

자세한 내용은 Amazon Simple Queue Service API 참조의 SendMessage를 참조하세요.

SQS-ReceiveMessage

지정된 대기열에서 하나 이상의 메시지(최대 10개)를 가져옵니다.

SQS-ReceiveMessage 1.0

파라미터	필수
QueueUrl	True
AttributeNames	False
MaxNumberOfMessages	False
MessageAttributeNames	False
ReceiveRequestAttemptId	False
VisibilityTimeout	False
WaitTimeSeconds	False
Region	False

자세한 내용은 Amazon Simple Queue Service API 참조의 ReceiveMessage를 참조하세요.

SQS-DeleteMessage

지정된 대기열에서 지정된 메시지를 삭제합니다.

SQS-DeleteMessage 1.0

파라미터	필수
ReceiptHandle	True
QueueUrl	True
Region	False

자세한 내용은 Amazon Simple Queue Service API 참조의 DeleteMessage를 참조하세요.

SQS-PurgeQueue

지정된 대기열의 모든 메시지를 삭제합니다.

SQS-PurgeQueue 1.0

파라미터	필수
QueueUrl	True
Region	False

자세한 내용은 Amazon Simple Queue Service API 참조의 PurgeQueue를 참조하세요.

AppConfig-GetConfiguration

구성에 대한 정보를 수신합니다.

AppConfig-GetConfiguration 1.0

파라미터	필수
애플리케이션	True
Environment	True
Configuration	True
ClientId	True
ClientConfigurationVersion	False
Region	False

자세한 내용은 AWS AppConfig API 참조의 GetConfiguration을 참조하세요.

Kinesis-PutRecord

단일 데이터 레코드를 Amazon Kinesis 데이터 스트림에 씁니다.

Kinesis-PutRecord 1.0

파라미터	필수
StreamName	True
Data	True
PartitionKey	True
SequenceNumberForOrdering	False
ExplicitHashKey	False
Region	False

자세한 내용은 Amazon Kinesis Data Streams API 참조의 PutRecord를 참조하세요.

StepFunctions-StartExecution

상태 시스템 실행을 시작합니다.

StepFunctions-StartExecution 1.0

파라미터	필수
StateMachineArn	True
이름	False
입력	False
Region	False

자세한 내용은 AWS Step Functions API 참조의 StartExecution을 참조하세요.

StepFunctions-StartSyncExecution

동기 상태 시스템 실행을 시작합니다.

StepFunctions-StartSyncExecution 1.0

파라미터	필수
StateMachineArn	True
이름	False
입력	False
Region	False
TraceHeader	False

자세한 내용은 AWS Step Functions API 참조의 StartSyncExecution을 참조하세요.

StepFunctions-StopExecution

실행을 중지합니다.

StepFunctions-StopExecution 1.0

파라미터	필수
ExecutionArn	True
원인	False
오류	False
Region	False

자세한 내용은 AWS Step Functions API 참조의 StopExecution을 참조하세요.

HTTP API에 대한 프라이빗 통합 작업

프라이빗 통합을 사용하면 VPC의 프라이빗 리소스(예: Application Load Balancer 또는 Amazon ECS 컨테이너 기반 애플리케이션)와의 API 통합을 생성할 수 있습니다.

프라이빗 통합을 사용하여 VPC 외부의 클라이언트가 액세스할 수 있도록 VPC에 리소스를 노출할 수 있습니다. API Gateway가 지원하는 권한 부여 방법을 사용하여 API에 대한 액세스를 제어할 수 있습니다.

프라이빗 통합을 생성하려면 먼저 VPC 링크를 생성해야 합니다. VPC 링크에 대한 자세한 내용은 HTTP API에 대한 VPC 링크 작업 단원을 참조하세요 .

VPC 링크를 생성한 후에는 Application Load Balancer, Network Load Balancer 또는 AWS Cloud Map 서비스에 등록된 리소스에 연결되는 프라이빗 통합을 설정할 수 있습니다.

프라이빗 통합을 생성하려면 모든 리소스를 동일한 AWS 계정(로드 밸런서 또는 AWS Cloud Map 서비스, VPC 링크 및 HTTP API 포함)이 소유해야 합니다.

기본적으로 프라이빗 통합 트래픽은 HTTP 프로토콜을 사용합니다. HTTPS를 사용하기 위해 프라이빗 통합 트래픽이 필요한 경우, `tlsConfig`를 지정할 수 있습니다.

> **Note**
>
> 프라이빗 통합의 경우 API Gateway에서 백엔드 리소스에 대한 요청에 API 엔드포인트의 스테이지 부분을 포함합니다. 예를 들어 API `test` 스테이지에 대한 요청은 프라이빗 통합 요청에 `test/`*`route-path`*를 포함합니다. 백엔드 리소스에 대한 요청에서 단계 이름을 제거하려면 파라미터 매핑을 사용하여 요청 경로를 `$request.path`에 덮어씁니다.

Application Load Balancer 또는 Network Load Balancer를 사용하여 프라이빗 통합 생성

프라이빗 통합을 생성하기 전에 VPC 링크를 생성해야 합니다. VPC 링크에 대한 자세한 내용은 HTTP API에 대한 VPC 링크 작업 단원을 참조하세요 .

Application Load Balancer 또는 Network Load Balancer에서 프라이빗 통합을 생성하려면 HTTP 프록시 통합을 생성하고, 사용할 VPC 링크를 지정하고, 로드 밸런서의 리스너 ARN을 제공합니다.

다음 명령을 사용하여 VPC 링크를 통해 로드 밸런서에 연결하는 프라이빗 통합을 생성합니다.

```
aws apigatewayv2 create-integration --api-id api-id --integration-type HTTP_PROXY \
    --integration-method GET --connection-type VPC_LINK \
    --connection-id VPC-link-ID \
    --integration-uri arn:aws:elasticloadbalancing:us-east-2:123456789012:listener/app/
my-load-balancer/50dc6c495c0c9188/0467ef3c8400ae65
    --payload-format-version 1.0
```

AWS Cloud Map 서비스 검색을 사용하여 프라이빗 통합 생성

프라이빗 통합을 생성하기 전에 VPC 링크를 생성해야 합니다. VPC 링크에 대한 자세한 내용은 HTTP API에 대한 VPC 링크 작업 단원을 참조하세요 .

AWS Cloud Map와의 통합을 위해 API Gateway는 `DiscoverInstances`를 사용하여 리소스를 식별합니다. 쿼리 파라미터를 사용하여 특정 리소스를 대상으로 지정할 수 있습니다. 등록된 리소스의 속성에는 IP 주소와 포트가 포함되어야 합니다. API Gateway는 `DiscoverInstances`에서 반환된 정상 리소스에 요청을 분산합니다. 자세한 내용은 AWS Cloud Map API 참조의 DiscoverInstances를 참조하세요.

> **Note**
>
> Amazon ECS를 사용하여 AWS Cloud Map의 항목을 채우는 경우 Amazon ECS 서비스 검색과 함께 SRV 레코드를 사용하도록 Amazon ECS 작업을 구성하거나 Amazon ECS Service Connect를 활성화해야 합니다. 자세한 내용은 Amazon Elastic Container Service 개발자 안내서의 서비스 상호 연결을 참조하세요.

AWS Cloud Map과의 프라이빗 통합을 생성하려면 HTTP 프록시 통합을 생성하고, 사용할 VPC 링크를 지정하고, AWS Cloud Map 서비스의 ARN을 제공합니다.

다음 명령을 사용하여 AWS Cloud Map 서비스 검색을 통해 리소스를 식별하는 프라이빗 통합을 생성합니다.

```
aws apigatewayv2 create-integration --api-id api-id --integration-type HTTP_PROXY  \
    --integration-method GET --connection-type VPC_LINK \
    --connection-id VPC-link-ID \
    --integration-uri arn:aws:servicediscovery:us-east-2:123456789012:service/srv-id?
stage=prod&deployment=green_deployment
    --payload-format-version 1.0
```

HTTP API에 대한 VPC 링크 작업

VPC 링크를 사용하여 HTTP API 라우팅을 VPC의 프라이빗 리소스(예: Application Load Balancer 또는 Amazon ECS 컨테이너 기반 애플리케이션)에 연결하는 프라이빗 통합을 생성할 수 있습니다. 프라이빗 통합 생성에 대한 자세한 내용은 HTTP API에 대한 프라이빗 통합 작업 단원을 참조하세요.

프라이빗 통합은 VPC 링크를 사용하여 API Gateway와 대상 VPC 리소스 간의 연결을 캡슐화합니다. 서로 다른 라우팅 및 API에서 VPC 링크를 재사용할 수 있습니다.

VPC 링크를 생성할 때 API Gateway는 계정에서 VPC 링크에 대한 탄력적 네트워크 인터페이스를 생성 및 관리합니다. 이 프로세스는 몇 분 정도 걸릴 수 있습니다. VPC 링크를 사용할 준비가 되면 상태가 `PENDING`에서 `AVAILABLE`로 바뀝니다.

> **Note**
>
> 60일 동안 VPC 링크를 통해 전송되는 트래픽이 없으면 이것이 `INACTIVE`이 됩니다. VPC 링크가 `INACTIVE` 상태에 있으면 API Gateway는 VPC 링크의 네트워크 인터페이스를 모두 삭제합니다. 이로 인해 VPC 링크를 사용하는 API 요청이 실패하게 됩니다. API 요청이 재개되면 API Gateway는 네트워크 인터페이스를 다시 프로비저닝합니다. 네트워크 인터페이스를 생성하고 VPC 링크를 다시 활성화하는 데 몇 분 정도 걸릴 수 있습니다. VPC 링크 상태를 사용하여 VPC 링크의 상태를 모니터링할 수 있습니다.

AWS CLI를 사용하여 VPC 링크 생성

다음 명령을 사용하여 VPC 링크를 생성합니다. VPC 링크를 생성하려면 관련된 모든 리소스를 동일한 AWS 계정이 소유해야 합니다.

```
aws apigatewayv2 create-vpc-link --name MyVpcLink \
    --subnet-ids subnet-aaaa subnet-bbbb \
    --security-group-ids sg1234 sg5678
```

> **Note**
>
> VPC 링크는 변경할 수 없습니다. VPC 링크를 생성한 후에는 해당 서브넷이나 보안 그룹을 변경할 수 없습니다.

AWS CLI을 사용하여 VPC 링크 삭제

다음 명령을 사용하여 VPC 링크를 삭제합니다.

```
aws apigatewayv2 delete-vpc-link --vpc-link-id abcd123
```

리전별 가용성

HTTP API에 대한 VPC 링크는 다음 리전 및 가용 영역에서 지원됩니다.

리전 이름	리전(Region)	지원되는 가용 영역
미국 동부(오하이오)	us-east-2	use2-az1, use2-az2, use2-az3
미국 동부(버지니아 북부)	us-east-1	use1-az1, use1-az2, use1-az4, use1-az5, use1-az6
미국 서부(캘리포니아 북부 지역)	us-west-1	usw1-az1, usw1-az3
미국 서부(오레곤)	us-west-2	usw2-az1, usw2-az2, usw2-az3, usw2-az4
Asia Pacific (Hong Kong)	ap-east-1	ape1-az2, ape1-az3
Asia Pacific (Mumbai)	ap-south-1	aps1-az1, aps1-az2, aps1-az3
Asia Pacific (Seoul)	ap-northeast-2	apne2-az1, apne2-az2, apne2-az3
아시아 태평양(싱가포르)	ap-southeast-1	apse1-az1, apse1-az2, apse1-az3
아시아 태평양(시드니)	ap-southeast-2	apse2-az1, apse2-az2, apse2-az3
Asia Pacific (Tokyo)	ap-northeast-1	apne1-az1, apne1-az2, apne1-az4
Canada (Central)	ca-central-1	cac1-az1, cac1-az2
Europe (Frankfurt)	eu-central-1	euc1-az1, euc1-az2, euc1-az3

리전 이름	리전(Region)	지원되는 가용 영역
유럽(아일랜드)	eu-west-1	euw1-az1, euw1-az2, euw1-az3
Europe (London)	eu-west-2	euw2-az1, euw2-az2, euw2-az3
Europe (Paris)	eu-west-3	euw3-az1, euw3-az3
Europe (Stockholm)	eu-north-1	eun1-az1, eun1-az2, eun1-az3
Middle East (Bahrain)	me-south-1	mes1-az1, mes1-az2, mes1-az3
남아메리카(상파울루)	sa-east-1	sae1-az1, sae1-az2, sae1-az3
AWS GovCloud (US-West)	us-gov-west-1	usgw1-az1, usgw1-az2, usgw1-az3

HTTP API에 대한 CORS 구성

CORS(Cross-Origin Resource Sharing)는 브라우저에서 실행 중인 스크립트에서 시작되는 HTTP 요청을 제한하는 브라우저 보안 기능입니다. API에 액세스할 수 없고 `Cross-Origin Request Blocked`가 포함된 오류 메시지를 수신하는 경우 CORS를 활성화해야 할 수 있습니다.

CORS는 일반적으로 다른 도메인이나 오리진에 호스팅된 API에 액세스하는 웹 애플리케이션을 빌드하는 데 필요합니다. CORS를 활성화하여 다른 도메인에 호스팅된 웹 애플리케이션에서 API에 대한 요청을 허용할 수 있습니다. 예를 들어, API가 `https://{api_id}.execute-api.{region}.amazonaws.com/`에 호스팅되고 `example.com`에 호스팅된 웹 애플리케이션에서 API를 호출하려는 경우 API가 CORS를 지원해야 합니다.

API에 대해 CORS를 구성하면 API Gateway는 API에 대해 구성된 OPTIONS 경로가 없더라도 preflight OPTIONS 요청에 대한 응답을 자동으로 전송합니다. CORS 요청의 경우 API Gateway는 구성된 CORS 헤더를 통합의 응답에 추가합니다.

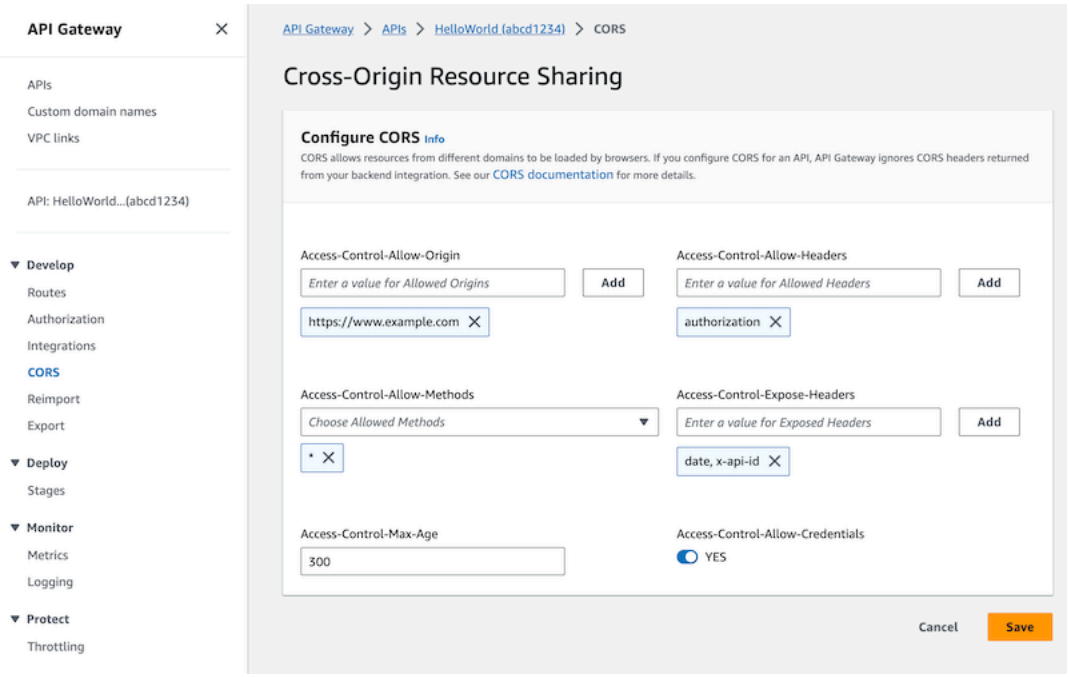
> **Note**
>
> API에 대해 CORS를 구성하는 경우, API Gateway는 백엔드 통합에서 반환된 CORS 헤더를 무시합니다.

CORS 구성에서 다음과 같은 매개 변수를 지정할 수 있습니다. API Gateway HTTP API 콘솔을 사용하여 이러한 파라미터를 추가하려면 값을 입력한 후 추가를 선택합니다.

CORS 헤더	CORS 구성 속성	예제 값
Access-Control-Allow-Origin	allowOrigins	• `https://www.example.com` • *(모든 오리진 허용) • `https://*` (https://로 시작하는 모든 오리진 허용) • `http://*`(http://로 시작하는 모든 오리진 허용)
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, *

CORS 헤더를 반환하려면 요청에 `origin` 헤더가 포함되어야 합니다.

CORS 구성은 다음과 비슷할 수 있습니다.



`$default` 라우팅 및 JWT 권한 부여자가 포함된 HTTP API에 CORS 구성

CORS를 활성화하고 HTTP API의 모든 라우팅에 대한 권한 부여를 구성할 수 있습니다. $default 라우팅에 대한 CORS 및 권한 부여를 활성화하는 경우 고려할 몇 가지 사항이 있습니다. $default 라우팅은 OPTIONS 요청을 포함하여 명시적으로 정의되지 않은 모든 메서드 및 라우팅에 대한 요청을 포착합니다. 권한 부여되지 않은 OPTIONS 요청을 지원하려면 OPTIONS /{proxy+} 라우팅을 권한 부여가 필요하지 않은 API에 추가하고 통합을 라우팅에 연결합니다. OPTIONS /{proxy+} 라우팅이 $default 라우팅보다 우선 순위가 높습니다. 따라서 클라이언트가 권한 부여 없이 API에 OPTIONS 요청을 제출할 수 있습니다. 라우팅 우선 순위에 대한 자세한 내용은 API 요청 라우팅 단원을 참조하세요.

AWS CLI를 사용하여 HTTP API에 대한 CORS 구성

다음 명령을 사용하여 https://www.example.com에서 CORS 요청을 활성화할 수 있습니다.

Example

```
aws apigatewayv2 update-api --api-id api-id --cors-configuration AllowOrigins="https://
www.example.com"
```

자세한 내용은 Amazon API Gateway 버전 2 API 참조의 CORS를 참조하세요.

API 요청 및 응답 변환

백엔드 통합에 도달하기 전에 클라이언트의 API 요청을 수정할 수 있습니다. API Gateway가 클라이언트에 응답을 반환하기 전에 통합에서 응답을 변경할 수도 있습니다. 파라미터 매핑을 사용하여 HTTP API에 대한 API 요청 및 응답을 수정합니다. 파라미터 매핑을 사용하려면 수정할 API 요청 또는 응답 파라미터를 지정하고 이러한 파라미터를 수정하는 방법을 지정합니다.

API 요청 변환

요청 파라미터를 사용하여 요청이 백엔드 통합에 도달하기 전에 요청을 변경합니다. 헤더, 쿼리 문자열 또는 요청 경로를 수정할 수 있습니다.

요청 파라미터는 키-값 맵입니다. 키는 변경할 요청 파라미터의 위치와 변경 방법을 식별합니다. 이 값은 파라미터에 대한 새 데이터를 지정합니다.

다음 표에는 지원되는 키가 나와 있습니다.

파라미터 매핑 키

유형	구문
헤더	`append|overwrite|remove:header.` *`headername`*
쿼리 문자열	`append|overwrite|remove:querystring.` *`querystring-name`*
경로	`overwrite:path`

다음 표에서는 파라미터에 매핑할 수 있는 지원되는 값을 보여 줍니다.

요청 파라미터 매핑 값

유형	구문	참고
헤더 값	$request.header.*name* 또는 ${request.header.*name*}	헤더 이름은 대/소문자를 구분하지 않습니다. API Gateway는 쉼표를 사용하여 여러 헤더 값을 결합합니다(예: "header1"

유형	구문	참고
		: "value1,value2"). 일부 헤더는 예약되어 있습니다. 자세한 내용은 the section called “예약된 헤더” 단원을 참조하세요.
쿼리 문자열 값	$request.querystring.*name* 또는 ${request.querystring.*name*}	쿼리 문자열 이름은 대/소문자를 구분합니다. API Gateway는 쉼표를 사용하여 여러 값을 결합합니다 (예: "querystring1" "Value1,Value2").
요청 본문	$request.body.*name* 또는 ${request.body.*name*}	JSON 경로 표현식입니다. 재귀적 하강($request.body..name)) 및 필터 표현식(?(expression))은 지원되지 않습니다. **Note** JSON 경로를 지정하면 API Gateway가 요청 본문을 100KB로 잘라낸 다음 선택 표현식을 적용합니다. 100KB보다 큰 페이로드를 보내려면 $request.body 을(를) 지정합니다.
요청 경로	$request.path 또는 ${request.path}	단계 이름이 없는 요청 경로입니다.

유형	구문	참고
경로 파라미터	$request.path.*name* 또는 ${request.path.*name*}	요청에 포함된 경로 파라미터의 값입니다. 예를 들어 라우팅이 `/pets/{petId}` 인 경우 요청의 `petId` 파라미터를 `$request.path.petId` (으)로 매핑할 수 있습니다.
컨텍스트 변수	$context.*variableName* 또는 ${context.*variableName* }	컨텍스트 변수의 값입니다. **Note** 특수 문자 `.` 및 `_`만 지원됩니다.
단계 변수	$stageVariables.*variableName* 또는 ${stageVariables.*variableName* }	스테이지 변수의 값입니다.
정적 값	*string*	상수 값입니다.

Note

선택 표현식에 여러 변수를 사용하려면 변수를 괄호로 묶습니다. 예: `${request.path.name} ${request.path.id}`.

API 응답 변환

응답 파라미터를 사용하여 클라이언트에 응답을 반환하기 전에 백엔드 통합에서 HTTP 응답을 변환합니다. API Gateway가 클라이언트에 응답을 반환하기 전에 응답의 헤더 또는 상태 코드를 수정할 수 있습니다.

통합에서 반환하는 각 상태 코드에 대한 응답 파라미터를 구성합니다. 응답 파라미터는 키-값 맵입니다. 키는 변경할 요청 파라미터의 위치와 변경 방법을 식별합니다. 이 값은 파라미터에 대한 새 데이터를 지정합니다.

다음 표에는 지원되는 키가 나와 있습니다.

응답 파라미터 매핑 키

유형	구문
헤더	`append|overwrite|remove:header. headername`
상태 코드	`overwrite:statuscode`

다음 표에서는 파라미터에 매핑할 수 있는 지원되는 값을 보여 줍니다.

응답 파라미터 매핑 값

유형	구문	참고
헤더 값	$response.header.*name* 또는 ${response.header.*name*}	헤더 이름은 대/소문자를 구분하지 않습니다. API Gateway는 쉼표를 사용하여 여러 헤더 값을 결합합니다(예: `"header1" : "value1,value2"`). 일부 헤더는 예약되어 있습니다. 자세한 내용은 the section called “예약된 헤더” 단원을 참조하세요.
응답 본문	$response.body.*name* 또는 ${response.body.*name*}	JSON 경로 표현식. 재귀적 하강(`$response.body..name`) 및 필터 표현식(`?(expression)`)은 지원되지 않습니다. **Note** JSON 경로를 지정하면 API Gateway가 응답 본문을 100KB로 잘라낸 다음 선택 표현식

유형	구문	참고
		을 적용합니다. 100KB 보다 큰 페이로드를 보내려면 `$response.body` 을(를) 지정합니다.
컨텍스트 변수	$context.*variableName* 또는 ${context.*variableName*}	지원되는 컨텍스트 변수의 값입니다.
단계 변수	$stageVariables.*variableName* 또는 ${stageVariables.*variableName*}	스테이지 변수의 값입니다.
정적 값	*string*	상수 값입니다.

> **Note**
>
> 선택 표현식에 여러 변수를 사용하려면 변수를 괄호로 묶습니다. 예: `${request.path.name} ${request.path.id}`.

예약된 헤더

다음 헤더가 예약되어 있습니다. 이러한 헤더에 대한 요청 또는 응답 매핑은 구성할 수 없습니다.

- access-control-*
- apigw-*
- 승인
- 연결
- Content-Encoding
- Content-Length
- Content-Location
- 전달됨

- Keep-Alive
- Origin
- 프록시-인증
- Proxy-Authorization
- TE
- 트레일러
- Transfer-Encoding
- 업그레이드
- x-amz-*
- x-amzn-*
- X-Forwarded-For
- X-Forwarded-Host
- X-Forwarded-Proto
- Via

예제

다음 AWS CLI 예제에서는 파라미터 매핑을 구성합니다. AWS CloudFormation 템플릿 예는 GitHub를 참조하세요.

API 요청에 헤더 추가

다음 예제에서는 백엔드 통합에 도달하기 전에 API 요청에 header1(이)라는 헤더를 추가합니다. API Gateway는 요청 ID로 헤더를 채웁니다.

```
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" }'
```

요청 헤더 이름 바꾸기

다음 예제에서는 요청 헤더의 이름을 header1에서 header2(으)로변경합니다.

```
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": "''"}'
```

통합에서 응답 변경

다음 예에서는 통합에 대한 응답 파라미터를 구성합니다. 통합에서 500 상태 코드를 반환하면 API Gateway가 상태 코드를 403으로 변경하고 응답에 header11을 추가합니다. 통합에서 404 상태 코드를 반환하면 API Gateway는 응답에 `error` 헤더를 추가합니다.

```
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"}  }'
```

구성된 파라미터 매핑 제거

다음 예제 명령은 `append:header.header1`에 대해 이전에 구성된 요청 파라미터를 제거합니다. 또한 200 상태 코드에 대해 이전에 구성된 응답 파라미터를 제거합니다.

```
aws apigatewayv2 update-integration \
    --api-id abcdef123 \
    --integration-id hijk456 \
    --request-parameters '{"append:header.header1" : ""}' \
    --response-parameters '{"200" : {}}'
```

HTTP API에 대한 OpenAPI 정의 작업

OpenAPI 3.0 정의 파일을 사용하여 HTTP API를 정의할 수 있습니다. 그런 다음 정의를 API Gateway로 가져와서 API를 생성할 수 있습니다. OpenAPI로 API Gateway 확장에 대한 자세한 내용은 OpenAPI 확장 단원을 참조하세요.

HTTP API 가져오기

OpenAPI 3.0 정의 파일을 가져와서 HTTP API를 생성할 수 있습니다.

REST API에서 HTTP API로 마이그레이션하려면 REST API를 OpenAPI 3.0 정의 파일로 내보낼 수 있습니다. 그런 다음 API 정의를 HTTP API로 가져옵니다. REST API 내보내기에 대한 자세한 내용은 API Gateway에서 REST API 내보내기 단원을 참조하세요.

> **Note**
>
> HTTP API는 REST API와 동일한 AWS 변수를 지원합니다. 자세한 내용은 AWSOpenAPI 가져오기를 위한 변수 단원을 참조하세요.

검증 정보 가져오기

API를 가져올 때 API Gateway는 세 가지 범주의 검증 정보를 제공합니다.

Info

속성은 OpenAPI 사양에 따라 유효하지만, HTTP API에 대해서는 해당 속성이 지원되지 않습니다.

예를 들어, HTTP API는 요청 검증을 지원하지 않으므로 다음 OpenAPI 3.0 코드 조각은 가져오기에 대한 정보를 생성합니다. API Gateway는 `requestBody` 및 `schema` 필드를 무시합니다.

```
"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"
        }
      }
    }
    ...
  }
  ...
}
```

경고

속성 또는 구조는 OpenAPI 사양에 따라 유효하지 않지만, API 생성을 차단하지 않습니다. API Gateway가 이러한 경고를 무시하고 API 생성을 계속할지 또는 경고 시 API 생성을 중지할지를 지정할 수 있습니다.

HTTP API는 Lambda 프록시 및 HTTP 프록시 통합만 지원하므로 다음과 같은 OpenAPI 3.0 문서에서는 가져오기에 대한 경고가 생성됩니다.

```
"x-amazon-apigateway-integration": {
  "type": "AWS",
  "httpMethod": "POST",
  "uri": "arn:aws:lambda:us-east-2:123456789012:function:HelloWorld",
  "payloadFormatVersion": "1.0"
}
```

오류

OpenAPI 사양이 올바르지 않거나 형식이 잘못되었습니다. API Gateway는 잘못된 형식의 문서에서 리소스를 만들 수 없습니다. 오류를 수정한 다음, 다시 시도해야 합니다.

HTTP API는 OpenAPI 3.0 사양만 지원하므로 다음과 같은 API 정의는 가져올 때 오류가 발생합니다.

```
{
  "swagger": "2.0.0",
  "info": {
    "title": "My API",
    "description": "An Example OpenAPI definition for Errors/Warnings/ImportInfo",
    "version": "1.0"
  }
  ...
}
```

또 다른 예로 OpenAPI를 사용하면 특정 작업에 연결된 여러 보안 요구 사항이 있는 API를 정의할 수 있지만 API Gateway는 이를 지원하지 않습니다. 각 작업에는 IAM 권한 부여, Lambda 권한 부여자 또는 JWT 권한 부여자가 하나만 있을 수 있습니다. 여러 보안 요구 사항을 모형화하려 시도하면 오류가 발생합니다.

AWS CLI를 사용하여 API 가져오기

다음 명령은 OpenAPI 3.0 정의 파일 `api-definition.json`을 HTTP API로 가져옵니다.

Example

```
aws apigatewayv2 import-api --body file://api-definition.json
```

Example

다음 예제 OpenAPI 3.0 정의를 가져와서 HTTP API를 생성할 수 있습니다.

```
{
  "openapi": "3.0.1",
  "info": {
    "title": "Example Pet Store",
    "description": "A Pet Store API.",
    "version": "1.0"
  },
  "paths": {
    "/pets": {
      "get": {
        "operationId": "GET HTTP",
        "parameters": [
          {
            "name": "type",
```

```
            "in": "query",
            "schema": {
              "type": "string"
            }
          },
          {
            "name": "page",
            "in": "query",
            "schema": {
              "type": "string"
            }
          }
        ],
        "responses": {
          "200": {
            "description": "200 response",
            "headers": {
              "Access-Control-Allow-Origin": {
                "schema": {
                  "type": "string"
                }
              }
            },
            "content": {
              "application/json": {
                "schema": {
                  "$ref": "#/components/schemas/Pets"
                }
              }
            }
          }
        },
        "x-amazon-apigateway-integration": {
          "type": "HTTP_PROXY",
          "httpMethod": "GET",
          "uri": "http://petstore.execute-api.us-west-1.amazonaws.com/petstore/pets",
          "payloadFormatVersion": 1.0
        }
      },
      "post": {
        "operationId": "Create Pet",
        "requestBody": {
          "content": {
            "application/json": {
```

```
            "schema": {
              "$ref": "#/components/schemas/NewPet"
            }
          }
        },
        "required": true
      },
      "responses": {
        "200": {
          "description": "200 response",
          "headers": {
            "Access-Control-Allow-Origin": {
              "schema": {
                "type": "string"
              }
            }
          },
          "content": {
            "application/json": {
              "schema": {
                "$ref": "#/components/schemas/NewPetResponse"
              }
            }
          }
        }
      },
      "x-amazon-apigateway-integration": {
        "type": "HTTP_PROXY",
        "httpMethod": "POST",
        "uri": "http://petstore.execute-api.us-west-1.amazonaws.com/petstore/pets",
        "payloadFormatVersion": 1.0
      }
    }
  },
  "/pets/{petId}": {
    "get": {
      "operationId": "Get Pet",
      "parameters": [
        {
          "name": "petId",
          "in": "path",
          "required": true,
          "schema": {
            "type": "string"
```

```
            }
          }
        ],
        "responses": {
          "200": {
            "description": "200 response",
            "headers": {
              "Access-Control-Allow-Origin": {
                "schema": {
                  "type": "string"
                }
              }
            },
            "content": {
              "application/json": {
                "schema": {
                  "$ref": "#/components/schemas/Pet"
                }
              }
            }
          }
        },
        "x-amazon-apigateway-integration": {
          "type": "HTTP_PROXY",
          "httpMethod": "GET",
          "uri": "http://petstore.execute-api.us-west-1.amazonaws.com/petstore/pets/
{petId}",
          "payloadFormatVersion": 1.0
        }
      }
    }
  },
  "x-amazon-apigateway-cors": {
    "allowOrigins": [
      "*"
    ],
    "allowMethods": [
      "GET",
      "OPTIONS",
      "POST"
    ],
    "allowHeaders": [
      "x-amzm-header",
      "x-apigateway-header",
```

```
      "x-api-key",
      "authorization",
      "x-amz-date",
      "content-type"
    ]
  },
  "components": {
    "schemas": {
      "Pets": {
        "type": "array",
        "items": {
          "$ref": "#/components/schemas/Pet"
        }
      },
      "Empty": {
        "type": "object"
      },
      "NewPetResponse": {
        "type": "object",
        "properties": {
          "pet": {
            "$ref": "#/components/schemas/Pet"
          },
          "message": {
            "type": "string"
          }
        }
      },
      "Pet": {
        "type": "object",
        "properties": {
          "id": {
            "type": "string"
          },
          "type": {
            "type": "string"
          },
          "price": {
            "type": "number"
          }
        }
      },
      "NewPet": {
        "type": "object",
```

```
        "properties": {
          "type": {
            "$ref": "#/components/schemas/PetType"
          },
          "price": {
            "type": "number"
          }
        }
      },
      "PetType": {
        "type": "string",
        "enum": [
          "dog",
          "cat",
          "fish",
          "bird",
          "gecko"
        ]
      }
    }
  }
}
```

API Gateway에서 HTTP API 내보내기

HTTP API를 생성한 후 API Gateway에서 API의 OpenAPI 3.0 정의를 내보낼 수 있습니다. 내보낼 스테이지를 선택하거나 API의 최신 구성을 내보낼 수 있습니다. 내보낸 API 정의를 API Gateway로 가져와 동일한 또 다른 API를 생성할 수도 있습니다. API 정의 가져오기에 대한 자세한 내용은 HTTP API 가져오기 단원을 참조하세요.

AWS CLI를 사용하여 스테이지의 OpenAPI 3.0 정의 내보내기

다음 명령은 `prod`라는 API 단계의 OpenAPI 정의를 `stage-definition.yaml`이라는 YAML 파일로 내보냅니다. 내보낸 정의 파일에는 기본적으로 API Gateway 확장이 포함됩니다.

```
aws apigatewayv2 export-api \
    --api-id api-id  \
    --output-type YAML  \
    --specification OAS30 \
    --stage-name prod \
    stage-definition.yaml
```

AWS CLI를 사용하여 API의 최신 변경 사항에 대한 OpenAPI 3.0 정의 내보내기

다음 명령은 HTTP API의 OpenAPI 정의를 `latest-api-definition.json`이라는 JSON 파일로 내보냅니다. 이 명령은 스테이지를 지정하지 않으므로 API Gateway는 스테이지에 배포되었는지 여부에 관계없이 API의 최신 구성을 내보냅니다. 내보낸 정의 파일에는API Gateway 확장이 포함되지 않습니다.

```
aws apigatewayv2 export-api \
    --api-id api-id  \
    --output-type JSON  \
    --specification OAS30 \
    --no-include-extensions \
    latest-api-definition.json
```

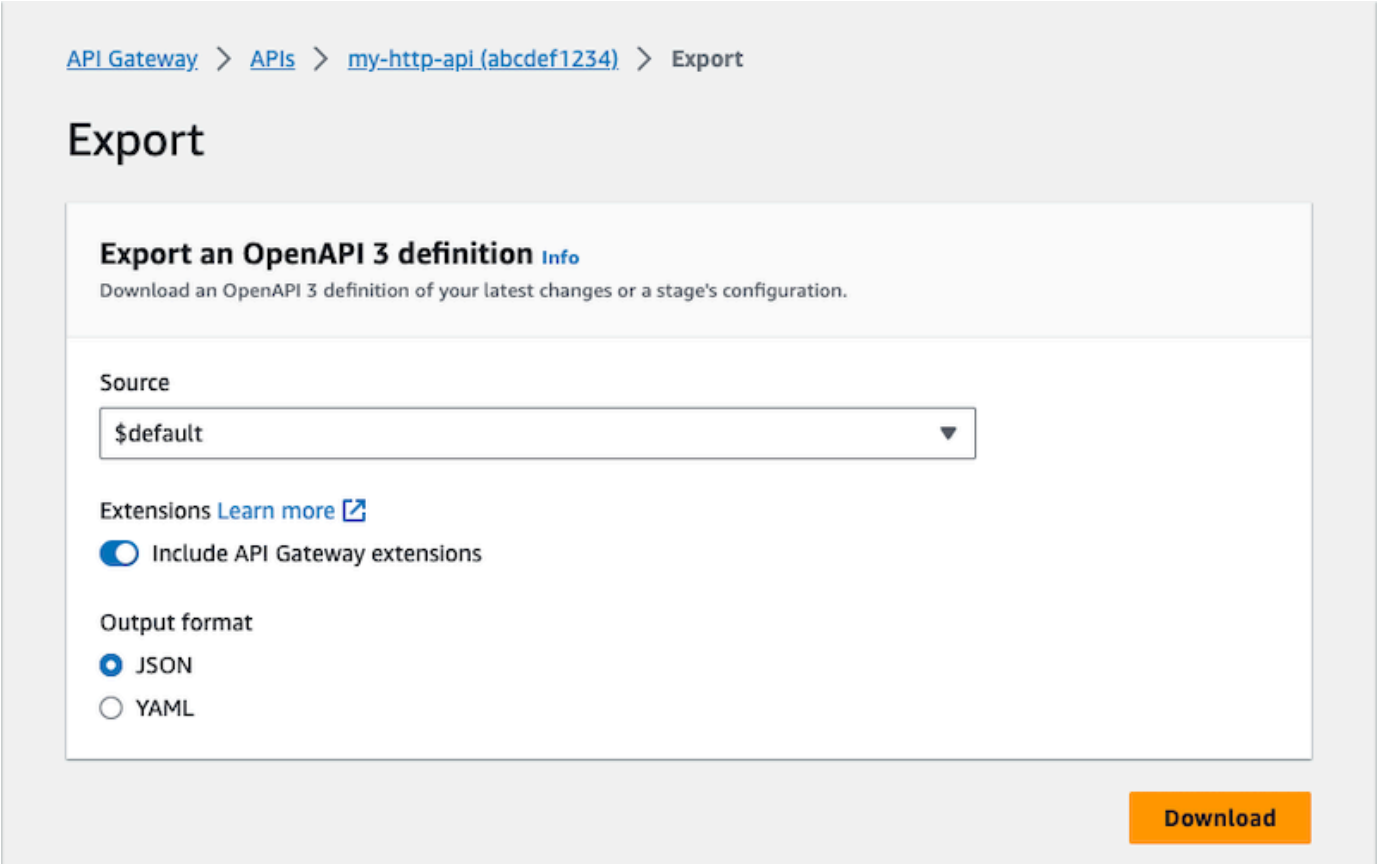
자세한 내용은 Amazon API Gateway 버전 2 API 참조의 ExportAPI를 참조하세요.

API Gateway 콘솔을 사용하여 OpenAPI 3.0 정의 내보내기

다음 절차에서는 HTTP API의 OpenAPI 정의를 내보내는 방법을 보여줍니다.

API Gateway 콘솔을 사용하여 OpenAPI 3.0 정의를 내보내려면

1. https://console.aws.amazon.com/apigateway에서 API Gateway 콘솔에 로그인합니다.
2. HTTP API를 선택합니다.
3. 기본 탐색 창의 개발에서 내보내기를 선택합니다.
4. API 내보내기를 위한 다음과 같은 옵션을 선택합니다.



a. 소스에서 OpenAPI 3.0 정의의 소스를 선택합니다. 내보낼 스테이지를 선택하거나 API의 최신 구성을 내보낼 수 있습니다.

b. API Gateway 확장을 포함하려면 API Gateway 확장 포함을 켭니다.

c. 출력 형식에서 출력 형식을 선택합니다.

5. 다운로드를 선택합니다.

고객이 호출할 수 있도록 HTTP API 게시

스테이지 및 사용자 지정 도메인 이름을 사용하여 클라이언트가 호출할 수 있도록 API를 게시할 수 있습니다.

API 스테이지는 API의 수명 주기 상태에 대한 논리적 참조(예: `dev`, `prod`, `beta`, `v2`)입니다. 각 스테이지는 API 배포에 대한 명명된 참조이며, 클라이언트 애플리케이션 프로그램에서 호출을 하는 데 사용할 수 있습니다. API의 각 스테이지에 대해 통합 및 설정을 서로 다르게 구성할 수 있습니다.

사용자 지정 도메인 이름을 사용하여 클라이언트가 API를 호출할 수 있도록 기본 URL인 `https://api-id.execute-api.region.amazonaws.com/stage` 보다 더 간단하고 직관적인 URL을 제공할 수 있습니다.

> **Note**
>
> API Gateway API의 보안을 강화하기 위해 `execute-api.{region}.amazonaws.com` 도메인은 PSL(Public Suffix List)에 등록됩니다. 보안 강화를 위해 API Gateway API 기본 도메인 이름에 민감한 쿠키를 설정해야 하는 경우 `__Host-` 접두사가 있는 쿠키를 사용하는 것이 좋습니다. 이렇게 하면 교차 사이트 요청 위조 시도 (CSRF) 로부터 도메인을 보호하는 데 도움이 됩니다. 자세한 내용은 Mozilla 개발자 네트워크의 Set-Cookie 페이지를 참조하세요.

주제

- HTTP API에 대한 스테이지 작업
- HTTP API에 대한 보안 정책
- HTTP API에 대한 사용자 지정 도메인 이름 설정

HTTP API에 대한 스테이지 작업

API 스테이지는 API의 수명 주기 상태에 대한 논리적 참조(예: `dev`, `prod`, `beta`, `v2`)입니다. API 스테이지는 API ID 및 스테이지 이름으로 식별되며, API를 호출하는 데 사용하는 URL에 포함됩니다. 각 스테이지는 API 배포에 대한 명명된 참조이며, 클라이언트 애플리케이션 프로그램에서 호출을 하는 데 사용할 수 있습니다.

API의 URL 기반에서 제공되는 `$default` 스테이지를 생성할 수 있습니다(예: `https://{api_id}.execute-api.{region}.amazonaws.com/`). 이 URL을 사용하여 API 단계를 호출합니다.

배포는 API 구성의 스냅샷입니다. 단계에 API를 배포한 후에는 클라이언트가 API를 호출할 수 있습니다. 변경 사항을 적용하려면 API를 배포해야 합니다. 자동 배포를 활성화하면 API에 대한 변경 사항이 자동으로 릴리스됩니다.

단계 변수

스테이지 변수는 HTTP API의 스테이지에 대해 정의할 수 있는 키-값 페어입니다. 환경 변수와 비슷한 역할을 하며, API 설정에 사용할 수 있습니다.

예를 들어 스테이지 변수를 정의한 다음, 해당 값을 HTTP 프록시 통합을 위한 HTTP 엔드포인트로서 설정할 수 있습니다. 또한 추후에 연결된 스테이지 변수 이름을 사용하여 엔드포인트를 참조할 수 있습니다. 이렇게 하면 각 스테이지의 서로 다른 엔드포인트에서 동일한 API 설정을 사용할 수 있습니다. 마찬가지로 스테이지 변수를 사용하여 API의 각 스테이지에 대해 서로 다른 AWS Lambda 함수 통합을 지정할 수 있습니다.

> **Note**
>
> 단계 변수는 자격 증명과 같은 중요한 데이터에 사용할 수 없습니다. 중요한 데이터를 통합에 전달하려면 AWS Lambda 권한 부여자를 사용합니다. Lambda 권한 부여자의 출력에서 중요한 데이터를 통합에 전달할 수 있습니다. 자세한 내용은 the section called “Lambda 권한 부여자 응답 형식” 단원을 참조하세요.

예시

스테이지 변수를 사용하여 HTTP 통합 엔드포인트를 사용자 지정하려면 먼저 스테이지(예: `url`)의 이름과 값을 `example.com`로 설정해야 합니다. 그런 다음 HTTP 프록시 통합을 설정합니다. 엔드포인트의 URL을 입력하는 대신, 단계 변수 값인 **`http://${stageVariables.url}`**을 사용하도록 API Gateway에 지시할 수 있습니다. 이 값은 API의 스테이지에 따라 실행 시간에 스테이지 변수 `${}`을 대체하도록 API Gateway에 지시합니다.

비슷한 방법으로 스테이지 변수를 참조하여 Lambda 함수 이름이나 AWS 역할 ARN을 지정할 수 있습니다.

Lambda 함수 이름을 단계 변수 값으로 지정할 때는 Lambda 함수에 대한 권한을 수동으로 구성해야 합니다. 이를 위해 AWS Command Line Interface(AWS CLI)를 사용할 수 있습니다.

```
aws lambda add-permission --function-name arn:aws:lambda:XXXXXX:your-lambda-function-
name --source-arn arn:aws:execute-api:us-east-1:YOUR_ACCOUNT_ID:api_id/*/HTTP_METHOD/
resource --principal apigateway.amazonaws.com --statement-id apigateway-access --action
 lambda:InvokeFunction
```

API Gateway 스테이지 변수 참조

HTTP 통합 URI

다음 예제에서 보듯 스테이지 변수를 HTTP 통합 URI의 일부로 사용할 수 있습니다.

- 프로토콜이 없는 전체 URI - `http://${stageVariables.<variable_name>}`

- 전체 도메인 - `http://${stageVariables.<variable_name>}/resource/operation`
- 하위 도메인 - `http://${stageVariables.<variable_name>}.example.com/resource/operation`
- 경로 - `http://example.com/${stageVariables.<variable_name>}/bar`
- 쿼리 문자열 - `http://example.com/foo?q=${stageVariables.<variable_name>}`

Lambda 함수

다음 예제와 같이 Lambda 함수 통합 이름이나 별칭 대신 스테이지 변수를 사용할 수 있습니다.

- `arn:aws:apigateway:<region>:lambda:path/2015-03-31/functions/arn:aws:lambda:<region>:<account_id>:function:${stageVariables.<function_variable_name>}/invocations`
- `arn:aws:apigateway:<region>:lambda:path/2015-03-31/functions/arn:aws:lambda:<region>:<account_id>:function:<function_name>:${stageVariables.<version_variable_name>}/invocations`

> **Note**
>
> Lambda 함수에 대해 단계 변수를 사용하려면 함수가 API와 동일한 계정에 있어야 합니다. 단계 변수는 교차 계정 Lambda 함수를 지원하지 않습니다.

AWS 통합 자격 증명

다음 예제와 같이 스테이지 변수를 AWS 사용자 또는 역할 자격 증명 ARN의 일부로 사용할 수 있습니다.

- `arn:aws:iam::<account_id>:${stageVariables.<variable_name>}`

HTTP API에 대한 보안 정책

API Gateway는 모든 HTTP API TLS_1_2 엔드포인트에 대한 보안 정책을 적용합니다.

보안 정책은 Amazon API Gateway에서 제공하는 최소 TLS 버전과 암호 제품군의 사전 정의된 조합입니다. TLS 프로토콜은 클라이언트와 서버 간의 변조 및 도청과 같은 네트워크 보안 문제를 해결합니다. 클라이언트가 사용자 지정의 도메인을 통해 API에 TLS 핸드셰이크를 설정하면 보안 정책은 클라이언

트가 선택할 수 있는 TLS 버전 및 암호 제품군 옵션을 적용합니다. 이 보안 정책은 TLS 1.2 및 TLS 1.3 트래픽은 허용하고 TLS 1.0 트래픽은 거부합니다.

HTTP API에 지원되는 TLS 프로토콜 및 암호

다음 표에서는 HTTP API에 지원되는 TLS 프로토콜 및 암호에 대해 설명합니다.

보안 정책	TLS_1_2
TLS 프로토콜	
TLSv1.3	♦
TLSv1.2	♦
TLS 암호	
TLS-AES-128-GCM-SHA256	♦
TLS-AES-256-GCM-SHA384	♦
TLS-CHACHA20-POLY1305-SHA256	♦
ECDHE-ECDSA-AES128-GCM-SHA256	♦
ECDHE-RSA-AES128-GCM-SHA256	♦
ECDHE-ECDSA-AES128-SHA256	♦
ECDHE-RSA-AES128-SHA256	♦
ECDHE-ECDSA-AES256-GCM-SHA384	♦
ECDHE-RSA-AES256-GCM-SHA384	♦
ECDHE-ECDSA-AES256-SHA384	♦
ECDHE-RSA-AES256-SHA384	♦
AES128-GCM-SHA256	♦
AES128-SHA256	♦

보안 정책	TLS_1_2
AES256-GCM-SHA384	♦
AES256-SHA256	♦

OpenSSL 및 RFC 암호 이름

OpenSSL과 IETF RFC 5246은 동일한 암호에 대해 서로 다른 이름을 사용합니다. 암호 이름 목록은 을 참조하십시오. the section called “OpenSSL 및 RFC 암호 이름”

REST API 및 API에 대한 정보 WebSocket

REST API 및 WebSocket API에 대한 자세한 내용은 및 을 참조하십시오. the section called “보안 정책 선택” the section called “ WebSocket API에 대한 보안 정책”

HTTP API에 대한 사용자 지정 도메인 이름 설정

사용자 지정 도메인 이름은 API 사용자에게 제공할 수 있는 더 간단하고 직관적인 URL입니다.

API를 배포한 후 사용자 및 사용자 고객은 다음 형식의 기본 URL을 사용하여 API를 호출할 수 있습니다.

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

여기서 `api-id` 는 API Gateway에서 생성하며, `region` (AWS 지역) 은 API를 생성할 때 사용자가 지정하고`stage`, API를 배포할 때 사용자가 지정합니다.

URL의 호스트 이름 부분(즉, *`api-id`*`.execute-api.`*`region`*`.amazonaws.com`)은 API 엔드포인트를 가리킵니다. 기본 API 엔드포인트는 기억하기가 어려우며 사용자에게 친숙하지 않을 수 있습니다.

사용자 지정 도메인 이름을 사용하면 API의 호스트 이름을 설정하고 기본 경로(예: `myservice`)를 선택하여 대체 URL을 API에 매핑할 수 있습니다. 예를 들어, 더 사용자 친화적인 API 기본 URL은 다음과 같습니다.

```
https://api.example.com/myservice
```

> **Note**
>
> 사용자 지정 도메인을 REST API 및 HTTP API에 연결할 수 있습니다. API Gateway 버전 2 API를 사용하여 REST API 및 HTTP API에 대한 리전 사용자 지정 도메인 이름을 생성하고 관리할 수 있습니다.
> HTTP API의 경우, TLS 1.2가 유일하게 지원되는 TLS 버전입니다.

도메인 이름 등록

API에 대한 사용자 지정 도메인 이름을 설정하려면 등록된 인터넷 도메인 이름이 있어야 합니다. 필요한 경우 Amazon Route 53를 사용하거나 사용자가 선택한 서드 파티 도메인 등록자를 사용하여 인터넷 도메인을 등록할 수 있습니다. API의 사용자 지정 도메인 이름은 하위 도메인의 이름이거나 등록된 인터넷 도메인의 루트 도메인("zone apex"라고도 함) 이름일 수 있습니다.

API Gateway에서 사용자 지정 도메인 이름을 생성한 후에는 DNS 공급자의 리소스 레코드를 생성하거나 업데이트하여 API 엔드포인트에 매핑해야 합니다. 이렇게 매핑하지 않으면 사용자 지정 도메인 이름이 목적지인 API 요청은 API Gateway에 도달할 수 없습니다.

리전 사용자 지정 도메인 이름

리전 API에 대해 사용자 지정 도메인 이름을 생성하면 API Gateway는 해당 API에 대해 리전 도메인 이름을 생성합니다. DNS 레코드를 설정하여 사용자 지정 도메인 이름이 리전 도메인 이름을 가리키도록 해야 합니다. 또한 사용자 지정 도메인 이름에 대한 인증서를 제출해야 합니다.

와일드카드 사용자 정의 도메인 이름

와일드카드 사용자 지정 도메인 이름을 사용하면 기본 할당량을 초과하지 않고 거의 무제한 수의 도메인 이름을 지원할 수 있습니다. 예를 들어 각 고객에게 고유한 도메인 이름인 *customername*`.api.example.com`을(를) 제공할 수 있습니다.

와일드카드 사용자 지정 도메인 이름을 생성하려면, 와일드카드(*)를 루트 도메인의 가능한 모든 하위 도메인을 나타내는 사용자 지정 도메인의 첫 번째 하위 도메인으로 지정할 수 있습니다.

예를 들어 와일드카드 사용자 정의 도메인 이름 `*.example.com`을 사용하면 `a.example.com`, `b.example.com` 및 `c.example.com` 같은 하위 도메인이 모두 동일한 도메인으로 라우팅됩니다.

와일드카드 사용자 지정 도메인 이름은 API Gateway의 표준 사용자 지정 도메인 이름과 별개의 구성을 지원합니다. 예를 들어 단일 AWS 계정에서 다르게 `*.example.com` 구성하고 `a.example.com` 작동하도록 할 수 있습니다.

와일드카드 사용자 지정 도메인 이름을 생성하려면 DNS 또는 이메일 검증 방법을 사용하여 검증한 ACM에서 발급한 인증서를 제공해야 합니다.

> **Note**
>
> 다른 AWS 계정에서 생성한 사용자 지정 도메인 이름이 와일드카드 사용자 지정 도메인 이름과 충돌하는 경우에는 와일드카드 사용자 지정 도메인 이름을 만들 수 없습니다. 예를 들어 계정 A에서 `a.example.com`가 생성된 경우, 계정 B는 와일드카드 사용자 지정 도메인 이름 `*.example.com`을 생성할 수 없습니다.
> 계정 A와 계정 B가 한 소유자를 공유하는 경우 AWS 지원 센터에 문의하여 예외를 요청할 수 있습니다.

사용자 지정 도메인 이름에 대한 인증서

> **Important**
>
> 사용자 지정 도메인 이름에 대한 인증서를 지정합니다. 애플리케이션이 인증서 고정 (SSL 피닝이라고도 함) 을 사용하여 ACM 인증서를 고정하는 경우 인증서를 갱신한 후 애플리케이션이 도메인에 연결하지 못할 수 있습니다. AWS 자세한 내용은 AWS Certificate Manager 사용 설명서의 인증서 고정 문제를 참조하세요.

ACM을 지원하는 리전에서 사용자 지정 도메인 이름에 대한 인증서를 제공하려면 ACM에 인증서를 요청해야 합니다. ACM을 지원하지 않는 리전에서 리전 사용자 지정 도메인 이름에 대한 인증서를 제공하려면 해당 리전의 API Gateway로 인증서를 가져와야 합니다.

SSL/TLS 인증서를 가져오려면 PEM 형식의 SSL/TLS 인증서 본문, 해당하는 프라이빗 키 및 사용자 지정 도메인 이름에 대한 인증서 체인을 제공해야 합니다. ACM에 저장된 각 인증서는 ARN으로 식별됩니다. 도메인 이름에 AWS 관리형 인증서를 사용하려면 해당 ARN을 참조하기만 하면 됩니다.

ACM을 사용하면 API에 대한 사용자 지정 도메인 이름을 간편하게 설정하고 사용할 수 있습니다. 지정된 도메인 이름에 대한 인증서를 생성하거나 인증서를 가져오고, API Gateway에서 ACM이 제공한 인증서의 ARN을 사용하여 도메인 이름을 설정한 다음, 사용자 지정 도메인 이름 아래의 기본 경로를 API의 배포된 단계에 매핑합니다. ACM에서 발행한 인증서를 사용하면 프라이빗 키 같은 민감한 인증서 세부 정보의 공개를 걱정할 필요가 없습니다.

사용자 지정 도메인 이름 설정에 대한 자세한 내용은 AWS Certificate Manager에서 인증서 준비하기 및 API Gateway에서 리전 사용자 지정 도메인 이름 설정 단원을 참조하십시오.

HTTP API에 대한 API 매핑 작업

API 매핑을 사용하여 API 스테이지를 사용자 지정 도메인 이름에 연결합니다. 도메인 이름을 생성하고 DNS 레코드를 구성한 후에는 API 매핑을 사용하여 사용자 지정 도메인 이름을 통해 API로 트래픽을 보냅니다.

