
Amazon API Gateway

Developer Guide



Amazon API Gateway: Developer Guide

Copyright © 2016 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

Table of Contents

What Is Amazon API Gateway?	1
API Gateway Concepts	2
Getting Started	4
Get Ready to Use API Gateway	4
Sign Up for AWS	5
Create an IAM User, Group or Role in Your AWS Account	5
Grant IAM Users Permissions to Access API Gateway Control and Execution Services	5
Next Step	6
Build an API to Expose an HTTP Endpoint	6
Create an API from an Example	7
Build the API Step By Step	14
Map Request Parameters	22
Map Response Payload	31
Build an API to Expose a Lambda Function	44
Step 1: Prerequisites	44
Step 2: Create an API	44
Step 3: Create a Resource	44
Step 4: Create Lambda Functions	44
Step 5: Create and Test a GET Method	48
Step 6: Create and Test a POST Method	49
Step 7: Deploy the API	50
Step 8: Test the API	50
Step 9: Clean Up	51
Next Steps	52
Create Lambda Invocation and Execution Roles	52
Build an API Using Proxy Integration and a Proxy Resource	54
Create and Test an API with HTTP Proxy Integration through a Proxy Resource	55
Create an API with Lambda Proxy Integration through a Proxy Resource	59
Create an AWS Service Proxy	64
Prerequisites	65
Step 1: Create the Resource	65
Step 2: Create the GET Method	66
Step 3: Create the AWS Service Proxy Execution Role	66
Step 4: Specify Method Settings and Test the Method	67
Step 5: Deploy the API	68
Step 6: Test the API	68
Step 7: Clean Up	69
Creating an API	70
Create an API in API Gateway	70
Create an API Using the API Gateway Console	70
Create an API Using the API Gateway Control Service API	71
Create an API Using the AWS SDK for API Gateway	71
Create an API Using the AWS CLI	71
Set up Method and Integration	71
Before Setting Up Methods and Integration	71
Configure How a Method Is Integrated with a Back End	72
Configure How a User Calls an API Method	74
Configure How Data Is Mapped between Method and Integration	75
Enable Support for Binary Payloads	78
Configure Mock Integration for a Method	94
Configure Proxy Integration for a Proxy Resource	97
After Setting Up Methods and Integration	105
Set Up Payload Mappings	105
Models	106
Mapping Templates	109

Tasks for Models and Mapping Templates	112
Create a Model	112
View a List of Models	113
Delete a Model	113
Photos Example	114
News Article Example	117
Sales Invoice Example	121
Employee Record Example	125
Request and Response Parameter-Mapping Reference	131
Map Data to Integration Request Parameters	131
Map Data to Method Response Headers	132
Transform Request and Response Bodies	133
Request and Response Payload-Mapping Reference	134
Accessing the \$context Variable	134
Accessing the \$input Variable	136
Accessing the \$stageVariables Variable	139
Accessing the \$util Variable	139
Integration Passthrough Behaviors	140
Import and Export API	141
Import an API	142
Export an API	145
API Gateway Extensions to Swagger	147
Controlling Access	160
Use IAM Permissions	160
Control Access to API Gateway with IAM Policies	160
Create and Attach a Policy to an IAM User	162
Statement Reference of IAM Policies for Managing API in API Gateway	163
Statement Reference of IAM Policies for Executing API in API Gateway	164
IAM Policy Examples for API Gateway APIs	165
IAM Policy Examples for API Execution Permissions	169
Enable CORS for a Resource	170
Prerequisites	171
Enable CORS Using the Console	171
Enable CORS Using Swagger Definition	172
Use API Gateway Custom Authorizers	174
API Gateway Custom Authorizers	174
Create the Custom Authorizer Lambda Function	175
Input to a Custom Authorizer	176
Output from an Amazon API Gateway Custom Authorizer	177
Configure Custom Authorizer	178
Call an API with Custom authorization	180
Use Amazon Cognito Your User Pool	182
Create a User Pool	183
Integrate an API with a User Pool	183
Call an API Integrated with a User Pool	186
Use Client-Side SSL Certificates	186
Generate a Client Certificate Using the API Gateway Console	187
Configure an API to Use SSL Certificates	187
Test Invoke	188
Configure Back End to Authenticate API	189
Supported Certificate Authorities for HTTP and HTTP Proxy Integration	189
Use API Gateway Usage Plans	212
What Is a Usage Plan?	212
How to Configure a Usage Plan?	212
Set Up API Keys Using the API Gateway Console	213
Create and Configure Usage Plans with the API Gateway Console	216
Set Up API Keys Using the API Gateway REST API	219
Create and Configure Usage Plans with the API Gateway REST API	220

API Gateway API Key File Format	222
Use API Keys without Usage Plans Enabled	223
Maintaining an API	227
View a List of APIs	227
Prerequisites	227
View a List of APIs with the API Gateway Console	227
Delete an API	227
Prerequisites	228
Delete an API with the API Gateway Console	228
Delete a Resource	228
Delete a Resource with the API Gateway Console	228
View a Methods List	228
Prerequisites	228
View a Methods List with the API Gateway Console	229
Delete a Method	229
Delete a Method with the API Gateway Console	229
Deploying an API	230
Deploy an API with the API Gateway Console	230
Prerequisites	230
Deploy an API with the API Gateway Console	230
Update deployment configuration with the API Gateway Console	231
Change a Stage to Use a Different Deployment with the API Gateway Console	231
Deploy an API in Stages	232
Create a Stage	232
View a List of Stages	232
Set Up a Stage	233
Delete a Stage	235
Manage API Request Throttling	236
Account-Level Throttling	236
Stage-Level and Method-Level Throttling	236
Enable API Caching	236
API Caching Overview	237
Enable API Caching	237
Override Stage Caching for Method Caching	238
Use Method/Integration Parameters as Cache Keys	239
Flush the API Stage Cache in API Gateway	240
Invalidate an API Gateway Cache Entry	240
Manage API Deployment with Stage Variables	242
Use Cases	242
Examples	242
Set Stage Variables	243
Use Stage Variables	246
Stage Variables Reference	253
Generate and Use an SDK for an API	254
Use the API Gateway Console to Generate the SDKs of an API	254
Use an Android SDK generated by API Gateway	266
Use iOS SDK Generated by API Gateway in Objective-C or Swift	268
Use a JavaScript SDK Generated by API Gateway	278
Use a Custom Domain Name	280
Prerequisites	281
Set Up a Custom Domain Name for an API Gateway API	281
Specify API Mappings for a Custom Domain Name	284
Base Path Examples of API Mappings for a Custom Domain Name	284
Upload and Renew an Expiring Certificate	285
Call Your API with Custom Domain Names	286
Documenting an API	287
Representation of API Documentation in API Gateway	287
Documentation Parts	288

Documentation Versions	294
Document an API Using the API Gateway Console	294
Document the API Entity	295
Document a RESOURCE Entity	297
Document a METHOD Entity	297
Document a QUERY_PARAMETER Entity	298
Document a PATH_PARAMETER Entity	299
Document a REQUEST_HEADER Entity	300
Document a REQUEST_BODY Entity	300
Document a RESPONSE Entity	300
Document a RESPONSE_HEADER Entity	301
Document a RESPONSE_BODY Entity	301
Document a MODEL Entity	301
Document an AUTHORIZER Entity	302
Document an API Using the API Gateway REST API	302
Document the API Entity	303
Document a RESOURCE Entity	304
Document a METHOD Entity	307
Document a QUERY_PARAMETER Entity	309
Document a PATH_PARAMETER Entity	311
Document a REQUEST_BODY Entity	312
Document a REQUEST_HEADER Entity	313
Document a RESPONSE Entity	314
Document a RESPONSE_HEADER Entity	314
Document an AUTHORIZER Entity	315
Document a MODEL Entity	317
Update Documentation Parts	318
List Documentation Parts	319
Publish API Documentation	319
Create a Documentation Snapshot and Associate it with an API Stage	319
Create a Documentation Snapshot	320
Update a Documentation Snapshot	320
Get a Documentation Snapshot	321
Associate a Documentation Snapshot with an API Stage	321
Download a Documentation Snapshot Associated with a Stage	322
Import API Documentation	325
Importing Documentation Parts Using the API Gateway REST API	325
Importing Documentation Parts Using the API Gateway Console	327
Control Access to API Documentation	327
Calling a Deployed API	329
Prerequisites	329
Obtain an API's Invoke URL in the API Gateway Console	330
Test a Method Using the Console	330
Prerequisites	330
Test a Method with the API Gateway Console	330
Use Postman to Test an API	331
Monitoring and Troubleshooting	332
Log API Management Calls with CloudTrail	332
API Gateway Information in CloudTrail	333
Understanding API Gateway Log File Entries	333
Monitor API execution with Amazon CloudWatch	334
Amazon API Gateway Dimensions and Metrics	334
View Metrics with the API Dashboard	336
View Metrics in the CloudWatch Console	336
Monitoring Tools in AWS	337
Selling an API as SaaS	339
Initialize AWS Marketplace Integration with API Gateway	339
List a Product on AWS Marketplace	339

Create the Metering Role	340
Associate Usage Plan with AWS Marketplace Product	340
Handle Customer Subscription to Usage Plans	341
Authorize a Customer to Access a Usage Plan	341
Associate a Customer with an API Key	342
Tutorials	343
Create an API as a Lambda Proxy	343
Set Up an IAM Role and Policy for an API to Invoke Lambda Functions	345
Create a Lambda Function in the Back End	346
Create API Resources for the Lambda Function	347
Create a GET Method with Query Strings to Call the Lambda Function	347
Create a POST Method with a JSON Payload to Call the Lambda Function	350
Create a GET Method with Path Parameters to Call the Lambda Function	352
Swagger Definitions of a Sample API as Lambda Proxy	356
Create an API as an Amazon S3 Proxy	359
Set Up IAM Permissions for the API to Invoke Amazon S3 Actions	360
Create API Resources to Represent Amazon S3 Resources	361
Expose an API Method to List the Caller's Amazon S3 Buckets	362
Expose API Methods to Access an Amazon S3 Bucket	369
Expose API Methods to Access an Amazon S3 Object in a Bucket	372
Call the API Using a REST API Client	374
Swagger Definitions of a Sample API as an Amazon S3 Proxy	378
Create an API as an Amazon Kinesis Proxy	388
Create an IAM Role and Policy for the API to Access Amazon Kinesis	390
Start to Create an API as an Amazon Kinesis Proxy	391
List Streams in Amazon Kinesis	391
Create, Describe, and Delete a Stream in Amazon Kinesis	393
Get Records from and Add Records to a Stream in Amazon Kinesis	400
Swagger Definitions of an API as a Kinesis Proxy	410
API Gateway REST API	418
Limits, Pricing and Known Issues	419
API Gateway Limits	419
API Gateway Limits for Configuring and Running an API	419
API Gateway Limits for Creating, Deploying and Managing an API	420
API Gateway Pricing	421
Known Issues	421
Document History	423
AWS Glossary	427

What Is Amazon API Gateway?

Amazon API Gateway supports the following two major functionalities:

- It lets you create, manage and host a RESTful API to expose AWS Lambda functions, HTTP endpoints as well as other services from the AWS family including, but not limited to, Amazon DynamoDB, Amazon S3 Amazon Kinesis. You can use this feature through the API Gateway REST API requests and responses, the API Gateway console, AWS Command-Line Interface (CLI), or an API Gateway SDK of supported platforms/languages. This feature is sometimes referred to as the API Gateway control service.
- It lets you or 3rd-party app developer to call a deployed API to access the integrated back-end features, using standard HTTP protocols or a platform- or language-specific SDK generated by API Gateway for the API. This feature is sometimes known as the API Gateway execution service.

The API you create in API Gateway consists of a set of resources and methods. A resource is a logical entity that can be accessed through a resource path using the API. A resource can have one or more operations that are defined by appropriate HTTP verbs such as GET, POST, and DELETE. A combination of a resource path and an operation identify a method in the API. Each method corresponds to a REST API request submitted by the user of your API and the corresponding response returned to the user. API Gateway integrates the method with a targeted back end by mapping the method request to an integration request acceptable by the back end and then mapping the integration response from the back end to the method response returned to the user. As an API developer, you can configure how methods are mapped to integrations and vice versa by stipulating what parameters to use and specifying mapping templates to transform payloads of given data models.

You can create an API by using the API Gateway management console, described in [Getting Started \(p. 4\)](#), or by using the API Gateway [API Gateway REST API \(p. 418\)](#). In addition, you can integrate API creation with [AWS CloudFormation templates](#) or [API Gateway Extensions to Swagger \(p. 147\)](#). For a list of regions where API Gateway is available, as well as the associated control service endpoints, see [Regions and Endpoints](#).

API Gateway helps developers deliver robust, secure, and scalable mobile and web application back ends. API Gateway allows developers to securely connect mobile and web applications to business logic hosted on AWS Lambda, APIs hosted on Amazon EC2, or other publicly addressable web services hosted inside or outside of AWS. With API Gateway, developers can create and operate APIs for their back-end services without developing and maintaining infrastructure to handle authorization and access control, traffic management, monitoring and analytics, version management, and software development kit (SDK) generation.

API Gateway is designed for web and mobile developers who want to provide secure, reliable access to back-end APIs for access from mobile apps, web apps, and server apps that are built internally or by

third-party ecosystem partners. The business logic behind the APIs can either be provided by a publicly accessible endpoint that API Gateway proxies call, or it can be entirely run as a Lambda function.

To better understand the terminology used in this documentation, you may find it useful to peruse the [API Gateway Concepts \(p. 2\)](#) section.

Amazon API Gateway Concepts

API Gateway	An AWS service that 1) supports creating, deploying and managing a RESTful application programming interface (API) to expose back-end HTTP endpoints, AWS Lambda function, or other AWS services; and 2) invoking exposed API methods through the front-end HTTP endpoints.
API Gateway API	A collection of resources and methods that are integrated with back-end HTTP endpoints, Lambda functions or other AWS services and can be deployed in one or more stages. API methods are invoked through front-end HTTP endpoints that can be associated with a registered custom domain names. Permissions to invoke a method can be granted using IAM roles and policies or API Gateway custom authorizers. An API can present a certificate to be authenticated by the back end. Typically, API resources are organized in a resource tree according to the application logic. Each API resource can expose one or more API methods that must have unique HTTP verbs supported by API Gateway.
API developer or API owner	An AWS account that owns an API Gateway deployment (for example, a service provider who also supports programmatic access.)
App developer or client developer	An app creator who may or may not have an AWS account and interacts with the API deployed by the API developer. An app developer can be represented by an API Key.
App user, end user or client endpoint	An entity that uses the application built by an app developer that interacts with APIs in Amazon API Gateway. An app user can be represented by an Amazon Cognito identity or a bearer token.
API Key	An alphanumeric string, which can be generated by API Gateway on behalf of an API owner or imported from an external source such as a CSV file, is used to identify an app developer of the API. An API owner can use API keys to permit or deny access of given APIs based on the apps in use.
API Deployment and stage	An API deployment is a point-in-time snapshot of the API Gateway API resources and methods. For a deployment to be accessible for invocation by a client, it must be associated with one or more stages. A stage is a logical reference to a life-cycle status of your API (e.g., 'dev', 'prod', 'beta', 'v2'). The identifier of an API stage consists of an API ID and stage name.
Method request	The public interface of an API method in API Gateway that defines the parameters and body that an app developer must send in the requests to access the back end through the API.
Integration request	An API Gateway internal interface that defines how API Gateway maps the parameters and body of a method request into the formats required by the back end.

Integration response	An API Gateway internal interface that defines how API Gateway maps data. The integration response includes the status codes, headers, and payload that are received from the back end into the formats defined for an app developer.
Method response	The public interface of an API that defines the status codes, headers, and body models that an app developer should expect from API Gateway.
Proxy Integration	A simplified API Gateway integration configuration. You can set up a proxy integration as an HTTP proxy integration type or a Lambda proxy integration type. For the HTTP proxy integration API Gateway passes the entire request and response between the front end and an HTTP back end. For the Lambda proxy integration API Gateway sends the entire request as an input to a back-end Lambda function and transforms the Lambda function output to a front-end HTTP response. The proxy integration is most commonly used with a proxy resource, which is represented by a greedy path variable (e.g., {proxy+}) combined with a catch-all ANY method.
Mapping template	Scripts, expressed in Velocity Template Language (VTL) , to transform a request body from the front-end data format to the back-end data format or to transform a response body from the back-end data format to the front-end data format. Mapping templates are specified in the integration request or integration response and they can reference data made available at run time in the forms of context and stage variables. An identity transformation is referred to as pass-through in which a payload is passed as-is from the client to the back end for a request and from the back end to the client for a response.
Model	Data schema specifying the data structure of a request or response payload. It is required for generating strongly typed SDK of an API, used for validating payload, and convenient for generating a sample mapping template to initiate creation of a production mapping template. Although useful, a model is not required for creating a mapping template.
Usage plan	A usage plan provides selected API clients with access to one or more deployed APIs with configurable throttling and quota limits enforced on individual client API keys.

Getting Started with Amazon API Gateway

The following walkthroughs include hands-on exercises, using the API Gateway console, to help you learn about API Gateway.

Topics

- [Get Ready to Use Amazon API Gateway \(p. 4\)](#)
- [Build an API Gateway API to Expose an HTTP Endpoint \(p. 6\)](#)
- [Build an API to Expose a Lambda Function \(p. 44\)](#)
- [Build an API Gateway API Using Proxy Integration and a Proxy Resource \(p. 54\)](#)
- [Create an AWS Service Proxy for Amazon SNS \(p. 64\)](#)

Get Ready to Use Amazon API Gateway

Before using API Gateway for the first time, you must have an AWS account set up. To create, configure and deploy an API in API Gateway, you must have appropriate IAM policy provisioned with permissible access rights to the API Gateway control service. To permit your API clients to invoke your API in API Gateway, you must set up the right IAM policy to allow the clients to call the API Gateway execution service. To allow API Gateway to invoke an AWS service in the back end, API Gateway must have permissions to assume the roles required to call the back-end AWS service. When an API Gateway API is set up to use AWS IAM roles and policies to control client access, the client must sign API Gateway API requests with [Signature Version 4](#).

Understanding of these topics are important to use API Gateway and to follow the tutorials and instructions presented here. This section provides brief discussions of or quick references to these topics.

Topics

- [Sign Up for AWS \(p. 5\)](#)

- [Create an IAM User, Group or Role in Your AWS Account \(p. 5\)](#)
- [Grant IAM Users Permissions to Access API Gateway Control and Execution Services \(p. 5\)](#)
- [Next Step \(p. 6\)](#)

Sign Up for AWS

Go to <http://aws.amazon.com/>, choose **Create an AWS Account**, and follow the instructions therein.

Create an IAM User, Group or Role in Your AWS Account

For better security practices, you should refrain from using your AWS root account to access API Gateway. Instead, create a new AWS Identity and Access Management (IAM) user or use an existing one in your AWS account, and then access API Gateway with that IAM user credentials.

To manage access for a user, you can create an IAM user, grant the user API Gateway access permissions. To create a new IAM user, see [Creating an IAM User](#).

To manage access for a group of users, you can create an IAM group, grant the group API Gateway access permissions and then add one or more IAM users to the group. To create an IAM group, see [Creating IAM Groups](#).

To delegate access to specific users, apps or service, you can create an IAM role, add the specified users or groups to the role, and grant the users or groups API Gateway access permissions. To create an IAM role, see [Creating IAM Roles](#).

When setting up your API, you need to specify the ARN of an IAM role to control access the API's methods. Make sure that this is ready when creating an API.

Grant IAM Users Permissions to Access API Gateway Control and Execution Services

In AWS, access permissions are stated as [policies](#). A policy created by AWS is a managed policy and one created by a user is an inline policy.

For the API Gateway control service, the managed policy of **AmazonAPIGatewayAdministrator** (`arn:aws:iam::aws:policy/AmazonAPIGatewayAdministrator`) grants the full access to create, configure and deploy an API in API Gateway:

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

To grant the stated permissions to a user, attach the policy to the user, a group containing the user. To attach a policy, see [Attaching Managed Policies](#).

Attaching the preceding policy to an IAM user provides the user with access to all API Gateway control service actions and resources associated with the AWS account. To learn how to restrict IAM users to a limited set of API Gateway control service actions and resources, see [Use IAM Permissions \(p. 160\)](#).

For the API Gateway execution service, the managed policy of **AmazonAPIGatewayInvokeFullAccess** (`arn:aws:iam::aws:policy/AmazonAPIGatewayInvokeFullAccess`) provides full access to invoke an API in API Gateway:

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

Attaching the preceding policy to an IAM user provides the user with access to all API Gateway execution service actions and resources associated with the AWS account. To learn how to restrict IAM users to a limited set of API Gateway execution service actions and resources, see [Use IAM Permissions \(p. 160\)](#).

To grant the state permissions to a user, attach the policy to the user, a group containing the user. To attach a policy, see [Attaching Managed Policies](#).

In this documentation, we will use managed policies, whenever possible. To create and use inline policies, see [Working with Inline Policies](#).

Note

To complete the steps above, you must have permission to create the IAM policy and attach it to the desired IAM user.

Next Step

You are now ready to start using API Gateway. See [Create an API Gateway API from an Example \(p. 7\)](#).

Build an API Gateway API to Expose an HTTP Endpoint

You can create an API in the Amazon API Gateway console from the ground up. In essence, you use the console as an API design studio to scope the API features, to experiment with its behaviors, to build the API, and to deploy your API in stages.

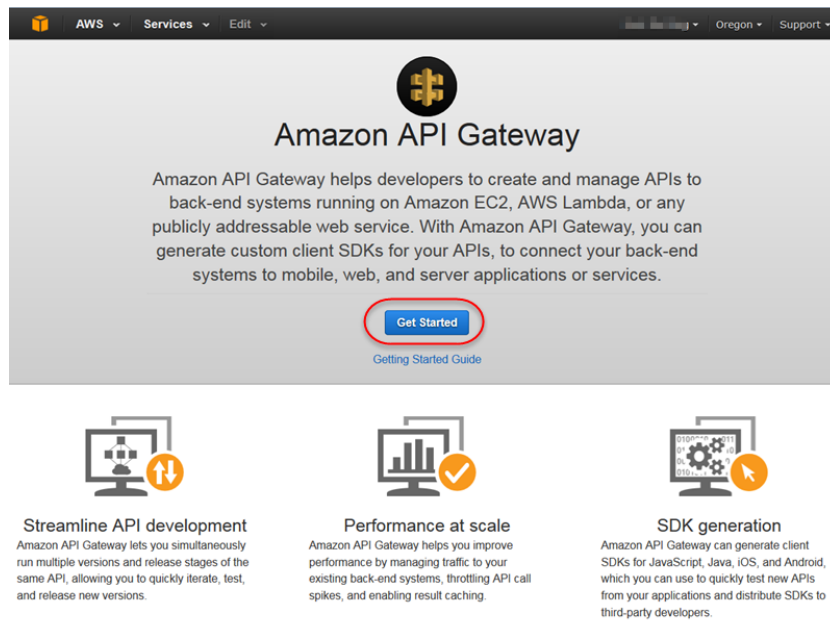
Topics

- [Create an API Gateway API from an Example](#) (p. 7)
- [Build the API Step By Step](#) (p. 14)
- [Map Request Parameters for an API Gateway API](#) (p. 22)
- [Map Response Payload](#) (p. 31)

Create an API Gateway API from an Example

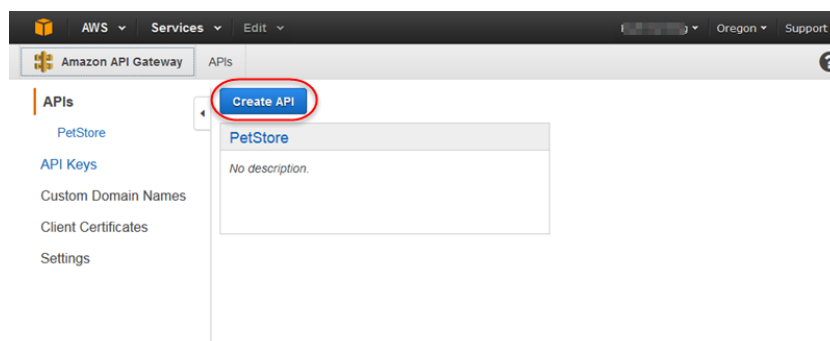
The Amazon API Gateway console now provides an option for you to create an API Gateway API by example, with helpful hints provided along the way. If you are new to API Gateway, you may find it useful as a learning tool. The following steps walk you through using this create-by-example option to create and test the example API.

1. Do one of the following:
 - a. For the first API in your account, choose **Get Started** from the API Gateway console welcome page:

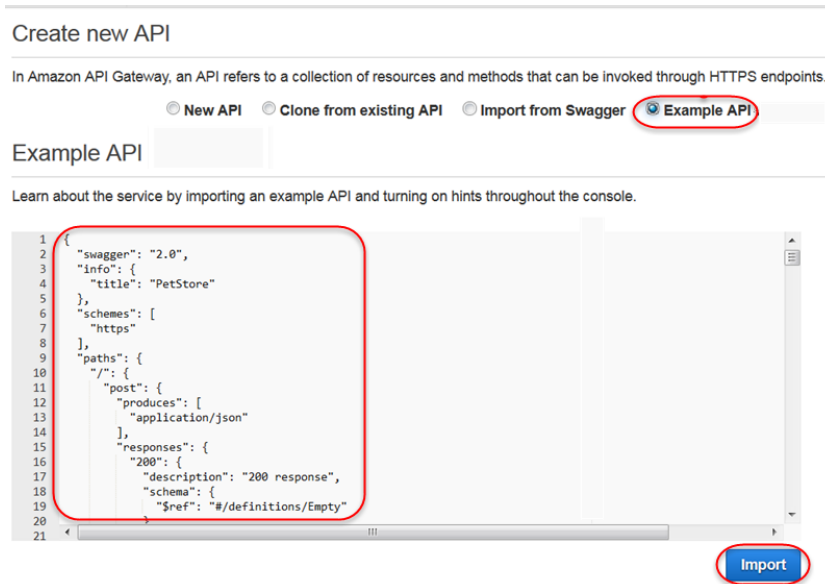


If prompted with a modal dialog box containing hints at a stage of the process, choose **OK** to close the modal dialog and continue.

- b. For your next API, choose **Create API** from the API Gateway **APIs** home page:

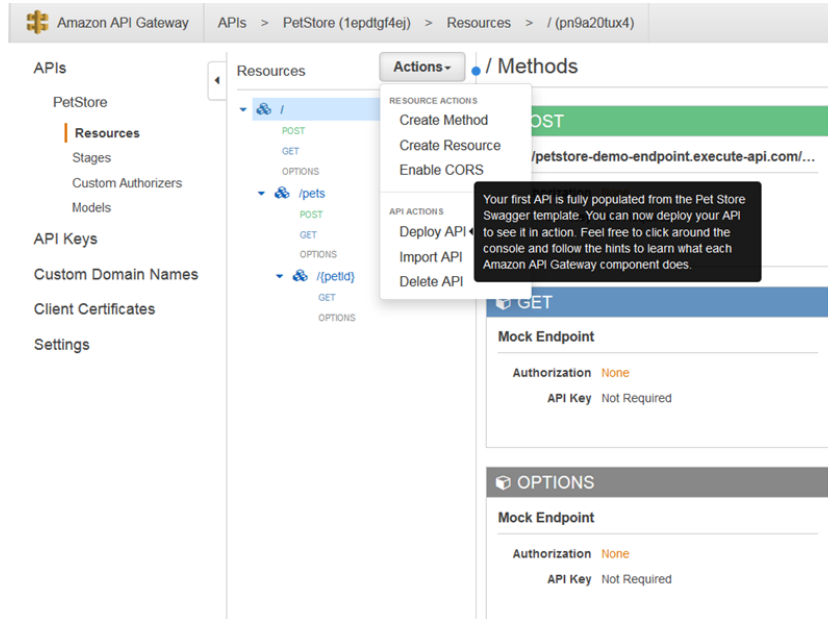


- Under **Create new API**, select **Examples API** and then choose **Import** to create the example API. For your first API, the API Gateway console will start with this option as default.



You can scroll down the Swagger definition for details of this example API before choosing **Import**.

- The resulting display shows the newly created API:



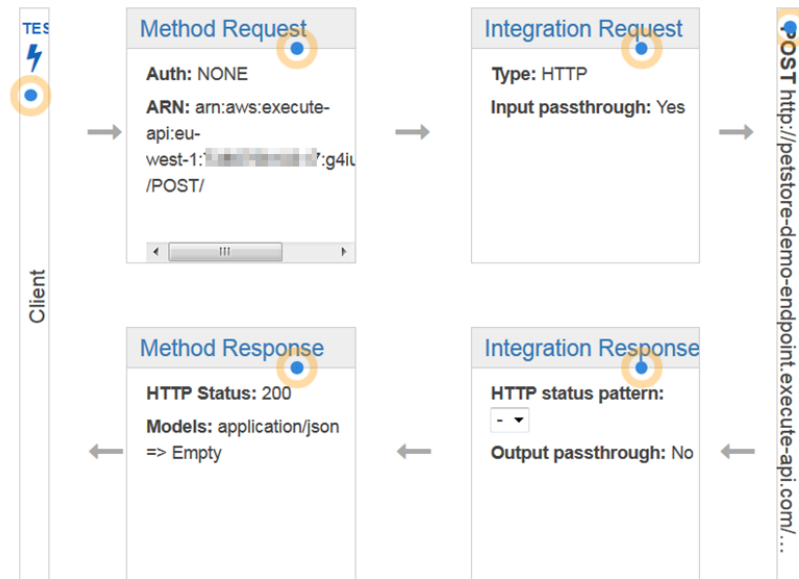
The API Gateway navigation pane on the left shows your available APIs, any API keys, custom domain names and client certificates that you created for your APIs, as well as the settings for logging your APIs' performance metrics. API-specific resources, deployment, custom authorizers and payload-mapping data models are organized under individual APIs.

The **Resources** pane in the middle shows the structure of the selected API as a tree of nodes. API methods defined on each resource are edges of the tree. When a resource is selected, all of

its methods are listed in the **Methods** pane on the right. Displayed under each method is a brief summary of the method, including its endpoint URL, authorization type, and API Key requirement.

4. To view the details of a method, to modify its set-up, or to test the method invocation, choose the method name from either the method list or the resource tree.

/ - POST - Method Execution



The resulting **Method Execution** pane for the chosen method presents a logical view of the method's structure and behaviors: a client accesses a back-end service by interacting with the API through **Method Request**. API Gateway translates the client request, if necessary, into the form acceptable to the back end before forwarding the request to the back end. The transformed request is known as the integration request and is depicted by **Integration Request** in the display. Similarly, the response from the back end goes through **Integration Response** and then **Method Response** before being received by the client. Again, if necessary, API Gateway maps the response from the form shaped in the back end to a form expected by the client.

For the `POST` method on this API's root (`/`) resource, the method's integration request shows that the method is integrated with the endpoint of `http://petstore-demo-endpoint.execute-api.com/petstore/pets` in the back end. The method request payload will be passed through to the integration request without modification. The `GET /` method request uses the `MOCK` integration type and is not tied to any endpoint in the back end. When the method is called, the API Gateway simply accepts the request and immediately returns a response, by way of from **Integration Response** to **Method Response**. You can use the mock integration to test an API without requiring a back-end endpoint. You can also use it to serve a local response. In fact, the example API uses it to return a local HTML page as the home page of the API. It uses a mapping template to generate the home page in **Integration Response**.

As an API developer, you control the behaviors of your API's front-end interactions by configuring the method request and a method response. You control the behaviors of your API's back-end interactions by setting up the integration request and integration response. They involve data mappings between a method and its corresponding integration. We will cover the method setup in [Build an API Gateway API to Expose an HTTP Endpoint \(p. 6\)](#). For now, we focus on testing the API to provide an end-to-end user experience.

5. Choose **Test** shown on **Client** (as shown in the previous image) to start testing. Enter the following `{"type": "dog", "price": 249.99}` payload into the **Request Body** before choosing the **Test** button.

[← Method Execution](#) / - POST - Method Test

Make a test call to your method with the provided input

Path
No path parameters exist for this resource. You can define path parameters by using the syntax `{myPathParam}` in a resource path.

Query Strings
No query string parameters exist for this method. You can add them via Method Request.


Headers
No header parameters exist for this method. You can add them via Method Request.

Stage Variables
No [stage variables](#) exist for this method.

Client Certificate
No client certificates have been generated.

Request Body

```
1 [{"type": "dog", "price": 249.99}]
```



The input specifies the attributes of the pet that we wish to add to the list of pets on the PetStore website.

6. The results display as follows:

Request: /

Status: 200

Latency: 1445 ms

Response Body

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

Response Headers

```
{"Access-Control-Allow-Origin":"*","Content-Type":"application/json"}
```

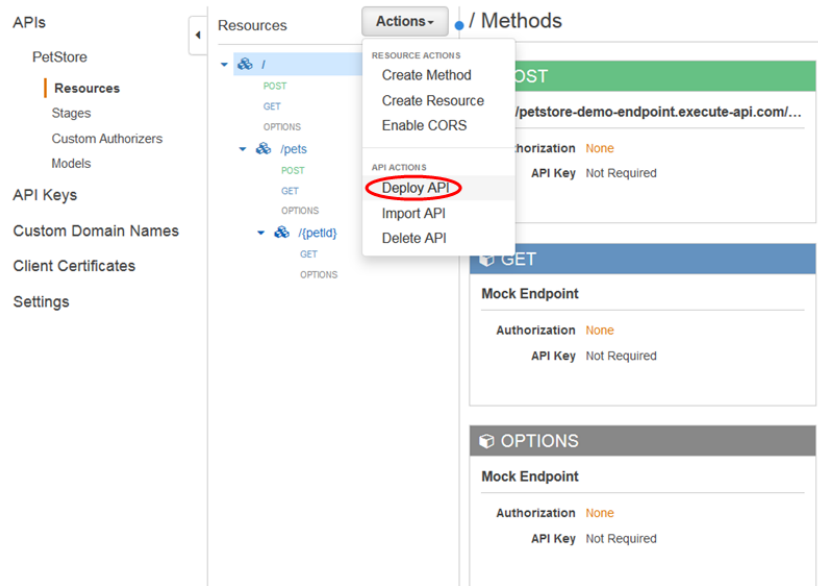
Logs

```
Execution log for request test-request
Mon Apr 04 04:59:05 UTC 2016 : Starting execution for request: test-invoke-request
Mon Apr 04 04:59:05 UTC 2016 : HTTP Method: POST, Resource Path: /
Mon Apr 04 04:59:05 UTC 2016 : Method request path: {}
Mon Apr 04 04:59:05 UTC 2016 : Method request query string: {}
Mon Apr 04 04:59:05 UTC 2016 : Method request headers: {}
Mon Apr 04 04:59:05 UTC 2016 : Method request body before transformations: {"type": "dog", "price": 249.99}
Mon Apr 04 04:59:05 UTC 2016 : Endpoint request URI: http://petstore-demo-endpoint.execute-api.com/petstore/pets
Mon Apr 04 04:59:05 UTC 2016 : Endpoint request headers: {x-amzn-apigateway-api-id=g4iukm23bf, Accept=application/json, User-Agent=AmazonAPIGateway_g4iukm23bf, Content-Type=application/json}
Mon Apr 04 04:59:05 UTC 2016 : Endpoint request body after transformations: {"type": "dog", "price": 249.99}
Mon Apr 04 04:59:06 UTC 2016 : Endpoint response body before transformations: {
  "pet": {
    "type": "dog",
    "price": 249.99
  },
  "message": "success"
}
Mon Apr 04 04:59:06 UTC 2016 : Endpoint response headers: {date=Mon, 04 Apr 2016 04:59:06 GMT, content-length=81, x-powered-by=Express, content-type=application/json; charset=utf-8, connection=keep-alive}
Mon Apr 04 04:59:06 UTC 2016 : Method response body after transformations: {
  "pet": {
    "type": "dog",
    "price": 249.99
  },
  "message": "success"
}
Mon Apr 04 04:59:06 UTC 2016 : Method response headers: {Access-Control-Allow-Origin=*, Content-Type=application/json}
Mon Apr 04 04:59:06 UTC 2016 : Successfully completed execution
Mon Apr 04 04:59:06 UTC 2016 : Method completed with status: 200
```

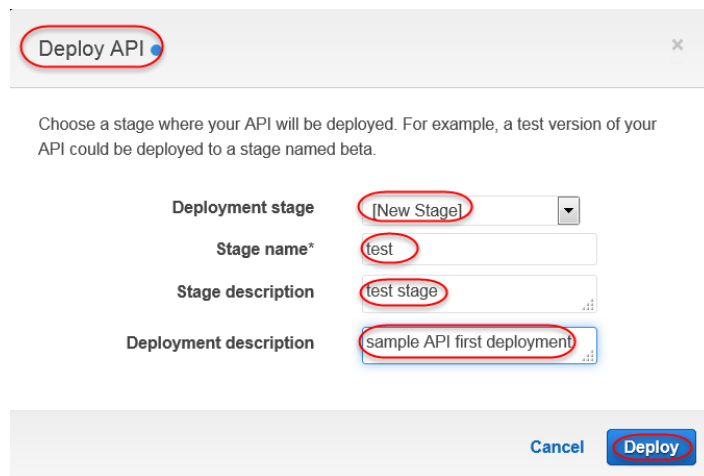
The **Logs** entry of the output shows the state changes from the method request to the integration request and from the integration response to the method response. This can be useful for troubleshooting any mapping errors that cause the request to fail. In this example, no mapping is applied: the method request is identical to the integration request and the integration response is the same as the method response.

To test the API using a client other than the API Gateway test-invoke-request feature, you must first deploy the API to a stage.

7. To deploy the sample API, select the **PetStore** API and the root / resource, and then choose **Deploy API** from the **Actions** menu.



In **Deploy API**, for **Deployment stage**, choose **[New Stage]** because this is the first deployment of the API. Type a name (e.g., `test`) in **Stage name** and, optionally, type descriptions in **Stage description** and **Deployment description**. Choose **Deploy**.



In the resulting **Stage Editor** pane, **Invoke URL** displays the URL to invoke the API's `GET /` method request.

protocols. For an example of such a request, see [Build an API Gateway API to Expose an HTTP Endpoint \(p. 6\)](#).

See Also

[Use API Gateway Custom Authorizers \(p. 174\)](#), [Deploying an API \(p. 230\)](#)

Build the API Step By Step

This section walks you through the steps to create resources, expose methods on a resource, configure a method to achieve the desired API behaviors, and to test and deploy the API.

1. From **Create new API**, select **New API**, type a name in **API Name**, optionally add a description in **Description**, and then choose **Create API**.

Create new API

In Amazon API Gateway, an API refers to a collection of resources and methods that can be invoked through HTTPS endpoints.

New API Clone from existing API Import from Swagger Example API

Name and description

Choose a friendly name and description for your API.

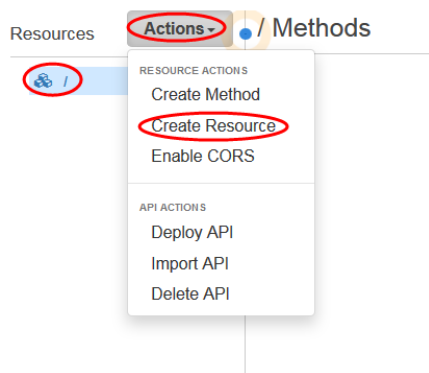
API name*

Description

* Required

As a result, an empty API is created. The **Resources** tree shows the root resource (/) without any methods. In this exercise, we will build the API with the HTTP integration of the PetStore demo website (<http://petstore-demo-endpoint.execute-api.com>.) For illustration purposes, we will create a /pets resource as a child of the root and expose a GET method on this resource for a client to retrieve a list of available Pets items from the PetStore website.

2. To create the /pets resource, select the root, choose **Actions** and then choose **Create Resource**.



Type `Pets` in **Resource Name**, leave the **Resource Path** value as given, and choose **Create Resource**.

New Child Resource

Use this page to create a new child resource for your resource.

Resource Name*

Resource Path*

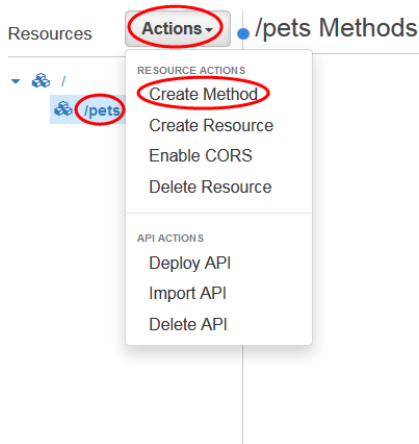
You can add path parameters using brackets. For example, the resource path **{username}** represents a path parameter called 'username'.

* Required

Cancel

Create Resource

3. To expose a GET method on the `/pets` resource, choose **Actions** and then **Create Method**.



Choose **GET** from the list under the `/pets` resource node and choose the checkmark icon to finish creating the method.



Note

Other options for an API method include:

- **POST**, primarily used to create child resources.
- **PUT**, primarily used to update existing resources (and, although not recommended, can be used to create child resources).
- **DELETE**, used to delete resources.
- **PATCH**, used to update resources.

- **HEAD**, primarily used in testing scenarios. It is the same as GET but does not return the resource representation.
- **OPTIONS**, which can be used by callers to get information about available communication options for the target service.

The method created is not yet integrated with the back end. The next step sets this up.

4. In the method's **Setup** pane, select **HTTP** for **Integration type**, select **GET** from the **HTTP method** drop-down list, type `http://petstore-demo-endpoint.execute-api.com/petstore/pets` as the **Endpoint URL** value, and then choose **Save**.

Note

For the integration request's **HTTP method**, you must choose one supported by the back end. For `HTTP` or `Mock` integration, it makes sense that the method request and the integration request use the same HTTP verb. For other integration types the method request will likely use an HTTP verb different from the integration request. For example, to call a Lambda function, the integration request must use `POST` to invoke the function, whereas the method request may use any HTTP verb depending on the logic of the Lambda function.

/pets - GET - Setup

Choose the integration point for your new method.

Integration type Lambda Function ⓘ **HTTP** ⓘ Mock ⓘ AWS Service ⓘ

Use HTTP Proxy integration ⓘ

HTTP method GET

Endpoint URL ndpoint.execute-api.com/petstore/pets

Save

When the method setup finishes, you are presented with the **Method Execution** pane, where you can further configure the method request to add query string or custom header parameters. You can also update the integration request to map input data from the method request to the format required by the back end.

The PetStore website allows you to retrieve a list of `Pet` items by the pet type (e.g., "Dog" or "Cat") on a given page. It uses the `type` and `page` query string parameters to accept such input. As such, we must add the query string parameters to the method request and map them into the corresponding query strings of the integration request.

5. In the GET method's **Method Execution** pane, choose **Method Request**, select `AWS_IAM` for **Authorization**, expand the **URL Query String Parameters** section, and choose **Add query string** to create two query string parameters named `type` and `page`. Choose the checkmark icon to save the newly added query string parameters.

[← Method Execution](#) /pets - GET - Method Request

Provide information about this method's authorization settings and the parameters it can receive.

Authorization Settings

Authorization **AWS_IAM** ⓘ

API Key Required false ✎

▼ URL Query String Parameters

Name	Caching	
type	<input type="checkbox"/>	✕
page	<input type="checkbox"/>	✕

[+ Add query string](#)

▶ HTTP Request Headers

▶ Request Models [Create a Model](#)

The client can now supply a pet type and a page number as query string parameters when submitting a request. These input parameters must be mapped into the integration's query string parameters to forward the input values to our PetStore website in the back end. Because the method uses `AWS_IAM`, you must sign the request to invoke the method.

6. From the method's **Integration Request** page, expand the **URL Query String Parameters** section. By default, the method request query string parameters are mapped to the like-named integration request query string parameters. This default mapping works for our demo API. We will leave them as given. To map a different method request parameter to the corresponding integration request parameter, choose the pencil icon for the parameter to edit the mapping expression, shown in the **Mapped from** column. To map a method request parameter to a different integration request parameter, first choose the delete icon to remove the existing integration request parameter, choose **Add query string** to specify a new name and the desired method request parameter mapping expression.

[← Method Execution](#) /pets - GET - Integration Request

Provide information about the target backend that this method will call and whether the incoming request data should be modified.

- Integration type**
- Lambda Function ⓘ
 - HTTP ⓘ
 - Mock ⓘ
 - AWS Service ⓘ

Use HTTP Proxy integration ⓘ

HTTP method GET ✎

Endpoint URL http://petstore-demo-endpoint.execute-api.com/petstore/pets ✎

▶ URL Path Parameters

▼ URL Query String Parameters

Name	Mapped from ⓘ	Caching	
type	method.request.querystring.type	<input type="checkbox"/>	✎ ✕
page	method.request.querystring.page	<input type="checkbox"/>	✎ ✕

+ Add query string

▶ HTTP Headers

▶ Body Mapping Templates

This completes building the simple demo API. It's time to test the API.

7. To test the API using the API Gateway console, choose **Test** from the GET-on-Pets method's **Method Execution** pane. In the **Method Test** pane, enter `DOG` and `2` for the **type** and **page** query strings, respectively, and then choose **Test**.

[← Method Execution](#) /pets - GET - **Method Test**

Make a test call to your method with the provided input

Path

No path parameters exist for this resource. You can define path parameters by using the syntax **{myPathParam}** in a resource path.

Query Strings

type

page

Headers

No header parameters exist for this method. You can add them via Method Request.

Stage Variables

No [stage variables](#) exist for this method.

Client Certificate

No client certificates have been generated.

Request Body

Request Body is not supported for GET methods.



The result is shown as follows. (You may need to scroll down to see the test result.)

Request: /pets?type=Dog&page=2

Status: 200

Latency: 1036 ms

Response Body

```
[
  {
    "id": 4,
    "type": "Dog",
    "price": 999.99
  },
  {
    "id": 5,
    "type": "Dog",
    "price": 249.99
  },
  {
    "id": 6,
    "type": "Dog",
    "price": 49.97
  }
]
```

Response Headers

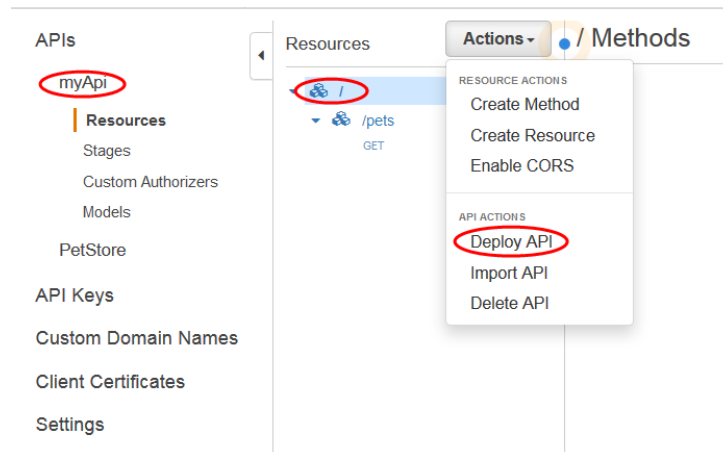
```
{"Content-Type":"application/json"}
```

Logs

```
Execution log for request test-request
Mon Apr 04 05:48:01 UTC 2016 : Starting execution for request: test-invoke-request
Mon Apr 04 05:48:01 UTC 2016 : HTTP Method: GET, Resource Path: /pets
Mon Apr 04 05:48:01 UTC 2016 : Method request path: {}
Mon Apr 04 05:48:01 UTC 2016 : Method request query string: {page=2, type=Dog}
Mon Apr 04 05:48:01 UTC 2016 : Method request headers: {}
Mon Apr 04 05:48:01 UTC 2016 : Method request body before transformations: null
```

Now that the test is successful, we can deploy the API to make it publicly available.

8. To deploy the API, select the API and then choose **Deploy API** from the **Actions** drop-down menu.



In the **Deploy API** dialog, choose a stage (or [New Stage] for the API's first deployment); enter a name (e.g., "test", "prod", "dev", etc.) in the **Stage name** input field; optionally, provide a description in **Stage description** and/or **Deployment description**; and then choose **Deploy**.

Deploy API

Choose a stage where your API will be deployed. For example, a test version of your API could be deployed to a stage named beta.

Deployment stage [New Stage]

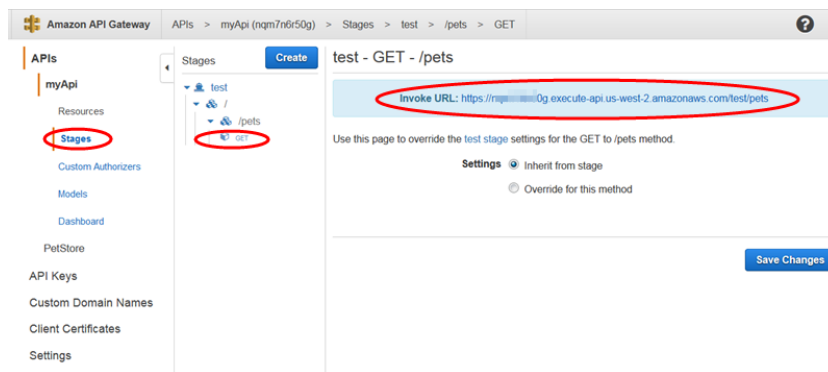
Stage name* test

Stage description GET on Pets only

Deployment description Initial deployment

Cancel Deploy

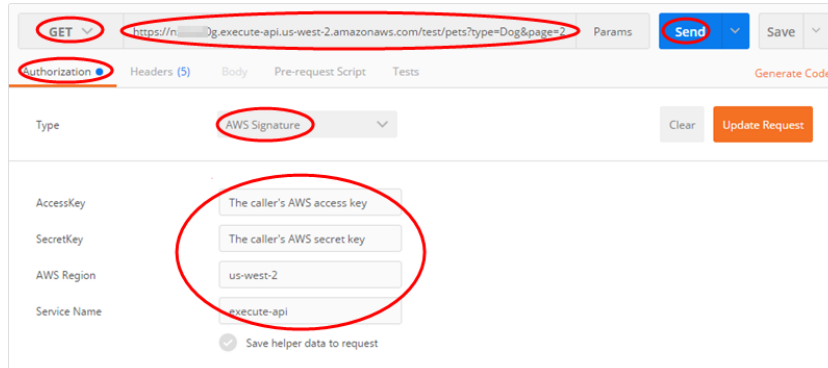
Once deployed, you can obtain the invocation URLs (**Invoke URL**) of the API's endpoints. For example, the GET on Pets method's invocation URL is as follows:



To invoke this API method in a client (e.g., a [Postman](#) browser), append the query string parameters to the stage-specific method invocation URL (as shown in the previous image) to create the complete method request URL:

```
https://api-id.execute-api.region.amazonaws.com/test/pets?type=Dog&page=2
```

Specify this URL in the address bar of the browser. Choose **GET** as the HTTP verb. Select **AWS Signature** for the **Authorization** type and then specify the required properties (as shown), following the [Signature Version 4](#) protocols. Finally, send the request.



If you use an SDK to create a client, you can call the methods exposed by the SDK to sign the request. For implementation details, see the [AWS SDK](#) of your choosing.

Note

When changes are made to your API, you must redeploy the API to make the new or updated features available before invoking the request URL again.

Map Request Parameters for an API Gateway API

In this walkthrough, you will learn how to map method request parameters to the corresponding integration request parameters for an API Gateway API. As an illustration, we will create an example API with the HTTP integration and use it to demonstrate how to use API Gateway to map a method request parameter to the corresponding integration request parameter and to access the publicly accessible HTTP endpoint of:

```
http://petstore-demo-endpoint.execute-api.com/petstore/pets
```

If you copy the above URL, paste it into the address bar of a web browser, and hit the `Enter` or `Return` key, you will get the following JSON-formatted response body:

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

The above endpoint can take two query parameters: `type` and `page`. For example, if you change the above URL to the following:

```
http://petstore-demo-endpoint.execute-api.com/petstore/pets?type=cat&page=2
```

you will receive the following JSON-formatted response payload, displaying page 2 of only the cats:

```
[
  {
    "id": 4,
    "type": "cat",
    "price": 999.99
  },
  {
    "id": 5,
    "type": "cat",
    "price": 249.99
  },
  {
    "id": 6,
    "type": "cat",
    "price": 49.97
  }
]
```

This endpoint also supports the use of an item ID, as expressed by a URL path parameter. For example, if you browse to the following:

```
http://petstore-demo-endpoint.execute-api.com/petstore/pets/1
```

The following JSON-formatted information about the item with an ID of 1 is displayed:

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

In addition to supporting GET operations, this endpoint also take POST requests with a payload. For example, if you use [Postman](#) to send a POST method request to the following:

```
http://petstore-demo-endpoint.execute-api.com/petstore/pets
```

including the header `Content-type: application/json` and the following request body:

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

you will receive following JSON object in the response body:

```
{
  "pet": {
    "type": "dog",
```

```
"price": 249.99
},
"message": "success"
}
```

We now expose these and other features by building an API Gateway API with the HTTP integration of this PetStore website. The tasks includes the following:

- Create an API with a resource of `https://my-api-id.execute-api.region-id.amazonaws.com/test/petstorewalkthrough/pets` acting as a proxy to the HTTP endpoint of `http://petstore-demo-endpoint.execute-api.com/petstore/pets`.
- Enable the API to accept two method request query parameters of `petType` and `petsPage`, map them to the `type` and `page` query parameters of the integration request, respectively, and pass the request to the HTTP endpoint.
- Support a path parameter of `{petId}` on the API's method request URL to specify an item ID, map it to the `{id}` path parameter in the integration request URL, and send the request to the HTTP endpoint.
- Enable the method request to accept the JSON payload of the format defined by the back end website, pass the payload without modifications through the integration request to the back-end HTTP endpoint.

Topics

- [Prerequisites \(p. 24\)](#)
- [Step 1: Create Resources \(p. 24\)](#)
- [Step 2: Create GET and POST Methods \(p. 25\)](#)
- [Step 3: Set Up and Test the Methods \(p. 25\)](#)
- [Step 4: Deploy the API \(p. 28\)](#)
- [Step 5: Test the API \(p. 29\)](#)
- [Next Steps \(p. 31\)](#)

Prerequisites

Before you begin this walkthrough, you should do the following:

1. Complete the steps in [Get Ready to Use API Gateway \(p. 4\)](#), including assigning API Gateway access permission to the IAM user.
2. At a minimum, follow the steps in [Build an API Gateway API to Expose an HTTP Endpoint \(p. 6\)](#) to create a new API named `MyDemoAPI` in the API Gateway console.

Step 1: Create Resources

In this step, you will create three resources that will enable the API to interact with the HTTP endpoint.

To create the first resource

1. In the **Resources** pane, select the resource root, as represented by a single forward slash (`/`), and then choose **Create Resource** from the **Actions** drop-down menu.
2. For **Resource Name**, type `petstorewalkthrough`.

This maps to `petstore` in the HTTP endpoint.

3. For **Resource Path**, accept the default of `/petstorewalkthrough`, and then choose **Create Resource**.

This maps to `/petstore` in the HTTP endpoint.

To create the second resource

1. In the **Resources** pane, choose `/petstorewalkthrough`, and then choose **Create Resource**.
2. For **Resource Name**, type `pets`.

This maps to `pets` in the HTTP endpoint.

3. For **Resource Path**, accept the default of `/petstorewalkthrough/pets`, and then choose **Create Resource**.

This maps to `/petstore/pets` in the HTTP endpoint.

To create the third resource

1. In the **Resources** pane, choose `/petstorewalkthrough/pets`, and then choose **Create Resource**.
2. For **Resource Name**, type `petId`. This maps to the item ID in the HTTP endpoint.
3. For **Resource Path**, overwrite `petid` with `{petId}`. Be sure to use curly braces (`{ }`) around `petId` so that `/petstorewalkthrough/pets/{petId}` is displayed, and then choose **Create Resource**.

This maps to `/petstore/pets/my-item-id` in the HTTP endpoint.

Step 2: Create GET and POST Methods

In this step, you will create two GET methods and a POST method to interact with the HTTP endpoint.

To create the first GET method

1. In the **Resources** pane, choose `/petstorewalkthrough/pets`, and then choose **Create Method** from the **Actions** drop-down menu.
2. For the HTTP method, choose **GET**, and then save your choice.

To create the second GET method

1. In the **Resources** pane, choose `/petstorewalkthrough/pets/{petId}`, and then choose **Create Method**.
2. For the HTTP method, choose **GET**, and then save your choice.

To create the POST method

1. In the **Resources** pane, choose `/petstorewalkthrough/pets` again, and then choose **Create Method**.
2. For the HTTP method, choose **POST**, and then save your choice.

Step 3: Set Up and Test the Methods

In this step, you will integrate the methods with the back-end HTTP endpoints, map the GET method request parameters to the corresponding integration request parameters, and then test the methods.

To set up and test the first GET method

This procedure demonstrates the following:

- Create and integrate the method request of `GET /petstorewalkthrough/pets` with the integration request of `GET http://petstore-demo-endpoint.execute-api.com/petstore/pets`.
- Map the method request query parameters of `petType` and `petsPage` to the integration request query string parameters of `type` and `page`, respectively.

1. In the **Resources** pane, choose `/petstorewalkthrough/pets`, choose **Create Method** from the **Actions** menu, and then choose **GET** under `/pets` from the drop-down list of the method names.
2. In the `/petstorewalkthrough/pets - GET - Setup` pane, choose **HTTP** for **Integration type** and choose **GET** for **HTTP method**.
3. For **Endpoint URL**, type `http://petstore-demo-endpoint.execute-api.com/petstore/pets`.
4. Choose **Save**.
5. In the **Method Execution** pane, choose **Method Request**, and then choose the arrow next to **URL Query String Parameters**.
6. Choose **Add query string**.
7. For **Name**, type `petType`.

This specifies the `petType` query parameter in the API's method request.

8. Choose the pencil icon to finish creating the method request URL query string parameter.
9. Choose **Add query string** again.
10. For **Name**, type `petsPage`.

This specifies the `petsPage` query parameter in the API's method request.

11. Choose the pencil icon to finish creating the method request URL query string parameter.
12. Choose **Method Execution**, choose **Integration Request**, and then choose the arrow next to **URL Query String Parameters**.
13. Delete the `petType` entry mapped from `method.request.querystring.petType` and the `petsPage` entry mapped from `method.request.querystring.petsPage`. This is because the endpoint expects query string parameters named `type` and `page` for the request URL, instead of the default values.
14. Choose **Add query string**.
15. For **Name**, type `type`. This creates the required query string parameter for the integration request URL.
16. For **Mapped from**, type `method.request.querystring.petType`.

This maps the method request's `petType` query parameter to the integration request's `type` query parameter.

17. Choose the pencil icon to finish creating the integration request URL query string parameter.
18. Choose **Add query string** again.
19. For **Name**, type `page`. This creates the required query string parameter for the integration request URL.
20. For **Mapped from**, type `method.request.querystring.petsPage`.

This maps the method request's `petsPage` query parameter to the integration request's `page` query parameter.

21. Choose the pencil icon to finish creating the integration request URL query string parameter.

22. Choose **Method Execution**, and in the **Client** box, choose **TEST**. In the **Query Strings** area, for **petType**, type `cat`. For **petsPage**, type `2`.
23. Choose **Test**. If successful, **Response Body** will display the following:

```
[
  {
    "id": 4,
    "type": "cat",
    "price": 999.99
  },
  {
    "id": 5,
    "type": "cat",
    "price": 249.99
  },
  {
    "id": 6,
    "type": "cat",
    "price": 49.97
  }
]
```

To set up and test the second GET method

This procedure demonstrates the following:

- Create and integrate the method request of `GET /petstorewalkthrough/pets/{petId}` with the integration request of `GET http://petstore-demo-endpoint.execute-api.com/petstore/pets/{id}`.
- Map the method request path parameters of `petId` to the integration request path parameters of `id`.

1. In the **Resources** list, choose `/petstorewalkthrough/pets/{petId}`, choose **Create Method** from the **Actions** drop-down menu, and then choose **GET** as the HTTP verb for the method.
2. In the **Setup** pane, choose **HTTP** for **Integration type** and choose **GET** for **HTTP method**.
3. For **Endpoint URL**, type `http://petstore-demo-endpoint.execute-api.com/petstore/pets/{id}`.
4. Choose **Save**.
5. In the **Method Execution** pane, choose **Integration Request**, and then choose the arrow next to **URL Path Parameters**.
6. Delete the `petId` entry mapped from `method.request.querystring.petId`. This is because the endpoint expects query string parameters named `id` for the request URL, instead of the default value.
7. Choose **Add path**.
8. For **Name**, type `id`.
9. For **Mapped from**, type `method.request.path.petId`.

This maps the method request's path parameter of `petId` to the integration request's path parameter of `id`.

10. Choose the pencil icon to finish creating the URL path parameter.
11. Choose **Method Execution**, and in the **Client** box, choose **TEST**. In the **Path** area, for **petId**, type `1`.
12. Choose **Test**. If successful, **Response Body** will display the following:

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

To set up and test the POST method

This procedure demonstrates the following:

- Create and integrate the method request of `POST /petstorewalkthrough/pets` with the integration request of `POST http://petstore-demo-endpoint.execute-api.com/petstore/pets`.
- Pass the method request JSON payload through to the integration request payload, without modification.

1. In the **Resources** pane, choose `/petstorewalkthrough/pets`, choose **Create Method** from the **Actions** drop-down menu, and then choose **POST** as the HTTP verb for the method.
2. In the **Setup** pane, choose **HTTP** for **Integration type** and choose **POST** for **HTTP method**.
3. For **Endpoint URL**, type `http://petstore-demo-endpoint.execute-api.com/petstore/pets`.
4. Choose **Save**.
5. In the **Method Execution** pane, in the **Client** box, choose **TEST**. Expand **Request Body**, and then type the following:

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

6. Choose **Test**. If successful, **Response Body** will display the following:

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

Step 4: Deploy the API

In this step, you will deploy the API so that you can begin calling it outside of the API Gateway console.

To deploy the API

1. In the **Resources** pane, choose **Deploy API**.
2. For **Deployment stage**, choose `test`.

Note

The input must be UTF-8 encoded (i.e., unlocalized) text.

3. For **Deployment description**, type `Calling HTTP endpoint walkthrough`.

4. Choose **Deploy**.

Step 5: Test the API

In this step, you will go outside of the API Gateway console and use your API to access the HTTP endpoint.

1. In the **Stage Editor** pane, next to **Invoke URL**, copy the URL to the clipboard. It should look something like this:

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

2. Paste this URL in the address box of a new browser tab.
3. Append `/petstorewalkthrough/pets` so that it looks like this:

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

Browse to the URL. The following information should be displayed:

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

4. After `petstorewalkthrough/pets`, type `?petType=cat&petsPage=2` so that it looks like this:

```
https://my-api-id.execute-api.region-id.amazonaws.com/test/  
petstorewalkthrough/pets?petType=cat&petsPage=2
```

5. Browse to the URL. The following information should be displayed:

```
[  
  {  
    "id": 4,  
    "type": "cat",  
    "price": 999.99  
  },  
  {  
    "id": 5,  
    "type": "cat",  
    "price": 249.99  
  }  
]
```

```
    },  
    {  
      "id": 6,  
      "type": "cat",  
      "price": 49.97  
    }  
  ]  
}
```

6. After `petstorewalkthrough/pets`, replace `?petType=cat&petsPage=2` with `/1` so that it looks like this:

```
https://my-api-id.execute-api.region-id.amazonaws.com/test/  
petstorewalkthrough/pets/1
```

7. Browse to the URL. The following information should be displayed:

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

8. Using a web debugging proxy tool or the cURL command-line tool, send a POST method request to the URL from the previous procedure. Be sure to append `/petstorewalkthrough/pets` so that it looks like this:

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

Also, be sure to append the following header:

```
Content-Type: application/json
```

And be sure to add the following code to the request body:

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

For example, if you use the cURL command-line tool, run a command similar to the following:

```
curl -H "Content-Type: application/json" -X POST -d "{\"type\":  
  \"dog\", \"price\": 249.99}" https://my-api-id.execute-api.region-  
id.amazonaws.com/test/petstorewalkthrough/pets
```

The following information should be returned in the response body:

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

```
}
```

You have reached the end of this walkthrough.

Next Steps

You may want to begin the next walkthrough, which shows you how to use models and mappings in API Gateway to transform the output of an API call from one data format to another. See [Map Response Payload \(p. 31\)](#).

Map Response Payload

In this walkthrough, you will learn how to use models and mapping templates in API Gateway to transform the output of an API call from one data schema to another. This walkthrough builds on the instructions and concepts in the [Build an API to Expose a Lambda Function \(p. 44\)](#) and the [Map Request Parameters \(p. 22\)](#). If you have not yet completed those walkthroughs, we suggest you do them first.

This walkthrough will use API Gateway to get example data from a publicly-accessible HTTP endpoint and from an AWS Lambda function you will create. Both the HTTP endpoint and the Lambda function return the same example data:

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

You will use models and mapping templates to transform this data to one or more output formats. In API Gateway, a model defines the format, also known as the schema or shape, of some data. In API Gateway, a mapping template is used to transform some data from one format to another. For more information, see [Set Up Payload Mappings \(p. 105\)](#).

The first model and mapping template is used to rename `id` to `number`, `type` to `class`, and `price` to `salesPrice`, as follows:

```
[
  {
    "number": 1,
    "class": "dog",
    "salesPrice": 249.99
  },
  {
    "number": 2,
    "class": "cat",
```

```
    "salesPrice": 124.99
  },
  {
    "number": 3,
    "class": "fish",
    "salesPrice": 0.99
  }
]
```

The second model and mapping template is used to combine `id` and `type` into `description`, and to rename `price` to `askingPrice`, as follows:

```
[
  {
    "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
  }
]
```

The third model and mapping template is used to combine `id`, `type`, and `price` into a set of listings, as follows:

```
{
  "listings": [
    "Item 1 is a dog. The asking price is 249.99.",
    "Item 2 is a cat. The asking price is 124.99.",
    "Item 3 is a fish. The asking price is 0.99."
  ]
}
```

Topics

- [Prerequisites \(p. 32\)](#)
- [Step 1: Create Models \(p. 33\)](#)
- [Step 2: Create Resources \(p. 35\)](#)
- [Step 3: Create GET Methods \(p. 36\)](#)
- [Step 4: Create a Lambda Function \(p. 36\)](#)
- [Step 5: Set Up and Test the Methods \(p. 37\)](#)
- [Step 6: Deploy the API \(p. 41\)](#)
- [Step 7: Test the API \(p. 41\)](#)
- [Step 8: Clean Up \(p. 43\)](#)
- [Next Steps \(p. 44\)](#)

Prerequisites

Before you begin this walkthrough, you should have already done the following:

1. Complete the steps in [Get Ready to Use API Gateway \(p. 4\)](#), including assigning API Gateway access permission to an IAM user.
2. Open the API Gateway console and create a new API named `MyDemoAPI`. For more information, see [Build an API Gateway API to Expose an HTTP Endpoint \(p. 6\)](#).
3. Create two resources: `petstorewalkthrough` and `pets`. For more information, see [Create Resources \(p. 24\)](#) in the [Map Request Parameters \(p. 22\)](#).
4. To use the Lambda portions of this walkthrough, make sure the IAM user has full access to work with Lambda. You can use the IAM console to attach the **AWSLambdaFullAccess** AWS managed policy to the IAM user.
5. Make sure the IAM user has access to create policies and roles in IAM. If you have not done so already, create a Lambda execution role named `APIGatewayLambdaExecRole` in IAM. For more information, see [Create Lambda Functions \(p. 44\)](#) in the [Build an API to Expose a Lambda Function \(p. 44\)](#).

Step 1: Create Models

In this step, you will create four models. The first three models represent the data output formats for use with the HTTP endpoint and the Lambda function. The last model represents the data input schema for use with the Lambda function.

To create the first output model

1. Sign in to the API Gateway console at <https://console.aws.amazon.com/apigateway>.
2. If **MyDemoAPI** is displayed, choose **Models**.
3. Choose **Create**.
4. For **Model name**, type `PetsModelNoFlatten`.
5. For **Content type**, type `application/json`.
6. For **Model description**, type `Changes id to number, type to class, and price to salesPrice`.
7. For **Model schema**, type the following JSON Schema-compatible definition:

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "title": "PetsModelNoFlatten",
  "type": "array",
  "items": {
    "type": "object",
    "properties": {
      "number": { "type": "integer" },
      "class": { "type": "string" },
      "salesPrice": { "type": "number" }
    }
  }
}
```

8. Choose **Create model**.

To create the second output model

1. Choose **Create**.
2. For **Model name**, type `PetsModelFlattenSome`.
3. For **Content type**, type `application/json`.

4. For **Model description**, type **Combines id and type into description, and changes price to askingPrice**.
5. For **Model schema**, type the following:

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "title": "PetsModelFlattenSome",
  "type": "array",
  "items": {
    "type": "object",
    "properties": {
      "description": { "type": "string" },
      "askingPrice": { "type": "number" }
    }
  }
}
```

6. Choose **Create model**.

To create the third output model

1. Choose **Create**.
2. For **Model name**, type **PetsModelFlattenAll**.
3. For **Content type**, type **application/json**.
4. For **Model description**, type **Combines id, type, and price into a set of listings**.
5. For **Model schema**, type the following:

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "title": "PetsModelFlattenAll",
  "type": "object",
  "properties": {
    "listings": {
      "type": "array",
      "items": {
        "type": "string"
      }
    }
  }
}
```

6. Choose **Create model**.

To create the input model

1. Choose **Create**.
2. For **Model name**, type **PetsLambdaModel**.
3. For **Content type**, type **application/json**.
4. For **Model description**, type **GetPetsInfo model**.
5. For **Model schema**, type the following:

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
```

```
"title": "PetsLambdaModel",
"type": "array",
"items": {
  "type": "object",
  "properties": {
    "id": { "type": "integer" },
    "type": { "type": "string" },
    "price": { "type": "number" }
  }
}
```

6. Choose **Create model**.

Step 2: Create Resources

In this step, you will create four resources. The first three resources will enable you to get the example data from the HTTP endpoint in the three output formats. The last resource will enable you to get the example data from the Lambda function in the output schema that combines `id` and `type` into `description` and renames `price` to `askingPrice`.

To create the first resource

1. In the links list, choose **Resources**.
2. In the **Resources** pane, choose `/petstorewalkthrough`, and then choose **Create Resource**.
3. For **Resource Name**, type `NoFlatten`.
4. For **Resource Path**, accept the default of `/petstorewalkthrough/noflatten`, and then choose **Create Resource**.

To create the second resource

1. In the **Resources** pane, choose `/petstorewalkthrough` again, and then choose **Create Resource**.
2. For **Resource Name**, type `FlattenSome`.
3. For **Resource Path**, accept the default of `/petstorewalkthrough/flattensome`, and then choose **Create Resource**.

To create the third resource

1. In the **Resources** pane, choose `/petstorewalkthrough` again, and then choose **Create Resource**.
2. For **Resource Name**, type `FlattenAll`.
3. For **Resource Path**, accept the default of `/petstorewalkthrough/flattenall`, and then choose **Create Resource**.

To create the fourth resource

1. In the **Resources** pane, choose `/petstorewalkthrough` again, and then choose **Create Resource**.
2. For **Resource Name**, type `LambdaFlattenSome`.
3. For **Resource Path**, accept the default of `/petstorewalkthrough/lambdaflattensome`, and then choose **Create Resource**.

Step 3: Create GET Methods

In this step, you will create a GET method for each of the resources you created in the previous step.

To create the first GET method

1. In the **Resources** list, choose `/petstorewalkthrough/flattenall`, and then choose **Create Method**.
2. From the drop-down list, choose **GET**, and then choose the checkmark icon to save your choice.
3. In the Setup pane, choose **HTTP** for the **Integration type** and **GET** for **HTTP method**, type `http://petstore-demo-endpoint.execute-api.com/petstore/pets` in **Endpoint URL**, and choose **Save**.

To create the second GET method

1. In the **Resources** list, choose `/petstorewalkthrough/lambdaflatten`, and then choose **Create Method**.
2. From the drop-down list, choose **GET**, and then choose the checkmark to save your choice.
3. In the Setup pane, choose **Lambda Function** for the **Integration type**, choose the region where you have created the [GetPetsInfo Lambda function \(p. 36\)](#) from the **Lambda Region** drop-down list, choose `GetPetsInfo` for **Lambda Function**, and choose **Save**. Choose **OK** when prompted to add permission to the Lambda function.

To create the third GET method

1. In the **Resources** list, choose `/petstorewalkthrough/flattensome`, and then choose **Create Method**.
2. From the drop-down list, choose **GET**, and then choose the checkmark icon to save your choice.
3. In the Setup pane, choose **HTTP** for the **Integration type** and **GET** for **HTTP method**, type `http://petstore-demo-endpoint.execute-api.com/petstore/pets` in **Endpoint URL**, and then choose **Save**.

To create the fourth GET method

1. In the **Resources** list, choose `/petstorewalkthrough/noflatten`, and then choose **Actions, Create Method**.
2. From the drop-down list, choose **GET**, and then choose the checkmark icon to save your choice.
3. In the Setup pane, choose **HTTP** for the **Integration type** and **GET** for **HTTP method**, type `http://petstore-demo-endpoint.execute-api.com/petstore/pets` in **Endpoint URL**, and then choose **Save**.

Step 4: Create a Lambda Function

In this step, you will create a Lambda function that returns the sample data.

To create the Lambda function

1. Open the AWS Lambda console at <https://console.aws.amazon.com/lambda/>.
2. Do one of the following:
 - If a welcome page appears, choose **Get Started Now**.
 - If the **Lambda: Function list** page appears, choose **Create a Lambda function**.
3. For **Name**, type `GetPetsInfo`.

4. For **Description**, type **Gets information about pets**.
5. For **Code template**, choose **None**.
6. Type the following code:

```
console.log('Loading event');

exports.handler = function(event, context) {
  context.done(null,
    [{ "id": 1, "type": "dog", "price": 249.99 },
      { "id": 2, "type": "cat", "price": 124.99 },
      { "id": 3, "type": "fish", "price": 0.99 }]); // SUCCESS with message
};
```

Tip

In the preceding code, written in Node.js, `console.log` writes information to an Amazon CloudWatch log. `event` contains the event's data. `context` contains callback context. Lambda uses `context.done` to perform follow-up actions. For more information about how to write Lambda function code, see the "Programming Model" section in [AWS Lambda: How it Works](#) and the sample walkthroughs in the *AWS Lambda Developer Guide*.

7. For **Handler name**, leave the default of `index.handler`.
8. For **Role**, choose the Lambda execution role, **APIGatewayLambdaExecRole**, you created in the [Build an API to Expose a Lambda Function \(p. 44\)](#).
9. Choose **Create Lambda function**.
10. In the list of functions, choose **GetPetsInfo** to show the function's details.
11. Make a note of the AWS region where you created this function. You will need it later.
12. In the pop-up list, choose **Edit or test function**.
13. For **Sample event**, replace any code that appears with the following:

```
{
}
```

Tip

The empty curly braces mean there are no input values for this Lambda function. This function simply returns the JSON object containing the pets information, so those key/value pairs are not required here.

14. Choose **Invoke**. **Execution result** shows `[{"id":1,"type":"dog","price":249.99}, {"id":2,"type":"cat","price":124.99}, {"id":3,"type":"fish","price":0.99}]`, which is also written to the CloudWatch logs.
15. Choose **Go to function list**.

Step 5: Set Up and Test the Methods

In this step, you will configure the method responses, integration requests and integration responses to specify the input and output data schemas (or models) for the GET methods associated with the HTTP endpoint and the Lambda function. You will also learn to test calling these methods using the API Gateway console.

To set up the integration for the first GET method and then test it

1. From the API's **Resources** tree, choose **GET** under the `/petstorewalkthrough/flattenall` node.

2. In the **Method Execution** pane, choose **Method Response**, and then choose the arrow next to **200**.
3. In the **Response Models for 200** area, for **application/json**, choose the pencil icon to start setting up the model for the method output. For **Models**, choose **PetsModelFlattenAll**, and then choose the checkmark icon to save the setting.
4. Choose **Method Execution**, choose **Integration Response**, and then choose the arrow next to **200**.
5. Expand the **Body Mapping Templates** section, choose **application/json** under **Content-Type**.
6. For **Generate template from model**, choose **PetsModelFlattenAll** to display a mapping template after the `PetsModelFlattenAll` model as a starting point.
7. Modify the mapping template code as follows:

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

#end
  ]
}
```

8. Choose **Save**.
9. Choose **Method Execution**, and in the **Client** box, choose **TEST**, and then choose **Test**. If successful, **Response Body** will display the following:

```
{
  "listings" : [
    "Item number 1 is a dog. The asking price is 249.99.",
    "Item number 2 is a cat. The asking price is 124.99.",
    "Item number 3 is a fish. The asking price is 0.99."
  ]
}
```

To set up integration for the second GET method and then test it

1. From the API's **Resources** tree, choose **GET** under the **/petstorewalkthrough/lambdaflatten** node.
2. In the **Method Execution** pane, choose **Integration Request** to set up mapping of the input to the `GetPetsInfo` Lambda function.
3. Expand the **Body Mapping Templates** section, choose **Add mapping template**.
4. For **Content-Type**, type **application/json** and choose the check-mark icon to save the choice. When prompted choose **Yes, secure this integration** to allow change of passthrough behavior.
5. For **Generate template**, choose **PetsLambdaModel** to display the input mapping script template.
6. Copy the following mapping script and paste it into the code editor before choosing **Save**:

```
#set($inputRoot = $input.path('$'))
[
#foreach($elem in $inputRoot)
  {
    "id" : $elem.id,
    "type" : "$elem.type",
```

```
    "price" : $elem.price
  }#if($foreach.hasNext),#end

#end
]
```

7. Go back to **Method Execution** and choose **Method Response**. Choose the arrow next to **200** to expand the section.
8. In the **Response Models for 200** area, choose the pencil icon on the row for the content type of **application/json**. Choose **PetsModelFlattenSome** for **Models**, and then choose the check mark icon to save the choice.
9. Go back to **Method Execution**. Choose **Integration Response**, and then choose the arrow next to **200**.
10. In the **Body Mapping Templates** section, choose **application/json** under **Content-Type**.
11. For **Generate template**, choose **PetsModelFlattenSome** to display the mapping script template for the output of this method.
12. Modify the code as follows, and then choose **Save**:

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

#end
]
```

13. Choose **Method Execution**, and in the **Client** box, choose **TEST**, and then choose **Test**. If successful, **Response Body** will display the following:

```
[
  {
    "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
  }
]
```

To set up integration for the third GET method and then test it

1. From the API's **Resources** tree, choose **GET** under the **/petstorewalkthrough/flattensome** node.
2. In the **Method Execution** pane, choose **Method Response**.
3. Choose the arrow next to **200**.
4. In the **Response Models for 200** area, for **application/json**, choose the pencil icon. For **Models**, choose **PetsModelFlattenSome**, and then choose the check-mark icon to save the choice.

5. Go back to **Method Execution** and choose **Integration Response**.
6. Choose the arrow next to **200** to expand the section.
7. Expand the **Body Mapping Templates** area. Choose **application/json** for **Content-Type**. For **Generate template**, choose **PetsModelFlattenSome** to display a mapping script template for the output of this method.
8. Modify the code as follows:

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

9. Choose **Save**.
10. Go back to **Method Execution** and choose **TEST** in the **Client** box. And then choose **Test**. If successful, **Response Body** will display the following:

```
[
  {
    "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
  }
]
```

To set up integration for the fourth GET method and then test it

1. From the API's **Resources** tree, choose **GET** under the **/petstorewalkthrough/noflatten** node.
2. In the **Method Execution** pane, choose **Method Response**, and then expand the **200** section.
3. In the **Response Models for 200** area, for **application/json**, choose the pencil icon to update the response model for this method.
4. Choose **PetsModelNoFlatten** as the model for the content type of **application/json**, and then choose the check-mark icon to save the choice.
5. Choose **Method Execution**, choose **Integration Response**, and then choose the arrow next to **200** to expand the section.
6. Expand the **Mapping Mapping Templates** section. Choose **application/json** for **Content-Type**. For **Generate templates**, choose **PetsModelNoFlatten** to display a mapping script template for the output of this method.
7. Modify the code as follows:

```
#set($inputRoot = $input.path('$'))
```

```
[
#foreach($elem in $inputRoot)
  {
    "number": $elem.id,
    "class": "$elem.type",
    "salesPrice": $elem.price
  }#if($foreach.hasNext),#end
#end
]
```

8. Choose **Save**.
9. Go back to **Method Execution**, and in the **Client** box, choose **TEST**, and then choose **Test**. If successful, **Response Body** will display the following:

```
[
  {
    "number": 1,
    "class": "dog",
    "salesPrice": 249.99
  },
  {
    "number": 2,
    "class": "cat",
    "salesPrice": 124.99
  },
  {
    "number": 3,
    "class": "fish",
    "salesPrice": 0.99
  }
]
```

Step 6: Deploy the API

In this step, you will deploy the API so that you can begin calling it outside of the API Gateway console.

To deploy the API

1. In the **Resources** pane, choose **Deploy API**.
2. For **Deployment stage**, choose **test**.
3. For **Deployment description**, type **Using models and mapping templates walkthrough**.
4. Choose **Deploy**.

Step 7: Test the API

In this step, you will go outside of the API Gateway console to interact with both the HTTP endpoint and the Lambda function.

1. In the **Stage Editor** pane, next to **Invoke URL**, copy the URL to the clipboard. It should look something like this:

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

2. Paste this URL in the address box of a new browser tab.

3. Append `/petstorewalkthrough/noflatten` so that it looks like this:

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

Browse to the URL. The following information should be displayed:

```
[  
  {  
    "number": 1,  
    "class": "dog",  
    "salesPrice": 249.99  
  },  
  {  
    "number": 2,  
    "class": "cat",  
    "salesPrice": 124.99  
  },  
  {  
    "number": 3,  
    "class": "fish",  
    "salesPrice": 0.99  
  }  
]
```

4. After `petstorewalkthrough/`, replace `noflatten` with `flattensome`.
5. Browse to the URL. The following information should be displayed:

```
[  
  {  
    "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  
  }  
]
```

6. After `petstorewalkthrough/`, replace `flattensome` with `flattenall`.
7. Browse to the URL. The following information should be displayed:

```
{  
  "listings" : [  
    "Item number 1 is a dog. The asking price is 249.99.",  
    "Item number 2 is a cat. The asking price is 124.99.",  
    "Item number 3 is a fish. The asking price is 0.99."  
  ]  
}
```

8. After `petstorewalkthrough/`, replace `flattenall` with `lambdaflatten`.
9. Browse to the URL. The following information should be displayed:

```
[
  {
    "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
  }
]
```

Step 8: Clean Up

If you no longer need the Lambda function you created for this walkthrough, you can delete it now. You can also delete the accompanying IAM resources.

Caution

If you delete a Lambda function your APIs rely on, those APIs will no longer work. Deleting a Lambda function cannot be undone. If you want to use the Lambda function again, you must re-create the function.

If you delete an IAM resource a Lambda function relies on, the Lambda function and any APIs that rely on it will no longer work. Deleting an IAM resource cannot be undone. If you want to use the IAM resource again, you must re-create the resource. If you plan to continue experimenting with the resources you created for this and the other walkthroughs, do not delete the Lambda invocation role or the Lambda execution role.

API Gateway does not currently support the deactivation or deletion of APIs that no longer work.

To delete the Lambda function

1. Sign in to the AWS Management Console and open the AWS Lambda console at <https://console.aws.amazon.com/lambda/>.
2. On the **Lambda: Function list** page, in the list of functions, choose the button next to **GetPetsInfo**, and then choose **Actions, Delete**. When prompted, choose **Delete** again.

To delete the associated IAM resources

1. Open the Identity and Access Management (IAM) console at <https://console.aws.amazon.com/iam/>.
2. In the **Details** area, choose **Roles**.
3. Select **APIGatewayLambdaExecRole**, and then choose **Role Actions, Delete Role**. When prompted, choose **Yes, Delete**.
4. In the **Details** area, choose **Policies**.
5. Select **APIGatewayLambdaExecPolicy**, and then choose **Policy Actions, Delete**. When prompted, choose **Delete**.

You have now reached the end of this walkthrough.

Next Steps

You may want to begin the next walkthrough, which shows you how to create an API Gateway API to access an AWS service. See [Create an AWS Service Proxy \(p. 64\)](#).

Build an API to Expose a Lambda Function

AWS Lambda provides an easy way to build back ends without managing servers. API Gateway and Lambda together can be powerful to create and deploy serverless Web applications. In this walkthrough, you learn how to create Lambda functions and build an API Gateway API to enable a Web client to call the Lambda functions synchronously. For more information about Lambda, see the [AWS Lambda Developer Guide](#). For information about asynchronous invocation of Lambda functions, see [Create an API as a Lambda Proxy \(p. 343\)](#).

Topics

- [Step 1: Prerequisites \(p. 44\)](#)
- [Step 2: Create an API \(p. 44\)](#)
- [Step 3: Create a Resource \(p. 44\)](#)
- [Step 4: Create Lambda Functions \(p. 44\)](#)
- [Step 5: Create and Test a GET Method \(p. 48\)](#)
- [Step 6: Create and Test a POST Method \(p. 49\)](#)
- [Step 7: Deploy the API \(p. 50\)](#)
- [Step 8: Test the API \(p. 50\)](#)
- [Step 9: Clean Up \(p. 51\)](#)
- [Next Steps \(p. 52\)](#)
- [Create Lambda Invocation and Execution Roles \(p. 52\)](#)

Step 1: Prerequisites

You must grant API Gateway access permission to the IAM user who will perform the tasks discussed here. The IAM user must have full access to work with Lambda. For this, you can use or customize the managed policy of **AWSLambdaFullAccess** (`arn:aws:iam::aws:policy/AWSLambdaFullAccess`) and attach it to the IAM user. For more information, see [Get Ready to Use API Gateway \(p. 4\)](#). The IAM user must also be allowed to create policies and roles in IAM. For this you can use or customize the managed policy of **IAMFullAccess** (`arn:aws:iam::aws:policy/IAMFullAccess`) and attach it to the user.

Step 2: Create an API

In this step, you will create a new API named `MyDemoAPI`. To create the new API, follow the instruction given in Step 1 of [Build the API Step By Step \(p. 14\)](#).

Step 3: Create a Resource

In this step, you will create a new resource named `MyDemoResource`. To create this resource, follow the instruction given in Step 2 of [Build the API Step By Step \(p. 14\)](#).

Step 4: Create Lambda Functions

Note

Creating Lambda functions may result in charges to your AWS account.

In this step, you will create two new Lambda functions. The first Lambda function, `GetHelloWorld`, will log the call to Amazon CloudWatch and return the JSON object `{"Hello": "World"}`. For more information about JSON, see [Introducing JSON](#).

The second Lambda function, `GetHelloWithName`, will take an input (`"name"`), log the call to CloudWatch, and return the JSON object `{"Hello": user-supplied-input-value}`. If no input value is provided, the value will be `"No-Name"`.

You will use the Lambda console to create the Lambda functions and set up the required execution role/policy. You will then use the API Gateway console to create an API to integrate API methods with the Lambda functions; the API Gateway console will set up the required Lambda invocation role/policy. If you set up the API without using the API Gateway console, such as when [importing an API from Swagger](#), you must explicitly create, if necessary, and set up an invocation role/policy for API Gateway to invoke the Lambda functions. For more information on how to set up Lambda invocation and execution roles, see [Create Lambda Invocation and Execution Roles \(p. 52\)](#). For more information about Lambda see [AWS Lambda Developer Guide](#).

To create the `GetHelloWorld` Lambda function

1. Open the AWS Lambda console at <https://console.aws.amazon.com/lambda/>.
2. Do one of the following:
 - If the welcome page appears, choose **Get Started Now**.
 - If the **Lambda > Functions** list page appears, choose **Create a Lambda function**.
3. From **Select blueprint**, choose the **hello-world** blueprint for `node.js`. You may need to type `Hello` as the search filter to bring the blueprint in focus.
4. In the **Configure triggers** pane, choose **Next**.
5. For **Name**, type `GetHelloWorld`.
6. For **Description**, type `Returns {"Hello": "World"}`.
7. For **Runtime**, choose **Node.js 4.3**.
8. Under **Lambda function code**, replace the default code statements in the inline code editor with the following:

```
'use strict';
console.log('Loading event');

exports.handler = function(event, context, callback) {
  console.log('{"Hello": "World"}');
  callback(null, {"Hello": "World"}); // SUCCESS with message
};
```

Tip

In the preceding Lambda function code in Node.js 4.3, the `console.log` method writes information to an Amazon CloudWatch log. The `event` parameter contains the input to the function. The `context` parameter contains execution context and `callback` the callback function. For the Lambda function to return successfully, call `callback(null, result)` to exit the function. To return an error message, call `callback(error)` upon exit. For more information about how to write Lambda function code, see the "Programming Model" section in [AWS Lambda: How it Works](#) and the sample walkthroughs in the *AWS Lambda Developer Guide*.

9. Under **Lambda function handler and role**, leave the default of `index.handler` for **Handler**.
10. For **Role**, choose `Create new role from template(s)`:
 - a. For **Role name**, type a name for your role (e.g., `execute_my_lambda`).

- b. For **Policy templates**, choose `Simple Microservice permissions`.

Tip

To use an existing IAM role, choose `Choose an existing role` for **Role** and then select an entry from the drop-down list of the existing roles. Alternatively, to create a custom role, choose `Create a Custom Role` and follow the instructions therein.

11. For **Advanced settings** leave the default setting as is.
12. Choose **Next**
13. Choose **Create function**.

Make a note of the AWS region where you created this function. You will need it later.

14. To test the newly created function, as a good practice, choose **Actions** and then select **Configure test event**.
15. For **Input test event**, replace any default code statements with the following, and then choose **Save and test**.

```
{ }
```

Tip

This function does not use any input. Therefore, we provide an empty JSON object as the input.

16. Choose **Test** to invoke the function. The **Execution result** section shows `{"Hello": "World"}`. The output is also written to CloudWatch Logs.

You can use the IAM console to view the IAM role (`execute_my_lambda`) created as part of the Lambda function creation. Attached to this IAM role is the following inline policy that grants users of your AWS account permission to call the CloudWatch `CreateLogGroup`, `CreateLogStreams`, and `PutLogEvents` actions on any of the CloudWatch resources.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents"
      ],
      "Resource": "arn:aws:logs:*:*:*"
    }
  ]
}
```

A trusted entity of this IAM role is `lambda.amazonaws.com`, which has the following trust relationship:

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

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

The combination of this trust relationship and the inline policy makes it possible for the user to invoke the Lambda function and for Lambda to call the supported CloudWatch actions on the user's behalf.

To create the `GetHelloWithName` Lambda function

1. Go back to the **Lambda > Functions** list to create the next Lambda function that takes an input value.
2. Choose **Create a Lambda function**.
3. From **Select blueprint**, select the **hello-world** blueprint for `nodejs`.
4. Type `GetHelloWithName` for **Name**.
5. For **Description**, type `Returns {"Hello":", a user-provided string, and "}`.
6. For **Runtime**, choose **Node.js 4.3**.
7. In the code editor under **Lambda function code** replace the default code statements with the following:

```
'use strict';
console.log('Loading event');

exports.handler = function(event, context, callback) {
  var name = (event.name === undefined ? 'No-Name' : event.name);
  console.log("Hello:" + name);
  callback(null, {"Hello":name}); // SUCCESS with message
};
```

8. Under **Lambda function handler and role**, leave the default of `index.handler` for **Handler**.
9. For **Role**, choose **Use existing role** and then choose the `execute_my_lambda` role, created previously, from the drop-down list of existing roles.
10. Leave the default values for **Advanced settings**. Then choose **Next**.
11. Choose **Create function**.

Make a note of the AWS region where you created this function. You will need it in later.

12. To test this newly created function, choose **Actions** and then **Configure test event**.
13. In **Input test event**, replace any default code statements with the following, and then choose **Save and test**.

```
{
  "name": "User"
}
```

Tip

The function calls `event.name` to read the input name. We expect it to return `{"Hello": "User"}`, given the above input.

You can experiment with this function by removing "name": "User" from the **Input test event** for the function and choosing **Save and test** again. You should see the output of {"Hello": "No-Name"} under **Execution result** in the Lambda console, as well as in CloudWatch Logs.

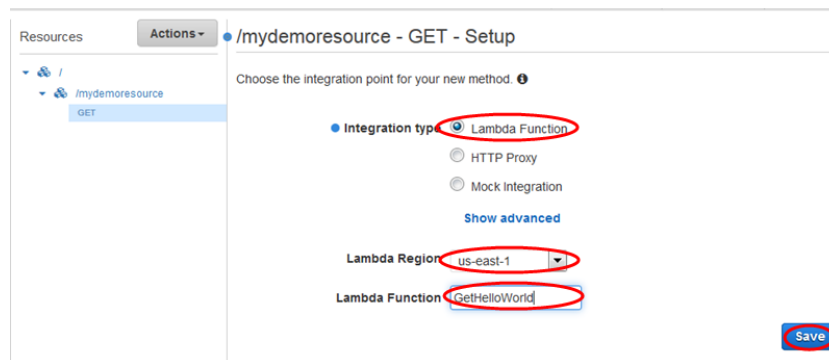
Step 5: Create and Test a GET Method

Switch back to the API Gateway console. In this step, you will create a GET method, connect it to your GetHelloWorld function in Lambda, and then test it. You use a GET method primarily to retrieve or read a representation of a resource. If successful, the GET method will return a JSON-formatted object.

To create and test the GET method

1. In the API Gateway console, from **APIs**, choose **MyDemoAPI**.
2. In the **Resources** pane, choose **/mydemoresource**. From **Actions**, choose **Create Method**. Choose **GET** from the HTTP method drop-down list and then choose the checkmark to create the method.
3. In the GET method **Setup** pane, for **Integration type**, choose **Lambda Function**. For **Lambda Region**, choose the region (.e.g, us-east-1) where you created the Lambda functions. In **Lambda Function**, type GetHelloWorld. Choose **Save** to finish setting up the integration request for this method.

For a list of region names and identifiers, see [AWS Lambda](#) in the *Amazon Web Services General Reference*.



4. For **Add Permission to Lambda Function**, choose **OK**.
5. In the **Method Execution** pane, choose **TEST** from the **Client** box, and then choose the **Test** button. If successful, **Response Body** will display the following:

```
{
  "Hello": "World"
}
```

By default, API Gateway will pass through the request from the API caller. For the GET method call you just created, as well as for HEAD method calls, a Lambda function will receive an empty JSON response by default and then return the response from the Lambda function without modifications.

In the next step, you will create a POST method call. For POST and PUT method calls, you can pass in a request body in JSON format, which the Lambda function will receive as its input event. Optionally, you can transform the input to the Lambda function by using mapping templates in API Gateway.

Step 6: Create and Test a POST Method

In this step, you will create a new POST method, connect it to your `GetHelloWithName` function in Lambda, and then test it. If successful, the POST method typically returns to the caller the URI of the newly created resource. In this walkthrough, the POST method will simply return a JSON-formatted object.

To create and test the POST method

1. In the **Resources** pane, choose **/mydemoresource**, and then choose **Create Method**.
2. For the HTTP method, choose **POST**, and then choose the checkmark to save your choice.
3. In the **Setup** pane, for **Integration Type**, choose **Lambda Function**.
4. For **Lambda Region**, choose the region identifier that corresponds to the region name in which you created the `GetHelloWithName` Lambda function.
5. For **Lambda Function**, type `GetHelloWithName`, and then choose **Save**.
6. When you are prompted to give API Gateway permission to invoke your Lambda function, choose **OK**.
7. In the **Method Execution** pane, in the **Client** box, and then choose **TEST**. Expand **Request Body**, and type the following:

```
{
  "name": "User"
}
```

8. Choose **Test**. If successful, **Response Body** will display the following:

```
{
  "Hello": "User"
}
```

9. Change **Request Body** by removing `"name": "User"` so that only a set of curly braces (`{ }`) remain, and then choose **Test** again. If successful, **Response Body** will display the following:

```
{
  "Hello": "No-Name"
}
```

The API Gateway console-assisted Lambda function integration uses the AWS service proxy integration type for Lambda. It streamlines the process to integrate an API method with a Lambda function by setting up, among other things, the required Lambda function invocation URI and the invocation role on behalf of the API developer.

The GET and POST methods discussed here are both integrated with a POST request in the back end:

```
POST /2015-03-31/functions/FunctionArn/invocations?Qualifier=Qualifier
HTTP/1.1
X-Amz-Invocation-Type: RequestReponse
...
Content-Type: application/json
Content-Length: PayloadSize

Payload
```

The `X-Amz-Invocation-Type: RequestResponse` header specifies that the Lambda function be invoked synchronously. `FunctionArn` is of the `arn:aws:lambda:region:account-id:function:FunctionName` format. In this walkthrough, the console sets `FunctionName` as `GetHelloWorld` for the `GET` method request and supplies an empty JSON payload when you test-invoke the method. For the `POST` method, the console sets `FunctionName` as `GetHelloWithName` and passes the caller-supplied method request payload to the integration request. You can regain full control of a method creation and setup by going through the AWS service proxy integration directly. For more information, see [Create an API as a Lambda Proxy](#) (p. 343).

Step 7: Deploy the API

You are now ready to deploy your API so that you can call it outside of the API Gateway console. In this step, you will create a stage. In API Gateway, a stage defines the path through which an API deployment is accessible. For example, you can define a `test` stage and deploy your API to it, so that a resource named `MyDemoAPI` is accessible through a URI that ends in `.../test/MyDemoAPI`.

To deploy the API

1. Choose the API from the **APIs** pane or choose a resource or method from the **Resources** pane. Choose **Deploy API** from the **Actions** drop-down menu.
2. For **Deployment stage**, choose **New Stage**.
3. For **Stage name**, type `test`.

Note

The input must be UTF-8 encoded (i.e., unlocalized) text.

4. For **Stage description**, type `This is a test`.
5. For **Deployment description**, type `Calling Lambda functions walkthrough`.
6. Choose **Deploy**.

Step 8: Test the API

In this step, you will go outside of the API Gateway console to call the `GET` and `POST` methods in the API you just deployed.

To test the GET-on-mydemoresource method

1. In the **Stage Editor** pane, copy the URL from **Invoke URL** to the clipboard. It should look something like this:

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

2. In a separate web browser tab or window, paste the URL into the address box. Append the path to your resource (`/mydemoresource`) to the end of the URL. The URL should look something like this:

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

3. Browse to this URL. If the `GET` method is successfully called, the web page will display:

```
{"Hello": "World" }
```

To test the POST-on-mydemoresource method

1. You will not be able to test a POST method request with your web browser's address bar. Instead, use an advanced REST API client, such as [Postman](#), or the [cURL](#) command-line tool.
2. Send a POST method request to the URL from the previous procedure. The URL should look something like this:

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

Be sure to append to the request headers the following header:

```
Content-Type: application/json
```

Also, be sure to add the following code to the request body:

```
{  
  "name": "User"  
}
```

For example, if you use the cURL command-line tool, run a command similar to the following:

```
curl -H "Content-Type: application/json" -X POST -d "{\"name\": \"User\"}"  
https://my-api-id.execute-api.region-id.amazonaws.com/test/mydemoresource
```

If the POST method is successfully called, the response should contain:

```
{"Hello": "User" }
```

Step 9: Clean Up

If you no longer need the Lambda functions you created for this walkthrough, you can delete them now. You can also delete the accompanying IAM resources.

Caution

If you plan to complete the other walkthroughs in this series, do not delete the Lambda execution role or the Lambda invocation role. If you delete a Lambda function that your APIs rely on, those APIs will no longer work. Deleting a Lambda function cannot be undone. If you want to use the Lambda function again, you must re-create the function.

If you delete an IAM resource that a Lambda function relies on, that Lambda function will no longer work, and any APIs that rely on that function will no longer work. Deleting an IAM resource cannot be undone. If you want to use the IAM resource again, you must re-create the resource.

To delete the Lambda functions

1. Sign in to the AWS Management Console and open the AWS Lambda console at <https://console.aws.amazon.com/lambda/>.
2. From the list of functions, choose **GetHelloWorld**, choose **Actions** and then choose **Delete function**. When prompted, choose **Delete** again.
3. From the list of functions, choose **GetHelloWithName**, choose **Actions**, and then choose **Delete function**. When prompted, choose **Delete** again.

To delete the associated IAM resources

1. Open the Identity and Access Management (IAM) console at <https://console.aws.amazon.com/iam/>.
2. From **Details**, choose **Roles**.
3. From the list of roles, choose **APIGatewayLambdaExecRole**, choose **Role Actions** and then choose **Delete Role**. When prompted, choose **Yes, Delete**.
4. From **Details**, choose **Policies**.
5. From the list of policies, choose **APIGatewayLambdaExecPolicy**, choose **Policy Actions** and then choose **Delete**. When prompted, choose **Delete**.

You have now reached the end of this walkthrough.

Next Steps

You may want to proceed to the next walkthrough, which shows how to map header parameters from the method request to the integration request and from the integration response to the method response. It uses the HTTP proxy integration to connect your API to HTTP endpoints in the back end.

For more information about API Gateway, see [What Is Amazon API Gateway? \(p. 1\)](#). For more information about REST, see [RESTful Web Services: A Tutorial](#).

Create Lambda Invocation and Execution Roles

Before you create AWS Lambda functions, you must assign appropriate permissions for the functions to execute the specified Amazon CloudWatch action (namely, writing to the CloudWatch Log) and for API Gateway to invoke the Lambda functions. You set up the permissions using IAM roles and policies for API Gateway to invoke your code and for Lambda to execute your code. For more information about invocation and execution roles/policies in Lambda see [Permission Model](#) in the *AWS Lambda Developer Guide*.

To create the Lambda invocation role and its policy

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.

If you are using the IAM-managed **AWSLambdaRole** policy, skip to Step 8 to create an invocation role.
2. In **Details**, choose **Policies**.
3. Do one of the following:
 - If a list of policies appears, choose **Create Policy**.
 - If the **Welcome to Managed Policies** page appears, choose **Get Started**, and then choose **Create Policy**.
4. For **Create Your Own Policy**, choose **Select**.
5. For **Policy Name**, type a name for the policy; for example, **APIGatewayLambdaInvokePolicy**.
6. For **Description**, type **Enables API Gateway to call Lambda functions**.
7. For **Policy Document**, type the following, and then choose **Create Policy**.

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

```



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

8. In **Details**, choose **Roles**.
9. Choose **Create New Role**.
10. For **Role Name**, type a name for the invocation role; for example, `APIGatewayLambdaInvokeRole`, and then choose **Next Step**.
11. Under **Select Role Type**, with the option button next to **AWS Service Roles** already chosen, for **Amazon API Gateway**, choose **Select**.
12. For **Attach Policy**, if the policy you want is in the list, choose it before choosing **Next Step**. Otherwise, simply choose **Next Step** to proceed.
13. For **Role ARN**, make a note of the invocation role's Amazon Resource Name (ARN). You will need this ARN in later steps when you specify the invocation role explicitly. The ARN should look similar to this: `arn:aws:iam::123456789012:role/APIGatewayLambdaInvokeRole`, where 123456789012 is your AWS account ID.
14. Choose **Create Role**.

The newly created IAM role will have the following trust policy.

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

The preceding policy document enables API Gateway to assume roles taken up by and, hence, take actions on behalf of your AWS account.

To create the Lambda execution role and its policies

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In **Details**, choose **Policies**.
3. Choose **Create Policy**.
4. For **Create Your Own Policy**, choose **Select**.
5. For **Policy Name**, type a name for the policy (for example, `APIGatewayLambdaExecPolicy`).
6. For **Description**, type `Enables Lambda to execute code`.
7. For **Policy Document**, type the following, and then choose **Create Policy**.

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

Note

The preceding policy document permits all log actions on Amazon CloudWatch Logs. Typically, you would add other permissions required by your Lambda function to interact with AWS services, such as uploading an object to an Amazon S3 bucket. In this walkthrough, the Lambda functions you create are very simple; they do not interact with AWS services.

8. In **Details**, choose **Roles**.
9. Choose **Create New Role**.
10. In **Role Name**, type a name for the execution role (for example, `APIGatewayLambdaExecRole`), and then choose **Next Step**.
11. Next to **AWS Lambda**, choose **Select**.

Note

IAM will attach the following resource-policy document in **Trust Relationships**:

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

This policy document enables Lambda to assume roles taken up by and, hence, to take actions on behalf of your AWS account.

Build an API Gateway API Using Proxy Integration and a Proxy Resource

As an API developer, you can configure a proxy resource in an API Gateway API to enable a client to access an HTTP back end or to call a Lambda function following various routes with a single integration setup on a catch-all `ANY` method. For more information about the proxy resource,

ANY method and proxy resource integration types, see [Configure Proxy Integration for a Proxy Resource](#) (p. 97).

This tutorial describes how to build and test two APIs with proxy resources, one for an HTTP back end and the other for a Lambda function. To follow the instructions, you must have signed up for an AWS account. For more information about signing up for AWS, see [Getting Started](#) (p. 4).

Topics

- [Create and Test an API with HTTP Proxy Integration through a Proxy Resource](#) (p. 55)
- [Create an API with Lambda Proxy Integration through a Proxy Resource](#) (p. 59)

Create and Test an API with HTTP Proxy Integration through a Proxy Resource

In this tutorial, we will create an API to integrate with the PetStore website through a proxy resource using the HTTP proxy integration.

Topics

- [Create an API with HTTP Proxy Integration through a Proxy Resource](#) (p. 55)
- [Test an API with HTTP Proxy Integration through Proxy Resource](#) (p. 57)

Create an API with HTTP Proxy Integration through a Proxy Resource

The following procedure walks you through the steps to create and test an API with a proxy resource for an HTTP back end using the API Gateway console. The HTTP back end is the PetStore website (<http://petstore-demo-endpoint.execute-api.com/>) from [Build an API Gateway API to Expose an HTTP Endpoint](#) (p. 6).

To build an API with HTTP Proxy Integration with the PetStore website through a proxy resource

1. Sign in to the API Gateway console at <https://console.aws.amazon.com/apigateway>.
2. To create an API, choose **Create new API** (for creating the first API) or **Create API** (for creating any subsequent API). Next, do the following:
 - a. Choose **New API**.
 - b. Type a name in **API Name**.
 - c. Optionally, add a brief description in **Description**.
 - d. Choose **Create API**.

For this tutorial, use `ProxyResourceForPetStore` for the API name.

3. To create a child resource, choose a parent resource item under the **Resources** tree and then choose **Create Resource** from the **Actions** drop-down menu. Then, do the following in the **New Child Resource** pane.
 - a. Select the **Configure as proxy resource** option to create a proxy resource. Otherwise, leave it de-selected.
 - b. Type a name in the **Resource Name*** input text field.
 - c. Type a new name or use the default name in the **Resource Path*** input text field.
 - d. Choose **Create Resource**.

- e. Select **Enable API Gateway CORS**, if required.

For this tutorial, select **Configure as proxy resource**. For **Resource Name**, use the default, `proxy`. For **Resource Path**, use `/ {proxy+}`. Select **Enable API Gateway CORS**.

> ProxyResourceForPetStore (miqyuu3lfg) > Resources > / (7ryftl47g0) > Create Show all hints

Resources Actions ▾

New Child Resource

Use this page to create a new child resource for your resource.

Configure as proxy resource ℹ

Resource Name*

Resource Path*

You can add path parameters using brackets. For example, the resource path `{username}` represents a path parameter called 'username'.
Configuring `/ {proxy+}` as a proxy resource catches all requests to sub-resources. For example, it works for a GET request to `/foo`. To catch requests to `/`, add a new ANY method on the `/` resource.

Enable API Gateway CORS ℹ

* Required Cancel Create

4. To set up the ANY method for integration with the HTTP back end, do the following:
 - a. Choose **HTTP Proxy** for **Integration type**.
 - b. Type an HTTP back-end resource URL in **Endpoint URL**.
 - c. Choose **Save** to finish configuring the ANY method.

For this tutorial, use `http://petstore-demo-endpoint.execute-api.com/ {proxy}` for the **Endpoint URL**.

/{proxy+} - ANY - Setup

API Gateway will configure your ANY method as a proxy integration. Proxy integrations can communicate with HTTP endpoints or Lambda functions. API Gateway sends the entire request to HTTP endpoints, including resource path, query string parameters, and body. For Lambda integrations, API Gateway applies a default mapping to send all of request information and responses follow a default interface. To learn more read our [documentation](#)

Integration type Lambda Function Proxy ⓘ
 HTTP Proxy ⓘ

Endpoint URL

Test an API with HTTP Proxy Integration through Proxy Resource

In the API we just created, the API's proxy resource path of `{proxy+}` becomes the placeholder of any of the back-end endpoints under `http://petstore-demo-endpoint.execute-api.com/`. For example, it can be `petstore`, `petstore/pets`, and `petstore/pets/{petId}`. The ANY method serves as a placeholder for any of the supported HTTP verbs at run time. These are illustrated in the following testing procedure.

To test an API integrated with the PetStore website using HTTP proxy integration through the proxy resource

1. To use the API Gateway console to test invoking the API, do the following.
 - a. Choose **ANY** on a proxy resource in the **Resources** tree.
 - b. Choose **Test** in the **Method Execution** pane.
 - c. From the **Method** drop-down list, choose an HTTP verb supported by the back end.
 - d. Under **Path**, type a specific path for the proxy resource supporting the chosen operation.
 - e. If required, type a supported query expression for the chosen operation under the **Query Strings** heading.
 - f. If required, type one or more supported header expressions for the chosen operation under the **Headers** heading.
 - g. If configured, set the required stage variable values for the chosen operation under the **Stage Variables** heading.
 - h. If prompted and required, choose an API Gateway-generated client certificate under the **Client Certificate** heading to the operation to be authenticated by the back end.
 - i. If prompted, type an appropriate request body in the text editor under the **Request Body** heading.
 - j. Choose **Test** to test invoking the method.

For this tutorial, use GET for the ANY method, use `petstore/pets` in place of the proxy resource path (`{proxy}`), and `type=fish` for the query string.

← Method Execution /{proxy+} - ANY - Method Test

Make a test call to your method with the provided input

Method

GET

Path

{proxy}

petstore/pets

Query Strings

{proxy}

type=fish

Headers

Use a colon (:) to separate header name and value, and new lines to declare multiple headers. eg.
Accept:application/json.

{proxy}

Stage Variables

No [stage variables](#) exist for this method.

Client Certificate

No client certificates have been generated.

Request Body

Request Body is not supported for GET methods.

⚡ Test

Because the back-end website supports the `GET /petstore/pets?type=fish` request, it will return a successful response similar to the following:

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

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

If you try to call `GET /petstore`, you will get a 404 response with an error message of `Cannot GET /petstore`. This is because the back end does not support the specified operation. On the other, if you call `GET /petstore/pets/1`, you will get a 200 OK response with the following payload, because the request is supported by the PetStore website.

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

2. To use a browser to call a GET method on a specific resource of the API, do the following.
 - a. If you have not done so, choose **Deploy API** from the **Actions** drop-down menu for the API you created. Follow the instructions to deploy the API to a specific stage. Note the **Invoke URL** that displays on the resulting **Stage Editor** page. This is the base URL of the API.
 - b. To submit a GET request on a specific resource, append the resource path, including possible query string expressions to the **Invoke URL** value obtained in the previous step, copy the complete URL into the address bar of a browser, and choose Enter.

For this tutorial, deploy the API to a test stage, append `petstore/pets?type=fish` to the API's Invoke URL. This produces a URL of `https://4z9giyi2c1.execute-api.us-west-2.amazonaws.com/test/petstore/pets?type=fish`.

The result should be the same as the result that is returned when you use `TestInvoke` from the API Gateway console.

Create an API with Lambda Proxy Integration through a Proxy Resource

As a prerequisite, we create a Lambda function as the back end of our API through the Lambda proxy integration with a proxy resource. As an illustration, we will create the following Node.js function, named `SimpleLambda4ProxyResource`, using the AWS Lambda console.

We then create an API with the Lambda proxy integration by using the `SimpleLambda4ProxyResource` function through a proxy resource by using the API Gateway console. In conclusion, we demonstrate how to test the API.

Topics

- [Lambda Function for Proxy Integration \(p. 60\)](#)
- [Create a Back End for Lambda Proxy Integration \(p. 61\)](#)
- [Create API with Lambda Proxy Integration \(p. 61\)](#)
- [Test API with Lambda Proxy Integration \(p. 62\)](#)

Lambda Function for Proxy Integration

The following Lambda function in Node.js is a "Hello, World!" application. The function shows how to parse the input `event` parameter that contains a request made by a client to an API Gateway proxy resource. This resource is integrated with the function using the Lambda proxy integration. The function also demonstrates how to format the output of the Lambda function for API Gateway to return the results as an HTTP response. For more information about the input and output formats that this type of Lambda function must follow, see [Input Format of a Lambda Function for Proxy Integration \(p. 103\)](#) and [Output Format of a Lambda Function for Proxy Integration \(p. 105\)](#).

```
'use strict';
console.log('Loading hello world function');

exports.handler = function(event, context) {
    var name = "World";
    var responseCode = 200;
    console.log("request: " + JSON.stringify(event));
    if (event.queryStringParameters !== null && event.queryStringParameters !==
    undefined) {
        if (event.queryStringParameters.name !==
        undefined && event.queryStringParameters.name !== null &&
        event.queryStringParameters.name !== "") {
            console.log("Received name: " +
            event.queryStringParameters.name);
            name = event.queryStringParameters.name;
        }

        if (event.queryStringParameters.httpStatus !==
        undefined && event.queryStringParameters.httpStatus !== null &&
        event.queryStringParameters.httpStatus !== "") {
            console.log("Received http status: " +
            event.queryStringParameters.httpStatus);
            responseCode = event.queryStringParameters.httpStatus;
        }
    }

    var responseBody = {
        message: "Hello " + name + "!",
        input: event
    };
    var response = {
        statusCode: responseCode,
        headers: {
            "x-custom-header" : "my custom header value"
        },
        body: JSON.stringify(responseBody)
    };
    console.log("response: " + JSON.stringify(response))
    context.succeed(response);
};
```

When used with a proxy resource in API Gateway, the input contains an API request marshalled by API Gateway. The input can include the request's HTTP method, path, query parameters, headers, and payload as well as the context and stage variables. This example Lambda function parses the query string parameters of `name` and `httpStatus` from the input (`event`). The function then returns a greeting to the named user in the `message` property of the `responseBody` object. The function also outputs the caller-supplied HTTP status code as part of a `response` object. If `name` and `httpStatus`

query parameters are not set, the default values ("World" and 200, respectively) are returned instead. In any case, the response body contains `event`, which is the input to the Lambda function. When structured as shown, the Lambda function output is un-marshalled by API Gateway and returned to the client as an HTTP response.

Create a Back End for Lambda Proxy Integration

Now let's create the Lambda function in API Gateway using the Lambda management console.

Create a Lambda function for an API with a proxy resource in the Lambda console

For this tutorial, use `LambdaForSimpleProxy` as the function name and choose the standard **Simple Microservice permissions** policy templates to create the required Lambda execution role.

1. Sign in to the Lambda console at <https://console.aws.amazon.com/lambda>.
2. From the upper-right corner of the console, choose an available region for the Lambda function.
3. From the main navigation pane, choose **Functions**.
4. Choose **Create a Lambda function** or **Get Started Now** to create your first Lambda function in a region, and then do the following.
 - a. On the **Select blueprint** page, choose **Skip**.
 - b. On the **Configure triggers** page, choose **Next**.
 - c. On the **Configure function** page, do the following:
 - Type a function name in the **Name** input field.
 - Type a brief function description in the **Description** input field.
 - From the **Runtime** drop-down list, choose **Node.js 4.3**.
 - d. Under **Lambda function code**, do the following:
 - Choose **Edit code inline** from the **Code entry type** drop-down list.
 - Type or copy your Node.js code into the inline code editor.
 - e. Under **Lambda function handler and role**, do the following:
 - Leave `index.handler` as-is for **Handler**.
 - Choose an existing or create a new IAM role for the function execution.
 - f. Keep the default values for **Advanced settings**.
 - g. Choose **Next**.
 - h. In the **Review** pane, choose **Create function**.

Note

Make note of the region where you created the Lambda function. You will need it next when creating the API for the function.

Create API with Lambda Proxy Integration

Now create an API with a proxy resource for a Lambda function by using the API Gateway console.

Build an API with a proxy resource for a Lambda function

1. Sign in to the API Gateway console at <https://console.aws.amazon.com/apigateway>.
2. To create an API, choose **Create new API** (for creating the first API) or **Create API** (for creating any subsequent API). Next, do the following:

- a. Choose **New API**.
- b. Type a name in **API Name**.
- c. Optionally, add a brief description in **Description**.
- d. Choose **Create API**.

For this tutorial, use `LambdaSimpleProxy` as the API name.

3. To create a child resource, choose a parent resource item under the **Resources** tree and then choose **Create Resource** from the **Actions** drop-down menu. Then, do the following in the **New Child Resource** pane.
 - a. Select the **Configure as proxy resource** option to create a proxy resource. Otherwise, leave it de-selected.
 - b. Type a name in the **Resource Name*** input text field.
 - c. Type a new name or use the default name in the **Resource Path*** input text field.
 - d. Choose **Create Resource**.
 - e. Select **Enable API Gateway CORS**, if required.

For this tutorial, use the root resource (`/`) as the parent resource. Select **Configure as proxy resource**. For **Resource Name**, use the default, `proxy`. For **Resource Path**, use `/{proxy+}`. De-select **Enable API Gateway CORS**.

4. To set up the `ANY` method for integration with the Lambda back end, do the following:
 - a. Choose **Lambda Function Proxy** for **Integration type**.
 - b. Choose a region from **Lambda Region**.
 - c. Type the name of your Lambda function in **Lambda Function**.
 - d. Choose **Save**.
 - e. Choose **OK** when prompted with **Add Permission to Lambda Function**.

For this tutorial, use the previously created `LambdaForProxyResource` for the **Lambda Function**.

For the proxy resource API that Lambda just created, API Gateway forwards the raw request from the client to the back end for the Lambda function to process. The request includes the request method, its path, query string and headers parameters, any payload, and context and stage variables. The next procedure describes how to test this.

Test API with Lambda Proxy Integration

The following procedure describes how to test the proxy integration.

Call the `LambdaForProxyResource` Lambda function through the proxy resource

1. To use a browser to call a GET method on a specific resource of the API, do the following.
 - a. If you have not done so, choose **Deploy API** from the **Actions** drop-down menu for the API you created. Follow the instructions to deploy the API to a specific stage. Note the **Invoke URL** that displays on the resulting **Stage Editor** page. This is the base URL of the API.
 - b. To submit a GET request on a specific resource, append the resource path, including possible query string expressions to the **Invoke URL** value obtained in the previous step, copy the complete URL into the address bar of a browser, and choose Enter.

For this tutorial, deploy the API to a test stage and append `hello?name=me` to the API's base URL to produce a URL of `https://wt6mne2s9k.execute-api.us-west-2.amazonaws.com/test/hello?name=me`.

The successful response returns a result similar to the following output from the back-end Lambda function. The input property captures the raw request from API Gateway. The response contains `httpMethod`, `path`, `headers`, `pathParameters`, `queryStringParameters`, `requestContext`, and `stageVariables`.

```
{
  "message": "Hello me!",
  "input": {
    "path": "/test/hello",
    "headers": {
      "Accept": "text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8",
      "Accept-Encoding": "gzip, deflate, lzma, sdch, br",
      "Accept-Language": "en-US,en;q=0.8",
      "CloudFront-Forwarded-Proto": "https",
      "CloudFront-Is-Desktop-Viewer": "true",
      "CloudFront-Is-Mobile-Viewer": "false",
      "CloudFront-Is-SmartTV-Viewer": "false",
      "CloudFront-Is-Tablet-Viewer": "false",
      "CloudFront-Viewer-Country": "US",
      "Host": "wt6mne2s9k.execute-api.us-west-2.amazonaws.com",
      "Upgrade-Insecure-Requests": "1",
      "User-Agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/52.0.2743.82 Safari/537.36 OPR/39.0.2256.48",
      "Via": "1.1 fb7cca60f0ecd82ce07790c9c5eef16c.cloudfront.net (CloudFront)",
      "X-Amz-Cf-Id": "nBsWBOOrSHMgnaROZJKlwGCZ9PcRcSpq_oSXZNQwQ10OTZL4cimZo3g==",
      "X-Forwarded-For": "192.168.100.1, 192.168.1.1",
      "X-Forwarded-Port": "443",
      "X-Forwarded-Proto": "https"
    },
    "pathParameters": {"proxy": "hello"},
    "requestContext": {
      "accountId": "123456789012",
      "resourceId": "us4z18",
      "stage": "test",
      "requestId": "41b45ea3-70b5-11e6-b7bd-69b5aaebc7d9",
      "identity": {
        "cognitoIdentityPoolId": "",
        "accountId": "",
        "cognitoIdentityId": "",
        "caller": "",
        "apiKey": "",
        "sourceIp": "192.168.100.1",
        "cognitoAuthenticationType": "",
        "cognitoAuthenticationProvider": "",
        "userArn": "",
        "userAgent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/52.0.2743.82 Safari/537.36 OPR/39.0.2256.48",
        "user": ""
      }
    }
  }
}
```

```
    },  
    "resourcePath": "/{proxy+}",  
    "httpMethod": "GET",  
    "apiId": "wt6mne2s9k"  
  },  
  "resource": "/{proxy+}",  
  "httpMethod": "GET",  
  "queryStringParameters": {"name": "me"},  
  "stageVariables": {"stageVarName": "stageVarValue"}  
}
```

You can use a different proxy resource path of `/hello/world?name=me` to call the Lambda function. The `input.path` property value of the JSON payload of the successful response will become `test/hello/world`.

If you use `POST` for the `ANY` method instead, the `input` property of the response payload contains a `body` JSON property. To test a `POST` request, you can use the API Gateway console `TestInvoke` feature, a `Curl` command, the `Postman` extension, or an `AWS SDK`.

2. To use the API Gateway console to test invoking the API, do the following.
 - a. Choose **ANY** on a proxy resource in the **Resources** tree.
 - b. Choose **Test** in the **Method Execution** pane.
 - c. From the **Method** drop-down list, choose an HTTP verb supported by the back end.
 - d. Under **Path**, type a specific path for the proxy resource supporting the chosen operation.
 - e. If required, type a supported query expression for the chosen operation under the **Query Strings** heading.
 - f. If required, type one or more supported header expressions for the chosen operation under the **Headers** heading.
 - g. If configured, set the required stage variable values for the chosen operation under the **Stage Variables** heading.
 - h. If prompted and required, choose an API Gateway-generated client certificate under the **Client Certificate** heading to the operation to be authenticated by the back end.
 - i. If prompted, type an appropriate request body in the text editor under the **Request Body** heading.
 - j. Choose **Test** to test invoking the method.

For this tutorial, use `GET` as the HTTP method, `hello` as the proxy resource path of `{proxy}`, and `name=me` as the query expression.

The successful response will return a JSON payload that is similar to the payload shown in the previous step.

Create an AWS Service Proxy for Amazon SNS

In addition to exposing Lambda functions or HTTP endpoints, you can also create an API Gateway API as a proxy to an AWS service, such as Amazon SNS, Amazon S3, Amazon Kinesis, enabling your client to access the back-end AWS services through your APIs. In this walkthrough, we illustrate this by creating an API to expose Amazon SNS. For examples of integrating an API with other AWS services, see [Tutorials \(p. 343\)](#).

An AWS service proxy can call only one action in an AWS service, and that action typically does not change. If you want more flexibility, you should call a Lambda function instead.

API Gateway does not retry when the endpoint times out. The API caller must implement a retry logic to handle endpoint timeouts.

This walkthrough builds on the instructions and concepts in the [Build an API to Expose a Lambda Function \(p. 44\)](#), which shows you how to use API Gateway to create a custom API, connect it to a set of AWS Lambda functions, and then call the Lambda functions from your API. If you have not yet completed that walkthrough, we suggest that you do it first.

Topics

- [Prerequisites \(p. 65\)](#)
- [Step 1: Create the Resource \(p. 65\)](#)
- [Step 2: Create the GET Method \(p. 66\)](#)
- [Step 3: Create the AWS Service Proxy Execution Role \(p. 66\)](#)
- [Step 4: Specify Method Settings and Test the Method \(p. 67\)](#)
- [Step 5: Deploy the API \(p. 68\)](#)
- [Step 6: Test the API \(p. 68\)](#)
- [Step 7: Clean Up \(p. 69\)](#)

Prerequisites

Before you begin this walkthrough, you should have already done the following:

1. Complete the steps in [Get Ready to Use API Gateway \(p. 4\)](#).
2. Make sure the IAM user has access to create policies and roles in IAM. You will need to create an IAM policy and role in this walkthrough.
3. At a minimum, open the API Gateway console and create a new API named `MyDemoAPI`. For more information, see [Build an API Gateway API to Expose an HTTP Endpoint \(p. 6\)](#).
4. Deploy the API at least once to a stage named `test`. For more information, see [Deploy the API \(p. 50\)](#) in the [Build an API to Expose a Lambda Function \(p. 44\)](#).
5. Complete the rest of the steps in the [Build an API to Expose a Lambda Function \(p. 44\)](#).
6. Create at least one topic in Amazon Simple Notification Service (Amazon SNS). You will use the deployed API to get a list of topics in Amazon SNS that are associated with your AWS account. To learn how to create a topic in Amazon SNS, see [Create a Topic](#). (You do not need to copy the topic ARN mentioned in step 5.)

Step 1: Create the Resource

In this step, you will create a resource that will enable the AWS service proxy to interact with the AWS service.

To create the resource

1. Sign in to the API Gateway console at <https://console.aws.amazon.com/apigateway>.
2. If **MyDemoAPI** is displayed, choose **Resources**.
3. In the **Resources** pane, choose the resource root, represented by a single forward slash (/), and then choose **Create Resource**.
4. For **Resource Name**, type `MyDemoAWSProxy`, and then choose **Create Resource**.

Step 2: Create the GET Method

In this step, you will create a GET method that will enable the AWS service proxy to interact with the AWS service.

To create the GET method

1. In the **Resources** pane, choose **/mydemoawsproxy**, and then choose **Create Method**.
2. For the HTTP method, choose **GET**, and then save your choice.

Step 3: Create the AWS Service Proxy Execution Role

In this step, you will create an IAM role that your AWS service proxy will use to interact with the AWS service. We call this IAM role an AWS service proxy execution role. Without this role, API Gateway cannot interact with the AWS service. In later steps, you will specify this role in the settings for the GET method you just created.

To create the AWS service proxy execution role and its policy

1. Sign in to the Identity and Access Management (IAM) console at <https://console.aws.amazon.com/iam/>.
2. Choose **Policies**.
3. Do one of the following:
 - If the **Welcome to Managed Policies** page appears, choose **Get Started**, and then choose **Create Policy**.
 - If a list of policies appears, choose **Create Policy**.
4. Next to **Create Your Own Policy**, choose **Select**.
5. For **Policy Name**, type a name for the policy (for example, **APIGatewayAWSProxyExecPolicy**).
6. For **Description**, type **Enables API Gateway to call AWS services**.
7. For **Policy Document**, type the following, and then choose **Create Policy**.

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

Note

This policy document allows the caller to get a list of the Amazon SNS topics for the AWS account.

8. Choose **Roles**.
9. Choose **Create New Role**.
10. For **Role Name**, type a name for the execution role (for example, **APIGatewayAWSProxyExecRole**), and then choose **Next Step**.
11. Next to **Amazon EC2**, choose **Select**.

Note

You choose **Select** here because you need to choose a standard AWS service role statement before you can continue. There is currently no option to choose a standard API Gateway service role statement. Later in this step, you will modify the standard Amazon EC2 service role statement for use with API Gateway.

12. In the list of policies, select **APIGatewayAWSProxyExecPolicy**, and then choose **Next Step**.
13. For **Role ARN**, make a note of the Amazon Resource Name (ARN) for the execution role. You will need it later. The ARN should look similar to: `arn:aws:iam::123456789012:role/APIGatewayAWSProxyExecRole`, where 123456789012 is your AWS account ID.
14. Choose **Create Role**.

The invocation role IAM just created enables Amazon EC2 to get a list of the Amazon SNS topics for the AWS account. You will change this role to enable API Gateway to get a list of the Amazon SNS topics for the AWS account instead.

15. In the list of roles, select **APIGatewayAWSProxyExecRole**.
16. In the **Trust Relationships** area, choose **Edit Trust Relationship**.
17. For **Policy Document**, replace `ec2.amazonaws.com` with `apigateway.amazonaws.com` so that the access control policy document now looks as follows:

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

This policy document enables API Gateway to take actions on behalf of your AWS account.

18. Choose **Update Trust Policy**.

Step 4: Specify Method Settings and Test the Method

In this step, you will specify the settings for the GET method so that it can interact with an AWS service through an AWS service proxy. You will then test the method.

To specify settings for the GET method and then test it

1. In the API Gateway console, in the **Resources** pane for the API named `MyDemoAPI`, in `/mydemoawsproxy`, choose **GET**.

2. In the **Setup** pane, for **Integration type**, choose **Show advanced**, and then choose **AWS Service Proxy**.
3. For **AWS Region**, choose the name of the AWS region where you want to get the Amazon SNS topics.
4. For **AWS Service**, choose **SNS**.
5. For **HTTP method**, choose **GET**.
6. For **Action**, type `ListTopics`.
7. For **Execution Role**, type the ARN for the execution role.
8. Leave **Path Override** blank.
9. Choose **Save**.
10. In the **Method Execution** pane, in the **Client** box, choose **TEST**, and then choose **Test**. If successful, **Response Body** will display a response similar to the following:

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

Step 5: Deploy the API

In this step, you will deploy the API so that you can begin calling it from outside of the API Gateway console.

To deploy the API

1. In the **Resources** pane, choose **Deploy API**.
2. For **Deployment stage**, choose `test`.
3. For **Deployment description**, type `Calling AWS service proxy walkthrough`.
4. Choose **Deploy**.

Step 6: Test the API

In this step, you will go outside of the API Gateway console and use your AWS service proxy to interact with the Amazon SNS service.

1. In the **Stage Editor** pane, next to **Invoke URL**, copy the URL to the clipboard. It should look like this:

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

2. Paste the URL into the address box of a new browser tab.
3. Append `/mydemoawsproxy` so that it looks like this:

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

Browse to the URL. Information similar to the following should be displayed:

```
{"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"}}
```

Step 7: Clean Up

You can delete the IAM resources the AWS service proxy needs to work.

Caution

If you delete an IAM resource an AWS service proxy relies on, that AWS service proxy and any APIs that rely on it will no longer work. Deleting an IAM resource cannot be undone. If you want to use the IAM resource again, you must re-create it.

To delete the associated IAM resources

1. Open the Identity and Access Management (IAM) console at <https://console.aws.amazon.com/iam/>.
2. In the **Details** area, click **Roles**.
3. Select **APIGatewayAWSPProxyExecRole**, and then choose **Role Actions, Delete Role**. When prompted, choose **Yes, Delete**.
4. In the **Details** area, choose **Policies**.
5. Select **APIGatewayAWSPProxyExecPolicy**, and then choose **Policy Actions, Delete**. When prompted, choose **Delete**.

You have reached the end of this walkthrough. For more detailed discussions about creating API as an AWS service proxy, see [Create an API as an Amazon S3 Proxy \(p. 359\)](#), [Create an API as a Lambda Proxy \(p. 343\)](#) or [Create an API as an Amazon Kinesis Proxy \(p. 388\)](#).

Creating an API in Amazon API Gateway

Topics

- [Create an API in API Gateway \(p. 70\)](#)
- [Set up API Gateway API Method and Integration \(p. 71\)](#)
- [Set Up Amazon API Gateway API Request and Response Payload Mappings \(p. 105\)](#)
- [Amazon API Gateway API Request and Response Parameter-Mapping Reference \(p. 131\)](#)
- [API Gateway API Request and Response Payload-Mapping Template Reference \(p. 134\)](#)
- [Import and Export API Gateway API with Swagger Definition Files \(p. 141\)](#)

Create an API in API Gateway

In Amazon API Gateway you can create an API using the API Gateway console, AWS CLI, the API Gateway control service REST API, and platform-specific or language-specific SDKs.

Topics

- [Create an API Using the API Gateway Console \(p. 70\)](#)
- [Create an API Using the API Gateway Control Service API \(p. 71\)](#)
- [Create an API Using the AWS SDK for API Gateway \(p. 71\)](#)
- [Create an API Using the AWS CLI \(p. 71\)](#)

Create an API Using the API Gateway Console

To create an API Gateway API using the API Gateway console, see [Build an API Gateway API to Expose an HTTP Endpoint \(p. 6\)](#).

You can learn how to create an API by following an example. For more information, see [Create an API Gateway API from an Example \(p. 7\)](#).

Alternatively, you can create an API by using the API Gateway [Import API \(p. 142\)](#) feature to upload an external API definition, such as one expressed in the [Swagger 2.0 with the API Gateway Extensions to Swagger \(p. 147\)](#). The example provided in [Create an API Gateway API from an Example \(p. 7\)](#) uses the Import API feature.

Create an API Using the API Gateway Control Service API

For more information about the API Gateway Control Service API, see [Amazon API Gateway REST API Reference](#).

Create an API Using the AWS SDK for API Gateway

For more information using a AWS SDK, see [AWS SDKs](#).

Create an API Using the AWS CLI

For an example of creating an API Gateway API Using AWS CLI, see [Create an API Gateway API for Lambda tutorial](#).

Set up API Gateway API Method and Integration

Before Setting Up Methods and Integration

- You must have the method available in API Gateway. Follow the instructions in [Build an API Gateway API to Expose an HTTP Endpoint \(p. 6\)](#).
- If you want the method to communicate with a Lambda function, you must have already created the Lambda invocation role and Lambda execution role in IAM and created the Lambda function with which your method will communicate in AWS Lambda. To create the roles and function, use the instructions in [Step 4: Create Lambda Functions \(p. 44\)](#) of the [Build an API to Expose a Lambda Function \(p. 44\)](#).
- If you want the method to communicate with an HTTP or HTTP proxy integration, you must have already created and have access to the HTTP endpoint URL with which your method will communicate.
- Verify that your certificates for HTTP and HTTP proxy endpoints are supported by API Gateway. For details see [Supported Certificate Authorities for HTTP and HTTP Proxy Integration \(p. 189\)](#)

Topics

- [Configure How API Gateway Integrates the Method with a Back End \(p. 72\)](#)
- [Configure How an API User Calls an API Method in Amazon API Gateway \(p. 74\)](#)
- [Configure How Data Is Mapped between a Method and its Integration in Amazon API Gateway \(p. 75\)](#)
- [Enable Support for Binary Payloads in API Gateway \(p. 78\)](#)
- [Configure Mock Integration for a Method in API Gateway \(p. 94\)](#)
- [Configure Proxy Integration for a Proxy Resource \(p. 97\)](#)

- [After Setting Up Methods and Integration \(p. 105\)](#)

Configure How API Gateway Integrates the Method with a Back End

The settings of an API method defines the method and describes its behaviors. To create a method, you must specify a resource, including the root ("/"), on which the method is exposed, a method type (GET, POST, etc.), and how it will be integrated with the targeted back end. The method request and response specify the contract with the calling app, stipulating which parameters the API can receive and what the response looks like. The integration request and response specifies how API Gateway interacts with their back end: enforcing secure communications over HTTPS with the back end and translating data formats between the client and back end. The following topics describe how to use the API Gateway console to specify a method settings.

1. In the **Resources** pane, choose the method.
2. In the **Method Execution** pane, choose **Integration Request**. For **Integration type**, choose one of the following:
 - Choose **Lambda Function** if your API will be communicating with a Lambda function.
 - Choose **HTTP Proxy** if your API will be communicating with an HTTP endpoint.
 - Choose **Show Advanced, AWS Service Proxy** if your API will be communicating directly with an AWS service.
 - Choose **Mock Integration** if your API is not yet final, but you want to generate API responses from API Gateway anyway to unblock dependent teams for testing. If you choose this option, skip the rest of the instructions in this topic and see [Configure Mock Integration for a Method \(p. 94\)](#).
3. If you chose **Lambda Function**, do the following:
 1. For **Lambda Region**, choose the region identifier that corresponds to the region where you created the Lambda function. For example, if you created the Lambda function in the US East (N. Virginia) region, you would choose `us-east-1`. For a list of region names and identifiers, see [AWS Lambda](#) in the *Amazon Web Services General Reference*.
 2. For **Lambda Function**, type the name of the Lambda function, and then choose the function's corresponding ARN.
 3. Choose **Save**.
4. If you chose **HTTP Proxy**, do the following:
 1. For **HTTP method**, choose the HTTP method type that most closely matches the method in the HTTP proxy.
 2. For **Endpoint URL**, type the URL of the HTTP proxy you want this method to use.
 3. Choose **Save**.
5. If you chose **Mock Integration**, do the following:
 - Choose **Save**.
6. If you chose **Show advanced, AWS Service Proxy**, do the following:
 1. For **AWS Region**, choose the AWS region you want this method to use to call the action.
 2. For **AWS Service**, choose the AWS service you want this method to call.
 3. For **HTTP method**, choose the HTTP method type that corresponds to the action. For HTTP method type, see the API reference documentation for the AWS service you chose for **AWS Service**.
 4. For **Action**, type the action you want to use. For a list of available actions, see the API reference documentation for the AWS service you chose for **AWS Service**.

5. For **Execution Role**, type the ARN of the IAM role the method will use to call the action.

To create the IAM role, you can adapt the instructions in "To create the Lambda invocation role and its policies" and "To create the Lambda execution role and its policy" in the [Create Lambda Functions \(p. 44\)](#) section of the [Build an API to Expose a Lambda Function \(p. 44\)](#); and specify an access policy of the following format, with the desired number of action and resource statements:

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

For the action and resource statement syntax, see the documentation for the AWS service you chose for **AWS Service**.

For the IAM role's trust relationship, specify the following, which enables API Gateway to take actions on behalf of your AWS account:

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

6. If the action you typed for **Action** provides a custom resource path you want this method to use, for **Path Override**, type this custom resource path. For the custom resource path, see the API reference documentation for the AWS service you chose for **AWS Service**.
7. Choose **Save**.
7. Do both of the following:
 - Specify how the method will receive requests from, and send responses to, callers (which API Gateway refers to as the API's method request/response), and how the method will authorize requests by following the instructions in [Configure How a User Calls an API Method \(p. 74\)](#).
 - Specify how the method will send requests to, and receive responses from, the Lambda function, HTTP proxy, or AWS service proxy (which API Gateway refers to as the API's integration request/response) by following the instructions in [Configure How Data Is Mapped between Method and Integration \(p. 75\)](#).

Configure How an API User Calls an API Method in Amazon API Gateway

To use the API Gateway console to specify an API's method request/response and the way in which the method will authorize requests, follow these instructions.

Note

These instructions assume you have already completed the steps in [Configure How a Method Is Integrated with a Back End \(p. 72\)](#).

1. With the method selected in the **Resources** pane, in the **Method Execution** pane, choose **Method Request**.
2. To assign custom access permissions to the method, in the **Authorization Settings** area, for **Authorization Type**, choose **Edit**, and then choose **AWS_IAM**. Only IAM roles with the correct IAM policy attached will be allowed to call this method. If you do not want to assign custom access permissions to the method, choose **NONE**.

- To create the IAM role, specify an access policy with a format like the following:

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

In this access policy, *resource-statement* is the value of the **ARN** field in the **Authorization Settings** section.

To create the IAM role, you can adapt the instructions in "To create the Lambda invocation role and its policy" and "To create the Lambda execution role and its policy" in the [Create Lambda Functions \(p. 44\)](#) section of the [Build an API to Expose a Lambda Function \(p. 44\)](#).

To save your choice, choose **Update**. Otherwise, choose **Cancel**.

Note

You can also enable an API key. For instructions, see [Use an API Key without Usage Plans \(p. 223\)](#).

3. To add a query string parameter to the method, do the following:
 - a. Choose the arrow next to **URL Query String Parameters**, and then choose **Add query string**.
 - b. For **Name**, type the name of the query string parameter.
 - c. Choose **Create a new query string**.

Note

To remove the query string parameter, choose **Cancel** or **Remove**.

To change the name of the query string parameter, you must remove it and create a new one.

4. To add a header parameter to the method, do the following:
 - a. Choose the arrow next to **HTTP Request Headers**, and then choose **Add header**.
 - b. For **Name**, type the name of the header parameter.
 - c. Optionally, check the **Caching** option to make this method as an API cache key. For more information, see [Use Method/Integration Parameters as Cache Keys \(p. 239\)](#).
 - d. Choose **Create**

Tip

To remove the header parameter, choose **Cancel** or **Remove**.

To change the name of the header parameter, you must remove the old header parameter and create a new one in its place.

5. For non-GET method types, expand **Request Models**, and for **Content Type** and **Model name**, type the content type and choose the name of a model that will transform caller-supplied data into the expected format.

To create a model, see [Create a Model \(p. 112\)](#).

6. To send a set of custom response headers, a custom response data format, or both, back to callers based on the HTTP status code returned by the method, do the following:
 - a. In the **Method Execution** pane, choose **Method Response**. By default, 200 response is included in the method responses. You can modify it, e.g., to have the method return 201 instead. In addition, you can add other responses, e.g., 409 for access denial and 500 for uninitialized stage variables used. Either choose the arrow icon next to **200** to specify settings for the 200 response, or choose **Add Response** to specify settings for any other HTTP response status code. If you choose **Add Response**, for **HTTP Status**, choose the response, choose **Create**, and choose the arrow next to the response.

Tip

You will use **Method Response** to specify all possible response codes for your API and use **Integration Response** to indicate to API Gateway how back-end errors are mapped to an HTTP status code.

- b. For each custom header you want to include in the response, in the **Response Headers** area, choose **Add Header**, type the name of the header, and then choose **Save**. (Choose **Remove** to remove a header from this list.)

To specify a response model to transform the output's data from one format to another, in the **Response Models** area, choose **Add Response Model**. Type the content type (for **Content type**), choose the model's name (for **Models**), and then choose **Save**. Choose **Add Response Model** to specify an additional model, or choose **Create a model** to define a new model. (Choose **Remove** to remove a response model selection from this list.)

Configure How Data Is Mapped between a Method and its Integration in Amazon API Gateway

Note

API Gateway does not currently support binary payloads. Binary data can be passed around in a payload as a JSON property value of a Base64-encoded string.

To use the API Gateway console to define the API's integration request/response, follow these instructions.

Note

These instructions assume you have already completed the steps in [Configure How a Method Is Integrated with a Back End](#) (p. 72).

1. With the method selected in the **Resources** pane, in the **Method Execution** pane, choose **Integration Request**.
2. For an HTTP proxy or an AWS service proxy, to associate a path parameter, a query string parameter, or a header parameter defined in the integration request with a corresponding path parameter, query string parameter, or header parameter in the method request of the HTTP proxy or AWS service proxy, do the following:
 - a. Choose the arrow next to **URL Path Parameters**, **URL Query String Parameters**, or **HTTP Headers** respectively, and then choose **Add path**, **Add query string**, or **Add header**, respectively.
 - b. For **Name**, type the name of the path parameter, query string parameter, or header parameter in the HTTP proxy or AWS service proxy.
 - c. For **Mapped from**, type the mapping value for the path parameter, query string parameter, or header parameter. Use one of the following formats:
 - `method.request.path.parameter-name` for a path parameter named *parameter-name* as defined in the **Method Request** page.
 - `method.request.querystring.parameter-name` for a query string parameter named *parameter-name* as defined in the **Method Request** page.
 - `method.request.header.parameter-name` for a header parameter named *parameter-name* as defined in the **Method Request** page.Alternatively, you can set a literal string value (enclosed by a pair of single quotes) to an integration header.
 - d. Choose **Create**. (To delete a path parameter, query string parameter, or header parameter, choose **Cancel** or **Remove** next to the parameter you want to delete.)
3. In the **Body Mapping Templates** area, choose an option for **Request body passthrough** to configure how the method request body of an unmapped content type will be passed through the integration request without transformation to the Lambda function, HTTP proxy, or AWS service proxy. There are three options:
 - Choose **When no template matches the request Content-Type header** if you want the method request body to pass through the integration request to the back end without transformation when the method request content type does not match any content types associated with the mapping templates, as defined in the next step.

Note

When calling the API Gateway API, you choose this option by setting `WHEN_NO_MATCH` as the `passthroughBehavior` property value on the [Integration](#) resource.

- Choose **When there are no templates defined (recommended)** if you want the method request body to pass through the integration request to the back end without transformation when no mapping template is defined in the integration request. If a template is defined when this option is selected, the method request of an unmapped content type will be rejected with an HTTP 415 Unsupported Media Type response.

Note

When calling the API Gateway API, you choose this option by setting `WHEN_NO_TEMPLATE` as the `passthroughBehavior` property value on the [Integration](#) resource.

- Choose **Never** if you do not want the method request to pass through when either the method request content type does not match any content type associated with the mapping

templates defined in the integration request or no mapping template is defined in the integration request. The method request of an unmapped content type will be rejected with an HTTP 415 Unsupported Media Type response.

Note

When calling the API Gateway API, you choose this option by setting `NEVER` as the `passthroughBehavior` property value on the [Integration](#) resource.

For more information about the integration passthrough behaviors, see [Integration Passthrough Behaviors](#) (p. 140).

4. To define a mapping template for an incoming request, choose **Add mapping template** under **Content-Type**. Type a content type (e.g., `application/json`) in the input text box and then choose the check mark icon to save the input. Then, type the mapping template manually or choose **Generate template** to create one from a model template. For more information, see [Set Up Payload Mappings](#) (p. 105).
5. You can map an integration response from the back-end to a method response of the API returned to the calling app. This includes returning to the client selected response headers from the available ones from the back end, transforming the data format of the back-end response payload to an API-specified format. You can specify such mapping by configuring **Method Response** and **Integration Response** from the **Method Execution** page.
 - a. In the **Method Execution** pane, choose **Integration Response**. Choose either the arrow next to **200** to specify settings for a 200 HTTP response code from the method, or choose **Add integration response** to specify settings for any other HTTP response status code from the method.
 - b. For **Lambda error regex** (for a Lambda function) or **HTTP status regex** (for an HTTP proxy or AWS service proxy), type a regular expression to specify which Lambda function error strings (for a Lambda function) or HTTP response status codes (for an HTTP proxy or AWS service proxy) map to this output mapping. For example, to map all 2xx HTTP response status codes from an HTTP proxy to this output mapping, type `2\d{2}` for **HTTP status regex**. To return an error message containing "Invalid Request" from a Lambda function to a 400 `Bad Request` response, type `.*Invalid request.*` as the **Lambda error regex** expression. On the other hand, to return 400 `Bad Request` for all unmapped error messages from Lambda, type `(\n|.)+` in **Lambda error regex**. This last regular expression can be used for the default error response of an API.

Note

The error patterns are matched against the entire string of the `errorMessage` property in the Lambda response, which is populated by `context.fail(errorMessage)` in Node.js or by `throw new MyException(errorMessage)` in Java. Also, escaped characters are unescaped before the regular expression is applied.

If you use `.` as the selection pattern to filter responses, be aware that it may not match a response containing a newline (`\n`) character.

- c. If enabled, for **Method response status**, choose the HTTP response status code you defined in the **Method Response** page.
- d. For **Header Mappings**, for each header you defined for the HTTP response status code in the **Method Response** page, specify a mapping value by choosing **Edit**. For **Mapping value**, use the format `integration.response.header.header-name` where `header-name` is the name of a response header from the backend. For example, to return the backend response's `Date` header as an API method's response's `Timestamp` header, the **Response header** column will contain an **Timestamp** entry and the associated **Mapping value** should be set to `integration.response.header.Date`.
- e. In the **Template Mappings** area, next to **Content type**, choose **Add**. In the **Content type** box, type the content type of the data that will be passed from the Lambda function, HTTP proxy, or AWS service proxy to the method. Choose **Update**.

- f. Select **Output passthrough** if you want the method to receive, but not modify, the data from the Lambda function, HTTP proxy, or AWS service proxy.
- g. If **Output passthrough** is cleared, for **Output mapping**, specify the output mapping template you want the Lambda function, HTTP proxy, or AWS service proxy to use to send data to the method. You can either type the mapping template manually or choose a model from **Generate template from model**.
- h. Choose **Save**.

Enable Support for Binary Payloads in API Gateway

In API Gateway, the API request and response can have a text or binary payload. By default, API Gateway treats the message body as a UTF-8-encoded JSON string. For API Gateway to handle binary payloads, you add the media types to the `binaryMediaTypes` list of the `RestApi` resource or set the `contentHandling` properties on the `Integration` and the `IntegrationResponse` resources. The `contentHandling` value can be `CONVERT_TO_BINARY`, `CONVERT_TO_TEXT`, or undefined. Depending on the `contentHandling` value, and whether the `Content-Type` header of the response or the `Accept` header of the incoming request matches an entry in the `binaryMediaTypes` list, API Gateway can encode the raw binary bytes as a Base64-encoded string, decode a Base64-encoded string back to its raw bytes, or pass the body through without modification.

You must configure the API as follows to support binary payloads for your API in API Gateway:

- Add the desired binary media types to the `binaryMediaTypes` list on the `RestApi` resource. If this property and the `contentHandling` property are not defined, the payloads are handled as UTF-8 encoded JSON strings.
- Set the `contentHandling` property of the `Integration` resource to `CONVERT_TO_BINARY` to have the request payload converted from a Base64-encoded string to its binary blob, or set the property to `CONVERT_TO_TEXT` to have the request payload converted from a binary blob to a Base64-encoded string. If this property is not defined, API Gateway passes the payload through without modification. This occurs when the `Content-Type` header value matches one of the `binaryMediaTypes` entries and the `passthrough behaviors` (p. 140) are also enabled for the API.
- Set the `contentHandling` property of the `IntegrationResponse` resource to `CONVERT_TO_BINARY` to have the response payload converted from a Base64-encoded string to its binary blob, or set the property to `CONVERT_TO_TEXT` to have the response payload converted from a binary blob to a Base64-encoded string. If `contentHandling` is not defined, and if the `Content-Type` header of the response and the `Accept` header of the original request match an entry of the `binaryMediaTypes` list, API Gateway passes through the body. This occurs when the `Content-Type` header and the `Accept` header are the same; otherwise, API Gateway converts the response body to the type specified in the `Accept` header.

Topics

- [Content Type Conversions in API Gateway](#) (p. 78)
- [Enable Binary Support Using the API Gateway Console](#) (p. 81)
- [Enable Binary Support Using API Gateway REST API](#) (p. 84)
- [Import and Export Content Encodings](#) (p. 87)
- [Examples of Binary Support](#) (p. 87)

Content Type Conversions in API Gateway

The following table shows how API Gateway converts the request payload for specific configurations of a request's `Content-Type` header, the `binaryMediaTypes` list of a `RestApi` resource, and the `contentHandling` property value of the `Integration` resource.

API Request Content Type Conversions in API Gateway

Method request payload	Request Content-Type header	binaryMediaType	contentHandling	Integration request payload
Text data	Any data type	Undefined	Undefined	UTF8-encoded string
Text data	Any data type	Undefined	CONVERT_TO_BINARY	Base64-decoded binary blob
Text data	Any data type	Undefined	CONVERT_TO_TEXT	UTF8-encoded string
Text data	A text data type	Set with matching media types	Undefined	Text data
Text data	A text data type	Set with matching media types	CONVERT_TO_BINARY	Base64-decoded binary blob
Text data	A text data type	Set with matching media types	CONVERT_TO_TEXT	Text data
Binary data	A binary data type	Set with matching media types	Undefined	Binary data
Binary data	A binary data type	Set with matching media types	CONVERT_TO_BINARY	Binary data
Binary data	A binary data type	Set with matching media types	CONVERT_TO_TEXT	Base64-encoded string

The following table shows how API Gateway converts the response payload for specific configurations of a request's `Accept` header, the `binaryMediaTypes` list of a [RestApi](#) resource, and the `contentHandling` property value of the [IntegrationResponse](#) resource.

API Gateway Response Content Type Conversions

Integration response payload	Request Accept header	binaryMediaType	contentHandling	Method response payload
Text or binary data	A text type	Undefined	Undefined	UTF8-encoded string
Text or binary data	A text type	Undefined	CONVERT_TO_BINARY	Base64-decoded blob
Text or binary data	A text type	Undefined	CONVERT_TO_TEXT	UTF8-encoded string
Text data	A text type	Set with matching media types	Undefined	Text data
Text data	A text type	Set with matching media types	CONVERT_TO_BINARY	Base64-decoded blob
Text data	A text type	Set with matching media types	CONVERT_TO_TEXT	UTF8-encoded string

Integration response payload	Request Accept header	binaryMediaType	contentHandling	Method response payload
Text data	A binary type	Set with matching media types	Undefined	Base64-decoded blob
Text data	A binary type	Set with matching media types	CONVERT_TO_BINARY	Base64-decoded blob
Text data	A binary type	Set with matching media types	CONVERT_TO_TEXT	UTF8-encoded string
Binary data	A text type	Set with matching media types	Undefined	Base64-encoded string
Binary data	A text type	Set with matching media types	CONVERT_TO_BINARY	Binary data
Binary data	A text type	Set with matching media types	CONVERT_TO_TEXT	Base64-encoded string
Binary data	A binary type	Set with matching media types	Undefined	Binary data
Binary data	A binary type	Set with matching media types	CONVERT_TO_BINARY	Binary data
Binary data	A binary type	Set with matching media types	CONVERT_TO_TEXT	Base64-encoded string

Tip

When a request contains multiple media types in its `Accept` header, API Gateway only honors the first `Accept` media type. In the situation where you cannot control the order of the `Accept` media types and the media type of your binary content is not the first in the list, you can add the first `Accept` media type in the `binaryMediaTypes` list of your API. API Gateway will return your content as binary. For example, to send a JPEG file using an `` element in a browser, the browser might send `Accept: image/webp, image/*, */*; q=0.8` in a request. By adding `image/webp` to the `binaryMediaTypes` list, the endpoint will receive the JPEG file as binary.

When converting a text payload to a binary blob, API Gateway assumes that the text data is a Base64-encoded string and outputs the binary data as a Base64-decoded blob. If the conversion fails, it returns a 500 response indicating an API configuration error. You do not provide a mapping template for such a conversion, although you must enable the [passthrough behaviors \(p. 140\)](#) on the API.

When converting a binary payload to a text string, API Gateway always applies a Base64 encoding on the binary data. You can define a mapping template for such a payload, but can only access the Base64-encoded string in the mapping template through `$input.body`, as shown in the following excerpt of an example mapping template.

```
{
  "data": "$input.body"
}
```

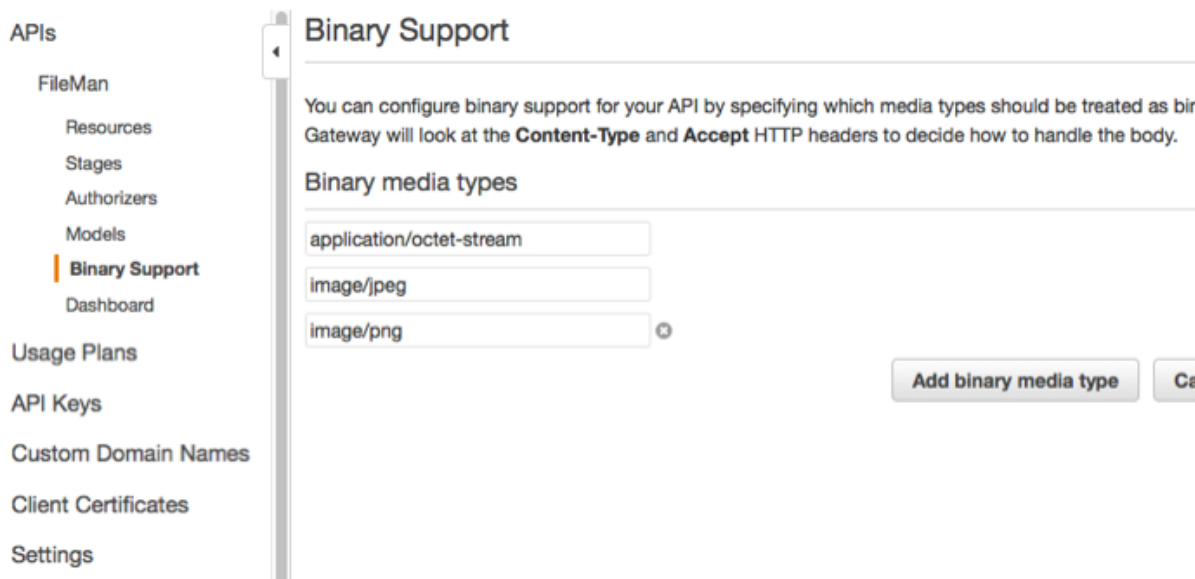
To have the binary payload passed through without modification, you must enable the [passthrough behaviors \(p. 140\)](#) on the API.

Enable Binary Support Using the API Gateway Console

The section explains how to enable binary support using the API Gateway console. As an example, we use an API integrated with Amazon S3. We focus on the tasks to set the supported media types and to specify how the payload should be handled. For detailed information on how to create an API integrated with Amazon S3, see [Create an API as an Amazon S3 Proxy \(p. 359\)](#).

To enable binary support using the API Gateway console

1. Set binary media types for the API:
 - a. Create a new API or choose an existing API. For this example, we name the API `FileMan`.
 - b. Choose **Binary Support** under the API.
 - c. In the **Binary Support** pane, choose **Edit**.
 - d. Choose **Add binary media types** and type a MIME type to be supported for your API.
 - e. Choose **Save**.



2. Set how message payloads are handled for the API method:
 - a. Create a new or choose an existing resource in the API. For this example, we use the `/ {folder} / {item}` resource.
 - b. Create a new or choose an existing method on the resource. As an example, we use the `GET / {folder} / {item}` method integrated with the `Object GET` action in Amazon S3.
 - c. In **Content Handling**, choose an option.

/{{folder}}/{{item}} - GET - Setup

Choose the integration point for your new method.

Integration type Lambda Function ⓘ
 HTTP ⓘ
 Mock ⓘ
 AWS Service ⓘ

AWS Region

AWS Service

AWS Subdomain

HTTP method

Action Type Use action name
 Use path override

Path override (optional)





Execution role
 ⓘ

Content Handling ⓘ

Choose **Passthrough** if you do not want to convert the body when the client and back end accepts the same binary format. Choose **Convert to text (if needed)** to convert the binary body to a Base64-encoded string when, for example, the back end requires that a binary request payload is passed in as a JSON property. And choose **Convert to binary (if needed)** when the client submits a Base64-encoded string and the back end requires the original binary format, or when the endpoint returns a Base64-encoded string and the client accepts only the binary output.

- d. Preserve the incoming request's `Accept` header in the integration request. You should do this if you've set `contentHandling` to `passthrough` and want to override that setting at run time.

▼ HTTP Headers

Name	Mapped from ⓘ	Caching	
Accept	method.request.header.Accept	<input type="checkbox"/>	 
Content-Type	method.request.header.Content-Type	<input type="checkbox"/>	 



 **Add header**

- e. Enable the passthrough behavior on the request body.

▼ Body Mapping Templates

Request body passthrough When no template matches the request Content-Type header ⓘ
 When there are no templates defined (recommended) ⓘ
 Never ⓘ

Content-Type

image/png  



 **Add mapping template**

- f. For conversion to text, define a mapping template to put the Base64-encoded binary data into the required format.

image/png

Generate template:

```
1 {  
2   "operation": "thumbnail",  
3   "base64Image": "$input.body"  
4 }
```

The format of this mapping template depends on the endpoint requirements of the input.

Enable Binary Support Using API Gateway REST API

The following tasks show how to enable binary support using the API Gateway REST API calls.

Topics

- [Add and Update Supported Binary Media Types to an API \(p. 84\)](#)
- [Configure Request Payload Conversions \(p. 84\)](#)
- [Configure Response Payload Conversions \(p. 85\)](#)
- [Convert Binary Data to Text Data \(p. 85\)](#)
- [Convert Text Data to a Binary Payload \(p. 86\)](#)
- [Pass through a Binary Payload \(p. 86\)](#)

Add and Update Supported Binary Media Types to an API

To enable API Gateway to support a new binary media type, you must add the binary media type to the `binaryMediaTypes` list of the `RestApi` resource. For example, to have API Gateway handle JPEG images, submit a `PATCH` request to the `RestApi` resource:

```
PATCH /restapis/<restapi_id>

{
  "patchOperations" : [ {
    "op" : "add",
    "path" : "/binaryMediaTypes/image~1jpeg"
  }
]
}
```

The MIME type specification of `image/jpeg` that is part of the `path` property value is escaped as `image~1jpeg`.

To update the supported binary media types, replace or remove the media type from the `binaryMediaTypes` list of the `RestApi` resource. For example, to change binary support from JPEG files to raw bytes, submit a `PATCH` request to the `RestApi` resource, as follows.

```
PATCH /restapis/<restapi_id>

{
  "patchOperations" : [{
    "op" : "replace",
    "path" : "/binaryMediaTypes/image~1jpeg",
    "value" : "application/octet-stream"
  },
  {
    "op" : "remove",
    "path" : "/binaryMediaTypes/image~1jpeg"
  }
]
}
```

Configure Request Payload Conversions

If the endpoint requires a binary input, set the `contentHandling` property of the `Integration` resource to `CONVERT_TO_BINARY`. To do so, submit a `PATCH` request, as shown next:


```
PATCH /restapis/<restapi_id>/resources/<resource_id>/methods/<http_method>/
integration

{
  "patchOperations" : [ {
    "op" : "replace",
    "path" : "/contentHandling",
    "value" : "CONVERT_TO_BINARY"
  } ]
}
```

Configure Response Payload Conversions

If the client accepts the result as a binary blob instead of a Base64-encoded payload returned from the endpoint, set the `contentHandling` property of the `IntegrationResponse` resource to `CONVERT_TO_BINARY` by submitting a `PATCH` request, as shown next:

```
PATCH /restapis/<restapi_id>/resources/<resource_id>/methods/<http_method>/
integration/responses/<status_code>

{
  "patchOperations" : [ {
    "op" : "replace",
    "path" : "/contentEncoding",
    "value" : "CONVERT_TO_BINARY"
  } ]
}
```

Convert Binary Data to Text Data

To send binary data as a JSON property of the input to AWS Lambda or Amazon Kinesis through API Gateway, do the following:

1. Enable the binary payload support of the API by adding the new binary media type of `application/octet-stream` to the API's `binaryMediaTypes` list.

```
PATCH /restapis/<restapi_id>

{
  "patchOperations" : [ {
    "op" : "add",
    "path" : "/binaryMediaTypes/application~1octet~1stream"
  } ]
}
```

2. Set `CONVERT_TO_TEXT` on the `contentHandling` property of the `Integration` resource and provide a mapping template to assign the Base64-encoded string of the binary data to a JSON property. In the following example, the JSON property is `body` and `$input.body` holds the Base64-encoded string.

```
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~loctet-stream",
  "value" : "{\"body\": \"$input.body\"}"
}
]
```

Convert Text Data to a Binary Payload

Suppose a Lambda function returns an image file as a Base64-encoded string. To pass this binary output to the client through API Gateway, do the following:

1. Update the API's `binaryMediaTypes` list by adding the binary media type of `application/octet-stream`, if it is not already in the list.

```
PATCH /restapis/<restapi_id>
{
  "patchOperations" : [ {
    "op" : "add",
    "path" : "/binaryMediaTypes/application~loctet-stream",
  } ]
}
```

2. Set the `contentHandling` property on the `Integration` resource to `CONVERT_TO_BINARY`. Do not define a mapping template. When you do not define a mapping template, API Gateway invokes the passthrough template to return the Base64-decoded binary blob as the image file to the client.

```
PATCH /restapis/<restapi_id>/resources/<resource_id>/methods/<http_method>/
integration/responses/<status_code>
{
  "patchOperations" : [
    {
      "op" : "replace",
      "path" : "/contentHandling",
      "value" : "CONTEXT_TO_BINARY"
    }
  ]
}
```

Pass through a Binary Payload

To store an image in an Amazon S3 bucket using API Gateway, do the following:

1. Update the API's `binaryMediaTypes` list by adding the binary media type of `application/octet-stream`, if it is not already in the list.

```
PATCH /restapis/<restapi_id>
```

```
{
  "patchOperations" : [ {
    "op" : "add",
    "path" : "/binaryMediaTypes/application~loctet~stream"
  }
]
```

2. On the `contentHandling` property of the `Integration` resource, set `CONVERT_TO_BINARY`. Set `WHEN_NO_MATCH` as the `passthroughBehavior` property value without defining a mapping template. This enables API Gateway to invoke the passthrough template.

```
PATCH /restapis/<restapi_id>/resources/<resource_id>/methods/<http_method>/
integration

{
  "patchOperations" : [
    {
      "op" : "replace",
      "path" : "/contentHandling",
      "value" : "CONTEXT_TO_BINARY"
    },
    {
      "op" : "replace",
      "path" : "/passthroughBehaviors",
      "value" : "WHEN_NO_MATCH"
    }
  ]
}
```

Import and Export Content Encodings

To import the `binaryMediaTypes` list on a `RestApi`, use the following API Gateway extension to the API's Swagger definition file. The extension is also used to export the API settings.

- [x-amazon-apigateway-binary-media-types Property \(p. 151\)](#)

To import and export the `contentHandling` property value on an `Integration` or `IntegrationResponse` resource, use the following API Gateway extensions to the Swagger definitions:

- [x-amazon-apigateway-integration \(p. 152\)](#)
- [x-amazon-apigateway-integration.response \(p. 157\)](#)

Examples of Binary Support

The following example demonstrates how to access a binary file in Amazon S3 through an API Gateway API. The sample API is presented in a Swagger file. The code example uses the API Gateway REST API calls.

Topics

- [Access Binary Files in Amazon S3 through an API Gateway API \(p. 88\)](#)
- [Access Binary Files in Lambda Using an API Gateway API \(p. 91\)](#)

Access Binary Files in Amazon S3 through an API Gateway API

The following examples show the Swagger file used to access images in Amazon S3, how to download an image from Amazon S3, and how to upload an image to Amazon S3.

Topics

- [Swagger File of a Sample API to Access Images in Amazon S3 \(p. 88\)](#)
- [Download an Image from Amazon S3 \(p. 90\)](#)
- [Upload an Image to Amazon S3 \(p. 90\)](#)

Swagger File of a Sample API to Access Images in Amazon S3

The following Swagger file shows a sample API that illustrates downloading an image file from Amazon S3 and uploading an image file to Amazon S3. This API exposes the `GET /s3?key={file-name}` and `PUT /s3?key={file-name}` methods for downloading and uploading a specified image file. The `GET` method returns the image file as a Base64-encoded string as part of a JSON output, following the supplied mapping template, in a 200 OK response. The `PUT` method takes a raw binary blob as input and returns

```
{
  "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",
        "responseTemplates": {
          "application/json": "{\n  \"image\": \"$input.body\"\n}"
        }
      }
    },
    "requestParameters": {
      "integration.request.path.key": "method.request.querystring.key"
    },
    "uri": "arn:aws:apigateway:us-east-1:123456789012:path/{key}",
    "passthroughBehavior": "when_no_match",
    "httpMethod": "GET",
    "type": "aws"
  }
},
"put": {
  "produces": [
    "application/json", "application/octet-stream"
  ],
  "parameters": [
    {
      "name": "key",
      "in": "query",
      "required": false,
      "type": "string"
    }
  ],
  "responses": {
    "200": {
      "description": "200 response",
      "schema": {
        "$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-east-1:123456789012: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"
    }
  }
}
```

Download an Image from Amazon S3

To download an image file (`image.jpg`) as a binary blob from Amazon S3:

```
GET /v1/s3?key=image.jpg HTTP/1.1
Host: abcdefghi.execute-api.us-east-1.amazonaws.com
Content-Type: application/json
Accept: application/octet-stream
```

The successful response looks like this:

```
200 OK HTTP/1.1

[raw bytes]
```

The raw bytes are returned because the `Accept` header is set to a binary media type of `application/octet-stream` and binary support is enabled for the API.

To download an image file (`image.jpg`) as a Base64-encoded string, formatted as a JSON property, from Amazon S3:

```
GET /v1/s3?key=image.jpg HTTP/1.1
Host: abcdefghi.execute-api.us-east-1.amazonaws.com
Content-Type: application/json
Accept: application/json
```

The successful response looks like this:

```
200 OK HTTP/1.1

{
  "image": "W3JhdYBieXRlc10="
}
```

Upload an Image to Amazon S3

To upload an image file (`image.jpg`) as a binary blob to Amazon S3:

```
PUT /v1/s3?key=image.jpg HTTP/1.1
Host: abcdefghi.execute-api.us-east-1.amazonaws.com
Content-Type: application/octet-stream
```

```
Accept: application/json  
  
[raw bytes]
```

The successful response looks like this:

```
200 OK HTTP/1.1
```

To upload an image file (`image.jpg`) as a Base64-encoded string to Amazon S3:

```
PUT /v1/s3?key=image.jpg HTTP/1.1  
Host: abcdefghi.execute-api.us-east-1.amazonaws.com  
Content-Type: application/json  
Accept: application/json  
  
W3JhdYBieXRlc10=
```

Notice that the input payload must be a Base64-encoded string, because the `Content-Type` header value is set to `application/json`. The successful response looks like this:

```
200 OK HTTP/1.1
```

Access Binary Files in Lambda Using an API Gateway API

The following example demonstrates how to access a binary file in AWS Lambda through an API Gateway API. The sample API is presented in a Swagger file. The code example uses the API Gateway REST API calls.

Topics

- [Swagger File of a Sample API to Access Images in Lambda \(p. 91\)](#)
- [Download an Image from Lambda \(p. 93\)](#)
- [Upload an Image to Lambda \(p. 94\)](#)

Swagger File of a Sample API to Access Images in Lambda

The following Swagger file shows an example API that illustrates downloading an image file from Lambda and uploading an image file to Lambda.

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

Download an Image from Lambda

To download an image file (`image.jpg`) as a binary blob from Lambda:

```
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
```

The successful response looks like this:

```
200 OK HTTP/1.1

[raw bytes]
```

To download an image file (`image.jpg`) as a Base64-encoded string, formatted as a JSON property, from Lambda:

```
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
```

The successful response looks like this:

```
200 OK HTTP/1.1

{
  "image": "W3JhdYBieXRlc10="
}
```

Upload an Image to Lambda

To upload an image file (`image.jpg`) as a binary blob to Lambda:

```
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]
```

The successful response looks like this:

```
200 OK
```

To upload an image file (`image.jpg`) as a Base64-encoded string to Lambda:

```
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=
```

The successful response looks like this:

```
200 OK
```

Configure Mock Integration for a Method in API Gateway

Amazon API Gateway supports mock integrations for API methods. This feature enables API developers to generate API responses from API Gateway directly, without the need for an integration back end. As an API developer, you can use this feature to unblock other dependent teams needing to work with an API before the project development is complete. You can also leverage this feature to provision a landing page of your API, which can provide an overview of and navigation to your API. For an example of such a landing page, see the integration request and response of the GET method on the root resource of the example API discussed in [Create an API Gateway API from an Example \(p. 7\)](#).

As an API developer, you decide how API Gateway responds to a mock integration request. For this, you configure the method's integration request and integration response to associate a response with a given status code. The tasks involve setting up a mapping template in the integration request to specify a supported status code in the request payload and setting up mapping templates, one for a supported status code, in the integration response to provide associated response payloads. At run time API Gateway retrieves the status code from the request payload and invokes the matching template to return the associated response payload. The integration request payload's content type must be `application/json` and its format must be of `{"statusCode": ddd, ... }`, where *ddd* stands for an HTTP status code. The integration response payload's content type can be any of those matching the response data, including `application/json`, `application/xml`, `text/html`, `text/plain` and etc.

In this section, you will learn how to use the API Gateway console to enable the mock integration for an API method.

Topics

- [Prerequisites \(p. 95\)](#)
- [Enable Mock Integration on a Method \(p. 95\)](#)
- [Example Request Templates \(p. 96\)](#)
- [Example Response Templates \(p. 97\)](#)

Prerequisites

- You must have the method available in API Gateway. Follow the instructions in [Build an API Gateway API to Expose an HTTP Endpoint \(p. 6\)](#).