API 매핑은 API, 스테이지 및 매핑에 사용할 경로(선택 사항)를 지정합니다. 예를 들어 API의 `production` 스테이지를 `https://api.example.com/orders`에 매핑할 수 있습니다.

HTTP 및 REST API 스테이지를 동일한 사용자 지정 도메인 이름에 매핑할 수 있습니다.

API 매핑을 생성하기 전에 API, 스테이지 및 사용자 지정 도메인 이름이 있어야 합니다. 사용자 지정 도메인 이름 생성에 대한 자세한 내용은 the section called “리전 사용자 지정 도메인 이름 설정” 단원을 참조하세요.

API 요청 라우팅

여러 수준(예: `orders/v1/items` 및 `orders/v2/items`)으로 API 매핑을 구성할 수 있습니다.

여러 수준 API 매핑의 경우 API Gateway는 일치하는 경로가 가장 긴 API 매핑으로 요청을 라우팅합니다. API Gateway는 호출할 API를 선택하기 위해 API 경로가 아닌 API 매핑에 대해 구성된 경로만 고려합니다. 요청과 일치하는 경로가 없으면 API Gateway는 빈 경로 (`none`)에 매핑한 API로 요청을 보냅니다.

여러 수준 API 매핑을 사용하는 사용자 지정 도메인 이름의 경우 API Gateway는 일치하는 경로가 가장 긴 API 매핑으로 요청을 라우팅합니다.

예를 들어 다음 API 매핑이 있는 사용자 지정 도메인 이름 `https://api.example.com`을(를) 살펴보겠습니다.

1. (`none`)이(가) API 1에 매핑됩니다.
2. `orders`이(가) API 2에 매핑됩니다.
3. `orders/v1/items`이(가) API 3에 매핑됩니다.
4. `orders/v2/items`이(가) API 4에 매핑됩니다.
5. `orders/v2/items/categories`이(가) API 5에 매핑됩니다.

요청	선택한 API	설명
`https://api.example.com/orders`	`API 2`	요청은 이 API 매핑과 정확히 일치합니다.
`https://api.example.com/orders/v1/items`	`API 3`	요청은 이 API 매핑과 정확히 일치합니다.
`https://api.example.com/orders/v2/items`	`API 4`	요청은 이 API 매핑과 정확히 일치합니다.
`https://api.example.com/orders/v1/items/123`	`API 3`	API Gateway는 일치하는 경로가 가장 긴 매핑을 선택합니다. 요청 끝에 있는 123은(는) 선택 항목에 영향을 주지 않습니다.
`https://api.example.com/orders/v2/items/categories/5`	`API 5`	API Gateway는 일치하는 경로가 가장 긴 매핑을 선택합니다.
`https://api.example.com/customers`	`API 1`	API Gateway는 빈 매핑을 캐치올(catch-all)로 사용합니다.
`https://api.example.com/ordersandmore`	`API 2`	API Gateway는 일치하는 접두사가 가장 긴 매핑을 선택합니다. 단일 수준 매핑으로 구성된 사용자 지정 도메인 이름(예: 단지 `https://api.example.com/orders` 및 `https://api.example.com/`)의 경우 `ordersandmore` 와 일치하는 경로가 없으므로 API Gateway는 `API 1`을 선택합니다.

제한 사항

- API 매핑에서 사용자 지정 도메인 이름과 매핑된 API는 동일한 AWS 계정에 있어야 합니다.
- API 매핑에는 문자, 숫자 및 문자 `$-_.+!*'()/`만 포함해야 합니다.
- API 매핑에서 경로의 최대 길이는 300자입니다.
- 각 도메인 이름에 대해 여러 수준의 API 매핑이 200개 있을 수 있습니다.
- TLS 1.2 보안 정책을 사용하여 HTTP API를 리전별 사용자 지정 도메인 이름에만 매핑할 수 있습니다.
- WebSocket API를 HTTP API 또는 REST API와 동일한 사용자 지정 도메인 이름에 매핑할 수 없습니다.

API 매핑 생성

API 매핑을 생성하려면 먼저 사용자 지정 도메인 이름, API 및 스테이지를 생성해야 합니다. 사용자 지정 도메인 이름 생성에 대한 자세한 내용은 the section called “리전 사용자 지정 도메인 이름 설정” 단원을 참조하세요.

예를 들어 모든 리소스를 생성하는 AWS Serverless Application Model 템플릿에 대해서는 GitHub의 SAM 세션을 참조하세요.

AWS Management Console

API 매핑 생성

1. https://console.aws.amazon.com/apigateway에서 API Gateway 콘솔에 로그인합니다.
2. 사용자 지정 도메인 이름을 선택합니다.
3. 이미 생성한 사용자 지정 도메인 이름을 선택합니다.
4. API 매핑을 선택합니다.
5. API 매핑 구성을 선택합니다.
6. 새 매핑 추가를 선택합니다.
7. API, 스테이지 및 경로(선택 사항)를 입력합니다.
8. 저장을 선택합니다.

AWS CLI

다음 AWS CLI 명령은 API 매핑을 생성합니다. 이 예에서 API Gateway는 지정된 API 및 스테이지로 `api.example.com/v1/orders` 요청을 보냅니다.

```
aws apigatewayv2 create-api-mapping \
    --domain-name api.example.com \
    --api-mapping-key v1/orders \
    --api-id a1b2c3d4 \
    --stage test
```

AWS CloudFormation

다음 AWS CloudFormation 예에서는 API 매핑을 생성합니다.

```
MyApiMapping:
  Type: 'AWS::ApiGatewayV2::ApiMapping'
  Properties:
    DomainName: api.example.com
    ApiMappingKey: 'orders/v2/items'
    ApiId: !Ref MyApi
    Stage: !Ref MyStage
```

HTTP API에 대한 기본 엔드포인트 비활성화

기본적으로 클라이언트는 API에 대해 API Gateway가 생성하는 `execute-api` 엔드포인트를 사용하여 API를 호출할 수 있습니다. 클라이언트가 사용자 지정 도메인 이름을 사용해야만 API에 액세스할 수 있도록 하려면 기본 `execute-api` 엔드포인트를 비활성화합니다.

> **Note**
>
> 기본 엔드포인트를 비활성화하면 API의 모든 스테이지에 영향이 미칩니다.

다음 AWS CLI 명령은 HTTP API의 기본 엔드포인트를 비활성화합니다.

```
aws apigatewayv2 update-api \
    --api-id abcdef123 \
```

```
    --disable-execute-api-endpoint
```

기본 엔드포인트를 비활성화한 후 자동 배포를 활성화하지 않는 한 변경 사항을 적용하려면 API를 배포해야 합니다.

다음 AWS CLI 명령은 배포를 생성합니다.

```
aws apigatewayv2 create-deployment \
    --api-id abcdef123 \
    --stage-name dev
```

HTTP API 보호

API Gateway는 악의적인 사용자나 트래픽 스파이크 같은 특정 위협으로부터 API를 보호할 수 있는 여러 가지 방법을 제공합니다. 제한 대상 설정 및 상호 TLS 활성화와 같은 전략을 사용하여 API를 보호할 수 있습니다. 이 섹션에서는 API Gateway를 사용하여 이러한 기능을 활성화하는 방법을 배울 수 있습니다.

주제

- HTTP API에 대한 요청 조절
- HTTP API에 대한 상호 TLS 인증 구성

HTTP API에 대한 요청 조절

API에 대한 제한을 구성하여 너무 많은 요청으로 인해 과부하되지 않도록 보호할 수 있습니다. 제한은 모두 최선의 방식으로 적용되며 보장된 요청 한도가 아닌 대상으로 간주해야 합니다.

API Gateway는 요청에 대해 토큰이 계산되는 토큰 버킷 알고리즘을 사용하여 API에 대한 요청을 제한합니다. 특히 API Gateway는 리전별로 계정의 모든 API에 대한 요청 제출 속도와 버스트를 검사합니다. 토큰 버킷 알고리즘에서 버스트는 이러한 제한의 사전 정의된 초과 실행을 허용할 수 있지만, 다른 요인으로도 제한을 초과할 수 있습니다.

요청 제출이 정상 상태의 요청 속도 및 버스트 제한을 초과할 경우 API Gateway에서 요청을 제한하기 시작합니다. 이 시점에서 클라이언트는 `429 Too Many Requests` 오류 응답을 받을 수 있습니다. 이러한 예외를 포착하면 클라이언트는 속도 제한 방식으로 실패한 요청을 다시 제출할 수 있습니다.

API 개발자는 개별 API 단계 또는 경로에 대한 목표 제한을 설정하여 계정의 모든 API에서 전반적인 성능을 향상시킬 수 있습니다.

리전별 계정 수준 조절

기본적으로 API Gateway는 리전별로 AWS 계정 내의 모든 API에서 안정적인 상태의 초당 요청(RPS)을 제한합니다. 또한 리전별로 AWS 계정 내 모든 API에 대해 버스트(즉, 최대 버킷 크기)를 제한합니다. API Gateway에서 버스트 제한은 API Gateway가 `429 Too Many Requests` 오류 응답을 반환하기 전에 수행할 동시 요청 제출의 최대 목표 수를 나타냅니다. 조절 할당량에 대한 자세한 내용은 할당량 및 중요 정보 단원을 참조하세요.

계정당 한도는 지정된 리전에 있는 계정의 모든 API에 적용됩니다. 계정 수준 속도 제한은 요청 시 늘릴 수 있습니다. 제한 시간이 더 짧고 페이로드가 더 작은 API를 사용하면 더 높은 제한이 가능합니다. 리전별로 계정 수준 조절 한도 증가를 요청하려면 AWS 지원 센터에 문의하시기 바랍니다. 자세한 내용은 할당량 및 중요 정보 섹션을 참조하세요. 이러한 제한은 AWS 제한 한도보다 높을 수 없습니다.

라우팅 수준 스로틀

라우팅 수준 스로틀을 설정해 API의 특정 단계 또는 개별 라우팅에 대해 계정 수준 요청 스로틀 한도를 재정의할 수 있습니다. 기본 경로 제한 한도는 계정 수준 속도 제한을 초과할 수 없습니다.

AWS CLI를 사용하여 라우팅 수준 스로틀을 구성할 수 있습니다. 다음 명령은 API의 지정된 단계 및 라우팅에 대한 사용자 지정 스로틀을 구성합니다.

```
aws apigatewayv2 update-stage \
    --api-id a1b2c3d4 \
    --stage-name dev \
    --route-settings '{"GET /pets":
{"ThrottlingBurstLimit":100,"ThrottlingRateLimit":2000}}'
```

HTTP API에 대한 상호 TLS 인증 구성

상호 TLS 인증에는 클라이언트와 서버 간의 양방향 인증이 필요합니다. 상호 TLS를 사용하는 경우 클라이언트가 API에 액세스하려면 자격 증명을 확인하기 위해 X.509 인증서를 제공해야 합니다. 상호 TLS는 사물 인터넷(IoT) 및 B2B 애플리케이션에 일반적으로 요구됩니다.

API Gateway가 지원하는 다른 권한 부여 및 인증 작업과 함께 상호 TLS를 사용할 수 있습니다. API Gateway는 클라이언트가 제공하는 인증서를 Lambda 권한 부여자와 백엔드 통합에 전달합니다.

> **Important**
>
> 기본적으로 클라이언트는 API에 대해 API Gateway가 생성하는 `execute-api` 엔드포인트를 사용하여 API를 호출할 수 있습니다. 클라이언트가 상호 TLS가 활성화된 사용자 지정 도메인

이름을 사용해야만 API에 액세스할 수 있도록 하려면 기본 `execute-api` 엔드포인트를 비활성화합니다. 자세한 내용은 the section called “기본 엔드포인트 비활성화” 단원을 참조하세요.

상호 TLS의 사전 조건

상호 TLS를 구성하려면 다음이 필요합니다.

- 사용자 지정 도메인 이름
- 사용자 지정 도메인 이름에 대해 AWS Certificate Manager에 구성된 하나 이상의 인증서
- Amazon S3에 구성 및 업로드된 트러스트 스토어

사용자 지정 도메인 이름

HTTP API에 상호 TLS를 활성화하려면 API에 대한 사용자 지정 도메인 이름을 구성해야 합니다. 사용자 지정 도메인 이름에 상호 TLS를 활성화한 다음 클라이언트에 사용자 지정 도메인 이름을 제공할 수 있습니다. 상호 TLS가 활성화된 사용자 지정 도메인 이름을 사용하여 API에 액세스하려면 클라이언트가 API 요청에서 신뢰할 수 있는 인증서를 제공해야 합니다. 추가 정보는 the section called “사용자 지정 도메인 이름” 단원에서 확인할 수 있습니다.

AWS Certificate Manager에서 발급된 인증서 사용

ACM에서 직접 공인 인증서를 요청하거나 공용 인증서 또는 자체 서명된 인증서를 가져올 수 있습니다. ACM에 인증서를 설정하려면 ACM으로 이동하세요. 인증서를 가져오려면 다음 단원을 계속 읽으세요.

가져온 인증서 또는 AWS Private Certificate Authority 인증서 사용

ACM으로 가져온 인증서 또는 AWS Private Certificate Authority에서 가져온 인증서를 상호 TLS와 함께 사용하려면 API Gateway에서 ACM이 발급한 `ownershipVerificationCertificate`가 필요합니다. 이 소유권 인증서는 도메인 이름을 사용할 수 있는 권한이 있는지 확인하는 데에만 사용됩니다. TLS 핸드셰이크에는 사용되지 않습니다. 아직 `ownershipVerificationCertificate`가 없으면 https://console.aws.amazon.com/acm/으로 이동하여 하나를 설정합니다.

도메인 이름의 수명 동안 이 인증서를 유효하게 유지해야 합니다. 인증서가 만료되고 자동 갱신이 실패하면 도메인 이름에 대한 모든 업데이트가 잠깁니다. 다른 변경 사항을 적용하기 전에 유효한 `ownershipVerificationCertificate`를 사용해 `ownershipVerificationCertificateArn`을 업데이트해야 합니다.

ownershipVerificationCertificate는 API Gateway의 다른 상호 TLS 도메인에 대한 서버 인증서로 사용할 수 없습니다. 인증서를 ACM으로 직접 다시 가져오는 경우 발급자는 동일하게 유지해야 합니다.

트러스트 스토어 구성

트러스트 스토어는 파일 확장자가 .pem인 텍스트 파일입니다. 인증 기관으로부터의 신뢰할 수 있는 인증서 목록입니다. 상호 TLS를 사용하려면 API에 액세스하도록 신뢰할 수 있는 X.509 인증서의 트러스트 스토어를 생성합니다.

발급 CA 인증서부터 루트 CA 인증서까지의 전체 신뢰 체인을 트러스트 스토어에 포함시켜야 합니다. API Gateway는 현재 신뢰 체인에 있는 모든 CA에서 발급한 클라이언트 인증서를 수락합니다. 공인 또는 사설 인증 기관의 인증서가 포함될 수 있습니다. 인증서의 체인 길이는 최대 4개가 될 수 있습니다. 자체 서명된 인증서를 제공할 수도 있습니다. 트러스트 스토어에서 지원되는 해싱 알고리즘은 다음과 같습니다.

- SHA-256 이상
- RSA-2048 이상
- ECDSA-256 이상

API Gateway에서는 여러 인증서 속성의 유효성을 검사합니다. Lambda 권한 부여자를 사용하여 클라이언트가 API를 호출할 때 인증서가 해지되었는지 확인하는 것과 같은 추가 검사를 수행할 수 있습니다. API Gateway는 다음 속성의 유효성을 검사합니다.

검증	설명
X.509 구문	인증서는 X.509 구문 요구 사항을 충족해야 합니다.
무결성	인증서 내용이 트러스트 스토어의 인증 기관에서 서명한 내용에서 변경되어서는 안 됩니다.
유효성	인증서의 유효 기간이 최신이어야 합니다.
이름 체인/키 체인	인증서의 이름과 주체는 끊어지지 않는 체인을 형성해야 합니다. 인증서의 체인 길이는 최대 4개가 될 수 있습니다.

단일 파일의 Amazon S3 버킷에 트러스트 스토어를 업로드합니다.

Example certificates.pem

```
-----BEGIN CERTIFICATE-----
<Certificate contents>
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
<Certificate contents>
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
<Certificate contents>
-----END CERTIFICATE-----
...
```

다음 AWS CLI 명령은 Amazon S3 버킷에 `certificates.pem`을 업로드합니다.

```
aws s3 cp certificates.pem s3://bucket-name
```

사용자 지정 도메인 이름에 대한 상호 TLS 구성

HTTP API에 대한 상호 TLS를 구성하려면 API에 대해 최소 TLS 버전이 1.2인 리전 사용자 지정 도메인 이름을 사용해야 합니다. 사용자 지정 도메인 이름 생성 및 구성에 대한 자세한 내용은 the section called “리전 사용자 지정 도메인 이름 설정” 단원을 참조하세요.

> **Note**
>
> 프라이빗 API에는 상호 TLS가 지원되지 않습니다.

트러스트 스토어를 Amazon S3에 업로드한 후 상호 TLS를 사용하도록 사용자 지정 도메인 이름을 구성할 수 있습니다. 다음 내용(슬래시 포함)을 터미널에 붙여 넣습니다.

```
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
```

도메인 이름을 생성한 후에는 API 작업에 대한 DNS 레코드와 기본 경로 매핑을 구성해야 합니다. 자세한 내용은 API Gateway에서 리전 사용자 지정 도메인 이름 설정 단원을 참조하십시오.

상호 TLS가 요구되는 사용자 지정 도메인 이름을 사용하여 API 호출

상호 TLS가 활성화된 API를 호출하려면 클라이언트가 API 요청에서 신뢰할 수 있는 인증서를 제공해야 합니다. 클라이언트가 API를 호출하려고 하면 API Gateway는 트러스트 스토어에서 클라이언트 인증서의 발급자를 찾습니다. API Gateway가 요청을 진행하려면 인증서의 발급자 및 루트 CA 인증서까지의 전체 신뢰 체인이 트러스트 스토어에 있어야 합니다.

다음 예제 `curl` 명령은 요청에 `api.example.com`,을 포함하여 `my-cert.pem`에 요청을 보냅니다. `my-key.key`는 인증서의 프라이빗 키입니다.

```
curl -v --key ./my-key.key --cert ./my-cert.pem api.example.com
```

트러스트 스토어가 인증서를 신뢰하는 경우에만 API가 호출됩니다. 다음 조건에서는 API Gateway가 TLS 핸드셰이크에 실패하고 `403` 상태 코드와 함께 요청을 거부합니다. 인증서가 다음 상태인 경우:

- 신뢰할 수 없음
- 만료됨
- 지원되는 알고리즘을 사용하지 않음

> **Note**
>
> API Gateway에서는 인증서가 해지되었는지 여부를 확인하지 않습니다.

트러스트 스토어 업데이트

트러스트 스토어의 인증서를 업데이트하려면 새 인증서 번들을 Amazon S3에 업로드합니다. 그런 다음 업데이트된 인증서를 사용하도록 사용자 지정 도메인 이름을 업데이트할 수 있습니다.

Amazon S3 버전 관리를 사용하여 트러스트 스토어의 여러 버전을 유지 관리합니다. 새 트러스트 스토어 버전을 사용하도록 사용자 지정 도메인 이름을 업데이트하면 인증서가 유효하지 않을 경우 API Gateway에서 경고가 반환됩니다.

API Gateway는 도메인 이름을 업데이트할 때만 인증서 경고를 생성합니다. API Gateway는 이전에 업로드한 인증서 만료를 알려주지 않습니다.

다음 AWS CLI 명령은 새 트러스트 스토어 버전을 사용하도록 사용자 지정 도메인 이름을 업데이트합니다.

```
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'
```

상호 TLS 비활성화

사용자 지정 도메인 이름에 상호 TLS를 사용하지 않도록 설정하려면 다음 명령과 같이 사용자 지정 도메인 이름에서 해당 트러스트 스토어를 제거합니다.

```
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=''
```

인증서 경고 문제 해결

상호 TLS가 포함된 사용자 지정 도메인 이름을 만들 때 트러스트 스토어의 인증서가 유효하지 않을 경우 API Gateway 에서 경고가 반환됩니다. 이는 새 트러스트 스토어를 사용하도록 사용자 지정 도메인 이름을 업데이트할 때에도 발생할 수 있습니다. 이러한 경고는 경고를 유발한 인증서와 인증서 주체에 문제가 있음을 나타냅니다. API에 대해 상호 TLS가 계속 활성화되어 있지만 일부 클라이언트는 API에 액세스하지 못할 수 있습니다.

경고를 유발한 인증서를 식별하려면 트러스트 스토어에서 인증서를 디코딩해야 합니다. `openssl` 같은 도구를 사용하여 인증서를 디코딩하고 해당 주체를 식별할 수 있습니다.

다음 명령은 주체를 포함하여 인증서의 내용을 표시합니다.

```
openssl x509 -in certificate.crt -text -noout
```

경고를 유발한 인증서를 업데이트하거나 제거한 다음 새 트러스트 스토어를 Amazon S3에 업로드합니다. 새 트러스트 스토어를 업로드한 후 새 트러스트 스토어를 사용하도록 사용자 지정 도메인 이름을 업데이트합니다.

도메인 이름 충돌 문제 해결

오류 `"The certificate subject <certSubject> conflicts with an existing certificate from a different issuer."`은(는) 여러 인증 기관이 이 도메인에 대한 인증서를

발급했음을 의미합니다. 인증서의 각 주체에 대해 API Gateway에는 상호 TLS 도메인에 대한 발급자가 하나만 있을 수 있습니다. 해당 주제에 대한 모든 인증서는 단일 발급자를 통해서만 받아야 합니다. 본인이 제어할 수 없지만 도메인 이름의 소유권을 증명할 수는 있는 인증서에 문제가 있는 경우 AWS Support에 문의를 눌러 티켓을 엽니다.

도메인 이름 상태 메시지 문제 해결

`PENDING_CERTIFICATE_REIMPORT`: 이것은 인증서를 ACM으로 다시 가져왔으며, 새 인증서에 SAN(주체 대체 이름)이 `ownershipVerificationCertificate`에 포함되지 않거나 인증서의 주체 또는 SAN이 도메인 이름을 포함하지 않기 때문에 유효성 검사에 실패했다는 의미입니다. 잘못 구성되었거나 잘못된 인증서를 가져왔을 수 있습니다. 유효한 인증서를 ACM으로 다시 가져와야 합니다. 검증에 대한 자세한 내용은 도메인 소유권 검증을 참조하세요.

`PENDING_OWNERSHIP_VERIFICATION`: 이것은 이전에 확인된 인증서가 만료되어 ACM이 인증서를 자동 갱신할 수 없다는 의미입니다. 인증서를 갱신하거나 새 인증서를 요청해야 합니다. 인증서 갱신에 대한 자세한 내용은 ACM의 관리형 인증서 갱신 문제 해결 가이드를 참조하세요.

HTTP API 모니터링

CloudWatch 지표와 CloudWatch Logs를 사용하여 HTTP API를 모니터링할 수 있습니다. 로그와 지표를 결합하여 오류를 기록하고 API의 성능을 모니터링할 수 있습니다.

> **Note**
>
> API Gateway는 다음과 같은 경우 로그 및 지표를 생성하지 않을 수 있습니다.
>
> - 413 요청 엔터티가 너무 큼 오류가 발생한 경우
> - 429개 초과로 요청이 너무 많음 오류가 발생한 경우
> - API 매핑이 없는 사용자 지정 도메인으로 보낸 요청에서 400개의 연속 오류가 발생한 경우
> - 내부 오류로 인해 500개의 연속 오류가 발생한 경우

주제

- HTTP API에 대한 지표 작업
- HTTP API의 로깅 구성

HTTP API에 대한 지표 작업

API Gateway에서 원시 데이터를 수집하고 읽기 가능한 실시간에 가까운 지표로 처리하는 CloudWatch를 사용하여 API 실행을 모니터링할 수 있습니다. 이러한 통계는 15개월 간 기록되므로 기록 정보에 액세스하고 웹 애플리케이션이나 서비스가 어떻게 수행되고 있는지 전체적으로 더 잘 파악할 수 있습니다. 기본적으로 API Gateway 지표 데이터는 1분 간격으로 CloudWatch로 자동 전송됩니다. 지표를 모니터링하려면 API용 CloudWatch 대시보드를 만드세요. CloudWatch 대시보드를 만드는 방법에 대한 자세한 내용은 Amazon CloudWatch 사용 설명서에서 CloudWatch 대시보드 생성을 참조하세요. 자세한 내용은 Amazon CloudWatch 사용 설명서의 Amazon CloudWatch란 무엇입니까?를 참조하십시오.

HTTP API에서는 다음과 같은 지표가 지원됩니다. 또한 세부 지표를 활성화하여 경로 수준 지표를 Amazon CloudWatch에 쓸 수도 있습니다.

측정치	설명
4xx	지정한 기간 내에 캡처된 클라이언트 측 오류 수
5xx	지정한 기간 내에 캡처된 서버 측 오류 수.
개수	지정된 기간 동안의 총 API 요청 수.
IntegrationLatency	API Gateway가 요청을 백엔드로 릴레이할 때부터 백엔드에서 응답을 수신할 때까지의 시간입니다.
Latency	API Gateway가 클라이언트에서 요청을 수신할 때부터 클라이언트에게 응답을 반환할 때까지의 시간입니다. 지연 시간에는 통합 지연 시간과 기타 API Gateway 오버헤드가 포함됩니다.

측정치	설명
DataProcessed	처리된 데이터의 양(바이트)입니다.

다음 표의 차원을 사용하여 API Gateway 지표를 필터링할 수 있습니다.

차원	설명
ApiId	지정한 API ID를 사용하여 API에 대한 API Gateway 지표를 필터링합니다.
ApiId, 스테이지	지정한 API ID와 스테이지 ID를 사용하여 API 스테이지에 대한 API Gateway 지표를 필터링합니다.
ApiId, 스테이지, 라우팅	지정한 API ID, 스테이지 ID 및 라우팅 ID를 사용하여 API 메서드에 대한 API Gateway 지표를 필터링합니다. 사용자가 세부 CloudWatch 지표를 명시적으로 활성화하지 않으면 API Gateway는 이러한 지표를 전송하지 않습니다. API Gateway V2 REST API의 UpdateStage 작업을 호출하여 `detailedMetricsEnabled` 속성을 `true`로 업데이트하면 됩니다. 또는 update-stage AWS CLI 명령을 호출하여 `DetailedMetricsEnabled` 속성을 `true`로 업데이트할 수 있습니다. 이러한 지표를 활성화할 경우 계정에 추

차원	설명
	가 비용이 발생합니다. 요금에 대한 자세한 내용은 Amazon CloudWatch 요금을 참조하십시오.

HTTP API의 로깅 구성

로깅을 활성화하여 CloudWatch Logs에 로그를 기록할 수 있습니다. 로깅 변수를 사용하여 로그의 내용을 사용자 지정할 수 있습니다.

HTTP API에 대한 로깅을 활성화하려면 다음을 수행해야 합니다.

1. 사용자가 로깅을 활성화하는 데 필요한 권한을 가지고 있는지 확인합니다.
2. CloudWatch Logs 로그 그룹을 생성합니다.
3. API 단계에 대한 CloudWatch Logs 로그 그룹의 ARN을 제공합니다.

로깅을 활성화하는 권한

API에 대한 로깅을 활성화하려면 사용자는 다음과 같은 권한을 가지고 있어야 합니다.

Example

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogGroups",
                "logs:DescribeLogStreams",
                "logs:GetLogEvents",
                "logs:FilterLogEvents"
            ],
            "Resource": "arn:aws:logs:us-east-2:123456789012:log-group:*"
        },
        {
            "Effect": "Allow",
            "Action": [
```

```
                "logs:CreateLogDelivery",
                "logs:PutResourcePolicy",
                "logs:UpdateLogDelivery",
                "logs:DeleteLogDelivery",
                "logs:CreateLogGroup",
                "logs:DescribeResourcePolicies",
                "logs:GetLogDelivery",
                "logs:ListLogDeliveries"
            ],
            "Resource": "*"
        }
    ]
}
```

로그 그룹 생성 및 HTTP API에 대한 로깅 활성화

AWS Management Console 또는 AWS CLI를 사용하여 로그 그룹을 생성하고 액세스 로깅을 활성화할 수 있습니다.

AWS Management Console

1. 로그 그룹 생성

   콘솔을 사용하여 로그 그룹을 생성하는 방법을 알아보려면 Amazon CloudWatch Logs 사용 설명서의 로그 그룹 생성을 참조하세요.
2. https://console.aws.amazon.com/apigateway에서 API Gateway 콘솔에 로그인합니다.
3. HTTP API를 선택합니다.
4. 기본 탐색 패널의 Monitor(모니터링) 탭에서 Logging(로깅) 선택합니다.
5. 로깅을 활성화할 단계를 선택하고 Select(선택)를 선택합니다.
6. Edit(편집)을 선택하여 액세스 로깅을 활성화합니다.
7. Access logging(액세스 로깅)을 활성화하고 CloudWatch Logs를 입력한 다음 로그 형식을 선택합니다.
8. Save를 선택합니다.

AWS CLI

다음 AWS CLI 명령은 로그 그룹을 생성합니다.

```
aws logs create-log-group --log-group-name my-log-group
```

로깅을 활성화하려면 로그 그룹의 Amazon 리소스 이름(ARN)이 필요합니다. ARN 형식은 arn:aws:logs:*region*: *account-id*:log-group:*log-group-name*입니다.

다음 AWS CLI 명령은 HTTP API의 `$default` 단계에 대한 로깅을 활성화합니다.

```
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"}'
```

로그 형식 예

일반적으로 사용되는 몇 가지 액세스 로그 형식의 예제가 API Gateway 콘솔에 표시되며 다음과 같이 나열됩니다.

- CLF(Common Log Format):

```
$context.identity.sourceIp - - [$context.requestTime] "$context.httpMethod
 $context.routeKey $context.protocol" $context.status $context.responseLength
 $context.requestId $context.extendedRequestId
```

- JSON:

```
{ "requestId":"$context.requestId", "ip": "$context.identity.sourceIp",
 "requestTime":"$context.requestTime",
 "httpMethod":"$context.httpMethod","routeKey":"$context.routeKey",
 "status":"$context.status","protocol":"$context.protocol",
 "responseLength":"$context.responseLength", "extendedRequestId":
 "$context.extendedRequestId" }
```

- XML:

```
<request id="$context.requestId"> <ip>$context.identity.sourceIp</ip> <requestTime>
$context.requestTime</requestTime> <httpMethod>$context.httpMethod</httpMethod>
 <routeKey>$context.routeKey</routeKey> <status>$context.status</status> <protocol>
```

```
$context.protocol</protocol> <responseLength>$context.responseLength</responseLength>
 <extendedRequestId>$context.extendedRequestId</extendedRequestId> </request>
```

- CSV(쉼표로 분리된 값):

```
$context.identity.sourceIp,$context.requestTime,$context.httpMethod,
$context.routeKey,$context.protocol,$context.status,$context.responseLength,
$context.requestId,$context.extendedRequestId
```

HTTP API 액세스 로그 사용자 정의

다음 변수를 사용하여 HTTP API 액세스 로그를 사용자 지정할 수 있습니다. HTTP API의 액세스 로그에 대해 자세히 알아보려면 HTTP API의 로깅 구성 단원을 참조하십시오.

파라미터	설명
`$context.accountId`	API 소유자의 AWS 계정 ID.
`$context.apiId`	식별자 API Gateway가 API에 할당합니다.
`$context.authorizer.claims.`*`property`*	메서드 호출자가 성공적으로 인증된 후 JSON 웹 토큰 (JWT) 에서 반환되는 클레임의 속성 (예:). `$context.authorizer.claims.username` 자세한 내용은 JWT 권한 부여자를 사용하여 HTTP API에 대한 액세스 제어 단원을 참조하세요. **Note** `$context.authorizer.claims` 를 호출하면 null이 반환됩니다.
`$context.authorizer.error`	권한 부여자로부터 반환된 오류 메시지입니다.
`$context.authorizer.principalId`	Lambda 권한 부여자가 반환하는 보안 주체 사용자 자격 증명입니다.

파라미터	설명
`$context.authorizer.` *property*	API Gateway Lambda 권한 부여자 함수에서 반환된 `context` 맵의 지정된 키-값 페어의 값입니다. 예를 들어, 권한 부여자는 다음 `context` 맵을 반환합니다. `"context" : { "key": "value", "numKey": 1, "boolKey": true }` `$context.authorizer.key` 를 호출하면 `"value"` 문자열이 반환되고, `$context.authorizer.numKey` 를 호출하면 `1`이 반환되고, `$context.authorizer.boolKey` 를 호출하면 `true`가 반환됩니다.
`$context.awsEndpointRequestId`	또는 AWS 헤더에 있는 엔드포인트의 요청 ID. `x-amz-request-id` `x-amzn-requestId`
`$context.awsEndpointRequestId2`	`x-amz-id-2` 헤더에 있는 AWS 엔드포인트의 요청 ID.
`$context.customDomain.basePathMatched`	들어오는 요청이 일치하는 API 매핑의 경로입니다. 클라이언트가 사용자 지정 도메인 이름을 사용하여 API에 액세스할 때 적용할 수 있습니다. 예를 들어 클라이언트가 `https://api.example.com/v1/orders/1234` 에 요청을 보내고 요청이 API 매핑을 경로 `v1/orders` 와(과) 일치시키는 경우 값은 `v1/orders` 입니다. 자세한 내용은 the section called “API 매핑” 단원을 참조하십시오.
`$context.dataProcessed`	처리된 데이터의 양(바이트)입니다.

파라미터	설명
$context.domainName	API를 호출하는 데 사용되는 정식 도메인 이름입니다. 이 이름은 수신되는 Host 헤더와 동일해야 합니다.
$context.domainPrefix	$context.domainName 의 첫 번째 레이블입니다.
$context.error.message	API Gateway 오류 메시지가 포함된 문자열입니다.
$context.error.messageString	$context.error.message 의 따옴표 붙은 값, 즉 "$context.error.message" 입니다.
$context.error.responseType	GatewayResponse 의 한 유형입니다. 자세한 내용은 the section called “지표” 및 the section called “오류 응답을 사용자 지정하도록 게이트웨이 응답 설정” 단원을 참조하십시오.
$context.extendedRequestId	$context.requestId 와 동일합니다.
$context.httpMethod	사용된 HTTP 메서드입니다. 유효한 값에는 DELETE, GET, HEAD, OPTIONS, PATCH, POST 및 PUT이 있습니다.
$context.identity.accountId	요청과 관련된 AWS 계정 ID. IAM 권한 부여를 사용하는 라우팅에 대해 지원됩니다.
$context.identity.caller	요청에 서명한 호출자의 보안 주체 ID입니다. IAM 권한 부여를 사용하는 라우팅에 대해 지원됩니다.

파라미터	설명
`$context.identity.cognitoAuthenticationProvider`	요청을 작성한 호출자가 사용한 Amazon Cognito 인증 공급자의 쉼표로 구분된 목록입니다. Amazon Cognito 자격 증명으로 요청을 서명한 경우에만 사용할 수 있습니다. 예를 들어, Amazon Cognito 사용자 풀의 자격 증명의 경우 `cognito-idp.`*`region`*`.amazonaws.com/`*`user_pool_id`*`,cognito-idp.`*`region`*`.amazonaws.com/`*`user_pool_id`*`:CognitoSignIn:`*`token subject claim`* 자세한 내용은 Amazon Cognito 개발자 안내서의 연동 자격 증명 사용을 참조하세요.
`$context.identity.cognitoAuthenticationType`	요청을 작성한 호출자의 Amazon Cognito 인증 유형입니다. Amazon Cognito 자격 증명으로 요청을 서명한 경우에만 사용할 수 있습니다. 가능한 값에는 인증 자격 증명에 대한 `authenticated` 및 미인증 자격 증명에 대한 `unauthenticated` 이(가) 포함됩니다.
`$context.identity.cognitoIdentityId`	요청을 작성한 호출자의 Amazon Cognito 자격 증명 ID입니다. Amazon Cognito 자격 증명으로 요청을 서명한 경우에만 사용할 수 있습니다.
`$context.identity.cognitoIdentityPoolId`	요청을 작성한 호출자의 Amazon Cognito 자격 증명 풀 ID입니다. Amazon Cognito 자격 증명으로 요청을 서명한 경우에만 사용할 수 있습니다.
`$context.identity.principalOrgId`	AWS 조직 ID입니다. IAM 권한 부여를 사용하는 라우팅에 대해 지원됩니다.

파라미터	설명
`$context.identity.clientCert.clientCertPem`	상호 TLS 인증 시 클라이언트가 제공한 PEM 인코딩된 클라이언트 인증서입니다. 클라이언트에서 상호 TLS가 활성화된 사용자 지정 도메인 이름을 사용하여 API에 액세스할 때 제공됩니다.
`$context.identity.clientCert.subjectDN`	클라이언트가 제공하는 인증서의 주체가 갖는 고유 이름입니다. 클라이언트에서 상호 TLS가 활성화된 사용자 지정 도메인 이름을 사용하여 API에 액세스할 때 제공됩니다.
`$context.identity.clientCert.issuerDN`	클라이언트가 제공하는 인증서의 발행자가 갖는 고유 이름입니다. 클라이언트에서 상호 TLS가 활성화된 사용자 지정 도메인 이름을 사용하여 API에 액세스할 때 제공됩니다.
`$context.identity.clientCert.serialNumber`	인증서의 일련 번호입니다. 클라이언트에서 상호 TLS가 활성화된 사용자 지정 도메인 이름을 사용하여 API에 액세스할 때 제공됩니다.
`$context.identity.clientCert.validity.notBefore`	인증서의 유효 기간이 시작되는 날짜입니다. 클라이언트에서 상호 TLS가 활성화된 사용자 지정 도메인 이름을 사용하여 API에 액세스할 때 제공됩니다.
`$context.identity.clientCert.validity.notAfter`	인증서의 유효 기간이 끝나는 날짜입니다. 클라이언트에서 상호 TLS가 활성화된 사용자 지정 도메인 이름을 사용하여 API에 액세스할 때 제공됩니다.
`$context.identity.sourceIp`	API Gateway 엔드포인트를 요청하는 즉시 TCP 연결의 소스 IP 주소입니다.
`$context.identity.user`	리소스 액세스에 대해 권한을 부여할 사용자의 보안 주체 ID입니다. IAM 권한 부여를 사용하는 라우팅에 대해 지원됩니다.
`$context.identity.userAgent`	API 호출자의 `User-Agent` 헤더입니다.

파라미터	설명
$context.identity.userArn	인증 후 식별된 실제 사용자의 ARN(Amazon Resource Name)입니다. IAM 권한 부여를 사용하는 라우팅에 대해 지원됩니다. 자세한 내용은 https://docs.aws.amazon.com/IAM/latest/UserGuide/id_users.html 단원을 참조하세요.
$context.integration.error	통합에서 반환된 오류 메시지입니다. $context.integrationErrorMessage 와 동일합니다.
$context.integration.integrationStatus	Lambda 프록시 통합의 경우, 백엔드 Lambda 함수 코드가 아닌 AWS Lambda에서 상태 코드가 반환되었습니다.
$context.integration.latency	통합 지연 시간(ms)입니다. $context.integrationLatency 와 동일합니다.
$context.integration.requestId	엔드포인트의 요청 ID. AWS $context.awsEndpointRequestId 와 동일합니다.
$context.integration.status	통합에서 반환된 상태 코드입니다. Lambda 프록시 통합의 경우 Lambda 함수 코드에서 반환된 상태 코드입니다.
$context.integrationErrorMessage	통합 오류 메시지가 포함된 문자열입니다.
$context.integrationLatency	통합 지연 시간(ms)입니다.
$context.integrationStatus	Lambda 프록시 통합의 경우 이 파라미터는 백엔드 Lambda 함수가 아니라 반환된 AWS Lambda상태 코드를 나타냅니다.
$context.path	요청 경로입니다. 예: /{stage}/root/child .

파라미터	설명
`$context.protocol`	요청 프로토콜입니다(예: `HTTP/1.1`). **Note** API Gateway API는 HTTP/2 요청을 수락할 수 있지만 API Gateway는 HTTP/1.1을 사용하여 백엔드 통합에 요청을 보냅니다. 따라서 클라이언트가 HTTP/2를 사용하는 요청을 보내더라도 요청 프로토콜은 HTTP/1.1로 기록됩니다.
`$context.requestId`	API Gateway가 API 요청에 할당하는 ID입니다.
`$context.requestTime`	CLF 형식 요청 시간(`dd/MMM/yyyy:HH:mm:ss +-hhmm`)입니다.
`$context.requestTimeEpoch`	Epoch 형식 요청 시간입니다.
`$context.responseLatency`	응답 지연 시간(ms)입니다.
`$context.responseLength`	바이트로 된 응답 페이로드 길이입니다.
`$context.routeKey`	API 요청의 경로 키입니다(예: `/pets`).
`$context.stage`	API 요청의 개발 단계입니다(예: `beta` 또는 `prod`).
`$context.status`	메서드 응답 상태입니다.

HTTP API 관련 문제 해결

다음 주제에서는 HTTP API를 사용할 때 발생할 수 있는 오류 및 문제에 대한 문제 해결 조언을 제공합니다.

주제

- HTTP API Lambda 통합 관련 문제 해결
- HTTP API JWT 권한 부여자 관련 문제 해결

HTTP API Lambda 통합 관련 문제 해결

다음은 AWS Lambda 통합를 HTTP API와 함께 사용할 때 발생할 수 있는 오류 및 문제에 대한 문제 해결 조언을 제공합니다.

문제: Lambda 통합이 있는 내 API가 **{"message":"Internal Server Error"}**를 반환합니다.

내부 서버 오류 문제를 해결하려면 로그 형식에 `$context.integrationErrorMessage` 로깅 변수를 추가하고 HTTP API의 로그를 봅니다. 이를 위해 다음을 수행합니다.

를 사용하여 로그 그룹을 만들려면 AWS Management Console

1. https://console.aws.amazon.com/cloudwatch/ 에서 CloudWatch 콘솔을 엽니다.
2. 로그 그룹을 선택합니다.
3. 로그 그룹 생성을 선택합니다.
4. 로그 그룹 이름을 입력한 다음 생성을 선택합니다.
5. 로그 그룹의 Amazon 리소스 이름(ARN)을 기록해 둡니다. *ARN ### arn:aws:logs: ##: ## ID:log-group:###. log-group-name* HTTP API에 대한 액세스 로깅을 활성화하려면 로그 그룹 ARN이 필요합니다.

`$context.integrationErrorMessage` 로깅 변수를 추가하려면

1. https://console.aws.amazon.com/apigateway에서 API Gateway 콘솔에 로그인합니다.
2. HTTP API를 선택합니다.
3. 모니터에서 로깅을 선택합니다.
4. API의 단계를 선택합니다.
5. 편집을 선택한 다음 액세스 로깅을 활성화합니다.
6. 로그 대상에 이전 단계에서 생성한 로그 그룹의 ARN을 입력합니다.
7. 로그 형식에서 CLF를 선택합니다. API Gateway는 예제 로그 형식을 생성합니다.
8. 로그 형식의 끝에 `$context.integrationErrorMessage`를 추가합니다.

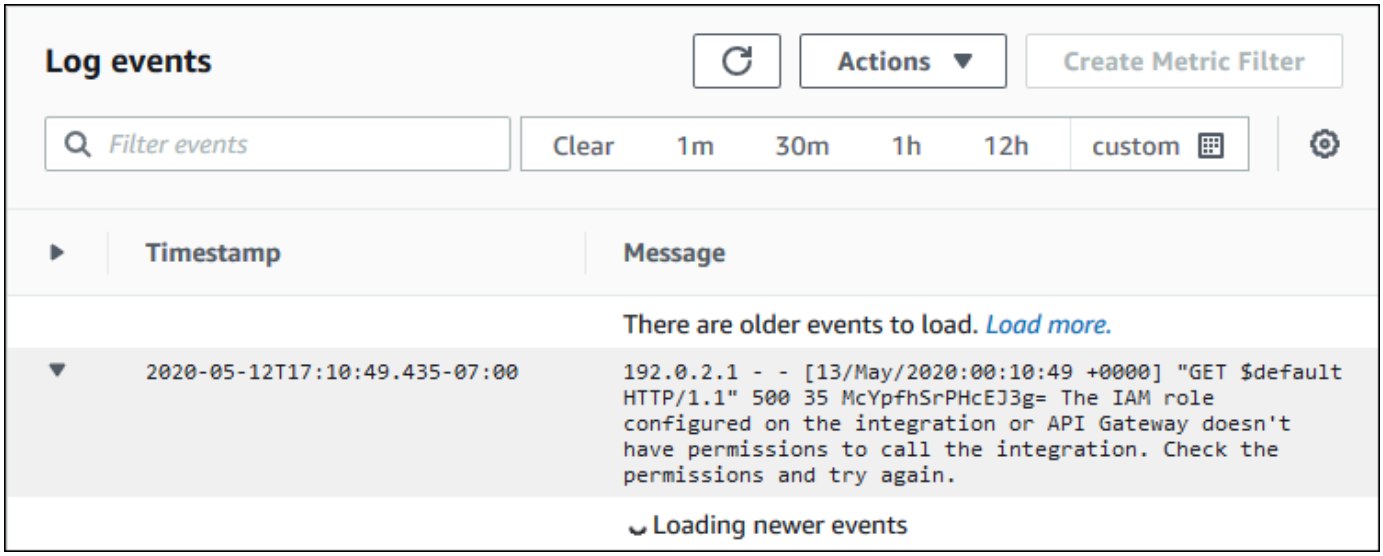
9. 저장을 선택합니다.

API 로그를 보려면

1. 로그를 생성합니다. 브라우저 또는 curl을 사용하여 API를 호출합니다.

```
$curl https://api-id.execute-api.us-west-2.amazonaws.com/route
```

2. https://console.aws.amazon.com/apigateway에서 API Gateway 콘솔에 로그인합니다.
3. HTTP API를 선택합니다.
4. 모니터에서 로깅을 선택합니다.
5. 로깅을 활성화한 API의 단계를 선택합니다.
6. [로그인 보기] 를 선택합니다 CloudWatch.
7. 최신 로그 스트림을 선택하여 HTTP API의 로그를 확인합니다.
8. 로그 항목은 다음과 같아야 합니다.



$context.integrationErrorMessage가 로그 형식에 추가되었으므로 문제를 요약해서 보여주는 오류 메시지가 로그에 표시됩니다.

Lambda 함수 코드에 문제가 있음을 나타내는 다른 오류 메시지가 로그에 포함될 수 있습니다. 이 경우 Lambda 함수 코드를 확인하고 Lambda 함수가 필수 형식으로 응답을 반환하는지 확인합니다. 로그에 오류 메시지가 포함되어 있지 않은 경우 $context.error.message 및 $context.error.responseType를 로그 형식에 추가하여 문제를 해결하는 데 도움이 되는 자세한 내용을 확인하세요.

이 경우 API Gateway에 Lambda 함수를 호출하는 데 필요한 권한이 없는 것으로 로그에 표시됩니다.

API Gateway 콘솔에서 Lambda 통합을 생성하면 API Gateway가 Lambda 함수를 호출할 권한을 자동으로 구성합니다. AWS CloudFormation, 또는 SDK를 사용하여 AWS CLI Lambda 통합을 생성할 때는 API Gateway가 함수를 호출할 수 있는 권한을 부여해야 합니다. 다음 예제 AWS CLI 명령은 다양한 HTTP API 경로에 Lambda 함수를 호출할 수 있는 권한을 부여합니다.

Example 예 - HTTP API의 **$default** 스테이지 및 **$default** 라우트에 대해

```
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 예 — HTTP API의 **prod** 스테이지 및 **test** 라우트에 대해

```
aws lambda add-permission \
    --function-name my-function \
    --statement-id apigateway-invoke-permissions \
    --action lambda:InvokeFunction \
    --principal apigateway.amazonaws.com \
    --source-arn "arn:aws:execute-api:us-west-2:123456789012:api-id/prod/*/test"
```

Lambda 콘솔의 권한 탭에서 함수 정책을 확인합니다.

API를 다시 호출해 보십시오. Lambda 함수의 응답이 표시되어야 합니다.

HTTP API JWT 권한 부여자 관련 문제 해결

다음은 HTTP API에서 JSON Web Token(JWT) 권한 부여자를 사용할 때 발생할 수 있는 오류 및 문제에 대한 문제 해결 조언을 제공합니다.

문제: 내 API가 **401 {"message":"Unauthorized"}**를 반환합니다.

API의 응답에서 `www-authenticate` 헤더를 확인합니다.

다음 명령은 `curl`을 사용하여 `$request.header.Authorization`을 ID 소스로 사용하는 JWT 권한 부여자를 통해 API에 요청을 보냅니다.

```
$curl -v -H "Authorization: token" https://api-id.execute-api.us-
west-2.amazonaws.com/route
```

API의 응답에는 www-authenticate 헤더가 포함됩니다.

```
...
< 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"}}
```

이 경우 www-authenticate 헤더는 유효한 대상에게 토큰이 발급되지 않았음을 나타냅니다. API Gateway에서 요청을 승인하려면 JWT의 aud 또는 client_id 클레임이 권한 부여자에 대해 구성된 대상 항목 중 하나와 일치해야 합니다. API aud Gateway는 존재하지 않는 client_id 경우에만 유효성을 검사합니다. aud과 client_id 가 모두 있는 경우 API Gateway는 평가합니다aud.

JWT를 디코딩하고 API에 필요한 발행자, 대상 및 범위와 일치하는지 확인할 수도 있습니다. 웹사이트 jwt.io는 브라우저에서 JWT를 디버깅할 수 있습니다. OpenID Foundation에는 JWT 작업을 위한 라이브러리 목록이 포함되어 있습니다.

JWT 권한 부여자에 대한 자세한 내용은 JWT 권한 부여자를 사용하여 HTTP API에 대한 액세스 제어 단원을 참조하세요.

WebSocket API 작업

API Gateway의 WebSocket API는 백엔드 HTTP 엔드포인트, Lambda 함수 또는 기타 AWS 서비스에 통합되어 있는 WebSocket 경로 모음입니다. API Gateway 기능을 사용하여 생성에서 프로덕션 API 모니터링에 이르기까지 API 수명 주기의 모든 측면을 지원할 수 있습니다.

API Gateway WebSocket API는 양방향입니다. 클라이언트는 서비스에 메시지를 보낼 수 있고, 서비스는 독립적으로 클라이언트에 메시지를 보낼 수 있습니다. 이러한 양방향 동작 덕분에 클라이언트가 명시적 요청을 하지 않아도 클라이언트에 데이터를 푸시 할 수 있으므로 풍부한 클라이언트/서비스 상호작용이 가능합니다. WebSocket API는 채팅 애플리케이션, 공동 작업 플랫폼, 멀티 플레이어 게임, 금융 거래 플랫폼과 같은 실시간 애플리케이션에서 자주 사용됩니다.

시작하는 데 활용할 예제 앱은 자습서: WebSocket API, Lambda 및 DynamoDB를 사용하여 서버리스 채팅 앱 구축 섹션을 참조하세요.

이 단원에서는 API Gateway를 사용하여 WebSocket API를 개발, 게시, 보호 및 모니터링하는 방법을 배울 수 있습니다.

주제

- API Gateway의 WebSocket API에 대한 정보
- API Gateway에서 WebSocket API 개발
- 고객이 호출할 수 WebSocket 있는 API 게시
- WebSocket API 보호
- WebSocket API 모니터링

API Gateway의 WebSocket API에 대한 정보

API Gateway에서는 AWS 서비스 (예: Lambda 또는 DynamoDB) 를 위한 스테이트풀 프론트엔드 또는 HTTP 엔드포인트로 WebSocket API를 생성할 수 있습니다. WebSocket API는 클라이언트 앱으로부터 수신한 메시지 내용을 기반으로 백엔드를 호출합니다.

요청을 수신하고 이에 응답하는 REST API와 달리 WebSocket API는 클라이언트 앱과 백엔드 간의 양방향 통신을 지원합니다. 백엔드는 연결된 클라이언트로 콜백 메시지를 보낼 수 있습니다.

WebSocket API에서 수신되는 JSON 메시지는 구성한 경로를 기반으로 백엔드 통합으로 전달됩니다. (비 JSON 메시지는 사용자가 구성한 `$default` 라우팅으로 전달됩니다.)

라우팅에는 라우팅 선택 표현식을 평가할 경우 예상되는 값인 라우팅 키가 포함되어 있습니다. `routeSelectionExpression`은 API 수준에서 정의되는 속성입니다. 메시지 페이로드에 존재할 것으로 예상되는 JSON 속성을 지정합니다. 라우팅 선택 표현식에 대한 자세한 내용은 the section called "" 단원을 참조하십시오.

예를 들어 JSON 메시지에 `action` 속성이 있고 이 속성을 기반으로 다른 작업을 수행하려는 경우 라우팅 선택 표현식은 `${request.body.action}`일 수 있습니다. 라우팅 테이블은 사용자가 테이블에 정의한 사용자 지정 라우팅 키 값에 따라 `action` 속성 값을 일치시켜 어떤 작업을 수행할 것인지 지정할 것입니다.

`$connect`, `$disconnect`, `$default` 등 세 가지 사전 정의된 라우팅을 사용할 수 있습니다. 또한 사용자 지정 라우팅을 생성할 수 있습니다.

- API Gateway는 클라이언트와 WebSocket API 간의 지속적인 연결이 시작될 때 `$connect` 경로를 호출합니다.
- API Gateway는 클라이언트 또는 서버가 API와의 연결을 끊을 때 `$disconnect` 라우팅을 호출합니다.
- 일치하는 라우팅이 발견되면 메시지를 기준으로 라우팅 선택 표현식이 평가된 후 API Gateway가 사용자 지정 라우팅을 호출합니다. 일치 여부에 따라 어떤 통합이 호출될지 결정됩니다.
- 일치하는 라우팅이 없거나 메시지를 기준으로 라우팅 선택 표현식을 평가할 수 없는 경우 API Gateway가 `$default` 라우팅을 호출합니다.

`$connect` 및 `$disconnect` 라우팅에 대한 자세한 내용은 the section called "연결된 사용자 및 클라이언트 앱 관리"를 참조하십시오.

`$default` 라우팅 및 사용자 지정 라우팅에 대한 자세한 내용은 the section called "백엔드 통합 호출"를 참조하십시오.

백엔드 서비스는 연결된 클라이언트 앱에 데이터를 보낼 수 있습니다. 자세한 내용은 the section called "백엔드 서비스에서 연결된 클라이언트로 데이터 전송" 단원을 참조하세요.

연결된 사용자 및 클라이언트 앱 관리: **`$connect`** 및 **`$disconnect`** 라우팅

주제

- $connect 라우팅
- $connect 라우팅에서 연결 정보 전달
- $disconnect 라우팅

$connect 라우팅

클라이언트 앱은 WebSocket 업그레이드 요청을 전송하여 WebSocket API에 연결합니다. 요청이 성공하면 연결이 설정되는 동안 $connect 라우팅이 실행됩니다.

WebSocket 연결은 상태 저장 연결이므로 $connect 경로에서만 인증을 구성할 수 있습니다. AuthNAuthZ/는 연결 시에만 수행됩니다.

$connect 라우팅과 연결된 통합 실행이 완료될 때까지 업그레이드 요청은 대기 중이고 실제 연결이 설정되지 않습니다. $connect 요청이 실패하면(예: AuthN/AuthZ 실패 또는 통합 실패로 인해) 연결이 이루어지지 않습니다.

> **Note**
>
> $connect에서 권한 부여에 실패하면 연결이 설정되지 않고 클라이언트는 401 또는 403 응답을 받게 됩니다.

$connect에 대한 통합 설정은 선택 사항입니다. 다음의 경우 $connect 통합 설정을 고려해야 합니다.

- 클라이언트가 Sec-WebSocket-Protocol 필드를 사용하여 서브프로토콜을 지정할 수 있도록 하고 싶습니다. 예제 코드는 WebSocket 서브프로토콜이 필요한 $connect 라우팅 설정 단원을 참조하세요.
- 클라이언트가 연결될 때 알림을 받고 싶습니다.
- 연결을 조절하거나 연결하는 사람을 통제하고 싶습니다.
- 백엔드가 콜백 URL을 사용하여 클라이언트로 메시지를 다시 보내도록 하고 싶습니다.
- 각 연결 ID 및 기타 정보를 데이터베이스(예: Amazon DynamoDB)에 저장하고 싶습니다.