Enable Mock Integration on a Method

1. Choose an API resource and create a method. In the **Setup** pane, choose **Mock Integration** , and then choose **Save**.
2. In the **Method Execution** pane, choose **Integration Request**.
3. By default, mock integrations return a 200 HTTP status code. To customize this default behavior, do the following:
 1. Expand **Mapping Templates**.
 2. For **Content-Type**, do one of the following:
 - If the desired content type is already visible (for example, **application/json**), then choose it.
 - If the desired content type is not already visible, then choose **Add mapping** template, type the desired content type (for example, `application/json`), and then choose **Create**.
 3. In the **Template** editor, type the content of the template you want API Gateway to use to determine which HTTP status code to use in the integration response. The template must output a JSON payload containing the `statusCode` property. For more information, see [Example Request Templates \(p. 96\)](#).
 4. Next to **Mapping template**, choose **Save**.
4. For each query string parameter or header parameter you want to add to the method, do the following:
 1. Choose **Method Execution**, and then choose **Method Request**.
 2. Choose the arrow next to **URL Query String Parameters** or **HTTP Request Headers**, and then choose **Add query string** or **Add header**, respectively.
 3. For **Name**, type the name of the query string parameter or header parameter, and then choose **Create a new query string** or **Create**, respectively.

Note

To remove a query string parameter or header parameter, choose **Cancel** or **Remove**. To change the name of a query string parameter or header parameter, you must remove it and create a new one in its place.

5. Choose **Method Execution**, and then choose **Method Response**.
6. Do one of the following:
 - If all of the **HTTP Status** entries you want to use are already visible (for example, **200**), then skip ahead to step 8.
 - If any of the **HTTP Status** entries you want to use are not already visible, then for each missing **HTTP Status** entry, choose **Add Response**, choose the HTTP status code that you want to use, and then choose **Create**.
7. Choose **Method Execution**, and then choose **Integration Response**.
8. Do one of the following:
 - If all of the **Method response status** entries you want to use are already visible (for example, **200**), then skip ahead to step 10.
 - If any of the **Method response status** entries you want to use are not already visible, then for each missing **Method response status** entry, choose **Add integration response**, for **Method response status** choose the **HTTP Status** entry you created earlier, and then choose **Save**.
9. For each **Method response status** entry you want to use, do the following:
 1. Expand the row that corresponds to the **Method response status** entry you want to use.
 2. For **HTTP status regex**, type the matching **HTTP Status** entry (for example, type **400** for a **400 HTTP Status** entry or **500** for a **500 HTTP Status** entry). Or specify a range of matching HTTP status codes (for example, `5\d{2}` matches all 5XX HTTP status codes).
 3. Expand **Mapping Templates**.
 4. For **Content-Type**, do one of the following:
 - If the desired content type is already visible (for example, **application/json**), then choose it.
 - If the desired content type is not already visible, then choose **Add mapping template**, type the desired content type (for example, **application/json**), and then choose **Create**.
 5. In the **Template** editor, type the contents of the template that you want API Gateway to use to respond to the caller. For more information, see [Example Response Templates \(p. 97\)](#).
 6. Next to **Mapping template**, choose **Save**.
10. Do one of the following to test the method:
 - Call the method from the API Gateway console. Follow the instructions in [Test a Method Using the Console \(p. 330\)](#).
 - Call the method from a web browser, a web debugging proxy tool or the cURL command-line tool, or from your own API. Follow the instructions in [Calling a Deployed API \(p. 329\)](#).

Example Request Templates

The following example shows a request template that always uses the 200 HTTP status code.

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

The following example shows a request template that uses the 200 HTTP status code if the request specifies the `petType` parameter of `cat`; 400 if the request specifies `dog`; and uses 500 otherwise. This example is based on the one in the [Map Request Parameters \(p. 22\)](#).

```
{
  #if( $input.params('petType') == "cat" )
    "statusCode": 200
  #elseif( $input.params('petType') == "dog" )
    "statusCode": 400
  #else
    "statusCode": 500
  #end
}
```

Example Response Templates

The following two examples show response templates that respond with the same information every time. These examples are based on the one in the [Map Request Parameters \(p. 22\)](#).

```
## Example 400 response.
{
  "Message": "Error: petType not valid."
}
```

```
## Example 500 response.
{
  "Message": "Error: petType not valid or not specified."
}
```

The following example shows a response template that responds with the same information every time, but includes the value the caller specified for the `petType` parameter. This example is based on the one in the [Map Request Parameters \(p. 22\)](#).

```
## Example 200 response for ?petType=cat (response will contain "type":
"cat").
{
  "id": 1,
  "name": "Kitty",
  "type": "$input.params('petType')"
}
```

Configure Proxy Integration for a Proxy Resource

To set up a proxy resource in an API Gateway API with a proxy integration, you perform the following three tasks:

- Create a proxy resource with a greedy path variable of `{proxy+}`.
- Set the `ANY` method on the proxy resource.
- Integrate the resource and method with a back end using the HTTP or Lambda integration type.

Note

Greedy path variables, `ANY` methods, and proxy integration types are independent features, although they are commonly used together. You can configure a specific HTTP method on a greedy resource or apply non-proxy integration types to a proxy resource.

API Gateway enacts certain restrictions and limitations when handling methods with either Lambda proxy integration or HTTP proxy integration. For details, [Known Issues \(p. 421\)](#).

Topics

- [API Gateway Proxy Resource \(p. 98\)](#)
- [API Gateway Proxy Integration Types \(p. 98\)](#)
- [Set Up a Proxy Resource with the HTTP Proxy Integration \(p. 98\)](#)
- [Set Up a Proxy Resource with the Lambda Proxy Integration \(p. 102\)](#)
- [Input Format of a Lambda Function for Proxy Integration \(p. 103\)](#)
- [Output Format of a Lambda Function for Proxy Integration \(p. 105\)](#)

API Gateway Proxy Resource

API Gateway defines a proxy resource with the following properties:

- A special path parameter denoted as `{proxy+}`. This path parameter represents any of the child resources under its parent resource of an API. In other words, `/parent/{proxy+}` can stand for any resource matching the path pattern of `/parent/*`. The `+` symbol indicates to API Gateway to intercept all requests on the matched resource. This special path parameter is also known as a greedy path variable. The `proxy` variable is the greedy path variable name and can be replaced by another string in the same way you treat a regular path parameter name.
- A special method, named `ANY`, used to define the same integration set up for all supported methods: `DELETE`, `GET`, `HEAD`, `OPTIONS`, `PATCH`, `POST`, and `PUT`.

API Gateway Proxy Integration Types

A proxy resource is most powerful when it is integrated with a back end using one of the following two proxy [integration](#) types:

- The HTTP proxy integration, designated by `HTTP_PROXY` in the API Gateway REST API, is for integrating a method request with a back-end HTTP endpoint. With this integration type, API Gateway simply passes the entire request and response between the front end and the back end, subject to certain [restrictions and limitations \(p. 421\)](#).
- The Lambda proxy integration, designated by `AWS_PROXY` in the API Gateway REST API, is for integrating a method request with a Lambda function in the back end. With this integration type, API Gateway applies a default mapping template to send the entire request to the Lambda function and transforms the output from the Lambda function to HTTP responses.

When applying the HTTP proxy integration to a proxy resource, you can set up your API to expose a portion or an entire endpoint hierarchy of the HTTP back end with a single integration set up. For example, suppose your back-end website is organized into multiple branches of tree nodes off the root node (`/site`) as: `/site/a0/a1/.../aN`, `/site/b0/b1/.../bM`, etc. If you integrate the `ANY` method on a proxy resource of `/api/{proxy+}` with the back-end endpoints with URL paths of `/site/{proxy}`, a single integration request can support any HTTP operations (`GET`, `POST`, etc.) on any of `[a0, a1, ..., aN, b0, b1, ...bM, ...]`. If you apply a proxy integration to a specific HTTP method, e.g., `GET`, instead, the resulting integration request will work with the specified (e.g., `GET`) operations on any of those back-end nodes.

Similarly, you can apply the Lambda proxy integration to a proxy resource of `/api/{proxy+}` to set up a single integration to have a back-end Lambda function react individually to changes in any of the API resources under `/api`.

Set Up a Proxy Resource with the HTTP Proxy Integration

To set up a proxy resource with the HTTP proxy integration type, create an API resource with a greedy path parameter (e.g., `/parent/{proxy+}`) and integrate this resource with an HTTP back-end

endpoint (e.g., `https://petstore-demo-endpoint.execute-api.com/petstore/{proxy}`) on the ANY method. The greedy path parameter must be at the end of the resource path.

As with a non-proxy resource, you can set up a proxy resource with the HTTP proxy integration using the API Gateway console, importing a Swagger definition file, or calling the API Gateway REST API directly. For detailed instructions about using the API Gateway console to configure a proxy resource with the HTTP integration, see [Create and Test an API with HTTP Proxy Integration through a Proxy Resource \(p. 55\)](#).

The following Swagger API definition file shows an example of an API with a proxy resource that is integrated with the [PetStore](#) website.

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

```
}  
}
```

In this example, a cache key is declared on the `method.request.path.proxy` path parameter of the proxy resource. This is the default setting when you create the API using the API Gateway console. The API's base path (`/test`, corresponding to a stage) is mapped to the website's PetStore page (`/petstore`). The single integration request serves to mirror the entire PetStore website using the API's greedy path variable and the catch-all `ANY` method. The following examples illustrate this mirroring.

- **Set ANY as GET and {proxy+} as pets**

Method request initiated from the front end:

```
GET https://4z9giyi2c1.execute-api.us-west-2.amazonaws.com/test/pets  
HTTP/1.1
```

Integration request sent to the back end:

```
GET http://petstore-demo-endpoint.execute-api.com/petstore/pets HTTP/1.1
```

The run-time instances of the `ANY` method and proxy resource are both valid. The call will return a `200 OK` response with the payload containing the first batch of pets, as returned from the back end.

- **Set ANY as GET and {proxy+} as pets?type=dog**

```
GET https://4z9giyi2c1.execute-api.us-west-2.amazonaws.com/test/pets?  
type=dog HTTP/1.1
```

Integration request sent to the back end:

```
GET http://petstore-demo-endpoint.execute-api.com/petstore/pets?type=dog  
HTTP/1.1
```

The run-time instances of the `ANY` method and proxy resource are both valid. The call will return a `200 OK` response with the payload containing the first batch of specified dogs, as returned from the back end.

- **Set ANY as GET and {proxy+} as pets/{petId}**

Method request initiated from the front end:

```
GET https://4z9giyi2c1.execute-api.us-west-2.amazonaws.com/test/pets/1  
HTTP/1.1
```

Integration request sent to the back end:

```
GET http://petstore-demo-endpoint.execute-api.com/petstore/pets/1 HTTP/1.1
```

The run-time instances of the `ANY` method and proxy resource are both valid. The call will return a `200 OK` response with the payload containing the specified pet, as returned from the back end.

- **Set ANY as POST and {proxy+} as pets**

Method request initiated from the front end:


```
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
}
```

Integration request sent to the back end:

```
POST http://petstore-demo-endpoint.execute-api.com/petstore/pets HTTP/1.1
Content-Type: application/json
Content-Length: ...

{
  "type" : "dog",
  "price" : 1001.00
}
```

The run-time instances of the `ANY` method and proxy resource are both valid. The call will return a `200 OK` response with the payload containing the newly created pet, as returned from the back end.

- **Set ANY as GET and {proxy+} as pets/cat**

Method request initiated from the front end:

```
GET https://4z9giyi2c1.execute-api.us-west-2.amazonaws.com/test/pets/cat
```

Integration request sent to the back end:

```
GET http://petstore-demo-endpoint.execute-api.com/petstore/pets/cat
```

The run-time instance of the proxy resource path does not correspond to a back-end endpoint and the resulting request is invalid. As a result, a `400 Bad Request` response is returned with the following error message.

```
{
  "errors": [
    {
      "key": "Pet2.type",
      "message": "Missing required field"
    },
    {
      "key": "Pet2.price",
      "message": "Missing required field"
    }
  ]
}
```

- **Set ANY as GET and {proxy+} as null**

Method request initiated from the front end:

```
GET https://4z9giyi2cl.execute-api.us-west-2.amazonaws.com/test
```

Integration request sent to the back end:

```
GET http://petstore-demo-endpoint.execute-api.com/petstore
```

The targeted resource is the parent of the proxy resource, but the run-time instance of the `ANY` method is not defined in the API on that resource. As a result, this GET request returns a `403 Forbidden` response with the "Missing Authentication Token" error message as returned by API Gateway. If the API exposes the `ANY` or `GET` method on the parent resource, (`/`), the call will return a `404 Not Found` response with the `Cannot GET /petstore` message as returned from the back end.

For any client request, if the targeted endpoint URL is invalid or the HTTP verb is valid but not supported, the back end returns a `404 Not Found` response. For an unsupported HTTP method, a `403 Forbidden` response is returned.

Set Up a Proxy Resource with the Lambda Proxy Integration

To set up a proxy resource with the Lambda proxy integration type, create an API resource with a greedy path parameter (e.g., `/parent/{proxy+}`) and integrate this resource with a Lambda function back end (e.g., `arn:aws:lambda:us-west-2:123456789012:function:SimpleLambda4ProxyResource`) on the `ANY` method. The greedy path parameter must be at the end of the API resource path. As with a non-proxy resource, you can set up the proxy resource using the API Gateway console, importing a Swagger definition file, or calling the API Gateway REST API directly.

For detailed instructions about using the API Gateway console to configure a proxy resource with the Lambda proxy integration, see [Create an API with Lambda Proxy Integration through a Proxy Resource](#) (p. 59).

The following Swagger API definition file shows an example of an API with a proxy resource that is integrated with the [SimpleLambda4ProxyResource](#) (p. 60) Lambda function.

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

With the Lambda proxy integration, at run time, API Gateway maps an incoming request into the input event parameter of the Lambda function. The input includes the request method, path, headers, any query parameters, any payload, associated context, and any defined stage variables. The input format is explained in [Input Format of a Lambda Function for Proxy Integration \(p. 103\)](#). For API Gateway to map the Lambda output to HTTP responses successfully, the Lambda function must output the result in the format explained in [Output Format of a Lambda Function for Proxy Integration \(p. 105\)](#).

With the Lambda proxy integration of a proxy resource through the ANY method, the single back-end Lambda function serves as the event handler for all requests through the proxy resource. For example, to log traffic patterns, you can have a mobile device send its location in terms of state, city, street, and building by submitting a request with `/state/city/street/house` in the URL path for the proxy resource. The back-end Lambda function can then parse the URL path and insert the location tuples into a DynamoDB table.

Input Format of a Lambda Function for Proxy Integration

Let's use an example to illustrate the input format of a Lambda function with the Lambda proxy integration. The following POST request shows an API deployed to `testStage` with a stage variable of `stageVariableName=stageVariableValue`:

```
POST /testStage/hello/world?name=me HTTP/1.1
Host: gy415nuibc.execute-api.us-east-1.amazonaws.com
Content-Type: application/json
headerName: headerValue

{
  "a": 1
}
```

API Gateway maps the entire client request to the input `event` parameter of the back-end Lambda function as follows:

```
{
  "message": "Hello me!",
  "input": {
    "resource": "/{proxy+}",
    "path": "/hello/world",
    "httpMethod": "POST",
    "headers": {
      "Accept": "*/*",
      "Accept-Encoding": "gzip, deflate",
      "cache-control": "no-cache",
      "CloudFront-Forwarded-Proto": "https",
      "CloudFront-Is-Desktop-Viewer": "true",
      "CloudFront-Is-Mobile-Viewer": "false",
      "CloudFront-Is-SmartTV-Viewer": "false",
      "CloudFront-Is-Tablet-Viewer": "false",
      "CloudFront-Viewer-Country": "US",
      "Content-Type": "application/json",
      "headerName": "headerValue",
      "Host": "gy415nuibc.execute-api.us-east-1.amazonaws.com",
      "Postman-Token": "9f583ef0-ed83-4a38-aef3-eb9ce3f7a57f",
      "User-Agent": "PostmanRuntime/2.4.5",
      "Via": "1.1 d98420743a69852491bbdea73f7680bd.cloudfront.net
(CloudFront)",
      "X-Amz-Cf-Id": "pn-
PWIJc6thYnZm5P0NMgOUglL1DYt10gdeJky8tqsg8iS_sgsKD1A==",
      "X-Forwarded-For": "54.240.196.186, 54.182.214.83",
      "X-Forwarded-Port": "443",
      "X-Forwarded-Proto": "https"
    },
    "queryStringParameters": {
      "name": "me"
    },
    "pathParameters": {
      "proxy": "hello/world"
    },
    "stageVariables": {
      "stageVariableName": "stageVariableValue"
    },
    "requestContext": {
      "accountId": "12345678912",
      "resourceId": "roq9wj",
      "stage": "testStage",
      "requestId": "deef4878-7910-11e6-8f14-25afc3e9ae33",
      "identity": {
        "cognitoIdentityPoolId": null,
        "accountId": null,
        "cognitoIdentityId": null,
        "caller": null,
        "apiKey": null,
        "sourceIp": "192.168.196.186",
        "cognitoAuthenticationType": null,
        "cognitoAuthenticationProvider": null,
        "userArn": null,
        "userAgent": "PostmanRuntime/2.4.5",
        "user": null
      },
      "resourcePath": "/{proxy+}",
      "httpMethod": "POST",
```

```
    "apiId": "gy415nuihc"
  },
  "body": "{\r\n\t\"a\": 1\r\n}"
}
```

Note

API Gateway enacts certain restrictions and limitations when handling methods with either Lambda proxy integration or HTTP proxy integration. For details, [Known Issues \(p. 421\)](#).

Output Format of a Lambda Function for Proxy Integration

With the Lambda proxy integration, API Gateway requires the back-end Lambda function to return output according to the following JSON format:

```
{
  "statusCode": httpStatusCode,
  "headers": { "headerName": "headerValue", ... },
  "body": "... "
}
```

A Lambda function in Node.js can supply a JSON object of this format as the input to the `context.succeed({ ... });` function call. If the function output is of a different format, API Gateway will return a 502 Bad Gateway error response.

After Setting Up Methods and Integration

The next step is to deploy the API to make it open for access. For instructions, see [Deploying an API \(p. 230\)](#).

To configure access control to your API, see [Controlling Access in API Gateway \(p. 160\)](#).

Set Up Amazon API Gateway API Request and Response Payload Mappings

In API Gateway, an API's method request can take a payload in a different format from the corresponding integration request payload, as required in the back end. Similarly, the back end may return an integration response payload different from the method response payload, as expected by the front end. API Gateway lets you map the payload from a method request to the corresponding integration request and from an integration response to the corresponding method response. You use mapping templates to specify the mapping and can create model to facilitate the template generation. The section explains how to map the API request and response payload using models and mapping templates.

Topics

- [Models \(p. 106\)](#)
- [Mapping Templates \(p. 109\)](#)
- [Tasks for Models and Mapping Templates \(p. 112\)](#)
- [Create a Model in API Gateway \(p. 112\)](#)
- [View a List of Models in API Gateway \(p. 113\)](#)
- [Delete a Model in API Gateway \(p. 113\)](#)

- [Photos Example \(API Gateway Models and Mapping Templates\)](#) (p. 114)
- [News Article Example \(API Gateway Models and Mapping Templates\)](#) (p. 117)
- [Sales Invoice Example \(API Gateway Models and Mapping Templates\)](#) (p. 121)
- [Employee Record Example \(API Gateway Models and Mapping Templates\)](#) (p. 125)

Models

In API Gateway, a model defines the structure or shape of a payload and is also known as the schema of the payload. API Gateway requires that the [JSON Schema](#) be used to describe the model of a JSON payload.

API Gateway maps a payload according to a mapping template. A model is useful, but not required, to generate a template. However, models are necessary for generating strongly typed SDK of your API. They can also be useful to validate a payload.

The following JSON object describes a sample data describing the fruit or vegetable inventory in the produce department of a likely supermarket:

```
{
  "department": "produce",
  "categories": [
    "fruit",
    "vegetables"
  ],
  "bins": [
    {
      "category": "fruit",
      "type": "apples",
      "price": 1.99,
      "unit": "pound",
      "quantity": 232
    },
    {
      "category": "fruit",
      "type": "bananas",
      "price": 0.19,
      "unit": "each",
      "quantity": 112
    },
    {
      "category": "vegetables",
      "type": "carrots",
      "price": 1.29,
      "unit": "bag",
      "quantity": 57
    }
  ]
}
```

The JSON object has three properties

- The `department` property has a string value (`produce`).
- The `categories` property is an array of two strings: `fruit` and `vegetables`.
- The `bins` property is an array of objects, each having the string- or number-valued properties of `category`, `type`, `price`, `unit` and `quantity`.

We express the corresponding model in the following JSON Schema notation:

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "title": "GroceryStoreInputModel",
  "type": "object",
  "properties": {
    "department": { "type": "string" },
    "categories": {
      "type": "array",
      "items": { "type": "string" }
    },
    "bins": {
      "type": "array",
      "items": {
        "type": "object",
        "properties": {
          "category": { "type": "string" },
          "type": { "type": "string" },
          "price": { "type": "number" },
          "unit": { "type": "string" },
          "quantity": { "type": "integer" }
        }
      }
    }
  }
}
```

In the preceding example:

- The `$schema` object represents a valid JSON Schema version identifier. In this example, it refers to JSON Schema, draft v4.
- The `title` object is a human-readable identifier for the model. In this example, it is `GroceryStoreInputModel`.
- The top-level, or root, construct in the JSON data is an object.
- The root object in the JSON data contains `department`, `categories`, and `bins` properties.
- The `department` property is a string object in the JSON data.
- The `categories` property is an array in the JSON data. The array contains string values in the JSON data.
- The `bins` property is an array in the JSON data. The array contains objects in the JSON data. Each of these objects in the JSON data contains a `category` string, a `type` string, a `price` number, a `unit` string, and a `quantity` integer (a number without a fraction or exponent part).

Alternatively, you could include part of this schema, for example, the item definition of the `bins` array, in a separate section of the same file and use the `$ref` primitive to reference this reusable definition in other parts of the schema. Using `$ref`, the above model definition file can be expressed as follows:

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "title": "GroceryStoreInputModel",
  "type": "object",
  "properties": {
    "department": { "type": "string" },
    "categories": {
      "type": "array",
```

```

    "items": { "type": "string" }
  },
  "bins": {
    "type": "array",
    "items": {
      "$ref": "#/definitions/Bin"
    }
  }
},
"definitions": {
  "Bin" : {
    "type": "object",
    "properties": {
      "category": { "type": "string" },
      "type": { "type": "string" },
      "price": { "type": "number" },
      "unit": { "type": "string" },
      "quantity": { "type": "integer" }
    }
  }
}
}

```

The `definitions` section contains the schema definition of the `Bin` item that is referenced in the `bins` array with `"ref": "#/definitions/Bin"`. Using reusable definitions this way makes your model definition easier to read.

In addition, you can also reference another model schema defined in an external model file by setting that model's URL as the value of the `$ref` property: `"$ref": "https://apigateway.amazonaws.com/restapis/{restapi_id}/models/{model_name}"`. For example, supposed you have the following full-fledged model named `Bin2` created under an API with an identifier of `fugvjdxtri`:

```

{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "title": "GroceryStoreInputModel",
  "type": "object",
  "properties": {
    "Bin" : {
      "type": "object",
      "properties": {
        "category": { "type": "string" },
        "type": { "type": "string" },
        "price": { "type": "number" },
        "unit": { "type": "string" },
        "quantity": { "type": "integer" }
      }
    }
  }
}

```

You can then reference it from the `GroceryStoreInputModel` from the same API, as shown as follows:

```

{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "title": "GroceryStoreInputModel",
  "type": "object",

```



```
"properties": {
  "department": { "type": "string" },
  "categories": {
    "type": "array",
    "items": { "type": "string" }
  },
  "bins": {
    "type": "array",
    "items": {
      "$ref": "https://apigateway.amazonaws.com/restapis/fugvjdxtri/models/
Bin2"
    }
  }
}
```

The referencing and referenced models must be from the same API.

The examples do not use advanced JSON Schema features, such as specifying required items; minimum and maximum allowed string lengths, numeric values, and array item lengths; regular expressions; and more. For more information, see [Introducing JSON](#) and [JSON Schema](#).

For more complex JSON data formats and their models, see the following examples:

- [Input Model \(Photos Example\) \(p. 115\)](#) and [Output Model \(Photos Example\) \(p. 116\)](#) in the [Photos Example \(p. 114\)](#)
- [Input Model \(News Article Example\) \(p. 118\)](#) and [Output Model \(News Article Example\) \(p. 120\)](#) in the [News Article Example \(p. 117\)](#)
- [Input Model \(Sales Invoice Example\) \(p. 122\)](#) and [Output Model \(Sales Invoice Example\) \(p. 124\)](#) in the [Sales Invoice Example \(p. 121\)](#)
- [Input Model \(Employee Record Example\) \(p. 126\)](#) and [Output Model \(Employee Record Example\) \(p. 129\)](#) in the [Employee Record Example \(p. 125\)](#)

To experiment with models in API Gateway, follow the instructions in [Map Response Payload \(p. 31\)](#), specifically [Step 1: Create Models \(p. 33\)](#).

Mapping Templates

In API Gateway, a mapping template is used to transform some data from one format to another. You create and use input mapping templates and output mapping templates when you need to inform API Gateway about the schema of the data being sent from or returned to the caller, respectively. API Gateway uses the [Velocity Template Language \(VTL\)](#) and [JSONPath expressions](#) to define mapping templates.

For an example of an input mapping template, consider the example JSON data from the previous section. The following input mapping template makes no transform to the JSON data as API Gateway receives the JSON data from the caller:

```
#set($inputRoot = $input.path('$'))
{
  "department": "$inputRoot.department",
  "categories": [
#foreach($elem in $inputRoot.categories)
    "$elem"#if($foreach.hasNext),#end
#end
  ],
```

```
"bins" : [  
#foreach($elem in $inputRoot.bins)  
  {  
    "category" : "$elem.category",  
    "type" : "$elem.type",  
    "price" : $elem.price,  
    "unit" : "$elem.unit",  
    "quantity" : $elem.quantity  
  }#if($foreach.hasNext),#end  
#end  
]
```

The preceding input mapping template is expressed as follows:

- Let the variable `$inputRoot` in the input mapping template represent the root object in the original JSON data.
- The values of the `department` object and `categories` and `bins` arrays in the input mapping template (represented by `$inputRoot.department`, `$inputRoot.categories`, and `$inputRoot.bins`) map to the corresponding values of the `department` object and `categories` and `bins` arrays in the root object in the original JSON data.
- In the input mapping template, each of the values in the `categories` array (represented by the first `$elem`), and each of the objects in the `bins` array (represented by the second `$elem`), map to the corresponding values in the `categories` array and objects in the `bins` array, respectively, within the root object in the original JSON data.
- For each of objects in the `bins` object, the values of the `category`, `type`, `price`, `unit`, and `quantity` objects in the input mapping template (represented by `$elem.category`, `$elem.type`, `$elem.price`, `$elem.unit`, and `$elem.quantity`, respectively) map to the corresponding values of the `category`, `type`, `price`, `unit`, and `quantity` objects in the original JSON data, respectively.

For an example of an output mapping template, first consider the following JSON data schema, which is based on the example JSON data from the previous section.

Note

None of the array and object names in this JSON data schema match the JSON data from the previous section:

```
{  
  "choices": [  
    {  
      "kind": "apples",  
      "suggestedPrice": "1.99 per pound",  
      "available": 232  
    },  
    {  
      "kind": "bananas",  
      "suggestedPrice": "0.19 per each",  
      "available": 112  
    },  
    {  
      "kind": "carrots",  
      "suggestedPrice": "1.29 per bag",  
      "available": 57  
    }  
  ]  
}
```

```
}
```

To transform the example JSON data from the previous section into this JSON data schema, you would use the following model:

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "title": "GroceryStoreOutputModel",
  "type": "object",
  "properties": {
    "choices": {
      "type": "array",
      "items": {
        "type": "object",
        "properties": {
          "kind": { "type": "string" },
          "suggestedPrice": { "type": "string" },
          "available": { "type": "integer" }
        }
      }
    }
  }
}
```

In the preceding example, the JSON schema is expressed as follows:

- The `$schema` object represents a valid JSON Schema version identifier. In this example, it refers to JSON Schema, draft v4.
- The `title` object is a human-readable identifier for the model. In this example, it is `GroceryStoreOutputModel`.
- The top-level, or root, construct in the JSON data is an object.
- The root object in the JSON data contains an array of objects.
- Each object in the array of objects contains a `kind` string, a `suggestedPrice` string, and an `available` integer (a number without a fraction or exponent part).

You would then use the following output mapping template, which is based on this model:

```
#set($inputRoot = $input.path('$'))
{
  "choices": [
#foreach($elem in $inputRoot.bins)
    {
      "kind": "$elem.type",
      "suggestedPrice": "$elem.price per $elem.unit",
      "available": $elem.quantity
    }#if($foreach.hasNext),#end
#end
  ]
}
```

The preceding output mapping template is expressed as follows:

- Let the variable `$inputRoot` in the output mapping template represent the root object in the original JSON data from the previous section. Note the variables in the output mapping template map to the original JSON data, not the desired transformed JSON data schema.

- The `choices` array in the output mapping template maps to the `bins` array with the root object in the original JSON data (`$inputRoot.bins`).
- In the output mapping template, each of the objects in the `choices` array (represented by `$elem`) map to the corresponding objects in the `bins` array within the root object in the original JSON data.
- In the output mapping template, for each of objects in the `choices` object, the values of the `kind` and `available` objects (represented by `$elem.type` and `$elem.quantity`) map to the corresponding values of the `type` and `value` objects in each of the objects in the original JSON data's `bins` array, respectively.
- In the output mapping template, for each of objects in the `choices` object, the value of the `suggestedPrice` object is a concatenation of the corresponding value of the `price` and `unit` objects in each of the objects in the original JSON data, respectively, with each value separated by the word `per`.

For more information about the Velocity Template Language, see [Apache Velocity - VTL Reference](#).
For more information about JSONPath, see [JSONPath - XPath for JSON](#).

To explore more complex mapping templates, see the following examples:

- [Input Mapping Template \(Photos Example\) \(p. 115\)](#) and [Output Mapping Template \(Photos Example\) \(p. 117\)](#) in the [Photos Example \(p. 114\)](#)
- [Input Mapping Template \(News Article Example\) \(p. 119\)](#) and [Output Mapping Template \(News Article Example\) \(p. 120\)](#) in the [News Article Example \(p. 117\)](#)
- [Input Mapping Template \(Sales Invoice Example\) \(p. 123\)](#) and [Output Mapping Template \(Sales Invoice Example\) \(p. 125\)](#) in the [Sales Invoice Example \(p. 121\)](#)
- [Input Mapping Template \(Employee Record Example\) \(p. 127\)](#) and [Output Mapping Template \(Employee Record Example\) \(p. 130\)](#) in the [Employee Record Example \(p. 125\)](#)

To experiment with mapping templates in API Gateway, follow the instructions in [Map Response Payload \(p. 31\)](#), specifically [Step 5: Set Up and Test the Methods \(p. 37\)](#).

Tasks for Models and Mapping Templates

For additional things you can do with models and mapping templates, see the following:

- [Create a Model \(p. 112\)](#)
- [View a List of Models \(p. 113\)](#)
- [Delete a Model \(p. 113\)](#)

Create a Model in API Gateway

Use the API Gateway console to create a model for an API.

Topics

- [Prerequisites \(p. 112\)](#)
- [Create a Model With the API Gateway Console \(p. 113\)](#)

Prerequisites

- You must have an API available in API Gateway. Follow the instructions in [Creating an API \(p. 70\)](#).

Create a Model With the API Gateway Console

1. Sign in to the API Gateway console at <https://console.aws.amazon.com/apigateway>.
2. In the box that contains the name of the API where you want to create the model, choose **Models**.
3. Choose **Create**.
4. For **Model Name**, type a name for the model.
5. For **Content Type**, type the model's content type (for example, `application/json` for JSON).
6. (Optional) For **Model description**, type a description for the model.
7. For **Model schema**, type the model's schema. For more information about model schemas, see [Set Up Payload Mappings \(p. 105\)](#).
8. Choose **Create model**.

View a List of Models in API Gateway

Use the API Gateway console to view a list of models.

Topics

- [Prerequisites \(p. 113\)](#)
- [View a List of Models with the API Gateway Console \(p. 113\)](#)

Prerequisites

- You must have at least one model in API Gateway. Follow the instructions in [Create a Model \(p. 112\)](#).

View a List of Models with the API Gateway Console

1. Sign in to the API Gateway console at <https://console.aws.amazon.com/apigateway>.
2. In the box that contains the name of the API, choose **Models**.

Delete a Model in API Gateway

Use the API Gateway console to delete a model.

Warning

Deleting a model may cause part or all of the corresponding API to become unusable by API callers. Deleting a model cannot be undone.

Delete a Model with the API Gateway Console

1. Sign in to the API Gateway console at <https://console.aws.amazon.com/apigateway>.
2. In the box that contains the name of the API for the model, choose **Models**.
3. In the **Models** pane, choose the model you want to delete, and then choose **Delete Model**.
4. When prompted, choose **Delete**.

Photos Example (API Gateway Models and Mapping Templates)

The following sections provide examples of models and mapping templates that could be used for a sample photo API in API Gateway. For more information about models and mapping templates in API Gateway, see [Set Up Payload Mappings](#) (p. 105).

Topics

- [Original Data \(Photos Example\)](#) (p. 114)
- [Input Model \(Photos Example\)](#) (p. 115)
- [Input Mapping Template \(Photos Example\)](#) (p. 115)
- [Transformed Data \(Photos Example\)](#) (p. 116)
- [Output Model \(Photos Example\)](#) (p. 116)
- [Output Mapping Template \(Photos Example\)](#) (p. 117)

Original Data (Photos Example)

The following is the original JSON data for the photos example:

```
{
  "photos": {
    "page": 1,
    "pages": "1234",
    "perpage": 100,
    "total": "123398",
    "photo": [
      {
        "id": "12345678901",
        "owner": "23456789@A12",
        "secret": "abc123d456",
        "server": "1234",
        "farm": 1,
        "title": "Sample photo 1",
        "ispublic": 1,
        "isfriend": 0,
        "isfamily": 0
      },
      {
        "id": "23456789012",
        "owner": "34567890@B23",
        "secret": "bcd234e567",
        "server": "2345",
        "farm": 2,
        "title": "Sample photo 2",
        "ispublic": 1,
        "isfriend": 0,
        "isfamily": 0
      }
    ]
  }
}
```

Input Model (Photos Example)

The following is the input model that corresponds to the original JSON data for the photos example:

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "title": "PhotosInputModel",
  "type": "object",
  "properties": {
    "photos": {
      "type": "object",
      "properties": {
        "page": { "type": "integer" },
        "pages": { "type": "string" },
        "perpage": { "type": "integer" },
        "total": { "type": "string" },
        "photo": {
          "type": "array",
          "items": {
            "type": "object",
            "properties": {
              "id": { "type": "string" },
              "owner": { "type": "string" },
              "secret": { "type": "string" },
              "server": { "type": "string" },
              "farm": { "type": "integer" },
              "title": { "type": "string" },
              "ispublic": { "type": "integer" },
              "isfriend": { "type": "integer" },
              "isfamily": { "type": "integer" }
            }
          }
        }
      }
    }
  }
}
```

Input Mapping Template (Photos Example)

The following is the input mapping template that corresponds to the original JSON data for the photos example:

```
#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": "$elem.owner",
        "secret": "$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
    ]
}
}
```

Transformed Data (Photos Example)

The following is one example of how the original photos example JSON data could be transformed for output:

```
{
  "photos": [
    {
      "id": "12345678901",
      "owner": "23456789@A12",
      "title": "Sample photo 1",
      "ispublic": 1,
      "isfriend": 0,
      "isfamily": 0
    },
    {
      "id": "23456789012",
      "owner": "34567890@B23",
      "title": "Sample photo 2",
      "ispublic": 1,
      "isfriend": 0,
      "isfamily": 0
    }
  ]
}
```

Output Model (Photos Example)

The following is the output model that corresponds to the transformed JSON data format:

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "title": "PhotosOutputModel",
  "type": "object",
  "properties": {
    "photos": {
      "type": "array",
      "items": {
        "type": "object",
        "properties": {
          "id": { "type": "string" },
          "owner": { "type": "string" },
          "title": { "type": "string" },
          "ispublic": { "type": "integer" },

```



```
        "isfriend": { "type": "integer" },
        "isfamily": { "type": "integer" }
    }
}
}
```

Output Mapping Template (Photos Example)

The following is the output mapping template that corresponds to the transformed JSON data format. The template variables here are based on the original, not transformed, JSON data format:

```
#set($inputRoot = $input.path('$'))
{
  "photos": [
#foreach($elem in $inputRoot.photos.photo)
    {
      "id": "$elem.id",
      "owner": "$elem.owner",
      "title": "$elem.title",
      "ispublic": $elem.ispublic,
      "isfriend": $elem.isfriend,
      "isfamily": $elem.isfamily
    }#if($foreach.hasNext),#end
#end
  ]
}
```

News Article Example (API Gateway Models and Mapping Templates)

The following sections provide examples of models and mapping templates that could be used for a sample news article API in API Gateway. For more information about models and mapping templates in API Gateway, see [Set Up Payload Mappings](#) (p. 105).

Topics

- [Original Data \(News Article Example\)](#) (p. 117)
- [Input Model \(News Article Example\)](#) (p. 118)
- [Input Mapping Template \(News Article Example\)](#) (p. 119)
- [Transformed Data \(News Article Example\)](#) (p. 119)
- [Output Model \(News Article Example\)](#) (p. 120)
- [Output Mapping Template \(News Article Example\)](#) (p. 120)

Original Data (News Article Example)

The following is the original JSON data for the news article example:

```
{
  "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
}
```

Input Model (News Article Example)

The following is the input model that corresponds to the original JSON data for the news article example:

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

Input Mapping Template (News Article Example)

The following is the input mapping template that corresponds to the original JSON data for the news article example:

```
#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
}
```

Transformed Data (News Article Example)

The following is one example of how the original news article example JSON data could be transformed for output:

```
{
  "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
}
```

Output Model (News Article Example)

The following is the output model that corresponds to the transformed JSON data format:

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

Output Mapping Template (News Article Example)

The following is the output mapping template that corresponds to the transformed JSON data format. The template variables here are based on the original, not transformed, JSON data format:

```
#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"
    }
#endforeach
  ]
}
```

```
    }#if($foreach.hasNext),#end  
#end  
  ],  
  "version": $inputRoot.version  
}
```

Sales Invoice Example (API Gateway Models and Mapping Templates)

The following sections provide examples of models and mapping templates that could be used for a sample sales invoice API in API Gateway. For more information about models and mapping templates in API Gateway, see [Set Up Payload Mappings](#) (p. 105).

Topics

- [Original Data \(Sales Invoice Example\)](#) (p. 121)
- [Input Model \(Sales Invoice Example\)](#) (p. 122)
- [Input Mapping Template \(Sales Invoice Example\)](#) (p. 123)
- [Transformed Data \(Sales Invoice Example\)](#) (p. 124)
- [Output Model \(Sales Invoice Example\)](#) (p. 124)
- [Output Mapping Template \(Sales Invoice Example\)](#) (p. 125)

Original Data (Sales Invoice Example)

The following is the original JSON data for the sales invoice example:

```
{  
  "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
}
```

Input Model (Sales Invoice Example)

The following is the input model that corresponds to the original JSON data for the sales invoice example:

```
{
  "$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" }
}
}

```

Input Mapping Template (Sales Invoice Example)

The following is the input mapping template that corresponds to the original JSON data for the sales invoice example:

```

#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
}
```

Transformed Data (Sales Invoice Example)

The following is one example of how the original sales invoice example JSON data could be transformed for output:

```
{
  "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
}
```

Output Model (Sales Invoice Example)

The following is the output model that corresponds to the transformed JSON data format:

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

Output Mapping Template (Sales Invoice Example)

The following is the output mapping template that corresponds to the transformed JSON data format. The template variables here are based on the original, not transformed, JSON data format:

```
#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
}
```

Employee Record Example (API Gateway Models and Mapping Templates)

The following sections provide examples of models and mapping templates that can be used for a sample employee record API in API Gateway. For more information about models and mapping templates in API Gateway, see [Set Up Payload Mappings \(p. 105\)](#).

Topics

- [Original Data \(Employee Record Example\) \(p. 126\)](#)
- [Input Model \(Employee Record Example\) \(p. 126\)](#)
- [Input Mapping Template \(Employee Record Example\) \(p. 127\)](#)
- [Transformed Data \(Employee Record Example\) \(p. 128\)](#)

- [Output Model \(Employee Record Example\) \(p. 129\)](#)
- [Output Mapping Template \(Employee Record Example\) \(p. 130\)](#)

Original Data (Employee Record Example)

The following is the original JSON data for the employee record example:

```
{
  "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"
}
```

Input Model (Employee Record Example)

The following is the input model that corresponds to the original JSON data for the employee record example:

```
{
  "$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" }
}

```

Input Mapping Template (Employee Record Example)

The following is the input mapping template that corresponds to the original JSON data for the employee record example:

```
#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"
}
```

Transformed Data (Employee Record Example)

The following is one example of how the original employee record example JSON data could be transformed for output:

```
{
  "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"
}
```

Output Model (Employee Record Example)

The following is the output model that corresponds to the transformed JSON data format:

```
{
  "$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" }
}

```

Output Mapping Template (Employee Record Example)

The following is the output mapping template that corresponds to the transformed JSON data format. The template variables here are based on the original, not transformed, JSON data format:

```

#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 Request and Response Parameter-Mapping Reference

This section explains how to set up data mappings from an API's method request data, including other data stored in [context](#) (p. 134), [stage](#) (p. 139) or [util](#) (p. 139) variables, to the corresponding integration request parameters and from an integration response data, including the other data, to the method response parameters. The method request data includes request parameters (path, query string, headers) and the body. The integration response data includes response parameters (headers), and the body. For more information about using the stage variables, see [Amazon API Gateway Stage Variables Reference](#) (p. 253).

Topics

- [Map Data to Integration Request Parameters](#) (p. 131)
- [Map Data to Method Response Headers](#) (p. 132)
- [Transform Request and Response Bodies](#) (p. 133)

Map Data to Integration Request Parameters

Integration request parameters, in the form of path variables, query strings or headers, can be mapped from any defined method request parameters and the payload.

Integration request data mapping expressions

Mapped data source	Mapping expression
Method request path	<code>method.request.path.<i>PARAM_NAME</i></code>
Method request query string	<code>method.request.querystring.<i>PARAM_NAME</i></code>
Method request header	<code>method.request.header.<i>PARAM_NAME</i></code>
Method request body	<code>method.request.body</code>
Method request body (JsonPath)	<code>method.request.body.<i>JSONPath_EXPRESSION</i></code> .
Stage variables	<code>stageVariables.<i>VARIABLE_NAME</i></code>
Context variables	<code>context.<i>VARIABLE_NAME</i></code> that must be one of the supported context variables (p. 134).
Static value	<code>'<i>STATIC_VALUE</i>'</code> . The <i>STATIC_VALUE</i> is a string literal and must be enclosed within a pair of single quotes.

Here, *PARAM_NAME* is the name of a method request parameter of the given parameter type. It must have been defined before it can be referenced. *JSONPath_EXPRESSION* is a JSONPath expression for a JSON field of the body of a request or response. However, the "\$." prefix is omitted in this syntax.

Example mappings from method request parameter in Swagger

The following example shows a Swagger snippet that maps 1) the method request's header, named `methodRequestHeaderParam`, into the integration request path parameter, named `integrationPathParam`; 2) the method request query string, named `methodRequestQueryParam`, into the integration request query string, named `integrationQueryParam`.

```
...
"requestParameters" : {
    "integration.request.path.integrationPathParam" :
    "method.request.header.methodRequestHeaderParam",
    "integration.request.querystring.integrationQueryParam" :
    "method.request.querystring.methodRequestQueryParam"
}
...
```

Integration request parameters can also be mapped from fields in the JSON request body using a [JSONPath expression](#). The following table shows the mapping expressions for a method request body and its JSON fields.

Example mapping from method request body in Swagger

The following example shows a Swagger snippet that maps 1) the method request body to the integration request header, named `body-header`, and 2) a JSON field of the body, as expressed by a JSON expression (`petstore.pets[0].name`, without the `$.` prefix).

```
...
"requestParameters" : {
    "integration.request.header.body-header" : "method.request.body",
    "integration.request.path.pet-name" :
    "method.request.body.petstore.pets[0].name",
}
...
```

Map Data to Method Response Headers

Method response header parameters can be mapped from any integration response header or from the integration response body.

Method response header mapping expressions

Mapped Data Source	Mapping expression
Integration response header	<code>integration.response.header.PARAM_NAME</code>
Integration response body	<code>integration.response.body</code>

Mapped Data Source	Mapping expression
Integration response body (JsonPath)	<code>integration.response.body.<i>JSONPath_EXPRESSION</i></code>
Stage variable	<code>stageVariables.<i>VARIABLE_NAME</i></code>
Context variable	<code>context.<i>VARIABLE_NAME</i></code> that must be one of the supported context variables (p. 134).
Static value	<code>'<i>STATIC_VALUE</i>'</code> . The <i>STATIC_VALUE</i> is a string literal and must be enclosed within a pair of single quotes.

Example data mapping from integration response in Swagger

The following example shows a Swagger snippet that maps 1) the integration response's `redirect.url`, JSONPath field into the request response's `location` header; and 2) the integration response's `x-app-id` header to the method response's `id` header.

```
...
"responseParameters" : {
    "method.response.header.location" :
    "integration.response.body.redirect.url",
    "method.response.header.id" : "integration.response.header.x-app-id",
}
...
```

Transform Request and Response Bodies

Integration request and method response bodies can be transformed from the method request and integration response bodies, respectively, by using [Mapping Templates](#) (p. 109) written in [Velocity Template Language \(VTL\)](#). JSON data can be manipulated using VTL logic and JSONPath expressions, and additional data can be included from HTTP parameters, the calling context, and stage variables.

Select Mapping Templates

The request mapping template used to transform the method request body into the integration request body is selected by the value of the "Content-Type" header sent in the client request.

The response mapping template used to transform the integration response body into the method response body is selected by the value of the "Accept" header sent in the client request.

For example, if the client sends headers of "Content-Type : application/xml", and "Accept : application/json", the request template with the `application/xml` key will be used for the integration request, and the response template with the `application/json` key will be used for the method response.

Only the MIME type is used from the `Accept` and `Content-Type` headers when selecting a mapping template. For example, a header of "Content-Type: application/json; charset=UTF-8" will have a request template with the `application/json` key selected.

API Gateway API Request and Response Payload-Mapping Template Reference

Amazon API Gateway defines a set of variables for working with models and mapping templates. This document describes those functions and provides examples for working with input payloads.

Topics

- [Accessing the \\$context Variable \(p. 134\)](#)
- [Accessing the \\$input Variable \(p. 136\)](#)
- [Accessing the \\$stageVariables Variable \(p. 139\)](#)
- [Accessing the \\$util Variable \(p. 139\)](#)
- [Integration Passthrough Behaviors \(p. 140\)](#)

Accessing the \$context Variable

The `$context` variable holds all the contextual information of your API call.

\$context Variable Reference

Parameter	Description
<code>\$context.apiId</code>	The identifier API Gateway assigns to your API.
<code>\$context.authorizer.claims.<i>property</i></code>	A property of the claims returned from the Amazon Cognito user pool after the method caller is successfully authenticated. Note Calling <code>\$context.authorizer.claims</code> returns null.
<code>\$context.authorizer.principalId</code>	The principal user identification associated with the token sent by the client and returned from an API Gateway custom authorizer Lambda function.
<code>\$context.authorizer.<i>property</i></code>	The value of the specified key-value pair of the <code>context</code> map returned from an API Gateway custom authorizer Lambda function. For example, if the authorizer returns the following <code>context</code> map: <pre>"context" : { "key": "value", "numKey": 1, "boolKey": true }</pre> calling <code>\$context.authorizer.key</code> returns <code>value</code> , calling <code>\$context.authorizer.numKey</code> returns <code>1</code> , and calling <code>\$context.authorizer.boolKey</code> returns <code>true</code> .

Parameter	Description
<code>\$context.httpMethod</code>	The HTTP method used. Valid values include: DELETE, GET, HEAD, OPTIONS, PATCH, POST, and PUT.
<code>\$context.identity.accountId</code>	The AWS account ID associated with the request.
<code>\$context.identity.apiKey</code>	The API owner key associated with your API.
<code>\$context.identity.caller</code>	The principal identifier of the caller making the request.
<code>\$context.identity.cognitoAuthenticationProvider</code>	The Amazon Cognito authentication provider used by the caller making the request. Available only if the request was signed with Amazon Cognito credentials. For information related to this and the other Amazon Cognito \$context variables, see Amazon Cognito Identity .
<code>\$context.identity.cognitoAuthenticationType</code>	The Amazon Cognito authentication type of the caller making the request. Available only if the request was signed with Amazon Cognito credentials.
<code>\$context.identity.cognitoIdentityId</code>	The Amazon Cognito identity ID of the caller making the request. Available only if the request was signed with Amazon Cognito credentials.
<code>\$context.identity.cognitoIdentityPoolId</code>	The Amazon Cognito identity pool ID of the caller making the request. Available only if the request was signed with Amazon Cognito credentials.
<code>\$context.identity.sourceIp</code>	The source IP address of the TCP connection making the request to API Gateway.
<code>\$context.identity.user</code>	The principal identifier of the user making the request.
<code>\$context.identity.userAgent</code>	The User Agent of the API caller.
<code>\$context.identity.userArn</code>	The Amazon Resource Name (ARN) of the effective user identified after authentication.
<code>\$context.requestId</code>	An automatically generated ID for the API call.
<code>\$context.resourceId</code>	The identifier API Gateway assigns to your resource.
<code>\$context.resourcePath</code>	The path to your resource. For more information, see Build an API Gateway API to Expose an HTTP Endpoint (p. 6) .
<code>\$context.stage</code>	The deployment stage of the API call (for example, Beta or Prod).

Example

You may want to use the `$context` variable if you're using AWS Lambda as the target backend that the API method calls. For example, you may want to perform two different actions depending on whether the stage is in Beta or in Prod.

Context Variables Template Example

The following example shows how to get context variables:

```
{
  "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"
}
```

Accessing the \$input Variable

The `$input` variable represents the input payload and parameters to be processed by your template. It provides four functions:

Function Reference

Variable and Function	Description
<code>\$input.body</code>	Returns the raw payload as a string.
<code>\$input.json(x)</code>	This function evaluates a JSONPath expression and returns the results as a JSON string. For example, <code>\$input.json('\$.pets')</code> will return a JSON string representing the pets structure. For more information about JSONPath, see JSONPath or JSONPath for Java .
<code>\$input.params()</code>	Returns a map of all the request parameters of your API call.
<code>\$input.params(x)</code>	Returns the value of a method request parameter from the path, query string, or header value (in that order) given a parameter name string <code>x</code> .
<code>\$input.path(x)</code>	Takes a JSONPath expression string (<code>x</code>) and returns an object representation of the result. This allows you to access and manipulate

Variable and Function	Description
	elements of the payload natively in Apache Velocity Template Language (VTL) . For example, <code>\$input.path('\$.pets').size()</code> For more information about JSONPath, see JSONPath or JSONPath for Java .

Examples

You may want to use the `$input` variable to get query strings and the request body with or without using models. You may also want to get the parameter and the payload, or a subsection of the payload, into your AWS Lambda function. The examples below show how to do this.

Example JSON Mapping Template

The following example shows how to use a mapping to read a name from the query string and then include the entire POST body in an element:

```
{
  "name" : "$input.params('name')",
  "body" : $input.json('$')
}
```

If the JSON input contains unescaped characters that cannot be parsed by JavaScript, a 400 response may be returned. Applying `$util.escapeJavaScript($input.json('$'))` above will ensure that the JSON input can be parsed properly.

Example Inputs Mapping Template

The following example shows how to pass a JSONPath expression to the `json()` method. You could also read a specific property of your request body object by using a period (`.`), followed by your property name:

```
{
  "name" : "$input.params('name')",
  "body" : $input.json('$.mykey')
}
```

If a method request payload contains unescaped characters that cannot be parsed by JavaScript, you may get 400 response. In this case, you need to call `$util.escapeJavaScript()` function in the mapping template, as shown as follows:

```
{
  "name" : "$input.params('name')",
  "body" : $util.escapeJavaScript($input.json('$.mykey'))
}
```

Param Mapping Template Example

The following parameter-mapping example passes all parameters, including path, querystring and header, through to the integration endpoint via a JSON payload

```
#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
}
```

In effect, this mapping template outputs all the request parameters in the payload as outlined as follows:

```
{
  "parameters" : {
    "path" : {
      "path_name" : "path_value",
      ...
    }
    "header" : {
      "header_name" : "header_value",
      ...
    }
    "querystring" : {
      "querystring_name" : "querystring_value",
      ...
    }
  }
}
```

Example Request and Response

Here's an example that uses all three functions:

Request Template:

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

Response:

```

{
  "id": "abc",
  "count": "3",
  "things": {
    "1": {},
    "2": {},
    "3": {}
  }
}
    
```

For more mapping examples, see [Set Up Payload Mappings \(p. 105\)](#)

Accessing the \$stageVariables Variable

The syntax for inserting a stage variable looks like this: `$stageVariables`.

\$stageVariables Reference

Syntax	Description
<code>\$stageVariables.<variable_name></code>	<code><variable_name></code> represents a stage variable name.
<code>\$stageVariables['<variable_name>']</code>	<code><variable_name></code> represents any stage variable name.
<code>#{stageVariables['<variable_name>']}</code>	<code><variable_name></code> represents any stage variable name.

Accessing the \$util Variable

The `$util` variable contains utility functions for use in mapping templates.

Note

Unless otherwise specified, the default character set is UTF-8.

\$util Variable Reference

Function	Description
<code>\$util.escapeJavaScript()</code>	Escapes the characters in a string using JavaScript string rules. <p>Note This function will turn any regular single quotes (') into escaped ones (\'). However, the escaped single quotes are not valid in JSON. Thus, when the output from this function is used in</p>

Function	Description
	<p>a JSON property, you must turn any escaped single quotes (<code>\'</code>) back to regular single quotes (<code>'</code>). This is shown in the following example:</p> <pre>\$util.escapeJavaScript(<i>data</i>).replaceAll("\'","'")</pre>
<code>\$util.parseJson()</code>	<p>Takes "stringified" JSON and returns an object representation of the result. You can use the result from this function to access and manipulate elements of the payload natively in Apache Velocity Template Language (VTL). For example, if you have the following payload:</p> <pre>{ "errorMessage": "{ \"key1\": \"var1\", \"key2\": { \"arr\": [1,2,3] } }" }</pre> <p>and use the following mapping template</p> <pre>#set (\$errorMessageObj = \$util.parseJson(\$input.path('\$errorMessage'))) { "errorMessageObjKey2ArrVal" : \$errorMessageObj.key2.arr[0] }</pre> <p>You will get the following output:</p> <pre>{ "errorMessageObjKey2ArrVal" : 1 }</pre>
<code>\$util.urlEncode()</code>	Converts a string into "application/x-www-form-urlencoded" format.
<code>\$util.urlDecode()</code>	Decodes an "application/x-www-form-urlencoded" string.
<code>\$util.base64Encode()</code>	Encodes the data into a base64-encoded string.
<code>\$util.base64Decode()</code>	Decodes the data from a base64-encoded string.

Integration Passthrough Behaviors

When a method request carries a payload and either the Content-Type header does not match any specified mapping template or no mapping template is defined, you can choose to pass the client supplied request payload through the integration request to the back end without transformation. The process is known as integration passthrough. The actual passthrough behavior of an incoming request is determined by the option you choose for a specified mapping template, during [integration request set-up \(p. 75\)](#), and the Content Type header that a client set in the incoming request. The following examples illustrate the possible passthrough behaviors.

Example 1: One mapping template is defined in the integration request for the `application/json` content type.

Content-Type header\Selected passthrough option	WHEN_NO_MATCH	WHEN_NO_TEMPLATE	NEVER
None (default to <code>application/json</code>)	The request payload is transformed using the template.	The request payload is transformed using the template.	The request payload is transformed using the template.
<code>application/json</code>	The request payload is transformed using the template.	The request payload is transformed using the template.	The request payload is transformed using the template.
<code>application/xml</code>	The request payload is not transformed and is sent to the back end as-is.	The request is rejected with an HTTP 415 Unsupported Media Type response.	The request is rejected with an HTTP 415 Unsupported Media Type response.

Example 2: One mapping template is defined in the integration request for the `application/xml` content type.

Content-Type header\Selected passthrough option	WHEN_NO_MATCH	WHEN_NO_TEMPLATE	NEVER
None (default to <code>application/json</code>)	The request payload is not transformed and is sent to the back end as-is.	The request is rejected with an HTTP 415 Unsupported Media Type response.	The request is rejected with an HTTP 415 Unsupported Media Type response.
<code>application/json</code>	The request payload is not transformed and is sent to the back end as-is.	The request is rejected with an HTTP 415 Unsupported Media Type response.	The request is rejected with an HTTP 415 Unsupported Media Type response.
<code>application/xml</code>	The request payload is transformed using the template.	The request payload is transformed using the template.	The request payload is transformed using the template.

Import and Export API Gateway API with Swagger Definition Files

As an alternative to using the Amazon API Gateway console to create and update your API, you can use the API Gateway Import API feature to upload API definitions into API Gateway from external API definition files, such as those using the [Swagger specification](#) with the [API Gateway extensions](#) (p. 147).

After an API is created and configured in API Gateway, you can download it as a Swagger definition file using the Amazon API Gateway Export API. The API Gateway console has enabled this feature for you to export an API using intuitive visual interfaces.

Topics

- [Import an API into API Gateway](#) (p. 142)

- [Export an API from API Gateway \(p. 145\)](#)
- [API Gateway Extensions to Swagger \(p. 147\)](#)

Import an API into API Gateway

You can use the API Gateway Import API feature to import an API from an external definition file into API Gateway. Currently, the Import API feature supports [Swagger v2.0](#) definition files.

With the Import API, you can either create a new API by submitting a [POST request](#) that includes a definition as the payload, or you can update an existing API by using a [PUT request](#) that contains a definition in the payload. You can update an API by overwriting it with a new definition, or merge a definition with an existing API. You specify the options in the request URL using a `mode` query parameter.

Note

For RAML API definitions, you can continue to use [API Gateway Importer](#).

Besides making explicit calls to the REST API, as described below, you can also use the Import API feature in the API Gateway console. The option is available as an item in the **Actions** drop-down menu. For an example of using the Import API feature from the API Gateway console, see [Create an API Gateway API from an Example \(p. 7\)](#).

Use the Import API to Create a New API

To use the Import API feature to create a new API, POST your API definition file to `https://apigateway.<region>.amazonaws.com/restapis?mode=import`. This request results in a new *RestApi*, along with *Resources*, *Models*, and other items defined in the definition file.

The following code snippet shows an example of the POST request with the payload of a JSON-formatted Swagger definition:

```
POST /restapis?mode=import
Host: apigateway.<region>.amazonaws.com
Content-Type: application/json
Content-Length: ...
```

Swagger API definition in JSON (p. 356)

Use the Import API to Update an Existing API

You can use the Import API feature to update an existing API when there are aspects of that API you would like to preserve, such as stages and stage variables, or references to the API from API Keys.

An API update can occur in two modes: merge or overwrite. Merging an API is useful when you have decomposed your external API definitions into multiple, smaller parts and only want to apply changes from one of those parts at a time. For example, this might occur if multiple teams are responsible for different parts of an API and have changes available at different rates. In this mode, items from the existing API that are not specifically defined in the imported definition will be left alone.

Overwriting an API is useful when an external API definition contains the complete definition of an API. In this mode, items from an existing API that are not specifically defined in the imported definition will be deleted.

To merge an API, submit a PUT request to `https://apigateway.<region>.amazonaws.com/restapis/<restapi_id>?mode=merge`. The `restapi_id` path parameter value specifies the API to which the supplied API definition will be merged.

The following code snippet shows an example of the PUT request to merge a Swagger API definition in JSON, as the payload, with the specified API already in API Gateway.

```
PUT /restapis/<restapi_id>?mode=merge
Host:apigateway.<region>.amazonaws.com
Content-Type: application/json
Content-Length: ...
```

A Swagger API definition in JSON (p. 356)

The merging update operation takes two complete API definitions and merges them together. For a small and incremental change, you can use the [resource update](#) operation.

To overwrite an API, submit a PUT request to `https://apigateway.<region>.amazonaws.com/restapis/<restapi_id>?mode=overwrite`. The `restapi_id` path parameter specifies the API that will be overwritten with the supplied API definitions.

The following code snippet shows an example of an overwriting request with the payload of a JSON-formatted Swagger definition:

```
PUT /restapis/<restapi_id>?mode=overwrite
Host:apigateway.<region>.amazonaws.com
Content-Type: application/json
Content-Length: ...
```

A Swagger API definition in JSON (p. 356)

When the `mode` query parameter is not specified, merge is assumed.

Note

The PUT operations are idempotent, but not atomic. That means if a system error occurs part way through processing, the API can end up in a bad state. However, repeating the operation will put the API into the same final state as if the first operation had succeeded.

Swagger `basePath`

In Swagger, you can use the `basePath` property to provide one or more path parts that precede each path defined in the `paths` property. Because API Gateway has several ways to express a resource's path, the Import API feature provides three options for interpreting the `basePath` property during an import:

ignore

If the Swagger file has a `basePath` value of `"/a/b/c"` and the `paths` property contains `"/e"` and `"/f"`, the following POST or PUT request:

```
POST /restapis?mode=import&basepath=ignore
```

```
PUT /restapis/<api_id>?basepath=ignore
```

will result in the following resources in the API:

- /
- /e
- /f

The effect is to treat the `basePath` as if it was not present, and all of the declared API resources are served relative to the host. This can be used, for example, when you have a custom domain name with an API mapping that does not include a *Base Path* and a *Stage* value that refers to your production stage.

Note

API Gateway will automatically create a root resource for you, even if it is not explicitly declared in your definition file.

prepend

If the Swagger file has a `basePath` value of `"/a/b/c"` and the `paths` property contains `"/e"` and `"/f"`, the following POST or PUT request:

```
POST /restapis?mode=import&basepath=prepend
```

```
PUT /restapis/api_id?basepath=prepend
```

will result in the following resources in the API:

- /
- /a
- /a/b
- /a/b/c
- /a/b/c/e
- /a/b/c/f

The effect is to treat the `basePath` as specifying additional resources (without methods) and to add them to the declared resource set. This can be used, for example, when different teams are responsible for different parts of an API and the `basePath` could reference the path location for each team's API part.

Note

API Gateway will automatically create intermediate resources for you, even if they are not explicitly declared in your definition.

split

If the Swagger file has a `basePath` value of `"/a/b/c"` and the `paths` property contains `"/e"` and `"/f"`, the following POST or PUT request:

```
POST /restapis?mode=import&basepath=split
```

```
PUT /restapis/api_id?basepath=split
```

will result in the following resources in the API:

- /
- /b
- /b/c
- /b/c/e
- /b/c/f

The effect is to treat top-most path part, "/a", as the beginning of each resource's path, and to create additional (no method) resources within the API itself. This could, for example, be used when "a" is a stage name that you want to expose as part of your API.

Errors during Import

During the import, errors can be generated for major issues like an invalid Swagger document. Errors are returned as exceptions (e.g., `BadRequestException`) in an unsuccessful response. When an error occurs, the new API definition is discarded and no change is made to the existing API.

Warnings during Import

During the import, warnings can be generated for minor issues like a missing model reference. If a warning occurs, the operation will continue if the `failonwarnings=false` query expression is appended to the request URL. Otherwise, the updates will be rolled back. By default, `failonwarnings` is set to `false`. In such cases, warnings are returned as a field in the resulting [RestApi resource](#). Otherwise, warnings are returned as a message in the exception.

Export an API from API Gateway

Once you created and configured an API in API Gateway, using the API Gateway console or otherwise, you can export it to a Swagger file using the API Gateway Export API, which is part of the Amazon API Gateway Control Service. You have options to include the API Gateway integration extensions, as well as the [Postman](#) extensions, in the exported Swagger definition file.

You cannot export an API if its payloads are not of the `application/json` type. If you try, you will get an error response stating that JSON body models are not found.

Request to Export an API

With the Export API, you export an existing API by submitting a GET request, specifying the to-be-exported API as part of URL paths. The request URL is of the following format:

```
https://<host>/restapis/<restapi_id>/stages/<stage_name>/exports/swagger
```

You can append the `extensions` query string to specify whether to include API Gateway extensions (with the `integration` value) or Postman extensions (with the `postman` value).

In addition, you can set the `Accept` header to `application/json` or `application/yaml` to receive the API definition output in JSON or YAML format, respectively.

For more information about submitting GET requests using the API Gateway Export API, see [Making HTTP Requests](#).

Note

If you define models in your API, they must be for the content type of "application/json" for API Gateway to export the model. Otherwise, API Gateway throws an exception with the "Only found non-JSON body models for ..." error message.

Download API Swagger Definition in JSON

To export and download an API in Swagger definitions in JSON format:

```
GET /restapis/<restapi_id>/stages/<stage_name>/exports/swagger
Host: apigateway.<region>.amazonaws.com
Accept: application/json
```

Here, *<region>* could be, for example, `us-east-1`. For all the regions where API Gateway is available, see [Regions and Endpoints](#)

Download API Swagger Definition in YAML

To export and download an API in Swagger definitions in YAML format:

```
GET /restapis/<restapi_id>/stages/<stage_name>/exports/swagger
Host: apigateway.<region>.amazonaws.com
Accept: application/yaml
```

Download API Swagger Definition with Postman Extensions in JSON

To export and download an API in Swagger definitions with the Postman extension in JSON format:

```
GET /restapis/<restapi_id>/stages/<stage_name>/exports/swagger?
extensions=postman
Host: apigateway.<region>.amazonaws.com
Accept: application/json
```

Download API Swagger Definition with API Gateway Integration in YAML

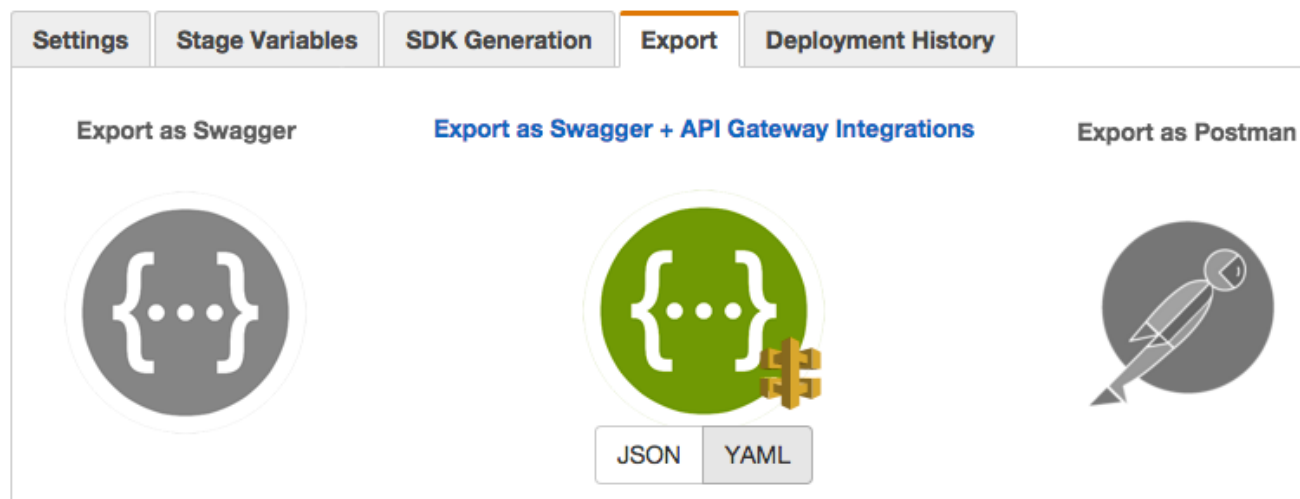
To export and download an API in Swagger definitions with API Gateway integration in YAML format:

```
GET /restapis/<restapi_id>/stages/<stage_name>/exports/swagger?
extensions=integration
Host: apigateway.<region>.amazonaws.com
```

```
Accept: application/yaml
```

Export API Using the API Gateway Console

From the **stage configuration** page in the API Gateway console, choose the **Export** tab and then one of the available options (**Export as Swagger**, **Export as Swagger + API Gateway Integrations** and **Export as Postman**) to download your API's Swagger definition.



API Gateway Extensions to Swagger

The API Gateway extensions support the AWS-specific authorization and API Gateway-specific API integrations. In this section, we will describe the API Gateway extensions to the Swagger specification.

Tip

To understand how the API Gateway extensions are used in an app, you can use the API Gateway console to create an API and export it to a Swagger definition file. For more information on how to export an API, see [Export an API \(p. 145\)](#).

Topics

- [x-amazon-apigateway-any-method Object \(p. 148\)](#)
- [x-amazon-apigateway-authorizer Object \(p. 148\)](#)
- [x-amazon-apigateway-authtype Property \(p. 150\)](#)
- [x-amazon-apigateway-binary-media-types Property \(p. 151\)](#)
- [x-amazon-apigateway-documentation Object \(p. 151\)](#)
- [x-amazon-apigateway-integration Object \(p. 152\)](#)
- [x-amazon-apigateway-integration.requestTemplates Object \(p. 154\)](#)
- [x-amazon-apigateway-integration.requestParameters Object \(p. 155\)](#)
- [x-amazon-apigateway-integration.responses Object \(p. 156\)](#)
- [x-amazon-apigateway-integration.response Object \(p. 157\)](#)
- [x-amazon-apigateway-integration.responseTemplates Object \(p. 158\)](#)
- [x-amazon-apigateway-integration.responseParameters Object \(p. 159\)](#)

x-amazon-apigateway-any-method Object

Specifies the [Swagger Operation Object](#) for the API Gateway catch-all ANY method in a [Swagger Path Item Object](#). This object can exist alongside other Operation objects and will catch any HTTP method that was not explicitly declared.

The following table lists the properties extended by API Gateway. For the other Swagger Operation properties, see the Swagger specification.

Properties

Property Name	Type	Description
x-amazon-apigateway-integration	x-amazon-apigateway-integration (p. 152)	Specifies the integration of the method with the back end. This is an extended property of the Swagger Operation object. The integration can be of type <code>AWS</code> , <code>AWS_PROXY</code> , <code>HTTP</code> , <code>HTTP_PROXY</code> , or <code>MOCK</code> .

x-amazon-apigateway-any-method Example

The following example integrates the ANY method on a proxy resource, `{proxy+}`, with a Lambda function, `TestSimpleProxy`.

```

"/{proxy+}": {
  "x-amazon-apigateway-any-method": {
    "produces": [
      "application/json"
    ],
    "parameters": [
      {
        "name": "proxy",
        "in": "path",
        "required": true,
        "type": "string"
      }
    ],
    "responses": {},
    "x-amazon-apigateway-integration": {
      "uri": "arn:aws:apigateway:us-east-1:lambda:path/2015-03-31/functions/arn:aws:lambda:us-east-1:123456789012:function:TestSimpleProxy/invocations",
      "passthroughBehavior": "when_no_match",
      "httpMethod": "POST",
      "type": "aws_proxy"
    }
  }
}

```

x-amazon-apigateway-authorizer Object

Defines a custom authorizer to be applied for authorization of method invocations in API Gateway. This object is an extended property of the [Swagger Security Definitions Operation](#) object.

Properties

Property Name	Type	Description
type	string	The type of the authorizer. This is a required property and the value must be "token".
authorizerUri	string	The Uniform Resource Identifier (URI) of the authorizer (a Lambda function). For example, <pre>"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"</pre>
authorizerCredentials	string	Credentials required for the authorizer, if any, in the form of an ARN of an IAM execution role. For example, "arn:aws:iam::account-id:IAM_role".
identityValidationExpression	string	A regular expression for validating the incoming identity. For example, "^x-[a-z]+\$".
authorizerResultTtlInSeconds	string	The number of seconds during which the resulting IAM policy is cached.

x-amazon-apigateway-authorizer Example

The following Swagger security definitions example specifies a custom authorizer named `test-authorizer`.

```
"securityDefinitions" : {
  "test-authorizer" : {
    "type" : "apiKey", // Required and the value
    must be "apiKey" for an API Gateway API.
    "name" : "Authorization", // The source header name
    identifying this authorizer.
    "in" : "header", // Required and the value
    must be "header" for an AAPI Gateway API.
    "x-amazon-apigateway-authtype" : "oauth2", // Specifies the
    authorization mechanism for the client.
    "x-amazon-apigateway-authorizer" : { // An API Gateway custom
    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  
  }  
}  
}
```

The following Swagger operation object snippet sets the GET /http to use the custom authorizer specified above.

```
"/http" : {  
  "get" : {  
    "responses" : { },  
    "security" : [ {  
      "test-authorizer" : [ ]  
    } ],  
    "x-amazon-apigateway-integration" : {  
      "type" : "http",  
      "responses" : {  
        "default" : {  
          "statusCode" : "200"  
        }  
      },  
      "httpMethod" : "GET",  
      "uri" : "http://api.example.com"  
    }  
  }  
}
```

x-amazon-apigateway-authtype Property

Specify the type of a custom authorizer. It is parsed by the API Gateway API import and export features.

This property is an extended property of the [Swagger Security Definitions Operation](#) object.

x-amazon-apigateway-authtype Example

The following example sets the type of a custom authorizer using OAuth 2.

```
"cust-authorizer" : {  
  "type" : "...", // required  
  "name" : "...", // name of the identity source header  
  "in" : "header", // must be header  
  "x-amazon-apigateway-authtype" : "oauth2", // Specifies the  
  authorization mechanism for the client.  
  "x-amazon-apigateway-authorizer" : {  
    ...  
  }  
}
```

The following security definition example specifies authorization using [AWS Signature Version 4](#):

```
"sigv4" : {  
  "type" : "apiKey",  
  "name" : "Authorization",  
  "in" : "header",  
  "x-amazon-apigateway-authtype" : "awsSigv4"  
}
```

x-amazon-apigateway-binary-media-types Property

Specifies the list of binary media types to be supported by API Gateway, such as `application/octet-stream`, `image/jpeg`, etc. This extension is a JSON Array.

x-amazon-apigateway-binary-media-types Example

The following example shows the encoding lookup order of an API.

```
"x-amazon-apigateway-binary-media-types": [ "application/octet", "image/  
jpeg" ]
```

x-amazon-apigateway-documentation Object

Defines the documentation parts to be imported into API Gateway. This object is a JSON object containing an array of the `DocumentationPart` instances.

Properties

Property Name	Type	Description
documentationParts	Array	An array of the exported or imported <code>DocumentationPart</code> instances.
version	String	The version identifier of the snapshot of the exported documentation parts.

x-amazon-apigateway-documentation Example

The following example of the API Gateway extension to Swagger defines `DocumentationParts` instances to be imported to or exported from an API in API Gateway.

```
{ ...  
  "x-amazon-apigateway-documentation": {  
    "version": "1.0.3",  
    "documentationParts": [  
      {  
        "location": {  
          "type": "API"  
        },  
      },  
      "properties": {  
        "description": "api description",  
        "info": {
```

```

        "description": "api info description 4",
        "version": "api info version 3"
    },
    {
        ... // Another DocumentationPart instance
    }
]
}

```

x-amazon-apigateway-integration Object

Specifies details of the back-end integration used for this method. This extension is an extended property of the [Swagger Operation](#) object. The result is an [API Gateway integration](#) object.

Properties

Property Name	Type	Description
type	string	The type of integration with the specified back end. The valid value is <code>http</code> (for integration with an HTTP back end) or <code>aws</code> (for integration with AWS Lambda functions or other AWS services, such as DynamoDB, SNS or SQS).
contentHandling	string	Request payload encoding conversion types. Valid values are 1) <code>CONVERT_TO_TEXT</code> , for converting a binary payload into a Base64-encoded string or converting a text payload into a utf-8-encoded string or passing through the text payload natively without modification, and 2) <code>CONVERT_TO_BINARY</code> , for converting a text payload into Base64-decoded blob or passing through a binary payload natively without modification.
uri	string	The endpoint URI of the back end. For integrations of the <code>aws</code> type, this is an ARN value. For the HTTP integration, this is the URL of the HTTP endpoint including the <code>https</code> or <code>http</code> scheme.
httpMethod	string	The HTTP method used in the integration request. For Lambda function invocations, the value must be <code>POST</code> .

Property Name	Type	Description
credentials	string	For AWS IAM role-based credentials, specify the ARN of an appropriate IAM role. If unspecified, credentials will default to resource-based permissions that must be added manually to allow the API to access the resource. For more information, see Granting Permissions Using a Resource Policy . Note: when using IAM credentials, please ensure that AWS STS regional endpoints are enabled for the region where this API is deployed for best performance.
requestTemplates	x-amazon-apigateway-integration.requestTemplates (p. 154)	Mapping templates for a request payload of specified MIME types.
requestParameters	x-amazon-apigateway-integration.requestParameters (p. 154)	Specifies mappings from method request parameters to integration request parameters. Supported request parameters are <code>querystring</code> , <code>path</code> , <code>header</code> , and <code>body</code> .
cacheNamespace	string	An API-specific tag group of related cached parameters.
cacheKeyParameters	An array of string	A list of request parameters whose values are to be cached.
responses	x-amazon-apigateway-integration.responses (p. 156)	Defines the method's responses and specifies desired parameter mappings or payload mappings from integration responses to method responses.

x-amazon-apigateway-integration Example

The following example integrates an API's `POST` method with a Lambda function in the back end. For demonstration purposes, the sample mapping templates shown in `requestTemplates` and `responseTemplates` of the examples below are assumed to apply to the following JSON-formatted payload: `{ "name": "value_1", "key": "value_2", "redirect": { "url" : "..."} }` to generate a JSON output of `{ "stage": "value_1", "user-id": "value_2" }` or an XML output of `<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'"
        }
      }
    }
  }
}

```

Note that double quotes (") of the JSON string in the mapping templates must be string-escaped (\").

x-amazon-apigateway-integration.requestTemplates Object

Specifies mapping templates for a request payload of the specified MIME types.


```
"integration.request.header.x-userid" : "method.request.header.x-user-
id",
"integration.request.path.op" : "method.request.path.service"
},
```

x-amazon-apigateway-integration.responses Object

Defines the method's responses and specifies parameter mappings or payload mappings from integration responses to method responses.

Properties

Property Name	Type	Description
<i>Response status pattern</i>	x-amazon-apigateway-integration.response (p. 157)	<p>Selection regular expression used to match the integration response to the method response. For HTTP integrations, this regex applies to the integration response status code. For Lambda invocations, the regex applies to the <code>errorMessage</code> field of the error information object returned by AWS Lambda as a failure response body when the Lambda function execution throws an exception.</p> <p>Note The <i>Response status pattern</i> property name refers to a response status code or regular expression describing a group of response status codes. It does not correspond to any identifier of an IntegrationResponse resource in the API Gateway REST API.</p>

x-amazon-apigateway-integration.responses Example

The following example shows a list of responses from 2xx and 302 responses. For the 2xx response, the method response is mapped from the integration response's payload of the `application/json` or `application/xml` MIME type. This response uses the supplied mapping templates. For the 302 response, the method response returns a `Location` header whose value is derived from the `redirect.url` property on the integration response's payload.

```
"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 Object

Defines a response and specifies parameter mappings or payload mappings from the integration response to the method response.

Properties

Property Name	Type	Description
statusCode	string	HTTP status code for the method response; for example, "200". This must correspond to a matching response in the Swagger Operation responses field.
responseTemplates	x-amazon-apigateway-integration.responseTemplates (p. 150)	Specifies MIME type-specific mapping templates for the response's payload.
responseParameters	x-amazon-apigateway-integration.responseParameters (p. 150)	Specifies parameter mappings for the response. Only the header and body parameters of the integration response can be mapped to the header parameters of the method.
contentHandling	string	Response payload encoding conversion types. Valid values are 1) <code>CONVERT_TO_TEXT</code> , for converting a binary payload into a Base64-encoded string or converting a text payload into a utf-8-encoded string or passing through the text payload natively without modification, and 2) <code>CONVERT_TO_BINARY</code> , for converting a text payload

Property Name	Type	Description
		into Base64-decoded blob or passing through a binary payload natively without modification.

x-amazon-apigateway-integration.response Example

The following example defines a 302 response for the method that derives a payload of the `application/json` or `application/xml` MIME type from the back end. The response uses the supplied mapping templates and returns the redirect URL from the integration response in the method's `Location` header.

```
{
  "statusCode" : "302",
  "responseTemplates" : {
    "application/json" : "#set ($root=$input.path('$')) { \"stage\": \"$root.name\", \"user-id\": \"$root.key\" }",
    "application/xml" : "#set ($root=$input.path('$')) <stage> $root.name</stage> "
  },
  "responseParameters" : {
    "method.response.header.Location" :
    "integration.response.body.redirect.url"
  }
}
```

x-amazon-apigateway-integration.responseTemplates Object

Specifies mapping templates for a response payload of the specified MIME types.

Properties

Property Name	Type	Description
<i>MIME type</i>	string	Specifies a mapping template to transform the integration response body to the method response body for a given MIME type. For information about creating a mapping template, see Mapping Templates (p. 109) . An example of the <i>MIME type</i> is <code>application/json</code> .

x-amazon-apigateway-integration.responseTemplate Example

The following example sets mapping templates for a request payload of the `application/json` and `application/xml` MIME types.

```
"responseTemplates" : {  
  "application/json" : "#set ($root=$input.path('$')) { \"stage\":  
  \"\${root.name}\", \"user-id\": \"\${root.key}\" }",  
  "application/xml" : "#set ($root=$input.path('$')) <stage>\${root.name}</  
stage> "  
}
```

x-amazon-apigateway-integration.responseParameters Object

Specifies mappings from integration method response parameters to method response parameters. Only the `header` and `body` types of the integration response parameters can be mapped to the `header` type of the method response.

Properties

Property Name	Type	Description
<code>method.response.header.<parameter name></code>	<code>string</code>	The named parameter value can be derived from the <code>header</code> and <code>body</code> types of the integration response parameters only.

x-amazon-apigateway-integration.responseParameters Example

The following example maps `body` and `header` parameters of the integration response to two `header` parameters of the method response.

```
"responseParameters" : {  
  "method.response.header.Location" :  
  "integration.response.body.redirect.url",  
  "method.response.header.x-user-id" : "integration.response.header.x-  
userid"  
}
```

Controlling Access in API Gateway

API Gateway supports multiple mechanisms of access control, including metering or tracking API uses by clients using API keys. The standard AWS IAM roles and policies offer flexible and robust access controls that can be applied to an entire API set or individual methods. Custom authorizers and Amazon Cognito user pools provide customizable authorization and authentication solutions.

Topics

- [Use IAM Permissions to Access API Gateway API \(p. 160\)](#)
- [Enable CORS for an API Gateway Resource \(p. 170\)](#)
- [Use Amazon API Gateway Custom Authorizers \(p. 174\)](#)
- [Use Amazon Cognito Your User Pool \(p. 182\)](#)
- [Use Client-Side SSL Certificates for Authentication by the Back End \(p. 186\)](#)
- [API Gateway-Supported Certificate Authorities for HTTP and HTTP Proxy Integrations \(p. 189\)](#)
- [Create and Use API Gateway Usage Plans \(p. 212\)](#)

Use IAM Permissions to Access API Gateway API

Topics

- [Control Access to API Gateway with IAM Policies \(p. 160\)](#)
- [Create and Attach a Policy to an IAM User \(p. 162\)](#)
- [Statement Reference of IAM Policies for Managing API in API Gateway \(p. 163\)](#)
- [Statement Reference of IAM Policies for Executing API in API Gateway \(p. 164\)](#)
- [IAM Policy Examples for API Gateway APIs \(p. 165\)](#)
- [IAM Policy Examples for API Execution Permissions \(p. 169\)](#)

Control Access to API Gateway with IAM Policies

When working with Amazon API Gateway, you access two services. You use one to create, configure, deploy and update your API and the other to actually execute your deployed API upon requests by a client. When setting access permissions in an IAM policy, you reference the API managing service as

apigateway and the API executing service as `execute-api`. The `apigateway` service supports the actions of GET, POST, PUT, PATCH, DELETE, OPTIONS, HEAD and the `execute-api` service supports the `Invoke` and `InvalidateCache` actions. To create an IAM policy using the Policy Generator in the IAM console, select **Manage Amazon API Gateway** as **AWS Service** to set permissions statements for `apigateway` and select **Amazon API Gateway** as **AWS Service** to set permission statements for `execute-api`.

You can use IAM to allow IAM users and roles in your AWS account to manage only certain API Gateway entities (for example, APIs, resources, methods, models, and stages) and perform only certain actions against those entities. You may want to do this, for example, if you have IAM users you want to allow to list, but not create, resources and methods for selected APIs. You may have other IAM users you want to allow to list and create new resources and methods for any API they have access to in API Gateway.

In the [Get Ready to Use API Gateway \(p. 4\)](#) instructions, you attached an access policy to an IAM user in your AWS account that contains a policy statement similar to this:

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

This statement allows the IAM user in your AWS account to perform all available actions and access all available resources in API Gateway to which your AWS account has access. In practice, you may not want to give the IAM users in your AWS account this much access.

You can also use IAM to enable users inside your organization to interact with only certain API methods in API Gateway.

In the [Configure How a User Calls an API Method \(p. 74\)](#) instructions, the API Gateway console may have displayed a resource ARN you used to create a policy statement similar to this:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "execute-api:Invoke"
      ],
      "Resource": [
        "arn:aws:execute-api:us-east-1:my-aws-account-id:my-api-id/my-stage/  
GET/my-resource-path"
      ]
    }
  ]
}
```

This statement allows the IAM user to call the GET method for the resource path associated with the specified resource ARN in API Gateway. In practice, you may want to give IAM users access to more methods.

Note

IAM policies are effective only if IAM authentication is enabled. If you, as the API owner, has enabled AWS identity and access management on a specific resource, users from other AWS accounts cannot access your API. If you do not enable IAM authentication on the resource, that resource is effectively public accessible.

Create and Attach a Policy to an IAM User

To create and attach an access policy to an IAM user that restricts the API Gateway entities the IAM user can manage or the API methods the IAM user can call, do the following:

1. Sign in to the Identity and Access Management (IAM) console at <https://console.aws.amazon.com/iam/>.
2. Choose **Policies**, and then choose **Create Policy**. (If a **Get Started** button appears, choose it, and then choose **Create Policy**.)
3. Next to **Create Your Own Policy**, choose **Select**.
4. For **Policy Name**, type any value that will be easy for you to refer to later, if needed.
5. For **Policy Document**, type a policy statement with the following format, and then choose **Create Policy**:

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

In this statement, substitute *action-statement* and *resource-statement* as needed, and add additional statements as needed, to specify the API Gateway entities you want to allow the IAM user to manage, the API methods the IAM user can call, or both. (By default, the IAM user will not have permissions unless a corresponding Allow statement is explicitly stated.)

6. Choose **Users**.
7. Choose the IAM user to whom you want to attach the policy.
8. For **Permissions**, for **Managed Policies**, choose **Attach Policy**.

9. Select the policy you just created, and then choose **Attach Policy**.

Statement Reference of IAM Policies for Managing API in API Gateway

The following information describes the `Action` and `Resource` format used in an IAM policy statement to grant or revoke permissions for managing API Gateway API entities, such as restapis, resources, methods, models, stages, custom domain names, API keys, etc.

Action Format of Permissions for Managing API in API Gateway

The API-managing `Action` expression has the following general format:

```
apigateway:action
```

where *action* is one of the following API Gateway actions:

- `*`, which represents all of the following actions.
- `GET`, which is used to get information about resources.
- `POST`, which is primarily used to create child resources.
- `PUT`, which is primarily used to update resources (and, although not recommended, can be used to create child resources).
- `DELETE`, which is used to delete resources.
- `PATCH`, which can be used to update resources.
- `HEAD`, which is the same as GET but does not return the resource representation. HEAD is used primarily in testing scenarios.
- `OPTIONS`, which can be used by callers to get information about available communication options for the target service.

Some examples of the `Action` expression include:

- `apigateway:*` for all API Gateway actions.
- `apigateway:GET` for just the GET action in API Gateway.

Resource Format of Permissions for Managing API in API Gateway

The API-managing `Resource` expression has the following general format:

```
arn:aws:apigateway:region::resource-path-specifier
```

where *region* is a target AWS region (such as `us-east-1` or `*` for all supported AWS regions), and *resource-path-specifier* is the path to the target resources.

Some example resource expressions include:

- `arn:aws:apigateway:region::/restapis/*` for all resources, methods, models, and stages in the AWS region of *region*.

- `arn:aws:apigateway:region::restapis/api-id/*` for all resources, methods, models, and stages in the API with the identifier of `api-id` in the AWS region of `region`.
- `arn:aws:apigateway:region::restapis/api-id/resources/resource-id/*` for all resources and methods in the resource with the identifier `resource-id`, which is in the API with the identifier of `api-id` in the AWS region of `region`.
- `arn:aws:apigateway:region::restapis/api-id/resources/resource-id/methods/*` for all of the methods in the resource with the identifier `resource-id`, which is in the API with the identifier of `api-id` in the AWS region of `region`.
- `arn:aws:apigateway:region::restapis/api-id/resources/resource-id/methods/GET` for just the GET method in the resource with the identifier `resource-id`, which is in the API with the identifier of `api-id` in the AWS region of `region`.
- `arn:aws:apigateway:region::restapis/api-id/models/*` for all of the models in the API with the identifier of `api-id` in the AWS region of `region`.
- `arn:aws:apigateway:region::restapis/api-id/models/model-name` for the model with the name of `model-name`, which is in the API with the identifier of `api-id` in the AWS region of `region`.
- `arn:aws:apigateway:region::restapis/api-id/stages/*` for all of the stages in the API with the identifier of `api-id` in the AWS region of `region`.
- `arn:aws:apigateway:region::restapis/api-id/stages/stage-name` for just the stage with the name of `stage-name` in the API with the identifier of `api-id` in the AWS region of `region`.

Statement Reference of IAM Policies for Executing API in API Gateway

The following information describes the Action and Resource format of IAM policy statements of access permissions for executing an API.

Action Format of Permissions for Executing API in API Gateway

The API-executing `Action` expression has the following general format:

```
execute-api:action
```

where `action` is an available API-executing action:

- `*`, which represents all of the following actions.
- **Invoke**, used to invoke an API upon a client request.
- **InvalidateCache**, used to invalidate API cache upon a client request.

Resource Format of Permissions for Executing API in API Gateway

The API-executing `Resource` expression has the following general format:

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

where:

- *region* is the AWS region (such as `us-east-1` or `*` for all AWS regions) that corresponds to the deployed API for the method.
- *account-id* is the 12-digit AWS account Id of the REST API owner.
- *api-id* is the identifier API Gateway has assigned to the API for the method. (`*` can be used for all APIs, regardless of the API's identifier.)
- *stage-name* is the name of the stage associated with the method (`*` can be used for all stages, regardless of the stage's name.)
- *HTTP-VERB* is the HTTP verb for the method. It can be one of the following: GET, POST, PUT, DELETE, PATCH, HEAD, OPTIONS.
- *resource-path-specifier* is the path to the desired method. (`*` can be used for all paths).

Some example resource expressions include:

- `arn:aws:execute-api:*:*:*` for any resource path in any stage, for any API in any AWS region. (This is equivalent to `*`).
- `arn:aws:execute-api:us-east-1:*:*` for any resource path in any stage, for any API in the AWS region of `us-east-1`.
- `arn:aws:execute-api:us-east-1:*:api-id/*` for any resource path in any stage, for the API with the identifier of *api-id* in the AWS region of `us-east-1`.
- `arn:aws:execute-api:us-east-1:*:api-id/test/*` for resource path in the stage of `test`, for the API with the identifier of *api-id* in the AWS region of `us-east-1`.
- `arn:aws:execute-api:us-east-1:*:api-id/test/*/mydemoresource/*` for any resource path along the path of *mydemoresource*, for any HTTP method in the stage of `test`, for the API with the identifier of *api-id* in the AWS region of `us-east-1`.
- `arn:aws:execute-api:us-east-1:*:api-id/test/GET/mydemoresource/*` for GET methods under any resource path along the path of *mydemoresource*, in the stage of `test`, for the API with the identifier of *api-id* in the AWS region of `us-east-1`.

IAM Policy Examples for API Gateway APIs

The following example policy documents shows various use cases to set access permissions for managing API resources in API Gateway. For permissions model and other background information, see [Control Access to API Gateway with IAM Policies](#) (p. 160).

Topics

- [Simple Read Permissions](#) (p. 165)
- [Read-Only Permissions on any APIs](#) (p. 166)
- [Full Access Permissions for any API Gateway Resources](#) (p. 167)
- [Full-Access Permissions for Managing API Stages](#) (p. 168)
- [Block Specified Users from Deleting any API Resources](#) (p. 168)

Simple Read Permissions

The following policy statement gives the user permission to get information about all of the resources, methods, models, and stages in the API with the identifier of `a123456789` in the AWS region of `us-east-1`:

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

```
    "Effect": "Allow",
    "Action": [
      "apigateway:GET"
    ],
    "Resource": [
      "arn:aws:apigateway:us-east-1::/restapis/a123456789/*"
    ]
  }
]
```

The following example policy statement gives the IAM user permission to list information for all resources, methods, models, and stages in any region. The user also has permission to perform all available API Gateway actions for the API with the identifier of a123456789 in the AWS region of us-east-1:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "apigateway:GET"
      ],
      "Resource": [
        "arn:aws:apigateway:*::/restapis/*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "apigateway:*"
      ],
      "Resource": [
        "arn:aws:apigateway:us-east-1::/restapis/a123456789/*"
      ]
    }
  ]
}
```

Read-Only Permissions on any APIs

The following policy document will permit attached entities (users, groups or roles) to retrieve any of the APIs of the caller's AWS account. This includes any of the child resources of an API, such as method, integration, etc.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Stmt1467321237000",
      "Effect": "Deny",
      "Action": [
        "apigateway:POST",
        "apigateway:PUT",
        "apigateway:PATCH",
        "apigateway:DELETE"
      ]
    }
  ]
}
```

```
    ],
    "Resource": [
      "arn:aws:apigateway:us-east-1:/*"
    ]
  },
  {
    "Sid": "Stmt1467321341000",
    "Effect": "Deny",
    "Action": [
      "apigateway:GET",
      "apigateway:HEAD",
      "apigateway:OPTIONS"
    ],
    "Resource": [
      "arn:aws:apigateway:us-east-1:/",
      "arn:aws:apigateway:us-east-1:/account",
      "arn:aws:apigateway:us-east-1:/clientcertificates",
      "arn:aws:apigateway:us-east-1:/domainnames",
      "arn:aws:apigateway:us-east-1:/apikeys"
    ]
  },
  {
    "Sid": "Stmt1467321344000",
    "Effect": "Allow",
    "Action": [
      "apigateway:GET",
      "apigateway:HEAD",
      "apigateway:OPTIONS"
    ],
    "Resource": [
      "arn:aws:apigateway:us-east-1:/restapis/*"
    ]
  }
]
```

The first `Deny` statement explicitly prohibits any calls of `POST`, `PUT`, `PATCH`, `DELETE` on any resources in API Gateway. This ensures that such permissions will not be overridden by other policy documents also attached to the caller. The second `Deny` statement blocks the caller to query the root (`/`) resource, account information (`/account`), client certificates (`/clientcertificates`), custom domain names (`/domainnames`) and API keys (`/apikeys`). Together, the three statements ensure that the caller can only query API-related resources. This can be useful in API testing when you do not want the tester to modify any of the code.

To restrict the above read-only access to specified APIs, replace the `Resource` property of `Allow` statement by the following:

```
"Resource": [ "arn:aws:apigateway:us-east-1:/restapis/restapi_id1/*",
              "arn:aws:apigateway:us-east-1:/restapis/restapi_id2/*" ]
```

Full Access Permissions for any API Gateway Resources

The following example policy document grants the full access to any of the API Gateway resource of the AWS account.

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

```
"Statement": [
  {
    "Sid": "Stmt1467321765000",
    "Effect": "Allow",
    "Action": [
      "apigateway:*"
    ],
    "Resource": [
      "*"
    ]
  }
]
```

In general, you should refrain from using such a broad and open access policy. It may be necessary to do so for your API development core team so that they can create, deploy, update, and delete any API Gateway resources.

Full-Access Permissions for Managing API Stages

The following example policy documents grants full-access permissions on Stage related resources of any API in the caller's AWS account.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "apigateway:*"
      ],
      "Resource": [
        "arn:aws:apigateway:us-east-1::/restapis/*/stages",
        "arn:aws:apigateway:us-east-1::/restapis/*/stages/*"
      ]
    }
  ]
}
```

The above policy document grants full access permissions only to the `stages` collection and any of the contained `stage` resources, provided that no other policies granting more accesses have been attached to the caller. Otherwise, you must explicitly deny all the other accesses.

Using the above policy, caller must find out the REST API's identifier beforehand because the user cannot call `GET /restapis` to query the available APIs. Also, if `arn:aws:apigateway:us-east-1::/restapis/*/stages` is not specified in the `Resource` list, the `Stages` resource becomes inaccessible. In this case, the caller will not be able to create a stage nor get the existing stages, although he or she can still view, update or delete a stage, provided that he stage's name is known.

To grant permissions for a specific API's stages, simply replace the `restapis/*` portion of the `Resource` specifications by `restapis/restapi_id`, where *restapi_id* is the identifier of the API of interest.

Block Specified Users from Deleting any API Resources

The following example IAM policy document blocks a specified user from deleting any API resources in API Gateway.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Stmt1467331998000",
      "Effect": "Allow",
      "Action": [
        "apigateway:GET",
        "apigateway:HEAD",
        "apigateway:OPTIONS",
        "apigateway:PATCH",
        "apigateway:POST",
        "apigateway:PUT"
      ],
      "Resource": [
        "arn:aws:apigateway:us-east-1::/restapis/*"
      ]
    },
    {
      "Sid": "Stmt1467332141000",
      "Effect": "Allow",
      "Action": [
        "apigateway:DELETE"
      ],
      "Condition": {
        "StringNotLike": {
          "aws:username": "johndoe"
        }
      },
      "Resource": [
        "arn:aws:apigateway:us-east-1::/restapis/*"
      ]
    }
  ]
}
```

This IAM policy grants full access permission to create, deploy, update and delete API for attached users, groups or roles, except for the specified user (johndoe), who cannot delete any API resources. It assumes that no other policy document granting `Allow` permissions on the root, API keys, client certificates or custom domain names has been attached to the caller.

To block the specified user from deleting specific API Gateway resources, e.g., a specific API or an API's resources, replace the `Resource` specification above by this:

```
"Resource": [ "arn:aws:apigateway:us-east-1::/restapis/restapi_id_1",
              "arn:aws:apigateway:us-east-1::/restapis/restapi_id_2/resources" ]
```

IAM Policy Examples for API Execution Permissions

For permissions model and other background information, see [Control Access to API Gateway with IAM Policies](#) (p. 160).

The following policy statement gives the user permission to call any POST method along the path of `mydemoresource`, in the stage of `test`, for the API with the identifier of `a123456789`, assuming the corresponding API has been deployed to the AWS region of `us-east-1`:

```
{
```

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

The following example policy statement gives the user permission to call any method on the resource path of `petstorewalkthrough/pets`, in any stage, for the API with the identifier of `a123456789`, in any AWS region where the corresponding API has been deployed:

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

Enable CORS for an API Gateway Resource

When your API's resources receive requests from a domain other than the API's own domain, you must enable cross-origin resource sharing (CORS) for selected methods on the resource. This amounts to having your API respond to the `OPTIONS` preflight request with at least the following CORS-required response headers:

- `Access-Control-Allow-Methods`
- `Access-Control-Allow-Headers`
- `Access-Control-Allow-Origin`

In API Gateway you enable CORS by setting up an `OPTIONS` method with the mock integration type to return the preceding response headers (with static values discussed in the following) as the method response headers. In addition, the actual CORS-enabled methods must also return the `Access-Control-Allow-Origin: '*'` header in at least its 200 response.

Tip

You must set up an `OPTIONS` method to handle preflight requests to support CORS. However, `OPTIONS` methods are optional if 1) an API resource exposes only the `GET`, `HEAD` or `POST` methods and 2) the request payload content type is `application/x-www-form-urlencoded`, `multipart/form-data` or `text/plain` and 3) the request does not contain

any custom headers. When possible, we recommend to use OPTIONS method to enable CORS in your API.

This section describes how to enable CORS for a method in API Gateway using the API Gateway console or the API Gateway [Import API](#).

Topics

- [Prerequisites \(p. 171\)](#)
- [Enable CORS on a Resource Using the API Gateway Console \(p. 171\)](#)
- [Enable CORS for a Resource Using the API Gateway Import API \(p. 172\)](#)

Prerequisites

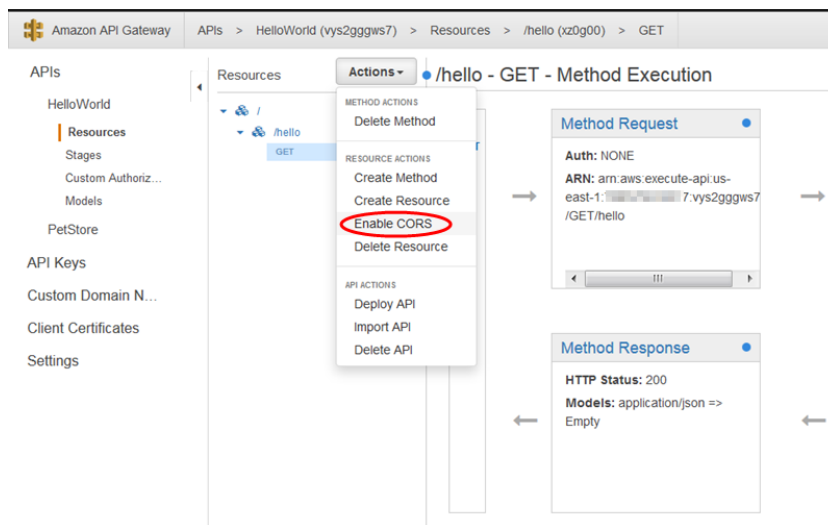
- You must have the method available in API Gateway. For instructions on how to create and configure a method, see [Build an API Gateway API to Expose an HTTP Endpoint \(p. 6\)](#).

Enable CORS on a Resource Using the API Gateway Console

1. Sign in to the API Gateway console at <https://console.aws.amazon.com/apigateway>.
2. In the API Gateway console, choose an API under **APIs**.
3. Choose a resource under **Resources**. This will enable CORS for all the methods on the resource.

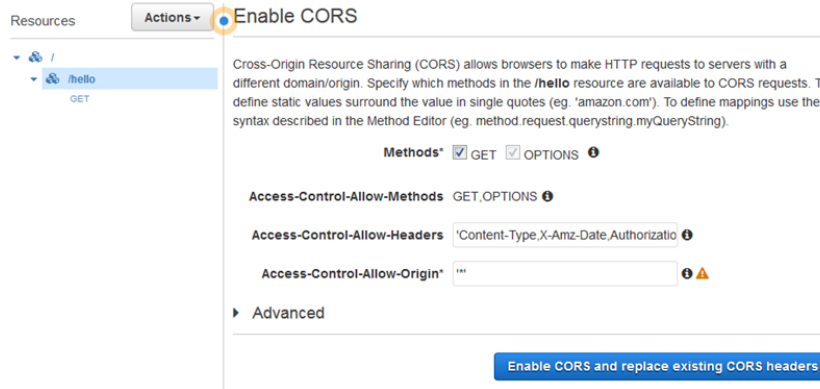
Alternatively, you could choose a method under the resource to enable CORS for just this method.

4. Choose **Enable CORS** from the **Actions** drop-down menu.

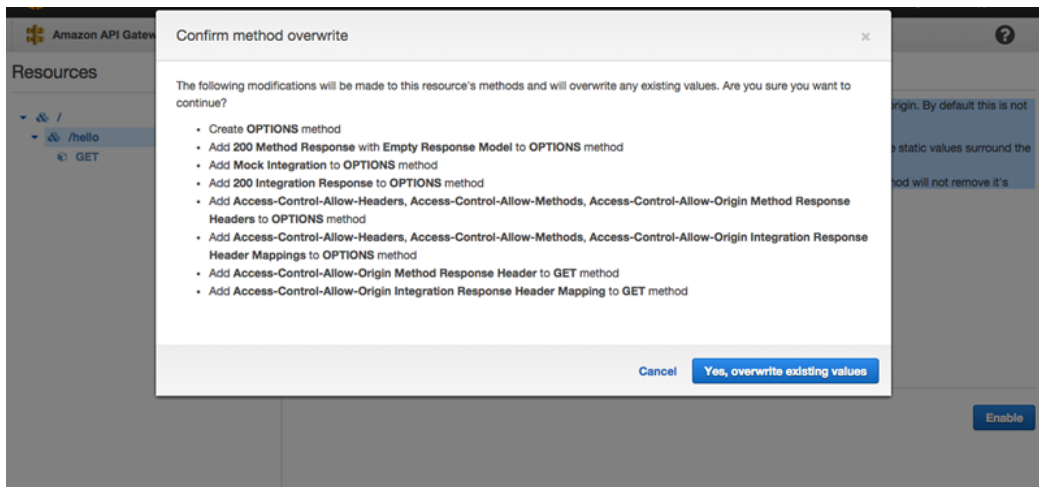


5. In the **Enable CORS** form, do the following:
 - a. In the **Access-Control-Allow-Headers** input field, type a static string of a comma-separated list of headers that the client must submit in the actual request of the resource. Use the console-provided header list of 'Content-Type,X-Amz-Date,Authorization,X-Api-Key,X-Amz-Security-Token' or specify your own headers.
 - b. Use the console-provided value of '*' as the **Access-Control-Allow-Origin** header value to allow access requests from all domains, or specify a named domain to all access requests from the specified domain.

- c. Choose **Enable CORS** and replace existing CORS headers.



6. In **Confirm method changes**, choose **Yes, overwrite existing values** to confirm the new CORS settings.



After CORS is enabled on the `GET` method, an `OPTIONS` method is added to the resource, if it is not already there. The `200` response of the `OPTIONS` method is automatically configured to return the three `Access-Control-Allow-*` headers to fulfill preflight handshakes. In addition, the actual (`GET`) method is also configured by default to return the `Access-Control-Allow-Origin` header in its `200` response as well. For other types of responses, you will need to manually configure them to return `Access-Control-Allow-Origin` header with `'*'` or specific origin domain names, if you do not want to return the Cross-origin access error.

Enable CORS for a Resource Using the API Gateway Import API

If you are using the [API Gateway Import API](#) (p. 142), you can set up CORS support using a Swagger file. You must first define an `OPTIONS` method in your resource that returns the required headers.

Note

Web browsers expect `Access-Control-Allow-Headers`, and `Access-Control-Allow-Origin` headers to be set up in each API method that accepts CORS requests. In addition, some browsers first make an HTTP request to an `OPTIONS` method in the same resource, and then expect to receive the same headers.

The following example creates an `OPTIONS` method and specifies mock integration. For more information, see [Configure Mock Integration for a Method \(p. 94\)](#).

```
/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:
        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 : "'*'"
          responseTemplates:
            application/json: |
              {}
      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"
```

Once you have configured the `OPTIONS` method for your resource, you can add the required headers to the other methods in the same resource that need to accept CORS requests.

1. Declare the **Access-Control-Allow-Origin** and **Headers** to the response types.

```
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. In the `x-amazon-apigateway-integration` tag, set up the mapping for those headers to your static values:

```
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 : "'*'"
```

Use Amazon API Gateway Custom Authorizers

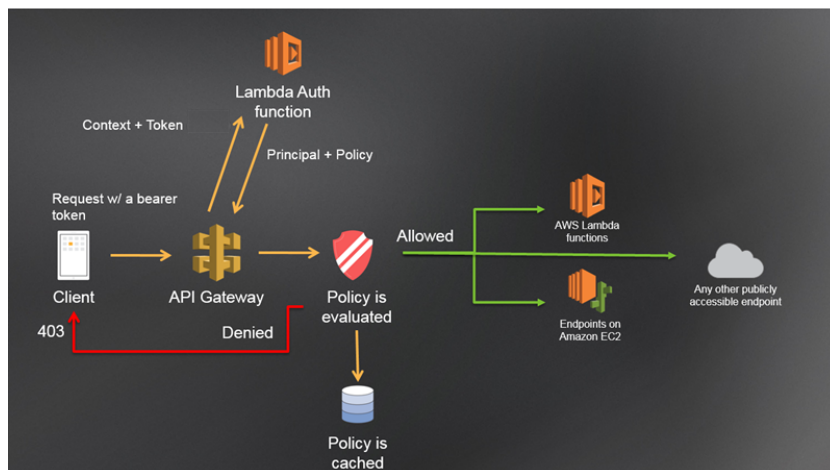
Topics

- [Amazon API Gateway Custom Authorizers \(p. 174\)](#)
- [Create the API Gateway Custom Authorizer Lambda Function \(p. 175\)](#)
- [Input to an Amazon API Gateway Custom Authorizer \(p. 176\)](#)
- [Output from an Amazon API Gateway Custom Authorizer \(p. 177\)](#)
- [Configure Custom Authorizer Using the API Gateway Console \(p. 178\)](#)
- [Call an API Using API Gateway Custom Authorization \(p. 180\)](#)

Amazon API Gateway Custom Authorizers

An Amazon API Gateway custom authorizer is a Lambda function that you provide to control access to your APIs using bearer token authentication strategies, such as OAuth or SAML.

When a client calls your API, API Gateway verifies whether a custom authorizer is configured for the API. If so, API Gateway calls the Lambda function, supplying the authorization token extracted from a specified request header. You use this Lambda function to implement various authorization strategies, such as JSON Web Token (JWT) verification and OAuth provider callout, to return IAM policies that authorize the request. If the returned policy is invalid or the permissions are denied, the API call will not succeed. For a valid policy, API Gateway caches the returned policy, associated with the incoming token and used for the current and subsequent requests, over a pre-configured time-to-live (TTL) period of up to 3600 seconds. You can set the TTL period to zero seconds to disable the policy caching. The default TTL value is 300 seconds. Currently, the maximum TTL value of 3600 seconds cannot be increased.



Create the API Gateway Custom Authorizer Lambda Function

Before creating an API Gateway custom authorizer, you must first create the AWS Lambda function that implements the logic to authenticate and authorize the caller. You can do so in the Lambda console, using the code template available from the API Gateway Custom Authorizer blueprint. Or you can create one from scratch. For illustration purposes, we will explain here the creation of the Lambda function without using the blueprint.

Note

The custom authorizer Lambda function presented here is for illustration purposes. In production code, you should follow the API Gateway Custom Authorizer blueprint to implement your authorizer Lambda function.

When creating the Lambda function for your API Gateway custom authorizer, you will be asked to assign an execution role for the Lambda function if it calls other AWS services. For the following example, the basic `AWSLambdaRole` will suffice. For more involved use cases, follow the [instructions](#) to grant permissions in an execution role for the Lambda function.

In the code editor of the Lambda console, enter the following Node.js code.

```
console.log('Loading function');

exports.handler = function(event, context) {
  var token = event.authorizationToken;
  // Call oauth provider, crack jwt token, etc.
  // In this example, the token is treated as the status for simplicity.

  switch (token) {
    case 'allow':
      context.succeed(generatePolicy('user', 'Allow',
event.methodArn));
      break;
    case 'deny':
      context.succeed(generatePolicy('user', 'Deny', event.methodArn));
      break;
    case 'unauthorized':
      context.fail("Unauthorized");
      break;
    default:
```

```
        context.fail("error");
    }
};

var generatePolicy = function(principalId, effect, resource) {
    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;
    }
    return authResponse;
}
```

The preceding Lambda function returns an Allow IAM policy on a specified method if the request's authorization token contains an 'allow' value, thereby permitting a caller to invoke the specified method. The caller receives a 200 OK response. The function returns a Deny policy against the specified method if the authorization token has a 'deny' value, thus blocking the caller from calling the method. The client will receive a 403 Forbidden response. If the token is 'unauthorized', the client will receive a 401 Unauthorized response. If the token is 'fail' or anything else, the client will receive a 500 Internal Server Error response. In both of the last two cases, the calls will not succeed.

Note

In production code, you may need to authenticate the user before granting authorizations. If so, you can add authentication logic in the Lambda function as well. Consult the provider-specific documentation for instructions on how to call such an authentication provider.

Besides returning an IAM policy in a custom authorizer, the Lambda function must also return the caller's principal identifier and can also return a key-value map named `context` to contain additional information. For example, see [Output from an Amazon API Gateway Custom Authorizer \(p. 177\)](#).

You can then configure the integration request to pass returned `context` map entries to the back end. When the `context` map entries refer to cached credentials, the back-end service can provide better latency and, hence, improved custom experience by leveraging the cached credentials to reduce the need to access the secret keys and open the authorization tokens for every request.

Before going further, you may want to test the Lambda function from within the Lambda Console. To do this, configure the sample event to provide the input and verify the result by examining the output. The next two sections explain the [Input to a Custom Authorizer \(p. 176\)](#) and [Output from an Amazon API Gateway Custom Authorizer \(p. 177\)](#).

Input to an Amazon API Gateway Custom Authorizer

When a custom authorizer is enabled on an API method, you must specify a custom header for the method caller to pass the required authorization token in the initial client request. Upon receiving the request, API Gateway extracts the token from the custom header as the input `authorizationToken` parameter value into the Lambda function and calls the custom authorizer with the following request payload.

```
{
  "type": "TOKEN",
  "authorizationToken": "<caller-supplied-token>",
  "methodArn": "arn:aws:execute-
api:<regionId>:<accountId>:<apiId>/<stage>/<method>/<resourcePath>"
}
```

In this example, the `type` property specifies the payload type. Currently, the only valid value is the `TOKEN` literal. The `<caller-supplied-token>` originates from the custom authorization header in a client request. The `methodArn` is the ARN of the incoming method request and is populated by API Gateway in accordance with the custom authorizer configuration.

For the custom authorizer shown in the preceding section, the `<caller-supplied-token>` string is `allow`, `deny`, `unauthorized`, or any other string value. An empty string value is the same as `unauthorized`. The following shows an example of such an input to obtain an `Allow` policy on the `GET` method of an API (`ymy8tbxw7b`) of the AWS account (`123456789012`) in any stage (`*`).

```
{
  "type": "TOKEN",
  "authorizationToken": "allow",
  "methodArn": "arn:aws:execute-api:us-west-2:123456789012:ymy8tbxw7b/* /
GET/"
}
```

Output from an Amazon API Gateway Custom Authorizer

The custom authorizer's Lambda function returns an output that must include the principal identifier (`principalId`) and a policy document (`policyDocument`) containing a list of policy statements. The output can also include a `context` map containing key-value pairs. The following shows an example of this output.

```
{
  "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>:<appId>/<stage>/<httpVerb>/[<resource>/<httpVerb>/
[...]]"
      }
    ]
  },
  "context": {
    "key": "value",
    "numKey": 1,
  }
}
```

```
    "boolKey": true
  }
}
```

Here, a policy statement stipulates whether to allow or deny (**Effect**) the API Gateway execution service to invoke (**Action**) the specified API method (**Resource**). You can use a wild card (*) to specify a resource type (method). For information about setting valid policies for calling an API, see [Statement Reference of IAM Policies for Executing API in API Gateway \(p. 164\)](#).

You can access the `principalId` value in a mapping template using the `$context.authorizer.principalId` variable. This is useful if you want to pass the value to the back end. For more information, see [Accessing the \\$context Variable \(p. 134\)](#).

You can access the `key`, `numKey`, or `boolKey` value (i.e., `value`, `1`, or `true`) of the context map in a mapping template by calling `$context.authorizer.key`, `$context.authorizer.numKey`, or `$context.authorizer.boolKey`, respectively. Notice that a JSON object or array is not a valid value of any key in the context map.

The following shows example output from the example custom authorizer. The example output contains a policy statement to block (**Deny**) calls to the **GET** method in an API (`yymy8tbxw7b`) of an AWS account (`123456789012`) in any stage (*).

```
{
  "principalId": "user",
  "policyDocument": {
    "Version": "2012-10-17",
    "Statement": [
      {
        "Action": "execute-api:Invoke",
        "Effect": "Deny",
        "Resource": "arn:aws:execute-api:us-west-2:123456789012:yymy8tbxw7b/*/"
      }
    ]
  }
}
```

Configure Custom Authorizer Using the API Gateway Console

After you create the Lambda function and verify that it works, you can configure the API Gateway Custom Authorizer in the API Gateway console.

Enable a Custom Authorizer on API Methods

1. Sign in to the API Gateway console.
2. Create a new or select an existing API and choose **Authorizers**.
3. Choose **Create**, select **Custom Authorizer**, and do the following:
 - In **Lambda region**, select the region where you upload your custom authorizer's Lambda function.
 - In **Lambda function**, select the Lambda function for your custom authorizer.

Note

You must first create a custom authorizer Lambda function in the region for it to be available in the drop-down list.

- In **Authorizer Name**, enter a name for your new custom authorizer.
- Leave **Execution role** blank to let the API Gateway console to set a resource-based policy to grant API Gateway permissions to invoke the authorizer Lambda function or type the name of an IAM role to allow API Gateway to invoke the authorizer Lambda function. For an example of such a role, see [Set Up an IAM Role and Policy for an API to Invoke Lambda Functions](#) (p. 345).
- In **Identity token source**, type the mapping expression for your authorizer's custom header.

Note

The custom header mapping expression is of the `method.request.header.<name>` format, where `<name>` is the name of a custom authorization header submitted as part of the client request. In the following example, this custom header name is `Auth`.

- In **Token validation expression**, you can optionally provide a RegEx statement for API Gateway to validate the input token before calling the custom authorizer Lambda function. This helps you avoid or reduce the chances of being charged for processing invalid tokens.
- In **Result TTL in seconds**, you can change or use the default (300) value to enable caching (>0) or disable caching (=0) of the policy returned from the Lambda function.

Note

The policy caching uses a cache key generated from the supplied token for the targeted API and custom authorizer in a specified stage. To enable caching, your authorizer must return a policy that is applicable to all methods across an API. To enforce method-specific policy, you can set the TTL value to zero to disable policy caching for the API.

The screenshot shows the 'New Custom Authorizer' form in the AWS API Gateway console. The form is titled 'New Custom Authorizer' and includes a 'Create' button in the top left. Below the title, there is a instruction: 'Provide a name, Lambda function, and identity token source for your authorizer.' The form contains several fields, each with an asterisk indicating it is required: 'Lambda region*' (dropdown menu showing 'us-west-2'), 'Lambda function*' (text input 'myCustomAuthorizer'), 'Authorizer name*' (text input 'myTestApiAuthorizer'), 'Execution role' (text input 'arn:aws:iam::myAccount:role/myRole'), 'Identity token source*' (text input 'method.request.header.Auth'), 'Token validation expression' (empty text input), and 'Result TTL in seconds*' (text input '300'). There are information icons (i) next to the Lambda function, Authorizer name, Execution role, Identity token source, and Result TTL in seconds fields. At the bottom right, there are 'Cancel' and 'Create' buttons. A note at the bottom left says '* Required'.

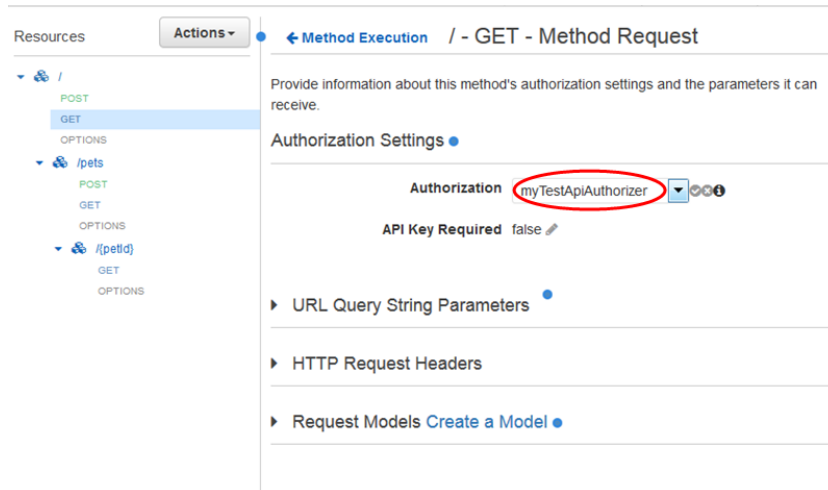
4. If you choose to let the API Gateway console to set the resource-based policy, the **Add Permission to Lambda Function** dialog will be displayed. Choose **OK**. After the custom authorization is created, you can test it with appropriate authorization token values to verify that it works as expected.

This completes the procedure to create a custom authorization. The next procedure shows how to configure an API method to use the custom authorizer.

Configure an API Method to Use the Custom Authorizer

1. Go back to the API. Create a new method or choose an existing method. If necessary, create a new resource.

2. In **Method Execution**, choose the **Method Request** link.
3. Under **Authorization Settings**, expand the **Authorization** drop-down list to select the custom authorizer you just created (**myTestApiAuthorizer**), and then choose the checkmark icon to save the choice.



4. Optionally, while still on the **Method Request** page, choose **Add header** if you also want to pass the custom authorization header to the back end. In **Name**, type a custom header name that matches the header mapping expression you used when you created the custom authorization, and then choose the checkmark icon to save the settings.
5. Choose **Deploy API** to deploy the API to a stage. Make a note of the **Invoke URL** value. You will need it when calling the API.

Call an API Using API Gateway Custom Authorization

After you configure your API to use the custom authorizer, you or your customers can call the API using the custom authorizer. Because it involves submitting a custom authorization token header in the requests, you need a REST client that supports this. In the following examples, API calls are made using the [Postman Chrome App](#).

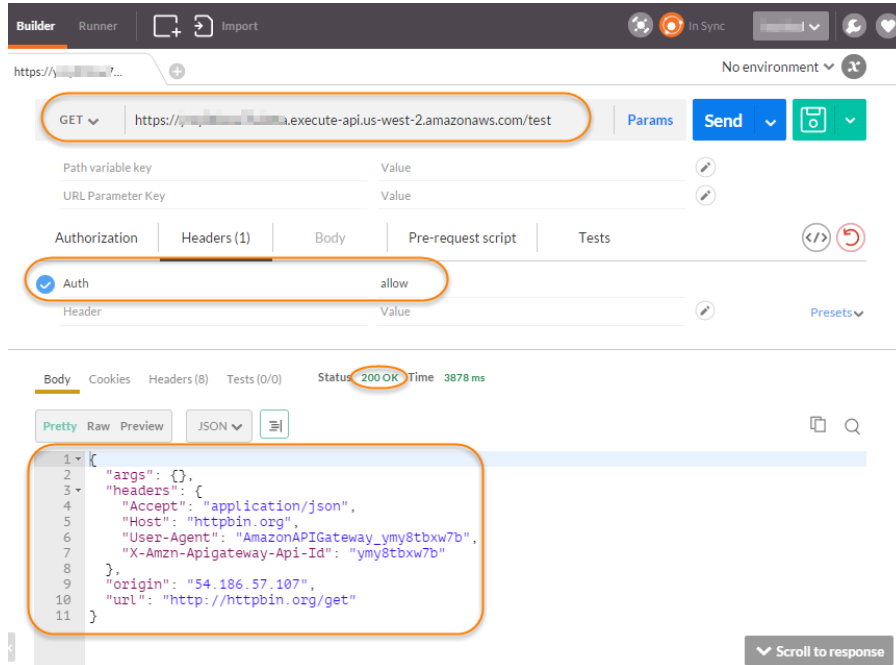
Note

When calling an authorizer-enabled method, API Gateway will not log the call to CloudWatch if the required token is not set, null or invalidated by the specified **Token validation expression**.

Calling an API with Custom Authorization Tokens

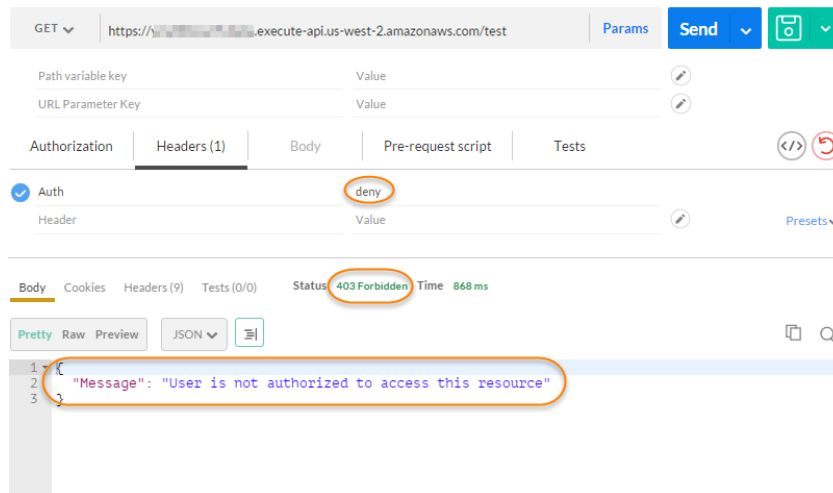
1. Open the **Postman Chrome App**, choose the **GET** method and paste the API's **Invoke URL** into the adjacent URL field.

Add the custom authorization token header and set the value to `allow`. Choose **Send**.



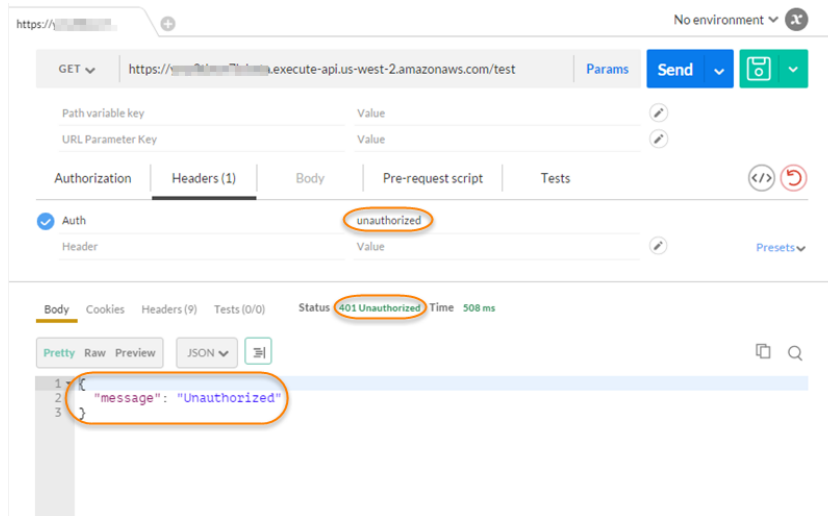
The response shows that the API Gateway custom authorizer returns a **200 OK** response and successfully authorizes the call to access the HTTP endpoint (`http://httpbin.org/get`) integrated with the method.

2. Still in Postman, change the custom authorization token header value to `deny`. Choose **Send**.



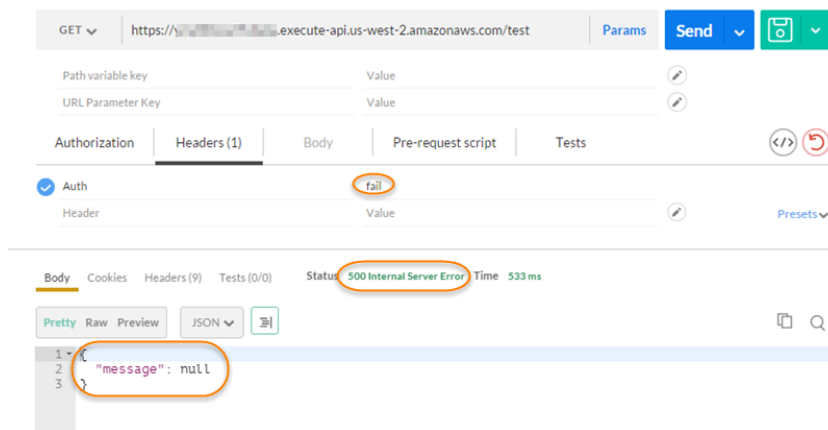
The response shows that the API Gateway custom authorizer returns a **403 Forbidden** response without authorizing the call to access the HTTP endpoint.

3. In Postman, change the custom authorization token header value to `unauthorized` and choose **Send**.



The response shows that API Gateway returns a **401 Unauthorized** response without authorizing the call to access the HTTP endpoint.

4. Now, change the custom authorization token header value to `fail`. Choose **Send**.



The response shows that API Gateway returns a **500 Internal Server Error** response without authorizing the call to access the HTTP endpoint.

Use Amazon Cognito Your User Pool

In addition to using [IAM roles and policies \(p. 160\)](#) or [custom authorizers \(p. 174\)](#), you can also use [a user pool in Amazon Cognito](#) to control who can access your API in API Gateway. A user pool serves as your own identity provider to maintain a user directory. It supports user registration and sign-in, as well as provisioning [identity tokens](#) for signed-in users.

A user pool is integrated with an API as a method authorizer. When calling the methods with such an authorizer enabled, an API client includes in the request headers the user's identity token provisioned from the user pool. API Gateway then validates the token to ensure it belongs to the configured user pool and authenticates the caller before passing the request to the back end.

To integrate an API with the Amazon Cognito identity provider, you, as an API developer, create and own a user pool, create an API Gateway authorizer connected to the user pool, and enable

the authorizer on selected API methods. You must also distribute to your API client developers the user pool ID, a client ID, and possibly the associated client secret that are provisioned from the user pool. The client will need this information to register users with the user pool, to provide the sign-in functionality, and to have the user's identity token provisioned from the user pool.

In this section, you will learn how to create a user pool, how to integrate an API Gateway API with the user pool, and how to invoke an API integrated with the user pool.

Topics

- [Create a User Pool \(p. 183\)](#)
- [Integrate an API with a User Pool \(p. 183\)](#)
- [Call an API Integrated with a User Pool \(p. 186\)](#)

Create a User Pool

Before integrating your API with a user pool, you must create the user pool in Amazon Cognito. For instructions on how to create a user pool, see [Setting up User Pools](#) in the *Amazon Cognito Developer Guide*.

Note

Make a note of the user pool ID, client ID and the client secret, if selected. The client will need to provide them to Amazon Cognito for the user to register with the user pool, to sign in to the user pool, and to get an identity token to be included in requests to call API methods configured with the user pool. Also, you will need to specify the user pool name when you configure the user pool as an authorizer in API Gateway, as described next.

Integrate an API with a User Pool

To integrate your API with a user pool, you must create in API Gateway a user pool authorizer connected to the user pool. The following procedure walks you through the steps to do this using the API Gateway console.

To create a user pool authorizer using the API Gateway console

1. Create a new API or select an existing API in API Gateway.
2. From the main navigation pane, choose **Authorizers** under the specified API.
3. Under **Authorizers**, choose **Create** and then choose **Cognito User Pool Authorizer**.
4. To configure this authorizer:
 - a. Choose a region for **Cognito region**.
 - b. For **Cognito User Pool**, choose an available user pool.
 - c. The **Authorizer name** field will be automatically populated with the chosen user pool name. However, you can customize it if you want to.
 - d. The **Identity token source** field will be set to `method.request.header.Authorization` by default. However, you can customize it if you want to. Using the default, `Authorization` will be the name of the incoming request header to contain an API caller's identity token.
 - e. Optionally, type a regular expression in the **App client ID regex** field to validate client IDs associated with the user pool.
 - f. Choose **Create** to finish integrating the user pool with the API.
5. Having created the authorizer, you can, optionally, test it by supplying an identity token provisioned from the user pool.

To enable a user pool authorizer on methods

1. Choose (or create) a method of your API.
2. Choose **Method Request**.
3. Under **Authorization Settings**, choose the edit icon by the **Authorization** field.
4. Choose one of the available **Amazon Cognito User Pool authorizers** from the drop-down list.
5. Choose the check-mark icon to save the settings.
6. Repeat these steps for other methods of your choosing.
7. If needed, choose **Integration Request** to add `context.authorizer.claims['property-name']` or `context.authorizer.claims.property-name` expressions in a body-mapping template to pass the specified identity claims property from the user pool to the back end. For simple property names, such as `sub` or `custom-sub`, the two notations are identical. For complex property names, such as `custom:role`, the dot notation may not be used. For example, the following mapping expressions pass the claim's **standard fields** of `sub` and `email` to the back end:

```
{
  "context" : {
    "sub" : "$context.authorizer.claims.sub",
    "email" : "$context.authorizer.claims.email"
  }
}
```

If you have declared a custom claim field when configuring your user pool, you can follow the same pattern to access the custom fields. The following example gets a custom `role` field of a claim:

```
{
  "context" : {
    "role" : "$context.authorizer.claims.role"
  }
}
```

If the custom claim field is declared as `custom:role`, use the following example to get the claim's property:

```
{
  "context" : {
    "role" : "$context.authorizer.claims['custom:role']"
  }
}
```

Instead of using the API Gateway console, you can also enable a Amazon Cognito user pool on a method by configuring the settings in a Swagger definition file and import the API definition into API Gateway.

To import a user pool authorizer with a Swagger definition file

1. Create (or export) a Swagger definition file for your API.
2. Add the user pool settings to the `securityDefinitions`:

```
"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. Enable the Cognito user pool (MyUserPool) as an authorizer for a method, as shown in the following GET method on the root resource.

```
"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. If needed, you can set other API configuration using appropriate Swagger definitions or extensions. For more information, see [Import and Export API](#) (p. 141).

Call an API Integrated with a User Pool

To call a method with a user pool authorizer configured, the client must do the following:

- Enable the user to sign up with the user pool.
- Enable the user to sign in to the user pool.
- Obtain an identity token of the signed-in user from the user pool.
- Include the identity token in the `Authorization` header (or another header you specified when creating the authorizer).

You can use one of the [AWS SDKs](#) to perform these tasks. For example:

- To use the Android SDK, see [Setting up the AWS Mobile SDK for Android to Work with User Pools](#).
- To use the iOS SDK, see [Setting Up the AWS Mobile SDK for iOS to Work with User Pools](#).
- To use JavaScript, see [Setting up the AWS SDK for JavaScript in the Browser to Work with User Pools](#).

The following procedure outlines the steps to perform these tasks. For more information, see the blog posts on [Using Android SDK with Amazon Cognito Your User Pools](#) and [Using Your Amazon Cognito User Pool for iOS](#).

To call an API integrated with a user pool

1. Sign up a first-time user to a specified user pool.
2. Sign in a user to the user pool.
3. Get the user's identity token.
4. Call API methods configured with a user pool authorizer, supplying the unexpired token in the `Authorization` header or another header of your choosing.
5. When the token expires, repeat Step 2-4. Identity tokens provisioned by Amazon Cognito expire within an hour.

For code examples, see an [Android Java sample](#) and an [iOS Objective-C sample](#).

Use Client-Side SSL Certificates for Authentication by the Back End

You can use API Gateway to generate an SSL certificate and use its public key in the back end to verify that HTTP requests to your back-end system are from API Gateway. This allows your HTTP back end to control and accept only requests originating from Amazon API Gateway, even if the back end is publicly accessible.

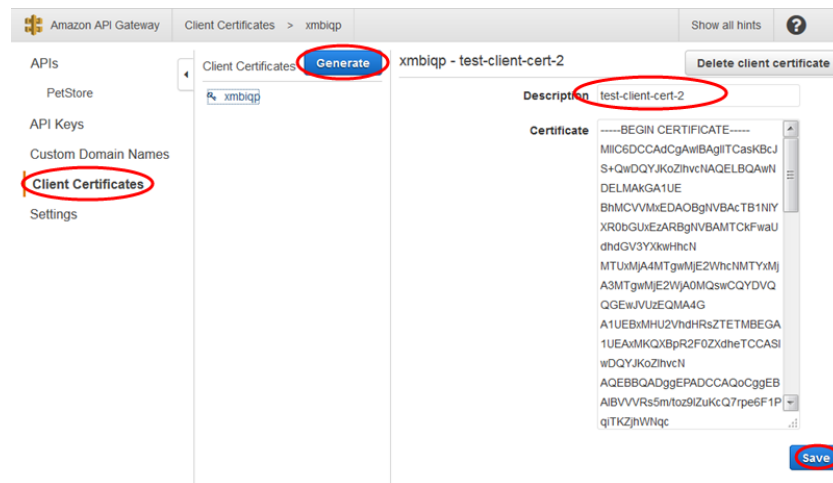
The API Gateway-generated SSL certificates are self-signed and only the public key of a certificate is visible in the API Gateway console or through the APIs.

Topics

- [Generate a Client Certificate Using the API Gateway Console \(p. 187\)](#)
- [Configure an API to Use SSL Certificates \(p. 187\)](#)
- [Test Invoke \(p. 188\)](#)
- [Configure Back End to Authenticate API \(p. 189\)](#)

Generate a Client Certificate Using the API Gateway Console

1. In the main navigation pane, choose **Client Certificates**.
2. From **Client Certificates**, choose **Generate**.
3. Optionally, For **Description**, enter a short descriptive title for the generated certificate. API Gateway generates a new certificate and returns the new certificate GUID, along with the PEM-encoded public key.
4. Choose the **Save** button to save the certificate to API Gateway.

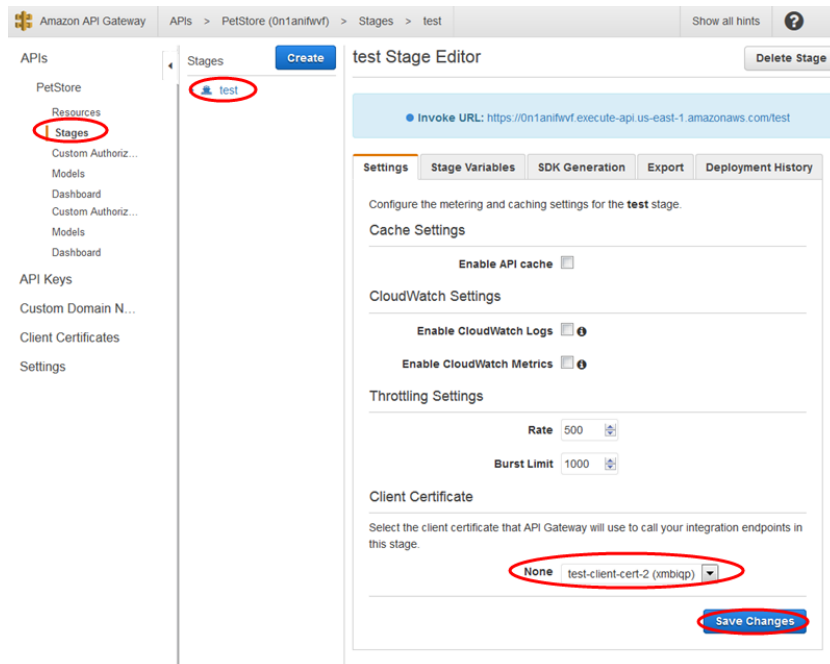


You are now ready to configure an API to use the certificate.

Configure an API to Use SSL Certificates

These instructions assume you have already completed [Generate a Client Certificate Using the API Gateway Console \(p. 187\)](#).

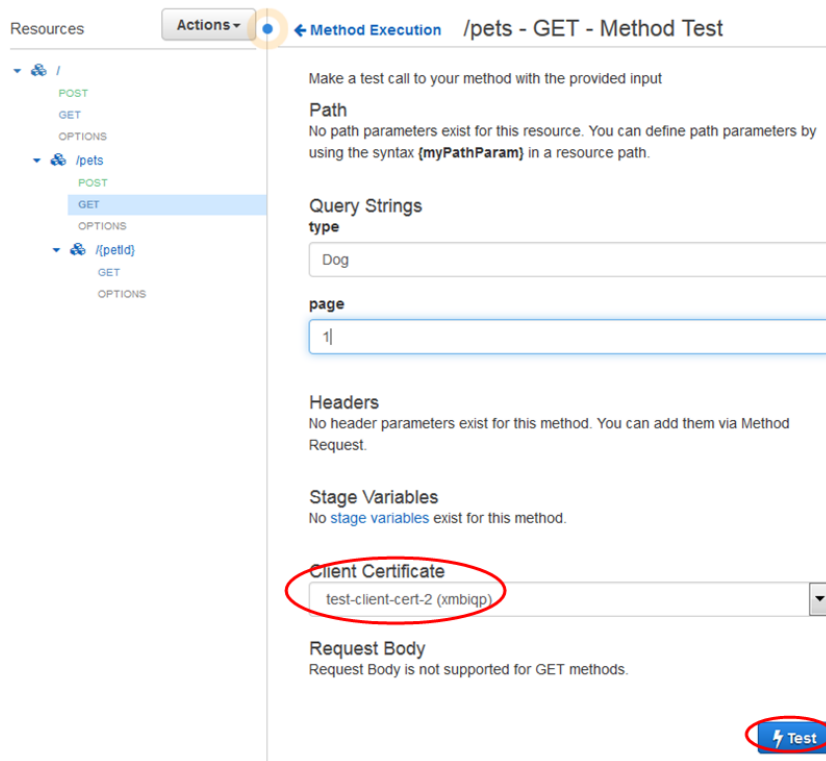
1. In the API Gateway console, create or open an API for which you want to use the client certificate. Make sure the API has been deployed to a stage.
2. Choose **Stages** under the selected API and then choose a stage.
3. In the **Stage Editor** panel, select a certificate under the **Client Certificate** section.
4. Choose the **Save Changes** button to save the settings.



After a certificate is selected for the API and saved, API Gateway will use the certificate for all calls to HTTP integrations in your API.

Test Invoke

1. Choose an API method. In **Client**, choose **Test**.
2. From **Client Certificate**, choose **Test** to invoke the method request.



API Gateway will present the chosen SSL certificate for the HTTP back end to authenticate the API.

Configure Back End to Authenticate API

These instructions assume you have already completed [Generate a Client Certificate Using the API Gateway Console](#) (p. 187) and [Configure an API to Use SSL Certificates](#) (p. 187).

When receiving HTTPS requests from API Gateway, your back end can authenticate your API using the PEM-encoded certificate generated by API Gateway, provided that the back end is properly configured. Most Web servers can be easily configured to do so.

For example, in Node.js you can use the `HTTPS` module to create an HTTPS back end and use the `client-certificate-auth` modules to authenticate client requests with PEM-encoded certificates. For more information, see [HTTPS](#) on the Nodejs.org website and see [client-certificate-auth](#) on the <https://www.npmjs.com/> website.

API Gateway-Supported Certificate Authorities for HTTP and HTTP Proxy Integrations

The following list shows the certificate authorities supported by API Gateway for HTTP and HTTP Proxy integrations.

```
Alias name: mozillacert81.pem
MD5:   D5:E9:81:40:C5:18:69:FC:46:2C:89:75:62:0F:AA:78
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:
```

Amazon API Gateway Developer Guide
Supported Certificate Authorities for
HTTP and HTTP Proxy Integration

```
Alias name: mozillacert99.pem
  MD5: 2B:70:20:56:86:82:A0:18:C8:07:53:12:28:70:21:72
  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:
Alias name: swissignplatinumg2ca
  MD5: C9:98:27:77:28:1E:3D:0E:15:3C:84:00:B8:85:03:E6
  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:
Alias name: mozillacert145.pem
  MD5: 60:84:7C:5A:CE:DB:0C:D4:CB:A7:E9:FE:02:C6:A9:C0
  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:
Alias name: mozillacert37.pem
  MD5: AB:57:A6:5B:7D:42:82:19:B5:D8:58:26:28:5E:FD:FF
  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:
Alias name: mozillacert4.pem
  MD5: 4F:EB:F1:F0:70:C2:80:63:5D:58:9F:DA:12:3C:A9:C4
  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:
Alias name: mozillacert70.pem
  MD5: 5E:80:9E:84:5A:0E:65:0B:17:02:F3:55:18:2A:3E:D7
  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:
Alias name: mozillacert88.pem
  MD5: 73:9F:4C:4B:73:5B:79:E9:FA:BA:1C:EF:6E:CB:D5:C9
  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:
Alias name: mozillacert134.pem
  MD5: FC:11:B8:D8:08:93:30:00:6D:23:F9:7E:EB:52:1E:02
  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:
Alias name: mozillacert26.pem
  MD5: DC:32:C3:A7:6D:25:57:C7:68:09:9D:EA:2D:A9:A2:D1
  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:
Alias name: buypassclass2ca
  MD5: 46:A7:D2:FE:45:FB:64:5A:A8:59:90:9B:78:44:9B:29
  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:
Alias name: chungwaepkirootca
  MD5: 1B:2E:00:CA:26:06:90:3D:AD:FE:6F:15:68:D3:6B:B3
  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:
Alias name: verisignclass2g2ca
  MD5: 2D:BB:E5:25:D3:D1:65:82:3A:B7:0E:FA:E6:EB:E2:E1
  SHA1: B3:EA:C4:47:76:C9:C8:1C:EA:F2:9D:95:B6:CC:A0:08:1B:67:EC:9D
```

Amazon API Gateway Developer Guide
Supported Certificate Authorities for
HTTP and HTTP Proxy Integration

```
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:
Alias name: mozillacert77.pem
MD5: CD:68:B6:A7:C7:C4:CE:75:E0:1D:4F:57:44:61:92:09
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:
Alias name: mozillacert123.pem
MD5: C1:62:3E:23:C5:82:73:9C:03:59:4B:2B:E9:77:49:7F
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:
Alias name: utndatacorpsgcca
MD5: B3:A5:3E:77:21:6D:AC:4A:C0:C9:FB:D5:41:3D:CA:06
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:
Alias name: mozillacert15.pem
MD5: 88:2C:8C:52:B8:A2:3C:F3:F7:BB:03:EA:AE:AC:42:0B
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:
Alias name: digicertglobalrootca
MD5: 79:E4:A9:84:0D:7D:3A:96:D7:C0:4F:E2:43:4C:89:2E
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:
Alias name: mozillacert66.pem
MD5: 3D:41:29:CB:1E:AA:11:74:CD:5D:B0:62:AF:B0:43:5B
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:
Alias name: mozillacert112.pem
MD5: 37:41:49:1B:18:56:9A:26:F5:AD:C2:66:FB:40:A5:4C
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:
Alias name: utnuserfirstclientauthemailca
MD5: D7:34:3D:EF:1D:27:09:28:E1:31:02:5B:13:2B:DD:F7
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:
Alias name: verisignc2gl.pem
MD5: B3:9C:25:B1:C3:2E:32:53:80:15:30:9D:4D:02:77:3E
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:
Alias name: mozillacert55.pem
MD5: 74:9D:EA:60:24:C4:FD:22:53:3E:CC:3A:72:D9:29:4F
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:
Alias name: mozillacert101.pem
MD5: DF:F2:80:73:CC:F1:E6:61:73:FC:F5:42:E9:C5:7C:EE
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:
Alias name: mozillacert119.pem
MD5: 94:14:77:7E:3E:5E:FD:8F:30:BD:41:B0:CF:E7:D0:30
```

Amazon API Gateway Developer Guide
Supported Certificate Authorities for
HTTP and HTTP Proxy Integration

```
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:
Alias name: verisignc3g1.pem
MD5: EF:5A:F1:33:EF:F1:CD:BB:51:02:EE:12:14:4B:96:C4
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:
Alias name: mozillacert44.pem
MD5: 72:E4:4A:87:E3:69:40:80:77:EA:BC:E3:F4:FF:F0:E1
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:
Alias name: mozillacert108.pem
MD5: 3E:45:52:15:09:51:92:E1:B7:5D:37:9F:B1:87:29:8A
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:
Alias name: mozillacert95.pem
MD5: 3D:3B:18:9E:2C:64:5A:E8:D5:88:CE:0E:F9:37:C2:EC
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:
Alias name: keynectisrootca
MD5: CC:4D:AE:FB:30:6B:D8:38:FE:50:EB:86:61:4B:D2:26
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:
Alias name: mozillacert141.pem
MD5: A9:23:75:9B:BA:49:36:6E:31:C2:DB:F2:E7:66:BA:87
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:
Alias name: equifaxsecureglobalebusinesscal
MD5: 51:F0:2A:33:F1:F5:55:39:07:F2:16:7A:47:C7:5D:63
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:
Alias name: affirmtrustpremiumca
MD5: C4:5D:0E:48:B6:AC:28:30:4E:0A:BC:F9:38:16:87:57
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:
Alias name: baltimorecodesigningca
MD5: 90:F5:28:49:56:D1:5D:2C:B0:53:D4:4B:EF:6F:90:22
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:
Alias name: mozillacert33.pem
MD5: 22:2D:A6:01:EA:7C:0A:F7:F0:6C:56:43:3F:77:76:D3
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:
Alias name: mozillacert0.pem
MD5: CA:3D:D3:68:F1:03:5C:D0:32:FA:B8:2B:59:E8:5A:DB
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:
Alias name: mozillacert84.pem
```

Amazon API Gateway Developer Guide
Supported Certificate Authorities for
HTTP and HTTP Proxy Integration

MD5: 49:63:AE:27:F4:D5:95:3D:D8:DB:24:86:B8:9C:07:53
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:
Alias name: mozillacert130.pem
MD5: 65:58:AB:15:AD:57:6C:1E:A8:A7:B5:69:AC:BF:FF:EB
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:
Alias name: mozillacert148.pem
MD5: 4C:56:41:E5:0D:BB:2B:E8:CA:A3:ED:18:08:AD:43:39
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:
Alias name: mozillacert22.pem
MD5: 02:26:C3:01:5E:08:30:37:43:A9:D0:7D:CF:37:E6:BF
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:
Alias name: verisignclgl.pem
MD5: 97:60:E8:57:5F:D3:50:47:E5:43:0C:94:36:8A:B0:62
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:
Alias name: mozillacert7.pem
MD5: 32:4A:4B:BB:C8:63:69:9B:BE:74:9A:C6:DD:1D:46:24
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:
Alias name: mozillacert73.pem
MD5: D6:39:81:C6:52:7E:96:69:FC:FC:CA:66:ED:05:F2:96
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:
Alias name: mozillacert137.pem
MD5: D3:D9:BD:AE:9F:AC:67:24:B3:C8:1B:52:E1:B9:A9:BD
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:
Alias name: swissignsilverg2ca
MD5: E0:06:A1:C9:7D:CF:C9:FC:0D:C0:56:75:96:D8:62:13
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:
Alias name: mozillacert11.pem
MD5: 87:CE:0B:7B:2A:0E:49:00:E1:58:71:9B:37:A8:93:72
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:
Alias name: mozillacert29.pem
MD5: D3:F3:A6:16:C0:FA:6B:1D:59:B1:2D:96:4D:0E:11:2E
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:
Alias name: mozillacert62.pem
MD5: EF:5A:F1:33:EF:F1:CD:BB:51:02:EE:12:14:4B:96:C4
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:

Amazon API Gateway Developer Guide
Supported Certificate Authorities for
HTTP and HTTP Proxy Integration

```
Alias name: mozillacert126.pem
  MD5: 77:0D:19:B1:21:FD:00:42:9C:3E:0C:A5:DD:0B:02:8E
  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:
Alias name: securetrustca
  MD5: DC:32:C3:A7:6D:25:57:C7:68:09:9D:EA:2D:A9:A2:D1
  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:
Alias name: soneraclasslca
  MD5: 33:B7:84:F5:5F:27:D7:68:27:DE:14:DE:12:2A:ED:6F
  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:
Alias name: mozillacert18.pem
  MD5: F1:6A:22:18:C9:CD:DF:CE:82:1D:1D:B7:78:5C:A9:A5
  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:
Alias name: mozillacert51.pem
  MD5: 18:98:C0:D6:E9:3A:FC:F9:B0:F5:0C:F7:4B:01:44:17
  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:
Alias name: mozillacert69.pem
  MD5: A6:B0:CD:85:80:DA:5C:50:34:A3:39:90:2F:55:67:73
  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:
Alias name: mozillacert115.pem
  MD5: 2B:9B:9E:E4:7B:6C:1F:00:72:1A:CC:C1:77:79:DF:6A
  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:
Alias name: verisignclass3g5ca
  MD5: CB:17:E4:31:67:3E:E2:09:FE:45:57:93:F3:0A:FA:1C
  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:
Alias name: utnuserfirsthardwareca
  MD5: 4C:56:41:E5:0D:BB:2B:E8:CA:A3:ED:18:08:AD:43:39
  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:
Alias name: addtrustqualifiedca
  MD5: 27:EC:39:47:CD:DA:5A:AF:E2:9A:01:65:21:A9:4C:BB
  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:
Alias name: mozillacert40.pem
  MD5: 56:5F:AA:80:61:12:17:F6:67:21:E6:2B:6D:61:56:8E
  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:
Alias name: mozillacert58.pem
  MD5: 01:5E:D8:6B:BD:6F:3D:8E:A1:31:F8:12:E0:98:73:6A
  SHA1: 8D:17:84:D5:37:F3:03:7D:EC:70:FE:57:8B:51:9A:99:E6:10:D7:B0
```

Amazon API Gateway Developer Guide
Supported Certificate Authorities for
HTTP and HTTP Proxy Integration

```
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:  
Alias name: verisignclass3g3ca  
MD5: CD:68:B6:A7:C7:C4:CE:75:E0:1D:4F:57:44:61:92:09  
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:  
Alias name: mozillacert104.pem  
MD5: 55:5D:63:00:97:BD:6A:97:F5:67:AB:4B:FB:6E:63:15  
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:  
Alias name: mozillacert91.pem  
MD5: 30:C9:E7:1E:6B:E6:14:EB:65:B2:16:69:20:31:67:4D  
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:  
Alias name: thawtepersonalfreemailca  
MD5: 53:4B:1D:17:58:58:1A:30:A1:90:F8:6E:5C:F2:CF:65  
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:  
Alias name: certplusclass3pprimaryca  
MD5: E1:4B:52:73:D7:1B:DB:93:30:E5:BD:E4:09:6E:BE:FB  
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:  
Alias name: verisignc3g4.pem  
MD5: 3A:52:E1:E7:FD:6F:3A:E3:6F:F3:6F:99:1B:F9:22:41  
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:  
Alias name: swissigngoldg2ca  
MD5: 24:77:D9:A8:91:D1:3B:FA:88:2D:C2:FF:F8:CD:33:93  
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:  
Alias name: mozillacert47.pem  
MD5: ED:41:F5:8C:50:C5:2B:9C:73:E6:EE:6C:EB:C2:A8:26  
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:  
Alias name: mozillacert80.pem  
MD5: 64:B0:09:55:CF:B1:D5:99:E2:BE:13:AB:A6:5D:EA:4D  
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:  
Alias name: mozillacert98.pem  
MD5: 43:5E:88:D4:7D:1A:4A:7E:FD:84:2E:52:EB:01:D4:6F  
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:  
Alias name: mozillacert144.pem  
MD5: A3:EC:75:0F:2E:88:DF:FA:48:01:4E:0B:5C:48:6F:FB  
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:  
Alias name: starfieldclass2ca  
MD5: 32:4A:4B:BB:C8:63:69:9B:BE:74:9A:C6:DD:1D:46:24
```

Amazon API Gateway Developer Guide
Supported Certificate Authorities for
HTTP and HTTP Proxy Integration

```
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:
Alias name: mozillacert36.pem
MD5: F0:96:B6:2F:C5:10:D5:67:8E:83:25:32:E8:5E:2E:E5
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:
Alias name: mozillacert3.pem
MD5: 39:16:AA:B9:6A:41:E1:14:69:DF:9E:6C:3B:72:DC:B6
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:
Alias name: globalsignr2ca
MD5: 94:14:77:7E:3E:5E:FD:8F:30:BD:41:B0:CF:E7:D0:30
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:
Alias name: mozillacert87.pem
MD5: 6C:39:7D:A4:0E:55:59:B2:3F:D6:41:B1:12:50:DE:43
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:
Alias name: mozillacert133.pem
MD5: D6:ED:3C:CA:E2:66:0F:AF:10:43:0D:77:9B:04:09:BF
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:
Alias name: mozillacert25.pem
MD5: CB:17:E4:31:67:3E:E2:09:FE:45:57:93:F3:0A:FA:1C
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:
Alias name: verisignclass1g2ca
MD5: DB:23:3D:F9:69:FA:4B:B9:95:80:44:73:5E:7D:41:83
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:
Alias name: mozillacert76.pem
MD5: 82:92:BA:5B:EF:CD:8A:6F:A6:3D:55:F9:84:F6:D6:B7
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:
Alias name: mozillacert122.pem
MD5: 1D:35:54:04:85:78:B0:3F:42:42:4D:BF:20:73:0A:3F
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:
Alias name: mozillacert14.pem
MD5: D4:74:DE:57:5C:39:B2:D3:9C:85:83:C5:C0:65:49:8A
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:
Alias name: equifaxsecureca
MD5: 67:CB:9D:C0:13:24:8A:82:9B:B2:17:1E:D1:1B:EC:D4
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:
Alias name: mozillacert65.pem
```


Amazon API Gateway Developer Guide
Supported Certificate Authorities for
HTTP and HTTP Proxy Integration

MD5: A2:6F:53:B7:EE:40:DB:4A:68:E7:FA:18:D9:10:4B:72
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:
Alias name: mozillacert111.pem
MD5: F9:03:7E:CF:E6:9E:3C:73:7A:2A:90:07:69:FF:2B:96
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:
Alias name: certumtrustednetworkca
MD5: D5:E9:81:40:C5:18:69:FC:46:2C:89:75:62:0F:AA:78
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:
Alias name: mozillacert129.pem
MD5: 92:65:58:8B:A2:1A:31:72:73:68:5C:B4:A5:7A:07:48
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:
Alias name: mozillacert54.pem
MD5: B5:E8:34:36:C9:10:44:58:48:70:6D:2E:83:D4:B8:05
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:
Alias name: mozillacert100.pem
MD5: CD:E0:25:69:8D:47:AC:9C:89:35:90:F7:FD:51:3D:2F
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:
Alias name: mozillacert118.pem
MD5: 8F:5D:77:06:27:C4:98:3C:5B:93:78:E7:D7:7D:9B:CC
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:
Alias name: gd-class2-root.pem
MD5: 91:DE:06:25:AB:DA:FD:32:17:0C:BB:25:17:2A:84:67
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:
Alias name: mozillacert151.pem
MD5: 86:38:6D:5E:49:63:6C:85:5C:DB:6D:DC:94:B7:D0:F7
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:
Alias name: thawteprimaryrootcag3
MD5: FB:1B:5D:43:8A:94:CD:44:C6:76:F2:43:4B:47:E7:31
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:
Alias name: quovadisrootca
MD5: 27:DE:36:FE:72:B7:00:03:00:9D:F4:F0:1E:6C:04:24
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:
Alias name: thawteprimaryrootcag2
MD5: 74:9D:EA:60:24:C4:FD:22:53:3E:CC:3A:72:D9:29:4F
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:

Amazon API Gateway Developer Guide
Supported Certificate Authorities for
HTTP and HTTP Proxy Integration

```
Alias name: deprecateditsecca
  MD5: A5:96:0C:F6:B5:AB:27:E5:01:C6:00:88:9E:60:33:E5
  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:
Alias name: entrustrootcag2
  MD5: 4B:E2:C9:91:96:65:0C:F4:0E:5A:93:92:A0:0A:FE:B2
  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:
Alias name: mozillacert43.pem
  MD5: 40:01:25:06:8D:21:43:6A:0E:43:00:9C:E7:43:F3:D5
  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:
Alias name: mozillacert107.pem
  MD5: BE:EC:11:93:9A:F5:69:21:BC:D7:C1:C0:67:89:CC:2A
  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:
Alias name: trustcenterclass4caii
  MD5: 9D:FB:F9:AC:ED:89:33:22:F4:28:48:83:25:23:5B:E0
  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:
Alias name: mozillacert94.pem
  MD5: 46:A7:D2:FE:45:FB:64:5A:A8:59:90:9B:78:44:9B:29
  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:
Alias name: mozillacert140.pem
  MD5: 5E:39:7B:DD:F8:BA:EC:82:E9:AC:62:BA:0C:54:00:2B
  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:
Alias name: ttelesecglobalrootclass3ca
  MD5: CA:FB:40:A8:4E:39:92:8A:1D:FE:8E:2F:C4:27:EA:EF
  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:
Alias name: amzninternalcorpca
  MD5: 7B:0E:9D:67:A9:3A:88:DD:BA:81:8D:A9:3C:74:AA:BB
  SHA1: 43:E3:E6:37:C5:88:05:67:91:37:E3:72:4D:01:7F:F4:1B:CE:3A:97
  SHA256:
01:29:04:6C:60:EF:5C:51:60:D3:9F:A2:3A:1D:0C:52:0A:AF:DA:4F:17:87:95:AA:66:82:01:9F:76:C9:
Alias name: starfieldservicesrootg2ca
  MD5: 17:35:74:AF:7B:61:1C:EB:F4:F9:3C:E2:EE:40:F9:A2
  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:
Alias name: mozillacert32.pem
  MD5: 0C:7F:DD:6A:F4:2A:B9:C8:9B:BD:20:7E:A9:DB:5C:37
  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:
Alias name: mozillacert83.pem
  MD5: 2C:8C:17:5E:B1:54:AB:93:17:B5:36:5A:DB:D1:C6:F2
  SHA1: A0:73:E5:C5:BD:43:61:0D:86:4C:21:13:0A:85:58:57:CC:9C:EA:46
```

Amazon API Gateway Developer Guide
Supported Certificate Authorities for
HTTP and HTTP Proxy Integration

```
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:
Alias name: verisignroot.pem
MD5: 8E:AD:B5:01:AA:4D:81:E4:8C:1D:D1:E1:14:00:95:19
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:
Alias name: mozillacert147.pem
MD5: B3:A5:3E:77:21:6D:AC:4A:C0:C9:FB:D5:41:3D:CA:06
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:
Alias name: camerfirmachambersca
MD5: 5E:80:9E:84:5A:0E:65:0B:17:02:F3:55:18:2A:3E:D7
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:
Alias name: mozillacert21.pem
MD5: E0:06:A1:C9:7D:CF:C9:FC:0D:C0:56:75:96:D8:62:13
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:
Alias name: mozillacert39.pem
MD5: CE:78:33:5C:59:78:01:6E:18:EA:B9:36:A0:B9:2E:23
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:
Alias name: mozillacert6.pem
MD5: 91:DE:06:25:AB:DA:FD:32:17:0C:BB:25:17:2A:84:67
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:
Alias name: verisignuniversalrootca
MD5: 8E:AD:B5:01:AA:4D:81:E4:8C:1D:D1:E1:14:00:95:19
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:
Alias name: mozillacert72.pem
MD5: 80:3A:BC:22:C1:E6:FB:8D:9B:3B:27:4A:32:1B:9A:01
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:
Alias name: geotrustuniversalca
MD5: 92:65:58:8B:A2:1A:31:72:73:68:5C:B4:A5:7A:07:48
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:
Alias name: mozillacert136.pem
MD5: 49:79:04:B0:EB:87:19:AC:47:B0:BC:11:51:9B:74:D0
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:
Alias name: mozillacert10.pem
MD5: F8:38:7C:77:88:DF:2C:16:68:2E:C2:E2:52:4B:B8:F9
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:
Alias name: mozillacert28.pem
MD5: 5C:48:DC:F7:42:72:EC:56:94:6D:1C:CC:71:35:80:75
```

Amazon API Gateway Developer Guide
Supported Certificate Authorities for
HTTP and HTTP Proxy Integration

```
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:
Alias name: affirmtrustnetworkingca
MD5: 42:65:CA:BE:01:9A:9A:4C:A9:8C:41:49:CD:C0:D5:7F
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:
Alias name: mozillacert61.pem
MD5: 42:81:A0:E2:1C:E3:55:10:DE:55:89:42:65:96:22:E6
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:
Alias name: mozillacert79.pem
MD5: C4:5D:0E:48:B6:AC:28:30:4E:0A:BC:F9:38:16:87:57
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:
Alias name: affirmtrustcommercialca
MD5: 82:92:BA:5B:EF:CD:8A:6F:A6:3D:55:F9:84:F6:D6:B7
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:
Alias name: mozillacert125.pem
MD5: D6:A5:C3:ED:5D:DD:3E:00:C1:3D:87:92:1F:1D:3F:E4
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:
Alias name: mozillacert17.pem
MD5: 21:D8:4C:82:2B:99:09:33:A2:EB:14:24:8D:8E:5F:E8
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:
Alias name: mozillacert50.pem
MD5: 2C:20:26:9D:CB:1A:4A:00:85:B5:B7:5A:AE:C2:01:37
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:
Alias name: mozillacert68.pem
MD5: 73:3A:74:7A:EC:BB:A3:96:A6:C2:E4:E2:C8:9B:C0:C3
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:
Alias name: starfieldrootg2ca
MD5: D6:39:81:C6:52:7E:96:69:FC:FC:CA:66:ED:05:F2:96
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:
Alias name: mozillacert114.pem
MD5: B8:A1:03:63:B0:BD:21:71:70:8A:6F:13:3A:BB:79:49
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:
Alias name: bypassclass3ca
MD5: 3D:3B:18:9E:2C:64:5A:E8:D5:88:CE:0E:F9:37:C2:EC
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:
Alias name: mozillacert57.pem
```

Amazon API Gateway Developer Guide
Supported Certificate Authorities for
HTTP and HTTP Proxy Integration

```
MD5: A8:0D:6F:39:78:B9:43:6D:77:42:6D:98:5A:CC:23:CA
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:
Alias name: verisignc2g3.pem
MD5: F8:BE:C4:63:22:C9:A8:46:74:8B:B8:1D:1E:4A:2B:F6
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:
Alias name: verisignclass2g3ca
MD5: F8:BE:C4:63:22:C9:A8:46:74:8B:B8:1D:1E:4A:2B:F6
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:
Alias name: mozillacert103.pem
MD5: E6:24:E9:12:01:AE:0C:DE:8E:85:C4:CE:A3:12:DD:EC
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:
Alias name: mozillacert90.pem
MD5: 69:C1:0D:4F:07:A3:1B:C3:FE:56:3D:04:BC:11:F6:A6
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:
Alias name: verisignc3g3.pem
MD5: CD:68:B6:A7:C7:C4:CE:75:E0:1D:4F:57:44:61:92:09
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:
Alias name: mozillacert46.pem
MD5: AA:8E:5D:D9:F8:DB:0A:58:B7:8D:26:87:6C:82:35:55
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:
Alias name: godaddyclass2ca
MD5: 91:DE:06:25:AB:DA:FD:32:17:0C:BB:25:17:2A:84:67
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:
Alias name: verisignc4g3.pem
MD5: DB:C8:F2:27:2E:B1:EA:6A:29:23:5D:FE:56:3E:33:DF
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:
Alias name: mozillacert97.pem
MD5: A2:33:9B:4C:74:78:73:D4:6C:E7:C1:F3:8D:CB:5C:E9
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:
Alias name: mozillacert143.pem
MD5: F1:BC:63:6A:54:E0:B5:27:F5:CD:E7:1A:E3:4D:6E:4A
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:
Alias name: mozillacert35.pem
MD5: 3F:45:96:39:E2:50:87:F7:BB:FE:98:0C:3C:20:98:E6
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:
```

Amazon API Gateway Developer Guide
Supported Certificate Authorities for
HTTP and HTTP Proxy Integration

```
Alias name: mozillacert2.pem
  MD5: 3A:52:E1:E7:FD:6F:3A:E3:6F:F3:6F:99:1B:F9:22:41
  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:
Alias name: utnuserfirstobjectca
  MD5: A7:F2:E4:16:06:41:11:50:30:6B:9C:E3:B4:9C:B0:C9
  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:
Alias name: mozillacert86.pem
  MD5: 10:FC:63:5D:F6:26:3E:0D:F3:25:BE:5F:79:CD:67:67
  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:
Alias name: mozillacert132.pem
  MD5: 14:F1:08:AD:9D:FA:64:E2:89:E7:1C:CF:A8:AD:7D:5E
  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:
Alias name: addtrustclasslca
  MD5: 1E:42:95:02:33:92:6B:B9:5F:C0:7F:DA:D6:B2:4B:FC
  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:
Alias name: mozillacert24.pem
  MD5: 7C:A5:0F:F8:5B:9A:7D:6D:30:AE:54:5A:E3:42:A2:8A
  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:
Alias name: verisignclg3.pem
  MD5: B1:47:BC:18:57:D1:18:A0:78:2D:EC:71:E8:2A:95:73
  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:
Alias name: mozillacert9.pem
  MD5: 37:85:44:53:32:45:1F:20:F0:F3:95:E1:25:C4:43:4E
  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:
Alias name: amzninternalrootca
  MD5: 08:09:73:AC:E0:78:41:7C:0A:26:33:51:E8:CF:E6:60
  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:
Alias name: mozillacert75.pem
  MD5: 67:CB:9D:C0:13:24:8A:82:9B:B2:17:1E:D1:1B:EC:D4
  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:
Alias name: entrustevca
  MD5: D6:A5:C3:ED:5D:DD:3E:00:C1:3D:87:92:1F:1D:3F:E4
  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:
Alias name: secomscrootca2
  MD5: 6C:39:7D:A4:0E:55:59:B2:3F:D6:41:B1:12:50:DE:43
  SHA1: 5F:3B:8C:F2:F8:10:B3:7D:78:B4:CE:EC:19:19:C3:73:34:B9:C7:74
```

Amazon API Gateway Developer Guide
Supported Certificate Authorities for
HTTP and HTTP Proxy Integration

```
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:
Alias name: camerfirmachambersignca
MD5: 9E:80:FF:78:01:0C:2E:C1:36:BD:FE:96:90:6E:08:F3
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:
Alias name: secomscreenshotcal
MD5: F1:BC:63:6A:54:E0:B5:27:F5:CD:E7:1A:E3:4D:6E:4A
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:
Alias name: mozillacert121.pem
MD5: 1E:42:95:02:33:92:6B:B9:5F:C0:7F:DA:D6:B2:4B:FC
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:
Alias name: mozillacert139.pem
MD5: 27:DE:36:FE:72:B7:00:03:00:9D:F4:F0:1E:6C:04:24
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:
Alias name: mozillacert13.pem
MD5: C5:A1:B7:FF:73:DD:D6:D7:34:32:18:DF:FC:3C:AD:88
SHA1: 06:08:3F:59:3F:15:A1:04:A0:69:A4:6B:A9:03:D0:06:E7:97:09:91
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:
Alias name: mozillacert64.pem
MD5: 06:9F:69:79:16:66:90:02:1B:8C:8C:A2:C3:07:6F:3A
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:
Alias name: mozillacert110.pem
MD5: D0:A0:5A:EE:05:B6:09:94:21:A1:7D:F1:B2:29:82:02
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:
Alias name: mozillacert128.pem
MD5: 0E:40:A7:6C:DE:03:5D:8F:D1:0F:E4:D1:8D:F9:6C:A9
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:
Alias name: entrust2048ca
MD5: EE:29:31:BC:32:7E:9A:E6:E8:B5:F7:51:B4:34:71:90
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:
Alias name: mozillacert53.pem
MD5: 7E:23:4E:5B:A7:A5:B4:25:E9:00:07:74:11:62:AE:D6
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:
Alias name: mozillacert117.pem
MD5: AC:B6:94:A5:9C:17:E0:D7:91:52:9B:B1:97:06:A6:E4
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:
Alias name: mozillacert150.pem
MD5: C5:E6:7B:BF:06:D0:4F:43:ED:C4:7A:65:8A:FB:6B:19
```

Amazon API Gateway Developer Guide
Supported Certificate Authorities for
HTTP and HTTP Proxy Integration

```
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:
Alias name: thawteserverca
MD5: EE:FE:61:69:65:6E:F8:9C:C6:2A:F4:D7:2B:63:EF:A2
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:
Alias name: secomvalicertclass1ca
MD5: 65:58:AB:15:AD:57:6C:1E:A8:A7:B5:69:AC:BF:FF:EB
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:
Alias name: mozillacert42.pem
MD5: 74:01:4A:91:B1:08:C4:58:CE:47:CD:F0:DD:11:53:08
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:
Alias name: verisignc2g6.pem
MD5: 7D:0B:83:E5:FB:7C:AD:07:4F:20:A9:B5:DF:63:ED:79
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:
Alias name: godaddyrootg2ca
MD5: 80:3A:BC:22:C1:E6:FB:8D:9B:3B:27:4A:32:1B:9A:01
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:
Alias name: gtocybertrustglobalca
MD5: CA:3D:D3:68:F1:03:5C:D0:32:FA:B8:2B:59:E8:5A:DB
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:
Alias name: mozillacert106.pem
MD5: 7B:30:34:9F:DD:0A:4B:6B:35:CA:31:51:28:5D:AE:EC
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:
Alias name: equifaxsecureebusinesscal
MD5: 14:C0:08:E5:A3:85:03:A3:BE:78:E9:67:4F:27:CA:EE
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:
Alias name: mozillacert93.pem
MD5: 78:4B:FB:9E:64:82:0A:D3:B8:4C:62:F3:64:F2:90:64
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:
Alias name: quovadisrootca3
MD5: 31:85:3C:62:94:97:63:B9:AA:FD:89:4E:AF:6F:E0:CF
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:
Alias name: quovadisrootca2
MD5: 5E:39:7B:DD:F8:BA:EC:82:E9:AC:62:BA:0C:54:00:2B
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:
Alias name: soneraclass2ca
```