$connect 라우팅에서 연결 정보 전달

프록시 및 비프록시 통합을 모두 사용하여 $connect 경로에서 데이터베이스 또는 기타 AWS 서비스 서비스로 정보를 전달할 수 있습니다.

프록시 통합을 사용하여 연결 정보 전달

이벤트의 Lambda 프록시 통합에서 연결 정보에 액세스할 수 있습니다. 다른 AWS 서비스 or AWS Lambda 함수를 사용하여 연결에 게시하십시오.

다음 Lambda 함수는 `requestContext` 객체를 사용하여 연결 ID, 도메인 이름, 단계 이름 및 쿼리 문자열을 기록하는 방법을 보여줍니다.

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}
```

비프록시 통합을 사용하여 연결 정보 전달

- 비프록시 통합으로 연결 정보에 액세스할 수 있습니다. 통합 요청을 설정하고 WebSocket API 요청 템플릿을 제공합니다. 다음 Velocity Template Language(VTL) 매핑 템플릿은 통합 요청을 제공합니다. 이 요청은 다음 세부 정보를 비프록시 통합에 전송합니다.

- 연결 ID
- 도메인 이름
- 단계 이름
- 경로
- 헤더
- 쿼리 문자열

이 요청은 연결 ID, 도메인 이름, 단계 이름, 경로, 헤더 및 쿼리 문자열을 비프록시 통합으로 보냅니다.

```
{
    "connectionId": "$context.connectionId",
    "domain": "$context.domainName",
    "stage": "$context.stage",
    "params": "$input.params()"
}
```

데이터 변환 설정에 대한 자세한 내용은 the section called “데이터 변환” 섹션을 참조하세요.

통합 요청을 완료하려면 통합 응답에 `StatusCode: 200`을 설정합니다. 통합 응답 설정에 대한 자세한 내용은 API Gateway 콘솔을 사용하여 통합 응답 설정 섹션을 참조하세요.

`$disconnect` 라우팅

연결이 종료된 후 `$disconnect` 라우팅이 실행됩니다.

서버 또는 클라이언트에 의해 연결이 종료될 수 있습니다. 실행될 때 연결이 이미 종결되었기 때문에, `$disconnect`가 최선의 이벤트입니다. API Gateway는 통합에 `$disconnect` 이벤트를 전달하기 위해 최선을 다하지만 전달을 보장할 수는 없습니다.

백엔드는 `@connections` API를 사용하여 연결 해제를 시작할 수 있습니다. 자세한 내용은 the section called “백엔드 서비스에서 `@connections` 명령 사용” 단원을 참조하세요.

백엔드 통합 호출: `$default` 라우팅 및 사용자 지정 라우팅

주제

- 라우팅을 사용하여 메시지 처리

- $default 라우팅
- 사용자 지정 라우팅
- API Gateway WebSocket API 통합을 사용하여 비즈니스 로직에 연결
- WebSocket API와 REST API 간의 중요한 차이점

라우팅을 사용하여 메시지 처리

API Gateway WebSocket API에서는 메시지를 클라이언트에서 백엔드 서비스로 전송하거나 그 반대로 전송할 수 있습니다. HTTP의 요청/응답 모델과 달리 백엔드에서는 클라이언트가 아무런 WebSocket 조치를 취하지 않아도 클라이언트에 메시지를 보낼 수 있습니다.

메시지는 JSON 또는 비 JSON일 수 있습니다. 그러나 JSON 메시지만 메시지 내용을 기반으로 특정 통합에 라우팅될 수 있습니다. 비 JSON 메시지는 `$default` 라우팅을 통해 백엔드로 전달됩니다.

> **Note**
>
> API Gateway는 최대 128KB의 메시지 페이로드를 지원하며 최대 프레임 크기는 32KB입니다. 메시지가 32KB를 초과하면 32KB 이하의 여러 프레임으로 분할해야 합니다. 큰 메시지(또는 프레임)가 수신되면 코드 1009와 함께 연결이 해제됩니다.
> 현재 이진 페이로드는 지원되지 않습니다. 이진 프레임이 수신되면 코드 1003과 함께 연결이 해제됩니다. 그러나 이진 페이로드를 텍스트로 변환할 수 있습니다. the section called “이진 미디어 유형”을(를) 참조하세요.

API Gateway의 WebSocket API를 사용하면 JSON 메시지를 라우팅하여 메시지 콘텐츠를 기반으로 특정 백엔드 서비스를 실행할 수 있습니다. 클라이언트가 WebSocket 연결을 통해 메시지를 보내면 API에 대한 라우팅 요청이 발생합니다. WebSocket 요청은 API Gateway에서 해당 라우팅 키가 있는 라우팅과 매치됩니다. WebSocket API Gateway 콘솔에서 를 사용하거나 AWS SDK를 사용하여 API에 대한 경로 요청을 설정할 수 있습니다. AWS CLI

> **Note**
>
> AWS CLI 및 AWS SDK에서 통합을 생성하기 전 또는 이후에 경로를 생성할 수 있습니다. 현재 콘솔은 통합의 재사용을 지원하지 않으므로 먼저 라우팅을 생성한 다음 해당 라우팅에 대한 통합을 생성해야 합니다.

통합 요청을 진행하기 전에 라우팅 요청의 확인을 수행하도록 API Gateway를 구성할 수 있습니다. 검증이 실패하면 API Gateway는 백엔드를 호출하지 않고 요청에 실패하고, 다음과 비슷한 "Bad request body" 게이트웨이 응답을 클라이언트에 보내고, 검증 결과를 Logs에 CloudWatch 게시합니다.

```
{"message" : "Bad request body", "connectionId": "{connectionId}", "messageId":
 "{messageId}"}
```

그러면 백엔드에 대한 불필요한 호출이 줄고 API의 다른 요구 사항에 집중할 수 있습니다.

양방향 통신을 활성화하기 위해 API의 라우팅에 대한 라우팅 응답을 정의할 수도 있습니다. 라우팅 응답은 특정 라우팅의 통합이 완료되면 클라이언트에 전송할 데이터를 설명합니다. 예를 들어 클라이언트가 응답을 받지 않고 백엔드로 메시지를 보내도록 하려면(단방향 통신)에는 라우팅에 대한 응답을 정의할 필요가 없습니다. 그러나 라우팅 응답을 제공하지 않으면 API Gateway는 통합 결과에 대한 정보를 클라이언트에 보내지 않습니다.

`$default` 라우팅

모든 WebSocket API Gateway API에는 `$default` 경로가 있을 수 있습니다. 이것은 다음과 같이 사용할 수 있는 특별한 라우팅 값입니다.

- 정의된 라우팅 키와 함께 사용할 경우, 정의된 라우팅 키와 일치하지 않는 수신 메시지에 대해 "대체" 라우팅(예: 특정 오류 메시지를 반환하는 일반 모의 통합)을 지정할 수 있습니다.
- 정의된 라우팅 키 없이 백엔드 구성 요소에 라우팅을 위임하는 프록시 모델을 지정하기 위해 사용할 수 있습니다.
- 비 JSON 페이로드에 대한 라우팅을 지정하는 데 사용할 수 있습니다.

사용자 지정 라우팅

메시지 내용을 기반으로 특정 통합을 호출하려는 경우 사용자 지정 라우팅을 생성하면 됩니다.

사용자 지정 라우팅은 사용자가 지정하는 라우팅 키와 통합을 사용합니다. 수신 메시지에 JSON 속성이 포함되어 있고 해당 속성이 라우팅 키 값과 일치하는 값으로 평가되면 API Gateway가 통합을 호출합니다. (자세한 내용은 the section called “API 정보 WebSocket ” 섹션을 참조하세요.)

예를 들어 대화방 애플리케이션을 만들고 싶다고 가정해 보겠습니다. 먼저 경로 선택 표현식이 `$request.body.action`인 WebSocket API를 만드는 것으로 시작할 수 있습니다. 그런 다음 `joinroom`, `sendmessage` 등 두 가지 라우팅을 정의합니다. 클라이언트 앱은 다음과 같은 메시지를 보내어 `joinroom` 라우팅을 호출할 수 있습니다.

```
{"action":"joinroom","roomname":"developers"}
```

그리고 다음과 같은 메시지를 보내어 sendmessage 라우팅을 호출할 수 있습니다.

```
{"action":"sendmessage","message":"Hello everyone"}
```

API Gateway WebSocket API 통합을 사용하여 비즈니스 로직에 연결

API Gateway WebSocket API의 경로를 설정한 후에는 사용하려는 통합을 지정해야 합니다. 라우팅이 라우팅 요청과 라우팅 응답으로 구성될 수 있는 것과 마찬가지로, 통합은 통합 요청과 통합 응답으로 구성될 수 있습니다. 통합 요청에는 클라이언트에서 제공한 요청을 처리하기 위해 백엔드가 기대한 정보가 들어 있습니다. 통합 응답에는 백엔드가 API Gateway로 반환하는 데이터가 포함되어 있으며, 이는 클라이언트에 보낼 메시지를 구성하는 데 사용할 수 있습니다(라우팅 응답이 정의된 경우).

통합 설정에 대한 자세한 내용은 the section called “통합” 단원을 참조하십시오.

WebSocket API와 REST API 간의 중요한 차이점

WebSocket API 통합은 다음과 같은 차이점을 제외하고 REST API 통합과 비슷합니다.

- 현재 API Gateway 콘솔에서는 먼저 라우팅을 생성한 다음, 해당 라우팅의 대상으로 통합을 생성해야 합니다. 그러나 API와 CLI에서는 순서와 상관없이 라우팅과 통합을 독립적으로 생성할 수 있습니다.
- 여러 라우팅에 하나의 통합을 사용할 수 있습니다. 예를 들어 서로 밀접한 관련이 있는 작업 집합의 경우, 하나의 Lambda 함수로 이동하도록 이러한 모든 라우팅을 설정하고 싶을 수 있습니다. 통합의 세부 사항을 여러 번 정의하는 대신, 한 번만 지정한 후 관련 라우팅에 각각 할당할 수 있습니다.

  > **Note**
  >
  > 현재 콘솔은 통합의 재사용을 지원하지 않으므로 먼저 라우팅을 생성한 다음 해당 라우팅에 대한 통합을 생성해야 합니다.
  > AWS CLI 및 AWS SDK에서 경로의 대상을 값 (여기서 *{integration-id}*"
  > 은 경로와 연결할 통합의 고유 ID) 으로 설정하여 통합을 재사용할 수 있습니다.
  > "integrations/*{integration-id}*"

- API Gateway는 라우팅 및 통합에 사용할 수 있는 여러 선택 표현식을 제공합니다. 콘텐츠 형식에 의존해 입력 템플릿 또는 출력 매핑을 선택할 필요가 없습니다. 라우팅 선택 표현식과 마찬가지로, 올

바른 항목을 선택하기 위해 API Gateway에서 평가할 선택 표현식을 정의할 수 있습니다. 일치하는 템플릿이 없으면 모든 템플릿이 `$default` 템플릿으로 대체됩니다.

- 통합 요청에서 템플릿 선택 표현식은 `$request.body.<json_path_expression>` 및 정적 값을 지원합니다.
- 통합 응답에서 템플릿 선택 표현식은 `$request.body.<json_path_expression>`, `$integration.response.statuscode`, `$integration.response.header.<headerName>` 및 정적 값을 지원합니다.

요청과 응답이 동기식으로 전송되는 HTTP 프로토콜에서 통신은 기본적으로 단방향입니다. WebSocket 프로토콜에서 통신은 양방향 통신입니다. 응답은 비동기식이며, 클라이언트의 메시지가 전송된 순서와 동일한 순서로 클라이언트가 수신하지는 않습니다. 또한 백엔드에서 클라이언트로 메시지를 보낼 수 있습니다.

> **Note**
>
> `AWS_PROXY` 또는 `LAMBDA_PROXY` 통합을 사용하도록 구성된 라우팅의 경우, 통신은 단방향이고 API Gateway는 백엔드 응답을 라우팅 응답에 자동으로 전달하지 않습니다. 예를 들어 `LAMBDA_PROXY` 통합의 경우 Lambda 함수가 반환하는 본문은 클라이언트에 반환되지 않습니다. 클라이언트가 통합 응답을 수신하도록 하려면 양방향 통신을 가능하게 할 라우팅 응답을 정의해야 합니다.

백엔드 서비스에서 연결된 클라이언트로 데이터 전송

API Gateway WebSocket API는 백엔드 서비스에서 연결된 클라이언트로 데이터를 전송할 수 있는 다음과 같은 방법을 제공합니다.

- 통합은 사용자가 정의한 라우팅 응답으로 클라이언트에 반환되는 응답을 보낼 수 있습니다.
- `@connections` API를 사용하여 POST 요청을 전송할 수 있습니다. 자세한 내용은 the section called “백엔드 서비스에서 @connections 명령 사용”을(를) 참조하세요.

API Gateway의 WebSocket 선택 표현식

주제

- 라우팅 응답 선택 표현식
- API 키 선택 표현식

- API 매핑 선택 표현식
- WebSocket 선택 표현식 요약

API Gateway는 요청 및 응답 컨텍스트를 평가하고 키를 생성하는 방법의 하나로 선택 표현식을 사용합니다. 이 경우 키는 일반적으로 API 개발자가 제공하는 가능한 값 집합 중에서 선택하는 데 사용됩니다. 지원되는 변수 집합은 특정 표현식에 따라 달라집니다. 각 표현식에 대해서는 아래에서 자세히 설명합니다.

모든 표현식에 대해 언어는 동일한 규칙 세트를 따릅니다.

- 변수에는 "$"라는 접두사가 붙습니다.
- 중괄호를 사용하여 변수 경계를 명시적으로 정의할 수 있습니다(예: `"${request.body.version}-beta"`).
- 여러 변수가 지원되지만 평가는 한 번만 발생합니다(recursive 평가 없음).
- 달러 문자($)는 "\"로 이스케이프될 수 있습니다. 예약된 `$default` 키에 매핑되는 표현식을 정의할 때 가장 유용합니다(예: `"\$default"`).
- 경우에 따라 패턴 형식이 필요합니다. 이 경우 표현식은 슬래시("/")로 래핑해야 합니다(예: `"/2\d\d/"` 상태 코드와 일치하도록 2*XX*).

라우팅 응답 선택 표현식

라우팅 응답은 백엔드에서 클라이언트로 응답을 모델링하는 데 사용됩니다. WebSocket API에서 라우팅 응답은 선택 사항입니다. 정의되면, WebSocket 메시지 수신 시 클라이언트에 응답을 반환해야 한다는 신호를 API Gateway에 보냅니다.

라우팅 응답 선택 표현식의 평가는 라우팅 응답 키를 생성합니다. 결국 이 키는 API와 연결된 `RouteResponses` 중에서 하나를 선택하는 데 사용됩니다. 그러나 현재는 `$default` 키만 지원됩니다.

API 키 선택 표현식

클라이언트가 유효한 API 키를 제공하는 경우에만 요청이 진행되어야 한다고 서비스가 판단할 때 이 표현식이 평가됩니다.

현재 지원되는 두 가지 값은 `$request.header.x-api-key` 및 `$context.authorizer.usageIdentifierKey`입니다.

API 매핑 선택 표현식

이 표현식은 사용자 지정 도메인을 사용하여 요청이 이루어질 때 선택되는 API 스테이지를 결정하기 위해 평가됩니다.

현재 지원되는 값은 `$request.basepath`입니다.

WebSocket 선택 표현식 요약

다음 표는 WebSocket API의 선택 표현식 사용 사례를 요약한 것입니다.

선택 표현식	다음에 대한 키 평가	참고	사용 사례
`Api.RouteSelectionExpression`	`Route.RouteKey`	`$defaul` catch-all 라우팅으로 지원됩니다.	클라이언트 요청 컨텍스트를 기반으로 WebSocket 메시지를 라우팅합니다.
`Route.ModelSelectionExpression`	`Route.RequestModels` 에 대한 키	선택. 비 프록시 통합에 제공되는 경우 모델 확인이 이루	동일한 라우팅 내에서 요청 확인을 동적으로 수행합니다.

선택 표현식	다음에 대한 키 평가	참고	사용 사례
		어집니다. `$defaul` catch-all로 지원됩니다.	

선택 표현식	다음에 대한 키 평가		참고	사용 사례
`Integration.TemplateSelectionExpression`	`Integration.RequestTemplates`	에 대한 키	선택. 수신 페이로드를 조작하기 위해 비프록시 통합에 제공될 수 있습니다. `${reque .body.: nPath}` 및 정적 값이 지원됩니다. `$defau` catch-all로 지원됩니다.	요청의 동적 속성을 기반으로 호출자의 요청을 조작합니다.

선택 표현식	다음에 대한 키 평가	참고	사용 사례
`Integration.IntegrationResponseSelectionExpression`	`IntegrationResponse.IntegrationResponseKey`	선택. 비 프록시 통합을 위해 제공될 수 있습니다. 오류 메시지(Lambda의) 또는 상태 코드(HTTP 통합의)에 대한 패턴 일치로 작동합니다. `$defaul` 비 프록시 통합이 성공적인 응답에 대한	백엔드의 응답을 조작합니다. 백엔드의 동적 응답에 따라 수행할 작업을 선택하십시오(예: 특정 오류를 명확히 처리).

선택 표현식	다음에 대한 키 평가	참고	사용 사례
		catch-all 역할을 하는 데 필요합니다.	

선택 표현식	다음에 대한 키 평가	참고	사용 사례
IntegrationResponse.TemplateSelectionExpression	IntegrationResponse.ResponseTemplates 에 대한 키	선택. 비 프록시 통합을 위해 제공될 수 있습니다. $default 가 지원됩니다.	경우에 따라 응답의 동적 속성은 동일한 라우팅 및 연결된 통합 내에서 다양한 변환을 지시할 수 있습니다. ${request.body.jsonPath},${integration.response.statuscode} ,${integration.response.header.headerName} ${integration.response.multivalueheader.heade

선택 표현식	다음에 대한 키 평가	참고	사용 사례
			rName}, 및 정적 값이 지원됩니다. $default는 catch-all로 지원됩니다.
Route.RouteResponseSelectionExpression	RouteResponse.RouteResponseKey	WebSoc 라우팅에 대한 양방향 통신을 시작하기 위해 제공되어야 합니다. 현재 이 값은 $defau 제한됩니다.	

선택 표현식	다음에 대한 키 평가		참고	사용 사례
`RouteResponse.ModelSelectionExpression`	`RouteResponse.RequestModels`	에 대한 키	현재 지원되지 않습니다.	

API Gateway에서 WebSocket API 개발

이 단원에서는 API Gateway API를 개발하는 동안 필요한 API Gateway 기능에 대해 자세히 설명합니다.

API Gateway API를 개발할 때 API의 여러 특성을 결정합니다. 이러한 특성은 API의 사용 사례에 따라 달라집니다. 예를 들어, 특정 클라이언트만 API를 호출하도록 허용하거나 모든 사용자가 API를 사용할 수 있도록 할 수 있습니다. API 호출로 Lambda 함수를 실행하거나, 데이터베이스 쿼리를 작성하거나, 애플리케이션을 호출할 수 있습니다.

주제

- API Gateway에서 WebSocket API 생성
- WebSocket API에 대한 라우팅 작업
- API Gateway에서 WebSocket API에 대한 액세스 제어 및 관리
- WebSocket API 연동 설정
- 요청 검증
- WebSocket API에 대한 데이터 변환 설정
- WebSocket API에 대한 이진 미디어 유형 작업
- WebSocket API 호출

API Gateway에서 WebSocket API 생성

AWS CLI `create-api` 명령을 사용하거나 AWS SDK에서 `CreateApi` 명령을 사용하면 API Gateway 콘솔에서 WebSocket API를 생성할 수 있습니다. 다음 절차는 새 WebSocket API를 생성하는 방법을 보여줍니다.

> **Note**
>
> WebSocket API는 TLS 1.2만 지원합니다. 이전 버전의 TLS는 지원되지 않습니다.

AWS CLI 명령을 사용하여 WebSocket API 생성

AWS CLI를 사용하여 WebSocket API를 생성하려면 다음 예와 같이 `create-api` 명령을 호출해야 합니다. 이 명령은 `$request.body.action` 라우팅 선택 표현식을 사용하여 API를 생성합니다.

```
aws apigatewayv2 --region us-east-1 create-api --name "myWebSocketApi3" --protocol-type
 WEBSOCKET --route-selection-expression '$request.body.action'
```

출력 예:

```
{
    "ApiKeySelectionExpression": "$request.header.x-api-key",
    "Name": "myWebSocketApi3",
    "CreatedDate": "2018-11-15T06:23:51Z",
    "ProtocolType": "WEBSOCKET",
    "RouteSelectionExpression": "'$request.body.action'",
    "ApiId": "aabbccddee"
}
```

API Gateway 콘솔을 사용하여 WebSocket API 생성

콘솔에서 WebSocket 프로토콜을 선택하고 API에 이름을 지정하여 WebSocket API를 만들 수 있습니다.

> **Important**
>
> API를 생성한 후에는 선택한 프로토콜을 변경할 수 없습니다. WebSocket API를 REST API로 또는 그 반대로 변환할 수는 없습니다.

API Gateway 콘솔을 사용하여 WebSocket API를 생성하려면

1. API Gateway 콘솔에 로그인하고 API 생성을 선택합니다.
2. WebSocket API에서 빌드를 선택합니다. 리전 엔드포인트만 지원됩니다.

3. API 이름에 API의 이름을 입력합니다.
4. 라우팅 선택 표현식에 값을 입력합니다. 예: `$request.body.action`.

   라우팅 선택 표현식에 대한 자세한 내용은 the section called “” 단원을 참조하십시오.
5. 다음 중 하나를 수행하십시오.
   - 라우팅이 없는 API를 생성하려면 빈 API 생성을 선택합니다.
   - 다음을 선택하여 API에 라우팅을 연결합니다.

   API를 생성한 후 라우팅을 연결할 수 있습니다.

WebSocket API에 대한 라우팅 작업

WebSocket API에서 수신되는 JSON 메시지는 사용자가 구성한 라우팅을 기반으로 백엔드 통합으로 전달됩니다. (비 JSON 메시지는 사용자가 구성한 `$default` 라우팅으로 전달됩니다.)

라우팅에는 라우팅 선택 표현식을 평가할 경우 예상되는 값인 라우팅 키가 포함되어 있습니다. `routeSelectionExpression`은 API 수준에서 정의되는 속성입니다. 메시지 페이로드에 존재할 것으로 예상되는 JSON 속성을 지정합니다. 라우팅 선택 표현식에 대한 자세한 내용은 the section called “” 단원을 참조하십시오.

예를 들어 JSON 메시지에 `action` 속성이 있고 이 속성을 기반으로 다른 작업을 수행하려는 경우, 라우팅 선택 표현식은 `${request.body.action}`일 수 있습니다. 라우팅 테이블은 사용자가 테이블에 정의한 사용자 지정 라우팅 키 값에 따라 `action` 속성 값을 일치시켜 어떤 작업을 수행할 것인지 지정할 것입니다.

`$connect`, `$disconnect`, `$default` 등 세 가지 사전 정의된 라우팅을 사용할 수 있습니다. 또한 사용자 지정 라우팅을 생성할 수 있습니다.

- API Gateway는 클라이언트와 WebSocket API 간 지속적인 연결이 시작될 때 `$connect` 라우팅을 호출합니다.
- API Gateway는 클라이언트 또는 서버가 API와의 연결을 끊을 때 `$disconnect` 라우팅을 호출합니다.
- 일치하는 라우팅이 발견되면 메시지를 기준으로 라우팅 선택 표현식이 평가된 후 API Gateway가 사용자 지정 라우팅을 호출합니다. 일치 여부에 따라 어떤 통합이 호출될지 결정됩니다.
- 일치하는 라우팅이 없거나 메시지를 기준으로 라우팅 선택 표현식을 평가할 수 없는 경우 API Gateway가 `$default` 라우팅을 호출합니다.

라우팅 선택 표현식

서비스가 수신 메시지에 대해 따라야 할 라우팅을 선택할 때 라우팅 선택 표현식이 평가됩니다. 서비스는 routeKey가 평가된 값과 정확하게 일치하는 라우팅을 사용합니다. 일치하는 값이 없고 라우팅 키가 $default인 라우팅이 존재하는 경우, 그 값이 선택됩니다. 평가된 값과 일치하는 라우팅이 없고 $default 라우팅이 없는 경우 서비스는 오류를 반환합니다. WebSocket 기반 API의 경우 표현식은 $request.body.*{path_to_body_element}* 형식이어야 합니다.

예를 들어 다음과 같은 JSON 메시지를 보낸다고 가정하겠습니다.

```
{
    "service" : "chat",
    "action" : "join",
    "data" : {
        "room" : "room1234"
   }
}
```

action 속성을 기반으로 API의 동작을 선택할 수 있습니다. 이 경우 다음 라우팅 선택 표현식을 정의할 수 있습니다.

```
$request.body.action
```

이 예에서 request.body는 메시지의 JSON 페이로드를 나타내며 .action은 JSONPath 표현식입니다. request.body 뒤에 모든 JSON 경로 표현식을 사용할 수 있지만 결과는 문자열로 변환됩니다. 예를 들어, JSONPath 표현식이 두 요소의 배열을 반환하면 "[item1, item2]" 문자열로 표시됩니다. 따라서 표현식을 배열이나 객체가 아닌 값으로 평가하는 것이 좋습니다.

단순히 정적 값을 사용하거나 여러 변수를 사용할 수 있습니다. 다음 표는 이전의 페이로드에 대한 예제와 평가된 결과를 보여줍니다.

표현식	평가된 결과	설명
$request.body.action	join	언래핑된 변수
${request.body.action}	join	래핑된 변수

표현식	평가된 결과	설명
`${request.body.service}/${request.body.action}`	`chat/join`	정적 값이 있는 여러 변수
`${request.body.action}-${request.body.invalidPath}`	`join-`	JSONPath를 찾지 못하면 변수는 ""로 해석됩니다.
`action`	`action`	정적 값
`\$default`	`$default`	정적 값

평가된 결과는 라우팅을 찾는 데 사용됩니다. 일치하는 라우팅 키를 가진 라우팅이 있는 경우, 해당 라우팅이 선택되어 메시지를 처리합니다. 일치하는 라우팅이 없으면 API Gateway가 사용 가능한 `$default` 라우팅을 찾습니다. `$default` 라우팅이 정의되지 않은 경우, API Gateway는 오류를 반환합니다.

API Gateway에서 WebSocket API에 대한 라우팅 설정

처음 새 WebSocket API를 생성할 때 `$connect`, `$disconnect`, `$default` 등 세 가지 사전 정의된 라우팅이 있습니다. 콘솔, API 또는 AWS CLI를 사용하여 생성할 수 있습니다. 필요에 따라 사용자 지정 라우팅을 생성할 수 있습니다. 자세한 내용은 the section called “API 정보 WebSocket ” 단원을 참조하세요.

> **Note**
>
> CLI에서는 통합을 생성하기 이전 또는 이후에 라우팅을 생성할 수 있으며 여러 라우팅에 대해 동일한 통합을 재사용할 수 있습니다.

API Gateway 콘솔을 사용하여 라우팅 생성

API Gateway 콘솔을 사용하여 라우팅을 생성하려면

1. API Gateway 콘솔에 로그인하고 API를 선택한 다음 라우팅을 선택하세요.
2. 경로 생성을 선택합니다.
3. 라우팅 키에 라우팅 키 이름을 입력합니다. 사전 정의된 경로($connect, $disconnect 및 $default) 또는 사용자 지정 경로를 생성할 수 있습니다.

   > **Note**
   >
   > 사용자 지정 라우팅을 생성할 때 라우팅 키 이름에 $ 접두사를 사용하지 마십시오. 이 접두사는 미리 정의된 라우팅에 예약되어 있습니다.

4. 경로의 통합 유형을 선택하고 구성합니다. 자세한 내용은 the section called “ WebSocket API Gateway 콘솔을 사용하여 API 통합 요청을 설정합니다.” 섹션을 참조하세요.

AWS CLI를 사용하여 라우팅 생성

AWS CLI를 사용하여 라우팅을 생성하려면 다음 예와 같이 `create-route`를 호출합니다.

```
aws apigatewayv2 --region us-east-1 create-route --api-id aabbccddee --route-key
 $default
```

출력 예:

```
{
    "ApiKeyRequired": false,
    "AuthorizationType": "NONE",
    "RouteKey": "$default",
    "RouteId": "1122334"
}
```

$connect에 대한 라우팅 요청 설정 지정

API에 대해 $connect 라우팅을 설정하면 다음 선택적 설정을 사용하여 API에 대한 권한 부여를 활성화할 수 있습니다. 자세한 내용은 the section called “$connect 라우팅” 단원을 참조하세요.

- 권한 부여: 권한 부여가 필요하지 않으면 NONE을 지정할 수 있습니다. 또는 다음을 지정할 수 있습니다.
  - AWS_IAM: 표준 AWS IAM 정책을 사용하여 API에 대한 액세스를 제어합니다.
  - CUSTOM: 이전에 생성한 Lambda 권한 부여자 함수를 지정하여 API에 대한 권한 부여를 구현합니다. 권한 부여자는 자신의 AWS 계정이나 다른 AWS 계정에 상주할 수 있습니다. Lambda 권한 부여자에 대한 자세한 내용은 API Gateway Lambda 권한 부여자 사용 단원을 참조하세요.

> **Note**
>
> API Gateway 콘솔에서 CUSTOM 설정은 the section called "콘솔을 사용하여 Lambda 권한 부여자 구성"에서 설명한 대로 권한 부여자 함수를 설정한 후에만 볼 수 있습니다.

> **Important**
>
> 권한 부여 설정은 $connect 라우팅뿐만 아니라 전체 API에 적용됩니다. $connect 라우팅은 모든 연결에서 호출되기 때문에 다른 라우팅을 보호합니다.

- API Key Required(API 키 필요): 선택적으로, API의 $connect 라우팅에 대한 API 키가 필요할 수 있습니다. API 키와 사용량 계획을 함께 사용하여 API 액세스를 제어하고 추적할 수 있습니다. 자세한 내용은 the section called "사용량 계획" 단원을 참조하세요.

API Gateway 콘솔을 사용하여 **$connect** 라우팅 요청 설정

API Gateway 콘솔을 사용하여 WebSocket API에 대한 $connect 라우팅 요청을 설정하려면 다음을 수행하세요.

1. API Gateway 콘솔에 로그인하고 API를 선택한 다음 라우팅을 선택하세요.
2. 경로에서 the section called "API Gateway 콘솔을 사용하여 라우팅 생성"을 따라 $connect를 선택하거나 $connect 경로를 생성합니다.
3. 경로 요청 설정 섹션에서 편집을 선택합니다.
4. 권한 부여에서 권한 부여 유형을 선택합니다.
5. $connect 경로용 API를 요구하려면 API 키 필요를 선택합니다.
6. 변경 사항 저장을 선택합니다.

API Gateway에서 WebSocket API에 대한 라우팅 응답 설정

WebSocket 라우팅은 양방향 또는 단방향 통신으로 구성할 수 있습니다. 경로 응답을 설정하지 않으면 API Gateway가 경로 응답을 통해 백엔드 응답을 전달하지 않습니다.

> **Note**
>
> WebSocket API에 대한 `$default` 경로 응답만 정의할 수 있습니다. 통합 응답을 사용하여 백엔드 서비스의 응답을 조작할 수 있습니다. 자세한 내용은 the section called “통합 응답 개요” 섹션을 참조하세요.

API Gateway 콘솔이나 AWS CLI 또는 AWS SDK를 사용하여 경로 응답 및 응답 선택 표현식을 구성할 수 있습니다.

라우팅 응답 선택 표현식에 대한 자세한 내용은 the section called “” 단원을 참조하십시오.

주제

- API Gateway 콘솔을 사용하여 라우팅 응답 설정
- AWS CLI를 사용하여 라우팅 응답 설정

API Gateway 콘솔을 사용하여 라우팅 응답 설정

WebSocket API를 생성하고 프록시 Lambda 함수를 기본 경로에 연결한 후, API Gateway 콘솔을 사용하여 경로 응답을 설정할 수 있습니다.

1. API Gateway 콘솔에 로그인하고 `$default` 경로에 프록시 Lambda 함수가 통합된 WebSocket API를 선택합니다.
2. Routes(경로)에서 `$default` 경로를 선택합니다.
3. 양방향 통신 활성화를 선택합니다.
4. Deploy API(API 배포)를 선택합니다.
5. API를 스테이지에 배포합니다.

API에 연결하려면 다음 wscat 명령을 사용합니다. `wscat`에 대한 자세한 정보는을 참조하십시오the section called “`wscat`를 사용하여 WebSocket API에 연결하고 메시지 보내기”.

```
wscat -c wss://api-id.execute-api.us-east-2.amazonaws.com/test
```

기본 경로를 호출하려면 enter(엔터) 버튼을 누릅니다. Lambda 함수 본문이 반환될 것입니다.

AWS CLI를 사용하여 라우팅 응답 설정

AWS CLI를 사용하여 WebSocket API에 대한 라우팅 응답을 설정하려면 다음 예와 같이 `create-route-response` 명령을 호출하십시오. `get-apis` 및 `get-routes`를 호출하여 API ID 및 라우팅 ID를 식별할 수 있습니다.

```
aws apigatewayv2 create-route-response \
    --api-id aabbccddee \
    --route-id 1122334  \
    --route-response-key '$default'
```

출력 예:

```
{
    "RouteResponseId": "abcdef",
    "RouteResponseKey": "$default"
}
```

WebSocket 서브프로토콜이 필요한 **$connect** 라우팅 설정

클라이언트는 WebSocket API에 연결하는 동안 WebSocket 서브프로토콜을 요청하기 위해 `Sec-WebSocket-Protocol` 필드를 사용할 수 있습니다. 클라이언트가 API가 지원하는 서브프로토콜을 요청하는 경우에만 연결을 허용하도록 `$connect` 라우팅에 대한 통합을 설정할 수 있습니다.

다음 예제 Lambda 함수는 `Sec-WebSocket-Protocol` 헤더를 클라이언트에 반환합니다. 이 함수는 클라이언트가 `myprotocol` 서브프로토콜을 지정하는 경우에만 API에 대한 연결을 설정합니다.

이 예제 API 및 Lambda 프록시 통합을 생성하는 AWS CloudFormation 템플릿에 대해서는 `ws-subprotocol.yaml` 단원을 참조하세요.

```
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;
}
```

wscat를 사용하면 클라이언트가 API가 지원하는 서브프로토콜을 요청하는 경우에만 API가 연결을 허용하는지 테스트할 수 있습니다. 다음 명령은 `-s` 플래그를 사용하여 연결 중에 서브프로토콜을 지정합니다.

다음 명령은 지원되지 않는 서브프로토콜을 사용하여 연결을 시도합니다. 클라이언트가 `chat1` 서브프로토콜을 지정했기 때문에 Lambda 통합에서 400 오류를 반환하고 연결이 실패합니다.

```
wscat -c wss://api-id.execute-api.region.amazonaws.com/beta -s chat1
error: Unexpected server response: 400
```

다음 명령은 연결 요청에 지원되는 서브프로토콜을 포함합니다. Lambda 통합은 연결을 허용합니다.

```
wscat -c wss://api-id.execute-api.region.amazonaws.com/beta -s chat1,myprotocol
connected (press CTRL+C to quit)
```

WebSocket API 호출에 대한 자세한 내용은 WebSocket API 호출 단원을 참조하세요.

API Gateway에서 WebSocket API에 대한 액세스 제어 및 관리

API Gateway는 WebSocket API에 대한 액세스를 제어 및 관리하기 위한 여러 메커니즘을 지원합니다.

다음 메커니즘은 인증 및 권한 부여에 사용할 수 있습니다.

- 표준 AWS IAM 역할 및 정책은 유연하고 강력한 액세스 제어를 제공합니다. IAM 역할 및 정책을 사용하여 API를 생성하고 관리할 수 있는 사용자와 API를 호출할 수 있는 사용자를 관리할 수 있습니다. 자세한 내용은 IAM 권한 부여 사용 단원을 참조하세요.
- IAM 태그는 액세스 제어를 위해 IAM 정책과 함께 사용할 수 있습니다. 자세한 내용은 태그를 사용하여 API Gateway REST API 리소스에 대한 액세스 제어 단원을 참조하세요.
- Lambda 권한 부여자는 Lambda 함수를 사용하여 API에 대한 액세스를 제어합니다. 자세한 내용은 Lambda REQUEST 권한 부여자 함수 단원을 참조하세요.

주제

- IAM 권한 부여 사용
- Lambda REQUEST 권한 부여자 함수

IAM 권한 부여 사용

API의 IAM WebSocket 인증은 REST API의 IAM 인증과 비슷하지만 다음과 같은 점이 있습니다.

- `execute-api` 작업은기존 작업(`ManageConnections`, `Invoke`) 이외에 `InvalidateCache`를 지원합니다. `ManageConnections`는 @connections API에 대한 액세스를 제어합니다.
- WebSocket 경로는 다른 ARN 형식을 사용합니다.

```
arn:aws:execute-api:region:account-id:api-id/stage-name/route-key
```

- `@connections` API는 REST API와 동일한 ARN 형식을 사용합니다.

```
arn:aws:execute-api:region:account-id:api-id/stage-name/POST/@connections
```

> **Important**
>
> IAM 권한 부여를 사용하는 경우 Signature Version 4(SigV4)로 서명해야 합니다.

예를 들어 클라이언트에 다음 정책을 설정할 수 있습니다. 이 예제는 모든 사람이 Invoke 스테이지에서 비밀 라우팅을 제외한 모든 라우팅에 대한 메시지(prod)를 보내도록 허용하고, 모든 사람이 모든 스테이지에 대해 연결된 클라이언트(ManageConnections)로 메시지를 보내는 것을 금지합니다.

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

Lambda **REQUEST** 권한 부여자 함수

API의 Lambda 권한 부여 함수는 다음과 같은 점을 제외하면 REST WebSocket API의 함수와 비슷합니다.

- `$connect`경로에 Lambda 권한 부여자 함수만 사용할 수 있습니다.
- 경로는 고정되어 있으므로 경로 변수(`event.pathParameters`)를 사용할 수 없습니다.
- `event.methodArn`은 HTTP 메서드가 없으므로 상응하는 REST API와 다릅니다. `$connect`의 경우, `methodArn`은 `"$connect"`로 끝납니다.

```
arn:aws:execute-api:region:account-id:api-id/stage-name/$connect
```

- `event.requestContext`의 컨텍스트 변수는 REST API에 대한 변수와 다릅니다.

다음 예제는 API에 대한 권한 부여자에 대한 입력을 보여줍니다. REQUEST WebSocket

```
{
    "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"
    }
}
```

다음 예제 Lambda 권한 부여자 함수는 REST API용 Lambda 권한 부여 함수 WebSocket 버전입니다. the section called “Lambda 콘솔에서 Lambda 권한 부여자 함수 생성”

Node.js

```
  // A simple REQUEST authorizer example to demonstrate how to use request
  // parameters to allow or deny a request. In this example, a request is
  // authorized if the client-supplied HeaderAuth1 header and QueryString1 query
parameter
  // in the request context match the specified values of
  // of 'headerValue1' and 'queryValue1' respectively.
          export const handler = function(event, context, callback) {
   console.log('Received event:', JSON.stringify(event, null, 2));

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

  // Parse the input for the parameter values
  var tmp = event.methodArn.split(':');
  var apiGatewayArnTmp = tmp[5].split('/');
  var awsAccountId = tmp[4];
  var region = tmp[3];
  var ApiId = apiGatewayArnTmp[0];
  var stage = apiGatewayArnTmp[1];
  var route = apiGatewayArnTmp[2];

  // Perform authorization to return the Allow policy for correct parameters and
  // the 'Unauthorized' error, otherwise.
  var authResponse = {};
  var condition = {};
```

```
    condition.IpAddress = {};

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

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

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

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

Python

```
# A simple REQUEST authorizer example to demonstrate how to use request
# parameters to allow or deny a request. In this example, a request is
# authorized if the client-supplied HeaderAuth1 header and QueryString1 query
 parameter
# in the request context match the specified values of
# of 'headerValue1' and 'queryValue1' respectively.

import json


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

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

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

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

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

    if (headers['HeaderAuth1'] ==
            "headerValue1" and queryStringParameters["QueryString1"] ==
 "queryValue1"):
        response = generateAllow('me', event['methodArn'])
        print('authorized')
        return json.loads(response)
    else:
```

```
        print('unauthorized')
        return 'unauthorized'

    # Help function to generate IAM policy


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

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

    authResponse_JSON = json.dumps(authResponse)

    return authResponse_JSON


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


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

위의 Lambda 함수를 API용 WebSocket 권한 부여 함수로 구성하려면 REST API와 REQUEST 동일한 절차를 따르십시오.

콘솔에서 이 Lambda 권한 부여자를 사용하도록 $connect 라우팅을 구성하려면 $connect 라우팅을 선택하거나 만듭니다. 경로 요청 설정 섹션에서 편집을 선택합니다. 권한 부여 드롭다운 메뉴에서 권한 부여자를 선택한 다음 변경 사항 저장을 선택합니다.

권한 부여자를 테스트하려면 새 연결을 생성해야 합니다. $connect에서 권한 부여자를 변경해도 이미 연결된 클라이언트에는 영향을 미치지 않습니다. WebSocket API에 연결할 때는 구성된 모든 ID 소스에 값을 제공해야 합니다. 예를 들어 다음 예제와 같이 wscat를 사용하여 유효한 쿼리 문자열과 헤더를 보내 연결할 수 있습니다.

```
wscat -c 'wss://myapi.execute-api.us-east-1.amazonaws.com/beta?
QueryString1=queryValue1' -H HeaderAuth1:headerValue1
```

유효한 자격 증명 값 없이 연결을 시도하면 401 응답을 받게 됩니다.

```
wscat -c wss://myapi.execute-api.us-east-1.amazonaws.com/beta
error: Unexpected server response: 401
```

WebSocket API 연동 설정

API 라우팅을 설정한 후 이를 백엔드의 엔드포인트와 통합해야 합니다. 백엔드 엔드포인트는 통합 엔드포인트라고도 하며 Lambda 함수, HTTP 엔드포인트 또는 서비스 작업일 수 있습니다. AWS API 통합에는 통합 요청 및 통합 응답이 있습니다.

이 섹션에서는 API에 대한 통합 요청 및 통합 응답을 설정하는 방법을 배울 수 있습니다. WebSocket

주제

- API Gateway에서 WebSocket API 통합 요청 설정
- API Gateway에서 WebSocket API 통합 응답 설정

API Gateway에서 WebSocket API 통합 요청 설정

통합 요청을 설정하려면 다음 단계를 수행합니다.

- 백엔드에 통합할 경로 키를 선택합니다.
- 호출할 백엔드 엔드포인트 지정 WebSocket API는 다음과 같은 통합 유형을 지원합니다.
  - AWS_PROXY
  - AWS
  - HTTP_PROXY

  - HTTP
  - MOCK

  통합 유형에 대한 자세한 내용은 API Gateway V2 REST API를 참조하십시오 IntegrationType.
- 필요할 경우 하나 이상의 요청 템플릿을 지정하여 라우팅 요청 데이터를 어떻게 통합 요청 데이터로 변환할 것인지 구성합니다.

WebSocket API Gateway 콘솔을 사용하여 API 통합 요청을 설정합니다.

API Gateway 콘솔을 사용하여 WebSocket API의 경로에 통합 요청을 추가하려면

1. API Gateway 콘솔에 로그인하고 API를 선택한 다음 라우팅을 선택하세요.
2. 라우팅에서 라우팅을 선택합니다.
3. 통합 요청 탭을 선택한 다음 통합 요청 설정 섹션에서 편집을 선택합니다.
4. 통합 유형에서 다음 중 하나를 선택합니다.

   - API가 이 계정이나 다른 계정에서 이미 생성한 함수와 AWS Lambda 통합되는 경우에만 Lambda 함수를 선택하십시오.

     AWS Lambda에서 새 Lambda 함수를 생성하거나, Lambda 함수에 리소스 권한을 설정하거나, 다른 Lambda 서비스 작업을 수행하려면 대신 Service를 선택하십시오.AWS
   - API가 기존 HTTP 엔드포인트와 통합될 예정이라면 HTTP를 선택합니다. 자세한 내용은 API Gateway에서 HTTP 통합 설정 단원을 참조하세요.
   - 통합 백엔드 없이 직접 API Gateway에서 API 응답을 생성하려면 모의(Mock)를 선택하세요. 자세한 설명은 API Gateway에서 모의 통합 설정 섹션을 참조하세요.
   - API가 AWS 서비스와 통합될 경우 서비스를 선택하십시오. AWS
   - API가 프라이빗 통합 엔드포인트로 VpcLink를 사용할 경우 VPC 링크를 선택합니다. 자세한 내용은 API Gateway 프라이빗 통합 설정 단원을 참조하세요.
5. Lambda 함수를 선택한 경우 다음을 수행합니다.

   a. Lambda 프록시 통합 사용의 경우, Lambda 프록시 통합 또는 교차 계정 Lambda 프록시 통합을 사용하려면 해당 확인란을 선택합니다.

   b. Lambda 함수의 경우, 다음 방법 중 하나로 함수를 지정합니다.

      - Lambda 함수가 동일한 계정에 있으면 함수 이름을 입력한 후 드롭다운 목록에서 해당 함수를 선택합니다.

> **Note**
>
> 함수 이름은 `HelloWorld`, `HelloWorld:1` 또는 `HelloWorld:alpha`에서와 같이, 별칭 또는 버전 사양을 선택적으로 포함할 수 있습니다.

- 함수가 다른 계정에 있으면 함수의 ARN을 입력합니다.

c. 기본 제한 시간 값인 29초를 사용하려면 기본 제한 시간을 활성화된 상태로 유지합니다. 사용자 지정 제한 시간을 설정하려면 기본 제한 시간을 선택하고 `50` ~ `29000`밀리초 사이의 제한 시간 값을 입력합니다.

6. HTTP를 선택한 경우, the section called “ 콘솔을 사용하여 통합 요청 설정”의 4단계 지침을 따릅니다.
7. Mock(모의)을 선택한 경우 Request Templates(요청 템플릿) 단계로 진행하십시오.
8. AWS 서비스를 선택한 경우, the section called “ 콘솔을 사용하여 통합 요청 설정”의 6단계 지침을 따릅니다.
9. VPC 링크를 선택한 경우 다음을 수행합니다.

   a. 프록시 통합 사용에서 요청이 `VPCLink`의 엔드포인트로 프록시되도록 하려면 확인란을 선택합니다.

   b. HTTP 메서드(HTTP method)에서 HTTP 백엔드의 메서드와 가장 가까운 HTTP 메서드 유형을 선택합니다.

   c. VPC 링크 드롭다운 목록에서 VPC 링크를 선택합니다. `[Use Stage Variables]`를 선택하고 목록 아래의 텍스트 상자에 **`${stageVariables.vpcLinkId}`**를 입력할 수 있습니다.

      API를 스테이지에 배포한 후 `vpcLinkId` 스테이지 변수를 정의하고, 그 값을 `VpcLink`의 ID로 설정할 수 있습니다.

   d. Endpoint URL(엔드포인트 URL)에서 이 통합을 사용하고자 하는 HTTP 백엔드의 URL을 입력합니다.

   e. 기본 제한 시간 값인 29초를 사용하려면 기본 제한 시간을 활성화된 상태로 유지합니다. 사용자 지정 제한 시간을 설정하려면 기본 제한 시간을 선택하고 `50` ~ `29000`밀리초 사이의 제한 시간 값을 입력합니다.

10. 변경 사항 저장를 선택합니다.
11. 요청 템플릿에서 다음의 작업을 수행합니다.

    a. 템플릿 선택 표현식을 입력하려면 요청 템플릿에서 편집을 선택합니다.

b. 템플릿 선택 표현식을 입력합니다. 메시지 페이로드에서 API Gateway가 찾는 표현식을 사용합니다. 검색되면 평가가 이루어지고, 그 결과는 메시지 페이로드의 데이터에 적용할 데이터 매핑 템플릿을 선택하는 데 사용되는 템플릿 키 값입니다. 다음 단계에서는 데이터 매핑 템플릿을 생성합니다. 편집을 선택하여 변경 사항을 저장합니다.

c. 템플릿 생성을 선택하여 데이터 매핑 템플릿을 생성합니다. 템플릿 키에는 메시지 페이로드의 데이터에 적용할 데이터 매핑 템플릿을 선택하는 데 사용되는 템플릿 키 값을 입력합니다. 그런 다음 매핑 템플릿을 입력합니다. 템플릿 생성을 선택합니다.

   템플릿 선택 표현식에 대한 내용은 the section called “템플릿 선택 표현식” 단원을 참조하십시오.

를 사용하여 통합 요청을 설정합니다. AWS CLI

다음 예제와 AWS CLI 같이 WebSocket API의 경로에 대한 통합 요청을 설정하면 모의 통합이 생성됩니다.

1. 다음 콘텐츠를 통해 `integration-params.json`이라는 파일을 생성합니다.

```
{"PassthroughBehavior": "WHEN_NO_MATCH", "TimeoutInMillis": 29000,
 "ConnectionType": "INTERNET", "RequestTemplates": {"application/json":
 "{\"statusCode\":200}"}, "IntegrationType": "MOCK"}
```

2. 다음 예와 같이 `create-integration` 명령을 사용하세요.

```
aws apigatewayv2 --region us-east-1 create-integration --api-id aabbccddee --cli-
input-json file://integration-params.json
```

다음은 이 예에 대한 샘플 출력입니다.

```
{
    "PassthroughBehavior": "WHEN_NO_MATCH",
    "TimeoutInMillis": 29000,
    "ConnectionType": "INTERNET",
    "IntegrationResponseSelectionExpression": "${response.statuscode}",
    "RequestTemplates": {
        "application/json": "{\"statusCode\":200}"
    },
    "IntegrationId": "0abcdef",
    "IntegrationType": "MOCK"
```

```
}
```

또는 다음 예제와 AWS CLI 같이 프록시 통합을 위한 통합 요청을 설정할 수 있습니다.

1. Lambda 콘솔에서 Lambda 함수를 생성하고 기본 Lambda 실행 역할을 부여하세요.
2. 다음 예제에서처럼 `create-integration` 명령을 실행합니다.

```
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
```

다음은 이 예에 대한 샘플 출력입니다.

```
{
    "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"
}
```

API에 대한 프록시 통합을 위한 Lambda 함수의 입력 형식 WebSocket

Lambda 프록시 통합을 사용할 경우 API Gateway에서 전체 클라이언트 요청을 백엔드 Lambda 함수의 입력 `event` 파라미터에 매핑합니다. 다음 예제는 API Gateway가 Lambda 프록시 통합으로 전송하는 `$disconnect` 라우트의 입력 이벤트와 라우트의 입력 이벤트 구조를 보여줍니다. `$connect`

Input from the $connect route

```
{
    headers: {
      Host: 'abcd123.execute-api.us-east-1.amazonaws.com',
      'Sec-WebSocket-Extensions': 'permessage-deflate; client_max_window_bits',
      'Sec-WebSocket-Key': '...',
      'Sec-WebSocket-Version': '13',
      'X-Amzn-Trace-Id': '...',
```

```
      'X-Forwarded-For': '192.0.2.1',
      'X-Forwarded-Port': '443',
      'X-Forwarded-Proto': 'https'
    },
    multiValueHeaders: {
      Host: [ 'abcd123.execute-api.us-east-1.amazonaws.com' ],
      'Sec-WebSocket-Extensions': [ 'permessage-deflate; client_max_window_bits' ],
      'Sec-WebSocket-Key': [ '...' ],
      'Sec-WebSocket-Version': [ '13' ],
      'X-Amzn-Trace-Id': [ '...' ],
      'X-Forwarded-For': [ '192.0.2.1' ],
      'X-Forwarded-Port': [ '443' ],
      'X-Forwarded-Proto': [ 'https' ]
    },
    requestContext: {
      routeKey: '$connect',
      eventType: 'CONNECT',
      extendedRequestId: 'ABCD1234=',
      requestTime: '09/Feb/2024:18:11:43 +0000',
      messageDirection: 'IN',
      stage: 'prod',
      connectedAt: 1707502303419,
      requestTimeEpoch: 1707502303420,
      identity: { sourceIp: '192.0.2.1' },
      requestId: 'ABCD1234=',
      domainName: 'abcd1234.execute-api.us-east-1.amazonaws.com',
      connectionId: 'AAAA1234=',
      apiId: 'abcd1234'
    },
    isBase64Encoded: false
  }
```

Input from the $disconnect route

```
{
    headers: {
      Host: 'abcd1234.execute-api.us-east-1.amazonaws.com',
      'x-api-key': '',
      'X-Forwarded-For': '',
      'x-restapi': ''
    },
    multiValueHeaders: {
```

```
    Host: [ 'abcd1234.execute-api.us-east-1.amazonaws.com' ],
    'x-api-key': [ '' ],
    'X-Forwarded-For': [ '' ],
    'x-restapi': [ '' ]
  },
  requestContext: {
    routeKey: '$disconnect',
    disconnectStatusCode: 1005,
    eventType: 'DISCONNECT',
    extendedRequestId: 'ABCD1234=',
    requestTime: '09/Feb/2024:18:23:28 +0000',
    messageDirection: 'IN',
    disconnectReason: 'Client-side close frame status not set',
    stage: 'prod',
    connectedAt: 1707503007396,
    requestTimeEpoch: 1707503008941,
    identity: { sourceIp: '192.0.2.1' },
    requestId: 'ABCD1234=',
    domainName: 'abcd1234.execute-api.us-east-1.amazonaws.com',
    connectionId: 'AAAA1234=',
    apiId: 'abcd1234'
  },
  isBase64Encoded: false
}
```

API Gateway에서 WebSocket API 통합 응답 설정

주제

- 통합 응답 개요
- 양방향 커뮤니케이션을 위한 통합 응답
- API Gateway 콘솔을 사용하여 통합 응답 설정
- 를 사용하여 통합 응답을 설정합니다. AWS CLI

통합 응답 개요

API Gateway의 통합 응답은 백엔드 서비스의 응답을 모델링하고 조작하는 한 방법입니다. REST API 설정과 WebSocket API 통합 응답에는 약간의 차이가 있지만 개념적으로는 동작이 동일합니다.

WebSocket 양방향 또는 단방향 통신을 위해 경로를 구성할 수 있습니다.

- 라우팅이 양방향 통신을 위해 구성되면 통합 응답을 통해 REST API에 대한 통합 응답과 마찬가지로 반환된 메시지 페이로드에 대한 변환을 구성할 수 있습니다.
- 경로가 단방향 통신을 위해 구성된 경우 통합 응답 구성에 관계없이 메시지가 처리된 후 WebSocket 채널을 통해 응답이 반환되지 않습니다.

경로 응답을 설정하지 않으면 API Gateway가 경로 응답을 통해 백엔드 응답을 전달하지 않습니다. 경로 응답 설정에 대한 자세한 내용은 the section called “WebSocket API 라우팅 응답 설정” 섹션을 참조하세요.

양방향 커뮤니케이션을 위한 통합 응답

통합은 프록시 통합과 비 프록시 통합으로 나눌 수 있습니다.

> **Important**
>
> 프록시 통합의 경우, API Gateway가 자동으로 백엔드 출력을 호출자에게 전체 페이로드로 전달합니다. 통합 응답이 없습니다.

비 프록시 통합의 경우 최소한 하나의 통합 응답을 설정해야 합니다.

- 명시적 선택을 할 수 없을 때 통합 응답 중 하나가 catch-all 역할을 하는 것이 이상적입니다. 이 기본 경우는 통합 응답 키 `$default` 설정으로 표현됩니다.
- 다른 모든 경우에는 통합 응답 키가 정규 표현식으로 사용됩니다. `"/expression/"` 형식을 따라야 합니다.

비 프록시 HTTP 통합의 경우:

- API Gateway가 백엔드 응답의 HTTP 상태 코드와 일치시키려고 합니다. 이 경우 통합 응답 키가 정규식으로 사용됩니다. 일치 항목을 찾을 수 없으면 `$default`가 통합 응답으로 선택됩니다.
- 템플릿 선택 표현식은 위에서 설명한 대로 작동합니다. 예:
  - `/2\d\d/`: 성공적인 응답 수신 및 변환
  - `/4\d\d/`: 잘못된 요청 오류 수신 및 변환
  - `$default`: 예기치 않은 모든 응답 수신 및 변환

템플릿 선택 표현식에 대한 자세한 내용은 the section called “템플릿 선택 표현식” 섹션을 참조하세요.

API Gateway 콘솔을 사용하여 통합 응답 설정

API Gateway 콘솔을 사용하여 WebSocket API에 대한 경로 통합 응답을 설정하려면:

1. https://console.aws.amazon.com/apigateway에서 API Gateway 콘솔에 로그인합니다.
2. WebSocket API를 선택하고 경로를 선택합니다.
3. 통합 요청 탭을 선택한 다음 통합 응답 설정 섹션에서 통합 응답 생성을 선택합니다.
4. 응답 키에 응답 선택 표현식을 평가한 후 보내는 메시지에 있는 응답 키에서 찾을 수 있는 값을 입력합니다. 예를 들어 잘못된 요청 오류를 수신하고 변환하려면 **/4\d\d/**를 입력하고 템플릿 선택 표현식과 일치하는 모든 응답을 수신하고 변환하려면 **$default**를 입력할 수 있습니다.
5. 템플릿 선택 표현식에 보내는 메시지를 평가하기 위한 선택 표현식을 입력합니다.
6. 응답 생성을 선택합니다.
7. 매핑 템플릿을 정의하여 반환된 메시지 페이로드의 변환을 구성할 수도 있습니다. 템플릿 생성을 선택합니다.
8. 키 이름을 입력합니다. 기본 템플릿 선택 표현식을 선택하는 경우 **\$default**를 입력합니다.
9. 응답 템플릿에서 코드 편집기에 매핑 템플릿을 입력합니다.
10. 템플릿 생성을 선택합니다.
11. API 배포를 선택하여 API를 배포합니다.

API에 연결하려면 다음 wscat 명령을 사용합니다. `wscat`에 대한 자세한 정보는을 참조하세요the section called “`wscat`를 사용하여 WebSocket API에 연결하고 메시지 보내기”.

```
wscat -c wss://api-id.execute-api.us-east-2.amazonaws.com/test
```

경로를 호출하면 반환된 메시지 페이로드가 반환됩니다.

를 사용하여 통합 응답을 설정합니다. AWS CLI

WebSocket API에 대한 통합 응답을 설정하려면 `create-integration-response`명령을 AWS CLI 호출하십시오. 다음 CLI 명령은 `$default` 통합 응답을 생성하는 예를 보여줍니다.

```
aws apigatewayv2 create-integration-response \
    --api-id vaz7da96z6 \
    --integration-id a1b2c3 \
    --integration-response-key '$default'
```

요청 검증

통합 요청을 진행하기 전에 라우팅 요청의 확인을 수행하도록 API Gateway를 구성할 수 있습니다. 검증이 실패하면 API Gateway는 백엔드를 호출하지 않고 요청에 실패하고, “잘못된 요청 본문” 게이트웨이 응답을 클라이언트에 보내고, 검증 결과를 Logs에 CloudWatch 게시합니다. 이런 식으로 유효성 검사를 이용하면 API 백엔드에 대한 불필요한 호출을 줄일 수 있습니다.

모델 선택 표현식

모델 선택 표현식을 사용하여 동일한 라우팅 내에서 요청을 동적으로 유효성 검사할 수 있습니다. 프록시 또는 비프록시 통합에 대한 모델 선택 표현식을 제공하는 경우 모델 유효성 검사가 수행됩니다. 일치하는 모델이 없으면 `$default` 모델을 폴백으로 정의해야 할 수도 있습니다. 일치하는 모델이 없고 `$default`가 정의되지 않으면 유효성 검사가 실패합니다. 선택 표현식은 `Route.ModelSelectionExpression`과 같고, `Route.RequestModels`의 키를 평가합니다.

WebSocket API의 경로를 정의할 때 선택적으로 모델 선택 표현식을 지정할 수 있습니다. 이 표현식은 요청이 수신될 때 본문 확인에 사용될 모델을 선택하기 위해 평가됩니다. 표현식은 라우팅의 `requestmodels`에서 항목 중 하나를 평가합니다.

모델은 JSON 스키마로 표시되고 요청 본문의 데이터 구조를 설명합니다. 이 선택 표현식의 속성상, 실행 시간에 특정 라우팅을 기준으로 유효성 검사를 수행할 모델을 동적으로 선택할 수 있습니다. 모델 생성에 대한 자세한 내용은 the section called “데이터 모델 이해” 단원을 참조하십시오.

API Gateway 콘솔을 사용하여 요청 확인 설정

다음 예제는 라우트에 요청 검증을 설정하는 방법을 보여줍니다.

먼저 모델을 생성한 다음 경로를 생성합니다. 다음으로 방금 생성한 경로에 요청 검증을 구성합니다. 마지막으로 API를 배포하고 테스트합니다. 이 자습서를 완료하려면 경로 선택 표현식으로 사용하는 WebSocket `$request.body.action` API와 새 경로의 통합 엔드포인트가 필요합니다.

또한 API에 연결하기 위해 `wscat`가 필요합니다. 자세한 설명은 the section called “`wscat`를 사용하여 WebSocket API에 연결하고 메시지 보내기” 섹션을 참조하세요.

모델을 생성하는 방법

1. https://console.aws.amazon.com/apigateway에서 API Gateway 콘솔에 로그인합니다.
2. WebSocket API를 선택하세요.
3. 기본 탐색 창에서 모델을 선택합니다.
4. Create model(모델 생성)을 선택합니다.

5. 이름에 **emailModel**를 입력합니다.
6. 콘텐츠 유형에 **application/json**을 입력합니다.
7. 모델 스키마에서 다음 모델을 입력합니다.

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "type" : "object",
    "required" : [ "address"],
    "properties" : {
        "address": {
            "type": "string"
        }
    }
}
```

   이 모델을 사용하려면 요청에 이메일 주소가 포함되어야 합니다.
8. 저장을 선택합니다.

이 단계에서는 WebSocket API의 경로를 생성합니다.

경로 생성

1. 기본 탐색 창에서 [Routes] 를 선택합니다.
2. 경로 생성(Create route)을 선택합니다.
3. 경로 키(Route key)에 **sendMessage**를 입력합니다.
4. 통합 유형을 선택하고 통합 엔드포인트를 지정합니다. 자세한 내용은 the section called “통합” 섹션을 참조하세요.
5. 경로 생성(Create route)을 선택합니다.

이 단계에서는 sendMessage 경로에 대한 요청 검증을 설정합니다.

요청 검증을 설정하려면

1. 라우트 요청 탭의 라우트 요청 설정에서 편집을 선택합니다.
2. 모델 선택 표현식에 을 입력합니다**${request.body.messageType}**.

   API Gateway는 messageType 속성을 사용하여 수신 요청을 검증합니다.

3. 요청 모델 추가를 선택합니다.
4. 모델 키에 를 입력합니다**email**.
5. 모델의 경우 이메일 모델을 선택합니다.

   API Gateway는 이 모델에 email 대해 messageType 속성이 로 설정된 수신 메시지를 검증합니다.

   > **Note**
   >
   > API Gateway가 모델 선택 표현식을 모델 키와 일치시킬 수 없는 경우 $default 모델을 선택합니다. $default모델이 없는 경우 검증이 실패합니다. 프로덕션 API의 경우 $default 모델을 생성하는 것이 좋습니다.

6. 변경 사항 저장를 선택합니다.

이 단계에서는 API를 배포하고 테스트합니다.

API를 배포하고 테스트하려면

1. Deploy API(API 배포)를 선택합니다.
2. 드롭다운 목록에서 원하는 스테이지를 선택하거나 새 스테이지의 이름을 입력하십시오.
3. 배포를 선택합니다.
4. 기본 탐색 창에서 스테이지를 선택합니다.
5. API WebSocket URL을 복사하세요. URL은 wss://*abcdef123*.execute-api.*us-east-2*.amazonaws.com/production와 같아야 합니다.
6. 새 터미널을 열고 다음 파라미터를 wscat 사용하여 명령을 실행합니다.

```
wscat -c wss://abcdef123.execute-api.us-west-2.amazonaws.com/production
```

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

7. 다음 명령어를 사용하여 API를 테스트하세요.

```
{"action": "sendMessage", "messageType": "email"}
```

```
{"message": "Invalid request body", "connectionId":"ABCD1=234",
 "requestId":"EFGH="}
```

API Gateway는 요청을 실패하게 됩니다.

다음 명령어를 사용하여 API에 유효한 요청을 보내세요.

```
{"action": "sendMessage", "messageType": "email", "address":
 "mary_major@example.com"}
```

WebSocket API에 대한 데이터 변환 설정

API Gateway에서 WebSocket API 메서드 요청은 백엔드의 요구에 따라 해당하는 통합 요청 페이로드에서 여러 유형의 페이로드를 취할 수 있습니다. 마찬가지로 백엔드에서는 프런트 엔드에서 기대하는 메서드 응답 페이로드가 아니라 통합 응답 페이로드를 반환할 수 있습니다.

API Gateway에서는 매핑 템플릿을 사용하여 방법 요청에서 해당 통합 요청으로, 통합 요청에서 해당 방법 요청으로 페이로드를 매핑할 수 있습니다. 템플릿 선택 표현식을 지정하여 필요한 데이터 변환을 수행하는 데 사용할 템플릿을 결정합니다.

데이터 매핑을 사용하면 라우팅 요청의 데이터를 백엔드 통합에 매핑할 수 있습니다. 자세한 내용은 the section called “데이터 매핑” 단원을 참조하십시오.

매핑 템플릿 및 모델

매핑 템플릿이란 VTL(Velocity Template Language)로 표현된 스크립트로, JSONPath 표현식을 사용하여 페이로드에 적용됩니다. API Gateway 매핑 템플릿에 대한 자세한 내용은 매핑 템플릿 이해 단원을 참조하세요.

페이로드는 JSON 스키마 draft 4에 따라 데이터 모델을 가질 수 있습니다. 매핑 템플릿을 생성하기 위해 모델을 정의할 필요는 없습니다. 그러나 API Gateway에서는 제공된 모델에 따라 템플릿 블루프린트를 생성하기 때문에 모델이 있으면 템플릿을 생성하는 데 도움이 됩니다. API Gateway 모델에 대한 자세한 내용은 데이터 모델 이해 단원을 참조하세요.

템플릿 선택 표현식

매핑 템플릿을 사용하여 페이로드를 변환하려면 통합 요청 또는 통합 응답에서 WebSocket API 템플릿 선택 표현식을 지정합니다. 이 표현식은 요청 본문을 (입력 템플릿을 통해) 요청 본문으로 변환하거

나 통합 응답 본문을 (출력 템플릿을 통해) 라우팅 응답 본문으로 변환하는 데 사용할 입력 또는 출력 템플릿(있는 경우)를 판별하기 위해 평가됩니다.

`Integration.TemplateSelectionExpression`은 `${request.body.jsonPath}`와 정적 값을 지원합니다.

`IntegrationResponse.TemplateSelectionExpression`은 `${request.body.jsonPath}`, `${integration.response.statuscode}`, `${integration.response.header.headerName}`, `${integration.response.multivalueheader.headerName}` 및 정적 값을 지원합니다.

통합 응답 선택 표현식

WebSocket API에 대해 통합 응답을 설정할 때 선택적으로 통합 응답 선택 표현식을 지정할 수 있습니다. 이 표현식은 통합이 반환될 때 어떤 `IntegrationResponse`가 선택되어야 하는지 결정합니다. 이 표현식의 값은 현재 아래에 정의된 대로 API 게이트웨이에 의해 제한됩니다. 이 표현식은 비 프록시 통합에만 해당한다는 점에 유의하십시오. 프록시 통합은 모델링이나 수정 없이 단순히 응답 페이로드를 호출자에게 다시 전달하기만 합니다.

이전의 다른 선택 표현식과 달리 이 표현식은 현재 패턴 일치 형식을 지원합니다. 표현식은 슬래시로 래핑해야 합니다.

현재 이 값은 `integrationType`에 따라 고정됩니다.

- Lambda 기반 통합의 경우 `$integration.response.body.errorMessage`입니다.
- `HTTP` 및 `MOCK` 통합의 경우 `$integration.response.statuscode`입니다.
- `HTTP_PROXY` 및 `AWS_PROXY`의 경우 페이로드가 호출자에게 패스스루 되도록 요청하기 때문에 표현식이 사용되지 않습니다.

WebSocket API에 대한 데이터 매핑 설정

데이터 매핑을 사용하면 라우팅 요청의 데이터를 백엔드 통합에 매핑할 수 있습니다.

> **Note**
>
> 웹 소켓 API에 대한 데이터 매핑은 에서 지원되지 않습니다AWS Management Console 데이터 매핑을 구성하려면 AWS CLI, AWS CloudFormation 또는 SDK를 사용해야 합니다.

주제

- 통합 요청 파라미터에 라우팅 요청 데이터 매핑
- 예시

통합 요청 파라미터에 라우팅 요청 데이터 매핑

통합 요청 파라미터는 정의된 라우팅 요청 파라미터, 요청 본문, context 또는 stage 변수 및 정적 값에서 매핑될 수 있습니다.

다음 표에서 *PARAM_NAME*은 지정된 파라미터 유형의 라우팅 요청 파라미터 이름입니다. 정규식 `'^[a-zA-Z0-9._$-]+$]'`와 일치해야 합니다. *JSONPath_EXPRESSION*은 요청 본문의 JSON 필드에 대한 JSONPath 표현식입니다.

통합 요청 데이터 매핑 표현식

매핑된 데이터 원본	매핑 표현식
요청 쿼리 문자열($connect 라우팅의 경우에만 지원)	`route.request.querystring.` *PARAM_NAME*
요청 헤더($connect 라우팅의 경우에만 지원)	`route.request.header.` *PARAM_NAME*
다중 값 요청 쿼리 문자열($connect 라우팅의 경우에만 지원)	`route.request.multivaluequerystring.` *PARAM_NAME*
다중 값 요청 헤더($connect 라우팅의 경우에만 지원)	`route.request.multivalueheader.` *PARAM_NAME*
요청 본문	`route.request.body.` *JSONPath_EXPRESSION*
단계 변수	`stageVariables.` *VARIABLE_NAME*
컨텍스트 변수	`context.`*VARIABLE_NAME* 은 지원되는 컨텍스트 변수 중 하나여야 합니다.
정적 값	*'STATIC_VALUE'* . *STATIC_VALUE* 는 문자열 리터럴이며 작은따옴표로 묶여야 합니다.

예시

다음 AWS CLI 예제에서는 데이터 매핑을 구성합니다. AWS CloudFormation 템플릿 예시를 보려면 websocket-data-mapping.yaml 단원을 참조하세요.

통합 요청의 헤더에 클라이언트의 connectionId 매핑

다음 예제 명령은 백엔드 통합에 대한 요청의 connectionId 헤더에 클라이언트의 connectionId를 매핑합니다.

```
aws apigatewayv2 update-integration \
    --integration-id abc123 \
    --api-id a1b2c3d4 \
    --request-parameters
 'integration.request.header.connectionId'='context.connectionId'
```

통합 요청의 헤더에 쿼리 문자열 파라미터 매핑

다음 예제 명령은 통합 요청의 authToken 헤더에 authToken 쿼리 문자열 파라미터를 매핑합니다.

먼저 라우팅의 요청 파라미터에 authToken 쿼리 문자열 파라미터를 추가합니다.

```
aws apigatewayv2 update-route --route-id 0abcdef \
    --api-id a1b2c3d4 \
    --request-parameters '{"route.request.querystring.authToken": {"Required": false}}'
```

그런 다음 백엔드 통합에 대한 요청의 authToken 헤더에 쿼리 문자열 파라미터를 매핑합니다.

```
aws apigatewayv2 update-integration \
    --integration-id abc123 \
    --api-id a1b2c3d4 \
    --request-parameters
 'integration.request.header.authToken'='route.request.querystring.authToken'
```

필요한 경우 경로의 요청 파라미터에서 authToken 쿼리 문자열 파라미터를 삭제합니다.

```
aws apigatewayv2 delete-route-request-parameter \
    --route-id 0abcdef \
    --api-id a1b2c3d4 \
    --request-parameter-key 'route.request.querystring.authToken'
```

API Gateway WebSocket API 매핑 템플릿 참조

이 섹션에서는 현재 API Gateway에서 WebSocket API에 대해 지원되는 변수 집합을 요약합니다.

파라미터	설명
`$context.connectionId`	클라이언트에 대한 콜백을 만드는 데 사용할 수 있는 고유한 연결 ID입니다.
`$context.connectedAt`	Epoch 형식 연결 시간입니다.
`$context.domainName`	WebSocket API의 도메인 이름입니다. 하드 코딩된 값 대신 클라이언트에 콜백하는 데 사용할 수 있습니다.
`$context.eventType`	이벤트 유형: `CONNECT`, `MESSAGE` 또는 `DISCONNECT` .
`$context.messageId`	메시지의 고유 서버 측 ID. `$context.eventType` 이 `MESSAGE`인 경우에만 사용 가능합니다.
`$context.routeKey`	선택한 라우팅 키.
`$context.requestId`	`$context.extendedRequestId` 와 동일합니다.
`$context.extendedRequestId`	API 호출에 대해 자동으로 생성된 ID로 디버깅/문제 해결에 유용한 정보가 포함합니다.
`$context.apiId`	식별자 API Gateway가 API에 할당합니다.
`$context.authorizer.principalId`	클라이언트가 전송한 토큰과 연결되고 API Gateway Lambda 권한 부여자(이전에는 사용자 지정 권한 부여자라고 함) Lambda 함수에서 반환한 보안 주체 사용자 식별입니다.
`$context.authorizer.` *`property`*	API Gateway Lambda 권한 부여자 함수에서 반환된 `context` 맵의 지정된 키-값 페어의 문자열

파라미터	설명
	화된 값입니다. 예를 들어, 권한 부여자는 다음 context 맵을 반환합니다. `"context" : { "key": "value", "numKey": 1, "boolKey": true }` $context.authorizer.key 를 호출하면 "value" 문자열이 반환되고, $context.authorizer.numKey 를 호출하면 "1" 문자열이 반환되고, $context.authorizer.boolKey 를 호출하면 "true" 문자열이 반환됩니다.
$context.error.messageString	$context.error.message 의 따옴표 붙은 값, 즉 "$context.error.message" 입니다.
$context.error.validationErrorString	세부적인 검증 오류 메시지를 포함하는 문자열입니다.
$context.identity.accountId	요청과 연결된 AWS 계정 ID입니다.
$context.identity.apiKey	키가 활성화된 API 요청에 연결된 API 소유자 키입니다.
$context.identity.apiKeyId	키가 활성화된 API 요청에 연결된 API 키 ID입니다.
$context.identity.caller	요청을 작성한 호출자의 보안 주체 ID입니다.

파라미터	설명
`$context.identity.cognitoAuthenticationProvider`	요청을 작성한 호출자가 사용한 Amazon Cognito 인증 공급자의 쉼표로 구분된 목록입니다. Amazon Cognito 자격 증명으로 요청을 서명한 경우에만 사용할 수 있습니다. 예를 들어, Amazon Cognito 사용자 풀의 자격 증명의 경우 `cognito-idp.region.amazonaws.com/user_pool_id ,cognito-idp.region.amazonaws.com/user_pool_id :CognitoSignIn: token subject claim` 자세한 내용은 Amazon Cognito 개발자 안내서의 연동 자격 증명 사용을 참조하세요.
`$context.identity.cognitoAuthenticationType`	요청을 작성한 호출자의 Amazon Cognito 인증 유형입니다. Amazon Cognito 자격 증명으로 요청을 서명한 경우에만 사용할 수 있습니다. 가능한 값에는 인증 자격 증명에 대한 `authenticated` 및 미인증 자격 증명에 대한 `unauthenticated` 이(가) 포함됩니다.
`$context.identity.cognitoIdentityId`	요청을 작성한 호출자의 Amazon Cognito 자격 증명 ID입니다. Amazon Cognito 자격 증명으로 요청을 서명한 경우에만 사용할 수 있습니다.
`$context.identity.cognitoIdentityPoolId`	요청을 작성한 호출자의 Amazon Cognito 자격 증명 풀 ID입니다. Amazon Cognito 자격 증명으로 요청을 서명한 경우에만 사용할 수 있습니다.
`$context.identity.sourceIp`	API Gateway 엔드포인트를 요청하는 즉시 TCP 연결의 소스 IP 주소입니다.
`$context.identity.user`	요청을 작성한 사용자의 보안 주체 ID입니다.
`$context.identity.userAgent`	API 호출자의 사용자 에이전트입니다.

파라미터	설명
`$context.identity.userArn`	인증 후 식별된 실제 사용자의 ARN(Amazon Resource Name)입니다.
`$context.requestTime`	CLF 형식 요청 시간(`dd/MMM/yyyy:HH:mm:ss +-hhmm`)입니다.
`$context.requestTimeEpoch`	Epoch 형식 요청 시간(밀리초)입니다.
`$context.stage`	API 호출의 개발 단계입니다(예: 베타 또는 프로덕션).
`$context.status`	응답 상태입니다.
`$input.body`	원시 페이로드를 문자열로 반환합니다.
`$input.json(x)`	이 함수는 JSONPath 표현식을 평가하고 결과를 JSON 문자열로 반환합니다. 예를 들어, `$input.json('$.pets')` 은 반려 동물 구조를 나타내는 JSON 문자열을 반환합니다. JSONPath에 대한 자세한 내용은 JSONPath 또는 Java용 JSONPath를 참조하십시오.

파라미터	설명
`$input.path(x)`	JSONPath 표현식 문자열 (x)을 가져와서 결과의 JSON 객체로 반환합니다. 이를 통해 기본적으로 Apache Velocity Template Language(VTL)에서 페이로드 요소에 액세스하고 조작할 수 있습니다. 예를 들어, `$input.path('$.pets')` 표현식이 다음과 같은 객체를 반환할 경우: `[ { "id": 1, "type": "dog", "price": 249.99 }, { "id": 2, "type": "cat", "price": 124.99 }, { "id": 3, "type": "fish", "price": 0.99 } ]` `$input.path('$.pets').count()` 는 을 반환합니다."3" JSONPath에 대한 자세한 내용은 JSONPath 또는 Java용 JSONPath를 참조하십시오.
`$stageVariables.` *<variable_name>*	*<variable_name>* 은 단계 변수 이름을 나타냅니다.
`$stageVariables['` *<variable_name>* `']`	*<variable_name>* 은 모든 단계 변수 이름을 나타냅니다.

파라미터	설명
`${stageVariables['` *`<variable_name>`* `']}`	*`<variable_name>`* 은 모든 단계 변수 이름을 나타냅니다.
`$util.escapeJavaScript()`	JavaScript 문자열 규칙을 사용하여 문자열의 문자를 이스케이프합니다. **Note** 이 함수는 일반 작은따옴표(')를 이스케이프된 작은따옴표(\')로 변환합니다. 그러나 이스케이프된 작은따옴표는 JSON에서 유효하지 않습니다. 따라서 이 함수의 결과를 JSON 속성에서 사용할 경우 이스케이프된 작은따옴표(\')를 다시 일반 작은따옴표(')로 변환해야 합니다. 방법은 다음 예제와 같습니다: `$util.escapeJavaScript( data).replaceAll("\\'",'')`

파라미터	설명

`$util.parseJson()`	"문자열화된" JSON을 가져와서 결과의 객체 표현을 반환합니다. 이 함수의 결과를 사용하여 기본적으로 Apache VTL(Velocity Template Language)에서 페이로드 요소에 액세스하고 조작할 수 있습니다. 예를 들어 다음과 같은 페이로드가 있고
	`{"errorMessage":"{\"key1\":\"var1\", \"key2\":{\"arr\":[1,2,3]}}"}`
	다음 매핑 템플릿을 사용하는 경우
	`#set ($errorMessageObj = $util.parseJson($input.path('$.errorMessage')))`
	`{`
	`"errorMessageObjKey2ArrVal" : $errorMessageObj.key2.arr[0]`
	`}`
	출력은 다음과 같습니다.
	`{`
	`"errorMessageObjKey2ArrVal" : 1`
	`}`

`$util.urlEncode()`	문자열을 "application/x-www-form-urlencoded" 형식으로 변환합니다.
`$util.urlDecode()`	"application/x-www-form-urlencoded" 문자열을 디코딩합니다.
`$util.base64Encode()`	데이터를 base64 인코딩 문자열로 인코딩합니다.
`$util.base64Decode()`	데이터를 base64 인코딩 문자열에서 디코딩합니다.

WebSocket API에 대한 이진 미디어 유형 작업

API Gateway WebSocket API는 현재 수신 메시지 페이로드에 이진 프레임을 지원하지 않습니다. 클라이언트 앱에서 이진 프레임을 보내면 API Gateway가 거부하고 코드 1003과 함께 클라이언트와의 연결을 끊습니다.

이 동작의 차선책이 있습니다. 클라이언트가 텍스트 인코딩된 이진 데이터(예: Base64)를 텍스트 프레임으로 보내는 경우, 통합의 `contentHandlingStrategy` 속성을 `CONVERT_TO_BINARY`로 설정하여 페이로드를 Base64 인코딩 문자열에서 이진으로 변환할 수 있습니다.

비 프록시 통합에서 이진 페이로드에 대한 라우팅 응답을 반환하기 위해 통합 응답의 `contentHandlingStrategy` 속성을 `CONVERT_TO_TEXT`로 설정하여 페이로드를 이진에서 Base64 인코딩 문자열로 변환할 수 있습니다.

WebSocket API 호출

WebSocket API를 배포하고 나면 클라이언트 애플리케이션이 WebSocket API에 연결하여 메시지를 보낼 수 있으며, 백엔드는 연결된 클라이언트 애플리케이션에 메시지를 보낼 수 있습니다.

- `wscat`를 사용하여 WebSocket API에 연결하고 메시지를 보내 클라이언트 동작을 시뮬레이션할 수 있습니다. the section called “`wscat`를 사용하여 WebSocket API에 연결하고 메시지 보내기”을(를) 참조하세요.
- 백엔드 서비스의 @connections API를 사용하여 연결된 클라이언트에 콜백 메시지를 보내거나 연결 정보를 얻거나 클라이언트 연결을 끊을 수 있습니다. the section called “백엔드 서비스에서 `@connections` 명령 사용”을(를) 참조하세요.
- 클라이언트 애플리케이션은 자체 WebSocket 라이브러리를 사용하여 WebSocket API를 호출할 수 있습니다.

`wscat`를 사용하여 WebSocket API에 연결하고 메시지 보내기

`wscat` 유틸리티는 API Gateway에서 만들고 배포한 WebSocket API를 테스트할 수 있는 편리한 도구입니다. 다음과 같이 `wscat`를 설치하여 사용할 수 있습니다.

1. https://www.npmjs.com/package/wscat에서 `wscat`를 다운로드합니다.
2. 다음 명령을 실행하여 `wscat`을 설치합니다.

```
npm install -g wscat
```

3. API에 연결하려면 다음 예와 같이 `wscat` 명령을 실행하십시오. 이 예제에서는 `Authorization` 설정이 `NONE`이라고 가정합니다.

```
wscat -c wss://aabbccddee.execute-api.us-east-1.amazonaws.com/test/
```

*aabbccddee*를 실제 API ID, 즉 API Gateway 콘솔에 표시되거나 AWS CLI `create-api` 명령에서 반환하는 ID로 바꿔야 합니다.

또한 API가 `us-east-1` 이외의 리전에 있는 경우, 올바른 리전으로 대체해야 합니다.

4. API를 테스트하려면 연결된 상태에서 다음과 같은 메시지를 입력하십시오.

```
{"{jsonpath-expression}":"{route-key}"}
```

여기에서 *{jsonpath-expression}*은 JSONPath 표현식이고 *{route-key}*는 API에 대한 라우팅 키입니다. 예:

```
{"action":"action1"}
{"message":"test response body"}
```

JSONPath에 대한 자세한 내용은 JSONPath 또는 Java용 JSONPath를 참조하십시오.

5. API 연결을 해제하려면 `ctrl-C`를 입력합니다.

백엔드 서비스에서 `@connections` 명령 사용

백엔드 서비스는 다음 WebSocket 연결 HTTP 요청을 사용하여 연결된 클라이언트에 콜백 메시지를 보내거나 연결 정보를 얻거나 클라이언트 연결을 끊을 수 있습니다.

> **Important**
>
> 이러한 요청은 IAM 권한 부여를 사용하므로 서명 버전 4(SigV4)로 서명해야 합니다. 이를 수행하기 위해 API Gateway 관리 API를 사용할 수 있습니다. 자세한 내용은 ApiGatewayManagementApi를 참조하십시오.

다음 명령에서 *{api-id}*를 실제 API ID, 즉 API Gateway 콘솔에 표시되거나 AWS CLI `create-api` 명령에서 반환하는 ID로 바꿔야 합니다. 이 명령을 사용하기 전에 연결을 설정해야 합니다.

클라이언트에게 콜백 메시지를 전송하려면 다음을 사용합니다.

```
POST https://{api-id}.execute-api.us-east-1.amazonaws.com/{stage}/
@connections/{connection_id}
```

Postman을 사용하거나 다음 예와 같이 awscurl를 호출하여 이 요청을 테스트할 수 있습니다.

```
awscurl --service execute-api -X POST -d "hello world" https://{prefix}.execute-api.us-
east-1.amazonaws.com/{stage}/@connections/{connection_id}
```

다음 예제와 같이 명령을 URL 인코딩해야 합니다.

```
awscurl --service execute-api -X POST -d "hello world" https://aabbccddee.execute-
api.us-east-1.amazonaws.com/prod/%40connections/R0oXAdfD0kwCH6w%3D
```

클라이언트의 가장 최신 연결 상태를 얻으려면 다음을 사용합니다.

```
GET https://{api-id}.execute-api.us-east-1.amazonaws.com/{stage}/
@connections/{connection_id}
```

클라이언트의 연결을 해제하려면 다음을 사용합니다.

```
DELETE https://{api-id}.execute-api.us-east-1.amazonaws.com/{stage}/
@connections/{connection_id}
```

통합에 $context 변수를 사용하여 동적으로 콜백 URL을 구성할 수 있습니다. 예를 들어 Lambda 프록시 통합과 Node.js Lambda 함수를 사용하는 경우 다음과 같이 URL을 구성하고 연결된 클라이언트에 메시지를 보낼 수 있습니다.

```
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,
  };
};
```

콜백 메시지를 보낼 때는 Lambda 함수에 API Gateway Management API를 직접 호출할 권한이 있어야 합니다. 연결이 설정되기 전 또는 클라이언트 연결이 끊긴 후에 메시지를 게시하면 `GoneException`을 포함하는 오류가 발생할 수 있습니다.

고객이 호출할 수 WebSocket 있는 API 게시

단순히 API Gateway API를 생성하고 개발하면 자동으로 사용자가 API를 호출할 수 있게 되는 것은 아닙니다. 호출이 가능하려면 API를 스테이지에 배포해야 합니다. 또한 사용자가 API에 액세스하는 데 사용할 URL을 사용자 지정할 수 있습니다. 브랜드와 일치하는 도메인이나 API의 기본 URL보다 기억에 남는 도메인을 지정할 수 있습니다.

이 섹션에서는 API를 배포하고 액세스를 위해 사용자에게 제공하는 URL을 사용자 지정하는 방법에 대해 알아볼 수 있습니다.

> **Note**
>
> API Gateway API의 보안을 강화하기 위해 `execute-api.{region}.amazonaws.com` 도메인은 PSL(Public Suffix List)에 등록됩니다. 보안 강화를 위해 API Gateway API 기본 도메인 이름에 민감한 쿠키를 설정해야 하는 경우 `__Host-` 접두사가 있는 쿠키를 사용하는 것이 좋습니다. 이렇게 하면 교차 사이트 요청 위조 시도 (CSRF) 로부터 도메인을 보호하는 데 도움이 됩니다. 자세한 내용은 Mozilla 개발자 네트워크의 Set-Cookie 페이지를 참조하세요.

주제

- WebSocket API에 대한 스테이지 작업
- API Gateway에서 WebSocket API 배포
- WebSocket API에 대한 보안 정책
- WebSocket API용 사용자 지정 도메인 이름 설정

WebSocket API에 대한 스테이지 작업

API 스테이지는 API의 수명 주기 상태에 대한 논리적 참조(예: `dev`, `prod`, `beta`, `v2`)입니다. API 스테이지는 API ID 및 스테이지 이름으로 식별되며, API를 호출하는 데 사용하는 URL에 포함됩니다. 각 스테이지는 API 배포에 대한 명명된 참조이며, 클라이언트 애플리케이션 프로그램에서 호출을 하는 데 사용할 수 있습니다.

배포는 API 구성의 스냅샷입니다. 단계에 API를 배포한 후에는 클라이언트가 API를 호출할 수 있습니다. 변경 사항을 적용하려면 API를 배포해야 합니다.

단계 변수

스테이지 변수는 WebSocket API의 스테이지에 대해 정의할 수 있는 키 값 페어입니다. 환경 변수와 비슷한 역할을 하며, API 설정에 사용할 수 있습니다.

예를 들어 스테이지 변수를 정의한 다음, 해당 값을 HTTP 프록시 통합을 위한 HTTP 엔드포인트로서 설정할 수 있습니다. 또한 추후에 연결된 스테이지 변수 이름을 사용하여 엔드포인트를 참조할 수 있습니다. 이렇게 하면 각 스테이지의 서로 다른 엔드포인트에서 동일한 API 설정을 사용할 수 있습니다. 마찬가지로 스테이지 변수를 사용하여 API의 각 스테이지에 대해 서로 다른 AWS Lambda 함수 통합을 지정할 수 있습니다.

> **Note**
>
> 단계 변수는 자격 증명과 같은 중요한 데이터에 사용할 수 없습니다. 중요한 데이터를 통합에 전달하려면 AWS Lambda 권한 부여자를 사용합니다. Lambda 권한 부여자의 출력에서 중요한 데이터를 통합에 전달할 수 있습니다. 자세한 내용은 the section called “Lambda 권한 부여자 응답 형식” 단원을 참조하세요.

예제

스테이지 변수를 사용하여 HTTP 통합 엔드포인트를 사용자 지정하려면 먼저 스테이지(예: `url`)의 이름과 값을 `example.com`로 설정해야 합니다. 그런 다음 HTTP 프록시 통합을 설정합니다. 엔드포인

트의 URL을 입력하는 대신, 단계 변수 값인 **http://${stageVariables.url}**을 사용하도록 API Gateway에 지시할 수 있습니다. 이 값은 API의 스테이지에 따라 실행 시간에 스테이지 변수 ${}을 대체하도록 API Gateway에 지시합니다.

비슷한 방법으로 스테이지 변수를 참조하여 Lambda 함수 이름이나 AWS 역할 ARN을 지정할 수 있습니다.

Lambda 함수 이름을 단계 변수 값으로 지정할 때는 Lambda 함수에 대한 권한을 수동으로 구성해야 합니다. 이를 위해 AWS Command Line Interface(AWS CLI)를 사용할 수 있습니다.

```
aws lambda add-permission --function-name arn:aws:lambda:XXXXXX:your-lambda-function-
name --source-arn arn:aws:execute-api:us-east-1:YOUR_ACCOUNT_ID:api_id/*/HTTP_METHOD/
resource --principal apigateway.amazonaws.com --statement-id apigateway-access --action
 lambda:InvokeFunction
```

API Gateway 스테이지 변수 참조

HTTP 통합 URI

다음 예제에서 보듯 스테이지 변수를 HTTP 통합 URI의 일부로 사용할 수 있습니다.

- 프로토콜이 없는 전체 URI - `http://${stageVariables.<variable_name>}`
- 전체 도메인 - `http://${stageVariables.<variable_name>}/resource/operation`
- 하위 도메인 - `http://${stageVariables.<variable_name>}.example.com/resource/operation`
- 경로 - `http://example.com/${stageVariables.<variable_name>}/bar`
- 쿼리 문자열 - `http://example.com/foo?q=${stageVariables.<variable_name>}`

Lambda 함수

다음 예제와 같이 Lambda 함수 이름이나 별칭 대신 스테이지 변수를 사용할 수 있습니다.

- `arn:aws:apigateway:<region>:lambda:path/2015-03-31/functions/arn:aws:lambda:<region>:<account_id>:function:${stageVariables.<function_variable_name>}/invocations`
- `arn:aws:apigateway:<region>:lambda:path/2015-03-31/functions/arn:aws:lambda:<region>:<account_id>:function:<function_name>:${stageVariables.<version_variable_name>}/invocations`

> **Note**
>
> Lambda 함수에 대해 단계 변수를 사용하려면 함수가 API와 동일한 계정에 있어야 합니다. 단계 변수는 교차 계정 Lambda 함수를 지원하지 않습니다.

AWS 통합 자격 증명

다음 예제와 같이 스테이지 변수를 AWS 사용자 또는 역할 자격 증명 ARN의 일부로 사용할 수 있습니다.

- `arn:aws:iam::<account_id>:${stageVariables.<variable_name>}`

API Gateway에서 WebSocket API 배포

WebSocket API를 생성한 후, 사용자가 호출할 수 있도록 배포해야 합니다.

API를 배포하려면 API 배포를 생성해 단계에 연결합니다. 각 단계는 API의 스냅샷이며 클라이언트 앱이 호출할 수 있습니다.

> **Important**
>
> API를 업데이트할 때마다 해당 API를 재배포해야 합니다. 스테이지 설정 이외의 내용을 변경하려면 다음 리소스에 대한 수정을 포함하여 재배포가 필요합니다.
>
> - 경로
> - 통합
> - 권한 부여자
>
> 기본적으로 API 하나당 10 스테이지로 제한됩니다. 배포에 스테이지를 재사용하는 것이 좋습니다.

배포된 WebSocket API를 호출하기 위해 클라이언트는 API의 URL로 메시지를 보냅니다. URL은 API의 호스트 이름과 스테이지 이름에 의해 결정됩니다.

> **Note**
>
> API Gateway는 최대 128KB의 페이로드를 지원하며 최대 프레임 크기는 32KB입니다. 메시지가 32KB를 초과하면 32KB 이하의 여러 프레임으로 분할되어야 합니다.

API의 기본 도메인 이름을 사용하면 지정된 단계(*{stageName}*)의 WebSocket API URL은 다음 형식이 됩니다.

```
wss://{api-id}.execute-api.{region}.amazonaws.com/{stageName}
```

WebSocket API의 URL을 사용자에게 더욱 친숙하게 만들려면 사용자 지정 도메인 이름(예: `api.example.com`)을 생성하여 API의 기본 호스트 이름을 대체할 수 있습니다. 이 구성 프로세스는 REST API의 프로세스와 동일합니다. 자세한 내용은 the section called “사용자 지정 도메인 이름” 단원을 참조하세요.

단계를 통해 API의 강력한 버전 관리가 가능합니다. 예를 들어 API를 `test` 단계와 `prod` 단계에 배포하고, `test` 단계를 테스트 빌드로, `prod` 단계를 안정적 빌드로 사용할 수 있습니다. 업데이트가 테스트를 통과한 후 `test` 단계를 `prod` 단계로 승격할 수 있습니다. 프로모션은 API를 `prod` 스테이지로 재배포하여 수행할 수 있습니다. 스테이지에 대한 자세한 내용은 the section called “단계 설정” 단원을 참조하십시오 .

주제

- AWS CLI를 사용하여 WebSocket API 배포 생성
- API Gateway 콘솔을 사용하여 WebSocket API 배포 만들기

AWS CLI를 사용하여 WebSocket API 배포 생성

AWS CLI를 사용해 배포를 생성하려면 다음 예와 같이 `create-deployment` 명령을 사용합니다.

```
aws apigatewayv2 --region us-east-1 create-deployment --api-id aabbccddee
```

출력 예:

```
{
    "DeploymentId": "fedcba",
    "DeploymentStatus": "DEPLOYED",
    "CreatedDate": "2018-11-15T06:49:09Z"
```

```
}
```

배포를 스테이지에 연결할 때까지는 배포된 API를 호출할 수 없습니다. 새 스테이지를 만들거나 이전에 만든 스테이지를 재사용할 수 있습니다.

새 스테이지를 만들어 배포에 연결하려면 다음 예와 같이 create-stage 명령을 사용합니다.

```
aws apigatewayv2 --region us-east-1 create-stage --api-id aabbccddee --deployment-id
 fedcba --stage-name test
```

출력 예:

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

기존 스테이지를 다시 사용하려면 deploymentId 명령을 사용하여 스테이지의 *{deployment-id}* 속성을 새로 생성한 배포 ID(update-stage)로 업데이트합니다.

```
aws apigatewayv2 update-stage --region {region} \
   --api-id {api-id} \
   --stage-name {stage-name} \
   --deployment-id {deployment-id}
```

API Gateway 콘솔을 사용하여 WebSocket API 배포 만들기

API Gateway 콘솔을 사용하여 WebSocket API에 대한 배포를 만들려면:

1. API Gateway 콘솔에 로그인하고 API를 선택합니다.
2. Deploy API(API 배포)를 선택합니다.

3. 드롭다운 목록에서 원하는 스테이지를 선택하거나 새 스테이지의 이름을 입력하십시오.

WebSocket API에 대한 보안 정책

API Gateway는 모든 WebSocket API TLS_1_2 엔드포인트에 대해 보안 정책을 적용합니다.

보안 정책은 Amazon API Gateway에서 제공하는 최소 TLS 버전과 암호 제품군의 사전 정의된 조합입니다. TLS 프로토콜은 클라이언트와 서버 간의 변조 및 도청과 같은 네트워크 보안 문제를 해결합니다. 클라이언트가 사용자 지정의 도메인을 통해 API에 TLS 핸드셰이크를 설정하면 보안 정책은 클라이언트가 선택할 수 있는 TLS 버전 및 암호 제품군 옵션을 적용합니다. 이 보안 정책은 TLS 1.2 및 TLS 1.3 트래픽은 허용하고 TLS 1.0 트래픽은 거부합니다.

API에 지원되는 TLS 프로토콜 및 암호 WebSocket

다음 표에는 API에 지원되는 TLS 프로토콜 및 암호가 설명되어 있습니다. WebSocket

보안 정책	TLS_1_2
TLS 프로토콜	
TLSv1.3	♦
TLSv1.2	♦
TLS 암호	
TLS_AES_128_GCM_SHA256	♦
TLS_AES_256_GCM_SHA384	♦
TLS_CHACHA20_POLY1305_SHA256	♦
ECDHE-ECDSA-AES128-GCM-SHA256	♦
ECDHE-RSA-AES128-GCM-SHA256	♦
ECDHE-ECDSA-AES128-SHA256	♦
ECDHE-RSA-AES128-SHA256	♦
ECDHE-ECDSA-AES256-GCM-SHA384	♦

보안 정책	TLS_1_2
ECDHE-RSA-AES256-GCM-SHA384	♦
ECDHE-ECDSA-AES256-SHA384	♦
ECDHE-RSA-AES256-SHA384	♦
AES128-GCM-SHA256	♦
AES128-SHA256	♦
AES256-GCM-SHA384	♦
AES256-SHA256	♦

OpenSSL 및 RFC 암호 이름

OpenSSL과 IETF RFC 5246은 동일한 암호에 다른 이름을 사용합니다. 암호 이름 목록은 을 참조하십시오. the section called “OpenSSL 및 RFC 암호 이름”

REST API 및 HTTP API에 대한 정보

REST API 및 HTTP API에 대한 자세한 내용은 및 을 참조하십시오. the section called “보안 정책 선택” the section called “HTTP API에 대한 보안 정책”

WebSocket API용 사용자 지정 도메인 이름 설정

사용자 지정 도메인 이름은 API 사용자에게 제공할 수 있는 더 간단하고 직관적인 URL입니다.

API를 배포한 후 사용자 및 사용자 고객은 다음 형식의 기본 URL을 사용하여 API를 호출할 수 있습니다.

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

여기서 `api-id` 는 API Gateway에서 생성하며, `region` (AWS 지역) 은 API를 생성할 때 사용자가 지정하고`stage`, API를 배포할 때 사용자가 지정합니다.

URL의 호스트 이름 부분(즉, *`api-id`*`.execute-api.`*`region`*`.amazonaws.com`)은 API 엔드포인트를 가리킵니다. 기본 API 엔드포인트는 기억하기가 어려우며 사용자에게 친숙하지 않을 수 있습니다.

사용자 지정 도메인 이름을 사용하면 API의 호스트 이름을 설정하고 기본 경로(예: `myservice`)를 선택하여 대체 URL을 API에 매핑할 수 있습니다. 예를 들어, 더 사용자 친화적인 API 기본 URL은 다음과 같습니다.

```
https://api.example.com/myservice
```

> **Note**
>
> API의 사용자 지정 도메인 이름은 WebSocket REST API 또는 HTTP API에 매핑할 수 없습니다.
> WebSocket API의 경우 지역별 사용자 지정 도메인 이름이 지원됩니다.
> WebSocket API의 경우 TLS 1.2만 지원되는 TLS 버전입니다.

도메인 이름 등록

API에 대한 사용자 지정 도메인 이름을 설정하려면 등록된 인터넷 도메인 이름이 있어야 합니다. 필요한 경우 Amazon Route 53를 사용하거나 사용자가 선택한 서드 파티 도메인 등록자를 사용하여 인터넷 도메인을 등록할 수 있습니다. API의 사용자 지정 도메인 이름은 하위 도메인의 이름이거나 등록된 인터넷 도메인의 루트 도메인("zone apex"라고도 함) 이름일 수 있습니다.

API Gateway에서 사용자 지정 도메인 이름을 생성한 후에는 DNS 공급자의 리소스 레코드를 생성하거나 업데이트하여 API 엔드포인트에 매핑해야 합니다. 이렇게 매핑하지 않으면 사용자 지정 도메인 이름이 목적지인 API 요청은 API Gateway에 도달할 수 없습니다.

리전 사용자 지정 도메인 이름

리전 API에 대해 사용자 지정 도메인 이름을 생성하면 API Gateway는 해당 API에 대해 리전 도메인 이름을 생성합니다. DNS 레코드를 설정하여 사용자 지정 도메인 이름이 리전 도메인 이름을 가리키도록 해야 합니다. 또한 사용자 지정 도메인 이름에 대한 인증서를 제출해야 합니다.

와일드카드 사용자 정의 도메인 이름

와일드카드 사용자 지정 도메인 이름을 사용하면 기본 할당량을 초과하지 않고 거의 무제한 수의 도메인 이름을 지원할 수 있습니다. 예를 들어 각 고객에게 고유한 도메인 이름인 *customername*`.api.example.com`을(를) 제공할 수 있습니다.

와일드카드 사용자 지정 도메인 이름을 생성하려면, 와일드카드(*)를 루트 도메인의 가능한 모든 하위 도메인을 나타내는 사용자 지정 도메인의 첫 번째 하위 도메인으로 지정할 수 있습니다.

예를 들어 와일드카드 사용자 정의 도메인 이름 `*.example.com`을 사용하면 `a.example.com`, `b.example.com` 및 `c.example.com` 같은 하위 도메인이 모두 동일한 도메인으로 라우팅됩니다.

와일드카드 사용자 지정 도메인 이름은 API Gateway의 표준 사용자 지정 도메인 이름과 별개의 구성을 지원합니다. 예를 들어 단일 AWS 계정에서 다르게 `*.example.com` 구성하고 `a.example.com` 작동하도록 할 수 있습니다.

`$context.domainName` 및 `$context.domainPrefix` 컨텍스트 변수를 사용하여 클라이언트가 API를 호출하는 데 사용한 도메인 이름을 확인할 수 있습니다. 컨텍스트 변수에 대해 자세히 알아보려면 API Gateway 매핑 템플릿과 액세스 로깅 변수 참조 단원을 참조하십시오.

와일드카드 사용자 지정 도메인 이름을 생성하려면 DNS 또는 이메일 검증 방법을 사용하여 검증한 ACM에서 발급한 인증서를 제공해야 합니다.

> **Note**
>
> 다른 AWS 계정에서 생성한 사용자 지정 도메인 이름이 와일드카드 사용자 지정 도메인 이름과 충돌하는 경우에는 와일드카드 사용자 지정 도메인 이름을 만들 수 없습니다. 예를 들어 계정 A에서 `a.example.com`가 생성된 경우, 계정 B는 와일드카드 사용자 지정 도메인 이름 `*.example.com`을 생성할 수 없습니다.
> 계정 A와 계정 B가 한 소유자를 공유하는 경우 AWS 지원 센터에 문의하여 예외를 요청할 수 있습니다.

사용자 지정 도메인 이름에 대한 인증서

> **Important**
>
> 사용자 지정 도메인 이름에 대한 인증서를 지정합니다. 애플리케이션이 인증서 고정 (SSL 피닝이라고도 함) 을 사용하여 ACM 인증서를 고정하는 경우 인증서를 갱신한 후 애플리케이션이 도메인에 연결하지 못할 수 있습니다. AWS 자세한 내용은 AWS Certificate Manager 사용 설명서의 인증서 고정 문제를 참조하세요.

ACM을 지원하는 리전에서 사용자 지정 도메인 이름에 대한 인증서를 제공하려면 ACM에 인증서를 요청해야 합니다. ACM을 지원하지 않는 리전에서 리전 사용자 지정 도메인 이름에 대한 인증서를 제공하려면 해당 리전의 API Gateway로 인증서를 가져와야 합니다.

SSL/TLS 인증서를 가져오려면 PEM 형식의 SSL/TLS 인증서 본문, 해당하는 프라이빗 키 및 사용자 지정 도메인 이름에 대한 인증서 체인을 제공해야 합니다. ACM에 저장된 각 인증서는 ARN으로 식별됩니다. 도메인 이름에 AWS 관리형 인증서를 사용하려면 해당 ARN을 참조하기만 하면 됩니다.

ACM을 사용하면 API에 대한 사용자 지정 도메인 이름을 간편하게 설정하고 사용할 수 있습니다. 지정된 도메인 이름에 대한 인증서를 생성하거나 인증서를 가져오고, API Gateway에서 ACM이 제공한 인증서의 ARN을 사용하여 도메인 이름을 설정한 다음, 사용자 지정 도메인 이름 아래의 기본 경로를 API의 배포된 단계에 매핑합니다. ACM에서 발행한 인증서를 사용하면 프라이빗 키 같은 민감한 인증서 세부 정보의 공개를 걱정할 필요가 없습니다.

사용자 지정 도메인 이름 설정

사용자 지정 도메인 이름 설정에 대한 자세한 내용은 AWS Certificate Manager에서 인증서 준비하기 및 API Gateway에서 리전 사용자 지정 도메인 이름 설정 단원을 참조하십시오.

WebSocket API에 대한 API 매핑 작업

API 매핑을 사용하여 API 스테이지를 사용자 지정 도메인 이름에 연결합니다. 도메인 이름을 생성하고 DNS 레코드를 구성한 후에는 API 매핑을 사용하여 사용자 지정 도메인 이름을 통해 API로 트래픽을 보냅니다.

API 매핑은 API, 스테이지 및 매핑에 사용할 경로(선택 사항)를 지정합니다. 예를 들어 API의 `production` 스테이지를 `wss://api.example.com/orders`에 매핑할 수 있습니다.

API 매핑을 생성하기 전에 API, 스테이지 및 사용자 지정 도메인 이름이 있어야 합니다. 사용자 지정 도메인 이름 생성에 대한 자세한 내용은 the section called “리전 사용자 지정 도메인 이름 설정” 단원을 참조하세요.

Restrictions

- API 매핑에서 사용자 지정 도메인 이름과 매핑된 API는 동일한 AWS 계정에 있어야 합니다.
- API 매핑에는 문자, 숫자 및 문자 `$-_.+!*'()`만 포함해야 합니다.
- API 매핑에서 경로의 최대 길이는 300자입니다.
- WebSocket API를 HTTP API 또는 REST API와 동일한 사용자 지정 도메인 이름에 매핑할 수 없습니다.

API 매핑 생성

API 매핑을 생성하려면 먼저 사용자 지정 도메인 이름, API 및 스테이지를 생성해야 합니다. 사용자 지정 도메인 이름 생성에 대한 자세한 내용은 the section called “리전 사용자 지정 도메인 이름 설정” 단원을 참조하세요.

AWS Management Console

API 매핑 생성

1. https://console.aws.amazon.com/apigateway에서 API Gateway 콘솔에 로그인합니다.
2. 사용자 지정 도메인 이름을 선택합니다.
3. 이미 생성한 사용자 지정 도메인 이름을 선택합니다.
4. API 매핑을 선택합니다.
5. API 매핑 구성을 선택합니다.
6. 새 매핑 추가를 선택합니다.
7. API, 스테이지 및 경로(선택 사항)를 입력합니다.
8. 저장을 선택합니다.

AWS CLI

다음 AWS CLI 명령은 API 매핑을 생성합니다. 이 예에서 API Gateway는 지정된 API 및 스테이지로 `api.example.com/v1` 요청을 보냅니다.

```
aws apigatewayv2 create-api-mapping \
    --domain-name api.example.com \
    --api-mapping-key v1 \
    --api-id a1b2c3d4 \
    --stage test
```

AWS CloudFormation

다음 AWS CloudFormation 예에서는 API 매핑을 생성합니다.

```
MyApiMapping:
  Type: 'AWS::ApiGatewayV2::ApiMapping'
  Properties:
    DomainName: api.example.com
    ApiMappingKey: 'v1'
```

```
    ApiId: !Ref MyApi
    Stage: !Ref MyStage
```

WebSocket API에 대한 기본 엔드포인트 비활성화

기본적으로 클라이언트는 API에 대해 API Gateway가 생성하는 `execute-api` 엔드포인트를 사용하여 API를 호출할 수 있습니다. 클라이언트가 사용자 지정 도메인 이름을 사용해야만 API에 액세스할 수 있도록 하려면 기본 `execute-api` 엔드포인트를 비활성화합니다.

> **Note**
>
> 기본 엔드포인트를 비활성화하면 API의 모든 스테이지에 영향이 미칩니다.

다음 AWS CLI 명령은 WebSocket API의 기본 엔드포인트를 비활성화합니다.

```
aws apigatewayv2 update-api \
    --api-id abcdef123 \
    --disable-execute-api-endpoint
```

기본 엔드포인트를 비활성화한 후 변경 사항을 적용하려면 API를 배포해야 합니다.

다음 AWS CLI 명령은 배포를 생성합니다.

```
aws apigatewayv2 create-deployment \
    --api-id abcdef123 \
    --stage-name dev
```

WebSocket API 보호

API에 대한 제한을 구성하여 너무 많은 요청으로 인해 과부하되지 않도록 보호할 수 있습니다. 제한은 모두 최선의 방식으로 적용되며 보장된 요청 한도가 아닌 대상으로 간주해야 합니다.

API Gateway는 요청에 대해 토큰이 계산되는 토큰 버킷 알고리즘을 사용하여 API에 대한 요청을 제한합니다. 특히 API Gateway는 리전별로 계정의 모든 API에 대한 요청 제출 속도와 버스트를 검사합니다. 토큰 버킷 알고리즘에서 버스트는 이러한 제한의 사전 정의된 초과 실행을 허용할 수 있지만, 다른 요인으로도 제한을 초과할 수 있습니다.

요청 제출이 정상 상태의 요청 속도 및 버스트 제한을 초과할 경우 API Gateway에서 요청을 제한하기 시작합니다. 이 시점에서 클라이언트는 429 Too Many Requests 오류 응답을 받을 수 있습니다. 이러한 예외를 포착하면 클라이언트는 속도 제한 방식으로 실패한 요청을 다시 제출할 수 있습니다.

API 개발자는 개별 API 단계 또는 경로에 대한 목표 제한을 설정하여 계정의 모든 API에서 전반적인 성능을 향상시킬 수 있습니다.

리전별 계정 수준 조절

기본적으로 API Gateway는 리전별로 AWS 계정 내의 모든 API에서 안정적인 상태의 초당 요청(RPS)을 제한합니다. 또한 리전별로 AWS 계정 내 모든 API에 대해 버스트(즉, 최대 버킷 크기)를 제한합니다. API Gateway에서 버스트 제한은 API Gateway가 429 Too Many Requests 오류 응답을 반환하기 전에 수행할 동시 요청 제출의 최대 목표 수를 나타냅니다. 조절 할당량에 대한 자세한 내용은 할당량 및 중요 정보 단원을 참조하세요.