Amazon API Gateway Developer Guide
Supported Certificate Authorities for
HTTP and HTTP Proxy Integration

MD5: A3:EC:75:0F:2E:88:DF:FA:48:01:4E:0B:5C:48:6F:FB
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:
Alias name: mozillacert31.pem
MD5: 7C:62:FF:74:9D:31:53:5E:68:4A:D5:78:AA:1E:BF:23
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:
Alias name: mozillacert49.pem
MD5: DF:3C:73:59:81:E7:39:50:81:04:4C:34:A2:CB:B3:7B
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:
Alias name: mozillacert82.pem
MD5: 7F:30:78:8C:03:E3:CA:C9:0A:E2:C9:EA:1E:AA:55:1A
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:
Alias name: mozillacert146.pem
MD5: 91:F4:03:55:20:A1:F8:63:2C:62:DE:AC:FB:61:1C:8E
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:
Alias name: baltimorecybertrustca
MD5: AC:B6:94:A5:9C:17:E0:D7:91:52:9B:B1:97:06:A6:E4
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:
Alias name: mozillacert20.pem
MD5: 24:77:D9:A8:91:D1:3B:FA:88:2D:C2:FF:F8:CD:33:93
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:
Alias name: mozillacert38.pem
MD5: 93:2A:3E:F6:FD:23:69:0D:71:20:D4:2B:47:99:2B:A6
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:
Alias name: mozillacert5.pem
MD5: A1:0B:44:B3:CA:10:D8:00:6E:9D:0F:D8:0F:92:0A:D1
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:
Alias name: mozillacert71.pem
MD5: 9E:80:FF:78:01:0C:2E:C1:36:BD:FE:96:90:6E:08:F3
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:
Alias name: verisignclass3g4ca
MD5: 3A:52:E1:E7:FD:6F:3A:E3:6F:F3:6F:99:1B:F9:22:41
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:
Alias name: mozillacert89.pem
MD5: DB:C8:F2:27:2E:B1:EA:6A:29:23:5D:FE:56:3E:33:DF
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:

Amazon API Gateway Developer Guide
Supported Certificate Authorities for
HTTP and HTTP Proxy Integration

```
Alias name: mozillacert135.pem
  MD5: 2C:8F:9F:66:1D:18:90:B1:47:26:9D:8E:86:82:8C:A9
  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:
Alias name: camerfirmachamberscommerceca
  MD5: B0:01:EE:14:D9:AF:29:18:94:76:8E:F1:69:33:2A:84
  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:
Alias name: mozillacert27.pem
  MD5: CF:F4:27:0D:D4:ED:DC:65:16:49:6D:3D:DA:BF:6E:DE
  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:
Alias name: verisignclg6.pem
  MD5: 2F:A8:B4:DA:F6:64:4B:1E:82:F9:46:3D:54:1A:7C:B0
  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:
Alias name: verisignclass3g2ca
  MD5: A2:33:9B:4C:74:78:73:D4:6C:E7:C1:F3:8D:CB:5C:E9
  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:
Alias name: mozillacert60.pem
  MD5: B7:52:74:E2:92:B4:80:93:F2:75:E4:CC:D7:F2:EA:26
  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:
Alias name: mozillacert78.pem
  MD5: 42:65:CA:BE:01:9A:4C:A9:8C:41:49:CD:C0:D5:7F
  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:
Alias name: gd_bundle-g2.pem
  MD5: 96:C2:50:31:BC:0D:C3:5C:FB:A7:23:73:1E:1B:41:40
  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:
Alias name: certumca
  MD5: 2C:8F:9F:66:1D:18:90:B1:47:26:9D:8E:86:82:8C:A9
  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:
Alias name: deutschetelekomrootca2
  MD5: 74:01:4A:91:B1:08:C4:58:CE:47:CD:F0:DD:11:53:08
  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:
Alias name: mozillacert124.pem
  MD5: 27:EC:39:47:CD:DA:5A:AF:E2:9A:01:65:21:A9:4C:BB
  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:
Alias name: mozillacert16.pem
  MD5: 41:03:52:DC:0F:F7:50:1B:16:F0:02:8E:BA:6F:45:C5
  SHA1: DA:C9:02:4F:54:D8:F6:DF:94:93:5F:B1:73:26:38:CA:6A:D7:7C:13
```

Amazon API Gateway Developer Guide
Supported Certificate Authorities for
HTTP and HTTP Proxy Integration

```
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:
Alias name: secomevrootca1
MD5: 22:2D:A6:01:EA:7C:0A:F7:F0:6C:56:43:3F:77:76:D3
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:
Alias name: mozillacert67.pem
MD5: C5:DF:B8:49:CA:05:13:55:EE:2D:BA:1A:C3:3E:B0:28
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:
Alias name: globalsignr3ca
MD5: C5:DF:B8:49:CA:05:13:55:EE:2D:BA:1A:C3:3E:B0:28
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:
Alias name: mozillacert113.pem
MD5: EE:29:31:BC:32:7E:9A:E6:E8:B5:F7:51:B4:34:71:90
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:
Alias name: gdroot-g2.pem
MD5: 80:3A:BC:22:C1:E6:FB:8D:9B:3B:27:4A:32:1B:9A:01
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:
Alias name: aolrootca2
MD5: D6:ED:3C:CA:E2:66:0F:AF:10:43:0D:77:9B:04:09:BF
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:
Alias name: trustcenteruniversalcai
MD5: 45:E1:A5:72:C5:A9:36:64:40:9E:F5:E4:58:84:67:8C
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:
Alias name: aolrootca1
MD5: 14:F1:08:AD:9D:FA:64:E2:89:E7:1C:CF:A8:AD:7D:5E
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:
Alias name: verisignc2g2.pem
MD5: 2D:BB:E5:25:D3:D1:65:82:3A:B7:0E:FA:E6:EB:E2:E1
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:
Alias name: mozillacert56.pem
MD5: FB:1B:5D:43:8A:94:CD:44:C6:76:F2:43:4B:47:E7:31
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:
Alias name: verisignclasslg3ca
MD5: B1:47:BC:18:57:D1:18:A0:78:2D:EC:71:E8:2A:95:73
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:
Alias name: mozillacert102.pem
MD5: AA:C6:43:2C:5E:2D:CD:C4:34:C0:50:4F:11:02:4F:B6
```

Amazon API Gateway Developer Guide
Supported Certificate Authorities for
HTTP and HTTP Proxy Integration

```
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:
Alias name: addtrustexternalca
MD5: 1D:35:54:04:85:78:B0:3F:42:42:4D:BF:20:73:0A:3F
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:
Alias name: verisignc3g2.pem
MD5: A2:33:9B:4C:74:78:73:D4:6C:E7:C1:F3:8D:CB:5C:E9
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:
Alias name: verisignclass3ca
MD5: EF:5A:F1:33:EF:F1:CD:BB:51:02:EE:12:14:4B:96:C4
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:
Alias name: mozillacert45.pem
MD5: 1B:2E:00:CA:26:06:90:3D:AD:FE:6F:15:68:D3:6B:B3
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:
Alias name: verisignc4g2.pem
MD5: 26:6D:2C:19:98:B6:70:68:38:50:54:19:EC:90:34:60
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:
Alias name: digicertassuredidrootca
MD5: 87:CE:0B:7B:2A:0E:49:00:E1:58:71:9B:37:A8:93:72
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:
Alias name: verisignclass1ca
MD5: 86:AC:DE:2B:C5:6D:C3:D9:8C:28:88:D3:8D:16:13:1E
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:
Alias name: mozillacert109.pem
MD5: 26:01:FB:D8:27:A7:17:9A:45:54:38:1A:43:01:3B:03
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:
Alias name: thawtepremiumserverca
MD5: A6:6B:60:90:23:9B:3F:2D:BB:98:6F:D6:A7:19:0D:46
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:
Alias name: verisigntsaca
MD5: F2:89:95:6E:4D:05:F0:F1:A7:21:55:7D:46:11:BA:47
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:
Alias name: mozillacert96.pem
MD5: CA:FB:40:A8:4E:39:92:8A:1D:FE:8E:2F:C4:27:EA:EF
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:
Alias name: mozillacert142.pem
```

Amazon API Gateway Developer Guide
Supported Certificate Authorities for
HTTP and HTTP Proxy Integration

MD5: 31:85:3C:62:94:97:63:B9:AA:FD:89:4E:AF:6F:E0:CF
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:
Alias name: thawteprimaryrootca
MD5: 8C:CA:DC:0B:22:CE:F5:BE:72:AC:41:1A:11:A8:D8:12
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:
Alias name: mozillacert34.pem
MD5: BC:6C:51:33:A7:E9:D3:66:63:54:15:72:1B:21:92:93
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:E7:AD:57:80:81:58:2E:02:07:97:A6:CB:DF:4F:FF:78:CE:83:96:B3:89:37:
Alias name: mozillacert1.pem
MD5: C5:70:C4:A2:ED:53:78:0C:C8:10:53:81:64:CB:D0:1D
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:
Alias name: xrampglobalca
MD5: A1:0B:44:B3:CA:10:D8:00:6E:9D:0F:D8:0F:92:0A:D1
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:
Alias name: mozillacert85.pem
MD5: AA:08:8F:F6:F9:7B:B7:F2:B1:A7:1E:9B:EA:EA:BD:79
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:
Alias name: valicertclass2ca
MD5: A9:23:75:9B:BA:49:36:6E:31:C2:DB:F2:E7:66:BA:87
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:
Alias name: mozillacert131.pem
MD5: 34:FC:B8:D0:36:DB:9E:14:B3:C2:F2:DB:8F:E4:94:C7
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:
Alias name: mozillacert149.pem
MD5: B0:01:EE:14:D9:AF:29:18:94:76:8E:F1:69:33:2A:84
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:
Alias name: geotrustprimaryca
MD5: 02:26:C3:01:5E:08:30:37:43:A9:D0:7D:CF:37:E6:BF
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:
Alias name: mozillacert23.pem
MD5: 8C:CA:DC:0B:22:CE:F5:BE:72:AC:41:1A:11:A8:D8:12
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:
Alias name: verisignc1g2.pem
MD5: DB:23:3D:F9:69:FA:4B:B9:95:80:44:73:5E:7D:41:83
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:

Amazon API Gateway Developer Guide
Supported Certificate Authorities for
HTTP and HTTP Proxy Integration

```
Alias name: mozillacert8.pem
  MD5: 22:4D:8F:8A:FC:F7:35:C2:BB:57:34:90:7B:8B:22:16
  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:
Alias name: mozillacert74.pem
  MD5: 17:35:74:AF:7B:61:1C:EB:F4:F9:3C:E2:EE:40:F9:A2
  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:
Alias name: mozillacert120.pem
  MD5: 64:9C:EF:2E:44:FC:C6:8F:52:07:D0:51:73:8F:CB:3D
  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:
Alias name: geotrustglobalca
  MD5: F7:75:AB:29:FB:51:4E:B7:77:5E:FF:05:3C:99:8E:F5
  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:
Alias name: mozillacert138.pem
  MD5: 91:1B:3F:6E:CD:9E:AB:EE:07:FE:1F:71:D2:B3:61:27
  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:
Alias name: mozillacert12.pem
  MD5: 79:E4:A9:84:0D:7D:3A:96:D7:C0:4F:E2:43:4C:89:2E
  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:
Alias name: comodoaaaaca
  MD5: 49:79:04:B0:EB:87:19:AC:47:B0:BC:11:51:9B:74:D0
  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:
Alias name: mozillacert63.pem
  MD5: F8:49:F4:03:BC:44:2D:83:BE:48:69:7D:29:64:FC:B1
  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:
Alias name: certplusclass2primaryca
  MD5: 88:2C:8C:52:B8:A2:3C:F3:F7:BB:03:EA:AE:AC:42:0B
  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:
Alias name: mozillacert127.pem
  MD5: F7:75:AB:29:FB:51:4E:B7:77:5E:FF:05:3C:99:8E:F5
  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:
Alias name: ttelesecglobalrootclass2ca
  MD5: 2B:9B:9E:E4:7B:6C:1F:00:72:1A:CC:C1:77:79:DF:6A
  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:
Alias name: mozillacert19.pem
  MD5: 37:A5:6E:D4:B1:25:84:97:B7:FD:56:15:7A:F9:A2:00
  SHA1: B4:35:D4:E1:11:9D:1C:66:90:A7:49:EB:B3:94:BD:63:7B:A7:82:B7
```

Amazon API Gateway Developer Guide
Supported Certificate Authorities for
HTTP and HTTP Proxy Integration

```
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:
Alias name: digicerthighassuranceevrootca
MD5: D4:74:DE:57:5C:39:B2:D3:9C:85:83:C5:C0:65:49:8A
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:
Alias name: amzninternalinfosecag3
MD5: E9:34:94:02:BA:BB:31:6B:22:E6:2B:A9:C4:F0:26:04
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:
Alias name: mozillacert52.pem
MD5: 21:BC:82:AB:49:C4:13:3B:4B:B2:2B:5C:6B:90:9C:19
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:
Alias name: mozillacert116.pem
MD5: AE:B9:C4:32:4B:AC:7F:5D:66:CC:77:94:BB:2A:77:56
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:
Alias name: globalsignca
MD5: 3E:45:52:15:09:51:92:E1:B7:5D:37:9F:B1:87:29:8A
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:
Alias name: mozillacert41.pem
MD5: 45:E1:A5:72:C5:A9:36:64:40:9E:F5:E4:58:84:67:8C
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:
Alias name: mozillacert59.pem
MD5: 8E:AD:B5:01:AA:4D:81:E4:8C:1D:D1:E1:14:00:95:19
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:
Alias name: mozillacert105.pem
MD5: 5B:04:69:EC:A5:83:94:63:18:A7:86:D0:E4:F2:6E:19
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:
Alias name: trustcenterclass2caii
MD5: CE:78:33:5C:59:78:01:6E:18:EA:B9:36:A0:B9:2E:23
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:
Alias name: mozillacert92.pem
MD5: C9:3B:0D:84:41:FC:A4:76:79:23:08:57:DE:10:19:16
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:
Alias name: verisignc3g5.pem
MD5: CB:17:E4:31:67:3E:E2:09:FE:45:57:93:F3:0A:FA:1C
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:
Alias name: geotrustprimarycag3
MD5: B5:E8:34:36:C9:10:44:58:48:70:6D:2E:83:D4:B8:05
```

```
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:
Alias name: geotrustprimarycag2
MD5: 01:5E:D8:6B:BD:6F:3D:8E:A1:31:F8:12:E0:98:73:6A
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:
Alias name: mozillacert30.pem
MD5: 15:AC:A5:C2:92:2D:79:BC:E8:7F:CB:67:ED:02:CF:36
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:
Alias name: affirmtrustpremiumeccca
MD5: 64:B0:09:55:CF:B1:D5:99:E2:BE:13:AB:A6:5D:EA:4D
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:
Alias name: mozillacert48.pem
MD5: B8:08:9A:F0:03:CC:1B:0D:C8:6C:0B:76:A1:75:64:23
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:
```

Create and Use API Gateway Usage Plans

After you create, test, and deploy your APIs, you can extend them as product offerings for your customers. You can provide usage plans to allow specified customers to access selected APIs at agreed-upon request rates and quotas that can meet their business requirements and budgetary constraints.

What Is a Usage Plan?

A usage plan provides access to one or more deployed API stages with configurable throttling and quota limits enforced on individual client API keys. API callers are identified by API keys that can be generated by API Gateway or imported from external sources. The throttling prescribes the request rate limits applied to each API key. The quotas are the maximum number of requests with a given API key submitted within a specified time interval. Individual API methods can be configured to require API key authorization based on usage plan configuration. An API stage is identified by an API identifier and a stage name.

Note

Throttling and quota limits apply to requests for individual API keys that are aggregated across all API stages within a usage plan.

How to Configure a Usage Plan?

The following steps outlines how you, as the API owner, configure a usage plan for your customers.

To configure a usage plan

1. Create one or more APIs, configure the methods to require an API key, and deploy the APIs in stages.
2. Generate API keys and distribute the keys to app developers (your customers) using your APIs.

3. Create the usage plan with the desired throttle and quota limits.
4. Associate selected API stages and API keys to the usage plan.

Callers of the API must supply an assigned API key in the `x-api-key` header in requests to the API.

Note

To enforce authorization of the API key in requests to the API, individual API methods must be configured to [require an API key \(p. 213\)](#). Setting this configuration ensures the incoming API key will be authorized according to the usage plan configuration.

The following sections provide detailed instructions for these tasks.

Topics

- [Set Up API Keys Using the API Gateway Console \(p. 213\)](#)
- [Create and Configure Usage Plans with the API Gateway Console \(p. 216\)](#)
- [Set Up API Keys Using the API Gateway REST API \(p. 219\)](#)
- [Create and Configure Usage Plans with the API Gateway REST API \(p. 220\)](#)
- [API Gateway API Key File Format \(p. 222\)](#)
- [Use API Keys in API Gateway without Usage Plans Enabled \(p. 223\)](#)

Set Up API Keys Using the API Gateway Console

To set up API keys, do the following:

- Configure API methods to require an API key.
- Create or import an API key for the API in a region.

Before setting up API keys, you must have created an API and deployed it to a stage.

For instructions on how to create and deploy an API using the API Gateway console, see [Creating an API \(p. 70\)](#) and [Deploying an API \(p. 230\)](#), respectively.

Topics

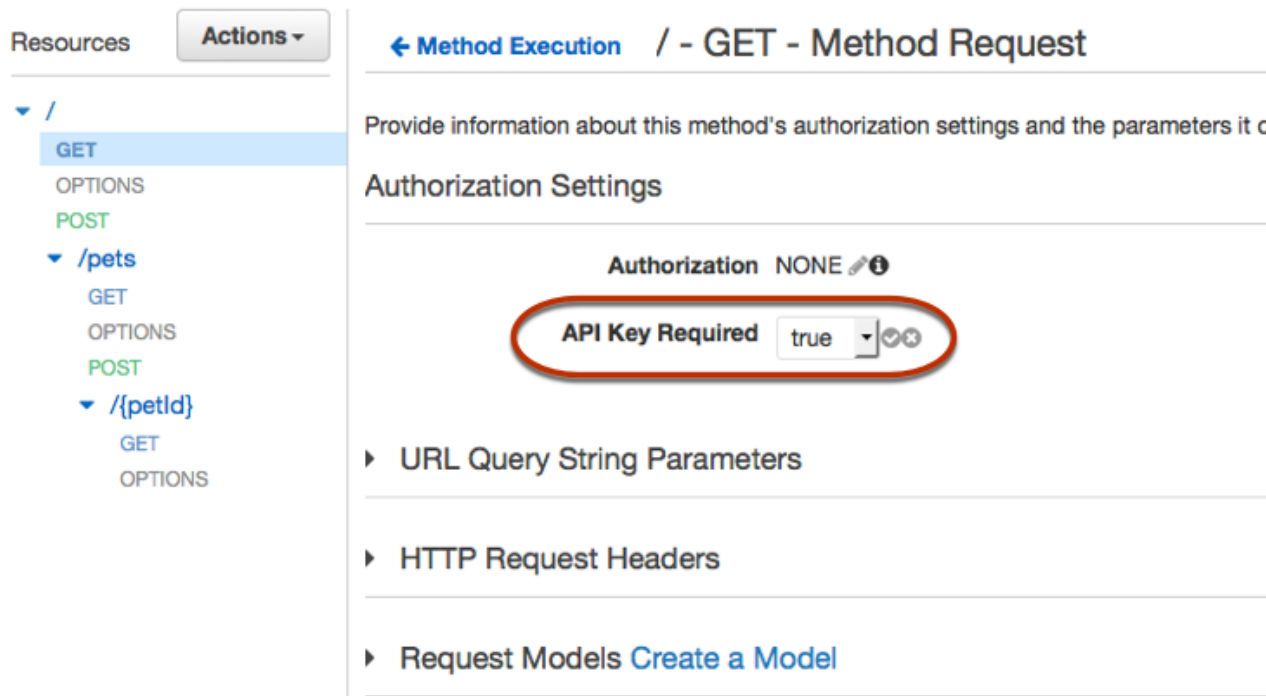
- [Require API Key on a Method \(p. 213\)](#)
- [Create an API Key \(p. 214\)](#)
- [Import API Keys \(p. 215\)](#)

Require API Key on a Method

The following procedure describes how to configure an API method to require an API key.

To configure an API method to require an API key

1. Sign in to the AWS Management Console and open the API Gateway console at <https://console.aws.amazon.com/apigateway/>.
2. In the API Gateway main navigation pane, choose **Resources**.
3. Under **Resources**, create a new method or choose an existing one.
4. Choose **Method Request**.
5. Under the **Authorization Settings** section, choose `true` for **API Key Required**.
6. Select the check-mark icon to save the settings.



7. Deploy or redeploy the API for the requirement to take effect.

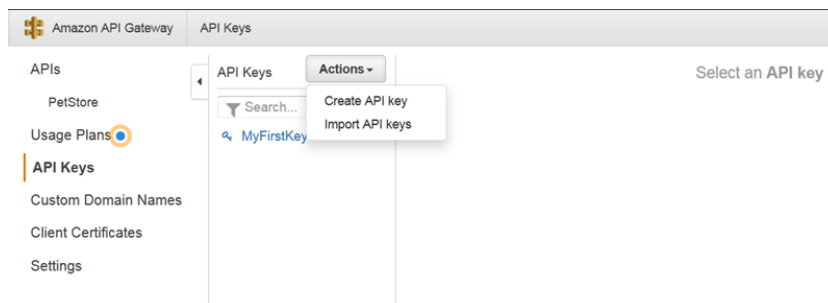
If the **API Key Required** option is set to `false` and you do not execute the steps above, any API key associated with an API stage will not be used for the method.

Create an API Key

If you have already created or imported API keys for use with usage plans, you can skip this and the next procedure.

To create an API key

1. Sign in to the AWS Management Console and open the API Gateway console at <https://console.aws.amazon.com/apigateway/>.
2. In the API Gateway main navigation pane, choose **API Keys**.
3. From the **Actions** drop-down menu, choose **Create API key**.



4. In **Create API Key**, do the following:
 - a. Type an API key name (e.g., `MyFirstKey`) in the **Name** input field.

- b. Choose **Auto Generate** to have API Gateway to generate the key value or choose **Custom** to enter the key manually.
- c. Choose **Save**.

Create API Key

Name*

API key* Auto Generate Custom

Description

* Required

5. Repeat the preceding steps to create more API keys, if needed.

Import API Keys

The following procedure describes how to import API keys to use with usage plans.

To import API keys

1. In the main navigation pane, choose **API Keys**.
2. From the **Actions** drop-down menu, choose **Import API keys**.
3. To load a comma-separated key file, choose **Select CSV File**. You can also type the keys manually. For information about the file format, see [API Gateway API Key File Format \(p. 222\)](#).

Import API Keys

Use the field below to upload your existing API Keys as comma separated values (CSV). API Keys will be created in API Gateway and associated with a Usage Plan. Learn about the CSV format in the [API Gateway documentation](#).

1	name,Key,Description,enabled,usageplanIds
2	ImportedKey,CWaiyZjNC212f9P7hcxG17Ae803jEdFu8pzfryqf,an imported key,true,abcdef

Fail on warnings Ignore warnings

4. Choose **Fail on warnings** to stop import when there is an error, or choose **Ignore warnings** to continue to import valid key entries when there is an error.
5. To start importing the selected API keys, choose **Import**.

Now that you have set up the API key, you can proceed to [create and use a usage plan \(p. 216\)](#).

Create and Configure Usage Plans with the API Gateway Console

Before creating a usage plan, make sure you have set up the desired API keys. For more information, see [Set Up API Keys Using the API Gateway Console \(p. 213\)](#).

This section describes steps needed to create and use a usage plan using the API Gateway console.

Topics

- [Migrate to Default Usage Plans \(p. 216\)](#)
- [Create Usage Plans \(p. 216\)](#)
- [Test a Usage Plan \(p. 218\)](#)
- [Maintain Plan Usage \(p. 218\)](#)

Migrate to Default Usage Plans

If you start to use API Gateway after the Usage Plans feature was rolled out on August 11, 2016, you will automatically have usage plans enabled for you in all supported regions.

If you started to use API Gateway before then, you will be prompted with the **Enable Usage Plans** option before using **Usage Plans** for the first time in the selected region. By enabling this option, you will have default usage plans created for every unique API stage associated with existing API keys. In the default usage plan, no throttle and quota limits are set initially, existing API keys are converted to a collection of [UsagePlanKey](#) resources, and existing API keys are converted to API stage Ids. The API will behave the same as before. However, you must use the [UsagePlan](#) `apiStages` property to associate specified API stage values (`apiId` and `stage`) with included API keys (via [UsagePlanKey](#)), instead of using the [ApiKey](#) `stageKeys` property.

If you choose to ignore the **Enable Usage Plans** option, you can continue to create and use API keys as described in [Use API Keys without Usage Plans Enabled \(p. 223\)](#).

Create Usage Plans

The following procedure describes how to create a usage plan.

To create a usage plan

1. In the Amazon API Gateway main navigation pane, choose **Usage Plans**, and then choose **Create**.
2. Under **Create Usage Plan**, do the following:
 - a. For **Name**, type a name for your plan (e.g., `Plan_A`).
 - b. For **Description**, type a description for your plan.
 - c. Select **Enable throttling** and set **Rate** (e.g., 100) and **Burst** (e.g., 200).
 - d. Choose **Enable quota** and set its limit (e.g., 5000) for a selected time interval (e.g., **Month**).
 - e. Choose **Save**.

Amazon API Gateway Developer Guide

Create and Configure Usage Plans with the API Gateway Console

Amazon API Gateway Usage Plans > Create Show all hints ?

Usage Plans Create

Create Usage Plan

Usage Plans help you meter API usage. With Usage Plans, you can enforce a throttling and quota limit on each API key. Throttling limits define the maximum number of requests per second available to each key. Quota limits define the number of requests each API key is allowed to make over a period.

Name* Plan_A

Description First usage plan

Throttling

Enable throttling ?

Rate 100 requests per second ?

Burst 200 requests ?

Quota

Enable quota ?

5000 requests per Month ?

* Required Save

3. To add a stage to the plan, do the following in the **Associated API Stages** pane:
 - a. Choose **Add API Stage**.
 - b. Choose an API (e.g., `PetStore`) from the **API** drop-down list.
 - c. Choose a stage (e.g., `stage_1`) from the **Stage** drop-down list.
 - d. Choose the check-mark icon to save.
 - e. Choose **Next**.

Associated API Stages

Associate API stages to this usage plan. Subscribers will only be allowed to access the API stages that are associated with the plan. Choose "Add Stage" below, then use the dropdown to select an API and stage to enable for this usage plan.

Add API Stage

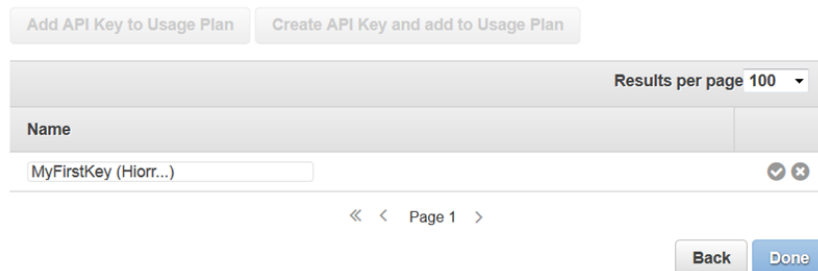
API	Stage	
PetStore	testStage	<input checked="" type="checkbox"/>

Back Next

4. To add a key to the plan, do the following in the **Usage Plan API Keys** pane:
 - a. To use an existing key, choose **Add API Key to Usage Plan**.
 - b. For **Name**, type a name for the key you want to add (e.g., `MyFirstKey`).
 - c. Choose the check-mark icon to save.
 - d. If desired, repeat the preceding steps to add other existing API keys to this usage plan.

Usage Plan API Keys

Subscribe an API key to this usage plan. Choose "Add API Key" below to search through your existing API keys. Once a key is associated with a plan, API Gateway will meter all requests from the key and apply the plan's throttling and quota limits.



Note

To add a new API key to the usage plan, choose **Create API Key and add to Usage Plan** and follow the instructions.

5. To finish creating the usage plan, choose **Done**.
6. If you want to add more API stages to the usage plan, choose **Add API Stage** to repeat the preceding steps.

Test a Usage Plan

To test the usage plan, you can use an AWS SDK, AWS CLI, or a REST API client like Postman. For an example of using [Postman](#) to test the usage plan, see [Test Usage Plans \(p. 221\)](#)

Maintain Plan Usage

Maintaining a usage plan involves monitoring the used and remaining quotas over a given time period and extending the remaining quotas by a specified amount. The following procedures describe how to monitor and extend quotas.

To monitor used and remaining quotas

1. In the API Gateway main navigation pane, choose **Usage Plans**.
2. Choose a usage plan from the list of the usage plans in the secondary navigation pane in the middle.
3. From within the specified plan, choose **API Keys**.
4. Choose an API key. Then choose **Usage** to view **Subscriber's Traffic** from the plan you are monitoring.
5. Optionally, choose **Export**, choose a **From** date and a **To** date, choose `JSON` or `CSV` for the exported data format, and then choose **Export**.

The following example shows an exported file.

```
{
  "thisPeriod": {
    "px1KW6...qBazOJH": [
      [
```

```
    0,  
    5000  
  ],  
  [  
    0,  
    5000  
  ],  
  [  
    0,  
    10  
  ]  
]  
},  
"startDate": "2016-08-01",  
"endDate": "2016-08-03"  
}
```

The usage data in the example shows the daily usage data for an API client, as identified by the API key (px1KW6...qBazOJH), between August 1, 2016, and August 3, 2016. Each daily usage data shows used and remaining quotas. In this example, the subscriber has not yet used any allotted quotas and the API owner or administrator has reduced the remaining quota from 5000 to 10 on the third day.

To extend the remaining quotas

1. Repeat steps 1-3 of the previous procedure.
2. On the usage plan page, choose **Extension** from the usage plan window.
3. Type a number for the **Remaining** request quotas.
4. Choose **Save**.

Set Up API Keys Using the API Gateway REST API

To set up API keys, do the following:

- Configure API methods to require an API key.
- Create or import an API key for the API in a region.

Before setting up API keys, you must have created an API and deployed it to a stage.

For the REST API calls to create and deploy an API, see [restapi:create](#) and [deployment:create](#), respectively.

Topics

- [Require an API Key on a Method \(p. 219\)](#)
- [Create or Import API Keys \(p. 220\)](#)

Require an API Key on a Method

To require an API key on a method, do one of the following:

- Call [method:put](#) to create a method, setting `apiKeyRequired` to `true` in the request payload.
- Call [method:update](#) to set `apiKeyRequired` to `true`.

Create or Import API Keys

To create or import an API key, do one of the following:

- Call `apikey:create` to create an API key.
- Call `apikey:import` to import an API key from a file. For the file format, see [API Gateway API Key File Format](#) (p. 222).

With the API key created, you can now proceed to [Create and Configure Usage Plans with the API Gateway REST API](#) (p. 220).

Create and Configure Usage Plans with the API Gateway REST API

Before configuring a usage plan, you must have already set up methods of a selected API to require API keys, deployed or redeployed the API to a stage, and the created or imported one or more API keys. For more information, see [Set Up API Keys Using the API Gateway REST API](#) (p. 219).

To configure a usage plan using the API Gateway REST API, use the following instructions, assuming you have already created the APIs to be added to the usage plan.

Topics

- [Migrate to Default Usage Plans](#) (p. 220)
- [Create a Usage Plan](#) (p. 220)
- [Manage a Usage Plan](#) (p. 221)
- [Test Usage Plans](#) (p. 221)

Migrate to Default Usage Plans

When creating a usage plan the first time, you can migrate existing API stages associated with selected API keys to a usage plan by calling `account:update` with the following body:

```
{
  "patchOperations" : [ {
    "op" : "add",
    "path" : "/features",
    "value" : "UsagePlans"
  } ]
}
```

For more information about migrating API stages associated with API keys see [Migrate to Default Usage Plans in the API Gateway Console](#) (p. 216).

Create a Usage Plan

The following procedure describes how to create a usage plan.

To create a usage plan with the REST API

1. Call `usageplan:create` to create a usage plan, specifying in the payload the name and description of the plan, associated API stages, rate limits, and quotas.

Make note of the resultant usage plan identifier. You will need it in the next step.

2. Do one of the following:

- a. Call `usageplankey:create` to add an API key to the usage plan, specifying `keyId` and `keyType` in the payload.

To add more API keys to the usage plan, repeat the above call, one API key at a time.

- b. Call `apikey:import` to add one or more API keys directly to the specified usage plan. The request payload should contain API key values, the associated usage plan identifier, the Boolean flags to indicate the keys are enabled for the usage plan, and, possibly, the API key names and descriptions.

The following example of the `apikey:import` request will add three API keys (as identified by `key`, `name`, and `description`) to one usage plan (as identified by `usageplanIds`):

```
POST /apikeys?mode=import&format=csv&failonwarnings=fase HTTP/1.1
Host: apigateway.us-east-1.amazonaws.com
Content-Type: text/csv
Authorization: ...

key,name,description,enabled,usageplanIds
abcdef1234ghijklmnop8901234567,importedKey_1,firstone,tRuE,
n371pt
abcdef1234ghijklmnop0123456789,importedKey_2,secondone,TRUE,n371pt
abcdef1234ghijklmnop9012345678,importedKey_3,
,true,n371pt
```

As a result, three `UsagePlanKey` resources will be created and added to the `UsagePlan`.

You can also add API keys to more than one usage plan this way. To do this, change each `usageplanIds` column value to a comma-separated string that contains the selected usage plan identifiers and is enclosed within a pair of quotes ("`n371pt,m282qs`" or '`n371pt,m282qs`').

Manage a Usage Plan

The following procedure describes how to manage a usage plan.

To manage a usage plan with the REST API

1. Call `usageplan:by-id` to get a usage plan of a given plan Id. To see the available usage plans, call `apigateway:usage-plans`.
2. Call `usageplan:update` to add a new API stage to the plan, to replace an existing API stage in the plan, to remove an API stage from the plan, or to modify the rate limits or quotas.
3. Call `usage:get` to query the usage data in a specified time interval.
4. Call `usage:update` to grant an extension to the current usage in a usage plan.

Test Usage Plans

As an example, let's use the PetStore API, created in [Create an API Gateway API from an Example \(p. 7\)](#). Assume the API is configured to use an API key of `Hiorr45VR...c4GJc`. The following steps describe how to test a usage plan.

To test your usage plan

- Make a `GET` request on the Pets resource (`/pets`), with the `?type=...&page=...` query parameters, of the API (e.g., `xbvx1pijch`) in a usage plan:

```
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

You must submit this request to the `execute-api` component of API Gateway and provide the required API key (e.g., `Hiorr45VR...c4GJc`) in the required `x-api-key` header.

The successful response returns a 200 OK status code and a payload containing the requested results from the back end. If you forget to set the `x-api-key` header or set it with an incorrect key, you will get a 403 Forbidden response. On the other hand, if you did not configure the method to require an API key, you will likely to get a 200 OK response whether you set the `x-api-key` header correctly or not and the throttle and quota limits of the usage plan are bypassed.

API Gateway API Key File Format

API Gateway can import API keys from external files of a comma-separated value (CSV) format and associate the imported keys with one or more usage plans. The imported file must contain the `Name` and `Key` columns. The column header names are not case-sensitive and columns can be in any order, as shown in the following example:

```
Key,name
apikey1234abcdefg hij0123456789,MyFirstApiKey
```

A `Key` value must be between 30 and 128 characters.

An API key file can also have the `Description`, `Enabled`, or `UsagePlanIds` column, as shown in the following example:

```
Name,key,description,Enabled,usageplanIds
MyFirstApiKey,apikey1234abcdefg hij0123456789,An imported key,TRUE,c7y23b
```

When a key is associated with more than one usage plan, the `UsagePlanIds` value is a comma-separated string of the usage plan Ids enclosed with a pair of double or single quotes, as shown in the following example:

```
Enabled,Name,key,UsageplanIds
true,MyFirstApiKey,apikey1234abcdefg hij0123456789,"c7y23b,glvrsr"
```

Unrecognized columns are permitted, but will be ignored. The default value is an empty string or a true Boolean value.

The same API key can be imported multiple times with the most recent version overwriting the previous one. Two API keys are identical if they have the same `key` value.

Use API Keys in API Gateway without Usage Plans Enabled

Since the API Gateway usage plans feature was launched on August 11, 2016, usage plans will be enabled for you in a region where you start using API Gateway the first time. In this case, you must use a usage plan to associate an API key with an API stage and follow the instructions in [Use API Gateway Usage Plans \(p. 212\)](#).

If, on the other hand, you created an API Gateway API before then and have not enabled usage plans in the region since then, you can follow the instructions given in this section to associate an API key with an API stage to identify API callers, to restrict API access to those users with matching API keys, and to curtail abusive uses with different API keys.

To use an API key without enabling usage plans, you need to perform the following tasks:

- Create an API key.
- Enabled it on specific API methods.
- Deploy the API to a stage.
- Associate the key with the API stage.
- Distribute the key to your customers and ask them to supply it in calls to the enabled API methods.

API keys are not meant as a security mechanism for controlling access to an API. To enable secure access control, use [IAM permissions \(p. 160\)](#), [custom authorizers \(p. 174\)](#) or a [Amazon Cognito Your User Pool \(p. 182\)](#).

Topics

- [Prerequisites \(p. 223\)](#)
- [Use an API Key without Usage Plans \(p. 223\)](#)

Prerequisites

1. You must not have API Gateway usage plans enabled.
2. You must have an API available in API Gateway. Follow the instructions in [Creating an API \(p. 70\)](#).
3. You must have deployed the API in API Gateway at least once. Follow the instructions in [Deploying an API \(p. 230\)](#).

Use an API Key without Usage Plans

Note

The following instructions apply only if usage plans are enabled for you. To associate an API key with an API stage in a usage plan, follow the instructions in [Use API Gateway Usage Plans \(p. 212\)](#).

To use the API Gateway console to enable an API key without usage plans, follow these instructions:

1. Sign in to the API Gateway console at <https://console.aws.amazon.com/apigateway>.
2. Enable API key on an method:
 - a. Choose a method under a resource of your choosing.
 - b. Choose the **Method Request** box
 - c. If **API Key Required** is set to **false**, choose the pencil icon next to it. Choose **true** from the drop-down list and then choose the check-mark icon to save the setting.

The screenshot shows the Amazon API Gateway console interface. On the left, a navigation pane lists resources: a root resource with GET, OPTIONS, and POST methods; a resource path `/pets` with GET, OPTIONS, and POST methods; and a resource path `/petId` with GET and OPTIONS methods. The main content area is titled 'Method Execution / - GET - Method Request'. Below the title, there is a section for 'Authorization Settings'. The current authorization is set to 'NONE'. A dropdown menu for 'API Key Required' is set to 'true' and is highlighted with a red circle. Below this, there are expandable sections for 'URL Query String Parameters', 'HTTP Request Headers', and 'Request Models Create a Model'.

Note that the steps above configure the API Gateway to enforce using API key on the method. Otherwise, the API key created following the instructions below will not be used for any of such calls.

3. Deploy or redeploy the API for the requirement to take effect.
4. Create an API key:
 - a. In the API Gateway main navigation pane, choose **API Keys**.
 - b. Choose **Create API Key** from the **Actions** drop-down menu.
 - c. For **Name**, type a name for the API key entry.
 - d. Choose either the **Auto Generate** option for API Gateway to generate the key or the **Custom** option for you specify the key value manually. A key value is an alphanumeric string of a size between 30 and 128 characters long.
 - e. Optionally, type a description for the API key in the **Description** text box.
 - f. Choose **Save**. Make a note of the key displayed in **API key**. Callers of the enabled method must specify the key value in the `x-api-key` header.

We have introduced a new feature called [Usage Plans](#). With usage plans you can set throttling and request quota limits on a per-API-key basis. There are no additional charges for enabling Usage Plans. Click "Enable Usage Plans" to migrate your existing API keys to a usage plan.

[Enable Usage Plans](#)

API Keys Actions ▾

Search...

- my-second-api-key
- my_fist_api_key
- myApiKey

Create API Key

Name* my-test-api-key

API key* Auto Generate Custom

Description my test api key

* Required

[Save](#)

- g. Choose **Show** next to **API key** to view the newly created API key. Your customers must provide this key as the `x-api-key` header value when they call this method.

my-test-api-key Delete A

ID pmyg3c3aih

Name my-test-api-key

API key [Show](#)

Description my test api key

Enabled Enabled ⓘ

The generated API key is enabled by default, allowing the API caller to access the API, provided that the supplied API key matches the one configured. To prevent the apps with the

specified API key from accessing the API, choose **Edit**, deselect the **Enabled** option, and then choose **Save**.

5. Associate an API key with an API stage (outside of a usage plan):
 - a. Under **API Stage Association**, choose the name of the API from the **Select API** drop-down list.
 - b. Choose the name of a stage of the chosen API from the **Select stage** drop-down list.
 - c. Choose **Add** to save the setting.

API Stage Association

Stages Enabled	
API	Stage
No associated stages	

Note: this step is available only when usage plans have not been enabled for your account and region.

6. Distribute the API key to your customers and ask them to add the key as the `x-api-key` header to call the key-required method. For example, if the API key of `hzYAV09Sg98nsNh81M8402kyXVy6K1xwHD8` is required on the `GET /` method in the `test` stage of an API (`yd4f8dz2vf`), the caller must submit the following request to invoke the method.

```
GET /test HTTP/1.1
Host: yd4f8dz2vf.execute-api.us-east-1.amazonaws.com
Content-Type: application/json
x-api-key: hzYAV09Sg98nsNh81M8402kyXVy6K1xwHD8
```

Maintaining an API in Amazon API Gateway

Topics

- [View a List of APIs in API Gateway \(p. 227\)](#)
- [Delete an API in API Gateway \(p. 227\)](#)
- [Delete a Resource in API Gateway \(p. 228\)](#)
- [View a Methods List in API Gateway \(p. 228\)](#)
- [Delete a Method in API Gateway \(p. 229\)](#)

View a List of APIs in API Gateway

Use the API Gateway console to view a list of APIs.

Topics

- [Prerequisites \(p. 227\)](#)
- [View a List of APIs with the API Gateway Console \(p. 227\)](#)

Prerequisites

- You must have an API available in API Gateway. Follow the instructions in [Creating an API \(p. 70\)](#).

View a List of APIs with the API Gateway Console

1. Sign in to the API Gateway console at <https://console.aws.amazon.com/apigateway>.
2. The list of APIs is displayed.

Delete an API in API Gateway

Use the API Gateway console to delete an API.

Warning

Deleting an API means that you can no longer call it. This action cannot be undone.

Topics

- [Prerequisites \(p. 228\)](#)
- [Delete an API with the API Gateway Console \(p. 228\)](#)

Prerequisites

- You must have deployed the API at least once. Follow the instructions in [Deploying an API \(p. 230\)](#).

Delete an API with the API Gateway Console

1. Sign in to the API Gateway console at <https://console.aws.amazon.com/apigateway>.
2. In the box that contains the name of the API you want to delete, choose **Resources**.
3. Choose **Delete API**.
4. When prompted to delete the API, choose **Ok**.

Delete a Resource in API Gateway

Use the API Gateway console to delete a resource.

Warning

When you delete a resource, you also delete its child resources and methods. Deleting a resource may cause part of the corresponding API to be unusable. Deleting a resource cannot be undone.

Delete a Resource with the API Gateway Console

1. Sign in to the API Gateway console at <https://console.aws.amazon.com/apigateway>.
2. In the box that contains the name of the API for the resource you want to delete, choose **Resources**.
3. In the **Resources** pane, choose the resource, and then choose **Delete Resource**.
4. When prompted, choose **Delete**.

View a Methods List in API Gateway

Use the API Gateway console to view a list of methods for a resource.

Topics

- [Prerequisites \(p. 228\)](#)
- [View a Methods List with the API Gateway Console \(p. 229\)](#)

Prerequisites

- You must have methods available in API Gateway. Follow the instructions in [Build an API Gateway API to Expose an HTTP Endpoint \(p. 6\)](#).

View a Methods List with the API Gateway Console

1. Sign in to the API Gateway console at <https://console.aws.amazon.com/apigateway>.
2. In the box that contains the name of the API, choose **Resources**.
3. The list of methods is displayed in the **Resources** pane.

Tip

You may need to choose the arrow next to one or more resources to display all of the available methods.

Delete a Method in API Gateway

Use the API Gateway console to delete a method.

Warning

Deleting a method may cause part of the corresponding API to become unusable. Deleting a method cannot be undone.

Delete a Method with the API Gateway Console

1. Sign in to the API Gateway console at <https://console.aws.amazon.com/apigateway>.
2. In the box that contains the name of the API for the method, choose **Resources**.
3. In the **Resources** pane, choose the arrow next to the resource for the method.
4. Choose the method, and then choose **Delete Method**.
5. When prompted, choose **Delete**.

Deploying an API in Amazon API Gateway

After an API is created, you must deploy it to make it public callable. A deployment takes place in stages. A stage corresponds to a version of the API in service. In each stage, you can configure stage-level throttling settings, in addition to enabling or disabling API cache or CloudWatch logs for the API's requests and responses. If the stage-level settings are enabled, you have options to override them for individual methods. You can also define stage variables and use them to pass deployment-specific environment data to the API integration at the run time.

Topics

- [Deploy an API with the Amazon API Gateway Console \(p. 230\)](#)
- [Deploy an API in Stages in Amazon API Gateway \(p. 232\)](#)
- [Manage API Request Throttling \(p. 236\)](#)
- [Enable Amazon API Gateway Caching in a Stage to Enhance API Performance \(p. 236\)](#)
- [Manage API Gateway API Deployment with Stage Variables \(p. 242\)](#)
- [Generate and Use an SDK for an API in API Gateway \(p. 254\)](#)
- [Use a Custom Domain Name in API Gateway \(p. 280\)](#)

Deploy an API with the Amazon API Gateway Console

Prerequisites

- You must specify settings for all of the methods in the API you want to deploy. Follow the instructions in [Set up Method and Integration \(p. 71\)](#).

Deploy an API with the API Gateway Console

Note

If you want to change a stage in API Gateway to use a different deployment, see [Change a Stage to Use a Different Deployment with the API Gateway Console \(p. 231\)](#) instead.

1. Sign in to the API Gateway console at <https://console.aws.amazon.com/apigateway>.
2. In the box that contains the name of the API you want to deploy, choose **Resources**.
3. In the **Resources** pane, choose **Deploy API**.
4. For **Deployment stage**, do one of the following:
 - To deploy the API to an existing stage, choose the name of the stage.
 - To deploy the API to a new stage, choose **New Stage**. For **Stage name**, type the name of the stage you want to use for the deployment.

Tip

The stage name should be meaningful, but short enough to be easy and fast to type. Your users will specify this name as part of the URL they will use to invoke the API.

5. (Optional) For **Stage description**, type a description for the stage.
6. (Optional) For **Deployment description**, type a description for the deployment.
7. Choose **Deploy**.

Update deployment configuration with the API Gateway Console

After an API is deployed to a stage, you can, optionally, modify the deployment by updating the stage settings or stage variables. After making any changes, you must redeploy the API. The following procedure demonstrates how to accomplish with the API Gateway Console.

1. If needed, choose the **Settings** tab in the **Stage Editor** pane of the API Gateway Console.

You can then choose to use or not use API cache, to enable or disable CloudWatch logs, to change throttling settings, or to select or deselect a client certificate.
2. If needed, choose the **Stage Variables** tab in the **Stage Editor** pane of the API Gateway Console.

You can then choose to update the values of selected stage variables.
3. If you made any change, choose the **Save Changes** button; go back to the **Resources** window; and then choose **Deploy API** again.

Note

If the updated settings, such as enabling logging, requires a new IAM role, you can add the required IAM role without redeploying the API. However, it can take a few minutes before the new IAM role takes effect. Before that happens, traces of your API calls will not be logged even if you have enabled the logging option.

Change a Stage to Use a Different Deployment with the API Gateway Console

Once you have deployed an API more than once, you can choose a specific deployment for a given stage. The following procedure shows how to do this.

1. You must have deployed to the stage at least twice. Follow the instructions in [Deploy an API with the API Gateway Console \(p. 230\)](#).
2. Sign in to the API Gateway console at <https://console.aws.amazon.com/apigateway>.
3. In the box that contains the name of the API with the stage you want to change, choose **Stages**.
4. Choose the stage you want to update the deployment.

5. On the **Deployment History** tab, choose the option button next to the deployment you want the stage to use.
6. Choose **Change Deployment**.

Deploy an API in Stages in Amazon API Gateway

In API Gateway, a stage defines the path through which an API deployment is accessible.

Use the API Gateway console to deploy an API in stages.

- [Create a Stage \(p. 232\)](#)
- [View a List of Stages \(p. 232\)](#)
- [Set Up a Stage \(p. 233\)](#)
- [Delete a Stage \(p. 235\)](#)

Create a Stage in API Gateway

Use the API Gateway console to create a stage for an API.

Topics

- [Prerequisites \(p. 232\)](#)
- [Create a Stage with the API Gateway Console \(p. 232\)](#)

Prerequisites

1. You must have an API available in API Gateway. Follow the instructions in [Creating an API \(p. 70\)](#).
2. You must have deployed the API at least once. Follow the instructions in [Deploying an API \(p. 230\)](#).

Create a Stage with the API Gateway Console

1. Sign in to the API Gateway console at <https://console.aws.amazon.com/apigateway>.
2. In the box that contains the name of the API, choose **Stages**.
3. Choose **Create Stage**.
4. For **Stage name**, type a name for the stage.
5. (Optional) For **Stage description**, type a description for the stage.
6. For **Deployment**, choose the date and time of the existing API deployment you want to associate with this stage.
7. Choose **Create**.

View a List of Stages in API Gateway

Use the API Gateway console to view a list of stages in API Gateway.

Topics

- [Prerequisites \(p. 233\)](#)
- [View a List of Stages with the API Gateway Console \(p. 233\)](#)

Prerequisites

1. You must have an API available in API Gateway. Follow the instructions in [Creating an API \(p. 70\)](#).
2. You must have deployed the API in API Gateway at least once. Follow the instructions in [Deploying an API \(p. 230\)](#).

View a List of Stages with the API Gateway Console

1. Sign in to the API Gateway console at <https://console.aws.amazon.com/apigateway>.
2. In the box that contains the name of the API, choose **Stages**.

Set Up a Stage

This section walks you through the options to set up an API deployment stage in the API Gateway console.

Topics

- [Prerequisites \(p. 233\)](#)
- [Set Up an API Deployment Stage with the API Gateway Console \(p. 233\)](#)

Prerequisites

- You must have a stage available in API Gateway. Follow the instructions in [Create a Stage \(p. 232\)](#).

Set Up an API Deployment Stage with the API Gateway Console

1. Sign in to the API Gateway console at <https://console.aws.amazon.com/apigateway>.
2. In the box that contains the name of the API for the stage where you want to specify settings, choose **Stages**.
3. In the **Stages** pane, choose the name of the stage.
4. To enable a cache for the API, on the **Settings** tab, in the **Cache Settings** area, select **Enable API cache**. Then, for **Cache capacity**, choose a cache size. You can use the default for other cache settings. For information on how to set up these, . Finally, choose **Save Changes**.

Important

By selecting this box, your AWS account may be charged for API caching.

Tip

To override enabled stage-level cache settings, expand the stage under the **Stages** secondary navigation pane, choose a method. Then back in the stage editor, choose **Override for this method** for **Settings**. In the ensuing **Cache Settings** area, clear **Enable Method Cache** or customize any other desired options, before choosing **Save Changes**. For more information about the method-level and other stage-level cache settings, see [Enable API Caching \(p. 236\)](#).

5. To generate code to call the API from Android, iOS, or JavaScript, you use the **SDK Generation** tab. For more information, see [Generate and Use an SDK for an API \(p. 254\)](#).
6. To enable Amazon CloudWatch Logs for all of the methods associated with this stage of this API Gateway API, do the following:

1. On the **Settings** tab, in the **CloudWatch Settings** area, select **Enable CloudWatch Logs**.

Tip

To enable method-level CloudWatch settings, expand the stage under the **Stages** secondary navigation pane, choose each method of interest, and, back in the stage editor, choose **Override for this method** for **Settings**. In the ensuing **CloudWatch Settings** area, make sure to select **Log to CloudWatch Logs** and any other desired options, before choosing **Save Changes**.

Important

Your account will be charged for accessing method-level CloudWatch metrics, but not the API- or stage- level metrics.

2. For **Log level**, choose **ERROR** to write only error-level entries to CloudWatch Logs, or choose **INFO** to include all **ERROR** events as well as extra informational events. No sensitive data will be logged unless the **Log full requests/responses data** option is selected.
3. To write entries to CloudWatch Logs that contain full API call request and response information, select **Log full requests/responses data**.
4. Choose **Save Changes**. The new settings will take effect after a new deployment.

Important

Whether you enable CloudWatch Logs for all or only some of the methods, you must also specify the ARN of an IAM role that enables API Gateway to write information to CloudWatch Logs on behalf of your IAM user. To do this, in the secondary navigation bar, in the first list next to the console home button, choose **Settings**. Then type the ARN of the IAM role in the **CloudWatch Logging role ARN** box. For common application scenarios, the IAM role could attach the managed policy of `AmazonAPIGatewayPushToCloudWatchLogs`, which contains the following access policy statement:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:DescribeLogGroups",
        "logs:DescribeLogStreams",
        "logs:PutLogEvents",
        "logs:GetLogEvents",
        "logs:FilterLogEvents"
      ],
      "Resource": "*"
    }
  ]
}
```

The IAM role must also contain the following trust relationship statement:

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

```
        "Service": "apigateway.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

To create the IAM role, you can adapt the instructions in "To create the Lambda invocation role and its policy" and "To create the Lambda execution role and its policy" in the [Create Lambda Functions \(p. 44\)](#) section of the [Build an API to Expose a Lambda Function \(p. 44\)](#).

For more information about CloudWatch, see the [Amazon CloudWatch User Guide](#).

7. To enable Amazon CloudWatch metrics for all of the methods associated with this API in API Gateway, in the **Stage Editor** pane, on the **Settings** tab, in the **CloudWatch Settings** area, select **Enable CloudWatch metrics**, and then choose **Save Changes**. The new settings will take effect after a new deployment.

Important

By selecting this box, your AWS account may be charged for using CloudWatch.

Tip

To enable CloudWatch metrics for only some methods, clear **Enable CloudWatch metrics**. In the **Stages** pane, choose each of the methods for which you want to enable CloudWatch metrics. For each method you choose, on the **Settings** tab for the method, choose **Override for this method**, and in the **CloudWatch Settings** area, select **Enable CloudWatch metrics**. Finally, choose **Save Changes**.

For more information about CloudWatch, see the [Amazon CloudWatch User Guide](#).

8. To set a default throttle limit for all of the methods associated with this API in API Gateway, in the **Stage Editor** pane, on the **Settings** tab, in the **Throttle Settings** area, do the following, and then choose **Save Changes**:
 - For **Burst Limit**, type the absolute maximum number of times API Gateway will allow this method to be called per second. (The value of **Burst Limit** must be equal to or greater than the value of **Rate**.) The default setting is 1000 request per second.
 - For **Rate**, type the number of times API Gateway will allow this method to be called per second on average. (The value of **Rate** must be equal to or less than the value of **Burst Limit**.) The default setting is 500 request per second.

Note

- When creating a stage, if not supplied, API Gateway will enforce the default values of 1000 for **Burst Limit** and 500 for **Rate** in the stage settings.
 - In addition, API Gateway enforces overall account level throttling at the default values of 1000 for **Burst Limit** and 500 for **Rate**. If you require a higher level of throttling on your account, contact the [AWS Support Center](#) to request an increase.
 - API Gateway uses the [token bucket](#) algorithm, including [average rate](#) and [burst size](#), for both account and method throttling.
9. To change the stage to use a different deployment, in the **Stage Editor** pane, on the **Change Deployment** tab, choose the option button next to the deployment you want the stage to use, and then choose **Change Deployment**.

Delete a Stage in API Gateway

Use the API Gateway console to delete a stage in API Gateway.

Warning

Deleting a stage may cause part or all of the corresponding API to be unusable by API callers. Deleting a stage cannot be undone.

Delete a Stage with the API Gateway Console

1. Sign in to the API Gateway console at <https://console.aws.amazon.com/apigateway>.
2. In the box that contains the name of the API for the stage, choose **Stages**.
3. In the **Stages** pane, choose the stage you want to delete, and then choose **Delete Stage**.
4. When prompted, choose **Delete**.

Manage API Request Throttling

Topics

- [Account-Level Throttling \(p. 236\)](#)
- [Stage-Level and Method-Level Throttling \(p. 236\)](#)

Amazon API Gateway throttles API requests to your API using the token bucket algorithm. For more information, see [token bucket algorithm](#).

Account-Level Throttling

By default, API Gateway limits the steady-state request rates to 1000 requests per second (rps) and allows bursts of up to 2000 rps across all APIs, stages, and methods within an AWS account. If necessary, you can request an increase to your account-level limits. For more information, see [API Gateway Limits \(p. 419\)](#).

You can view account-level throttling limits in the API Gateway console. The console displays the default account-level settings before these settings are overridden by any customization. You can also read the account-level throttling limits by using the [API Gateway REST API \(p. 418\)](#).

Stage-Level and Method-Level Throttling

As an API owner, you can override the account-level request throttling limits for a specific stage or for individual methods in an API. Actual stage-level and method-level throttling limits are bounded by the account-level rate limits, even if you set the stage-level or method-level throttling limits greater than the account-level limits.

You can set the stage-level or method-level throttling limits by using the API Gateway console or by calling the [API Gateway REST API \(p. 418\)](#). For instructions using the console, see [Set Up a Stage \(p. 233\)](#).

Enable Amazon API Gateway Caching in a Stage to Enhance API Performance

Topics

- [Amazon API Gateway Caching Overview \(p. 237\)](#)
- [Enable Amazon API Gateway Caching \(p. 237\)](#)
- [Override API Gateway Stage-Level Caching for Method Caching \(p. 238\)](#)
- [Use Method or Integration Parameters as Cache Keys to Index Cached Responses \(p. 239\)](#)
- [Flush the API Stage Cache in API Gateway \(p. 240\)](#)
- [Invalidate an API Gateway Cache Entry \(p. 240\)](#)

Amazon API Gateway Caching Overview

You can enable API caching in Amazon API Gateway to cache your endpoint's response. With caching, you can reduce the number of calls made to your endpoint and also improve the latency of the requests to your API. When you enable caching for a stage, API Gateway caches responses from your endpoint for a specified time-to-live (TTL) period, in seconds. API Gateway then responds to the request by looking up the endpoint response from the cache instead of making a request to your endpoint. The default TTL value for API caching is 300 seconds. The maximum TTL value is 3600 seconds. TTL=0 means caching is disabled.

Note

Caching is charged by the hour and is not eligible for the AWS free tier.

Enable Amazon API Gateway Caching

In API Gateway, you can enable caching for all methods for a specified stage. When you enable caching, you must choose a cache capacity. In general, a larger capacity gives a better performance, but also costs more.

API Gateway enables caching by creating a dedicated cache instance. This process can take up to 4 minutes.

API Gateway changes caching capacity by removing the existing cache instance and recreating a new one with a modified capacity. All existing cached data is deleted.

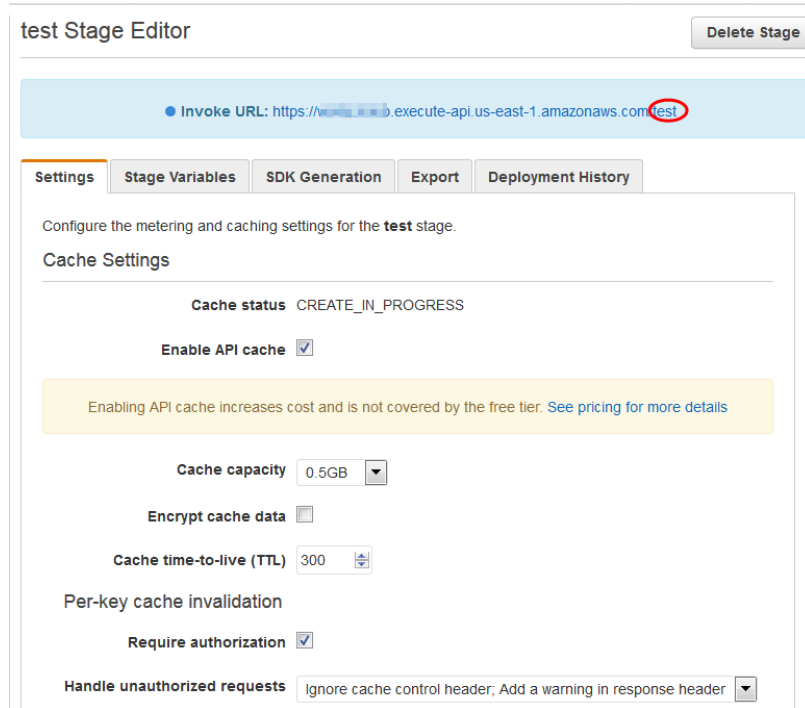
In the API Gateway console, you configure caching in the **Settings** tab of a named **Stage Editor**.

1. Go to the API Gateway console.
2. Navigate to the **Stage Editor** for the stage for which you want to enable caching.
3. Choose **Settings**.
4. Select **Enable API cache**.
5. Wait for the cache creation to complete.

Note

Creating or deleting a cache takes about 4 minutes for API Gateway to complete. When cache is created, the **Cache status** value changes from `CREATE_IN_PROGRESS` to `AVAILABLE`. When cache deletion is completed, the **Cache status** value changes from `DELETE_IN_PROGRESS` to an empty string.

When you enable caching within a stage's **Cache Settings**, you enable caching for all methods in that stage.



If you would like to verify if caching is functioning as expected, you have two general options:

- Inspect the CloudWatch metrics of CacheHitCount and CacheMissCount for your API and stage.
- Put a timestamp in the response.

Note

You should not use the X-Cache header from the CloudFront response to determine if your API is being served from your API Gateway cache instance.

Override API Gateway Stage-Level Caching for Method Caching

If you want more granularity in your caching settings, you can override the stage-level caching for individual methods. This includes disabling caching for a specific method, increasing or decreasing its TTL period, and turning on or off encryption of the cached response. If you anticipate that a method will receive sensitive data in its responses, in **Cache Settings**, choose **Encrypt cache data**.

test - GET - /streams

Invoke URL: <https://execute-api.us-east-1.amazonaws.com/test/streams>

Use this page to override the test stage settings for the GET to /streams method.

Settings Inherit from stage
 Override for this method

CloudWatch Settings

Enable CloudWatch Logs

Enable CloudWatch Metrics

Throttling Settings

Rate 500

Burst Limit 1000

Cache Settings

Configure the cache for GET to /streams

Enable Method Cache

Encrypt cache data

Cache time-to-live (TTL) 300

Per-key cache invalidation

Require authorization

Handle unauthorized requests

Use Method or Integration Parameters as Cache Keys to Index Cached Responses

When a cached method or integration has parameters, which can take the form of custom headers, URL paths, or query strings, you can use some or all of the parameters to form cache keys. API Gateway can cache the method's responses, depending on the parameter values used.

For example, suppose you have a request of the following format:

```
GET /users?type=... HTTP/1.1
host: example.com
...
```

In this request, `type` can take a value of `admin` or `regular`. If you include the `type` parameter as part of the cache key, the responses from `GET /users?type=admin` will be cached separately from those from `GET /users?type=regular`.

When a method or integration request takes more than one parameter, you can choose to include some or all of the parameters to create the cache key. For example, you can include only the `type` parameter in the cache key for the following request, made in the listed order within a TTL period:

```
GET /users?type=admin&department=A HTTP/1.1
```

```
host: example.com  
...
```

The response from this request will be cached and will be used to serve the following request:

```
GET /users?type=admin&department=B HTTP/1.1  
host: example.com  
...
```

To include a method or integration request parameter as part of a cache key in the API Gateway console, select **Caching** after you add the parameter.

← Method Execution /streams - GET - Method Request

Provide information about this method's authorization settings and the parameters it can receive.

Authorization Settings ●

Authorization NONE ⓘ

API Key Required false ✎

▼ URL Query String Parameters ●

Name	Caching	
query	<input checked="" type="checkbox"/>	✕

➕ Add query string

▶ HTTP Request Headers

▶ Request Models [Create a Model](#) ●

Flush the API Stage Cache in API Gateway

When API caching is enabled, you can flush your API stage's entire cache to ensure your API's clients get the most recent responses from your integration endpoints.

To flush the API stage cache, you can choose the **Flush Cache** button under the **Stage** tab in the API Gateway console. Notice that flushing the cache will cause the responses to ensuing requests to be serviced from the back end until the cache is build up again. During this period, the number of requests sent to the integration endpoint may increase. That may affect the overall latency of your API.

Invalidate an API Gateway Cache Entry

A client of your API can invalidate an existing cache entry and reloads it from the integration endpoint for individual requests. The client must send a request that contains the `Cache-Control: max-age=0` header. The client receives the response directly from the integration endpoint instead of the cache, provided that the user is authorized to do so. This replaces the existing cache entry with the new response, which is fetched from the integration endpoint.

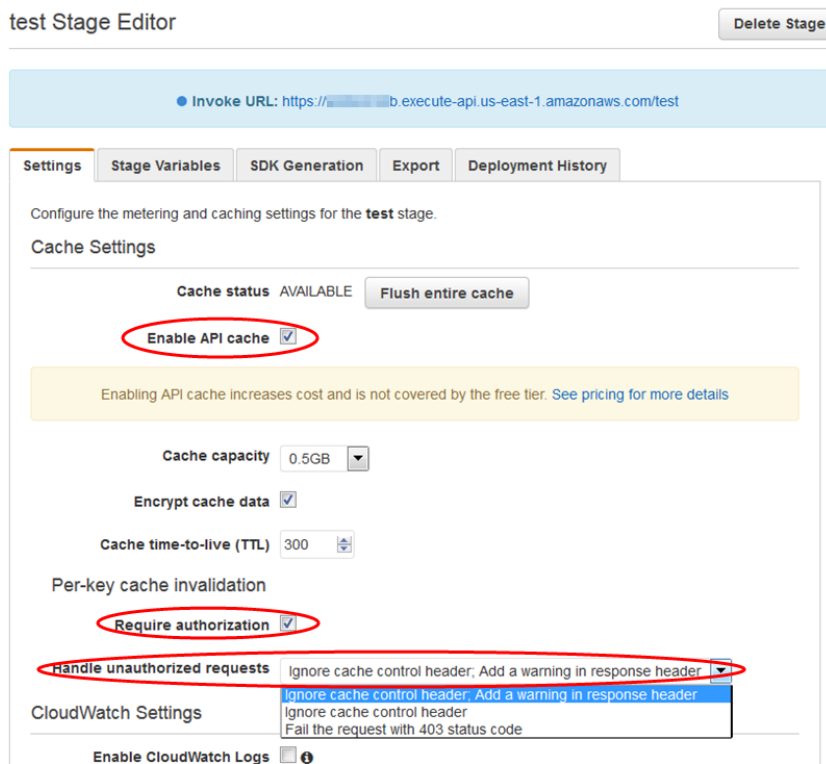
To grant permission for a caller, attach a policy of the following format to an IAM execution role for the user.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "execute-api:InvalidateCache"
      ],
      "Resource": [
        "arn:aws:execute-api:region:account-id:api-id/stage-name/HTTP-VERB/resource-path-specifier"
      ]
    }
  ]
}
```

This policy allows the API Gateway execution service to invalidate cache for requests on the specified resource (or resources). To specify a group of targeted resources, use a wildcard (*) character for `account-id`, `api-id`, and other entries in the ARN value of `Resource`. For more information on how to set permissions for the API Gateway execution service, see [Use IAM Permissions \(p. 160\)](#)

If you do not impose an `InvalidateCache` policy, any client can invalidate the API cache. If all or most of the clients invalidate the API cache, there could be significant latency impact on your API.

When the policy is in place, caching is enabled, and authorization is required, you can control how unauthorized requests are handled by choosing an option from **Handle unauthorized requests** in the API Gateway console.



The three options result in the following behaviors:

- **Fail the request with 403 status code:** returns a 403 Unauthorized response.

To set this option using the API, use `FAIL_WITH_403`.

- **Ignore cache control header; Add a warning in response header:** process the request and add a warning header in the response.

To set this option using the API, use `SUCCEED_WITH_RESPONSE_HEADER`.

- **Ignore cache control header:** process the request and do not add a warning header in the response.

To set this option using the API, use `SUCCEED_WITHOUT_RESPONSE_HEADER`.

Manage API Gateway API Deployment with Stage Variables

Stage variables are name-value pairs that you can define as configuration attributes associated with a deployment stage of an API. They act like environment variables and can be used in your API setup and mapping templates.

For example, you can define a stage variable in a stage configuration, and then set its value as the URL string of an HTTP integration for a method in your API. Later, you can reference the URL string using the associated stage variable name from the API setup. This way, you can use the same API setup with a different endpoint at each stage by resetting the stage variable value to the corresponding URLs. You can also access stage variables in the mapping templates, or pass configuration parameters to your AWS Lambda or HTTP back end.

For more information about mapping templates, see [Request and Response Payload-Mapping Reference \(p. 134\)](#).

Use Cases

With deployment stages in API Gateway, you can manage multiple release stages for each API, such as alpha, beta, and production. Using stage variables you can configure an API deployment stage to interact with different back-end endpoints. For example, your API can pass a GET request as an HTTP proxy to the back-end web host (for example, `http://example.com`). In this case, the back-end web host is configured in a stage variable so that when developers call your production endpoint, API Gateway calls `example.com`. When you call your beta endpoint, API Gateway uses the value configured in the stage variable for the beta stage, and calls a different web host (for example, `beta.example.com`). Similarly, stage variables can be used to specify a different AWS Lambda function name for each stage in your API.

You can also use stage variables to pass configuration parameters to a Lambda function through your mapping templates. For example, you may want to re-use the same Lambda function for multiple stages in your API, but the function should read data from a different Amazon DynamoDB table depending on which stage is being called. In the mapping templates that generate the request for the Lambda function, you can use stage variables to pass the table name to Lambda.

Examples

To use a stage variable to customize the HTTP integration endpoint, you must first configure a stage variable of a specified name, e.g., `url`, and then assign it a value, e.g., `example.com`. Next, from your method configuration, set up an HTTP proxy integration, and instead of entering the endpoint's URL,

you can tell API Gateway to use the stage variable value, `http://${stageVariables.url}`. This value tells API Gateway to substitute your stage variable `${}` at runtime, depending on which stage your API is running. You can reference stage variables in a similar way to specify a Lambda function name, an AWS Service Proxy path, or an AWS role ARN in the credentials field.

When specifying a Lambda function name as a stage variable value, you must configure the permissions on the Lambda function manually. You can use the AWS Command Line Interface to do this.

```
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
```

The following example assigns API Gateway permission to invoke a Lambda function named `helloWorld` hosted in the US West (Oregon) region of an AWS account on behalf of the API method.

```
arn arn:aws:execute-api:us-west-2:123123123123:bmmuvptwze/*/GET/hello
```

Here is the same command using the AWS CLI.

```
aws lambda add-permission --function-name arn:aws:lambda:us-  
east-1:123123123123:function:helloWorld --source-arn arn:aws:execute-  
api:us-west-2:123123123123:bmmuvptwze/*/GET/hello --principal  
apigateway.amazonaws.com --statement-id apigateway-access --action  
lambda:InvokeFunction
```

Set Stage Variables Using the Amazon API Gateway Console

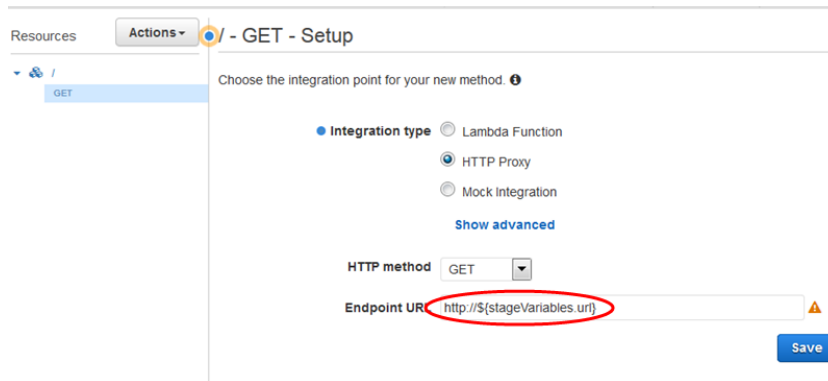
In this tutorial, you will learn how to set stage variables for two deployment stages of a sample API, using the Amazon API Gateway console.

Prerequisites

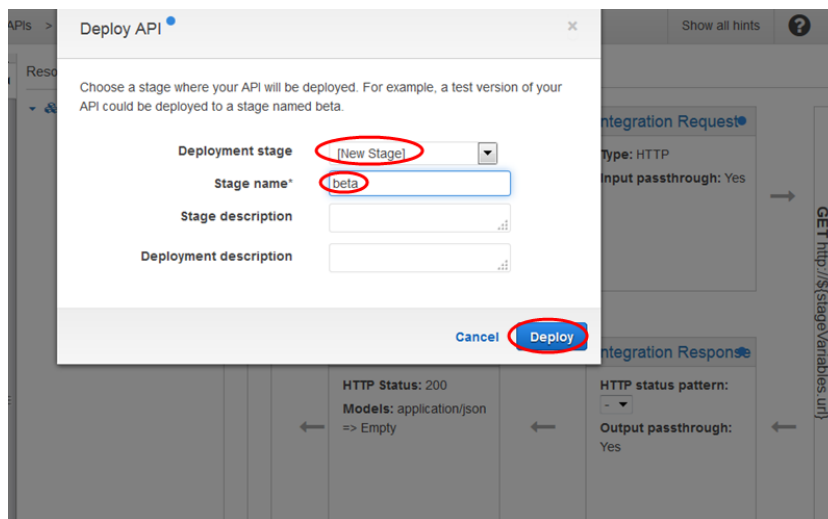
1. You must have an API available in API Gateway. Follow the instructions in [Creating an API \(p. 70\)](#).
2. You must have deployed the API at least once. Follow the instructions in [Deploying an API \(p. 230\)](#).
3. You must have created the first stage for a deployed API. Follow the instructions in [Create a Stage \(p. 232\)](#).

To Declare Stage Variables Using the API Gateway Console

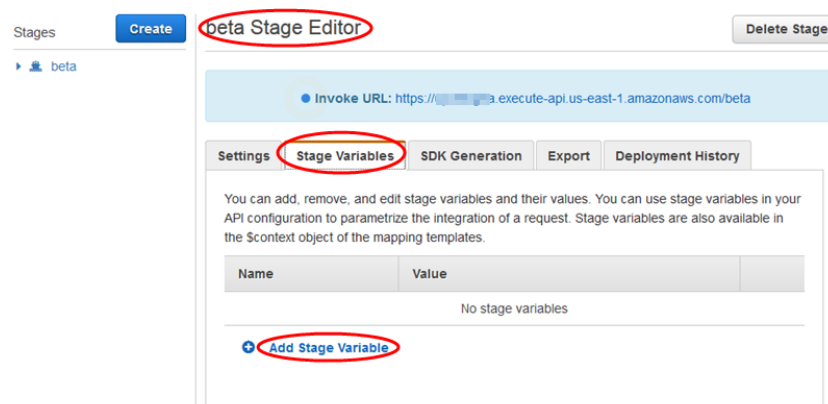
1. Sign in to the API Gateway console at <https://console.aws.amazon.com/apigateway>.
2. Create an API, create a `GET` method on the API's root resource, if you have not already done so. Set the HTTP **Endpoint URL** value as `"http://${stageVariables.url}"`, and then choose **Save**.



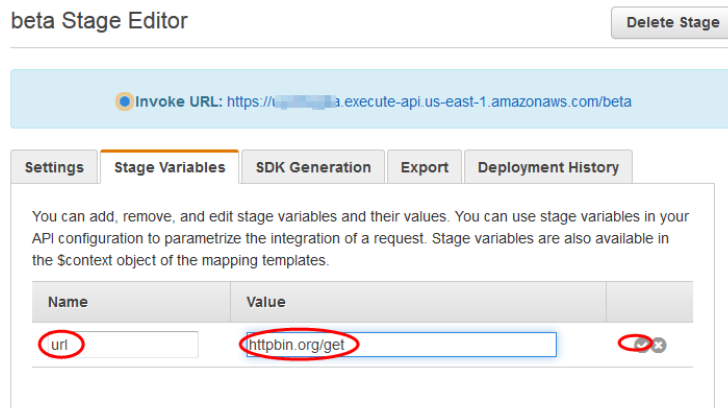
3. Choose **Deploy API**. Choose **New Stage** and enter "beta" for **Stage name**. Choose **Deploy**.



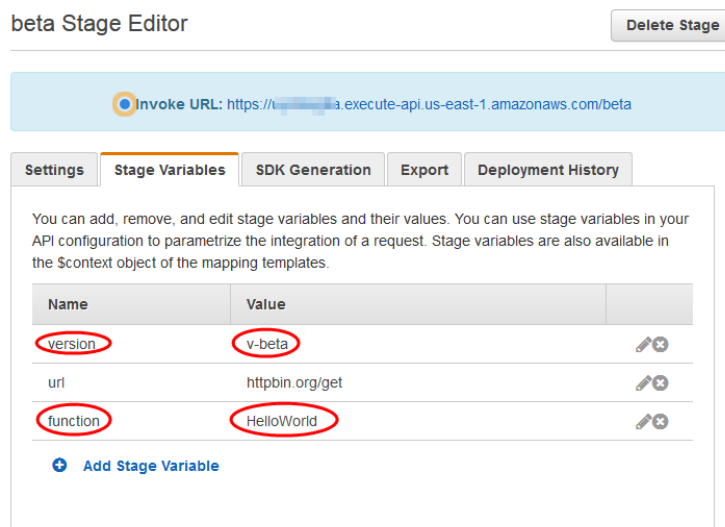
4. In the **beta Stage Editor** panel; choose the **Stage Variables** tab; and then choose **Add Stage Variable**.



5. Enter the "url" string in the **Name** field and the "httpbin.org/get" in the **Value** field. Choose the checkmark icon to save the setting for the stage variable.



- Repeat the above step to add two more stage variables: `version` and `function`. Set their values as "v-beta" and "HelloWorld", respectively.



Note

When setting a Lambda function as the value of a stage variable, use the function's local name, possibly including its alias or version specification, as in `HelloWorld`, `HelloWorld:1` or `HelloWorld:alpha`. Do not use the function's ARN (for example, `arn:aws:lambda:us-east-1:123456789012:function:HelloWorld`). The API Gateway console assumes the stage variable value for a Lambda function as the unqualified function name and will expand the given stage variable into an ARN.

- From the **Stages** navigation pane, choose **Create**. For **Stage name**, type `prod`. Select a recent deployment from **Deployment** and then choose **Create**.

Stages **Create** Create Stage

beta

Create a stage where your APIs will be deployed. For example, a test version of your API could be deployed to a stage named beta.

Stage name* prod

Stage description

Deployment* 04-15-2016 10:38

Deployment Description

Create

- As with the **beta** stage, set the same three stage variables (**url**, **version**, and **function**) to different values ("petstore-demo-endpoint.execute-api.com/petstore/pets", "v-prod", and "HelloEveryone"), respectively.

prod Stage Editor **Delete Stage**

Invoke URL: https://...a.execute-api.us-east-1.amazonaws.com/prod

Settings **Stage Variables** SDK Generation Export Deployment History

You can add, remove, and edit stage variables and their values. You can use stage variables in your API configuration to parametrize the integration of a request. Stage variables are also available in the \$context object of the mapping templates.

Name	Value	
version	v-prod	
url	petstore-demo-endpoint.execute-api.com/petstore/pets	
function	HelloEveryone	

Add Stage Variable

Use Amazon API Gateway Stage Variables

You can use API Gateway stage variables to access the HTTP and Lambda back ends for different API deployment stages and to pass stage-specific configuration metadata into an HTTP back end as a query parameter and into a Lambda function as a payload generated in an input mapping template.

Prerequisites

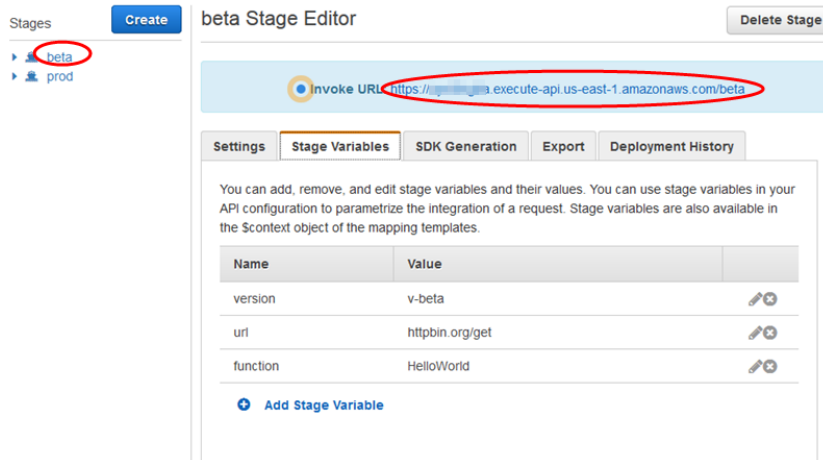
You must create two stages with a `url` variable set to two different HTTP endpoints: a `function` stage variable assigned to two different Lambda functions, and a `version` stage variable containing stage-specific metadata. Follow the instructions in [Set Stage Variables Using the Amazon API Gateway Console](#) (p. 243).

Access an HTTP endpoint through an API with a stage variable

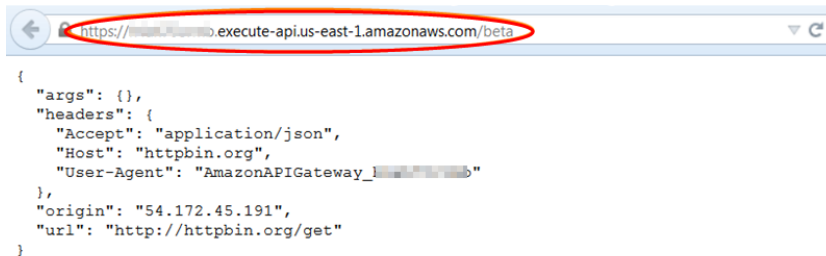
- In the **Stages** navigation pane, choose **beta**. In **beta Stage Editor**, choose the **Invoke URL** link. This starts the **beta** stage `GET` request on the root resource of the API.

Note

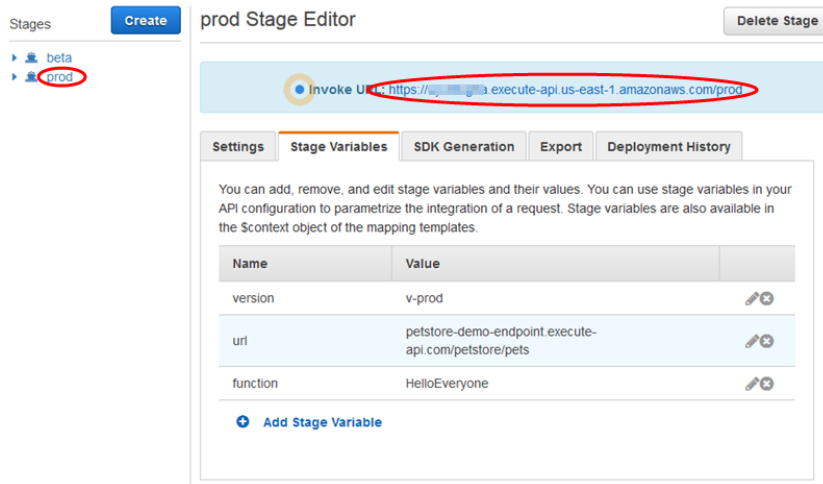
The **Invoke URL** link points to the root resource of the API in its **beta** stage. Navigating to the URL by choosing the link calls the **beta** stage **GET** method on the root resource. If methods are defined on child resources and not on the root resource itself, choosing the **Invoke URL** link will return a `{"message": "Missing Authentication Token"}` error response. In this case, you must append the name of a specific child resource to the **Invoke URL** link.



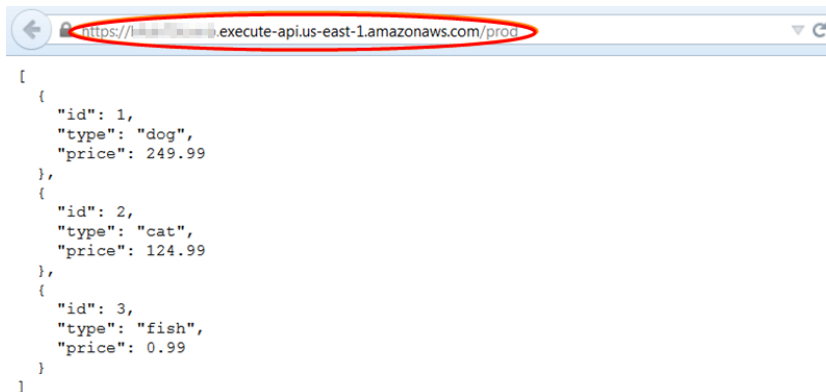
2. The response you get from the **beta** stage **GET** request is shown next. You can also verify the result by using a browser to navigate to **http://httpbin.org/get**. This value was assigned to the `url` variable in the **beta** stage. The two responses are identical.