계정당 한도는 지정된 리전에 있는 계정의 모든 API에 적용됩니다. 계정 수준 속도 제한은 요청 시 늘릴 수 있습니다. 제한 시간이 더 짧고 페이로드가 더 작은 API를 사용하면 더 높은 제한이 가능합니다. 리전별로 계정 수준 조절 한도 증가를 요청하려면 AWS 지원 센터에 문의하시기 바랍니다. 자세한 내용은 할당량 및 중요 정보 섹션을 참조하세요. 이러한 제한은 AWS 제한 한도보다 높을 수 없습니다.

라우팅 수준 스로틀

라우팅 수준 스로틀을 설정해 API의 특정 단계 또는 개별 라우팅에 대해 계정 수준 요청 스로틀 한도를 재정의할 수 있습니다. 기본 경로 제한 한도는 계정 수준 속도 제한을 초과할 수 없습니다.

AWS CLI를 사용하여 라우팅 수준 스로틀을 구성할 수 있습니다. 다음 명령은 API의 지정된 단계 및 라우팅에 대한 사용자 지정 스로틀을 구성합니다.

```
aws apigatewayv2 update-stage \
    --api-id a1b2c3d4 \
    --stage-name dev \
    --route-settings '{"messages":
{"ThrottlingBurstLimit":100,"ThrottlingRateLimit":2000}}'
```

WebSocket API 모니터링

CloudWatch 지표 및 CloudWatch Logs를 사용하여 WebSocket API를 모니터링할 수 있습니다. 로그와 지표를 결합하여 오류를 기록하고 API의 성능을 모니터링할 수 있습니다.

> **Note**
>
> API Gateway는 다음과 같은 경우 로그 및 지표를 생성하지 않을 수 있습니다.
>
> - 413 요청 엔터티가 너무 큼 오류가 발생한 경우
> - 429개 초과로 요청이 너무 많음 오류가 발생한 경우
> - API 매핑이 없는 사용자 지정 도메인으로 보낸 요청에서 400개의 연속 오류가 발생한 경우
> - 내부 오류로 인해 500개의 연속 오류가 발생한 경우

주제

- CloudWatch 지표로 WebSocket API 실행 모니터링
- WebSocket API의 로깅 구성

CloudWatch 지표로 WebSocket API 실행 모니터링

Amazon CloudWatch 지표를 사용하여 WebSocket API를 모니터링할 수 있습니다. 구성은 REST API에 사용된 것과 유사합니다. 자세한 내용은 Amazon CloudWatch 지표를 사용한 REST API 실행 모니터링 단원을 참조하세요.

다음 지표는 WebSocket API에서 지원됩니다.

측정치	설명
ConnectCount	`$connect` 라우팅 통합으로 전송된 메시지 수입니다.
MessageCount	클라이언트가 WebSocket API와 주고 받은 메시지 수입니다.
IntegrationError	통합에서 4XX/5XX 응답을 반환하는 요청의 수입니다.
ClientError	통합이 호출되기 전에 API Gateway에서 반환한 4XX 응답이 포함된 요청의 수입니다.

측정치	설명
ExecutionError	통합을 호출할 때 발생한 오류.
IntegrationLatency	API Gateway에서 통합에 요청을 보내고 API Gateway에서 통합으로부터 응답을 받기까지 시간 차이. 콜백 및 모의 통합을 위해 숨겨집니다.

다음 표의 차원을 사용하여 API Gateway 지표를 필터링할 수 있습니다.

차원	설명
ApiId	지정한 API ID를 사용하여 API에 대한 API Gateway 지표를 필터링합니다.
ApiId, 스테이지	지정한 API ID와 스테이지 ID를 사용하여 API 스테이지에 대한 API Gateway 지표를 필터링합니다.
ApiId, 스테이지, 라우팅	지정한 API ID, 스테이지 ID 및 라우팅 ID를 사용하여 API 메서드에 대한 API Gateway 지표를 필터링합니다. 사용자가 세부 CloudWatch 지표를 명시적으로 활성화하지 않으면 API Gateway는 이러한 지표를 전송하지 않습니다. API Gateway V2 REST API의 UpdateStage 작업을 호출하여 `detailedMetricsEnabled` 속성을 `true`로 업데이트하면 됩니다. 또는 update-st

차원	설명
	age AWS CLI 명령을 호출하여 DetailedMetricsEnabled 속성을 true로 업데이트할 수 있습니다. 이러한 지표를 활성화할 경우 계정에 추가 비용이 발생합니다. 요금에 대한 자세한 내용은 Amazon CloudWatch 요금을 참조하십시오.

WebSocket API의 로깅 구성

로깅을 활성화하여 CloudWatch Logs에 로그를 기록할 수 있습니다. CloudWatch의 API 로깅에는 실행 로깅과 액세스 로깅이라는 두 가지 유형이 있습니다. 실행 로깅에서 API Gateway는 CloudWatch Logs를 관리합니다. 이 프로세스에는 로그 그룹 및 로그 스트림을 생성하는 작업과 호출자의 요청 및 응답을 로그 스트림에 보고하는 작업이 포함됩니다.

액세스 로깅에서 API 개발자가 원하는 것은 누가 API에 액세스했고 호출자가 API에 어떻게 액세스했는지 로깅하는 것입니다. 자체 로그 그룹을 생성하거나 API Gateway에서 관리할 수 있는 기존 로그 그룹을 선택할 수 있습니다. 액세스 세부 정보를 지정하려면 $context 변수(선택한 형식으로 표현)를 선택하고 로그 그룹을 대상으로 선택합니다.

CloudWatch 로깅을 설정하는 자세한 방법은 the section called “API Gateway 콘솔을 사용하여 CloudWatch API 로깅 설정” 단원을 참조하십시오.

로그 형식을 지정할 때 기록할 컨텍스트 변수를 선택할 수 있습니다. 다음 변수가 지원됩니다.

파라미터	설명
$context.apiId	식별자 API Gateway가 API에 할당합니다.
$context.authorize.error	권한 부여 오류 메시지입니다.
$context.authorize.latency	권한 부여 지연 시간(ms)입니다.
$context.authorize.status	권한 부여 시도에서 반환된 상태 코드입니다.

파라미터	설명
`$context.authorizer.error`	권한 부여자로부터 반환된 오류 메시지입니다.
`$context.authorizer.integrationLatency`	Lambda 권한 부여자 지연 시간(ms)입니다.
`$context.authorizer.integrationStatus`	Lambda 권한 부여자로부터 반환된 상태 코드입니다.
`$context.authorizer.latency`	권한 부여자 지연 시간(ms)입니다.
`$context.authorizer.requestId`	AWS 엔드포인트의 요청 ID입니다.
`$context.authorizer.status`	권한 부여자로부터 반환된 상태 코드입니다.
`$context.authorizer.principalId`	클라이언트가 전송한 토큰에 연결되고 API Gateway Lambda 권한 부여자 Lambda 함수에서 반환되는 보안 주체 사용자 ID입니다. (이전에는 Lambda를 사용자 지정 권한 부여자라고도 함).

파라미터	설명
$context.authorizer. *property*	API Gateway Lambda 권한 부여자 함수에서 반환된 context 맵의 지정된 키-값 페어의 문자열화된 값입니다. 예를 들어, 권한 부여자는 다음 context 맵을 반환합니다. `"context" : { "key": "value", "numKey": 1, "boolKey": true }` $context.authorizer.key 를 호출하면 "value" 문자열이 반환되고, $context.authorizer.numKey 를 호출하면 "1" 문자열이 반환되고, $context.authorizer.boolKey 를 호출하면 "true" 문자열이 반환됩니다.
$context.authenticate.error	인증 시도에서 반환된 오류 메시지입니다.
$context.authenticate.latency	인증 지연 시간(ms)입니다.
$context.authenticate.status	인증 시도에서 반환된 상태 코드입니다.
$context.connectedAt	Epoch 형식 연결 시간입니다.
$context.connectionId	클라이언트에 대한 콜백을 만드는 데 사용할 수 있는 고유한 연결 ID입니다.
$context.domainName	WebSocket API의 도메인 이름입니다. 이 이름은 하드 코딩된 값 대신에 클라이언트를 콜백하는 데 사용할 수 있습니다.

파라미터	설명
$context.error.message	API Gateway 오류 메시지가 포함된 문자열입니다.
$context.error.messageString	$context.error.message 의 따옴표 붙은 값, 즉 "$context.error.message" 입니다.
$context.error.responseType	오류 응답 유형.
$context.error.validationErrorString	자세한 유효성 검사 오류 메시지가 포함된 문자열입니다.
$context.eventType	이벤트 유형: CONNECT, MESSAGE 또는 DISCONNECT .
$context.extendedRequestId	$context.requestId 와 동일합니다.
$context.identity.accountId	요청과 연결된 AWS 계정 ID입니다.
$context.identity.apiKey	키가 활성화된 API 요청에 연결된 API 소유자 키입니다.
$context.identity.apiKeyId	키가 활성화된 API 요청에 연결된 API 키 ID입니다.
$context.identity.caller	요청에 서명한 호출자의 보안 주체 ID입니다. IAM 권한 부여를 사용하는 라우팅에 대해 지원됩니다.

파라미터	설명
`$context.identity.cognitoAuthenticationProvider`	요청을 작성한 호출자가 사용한 Amazon Cognito 인증 공급자의 쉼표로 구분된 목록입니다. Amazon Cognito 자격 증명으로 요청을 서명한 경우에만 사용할 수 있습니다. 예를 들어, Amazon Cognito 사용자 풀의 자격 증명의 경우 `cognito-idp.region.amazonaws.com/user_pool_id ,cognito-idp.region.amazonaws.com/user_pool_id :CognitoSignIn: token subject claim` 자세한 내용은 Amazon Cognito 개발자 안내서의 연동 자격 증명 사용을 참조하세요.
`$context.identity.cognitoAuthenticationType`	요청을 작성한 호출자의 Amazon Cognito 인증 유형입니다. Amazon Cognito 자격 증명으로 요청을 서명한 경우에만 사용할 수 있습니다. 가능한 값에는 인증 자격 증명에 대한 `authenticated` 및 미인증 자격 증명에 대한 `unauthenticated` 이(가) 포함됩니다.
`$context.identity.cognitoIdentityId`	요청을 작성한 호출자의 Amazon Cognito 자격 증명 ID입니다. Amazon Cognito 자격 증명으로 요청을 서명한 경우에만 사용할 수 있습니다.
`$context.identity.cognitoIdentityPoolId`	요청을 작성한 호출자의 Amazon Cognito 자격 증명 풀 ID입니다. Amazon Cognito 자격 증명으로 요청을 서명한 경우에만 사용할 수 있습니다.
`$context.identity.principalOrgId`	AWS 조직 ID입니다. IAM 권한 부여를 사용하는 라우팅에 대해 지원됩니다.
`$context.identity.sourceIp`	API 게이트웨이에 대한 요청을 작성한 TCP 연결의 소스 IP 주소입니다.

파라미터	설명
$context.identity.user	리소스 액세스에 대해 권한을 부여할 사용자의 보안 주체 ID입니다. IAM 권한 부여를 사용하는 라우팅에 대해 지원됩니다.
$context.identity.userAgent	API 호출자의 사용자 에이전트입니다.
$context.identity.userArn	인증 후 식별된 실제 사용자의 ARN(Amazon Resource Name)입니다.
$context.integration.error	통합에서 반환된 오류 메시지입니다.
$context.integration.integrationStatus	Lambda 프록시 통합의 경우 백엔드 Lambda 함수 코드가 아니라 AWS Lambda에서 반환된 상태 코드입니다.
$context.integration.latency	통합 지연 시간(ms)입니다. $context.integrationLatency 와 동일합니다.
$context.integration.requestId	AWS 엔드포인트의 요청 ID입니다. $context.awsEndpointRequestId 와 동일합니다.
$context.integration.status	통합에서 반환된 상태 코드입니다. Lambda 프록시 통합의 경우 Lambda 함수 코드에서 반환된 상태 코드입니다. $context.integrationStatus 와 동일합니다.
$context.integrationLatency	통합 지연 시간(밀리초)으로서 액세스 로깅에만 사용할 수 있습니다.
$context.messageId	메시지의 고유 서버 측 ID. $context.eventType 이 MESSAGE인 경우에만 사용 가능합니다.
$context.requestId	$context.extendedRequestId 와 동일합니다.

파라미터	설명
$context.requestTime	CLF 형식 요청 시간(dd/MMM/yyyy:HH:mm:ss +-hhmm)입니다.
$context.requestTimeEpoch	Epoch 형식 요청 시간(밀리초)입니다.
$context.routeKey	선택한 라우팅 키.
$context.stage	API 호출의 개발 단계입니다(예: 베타 또는 프로덕션).
$context.status	응답 상태입니다.
$context.waf.error	에서 반환된 오류 메시지입니다AWS WAF
$context.waf.latency	AWS WAF 지연 시간(ms)입니다.
$context.waf.status	AWS WAF에서 반환된 상태 코드입니다.

일반적으로 사용되는 몇 가지 액세스 로그 형식의 예가 API Gateway 콘솔에 표시되며 다음과 같이 나열됩니다.

- CLF(Common Log Format):

```
$context.identity.sourceIp $context.identity.caller \
$context.identity.user [$context.requestTime] "$context.eventType $context.routeKey
 $context.connectionId" \
$context.status $context.requestId
```

  연속 문자(\)는 시각적 보조 수단으로 사용됩니다. 로그 형식은 한 줄이어야 합니다. 로그 형식의 끝에 줄바꿈 문자(\n)를 추가하여 각 로그 항목의 끝에 줄바꿈 문자를 포함할 수 있습니다.

- 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"
}
```

연속 문자(\)는 시각적 보조 수단으로 사용됩니다. 로그 형식은 한 줄이어야 합니다. 로그 형식의 끝에 줄바꿈 문자(\n)를 추가하여 각 로그 항목의 끝에 줄바꿈 문자를 포함할 수 있습니다.

- 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>
```

연속 문자(\)는 시각적 보조 수단으로 사용됩니다. 로그 형식은 한 줄이어야 합니다. 로그 형식의 끝에 줄바꿈 문자(\n)를 추가하여 각 로그 항목의 끝에 줄바꿈 문자를 포함할 수 있습니다.

- CSV(쉼표로 분리된 값):

```
$context.identity.sourceIp,$context.identity.caller, \
$context.identity.user,$context.requestTime,$context.eventType, \
$context.routeKey,$context.connectionId,$context.status, \
$context.requestId
```

연속 문자(\)는 시각적 보조 수단으로 사용됩니다. 로그 형식은 한 줄이어야 합니다. 로그 형식의 끝에 줄바꿈 문자(\n)를 추가하여 각 로그 항목의 끝에 줄바꿈 문자를 포함할 수 있습니다.

API Gateway Amazon 리소스 이름(ARN) 참조

다음 표에는 API Gateway 리소스의 Amazon 리소스 이름(ARN)이 나와 있습니다. AWS Identity and Access Management 정책에서의 ARN 사용에 대한 자세한 내용은 Amazon API Gateway에서 IAM을 사용하는 방식 및 IAM 권한을 사용하여 API에 대한 액세스 제어 단원을 참조하십시오.

HTTP API 및 WebSocket API 리소스

Resource	ARN
AccessLogSettings	`arn:partition :apigateway: region::/apis/api-id/stages/stage-name /accesslogsettings`
Api	`arn:partition :apigateway: region::/apis/api-id`
Apis	`arn:partition :apigateway: region::/apis`
ApiMapping	`arn:partition :apigateway: region::/domainnames/ domain-name /apimappings/ id`
ApiMappings	`arn:partition :apigateway: region::/domainnames/ domain-name /apimappings`
권한 부여자	`arn:partition :apigateway: region::/apis/api-id/authorizers/ id`
권한 부여자	`arn:partition :apigateway: region::/apis/api-id/authorizers`

Resource	ARN
Cors	arn:*partition* :apigatew ay: *region*::/apis/*api-id*/cors
배포	arn:*partition* :apigatew ay: *region*::/apis/*api-id*/deployme nts/ *id*
배포	arn:*partition* :apigatew ay: *region*::/apis/*api-id*/deployme nts
DomainName	arn:*partition* :apigatew ay: *region*::/domainnames/ *domain-na me*
DomainNames	arn:*partition* :apigatew ay: *region*::/domainnames
ExportedAPI	arn:*partition* :apigatew ay: *region*::/apis/*api-id*/exports/ *specification*
통합	arn:*partition* :apigatew ay: *region*::/apis/*api-id*/integrat ions/ *integration-id*
통합	arn:*partition* :apigatew ay: *region*::/apis/*api-id*/integrat ions
IntegrationResponse	arn:*partition* :apigatew ay: *region*::/apis/*api-id*/integrat ionresponses/ *integration-respon se*

Resource	ARN
IntegrationResponses	arn:*partition* :apigatew ay: *region*::/apis/*api-id*/integrat ionresponses
모델	arn:*partition* :apigatew ay: *region*::/apis/*api-id*/models/*id*
모델	arn:*partition* :apigatew ay: *region*::/apis/*api-id*/models
ModelTemplate	arn:*partition* :apigatew ay: *region*::/apis/*api-id*/models/*id*/ template
경로	arn:*partition* :apigatew ay: *region*::/apis/*api-id*/routes/*id*
경로	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*

Resource	ARN
단계	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

REST API 리소스

Resource	ARN
계정	arn:*partition* :apigatew ay: *region*::/account
ApiKey	arn:*partition* :apigatew ay: *region*::/apikeys/ *id*
ApiKeys	arn:*partition* :apigatew ay: *region*::/apikeys
권한 부여자	arn:*partition* :apigatew ay: *region*::/restapis/ *api-id*/ authorizers/ *id*
권한 부여자	arn:*partition* :apigatew ay: *region*::/restapis/ *api-id*/ authorizers

Resource	ARN
BasePathMapping	arn:*partition* :apigatew ay: *region*::/domainnames/ *domain-na me* /basepathmappings/ *basepath*
BasePathMappings	arn:*partition* :apigatew ay: *region*::/domainnames/ *domain-na me* /basepathmappings
ClientCertificate	arn:*partition* :apigatew ay: *region*::/clientcertifica tes/ *id*
ClientCertificates	arn:*partition* :apigatew ay: *region*::/clientcertificates
배포	arn:*partition* :apigatew ay: *region*::/restapis/ *api-id*/ deployments/ *id*
배포	arn:*partition* :apigatew ay: *region*::/restapis/ *api-id*/ deployments
DocumentationPart	arn:*partition* :apigatew ay: *region*::/restapis/ *api-id*/ documentation/parts/ *id*
DocumentationParts	arn:*partition* :apigatew ay: *region*::/restapis/ *api-id*/ documentation/parts
DocumentationVersion	arn:*partition* :apigatew ay: *region*::/restapis/ *api-id*/ documentation/versions/ *version*

Resource	ARN
DocumentationVersions	arn:*partition* :apigatew ay: *region*::/restapis/ *api-id*/ documentation/versions
DomainName	arn:*partition* :apigatew ay: *region*::/domainnames/ *domain-na me*
DomainNames	arn:*partition* :apigatew ay: *region*::/domainnames
GatewayResponse	arn:*partition* :apigatew ay: *region*::/restapis/ *api-id*/ gatewayresponses/ *response-type*
GatewayResponses	arn:*partition* :apigatew ay: *region*::/restapis/ *api-id*/ gatewayresponses
통합	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*
메서드	arn:*partition* :apigatew ay: *region*::/restapis/ *api-id*/ resources/ *resource-id* /methods/ *http-method*

Resource	ARN
MethodResponse	arn:*partition* :apigateway: *region*::/restapis/ *api-id*/resources/ *resource-id* /methods/ *http-method* /responses/ *status-code*
모델	arn:*partition* :apigateway: *region*::/restapis/ *api-id*/models/*model-name*
모델	arn:*partition* :apigateway: *region*::/restapis/ *api-id*/models
RequestValidator	arn:*partition* :apigateway: *region*::/restapis/ *api-id*/requestvalidators/ *id*
RequestValidators	arn:*partition* :apigateway: *region*::/restapis/ *api-id*/requestvalidators
Resource	arn:*partition* :apigateway: *region*::/restapis/ *api-id*/resources/ *id*
리소스	arn:*partition* :apigateway: *region*::/restapis/ *api-id*/resources
RestApi	arn:*partition* :apigateway: *region*::/restapis/ *api-id*
RestApis	arn:*partition* :apigateway: *region*::/restapis

Resource	ARN
단계	arn:*partition* :apigatew ay: *region*::/restapis/ *api-id*/ stages/*stage-name*
Stages	arn:*partition* :apigatew ay: *region*::/restapis/ *api-id*/ stages
Tags	arn:*partition* :apigatew ay: *region*::/tags/*url-encoded-resource-arn*
Template	arn:*partition* :apigatew ay: *region*::/restapis/models / *model-name* /template
UsagePlan	arn:*partition* :apigatew ay: *region*::/usageplans/ *usageplan-id*
UsagePlans	arn:*partition* :apigatew ay: *region*::/usageplans
UsagePlanKey	arn:*partition* :apigatew ay: *region*::/usageplans/ *usageplan-id* /keys/*id*
UsagePlanKeys	arn:*partition* :apigatew ay: *region*::/usageplans/ *usageplan-id* /keys
VpcLink	arn:*partition* :apigatew ay: *region*::/vpclinks/ *vpclink-id*
VpcLinks	arn:*partition* :apigatew ay: *region*::/vpclinks

execute-api(HTTP API, WebSocket API 및 REST API)

Resource	ARN
WebSocket API 엔드포인트	`arn:`*`partition`*` :execute-api: `*`region`*`:`*`account-id`*` :`*`api-id`*`/`*`stage`*`/`*`route-key`*
HTTP API 및 REST API 엔드포인트*	`arn:`*`partition`*` :execute-api: `*`region`*`:`*`account-id`*` :`*`api-id`*`/`*`stage`*`/`*`http-method`*` /`*`resource-path`*
Lambda 권한 부여자**	`arn:`*`partition`*` :execute-api: `*`region`*`:`*`account-id`*` :`*`api-id`*`/authorizers/ `*`authorizer-id`*

* HTTP API용 `$default` 라우팅 엔드포인트의 ARN은 `arn:`*`partition`*`:execute-api:`*`region`*`:`*`account-id`*`:`*`api-id`*`/*/$default`입니다.

** 이 ARN은 Lambda 권한 부여자 함수에 대한 리소스 정책에서 `SourceArn` 조건을 설정할 때만 적용됩니다. 관련 예제는 the section called “Lambda 권한 부여자 만들기” 섹션을 참조하세요

OpenAPI에 대한 API Gateway 확장 작업

API Gateway 확장 프로그램은 REST API 및 HTTP API에 대한 AWS특정 권한 부여 및 API 게이트웨이별 API 통합을 지원합니다. 이 단원에서는 OpenAPI 사양에 대한 API Gateway 확장을 설명합니다.

> **Tip**
>
> 애플리케이션에서 API Gateway 확장을 사용하는 방법을 이해할 수 있도록 API Gateway 콘솔을 사용하여 REST API 또는 HTTP API를 생성해 OpenAPI 정의 파일로 내보낼 수 있습니다. API를 내보내는 방법에 대한 자세한 내용은 API Gateway에서 REST API 내보내기 및 API Gateway에서 HTTP API 내보내기 단원을 참조하세요.

주제

- x-amazon-apigateway-any-메서드 객체
- x-amazon-apigateway-cors 객체
- x-amazon-apigateway-api-키-소스 속성
- x-amazon-apigateway-auth 오브젝트
- x-amazon-apigateway-authorizer 객체
- x-amazon-apigateway-authtype 프로퍼티
- x-amazon-apigateway-binary-미디어 유형 속성
- x-amazon-apigateway-documentation 객체
- x-amazon-apigateway-endpoint-구성 객체
- x-amazon-apigateway-gateway-응답 객체
- x-amazon-apigateway-gateway- responses.GatewayResponse 객체
- x-amazon-apigateway-gateway-responses.responseParameters 객체
- x-amazon-apigateway-gateway-responses.ResponseTemplates 객체
- x-amazon-apigateway-importexport-버전
- x-amazon-apigateway-integration 객체
- x-amazon-apigateway-integrations 객체
- x-amazon-apigateway-integration. 요청 템플릿 객체
- x-amazon-apigateway-integration. 요청 매개변수 객체
- x-amazon-apigateway-integration.response 객체

- x-amazon-apigateway-integration.response 객체
- x-amazon-apigateway-integration. 응답 템플릿 객체
- x-amazon-apigateway-integration. 응답 매개변수 객체
- x-amazon-apigateway-integration.tlsConfig 객체
- x-amazon-apigateway-minimum- 압축 크기
- x-amazon-apigateway-policy
- x-amazon-apigateway-request-유효성 검사기 속성
- x-amazon-apigateway-request-유효성 검사기 개체
- x-amazon-apigateway-request- 유효성 검사기. 요청 유효성 검사기 개체
- x-amazon-apigateway-tag- 값 속성

x-amazon-apigateway-any-메서드 객체

OpenAPI 경로 항목 객체(OpenAPI Path Item Object)에서 API Gateway catch-all ANY 메서드에 대한 OpenAPI 작업 객체(OpenAPI Operation Object)를 지정합니다. 이 객체는 다른 작업 객체들과 함께 존재할 수 있고, 명시적으로 선언되지 않은 모든 HTTP 메서드를 포착합니다.

다음 표에서는 API Gateway에서 확장한 속성을 나열합니다. 다른 OpenAPI 작업 속성은 OpenAPI 사양을 참조하세요.

속성

속성 이름	유형	설명
`isDefaultRoute`	`Boolean`	경로가 `$default` 경로인지 여부를 지정합니다. HTTP API에서만 지원됩니다. 자세한 내용은 HTTP API에 대한 라우팅 작업 단원을 참조하세요.
`x-amazon-apigateway-integration`	x-amazon-apigateway-integration 객체	백엔드와 메서드의 통합을 지정합니다. OpenAPI 작업 객체의 확장된 속성입니다. 통합은 `AWS`, `AWS_PROXY` , `HTTP`, `HTTP_PROXY` 또는 `MOCK` 유형일 수 있습니다.

x-amazon-apigateway-any-메서드 예제

다음 예는 프록시 리소스 ANY의 {proxy+} 메서드를 Lambda 함수 TestSimpleProxy와 통합합니다.

```
    "/{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"
        }
```

다음 예제에서는 Lambda 함수인 $default와 통합되는 HTTP API에 대한 HelloWorld 경로를 생성합니다.

```
"/$default": {
    "x-amazon-apigateway-any-method": {
      "isDefaultRoute": true,
      "x-amazon-apigateway-integration": {
        "type": "AWS_PROXY",
        "httpMethod": "POST",
        "uri": "arn:aws:apigateway:us-east-1:lambda:path/2015-03-31/functions/
arn:aws:lambda:us-east-1:123456789012:function:HelloWorld/invocations",
        "timeoutInMillis": 1000,
        "connectionType": "INTERNET",
        "payloadFormatVersion": 1.0
      }
   }
```

```
}
```

x-amazon-apigateway-cors 객체

HTTP API에 대한 CORS(Cross-Origin Resource Sharing) 구성을 지정합니다. 확장은 루트 수준 OpenAPI 구조에 적용됩니다. 자세한 내용은 HTTP API에 대한 CORS 구성 단원을 참조하세요.

속성

속성 이름	유형	설명
allowOrigins	Array	허용된 오리진을 지정합니다.
allowCredentials	Boolean	자격 증명이 CORS 요청에 포함되는지 여부를 지정합니다.
exposeHeaders	Array	노출되는 헤더를 지정합니다.
maxAge	Integer	브라우저가 preflight 요청 결과를 캐시해야 하는 시간(초)을 지정합니다.
allowMethods	Array	허용되는 HTTP 메서드를 지정합니다.
allowHeaders	Array	허용되는 헤더를 지정합니다.

x-amazon-apigateway-cors 예

다음은 HTTP API에 대한 CORS 구성 예제입니다.

```
"x-amazon-apigateway-cors": {
    "allowOrigins": [
      "https://www.example.com"
    ],
    "allowCredentials": true,
    "exposeHeaders": [
      "x-apigateway-header",
      "x-amz-date",
      "content-type"
```

```
    ],
    "maxAge": 3600,
    "allowMethods": [
      "GET",
      "OPTIONS",
      "POST"
    ],
    "allowHeaders": [
      "x-apigateway-header",
      "x-amz-date",
      "content-type"
    ]
}
```

x-amazon-apigateway-api-키-소스 속성

API 키를 수신할 소스를 지정하여 키가 필요한 API 메서드를 조절합니다. 이 API 수준 속성은 `String` 유형입니다.

요청을 위한 API 키의 소스를 지정합니다. 유효한 값은 다음과 같습니다.

- 요청의 `HEADER` 헤더에서 API 키를 수신하기 위한 `X-API-Key`.
- Lambda 권한 부여자(이전에는 사용자 지정 권한 부여자라고 함)의 `AUTHORIZER`로부터 API 키를 수신하기 위한 `UsageIdentifierKey`.

x-amazon-apigateway-api-키-소스 예제

다음 예는 `X-API-Key` 헤더를 API 키 원본으로 설정합니다.

OpenAPI 2.0

```
{
  "swagger" : "2.0",
  "info" : {
    "title" : "Test1"
   },
  "schemes" : [ "https" ],
  "basePath" : "/import",
  "x-amazon-apigateway-api-key-source" : "HEADER",
```

```
    .
    .
    .
}
```

OpenAPI 3.0.1

```
{
  "openapi" : "3.0.1",
  "info" : {
    "title" : "Test1"
  },
  "servers" : [ {
    "url" : "/{basePath}",
    "variables" : {
      "basePath" : {
        "default" : "import"
      }
    }
  } ],
  "x-amazon-apigateway-api-key-source" : "HEADER",
    .
    .
    .
}
```

x-amazon-apigateway-auth 오브젝트

API Gateway에서 메서드 호출의 권한 부여에 적용할 권한 부여 유형을 정의합니다.

속성

속성 이름	유형	설명
type	string	권한 부여 유형을 지정합니다. 열린 액세스를 위해 "NONE"을 지정합니다. IAM 권한을 사용하도록 "AWS_IAM" 을 지정합니다. 값은 대/소문자를 구분합니다.

x-amazon-apigateway-auth 예

다음 예제는 API 메서드에 대한 권한 부여 유형을 설정합니다.

OpenAPI 3.0.1

```
{
  "openapi": "3.0.1",
  "info": {
    "title": "openapi3",
    "version": "1.0"
  },
  "paths": {
    "/protected-by-iam": {
      "get": {
        "x-amazon-apigateway-auth": {
          "type": "AWS_IAM"
        }
      }
    }
  }
}
```

x-amazon-apigateway-authorizer 객체

API Gateway에서 메서드 호출의 권한 부여를 위해 적용할 Lambda 권한 부여자, Amazon Cognito 사용자 풀 또는 JWT 권한 부여자를 정의합니다. 이 확장은 OpenAPI 2 및 OpenAPI 3의 보안 정의에 적용됩니다.

속성

속성 이름	유형	설명
`type`	`string`	권한 부여자의 유형입니다. 필수 속성입니다. REST API의 경우 호출자 자격 증명이 권한 부여 토큰에 내장된 권한 부여자에 대해 `token`을(를) 지정합니다. 호

속성 이름	유형	설명
		출자 자격 증명이 요청 파라미터에 포함된 권한 부여자의 경우 `request`를 지정합니다. Amazon Cognito 사용자 풀을 사용하여 API에 대한 액세스를 제어하는 권한 부여자의 경우 `cognito_user_pools`를 지정합니다. HTTP API에 대해 호출자 자격 증명이 요청 파라미터에 포함된 Lambda 권한 부여자의 경우 `request`를 지정합니다. JWT 권한 부여자에 대해 `jwt`를 지정합니다.
`authorizerUri`	`string`	권한 부여자 Lambda 함수의 URI(Uniform Resource Identifier)입니다. 구문은 다음과 같습니다. `"arn:aws:apigateway:us-east-1:lambda:path/2015-03-31/functions/arn:aws:lambda:us-east-1:`*`account-id`*`:function:`*`auth_function_name`*`/invocations"`
`authorizerCredentials`	`string`	IAM 실행 역할의 ARN 형식으로 된, 권한 부여자를 호출하는 데 필요한 자격 증명입니다. 예: "arn:aws:iam::*account-id* :*IAM_role*".

속성 이름	유형	설명
authorizerPayloadFormatVersion	string	HTTP API의 경우, API Gateway에서 Lambda 권한 부여자로 전송하는 데이터의 형식과 API Gateway에서 Lambda의 응답을 해석하는 방법을 지정합니다. 자세한 내용은 the section called “페이로드 형식 버전” 단원을 참조하세요.
enableSimpleResponses	Boolean	HTTP API에 대해 request 권한 부여자가 부울 값을 반환할지 아니면 IAM 정책을 반환할지 여부를 지정합니다. authorizerPayloadFormatVersion 이 2.0인 권한 부여자의 경우에만 지원됩니다. 이 속성을 사용하면 Lambda 권한 부여자 함수가 부울 값을 반환합니다. 자세한 내용은 the section called “형식 2.0에 대한 Lambda 함수 응답” 단원을 참조하세요.
identitySource	string	ID 소스로서 요청 파라미터의 매핑 표현식에 대한 쉼표로 구분된 목록입니다. request 및 jwt 유형의 권한 부여자에만 적용됩니다.
jwtConfiguration	Object	JWT 권한 부여자의 발급자 및 대상 그룹을 지정합니다. 자세히 알아보려면 API Gateway 버전 2 API 참조의 JWTConfiguration을 참조하세요. HTTP API에서만 지원됩니다.

속성 이름	유형	설명
identityValidationExpression	string	수신되는 자격 증명으로서의 토큰을 확인하는 표현식입니다. 예: "^x-[a-z]+". REST API에서만 지원됩니다.
authorizerResultTtlInSeconds	string	권한 부여자 결과가 캐싱되는 시간(초)입니다.
providerARNs	string의 어레이	COGNITO_USER_POOLS 에 대한 Amazon Cognito 사용자 풀의 목록입니다.

x-amazon-apigateway-authorizer REST API의 예시

다음 OpenAPI 보안 정의 예제에서는 test-authorizer라는 "토큰" 유형의 Lambda 권한 부여자를 지정합니다.

```
 "securityDefinitions" : {
   "test-authorizer" : {
     "type" : "apiKey",                          // Required and the value must be
"apiKey" for an API Gateway API.
     "name" : "Authorization",                   // The name of the header containing
the authorization token.
     "in" : "header",                            // Required and the value must be
"header" for an API Gateway API.
     "x-amazon-apigateway-authtype" : "oauth2", // Specifies the authorization
mechanism for the client.
     "x-amazon-apigateway-authorizer" : {       // An API Gateway Lambda authorizer
definition
       "type" : "token",                         // Required property and the value
must "token"
       "authorizerUri" : "arn:aws:apigateway:us-east-1:lambda:path/2015-03-31/
functions/arn:aws:lambda:us-east-1:account-id:function:function-name/invocations",
       "authorizerCredentials" : "arn:aws:iam::account-id:role",
       "identityValidationExpression" : "^x-[a-z]+",
       "authorizerResultTtlInSeconds" : 60
     }
```

```
    }
  }
```

다음 OpenAPI 작업 객체 코드 조각은 위에 나온 Lambda 권한 부여자를 사용하도록 `GET /http`를 설정합니다.

```
  "/http" : {
     "get" : {
       "responses" : { },
       "security" : [ {
         "test-authorizer" : [ ]
       } ],
       "x-amazon-apigateway-integration" : {
         "type" : "http",
         "responses" : {
           "default" : {
             "statusCode" : "200"
           }
         },
         "httpMethod" : "GET",
         "uri" : "http://api.example.com"
       }
     }
   }
```

다음 OpenAPI 보안 정의 예제에서는 단일 헤더 파라미터(auth)를 자격 증명 원본으로 사용하여 "요청" 유형의 Lambda 권한 부여자를 지정합니다. `securityDefinitions`의 이름은 `request_authorizer_single_header`입니다.

```
"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
      }
    }
}
```

다음 OpenAPI 보안 정의 예제에서는 자격 증명 원본으로 헤더(HeaderAuth1) 및 쿼리 문자열 파라미터 QueryString1을 사용하여 "요청" 유형이 Lambda 권한 부여자를 지정합니다.

```
"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
      }
    }
}
```

다음 OpenAPI 보안 정의 예제에서는 단일 단계 변수(stage)를 자격 증명 원본으로 사용하여 "요청" 유형의 API Gateway Lambda 권한 부여자를 지정합니다.

```
"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
      }
    }
}
```

다음 OpenAPI 보안 정의 예제에서는 Amazon Cognito 사용자 풀을 권한 부여자로 지정합니다.

```
 "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"
        ]
      }
    }
```

다음 OpenAPI 작업 객체 코드 조각에서는 사용자 지정 범위 없이 이전 Amazon Cognito 사용자 풀을 권한 부여자로 사용하도록 `GET /http`를 설정합니다.

```
  "/http" : {
    "get" : {
      "responses" : { },
      "security" : [ {
```

```
          "cognito-pool" : [ ]
        } ],
        "x-amazon-apigateway-integration" : {
          "type" : "http",
          "responses" : {
            "default" : {
              "statusCode" : "200"
            }
          },
          "httpMethod" : "GET",
          "uri" : "http://api.example.com"
        }
      }
    }
```

x-amazon-apigateway-authorizer HTTP API에 대한 예제

다음 OpenAPI 3.0 예제에서는 `Authorization` 헤더를 자격 증명 소스로 사용하여 Amazon Cognito를 자격 증명 공급자로 사용하는 HTTP API에 대한 JWT 권한 부여자를 생성합니다.

```
"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"
    }
  }
}
```

다음 OpenAPI 3.0 예제에서는 이전 예제와 동일한 JWT 권한 부여자를 생성합니다. 하지만 이 예제에서는 OpenAPI `openIdConnectUrl` 속성을 사용하여 발행자를 자동으로 검색합니다. `openIdConnectUrl`는 완전한 형태여야 합니다.

```
"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"
    }
  }
}
```

다음은 HTTP API에 대한 Lambda 권한 부여자를 생성하는 예제입니다. 이 예제 권한 부여자는 `Authorization` 헤더를 자격 증명 원본으로 사용합니다. 권한 부여자는 `2.0` 페이로드 포맷 버전을 사용하고 `enableSimpleResponses`가 `true`로 설정되어 있기 때문에 부울 값을 반환합니다.

```
"securitySchemes" : {
  "lambda-authorizer" : {
    "type" : "apiKey",
    "name" : "Authorization",
    "in" : "header",
    "x-amazon-apigateway-authorizer" : {
      "type" : "request",
      "identitySource" : "$request.header.Authorization",
      "authorizerUri" : "arn:aws:apigateway:us-west-2:lambda:path/2015-03-31/functions/
arn:aws:lambda:us-west-2:123456789012:function:function-name/invocations",
      "authorizerPayloadFormatVersion" : "2.0",
      "authorizerResultTtlInSeconds" : 300,
      "enableSimpleResponses" : true
    }
  }
}
```

x-amazon-apigateway-authtype 프로퍼티

REST API의 경우 이 확장을 사용하여 Lambda 권한 부여자의 사용자 지정 유형을 정의할 수 있습니다. 이 경우 값은 자유 형식입니다. 예를 들어 API에는 서로 다른 내부 스키마를 사용하는 여러 Lambda

권한 부여자가 있을 수 있습니다. 이 확장을 사용하여 Lambda 권한 부여자의 내부 스키마를 식별할 수 있습니다.

보다 일반적으로 HTTP API 및 REST API에서는 동일한 보안 스키마를 공유하는 여러 작업에서 IAM 권한 부여를 정의하는 방법으로 사용할 수도 있습니다. 이 경우 용어 `awsSigv4`은(는) 접두사 `aws`이(가) 붙은 용어와 함께 예약 용어입니다.

이 확장은 OpenAPI 2 및 OpenAPI 3의 `apiKey` 유형 보안 스키마에 적용됩니다.

x-amazon-apigateway-authtype 예시

다음 OpenAPI 3 예제는 REST API 또는 HTTP API의 여러 리소스에 대한 IAM 권한 부여를 정의합니다.

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

다음 OpenAPI 3 예제는 REST API에 대한 사용자 지정 스키마를 사용하여 Lambda 권한 부여자를 정의합니다.

```
{
  "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"
}
```

다음 사항도 참조하세요.

authorizer.authType

x-amazon-apigateway-binary-미디어 유형 속성

API Gateway에서 지원하는 이진 미디어 유형의 목록을 지정합니다(예: `application/octet-stream` 및 `image/jpeg`). 이 확장은 JSON 배열입니다. 이는 OpenAPI 문서의 최상위 공급업체 확장으로 포함되어야 합니다.

x-amazon-apigateway-binary-미디어 유형 예제

다음 예에서는 API의 인코딩 조회 순서를 보여줍니다.

```
"x-amazon-apigateway-binary-media-types": [ "application/octet", "image/jpeg" ]
```

x-amazon-apigateway-documentation 객체

API Gateway로 가져올 설명서 부분을 정의합니다. 이 객체는 `DocumentationPart` 인스턴스의 어레이를 포함하는 JSON 객체입니다.

속성

속성 이름	유형	설명
documentationParts	Array	내보내거나 가져온 DocumentationPart 인스턴스의 어레이입니다.
version	String	내보낸 설명서 부분의 스냅샷의 버전 ID입니다.

x-amazon-apigateway-documentation 예

OpenAPI에 대한 API Gateway 확장의 다음 예에서는 API Gateway의 API에서 내보내거나 가져올 DocumentationParts 인스턴스를 정의합니다.

```
{ ...
  "x-amazon-apigateway-documentation": {
    "version": "1.0.3",
    "documentationParts": [
       {
         "location": {
          "type": "API"
       },
        "properties": {
          "description": "API description",
          "info": {
            "description": "API info description 4",
            "version": "API info version 3"
          }
        }
      },
      {
         … // Another DocumentationPart instance
      }
    ]
  }
}
```

x-amazon-apigateway-endpoint-구성 객체

API에 대한 엔드포인트 구성의 세부 사항을 지정합니다. 이 확장은 OpenAPI 작업 객체의 확장된 속성입니다. 이 객체는 Swagger 2.0의 최상위 공급 업체 확장에 있어야 합니다. OpenAPI 3.0의 경우 서버 객체의 공급 업체 확장 아래에 있어야 합니다.

속성

속성 이름	유형	설명
`disableExecuteApiEndpoint`	불	클라이언트가 기본 `execute-api` 엔드포인트를 사용하여 API를 호출할 수 있는지 여부를 지정합니다. 기본적으로 클라이언트는 기본 `https://{api_id}.execute-api.{region}.amazonaws.com` 엔드포인트로 API를 호출할 수 있습니다. 클라이언트가 사용자 지정 도메인 이름을 사용하여 API를 호출하도록 요구하려면 `true`를 지정합니다.
`vpcEndpointIds`	`String`의 어레이	REST API의 Route 53 별칭 레코드를 생성하는 데 사용할 VpcEndpoint 식별자 목록입니다. `PRIVATE` 엔드포인트 유형인 REST API에만 지원됩니다.

x-amazon-apigateway-endpoint-구성 예제

다음 예제는 지정된 VPC 종단점을 REST API에 연결합니다.

```
"x-amazon-apigateway-endpoint-configuration": {
    "vpcEndpointIds": ["vpce-0212a4ababd5b8c3e", "vpce-01d622316a7df47f9"]
}
```

다음 예제는 API에 대한 기본 엔드포인트를 비활성화합니다.

```
"x-amazon-apigateway-endpoint-configuration": {
    "disableExecuteApiEndpoint": true
}
```

x-amazon-apigateway-gateway-응답 객체

API에 대한 게이트웨이 응답을 키-값 쌍의 문자열 대 GatewayResponse맵으로 정의합니다. 확장은 루트 수준 OpenAPI 구조에 적용됩니다.

속성

속성 이름	유형	설명
responseType	x-amazon-apigateway-gateway- 응답. 게이트웨이 응답	지정된 *responseType* 에 대한 GatewayResponse 입니다.

x-amazon-apigateway-gateway- 응답 예제

다음 OpenAPI에 대한 API Gateway 확장 예제는 유형용 GatewayResponse인스턴스와 DEFAULT_4XX 유형용 인스턴스 두 개가 포함된 GatewayResponses맵을 정의합니다. INVALID_API_KEY

```
{
  "x-amazon-apigateway-gateway-responses": {
    "DEFAULT_4XX": {
      "responseParameters": {
        "gatewayresponse.header.Access-Control-Allow-Origin": "'domain.com'"
      },
      "responseTemplates": {
        "application/json": "{\"message\": test 4xx b }"
      }
    },
    "INVALID_API_KEY": {
      "statusCode": "429",
      "responseTemplates": {
        "application/json": "{\"message\": test forbidden }"
```

```
      }
    }
  }
}
```

x-amazon-apigateway-gateway- responses.GatewayResponse 객체

상태 코드, 적용 가능한 응답 파라미터 또는 응답 템플릿을 포함하여 지정된 응답 유형의 게이트웨이 응답을 정의합니다.

속성

속성 이름	유형	설명
responseParameters	x-amazon-apigateway-gateway- 응답, 응답 파라미터	매개 변수, 즉 헤더 GatewayResponse매개 변수를 지정합니다. 파라미터 값은 수신되는 요청 파라미터 값 또는 정적 사용자 지정 값을 가질 수 있습니다.
responseTemplates	x-amazon-apigateway-gateway- 응답. 응답 템플릿	게이트웨이 응답의 매핑 템플릿을 지정합니다. 템플릿은 VTL 엔진에 의해 처리되지 않습니다.
statusCode	`string`	게이트웨이 응답에 대한 HTTP 상태 코드입니다.

x-amazon-apigateway-gateway- 응답. 게이트웨이 응답 예제

OpenAPI에 대한 API Gateway 확장의 다음 예제는 상태 코드456, 수신 요청의 `api-key` 헤더 값 및 메시지를 반환하도록 `INVALID_API_KEY` 응답을 사용자 지정하는 `"Bad api-key"` a를 정의합니다. GatewayResponse

```
    "INVALID_API_KEY": {
```

```
    "statusCode": "456",
    "responseParameters": {
      "gatewayresponse.header.api-key": "method.request.header.api-key"
    },
    "responseTemplates": {
      "application/json": "{\"message\": \"Bad api-key\" }"
    }
  }
```

x-amazon-apigateway-gateway-responses.responseParameters 객체

들어오는 요청 파라미터로부터 또는 리터럴 문자열을 사용하여 게이트웨이 응답 파라미터를 생성하기 위한 키-값 쌍의 string-to-string 맵을 정의합니다. REST API에서만 지원됩니다.

속성

속성 이름	유형	설명
gatewayresponse. *param-position* .*param-name*	string	*param-position* 은 header, path 또는 querystring 일 수 있습니다. 자세한 설명은 통합 요청 파라미터에 메서드 요청 데이터 매핑 섹션을 참조하세요.

x-amazon-apigateway-gateway-responses.responseParameter 예제

다음 OpenAPI 확장 예제는 도메인의 리소스에 대한 CORS 지원을 활성화하는 GatewayResponse응답 파라미터 매핑 식을 보여줍니다. `*.example.domain`

```
    "responseParameters": {
      "gatewayresponse.header.Access-Control-Allow-Origin": '*.example.domain',
      "gatewayresponse.header.from-request-header" : method.request.header.Accept,
      "gatewayresponse.header.from-request-path" : method.request.path.petId,
      "gatewayresponse.header.from-request-query" : method.request.querystring.qname
    }
```

x-amazon-apigateway-gateway-responses.ResponseTemplates 객체

지정된 게이트웨이 응답에 대한 GatewayResponse매핑 템플릿을 키-값 쌍의 string-to-string 맵으로 정의합니다. 각 키-값 페어에 대해 키는 콘텐츠 유형입니다. 예를 들어, “application/json”과 그 값은 간단한 변수 대체를 위해 문자열로 변환된 매핑 템플릿입니다. GatewayResponse 매핑 템플릿은 VTL(Velocity Template Language) 엔진에 의해 처리되지 않습니다.

속성

속성 이름	유형	설명
content-type	string	게이트웨이 응답 본문을 사용자 지정하는 단순 변수 대체만 지원하는 GatewayResponse 본문 매핑 템플릿입니다.

x-amazon-apigateway-gateway- response. 응답 템플릿 예제

다음 OpenAPI 확장 예제는 API Gateway에서 생성된 오류 응답을 앱별 형식으로 사용자 지정하는 GatewayResponse매핑 템플릿을 보여줍니다.

```
      "responseTemplates": {
        "application/json": "{ \"message\": $context.error.messageString, \"type\": 
$context.error.responseType, \"statusCode\": '488' }"
      }
```

다음 OpenAPI 확장 예제는 API Gateway에서 생성된 오류 응답을 정적 오류 메시지로 재정의하는 GatewayResponse매핑 템플릿을 보여줍니다.

```
      "responseTemplates": {
        "application/json": "{ \"message\": 'API-specific errors' }"
      }
```

x-amazon-apigateway-importexport-버전

HTTP API에 대한 API Gateway 가져오기 및 내보내기 알고리즘의 버전을 지정합니다. 현재 지원되는 값은 `1.0`입니다. 자세한 내용은 API Gateway 버전 2 API 참조의 exportVersion을 참조하세요.

x-amazon-apigateway-importexport-버전 예제

다음 예제에서는 가져오기 및 내보내기 버전을 `1.0`으로 설정합니다.

```
{
    "openapi": "3.0.1",
    "x-amazon-apigateway-importexport-version": "1.0",
    "info": { ...
```

x-amazon-apigateway-integration 객체

이 메서드에 사용된 백엔드 통합의 세부 정보를 지정합니다. 이 확장은 OpenAPI 작업 객체의 확장된 속성입니다. 결과는 API Gateway 통합 객체입니다.

속성

속성 이름	유형	설명
`cacheKeyParameters`	`string`의 어레이	값을 캐시할 요청 파라미터의 목록입니다.
`cacheNamespace`	`string`	캐시된 관련 파라미터의 API 관련 태그 그룹입니다.
`connectionId`	`string`	프라이빗 통합을 VpcLink위한 a의 ID.
`connectionType`	`string`	통합 연결 유형. 유효 값은 프라이빗 통합의 경우, `"VPC_LINK"` 이고 그 밖의 경우에는 `"INTERNET"` 입니다.
`credentials`	`string`	AWS IAM 역할 기반 자격 증명의 경우 적절한 IAM 역할의

속성 이름	유형	설명
		ARN을 지정하십시오. 지정되어 있지 않으면 자격 증명이 기본적으로 리소스 기반 권한으로 설정되어 API가 리소스에 액세스할 수 있으려면 이 권한을 수동으로 추가해야 합니다. 자세한 내용은 리소스 정책을 사용하여 권한 부여 단원을 참조하세요. 참고: IAM 자격 증명을 사용할 때 최상의 성능을 위해 이 API가 배포된 리전에서 AWS STS 리전 엔드포인트가 활성화되어 있는지 확인하세요.
`contentHandling`	`string`	요청 페이로드 인코딩 변환 유형입니다. 유효한 값은 1) 이진 페이로드를 Base64 인코딩 문자열로 변환하거나, 텍스트 페이로드를 `utf-8` 인코딩 문자열로 변환하거나, 기본적으로 수정 없이 텍스트 페이로드를 패스스루하는 경우에는 `CONVERT_TO_TEXT` 이고, 2) 텍스트 페이로드를 Base64 디코딩 BLOB로 변환하거나 기본적으로 수정 없이 이진 페이로드를 패스스루하는 경우에는 `CONVERT_TO_BINARY` 입니다.

속성 이름	유형	설명
`httpMethod`	`string`	통합 요청에서 사용된 HTTP 메서드입니다. Lambda 함수 호출의 경우 값은 `POST`여야 합니다.
`integrationSubtype`	`string`	서비스 통합을 위한 통합 하위 유형을 지정합니다. AWS HTTP API에서만 지원됩니다. 지원되는 통합 하위 유형은 the section called “AWS 서비스 통합 참조” 단원을 참조하세요.
`passthroughBehavior`	`string`	매핑되지 않은 콘텐츠 형식의 요청 페이로드를 수정 없이 통합 요청을 통해 전달하는 방식을 지정합니다. 지원되는 값은 `when_no_templates` , `when_no_match` 및 `never`입니다. 자세한 내용은 Integration.passthroughBehavior를 참조하세요.
`payloadFormatVersion`	`string`	통합으로 전송되는 페이로드의 형식을 지정합니다. HTTP API에 필요합니다. HTTP API의 경우 Lambda 프록시 통합에 대해 지원되는 값은 `1.0` 및 `2.0`입니다. 다른 모든 통합의 경우 `1.0`은 유일하게 지원되는 값입니다. 자세한 내용은 the section called “AWS Lambda 통합” 및 the section called “AWS 서비스 통합 참조” 단원을 참조하세요.

속성 이름	유형	설명
requestParameters	x-amazon-apigateway-integration. 요청 매개변수 객체	REST API의 경우 메서드 요청 파라미터에서 통합 요청 파라미터로의 매핑을 지정합니다. 지원되는 요청 파라미터는 querystring , path, header 및 body입니다. HTTP API의 경우 요청 파라미터는 지정된 AWS_PROXY 과의 integrationSubtype 통합에 전달되는 파라미터를 지정하는 키-값 맵입니다. 정적 값을 제공하거나 런타임 시 평가되는 요청 데이터, 스테이지 변수 또는 컨텍스트 변수를 매핑할 수 있습니다. 자세한 내용은 the section called “AWS 서비스 통합” 단원을 참조하세요.
requestTemplates	x-amazon-apigateway-integration. 요청 템플릿 객체	지정된 MIME 유형의 요청 페이로드에 대한 매핑 템플릿입니다.
responses	x-amazon-apigateway-integration.response 객체	메서드의 응답을 정의하고 통합 응답에서 메서드 응답으로 원하는 파라미터 매핑 또는 페이로드 매핑을 지정합니다.
timeoutInMillis	integer	50ms~29,000ms 사이의 통합 제한 시간.

속성 이름	유형	설명
type	string	지정된 백엔드와의 통합 유형입니다. 유효한 값은 다음과 같습니다. • http 또는 http_proxy : HTTP 백엔드와 통합 시 • aws_proxy , AWS Lambda 함수와의 통합을 위한 것입니다. • aws, AWS Lambda 함수 또는 Amazon DynamoDB, Amazon 단순 알림 서비스 또는 Amazon 단순 대기열 서비스와 같은 AWS 다른 서비스와의 통합에 사용합니다. • mock: 백엔드를 호출하지 않고 API Gateway와 통합 시 통합 유형에 대한 자세한 내용은 integration:type을 참조하세요.
tlsConfig	the section called “x-amazon-apigateway-integration.tlsConfig”	통합을 위한 TLS 구성을 지정합니다.
uri	string	백엔드의 엔드포인트 URI입니다. aws 유형 통합의 경우 이는 ARN 값입니다. HTTP 통합의 경우 이는 https 또는 http 스키마를 포함한 HTTP 엔드포인트의 URL입니다.

x-amazon-apigateway-integration 예제

HTTP API의 경우, OpenAPI 정의의 구성 요소 섹션에서 통합을 정의할 수 있습니다. 자세한 내용은 x-amazon-apigateway-integrations 객체 단원을 참조하세요.

```
"x-amazon-apigateway-integration": {
    "$ref": "#/components/x-amazon-apigateway-integrations/integration1"
}
```

다음 예제에서는 Lambda 함수와의 통합을 생성합니다. 데모를 위해 `requestTemplates`에 표시된 샘플 매핑 템플릿과 아래 예의 `responseTemplates`는 `{ "name":"value_1", "key":"value_2", "redirect": {"url" :"..."} }`의 JSON 출력 또는 `{ "stage":"value_1", "user-id":"value_2" }`의 XML 출력을 생성하기 위해 다음 JSON 형식의 페이로드에 적용되는 것으로 가정됩니다. `<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'"
            }
        }
    }
}
```

매핑 템플릿에서 JSON 문자열의 큰따옴표 (")는 문자열 이스케이프 (\")여야 합니다.

x-amazon-apigateway-integrations 객체

통합의 모음을 정의합니다. OpenAPI 정의의 구성 요소 섹션에서 통합을 정의하고 여러 라우팅에서 통합을 다시 사용할 수 있습니다. HTTP API에서만 지원됩니다.

속성

속성 이름	유형	설명
##	x-amazon-apigateway-integration 객체	통합 객체의 모음입니다.

x-amazon-apigateway-integrations 예

다음 예제에서는 두 개의 통합을 정의하는 HTTP API를 생성하고, `$ref": "#/components/x-amazon-apigateway-integrations/`*integration-name*을 사용하여 통합을 참조합니다.

```
{
  "openapi": "3.0.1",
  "info":
    {
      "title": "Integrations",
      "description": "An API that reuses integrations",
      "version": "1.0"
    },
  "servers": [
  {
    "url": "https://example.com/{basePath}",
    "description": "The production API server",
    "variables":
      {
        "basePath":
          {
            "default": "example/path"
          }
      }
  }],
  "paths":
    {
      "/":
        {
          "get":
            {
              "x-amazon-apigateway-integration":
                {
                  "$ref": "#/components/x-amazon-apigateway-integrations/integration1"

                }
            }
        },
       "/pets":
        {
          "get":
            {
              "x-amazon-apigateway-integration":
```

```
                {
                  "$ref": "#/components/x-amazon-apigateway-integrations/integration1"

                }
            }
        },
       "/checkout":
        {
          "get":
            {
              "x-amazon-apigateway-integration":
                {
                  "$ref": "#/components/x-amazon-apigateway-integrations/integration2"
                }
            }
        }
    },
    "components": {
      "x-amazon-apigateway-integrations":
        {
          "integration1":
            {
              "type": "aws_proxy",
              "httpMethod": "POST",
              "uri": "arn:aws:apigateway:us-east-2:lambda:path/2015-03-31/functions/
arn:aws:lambda:us-east-2:123456789012:function:my-function/invocations",
              "passthroughBehavior": "when_no_templates",
              "payloadFormatVersion": "1.0"
            },
          "integration2":
            {
              "type": "aws_proxy",
              "httpMethod": "POST",
              "uri": "arn:aws:apigateway:us-east-2:lambda:path/2015-03-31/functions/
arn:aws:lambda:us-east-2:123456789012:function:example-function/invocations",
              "passthroughBehavior": "when_no_templates",
              "payloadFormatVersion" : "1.0"
            }
        }
    }
}
```

x-amazon-apigateway-integration. 요청 템플릿 객체

지정된 MIME 유형의 요청 페이로드에 대한 매핑 템플릿을 지정합니다.

속성

속성 이름	유형	설명
MIME type	string	MIME 유형의 예는 application/json 입니다. 매핑 템플릿 생성에 대한 자세한 내용은 PetStore 매핑 템플릿 단원을 참조하세요.

x-amazon-apigateway-integration. 요청 템플릿 예제

다음 예는 application/json 및 application/xml MIME 유형의 요청 페이로드에 대한 매핑 템플릿을 설정합니다.

```
"requestTemplates" : {
    "application/json" : "#set ($root=$input.path('$')) { \"stage\": \"$root.name\",
 \"user-id\": \"$root.key\" }",
    "application/xml" : "#set ($root=$input.path('$')) <stage>$root.name</stage> "
}
```

x-amazon-apigateway-integration. 요청 매개변수 객체

REST API의 경우 명명된 메서드 요청 파라미터에서 통합 요청 파라미터로의 매핑을 지정합니다. 참조되기 전에 메서드 요청 파라미터를 정의해야 합니다.

HTTP API에 대해 지정된 AWS_PROXY과의 integrationSubtype 통합에 전달되는 파라미터를 지정합니다.

속성

속성 이름	유형	설명
integration.request. *<param-type>* .*<param-name>*	string	REST API의 경우 값은 주로 method.request. *<param-type>* .*<param-name>* 형식의 사전 정의된 메서드 요청 파라미터입니다. 여기서 <param-type> 은 querystring , path, header 또는 body일 수 있습니다. 하지만 $context. *VARIABLE_NAME* , $stageVariables. *VARIABLE_NAME* , *STATIC_VALUE* 도 유효합니다. body 파라미터의 경우 <param-name> 은 $. 접두사가 없는 JSON 경로 표현식입니다.
parameter	string	HTTP API의 경우 요청 파라미터는 지정된 AWS_PROXY 과의 integrationSubtype 통합에 전달되는 파라미터를 지정하는 키-값 맵입니다. 정적 값을 제공하거나 런타임 시 평가되는 요청 데이터, 스테이지 변수 또는 컨텍스트 변수를 매핑할 수 있습니다. 자세한 내용은 the section called “AWS 서비스 통합” 단원을 참조하세요.

x-amazon-apigateway-integration.requestParameters 예제

다음에 나오는 요청 파라미터 매핑 예제에서는 메서드 요청의 쿼리(version), 헤더(x-user-id) 및 경로(service) 파라미터를 통합 요청의 쿼리(stage), 헤더(x-userid) 및 경로 파라미터(op)로 각각 변환합니다.

> **Note**
>
> OpenAPI 또는 AWS CloudFormation를 통해 리소스를 생성하는 경우 정적 값은 작은따옴표로 묶어야 합니다.
> 콘솔에서 이 값을 추가하려면 상자에 application/json을 따옴표 없이 입력합니다.

```
"requestParameters" : {
    "integration.request.querystring.stage" : "method.request.querystring.version",
    "integration.request.header.x-userid" : "method.request.header.x-user-id",
    "integration.request.path.op" : "method.request.path.service"
},
```

x-amazon-apigateway-integration.response 객체

메서드의 응답을 정의하고 통합 응답에서 메서드 응답으로 파라미터 매핑 또는 페이로드 매핑을 지정합니다.

속성

속성 이름	유형	설명
## ## ##	x-amazon-apigateway-integration.response 객체	메서드 응답에 대한 통합 응답을 일치시키는 데 사용되는 정규식이거나 구성하지 않은 응답을 포착하는 default입니다. HTTP 통합의 경우 이 regex가 통합 응답 상태 코드

속성 이름	유형	설명
		에 적용됩니다. Lambda 호출의 경우, Lambda 함수 실행에서 예외가 발생할 때 정규 표현식이 실패 응답 AWS Lambda 본문으로 반환된 오류 정보 객체의 필드에 적용됩니다. `errorMessage` **Note** `## ## ##` 속성 이름은 응답 상태 코드 그룹을 설명하는 응답 상태 코드 또는 정규 표현식을 참조합니다. API Gateway REST API에 있는 IntegrationResponse리소스의 식별자와는 일치하지 않습니다.

`x-amazon-apigateway-integration.responses` 예제

다음 예에서는 `2xx` 및 `302` 응답의 응답 목록을 표시합니다. `2xx` 응답의 경우 메서드 응답이 `application/json` 또는 `application/xml` MIME 유형의 통합 응답의 페이로드에서 매핑됩니다. 이 응답에서는 제공된 매핑 템플릿을 사용합니다. `302` 응답의 경우 메서드 응답은 통합 응답의 페이로드에 있는 `Location` 속성에서 값이 파생되는 `redirect.url` 헤더를 반환합니다.

```
"responses" : {
    "2\\d{2}" : {
        "statusCode" : "200",
        "responseTemplates" : {
            "application/json" : "#set ($root=$input.path('$')) { \"stage\":
 \"$root.name\", \"user-id\": \"$root.key\" }",
```

```
            "application/xml" : "#set ($root=$input.path('$')) <stage>$root.name</
stage> "
        }
    },
    "302" : {
        "statusCode" : "302",
        "responseParameters" : {
            "method.response.header.Location": "integration.response.body.redirect.url"
        }
    }
}
```

x-amazon-apigateway-integration.response 객체

응답을 정의하고 통합 응답에서 메서드 응답으로 파라미터 매핑 또는 페이로드 매핑을 지정합니다.

속성

속성 이름	유형	설명
`statusCode`	`string`	메서드 응답에 대한 HTTP 상태 코드입니다(예: "200"). 이는 OpenAPI 작업 `responses` 필드의 일치하는 응답에 해당해야 합니다.
`responseTemplates`	x-amazon-apigateway-integration. 응답 템플릿 객체	응답의 페이로드에 대한 MIME 유형 관련 매핑 템플릿을 지정합니다.
`responseParameters`	x-amazon-apigateway-integration. 응답 매개변수 객체	응답에 대한 파라미터 매핑을 지정합니다. 통합 응답의 `header` 및 `body` 파라미터만 메서드의 `header` 파라미터에 매핑될 수 있습니다.
`contentHandling`	`string`	응답 페이로드 인코딩 변환 유형입니다. 유효한 값은 1) 이

속성 이름	유형	설명
		진 페이로드를 Base64 인코딩 문자열로 변환하거나, 텍스트 페이로드를 utf-8 인코딩 문자열로 변환하거나, 기본적으로 수정 없이 텍스트 페이로드를 패스스루하는 경우에는 CONVERT_TO_TEXT 이고, 2) 텍스트 페이로드를 Base64 디코딩 BLOB로 변환하거나 기본적으로 수정 없이 이진 페이로드를 패스스루하는 경우에는 CONVERT_TO_BINARY 입니다.

x-amazon-apigateway-integration.response 예제

다음 예는 백엔드에서 302 또는 application/json MIME 유형의 페이로드를 추출하는 메서드에 대한 application/xml 응답을 정의합니다. 응답에서는 제공된 매핑 템플릿을 사용하고 메서드의 Location 헤더에 있는 통합 응답에서 리디렉션 URL을 반환합니다.

```
{
    "statusCode" : "302",
    "responseTemplates" : {
         "application/json" : "#set ($root=$input.path('$')) { \"stage\": \"$root.name
\", \"user-id\": \"$root.key\" }",
         "application/xml" : "#set ($root=$input.path('$')) <stage>$root.name</stage> "
    },
    "responseParameters" : {
        "method.response.header.Location": "integration.response.body.redirect.url"
    }
}
```

x-amazon-apigateway-integration. 응답 템플릿 객체

지정된 MIME 유형의 응답 페이로드에 대한 매핑 템플릿을 지정합니다.

속성

속성 이름	유형	설명
MIME type	string	지정된 MIME 유형에 대한 통합 응답 본문을 메서드 응답 본문으로 변환하는 매핑 템플릿을 지정합니다. 매핑 템플릿 생성에 대한 자세한 내용은 PetStore 매핑 템플릿 단원을 참조하세요. *MIME ##*의 예는 application/json 입니다.

x-amazon-apigateway-integration. 응답 템플릿 예제

다음 예는 application/json 및 application/xml MIME 유형의 요청 페이로드에 대한 매핑 템플릿을 설정합니다.

```
"responseTemplates" : {
    "application/json" : "#set ($root=$input.path('$')) { \"stage\": \"$root.name\",
 \"user-id\": \"$root.key\" }",
    "application/xml" : "#set ($root=$input.path('$')) <stage>$root.name</stage> "
}
```

x-amazon-apigateway-integration. 응답 매개변수 객체

통합 메서드 응답 파라미터에서 메서드 응답 파라미터로의 매핑을 지정합니다. header, body 또는 정적 값을 메서드 응답의 header 유형에 매핑할 수 있습니다. REST API에서만 지원됩니다.

속성

속성 이름	유형	설명
`method.response.header. <param-name>`	`string`	명명된 파라미터 값은 통합 응답 파라미터의 `header` 및 `body` 유형에서 추출할 수 있습니다.

`x-amazon-apigateway-integration.responseParameters` 예제

다음 예는 통합 응답의 `body` 및 `header` 파라미터를 메서드 응답의 두 `header` 파라미터에 매핑합니다.

```
"responseParameters" : {
    "method.response.header.Location" : "integration.response.body.redirect.url",
    "method.response.header.x-user-id" : "integration.response.header.x-userid"
}
```

x-amazon-apigateway-integration.tlsConfig 객체

통합을 위한 TLS 구성을 지정합니다.

속성

속성 이름	유형	설명
`insecureSkipVerification`	`Boolean`	REST API에서만 지원됩니다. 통합 엔드포인트에 대한 인증서가 지원되는 인증 기관에서 발행되었다는 확인을 API Gateway에서 건너뛸지 여부를 지정합니다. 권장되지 않는 방법이지만 사설 인증 기관에서 서명한 인증서 또는 자체 서명된 인증서를 사용할 수

속성 이름	유형	설명
		있습니다. 활성화된 경우 API Gateway가 인증서의 만료 날짜, 호스트 이름 및 루트 인증 기관의 존재 여부를 확인하는 등 기본적인 인증서 유효성 검사를 수행합니다. 사설 기관에 속하는 루트 인증서는 다음 제약 조건을 충족해야 합니다. • x509 확장 `keyUsage`에 `keyCertSign` 이 있어야 합니다. • x509 확장 `basicConstraints` 에 `CA:TRUE`이 있어야 합니다. `HTTP` 및 `HTTP_PROXY` 통합에서만 지원됩니다. ⚠ Warning `insecureSkipVerification` 의 활성화는 특히 퍼블릭 HTTPS 엔드포인트와의 통합에 대해 권장되지 않습니다. `insecureSkipVerification` 활성화하면 공격 위험이 높아집니다. man-in-the-middle

속성 이름	유형	설명
serverNameToVerify	string	HTTP API 프라이빗 통합에서만 지원됩니다. 서버 이름을 지정하면 API Gateway는 이 이름을 사용하여 통합 인증서의 호스트 이름을 확인합니다. 서버 이름은 서버 이름 표시(SNI) 또는 가상 호스팅을 지원하기 위해 TLS 핸드셰이크에도 포함됩니다.

x-amazon-apigateway-integration.tlsConfig 예제

다음 OpenAPI 3.0 예제는 REST API HTTP 프록시 통합을 위한 insecureSkipVerification을 활성화합니다.

```
"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",
}
```

다음 OpenAPI 3.0 예제는 HTTP API 프라이빗 통합을 위한 serverNameToVerify를 지정합니다.

```
"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- 압축 크기

REST API에 대한 최소 압축 크기를 지정합니다. 압축을 활성화하려면 0에서 10485760 사이의 정수를 지정합니다. 자세한 내용은 API에 대한 페이로드 압축 활성화 단원을 참조하세요.

x-amazon-apigateway-minimum-압축 크기 예제

다음 예제에서는 REST API에 대한 최소 압축 크기(5242880바이트)를 지정합니다.

```
"x-amazon-apigateway-minimum-compression-size": 5242880
```

x-amazon-apigateway-policy

REST API에 대한 리소스 정책을 지정합니다. 리소스 정책에 대해 자세히 알아보려면 API Gateway 리소스 정책을 사용하여 API에 대한 액세스 제어 단원을 참조하세요. 리소스 정책 예제는 API Gateway 리소스 정책 예제 단원을 참조하세요.

`x-amazon-apigateway-policy` 예제

다음 예제에서는 REST API에 대한 리소스 정책을 지정합니다. 리소스 정책은 지정된 소스 IP 주소 블록에서 API로 들어오는 트래픽을 거부(차단)합니다. 가져오기 시 현재 지역`arn:aws:execute-api:`*`region`*`:`*`account-id`*`:`*`api-id`*`/*`, AWS 계정 ID, 현재 REST API ID를 사용하여 `"execute-api:/*"` 변환됩니다.

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

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

x-amazon-apigateway-request-유효성 검사기 속성

포함하는 API 또는 메서드에 대한 요청 확인을 활성화하기 위해 *request_validator_name* 맵의 x-amazon-apigateway-request-유효성 검사기 개체을 참조하여 요청 검사기를 지정합니다. 이 확장의 값은 JSON 문자열입니다.

이 확장은 API 레벨 또는 메서드 레벨에서 지정할 수 있습니다. 메서드 레벨 검사기에서 재정의하지 않은 경우 API 레벨 검사기가 모든 메서드에 적용됩니다.

`x-amazon-apigateway-request-validator` 예제

다음 예는 `basic` 요청에서 `parameter-only` 요청 검사기를 적용하는 동안 `POST /validation` 요청 검사기를 API 레벨에서 적용합니다.

OpenAPI 2.0

```
{
  "swagger": "2.0",
  "x-amazon-apigateway-request-validators" : {
    "basic" : {
      "validateRequestBody" : true,
      "validateRequestParameters" : true
    },
```

```
    "params-only" : {
      "validateRequestBody" : false,
      "validateRequestParameters" : true
    }
  },
  "x-amazon-apigateway-request-validator" : "basic",
  "paths": {
    "/validation": {
      "post": {
        "x-amazon-apigateway-request-validator" : "params-only",
        ...
     }
}
```

x-amazon-apigateway-request-유효성 검사기 개체

포함하는 API에 대해 지원되는 요청 검사기를 검사기 이름과 해당하는 요청 확인 규칙 간의 맵으로 정의합니다. 이 확장은 REST API에 적용됩니다.

속성

속성 이름	유형	설명
request_validator_name	x-amazon-apigateway-request - 유효성 검사기. 요청 유효성 검사기 개체	명명된 검사기로 이루어진 확인 규칙을 지정합니다. 예: `"basic" : { "validateRequestBody" : true, "validateRequestParameters" : true },` 이 검사기를 특정 메서드에 적용하려면 검사기 이름(basic)을 x-amazon-apigateway-request-유효성 검사기 속성 속성의 값으로 참조합니다.

x-amazon-apigateway-request-validators 예제

다음 예는 검사기 이름 및 연관된 요청 검사기 규칙 간의 맵으로 API에 대한 요청 검사기 세트를 보여 줍니다.

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- 유효성 검사기. 요청 유효성 검사기 개체

요청 검사기의 확인 규칙을 x-amazon-apigateway-request-유효성 검사기 개체 맵 정의의 한 부분으로 지정합니다.

속성

속성 이름	유형	설명
validateRequestBody	Boolean	요청 본문을 확인할지 여부를 지정합니다(true 또는 false).

속성 이름	유형	설명
validateRequestParameters	Boolean	필수 요청 파라미터를 확인할지 여부를 지정합니다(true 또는 false).

x-amazon-apigateway-request-validators.requestValidator 예제

다음 예는 파라미터 전용 요청 검사기를 표시합니다.

```
"params-only": {
    "validateRequestBody" : false,
    "validateRequestParameters" : true
}
```

x-amazon-apigateway-tag- 값 속성

HTTP API에 대한 AWS 태그의 값을 지정합니다. 이 x-amazon-apigateway-tag-value 속성을 루트 수준 OpenAPI 태그 객체의 일부로 사용하여 HTTP API용 AWS 태그를 지정할 수 있습니다. x-amazon-apigateway-tag-value 속성 없이 태그 이름을 지정하면 API Gateway에서 값에 대해 빈 문자열이 포함된 태그가 생성됩니다.

태그 지정에 대한 자세한 내용은 API Gateway 리소스 태그 지정 단원을 참조하세요.

속성

속성 이름	유형	설명
name	String	태그 키를 지정합니다.
x-amazon-apigateway-tag-value	String	태그 값을 지정합니다.

x-amazon-apigateway-tag-value 예제

다음 예제에서는 HTTP API에 대해 두 개의 태그를 지정합니다.

- "Owner": "Admin"
- "Prod": ""

```
"tags": [
    {
      "name": "Owner",
      "x-amazon-apigateway-tag-value": "Admin"
    },
    {
      "name": "Prod"
    }
]
```

Amazon API Gateway의 보안

AWS에서 클라우드 보안을 가장 중요하게 생각합니다. AWS 고객은 보안에 매우 민감한 조직의 요구 사항에 부합하도록 구축된 데이터 센터 및 네트워크 아키텍처의 혜택을 누릴 수 있습니다.

보안은 AWS와 귀하의 공동 책임입니다. 공동 책임 모델은 이 사항을 클라우드의 보안 및 클라우드 내 보안으로 설명합니다.

- 클라우드의 보안 - AWS는 AWS 클라우드에서 AWS 서비스를 실행하는 인프라를 보호합니다. AWS는 또한 안전하게 사용할 수 있는 서비스를 제공합니다. 타사 감사자는 AWS 규정 준수 프로그램의 일환으로 보안 효과를 정기적으로 테스트하고 검증합니다. Amazon API Gateway에 적용되는 규정 준수 프로그램에 대한 자세한 내용은 규정 준수 프로그램 제공 범위 내 AWS 서비스 를 참조하세요.
- 클라우드 내 보안 – 귀하의 책임은 귀하가 사용하는 AWS 서비스에 의해 결정됩니다. 또한 귀하는 데이터의 민감도, 회사 요구 사항, 관련 법률 및 규정을 비롯한 기타 요소에 대해서도 책임이 있습니다.

이 설명서는 API Gateway를 사용할 때 공동 책임 모델을 적용하는 방법을 이해하는 데 도움이 됩니다. 다음 주제에서는 보안 및 규정 준수 목적에 맞게 API Gateway를 구성하는 방법을 보여줍니다. 또한 API Gateway 리소스를 모니터링하고 보호하는 데 도움이 되는 다른 AWS 서비스를 사용하는 방법을 배우게 됩니다.

자세한 내용은 Amazon API Gateway의 보안 개요를 참조하세요.

주제

- Amazon API Gateway에서의 데이터 보호
- Amazon API Gateway의 Identity and Access Management
- Amazon API Gateway에서 로깅 및 모니터링
- Amazon API Gateway에 대한 규정 준수 확인
- Amazon API Gateway의 복원성
- Amazon API Gateway의 인프라 보안
- Amazon API Gateway의 취약성 분석
- Amazon API Gateway의 보안 모범 사례

Amazon API Gateway에서의 데이터 보호

AWS 공동 책임 모델은 Amazon API Gateway의 데이터 보호에 적용됩니다. 이 모델에서 설명하는 것처럼 AWS는 모든 AWS 클라우드를 실행하는 글로벌 인프라를 보호할 책임이 있습니다. 이 인프라에서 호스팅되는 콘텐츠에 대한 제어를 유지하는 것은 사용자의 책임입니다. 사용하는 AWS 서비스의 보안 구성과 관리 작업에 대한 책임도 사용자에게 있습니다. 데이터 프라이버시에 대한 자세한 내용은 데이터 프라이버시 FAQ를 참조하십시오. 유럽의 데이터 보호에 대한 자세한 내용은 AWS 보안 블로그의 AWS 공동 책임 모델 및 GDPR 블로그 게시물을 참조하십시오.

데이터를 보호하려면 AWS 계정 보안 인증 정보를 보호하고 AWS IAM Identity Center 또는 AWS Identity and Access Management(IAM)을 통해 개별 사용자 계정을 설정하는 것이 좋습니다. 이러한 방식에는 각 사용자에게 자신의 직무를 충실히 이행하는 데 필요한 권한만 부여됩니다. 또한 다음과 같은 방법으로 데이터를 보호하는 것이 좋습니다.

- 각 계정에 복수 인증(MFA)을 사용합니다.
- SSL/TLS를 사용하여 AWS리소스와 통신합니다. TLS 1.2는 필수이며 TLS 1.3을 권장합니다.
- AWS CloudTrail(으)로 API 및 사용자 활동 로깅을 설정합니다.
- AWS 암호화 솔루션을 AWS 서비스 내의 모든 기본 보안 제어와 함께 사용합니다.
- Amazon S3에 저장된 민감한 데이터를 검색하고 보호하는 데 도움이 되는 Amazon Macie와 같은 고급 관리형 보안 서비스를 사용합니다.
- 명령행 인터페이스 또는 API를 통해 AWS에 액세스할 때 FIPS 140-2 검증된 암호화 모듈이 필요한 경우, FIPS 엔드포인트를 사용합니다. 사용 가능한 FIPS 엔드포인트에 대한 자세한 내용은 FIPS(Federal Information Processing Standard) 140-2를 참조하십시오.

고객의 이메일 주소와 같은 기밀 정보나 중요한 정보는 태그나 이름 필드와 같은 자유 양식 텍스트 필드에 입력하지 않는 것이 좋습니다. 여기에는 콘솔, API, AWS CLI 또는 AWS SDK를 사용하여 API Gateway 또는 기타 AWS 서비스 서비스에서 작업하는 경우가 포함됩니다. 이름에 사용되는 태그 또는 자유 형식 텍스트 필드에 입력하는 모든 데이터는 청구 또는 진단 로그에 사용될 수 있습니다. 외부 서버에 URL을 제공할 때 해당 서버에 대한 요청을 검증하기 위해 보안 인증 정보를 URL에 포함해서는 안 됩니다.

Amazon API Gateway에서의 데이터 암호화

데이터 보호란 전송 중(API Gateway 안팎으로 데이터가 이동 중)과 유휴 시(에 데이터가 저장된 동안)에 데이터를 보호하는 것을 말합니다AWS

Amazon API Gateway에서 저장된 데이터 암호화

REST API에 대한 캐싱을 활성화하도록 선택한 경우 캐시 암호화를 활성화할 수 있습니다. 자세한 내용은 응답성 향상을 위한 API 캐싱 활성화 단원을 참조하십시오.

데이터 보호에 대한 자세한 내용은 AWS 보안 블로그의 AWS 공동 책임 모델 및 GDPR 블로그 게시물을 참조하세요.

Amazon API Gateway에서 전송 중 데이터 암호화

Amazon API Gateway를 통해 생성된 API는 HTTPS 엔드포인트만 제공합니다. API Gateway는 암호화되지 않은(HTTP) 엔드포인트를 지원하지 않습니다.

API Gateway는 기본 `execute-api` 엔드포인트의 인증서를 관리합니다. 사용자 지정 도메인 이름을 구성하는 경우 도메인 이름에 대한 인증서를 지정합니다. 인증서를 고정하지 않는 것이 좋습니다.

보안 강화를 위해 API Gateway 사용자 지정 도메인에 적용할 최소 전송 계층 보안(TLS) 프로토콜 버전을 선택할 수 있습니다. WebSocket API 및 HTTP API는 TLS 1.2만 지원합니다. 자세한 내용은 API Gateway에서 사용자 지정 도메인에 대한 보안 정책 선택 단원을 참조하십시오.

계정에서 사용자 정의 SSL 인증서를 사용하여 Amazon CloudFront 배포를 설정하고 리전 API를 통해 사용할 수 있습니다. 그런 다음 보안 및 규정 준수 사항에 따라 TLS 1.1 이상을 사용하는 CloudFront 배포에 대한 보안 정책을 구성할 수 있습니다.

데이터 보호에 대한 자세한 내용은 AWS 보안 블로그의 REST API 보호 및 AWS 공동 책임 모델 및 GDPR 블로그 게시물을 참조하세요.

인터네트워크 트래픽 개인 정보 보호

Amazon API Gateway를 사용하면 Amazon Virtual Private Cloud(VPC)에서만 액세스할 수 있는 프라이빗 REST API를 생성할 수 있습니다. VPC는 VPC에서 생성하는 엔드포인트 네트워크 인터페이스인 인터페이스 VPC 종단점를 사용합니다. 여러 AWS 계정은 물론 선택한 VPC 및 VPC 엔드포인트에서 API에 액세스하는 것을 리소스 정책으로 허용하거나 거부할 수 있습니다. 각 엔드포인트를 사용하여 여러 개의 프라이빗 API에 액세스할 수 있습니다. 또한 AWS Direct Connect를 사용하여 온프레미스 네트워크에서 Amazon VPC에 연결한 다음 그 연결을 통해 프라이빗 API에 액세스할 수도 있습니다. 어떤 경우에든 프라이빗 API로 가는 트래픽은 안전한 연결을 사용하고, Amazon 네트워크를 벗어나지 않으며, 퍼블릭 인터넷과 격리됩니다. 자세한 내용은 the section called “프라이빗 API” 단원을 참조하십시오.

Amazon API Gateway의 Identity and Access Management

AWS Identity and Access Management (IAM)는 관리자가 AWS 리소스에 대한 액세스를 안전하게 제어할 수 있도록 지원하는 AWS 서비스입니다. IAM 관리자는 어떤 사용자가 API Gateway 리소스를 사용하도록 인증(로그인)되고 권한 부여(권한 있음)될 수 있는지를 제어합니다. IAM은 추가 비용 없이 사용할 수 있는 AWS 서비스입니다.

주제

- 대상
- 자격 증명을 통한 인증
- 정책을 사용한 액세스 관리
- Amazon API Gateway에서 IAM을 사용하는 방식
- Amazon API Gateway 자격 증명 기반 정책 예제
- Amazon API Gateway 리소스 기반 정책 예제
- Amazon API Gateway 자격 증명 및 액세스 문제 해결
- API Gateway에 서비스 연결 역할 사용

대상

AWS Identity and Access Management(IAM) 사용 방법은 API Gateway에서 하는 작업에 따라 달라집니다.

서비스 사용자 - API Gateway 서비스를 사용하여 작업을 수행하는 경우 필요한 자격 증명과 권한을 관리자가 제공합니다. 더 많은 API Gateway 기능을 사용하여 작업을 수행하게 되면 추가 권한이 필요할 수 있습니다. 액세스 권한 관리 방식을 이해하면 적절한 권한을 관리자에게 요청할 수 있습니다. API Gateway의 기능에 액세스할 수 없는 경우 Amazon API Gateway 자격 증명 및 액세스 문제 해결 단원을 참조하십시오.

서비스 관리자 - 회사에서 API Gateway 리소스를 책임지고 있는 경우 API Gateway 사용하는 서비스에 대한 완전한 액세스 권한을 가지고 있을 것입니다. 서비스 관리자는 서비스 사용자가 액세스해야 하는 API Gateway 기능과 리소스를 결정합니다. 그런 다음, IAM 관리자에게 요청을 제출하여 서비스 사용자의 권한을 변경해야 합니다. 이 페이지의 정보를 검토하여 IAM의 기본 개념을 이해하십시오. 회사가 API Gateway에서 IAM을 사용할 수 있는 방법에 대해 자세히 알아보려면 Amazon API Gateway에서 IAM을 사용하는 방식 단원을 참조하십시오.

IAM 관리자 - IAM 관리자는 API Gateway에 대한 액세스 권한 관리 정책을 작성하는 방법에 대해 자세히 알아보려고 할 수 있습니다. IAM에서 사용할 수 있는 API Gateway 자격 증명 기반 정책 예제를 보려면 Amazon API Gateway 자격 증명 기반 정책 예제 단원을 참조하십시오.

자격 증명을 통한 인증

인증은 ID 보안 인증을 사용하여 AWS에 로그인하는 방식입니다. AWS 계정 루트 사용자이나 IAM 사용자로, 또는 IAM 역할을 수임하여 인증(AWS에 로그인)받아야 합니다.

보안 인증 소스를 통해 제공된 보안 인증 정보를 사용하여 페더레이션형 ID로 AWS에 로그인할 수 있습니다. AWS IAM Identity Center (IAM Identity Center) 사용자, 회사의 Single Sign-On 인증, Google 또는 Facebook 보안 인증이 페더레이션형 ID의 예입니다. 페더레이션형 ID로 로그인할 때 관리자가 이전에 IAM 역할을 사용하여 ID 페더레이션을 설정했습니다. 페더레이션을 사용하여 AWS에 액세스하면 간접적으로 역할을 수임합니다.

사용자 유형에 따라AWS Management Console 또는 AWS 액세스 포털에 로그인할 수 있습니다. AWS에 로그인하는 방법에 대한 자세한 내용은 AWS 로그인 사용 설명서의 AWS 계정에 로그인하는 방법을 참조하십시오.

AWS에 프로그래밍 방식으로 액세스하는 경우, AWS에서는 보안 인증 정보를 사용하여 요청에 암호화 방식으로 서명할 수 있는 소프트웨어 개발 키트(SDK) 및 명령줄 인터페이스(CLI)를 제공합니다. AWS 도구를 사용하지 않는다면 요청에 직접 서명해야 합니다. 권장 방법을 사용하여 요청에 직접 서명하는 방법에 대한 자세한 내용은 IAM 사용 설명서의 AWS API 요청에 서명을 참조하십시오.

사용하는 인증 방법에 상관없이 추가 보안 정보를 제공해야 할 수도 있습니다. 예를 들어, AWS는 다중 인증(MFA)을 사용하여 계정의 보안을 강화하는 것을 권장합니다. 자세한 내용은 AWS IAM Identity Center 사용 설명서의 다중 인증 및 IAM 사용 설명서의 AWS에서 다중 인증(MFA) 사용을 참조하십시오.

AWS 계정 루트 사용자

AWS 계정를 생성할 때는 해당 계정의 모든 AWS 서비스 및 리소스에 대해 완전한 액세스 권한이 있는 단일 로그인 ID로 시작합니다. 이 보안 인증은 AWS 계정 루트 사용자라고 하며, 계정을 생성할 때 사용한 이메일 주소와 암호로 로그인하여 액세스합니다. 일상적인 작업에는 루트 사용자를 사용하지 않는 것이 좋습니다. 루트 사용자 보안 인증 정보를 보호하고 루트 사용자만 수행할 수 있는 작업을 수행하는 데 사용합니다. 루트 사용자로 로그인해야 하는 전체 작업 목록은 IAM 사용 설명서의 루트 사용자 보안 인증이 필요한 태스크를 참조하십시오.

IAM 사용자 및 그룹

IAM 사용자는 단일 개인 또는 애플리케이션에 대한 특정 권한을 가지고 있는 AWS 계정 내 보안 인증입니다. 가능하다면 암호 및 액세스 키와 같은 장기 보안 인증 정보가 있는 IAM 사용자를 생성하는 대신, 임시 보안 인증 정보를 사용하는 것이 좋습니다. 하지만 IAM 사용자의 장기 보안 인증 정보가 필요한 특정 사용 사례가 있는 경우, 액세스 키를 교체하는 것이 좋습니다. 자세한 내용은 IAM 사용 설명서의 장기 보안 인증이 필요한 사용 사례의 경우 정기적으로 액세스 키 교체를 참조하십시오.

IAM 그룹은 IAM 사용자 컬렉션을 지정하는 보안 인증입니다. 사용자는 그룹으로 로그인할 수 없습니다. 그룹을 사용하여 여러 사용자의 권한을 한 번에 지정할 수 있습니다. 그룹을 사용하면 대규모 사용자 집합의 권한을 더 쉽게 관리할 수 있습니다. 예를 들어, IAMAdmins(이)라는 그룹이 있고 이 그룹에 IAM 리소스를 관리할 권한을 부여할 수 있습니다.

사용자는 역할과 다릅니다. 사용자는 한 사람 또는 애플리케이션과 고유하게 연결되지만, 역할은 해당 역할이 필요한 사람이라면 누구나 수임할 수 있습니다. 사용자는 영구적인 장기 보안 인증을 가지고 있지만, 역할은 임시 보안 인증만 제공합니다. 자세한 정보는 IAM 사용 설명서의 IAM 사용자를 만들어야 하는 경우(역할이 아님)를 참조하십시오.

IAM 역할

IAM 역할은 특정 권한을 가지고 있는 AWS 계정 계정 내 자격 증명입니다. IAM 사용자와 유사하지만, 특정 개인과 연결되지 않습니다. 역할을 전환하여 AWS Management Console에서 IAM 역할을 임시로 수임할 수 있습니다. AWS CLI 또는 AWS API 태스크를 호출하거나 맞춤 URL을 사용하여 역할을 수임할 수 있습니다. 역할 사용 방법에 대한 자세한 정보는 IAM 사용 설명서의 IAM 역할 사용을 참조하십시오.

임시 보안 인증이 있는 IAM 역할은 다음과 같은 상황에서 유용합니다.

- 페더레이션 사용자 액세스 - 페더레이션형 ID에 권한을 부여하려면 역할을 생성하고 해당 역할의 권한을 정의합니다. 페더레이션형 ID가 인증되면 이 ID는 역할과 연결되며 역할에 의해 정의된 권한이 부여됩니다. 페더레이션 역할에 대한 자세한 내용은 IAM 사용 설명서의 타사 자격 증명 공급자의 역할 만들기를 참조하십시오. IAM 자격 증명 센터를 사용하는 경우 권한 집합을 구성합니다. 인증 후 아이덴티티가 액세스할 수 있는 항목을 제어하기 위해 IAM Identity Center는 권한 세트를 IAM의 역할과 연관 짓습니다. 권한 세트에 대한 자세한 내용은 AWS IAM Identity Center 사용 설명서의 권한 세트를 참조하십시오.
- 임시 IAM 사용자 권한 - IAM 사용자 또는 역할은 IAM 역할을 수임하여 특정 작업에 대한 다양한 권한을 임시로 받을 수 있습니다.
- 크로스 계정 액세스 - IAM 역할을 사용하여 다른 계정의 사용자(신뢰할 수 있는 보안 주체)가 내 계정의 리소스에 액세스하도록 허용할 수 있습니다. 역할은 크로스 계정 액세스를 부여하는 기본적인 방

법입니다. 그러나 일부 AWS 서비스를 사용하면 (역할을 프록시로 사용하는 대신) 리소스에 정책을 직접 연결할 수 있습니다. 크로스 계정 액세스를 위한 역할과 리소스 기반 정책의 차이점을 알아보려면 IAM 사용 설명서의 IAM 역할과 리소스 기반 정책의 차이를 참조하십시오.

- 교차 서비스 액세스 – 일부 AWS 서비스는 다른 AWS 서비스의 기능을 사용합니다. 예를 들어 서비스에서 호출하면 일반적으로 해당 서비스는 Amazon EC2에서 애플리케이션을 실행하거나 Amazon S3에 객체를 저장합니다. 서비스는 호출하는 보안 주체의 권한을 사용하거나, 서비스 역할을 사용하거나, 또는 서비스 연결 역할을 사용하여 이 작업을 수행할 수 있습니다.
  - 전달 액세스 세션(FAS) - IAM 사용자 또는 역할을 사용하여 AWS에서 작업을 수행하는 사람은 보안 주체로 간주됩니다. 일부 서비스를 사용하는 경우 다른 서비스에서 다른 작업을 시작하는 작업을 수행할 수 있습니다. FAS는 AWS 서비스를 직접 호출하는 보안 주체의 권한과 요청하는 AWS 서비스를 함께 사용하여 다운스트림 서비스에 대한 요청을 수행합니다. FAS 요청은 서비스에서 완료를 위해 다른 AWS 서비스 또는 리소스와의 상호 작용이 필요한 요청을 받은 경우에만 이루어집니다. 이 경우 두 작업을 모두 수행할 수 있는 권한이 있어야 합니다. FAS 요청 시 정책 세부 정보는 전달 액세스 세션을 참조하세요.
  - 서비스 역할 - 서비스 역할은 서비스가 사용자를 대신하여 작업을 수행하기 위해 수임하는 IAM 역할입니다. IAM 관리자는 IAM 내에서 서비스 역할을 생성, 수정 및 삭제할 수 있습니다. 자세한 정보는 IAM 사용자 가이드의 AWS 서비스에 대한 권한을 위임할 역할 생성을 참조하세요.
  - 서비스 연결 역할 – 서비스 연결 역할은 AWS 서비스에 연결된 서비스 역할의 한 유형입니다. 서비스는 사용자를 대신하여 작업을 수행하기 위해 역할을 수임할 수 있습니다. 서비스 연결 역할은 AWS 계정에 나타나고, 서비스가 소유합니다. IAM 관리자는 서비스 연결 역할의 권한을 볼 수 있지만 편집은 할 수 없습니다.
- Amazon EC2에서 실행 중인 애플리케이션 – IAM 역할을 사용하여 EC2 인스턴스에서 실행되고 AWS CLI 또는 AWS API 요청을 수행하는 애플리케이션의 임시 보안 인증을 관리할 수 있습니다. 이는 EC2 인스턴스 내에 액세스 키를 저장할 때 권장되는 방법입니다. EC2 인스턴스에 AWS 역할을 할당하고 해당 역할을 모든 애플리케이션에서 사용할 수 있도록 하려면 인스턴스에 연결된 인스턴스 프로필을 생성합니다. 인스턴스 프로파일에는 역할이 포함되어 있으며 EC2 인스턴스에서 실행되는 프로그램이 임시 보안 인증을 얻을 수 있습니다. 자세한 정보는 IAM 사용 설명서의 IAM 역할을 사용하여 Amazon EC2 인스턴스에서 실행되는 애플리케이션에 권한 부여를 참조하십시오.

IAM 역할을 사용할지 또는 IAM 사용자를 사용할지를 알아보려면 IAM 사용 설명서의 IAM 역할(사용자 대신)을 생성하는 경우를 참조하십시오.

정책을 사용한 액세스 관리

정책을 생성하고 AWS ID 또는 리소스에 연결하여 AWS 내 액세스를 통제합니다. 정책은 자격 증명 또는 리소스와 연결될 때 해당 권한을 정의하는 AWS의 객체입니다. AWS는 보안 주체(사용자, 루트 사

용자 또는 역할 세션)가 요청을 보낼 때 이러한 정책을 평가합니다. 정책의 권한이 요청 허용 또는 거부 여부를 결정합니다. 대부분의 정책은 AWS에 JSON 설명서로서 저장됩니다. JSON 정책 문서의 구조와 콘텐츠에 대한 자세한 정보는 IAM 사용 설명서의 JSON 정책 개요를 참조하십시오.

관리자는 AWSJSON 정책을 사용하여 누가 무엇에 액세스할 수 있는지를 지정할 수 있습니다. 즉, 어떤 보안 주체가 어떤 리소스와 어떤 조건에서 작업을 수행할 수 있는지를 지정할 수 있습니다.

기본적으로, 사용자와 역할에는 어떠한 권한도 없습니다. 사용자에게 사용자가 필요한 리소스에서 작업을 수행할 권한을 부여하려면 IAM 관리자가 IAM 정책을 생성하면 됩니다. 그런 다음 관리자가 IAM 정책을 역할에 추가하고, 사용자가 역할을 수임할 수 있습니다.

IAM 정책은 작업을 수행하기 위해 사용하는 방법과 상관없이 작업에 대한 권한을 정의합니다. 예를 들어, `iam:GetRole` 작업을 허용하는 정책이 있다고 가정합니다. 해당 정책이 있는 사용자는 AWS Management Console, AWS CLI또는 AWSAPI에서 역할 정보를 가져올 수 있습니다.

보안 인증 기반 정책

보안 인증 기반 정책은 IAM 사용자, 사용자 그룹 또는 역할과 같은 보안 인증에 연결할 수 있는 JSON 권한 정책 설명서입니다. 이러한 정책은 사용자와 역할이 어떤 리소스와 어떤 조건에서 어떤 태스크를 수행할 수 있는지를 통제합니다. 자격 증명 기반 정책을 생성하는 방법을 알아보려면 IAM 사용 설명서의 IAM 정책 생성을 참조하십시오.

보안 인증 기반 정책은 인라인 정책 또는 관리형 정책으로 한층 더 분류할 수 있습니다. 인라인 정책은 단일 사용자, 그룹 또는 역할에 직접 포함됩니다. 관리형 정책은 AWS 계정에 속한 다수의 사용자, 그룹 및 역할에 독립적으로 추가할 수 있는 정책입니다. 관리형 정책에는 AWS관리형 정책과 고객 관리형 정책이 포함되어 있습니다. 관리형 정책 또는 인라인 정책을 선택하는 방법을 알아보려면 IAM 사용 설명서의 관리형 정책과 인라인 정책의 선택을 참조하십시오.

리소스 기반 정책

리소스 기반 정책은 리소스에 연결하는 JSON 정책 설명서입니다. 리소스 기반 정책의 예는 IAM 역할 신뢰 정책과 Amazon S3 버킷 정책입니다. 리소스 기반 정책을 지원하는 서비스에서 서비스 관리자는 이러한 정책을 사용하여 특정 리소스에 대한 액세스를 통제할 수 있습니다. 정책이 연결된 리소스의 경우 정책은 지정된 보안 주체가 해당 리소스와 어떤 조건에서 어떤 작업을 수행할 수 있는지를 정의합니다. 리소스 기반 정책에서 보안 주체를 지정해야 합니다. 보안 주체에는 계정, 사용자, 역할, 페더레이션 사용자 또는 AWS 서비스가 포함될 수 있습니다.

리소스 기반 정책은 해당 서비스에 있는 인라인 정책입니다. 리소스 기반 정책에서는 IAM의 AWS관리형 정책을 사용할 수 없습니다.

ACL(액세스 통제 목록)

액세스 제어 목록(ACL)은 어떤 보안 주체(계정 멤버, 사용자 또는 역할)가 리소스에 액세스할 수 있는 권한을 가지고 있는지를 제어합니다. ACL은 JSON 정책 설명서 형식을 사용하지 않지만 리소스 기반 정책과 유사합니다.

Amazon S3, AWS WAF 및 Amazon VPC는 ACL을 지원하는 대표적인 서비스입니다. ACL에 대해 자세히 알아보려면 Amazon Simple Storage Service 개발자 안내서의 액세스 제어 목록(ACL) 개요를 참조하십시오.

기타 정책 유형

AWS은(는) 비교적 일반적이지 않은 추가 정책 유형을 지원합니다. 이러한 정책 유형은 더 일반적인 정책 유형에 따라 사용자에게 부여되는 최대 권한을 설정할 수 있습니다.

- 권한 경계 – 권한 경계는 보안 인증 기반 정책에 따라 IAM 엔터티(IAM 사용자 또는 역할)에 부여할 수 있는 최대 권한을 설정하는 고급 기능입니다. 개체에 대한 권한 경계를 설정할 수 있습니다. 그 결과로 얻는 권한은 개체의 보안 인증 기반 정책과 그 권한 경계의 교집합입니다. `Principal` 필드에서 사용자나 역할을 보안 주체로 지정하는 리소스 기반 정책은 권한 경계를 통해 제한되지 않습니다. 이러한 정책 중 하나에 포함된 명시적 거부는 허용을 재정의합니다. 권한 경계에 대한 자세한 정보는 IAM 사용 설명서의 IAM 엔터티에 대한 권한 경계를 참조하십시오.
- 서비스 통제 정책(SCP) – SCP는 AWS Organizations에서 조직 또는 조직 단위(OU)에 최대 권한을 지정하는 JSON 정책입니다. AWS Organizations는 기업이 소유하는 여러 개의 AWS 계정을 그룹화하고 중앙에서 관리하기 위한 서비스입니다. 조직에서 모든 기능을 활성화할 경우 서비스 통제 정책(SCP)을 임의의 또는 모든 계정에 적용할 수 있습니다. SCP는 각 AWS 계정 루트 사용자를 비롯하여 멤버 계정의 개체에 대한 권한을 제한합니다. 조직 및 SCP에 대한 자세한 정보는 AWS Organizations 사용 설명서의 SCP 작동 방식을 참조하십시오.
- 세션 정책 – 세션 정책은 역할 또는 페더레이션 사용자에 대해 임시 세션을 프로그래밍 방식으로 생성할 때 파라미터로 전달하는 고급 정책입니다. 결과적으로 얻는 세션의 권한은 사용자 또는 역할의 보안 인증 기반 정책의 교차와 세션 정책입니다. 또한 권한을 리소스 기반 정책에서 가져올 수도 있습니다. 이러한 정책 중 하나에 포함된 명시적 거부는 허용을 재정의합니다. 자세한 정보는 IAM 사용 설명서의 세션 정책을 참조하십시오.

여러 정책 유형

여러 정책 유형이 요청에 적용되는 경우, 결과 권한은 이해하기가 더 복잡합니다. 여러 정책 유형이 관련될 때 AWS가 요청을 허용할지 여부를 결정하는 방법을 알아보려면 IAM 사용 설명서의 정책 평가 로직을 참조하세요.

Amazon API Gateway에서 IAM을 사용하는 방식

IAM을 사용하여 API Gateway에 대한 액세스를 관리하기 전에 API Gateway에서 사용할 수 있는 IAM 기능을 이해해야 합니다. API Gateway 및 기타 AWS 서비스에서 IAM을 사용하는 방법을 전체적으로 알아보려면 IAM 사용 설명서의 IAM으로 작업하는 AWS 서비스를 참조하세요.

주제

- API Gateway 자격 증명 기반 정책
- API Gateway 리소스 기반 정책
- API Gateway 태그 기반 권한 부여
- API Gateway IAM 역할

API Gateway 자격 증명 기반 정책

IAM 자격 증명 기반 정책을 사용하면 허용되거나 거부되는 작업과 리소스뿐 아니라 작업이 허용되거나 거부되는 조건을 지정할 수 있습니다. API Gateway는 특정 작업, 리소스 및 조건 키를 지원합니다. API Gateway 관련 작업, 리소스 및 조건 키에 대한 자세한 내용은 Amazon API Gateway Management에 사용되는 작업, 리소스 및 조건 키 및 Amazon API Gateway Management V2에 사용되는 작업, 리소스 및 조건 키를 참조하세요. JSON 정책에서 사용하는 모든 요소에 대한 자세한 내용은 IAM 사용 설명서의 IAM JSON 정책 요소 참조를 참조하세요.

다음 예제에서는 사용자가 프라이빗 REST API만 생성하거나 업데이트할 수 있도록 허용하는 자격 증명 기반 정책을 보여 줍니다. 더 많은 예제는 the section called “자격 증명 기반 정책 예제”를 참조하세요.

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

작업

JSON 정책의 `Action` 요소는 정책에서 액세스를 허용하거나 거부하는 데 사용할 수 있는 작업을 설명합니다.

API Gateway의 정책 작업은 작업 앞에 `apigateway:` 접두사를 사용합니다. 정책 설명에는 `Action` 또는 `NotAction` 요소가 반드시 추가되어야 합니다. API Gateway는 이 서비스로 수행할 수 있는 작업을 설명하는 고유한 작업 세트를 정의합니다.

API 관리 `Action` 표현식은 `apigateway:`*`action`* 형식을 갖습니다. 여기서 *`action`*은 API Gateway 작업 GET, POST, PUT, DELETE, PATCH(리소스 업데이트용) 또는 *(모든 이전 작업) 중 하나입니다.

다음은 `Action` 표현식의 몇 가지 예입니다.

- **`apigateway:*`**: 모든 API Gateway 작업
- **`apigateway:GET`**: API Gateway의 GET 작업만

명령문 하나에 여러 작업을 지정하려면 다음과 같이 쉼표로 구분합니다.

```
"Action": [
      "apigateway:action1",
      "apigateway:action2"
```

특정 API Gateway 작업에 사용할 HTTP 동사에 대한 자세한 내용은 Amazon API Gateway 버전 1 API 참조(REST API) 및 Amazon API Gateway 버전 2 API 참조(WebSocket 및 HTTP API)를 참조하세요.

자세한 내용은 the section called “자격 증명 기반 정책 예제” 섹션을 참조하세요.

리소스

관리자는 AWS JSON 정책을 사용하여 누가 무엇에 액세스할 수 있는지를 지정할 수 있습니다. 즉, 어떤 보안 주체가 어떤 리소스와 어떤 조건에서 작업을 수행할 수 있는지 지정할 수 있습니다.

`Resource` JSON 정책 요소는 작업이 적용되는 하나 이상의 객체를 지정합니다. 명령문에는 `Resource` 또는 `NotResource` 요소가 반드시 추가되어야 합니다. 모범 사례에 따라 Amazon 리소스 이름(ARN)을 사용하여 리소스를 지정합니다. 리소스 수준 권한이라고 하는 특정 리소스 타입을 지원하는 작업에 대해 이 태스크를 수행할 수 있습니다.

작업 나열과 같이 리소스 수준 권한을 지원하지 않는 작업의 경우 와일드카드(*)를 사용하여 명령문이 모든 리소스에 적용됨을 나타냅니다.

```
"Resource": "*"
```

API Gateway 리소스에는 다음과 같은 ARN 형식이 있습니다.

```
arn:aws:apigateway:region::resource-path-specifier
```

예를 들어, ID *api-id*와(과) 하위 리소스(예: 명령문의 권한 부여자)가 있는 REST API를 지정하려면 다음 ARN을 사용합니다.

```
"Resource": "arn:aws:apigateway:us-east-2::/restapis/api-id/*"
```

특정 계정에 속하는 모든 REST API 및 하위 리소스를 지정하려면 와일드카드(*)를 사용합니다.

```
"Resource": "arn:aws:apigateway:us-east-2::/restapis/*"
```

API Gateway 리소스 유형 및 해당 ARN 목록은 API Gateway Amazon 리소스 이름(ARN) 참조 단원을 참조하세요.

조건 키

관리자는 AWS JSON 정책을 사용하여 누가 무엇에 액세스할 수 있는지를 지정할 수 있습니다. 즉, 어떤 보안 주체가 어떤 리소스와 어떤 조건에서 작업을 수행할 수 있는지 지정할 수 있습니다.

`Condition` 요소(또는 `Condition` 블록)를 사용하면 정책이 발효되는 조건을 지정할 수 있습니다. `Condition` 요소는 선택 사항입니다. 같음이나 미만 같은 조건 연산자를 사용하여 정책의 조건을 요청의 값과 일치시키는 조건식을 생성할 수 있습니다.

한 문에서 여러 `Condition` 요소를 지정하거나 단일 `Condition` 요소에서 여러 키를 지정하는 경우, AWS는 논리적 `AND` 태스크를 사용하여 평가합니다. 단일 조건 키의 여러 값을 지정하는 경우, AWS는 논리적 `OR` 태스크를 사용하여 조건을 평가합니다. 명령문의 권한을 부여하기 전에 모든 조건을 충족해야 합니다.

조건을 지정할 때 자리표시자 변수를 사용할 수도 있습니다. 예를 들어, IAM 사용자에게 IAM 사용자 이름으로 태그가 지정된 경우에만 리소스에 액세스할 수 있는 권한을 부여할 수 있습니다. 자세한 정보는 IAM 사용 설명서의 IAM 정책 요소: 변수 및 태그를 참조하십시오.

AWS은(는) 전역 조건 키와 서비스별 조건 키를 지원합니다. 모든 AWS 전역 조건 키를 보려면 IAM 사용 설명서의 AWS 전역 조건 컨텍스트 키를 참조하세요.

API Gateway는 자체 조건 키 세트를 정의하고 일부 전역 조건 키 사용도 지원합니다. API Gateway 조건 키 목록은 IAM 사용 설명서의 Amazon API Gateway의 조건 키를 참조하세요. 조건 키를 사용할 수 있는 작업과 리소스에 대한 내용은 Amazon API Gateway에서 정의한 작업을 참조하세요.

속성 기반 액세스 제어를 포함하여 태그 지정에 대한 자세한 내용은 태그 지정 섹션을 참조하세요.

예제

API Gateway 자격 증명 기반 정책의 예는 Amazon API Gateway 자격 증명 기반 정책 예제 단원을 참조하세요.

API Gateway 리소스 기반 정책

리소스 기반 정책은 지정된 보안 주체가 API Gateway 리소스에 대해 어떤 작업을 어떤 조건에서 수행할 수 있는지를 지정하는 JSON 정책 문서입니다. API Gateway는 REST API에 대한 리소스 기반 권한 정책을 지원합니다. 리소스 정책을 사용하여 REST API를 호출할 수 있는 사용자를 제어합니다. 자세한 내용은 the section called “API Gateway 리소스 정책 사용” 섹션을 참조하세요.

예제

API Gateway 리소스 기반 정책의 예는 API Gateway 리소스 정책 예제 단원을 참조하세요.

API Gateway 태그 기반 권한 부여

API Gateway 리소스에 태그를 연결하거나 요청에서 태그를 API Gateway에 전달할 수 있습니다. 태그를 기반으로 액세스를 제어하려면 `apigateway:ResourceTag/key-name`, `aws:RequestTag/key-name` 또는 `aws:TagKeys` 조건 키를 사용하여 정책의 조건 요소에 태그 정보를 제공합니다. API Gateway 리소스에 태그를 지정하는 방법에 대한 자세한 내용은 the section called “속성 기반 액세스 제어” 단원을 참조하십시오.

리소스의 태그를 기반으로 리소스에 대한 액세스를 제한하는 자격 증명 기반 정책의 예는 태그를 사용하여 API Gateway REST API 리소스에 대한 액세스 제어 단원을 참조하세요.

API Gateway IAM 역할

IAM 역할은 특정 권한을 가지고 있는 AWS 계정 내 개체입니다.

API Gateway에 임시 자격 증명 사용

임시 자격 증명을 사용하여 페더레이션을 통해 로그인하거나, IAM 역할을 맡거나, 교차 계정 역할을 맡을 수 있습니다. AssumeRole 또는 GetFederationToken 같은 AWS STS API 작업을 호출하여 임시 보안 자격 증명을 가져옵니다.

API Gateway는 임시 자격 증명 사용을 지원합니다.

서비스 연결 역할

서비스 연결 역할을 사용하면 AWS 서비스는 다른 서비스의 리소스에 액세스하여 사용자 대신 작업을 완료할 수 있습니다. 서비스 연결 역할은 IAM 계정에 나타나며 서비스가 소유합니다. IAM 관리자는 서비스 연결 역할의 권한을 볼 수 있지만 편집은 할 수 없습니다.