3. In the **Stages** navigation pane, choose the **prod** stage. From **prod Stage Editor**, choose the **Invoke URL** link. This starts the **prod** stage **GET** request on the root resource of the API.



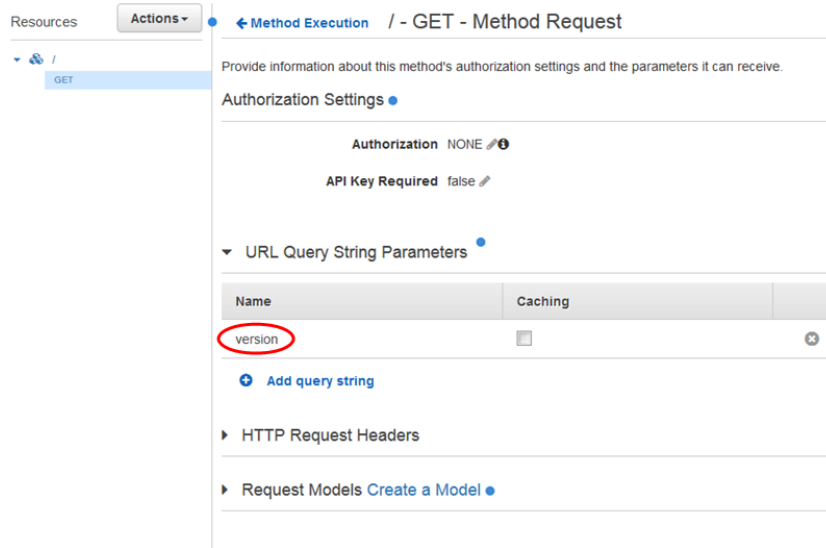
4. The response you get from the **prod** stage **GET** request is shown next. You can verify the result by using a browser to navigate to <http://petstore-demo-endpoint-execute-api.com/petstore/pets>. This value was assigned to the `url` variable in the **prod** stage. The two responses are identical.



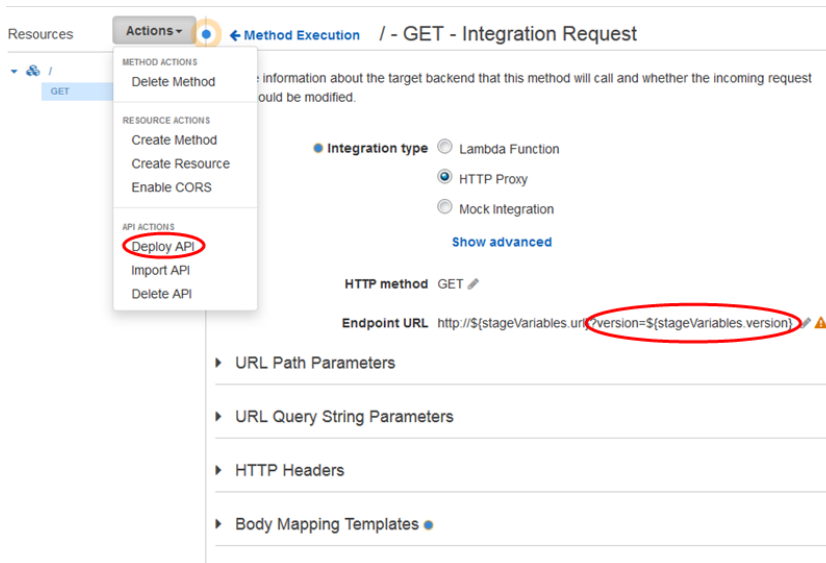
Pass stage-specific metadata to an HTTP back end via a stage variable in a query parameter expression

This procedure describes how to use a stage variable value in a query parameter expression to pass stage-specific metadata into an HTTP back end. We will use the `version` stage variable declared in [Set Stage Variables Using the Amazon API Gateway Console \(p. 243\)](#).

1. In the **Resource** navigation pane, choose the **GET** method. To add a query string parameter to the method's URL, in **Method Execution**, choose **Method Request** . Type **version** for the parameter name.



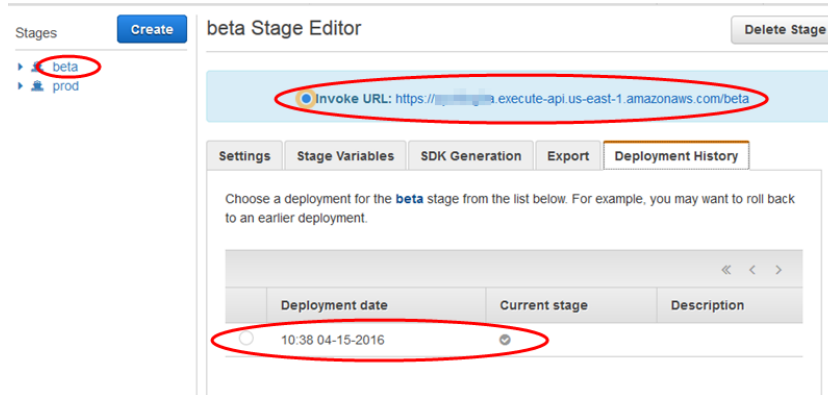
2. In **Method Execution** choose **Integration Request**. Edit the **Endpoint URL** value to append `?version=${stageVariables.version}` to the previously defined URL value, which, in this case, is also expressed with the `url` stage variable. Choose **Deploy API** to deploy these changes.



3. In the **Stages** navigation pane, choose the **beta** stage. From **beta Stage Editor**, verify that the current stage is in the most recent deployment, and then choose the **Invoke URL** link.

Note

We use the beta stage here because the HTTP endpoint, as specified by the `url` variable, "http://httpbin.org/get", accepts query parameter expressions and returns them as the `args` object in its response.



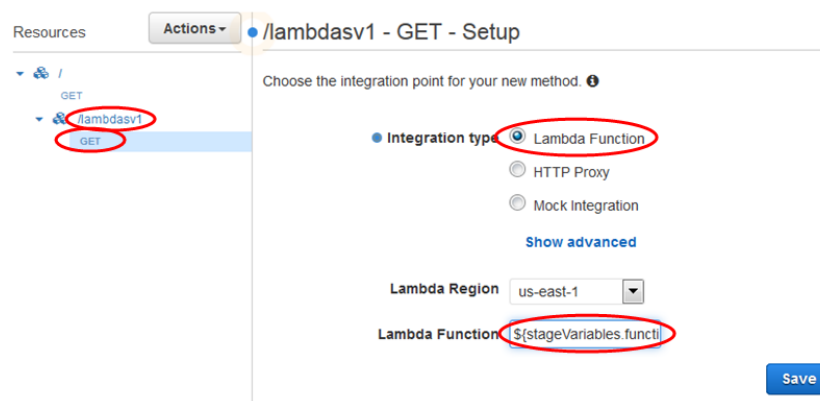
- The response is shown next. Notice that `v-beta`, assigned to the `version` stage variable, is passed in the back end as the `version` argument.

```
{
  "args": {
    "version": "v-beta"
  },
  "headers": {
    "Accept": "application/json",
    "Host": "httpbin.org",
    "User-Agent": "AmazonAPIGateway_h4ah70cvmb"
  },
  "origin": "52.91.42.97",
  "url": "http://httpbin.org/get?version=v-beta"
}
```

Call Lambda function through API with a stage variable

This procedure describes how to use a stage variable to call a Lambda function as a back end of your API. We will use the `function` stage variable declared earlier. For more information, see [Set Stage Variables Using the Amazon API Gateway Console](#) (p. 243).

- In the **Resources** pane, create a `/lambdasv1` child resource under the root directory, and then create a `GET` method on the child resource. Set the **Integration type** to **Lambda Function**, and in **Lambda Function**, type `${stageVariables.function}`. Choose **Save**.



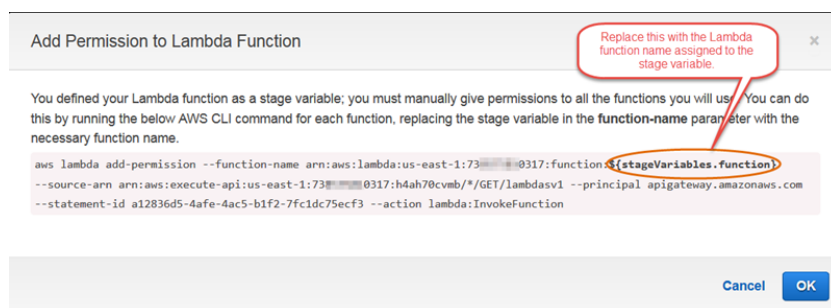
Tip

When prompted with **Add Permission to Lambda Function**, make a note of the AWS CLI command before choosing **OK**. You must run the command on each Lambda function

that is or will be assigned to the `function` stage variable for each of the newly created API methods. For example, if the `stageVariables.function` value is `HelloWorld` and you have not added permission to this function yet, you must run the following AWS CLI command:

```
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/lambda$stageVariables.function --principal apigateway.amazonaws.com --statement-id statement-id-guid --action lambda:InvokeFunction
```

Failing to do so results in a 500 Internal Server Error response when invoking the method. Make sure to replace `stageVariables.function` with the Lambda function name that is assigned to the stage variable.



2. Deploy the API to available stages.
3. In the **Stages** navigation pane, choose the **beta** stage. Verify that your most recent deployment is in **beta Stage Editor**. Copy the **Invoke URL** link, paste it into the address bar of your browser, and append `/lambda$stageVariables.function` to that URL. This calls the underlying Lambda function through the `GET` method on the **LambdaSv1** child resource of the API.

Note

Your `HelloWorld` Lambda function implements the following code.

```
exports.handler = function(event, context) {  
  if (event.version)  
    context.succeed('Hello, World! (' + event.version + ')');  
  else  
    context.succeed("Hello, world! (v-unknown)");  
};
```

This implementation results in the following response.

```
"Hello, world! (v-unknown)"
```

Pass stage-specific metadata to a Lambda function via a stage variable

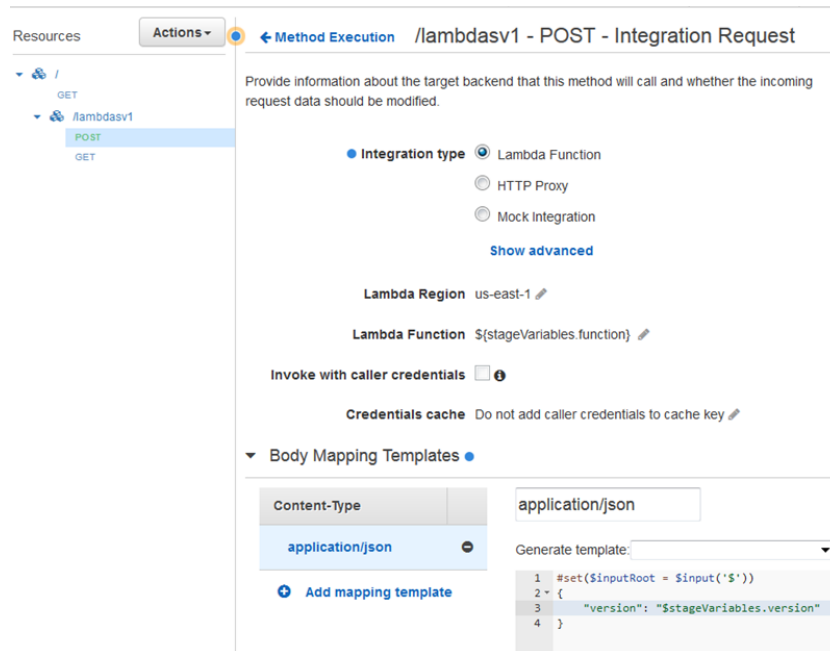
This procedure describes how to use a stage variable to pass stage-specific configuration metadata into a Lambda function. We will use a `POST` method and an input mapping template to generate payload using the `version` stage variable declared earlier.

1. In the **Resources** pane, choose the `/lambdasv1` child resource. Create a `POST` method on the child resource, set the **Integration type** to **Lambda Function**, and type `${stageVariables.function}` in **Lambda Function**. Choose **Save**.

Tip

This step is similar to the step we used to create the `GET` method. For more information, see [Call Lambda function through API with a stage variable](#) (p. 250).

2. From the **/Method Execution** pane, choose **Integration Request**. In the **Integration Request** pane, expand **Mapping Templates**, and then choose **Add mapping template** to add a template for the `application/json` content-type, as shown in the following.



Note

In a mapping template, a stage variable must be referenced within quotes (as in `"${stageVariables.version}"` or `"${stageVariables.version}"`), whereas elsewhere it must be referenced without quotes (as in `${stageVariables.function}`).

3. Deploy the API to available stages.
4. In the **Stages** navigation pane, choose **beta**. In **beta Stage Editor**, verify that the current stage has the most recent deployment. Copy the **Invoke URL** link, paste it into the URL input field of a REST API client, append `/lambdasv1` to that URL, and then submit a `POST` request to the underlying Lambda function.

Note

You will get the following response.

```
"Hello, world! (v-beta)"
```

To summarize, we have demonstrated how to use API Gateway stage variables to target different HTTP and Lambda back ends for different stages of API deployment. In addition, we also showed how to use the stage variables to pass stage-specific configuration data into HTTP and Lambda back ends. Together, these procedures demonstrate the versatility of the API Gateway stage variables in managing API development.

Amazon API Gateway Stage Variables Reference

You can use API Gateway stage variables in the following cases.

Parameter Mapping Expressions

A stage variable can be used in a parameter mapping expression for an API method's request or response header parameter, without any partial substitution. In the following example, the stage variable is referenced without the `$` and the enclosing `{...}`.

- `stageVariables.<variable_name>`

Mapping Templates

A stage variable can be used anywhere in a mapping template, as shown in the following examples.

- `{ "name" : "$stageVariables.<variable_name>" }`
- `{ "name" : "${stageVariables.<variable_name>}" }`

HTTP Integration URIs

A stage variable can be used as part of an HTTP integration URL, as shown in the following examples.

- A full URI without protocol, e.g., `http://${stageVariables.<variable_name>}`
- A full domain: e.g., `http://${stageVariables.<variable_name>}/resource/operation`
- A subdomain: e.g., `http://${stageVariables.<variable_name>}.example.com/resource/operation`
- A path, e.g., `http://example.com/${stageVariables.<variable_name>}/bar`
- A query string, e.g., `http://example.com/foo?q=${stageVariables.<variable_name>}`

AWS Integration URIs

A stage variable can be used as part of AWS URI action or path components, as shown in the following example.

- `arn:aws:apigateway:<region>:<service>:${stageVariables.<variable_name>}`

AWS Integration URIs (Lambda Functions)

A stage variable can be used in place of a Lambda function name, or version/alias, as shown in the following examples.

- `arn:aws:apigateway:<region>:lambda:path/2015-03-31/functions/arn:aws:lambda::<account_id>:function:${stageVariables.<function_variable_name>}/invocations`
- `arn:aws:apigateway:<region>:lambda:path/2015-03-31/functions/arn:aws:lambda::<account_id>:function:<function_name>:${stageVariables.<version_variable_name>}/invocations`

AWS Integration Credentials

A stage variable can be used as part of AWS user/role credential ARN, as shown in the following example.

- `arn:aws:iam::<account_id>:${stageVariables.<variable_name>}`

Generate and Use an SDK for an API in API Gateway

To call your API in a platform- and language-specific fashion, you must generate a platform- and language-specific SDK for the API. Currently, API Gateway supports generating SDK for an API, deployed to a specific stage, in JavaScript, in Java for Android, and in Objective-C or Swift for iOS.

This section explains how to generate an SDK for an API Gateway API and demonstrates how to use the generated SDK in platform/language-specific apps.

To facilitate the discussions, we will reference this API Gateway [API \(p. 257\)](#) that exposes this [Simple Calculator \(p. 255\)](#) Lambda function.

Before proceeding further, make sure you have created or imported the API and deployed it at least once in API Gateway. For the instructions, see [Deploying an API \(p. 230\)](#).

Topics

- [Use the API Gateway Console to Generate the SDKs of an API \(p. 254\)](#)
- [Use an Android SDK generated by API Gateway \(p. 266\)](#)
- [Use iOS SDK Generated by API Gateway in Objective-C or Swift \(p. 268\)](#)
- [Use a JavaScript SDK Generated by API Gateway \(p. 278\)](#)

Use the API Gateway Console to Generate the SDKs of an API

To generate a platform/language-specific SDK of an API in API Gateway, you must have created the API, tested it, and deployed it in a stage. If you update an API, the generated SDK does not include the updates unless the API is redeployed.

To generate the SDKs of an API in API Gateway

1. Sign in to the API Gateway console at <https://console.aws.amazon.com/apigateway>.
2. In the box that contains the name of the API for the stage, choose **Stages**.
3. In the **Stages** pane, choose the name of the stage.
4. On the **SDK Generation** tab, for **Platform**, choose the platform.
 - a. For **Android**, specify the following:
 - For **Group ID**, type the unique identifier for the corresponding project. This is used in the `pom.xml` file (for example, `com.mycompany`).
 - For **Invoker package**, type the namespace for the generated client classes (for example, `com.mycompany.clientsdk`).
 - For **Artifact ID**, type the name of the compiled `.jar` file without the version. This is used in the `pom.xml` file (for example, `aws-apigateway-api-sdk`).

- For **Artifact version**, type the artifact version number for the generated client. This is used in the `pom.xml` file and should follow a *major.minor.patch* pattern (for example, `1.0.0`).
- b. For **iOS (Objective-C)** or **iOS (Swift)**, type a unique prefix in the **Prefix** box. The effect of prefix is as follows: if you assign, for example, `SIMPLE_CALC` as the prefix for the SDK of the [SimpleCalc \(p. 257\)](#) API with `Input`, `Output`, and `Result` models, the generated SDK will contain the `SIMPLE_CALCSimpleCalcClient` class that encapsulates the API, including the method requests/responses. In addition, the generated SDK will contain the `SIMPLE_CALCInput`, `SIMPLE_CALCOutput`, and `SIMPLE_CALCResult` classes to represent the input, output, and results, respectively, to represent the request input and response output. For more information, see [Use iOS SDK Generated by API Gateway in Objective-C or Swift \(p. 268\)](#).
5. Choose **Generate SDK**, and then follow the on-screen directions to download the SDK generated by API Gateway.
 6. Do one of the following:
 - If you chose **Android** for **Platform**, follow the instructions in [Use an Android SDK generated by API Gateway \(p. 266\)](#).
 - If you chose **iOS** for **Platform**, follow the instructions in [Use iOS SDK Generated by API Gateway in Objective-C or Swift \(p. 268\)](#).
 - If you chose **JavaScript** for **Platform**, follow the instructions in [Use a JavaScript SDK Generated by API Gateway \(p. 278\)](#).

Next, we show how to use the generated SDK to call the underlying API. To put the discussions in context, we will make reference the following example API and its SDKs.

Topics

- [Simple Calculator Lambda Function \(p. 255\)](#)
- [Simple Calculator API in API Gateway \(p. 257\)](#)
- [Simple Calculator API Swagger Definition \(p. 261\)](#)

Simple Calculator Lambda Function

As an illustration, we will use a Node.js Lambda function that performs the binary operations of addition, subtraction, multiplication and division.

Topics

- [Simple Calculator Lambda Function Input Format \(p. 255\)](#)
- [Simple Calculator Lambda Function Output Format \(p. 255\)](#)
- [Simple Calculator Lambda Function Implementation \(p. 256\)](#)
- [Create the Simple Calculator Lambda Function \(p. 256\)](#)

Simple Calculator Lambda Function Input Format

This function takes an input of the following format:

```
{ "a": "Number", "b": "Number", "op": "string" }
```

where `op` can be any of `(+, -, *, /, add, sub, mul, div)`.

Simple Calculator Lambda Function Output Format

When an operation succeeds, it returns the result of the following format:

```
{ "a": "Number", "b": "Number", "op": "string", "c": "Number" }
```

where `c` holds the result of the calculation.

Simple Calculator Lambda Function Implementation

The implementation of the Lambda function is as follows:

```
console.log('Loading the Calc function');

exports.handler = function(event, context) {
  console.log('Received event:', JSON.stringify(event, null, 2));
  if (event.a === undefined || event.b === undefined || event.op ===
  undefined) {
    context.fail("400 Invalid Input");
  }

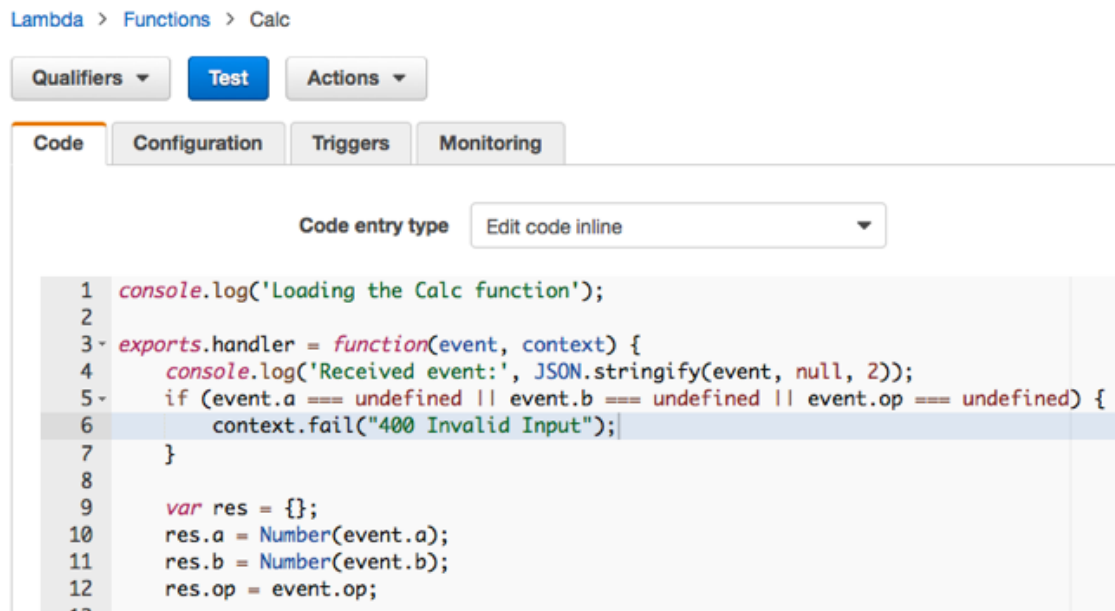
  var res = {};
  res.a = Number(event.a);
  res.b = Number(event.b);
  res.op = event.op;

  if (isNaN(event.a) || isNaN(event.b)) {
    context.fail("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":
      res.c = res.b===0 ? NaN : Number(event.a) / Number(event.b);
      break;
    default:
      context.fail("400 Invalid Operator");
      break;
  }
  context.succeed(res);
};
```

Create the Simple Calculator Lambda Function

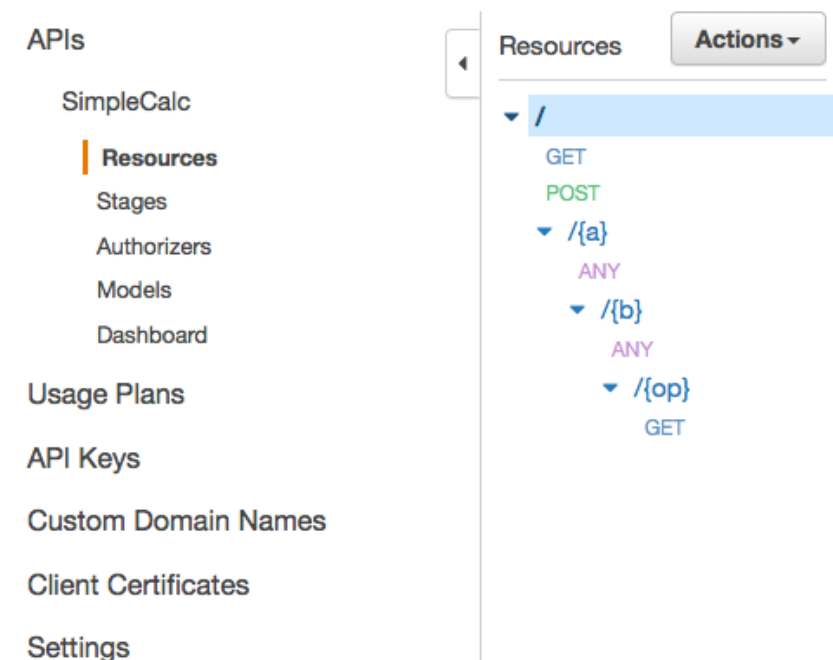
You can use the AWS Lambda console at <https://console.aws.amazon.com/lambda/> to create the function, pasting the above code listing into the online code editor as follows.



```
1 console.log('Loading the Calc function');
2
3 exports.handler = function(event, context) {
4   console.log('Received event:', JSON.stringify(event, null, 2));
5   if (event.a === undefined || event.b === undefined || event.op === undefined) {
6     context.fail("400 Invalid Input");
7   }
8
9   var res = {};
10  res.a = Number(event.a);
11  res.b = Number(event.b);
12  res.op = event.op;
```

Simple Calculator API in API Gateway

Our simple calculator API exposes three methods (GET, POST, GET) to invoke the [Simple Calculator Lambda Function \(p. 255\)](#) (Calc). A graphical representation of this API is shown as follows:



These three methods show different ways to supply the input for the back-end Lambda function to perform the same operation:

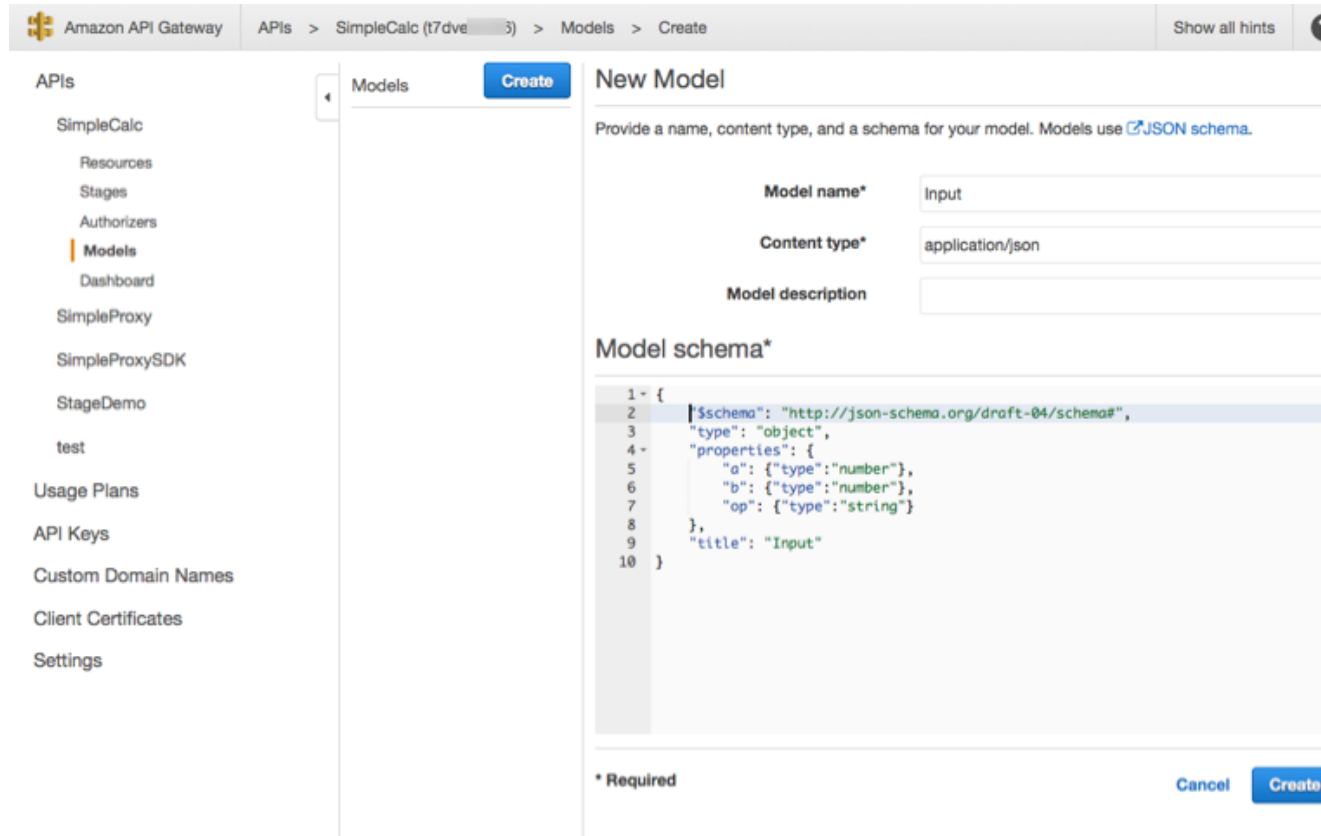
- The GET `/?a=...&b=...&op=...` method uses the query parameters to specify the input.
- The POST `/?` method uses a JSON payload of `{"a": "Number", "b": "Number", "op": "string"}` to specify the input.

- The `GET /{a}/{b}/{op}` method uses the path parameters to specify the input.

Before showing how to call these methods using an SDK generated by API Gateway for this API, let's recall briefly how to set them up. For detailed instructions, see [Creating an API \(p. 70\)](#). If you're new to API Gateway, see [Getting Started \(p. 4\)](#) first.

Create Models for Input and Output

To specify strongly typed input in the SDK, we create an `Input` model for the API:



Similarly, to describe the response body data type, we create the following models in the API Gateway:

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "type": "object",
  "properties": {
    "c": {"type": "number"}
  },
  "title": "Output"
}
```

and

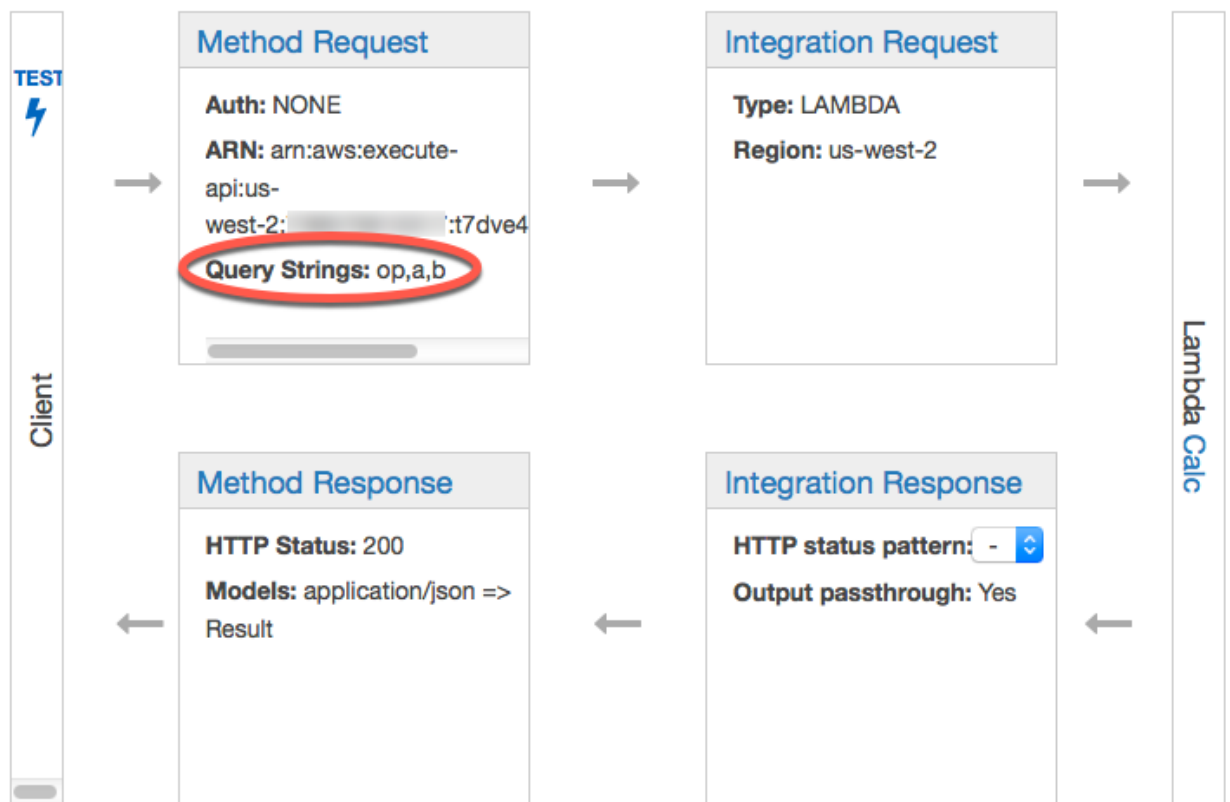
```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "type": "object",
```

```
"properties":{
  "input":{
    "$ref":"https://apigateway.amazonaws.com/restapis/t7dve4zn36/
models/Input"
  },
  "output":{
    "$ref":"https://apigateway.amazonaws.com/restapis/t7dve4zn36/
models/Output"
  }
},
"title":"Result"
}
```

Set Up GET / Method Query Parameters

For the GET /?a=..&b=..&op=.. method, the query parameters are declared in **Method Request**:

/ - GET - Method Execution



Set Up Data Model for the Payload as Input to the Back End


For the POST / method, we create the Input model and add it to the method request to define the shape of input data.

[← Method Execution](#) / - POST - Method Request

Provide information about this method's authorization settings and the parameters it can receive.

Authorization Settings


Authorization NONE  

API Key Required false 

▶ **URL Query String Parameters**

▶ **HTTP Request Headers**

▼ **Request Models** [Create a Model](#)

Content type	Model name	
application/json	Input	

 [Add model](#)

With this model, your API customers can parse a successful output by reading properties of a `Result` object. Without this model, customers would be required to create dictionary object to represent the JSON output.

Set Up Data Model for the Result Output from the Back End

For all three methods, we create the `Result` model and add it to the method's `Method Response` to define the shape of output returned by the Lambda function.

← Method Execution /{a}/{b}/{op} - GET - Method Response

Provide information about this method's response types, their headers and content types.

HTTP Status
200

Response Headers for 200

Name
No headers

[+ Add Header](#)

Response Models for 200

[Create a model](#)

Content type	Models
application/json	Result

[+ Add Response Model](#)

[+ Add Response](#)

With this model, your API customers can call the SDK to specify the input by instantiating an `Input` object. Without this model, your customers would be required to create dictionary object to represent the JSON input to the Lambda function.

In addition, you can also create and set up the API following the Swagger API [definitions](#) (p. 261).

Simple Calculator API Swagger Definition

The following is the Swagger definition of the simple calculator API. You can import it into your account. However, you need to reset the resource-based permissions on the [Lambda function](#) (p. 255) after the import. To do so, re-select the Lambda function that you created in your account from the **Integration Request** in the API Gateway console. This will cause the API Gateway console to reset the required permissions. Alternatively, you can use AWS Command Line Interface for Lambda command of [add-permission](#).

```
{
  "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\n\n  \"a\" : $input.params('a'),\n  \"b\" : $input.params('b'),\n  \"op\" :\n  \"$input.params('op')\"\n}\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",
    "responses": {
      "default": {
        "statusCode": "200",
        "responseTemplates": {
          "application/json": "#set($inputRoot = $input.path('$'))\n{\n\n\n  \"input\" : {\n    \"a\" : $inputRoot.a,\n    \"b\" : $inputRoot.b,\n    \"op\" : \"$inputRoot.op\"\n  },\n  \"output\" : {\n    \"c\" : $inputRoot.c\n  }\n}\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
\n  \"input\" : {\n    \"a\" : $inputRoot.a,\n    \"b\" : $inputRoot.b,\n
\n  \"op\" : \"$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}\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\" : $inputRoot.c\n  }\n}\n}\n}\n},\n\"type\": \"aws\"\n}\n}\n},\n\"definitions\": {\n  \"Input\": {\n    \"type\": \"object\",\n    \"properties\": {\n      \"a\": {\n        \"type\": \"number\"\n      },\n      \"b\": {\n        \"type\": \"number\"\n      },\n      \"op\": {\n        \"type\": \"string\"\n      }\n    },\n    \"title\": \"Input\"\n  },\n  \"Output\": {\n    \"type\": \"object\",\n    \"properties\": {\n      \"c\": {\n        \"type\": \"number\"\n      }\n    },\n    \"title\": \"Output\"\n  },\n  \"Result\": {\n    \"type\": \"object\",\n    \"properties\": {\n      \"input\": {\n        \"$ref\": \"#/definitions/Input\"\n      },\n      \"output\": {\n        \"$ref\": \"#/definitions/Output\"\n      }\n    },\n    \"title\": \"Result\"\n  }\n}\n}
```

Use an Android SDK generated by API Gateway

In this section, we will outline the steps to use an Android SDK generated by API Gateway of an API. Before proceeding further, you must have already completed the steps in [Use the API Gateway Console to Generate the SDKs of an API \(p. 254\)](#).

To install and use an Android SDK Generated by API Gateway

1. Extract the contents of the API Gateway-generated .zip file that you downloaded earlier.
2. Download and install [Apache Maven](#) (preferably version 3.x).
3. Download and install the [JDK](#) (preferably version 1.7 or later).
4. Set the `JAVA_HOME` environment variable.
5. Run the `mvn install` command to install the compiled artifact files to your local Maven repository. This creates a `target` folder containing the compiled SDK library.
6. Copy the SDK file (the name of which is derived from the **Artifact Id** and **Artifact Version** you specified when generating the SDK, e.g., `simple-calcsdk-1.0.0.jar`) from the `target` folder, along with all of the other libraries from the `target/lib` folder, into your project's `lib` folder.

If you use Android Studio, create a `libs` folder under your client app module and copy the required .jar file into this folder. Verify that the dependencies section in the module's gradle file contains the following.

```
compile fileTree(include: ['*.jar'], dir: 'libs')
compile fileTree(include: ['*.jar'], dir: 'app/libs')
```

Make sure no duplicated .jar files are declared.

7. Use the `ApiClientFactory` class to initialize the API Gateway-generated SDK. For example:

```
ApiClientFactory factory = new ApiClientFactory();

// Create an instance of your SDK.
final MyApiClient client = factory.build(MyApiClient.class);

// Invoke a method:
//     If the API exposes a 'GET /?a=1&b=2&op=+' method, you can call the
//     following SDK method:
//
Result output = client.rootGet("1", "2", "+");

//     where Result is the SDK class of the like-named API model.
//
//     If the API exposes a 'GET /{a}/{b}/{op}' method, 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.

//     You can call the following SDK methods invoke this POST API method:
//     POST /
//     host: ...
//     Content-Type: application/json
//
//     { "a": 1, "b": 2, "op": "+" }
Input body = new Input();
input.a=1;
input.b=2;
input.op="+";
Result output = client.rootPost(body);

// 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. To use an Amazon Cognito credentials provider to authorize calls to your API, use the `ApiClientFactory` class to pass a set of AWS credentials by using the SDK generated by API Gateway, as shown in the following example.

```
// 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. To set an API key by using the API Gateway- generated SDK, use code similar to the following.

```
ApiClientFactory factory = new ApiClientFactory()  
    .apiKey("YOUR_API_KEY");
```

Use iOS SDK Generated by API Gateway in Objective-C or Swift

In this tutorial, we will show how to use an iOS SDK generated by API Gateway in an Objective-C or Swift app to call the underlying API. We will use the [SimpleCalc API \(p. 257\)](#) as an example to illustrate the following topics:

- How to install the required AWS Mobile SDK components into your Xcode project
- How to create the API client object before calling the API's methods
- How to call the API methods through the corresponding SDK methods on the API client object
- How to prepare a method input and parse its result using the corresponding model classes of the SDK

Topics

- [Use an iOS SDK generated by API Gateway in an Objective-C App \(p. 269\)](#)
- [Use an iOS SDK generated by API Gateway in a Swift App \(p. 273\)](#)

Use an iOS SDK generated by API Gateway in an Objective-C App

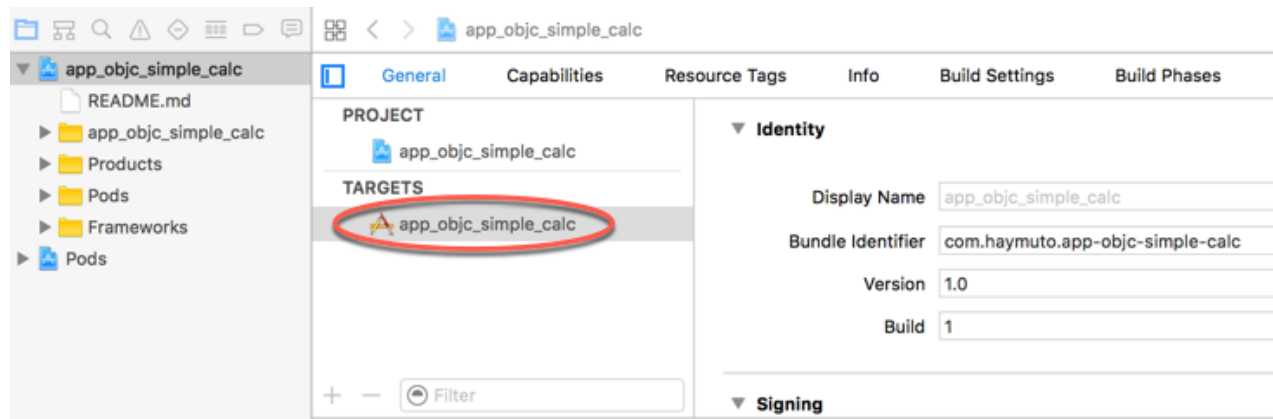
Before beginning the following procedure, you must complete the steps in [Use the API Gateway Console to Generate the SDKs of an API \(p. 254\)](#) for iOS in Objective-C and download the .zip file of the generated SDK.

Install the AWS Mobile SDK and an iOS SDK generated by API Gateway in an Objective-C Project

The following procedure describes how to install the SDK.

To install and use an iOS SDK generated by API Gateway in Objective-C

1. Extract the contents of the API Gateway-generated .zip file you downloaded earlier. Using the [SimpleCalc API \(p. 257\)](#), you may want to rename the unzipped SDK folder to something like `sdk_objc_simple_calc`. In this SDK folder there is a `README.md` file and a `Podfile` file. The `README.md` file contains the instructions to install and use the SDK. This tutorial provides details about these instructions. The installation leverages [CocoaPods](#) to import required API Gateway libraries and other dependent AWS Mobile SDK components. You must update the `Podfile` to import the SDKs into your app's Xcode project. The unarchived SDK folder also contains a `generated-src` folder that contains the source code of the generated SDK of your API.
2. Launch Xcode and create a new iOS Objective-C project. Make a note of the project's target. You will need to set it in the `Podfile`.



3. To import the AWS Mobile SDK for iOS into the Xcode project by using CocoaPods, do the following:
 1. Install CocoaPods by running the following command in a terminal window:

```
sudo gem install cocoapods
pod setup
```

2. Copy the `Podfile` file from the extracted SDK folder into the same directory containing your Xcode project file. Replace the following block:

```
target '<YourXCodeTarget>' do
  pod 'AWSAPIGateway', '~> 2.4.7'
end
```

with your project's target name:

```
target 'app_objc_simple_calc' do
  pod 'AWSAPIGateway', '~> 2.4.7'
end
```

If your Xcode project already contains a file named `Podfile`, add the following line of code to it:

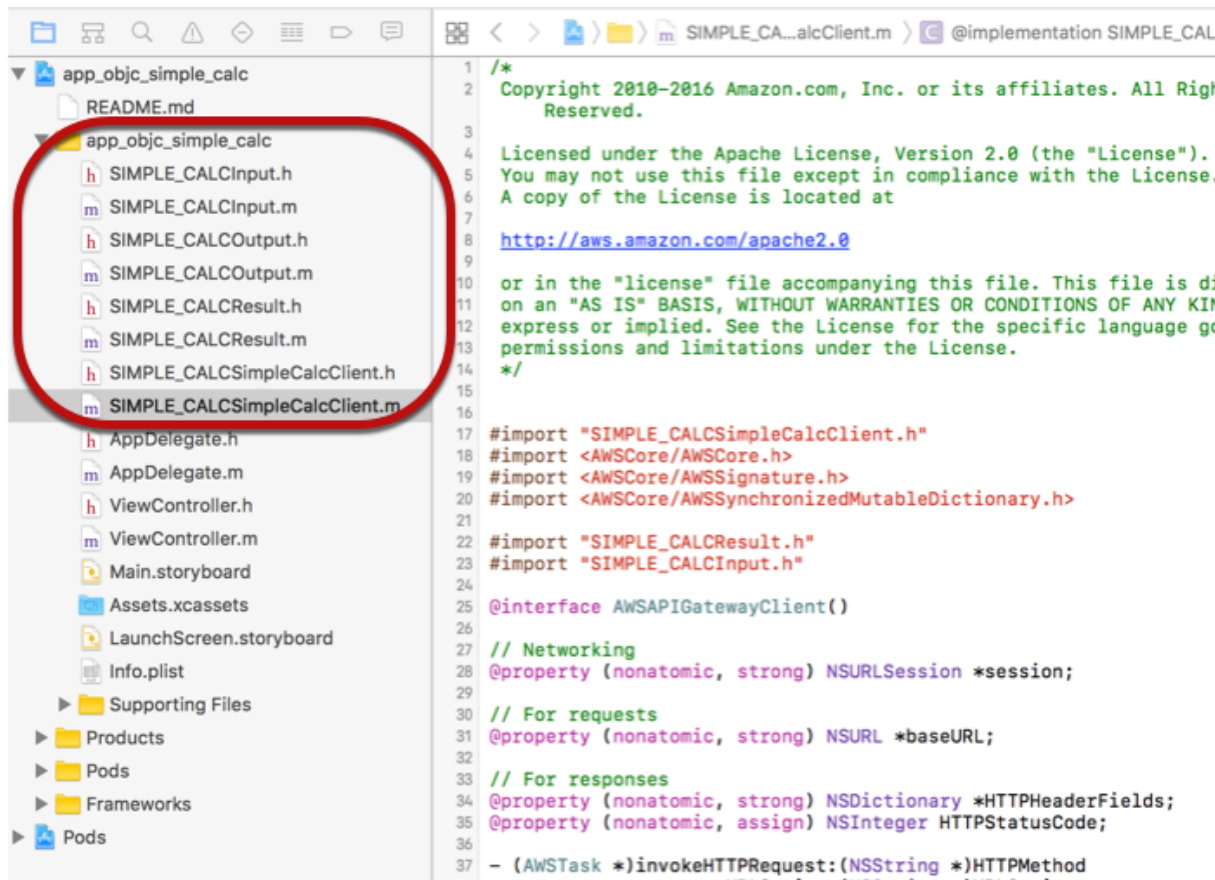
```
pod 'AWSAPIGateway', '~> 2.4.7'
```

3. Open a terminal window and run the following command:

```
pod install
```

This installs the API Gateway component and other dependent AWS Mobile SDK components.

4. Close the Xcode project and then open the `.xcworkspace` file to relaunch Xcode.
5. Add all of the `.h` and `.m` files from the extracted SDK's `generated-src` directory into your Xcode project.



To import the AWS Mobile SDK for iOS Objective-C into your project by explicitly downloading AWS Mobile SDK or using [Carthage](#), follow the instructions in the `README.md` file. Be sure to use only one of these options to import the AWS Mobile SDK.

Call API Methods Using the iOS SDK generated by API Gateway in an Objective-C Project

When you generated the SDK with the prefix of `SIMPLE_CALC` for this [SimpleCalc API \(p. 257\)](#) with two models for input (`Input`) and output (`Result`) of the methods, in the SDK, the resulting API client class becomes `SIMPLE_CALCSimpleCalcClient` and the corresponding data classes are `SIMPLE_CALCInput` and `SIMPLE_CALCResult`, respectively. The API requests and responses are mapped to the SDK methods as follows:

- The API request of

```
GET /?a=...&b=...&op=...
```

becomes the SDK method of

```
(AWSTask *)rootGet:(NSString *)op a:(NSString *)a b:(NSString *)b
```

The `AWSTask.result` property is of the `SIMPLE_CALCResult` type if the `Result` model was added to the method response. Otherwise, the property is of the `NSDictionary` type.

- This API request of

```
POST /  
  
{  
  "a": "Number",  
  "b": "Number",  
  "op": "String"  
}
```

becomes the SDK method of

```
(AWSTask *)rootPost:(SIMPLE_CALCInput *)body
```

- The API request of

```
GET /{a}/{b}/{op}
```

becomes the SDK method of

```
(AWSTask *)aBOPGet:(NSString *)a b:(NSString *)b op:(NSString *)op
```

The following procedure describes how to call the API methods in Objective-C app source code; for example, as part of the `viewDidLoad` delegate in a `ViewController.m` file.

To call the API through the iOS SDK generated by API Gateway

1. Import the API client class header file to make the API client class callable in the app:

```
#import "SIMPLE_CALCSimpleCalc.h"
```

The `#import` statement also imports `SIMPLE_CALCInput.h` and `SIMPLE_CALCResult.h` for the two model classes.

2. Instantiate the API client class:

```
SIMPLE_CALCSimpleCalcClient *apiInstance = [SIMPLE_CALCSimpleCalcClient
defaultClient];
```

To use Amazon Cognito with the API, set the `defaultServiceConfiguration` property on the default `AWSServiceManager` object, as shown in the following, before calling the `defaultClient` method to create the API client object (shown in the preceding example):

```
AWSCognitoCredentialsProvider *creds = [[AWSCognitoCredentialsProvider
alloc] initWithRegionType:AWSRegionUSEast1
identityPoolId:your_cognito_pool_id];
AWSServiceConfiguration *configuration = [[AWSServiceConfiguration alloc]
initWithRegion:AWSRegionUSEast1 credentialsProvider:creds];
AWSServiceManager.defaultServiceManager.defaultServiceConfiguration =
configuration;
```

3. Call the `GET /?a=1&b=2&op=+` method to perform 1+2:

```
[[apiInstance rootGet:@"+" a:@"1" b:@"2"] continueWithBlock:^id
_Nullable(AWSTask * _Nonnull task) {
    _textField1.text = [self handleApiResponse:task];
    return nil;
}];
```

where the helper function `handleApiResponse:task` formats the result as a string to be displayed in a text field (`_textField1`).

```
- (NSString *)handleApiResponse:(AWSTask *)task {
    if (task.error != nil) {
        return [NSString stringWithFormat:@"Error: %@",
task.error.description];
    } else if (task.result != nil && [task.result isKindOfClass:
SIMPLE_CALCResult class]) {
        return [NSString stringWithFormat:@"%@ %@ %@ = %@
\n",task.result.input.a, task.result.input.op, task.result.input.b,
task.result.output.c];
    }
    return nil;
}
```

The resulting display is `1 + 2 = 3`.

4. Call the `POST /` with a payload to perform 1-2:

```
SIMPLE_CALCInput *input = [[SIMPLE_CALCInput alloc] init];
input.a = [NSNumber numberWithInt:1];
input.b = [NSNumber numberWithInt:2];
input.op = @"-";
[[apiInstance rootPost:input] continueWithBlock:^id _Nullable(AWSTask
* _Nonnull task) {
    _textField2.text = [self handleApiResponse:task];
    return nil;
}];
```

The resulting display is `1 - 2 = -1`.

5. Call the `GET /{a}/{b}/{op}` to perform `1/2`:

```
[[apiInstance aBopGet:@"1" b:@"2" op:@"div"] continueWithBlock:^id  
_Nullable(AWSTask * _Nonnull task) {  
    _textField3.text = [self handleApiResponse:task];  
    return nil;  
}];
```

The resulting display is `1 div 2 = 0.5`. Here, `div` is used in place of `/` because the [simple Lambda function \(p. 255\)](#) in the back end does not handle URL encoded path variables.

Use an iOS SDK generated by API Gateway in a Swift App

Before beginning the following procedure, you must complete the steps in [Use the API Gateway Console to Generate the SDKs of an API \(p. 254\)](#) for iOS in Swift and download the .zip file of the generated SDK.

Topics

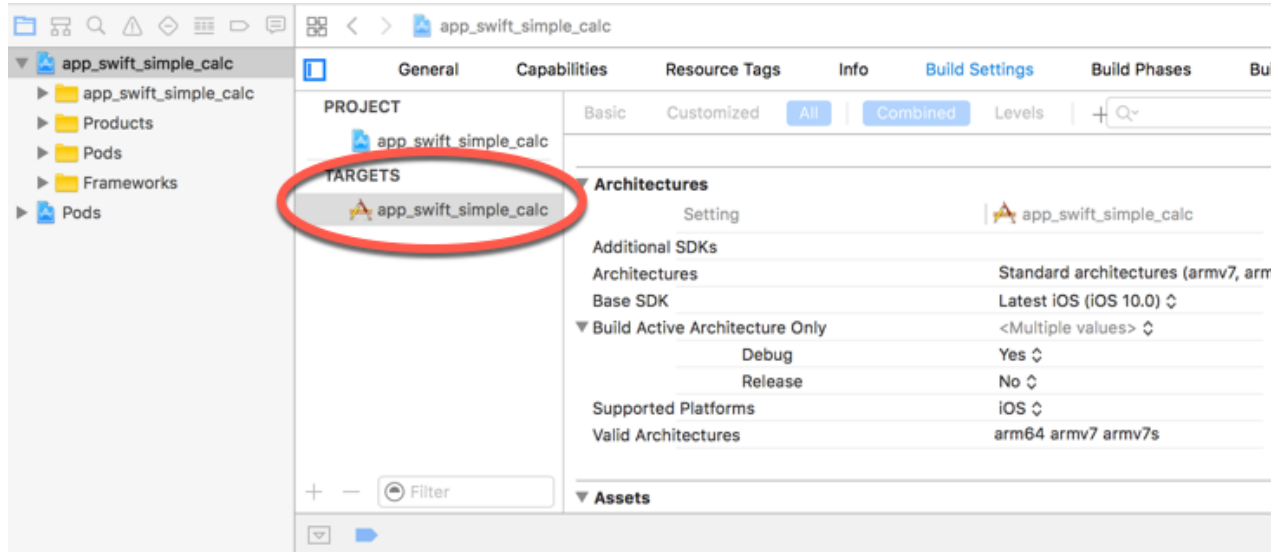
- [Install AWS Mobile SDK and API Gateway-Generated SDK in a Swift Project \(p. 273\)](#)
- [Call API methods through the iOS SDK generated by API Gateway in a Swift Project \(p. 276\)](#)

Install AWS Mobile SDK and API Gateway-Generated SDK in a Swift Project

The following procedure describes how to install the SDK.

To install and use an iOS SDK generated by API Gateway in Swift

1. Extract the contents of the API Gateway-generated .zip file you downloaded earlier. Using the [SimpleCalc API \(p. 257\)](#), you may want to rename the unzipped SDK folder to something like `sdk_swift_simple_calc`. In this SDK folder there is a `README.md` file and a `Podfile` file. The `README.md` file contains the instructions to install and use the SDK. This tutorial provides details about these instructions. The installation leverages [CocoaPods](#) to import required AWS Mobile SDK components. You must update the `Podfile` to import the SDKs into your Swift app's XCode project. The unarchived SDK folder also contains a `generated-src` folder that contains the source code of the generated SDK of your API.
2. Launch Xcode and create a new iOS Swift project. Make a note of the project's target. You will need to set it in the `Podfile`.



3. To import the required AWS Mobile SDK components into the Xcode project by using CocoaPods, do the following:
 1. If it is not installed, install CocoaPods by running the following command in a terminal window:

```
sudo gem install cocoapods  
pod setup
```

2. Copy the `Podfile` file from the extracted SDK folder into the same directory containing your Xcode project file. Replace the following block:

```
target '<YourXCodeTarget>' do  
  pod 'AWSAPIGateway', '~> 2.4.7'  
end
```

with your project's target name as shown:

```
target 'app_swift_simple_calc' do  
  pod 'AWSAPIGateway', '~> 2.4.7'  
end
```

If your Xcode project already contains a `Podfile` with the correct target, you can simply add the following line of code to the `do ... end` loop:

```
pod 'AWSAPIGateway', '~> 2.4.7'
```

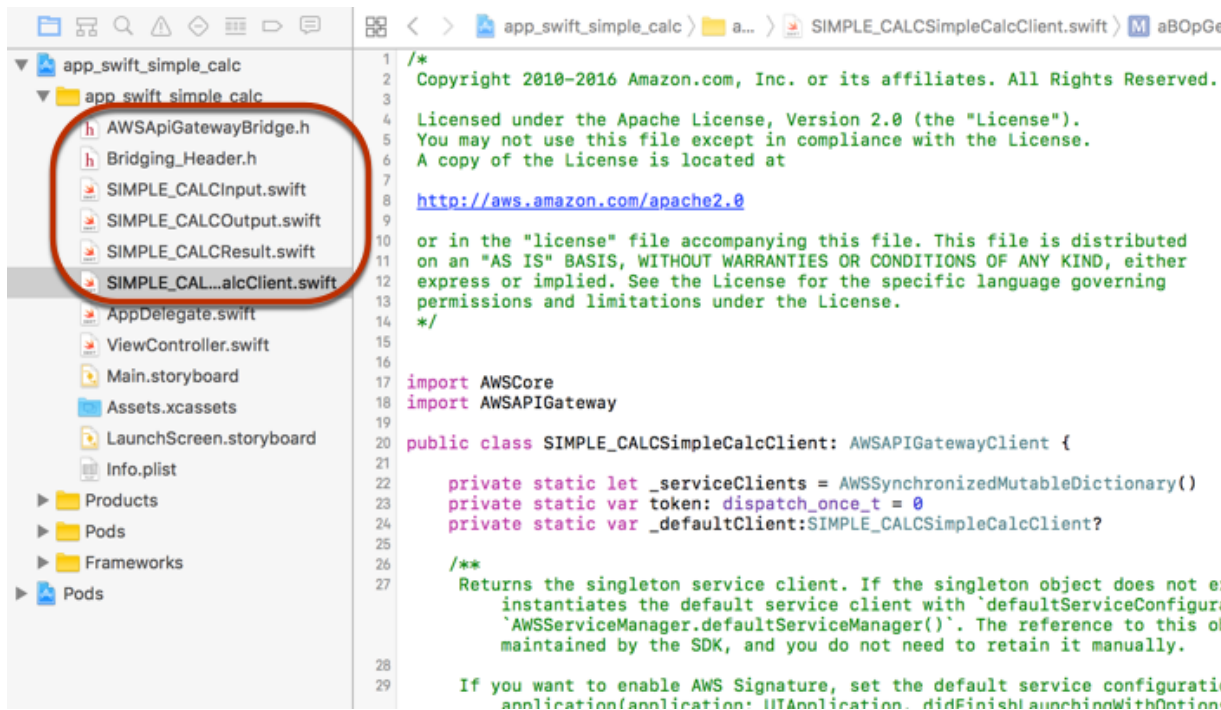
3. Open a terminal window and run the following command in the app directory:

```
pod install
```

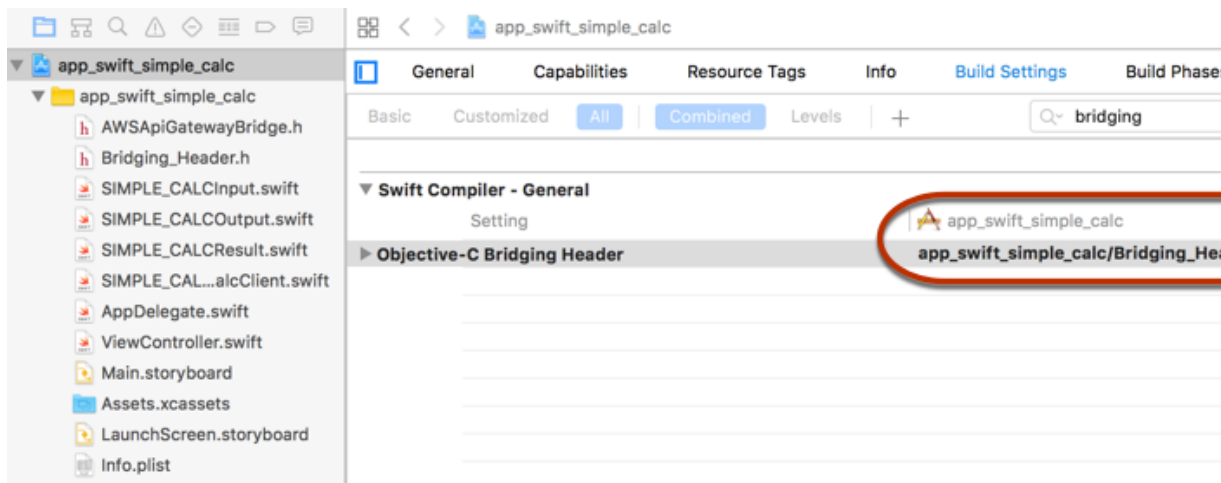
This installs the API Gateway component and any dependent AWS Mobile SDK components into the app's project.

4. Close the Xcode project and then open the `*.xcworkspace` file to relaunch Xcode.

5. Add all of the SDK's header files (.h) and Swift source code files (.swift) from the extracted generated-src directory to your Xcode project.



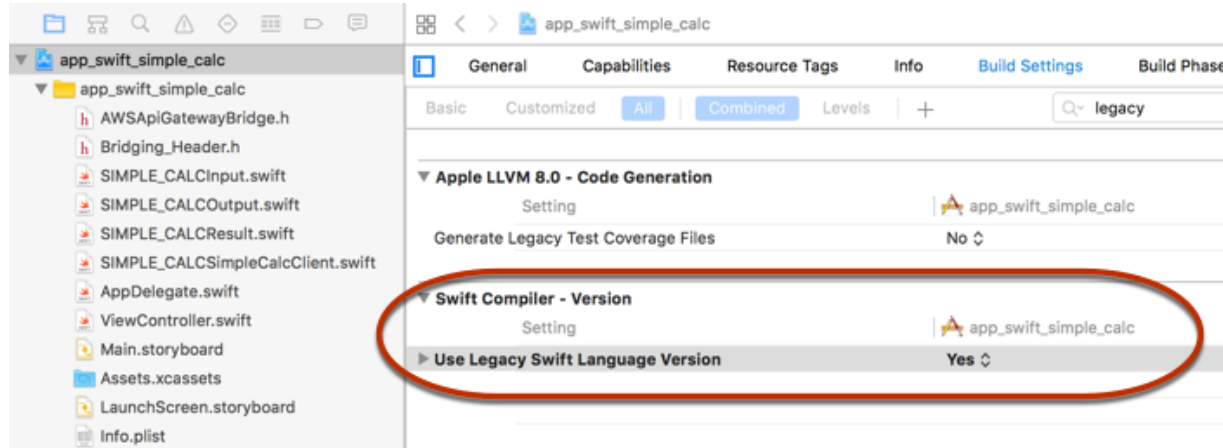
6. To enable calling the Objective-C libraries of the AWS Mobile SDK from your Swift code project, set the Bridging_Header.h file path on the **Objective-C Bridging Header** property under the **Swift Compiler - General** setting of your Xcode project configuration:



Tip

You can type `bridging` in the search box of Xcode to locate the **Objective-C Bridging Header** property.

7. Build the Xcode project to verify that it is properly configured before proceeding further. If your Xcode uses a more recent version of Swift than the one supported for the AWS Mobile SDK, you will get Swift compiler errors. In this case, set the **Use Legacy Swift Language Version** property to **Yes** under the **Swift Compiler - Version** setting:



To import the AWS Mobile SDK for iOS in Swift into your project by explicitly downloading the AWS Mobile SDK or using [Carthage](#), follow the instructions in the `README.md` file that comes with the SDK package. Be sure to use only one of these options to import the AWS Mobile SDK.

Call API methods through the iOS SDK generated by API Gateway in a Swift Project

When you generated the SDK with the prefix of `SIMPLE_CALC` for this [SimpleCalc API](#) (p. 257) with two models to describe the input (`Input`) and output (`Result`) of the API's requests and responses, in the SDK, the resulting API client class becomes `SIMPLE_CALCSimpleCalcClient` and the corresponding data classes are `SIMPLE_CALCInput` and `SIMPLE_CALCResult`, respectively. The API requests and responses are mapped to the SDK methods as follows:

- The API request of

```
GET /?a=...&b=...&op=...
```

becomes the SDK method of

```
public func rootGet(op: String?, a: String?, b: String?) -> AWSTask
```

The `AWSTask.result` property is of the `SIMPLE_CALCResult` type if the `Result` model was added to the method response. Otherwise, it is of the `NSDictionary` type.

- This API request of

```
POST /  
  
{  
  "a": "Number",  
  "b": "Number",  
  "op": "String"  
}
```

becomes the SDK method of


```
public func rootPost(body: SIMPLE_CALCInput) -> AWSTask
```

- The API request of

```
GET /{a}/{b}/{op}
```

becomes the SDK method of

```
public func aBopGet(a: String, b: String, op: String) -> AWSTask
```

The following procedure describes how to call the API methods in Swift app source code; for example, as part of the `viewDidLoad()` delegate in a `ViewController.m` file.

To call the API through the iOS SDK generated by API Gateway

1. Instantiate the API client class:

```
let client = SIMPLE_CALCSimpleCalcClient.defaultClient()
```

To use Amazon Cognito with the API, set a default AWS service configuration (shown following) before getting the `defaultClient` method (shown previously):

```
let credentialsProvider =  
    AWSCognitoCredentialsProvider(regionType: AWSRegionType.USEast1,  
    identityPoolId: "my_pool_id")  
let configuration = AWSServiceConfiguration(region: AWSRegionType.USWest2,  
    credentialsProvider: credentialsProvider)  
AWSServiceManager.defaultServiceManager().defaultServiceConfiguration =  
    configuration
```

2. Call the `GET /?a=1&b=2&op=+` method to perform 1+2:

```
client.rootGet("+", a: "1", b:"2").continueWithBlock {(task: AWSTask) ->  
    AnyObject? in  
    self.showResult(task)  
    return nil  
}
```

where the helper function `self.showResult(task)` prints the result or error to the console; for example:

```
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)")  
        }  
    }  
}
```

```
    }  
  }  
}
```

In a production app, you can display the result or error in a text field. The resulting display is $1 + 2 = 3$.

3. Call the `POST /` with a payload to perform $1-2$:

```
let body = SIMPLE_CALCInput()  
body.a=1  
body.b=2  
body.op="-"  
client.rootPost(body).continueWithBlock {(task: AWSTask) -> AnyObject? in  
  self.showResult(task)  
  return nil  
}
```

The resultant display is $1 - 2 = -1$.

4. Call the `GET /{a}/{b}/{op}` to perform $1/2$:

```
client.aBOPGet("1", b:"2", op:"div").continueWithBlock {(task: AWSTask) ->  
  AnyObject? in  
  self.showResult(task)  
  return nil  
}
```

The resulting display is $1 \text{ div } 2 = 0.5$. Here, `div` is used in place of `/` because the [simple Lambda function \(p. 255\)](#) in the back end does not handle URL encoded path variables.

Use a JavaScript SDK Generated by API Gateway

Note

These instructions assume you have already completed the instructions in [Use the API Gateway Console to Generate the SDKs of an API \(p. 254\)](#).

1. Extract the contents of the API Gateway-generated .zip file you downloaded earlier.
2. Enable cross-origin resource sharing (CORS) for all of the methods the SDK generated by API Gateway will call. For instructions, see [Enable CORS for a Resource \(p. 170\)](#).
3. In your web page, include references to the following scripts.

```
<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. In your code, initialize the SDK generated by API Gateway by using code similar to the following.

```
var apigClient = apigClientFactory.newClient();
```

5. Call the API methods in API Gateway by using code similar to the following. Each call returns a promise with a success and failure callbacks.

```
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.  
  });
```

Here, the *methodName* is constructed from the method request's resource path and the HTTP. For example, for the GET / request, the *methodName* is rootGet. For the GET /a/b/op request, the *methodName* will be aBOPGet.

6. To initialize the SDK generated by API Gateway with AWS credentials, use code similar to the following. If you use AWS credentials, all requests to the API will be signed. This means you must set the appropriate CORS Accept headers for each request.

```
var apigClient = apigClientFactory.newClient({  
  accessKey: 'ACCESS_KEY',  
  secretKey: 'SECRET_KEY',
```

```
});
```

7. To use an API key with the SDK generated by API Gateway, you can pass the API key as a parameter to the `Factory` object by using code similar to the following. If you use an API key, it is specified as part of the `x-api-key` header and all requests to the API will be signed. This means you must set the appropriate CORS Accept headers for each request.

```
var apigClient = apigClientFactory.newClient({  
    apiKey: 'API_KEY'  
});
```

Use a Custom Domain Name in API Gateway

After deploying your API, you (and the client) can invoke the API using the default root URL of the `https://api-id.execute-api.region.amazonaws.com` format. To provide a simpler and more intuitive URL for your API users, you can use API Gateway to set up a custom domain name (e.g., `api.example.com`) and choose a base path (e.g., `myservice`) to present an alternative URL (e.g., `https://api.example.com/myservice`) for the API. You can also use an empty base path for an API. In this case, the API's URL is the same as the custom domain (e.g., `https://api.example.com`).

For every API you create, API Gateway sets up an Amazon CloudFront distribution for the API. Requests with the default API URL are routed to the corresponding CloudFront distribution. Similarly, every custom domain name is backed by a CloudFront distribution. An API request with the custom domain name passes through the custom domain name's CloudFront distribution before reaching the API's CloudFront distribution. API Gateway supports custom domain names for APIs by leveraging Server Name Indication (SNI) on the CloudFront distribution. For more information on using custom domain names on a CloudFront distribution, including the required certificate format and the maximum size of a certificate key length, see [Using Alternate Domain Names and HTTPS](#) in the *Amazon CloudFront Developer Guide*.

Creating a custom domain name for your APIs involves deleting an existing CloudFront distribution and creating a new one. The process may take approximately 30 minutes before the new custom domain name becomes available. For more information, see [Updating CloudFront Distributions](#).

To enable a custom domain name, you, as the API owner, must provide a server-side SSL certificate to verify the custom domain name targeted by the client requests. You do this when setting up the domain name initially and then when renewing an expiring certificate subsequently. In addition, you must have registered the custom domain name with a domain name registrar. After setting up a custom domain name in API Gateway, you must create or update your domain name service (DNS) provider's resource record to map the custom domain name to its CloudFront distribution domain name. For the SSL certificate, you must also have obtained from a certificate authority the PEM-formatted SSL certificate body, its private key, and the certificate chain for the custom domain name. This section describes how to configure a domain name for an API, to set up the certificate for a custom domain name, to map a base path to an API, and to upload a new certificate to replace an expiring one. We will also provide general guidance, by way of examples, on how to obtain the server-side certificate and create a DNS alias record.

Topics

- [Prerequisites \(p. 281\)](#)
- [Set Up a Custom Domain Name for an API Gateway API \(p. 281\)](#)
- [Specify API Mappings for a Custom Domain Name \(p. 284\)](#)
- [Base Path Examples of API Mappings for a Custom Domain Name \(p. 284\)](#)
- [Upload and Renew an Expiring Certificate \(p. 285\)](#)
- [Call Your API with Custom Domain Names \(p. 286\)](#)

Prerequisites

Note

API Gateway, fronted by CloudFront, does not support self-signed SSL certificates.

The following steps describe how to prepare to use custom domain names in API Gateway.

To prepare to use custom domain names in API Gateway

1. Register your custom domain name. See the [Accredited Registrar Directory](#) at the ICANN website.
2. Get a PEM-encoded SSL certificate for your custom domain name from a certificate authority. For a partial list, see [Third-Party Certificate Authorities](#) at the DMOZ website.

Here are the general steps to obtain an SSL certificate from your chosen certificate authority:

- a. Generate a private key for the certificate and save output to a file, using the [OpenSSL](#) toolkit at the OpenSSL website:

```
openssl genrsa -out private-key-file 2048
```

Note

Amazon API Gateway leverages Amazon CloudFront to support certificates for custom domain names. As such, the requirements and constraints of a custom domain name SSL certificate are dictated by CloudFront. For example, the maximum size of the public key is 2048 and the private key size can be 1024, 2048 and 4096. The public key size is determined by the CA you use. Inquire your CA to return keys of a size different from the default length. For more information, see [Secure access to your objects](#) and [Create signed URLs and signed cookies](#).

- b. Generate a certificate signing request (CSR) with the previously generated private key, using OpenSSL:

```
openssl req -new -sha256 -key private-key-file -out CSR-file
```

- c. Submit the CSR to the certificate authority and save the resulting certificate.
- d. Download the certificate chain from the certificate authority.

Note

If you obtain the private key in another way and the key is encrypted, you can use the following command to decrypt the key before submitting it to API Gateway for setting up a custom domain name.

```
openssl pkcs8 -topk8 -inform pem -in MyEncryptedKey.pem -outform pem -nocrypt -out MyDecryptedKey.pem
```

Set Up a Custom Domain Name for an API Gateway API

The following procedure describes how to set up a custom domain name.

To set up a custom domain name for an API Gateway API

1. Sign in to the API Gateway console at <https://console.aws.amazon.com/apigateway>.

2. Choose **Custom Domain Names** from the main navigation pane.
3. Choose **Create** in the secondary navigation pane.
4. In **Create Custom Domain Name**, specify the following:
 - a. For **Domain name**, type your domain name (for example, `api.example.com`).
 - b. For **Certificate name**, type a name for future reference (for example, `my-example-certificate`).
 - c. For **Certificate body**, type or paste the body of the PEM-formatted server certificate from your certificate authority. The following shows an abbreviated example of such a certificate.

```
-----BEGIN CERTIFICATE-----  
EXAMPLECA+KgAwIBAgIQJ1XxJ8P1++gOfQtj0IBoqDANBgkqhkiG9w0BAQUFADBB  
...  
az8CglaiLcXLBQ7EaWIhhgEXAMPLE  
-----END CERTIFICATE-----
```

- d. For **Certificate private key**, type or paste your PEM-formatted certificate's private key. The following shows an abbreviated example of such a key.

```
-----BEGIN RSA PRIVATE KEY-----  
EXAMPLEBAAKCAQEA2Qb3LDHD7StY7Wj6U2/opV6Xu37qUCCKeDWhwpZMYJ9/nETO  
...  
1qGvJ3u04vDnzaYN5WoyN5LFckr1A71+CszD1CGSqBVDWEXAMPLE  
-----END RSA PRIVATE KEY-----
```

- e. For **Certificate chain**, type or paste the PEM-formatted intermediate certificates and, optionally, the root certificate, one after the other without any blank lines. If you include the root certificate, your certificate chain must start with intermediate certificates and end with the root certificate. Use the intermediate certificates provided by your certificate authority. Do not include any intermediaries that are not in the chain of trust path. The following shows an abbreviated example.

```
-----BEGIN CERTIFICATE-----  
EXAMPLECA4ugAwIBAgIQWrYdrB5NogYUx1U9Pamy3DANBgkqhkiG9w0BAQUFADCB  
...  
8/ifBlIK3se2e4/hEfcEejX/arxbx1BJCHBv1EPNnsdw8EXAMPLE  
-----END CERTIFICATE-----
```

Here is another example.

```
-----BEGIN CERTIFICATE-----  
Intermediate certificate 2  
-----END CERTIFICATE-----  
-----BEGIN CERTIFICATE-----  
Intermediate certificate 1  
-----END CERTIFICATE-----  
-----BEGIN CERTIFICATE-----  
Optional: Root certificate  
-----END CERTIFICATE-----
```

5. Choose **Save**.
6. While the new custom domain name is being created, the console displays the following information to have an alias resource record created in your DNS provider to map your custom domain name (`api.example.com`) to the API's CloudFront distribution domain name (`distribution-id.cloudfront.net`).

api.example.com Delete Custom Domain Name

Create an Alias resource record with your DNS provider to map api.example.com to d3boq9ikohtgw.cloudfront.net

Certificate name my-example-cert

Distribution domain name d3boq9ikohtgw.cloudfront.net

The custom domain name is being created. This process can take up to 40 minutes.

API Mappings

Base path	API	Stage
No API mappings		

[+ Create API mapping](#)

Make note of the CloudFront distribution's domain name shown in the output. You will need it to set the custom domain's CNAME or alias record in your DNS.

7. In this step, we will use Amazon Route 53 as an example DNS provider to show how to set up the alias record to map the custom domain to its CloudFront distribution. The instructions can be adapted to other DNS providers.
 - a. Go to the Amazon Route 53 console.
 - b. If necessary, register a custom domain name.
 - c. Create a hosted zone.
 - d. Create a record set (e.g., api.example.com.)
 - e. Choose **Yes** for **Alias**, type the CloudFront domain name (e.g., d3boq9ikohtgw.cloudfront.net) in **Alias Target**, and then choose **Create**.

Create Record Set

Name: api.example.com

Type: A - IPv4 address

Alias: Yes No

Alias Target: d3boq9ikohtgw.cloudfront.net

Alias Hosted Zone ID: Z2FDTNDATAQYW2

Routing Policy: Simple

Route 53 responds to queries based only on the values in this record. [Learn More](#)

Evaluate Target Health: Yes No

Create

For most DNS providers, including Amazon Route 53, your custom domain name is added to the hosted zone as a CNAME resource record set. The CNAME record name specifies the custom domain name you typed earlier for **Domain Name** (for example, api.example.com). The CNAME record value specifies the domain name for the CloudFront distribution. However, use of a CNAME record will not work if your custom domain is a zone apex (i.e., example.com

instead of `api.example.com`). A zone apex is also commonly known as the root domain of your organization.

With Amazon Route 53 you can also create an alias resource record set for your custom domain name and specify the CloudFront distribution as the alias target. This means that Amazon Route 53 can route your custom domain name even if it is a zone apex. For more information, see [Choosing Between Alias and Non-Alias Resource Record Sets](#) in the *Amazon Route 53 Developer Guide*.

Specify API Mappings for a Custom Domain Name

After you have set up a custom domain name, you must configure how individual APIs are invoked with the custom domain name. This amounts to specifying an API's URL with the given domain name. For example, if you have created an API named `PetStore` and another API named `PetShop` and set up a custom domain name of `api.example.com` in API Gateway, you can set the `PetStore` API's URL as `https://api.example.com` or `https://api.example.com/myPetStore`. This involves setting up the API's base path. The first example uses an empty base path and the second example uses `myPetStore` as the base path of the API, relative to the domain name. Similarly, you can use `https://api.example.com/yourPetStore` as the `PetShop` API's URL. The base path is `yourPetShop`. Thus, base paths can be used to host multiple APIs behind a single custom domain name.

Complete the steps in [Set Up a Custom Domain Name for an API Gateway API \(p. 281\)](#) before setting the base path for API mappings.

To set the base path for API mappings

1. For each URL variation you want to enable, choose **Create API mapping**.
2. (Optional) For **Base path**, type the base path name that API callers must provide as part of the URL. This value must be unique for all of the mappings across a single API. Leave this blank if you do not want callers to specify a base path name after the domain name.
3. For **API**, choose the name of an available API from the selected region in your AWS account.
4. (Optional) For **Stage**, choose the name of the API's stage you want to use for this mapping. Leave this blank if you want callers to explicitly specify the stage name after any base path name.
5. Choose **Save**.

Note

To delete a mapping after you create it, next to the mapping that you want to delete, choose **Remove**.

Base Path Examples of API Mappings for a Custom Domain Name

The following examples use a custom domain name of `api.example.com`:

- Leave **Base Path** blank, specify an **API** of `MyDemoAPI`, and specify a **Stage** value of `prod` to enable calls to `https://api.example.com` to be forwarded to `https://my-api-id.execute-api.region-id.amazonaws.com/prod` (where `my-api-id` is the identifier API Gateway assigns to the API named `MyDemoAPI`).
- Leave **Base Path** blank, specify an **API** of `MyDemoAPI`, and leave **Stage** blank to enable calls to `https://api.example.com/prod` to be forwarded to `https://my-api-id.execute-api.region-id.amazonaws.com/prod` (where `my-api-id` is the identifier API Gateway assigns to the API named `MyDemoAPI`).

- Specify a **Base Path** value of `billing`, specify an **API** of `MyDemoAPI`, and leave **Stage** blank to enable calls to `https://api.example.com/billing/beta` to be forwarded to `https://my-api-id.execute-api.region-id.amazonaws.com/beta` (where `my-api-id` is the identifier API Gateway assigns to the API named `MyDemoAPI`).
- Specify a **Base Path** value of `scheduling`, specify an **API** of `MyDemoAPI`, and specify a **Stage** value of `gamma` to enable calls to `https://api.example.com/scheduling` to be forwarded to `https://my-api-id.execute-api.region-id.amazonaws.com/gamma` (where `my-api-id` is the identifier API Gateway assigns to the API named `MyDemoAPI`).

Upload and Renew an Expiring Certificate

The following steps describe how to upload and renew an expiring certificate for a custom domain name using the API Gateway console. You cannot rotate custom domain name certificates programmatically.

To upload a new certificate for a custom domain name

1. Choose **Custom Domain Names** from the API Gateway console main navigation pane.
2. Select a custom name under the **Domain Names** pane.
3. Choose **Upload**

Note

The upload feature will not be available when the certificate is being initialized or rotated for the selected custom domain name. However, upload for a different domain name is still available because the upload feature is independent for each custom domain name.

4. In **Upload Backup Certificate for `a-domain-name`** specify the following:
 - Type a name for the new certificate in **Certificate name**. The name should be different from the name of the expiring certificate.
 - Type or paste the PEM-formatted new certificate body in **Certificate body**.
 - Type or paste the PEM-formatted new certificate key in **Certificate private key**
 - Type or paste the PEM-formatted new certificate chain in **Certificate chain**.

Then, choose **Save**.

5. Choose **Rotate** to start replacing the old certificate by the new certificate.

Note

The certificate rotation takes up to 40 minutes to finish. The custom domain name is available during the rotation.

The screenshot shows the 'Upload Backup Certificate' interface in the Amazon API Gateway console. At the top, the domain name 'api.example.com' is displayed with a 'Delete Custom Domain Name' button. Below this, a message instructs the user to create an Alias resource record with their DNS provider to map 'api.example.com' to 'd3boq9ikothtgw.cloudfront.net'. The 'Certificate name' is set to 'my-example-certificate-2' and the 'Distribution domain name' is 'd3boq9ikothtgw.cloudfront.net'. A 'Backup Certificate' section shows 'my-example-certificate-3' with 'Rotate' and 'Upload' buttons. Below the certificate information, there is an 'API Mappings' section with a table:

Base path	API	Stage	
petstore	PetStore	test	

At the bottom of the console, there is a '+ Create API mapping' button.

Call Your API with Custom Domain Names

Calling an API with a custom domain name is the same as calling the API with its default domain name, provided that the correct URL is used.

API Gateway supports custom domain names for an API by using [Server Name Indication \(SNI\)](#). After a custom domain name is configured with the API, you can call the API with the custom domain name by using a browser or a client library that supports SNI.

API Gateway enforces SNI on the CloudFront distribution. For information on how CloudFront uses custom domain names, see [Amazon CloudFront Custom SSL](#).

Documenting an API Gateway API

To help customers understand and use your API, you should document the API. To help you document your API, API Gateway lets you add and update the help content for individual API entities as an integral part of your API development process. API Gateway stores the source content and enables you to archive different versions of the documentation. You can associate a documentation version with an API stage, export a stage-specific documentation snapshot to an external Swagger file, and distribute the file as a publication of the documentation.

To document your API, you can call the [API Gateway REST API](#), use one of the [AWS SDKs](#) or [AWS CLIs](#) for API Gateway, or use the API Gateway console. In addition, you can import or export the documentation parts defined in an external Swagger file. Before explaining how to document your API, we'll show how API documentation is represented in API Gateway.

Topics

- [Representation of API Documentation in API Gateway](#) (p. 287)
- [Document an API Using the API Gateway Console](#) (p. 294)
- [Document an API Using the API Gateway REST API](#) (p. 302)
- [Publish API Documentation](#) (p. 319)
- [Import API Documentation](#) (p. 325)
- [Control Access to API Documentation](#) (p. 327)

Representation of API Documentation in API Gateway

API Gateway API documentation consists of individual documentation parts associated with specific API entities that include API, resource, method, request, response, message parameters (i.e., path, query, header), as well as authorizers and models.

In API Gateway, a documentation part is represented by a [DocumentationPart](#) resource. The API documentation as a whole is represented by the [DocumentationParts](#) collection.

Documenting an API involves creating `DocumentationPart` instances, adding them to the `DocumentationParts` collection, and maintaining versions of the documentation parts as your API evolves.

Topics

- [Documentation Parts](#) (p. 288)
- [Documentation Versions](#) (p. 294)

Documentation Parts

A `DocumentationPart` resource is a JSON object that stores the documentation content applicable to an individual API entity. Its `properties` field contains the documentation content as a map of key-value pairs. Its `location` property identifies the associated API entity.

The shape of a content map is determined by you, the API developer. The value of a key-value pair can be a string, number, boolean, object, or array. The shape of the `location` object depends on the targeted entity type.

The `DocumentationPart` resource supports content inheritance: the documentation content of an API entity is applicable to children of that API entity. For more information about the definition of child entities and content inheritance, see [Inherit Content from an API Entity of More General Specification](#) (p. 289).

Location of a Documentation Part

The `location` property of a `DocumentationPart` instance identifies an API entity to which the associated content applies. The API entity can be an API Gateway REST API resource, such as [RestApi](#), [Resource](#), [Method](#), [MethodResponse](#), [Authorizer](#), or [Model](#). The entity can also be a message parameter, such as a URL path parameter, a query string parameter, a request or response header parameter, a request or response body, or response status code.

To specify an API entity, set the `type` attribute of the `location` object to be one of `API`, `AUTHORIZER`, `MODEL`, `RESOURCE`, `METHOD`, `PATH_PARAMETER`, `QUERY_PARAMETER`, `REQUEST_HEADER`, `REQUEST_BODY`, `RESPONSE`, `RESPONSE_HEADER`, or `RESPONSE_BODY`.

Depending on the `type` of an API entity, you might specify other `location` attributes, including [method](#), [name](#), [path](#), and [statusCode](#). Not all of these attributes are valid for a given API entity. For example, `type`, `path`, `name`, and `statusCode` are valid attributes of the `RESPONSE` entity; only `type` and `path` are valid location attributes of the `RESOURCE` entity. It is an error to include an invalid field in the `location` of a `DocumentationPart` for a given API entity.

Not all valid `location` fields are required. For example, `type` is both the valid and required `location` field of all API entities. However, `method`, `path`, and `statusCode` are valid but not required attributes for the `RESPONSE` entity. When not explicitly specified, a valid `location` field assumes its default value. The default `path` value is `/`, i.e., the root resource of an API. The default value of `method`, or `statusCode` is `*`, meaning any method, or status code values, respectively.

Content of a Documentation Part

The `properties` value is encoded as a JSON string. The `properties` value contains any information you choose to meet your documentation requirements. For example, the following is a valid content map:

```
{
  "info": {
    "description": "My first API with Amazon API Gateway."
  },
  "x-custom-info" : "My custom info, recognized by Swagger.",
  "my-info" : "My custom info not recognized by Swagger."
}
```

To set it as a value of `properties` using the API Gateway REST API, encode this object as a JSON string:

```
"{\n\t\t\"info\": {\n\t\t\t\"description\": \"My first API with Amazon API Gateway.\n\t\t\t}\n\t\t}, ... \n}"
```

Although API Gateway accepts any valid JSON string as the content map, the content attributes are treated as two categories: those that can be recognized by Swagger and those that cannot. In the preceding example, `info`, `description`, and `x-custom-info` are recognized by Swagger as a standard Swagger object, property, or extension. In contrast, `my-info` is not compliant with the Swagger specification. API Gateway propagates Swagger-compliant content attributes into the API entity definitions from the associated `DocumentationPart` instances. API Gateway does not propagate the non-compliant content attributes into the API entity definitions.

As another example, here is `DocumentationPart` targeted for a `Resource` entity:

```
{
  "location" : {
    "type" : "RESOURCE",
    "path": "/pets"
  },
  "properties" : {
    "summary" : "The /pets resource represents a collection of pets in
PetStore.",
    "description": "... a child resource under the root...",
  }
}
```

Here, both `type` and `path` are valid fields to identify the target of the `RESOURCE` type. For the root resource (`/`), you can omit the `path` field.

```
{
  "location" : {
    "type" : "RESOURCE"
  },
  "properties" : {
    "description" : "The root resource with the default path
specification."
  }
}
```

This is the same as the following `DocumentationPart` instance:

```
{
  "location" : {
    "type" : "RESOURCE",
    "path": "/"
  },
  "properties" : {
    "description" : "The root resource with an explicit path
specification"
  }
}
```

Inherit Content from an API Entity of More General Specifications

The default value of an optional `location` field provides a patterned description of an API entity. Using the default value in the `location` object, you can add a general description in the `properties` map to a `DocumentationPart` instance with this type of `location` pattern. API Gateway extracts the applicable Swagger documentation attributes from the `DocumentationPart` of the generic API entity

and injects them into a specific API entity with the `location` fields matching the general `location` pattern, or matching the exact value, unless the specific entity already has a `DocumentationPart` instance associated with it. This behavior is also known as content inheritance from an API entity of more general specifications.

Content inheritance does not apply to API entities of the `API`, `AUTHORIZER`, `MODEL`, or `RESOURCE` type.

When an API entity matches more than one `DocumentationPart`'s location pattern, the entity will inherit the documentation part with the location fields of the highest precedence and specificities. The order of precedence is `path > method > statusCode`. For matching with the `path` field, API Gateway chooses the entity with the most specific path value. The following table shows this with a few examples.

Case	path	method	statusCode	name	Remarks
1	/pets	*	*	id	Documentation associated with this location pattern will be inherited by entities matching the location pattern.
2	/pets	*	200	id	Documentation associated with this location pattern will be inherited by entities matching the location pattern when both Case 1 and Case 2 are matched, because Case 2 is more specific than Case 1.
3	/pets	GET	*	id	Documentation associated with this location pattern will be inherited by entities matching the location pattern when Cases 1, 2, and 3 are matched, because Case 3 has a higher precedence than Case 2 and is more specific than Case 1.
4	/pets	GET	200	id	Documentation associated with this location pattern will be inherited by entities matching the location pattern when Cases 1, 2, 3 and 4 are matched, because Case 4 is more specific than Case 1, 2 or 3.
5	/pets/ petId	*	*	id	Documentation associated with this location pattern will be inherited when Cases 1, 2, 3, 4 and 5 are matched, because Case 5 has a more specific <code>path</code> field and <code>path</code> takes the highest precedence.

Here is another example to contrast a more generic `DocumentationPart` instance to a more specific one. The following general error message of "Invalid request error" is injected into the Swagger definitions of the 400 error responses, unless overridden.

```
{
  "location" : {
    "type" : "RESPONSE",
    "statusCode": "400"
  },
  "properties" : {
    "description" : "Invalid request error."
  }
}
```

With the following overwrite, the 400 responses to any methods on the `/pets` resource has a description of "Invalid petId specified" instead.

```
{
  "location" : {
    "type" : "RESPONSE",
    "path": "/pets",
    "statusCode": "400"
  },
  "properties" : "{
    "description" : "Invalid petId specified."
  }"
}
```

Valid Location Fields of [DocumentationPart](#)

The following table shows the valid and required fields as well as applicable default values of a [DocumentationPart](#) resource that is associated with a given type of API entities.

API entity	Valid location fields	Required location fields	Default field values	Inheritable Content
API	<pre>{ "location": { "type": "API" }, ... }</pre>	type	N/A	No
Resource	<pre>{ "location": { "type": "RESOURCE", "path": "<i>resource_path</i>" }, ... }</pre>	type	The default value of path is <code>/</code> .	No
Method	<pre>{ "location": { "type": "METHOD", "path": "<i>resource_path</i>", "method": "<i>http_verb</i>" }, ... }</pre>	type	The default values of path and method are <code>/</code> and <code>*</code> , respectively.	Yes, matching path by prefix and matching method of any values.
Query parameter	<pre>{ "location": {</pre>	type	The default values of path and method are <code>/</code> and <code>*</code> , respectively.	Yes, matching path by prefix and matching

API entity	Valid location fields	Required location fields	Default field values	Inheritable Content
	<pre> "type": "QUERY_PARAMETER", "path": "resource_path", "method": "HTTP_verb", "name": "query_parameter_name" }, ... </pre>			method by exact values.
Request body	<pre> { "location": { "type": "REQUEST", "path": "resource_path", "method": "http_verb" }, ... } </pre>	type	The default values of path, and method are / and *, respectively.	Yes, matching path by prefix, and matching method by exact values.
Request header parameter	<pre> { "location": { "type": "REQUEST_HEADER", "path": "resource_path", "method": "HTTP_verb", "name": "header_name" }, ... } </pre>	type, name	The default values of path and method are / and *, respectively.	Yes, matching path by prefix and matching method by exact values.
Request path parameter	<pre> { "location": { "type": "PATH_PARAMETER", "path": "resource/{path_parameter_name}", "method": "HTTP_verb", "name": "path_parameter_name" }, ... } </pre>	type, name	The default values of path and method are / and *, respectively.	Yes, matching path by prefix and matching method by exact values.

API entity	Valid location fields	Required location fields	Default field values	Inheritable Content
Response	<pre>{ "location": { "type": "RESPONSE", "path": "<i>resource_path</i>", "method": "<i>http_verb</i>", "statusCode": "<i>status_code</i>" }, ... }</pre>	type	The default values of path, method, and statusCode are /, * and *, respectively.	Yes, matching path by prefix and matching method and statusCode by exact values.
Response header	<pre>{ "location": { "type": "RESPONSE_HEADER", "path": "<i>resource_path</i>", "method": "<i>http_verb</i>", "statusCode": "<i>status_code</i>", "name": "<i>header_name</i>" }, ... }</pre>	type, name	The default values of path, method and statusCode are /, * and *, respectively.	Yes, matching path by prefix and matching method, and statusCode by exact values.
Response body	<pre>{ "location": { "type": "RESPONSE_BODY", "path": "<i>resource_path</i>", "method": "<i>http_verb</i>", "statusCode": "<i>status_code</i>" }, ... }</pre>	type	The default values of path, method and statusCode are /, * and *, respectively.	Yes, matching path by prefix and matching method, and statusCode by exact values.

API entity	Valid location fields	Required location fields	Default field values	Inheritable Content
Authorizer	<pre>{ "location": { "type": "AUTHORIZER", "name": "authorizer_name" }, ... }</pre>	type	N/A	No
Model	<pre>{ "location": { "type": "MODEL", "name": "model_name" }, ... }</pre>	type	N/A	No

Documentation Versions

A documentation version is a snapshot of the [DocumentationParts](#) collection of an API and is tagged with a version identifier. Publishing the documentation of an API involves creating a documentation version, associating it with an API stage, and exporting that stage-specific version of the API documentation to an external Swagger file. In API Gateway, a documentation snapshot is represented as a [DocumentationVersion](#) resource.

As you update an API, you create new versions of the API. In API Gateway, you maintain all the documentation versions using the [DocumentationVersions](#) collection.

Document an API Using the API Gateway Console

In this section, we describe how to create and maintain documentation parts of an API using the API Gateway console.

A prerequisite for creating and editing the documentation of an API is that you must have already created the API. In this section, we use the [PetStore](#) API as an example. To create an API using the API Gateway console, follow the instructions in [Create an API Gateway API from an Example \(p. 7\)](#).

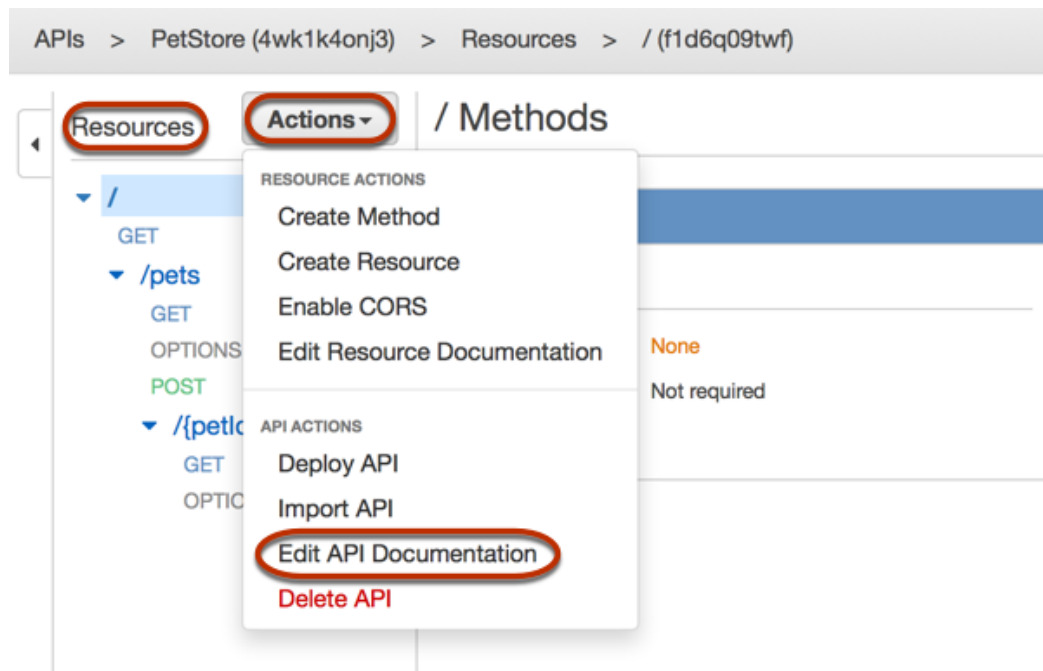
Topics

- [Document the API Entity \(p. 295\)](#)
- [Document a RESOURCE Entity \(p. 297\)](#)
- [Document a METHOD Entity \(p. 297\)](#)
- [Document a QUERY_PARAMETER Entity \(p. 298\)](#)
- [Document a PATH_PARAMETER Entity \(p. 299\)](#)

- [Document a REQUEST_HEADER Entity \(p. 300\)](#)
- [Document a REQUEST_BODY Entity \(p. 300\)](#)
- [Document a RESPONSE Entity \(p. 300\)](#)
- [Document a RESPONSE_HEADER Entity \(p. 301\)](#)
- [Document a RESPONSE_BODY Entity \(p. 301\)](#)
- [Document a MODEL Entity \(p. 301\)](#)
- [Document an AUTHORIZER Entity \(p. 302\)](#)

Document the API Entity

To add a documentation part for the API entity, choose **Resources** from the **PetStore** API. Choose the **Actions** → **Edit API Documentation** menu item.



If a documentation part was not created for the API, you get the documentation part's `properties` map editor. Type the following `properties` map in the text editor and then choose **Save** to create the documentation part.

```
{
  "info": {
    "description": "Your first API Gateway API.",
    "contact": {
      "name": "John Doe",
      "email": "john.doe@api.com",
    }
  }
}
```

Note

You do not encode the `properties` map into a JSON string, as you must do when using the API Gateway REST API. The API Gateway console stringifies the JSON object for you.

Documentation ✕

Provide your API documentation in JSON format in the form below.

Type API

```
1 {  
2   "info": {  
3     "description" : "Your first API Gateway API.",  
4     "contact": {  
5       "name": "John Doe",  
6       "email": "john.doe@api.com"  
7     }  
8   }  
9 }
```

If a documentation part has already been created, you first get the `properties` map viewer, as shown in the following.

Documentation ×

Specify your documentation part in JSON format in the form below. For more information, see the [Documentation Parts Documentation](#).

Type API

```
{
  "info": {
    "description": "Your first API Gateway API.",
    "contact": {
      "name": "John Doe",
      "email": "john.doe@api.com"
    }
  }
}
```

Delete **Close** **Edit**

Choosing **Edit** brings up the `properties` map editor as shown previously.

Document a RESOURCE Entity

To add or edit the documentation part for the API's root resource, choose `/` under the **Resource** tree, and then choose the **Actions** → **Edit Resource Documentation** menu item.

If no documentation part was created for this entity, you get the **Documentation** window. Type a valid `properties` map in the editor. Then choose **Save** and **Close**.

```
{
  "description": "The PetStore's root resource."
}
```

If a documentation part has already been defined for the RESOURCE entity, you get the documentation viewer. Choose **Edit** to open the **Documentation** editor. Modify the existing `properties` map. Choose **Save** and then choose **Close**.

If necessary, repeat these steps to add a documentation part to other RESOURCE entities.

Document a METHOD Entity

To add or edit documentation for a METHOD entity, using the GET method on the root resource as an example, choose **GET** under the `/` resource and then choose the **Actions** → **Edit Method Documentation** menu item.

For the new documentation part, type the following `properties` map in the **Documentation** editor in the **Documentation** window. Then choose **Save** and **Close**.

```
{  
  "tags" : [ "pets" ],  
  "description" : "PetStore HTML web page containing API usage information"  
}
```

For the existing documentation, choose **Edit** from the **Documentation** viewer. Edit the documentation content in the **Documentation** editor and choose **Save**. Choose **Close**.

From the **Documentation** viewer, you can also delete the documentation part.

If necessary, repeat these steps to add a documentation part to other methods.

Document a `QUERY_PARAMETER` Entity

To add or edit a documentation part for a request query parameter, using the `GET /pets?type=...&page=...` method as an example, choose **GET** under **/pets** from the **Resources** tree. Choose **Method Request** in the **Method Execution** window. Expand the **URL Query String Parameters** section. Choose the **page** query parameter, for example, and choose the book icon to open the **Documentation** viewer or editor.

← Method Execution /pets - GET - Method Request

Provide information about this method's authorization settings and the parameters it can receive.

Authorization Settings

Authorization NONE

API Key Required true

▼ URL Query String Parameters

Name	Caching	
page	<input type="checkbox"/>	
type	<input type="checkbox"/>	

Add query string

Alternatively, you can choose **Documentation** under the **PetStore** API from the main navigation pane. Then choose `Query Parameter` for **Type**. For the PetStore example API, this shows the documentation parts for the `page` and `type` query parameters.

Documentation

Create Documentation Part Import Documentation Publish Document

Add documentation to help developers understand how to interact with your API. Documentation parts can be shared across multiple resources and methods by specifying a wildcard value (*) for method or status code, eg. documentation for a 200 response can be used in multiple locations. You can also import documentation by supplying a Swagger definition file, and publish documentation to a stage. For more information, reference the [documentation](#).

Type Query Parameter Path Method All Name

Type Query Parameter
Path /pets
Method GET
Name page
{
 "description": "Page number of results to return."
}

Edit Clone

Type Query Parameter
Path /pets
Method GET
Name type
{
 "description": "The type of pet to retrieve"
}

Edit

For an API with query parameters defined for other methods, you can filter your selection by specifying the `path` of the affected resource for **Path**, choosing the desired HTTP method from **Method**, or typing the query parameter name in **Name**.

For example, choose the `page` query parameter. Choose **Edit** to modify the existing documentation. Choose **Save** to save the change.

To add a new documentation part for a `QUERY_PARAMETER` entity, choose **Create Documentation Part**. Choose `Query Parameter` for **Type**. Type a resource path (e.g., `/pets`) in **Path**. Choose an HTTP verb (e.g., `GET`) for **Method**. Type a `properties` description in the text editor. Then choose **Save**.

If necessary, repeat these steps to add a documentation part to other request query parameters.

Document a `PATH_PARAMETER` Entity

To add or edit documentation for a path parameter, go to **Method Request** of the method on the resource specified by the path parameter. Expand the **Request Paths** section. Choose the book icon for the path parameter to open the **Documentation** viewer or editor. Add or modify the properties of the documentation part.

Alternatively, choose **Documentation** under the **PetStore** API from the main navigation pane. Choose `Path Parameter` for **Type**. Choose **Edit** on a path parameter from the list. Modify the content and then choose **Save**.

To add documentation for a path parameter not listed, choose **Create Documentation Part**. Choose **Path Parameter** for **Type**. Set a resource path in **Path**, choose a method from **Method**, and set a path parameter name in **Name**. Add the documentation's properties and choose **Save**.

If required, repeat these steps to add or edit a documentation part to other path parameters.

Document a `REQUEST_HEADER` Entity

To add or edit documentation for a request header, go to **Method Request** of the method with the header parameter. Expand the **HTTP Request Headers** section. Choose the book icon for the header to open the **Documentation** viewer or editor. Add or modify the properties of the documentation part.

Alternatively, choose **Documentation** under the API from the main navigation pane. Then choose `Request Header` for **Type**. Choose **Edit** on a listed request header to change the documentation. To add documentation for an unlisted request header, choose **Create Documentation Part**. Choose `Request Header` for **Type**. Specify a resource path in **Path**. Choose a method for **Method**. Type a header name in **Name**. Then add and save a `properties` map.

If required, repeat these steps to add or edit a documentation part to other request headers.

Document a `REQUEST_BODY` Entity

To add or edit documentation for a request body, go to **Method Request** for a method. Choose the book icon for **Request Body** to open the **Documentation** viewer and then editor. Add or modify the properties of the documentation part.

Alternatively, choose **Documentation** under the API from the main navigation pane. Then choose `Request Body` for **Type**. Choose **Edit** on a listed request body to change the documentation. To add documentation for an unlisted request body, choose **Create Documentation Part**. Choose `Request Body` for **Type**. Set a resource path in **Path**. Choose an HTTP verb for **Method**. Then add and save a `properties` map.

If required, repeat these steps to add or edit a documentation part to other request bodies.

Document a `RESPONSE` Entity

To add or edit documentation for a response body, go to **Method Response** of a method. Choose the book icon for **Method Response** to open the **Documentation** viewer and then editor. Add or modify the properties of the documentation part.



Alternatively, choose **Documentation** under the API from the main navigation pane. Then choose `Response (status code)` for **Type**. Choose **Edit** on a listed response of a specified HTTP status code to change the documentation. To add documentation for an unlisted response body, choose **Create Documentation Part**. Choose `Response (status code)` for **Type**. Set a resource path in **Path**. Choose an HTTP verb for **Method**. Type an HTTP status code in **Status Code**. Then add and save the documentation part properties.

If required, repeat these steps to add or edit a documentation part to other responses.

Document a RESPONSE_HEADER Entity

To add or edit documentation for a response header, go to **Method Response** of a method. Expand a response section of a given HTTP status. Choose the book icon for a response header under **Response Headers for *Status Code*** to open the **Documentation** viewer and then editor. Add or modify the properties of the documentation part.

Alternatively, choose **Documentation** under the API from the main navigation pane. Then choose **Response Header** for **Type**. Choose **Edit** on a listed response header to change the documentation. To add documentation for an unlisted response header, choose **Create Documentation Part**. Choose **Response Header** for **Type**. Set a resource path in **Path**. Choose an HTTP verb for **Method**. Type an HTTP status code in **Status Code**. Type the response header name in **Name**. Then add and save the documentation part properties.

If required, repeat these steps to add or edit a documentation part to other response headers.

Document a RESPONSE_BODY Entity

To add or edit documentation for a response body, go to **Method Response** of a method. Expand the response section of a given HTTP status. Choose the book icon for **Response Body for *Status Code*** to open the **Documentation** viewer and then editor. Add or modify the properties of the documentation part.

Alternatively, choose **Documentation** under the API from the main navigation pane. Then choose **Response Body** for **Type**. Choose **Edit** on a listed response body to change the documentation. To add documentation for an unlisted response body, choose **Create Documentation Part**. Choose **Response Body** for **Type**. Set a resource path in **Path**. Choose an HTTP verb for **Method**. Type an HTTP status code in **Status Code**. Then add and save the documentation part properties.

If required, repeat these steps to add or edit a documentation part to other response bodies.

Document a MODEL Entity

Documenting a MODEL entity involves creating and managing `DocumentationPart` instances for the model and each of the model's `properties`. For example, for the `Error` model that comes with every API by default has the following schema definition,

```
{
  "$schema" : "http://json-schema.org/draft-04/schema#",
  "title" : "Error Schema",
  "type" : "object",
  "properties" : {
    "message" : { "type" : "string" }
  }
}
```

and requires two `DocumentationPart` instances, one for the `Model` and the other for its `message` property:

```
{
  "location": {
    "type": "MODEL",
    "name": "Error"
  },
  "properties": {
    "title": "Error Schema",
```

```
    "description": "A description of the Error model"
  }
}
```

and

```
{
  "location": {
    "type": "MODEL",
    "name": "Error.message"
  },
  "properties": {
    "description": "An error message."
  }
}
```

When the API is exported, the `DocumentationPart`'s properties will override the values in the original schema.

To add or edit documentation for a model, go to **Models** of the API in the main navigation pane. Choose the book icon for the name of a listed model to open the **Documentation** viewer and then editor. Add or modify the properties of the documentation part.

Alternatively, choose **Documentation** under the API from the main navigation pane. Then choose `Model` for **Type**. Choose **Edit** on a listed model to change the documentation. To add documentation for an unlisted model, choose **Create Documentation Part**. Choose **Model** for **Type**. Give a name to the model in **Name**. Then add and save the documentation part properties.

If required, repeat these steps to add or edit a documentation part to other models.

Document an `AUTHORIZER` Entity

To add or edit documentation for an authorizer, go to **Authorizers** for the API in the main navigation pane. Choose the book icon for the listed authorizer to open the **Documentation** viewer and then editor. Add or modify the properties of the documentation part.

Alternatively, choose **Documentation** under the API from the main navigation pane. Then choose `Authorizer` for **Type**. Choose **Edit** on a listed authorizer to change the documentation. To add documentation for an unlisted authorizer, choose **Create Documentation Part**. Choose **Authorizer** for **Type**. Give a name to the authorizer in **Name**. Then add and save the documentation part properties.

If required, repeat these steps to add or edit a documentation part to other authorizers.

To add a documentation part for an authorizer, choose **Create Documentation Part**. Choose **Authorizer** for **Type**. Specify a value for the valid `location` field of **Name** for the authorizer.

Add and save the documentation content in the `properties` map editor.

If required, repeat these steps to add a documentation part to another authorizer.

Document an API Using the API Gateway REST API

In this section, we describe how to create and maintain documentation parts of an API using the API Gateway REST API.

Before creating and editing the documentation of an API, first create the API. In this section, we use the [PetStore](#) API as an example. To create an API using the API Gateway console, follow the instructions in [Create an API Gateway API from an Example](#) (p. 7).

Topics

- [Document the API Entity](#) (p. 303)
- [Document a RESOURCE Entity](#) (p. 304)
- [Document a METHOD Entity](#) (p. 307)
- [Document a QUERY_PARAMETER Entity](#) (p. 309)
- [Document a PATH_PARAMETER Entity](#) (p. 311)
- [Document a REQUEST_BODY Entity](#) (p. 312)
- [Document a REQUEST_HEADER Entity](#) (p. 313)
- [Document a RESPONSE Entity](#) (p. 314)
- [Document a RESPONSE_HEADER Entity](#) (p. 314)
- [Document an AUTHORIZER Entity](#) (p. 315)
- [Document a MODEL Entity](#) (p. 317)
- [Update Documentation Parts](#) (p. 318)
- [List Documentation Parts](#) (p. 319)

Document the API Entity

To add documentation for an [API](#), add a [DocumentationPart](#) resource for the API entity:

```
POST /restapis/restapi_id/documentation/parts HTTP/1.1
Host: apigateway.region.amazonaws.com
Content-Type: application/json
X-Amz-Date: YYYYMMDDTtttttZ
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}"
}
```

If successful, the operation returns a 201 Created response containing the newly created `DocumentationPart` instance in the payload. For example:

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

If the documentation part has already been added, a 409 Conflict response returns, containing the error message of Documentation part already exists for the specified location: type 'API'. In this case, you must call the [documentationpart:update](#) operation.

```
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}"  
  } ]  
}
```

The successful response returns a 200 OK status code with the payload containing the updated DocumentationPart instance in the payload.

Document a RESOURCE Entity

To add documentation for the root resource of an API, add a [DocumentationPart](#) resource targeted for the corresponding [Resource](#) resource:

```
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}"  
}
```

If successful, the operation returns a 201 Created response containing the newly created DocumentationPart instance in the payload. For example:

```
{
```

```

    "_links": {
      "curies": {
        "href": "http://docs.aws.amazon.com/apigateway/latest/developerguide/
restapi-documentationpart-{rel}.html",
        "name": "documentationpart",
        "templated": true
      },
      "self": {
        "href": "/restapis/4wklk4onj3/documentation/parts/p76vqo"
      },
      "documentationpart:delete": {
        "href": "/restapis/4wklk4onj3/documentation/parts/p76vqo"
      },
      "documentationpart:update": {
        "href": "/restapis/4wklk4onj3/documentation/parts/p76vqo"
      }
    },
    "id": "p76vqo",
    "location": {
      "path": "/",
      "method": null,
      "name": null,
      "statusCode": null,
      "type": "RESOURCE"
    },
    "properties": "{\n\t\"description\" : \"The PetStore root resource.\"\n}"
  }

```

When the resource path is not specified, the resource is assumed to be the root resource. You can add "path": "/" to properties to make the specification explicit.

To create documentation for a child resource of an API, add a [DocumentationPart](#) resource targeted for the corresponding [Resource](#) resource:

```

POST /restapis/restapi_id/documentation/parts HTTP/1.1
Host: apigateway.region.amazonaws.com
Content-Type: application/json
X-Amz-Date: YYYYMMDDTtttttZ
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}"
}

```

If successful, the operation returns a 201 Created response containing the newly created DocumentationPart instance in the payload. For example:

```

{
  "_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\n}"
}

```

To add documentation for a child resource specified by a path parameter, add a [DocumentationPart](#) resource targeted for the [Resource](#) resource:

```

POST /restapis/restapi_id/documentation/parts HTTP/1.1
Host: apigateway.region.amazonaws.com
Content-Type: application/json
X-Amz-Date: YYYYMMDDTtttttZ
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\n}"
}

```

If successful, the operation returns a 201 Created response containing the newly created DocumentationPart instance in the payload. For example:

```

{
  "_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

The [DocumentationPart](#) instance of a RESOURCE entity cannot be inherited by any of its child resources.

Document a METHOD Entity

To add documentation for a method of an API, add a [DocumentationPart](#) resource targeted for the corresponding [Method](#) resource:

```

POST /restapis/restapi_id/documentation/parts HTTP/1.1
Host: apigateway.region.amazonaws.com
Content-Type: application/json
X-Amz-Date: YYYYMMDDTtttttZ
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}"
}

```

If successful, the operation returns a 201 Created response containing the newly created `DocumentationPart` instance in the payload. For example:

```

{
  "_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}"
}
```

If successful, the operation returns a 201 Created response containing the newly created DocumentationPart instance in the payload. For example:

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

If the `location.method` field is not specified in the preceding request, it is assumed to be ANY method that is represented by a wild card * character.

To update the documentation content of a `METHOD` entity, call the `documentationpart:update` operation, supplying a new `properties` map:

```
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\n}"
  } ]
}
```

The successful response returns a 200 OK status code with the payload containing the updated `DocumentationPart` instance in the payload. For example:

```
{
  "_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\n}"
}
```

Document a `QUERY_PARAMETER` Entity

To add documentation for a request query parameter, add a `DocumentationPart` resource targeted for the `QUERY_PARAMETER` type, with the valid fields of `path` and `name`.

```
POST /restapis/restapi_id/documentation/parts HTTP/1.1
Host: apigateway.region.amazonaws.com
Content-Type: application/json
X-Amz-Date: YYYYMMDDTttttttZ
Authorization: AWS4-HMAC-SHA256 Credential=access_key_id/YYYYMMDD/region/
apigateway/aws4_request, SignedHeaders=content-length;content-type;host;x-
amz-date, Signature=sigv4_secret

{
  "location" : {
    "type" : "QUERY_PARAMETER",
    "path" : "/pets",
    "method" : "GET",
    "name" : "page"
  },
  "properties": "{\n\t\"description\" : \"Page number of results to return.
\n\n}"
}
```

If successful, the operation returns a 201 Created response containing the newly created DocumentationPart instance in the payload. For example:

```
{
  "_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\n}"
}
```

The documentation part's properties map of a QUERY_PARAMETER entity can be inherited by one of its child QUERY_PARAMETER entities. For example, if you add a `treats` resource after `/pets/{petId}`, enable the GET method on `/pets/{petId}/treats`, and expose the `page` query parameter, this child query parameter inherits the `DocumentationPart`'s properties map from the like-named query parameter of the GET `/pets` method, unless you explicitly add a

DocumentationPart resource to the page query parameter of the GET /pets/{petId}/treats method.

Document a PATH_PARAMETER Entity

To add documentation for a path parameter, add a DocumentationPart resource for the PATH_PARAMETER entity.

```
POST /restapis/restapi_id/documentation/parts HTTP/1.1
Host: apigateway.region.amazonaws.com
Content-Type: application/json
X-Amz-Date: YYYYMMDDTtttttZ
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\n}"
}
```

If successful, the operation returns a 201 Created response containing the newly created DocumentationPart instance in the payload. For example:

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

```
}
```

Document a `REQUEST_BODY` Entity

To add documentation for a request body, add a [DocumentationPart](#) resource for the request body.

```
POST /restapis/restapi_id/documentation/parts HTTP/1.1
Host: apigateway.region.amazonaws.com
Content-Type: application/json
X-Amz-Date: YYYYMMDDTtttttZ
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}\""
}
```

If successful, the operation returns a 201 Created response containing the newly created `DocumentationPart` instance in the payload. For example:

```
{
  "_links": {
    "curies": {
      "href": "http://docs.aws.amazon.com/apigateway/latest/developerguide/
restapi-documentationpart-{rel}.html",
      "name": "documentationpart",
      "templated": true
    },
    "self": {
      "href": "/restapis/4wklk4onj3/documentation/parts/kgmfr1"
    },
    "documentationpart:delete": {
      "href": "/restapis/4wklk4onj3/documentation/parts/kgmfr1"
    },
    "documentationpart:update": {
      "href": "/restapis/4wklk4onj3/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}\""
}
```

Document a `REQUEST_HEADER` Entity

To add documentation for a request header, add a `DocumentationPart` resource for the request header.

```
POST /restapis/restapi_id/documentation/parts HTTP/1.1
Host: apigateway.region.amazonaws.com
Content-Type: application/json
X-Amz-Date: YYYYMMDDTtttttZ
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\n}"
}
```

If successful, the operation returns a 201 Created response containing the newly created `DocumentationPart` instance in the payload. For example:

```
{
  "_links": {
    "curies": {
      "href": "http://docs.aws.amazon.com/apigateway/latest/developerguide/
restapi-documentationpart-{rel}.html",
      "name": "documentationpart",
      "templated": true
    },
    "self": {
      "href": "/restapis/4wklk4onj3/documentation/parts/h0m3uf"
    },
    "documentationpart:delete": {
      "href": "/restapis/4wklk4onj3/documentation/parts/h0m3uf"
    },
    "documentationpart:update": {
      "href": "/restapis/4wklk4onj3/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\n}"
}
```

Document a RESPONSE Entity

To add documentation for a response of a status code, add a [DocumentationPart](#) resource targeted for the corresponding [MethodResponse](#) resource.

```
POST /restapis/restapi_id/documentation/parts HTTP/1.1
Host: apigateway.region.amazonaws.com
Content-Type: application/json
X-Amz-Date: YYYYMMDDTtttttZ
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}"
}
```

If successful, the operation returns a 201 Created response containing the newly created [DocumentationPart](#) instance in the payload. For example:

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

Document a RESPONSE_HEADER Entity

To add documentation for a response header, add a [DocumentationPart](#) resource for the response header.

```
POST /restapis/restapi_id/documentation/parts HTTP/1.1
```

```
Host: apigateway.region.amazonaws.com
Content-Type: application/json
X-Amz-Date: YYYYMMDDTtttttZ
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}"
```

If successful, the operation returns a 201 Created response containing the newly created DocumentationPart instance in the payload. For example:

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

The documentation of this Content-Type response header is the default documentation for the Content-Type headers of any responses of the API.

Document an AUTHORIZER Entity

To add documentation for an API authorizer, add a [DocumentationPart](#) resource targeted for the specified authorizer.

```
POST /restapis/restapi_id/documentation/parts HTTP/1.1
```

```
Host: apigateway.region.amazonaws.com
Content-Type: application/json
X-Amz-Date: YYYYMMDDTtttttZ
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\n}"
}
```

If successful, the operation returns a 201 Created response containing the newly created DocumentationPart instance in the payload. For example:

```
{
  "_links": {
    "curies": {
      "href": "http://docs.aws.amazon.com/apigateway/latest/developerguide/
restapi-documentationpart-{rel}.html",
      "name": "documentationpart",
      "templated": true
    },
    "self": {
      "href": "/restapis/4wklk4onj3/documentation/parts/pw3qw3"
    },
    "documentationpart:delete": {
      "href": "/restapis/4wklk4onj3/documentation/parts/pw3qw3"
    },
    "documentationpart:update": {
      "href": "/restapis/4wklk4onj3/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\n}"
}
```

Note

The [DocumentationPart](#) instance of an AUTHORIZER entity cannot be inherited by any of its child resources.

Document a MODEL Entity

Documenting a MODEL entity involves creating and managing `DocumentationPart` instances for the model and each of the model's `properties`. For example, for the `Error` model that comes with every API by default has the following schema definition,

```
{
  "$schema" : "http://json-schema.org/draft-04/schema#",
  "title" : "Error Schema",
  "type" : "object",
  "properties" : {
    "message" : { "type" : "string" }
  }
}
```

and requires two `DocumentationPart` instances, one for the `Model` and the other for its `message` property:

```
{
  "location": {
    "type": "MODEL",
    "name": "Error"
  },
  "properties": {
    "title": "Error Schema",
    "description": "A description of the Error model"
  }
}
```

and

```
{
  "location": {
    "type": "MODEL",
    "name": "Error.message"
  },
  "properties": {
    "description": "An error message."
  }
}
```

When the API is exported, the `DocumentationPart`'s properties will override the values in the original schema.

To add documentation for an API model, add a `DocumentationPart` resource targeted for the specified model.

```
POST /restapis/restapi_id/documentation/parts HTTP/1.1
Host: apigateway.region.amazonaws.com
Content-Type: application/json
X-Amz-Date: YYYYMMDDTtttttZ
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\n}"
}
```

If successful, the operation returns a 201 Created response containing the newly created `DocumentationPart` instance in the payload. For example:

```
{
  "_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\n}"
}
```

Repeat the same step to create a `DocumentationPart` instance for any of the model's properties.

Note

The `DocumentationPart` instance of a `MODEL` entity cannot be inherited by any of its child resources.

Update Documentation Parts

To update the documentation parts of any type of API entities, submit a PATCH request on a `DocumentationPart` instance of a specified part identifier to replace the existing `properties` map with a new one.

```
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"  
  } ]  
}
```

The successful response returns a 200 OK status code with the payload containing the updated `DocumentationPart` instance in the payload.

You can update multiple documentation parts in a single `PATCH` request.

List Documentation Parts

To list the documentation parts of any type of API entities, submit a GET request on a `DocumentationParts` collection.

```
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
```

The successful response returns a 200 OK status code with the payload containing the available `DocumentationPart` instances in the payload.

Publish API Documentation

To publish the documentation for an API, create, update, or get a documentation snapshot, and then associate the documentation snapshot with an API stage. When creating a documentation snapshot, you can also associate it with an API stage at the same time.

Topics

- [Create a Documentation Snapshot and Associate it with an API Stage \(p. 319\)](#)
- [Create a Documentation Snapshot \(p. 320\)](#)
- [Update a Documentation Snapshot \(p. 320\)](#)
- [Get a Documentation Snapshot \(p. 321\)](#)
- [Associate a Documentation Snapshot with an API Stage \(p. 321\)](#)
- [Download a Documentation Snapshot Associated with a Stage \(p. 322\)](#)

Create a Documentation Snapshot and Associate it with an API Stage

To create a snapshot of an API's documentation parts and associate it with an API stage at the same time, submit the following `POST` request:

```
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"
}
```

If successful, the operation returns a 200 OK response, containing the newly created `DocumentationVersion` instance as the payload.

Alternatively, you can create a documentation snapshot without associating it with an API stage first and then call [restapi:update](#) to associate the snapshot with a specified API stage. You can also update or query an existing documentation snapshot and then update its stage association. We show the steps in the next four sections.

Create a Documentation Snapshot

To create a snapshot of an API's documentation parts, create a new [DocumentationVersion](#) resource and add it to the [DocumentationVersions](#) collection of the API:

```
POST /restapis/restapi_id/documentation/versions HTTP/1.1
Host: apigateway.region.amazonaws.com
Content-Type: application/json
X-Amz-Date: YYYYMMDDTttttttZ
Authorization: AWS4-HMAC-SHA256 Credential=access_key_id/YYYYMMDD/region/
apigateway/aws4_request, SignedHeaders=content-length;content-type;host;x-
amz-date, Signature=sigv4_secret

{
  "documentationVersion" : "1.0.0",
  "description" : "My API Documentation v1.0.0"
}
```

If successful, the operation returns a 200 OK response, containing the newly created `DocumentationVersion` instance as the payload.

Update a Documentation Snapshot

You can only update a documentation snapshot by modifying the `description` property of the corresponding [DocumentationVersion](#) resource. The following example shows how to update the description of the documentation snapshot as identified by its version identifier, `version`, e.g., `1.0.0`.

```
PATCH /restapis/restapi_id/documentation/versions/version HTTP/1.1
Host: apigateway.region.amazonaws.com
Content-Type: application/json
X-Amz-Date: YYYYMMDDTttttttZ
Authorization: AWS4-HMAC-SHA256 Credential=access_key_id/YYYYMMDD/region/
apigateway/aws4_request, SignedHeaders=content-length;content-type;host;x-
amz-date, Signature=sigv4_secret
```

```
{
  "patchOperations": [{
    "op": "replace",
    "path": "/description",
    "value": "My API for testing purposes."
  }]
}
```

If successful, the operation returns a 200 OK response, containing the updated `DocumentationVersion` instance as the payload.

Get a Documentation Snapshot

To get a documentation snapshot, submit a `GET` request against the specified [DocumentationVersion](#) resource. The following example shows how to get a documentation snapshot of a given version identifier, 1.0.0.

```
GET /restapis/<restapi_id>/documentation/versions/1.0.0 HTTP/1.1
Host: apigateway.region.amazonaws.com
Content-Type: application/json
X-Amz-Date: YYYYMMDDTttttttZ
Authorization: AWS4-HMAC-SHA256 Credential=access_key_id/YYYYMMDD/region/
apigateway/aws4_request, SignedHeaders=content-length;content-type;host;x-
amz-date, Signature=sigv4_secret
```

Associate a Documentation Snapshot with an API Stage

To publish the API documentation, associate a documentation snapshot with an API stage. You must have already created an API stage before associating the documentation version with the stage.

To associate a documentation snapshot with an API stage using the [API Gateway REST API](#), call the `stage:update` operation to set the desired documentation version on the `stage.documentationVersion` property:

```
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"
  }]
}
```

The following procedure describes how to publish a documentation version.

To publish a documentation version using the API Gateway console

1. Choose **Documentation** for the API from the main navigation pane in the API Gateway console.
2. Choose **Publish Documentation** in the **Documentation** pane.
3. Set up the publication:
 - a. Choose an available name for **Stage**.
 - b. Type a version identifier, e.g., 1.0.0, in **Version**.
 - c. Optionally, provide a description about the publication in **Description**.
4. Choose **Publish**.

You can now proceed to download the published documentation by exporting the documentation to an external Swagger file.

Download a Documentation Snapshot Associated with a Stage

After a version of the documentation parts is associated with a stage, you can export the documentation parts together with the API entity definitions, to an external file, using the API Gateway console, the API Gateway REST API, one of its SDKs, or the AWS CLI for API Gateway. The process is the same as for exporting the API. The exported file format can be JSON or YAML.

Using the API Gateway REST API, you can also explicitly set the `extension=documentation,integrations,authorizers` query parameter to include the API documentation parts, API integrations and authorizers in an API export. By default, documentation parts are included, but integrations and authorizers are excluded, when you export an API. The default output from an API export is suited for distribution of the documentation.

To export the API documentation in an external JSON Swagger file using the API Gateway REST API, submit the following `GET` request:

```
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: YYYYMMDDTtttttZ  
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
```

Here, the `x-amazon-apigateway-documentation` object contains the documentation parts and the API entity definitions contains the documentation properties supported by Swagger. The output does not include details of integration or custom authorizers. To include both details, set `extensions=integrations,authorizers,documentation`. To include details of integrations but not of authorizers, set `extensions=integrations,documentation`.

You must set the `Accept:application/json` header in the request to output the result in a JSON file. To produce the YAML output, change the request header to `Accept:application/yaml`.

As an example, we will look at an API that exposes a simple `GET` method on the root resource (`/`). This API has four API entities defined in a Swagger definition file, one for each of the `API`, `MODEL`, `METHOD`, and `RESPONSE` types. A documentation part has been added to each of the `API`, `METHOD`, and `RESPONSE` entities. Calling the preceding documentation-exporting command, we get the following

output, with the documentation parts listed within the `x-amazon-apigateway-documentation` object as an extension to a standard Swagger file.

```
{
  "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"
}
```

For a Swagger-compliant attribute defined in the `properties` map of a documentation part, API Gateway inserts the attribute into the associated API entity definition. An attribute of `x-something` is a standard Swagger extension. This extension gets propagated into the API entity definition. For example, see the `x-example` attribute for the `GET` method. An attribute like `foo` is not part of the Swagger specification and is not injected into its associated API entity definitions.

If a documentation-rendering tool (e.g., [Swagger UI](#)) parses the API entity definitions to extract documentation attributes, any non Swagger-compliant `properties` attributes of a `DocumentationPart` instance are not available for the tool. However, if a documentation-rendering tool parses the `x-amazon-apigateway-documentation` object to get content, or if the tool calls [restapi:documentation-parts](#) and [documentationpart:by-id](#) to retrieve documentation parts from API Gateway, all the documentation attributes are available for the tool to display.

To export the documentation with API entity definitions containing integration details to a JSON Swagger file, submit the following `GET` request:

```
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: YYYYMMDDTtttttZ
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
```


To export the documentation with API entity definitions containing details of integrations and authorizers to a YAML Swagger file, submit the following GET request:

```
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: YYYYMMDDTtttttZ
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
```

To use the API Gateway console to export and download the published documentation of an API, follow the instructions in [Export API Using the API Gateway Console \(p. 147\)](#).

Import API Documentation

As with importing API entity definitions, you can import documentation parts from an external Swagger file into an API in API Gateway. You specify the to-be-imported documentation parts within the [x-amazon-apigateway-documentation Object \(p. 151\)](#) extension in a valid Swagger 2.0 definition file. Importing documentation does not alter the existing API entity definitions.

You have an option to merge the newly specified documentation parts into existing documentation parts in API Gateway or to overwrite the existing documentation parts. In the MERGE mode, a new documentation part defined in the Swagger file is added to the `DocumentationParts` collection of the API. If an imported `DocumentationPart` already exists, an imported attribute replaces the existing one if the two are different. Other existing documentation attributes remain unaffected. In the OVERWRITE mode, the entire `DocumentationParts` collection is replaced according to the imported Swagger definition file.

Importing Documentation Parts Using the API Gateway REST API

To import API documentation using the API Gateway REST API, call the `documentationpart:import` operation. The following example shows how to overwrite existing documentation parts of an API with a single GET / method, returning a 200 OK response when successful.

```
PUT /restapis/<restapi_id>/documentation/
parts&mode=overwrite&failonwarnings=true
Host: apigateway.region.amazonaws.com
Content-Type: application/json
X-Amz-Date: YYYYMMDDTtttttZ
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"  
    }  
  }  
]  
}
```

When successful, this request returns a 200 OK response containing the imported `DocumentationPartId` in the payload.

```
{  
  "ids": [  
    "kg3mth",  
    "796rtf",  
    "zhek4p",  
    "5ukm9s"  
  ]  
}
```

In addition, you can also call `restapi:import` or `restapi:put`, supplying the documentation parts in the `x-amazon-apigateway-documentation` object as part of the input Swagger file of the API definition. To exclude the documentation parts from the API import, set `ignore=documentation` in the request query parameters.

Importing Documentation Parts Using the API Gateway Console

The following instructions describe how to import documentation parts.

To use the console to import documentation parts of an API from an external file

1. Choose **Documentation** for the API from the main navigation pane on the console.
2. Choose **Import Documentation** in the **Documentation** pane.
3. Choose **Select Swagger File** to load a file from a drive, or copy and paste a file contents into the file view. For an example, see the payload of the example request in [Importing Documentation Parts Using the API Gateway REST API](#) (p. 325).
4. Optionally, choose **Fail on warnings** or **Ignore warnings**, and choose `Merge` or `Overwrite` from **Import mode**.
5. Choose **Import**.

Control Access to API Documentation

If you have a dedicated documentation team to write and edit your API documentation, you can configure separate access permissions for your developers (for API development) and for your writers

or editors (for content development). This is especially appropriate when a third-party vendor is involved in creating the documentation for you.

To grant your documentation team the access to create, update, and publish your API documentation, you can assign the documentation team an IAM role with the following IAM policy, where *account_id* is the AWS account ID of your documentation team.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "StmntDocPartsAddEditViewDelete",
      "Effect": "Allow",
      "Action": [
        "apigateway:GET",
        "apigateway:PUT",
        "apigateway:POST",
        "apigateway:PATCH",
        "apigateway:DELETE"
      ],
      "Resource": [
        "arn:aws:apigateway::account_id:/restapis/*/documentation/*"
      ]
    }
  ]
}
```

For information on setting permissions to access API Gateway resources, see [Control Access to API Gateway with IAM Policies \(p. 160\)](#).

Calling a Deployed API in Amazon API Gateway

Calling a deployed API involves submitting requests to the `execute-api` component of API Gateway. The request URL is the **Invoke URL** generated by API Gateway when the API is successfully deployed. You can obtain this invocation URL from the API Gateway console or you can construct it yourself according to the following format:

```
https://{restapi_id}.execute-api.{region}.amazonaws.com/{stage_name}/
```

If your API permits anonymous access, you can use any web browser to invoke any GET-method calls by pasting the Invoke URL to the browser's address bar. For other methods or any authentication-required calls, the invocation will be more involved because you must specify a payload or sign the requests. You can handle these in a script behind an HTML page or in a client app using one of the AWS SDKs.

For testing, you can use the API Gateway console to call an API using the API Gateway's TestInvoke feature, which bypasses the Invoke URL and allows API testing before the API is deployed. Alternatively, you can use the [Postman](#) Chrome extension to test a successfully deployed API, without writing a script or a client.

Topics

- [Prerequisites](#) (p. 329)
- [Obtain an API's Invoke URL in the API Gateway Console](#) (p. 330)
- [Test a Method Using the API Gateway Console](#) (p. 330)
- [Use Postman to Test an API](#) (p. 331)

Prerequisites

- You must have already deployed the API in API Gateway. Follow the instructions in [Deploying an API](#) (p. 230).

Obtain an API's Invoke URL in the API Gateway Console

1. Sign in to the API Gateway console at <https://console.aws.amazon.com/apigateway>.
2. In the box that contains the name of the API you want to call, choose **Stages**.
3. In the **Stages** pane, choose the name of the stage.
4. The URL displayed next to **Invoke URL** should look something like this, where *my-api-id* is the identifier API Gateway assigns to your API, *region-id* is the AWS region identifier (for example, *us-east-1*) where you deployed your API, and *stage-name* is the name of the stage for the API you want to call:

```
https://my-api-id.execute-api.region-id.amazonaws.com/stage-name/  
{resourcePath}
```

Depending on the method type you want to call and the tool you want to use, copy this URL to your clipboard, and then paste and modify it to call the API from a web browser, a web debugging proxy tool or the cURL command-line tool, or from your own API.

If you are not familiar with which method to call or the format you must use to call it, browse the list of available methods by following the instructions in [View a Methods List \(p. 228\)](#).

To call the method directly from the API Gateway console, see [Test a Method Using the Console \(p. 330\)](#).

For more options, contact the API owner.

Test a Method Using the API Gateway Console

Use the API Gateway console to test a method.

Topics

- [Prerequisites \(p. 330\)](#)
- [Test a Method with the API Gateway Console \(p. 330\)](#)

Prerequisites

- You must specify the settings for the methods you want to test. Follow the instructions in [Set up Method and Integration \(p. 71\)](#).

Test a Method with the API Gateway Console

Important

Testing methods with the API Gateway console may result in changes to resources that cannot be undone. Testing a method with the API Gateway console is the same as calling the method outside of the API Gateway console. For example, if you use the API Gateway console to call a method that deletes an API's resources, if the method call is successful, the API's resources will be deleted.

1. Sign in to the API Gateway console at <https://console.aws.amazon.com/apigateway>.

2. In the box that contains the name of the API for the method, choose **Resources**.
3. In the **Resources** pane, choose the method you want to test.
4. In the **Method Execution** pane, in the **Client** box, choose **TEST**. Type values in any of the displayed boxes (such as **Query Strings**, **Headers**, and **Request Body**).

For additional options you may need to specify, contact the API owner.

5. Choose **Test**. The following information will be displayed:
 - **Request** is the resource's path that was called for the method.
 - **Status** is the response's HTTP status code.
 - **Latency** is the time between the receipt of the request from the caller and the returned response.
 - **Response Body** is the HTTP response body.
 - **Response Headers** are the HTTP response headers.

Tip

Depending on the mapping, the HTTP status code, response body, and response headers may be different from those sent from the Lambda function, HTTP proxy, or AWS service proxy.

- **Logs** are the simulated Amazon CloudWatch Logs entries that would have been written if this method were called outside of the API Gateway console.

Note

Although the CloudWatch Logs entries are simulated, the results of the method call are real.

Use Postman to Test an API

Use the [Postman](#) Chrome extension is a convenient tool to test an API in API Gateway.

1. Launch Postman.
2. Enter the endpoint URL of a request in the address bar and choose the appropriate HTTP method from the drop-down list to the left of the address bar.
3. If required, choose the **Authorization** tab. Choose **AWS Signature** for the authorization **Type**. Enter your AWS IAM user's access key ID in the **AccessKey** input field. Enter your IAM user secret key in **SecretKey**. Specify an appropriate AWS region that matches the region specified in the invocation URL. Enter `execute-api` in **Service Name**.
4. Choose the **Headers** tab. Optionally, delete any existing headers. This can clear any stale settings that may cause errors. Add any required custom headers. For example, if API keys are enabled, you can set the `x-api-key:{api_key}` name/value pair here.
5. Choose **Send** to submit the request and receive a response.

For an example of using Postman, see [Call an API with Custom authorization \(p. 180\)](#).

Monitoring and Troubleshooting in API Gateway

Topics

- [Log API management calls to Amazon API Gateway Using AWS CloudTrail \(p. 332\)](#)
- [Monitor API execution with Amazon CloudWatch \(p. 334\)](#)

For API execution, API Gateway automatically reports to Amazon CloudWatch your API's execution metrics on the API- and stage-levels. The metrics include statistics about caching, latency and detected errors. You can also opt in for API Gateway to send to CloudWatch method-level metrics, using the [API Gateway console \(p. 233\)](#) or calling the [API Gateway REST API](#) or one of its SDKs. Based on these metrics, you can set CloudWatch custom alarms for troubleshooting any performance issues of your APIs. For more information about CloudWatch, see the [Amazon CloudWatch User Guide](#).

For API management using API Gateway REST API, you can create AWS CloudTrail trails to log events involved in the API Gateway REST API calls. You can use the logs to troubleshoot API creation, deployment and updates. You can also use Amazon CloudWatch to monitor the CloudTrail logs. To learn more about CloudTrail, see the [AWS CloudTrail User Guide](#).

Note

CloudTrail logs API Gateway REST API calls an API developer or owner made against the `apigateway` component, whereas CloudWatch logs API calls an API customer or client made against the `execute-api` component of API Gateway.

Log API management calls to Amazon API Gateway Using AWS CloudTrail

You can use AWS CloudTrail to capture API Gateway REST API calls in your AWS account and deliver the log files to an Amazon S3 bucket you specify. Examples of these API calls include creating a new API, resource, or method in API Gateway. CloudTrail captures such API calls from the API Gateway console or from the API Gateway APIs directly. Using the information collected by CloudTrail, you can determine which request was made to API Gateway, the source IP address from which the request was made, who made the request, when it was made, and so on. To learn more about CloudTrail, including how to configure and enable it, see the [AWS CloudTrail User Guide](#).

API Gateway Information in CloudTrail

When CloudTrail logging is enabled in your AWS account, API calls made to API Gateway actions are tracked in log files. API Gateway records are written together with other AWS service records in a log file. CloudTrail determines when to create and write to a new file based on a time period and file size.

All of the API Gateway actions are logged and documented in the [API Gateway REST API \(p. 418\)](#). For example, calls to create a new API, resource, or method in API Gateway generate entries in CloudTrail log files.

Every log entry contains information about who generated the request. The user identity information in the log helps you determine whether the request was made with root or IAM user credentials, with temporary security credentials for a role or federated user, or by another AWS service. For more information, see the **userIdentity** field in the [CloudTrail Event Reference](#).

You can store your log files in your bucket for as long as you want, but you can also define Amazon S3 lifecycle rules to archive or delete log files automatically. By default, your log files are encrypted by using Amazon S3 server-side encryption (SSE).

You can choose to have CloudTrail publish Amazon SNS notifications when new log files are delivered so you can take action quickly. For more information, see [Configuring Amazon SNS Notifications](#).

You can also aggregate API Gateway log files from multiple AWS regions and multiple AWS accounts into a single Amazon S3 bucket. For more information, see [Aggregating CloudTrail Log Files to a Single Amazon S3 Bucket](#).

Understanding API Gateway Log File Entries

CloudTrail log files can contain one or more log entries where each entry is made up of multiple JSON-formatted events. A log entry represents a single request from any source and includes information about the requested action, any parameters, the date and time of the action, and so on. The log entries are not guaranteed to be in any particular order. That is, they are not an ordered stack trace of the public API calls.

The following example shows a CloudTrail log entry that demonstrates the API Gateway get resource action:

```
{
  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 ...
]
}
```

Monitor API execution with Amazon CloudWatch

You can monitor API execution using CloudWatch, which collects and processes raw data from API Gateway into readable, near real-time metrics. These statistics are recorded for a period of two weeks, so that you can access historical information and gain a better perspective on how your web application or service is performing. By default, API Gateway metric data is automatically sent to CloudWatch in one-minute periods. For more information, see [What Are Amazon CloudWatch, Amazon CloudWatch Events, and Amazon CloudWatch Logs?](#) in the *Amazon CloudWatch User Guide*.

The metrics reported by API Gateway provide information that you can analyze in different ways. The list below shows some common uses for the metrics. These are suggestions to get you started, not a comprehensive list.

- Monitor the **IntegrationLatency** metrics to measure the responsiveness of the back end.
- Monitor the **Latency** metrics to measure the overall responsiveness of your API calls.
- Monitor the **CacheHitCount** and **CacheMissCount** metrics to optimize cache capacities to achieve a desired performance.

Topics

- [Amazon API Gateway Dimensions and Metrics](#) (p. 334)
- [View CloudWatch Metrics with the API Dashboard in API Gateway](#) (p. 336)
- [View API Gateway Metrics in the CloudWatch Console](#) (p. 336)
- [Monitoring Tools in AWS](#) (p. 337)

Amazon API Gateway Dimensions and Metrics

The metrics and dimensions that API Gateway sends to Amazon CloudWatch are listed below. For more information, see [Monitor API Execution with Amazon CloudWatch](#) in the *Amazon API Gateway Developer Guide*.

API Gateway Metrics

Amazon API Gateway sends metric data to CloudWatch every minute.

The `AWS/ApiGateway` namespace includes the following metrics.

Metric	Description
4XXError	The number of client-side errors captured Unit: count
5XXError	The number of server-side errors captured. Unit: count
CacheHitCount	The number of requests served from the API cache. Unit: count
CacheMissCount	The number of requests served from the back end when API caching is enabled. Unit: count
Count	The number of calls to API methods. Unit: count
IntegrationLatency	The time between when API Gateway relays a request to the back end and when it receives a response from the back end. Unit: millisecond
Latency	The time between when API Gateway receives a request from a client and when it returns a response to the client. Unit: millisecond

Dimensions for Metrics

You can use the dimensions in the following table to filter API Gateway metrics.

Dimension	Description
ApiName	Filters API Gateway metrics for an API of the specified API name.
ApiName, Method, Resource, Stage	Filters API Gateway metrics for an API method of the specified API, stage, resource, and method. API Gateway will not send such metrics unless you have explicitly enabled detailed CloudWatch metrics. You can do this in the console by selecting Enable CloudWatch Metrics under a stage Settings tab. Alternatively, you can call the <code>stage:update</code> action of the API Gateway REST API to update the <code>metricsEnabled</code> property to <code>true</code> . Enabling such metrics will incur additional charges to your account. For pricing information, see Amazon CloudWatch Pricing .

Dimension	Description
ApiName, Stage	Filters API Gateway metrics for an API stage of the specified API and stage.

View CloudWatch Metrics with the API Dashboard in API Gateway

You can use the API dashboard in the API Gateway Console to display the CloudWatch metrics of your deployed API in API Gateway. These are shown as a summary of API activity over time.

Topics

- [Prerequisites \(p. 336\)](#)
- [Examine API activities in the Dashboard \(p. 336\)](#)

Prerequisites

1. You must have an API created in API Gateway. Follow the instructions in [Creating an API \(p. 70\)](#).
2. You must have the API deployed at least once. Follow the instructions in [Deploying an API \(p. 230\)](#).
3. To get CloudWatch metrics for individual methods, you must have CloudWatch Logs enabled for those methods in a given stage. The process is prescribed in [Set Up a Stage \(p. 233\)](#). Your account will be charged for accessing method-level logs, but not for accessing API- or stage-level logs.

Examine API activities in the Dashboard

1. Sign in to the API Gateway console at <https://console.aws.amazon.com/apigateway>.
2. Choose the name of the API.
3. Under the selected API, choose **Dashboard**.
4. To display a summary of API activity over time, for **Stage**, choose the desired stage.
5. Use **From** and **To** to enter the date range.
6. Refresh, if needed, and view individual metrics displayed in separate graphs titled **API Calls**, **Integration Latency**, **Latency**, **4xx Error** and **5xx Error**. The **CacheHitCount** and **CacheMissCount** graphs will be displayed only if API caching has been enabled.

Tip

To examine method-level CloudWatch metrics, make sure that you have enabled CloudWatch Logs on a method level. For more information about how to set up method-level logging, see [Set Up an API Deployment Stage with the API Gateway Console \(p. 233\)](#).

View API Gateway Metrics in the CloudWatch Console

To view metrics using the CloudWatch console

Metrics are grouped first by the service namespace, and then by the various dimension combinations within each namespace.

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. If necessary, change the region. From the navigation bar, select the region where your AWS resources reside. For more information, see [Regions and Endpoints](#).
3. In the navigation pane, choose **Metrics**.
4. In the **CloudWatch Metrics by Category** pane, under the metrics category for API Gateway, select a metrics category, and then in the upper pane, scroll down to view the full list of metrics.

To view metrics using the AWS CLI

- At a command prompt, use the following command:

```
aws cloudwatch list-metrics --namespace "AWS/ApiGateway"
```

Monitoring Tools in AWS

AWS provides various tools that you can use to monitor API Gateway. You can configure some of these tools to do the monitoring for you automatically, while other tools require manual intervention. We recommend that you automate monitoring tasks as much as possible.

Automated Monitoring Tools in AWS

You can use the following automated monitoring tools to watch API Gateway and report when something is wrong:

- **Amazon CloudWatch Alarms** – Watch a single metric over a time period that you specify, and perform one or more actions based on the value of the metric relative to a given threshold over a number of time periods. The action is a notification sent to an Amazon Simple Notification Service (Amazon SNS) topic or Auto Scaling policy. CloudWatch alarms do not invoke actions simply because they are in a particular state; the state must have changed and been maintained for a specified number of periods. For more information, see [Monitor API execution with Amazon CloudWatch](#) (p. 334).
- **Amazon CloudWatch Logs** – Monitor, store, and access your log files from AWS CloudTrail or other sources. For more information, see [Monitoring Log Files](#) in the *Amazon CloudWatch User Guide*.
- **Amazon CloudWatch Events** – Match events and route them to one or more target functions or streams to make changes, capture state information, and take corrective action. For more information, see [Using Events](#) in the *Amazon CloudWatch User Guide*.
- **AWS CloudTrail Log Monitoring** – Share log files between accounts, monitor CloudTrail log files in real time by sending them to CloudWatch Logs, write log processing applications in Java, and validate that your log files have not changed after delivery by CloudTrail. For more information, see [Working with CloudTrail Log Files](#) in the *AWS CloudTrail User Guide*.

Manual Monitoring Tools

Another important part of monitoring API Gateway involves manually monitoring those items that the CloudWatch alarms don't cover. The API Gateway, CloudWatch, and other AWS console dashboards provide an at-a-glance view of the state of your AWS environment. We recommend that you also check the log files on API execution.

- API Gateway dashboard shows the following statistics for a given API stage during a specified period of time:
 - **API Calls**

- **Cache Hit**, only when API caching is enabled.
- **Cache Miss**, only when API caching is enabled.
- **Latency**
- **Integration Latency**
- **4XX Error**
- **5XX Error**
- The CloudWatch home page shows:
 - Current alarms and status
 - Graphs of alarms and resources
 - Service health status

In addition, you can use CloudWatch to do the following:

- Create [customized dashboards](#) to monitor the services you care about
- Graph metric data to troubleshoot issues and discover trends
- Search and browse all your AWS resource metrics
- Create and edit alarms to be notified of problems

Creating CloudWatch Alarms to Monitor API Gateway

You can create a CloudWatch alarm that sends an Amazon SNS message when the alarm changes state. An alarm watches a single metric over a time period you specify, and performs one or more actions based on the value of the metric relative to a given threshold over a number of time periods. The action is a notification sent to an Amazon SNS topic or Auto Scaling policy. Alarms invoke actions for sustained state changes only. CloudWatch alarms do not invoke actions simply because they are in a particular state; the state must have changed and been maintained for a specified number of periods.

Selling an API Gateway API through AWS Marketplace

After you build, test, and deploy your API, you can package it in an API Gateway [usage plan](#) and sell the plan as a Software as a Service (SaaS) product through AWS Marketplace. API buyers subscribing to your product offering are billed by AWS Marketplace based on the number of requests made to the [usage plan](#) (p. 212).

To sell your API on AWS Marketplace, you must set up the sales channel to integrate AWS Marketplace with API Gateway. Generally speaking, this involves listing your product on AWS Marketplace, setting up an IAM role with appropriate policies to allow API Gateway to send usage metrics to AWS Marketplace, associating an AWS Marketplace product with an API Gateway usage plan, and associating an AWS Marketplace buyer with an API Gateway API key. Details are discussed in the following sections.

To enable your customers to buy your product on AWS Marketplace, you must register your developer portal (an external application) with AWS Marketplace. The developer portal must handle the subscription requests that are redirected from the AWS Marketplace console.

For more information about selling your API as a SaaS product on AWS Marketplace, see [AWS Marketplace SaaS Subscriptions - Seller Integration Guide](#).

Topics

- [Initialize AWS Marketplace Integration with API Gateway](#) (p. 339)
- [Handle Customer Subscription to Usage Plans](#) (p. 341)

Initialize AWS Marketplace Integration with API Gateway

The following tasks are for one-time initialization of AWS Marketplace integration with API Gateway, which enables you to sell your API as a SaaS product.

List a Product on AWS Marketplace

To list your usage plan as a SaaS product, submit a product load form through [AWS Marketplace](#). The product must contain a dimension named `apigateway` of the `requests` type. This dimension defines the price-per-request and is used by API Gateway to meter requests to your API.

Create the Metering Role

Create an IAM role named `ApiGatewayMarketplaceMeteringRole` with the following execution policy and trust policy. This role allows API Gateway to send usage metrics to AWS Marketplace on your behalf.

Execution Policy of the Metering Role

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "aws-marketplace:BatchMeterUsage",
        "aws-marketplace:ResolveCustomer"
      ],
      "Resource": "*",
      "Effect": "Allow"
    }
  ]
}
```

Trusted Relationship Policy of the Metering Role

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

Associate Usage Plan with AWS Marketplace Product

When you list a product on AWS Marketplace, you receive an AWS Marketplace product code. To integrate API Gateway with AWS Marketplace, associate your usage plan with the AWS Marketplace product code. You enable the association by setting the API Gateway `UsagePlan`'s `productCode` field to your AWS Marketplace product code, using the API Gateway console, the API Gateway REST API, the AWS CLI for API Gateway, or AWS SDK for API Gateway. The following code example uses the API Gateway REST API:

```
PATCH /usageplans/USAGE_PLAN_ID
Host: apigateway.region.amazonaws.com
Authorization: ...

{
  "patchOperations" : [{
```



```
    "path" : "/productCode",  
    "value" : "MARKETPLACE_PRODUCT_CODE",  
    "op" : "replace"  
  }  
}]  
}
```

Handle Customer Subscription to Usage Plans

The following tasks are handled by your developer portal application.

When a customer subscribes to your product through AWS Marketplace, AWS Marketplace forwards a `POST` request to the SaaS subscriptions URL that you registered when listing your product on AWS Marketplace. The `POST` request comes with an `x-amzn-marketplace-token` header parameter containing buyer information. Follow the instructions in the Register Application section of the [SaaS Seller Integration Guide](#) to handle this redirect in your developer portal application.

Responding to a customer's subscribing request, AWS Marketplace sends a `subscribe-success` notification to an Amazon SNS topic that you can subscribe to (See Step 6.4 of the [SaaS Seller Integration Guide](#)). To accept the customer subscription request, you handle the `subscribe-success` notification by creating or retrieving an API Gateway API key for the customer, associating the customer's AWS Marketplace-provisioned `customerId` with the API keys, and then adding the API key to your usage plan.

When the customer's subscription request completes, the developer portal application should present the customer with the associated API key and inform the customer that the API key must be included in the `x-api-key` header in requests to the API.

When a customer cancels a subscription to a usage plan, AWS Marketplace sends an `unsubscribe-success` notification to the SNS topic. To complete the process of unsubscribing the customer, you handle the `unsubscribe-success` notification by removing the customer's API keys from the usage plan.

Authorize a Customer to Access a Usage Plan

To authorize access to your usage plan for a given customer, use the API Gateway API to fetch or create an API key for the customer and add the API key to the usage plan.

The following example shows how to call the API Gateway REST API to create a new API key with a specific AWS Marketplace `customerId` value (`MARKETPLACE_CUSTOMER_ID`).

```
POST apikeys HTTP/1.1  
Host: apigateway.region.amazonaws.com  
Authorization: ...  
  
{  
  "name" : "my_api_key",  
  "description" : "My API key",  
  "enabled" : "false",  
  "stageKeys" : [ {  
    "restApiId" : "uycll6xg9a",  
    "stageName" : "prod"  
  } ],  
  "customerId" : "MARKETPLACE_CUSTOMER_ID"  
}
```

The following example shows how to get an API key with a specific AWS Marketplace `customerId` value (`MARKETPLACE_CUSTOMER_ID`).

```
GET /apikeys?customerId=MARKETPLACE_CUSTOMER_ID HTTP/1.1
Host: apigateway.region.amazonaws.com
Authorization: ...
```

To add an API key to a usage plan, create a `UsagePlanKey` with the API key for the relevant usage plan. The following example shows how to accomplish this using the API Gateway REST API, where `n371pt` is the usage plan ID and `q5ugs7qjjh` is an example API `keyId` returned from the preceding examples.

```
POST /usageplans/n371pt/keys HTTP/1.1
Host: apigateway.region.amazonaws.com
Authorization: ...
```

```
{
  "keyId": "q5ugs7qjjh",
  "keyType": "API_KEY"
}
```

Associate a Customer with an API Key

You must update the `ApiKey`'s `customerId` field to the AWS Marketplace customer ID of the customer. This associates the API key with the AWS Marketplace customer, which enables metering and billing for the buyer. The following code example calls the API Gateway REST API to do that.

```
PATCH /apikeys/q5ugs7qjjh
Host: apigateway.region.amazonaws.com
Authorization: ...

{
  "patchOperations" : [{
    "path" : "/customerId",
    "value" : "MARKETPLACE_CUSTOMER_ID",
    "op" : "replace"
  }]
}
```

Tutorials for Using Amazon API Gateway

The following tutorials provide hands-on exercises to help you learn about API Gateway.

Topics

- [Create an API Gateway API as an AWS Lambda Proxy \(p. 343\)](#)
- [Create an API as an Amazon S3 Proxy \(p. 359\)](#)
- [Create an API Gateway API as an Amazon Kinesis Proxy \(p. 388\)](#)

Create an API Gateway API as an AWS Lambda Proxy

Note

To integrate your API Gateway API with Lambda, you must choose a region where both the API Gateway and Lambda services are available. For region availability, see [Regions and Endpoints](#).

If your API makes only synchronous calls to Lambda functions in the back end, you should use the **Lambda Function** integration type. For instructions, see [Build an API to Expose a Lambda Function \(p. 44\)](#).

If your API makes asynchronous calls to Lambda functions, you must use the **AWS Service Proxy** integration type described in this section. The instructions apply to requests for synchronous Lambda function invocations as well. For the asynchronous invocation, you must explicitly add the `X-Amz-Invocation-Type:Event` header to the integration request. For the synchronous invocation, you can add the `X-Amz-Invocation-Type:RequestResponse` header to the integration request or leave it unspecified. The following example shows the integration request of an asynchronous Lambda function invocation:

```
POST /2015-03-31/functions/FunctionArn/invocations?Qualifier=Qualifier
HTTP/1.1
X-Amz-Invocation-Type: Event
...
Authorization: ...
Content-Type: application/json
Content-Length: PayloadSize

Payload
```

In this example, *FunctionArn* is the ARN of the Lambda function to be invoked. The `Authorization` header is required by secure invocation of Lambda functions over HTTPS. For more information, see the [Invoke](#) action in the *AWS Lambda Developer Guide*.

To illustrate how to create and configure an API as an AWS service proxy for Lambda, we will create a Lambda function (`calc`) that performs addition (+), subtraction (-), multiplication (*), and division (/). When a client submits a method request to perform any of these operations, API Gateway will post the corresponding integration request to call the specified Lambda function, passing the required input (two operands and one operator) as a JSON payload. A synchronous call will return the result, if any, as the JSON payload. An asynchronous call will return no data.

The API can expose a GET or POST method on the `/calc` resource to invoke the Lambda function. With the GET method, a client supplies the input to the back-end Lambda function through three query string parameters (`operand1`, `operand2`, and `operator`). These are mapped to the JSON payload of the integration request. With the POST method, a client provides the input to the Lambda function as a JSON payload of the method request, which is then passed through to the integration request. Alternatively, the API can expose a GET method on the `/calc/{operand1}/{operand2}/{operator}` resource. With this method, the client specifies the Lambda function input as the values of the path parameters. Parameter mappings and mapping templates are used to translate the method request data into the Lambda function input and to translate the output from the integration responses to the method response.

This section provides more detailed discussions for the following tasks:

- Create the `calc` Lambda function to implement the arithmetic operations, accepting and returning JSON-formatted input and output.
- Expose GET on the `/calc` resource to invoke the Lambda function, supplying the input as query strings.
- Expose POST on the `/calc` resource to invoke the Lambda function, supplying the input in the payload.
- Expose GET on the `/calc/{operand1}/{operand2}/{operator}` resource to invoke the Lambda function, specifying the input in the path parameters.

You can import the sample API as a Lambda proxy from the [Swagger Definitions of a Sample API as Lambda Proxy \(p. 356\)](#). To do so, copy the Swagger definition, paste it into a file, and use the [API Gateway Swagger Importer](#). For more information, see [Getting Started with the API Gateway Swagger Importer](#).

To use the API Gateway console to create the API, you must first sign up for an AWS account.

If you do not have an AWS account, use the following procedure to create one.

To sign up for AWS

1. Open <http://aws.amazon.com/> and choose **Create an AWS Account**.

2. Follow the online instructions.

To allow the API to invoke Lambda functions, you must have an IAM role that has appropriate IAM policies attached to it. The next section describes how to verify and to create, if necessary, the required IAM role and policies.

Topics

- [Set Up an IAM Role and Policy for an API to Invoke Lambda Functions](#) (p. 345)
- [Create a Lambda Function in the Back End](#) (p. 346)
- [Create API Resources for the Lambda Function](#) (p. 347)
- [Create a GET Method with Query Strings to Call the Lambda Function](#) (p. 347)
- [Create a POST Method with a JSON Payload to Call the Lambda Function](#) (p. 350)
- [Create a GET Method with Path Parameters to Call the Lambda Function](#) (p. 352)
- [A Sample API as a Lambda Proxy in Swagger with API Gateway Extensions](#) (p. 356)

Set Up an IAM Role and Policy for an API to Invoke Lambda Functions

The API will use the [InvokeFunction](#) action to call a Lambda function. At minimum, you must attach the following IAM policy to an IAM role for API Gateway to assume the policy.

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

If you do not enact this policy, the API caller will receive a 500 Internal Server Error response. The response contains the "Invalid permissions on Lambda function" error message. For a complete list of error messages returned by Lambda, see the [Invoke](#) topic.

An API Gateway assumable role is an IAM role with the following trusted relationship:

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

```
}  
]  
}
```

Create a Lambda Function in the Back End

Copy the following Lambda function and paste it into the code editor in the Lambda console.