API Gateway는 서비스 연결 역할을 지원합니다. API Gateway 서비스 연결 역할을 생성하거나 관리하는 방법에 대한 자세한 내용은 API Gateway에 서비스 연결 역할 사용 단원을 참조하세요.

서비스 역할

서비스가 귀하를 대신해 서비스 역할을 맡을 수 있습니다. 서비스 역할을 사용하면 서비스는 다른 서비스의 리소스에 액세스해 사용자를 대신해 작업을 완료할 수 있습니다. 서비스 역할은 IAM 계정에 표시되며 계정에서 소유하므로 관리자가 이 역할에 대한 권한을 변경할 수 있습니다. 그러나 권한을 변경하면 서비스의 기능이 손상될 수 있습니다.

API Gateway는 서비스 역할을 지원합니다.

Amazon API Gateway 자격 증명 기반 정책 예제

기본적으로 IAM 사용자 및 역할은 API Gateway 리소스를 생성하거나 수정할 수 있는 권한이 없습니다. 또한 AWS Management Console, AWS CLI 또는 AWS SDK를 사용해 작업을 수행할 수 없습니다. IAM 관리자는 지정된 리소스에서 특정 API 작업을 수행할 수 있는 권한을 사용자와 역할에게 부여하는 IAM 정책을 생성해야 합니다. 그런 다음 관리자는 해당 권한이 필요한 IAM 사용자 또는 그룹에 이러한 정책을 연결해야 합니다.

IAM 정책을 생성하는 방법에 대한 자세한 내용은 IAM 사용 설명서의 JSON 탭에서 정책 생성을 참조하세요. API Gateway 관련 작업, 리소스 및 조건에 대한 자세한 내용은 Amazon API Gateway Management에 사용되는 작업, 리소스 및 조건 키 및 Amazon API Gateway Management V2에 사용되는 작업, 리소스 및 조건 키를 참조하세요.

주제

- 정책 모범 사례
- 사용자가 자신의 권한을 볼 수 있도록 허용
- 단순 읽기 권한
- REQUEST 또는 JWT 권한 부여자만 생성
- 기본 execute-api 엔드포인트가 비활성화되어 있어야 함
- 사용자가 프라이빗 REST API만 생성하거나 업데이트할 수 있도록 허용
- API 경로에 권한이 있어야 함
- 사용자가 VPC 링크를 생성하거나 업데이트하지 못하도록 방지

정책 모범 사례

ID 기반 정책에 따라 계정에서 사용자가 API Gateway 리소스를 생성, 액세스 또는 삭제할 수 있는지 여부가 결정됩니다. 이 작업으로 인해 AWS 계정에 비용이 발생할 수 있습니다. 자격 증명 기반 정책을 생성하거나 편집할 때는 다음 지침과 권장 사항을 따르십시오.

- AWS 관리형 정책으로 시작하고 최소 권한을 향해 나아가기 - 사용자 및 워크로드에 권한 부여를 시작하려면 많은 일반 사용 사례에 대한 권한을 부여하는 AWS 관리형 정책을 사용합니다. 관리형 정책은 AWS 계정에서 사용할 수 있습니다. 사용 사례에 고유한 AWS 고객 관리형 정책을 정의하여 권한을 줄이는 것이 좋습니다. 자세한 정보는 IAM 사용 설명서의 AWS관리형 정책 또는 AWS 직무에 대한 관리형 정책을 참조하십시오.

- 최소 권한 적용 – IAM 정책을 사용하여 권한을 설정하는 경우 태스크를 수행하는 데 필요한 권한만 부여합니다. 이렇게 하려면 최소 권한으로 알려진 특정 조건에서 특정 리소스에 대해 수행할 수 있는 작업을 정의합니다. IAM을 사용하여 권한을 적용하는 방법에 대한 자세한 정보는 IAM 사용 설명서에 있는 IAM의 정책 및 권한을 참조하십시오.
- IAM 정책의 조건을 사용하여 액세스 추가 제한 – 정책에 조건을 추가하여 작업 및 리소스에 대한 액세스를 제한할 수 있습니다. 예를 들어, SSL을 사용하여 모든 요청을 전송해야 한다고 지정하는 정책 조건을 작성할 수 있습니다. 특정 AWS 서비스(예: AWS CloudFormation)를 통해 사용되는 경우에만 서비스 작업에 대한 액세스 권한을 부여할 수도 있습니다. 자세한 정보는 IAM 사용 설명서의 IAM JSON 정책 요소: 조건을 참조하십시오.
- IAM Access Analyzer를 통해 IAM 정책을 검증하여 안전하고 기능적인 권한 보장 – IAM Access Analyzer에서는 IAM 정책 언어(JSON)와 모범 사례가 정책에서 준수되도록 신규 및 기존 정책을 검증합니다. IAM Access Analyzer는 100개 이상의 정책 확인 항목과 실행 가능한 권장 사항을 제공하여 안전하고 기능적인 정책을 작성하도록 돕습니다. 자세한 정보는 IAM 사용 설명서의 IAM Access Analyzer 정책 검증을 참조하십시오.
- 다중 인증(MFA) 필요 – AWS 계정 계정에 IAM 사용자 또는 루트 사용자가 필요한 시나리오가 있는 경우 추가 보안을 위해 MFA를 설정합니다. API 작업을 호출할 때 MFA가 필요하면 정책에 MFA 조건을 추가합니다. 자세한 정보는 IAM 사용 설명서의 MFA 보호 API 액세스 구성을 참조하십시오.

IAM의 모범 사례에 대한 자세한 내용은 IAM 사용 설명서의 IAM의 보안 모범 사례를 참조하십시오.

사용자가 자신의 권한을 볼 수 있도록 허용

이 예는 IAM 사용자가 자신의 사용자 보안 인증에 연결된 인라인 및 관리형 정책을 볼 수 있도록 허용하는 정책을 생성하는 방법을 보여줍니다. 이 정책에는 콘솔에서 또는 AWS CLI나 AWS API를 사용하여 프로그래밍 방식으로 이 작업을 완료할 수 있는 권한이 포함됩니다.

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

단순 읽기 권한

이 예제 정책은 us-east-1의 AWS 리전에서 a123456789의 식별자를 사용하여 HTTP 또는 WebSocket API의 모든 리소스에 대한 정보를 가져올 수 있는 권한을 사용자에게 부여합니다. 리소스 `arn:aws:apigateway:us-east-1::/apis/a123456789/*`에는 권한 부여자 및 배포와 같은 API의 모든 하위 리소스가 포함됩니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "apigateway:GET"
      ],
      "Resource": [
        "arn:aws:apigateway:us-east-1::/apis/a123456789/*"
      ]
    }
  ]
}
```

REQUEST 또는 JWT 권한 부여자만 생성

이 예제 정책은 사용자가 가져오기를 포함하여 REQUEST 또는 JWT 권한 부여자로만 API를 생성할 수 있도록 허용합니다. 정책의 Resource 섹션에서 arn:aws:apigateway:us-east-1::/apis/??????????에 따라 리소스에는 최대 10자(API의 하위 리소스 제외)를 포함해야 합니다. 이 예제에서는 사용자가 API를 가져와서 한 번에 여러 권한 부여자를 생성할 수 있기 때문에 ForAllValues 섹션에서 Condition을(를) 사용합니다.

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

기본 **execute-api** 엔드포인트가 비활성화되어 있어야 함

이 예제 정책은 사용자가 DisableExecuteApiEndpoint이(가) true인 요구 사항을 사용하여 API를 생성, 업데이트 또는 가져올 수 있도록 허용합니다. DisableExecuteApiEndpoint이(가) true인 경우 클라이언트는 API를 호출하는 데 기본 execute-api 엔드포인트를 사용할 수 없습니다.

BoolIfExists 조건을 사용하여 DisableExecuteApiEndpoint 조건 키가 채워지지 않은 API를 업데이트하는 호출을 처리합니다. 사용자가 API를 생성하거나 가져오려고 하면 DisableExecuteApiEndpoint 조건 키가 항상 채워집니다.

apis/* 리소스는 권한 부여자나 메서드와 같은 하위 리소스도 캡처하므로 Deny 명령문을 통해 API만 사용하도록 명시적으로 범위를 지정합니다.

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

사용자가 프라이빗 REST API만 생성하거나 업데이트할 수 있도록 허용

이 예제 정책은 조건 키를 사용하여 사용자가 PRIVATE API만 생성하도록 요구하고 API를 PRIVATE에서 다른 유형(예: REGIONAL)으로 변경할 수 있는 업데이트를 방지합니다.

ForAllValues는 API에 추가된 모든 EndpointType이 PRIVATE여야 한다고 요구하기 위해 사용합니다. PRIVATE인 경우에 한해 리소스 조건 키를 사용하여 API에 대한 업데이트를 허용합니다. ForAllValues는 조건 키가 있는 경우에만 적용됩니다.

비API 리소스(예: 권한 부여자) 허용을 방지하기 위해 최소 일치(?)를 사용하여 API ID에 대해 명시적으로 일치시킵니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ScopePutToPrivateApis",
            "Effect": "Allow",
            "Action": [
                "apigateway:PUT"
            ],
            "Resource": [
                "arn:aws:apigateway:us-east-1::/restapis",
                "arn:aws:apigateway:us-east-1::/restapis/??????????"
            ],
            "Condition": {
                "ForAllValues:StringEquals": {
                    "apigateway:Resource/EndpointType": "PRIVATE"
                }
            }
        },
        {
            "Sid": "ScopeToPrivateApis",
            "Effect": "Allow",
            "Action": [
                "apigateway:DELETE",
                "apigateway:PATCH",
                "apigateway:POST"
            ],
            "Resource": [
                "arn:aws:apigateway:us-east-1::/restapis",
                "arn:aws:apigateway:us-east-1::/restapis/??????????"
            ],
```

```
            "Condition": {
                "ForAllValues:StringEquals": {
                    "apigateway:Request/EndpointType": "PRIVATE",
                    "apigateway:Resource/EndpointType": "PRIVATE"
                }
            }
        },
        {
            "Sid": "AllowResourcePolicyUpdates",
            "Effect": "Allow",
            "Action": [
                "apigateway:UpdateRestApiPolicy"
            ],
            "Resource": [
                "arn:aws:apigateway:us-east-1::/restapis/*"
            ]
        }
    ]
}
```

API 경로에 권한이 있어야 함

이 정책은 경로에 권한이 없는 경우 경로를 생성하거나 업데이트하려는 시도(가져오기를 통한 시도 포함)가 실패하게 합니다. ForAnyValue은(는) 경로가 생성되거나 업데이트되지 않는 경우와 같이 키가 없으면 false로 평가됩니다. 가져오기를 통해 여러 경로를 생성할 수 있기 때문에 ForAnyValue을(를) 사용합니다.

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

사용자가 VPC 링크를 생성하거나 업데이트하지 못하도록 방지

이 정책은 사용자가 VPC 링크를 생성하거나 업데이트하는 것을 방지합니다. VPC 링크를 사용하면 Amazon VPC 내의 리소스를 VPC 외부의 클라이언트에 노출할 수 있습니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "DenyVPCLink",
      "Effect": "Deny",
      "Action": [
        "apigateway:POST",
        "apigateway:PUT",
        "apigateway:PATCH"
      ],
      "Resource": [
        "arn:aws:apigateway:us-east-1::/vpclinks",
        "arn:aws:apigateway:us-east-1::/vpclinks/*"
      ]
```

```
    }
  ]
}
```

Amazon API Gateway 리소스 기반 정책 예제

리소스 기반 정책 예제는 the section called “API Gateway 리소스 정책 예제” 단원을 참조하십시오.

Amazon API Gateway 자격 증명 및 액세스 문제 해결

다음 정보를 사용하여 API Gateway 및 IAM으로 작업할 때 발생할 수 있는 일반적인 문제를 진단하고 수정할 수 있습니다.

주제

- API Gateway에서 작업을 수행할 권한이 없음
- 저에게 iam:PassRole을 수행할 권한이 없습니다.
- AWS 계정 외부의 사용자가 API Gateway 리소스에 액세스할 수 있도록 허용하려고 함

API Gateway에서 작업을 수행할 권한이 없음

작업을 수행할 권한이 없다는 오류가 수신되면, 작업을 수행할 수 있도록 정책을 업데이트해야 합니다.

다음 예제 오류는 `mateojackson`IAM user(IAM 사용자)가 콘솔을 사용하여 가상 *`my-example-widget`* 리소스에 대한 세부 정보를 보려고 하지만 가상 `apigateway:`*`GetWidget`* 권한이 없을 때 발생합니다.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 apigateway:GetWidget on resource: my-example-widget
```

이 경우 `apigateway:`*`GetWidget`* 작업을 사용하여 *`my-example-widget`* 리소스에 액세스할 수 있도록 `mateojackson` 사용자 정책을 업데이트해야 합니다.

도움이 필요한 경우 AWS 관리자에게 문의하십시오. 관리자는 로그인 보안 인증을 제공한 사용자입니다.

저에게 iam:PassRole을 수행할 권한이 없습니다.

`iam:PassRole` 작업을 수행할 수 있는 권한이 없다는 오류가 수신되면 API Gateway에 역할을 전달할 수 있도록 정책을 업데이트해야 합니다.

일부 AWS 서비스에서는 새 서비스 역할 또는 서비스 연결 역할을 생성하는 대신, 해당 서비스에 기존 역할을 전달할 수 있습니다. 이렇게 하려면 사용자가 서비스에 역할을 전달할 수 있는 권한을 가지고 있어야 합니다.

다음 예제 오류는 `marymajor`라는 IAM 사용자가 콘솔을 사용하여 API Gateway에서 작업을 수행하려고 하는 경우에 발생합니다. 하지만 작업을 수행하려면 서비스 역할이 부여한 권한이 서비스에 있어야 합니다. Mary는 서비스에 역할을 전달할 수 있는 권한을 가지고 있지 않습니다.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

이 경우 Mary가 `iam:PassRole` 작업을 수행할 수 있도록 Mary의 정책을 업데이트해야 합니다.

도움이 필요한 경우 AWS 관리자에게 문의하십시오. 관리자는 로그인 보안 인증을 제공한 사용자입니다.

AWS 계정 외부의 사용자가 API Gateway 리소스에 액세스할 수 있도록 허용하려고 함

다른 계정의 사용자 또는 조직 외부의 사람이 리소스에 액세스하는 데 사용할 수 있는 역할을 생성할 수 있습니다. 역할을 수임할 신뢰할 수 있는 사람을 지정할 수 있습니다. 리소스 기반 정책 또는 액세스 제어 목록(ACL)을 지원하는 서비스의 경우 이러한 정책을 사용하여 다른 사람에게 리소스에 대한 액세스 권한을 부여할 수 있습니다.

자세히 알아보려면 다음을 참조하십시오.

- API Gateway가 이러한 기능을 지원하는지 여부를 알아보려면 Amazon API Gateway에서 IAM을 사용하는 방식 단원을 참조하세요.
- 소유하고 있는 AWS 계정의 리소스에 대한 액세스 권한을 제공하는 방법을 알아보려면 IAM 사용 설명서의 자신이 소유한 다른 AWS 계정의 IAM 사용자에 대한 액세스 권한 제공을 참조하세요.
- 리소스에 대한 액세스 권한을 타사 AWS 계정에게 제공하는 방법을 알아보려면 IAM 사용 설명서의 타사가 소유한 AWS 계정에 대한 액세스 제공을 참조하십시오.
- ID 페더레이션을 통해 액세스 권한을 제공하는 방법을 알아보려면 IAM 사용 설명서의 외부에서 인증된 사용자에게 액세스 권한 제공(ID 페더레이션)을 참조하십시오.
- 교차 계정 액세스를 위한 역할과 리소스 기반 정책 사용의 차이점을 알아보려면 IAM 사용 설명서의 IAM 역할과 리소스 기반 정책의 차이를 참조하십시오.

API Gateway에 서비스 연결 역할 사용

Amazon API Gateway는 AWS Identity and Access Management (IAM) 서비스 연결 역할을 사용합니다. 서비스 연결 역할은 API Gateway에 직접 연결된 고유한 유형의 IAM 역할입니다. 서비스 연결 역할은 API Gateway에서 사전 정의하며 서비스가 사용자를 대신하여 다른 AWS 서비스를 호출하는 데 필요한 모든 권한을 포함합니다.

서비스 연결 역할을 사용하면 필요한 권한을 수동으로 추가할 필요가 없으므로 API Gateway를 더 쉽게 설정할 수 있습니다. API Gateway에서 서비스 연결 역할의 권한을 정의하므로 다르게 정의되지 않은 한, API Gateway만 해당 역할을 수임할 수 있습니다. 정의된 권한에는 신뢰 정책과 권한 정책이 포함되며, 이 권한 정책은 다른 IAM 엔터티에 연결할 수 없습니다.

먼저 관련 리소스를 삭제한 후에만 서비스 연결 역할을 삭제할 수 있습니다. 이렇게 하면 리소스에 대한 액세스 권한을 실수로 삭제할 수 없기 때문에 API Gateway 리소스가 보호됩니다.

서비스 연결 역할을 지원하는 기타 서비스에 대한 자세한 내용은 AWS IAM과 작동하는 서비스를 참조하고 서비스 연결 역할(Service-Linked Role) 열에 예(Yes)가 있는 서비스를 찾아보세요. 해당 서비스에 대한 서비스 연결 역할 설명서를 보려면 예 링크를 선택합니다.

API Gateway에 대한 서비스 연결 역할 권한

API Gateway는 API Gateway가 사용자를 대신하여 Elastic Load Balancing, Amazon Data Firehose 및 기타 서비스 리소스에 액세스할 수 있도록 허용이라는 AWSServiceRoleForAPIGateway서비스 연결 역할을 사용합니다.

AWSServiceRoleForAPIGateway 서비스 연결 역할은 다음 서비스를 신뢰하여 역할을 수임합니다.

- `ops.apigateway.amazonaws.com`

역할 권한 정책은 API Gateway가 지정된 리소스에서 다음 작업을 완료하도록 허용합니다.

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

IAM 개체(사용자, 그룹, 역할 등)가 서비스 연결 역할을 작성하고 편집하거나 삭제할 수 있도록 권한을 구성할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 서비스 연결 역할 권한을 참조하세요.

API Gateway에 대한 서비스 연결 역할 생성

서비스 링크 역할은 수동으로 생성할 필요가 없습니다. , 또는 API에서 API, 사용자 지정 도메인 이름 또는 VPC 링크를 생성하면 AWS API Gateway에서 서비스 연결 역할을 자동으로 생성합니다. AWS Management Console AWS CLI

이 서비스 연결 역할을 삭제했다가 다시 생성해야 하는 경우 동일한 프로세스를 사용하여 계정에서 역할을 다시 생성할 수 있습니다. API, 사용자 지정 도메인 이름 또는 VPC 링크를 생성하면 API Gateway가 서비스 연결 역할을 자동으로 다시 생성합니다.

API Gateway에 대한 서비스 연결 역할 편집

API Gateway에서는 AWSServiceRoleForAPIGateway 서비스 연결 역할을 편집할 수 없습니다. 서비스 연결 역할을 생성한 후에는 다양한 엔터티가 역할을 참조할 수 있기 때문에 역할 이름을 변경할 수 없습니다. 하지만 IAM을 사용하여 역할의 설명을 편집할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 서비스 연결 역할 편집을 참조하세요.

API Gateway에 대한 서비스 연결 역할 삭제

서비스 연결 역할이 필요한 기능 또는 서비스가 더 이상 필요 없는 경우에는 해당 역할을 삭제할 것을 권합니다. 이렇게 하면 적극적으로 모니터링하거나 유지 관리하지 않은 미사용 엔터티가 없습니다. 단, 서비스 연결 역할에 대한 리소스를 먼저 정리해야 수동으로 삭제할 수 있습니다.

> **Note**
>
> 리소스를 삭제하려고 할 때 API Gateway 서비스가 역할을 사용 중이면 삭제에 실패할 수 있습니다. 이 문제가 발생하면 몇 분 기다렸다가 작업을 다시 시도하세요.

에서 사용하는 API Gateway 리소스를 삭제하려면 AWSServiceRoleForAPIGateway

1. https://console.aws.amazon.com/apigateway/에서 Amazon API Gateway 콘솔을 엽니다.
2. 서비스 연결 역할을 사용하는 API, 사용자 지정 도메인 이름 또는 VPC 링크로 이동합니다.
3. 콘솔을 사용하여 리소스를 삭제합니다.
4. 이 절차를 반복하여 서비스 연결 역할을 사용하는 모든 API, 사용자 지정 도메인 이름 또는 VPC 링크를 삭제합니다.

IAM을 사용하여 수동으로 서비스 링크 역할을 삭제하려면

IAM 콘솔 AWS CLI, 또는 AWS API를 사용하여 AWSServiceRoleForAPIGateway 서비스 연결 역할을 삭제합니다. 자세한 내용은 IAM 사용 설명서의 서비스 연결 역할 삭제를 참조하십시오.

API Gateway 서비스 연결 역할이 지원되는 리전

API Gateway는 서비스가 제공되는 모든 리전에서 서비스 연결 역할을 사용할 수 있도록 지원합니다. 자세한 내용은 AWS 서비스 엔드포인트를 참조하십시오.

API Gateway에서 AWS 관리형 정책을 업데이트합니다.

이 서비스가 이러한 변경 사항을 추적하기 시작한 이후 API Gateway의 AWS 관리형 정책 업데이트에 대한 세부 정보를 확인하십시오. 이 페이지의 변경 사항에 대한 자동 알림을 받아보려면 API Gateway 문서 기록 페이지에서 RSS 피드를 구독하세요.

변경 사항	설명	날짜
`AWSServiceRoleForAPIGateway` 정책에 `acm:GetCertificate` 지원이 추가되었습니다.	이 `AWSServiceRoleForAPIGateway` 정책이 ACM `GetCertificate` API 작업 호출 권한을 포함합니다.	2021년 7월 12일
API Gateway 변경 내용 추적 시작	API Gateway는 AWS 관리형 정책의 변경 사항을 추적하기 시작했습니다.	2021년 7월 12일

Amazon API Gateway에서 로깅 및 모니터링

모니터링은 API Gateway 및 AWS 솔루션의 안정성, 가용성 및 성능을 유지하는 데 있어 중요한 부분입니다. 다중 지점 장애가 발생할 경우 이를 보다 쉽게 디버깅할 수 있도록 AWS 솔루션의 모든 부분에서 모니터링 데이터를 수집해야 합니다. AWS API Gateway 리소스를 모니터링하고 잠재적 사고에 대응하기 위한 몇 가지 도구를 제공합니다.

아마존 CloudWatch 로그

요청 실행 또는 API에 대한 클라이언트 액세스와 관련된 문제를 디버깅하는 데 도움이 되도록 CloudWatch 로그를 활성화하여 API 호출을 기록할 수 있습니다. 자세한 설명은 the section called “CloudWatch 로그” 섹션을 참조하세요.

아마존 CloudWatch 알람

CloudWatch 경보를 사용하면 지정한 기간 동안 단일 지표를 관찰할 수 있습니다. 지표가 지정된 임계값을 초과하는 경우 Amazon Simple Notification Service 주제 또는 AWS Auto Scaling 정책으로 알림이 전송됩니다. CloudWatch 지표가 특정 상태에 있을 때는 경보가 작업을 호출하지 않습니다. 대신, 상태가 변경되어 지정된 기간 동안 유지되어야 합니다. 자세한 설명은 the section called “CloudWatch 지표” 섹션을 참조하세요.

Firehose에 대한 액세스 로깅

API에 대한 클라이언트 액세스와 관련된 문제를 디버깅하기 위해 Firehose가 API 호출을 기록하도록 설정할 수 있습니다. 자세한 설명은 the section called “Firehose” 섹션을 참조하세요.

AWS CloudTrail

CloudTrail API Gateway에서 사용자, 역할 또는 AWS 서비스가 수행한 작업의 기록을 제공합니다. 에서 수집한 CloudTrail 정보를 사용하여 API Gateway에 이루어진 요청, 요청이 이루어진 IP 주소, 요청한 사람, 요청 시기 및 추가 세부 정보를 확인할 수 있습니다. 자세한 설명은 the section called “AWS CloudTrail 작업” 섹션을 참조하세요.

AWS X-Ray

X-Ray는 애플리케이션이 AWS 처리하는 요청에 대한 데이터를 수집하고 이를 사용하여 애플리케이션 관련 문제와 최적화 기회를 식별하는 데 사용할 수 있는 서비스 맵을 구성하는 서비스입니다. 자세한 설명은 the section called “ 설정AWS X-Ray” 섹션을 참조하세요.

AWS Config

AWS Config 계정의 AWS 리소스 구성을 자세히 볼 수 있습니다. 리소스 간에 어떤 관계가 있는지 파악하고, 구성 변경 이력을 확인하고, 시간이 지나면서 구성과 관계가 어떻게 변하는지 확인할 수 있습니다. 를 AWS Config 사용하여 리소스 구성의 데이터 규정 준수를 평가하는 규칙을 정의할 수 있습니다. AWS Config 규칙은 API Gateway 리소스에 대한 이상적인 구성 설정을 나타냅니다. 리소스가 규칙을 위반하여 규정을 준수하지 않는 것으로 플래그가 지정된 경우 Amazon Simple Notification Service (Amazon SNS) 주제를 사용하여 AWS Config 경고할 수 있습니다. 자세한 내용은 the section called “AWS Config 작업” 단원을 참조하십시오.

AWS CloudTrail을 사용하여 Amazon API Gateway API에 대한 호출 로깅

Amazon API Gateway는 사용자, 역할, AWS 서비스가 API Gateway에서 수행한 작업의 레코드를 제공하는 서비스, AWS CloudTrail와 통합됩니다. CloudTrail은 API Gateway 콘솔의 호출 및 API Gateway 서비스 API에 대한 코드 호출을 포함하여 API Gateway 서비스 API에 대한 모든 REST API 호출을 이벤트로 캡처합니다.

> **Note**
>
> TestInvokeAuthorizer 및 TestInvokeMethod는 CloudTrail에서 로깅되지 않습니다.

추적을 생성하면, API Gateway 이벤트를 포함한 CloudTrail 이벤트를 지속적으로 Amazon S3 버킷에 배포할 수 있습니다. 추적을 구성하지 않은 경우에도 CloudTrail 콘솔의 Event history(이벤트 기록)에서 최신 이벤트를 볼 수 있습니다.

CloudTrail에서 수집한 정보를 사용하여 API Gateway에 수행된 요청, 요청이 수행된 IP 주소, 요청을 수행한 사람, 요청이 수행된 시간 및 추가 세부 정보를 확인할 수 있습니다.

CloudTrail에 대한 자세한 내용은 AWS CloudTrail 사용 설명서를 참조하세요.

CloudTrail의 API Gateway 정보

CloudTrail은 계정 생성 시 AWS 계정에서 사용되도록 설정됩니다. Amazon API Gateway에서 활동이 발생하면, 해당 활동이 이벤트 기록(Event history)에 다른 AWS 서비스 이벤트와 함께 CloudTrail 이벤트에 기록됩니다. AWS 계정에서 최신 이벤트를 확인, 검색 및 다운로드할 수 있습니다. 자세한 내용은 CloudTrail 이벤트 기록을 사용하여 이벤트 보기를 참조하십시오.

API Gateway의 이벤트를 포함하여 AWS 계정에 이벤트를 지속적으로 기록하려면 추적을 생성합니다. CloudTrail은 추적을 사용하여 Amazon S3 버킷으로 로그 파일을 전송할 수 있습니다. 콘솔에서 추적을 생성하면 기본적으로 모든 AWS 리전에 추적이 적용됩니다. 추적은 AWS 파티션에 있는 모든 리전의 이벤트를 로깅하고 지정된 Amazon S3 버킷으로 로그 파일을 전송합니다. 또는 CloudTrail 로그에서 수집된 이벤트 데이터를 추가 분석 및 처리하도록 다른 AWS 서비스를 구성할 수 있습니다. 자세한 내용은 다음을 참조하세요.

- 추적 생성 개요
- CloudTrail 지원 서비스 및 통합
- CloudTrail에 대한 Amazon SNS 알림 구성
- 여러 리전에서 CloudTrail 로그 파일 받기 및 여러 계정에서 CloudTrail 로그 파일 받기

모든 Amazon API Gateway 작업은 CloudTrail에서 로깅되며 에서 문서화됩니다API 참조 예를 들어, API Gateway에서 새 API, 리소스 또는 메서드를 생성하도록 호출하면 CloudTrail 로그 파일에 항목이 생성됩니다.

모든 이벤트 및 로그 항목에는 요청을 생성한 사용자에 대한 정보가 들어 있습니다. 자격 증명 정보를 이용하면 다음을 쉽게 판단할 수 있습니다.

- 요청을 루트로 했는지 아니면 IAM 사용자 자격 증명으로 했는지 여부.
- 역할 또는 연합된 사용자에 대한 임시 보안 자격 증명을 사용하여 요청이 생성되었는지 여부.
- 다른 AWS 서비스에서 요청했는지 여부.

자세한 내용은 CloudTrail userIdentity 요소를 참조하십시오.

API Gateway 로그 파일 항목 이해

추적이란 지정한 Amazon S3 버킷에 이벤트를 로그 파일로 입력할 수 있게 하는 구성입니다. CloudTrail 로그 파일에는 하나 이상의 로그 항목이 포함될 수 있습니다. 이벤트는 모든 소스의 단일 요청을 나타내며 요청된 작업, 작업 날짜와 시간, 요청 파라미터 등에 대한 정보를 포함하고 있습니다. CloudTrail 로그 파일은 퍼블릭 API 호출에 대한 순서 지정된 스택 추적이 아니기 때문에 특정 순서로 표시되지 않습니다.

다음 예에 API Gateway `GetResource` 작업을 보여 주는 CloudTrail 로그 항목이 나와 있습니다.

```
{
    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 ...
    ]
}
```

로 API Gateway API 구성 모니터링AWS Config

AWS Config를 사용하여 API Gateway API 리소스에 대한 구성 변경을 기록하고, 리소스 변경을 기반으로 알림을 보낼 수 있습니다. API Gateway 리소스에 대한 구성 변경 기록을 유지 관리하면 운영 문제 해결, 감사 및 규정 준수 사용 사례에 도움이 됩니다.

AWS Config는 다음에 대한 변경을 추적할 수 있습니다.

- 다음과 같은 API 단계 구성
  - 캐시 클러스터 설정
  - 스로틀 설정
  - 액세스 로그 설정
  - 단계의 활성 배포 세트
- 다음과 같은 API 구성
  - 엔드포인트 구성
  - version
  - protocol
  - tags

또한 AWS Config 규칙 기능을 사용하여 구성 규칙을 정의하고, 이러한 규칙에 대한 위반을 자동으로 감지 및 추적하고 알릴 수 있습니다. 이러한 리소스 구성 속성의 변경 사항을 추적하면 API Gateway 리소스에 대해 변경으로 트리거되는 AWS Config 규칙을 작성하고, 모범 사례를 기준으로 리소스 구성을 테스트할 수도 있습니다.

계정에서 AWS Config 콘솔 또는 AWS Config를 사용하여 AWS CLI를 활성화할 수 있습니다. 그런 다음 변경 사항을 추적하려는 리소스 유형을 선택합니다. 이전에 구성한 AWS Config가 모든 리소스 유형을 기록하는 경우 이러한 API Gateway 리소스가 계정에 자동으로 기록됩니다. AWS Config의 Amazon API Gateway에 대한 지원은 모든 AWS 퍼블릭 리전 및 AWS GovCloud (US)에서 사용할 수 있습니다. 지원되는 리전의 전체 목록은 AWS 일반 참조에서 Amazon API Gateway 엔드포인트 및 할당량을 참조하세요.

주제

- 지원되는 리소스 유형
- 설정AWS Config
- API Gateway 리소스를 기록하도록 AWS Config 구성
- AWS Config 콘솔에서 API Gateway 구성 세부 정보 보기
- AWS Config 규칙을 사용하여 API Gateway 리소스 평가

지원되는 리소스 유형

다음 API Gateway 리소스 유형은 AWS Config와 통합되며, AWS Config 지원 AWS 리소스 유형과 리소스 관계에서 설명합니다.

- `AWS::ApiGatewayV2::Api`(WebSocket 및 HTTP API)
- `AWS::ApiGateway::RestApi`(REST API)
- `AWS::ApiGatewayV2::Stage`(WebSocket 및 HTTP API 단계)
- `AWS::ApiGateway::Stage`(REST API 단계)

AWS Config에 대한 자세한 내용은 AWS Config개발자 안내서 단원을 참조하십시오. 요금에 대한 자세한 내용은 AWS Config 요금 정보 페이지를 참조하십시오.

> **Important**
>
> API를 배포한 후 다음과 같은 API 속성을 변경한 경우, 반드시 API를 재배포하여 변경을 전파해야 합니다. 이렇게 하지 않으면 AWS Config 콘솔에서는 속성 변경이 표시되지만, 이전 속성 설정이 여전히 유효하며 API의 런타임 동작이 변경되지 않습니다.
>
> - **`AWS::ApiGateway::RestApi`** – `binaryMediaTypes`, `minimumCompressionSize`, `apiKeySource`

- **AWS::ApiGatewayV2::Api** – apiKeySelectionExpression

설정AWS Config

처음 AWS Config를 설정한다면 AWS Config 개발자 안내서의 다음 주제를 참조하세요.

- 콘솔을 사용하여 AWS Config 설정
- AWS CLI를 사용하여 AWS Config 설정

API Gateway 리소스를 기록하도록 AWS Config 구성

기본적으로 AWS Config는 해당 환경이 실행 중인 리전에서 검색하는 지원되는 모든 유형의 리전 리소스에 대한 구성 변경을 기록합니다. 특정 리소스 유형만의 변경 또는 글로벌 리소스 변경을 기록하도록 AWS Config를 사용자 지정할 수 있습니다.

리전 리소스와 글로벌 리소스를 확인하고 AWS Config 구성을 사용자 지정하는 방법을 알아보려면 AWS Config가 기록할 리소스 선택을 참조하십시오.

AWS Config 콘솔에서 API Gateway 구성 세부 정보 보기

AWS Config 콘솔을 사용해 API Gateway 리소스를 찾고, 현재 및 과거 구성에 대한 세부 정보를 얻을 수 있습니다. 다음 절차는 API Gateway API에 대한 정보를 찾는 방법을 보여 줍니다.

AWS config 콘솔에서 API Gateway 리소스 찾기

1. AWS Config 콘솔을 엽니다.
2. 리소스를 선택합니다.
3. 리소스 인벤토리 페이지에서 리소스를 선택합니다.
4. 리소스 유형(Resource type) 메뉴를 열고 APIGateway 또는 APIGatewayV2로 스크롤 한 후 API Gateway 리소스 유형 중 하나 이상을 선택합니다.
5. Look up(조회)을 선택합니다.
6. AWS Config가 표시할 리소스 목록의 리소스 ID를 선택합니다. AWS Config는 선택한 리소스에 대한 구성의 세부 정보와 기타 정보를 표시합니다.
7. 기록이 된 구성의 세부 정보를 완전히 확인하려면 세부 정보 보기를 선택합니다.

이 페이지의 리소스 및 보기 정보를 찾는 방법에 대한 자세한 내용은 AWS Config 개발자 안내서의 AWS 리소스 구성 및 기록 보기를 참조하세요.

AWS Config 규칙을 사용하여 API Gateway 리소스 평가

API Gateway 리소스에 대한 이상적인 구성 설정을 나타내는 AWS Config 규칙을 생성할 수 있습니다. 미리 정의된 AWS Config 관리형 규칙을 사용하거나, 사용자 지정 규칙을 정의할 수 있습니다. AWS Config는 사용자 리소스의 구성을 계속 추적해 변경 사항이 사용자 규칙의 조건을 위반하는지 여부를 결정합니다. AWS Config 콘솔에는 규칙과 리소스의 준수 상태가 표시됩니다.

리소스가 규칙을 위반하고 규정 위반으로 플래그가 지정된 경우 AWS Config에서는 Amazon Simple Notification Service 개발자 안내서(Amazon SNS) 주제를 사용하여 알림을 제공할 수 있습니다. 이러한 AWS Config 알림의 데이터를 프로그래밍 방식으로 사용하려면 Amazon Simple Queue Service(Amazon SQS) 대기열을 Amazon SNS 주제의 알림 엔드포인트로 사용합니다.

규칙 설정 및 사용에 대한 자세한 내용은 AWS Config 개발자 안내서의 규칙으로 리소스 평가를 참조하세요.

Amazon API Gateway에 대한 규정 준수 확인

AWS 서비스가 특정 규정 준수 프로그램의 범위에 포함되는지 알아보려면 규정 준수 프로그램 제공 범위 내 AWS 서비스를 참조하고 관심 있는 규정 준수 프로그램을 선택하십시오. 일반적인 정보는 AWS 규정 준수 프로그램을 참조하십시오.

AWS Artifact를 사용하여 제3자 감사 보고서를 다운로드할 수 있습니다. 자세한 설명은 AWS Artifact에서 보고서 다운로드를 참조하세요.

AWS 서비스 사용 시 규정 준수 책임은 데이터의 민감도, 회사의 규정 준수 목표 및 관련 법률과 규정에 따라 결정됩니다. AWS에서는 규정 준수를 지원할 다음과 같은 리소스를 제공합니다.

- 보안 및 규정 준수 빠른 시작 가이드 – 이 배포 가이드에서는 아키텍처 고려 사항에 대해 설명하고 보안 및 규정 준수에 중점을 둔 기본 AWS 환경을 배포하기 위한 단계를 제공합니다.
- Architecting for HIPAA Security and Compliance on Amazon Web Services(Amazon Web Services에서 HIPAA 보안 및 규정 준수 기술 백서 설계) - 이 백서는 기업에서 AWS을(를) 사용하여 HIPAA를 준수하는 애플리케이션을 만드는 방법을 설명합니다.

> **Note**
>
> 모든 AWS 서비스에 HIPAA 자격이 있는 것은 아닙니다. 자세한 설명은 HIPAA 적격 서비스 참조를 참조하세요.

- AWS 규정 준수 리소스 - 고객 조직이 속한 산업 및 위치에 적용될 수 있는 워크북 및 가이드 콜렉션입니다.
- AWS 고객 규정 준수 가이드 - 규정 준수의 관점에서 공동 책임 모델을 이해합니다. 이 가이드에서는 AWS 서비스를 보호하기 위한 모범 사례를 요약하고 여러 프레임워크(미국 표준 기술 연구소(NIST), 결제 카드 산업 보안 표준 위원회(PCI), 국제 표준화기구(ISO) 등)에서 보안 제어에 대한 지침을 매핑합니다.
- AWS Config 개발자 가이드의 규칙을 사용하여 리소스 평가 – AWS Config 서비스는 내부 사례, 산업 지침 및 규제에 대한 리소스 구성의 준수 상태를 평가합니다.
- AWS Security Hub – 이 AWS 서비스은(는) AWS 내의 보안 상태에 대한 포괄적인 보기를 제공합니다. Security Hub는 보안 통제를 사용하여 AWS 리소스를 평가하고 보안 업계 표준 및 모범 사례에 대한 규정 준수를 확인합니다. 지원되는 서비스 및 통제 목록은 Security Hub 통제 참조를 참조하세요.
- AWS Audit Manager - 이 AWS 서비스은(는) AWS 사용을 지속적으로 감사하여 리스크를 관리하고 규정 및 업계 표준을 준수하는 방법을 간소화할 수 있도록 지원합니다.

Amazon API Gateway의 복원성

AWS 글로벌 인프라는 AWS 리전 및 가용 영역을 중심으로 구축됩니다. AWS 리전은 물리적으로 분리되고 격리된 다수의 가용 영역을 제공하며 이러한 가용 영역은 짧은 지연 시간, 높은 처리량 및 높은 중복성을 갖춘 네트워크에 연결되어 있습니다. 가용 영역을 사용하면 중단 없이 영역 간에 자동으로 장애 조치가 이루어지는 애플리케이션 및 데이터베이스를 설계하고 운영할 수 있습니다. 가용 영역은 기존의 단일 또는 다중 데이터 센터 인프라보다 가용성, 내결함성, 확장성이 뛰어납니다.

AWS 리전 및 가용 영역에 대한 자세한 내용은 AWS 글로벌 인프라를 참조하세요.

API가 너무 많은 요청에 의해 압도되는 것을 방지하기 위해 API에 대한 API Gateway 요청을 조절합니다. 특히 API Gateway에서는 계정의 모든 API에 대한 요청 제출 버스트 및 안정적인 상태의 속도에 관한 한도를 설정합니다. API에 대한 사용자 지정 조절을 구성할 수 있습니다. 자세한 내용은 처리량 향상을 위해 API 요청 조절 단원을 참조하세요.

Route 53 상태 확인을 사용하여 기본 리전의 API Gateway API에서 보조 리전의 API Gateway API로의 DNS 장애 조치를 제어할 수 있습니다. 관련 예제는 the section called “DNS 장애 조치” 섹션을 참조하세요

Amazon API Gateway의 인프라 보안

관리형 서비스인 Amazon API Gateway는 AWS 글로벌 네트워크 보안으로 보호됩니다. AWS 보안 서비스와 AWS의 인프라 보호 방법에 대한 자세한 내용은 AWS 클라우드 보안을 참조하세요. 인프라 보안에 대한 모범 사례를 사용하여 AWS 환경을 설계하려면 보안 원칙 AWS Well-Architected Framework의 인프라 보호를 참조하세요.

AWS에서 게시한 API 호출을 사용하여 네트워크를 통해 API Gateway에 액세스합니다. 고객은 다음을 지원해야 합니다.

- 전송 계층 보안(TLS). TLS 1.2는 필수이며 TLS 1.3을 권장합니다.
- DHE(Ephemeral Diffie-Hellman) 또는 ECDHE(Elliptic Curve Ephemeral Diffie-Hellman)와 같은 완전 전송 보안(PFS)이 포함된 암호 제품군. Java 7 이상의 최신 시스템은 대부분 이러한 모드를 지원합니다.

또한 요청은 액세스 키 ID 및 IAM 주체와 관련된 보안 액세스 키를 사용하여 서명해야 합니다. 또는 AWS Security Token Service(AWS STS)를 사용하여 임시 보안 자격 증명을 생성하여 요청에 서명할 수 있습니다.

이러한 API 작업은 어떤 네트워크 위치에서든 호출할 수 있지만, API Gateway는 소스 IP 주소에 따른 제한 사항을 포함할 수 있는 리소스 기반 액세스 정책을 지원합니다. 리소스 기반 정책을 사용하여 특정 Amazon Virtual Private Cloud(Amazon VPC) 엔드포인트 또는 특정 VPC에서 액세스를 제어할 수도 있습니다. 그러면 AWS 네트워크의 특정 VPC에서만 특정 API Gateway 리소스에 대한 네트워크 액세스가 효과적으로 격리됩니다.

Amazon API Gateway의 취약성 분석

구성 및 IT 제어는 AWS와 고객 간의 공동 책임입니다. 자세한 내용은 AWS 공동 책임 모델을 참조하세요.

Amazon API Gateway의 보안 모범 사례

API Gateway는 자체 보안 정책을 개발하고 구현할 때 고려해야 할 여러 보안 기능을 제공합니다. 다음 모범 사례는 일반적인 지침이며 완벽한 보안 솔루션을 나타내지는 않습니다. 이러한 모범 사례는 환경에 적절하지 않거나 충분하지 않을 수 있으므로 참고용으로만 사용해 주세요.

최소 권한 액세스 구현

IAM 정책을 사용하여 API Gateway API 생성, 읽기, 업데이트 또는 삭제에 대한 최소 권한 액세스를 구현합니다. 자세한 내용은 Amazon API Gateway의 Identity and Access Management 단원을 참조하세요. API Gateway에서는 생성한 API에 대한 액세스를 제어하는 몇 가지 옵션을 제공합니다. 자세한 내용은 API Gateway의 REST API에 대한 액세스 제어 및 관리, API Gateway에서 WebSocket API에 대한 액세스 제어 및 관리 및 JWT 권한 부여자를 사용하여 HTTP API에 대한 액세스 제어 단원을 참조하세요.

로깅 구현

CloudWatch 로그 또는 Amazon Data Firehose를 사용하여 API에 대한 요청을 기록할 수 있습니다. 자세한 내용은 REST API 모니터링, WebSocket API의 로깅 구성 및 HTTP API의 로깅 구성 단원을 참조하세요.

아마존 CloudWatch 알람 구현

CloudWatch 경보를 사용하면 지정한 기간 동안 단일 지표를 관찰할 수 있습니다. 지표가 지정된 임계값을 초과하는 경우 Amazon Simple Notification Service 주제 또는 AWS Auto Scaling 정책으로 알림이 전송됩니다. CloudWatch 지표가 특정 상태에 있을 때는 경보가 작업을 호출하지 않습니다. 대신, 상태가 변경되어 지정된 기간 동안 유지되어야 합니다. 자세한 설명은 the section called “CloudWatch 지표” 섹션을 참조하세요.

활성화합니다. AWS CloudTrail

CloudTrail API Gateway에서 사용자, 역할 또는 AWS 서비스가 수행한 작업의 기록을 제공합니다. 에서 수집한 CloudTrail 정보를 사용하여 API Gateway에 이루어진 요청, 요청이 이루어진 IP 주소, 요청한 사람, 요청 시기 및 추가 세부 정보를 확인할 수 있습니다. 자세한 설명은 the section called “AWS CloudTrail 작업” 섹션을 참조하세요.

활성화 AWS Config

AWS Config 계정의 AWS 리소스 구성을 자세히 볼 수 있습니다. 리소스 간에 어떤 관계가 있는지 파악하고, 구성 변경 이력을 확인하고, 시간이 지나면서 구성과 관계가 어떻게 변하는지 확인할 수 있습니다. 를 AWS Config 사용하여 리소스 구성의 데이터 규정 준수를 평가하는 규칙을 정의할 수 있습니다. AWS Config 규칙은 API Gateway 리소스에 대한 이상적인 구성 설정을 나타냅니다.

리소스가 규칙을 위반하여 규정을 준수하지 않는 것으로 플래그가 지정된 경우 Amazon Simple Notification Service (Amazon SNS) 주제를 사용하여 AWS Config 경고할 수 있습니다. 자세한 내용은 the section called “AWS Config 작업” 섹션을 참조하세요.

사용: AWS Security Hub

AWS Security Hub을 사용하여 보안 모범 사례와 관련된 API Gateway의 사용량을 모니터링하세요. Security Hub는 보안 제어를 사용하여 리소스 구성 및 보안 표준을 평가하여 다양한 규정 준수 프레임워크를 준수할 수 있도록 지원합니다. Security Hub를 사용하여 Lambda 리소스를 평가하는 방법에 대한 자세한 내용은 AWS Security Hub 사용자 설명서의 Amazon API Gateway 제어를 참조하세요.

API Gateway 리소스 태그 지정

태그는 사용자 또는 AWS가 AWS 리소스에 할당하는 메타데이터 레이블입니다. 각 태그에는 다음 두 가지 부분이 있습니다.

- 태그 키 (예: `CostCenter`, `Environment` 또는 `Project`) 태그 키는 대/소문자를 구별합니다.
- 태그 값(예: `111122223333` 또는 `Production`)으로 알려진 선택적 필드 태그 값을 생략하는 것은 빈 문자열을 사용하는 것과 같습니다. 태그 키처럼 태그 값은 대/소문자를 구분합니다.

태그는 다음을 지원합니다.

- 할당된 태그를 기반으로 리소스에 대한 액세스를 제어합니다. AWS Identity and Access Management(IAM) 정책 조건에서 태그 키와 값을 지정하여 액세스를 제어합니다. 태그 기반 액세스 제어에 대한 자세한 내용은 IAM 사용 설명서의 태그를 사용한 액세스 제어를 참조하십시오.
- AWS 비용을 추적합니다. AWS Billing and Cost Management 대시보드에서 이러한 태그를 활성화합니다. AWS는 태그를 사용하여 비용을 분류하고 월별 비용 할당 보고서를 전달합니다. 자세한 내용은 AWS Billing 사용 설명서의 비용 할당 태그 사용을 참조하세요.
- AWS 리소스를 식별하고 정리합니다. 많은 AWS 서비스가 태그 지정을 지원하므로 다른 서비스의 리소스에 동일한 태그를 할당하여 해당 리소스의 관련 여부를 나타낼 수 있습니다. 예를 들어, API Gateway 단계에 CloudWatch Events 규칙에 할당한 동일한 태그를 할당할 수 있습니다.

태그 사용에 대한 팁은 AWS Answers 블로그의 게시글 AWS Tagging Strategies를 참조하세요.

다음 단원에서는 Amazon API Gateway의 태그에 대한 추가 정보를 제공합니다.

주제

- 태그를 지정할 수 있는 API Gateway 리소스
- 태그를 사용하여 API Gateway REST API 리소스에 대한 액세스 제어

태그를 지정할 수 있는 API Gateway 리소스

태그는 Amazon API Gateway V2 API의 다음과 같은 HTTP API 또는 WebSocket API 리소스에 설정될 수 있습니다.

- `Api`

- DomainName
- Stage
- VpcLink

또한 태그는 Amazon API Gateway V1 API의 다음과 같은 REST API 리소스에 설정될 수 있습니다.

- ApiKey
- ClientCertificate
- DomainName
- RestApi
- Stage
- UsagePlan
- VpcLink

다른 리소스에는 태그를 직접 설정할 수 없습니다. 그러나 Amazon API Gateway V1 API에서는 하위 리소스가 상위 리소스에 설정된 태그를 상속합니다. 예:

- RestApi 리소스에 태그가 설정되면 해당 태그는 속성 기반 액세스 제어에 대한 RestApi의 다음 하위 리소스에 의해 상속됩니다.
  - Authorizer
  - Deployment
  - Documentation
  - GatewayResponse
  - Integration
  - Method
  - Model
  - Resource
  - ResourcePolicy
  - Setting
  - Stage
- DomainName에 태그가 설정되면 해당 태그는 아래에 있는 BasePathMapping 리소스에 의해 상속됩니다.

- UsagePlan에 태그가 설정되면 해당 태그는 아래에 있는 UsagePlanKey 리소스에 의해 상속됩니다.

> **Note**
>
> 태그 상속은 속성 기반 액세스 제어에만 적용됩니다. 예를 들어 상속된 태그를 사용하여 AWS Cost Explorer의 비용을 모니터링할 수 없습니다. API Gateway는 리소스에 대해 GetTags 호출 시 상속된 태그를 반환하지 않습니다.

Amazon API Gateway V1 API의 태그 상속

이전에는 단계에만 태그를 설정할 수 있었습니다. 이제는 다른 리소스에도 태그를 설정할 수 있으므로 Stage에서 다음 두 가지 방법으로 태그를 수신할 수 있습니다.

- Stage에 태그를 직접 설정할 수 있습니다.
- 단계가 해당 상위 RestApi에서 태그를 상속할 수 있습니다.

단계가 두 방법으로 태그를 수신하는 경우 단계에 직접 설정된 태그가 우선합니다. 예를 들어, 단계가 상위 REST API에서 다음 태그를 상속하는 경우 다음과 같습니다.

```
{
 'foo': 'bar',
 'x':'y'
}
```

단계에 다음 태그가 직접 설정되어 있는 경우 다음과 같습니다.

```
{
 'foo': 'bar2',
 'hello': 'world'
}
```

단계에 다음 값과 함께 다음 태그가 포함됩니다.

```
{
 'foo': 'bar2',
```

```
  'hello': 'world'
  'x':'y'
}
```

태그 제한 및 사용 규칙

다음 제한 및 사용 규칙은 API Gateway 리소스와 함께 태그를 사용할 때 적용합니다.

- 각 리소스는 최대 50개의 태그를 보유할 수 있습니다.
- 각 리소스에 대해 각 태그 키는 고유하며 하나의 값만 가질 수 있습니다.
- 태그 키의 최대 길이는 UTF-8 형식의 유니코드 문자 128자입니다.
- 태그 값의 최대 길이는 UTF-8 형식의 유니코드 문자 256자입니다.
- 키와 값에 허용되는 문자는 UTF-8로 표현할 수 있는 문자, 숫자, 공백 및 . : + = @ _ / -(하이픈) 문자도 있습니다. Amazon EC2 리소스는 모든 문자를 허용합니다.
- 태그 키와 값은 대/소문자를 구분합니다. 태그를 대문자로 사용하는 전략을 세우고 이러한 전략을 모든 리소스 유형에 대해 일관되게 구현하는 것이 가장 좋습니다. 예를 들어, `Costcenter`, `costcenter` 또는 `CostCenter`를 사용할지 결정하고 모든 태그에 대해 동일한 규칙을 사용합니다. 대/소문자가 일치하지 않는 유사한 태그를 사용하지 마십시오.
- `aws:` 접두사는 AWS가 사용하도록 예약되어 있으므로 태그에 사용할 수 없습니다. 이 접두사가 지정된 태그 키나 값은 편집하거나 삭제할 수 없습니다. 이 접두사가 지정된 태그는 리소스당 태그 수 제한에 포함되지 않습니다.

태그를 사용하여 API Gateway REST API 리소스에 대한 액세스 제어

AWS Identity and Access Management 정책의 조건은 API Gateway 리소스에 대한 권한을 지정하는 데 사용하는 구문의 일부입니다. IAM 정책 지정에 대한 자세한 내용은 the section called “IAM 권한 사용” 단원을 참조하세요. API Gateway에서는 리소스에 태그가 있을 수 있고 일부 작업에 태그가 포함될 수 있습니다. IAM 정책을 생성하면 태그 조건 키를 사용하여 다음을 제어할 수 있습니다.

- API Gateway 리소스에 이미 있는 태그를 기반으로 어떤 사용자가 해당 리소스에 대해 작업을 수행할 수 있는지 여부
- 작업의 요청에서 전달될 수 있는 태그
- 요청에서 특정 키를 사용할 수 있는지 여부를 제어합니다.

속성 기반 액세스 제어 태그를 사용하면 API 수준의 제어를 보다 세부적으로 제어할 수 있을 뿐만 아니라, 리소스 기반 액세스 제어보다 동적으로 제어할 수 있습니다. 요청에서 제공된 태그(요청 태그) 또는 작업 중인 리소스에 대한 태그(리소스 태그)를 기반으로 작업을 허용하거나 허용하지 않는 IAM 정책을 생성할 수 있습니다. 일반적으로 리소스 태그는 이미 존재하는 리소스에 대한 태그입니다. 요청 태그는 새 리소스를 생성할 때 사용되는 태그입니다.

태그 조건 키의 전체 구문 및 의미는 IAM 사용 설명서의 태그를 사용한 액세스 제어를 참조하십시오.

다음 예제에서는 API Gateway 사용자에 정책의 태그 조건을 지정하는 방법을 설명합니다.

리소스 태그 기반 작업 한도 지정

다음 정책 예에서는 해당 리소스에 값이 `prod`인 태그 `Environment`가 없는 경우 모든 리소스에서 모든 작업을 수행할 수 있는 권한을 사용자에게 부여합니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "apigateway:*",
      "Resource": "*"
    },
    {
      "Effect": "Deny",
      "Action": [
        "apigateway:*"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/Environment": "prod"
        }
      }
    }
  ]
}
```

리소스 태그 기반 작업 허용

다음 정책 예에서는 해당 리소스에 값이 Development인 태그 Environment가 있는 경우 사용자가 API Gateway 리소스에서 모든 작업을 수행할 수 있도록 허용합니다. Deny 문은 사용자가 Environment 태그의 값을 변경하지 못하도록 합니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ConditionallyAllow",
      "Effect": "Allow",
      "Action": [
        "apigateway:*"
      ],
      "Resource": [
        "arn:aws:apigateway:*::*"
      ],
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/Environment": "Development"
        }
      }
    },
    {
      "Sid": "AllowTagging",
      "Effect": "Allow",
      "Action": [
        "apigateway:*"
      ],
      "Resource": [
        "arn:aws:apigateway:*::/tags/*"
      ]
    },
    {
      "Sid": "DenyChangingTag",
      "Effect": "Deny",
      "Action": [
        "apigateway:*"
      ],
      "Resource": [
        "arn:aws:apigateway:*::/tags/*"
      ],
```

```
      "Condition": {
        "ForAnyValue:StringEquals": {
          "aws:TagKeys": "Environment"
        }
      }
    }
  ]
}
```

태그 지정 작업 거부

다음 정책 예에서는 사용자가 태그 변경을 제외한 모든 API Gateway 작업을 수행할 수 있도록 허용합니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "apigateway:*"
      ],
      "Resource": [
        "*"
      ],
    },
    {
      "Effect": "Deny",
      "Action": [
        "apigateway:*"
      ],
      "Resource": "arn:aws:apigateway:*::/tags*",
    }
  ]
}
```

태그 지정 작업 허용

다음 정책 예에서는 사용자가 모든 API Gateway 리소스를 가져오고 해당 리소스에 대한 태그를 변경할 수 있도록 허용합니다. 리소스의 태그를 가져오려면 사용자에게 해당 리소스에 대한 GET 권한이 있어야 합니다. 리소스의 태그를 업데이트하려면 사용자에게 해당 리소스에 대한 PATCH 권한이 있어야 합니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "apigateway:GET",
        "apigateway:PUT",
        "apigateway:POST",
        "apigateway:DELETE"
      ],
      "Resource": [
        "arn:aws:apigateway:*::/tags/*",
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "apigateway:GET",
        "apigateway:PATCH",
      ],
      "Resource": [
        "arn:aws:apigateway:*::*",
      ]
    }
  ]
}
```

API 참조

Amazon API Gateway는 자체 HTTP 및 WebSocket API를 만들고 배포하기 위한 API를 제공합니다. 또한 API Gateway API는 표준 AWS SDK에서 사용할 수 있습니다.

AWS SDK가 있는 언어를 사용하는 경우 API Gateway REST API를 직접 사용하는 대신 SDK를 사용하는 것이 좋습니다. SDK를 사용하면 인증을 더 간단하게 만들고, 개발 환경에 쉽게 통합할 수 있으며, API Gateway 명령에 쉽게 액세스할 수 있습니다.

AWS SDK 및 API Gateway REST API 참조 문서를 찾을 수 있는 위치는 다음과 같습니다.

- Amazon Web Services용 도구
- Amazon API Gateway REST API 참조
- Amazon API Gateway WebSocket 및 HTTP API 참조

Amazon API Gateway 할당량 및 중요 정보

주제

- 리전당 API Gateway 계정 수준 할당량
- HTTP API 할당량
- WebSocket API 구성 및 실행에 대한 API Gateway 할당량
- REST API 구성 및 실행에 대한 API Gateway 할당량
- API 생성, 배포 및 관리를 위한 API Gateway 할당량
- Amazon API Gateway 중요 정보

특별한 언급이 없는 한 요청 시 할당량을 높일 수 있습니다. 할당량 증가를 요청하려면 Service Quotas를 사용하거나 AWS 지원 센터에 문의할 수 있습니다.

권한 부여가 활성화된 메서드의 경우, 메서드 ARN(예: `arn:aws:execute-api:{region-id}:{account-id}:{api-id}/{stage-id}/{method}/{resource}/{path}`)의 최대 길이는 1,600바이트입니다. 경로 파라미터 값(실행 시간에 크기가 결정됨) 으로 인해 ARN 길이는 한도를 초과할 수 있습니다. 이 경우 API 클라이언트는 `414 Request URI too long` 응답을 수신합니다.

> **Note**
>
> 이는 리소스 정책이 사용될 때 URI 길이를 제한합니다. 리소스 정책이 필요한 프라이빗 API의 경우 모든 프라이빗 API의 URI 길이가 제한됩니다.

리전당 API Gateway 계정 수준 할당량

다음 할당량은 Amazon API Gateway에서 리전별로 계정당 적용됩니다.

리소스 또는 작업	기본 할당량	높일 수 있음
HTTP API, REST API, WebSocket API 및 WebSocket 콜백 API에서의 리	5,000개의 요청을 처리할 수 있는 최대 버킷 용량을 사용하여 토큰 버킷 알고리즘에서 제공하는 추가 버스트 용량이 있는 초당 10,000개의 요청(RPS). *	예

리소스 또는 작업	기본 할당량	높일 수 있음
전별 계정당 조절 할당량	Note 버스트 할당량은 리전의 해당 계정에 대한 전체 RPS 할당량에 따라 API Gateway 서비스 팀에서 결정됩니다. 이 할당량은 고객이 조절하거나 변경을 요청할 수 없습니다.	
리전 API	600	아니요
엣지 최적화 API	120	아니요

* 아프리카(케이프타운) 및 유럽(밀라노) 리전의 경우 기본 조절 할당량은 2500 RPS이고, 기본 버스트 할당량은 1250 RPS입니다.

HTTP API 할당량

API Gateway에서의 HTTP API 구성 및 실행에는 다음 할당량이 적용됩니다.

리소스 또는 작업	기본 할당량	높일 수 있음
API당 라우팅	300	예
API당 통합 수	300	아니요
최대 통합 제한 시간	30초	아니요
API당 단계	10	예
도메인별 멀티 레벨 API 매핑	200	아니요
단계당 태그 수	50	아니요

리소스 또는 작업	기본 할당량	높일 수 있음
요청 라인 및 헤더 값의 총 결합 크기	10,240바이트	아니요
페이로드 크기	10MB	아니요
리전별 계정당 사용자 지정 도메인	120	예
액세스 로그 템플릿 크기	3KB	아니요
Amazon CloudWatch Logs 로그 항목	1MB	아니요
API별 권한 부여자	10	예
권한 부여자별 대상 그룹	50	아니요
경로별 범위	10	아니요
JSON Web Key Set 엔드포인트에 대한 제한 시간	1500ms	아니요
JSON Web Key Set 엔드포인트의 응답 크기	150000바이트	아니요
OpenID Connect 검색 엔드포인트에 대한 제한 시간	1500ms	아니요
Lambda 권한 부여자 응답 시간 제한	10,000ms	아니요

리소스 또는 작업	기본 할당량	높일 수 있음
리전별 계정당 VPC 링크	10	예
VPC 링크당 서브넷	10	예
단계별 단계 변수	100	아니요
단계 변수의 키의 길이(문자 수)	64	아니요
단계 변수의 값의 길이(문자 수)	512	아니요

WebSocket API 구성 및 실행에 대한 API Gateway 할당량

Amazon API Gateway에서의 WebSocket API 구성 및 실행에는 다음 할당량이 적용됩니다.

리소스 또는 작업	기본 할당량	높일 수 있음
리전별 계정당 초당 새 연결 수(모든 WebSocket API에서)	500	예
동시 연결	해당 사항 없음 *	해당 사항 없음
AWS LambdaAPI별 권한 부여자	10	예
AWS Lambda 권한 부여자 결과 크기	8KB	아니요
API당 라우팅	300	예

리소스 또는 작업	기본 할당량	높일 수 있음
API당 통합 수	300	예
통합 시간제한	Lambda, Lambda 프록시, HTTP, HTTP 프록시 및 AWS 통합을 포함하는 모든 통합 유형의 경우, 50밀리초 - 29초.	아니요
API당 단계	10	예
WebSocket 프레임 크기	32KB	아니요
페이로드 크기 관리	128KB **	아니요
WebSocket API 연결 기간	2시간	아니요
유휴 연결 제한 시간	10분	아니요
WebSocket API에 대한 URL의 길이 (문자 수)	4096	아니요

* API Gateway는 동시 연결에 할당량을 적용하지 않습니다. 최대 동시 연결 수는 초당 새 연결 속도와 최대 연결 지속 시간(2시간)에 의해 결정됩니다. 예를 들어 기본 할당량이 초당 500개의 새 연결인 경우 클라이언트가 2시간에 걸쳐 최대 속도로 연결하는 경우 API Gateway는 최대 3,600,000개의 동시 연결을 제공할 수 있습니다.

** WebSocket 프레임 크기 할당량이 32KB이기 때문에 32KB를 초과하는 메시지는 32KB 이하의 여러 프레임으로 분할되어야 합니다. 이는 `@connections` 명령에 적용됩니다. 큰 메시지(또는 큰 프레임 크기)가 수신되면 코드 1009와 함께 연결이 해제됩니다.

REST API 구성 및 실행에 대한 API Gateway 할당량

Amazon API Gateway에서의 REST API 구성 및 실행에는 다음 할당량이 적용됩니다.

리소스 또는 작업	기본 할당량	높일 수 있음
리전별 계정당 사용자 지정 도메인 이름	120	예
도메인별 멀티 레벨 API 매핑	200	아니요
엣지 최적화된 API에 대한 URL의 길이(문자 수)	8192	아니요
지역 API에 대한 URL의 길이(문자 수)	10240	아니요
리전별 계정당 프라이빗 API	600	아니요
API Gateway 리소스 정책의 길이(문자 수 단위)	8192	예
리전별 계정당 API 키	10000	아니요
리전별 계정당 클라이언트 인증서	60	예
API별 권한 부여자(AWS Lambda 및 Amazon Cognito)	10	예
API당 설명서 부분	2000	예
API당 리소스	300	예

리소스 또는 작업	기본 할당량	높일 수 있음
API당 단계	10	예
단계별 단계 변수	100	아니요
단계 변수의 키의 길이(문자 수)	64	아니요
단계 변수의 값의 길이(문자 수)	512	아니요
리전별 계정당 사용량 계획	300	예
API 키당 사용량 계획	10	예
리전별 계정당 VPC 링크	20	예
API 캐싱 TTL	기본적으로 300초이고 API 소유자가 0~3600으로 구성 가능합니다.	상한용이 아님(3600)
캐싱된 응답 크기	1048576바이트 캐시 데이터 암호화는 캐싱되는 항목의 크기를 증가시킬 수 있습니다.	아니요
통합 시간제한	Lambda, Lambda 프록시, HTTP, HTTP 프록시 및 AWS 통합을 포함하는 모든 통합 유형의 경우, 50밀리초 - 29초.	하한 또는 상한에는 해당되지 않음.
모든 헤더 값의 전체 결합 크기	10240바이트	아니요

리소스 또는 작업	기본 할당량	높일 수 있음
프라이빗 API의 경우 모든 헤더 값의 전체 결합 크기	8000바이트	아니요
페이로드 크기	10MB	아니요
단계당 태그 수	50	아니요
매핑 템플릿의 `#foreach ... #end` 루프의 반복 횟수	1000	아니요
권한 부여가 포함된 메서드의 ARN 길이	1600 bytes	아니요
사용량 계획의 단계에 대한 메서드 수준 제한 설정	20	예
API당 모델 크기	400KB	아니요

restapi:import 또는 restapi:put의 경우, API 정의 파일의 최대 크기는 6MB입니다.

모든 API별 할당량은 특정 API에서만 증가시킬 수 있습니다.

API 생성, 배포 및 관리를 위한 API Gateway 할당량

API Gateway에서 AWS CLI, API Gateway 콘솔 또는 API Gateway REST API 및 SDK를 사용하여 API를 생성, 배포 및 관리하는 경우 다음과 같이 고정된 할당량이 적용됩니다. 이러한 할당량은 늘릴 수 없습니다.

작업	기본 할당량	높일 수 있음
CreateApiKey	계정 한 개에 대해 초당 5개의 요청	아니요
CreateDeployment	계정 한 개에 대해 5초당 1개의 요청	아니요
CreateDocumentationVersion	계정 한 개에 대해 20초당 1개의 요청	아니요
CreateDomainName	계정 한 개에 대해 30초당 1개의 요청	아니요
CreateResource	계정 한 개에 대해 초당 5개의 요청	아니요
CreateRestApi	리전 또는 프라이빗 API • 계정 한 개에 대해 3초당 1개의 요청 엣지 최적화 API • 계정 한 개에 대해 30초당 1개의 요청	아니요
CreateVpcLink(V2)	계정 한 개에 대해 15초당 1개의 요청	아니요
DeleteApiKey	계정 한 개에 대해 초당 5개의 요청	아니요
DeleteDomainName	계정 한 개에 대해 30초당 1개의 요청	아니요
DeleteResource	계정 한 개에 대해 초당 5개의 요청	아니요

작업	기본 할당량	높일 수 있음
DeleteRestApi	계정 한 개에 대해 30초당 1개의 요청	아니요
GetResources	계정 한 개에 대해 2초당 5개의 요청	아니요
DeleteVpcLink(V2)	계정 한 개에 대해 30초당 1개의 요청	아니요
ImportDocumentationParts	계정 한 개에 대해 30초당 1개의 요청	아니요
ImportRestApi	리전 또는 프라이빗 API • 계정 한 개에 대해 3초당 1개의 요청 엣지 최적화 API • 계정 한 개에 대해 30초당 1개의 요청	아니요
PutRestApi	계정 한 개에 대해 초당 1개의 요청	아니요
UpdateAccount	계정 한 개에 대해 20초당 1개의 요청	아니요
UpdateDomainName	계정 한 개에 대해 30초당 1개의 요청	아니요
UpdateUsagePlan	계정 한 개에 대해 20초당 1개의 요청	아니요
기타 작업	총 계정 할당량까지 할당량이 없습니다.	아니요

작업	기본 할당량	높일 수 있음
총 작업	초당 40개의 요청이 버스트 할당량인 초당 10개의 요청입니다.	아니요

Amazon API Gateway 중요 정보

주제

- Amazon API Gateway의 REST 및 WebSocket API 중요 정보
- Amazon API Gateway의 WebSocket API 중요 정보
- Amazon API Gateway의 REST API 중요 정보

Amazon API Gateway의 REST 및 WebSocket API 중요 정보

- API Gateway는 REST 및 WebSocket API에서 사용자 지정 도메인 이름 공유를 지원하지 않습니다.
- 단계 이름에는 영숫자, 하이픈, 밑줄만 사용할 수 있습니다. 최대 길이는 128자입니다.
- `/ping` 및 `/sping` 경로가 서비스 상태 확인을 위해 예약되어 있습니다. 사용자 지정 도메인이 있는 API 루트 수준의 리소스에서 이를 사용하면 정상적인 결과가 나오지 않습니다.
- API Gateway는 현재 로그 이벤트를 1,024바이트로 제한합니다. 요청 및 응답 본문과 같이 1,024바이트가 넘는 로그 이벤트는 CloudWatch Logs에 제출되기 전에 API Gateway에서 잘립니다.
- CloudWatch 지표는 현재 차원 이름 및 값을 유효한 XML 문자 255자로 제한합니다. (자세한 내용은 CloudWatch 사용 설명서를 참조하세요.) 차원 값은 API 이름, 레이블(스테이지) 이름 및 리소스 이름을 포함하여 사용자가 정의한 이름의 함수입니다. 이 이름을 선택할 때는 CloudWatch 지표 제한을 초과하지 않도록 주의하세요.
- 매핑 템플릿의 최대 크기는 300KB입니다.

Amazon API Gateway의 WebSocket API 중요 정보

- API Gateway는 최대 128KB의 메시지 페이로드를 지원하며 최대 프레임 크기는 32KB입니다. 메시지가 32KB를 초과하면 32KB 이하의 여러 프레임으로 분할해야 합니다. 이보다 더 큰 메시지가 수신되면 코드 1009와 함께 연결이 해제됩니다.

Amazon API Gateway의 REST API 중요 정보

- 임의의 요청 URL 쿼리 문자열에 대해서는 일반 텍스트 파이프 문자(|)가 지원되지 않으며 URL 인코딩되어야 합니다.
- 임의의 요청 URL 쿼리 문자열에 대해서는 세미콜론 문자(;)가 지원되지 않으며 세미콜론 문자를 사용할 경우 데이터가 분할됩니다. 일반적으로 REST API는 URL 인코딩 요청 파라미터를 백엔드 통합에 전달하기 전에 디코딩합니다.
- API Gateway 콘솔을 사용하여 API를 테스트할 때 자체 서명된 인증서를 백엔드에 제시하거나, 인증서 체인에서 중간 인증서가 누락되었거나, 기타 확인되지 않은 인증서 관련 예외가 백엔드에서 발생하는 경우 "알 수 없는 엔드포인트 오류" 응답을 받을 수 있습니다.
- 프라이빗 통합이 포함된 API `Resource` 또는 `Method` 엔터티의 경우, `VpcLink`의 하드 코딩된 참조를 모두 제거한 후 삭제해야 합니다. 그러지 않으면 현수 통합이 되고, `Resource` 또는 `Method` 엔터티가 삭제되어도 VPC 링크가 여전히 사용 중이라고 하는 오류를 수신하게 됩니다. 이 동작은 프라이빗 통합이 단계 변수를 통해 `VpcLink`를 참조할 때는 적용되지 않습니다.
- 다음 백엔드에서는 API Gateway와 호환되는 방식으로 SSL 클라이언트 인증을 지원하지 않을 수 있습니다.
  - NGINX
  - Heroku
- API Gateway는 대부분의 OpenAPI 2.0 사양 및 OpenAPI 3.0 사양을 지원하며, 다음은 예외입니다.
  - 경로 세그먼트에는 영숫자, 밑줄, 하이픈, 점, 쉼표, 콜론, 중괄호만 사용할 수 있습니다. 경로 파라미터는 별도의 경로 세그먼트여야 합니다. 예를 들어 "resource/{path_parameter_name}"은 유효하지만, "resource{path_parameter_name}"은 유효하지 않습니다.
  - 모델 이름에는 영숫자만 사용할 수 있습니다.
  - 파라미터 입력에는 다음과 같은 속성만 지원됩니다.`name` ,`in` ,`required` ,`type` ,`description` 다른 속성은 무시됩니다.
  - `securitySchemes` 유형을 사용할 경우 `apiKey`여야 합니다. 하지만 OAuth 2 및 HTTP 기본 인증은 Lambda 권한 부여자를 통해 지원되고, OpenAPI 구성은 공급자 확장 기능을 통해 가능합니다.
  - `deprecated` 필드는 지원되지 않으며 내보낸 API에서 제거됩니다.
  - API Gateway 모델은 OpenAPI가 사용하는 JSON 대신 JSON 스키마 draft 4를 사용하여 정의합니다.
  - 스키마 객체에서 `discriminator` 파라미터는 지원되지 않습니다.
  - `example` 태그는 지원되지 않습니다.

- `exclusiveMinimum`은 API Gateway에서 지원되지 않습니다.
- `maxItems` 및 `minItems` 태그는 단순 요청 확인에 포함되지 않습니다. 이 문제를 해결하려면 확인을 수행하기 전에 가져온 후 업데이트합니다.
- `oneOf`는 OpenAPI 2.0 또는 SDK 생성에서 지원되지 않습니다.
- `readOnly` 필드는 지원되지 않습니다.
- `$ref`는 다른 파일을 참조하는 데 사용할 수 없습니다.
- OpenAPI 문서 루트에서는 `"500": {"$ref": "#/responses/UnexpectedError"}` 형식의 응답 정의를 사용할 수 없습니다. 이 문제를 해결하려면 참조를 인라인 스키마로 바꿉니다.
- `Int32` 또는 `Int64` 유형의 숫자는 지원되지 않습니다. 예를 들면 다음과 같습니다.

```
"elementId": {
    "description": "Working Element Id",
    "format": "int32",
    "type": "number"
}
```

- 10진수 형식 유형(`"format": "decimal"`)은 스키마 정의에서 지원되지 않습니다.
- 메서드 응답에서 스키마 정의는 객체 유형이어야 하며 기본 유형일 수 없습니다. 예를 들어 `"schema": { "type": "string"}`은 지원되지 않습니다. 그러나 다음 객체 유형을 사용하여 이 문제를 해결할 수 있습니다.

```
"schema": {
     "$ref": "#/definitions/StringResponse"
            }

 "definitions": {
    "StringResponse": {
      "type": "string"
    }
  }
```

- API Gateway는 OpenAPI 사양에 정의된 루트 수준 보안을 사용하지 않습니다. 따라서 보안은 적절하게 적용될 수 있는 운영 수준에서 정의되어야 합니다.
- `default` 키워드는 지원되지 않습니다.

- API Gateway는 Lambda 통합 또는 HTTP 통합으로 메서드를 처리할 때 다음과 같은 제한 및 제약을 적용합니다.
  - 헤더 이름과 쿼리 파라미터는 대/소문자를 구분하여 처리합니다.

- 다음 표에는 통합 엔드포인트로 헤더를 전송하거나 통합 엔드포인트에서 이를 다시 전송하는 경우, 끊기거나 다시 매핑되거나 수정될 수 있는 헤더가 나열됩니다. 이 표에서
  - Remapped은 헤더 이름이 *<string>*에서 X-Amzn-Remapped-*<string>*으로 변경됨을 의미합니다.

    Remapped Overwritten은 헤더 이름이 *<string>*에서 X-Amzn-Remapped-*<string>*으로 변경됨을 의미합니다.

헤더 이름	요청(**http/http_proxy /lambda**)	응답(**http/http_proxy /lambda**)
Age	패스스루	패스스루
Accept	패스스루	끊김/전달/전달
Accept-Charset	패스스루	패스스루
Accept-Encoding	패스스루	패스스루
Authorization	패스스루 *	다시 매핑됨
Connection	전달/전달/끊김	다시 매핑됨
Content-Encoding	전달/끊김/전달	패스스루
Content-Length	전달(본문에 기반하여 생성됨)	패스스루
Content-MD5	끊김	다시 매핑됨

헤더 이름	요청(**http**/**http_proxy** /**lambda**)	응답(**http**/**http_proxy** /**lambda**)
Content-Type	패스스루	패스스루
Date	패스스루	재매핑됨 덮어씀
Expect	끊김	끊김
Host	통합 엔드포인트 덮어씀	끊김
Max-Forwards	끊김	다시 매핑됨
Pragma	패스스루	패스스루
Proxy-Authenticate	끊김	끊김
Range	패스스루	패스스루
Referer	패스스루	패스스루
Server	끊김	재매핑됨 덮어씀
TE	끊김	끊김
Transfer-Encoding	끊김/끊김/예외	끊김
Trailer	끊김	끊김

헤더 이름	요청(**http/http_proxy /lambda**)	응답(**http/http_proxy /lambda**)
Upgrade	끊김	끊김
User-Agent	패스스루	다시 매핑됨
Via	끊김/끊김/전달	전달/끊김/끊김
Warn	패스스루	패스스루
WWW-Authenticate	끊김	다시 매핑됨

* 서명 버전 4 서명이 포함되어 있거나 AWS_IAM 권한 부여가 사용되는 경우 Authorization 헤더는 삭제됩니다.

- API Gateway에서 생성된 API의 Android SDK는 java.net.HttpURLConnection 클래스를 사용합니다. Android 4.4 이전을 실행하는 장치에서 이 클래스는 WWW-Authenticate 헤더를 X-Amzn-Remapped-WWW-Authenticate에 다시 매핑하여 나온 401 응답에 대해 처리되지 않은 예외를 발생시킵니다.
- API Gateway에서 생성한 Java, Android 및 API의 iOS SDK와 달리, API Gateway에서 생성한 API의 JavaScript SDK는 500 수준 오류에 대한 재시도를 지원하지 않습니다.
- 메서드의 테스트 호출은 application/json의 기본 콘텐츠 유형을 사용하며 다른 콘텐츠 유형의 사양을 무시합니다.
- X-HTTP-Method-Override 헤더를 전달하여 API에 요청을 보낼 때 API Gateway는 메서드를 재정의합니다. 따라서 헤더를 백엔드에 전달하려면 헤더를 통합 요청에 추가해야 합니다.
- 요청의 Accept 헤더에 여러 미디어 유형이 포함되어 있는 경우 API Gateway에서는 첫 번째 Accept 미디어 유형만 인식합니다. Accept 미디어 유형의 순서를 제어할 수 없고 이진 콘텐츠의 미디어 유형이 목록의 맨 앞에 있지 않은 경우 API의 Accept 목록에서 첫 번째 binaryMediaTypes 미디어 유형을 추가할 수 있습니다. 그러면 API Gateway에서 콘텐츠를 이진 유형으로 반환합니다. 예를 들어 브라우저에서 <img> 요소를 사용하여 JPEG 파일을 보내려는 경우 브라우저에서 요청에 Accept:image/webp,image/*,*/*;q=0.8을 전송할 수 있습니다.

`image/webp`를 `binaryMediaTypes` 목록에 추가하여 엔드포인트에서 JPEG 파일을 이진 유형으로 수신합니다.

- `413 REQUEST_TOO_LARGE`에 대한 기본 게이트웨이 응답 사용자 지정은 현재 지원되지 않습니다.
- API Gateway에는 모든 통합 응답에 대한 `Content-Type` 헤더가 포함되어 있습니다. 기본적으로 콘텐츠 유형은 `application/json`입니다.

문서 기록

다음 표에는 Amazon API Gateway의 최신 릴리스가 발표된 이후 이 설명서에서 변경된 중요 사항이 기술되어 있습니다. 이 문서에 대한 업데이트 알림을 받으려면 상위 메뉴에서 RSS 버튼을 선택하여 RSS 피드를 구독하면 됩니다.

- 설명서 최종 업데이트: 2023년 12월 10일

변경 사항	설명	날짜
REST API 및 WebSocket API 콘솔 업데이트	REST API 및 WebSocket API에 대한 콘솔 정보가 업데이트되었습니다.	2023년 12월 10일
문서 업데이트	개념 정보를 업데이트하고 API Gateway REST API에 대한 데이터 변환 및 요청 확인 주제의 새 자습서를 만들었습니다. 자세한 내용은 API Gateway에서 요청 확인 사용 및 REST API에 대한 데이터 변환 설정을 참조하세요.	2023년 6월 22일
다중 리전 API Gateway를 위한 DNS 장애 조치 구성	Amazon Route 53 상태 확인을 사용하여 기본 지역의 API Gateway REST API에서 보조 지역의 API Gateway REST AWS 리전 API로 DNS 장애 조치를 제어하는 지원이 추가되었습니다. 자세한 내용은 DNS 장애 조치에 대한 사용자 지정 상태 확인 구성을 참조하세요.	2022년 10월 31일
문서 업데이트	REST API 및 HTTP API API에 대한 핵심 기능 요약을 업데이트했습니다. 자세한 내용은	2022년 5월 31일

	REST API와 HTTP API 중에서 선택을 참조하세요.	
관리형 정책 업데이트	`AWSServiceRoleForAPIGateway` 정책에 `acm:GetCertificate` 지원이 추가되었습니다. 자세한 내용은 API Gateway에 서비스 연결 역할 사용을 참조하세요.	2021년 7월 12일
HTTP API에 대한 파라미터 매핑	HTTP API에 대한 파라미터 매핑 지원이 추가되었습니다. 자세한 내용은 API 요청 및 응답 변환을 참조하세요.	2021년 1월 7일
REST API의 기본 엔드포인트 비활성화	REST API의 기본 엔드포인트를 비활성화하는 지원이 추가되었습니다. 자세한 내용은 REST API의 기본 엔드포인트 비활성화를 참조하십시오.	2020년 10월 29일
상호 TLS 인증	REST API 및 HTTP API에 대한 상호 TLS 인증 지원이 추가되었습니다. 자세한 내용은 REST API에 대한 상호 TLS 인증 구성 및 HTTP API에 대한 상호 TLS 인증 구성을 참조하세요.	2020년 9월 17일
HTTP API 권한 AWS Lambda 부여자	HTTP API에 대한 AWS Lambda 권한 부여자에 대한 지원이 추가되었습니다. 자세한 내용은 HTTP API용 AWS Lambda 권한 부여자와 작업하기를 참조하십시오.	2020년 9월 9일

HTTP API 서비스 통합 AWS	HTTP API의 AWS 서비스 통합에 대한 지원이 추가되었습니다. 자세한 내용은 HTTP API용 AWS 서비스 통합 작업을 참조하십시오.	2020년 8월 20일
HTTP API 와일드카드 사용자 지정 도메인	HTTP API에 대한 와일드카드 사용자 지정 도메인 이름 지원이 추가되었습니다. 자세한 내용은 와일드카드 사용자 지정 도메인 이름을 참조하세요.	2020년 8월 10일
서버리스 개발자 포털 개선	관리자 패널에 사용자 관리 기능이 추가되었으며 API 정의 내보내기에 대한 지원이 추가되었습니다. 자세한 내용은 서버리스 개발자 포털을 사용하여 API Gateway API 목록화를 참조하세요.	2020년 6월 25일
WebSocket API 지원 `Sec-WebSocket-Protocol`	`Sec-WebSocket-Protocol` 필드에 대한 지원이 추가되었습니다. 자세한 내용은 WebSocket 하위 프로토콜이 필요한 $connect 경로 설정을 참조하십시오.	2020년 6월 16일
HTTP API 내보내기	HTTP API의 OpenAPI 3.0 정의를 내보내는 지원이 추가되었습니다. 자세한 내용은 API Gateway에서 HTTP API 내보내기를 참조하십시오.	2020년 4월 20일
보안 설명서	보안 설명서가 추가되었습니다. 자세한 내용은 Amazon API Gateway의 보안을 참조하세요.	2020년 3월 31일

재구성된 설명서	개발자 안내서를 재구성했습니다.	2020년 3월 12일
HTTP API 정식 출시	HTTP API가 정식 출시되었습니다. 자세한 내용은 HTTP API 작업을 참조하세요.	2020년 3월 12일
HTTP API 로깅	HTTP API 로그에서 `$context.integrationErrorMessage` 에 대한 지원이 추가되었습니다. 자세한 내용은 HTTP API 로깅 변수를 참조하세요.	2020년 2월 26일
AWS OpenAPI 임포트를 위한 변수	OpenAPI 정의에 AWS 변수에 대한 지원이 추가되었습니다. 자세한 내용은 OpenAPI 가져오기를 위한AWS 변수를 참조하세요.	2020년 2월 17일
HTTP API	베타 버전의 HTTP API가 릴리스되었습니다. 자세한 내용은 HTTP API를 참조하세요.	2019년 12월 4일
와일드카드 사용자 정의 도메인 이름	와일드카드 사용자 지정 도메인 이름에 대한 지원이 추가되었습니다. 자세한 내용은 와일드카드 사용자 지정 도메인 이름을 참조하세요.	2019년 10월 21일
Amazon Data Firehose 로깅	Amazon Data Firehose에 대한 지원을 로깅 데이터에 대한 액세스 대상으로 추가했습니다. 자세한 내용은 Amazon Data Firehose를 API Gateway 액세스 로깅의 대상으로 사용하기 섹션을 참조하십시오.	2019년 10월 15일

프라이빗 API 호출을 위한 Route53 별칭	프라이빗 API 호출을 위한 추가 Route53 별칭 DNS 레코드에 대한 지원이 추가되었습니다. 자세한 내용은 Route53 별칭을 사용하여 프라이빗 API 액세스를 참조하세요.	2019년 9월 18일
API에 대한 태그 기반 액세스 제어 WebSocket	API에 대한 태그 기반 액세스 제어에 대한 지원이 추가되었습니다. WebSocket 자세한 내용은 태그할 수 있는 API Gateway 리소스를 참조하세요.	2019년 6월 27일
사용자 지정 도메인에 대한 TLS 버전 선택	사용자 지정 도메인에 배포된 API에 대한 TLS(전송 계층 보안) 버전 선택 지원이 추가되었습니다. API Gateway의 사용자 지정 도메인에 대한 최소 TLS 버전 선택의 노트를 참조하세요.	2019년 6월 20일
프라이빗 API에 대한 VPC 종단점 정책	인터페이스 VPC 종단점에 엔드포인트 정책을 연결하여 프라이빗 API의 보안을 향상시키는 지원이 추가되었습니다. 자세한 내용은 API Gateway의 프라이빗 API에 대한 VPC 종단점 정책 사용을 참조하세요.	2019년 6월 4일
업데이트된 설명서	Amazon API Gateway 시작하기를 다시 작성했습니다. 자습서를 Amazon API Gateway 자습서로 이동했습니다.	2019년 5월 29일

REST API를 위한 태그 기반 액세스 제어	REST API에 대한 태그 기반 액세스 제어에 대한 지원을 추가했습니다. 자세한 내용은 IAM 정책과 함께 태그를 사용하여 API Gateway 리소스에 대한 액세스 제어를 참조하세요.	2019년 5월 23일
업데이트된 설명서	6가지 항목: Amazon API Gateway란 무엇인가?, 자습서: HTTP 프록시 통합을 사용하여 API 빌드, 자습서: 세 가지 비 프록시 통합을 사용하여 Calc REST API 만들기, API Gateway 매핑 템플릿과 액세스 로깅 변수 참조 정보, API Gateway Lambda 권한 부여자 사용, API Gateway REST API 리소스에 대해 CORS 활성화를 다시 작성했습니다.	2019년 4월 5일
서버리스 개발자 포털 개선	관리자 패널 및 기타 기능을 개선하여 Amazon API Gateway 개발자 포털에서 API를 쉽게 게시할 수 있도록 했습니다. 자세한 내용은 개발자 포털을 사용하여 API 목록화를 참조하세요.	2019년 3월 28일
에 대한 지원 AWS Config	에 대한 지원이 추가되었습니다 AWS Config. 자세한 내용은 API Gateway API 구성 모니터링을 참조하십시오 AWS Config.	2019년 3월 20일

에 대한 지원 AWS CloudFormation	AWS CloudFormation 템플릿 참조에 API Gateway V2 API를 추가했습니다. 자세한 내용은 Amazon API Gateway V2 리소스 유형 참조를 참조하세요.	2019년 2월 7일
WebSocket API에 대한 지원	WebSocket API에 대한 지원이 추가되었습니다. 자세한 내용은 Amazon API Gateway에서 WebSocket API 생성을 참조하십시오.	2018년 12월 18일
다음을 통해 서버리스 개발자 포털을 이용할 수 있습니다. AWS Serverless Application Repository	Amazon API Gateway 개발자 포털 서버리스 애플리케이션을 이제 에서 AWS Serverless Application Repository(추가로 GitHub) 에서 사용할 수 있습니다. 자세한 내용은 개발자 포털을 사용하여 API Gateway API 목록화를 참조하세요.	2018년 11월 16일
에 대한 지원 AWS WAF	AWS WAF(웹 애플리케이션 방화벽)에 대한 지원을 추가했습니다. 자세한 내용은 API를 사용하여 액세스 제어를 참조하십시오 AWS WAF.	2018년 11월 5일
서버리스 개발자 포털	Amazon API Gateway가 이제 API Gateway API를 게시하기 위해 배포 가능한 서버리스 애플리케이션으로 완전히 사용자 지정 가능한 개발자 포털을 제공합니다. 자세한 내용은 개발자 포털을 사용하여 API Gateway API 목록화를 참조하세요.	2018년 10월 29일

다중 값 헤더 및 쿼리 문자열 파라미터에 대한 지원	Amazon API Gateway가 이제 동일한 이름을 지닌 복수의 헤더 및 쿼리 문자열 파라미터를 지원합니다. 자세한 내용은 다중 값 헤더 및 쿼리 문자열 파라미터에 대한 지원을 참조하세요.	2018년 10월 4일
OpenAPI 지원	Amazon API Gateway는 이제 OpenAPI (Swagger) 2.0뿐만 아니라 OpenAPI 3.0까지 지원합니다.	2018년 9월 27일
업데이트된 설명서	새로운 주제 추가: Amazon API Gateway 리소스 정책의 권한 부여 워크플로우에 대한 영향	2018년 9월 27일
능동적 AWS X-Ray 통합	이제 API를 통해 기본 서비스로 이동하는 사용자 요청의 지연 시간을 추적하고 분석하는 데 사용할 AWS X-Ray 수 있습니다. 자세한 내용은 API Gateway API 실행 추적을 참조하십시오 AWS X-Ray.	2018년 9월 6일
캐싱 개선	API 단계에 대한 캐싱을 활성화할 경우, GET 메서드만 기본적으로 캐싱을 활성화합니다. 이렇게 하면 API의 안전성이 보장됩니다. 메서드 설정을 재정의함으로써 다른 메서드에 대해 캐싱을 활성화할 수 있습니다. 자세한 내용은 응답 개선을 위해 API 캐싱 활성화 단원을 참조하세요.	2018년 8월 20일

서비스 한도 개정	제한 개정: 한 계정에 대한 API의 수를 늘렸습니다. 생성/가져오기/배포 API용 API 속도 제한을 늘렸습니다. 일부 속도를 분당에서 초당으로 변경했습니다. 자세한 내용은 제한을 참조하세요.	2018년 7월 13일
API 요청 및 응답 파라미터와 헤더 재정의	요청 헤더와 쿼리 문자열 및 경로는 물론 응답 헤더와 상태 코드에 대한 재정의 지원을 추가했습니다. 자세한 내용은 매핑 템플릿을 사용한 API의 요청 및 응답 파라미터와 헤더 재정의 단원을 참조하세요.	2018년 7월 12일
사용량 계획의 메서드 수준 조절	기본 메서드 단위 조절 한도의 설정은 물론 사용량 계획 설정에서 개별 API 메서드의 조절 한도 설정에 대한 지원을 추가했습니다. 이것들은 기존의 단계 설정에서 계정 수준의 조절 및 기본 메서드 수준의 조절 제한을 설정하던 것 외에 추가된 설정입니다. 자세한 내용은 처리량 향상을 위해 API 요청 조절 단원을 참조하세요.	2018년 7월 11일

이제 RSS에서 API Gateway 개발자 안내서 업데이트 알림 이용 가능	API Gateway 개발자 안내서의 HTML 버전이 이 문서 이력 페이지에 기록된 RSS 업데이트 피드를 현재 지원합니다. RSS 피드에는 2018년 6월 27일 이후의 업데이트가 포함됩니다. 이전에 발표한 업데이트도 이 페이지에서 이용할 수 있습니다. 피드를 구독하려면 상단 메뉴판에서 RSS 버튼을 누릅니다.	2018년 27월 6일

이전 업데이트

다음 표에서는 2018년 6월 27일 이전 API Gateway 개발자 안내서의 각 릴리스에서 변경된 중요 사항에 대해 설명합니다.

변경 사항	설명	변경 날짜
프라이빗 API	인터페이스 VPC 엔드포인트를 통해 노출되는 프라이빗 API에 대한 지원을 추가했습니다. 프라이빗 API로 가는 트래픽은 Amazon 네트워크를 벗어나지 않으며, 퍼블릭 인터넷과 격리됩니다.	2018년 6월 14일
교차 계정 Lambda 권한 부여자 그리고 통합 및 교차 계정 Amazon Cognito 사용자 풀 권한 부여자	다른 AWS 계정의 AWS Lambda 함수를 Lambda 권한 부여 함수 또는 API 통합 백엔드로 사용합니다. 또는 Amazon Cognito 사용자 풀을 권한 부여자로 사용합니다. 다른 계정은 Amazon API Gateway를 사용할 수 있는 모든 리전에 존재할 수 있습니다. 자세한 내용은 the section called “교차 계정 Lambda 권한 부여자 구성”, the section called “자습서: 교차 계정 Lambda 프록시 통합을 사용하여 API 빌드”, the section called “REST API를 위한 교차 계정 Amazon Cognito 권한 부여자 구성” 단원을 참조하십시오.	2018년 2월 4일
API에 대한 리소스 정책	API Gateway 리소스 정책을 사용하여 AWS 계정의 사용자가 API에 안전하게 액세스하도록 지원하거나 지정된 소	2018년 2월 4일

변경 사항	설명	변경 날짜
	스 IP 주소 범위나 CIDR 블록에서만 API가 호출되도록 허용할 수 있습니다. 자세한 내용은 the section called “API Gateway 리소스 정책 사용” 단원을 참조하세요.	
API 게이트웨이 리소스 태그 지정	API Gateway에서 API 요청 및 캐싱의 비용 할당을 위해 최대 50개의 태그로 API 단계에 태그를 지정합니다. 자세한 내용은 the section called “태그 설정” 단원을 참조하세요.	2017년 12월 19일
페이로드 압축 및 압축 해제	지원되는 콘텐츠 코딩 중 하나를 사용하여 압축된 페이로드가 포함된 API 호출을 활성화합니다. 본문 매핑 템플릿이 지정된 경우, 압축된 페이로드는 매핑 대상입니다. 자세한 내용은 the section called “콘텐츠 인코딩” 단원을 참조하세요.	2017년 12월 19일
사용자 지정 권한 부여자에서 생성된 API 키	사용자 지정 권한 부여자에서 API Gateway로 API 키를 반환하여 키가 필요한 API 메서드의 사용량 계획을 적용합니다. 자세한 내용은 the section called “API 키 소스 선택” 단원을 참조하세요.	2017년 12월 19일
OAuth 2 범위를 사용한 권한 부여	`COGNITO_USER_POOLS` 권한 부여자에 OAuth 2 범위를 사용하여 메서드 호출의 권한 부여를 활성화합니다. 자세한 내용은 the section called “REST API에 대한 권한 부여자로 Amazon Cognito 사용자 풀 사용” 단원을 참조하세요.	2017년 12월 14일
프라이빗 통합 및 VPC 링크	API Gateway 프라이빗 통합으로 API를 생성하여 클라이언트가 리소스를 통해 VPC 외부에서 Amazon VPC의 HTTP/HTTPS 리소스에 액세스할 수 있도록 합니다. VpcLink 자세한 내용은 the section called “자습서: 프라이빗 통합을 사용하여 API 빌드” 및 the section called “프라이빗 통합” 단원을 참조하십시오.	2017년 11월 30일
API 테스트를 위한 canary 배포	기존 API 배포에 canary 릴리스를 추가하여 새로운 버전의 API를 테스트하고 현재 버전은 같은 단계에서 계속 작동시킵니다. canary 릴리스의 스테이지 트래픽 비율을 설정하고 카나리아별 실행 및 액세스가 별도의 로그 로그에 기록되도록 할 수 있습니다. CloudWatch 자세한 설명은 the section called “Canary 릴리스 배포 설정” 섹션을 참조하세요.	2017년 11월 28일

변경 사항	설명	변경 날짜
액세스 로깅	$context 변수에서 추출된 데이터를 사용하여 원하는 형식으로 API에 대한 클라이언트 액세스를 기록합니다. 자세한 내용은 the section called “CloudWatch 로그” 단원을 참조하세요.	2017년 11월 21일
API의 Ruby SDK	API용 Ruby SDK를 생성하여 API 메서드 호출에 사용하십시오. 자세한 내용은 the section called “API의 Ruby SDK 생성” 및 the section called “API Gateway에서 생성한 REST API용 Ruby SDK 사용” 단원을 참조하십시오.	2017년 11월 20일
리전 API 엔드포인트	비모바일 클라이언트용 API를 생성하기 위해 리전 API 엔드포인트를 지정합니다. EC2 인스턴스와 같은 비모바일 클라이언트는 API가 배포되는 AWS 리전과 동일한 리전에서 실행됩니다. 엣지 최적화 API와 마찬가지로 리전 API에 대해 사용자 지정 도메인 이름을 생성할 수 있습니다. 자세한 내용은 the section called “리전 API 설정” 및 the section called “리전 사용자 지정 도메인 이름 설정” 단원을 참조하십시오.	2017년 11월 2일
사용자 지정 요청 권한 부여자	사용자 지정 요청 권한 부여자를 사용해 요청 파라미터의 사용자 인증 정보를 제공하여 API 메서드 호출을 승인합니다. 요청 파라미터는 헤더 및 쿼리 문자열 파라미터와 단계 및 컨텍스트 변수를 포함합니다. 자세한 내용은 API Gateway Lambda 권한 부여자 사용 단원을 참조하세요.	2017년 9월 15일
게이트웨이 응답 사용자 지정	통합 백엔드에 도달하지 못한 API 요청에 대해 API Gateway에서 생성한 게이트웨이 응답을 사용자 지정합니다. 사용자 지정된 게이트웨이 메시지는 필요한 CORS 헤더를 반환하는 것을 포함하여 API 관련 사용자 지정 오류 메시지를 호출자에게 제공하거나, 게이트웨이 응답 데이터를 외부 교환의 형식으로 변환할 수 있습니다. 자세한 내용은 오류 응답을 사용자 지정하도록 게이트웨이 응답 설정 단원을 참조하세요.	2017년 6월 6일

변경 사항	설명	변경 날짜
Lambda 사용자 지정 오류 속성을 메서드 응답 헤더에 매핑	Lambda에서 반환된 사용자 지정 오류 속성 각각을 `integration.response.body` 파라미터를 사용하여 메서드 응답 헤더 파라미터에 매핑하고, 실행 시간에 문자열화된 사용자 지정 오류 객체를 API Gateway에서 역직렬화합니다. 자세한 내용은 API Gateway에서 사용자 지정 Lambda 오류 처리 단원을 참조하세요.	2017년 6월 6일
조절 한도 증가	계정 수준의 안정적인 상태 요청 속도 제한을 초당 10,000개 요청(rps)으로, 버스트 제한을 5,000개 동시 요청으로 늘립니다. 자세한 내용은 처리량 향상을 위해 API 요청 조절 단원을 참조하세요.	2017년 6월 6일
메서드 요청 확인	API Gateway가 수신되는 요청을 확인할 수 있도록 API 레벨 또는 메서드 레벨에서 기본 요청 검사기를 구성합니다. API Gateway는 필요한 파라미터가 설정되어 있고 공백이 아닌지 확인하고, 해당하는 페이로드의 형식이 구성된 모델을 준수하는지 확인합니다. 자세한 내용은 API Gateway에서 요청 확인 사용 단원을 참조하세요.	2017년 11월 4일
ACM과 통합	API의 사용자 지정 도메인 이름에 대해 ACM 인증서를 사용합니다. ACM에서 인증서를 AWS Certificate Manager 생성하거나 기존 PEM 형식 인증서를 ACM으로 가져올 수 있습니다. 그런 다음 API에 대한 사용자 지정 도메인 이름을 설정할 때 인증서의 ARN을 참조합니다. 자세한 내용은 REST API에 대한 사용자 지정 도메인 이름 설정 단원을 참조하세요.	2017년 3월 9일
API의 Java SDK 생성 및 호출	API Gateway에서 API에 대한 Java SDK를 생성하고 SDK를 사용하여 Java 클라이언트에서 API를 호출하도록 합니다. 자세한 내용은 API Gateway에서 생성한 REST API용 Java SDK 사용 단원을 참조하세요.	2017년 1월 13일

변경 사항	설명	변경 날짜
과 AWS Marketplace통합	를 통해 사용량 요금제의 API를 SaaS 제품으로 판매하십시오. AWS Marketplace AWS Marketplace 를 사용하여 API의 범위를 확장합니다. 고객 청구를 대신 맡기세요. AWS Marketplace API Gateway에서 사용자 권한 부여 및 사용량 측정을 처리하도록 합니다. 자세한 내용은 API를 SaaS로 판매 단원을 참조하세요.	2016년 1월 12일
API에 대한 설명서 지원 활성화	API Gateway의 DocumentationPart리소스에 API 엔티티에 대한 설명서를 추가합니다. 컬렉션 `DocumentationPart` 인스턴스의 스냅샷을 API 단계와 연결하여 DocumentationVersion생성하십시오. 설명서 버전을 외부 파일(예: Swagger 파일)로 내보내서 API 설명서를 게시합니다. 자세한 내용은 REST API 문서화 단원을 참조하세요.	2016년 1월 12일
업데이트된 사용자 지정 권한 부여자	이제 사용자 지정 권한 부여자 Lambda 함수에서 호출자의 보안 주체 ID를 반환합니다. 또한 이 함수는 다른 정보를 `context` 맵 및 IAM 정책의 키-값 페어로 반환할 수 있습니다. 자세한 내용은 Amazon API Gateway Lambda 권한 부여자의 출력 단원을 참조하세요.	2016년 1월 12일
이진 페이로드 지원	요청 또는 응답의 바이너리 페이로드를 지원하도록 binaryMediaTypesAPI에 설정합니다. 통합에서 `contentHandling` 속성을 설정하거나 IntegrationResponse바이너리 페이로드를 네이티브 바이너리 블럽, Base64로 인코딩된 문자열 또는 수정 없이 패스스루로 처리할지 여부를 지정합니다. 자세한 설명은 REST API에 대한 이진 미디어 유형 작업 섹션을 참조하세요.	2016년 11월 17일
API의 프록시 리소스를 통해 HTTP 또는 Lambda 백엔드와 프록시 통합을 활성화.	`{proxy+}` 형식의 복잡한 경로 파라미터 및 catch-all ANY 메서드를 사용하여 프록시 리소스를 생성합니다. 프록시 리소스는 각각 HTTP 또는 Lambda 프록시 통합을 사용하여 HTTP 또는 Lambda 백엔드와 통합됩니다. 자세한 내용은 프록시 리소스를 사용하여 프록시 통합 설정 단원을 참조하세요.	2016년 9월 20일

변경 사항	설명	변경 날짜
하나 이상의 사용량 계획을 제공하여 고객에 대한 제품으로 선택된 API를 API Gateway에서 확장.	선택된 API 클라이언트가 합의된 요청 비율 및 할당량으로 지정된 API 단계에 액세스할 수 있도록 사용량 계획을 API Gateway에서 작성합니다. 자세한 내용은 API 키를 사용하여 사용량 계획 생성 및 사용 단원을 참조하세요.	2016년 8월 11일
Amazon Cognito의 사용자 풀로 메서드 레벨 권한 부여 사용	Amazon Cognito에 사용자 풀을 생성하고 고유 ID 공급자로 사용합니다. 사용자 풀에 등록된 사용자에 대한 액세스 권한을 부여하는 메서드 수준의 권한 부여자로 사용자 풀을 구성할 수 있습니다. 자세한 내용은 Amazon Cognito 사용자 풀을 권한 부여자로 사용하여 REST API에 대한 액세스 제어 단원을 참조하세요.	2016년 7월 28일
`AWS/ApiGateway` 네임스페이스에서 Amazon CloudWatch 지표 및 차원을 활성화합니다.	API Gateway 지표는 이제 의 CloudWatch 네임스페이스로 표준화되었습니다. `AWS/ApiGateway` API Gateway 콘솔과 Amazon CloudWatch 콘솔 모두에서 확인할 수 있습니다. 자세한 내용은 Amazon API Gateway 차원 및 지표 섹션을 참조하세요.	2016년 7월 28일
사용자 지정 도메인 이름에 대한 인증서 교체 활성화	인증서 교체를 통해 사용자 지정 도메인 이름에 대한 만료 인증서를 업로드하고 갱신할 수 있습니다. 자세한 내용은 ACM에 가져온 인증서 교체 단원을 참조하세요.	2016년 4월 27일
업데이트된 Amazon API Gateway 콘솔에 대한 설명서 변경 사항	업데이트된 API Gateway 콘솔을 사용하여 API를 생성 및 설정하는 방법을 알아봅니다. 자세한 내용은 자습서: 예제를 가져와 REST API 생성 및 자습서: HTTP 비 프록시 통합을 사용하여 REST API 빌드 단원을 참조하십시오.	2016년 4월 5일
외부 API 정의에서 API를 새로 생성하거나 기존 API를 업데이트하는 API 가져오기 기능 활성화.	API 가져오기 기능을 사용하면 API Gateway 확장이 포함된 Swagger 2.0으로 표시된 외부 API 정의를 업로드하여 API를 새로 생성하거나 기존 API를 업데이트할 수 있습니다. API 가져오기에 대한 자세한 내용은 OpenAPI를 사용하여 REST API 구성 단원을 참조하십시오.	2016년 4월 5일

변경 사항	설명	변경 날짜
원시 페이로드를 문자열로 액세스하는 `$input.body` 변수 및 JSON 문자열을 JSON 객체로 변환하는 `$util.parseJson()` 함수를 매핑 템플릿에 공개.	`$input.body` 및 `$util.parseJson()` 에 대한 자세한 내용은 API Gateway 매핑 템플릿과 액세스 로깅 변수 참조 단원을 참조하십시오.	2016년 4월 5일
메서드 레벨 캐시 무효화가 포함된 클라이언트 요청 활성화 및 요청 조절 관리 개선.	Flush API stage-level cache and invalidate individual cache entry. 자세한 내용은 API Gateway에서 API 단계 캐시 비우기 및 API Gateway 캐시 항목 무효화 단원을 참조하십시오. API 요청 조절을 관리하기 위한 콘솔 환경 개선. 자세한 내용은 처리량 향상을 위해 API 요청 조절 단원을 참조하세요.	2016년 3월 25일
사용자 지정 권한 부여를 사용하여 API Gateway API 활성화 및 호출	AWS Lambda 함수를 생성하고 구성하여 사용자 지정 인증을 구현하십시오. 이 함수에서 API Gateway API의 클라이언트 요청에 허용 또는 거부 권한을 부여하는 IAM 정책 문서를 반환합니다. 자세한 내용은 API Gateway Lambda 권한 부여자 사용 단원을 참조하세요.	2016년 2월 11일
Swagger 정의 파일 및 확장을 사용하여 API Gateway API 가져오기 및 내보내기	API Gateway 확장이 포함된 Swagger 사양을 사용하여 API Gateway API를 생성 및 업데이트합니다. API Gateway Importer를 사용하여 Swagger 정의를 가져옵니다. API Gateway 콘솔 또는 API Gateway 내보내기 API를 사용하여 API Gateway API를 Swagger 정의 파일로 내보냅니다. 자세한 내용은 OpenAPI를 사용하여 REST API 구성 및 API Gateway에서 REST API 내보내기 단원을 참조하십시오.	2015년 12월 18일
요청 또는 응답 본문이나 본문의 JSON 필드를 요청 또는 응답 파라미터에 매핑.	메서드 요청 본문 또는 해당 JSON 필드를 통합 요청의 경로, 쿼리 문자열 또는 헤더에 매핑합니다. 통합 응답 본문 또는 해당하는 JSON 필드를 요청 응답의 헤더에 매핑합니다. 자세한 내용은 Amazon API Gateway API 요청 및 응답 데이터 매핑 참조 단원을 참조하세요.	2015년 12월 18일

변경 사항	설명	변경 날짜
Amazon API Gateway에서 스테이지 변수 작업	구성 속성을 Amazon API Gateway의 API 배포 단계와 연관시키는 방법을 알아봅니다. 자세한 내용은 REST API 배포에 대한 스테이지 변수 설정 단원을 참조하세요.	2015년 11월 5일
방법: 메서드에 대한 CORS 활성화	이제 Amazon API Gateway에서 메서드에 대한 CORS(cross-origin 리소스 공유)를 더 쉽게 활성화할 수 있습니다. 자세한 내용은 REST API 리소스에 대한 CORS 활성화 단원을 참조하세요.	2015년 11월 3일
방법: 클라이언트 측 SSL 인증 사용	Amazon API Gateway를 사용하여 HTTP 백엔드에 대한 호출을 인증하기 위해 사용할 수 있는 SSL 인증서를 생성합니다. 자세한 내용은 백엔드 인증을 위한 SSL 인증서 생성 및 구성 단원을 참조하세요.	2015년 9월 22일
메서드 모의 통합	API를 Amazon API Gateway와 모의 통합하는 방법을 알아봅니다. 이 기능을 통해 개발자는 최종 통합 백엔드가 사전에 없어도 API Gateway에서 직접 API 응답을 생성할 수 있습니다.	2015년 9월 1일
Amazon Cognito 자격 증명 풀	요청을 Amazon Cognito 자격 증명으로 암호화한 경우 Amazon Cognito Identity에 대한 정보를 반환하도록 Amazon API Gateway에서 `$context` 변수의 범위를 확장했습니다. 또한 URL과 문자열에서 문자를 JavaScript 이스케이프하고 인코딩하기 위한 `$util` 변수를 추가했습니다. 자세한 내용은 API Gateway 매핑 템플릿과 액세스 로깅 변수 참조 단원을 참조하세요.	2015년 8월 28일
Swagger 통합	에서 Swagger 가져오기 도구를 사용하여 Swagger API 정의를 GitHub Amazon API Gateway로 가져올 수 있습니다. 가져오기 도구를 사용하여 API 및 메서드를 생성 및 배포하는 OpenAPI에 대한 API Gateway 확장 작업에 대해 알아봅니다. Swagger 가져오기 도구를 사용하여 기존 API를 업데이트할 수도 있습니다.	2015년 7월 21일
매핑 템플릿 참조	API Gateway 매핑 템플릿과 액세스 로깅 변수 참조 단원에서 `$input` 파라미터 및 해당 함수에 대해 알아봅니다.	2015년 7월 18일

변경 사항	설명	변경 날짜
최초 공개 릴리스	API Gateway 개발자 안내서가 처음으로 릴리스되었습니다.	2015년 7월 9일

AWS 용어집

최신 AWS 용어는 AWS 용어집 참조서의 AWS 용어집을 참조하세요.

기계 번역으로 제공되는 번역입니다. 제공된 번역과 원본 영어의 내용이 상충하는 경우에는 영어 버전이 우선합니다.