```
exports.handler = function(event, context) {  
  //console.log('Received event:', JSON.stringify(event, null, 2));  
  var res = {};  
  res.a = event.a;  
  res.b = event.b;  
  res.op = event.op;  
  
  switch(event.op)  
  {  
    case "+":  
      res.c = Number(event.a) + Number(event.b);  
      break;  
    case "-":  
      res.c = Number(event.a) - Number(event.b);  
      break;  
    case "*":  
      res.c = Number(event.a) * Number(event.b);  
      break;  
    case "/":  
      res.c = Number(event.b)===0 ? NaN : Number(event.a) /  
Number(event.b);  
      break;  
    default:  
      res.c = "Invalid op";  
  }  
  context.succeed(res);  
};
```

This function requires two operands (*a* and *b*) and an operator (*op*) from the *event* input parameter. The input is a JSON object of the following format:

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

This function returns the calculated result (*c*) and the input. For an invalid input, the function returns either the null value or the "Invalid op" string as the result. The output is of the following JSON format:

```
{
```

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

You should test the function in the Lambda console before integrating it with the API, which is created next.

Create API Resources for the Lambda Function

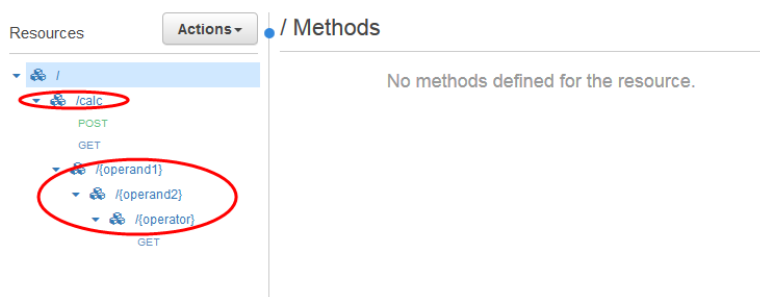
The following procedure describes how to create API resources for the Lambda function.

To create API resources for Lambda functions

1. In the API Gateway console, create an API named **LambdaGate**. You can create child resources to represent different Lambda functions; in the following, you will create a single child resource of the API root.
2. For the simple calculator function you created, create the **/calc** resource off the API's root. You will expose the GET and POST methods on this resource for the client to invoke the back-end Lambda function, supplying the required input as query string parameters (to be declared as ?operand1=...&operand2=...&operator=...) in the GET request and as a JSON payload in the POST request, respectively.

You will also create the **/calc/{operand1}/{operand2}/{operator}** to expose the GET method to invoke the Lambda function and to supply the required input as the three path parameters (**operand1**, **operand2**, and **operator**).

We will show how to apply API Gateway request and response data mapping to normalize the input to the back end Lambda function.



Create a GET Method with Query Strings to Call the Lambda Function

Use the following steps to expose a GET method with query strings to call a Lambda function.

To set up the GET method with query strings to invoke the Lambda function

1. Choose **Create Method** in the API Gateway console to create a GET method for the API's **/calc** resource.

In the method's **Set up** pane, configure the method with the following settings.

Amazon API Gateway Developer Guide

Create a GET Method with Query Strings to Call the Lambda Function

[← Method Execution](#) /calc - GET - Integration Request

Provide information about the target backend that this method will call and whether the incoming request data should be modified.

Integration type Lambda Function
 HTTP Proxy
 Mock Integration
 AWS Service Proxy

AWS Region us-west-2 [✎](#)

AWS Service Lambda [✎](#)

AWS Subdomain [✎](#)

HTTP method POST [✎](#)

Path override /2015-03-31/functions/arn:aws:lambda:us-west-2:██████████:function:Calc/invocations [✎](#)

Execution role arn:aws:iam::██████████:role/apigAwsProxyRole [✎](#)

Credentials cache Do not add caller credentials to cache key [✎](#)

▶ URL Path Parameters

▶ URL Query String Parameters

▶ HTTP Headers

▼ Mapping Templates

You must use the POST method for the integration request when calling a Lambda function, although you can use any other HTTP verbs for the method request.

The **Path override** value must be the URL path of the Lambda [Invoke](#) action. The path is of the following format:

```
/2015-03-31/functions/FunctionName/invocations?Qualifier=version
```

where *FunctionName* is the ARN of the Lambda function to be invoked. The optional *Qualifier* query string can be used to select a version of the function. If it not specified, the `$LATEST` version will be used.

You can also add the `X-Amz-Invocation-Type: Event | RequestResponse | DryRun` header to have the action invoked asynchronously, as request and response, or as a test run, respectively. If the header is not specified, the action will be invoked as request and response. For the example shown here, this header has the default value.

We will come back to setting up **Mapping Templates** after setting up the query string parameters to hold the input data for the Lambda function.

2. In **Method Request** for the **GET** method on `/calc`, expand the **URL Query String Parameters** section. Choose **Add query string** to add the **operand1**, **operand2**, and **operator** query string parameters.

Amazon API Gateway Developer Guide

Create a GET Method with Query Strings to Call the Lambda Function

[← Method Execution](#) /calc - GET - Method Request

Provide information about this method's authorization settings and the parameters it can receive.

Authorization Settings

Authorization NONE

API Key Required false

URL Query String Parameters

Name	Caching	
operand2	<input type="checkbox"/>	
operator	<input type="checkbox"/>	
operand1	<input type="checkbox"/>	

[+ Add query string](#)

▶ HTTP Request Headers

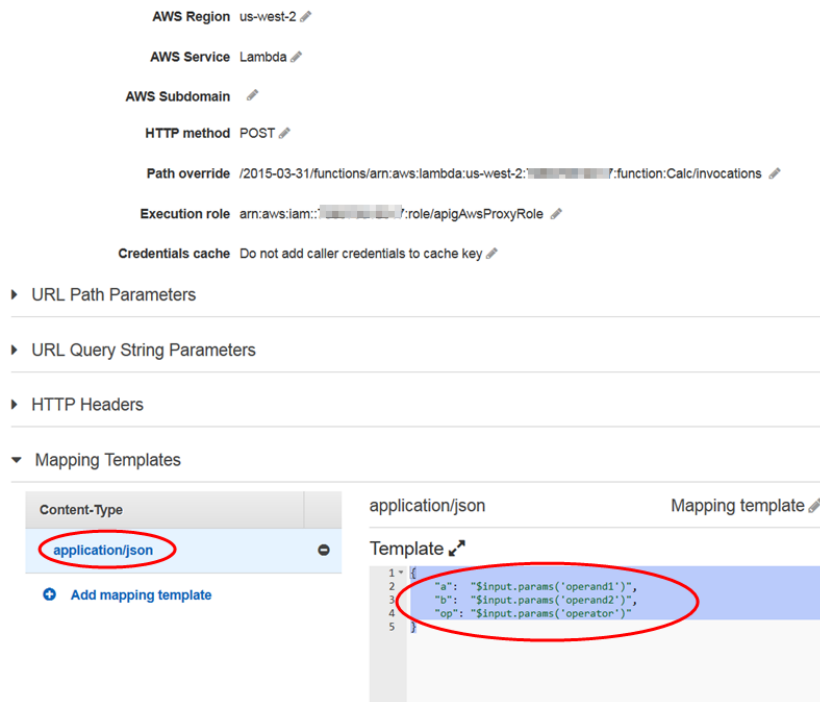
▶ Request Models [Create a Model](#)

- Go back to **Integration Request**. Expand the **Mapping Templates** section. If necessary, in **Content-Type**, under **application/json**, choose **Add mapping template**. Type the following in the **Mapping template** editor:

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

Amazon API Gateway Developer Guide

Create a POST Method with a JSON Payload to Call the Lambda Function



This template maps the three query string parameters declared in **Method Request** into designated property values of the JSON object as the input to the back-end Lambda function. The transformed JSON object will be included as the integration request payload.

4. You can now choose **Test** to verify that the GET method on the **/calc** resource has been properly set up to invoke the Lambda function.

Create a POST Method with a JSON Payload to Call the Lambda Function

The following steps describe how to expose a POST method with a JSON payload.

To set up the POST method with a JSON payload to invoke a Lambda function

1. Choose **Create Method** in the API Gateway console to create a POST method for the **LambdaGate** API's **/calc** resource.

In the method's **Set Up** panel, configure the POST method with the following settings.

Amazon API Gateway Developer Guide

Create a POST Method with a JSON Payload to Call the Lambda Function

[← Method Execution](#) /calc - POST - Integration Request

Provide information about the target backend that this method will call and whether the incoming request data should be modified.

Integration type Lambda Function
 HTTP Proxy
 Mock Integration
 AWS Service Proxy

AWS Region us-west-2 [↗](#)

AWS Service Lambda [↗](#)

AWS Subdomain [↗](#)

HTTP method POST [↗](#)

Path override /2015-03-31/functions/arn:aws:lambda:us-west-2:7: :function:Calc/invocations [↗](#)

Execution role arn:aws:iam:: :role/apigAwsProxyRole [↗](#)

Credentials cache Do not add caller credentials to cache key [↗](#)

▶ URL Path Parameters

▶ URL Query String Parameters

▶ HTTP Headers

▶ Mapping Templates

Using a POST request with a JSON payload is the simplest way to invoke a Lambda function, because no mappings are needed.

2. You can now choose **Test** to verify the POST method works as expected. The following input:

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

should produce the following output:

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

If you would like to implement POST as an asynchronous call, you can add an `InvocationType:Event` header in the method request and map it to the `X-Amz-Invocation-Type` header in the integration request, using the header mapping expression `of method.request.header.InvocationType`. You must inform the clients to include the `InvocationType:Event` header in the method request. Alternatively, you can set the `X-Amz-`

`Invocation-Type` header with the `'Event'` string literal in the integration request, without requiring the client to include the header. The asynchronous call will return an empty response, instead.

Create a GET Method with Path Parameters to Call the Lambda Function

The following steps describe how to set up the GET method with path parameters to call the Lambda function.

To set up the GET method with URL path parameters to invoke the Lambda function

1. Choose **Create Method** in the API Gateway console to create a GET method for the API's `/calc/{operand1}/{operand2}/{operator}` resource.

In the method's **Set up** pane, configure this GET method with the following settings.

← Method Execution /calc/{operand1}/{operand2}/{operator} - GET - Integration ...

Provide information about the target backend that this method will call and whether the incoming request data should be modified.

Integration type Lambda Function
 HTTP Proxy
 Mock Integration
 AWS Service Proxy

AWS Region us-west-2

AWS Service Lambda

AWS Subdomain

HTTP method POST

Path override /2015-03-31/functions/arn:aws:lambda:us-west-2:...:function:Calc/invocations

Execution role arn:aws:iam::...:role/apigAwsProxyRole

Credentials cache Do not add caller credentials to cache key

▶ URL Path Parameters

▶ URL Query String Parameters

▶ HTTP Headers

▼ Mapping Templates

Content-Type

Next, we will set up **Mapping Templates** to translate the URL path parameters into the integration request JSON payload as the input to the Lambda function.

2. In **Method Request** for the **GET** method on `/calc/{operand1}/{operand2}/{operator}`, expand the **Request Paths** section to verify that the path parameters are there.

Amazon API Gateway Developer Guide

Create a GET Method with Path Parameters to Call the Lambda Function

← Method Execution /calc/{operand1}/{operand2}/{operator} - GET - Method Re...

Provide information about this method's authorization settings and the parameters it can receive.

Authorization Settings

Authorization NONE ⓘ

API Key Required false ⓘ

Request Paths

Name	Caching
operand1	<input type="checkbox"/>
operand2	<input type="checkbox"/>
operator	<input type="checkbox"/>

URL Query String Parameters

HTTP Request Headers

- Go back to **Integration Request**. Expand the **Mapping Templates** section. If necessary, in **Content-Type**, under **application/json**, choose **Add mapping template**.

Integration type Lambda Function

HTTP Proxy

Mock Integration

AWS Service Proxy

AWS Region us-west-2 ⓘ

AWS Service Lambda ⓘ

AWS Subdomain ⓘ

HTTP method POST ⓘ

Path override /2015-03-31/functions/arn:aws:lambda:us-west-2:██████████:function:Calc/invocations ⓘ

Execution role arn:aws:iam::██████████:role/apigAwsProxyRole ⓘ

Credentials cache Do not add caller credentials to cache key ⓘ

URL Path Parameters

URL Query String Parameters

HTTP Headers

Mapping Templates

Content-Type application/json Mapping template ⓘ

application/json

Template ⓘ Select a model to generate a template

```
1 {
2   "a": "$input.params('operand1')",
3   "b": "$input.params('operand2')",
4   "op": "#if($input.params('operator')=='%2F')/#{else}$input
5   params('operator')#end
6 }
```

Type the following in the **Mapping Template** editor:

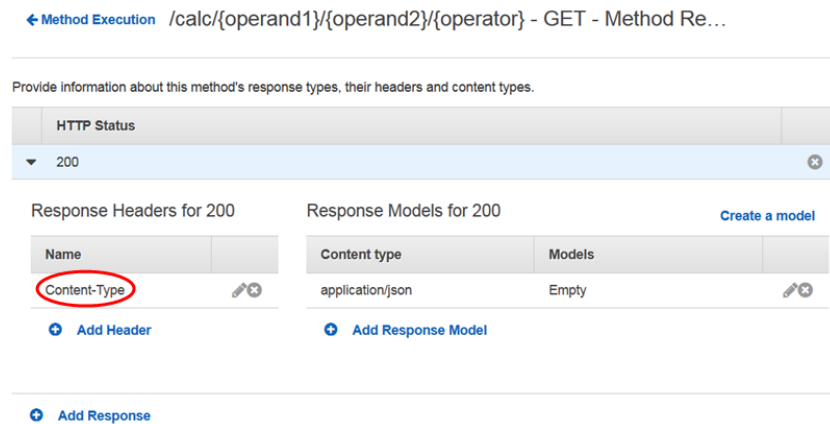
```
{
  "a": "$input.params('operand1')",
```

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

This template maps the three URL path parameters, declared when the `/calc/{operand1}/{operand2}/{operator}` resource was created, into designated property values of the JSON object. Because URL paths must be URL-encoded, the division operator must be specified as `%2F` instead of `/`. This template maps these translations as well. The transformed JSON object will be included as the integration request payload.

- As another exercise, we demonstrate how to translate the JSON returned from the Lambda function to show the output as a plain text string to the caller. This involves resetting the method request's Content-Type header to "text/plain" and providing a mapping template to translate the JSON output into a plain string.

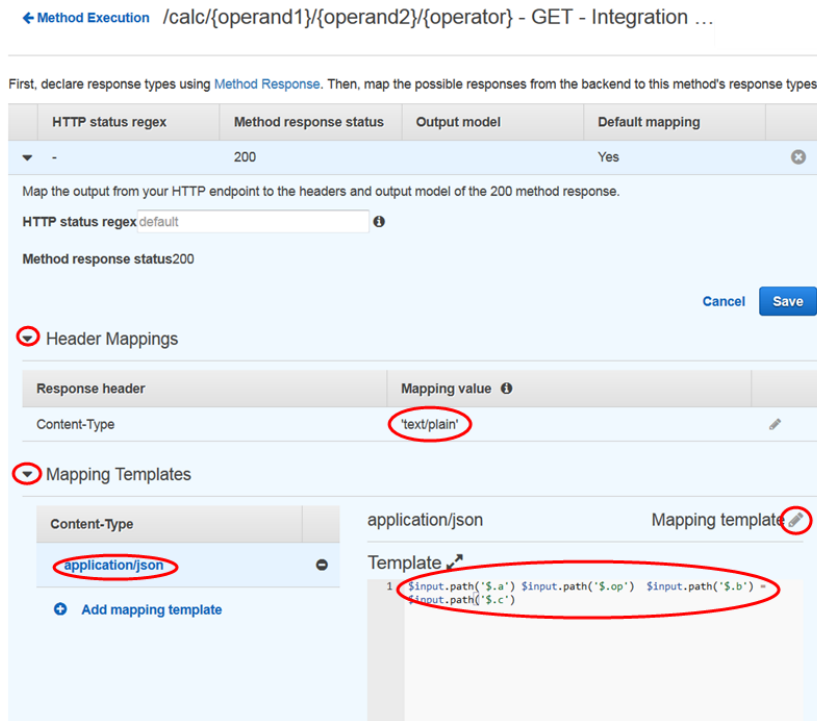
First, make sure that **Content-Type** header is included in the **Response Headers for 200** section in **Method Response**.



- In **Integration Response**, expand the 200 method response entry. Expand the **Header Mappings** section. In **Mapping value** for **Content-Type**, type `'text/plain'`. This header mapping expression overrides the Content-Type header with a literal string, which must be enclosed within a pair of single quotes.

Amazon API Gateway Developer Guide

Create a GET Method with Path Parameters to Call the Lambda Function



Next, expand the **Mapping Templates** section, highlight the **application/json** entry under the **Content-Type** header (of integration response), open the **Mapping template** editor, enter and save the following mapping script:

```
$input.path('$$.a') $input.path('$$.op') $input.path('$$.b') = $input.path('$$.c')
```

6. Choose **Test** to verify the GET method on the `/calc/{operand1}/{operand2}/{operator}` works as expected. The following request URL:

```
/calc/1/2/%2F
```

should produce the following plain text output:

```
1 / 2 = 0.5
```

Note

As part of a URL, the division operator (/) is URL-encoded (%2F).

7. After testing the API using the Test Invoke in the API Gateway console, you must deploy the API to make it public available. If you update the API, such as adding, modifying or deleting a resource or method, updating any data mapping, you must redeploy the API to make the new features or updates available.

A Sample API as a Lambda Proxy in Swagger with API Gateway Extensions

```
{
  "swagger": "2.0",
  "info": {
    "version": "2016-02-23T05:35:54Z",
    "title": "LambdaGate"
  },
  "host": "a123456789.execute-api.us-east-1.amazonaws.com",
  "basePath": "/test",
  "schemes": [
    "https"
  ],
  "paths": {
    "/calc": {
      "get": {
        "produces": [
          "application/json"
        ],
        "parameters": [
          {
            "name": "operand2",
            "in": "query",
            "required": false,
            "type": "string"
          },
          {
            "name": "operator",
            "in": "query",
            "required": false,
            "type": "string"
          },
          {
            "name": "operand1",
            "in": "query",
            "required": false,
            "type": "string"
          }
        ],
        "responses": {
          "200": {
            "description": "200 response",
            "schema": {
              "$ref": "#/definitions/Empty"
            },
            "headers": {
              "operand_1": {
                "type": "string"
              },
              "operand_2": {
                "type": "string"
              },
              "operator": {
                "type": "string"
              }
            }
          }
        }
      }
    }
  }
}
```



```

    }
  },
  "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 => $res.c\"\n}"
        }
      },
      "requestTemplates": {
        "application/json": "{\n  \"a\":
\"$input.params('operand1')\", \n  \"b\": \"$input.params('operand2')\",
\n  \"op\": \"$input.params('operator')\" \n}"
      },
      "uri": "arn:aws:apigateway:us-west-2:lambda:path//2015-03-31/
functions/arn:aws:lambda:us-west-2:123456789012:function:Calc/invocations",
      "httpMethod": "POST",
      "type": "aws"
    }
  },
  "post": {
    "produces": [
      "application/json"
    ],
    "parameters": [],
    "responses": {
      "200": {
        "description": "200 response",
        "schema": {
          "$ref": "#/definitions/Empty"
        },
        "headers": {}
      }
    }
  },
  "x-amazon-apigateway-integration": {
    "credentials": "arn:aws:iam::123456789012:role/apigAwsProxyRole",
    "responses": {
      "default": {
        "statusCode": "200",
        "responseTemplates": {
          "application/json": "__passthrough__"
        }
      }
    },
    "uri": "arn:aws:apigateway:us-west-2:lambda:path//2015-03-31/
functions/arn:aws:lambda:us-west-2:123456789012:function:Calc/invocations",

```

```

    "httpMethod": "POST",
    "type": "aws"
  }
},
"/calc/{operand1}/{operand2}/{operator}": {
  "get": {
    "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": {
          "$ref": "#/definitions/Empty"
        },
        "headers": {
          "Content-Type": {
            "type": "string"
          }
        }
      }
    }
  },
  "x-amazon-apigateway-integration": {
    "credentials": "arn:aws:iam::123456789012:role/apigAwsProxyRole",
    "responses": {
      "default": {
        "statusCode": "200",
        "responseParameters": {
          "method.response.header.Content-Type": "'text/plain'"
        },
        "responseTemplates": {
          "application/json": "\${input.path('$a')} ${input.path('$op')}
${input.path('$b')} = ${input.path('$c')}\n"
        }
      }
    }
  },
  "requestTemplates": {

```

```
        "application/json": "\n{\n  \"a\":\n    \"${input.params('operand1')}\",\n    \"b\": \"${input.params('operand2')}\",\n    \"op\": \"#if(${input.params('operator')}=='%2F')\"/\n    \"#{else}\"${input.params('operator')}\"#end\n  }\n}\",\n        "uri": "arn:aws:apigateway:us-west-2:lambda:path//2015-03-31/\nfunctions/arn:aws:lambda:us-west-2:123456789012:function:Calc/invocations",\n        "httpMethod": "POST",\n        "type": "aws"\n      }\n    }\n  },\n  "definitions": {\n    "Empty": {\n      "type": "object"\n    }\n  }\n}
```

Create an API as an Amazon S3 Proxy

As an example to showcase using an API in API Gateway to proxy Amazon S3, this section describes how to create and configure an API to expose the following Amazon S3 operations:

- Expose GET on the API's root resource to [list all of the Amazon S3 buckets of a caller](#).
- Expose GET on a Folder resource to [view a list of all of the objects in an Amazon S3 bucket](#).
- Expose PUT on a Folder resource to [add a bucket to Amazon S3](#).
- Expose DELETE on a Folder resource to [remove a bucket from Amazon S3](#).
- Expose GET on a Folder/Item resource to [view or download an object from an Amazon S3 bucket](#).
- Expose PUT on a Folder/Item resource to [upload an object to an Amazon S3 bucket](#).
- Expose HEAD on a Folder/Item resource to [get object metadata in an Amazon S3 bucket](#).
- Expose DELETE on a Folder/Item resource to [remove an object from an Amazon S3 bucket](#).

Note

To integrate your API Gateway API with Amazon S3, you must choose a region where both the API Gateway and Amazon S3 services are available. For region availability, see [Regions and Endpoints](#).

You may want to import the sample API as an Amazon S3 proxy, as shown in [Swagger Definitions of the Sample API as an Amazon S3 Proxy \(p. 378\)](#). For instructions on how to import an API using the Swagger definition, see [Import an API \(p. 142\)](#).

To use the API Gateway console to create the API, you must first sign up for an AWS account.

If you do not have an AWS account, use the following procedure to create one.

To sign up for AWS

1. Open <http://aws.amazon.com/> and choose **Create an AWS Account**.
2. Follow the online instructions.

Topics

- [Set Up IAM Permissions for the API to Invoke Amazon S3 Actions \(p. 360\)](#)
- [Create API Resources to Represent Amazon S3 Resources \(p. 361\)](#)
- [Expose an API Method to List the Caller's Amazon S3 Buckets \(p. 362\)](#)
- [Expose API Methods to Access an Amazon S3 Bucket \(p. 369\)](#)
- [Expose API Methods to Access an Amazon S3 Object in a Bucket \(p. 372\)](#)
- [Call the API Using a REST API Client \(p. 374\)](#)
- [Swagger Definitions of the Sample API as an Amazon S3 Proxy \(p. 378\)](#)

Set Up IAM Permissions for the API to Invoke Amazon S3 Actions

To allow the API to invoke required Amazon S3 actions, you must have appropriate IAM policies attached to an IAM role. The next section describes how to verify and to create, if necessary, the required IAM role and policies.

For your API to view or list Amazon S3 buckets and objects, you can use the IAM-provided `AmazonS3ReadOnlyAccess` policy in the IAM role. The ARN of this policy is `arn:aws:iam::aws:policy/AmazonS3ReadOnlyAccess`, which is as shown as follows:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:Get*",
        "s3:List*"
      ],
      "Resource": "*"
    }
  ]
}
```

This policy document states that any of the Amazon S3 `Get*` and `List*` actions can be invoked on any of the Amazon S3 resources.

For your API to update Amazon S3 buckets and objects, you can use a custom policy for any of the Amazon S3 `Put*` actions as shown as follows:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "s3:Put*",
      "Resource": "*"
    }
  ]
}
```

For your API to work with Amazon S3 `Get*`, `List*` and `Put*` actions, you can add the above read-only and put-only policies to the IAM role.

For your API to invoke the Amazon S3 `Post*` actions, you must use an Allow policy for the `s3:Post*` actions in the IAM role. For a complete list of Amazon S3 actions, see [Specifying Amazon S3 Permissions in a Policy](#).

For your API to create, view, update, and delete buckets and objects in Amazon S3, you can use the IAM -provided `AmazonS3FullAccess` policy in the IAM role. The ARN is `arn:aws:iam::aws:policy/AmazonS3FullAccess`.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "s3:*",
      "Resource": "*"
    }
  ]
}
```

Having chosen the desired IAM policies to use, create an IAM role and attach to it the policies. The resulting IAM role must contain the following trust policy for API Gateway to assume this role at runtime.

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

When using the IAM console to create the role, choose the **Amazon API Gateway** role type to ensure that this trust policy is automatically included.

Create API Resources to Represent Amazon S3 Resources

We will use the API's root (`/`) resource as the container of an authenticated caller's Amazon S3 buckets. We will also create a `Folder` and `Item` resources to represent a particular Amazon S3 bucket and a particular Amazon S3 object, respectively. The folder name and object key will be specified, in the form of path parameters as part of a request URL, by the caller.

To create an API resource that exposes the Amazon S3 service features

1. In the API Gateway console, create an API named **MyS3**. This API's root resource (`/`) represents the Amazon S3 service.
2. Under the API's root resource, create a child resource named **Folder** and set the required **Resource Path** as `/{folder}`.

- For the API's **Folder** resource, create an **Item** child resource. Set the required **Resource Path** as **/{item}**.

The screenshot shows the AWS API Gateway console interface for creating a new child resource. On the left, the 'Resources' panel shows a tree view with a root resource '/' and a child resource '/{folder}'. The '/{folder}' resource is selected and highlighted. The main panel is titled 'New Child Resource' and contains the following configuration options:

- Configure as proxy resource:** (unchecked)
- Resource Name*:** Item
- Resource Path*:** /{folder}/ {item} (circled in red)
- Enable API Gateway CORS:** (unchecked)

Below the configuration fields, there is a text block explaining path parameters: "You can add path parameters using brackets. For example, a resource path {username} represents a path parameter 'username'. Configuring /{folder}/{proxy+} as a proxy resource catches all requests to its sub-resources. For example, for a GET request to /{folder}/foo. To handle requests to add a new ANY method on the /{folder} resource."

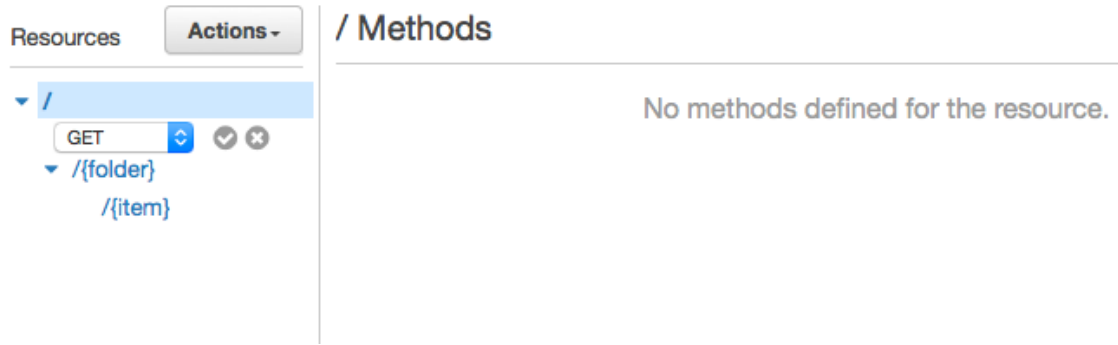
At the bottom of the form, there is a legend for required fields (* Required), a 'Cancel' button, and a 'Create' button (highlighted with a red border).

Expose an API Method to List the Caller's Amazon S3 Buckets

Getting the list of Amazon S3 buckets of the caller involves invoking the [GET Service](#) action on Amazon S3. On the API's root resource, (/), create the GET method. Configure the GET method to integrate with the Amazon S3, as follows.

To create and initialize the API's `GET /` method

- Choose **Create method** on the root node (/) from the **Actions** drop-down menu at the top-right corner of the **Resources** panel.
- Choose the `GET` from the drop-down list of HTTP verbs, and choose the check-mark icon to start creating the method.



3. In the **/ - GET - Setup** pane, choose **AWS Service Proxy** for the **Integration type**.
4. From the list, choose an **AWS Region**.
5. From **AWS Service**, choose **S3**.
6. From **HTTP method**, choose **GET**.
7. For **Action Type**, choose **Use path override**.
8. (Optional) In **Path override** type **/**.
9. Copy the previously created IAM role's ARN (from the IAM console) and paste it into **Execution role**.

Choose the integration point for your new method. ⓘ

Integration type Lambda Function
 HTTP Proxy
 Mock Integration
 AWS Service Proxy 1

AWS Region us-east-1 2

AWS Service S3 3

AWS Subdomain

HTTP method GET 4

Action Type Use action name
 Use path override 5

Path override (optional) / 6

Execution role arn:aws:iam::777777777777:role/apigAwsProxyRole 7 ⓘ

8 Save

10. Choose **Save** to finish setting up this method.

This setup integrates the front-end GET `https://your-api-host/stage/` request with the back-end GET `https://your-s3-host/`.

Note

After the initial setup, you can modify these settings in the **Integration Request** page of the method.

To control who can call this method of our API, we turn on the method authorization flag and set it to `AWS_IAM`.

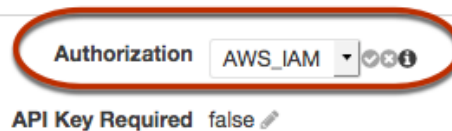
To enable IAM to control access to the GET / method

1. From the **Method Execution**, choose **Method Request**.
2. Choose the pencil icon next to **Authorization**
3. Choose `AWS_IAM` from the drop-down list.
4. Choose the check-mark icon to save the setting.

[← Method Execution](#) / - GET - Method Request

Provide information about this method's authorization settings and the parameters it can receive.

Authorization Settings



- ▶ [URL Query String Parameters](#)
 - ▶ [HTTP Request Headers](#)
 - ▶ [Request Models](#) [Create a Model](#)
-

For our API to return successful responses and exceptions properly to the caller, let us declare the 200, 400 and 500 responses in **Method Response**. We use the default mapping for 200 responses so that back-end responses of the status code not declared here will be returned to the caller as 200 ones.

To declare response types for the GET / method

1. From the **Method Execution** pane, choose the **Method Response** box. The API Gateway declares the 200 response by default.
2. Choose **Add response**, enter `400` in the input text box, and choose the check-mark to finish the declaration.
3. Repeat the above step to declare the 500 response type. The final setting is shown as follows:

[← Method Execution](#) / - GET - Method Response

Provide information about this method's response types, their headers and content types.

HTTP Status	
▶ 200	✕
▶ 400	✕
▶ 500	✕

[+ Add Response](#)

Because the successful integration response from Amazon S3 returns the bucket list as an XML payload and the default method response from API Gateway returns a JSON payload, we must map the back-end Content-Type header parameter value to the front-end counterpart. Otherwise, the client will receive `application/json` for the content type when the response body is actually an XML string. The following procedure shows how to set this up. In addition, we also want to display to the client other header parameters, such as Date and Content-Length.







To set up response header mappings for the GET / method



1. In the API Gateway console, choose **Method Response**. Add the **Content-Type** header for the 200 response type.



[← Method Execution](#) / - GET - Method Response

Provide information about this method's response types, their headers and content types.

HTTP Status	
▼	200

Response Headers for 200	
Name	
Timestamp	 
Content-Length	 
Content-Type	 
+ Add Header	

Response Models for 200		Create a model
Content type	Models	
application/json	Empty	 
+ Add Response Model		

▶	400	
▶	500	
+ Add Response		

2. In **Integration Response**, for **Content-Type**, type `integration.response.header.Content-Type` for the method response.

← Method Execution / - GET - Integration Response

First, declare response types using [Method Response](#). Then, map the possible responses from the backend to this method's response types.

HTTP status regex	Method response status	Output model	Default mapping
-	200		Yes

Map the output from your HTTP endpoint to the headers and output model of the 200 method response.

HTTP status regex ⓘ

Method response status 200

Cancel Save

▼ Header Mappings

Response header	Mapping value ⓘ
Timestamp	integration.response.header.Date
Content-Length	integration.response.header.Content-Length
Content-Type	integration.response.header.Content-Type

▶ Body Mapping Templates

With the above header mappings, API Gateway will translate the `Date` header from the back end to the `Timestamp` header for the client.

3. Still in **Integration Response**, choose **Add integration response**, type an appropriate regular expression in the **HTTP status regex** text box for a remaining method response status. Repeat until all the method response status are covered.

[← Method Execution](#) / - GET - Integration Response

First, declare response types using [Method Response](#). Then, map the possible responses from the backend to this method's response types.

	HTTP status regex	Method response status	Output model	Default mapping	
▶	-	200		Yes	✕
▶	4\d{2}	400		No	✕
▼	5\d{2}	500		No	✕

Map the output from your HTTP endpoint to the headers and output model of the 500 method response.

ⓘ

Method response status500

Cancel

▼ Header Mappings

Response header	Mapping value ⓘ	
No method response headers.		

▶ Body Mapping Templates

As a good practice, let us test our API we have configured so far.

Test the GET method on the API root resource

1. Go back to **Method Execution**, choose **Test** from the **Client** box.
2. Choose **Test** in the **GET / - Method Test** pane. An example result is shown as follows.

Notice that we used the `{bucket}` path parameter in the Amazon S3 endpoint URLs to specify the bucket. We will need to map the `{folder}` path parameter of the method requests to the `{bucket}` path parameter of the integration requests.

3. To map `{folder}` to `{bucket}`:
 - a. Choose **Method Execution** and then **Integration Request**.
 - b. Expand **URL Path Parameters** and choose **Add path**
 - c. Type `bucket` in the **Name** column and `method.request.path.folder` in the **Mapped from** column. Choose the check-mark icon to save the mapping.

▼ URL Path Parameters

Name	Mapped from ⓘ	Caching	
bucket	method.request.path.folder		<input checked="" type="checkbox"/>

4. In **Method Request**, add the `Content-Type` to the **HTTP Request Headers** section.

▼ HTTP Request Headers

Name	Caching	
Content-Type	<input type="checkbox"/>	✕

+ Add header

This is mostly needed for testing, when using the API Gateway console, when you must specify `application/xml` for an XML payload.

5. In **Integration Request**, set up the following header mappings, following the instructions described in [Expose an API Method to List the Caller's Amazon S3 Buckets](#) (p. 362).

▼ HTTP Headers

Name	Mapped from ⓘ	Caching	
x-amz-acl	'authenticated-read'	<input type="checkbox"/>	✎ ✕
Expect	'100-continue'	<input type="checkbox"/>	✎ ✕
Content-Type	method.request.header.Content-Type	<input type="checkbox"/>	✎ ✕

+ Add header

The `x-amz-acl` header is for specifying access control on the folder (or the corresponding Amazon S3 bucket). For more information, see [Amazon S3 PUT Bucket Request](#). The `Expect: '100-continue'` header ensures that a request payload is submitted only when the request parameters are validated.

6. To test the `PUT` method, choose **Test** in the **Client** box from **Method Execution**, and enter the following as input to the testing:
 - a. In **folder**, type a bucket name,
 - b. For the **Content-Type** header, type `application/xml`.
 - c. In **Request Body**, provide the bucket region as the location constraint, declared in an XML fragment as the request payload. For example,

```
<CreateBucketConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <LocationConstraint>us-west-2</LocationConstraint>
</CreateBucketConfiguration>
```


← Method Execution **/{folder}/{item} - PUT - Integration Request**

Provide information about the target backend that this method will call and whether the incoming request data should be modified.

- Integration type**
- Lambda Function ⓘ
 - HTTP ⓘ
 - Mock ⓘ
 - AWS Service ⓘ

AWS Region us-west-2 ✎

AWS Service S3 ✎

AWS Subdomain ✎

HTTP method PUT ✎

Path override {bucket}/{object} ✎

Execution role arn:aws:iam::[redacted]:role/apigAwsProxyRole ✎

Credentials cache Do not add caller credentials to cache key ✎

▼ URL Path Parameters

Name	Mapped from ⓘ	Caching	
object	method.request.path.item	<input type="checkbox"/>	✎⊗
bucket	method.request.path.folder	<input type="checkbox"/>	✎⊗

➕ Add path

The same is true for the GET and DELETE methods.

As an illustration, the following screen shot shows the output when testing the GET method on a `{folder}/{item}` resource using the API Gateway console. The request correctly returns the plain text of ("Welcome to README.txt") as the content of the specified file (README.txt) in the given Amazon S3 bucket (apig-demo).

← Method Execution `/{folder}/{item}` - GET - Method Test

Make a test call to your method with the provided input

Path
folder:
item:

Request: /apig-demo/README.txt
Status: 200
Latency: 486 ms

Response Body

Response Headers

Query Strings
No query string parameters exist for this method. You can add them via Method Request.

Headers
No header parameters exist for this method. You can add them via Method Request.

Stage Variables
No stage variables exist for this method.

Client Certificate

Request Body
Request Body is not supported for GET methods.

Logs

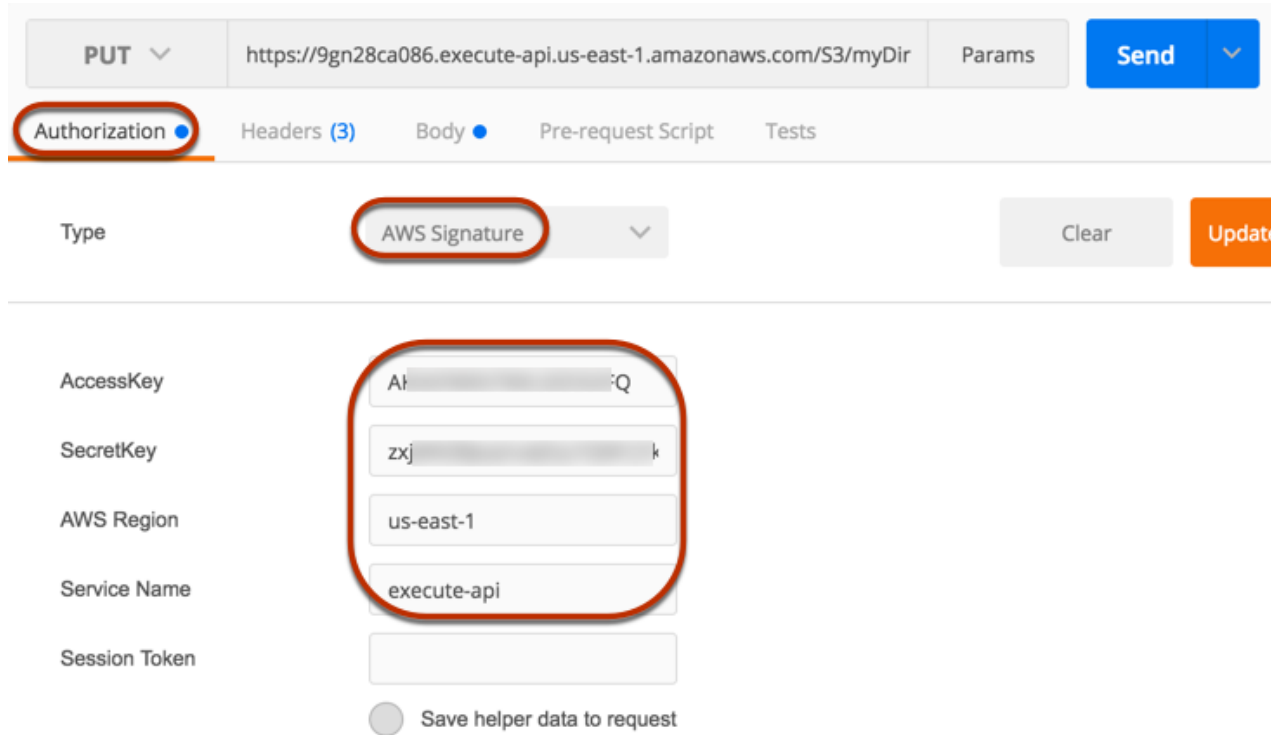
```
Execution log for request test-request
Fri Feb 19 03:35:20 UTC 2016 : Starting execution for request: test-invoke-request
Fri Feb 19 03:35:20 UTC 2016 : API Key: test-invoke-api-key
Fri Feb 19 03:35:20 UTC 2016 : Method request path: {item=README.txt, folder=apig-demo}
Fri Feb 19 03:35:20 UTC 2016 : Method request query string: {}
Fri Feb 19 03:35:20 UTC 2016 : Method request headers: {}
Fri Feb 19 03:35:20 UTC 2016 : Method request body before transformations: null
Fri Feb 19 03:35:20 UTC 2016 : Endpoint request URI: http://s3-us-west-2.amazonaws.com/apig-demo/README.txt
Fri Feb 19 03:35:20 UTC 2016 : Endpoint request headers: {Authorization=*****a99006, X-Amz
```

Call the API Using a REST API Client

To provide an end-to-end tutorial, we now show how to call the API using [Postman](#), which supports the AWS IAM authorization.

To call our Amazon S3 proxy API using Postman

1. Deploy or redeploy the API. Make a note of the base URL of the API that is displayed next to **Invoke URL** at the top of the **Stage Editor**.
2. Launch Postman.
3. Choose **Authorization** and then choose **AWS Signature**. Type your IAM user's Access Key ID and Secret Access Key into the **AccessKey** and **SecretKey** input fields, respectively. Type the AWS region to which your API is deployed in the **AWS Region** text box. Type `execute-api` in the **Service Name** input field.



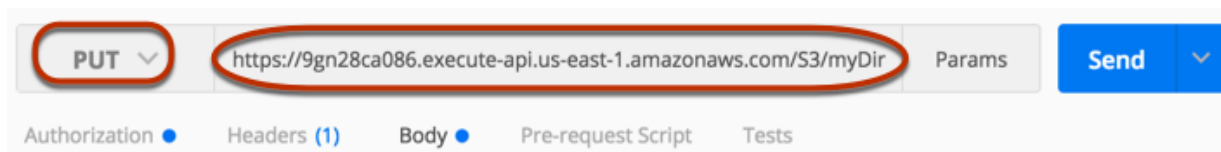
You can create a pair of the keys from the **Security Credentials** tab from your IAM user account in the IAM Management Console.

- To add a bucket named `apig-demo-5` to your Amazon S3 account in the `us-west-2` region:

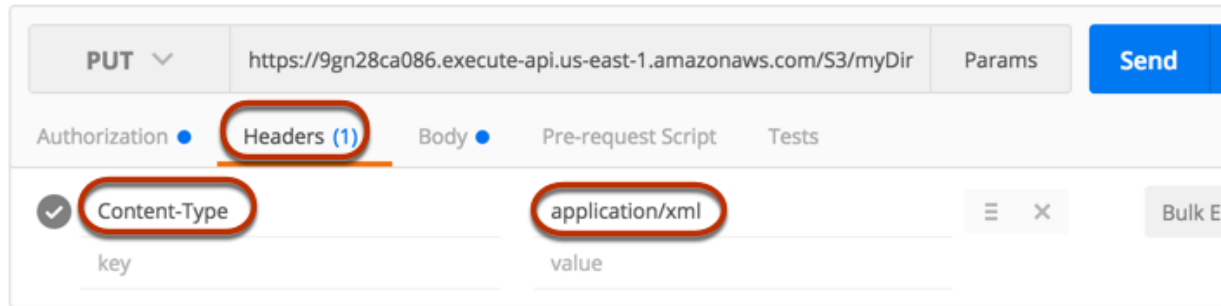
Note

Be sure that the bucket name must be globally unique.

- Choose **PUT** from the drop-down method list and type the method URL (`https://api-id.execute-api.aws-region.amazonaws.com/stage/folder-name`

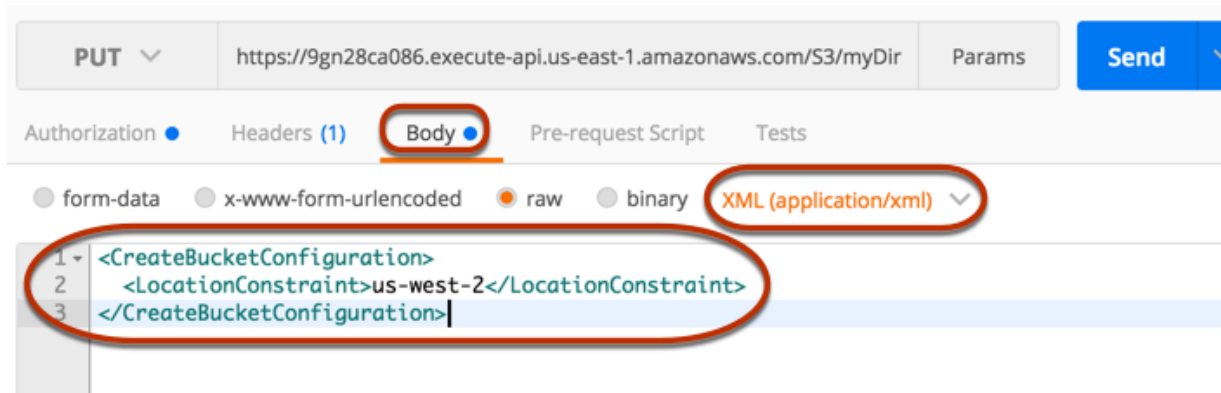


- Set the `Content-Type` header value as `application/xml`. You may need to delete any existing headers before setting the content type.



- c. Choose **Body** menu item and type the following XML fragment as the request body:

```
<CreateBucketConfiguration>
  <LocationConstraint>us-west-2</LocationConstraint>
</CreateBucketConfiguration>
```



- d. Choose **Send** to submit the request. If successful, you should receive a 200 OK response with an empty payload.
5. To add a text file to a bucket, follow the instructions above. If you specify a bucket name of `apig-demo-5` for `{folder}` and a file name of `Readme.txt` for `{item}` in the URL and provide a text string of `Hello, World!` as the request payload, the request becomes

```
PUT /S3/apig-demo-5/Readme.txt HTTP/1.1
Host: 9gn28ca086.execute-api.us-east-1.amazonaws.com
Content-Type: application/xml
X-Amz-Date: 20161015T062647Z
Authorization: AWS4-HMAC-SHA256 Credential=access-
key-id/20161015/us-east-1/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!
```

If everything goes well, you should receive a 200 OK response with an empty payload.

6. To get the content of the `Readme.txt` file we just added to the `apig-demo-5` bucket, do a GET request like the following one:

```
GET /S3/apig-demo-5/Readme.txt HTTP/1.1
Host: 9gn28ca086.execute-api.us-east-1.amazonaws.com
Content-Type: application/xml
X-Amz-Date: 20161015T063759Z
Authorization: AWS4-HMAC-SHA256 Credential=access-
key-id/20161015/us-east-1/execute-api/aws4_request,
SignedHeaders=content-type;host;x-amz-date,
Signature=ba09b72b585acf0e578e6ad02555c00e24b420b59025bc7bb8d3f7aed1471339
Cache-Control: no-cache
Postman-Token: d60fcb59-d335-52f7-0025-5bd96928098a
```

If successful, you should receive a 200 OK response with the Hello, World! text string as the payload.

7. To list items in the apig-demo-5 bucket, submit the following request:

```
GET /S3/apig-demo-5 HTTP/1.1
Host: 9gn28ca086.execute-api.us-east-1.amazonaws.com
Content-Type: application/xml
X-Amz-Date: 20161015T064324Z
Authorization: AWS4-HMAC-SHA256 Credential=access-
key-id/20161015/us-east-1/execute-api/aws4_request,
SignedHeaders=content-type;host;x-amz-date,
Signature=4ac9bd4574a14e01568134fd16814534d9951649d3a22b3b0db9f1f5cd4dd0ac
Cache-Control: no-cache
Postman-Token: 9c43020a-966f-61e1-81af-4c49ad8d1392
```

If successful, you should receive a 200 OK response with an XML payload showing a single item in the specified bucket, unless you added more files to the bucket before submitting this request.

```
<?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...603add12ee</ID>
      <DisplayName>user-name</DisplayName>
    </Owner>
    <StorageClass>STANDARD</StorageClass>
  </Contents>
</ListBucketResult>
```

Swagger Definitions of the Sample API as an Amazon S3 Proxy

The following Swagger definitions describe the sample API , referenced in this tutorial, as an Amazon S3 proxy.

```
{
  "swagger": "2.0",
  "info": {
    "version": "2016-10-13T23:04:43Z",
    "title": "MyS3"
  },
  "host": "9gn28ca086.execute-api.us-east-1.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::<replaceable>123456789012</replaceable>: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::<replaceable>123456789012</
replaceable>: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::<replaceable>123456789012/</replaceable>:role/apigAwsProxyRole",
    "responses": {
      "4\\d{2}": {
        "statusCode": "400"
      },
      "default": {
        "statusCode": "200",
        "responseParameters": {
          "integration.response.header.Content-Type":
"integration.response.header.Content-Type",
          "integration.response.header.Content-Length":
"integration.response.header.Content-Length"
        }
      },
      "5\\d{2}": {
        "statusCode": "500"
      }
    },
    "requestParameters": {
      "integration.request.header.x-amz-acl": "'authenticated-read'",
      "integration.request.header.Expect": "'100-continue'",
      "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::<replaceable>123456789012</replaceable>:role/apigAwsProxyRole",
    "responses": {
      "4\\d{2}": {
        "statusCode": "400"
      },
      "default": {
        "statusCode": "200",
        "responseParameters": {
          "integration.response.header.Content-Type":
"integration.response.header.Content-Type",
          "integration.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::<replaceable>123456789012/<replaceable>: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::<replaceable>123456789012</replaceable>: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::<replaceable>123456789012</replaceable>: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.header.x-amz-acl": "'authenticated-read'",
      "integration.request.header.Expect": "'100-continue'",

```

```
        "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::<replaceable>123456789012</replaceable>: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"
    }
}
}
```

Create an API Gateway API as an Amazon Kinesis Proxy

This section describes how to create and configure an API Gateway API as an AWS proxy to access Amazon Kinesis.

Note

To integrate your API Gateway API with Amazon Kinesis, you must choose a region where both the API Gateway and Amazon Kinesis services are available. For region availability, see [Regions and Endpoints](#).

For the purpose of illustration, we will create an example API to enable a client to do the following:

1. List the user's available streams in Amazon Kinesis
2. Create, describe, or delete a specified stream
3. Read data records from or write data records into the specified stream

To accomplish the preceding tasks, the API exposes methods on various resources to invoke the following, respectively:

1. The `ListStreams` action in Amazon Kinesis
2. The `CreateStream`, `DescribeStream`, or `DeleteStream` action
3. The `GetRecords` or `PutRecords` (including `PutRecord`) action in Amazon Kinesis

Specifically, we will build the API as follows:

- Expose an HTTP GET method on the API's `/streams` resource and integrate the method with the [ListStreams](#) action in Amazon Kinesis to list the streams in the caller's account.
- Expose an HTTP POST method on the API's `/streams/{stream-name}` resource and integrate the method with the [CreateStream](#) action in Amazon Kinesis to create a named stream in the caller's account.
- Expose an HTTP GET method on the API's `/streams/{stream-name}` resource and integrate the method with the [DescribeStream](#) action in Amazon Kinesis to describe a named stream in the caller's account.
- Expose an HTTP DELETE method on the API's `/streams/{stream-name}` resource and integrate the method with the [DeleteStream](#) action in Amazon Kinesis to delete a stream in the caller's account.
- Expose an HTTP PUT method on the API's `/streams/{stream-name}/record` resource and integrate the method with the [PutRecord](#) action in Amazon Kinesis. This enables the client to add a single data record to the named stream.
- Expose an HTTP PUT method on the API's `/streams/{stream-name}/records` resource and integrate the method with the [PutRecords](#) action in Amazon Kinesis. This enables the client to add a list of data records to the named stream.
- Expose an HTTP GET method on the API's `/streams/{stream-name}/records` resource and integrate the method with the [GetRecords](#) action in Amazon Kinesis. This enables the client to list data records in the named stream, with a specified shard iterator. A shard iterator specifies the shard position from which to start reading data records sequentially.
- Expose an HTTP GET method on the API's `/streams/{stream-name}/sharditerator` resource and integrate the method with the [GetShardIterator](#) action in Amazon Kinesis. This helper method must be supplied to the `ListStreams` action in Amazon Kinesis.

You can apply the instructions presented here to other Amazon Kinesis actions. For the complete list of the Amazon Kinesis actions, see [Amazon Kinesis API Reference](#).

Instead of using the API Gateway console to create the sample API, you can import the sample API into API Gateway, using either the API Gateway [Import API](#) or the [API Gateway Swagger Importer](#). For information on how to use the Import API, see [Import an API \(p. 142\)](#). For information on how to use the API Gateway Swagger Importer, see [Getting Started with the API Gateway Swagger Importer](#).

If you do not have an AWS account, use the following procedure to create one.

To sign up for AWS

1. Open <http://aws.amazon.com/> and choose **Create an AWS Account**.
2. Follow the online instructions.

Create an IAM Role and Policy for the API to Access Amazon Kinesis

To allow the API to invoke Amazon Kinesis actions, you must have appropriate IAM policies attached to an IAM role. This section explains how to verify and to create, if necessary, the required IAM role and policies.

To enable read-only access to Amazon Kinesis, you can use the `AmazonKinesisReadOnlyAccess` policy that allows the `Get*`, `List*`, and `Describe*` actions in Amazon Kinesis to be invoked.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "kinesis:Get*",
        "kinesis:List*",
        "kinesis:Describe*"
      ],
      "Resource": "*"
    }
  ]
}
```

This policy is available from the IAM console and its ARN is `arn:aws:iam::aws:policy/AmazonKinesisReadOnlyAccess`.

To enable read-write actions in Amazon Kinesis, you can use the `AmazonKinesisFullAccess` policy.

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

This policy is also available from the IAM console. Its ARN is `arn:aws:iam::aws:policy/AmazonKinesisFullAccess`.

After you decide which IAM policy to use, attach it to a new or existing IAM role. Make sure that the API Gateway control service (`apigateway.amazonaws.com`) is a trusted entity of the role and is allowed to assume the execution role (`sts:AssumeRole`).

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

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

If you create the execution role in the IAM console and choose the **Amazon API Gateway** role type, this trust policy is automatically attached.

Note the ARN of the execution role. You will need it when creating an API method and setting up its integration request.

Start to Create an API as an Amazon Kinesis Proxy

Use the following steps to create the API in the API Gateway console.

To create an API as an AWS service proxy for Amazon Kinesis

1. In the API Gateway console, choose **Create API**.
2. In **API name**, type `kinesisProxy`. Leave the default values in the other fields.
3. For **Clone from API**, choose **Do not clone from existing API**.
4. Type a description in **Description**.
5. Choose **Create API**.

After the API is created, the API Gateway console displays the **Resources** page, which contains only the API's root (/) resource.

List Streams in Amazon Kinesis

To list streams in Amazon Kinesis, add a `/streams` resource to the API's root, expose a GET method on the resource, and integrate the method to the `ListStreams` action of Amazon Kinesis.

The following procedure describes how to list Amazon Kinesis streams by using the API Gateway console.

To list Amazon Kinesis streams by using the API Gateway console

1. Select the API root resource. In **Actions**, choose **Create Resource**.
In **Resource Name**, type `Streams`, leave **Resource Path** as the default, and choose **Create Resource**.
2. Select the `/Streams` resource. From **Actions**, choose **Create Method**, choose **GET** from the list, and then choose the checkmark icon to finish creating the method.

Note

You can choose any of the available HTTP verbs for a method request. We use `GET` here, because listing streams is a `READ` operation.

3. In the method's **Setup** pane, choose **Show Advanced** and then choose **AWS Service Proxy**.
 - a. For **AWS Region**, choose a region (e.g., `us-east-1`).

- b. For **AWS Service**, choose **Kinesis**.
- c. For **HTTP method**, choose **POST**.

Note

For the integration request with Amazon Kinesis, you must choose the `POST` HTTP verb to invoke the action, although you can use any of the available HTTP verbs for the API's method request.

- d. For **Action Type**, choose **Use action name**.
- e. For **Action**, type `ListStreams`.
- f. For **Execution role**, type the ARN for your execution role.
- g. Choose **Save** to finish the initial setup of the method.

[← Method Execution](#) /streams - GET - Integration Request

Provide information about the target backend that this method will call and whether the incoming request data should be modified.

Integration type Lambda Function
 HTTP Proxy
 Mock Integration
 AWS Service Proxy

AWS Region us-east-1

AWS Service Kinesis

AWS Subdomain

HTTP method POST

Action ListStreams

Execution role arn:aws:iam::[redacted]:role/apigAwsProxyRole

Credentials cache Do not add caller credentials to cache key

The initial setup of the integration request will suffice if there is no need to map data between the method and integration requests and/or between the method and integration responses. Examples discussed in this topic require data mapping, which is covered in the second half of the **Integration Request** pane.

- 4. In the **Integration Request** pane, expand the **HTTP Headers** section:
 - a. Choose **Add header**.
 - b. In the **Name** column, type `Content-Type`.
 - c. In the **Mapped from** column, type `'application/x-amz-json-1.1'`.
 - d. Choose the checkmark icon to save the setting.
- 5. Expand the **Body Mapping Templates** section:
 - a. Choose **Add mapping template**.
 - b. For **Content-Type**, type `application/json`.
 - c. Choose the checkmark icon to save the setting.
 - d. Choose the pencil icon to the right of **Mapping template**.
 - e. Choose **Mapping template** from the drop-down list to open the **Template** editor.
 - f. Type `{}` in the template editor.
 - g. Choose the checkmark icon to save the mapping template.

The `ListStreams` request takes a payload of the following JSON format:

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

However, the properties are optional. To use the default values, we opted for an empty JSON payload here.

▶ URL Path Parameters

▶ URL Query String Parameters

▼ HTTP Headers

Name	Mapped from ⓘ	Caching	
Content-Type	'application/x-amz-json-1.1'	<input type="checkbox"/>	

[+ Add header](#)

▼ Body Mapping Templates

Content-Type		application/json	Mapping template
application/json		Template	<pre>1 { 2 }</pre>

[+ Add mapping template](#)

6. Test the GET method on the Streams resource to invoke the `ListStreams` action in Amazon Kinesis:

From the API Gateway console, select the `/streams/GET` entry from the **Resources** pane, choose the **Test** invocation option, and then choose **Test**.

If you have already created two streams named "myStream" and "yourStream" in Amazon Kinesis, the successful test will return a 200 OK response containing the following payload:

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

Create, Describe, and Delete a Stream in Amazon Kinesis

Creating, describing, and deleting a stream in Amazon Kinesis involves making the following Amazon Kinesis REST API requests, respectively:

```
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

{
  "ExclusiveStartShardId": "string",
  "Limit": number,
  "StreamName": "string"
}
```

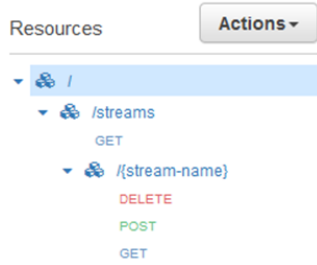
```
POST /?Action>DeleteStream HTTP/1.1
Host: kinesis.region.domain
...
Content-Type: application/x-amz-json-1.1
Content-Length: PayloadSizeBytes

{
  "StreamName": "string"
}
```

We can build our API to accept the required input as a JSON payload of the method request and pass the payload through to the integration request. However, to provide more examples of data mapping between method and integration requests, and method and integration responses, we will create our API slightly differently.

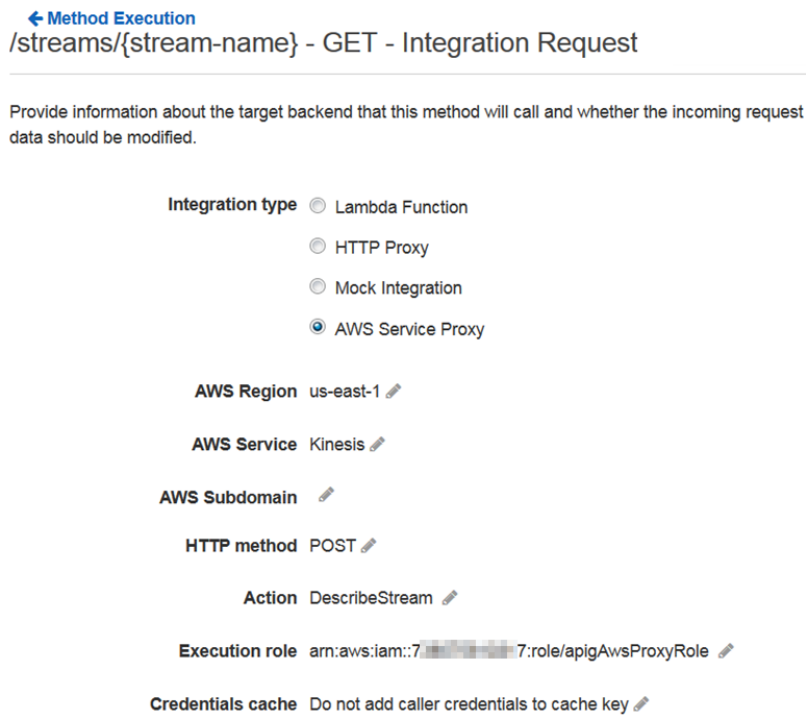
We will expose the GET, POST, and Delete HTTP methods on a to-be-named `Stream` resource. We will use the `{stream-name}` path variable to hold the to-be-named stream resource and integrate these API methods with the Amazon Kinesis' `DescribeStream`, `CreateStream`, and `DeleteStream` actions, respectively. We require that the client pass other input data as headers, query parameters, or the payload of a method request, and we provide mapping templates to transform the data to the required integration request payload.

After the methods are created on a to-be-named stream resource, the structure of the API looks like the following:



To configure and test the GET method on a stream resource

1. Set up the GET method to describe a named stream in Amazon Kinesis, as shown in the following.



2. Map data from the GET method request to the integration request, as shown in the following:

HTTP method POST

Action DescribeStream

Execution role arn:aws:iam::7[redacted]:role/apigAwsProxyRole

Credentials cache Do not add caller credentials to cache key

▶ URL Path Parameters

▶ URL Query String Parameters

▶ HTTP Headers

▼ Body Mapping Templates

Content-Type	Mapping template
application/json	Template

Add mapping template

```
1 {
2   "StreamName": "$input.params('stream-name')"
3 }
```

3. Test the GET method to invoke the `DescribeStream` action in Amazon Kinesis:

From the API Gateway console, select `/streams/{stream-name}/GET` in the **Resources** pane, choose **Test** to start testing, type the name of an existing Amazon Kinesis stream in the **Path** field for `stream-name`, and choose **Test**. If the test is successful, a 200 OK response is returned with a payload similar to the following:

```
{
  "StreamDescription": {
    "HasMoreShards": false,
    "RetentionPeriodHours": 24,
    "Shards": [
      {
        "HashKeyRange": {
          "EndingHashKey": "68056473384187692692674921486353642290",
          "StartingHashKey": "0"
        },
        "SequenceNumberRange": {
          "StartingSequenceNumber":
"49559266461454070523309915164834022007924120923395850242"
        },
        "ShardId": "shardId-000000000000"
      },
      ...
      {
        "HashKeyRange": {
          "EndingHashKey": "340282366920938463463374607431768211455",
          "StartingHashKey": "272225893536750770699685945414569164"
        },
        "SequenceNumberRange": {
          "StartingSequenceNumber":
"49559266461543273504104037657400164881014714369419771970"
        },
      },
    ]
  }
}
```



```
        "ShardId": "shardId-000000000004"  
      }  
    ],  
    "StreamARN": "arn:aws:kinesis:us-east-1:12345678901:stream/myStream",  
    "StreamName": "myStream",  
    "StreamStatus": "ACTIVE"  
  }  
}
```

After you deploy the API, you can make a REST request against this API method:

```
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
```

To configure and test the POST method on a stream resource

1. Set up the POST method on a stream resource to create the stream in Amazon Kinesis, as shown in the following:

[← Method Execution](#)
/streams/{stream-name} - POST - Integration Request

Provide information about the target backend that this method will call and whether the incoming request data should be modified.

Integration type Lambda Function
 HTTP Proxy
 Mock Integration
 AWS Service Proxy

AWS Region us-east-1

AWS Service Kinesis

AWS Subdomain

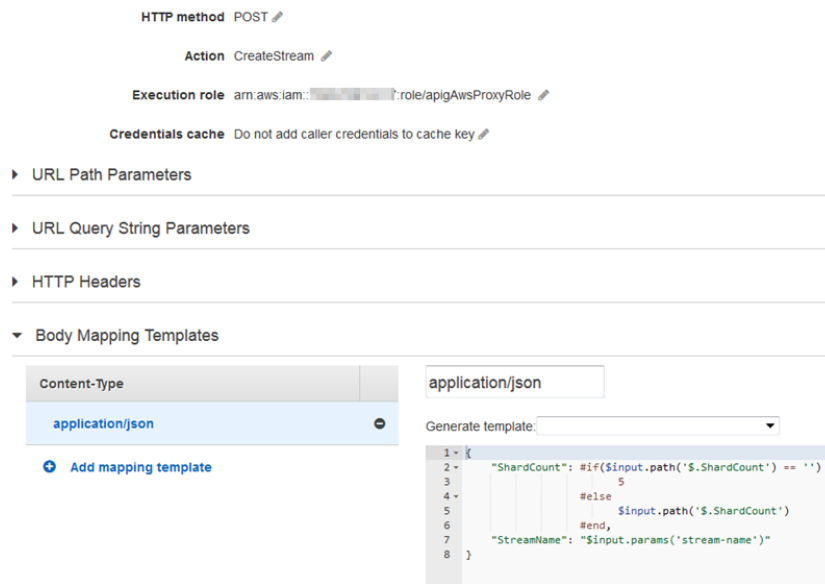
HTTP method POST

Action CreateStream

Execution role arn:aws:iam::7:role/apigAwsProxyRole

Credentials cache Do not add caller credentials to cache key

2. Map data from the POST method request to the integration request, as shown in the following:



In this example, we use the following mapping template to set `ShardCount` to a fixed value of 5 if the client does not specify a value in the method request payload. Otherwise, we pass the client-supplied value to the back end.

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

The preceding `if ($input.path('$.ShardCount') == '') ... Boolean expression` tests if the method request's JSON payload does not have the `ShardCount` property declared or if the property value is empty.

3. Test the POST method to create a named stream in Amazon Kinesis:

From the API Gateway console, select `/streams/{stream-name}/POST` in the **Resources** pane, choose **Test** to start testing, type the name of an existing Amazon Kinesis stream in **Path** for `stream-name`, and choose **Test**. If the test is successful, a 200 OK response is returned with no data.

After you deploy the API, you can also make a REST API request against the POST method on a Stream resource to invoke the `CreateStream` action in Amazon Kinesis:

```
POST https://your-api-id.execute-api.region.amazonaws.com/stage/streams/yourStream HTTP/1.1
Host: your-api-id.execute-api.region.amazonaws.com
Content-Type: application/json
Authorization: ...
X-Amz-Date: 20160323T194451Z

{
  "ShardCount": 5
```

```
}
```


Configure and test the DELETE method on a stream resource


1. Set up the DELETE method to invoke the `DeleteStream` action in Amazon Kinesis, as shown in the following.


[← Method Execution](#)
/streams/{stream-name} - DELETE - Integration Request


Provide information about the target backend that this method will call and whether the incoming request data should be modified.


Integration type Lambda Function
 HTTP Proxy
 Mock Integration
 AWS Service Proxy


AWS Region us-east-1 


AWS Service Kinesis 

AWS Subdomain 

HTTP method POST 

Action DeleteStream 

Execution role arn:aws:iam::7:role/apigAwsProxyRole 

Credentials cache Do not add caller credentials to cache key 

2. Map data from the DELETE method request to the integration request, as shown in the following:

Amazon API Gateway Developer Guide

Get Records from and Add Records to a Stream in Amazon Kinesis

HTTP method POST

Action DeleteStream

Execution role arn:aws:iam::738575810317:role/apigAwsProxyRole

Credentials cache Do not add caller credentials to cache key

▶ URL Path Parameters

▶ URL Query String Parameters

▼ HTTP Headers

Name	Mapped from	Caching	
Content-Type	'application/x-amz-json-1.1'	<input type="checkbox"/>	

Add header

▼ Body Mapping Templates

Content-Type	Mapping template
application/json	Template

```
1- {
2   "StreamName": "$input.params('stream-name')
3 }
```

Add mapping template

3. Test the DELETE method to delete a named stream in Amazon Kinesis:

From the API Gateway console, select the **/streams/{stream-name}/DELETE** method node in the **Resources** pane, choose **Test** to start testing, type the name of an existing Amazon Kinesis stream in **Path** for `stream-name`, and choose **Test**. If the test is successful, a 200 OK response is returned with no data.

After you deploy the API, you can also make the following REST API request against the DELETE method on the Stream resource to call the `DeleteStream` action in Amazon Kinesis:

```
DELETE https://your-api-id.execute-api.region.amazonaws.com/stage/streams/yourStream HTTP/1.1
Host: your-api-id.execute-api.region.amazonaws.com
Content-Type: application/json
Authorization: ...
X-Amz-Date: 20160323T194451Z

{}
```

Get Records from and Add Records to a Stream in Amazon Kinesis

After you create a stream in Amazon Kinesis, you can add data records to the stream and read the data from the stream. Adding data records involves calling the [PutRecords](#) or [PutRecord](#) action in

Amazon Kinesis. The former adds multiple records whereas the latter adds a single record to the stream.

```
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"
}
```

or

```
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"
}
```

Here, `StreamName` identifies the target stream to add records. `StreamName`, `Data`, and `PartitionKey` are required input data. In our example, we use the default values for all of the optional input data and will not explicitly specify values for them in the input to the method request.

Reading data in Amazon Kinesis amounts to calling the [GetRecords](#) action:

```
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
}
```

Here, the source stream from which we are getting records is specified in the required `ShardIterator` value, as is shown in the following Amazon Kinesis action to obtain a shard iterator:

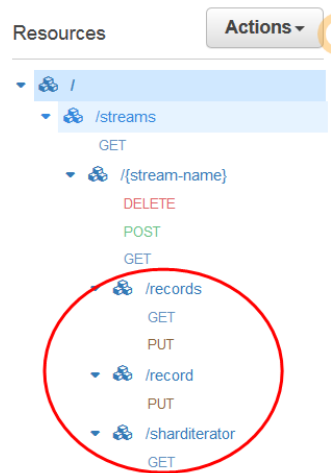
```
POST /?Action=GetShardIterator HTTP/1.1
Host: kinesis.region.domain
Authorization: AWS4-HMAC-SHA256 Credential=..., ...
...
Content-Type: application/x-amz-json-1.1
Content-Length: PayloadSizeBytes

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

For the `GetRecords` and `PutRecords` actions, we expose the `GET` and `PUT` methods, respectively, on a `/records` resource that is appended to a named stream resource (`/{stream-name}`). Similarly, we expose the `PutRecord` action as a `PUT` method on a `/record` resource.

Because the `GetRecords` action takes as input a `ShardIterator` value, which is obtained by calling the `GetShardIterator` helper action, we expose a `GET` helper method on a `ShardIterator` resource (`/sharditerator`).

The following figure shows the API structure of resources after the methods are created:



The following four procedures describe how to set up each of the methods, how to map data from the method requests to the integration requests, and how to test the methods.

To configure and test the PUT method on the record resource in the API to invoke the PutRecord action in Amazon Kinesis:

1. Set up the `PUT` method, as shown in the following:

Amazon API Gateway Developer Guide

Get Records from and Add Records to a Stream in Amazon Kinesis

← Method Execution /streams/{stream-name}/record - PUT - Integration Request

Provide information about the target backend that this method will call and whether the incoming request data should be modified.

Integration type Lambda Function
 HTTP Proxy
 Mock Integration
 AWS Service Proxy

AWS Region

AWS Service

AWS Subdomain

HTTP method

Action

Execution role

Credentials cache

2. Configure data mapping for the PUT-on-Record method, as shown in the following:

Action

Execution role

Credentials cache

▶ URL Path Parameters

▶ URL Query String Parameters

▼ HTTP Headers

Name	Mapped from	Caching	
Content-Type	<input type="text" value="'application/x-amz-json-1.1'"/>	<input type="checkbox"/>	<input type="text"/>

➕ Add header

▼ Body Mapping Templates

Content-Type	Mapping template
<input type="text" value="application/json"/>	<pre>1 { 2 "StreamName": "\$input.params('stream-name')", 3 "Data": "\$util.base64Encode(\$input.path('\$Data'))", 4 "PartitionKey": "\$input.path('\$PartitionKey')", 5 }</pre>

➕ Add mapping template

The preceding mapping template assumes that the method request payload is of the following format:

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

This data can be modeled by the following JSON schema:

```
{
```

Amazon API Gateway Developer Guide

Get Records from and Add Records to a Stream in Amazon Kinesis

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

You can create a model to include this schema and use the model to facilitate generating the mapping template. However, you can generate a mapping template without using any model.

3.

To test the PUT method, set the `stream-name` path variable to an existing stream, supply a payload of the preceding format, and then submit the method request. The successful result is a 200 OK response with a payload of the following format:

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

To configure and test the PUT method on the records resource in the API to invoke the PutRecords action in Amazon Kinesis

1. Set up the PUT method, as shown in the following:

[← Method Execution](#) /streams/{stream-name}/records - PUT - Integration Request

Provide information about the target backend that this method will call and whether the incoming request data should be modified.

Integration type Lambda Function
 HTTP Proxy
 Mock Integration
 AWS Service Proxy

AWS Region

AWS Service

AWS Subdomain

HTTP method

Action

Execution role

Credentials cache

2. Configure data mapping for the PUT method, as shown in the following:

Amazon API Gateway Developer Guide Get Records from and Add Records to a Stream in Amazon Kinesis

Action PutRecords

Execution role arn:aws:iam::7:role/apigAwsProxyRole

Credentials cache Do not add caller credentials to cache key

- ▶ URL Path Parameters
- ▶ URL Query String Parameters
- ▶ HTTP Headers
- ▼ Body Mapping Templates

Content-Type	application/json	Mapping template
application/json		Template

Add mapping template

```
1 {
2   "StreamName": "$input.params('stream-name')",
3   "Records": [
4     #foreach($elem in $input.path('$.records'))
5     {
6       "Data": "$util.base64Encode($elem.data)",
7       "PartitionKey": "$elem.partition-key"
8     }#if($foreach.hasNext),#end
9   ]#end
10 }
11 }
```

The preceding mapping template assumes the method request payload can be modeled by following JSON schema:

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

3.

To test the PUT method, set the `stream-name` path variable to an existing stream, supply a payload as previously shown, and submit the method request. The successful result is a 200 OK response with a payload of the following format:

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

Amazon API Gateway Developer Guide Get Records from and Add Records to a Stream in Amazon Kinesis

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

The response payload will be similar to the following output:

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

To configure and test the GET method on the ShardIterator resource in the API to invoke the GetShardIterator action in Amazon Kinesis

The GET-on-ShardIterator method is a helper method to acquire a required shard iterator before calling the GET-on-Records method.

1. Set up the GET-on-ShardIterator method, as shown in the following:

[← Method Execution](#) /streams/{stream-name}/sharditerator - GET - Integration R...

Provide information about the target backend that this method will call and whether the incoming request data should be modified.

Integration type Lambda Function
 HTTP Proxy
 Mock Integration
 AWS Service Proxy

AWS Region

AWS Service

AWS Subdomain

HTTP method

Action

Execution role

Credentials cache

- 2.

Amazon API Gateway Developer Guide

Get Records from and Add Records to a Stream in Amazon Kinesis

The `GetShardIterator` action requires an input of a `ShardId` value. To pass a client-supplied `ShardId` value, we add a `shard-id` query parameter to the method request, as shown in the following:

← Method Execution /streams/{stream-name}/sharditerator - GET - Method Req...

Provide information about this method's authorization settings and the parameters it can receive.

Authorization Settings

Authorization NONE ⓘ

API Key Required false ⓘ

Request Paths

Name	Caching
stream-name	<input type="checkbox"/>

URL Query String Parameters

Name	Caching	
shard-id	<input type="checkbox"/>	+

+ Add query string

HTTP Request Headers

Request Models [Create a Model](#)

In the following mapping template, we add the translation of the `shard-id` query parameter value to the `ShardId` property value of the JSON payload for the `GetShardIterator` action in Amazon Kinesis.

3. Configure data mapping for the GET-on-ShardIterator method:

Action GetShardIterator ⓘ

Execution role arn:aws:iam::7:role/apigAwsProxyRole ⓘ

Credentials cache Do not add caller credentials to cache key ⓘ

URL Path Parameters

URL Query String Parameters

HTTP Headers

Name	Mapped from ⓘ	Caching	
Content-Type	'application/x-amz-json-1.1'	<input type="checkbox"/>	ⓘ

+ Add header

Body Mapping Templates

Content-Type	Mapping template ⓘ
application/json	Mapping template ⓘ

+ Add mapping template

```
1 {
2   "ShardId": "${input.params('shard-id')}",
3   "ShardIteratorType": "TRIM_HORIZON",
4   "StreamName": "${input.params('stream-name')}"
5 }
```

- 4.

Using the **Test** option in the API Gateway console, enter an existing stream name as the `stream-name` **Path** variable value, set the `shard-id` **Query string** to an existing `ShardId` value (e.g., `shard-000000000004`), and choose **Test**.

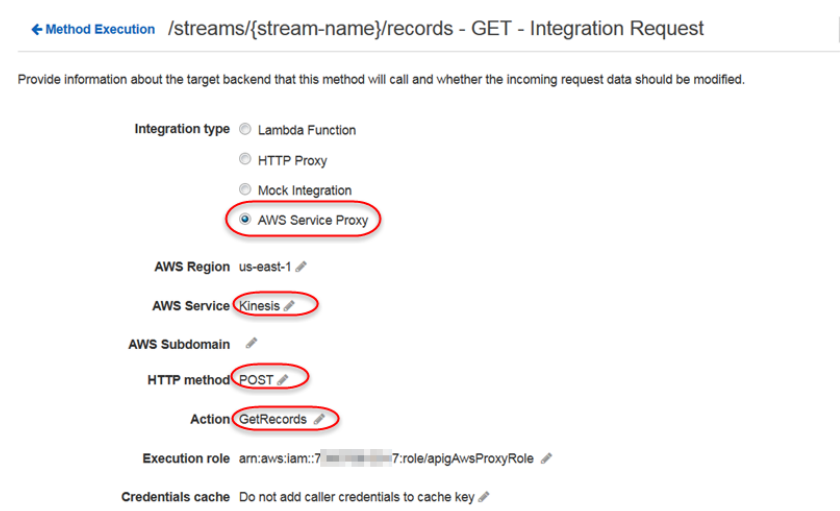
The successful response payload will be similar to the following output:

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

Make note of the `ShardIterator` value. You will need it to get records from a stream.

To configure and test the GET Method on the records resource in the API to invoke the GetRecords action in Amazon Kinesis

1. Set up the GET method, as shown in the following:



2. The `GetRecords` action requires an input of a `ShardIterator` value. To pass a client-supplied `ShardIterator` value, we add a `Shard-Iterator` header parameter to the method request, as shown in the following:

Amazon API Gateway Developer Guide

Get Records from and Add Records to a Stream in Amazon Kinesis

← Method Execution /streams/{stream-name}/records - GET - Method Request

Provide information about this method's authorization settings and the parameters it can receive.

Authorization Settings

Authorization NONE ⓘ

API Key Required false ✎

Request Paths

Name	Caching
stream-name	<input type="checkbox"/>

URL Query String Parameters

HTTP Request Headers

Name	Caching	
Shard-Iterator	<input type="checkbox"/>	+

+ Add header

Request Models [Create a Model](#)

In the following mapping template, we add the mapping from the `Shard-Iterator` header value to the `ShardIterator` property value of the JSON payload for the `GetRecords` action in Amazon Kinesis.

3. Configure data mapping for the GET-on-Records method:

Action `GetRecords` ✎

Execution role `arn:aws:iam::777777777777:role/apigAwsProxyRole` ✎

Credentials cache Do not add caller credentials to cache key ✎

URL Path Parameters

URL Query String Parameters

HTTP Headers

Name	Mapped from ⓘ	Caching	
Content-Type	<code>application/x-amz-json-1.1</code>	<input type="checkbox"/>	✎

+ Add header

Body Mapping Templates

Content-Type	Mapping template ✎
<code>application/json</code>	<pre>Template ↕ 1- { 2 "ShardIterator": "\$input.params('Shard-Iterator')" 3 }</pre>

+ Add mapping template

4. Using the **Test** option in the API Gateway console, type an existing stream name as the `stream-name` **Path** variable value, set the `Shard-Iterator` **Header** to the `ShardIterator` value obtained from the test run of the GET-on-ShardIterator method (above), and choose **Test**.

The successful response payload will be similar to the following output:

```
{  
  "MillisBehindLatest": 0,  
}
```

```
"NextShardIterator": "AAAAAAAAAAAF...",  
"Records": [ ... ]  
}
```

Swagger Definitions of a Sample API as an Amazon Kinesis Proxy

```
{  
  "swagger": "2.0",  
  "info": {  
    "version": "2016-03-31T18:25:32Z",  
    "title": "KinesisProxy"  
  },  
  "host": "wd4zclrobb.execute-api.us-east-1.amazonaws.com",  
  "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": {  
        "credentials": "arn:aws:iam::123456789012:role/apigAwsProxyRole",  
        "responses": {  
          "default": {  
            "statusCode": "200"  
          }  
        },  
        "requestTemplates": {  
          "application/json": "{\n}"  
        },  
        "uri": "arn:aws:apigateway:us-east-1:kinesis:action/ListStreams",  
        "httpMethod": "POST",  
        "requestParameters": {  
          "integration.request.header.Content-Type": "'application/x-amz-  
json-1.1'"  
        },  
        "type": "aws"  
      }  
    }  
  }  
}
```



```

    "schema": {
      "$ref": "#/definitions/Empty"
    }
  },
  "x-amazon-apigateway-integration": {
    "credentials": "arn:aws:iam::123456789012:role/apigAwsProxyRole",
    "responses": {
      "default": {
        "statusCode": "200"
      }
    },
    "requestTemplates": {
      "application/json": "{\n  \n  \"ShardCount\": 5,\n  \n  \"StreamName\n\": \"\${input.params('stream-name')}\n\n}"
    },
    "uri": "arn:aws:apigateway:us-east-1:kinesis:action/CreateStream",
    "httpMethod": "POST",
    "requestParameters": {
      "integration.request.header.Content-Type": "'application/x-amz-
json-1.1'"
    },
    "type": "aws"
  }
},
"delete": {
  "consumes": [
    "application/json"
  ],
  "produces": [
    "application/json"
  ],
  "parameters": [
    {
      "name": "stream-name",
      "in": "path",
      "required": true,
      "type": "string"
    }
  ],
  "responses": {
    "200": {
      "description": "200 response",
      "schema": {
        "$ref": "#/definitions/Empty"
      },
      "headers": {
        "Content-Type": {
          "type": "string"
        }
      }
    },
    "400": {
      "description": "400 response",
      "headers": {
        "Content-Type": {
          "type": "string"
        }
      }
    }
  }
}

```



```

    "name": "stream-name",
    "in": "path",
    "required": true,
    "type": "string"
  }
],
"responses": {
  "200": {
    "description": "200 response",
    "schema": {
      "$ref": "#/definitions/Empty"
    }
  }
},
"x-amazon-apigateway-integration": {
  "credentials": "arn:aws:iam::123456789012:role/apigAwsProxyRole",
  "responses": {
    "default": {
      "statusCode": "200"
    }
  },
  "requestTemplates": {
    "application/json": "{\n  \n  \"StreamName\n\": \n  \"${input.params('stream-name')}\", \n  \n  \"Data\":\n  \"${util.base64Encode($input.path('$.Data'))}\", \n  \n  \"PartitionKey\":\n  \"${input.path('$.PartitionKey')}\" \n  \n}"
  },
  "uri": "arn:aws:apigateway:us-east-1:kinesis:action/PutRecord",
  "httpMethod": "POST",
  "requestParameters": {
    "integration.request.header.Content-Type": "'application/x-amz-
json-1.1'"
  },
  "type": "aws"
}
},
"/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"
      }
    ]
  }
},
],

```



```

    "200": {
      "description": "200 response",
      "schema": {
        "$ref": "#/definitions/Empty"
      }
    },
    "x-amazon-apigateway-integration": {
      "credentials": "arn:aws:iam::123456789012:role/apigAwsProxyRole",
      "responses": {
        "default": {
          "statusCode": "200"
        }
      },
      "requestTemplates": {
        "application/json": "{\n  \\"StreamName\":\n  \\"$input.params('stream-name')\"," ,\n  \\"Records\": [\n    #foreach($elem\n    in $input.path('$.records'))\n      {\n        \\"Data\":\n        \\"$util.base64Encode($elem.data)\",\n        \\"PartitionKey\":\n        \\"$elem.partition-key\\"\n      }#if($foreach.hasNext),#end\n    #end\n  ]\n}",
        "application/x-amz-json-1.1": "#set($inputRoot =\n  $input.path('$'))\n{\n  \\"StreamName\": \\"$input.params('stream-name')\"," ,\n  \\"records\": [\n    #foreach($elem in $inputRoot.records)\n      {\n        \\"Data\": \\"$elem.data\"," ,\n        \\"partition-key\": \\"$elem.partition-key\\"\n      }#if($foreach.hasNext),#end\n    #end\n  ]\n}"
      },
      "uri": "arn:aws:apigateway:us-east-1:kinesis:action/PutRecords",
      "httpMethod": "POST",
      "requestParameters": {
        "integration.request.header.Content-Type": "'application/x-amz-  

json-1.1'"
      }
    },
    "type": "aws"
  }
},
"/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": {
          "$ref": "#/definitions/Empty"
        }
      },
      "x-amazon-apigateway-integration": {
        "credentials": "arn:aws:iam:123456789012:role/apigAwsProxyRole",
        "responses": {
          "default": {
            "statusCode": "200"
          }
        },
        "requestTemplates": {
          "application/json": "{\n  \\"ShardId\\": \\"$input.params('shard-id')\\",\n  \\"ShardIteratorType\\": \\"TRIM_HORIZON\\",\n  \\"StreamName\\": \\"$input.params('stream-name')\\"}\n"
        },
        "uri": "arn:aws:apigateway:us-east-1:kinesis:action/GetShardIterator",
        "httpMethod": "POST",
        "requestParameters": {
          "integration.request.header.Content-Type": "'application/x-amz-json-1.1'"
        },
        "type": "aws"
      }
    }
  },
  "definitions": {
    "PutRecordsMethodRequestPayload": {
      "type": "object",
      "properties": {
        "records": {
          "type": "array",
          "items": {
            "type": "object",
            "properties": {
              "data": {
                "type": "string"
              },
              "partition-key": {
                "type": "string"
              }
            }
          }
        }
      }
    },
    "Empty": {
      "type": "object"
    }
  }
}

```

Amazon API Gateway REST API

When you use the Amazon API Gateway console to create, configure, update, and deploy an API, the console calls the API Gateway REST API behind the scenes to make things happen.

When you use AWS Command Line Interface to create, configure, update, and deploy an API, the AWS CLI tool calls the API Gateway REST API as well. For an example, see [Create an API using API Gateway and Test It](#) in the *AWS Lambda Developer Guide* . For more information, see [AWS Command Line Interface User Guide](#).

When you use an [AWS SDK](#) to create, configure, update, and deploy an API, the SDK calls the API Gateway REST API behind the scenes.

Instead, you can call the API Gateway REST API directly to create, configure, update, and deploy an API in API Gateway.

For more information on how to use the API Gateway REST API, see [Amazon API Gateway REST API Reference](#).

Amazon API Gateway Limits and Pricing

Topics

- [API Gateway Limits \(p. 419\)](#)
- [API Gateway Pricing \(p. 421\)](#)
- [Known Issues \(p. 421\)](#)

API Gateway Limits

Unless noted otherwise, the limits can be increased upon request. To request a limit increase, contact the [AWS Support Center](#).

API Gateway Limits for Configuring and Running an API

The following limits apply to configuring and running an API in Amazon API Gateway.

Resource or Operation	Default Limit	Can Be Increased
Throttle limits per account	1000 request per second (rps) with a burst limit of 2000 rps.	Yes
APIs per account	60	Yes
API keys per account	500	Yes
Client certificates per account	60	Yes
Custom authorizers per API	10	Yes

Resource or Operation	Default Limit	Can Be Increased
Documentation parts per account	2000	Yes
Resources per API	300	Yes
Stages per API	10	Yes
Usage plans per account	300	Yes
API caching TTL	300 seconds by default and configurable between 0 and 3600 by an API owner.	Not for the upper bound (3600)
Integration timeout	30 seconds for all integration types, including Lambda, Lambda proxy, HTTP, HTTP proxy, and AWS integrations.	No
Payload size	10 MB	No
Number of iterations in a <code>#foreach ... #end</code> loop in mapping templates	1000	No
ARN length of a method with authorization	1600 bytes	No

When authorization is enabled on a method, the maximum length of the method's ARN (e.g., `arn:aws:execute-api:{region-id}:{account-id}:{api-id}/{stage-id}/{method}/{resource}/{path}`) is 1600 bytes. The path parameter values, the size of which are determined at run time, can cause the ARN length to exceed the limit. When this happens, the API client will receive a `414 Request URI too long` response.

API Gateway Limits for Creating, Deploying and Managing an API

The following fixed limits apply to creating, deploying, and managing an API in API Gateway, using the AWS CLI, the API Gateway console, or the API Gateway REST API and its SDKs. These limits cannot be increased.

Action	Default Limit	Can Be Increased
CreateRestApi	2 requests per minute (rpm) per account.	No
ImportRestApi	2 requests per minute per account	No
PutRestApi	60 requests per minutes per account	No
DeleteRestApi	2 requests per minutes per account	No

Action	Default Limit	Can Be Increased
CreateDeployment	3 requests per minutes per account	No
UpdateAccount	3 requests per minutes per account	No
GetResources	150 requests per minutes per account	No
CreateResource	300 requests per minutes per account	No
DeleteResource	300 requests per minutes per account	No
CreateDomainName	2 requests per minutes per account	No
UpdateUsagePlan	3 requests per minutes per account	No
Other operations	No limit up to the total account limit.	No
Total operations	10 request per second (rps) with a burst limit of 40 rps.	No

API Gateway Pricing

For API Gateway region-specific pricing information, see [Amazon API Gateway Pricing](#).

Note

API caching in Amazon API Gateway is not eligible for the AWS Free Tier.

Known Issues

- Cross-account authentication is not currently supported in API Gateway. An API caller must be an IAM user of the same AWS account of the API owner.
- When using the API Gateway console to test an API, you may get an "unknown endpoint errors" response if a self-signed certificate is presented to the back end, the intermediate certificate is missing from the certificate chain, or any other unrecognizable certificate-related exceptions thrown by the back end.
- API Gateway enacts the following restrictions and limitations when handling methods with either Lambda proxy integration or HTTP proxy integration.
 - Duplicated query string parameters are not supported.
 - Duplicated headers are not supported.
 - The `Host` header will not be forwarded to HTTP endpoints.
 - The following headers may be remapped to `x-amzn-Remapped-HEADER` when sent to your integration endpoint or sent back by your integration endpoint:
 - `Accept`
 - `Accept-Charset`

- Accept-Encoding
- Age
- Authorization
- Connection
- Content-Encoding
- Content-Length
- Content-MD5
- Content-Type
- Date
- Expect
- Host
- Max-Forwards
- Pragma
- Proxy-Authenticate
- Range
- Referer
- Server
- TE
- Trailer
- Transfer-Encoding
- Upgrade
- User-Agent
- Via
- Warn

Document History

The following table describes the important changes to the documentation since the last release of the API Gateway Developer Guide.

- **Latest documentation update:** December 1, 2016

Change	Description	Date Changed
Integrating with AWS Marketplace	Sell your API in a usage plan as a SaaS product through AWS Marketplace. Use AWS Marketplace to extend the reach of your API. Rely on AWS Marketplace for customer billing on your behalf. Let API Gateway handle user authorization and usage metering. For more information, see Selling an API as SaaS (p. 339) .	December 1, 2016
Enabling Documentation Support for your API	Add documentation for API entities in DocumentationPart resources in API Gateway. Associate a snapshot of the collection <code>DocumentationPart</code> instances with an API stage to create a DocumentationVersion . Publish API documentation by exporting a documentation version to an external file, such as a Swagger file. For more information, see Documenting an API (p. 287) .	December 1, 2016
Updated custom authorizer	A customer authorizer Lambda function now returns the caller's principal identifier. The function also can return other information as key-value pairs of the <code>context</code> map and an IAM policy. For more information, see Output from an Amazon API Gateway Custom Authorizer (p. 177) .	December 1, 2016
Supporting binary payloads	Set binaryMediaTypes on your API to support binary payloads of a request or response. Set the <code>contentHandling</code> property on an Integration or IntegrationResponse to specify whether to handle a binary payload as the native binary blob, as a Base64-encoded string, or as a passthrough without modifications. For more information, see Enable Support for Binary Payloads in API Gateway (p. 78) .	November 17, 2016

Change	Description	Date Changed
Enabling a proxy integration with an HTTP or Lambda back end through a proxy resource of an API.	Create a proxy resource with a greedy path parameter of the form <code>{proxy+}</code> and the catch-all <code>ANY</code> method. The proxy resource is integrated with an HTTP or Lambda back end using the HTTP or Lambda proxy integration, respectively. For more information, see Configure Proxy Integration for a Proxy Resource (p. 97) .	September 20, 2016
Extending selected APIs in API Gateway as product offerings for your customers by providing one or more usage plans.	Create a usage plan in API Gateway to enable selected API clients to access specified API stages at agreed-upon request rates and quotas. For more information, see Use API Gateway Usage Plans (p. 212) .	August 11, 2016
Enabling method-level authorization with a user pool in Amazon Cognito	Create a user pool in Amazon Cognito and use it as your own identity provider. You can configure the user pool as a method-level authorizer to grant access for users who are registered with the user pool. For more information, see Use Amazon Cognito Your User Pool (p. 182) .	July 28, 2016
Enabling Amazon CloudWatch metrics and dimensions under the AWS/ApiGateway namespace.	The API Gateway metrics are now standardized under the CloudWatch namespace of AWS/ApiGateway. You can view them in both the API Gateway console and the Amazon CloudWatch console. For more information, see Amazon API Gateway Dimensions and Metrics (p. 334) .	July 28, 2016
Enabling certificate rotation for a custom domain name	Certificate rotation allows you to upload and renew an expiring certificate for a custom domain name. For more information, see Upload and Renew an Expiring Certificate (p. 285) .	April 27, 2016
Documenting changes for the updated Amazon API Gateway console.	Learn how to create and set up an API using the updated API Gateway console. For more information, see Create an API Gateway API from an Example (p. 7) and Build an API Gateway API to Expose an HTTP Endpoint (p. 6) .	April 5, 2016
Enabling the Import API feature to create a new or update an existing API from external API definitions.	With the Import API features, you can create a new API or update an existing one by uploading an external API definition expressed in Swagger 2.0 with the API Gateway extensions. For more information about the Import API, see Import an API (p. 142) .	April 5, 2016
Exposing the <code>\$input.body</code> variable to access the raw payload as string and the <code>\$util.parseJson()</code> function to turn a JSON string into a JSON object in a mapping template.	For more information about <code>\$input.body</code> and <code>\$util.parseJson()</code> , see Request and Response Payload-Mapping Reference (p. 134) .	April 5, 2016

Change	Description	Date Changed
Enabling client requests with method-level cache invalidation, and improving request throttling management.	Flush API stage-level cache and invalidate individual cache entry. For more information, see Flush the API Stage Cache in API Gateway (p. 240) and Invalidate an API Gateway Cache Entry (p. 240) . Improve the console experience for managing API request throttling. For more information, see Manage API Request Throttling (p. 236) .	March 25, 2016
Enabling and calling API Gateway API using custom authorization	Create and configure an AWS Lambda function to implement custom authorization. The function returns an IAM policy document that grants the Allow or Deny permissions to client requests of an API Gateway API. For more information, see Use API Gateway Custom Authorizers (p. 174) .	February 11, 2016
Importing and exporting API Gateway API using a Swagger definition file and extensions	Create and update your API Gateway API using the Swagger specification with the API Gateway extensions. Import the Swagger definitions using the API Gateway Importer. Export an API Gateway API to a Swagger definition file using the API Gateway console or API Gateway Export API. For more information, see Import and Export API (p. 141) .	December 18, 2015
Mapping request or response body or body's JSON fields to request or response parameters.	Map method request body or its JSON fields into integration request's path, query string, or headers. Map integration response body or its JSON fields into request response's headers. For more information, see Request and Response Parameter-Mapping Reference (p. 131) .	December 18, 2015
Working with Stage Variables in Amazon API Gateway	Learn how to associate configuration attributes with a deployment stage of an API in Amazon API Gateway. For more information, see Manage API Gateway API Deployment with Stage Variables (p. 242) .	November 5, 2015
How to: Enable CORS for a Method	It is now easier to enable cross-origin resource sharing (CORS) for methods in Amazon API Gateway. For more information, see Enable CORS for a Resource (p. 170) .	November 3, 2015
How to: Use Client Side SSL Authentication	Use Amazon API Gateway to generate SSL certificates that you can use to authenticate calls to your HTTP backend. For more information, see Use Client-Side SSL Certificates (p. 186) .	September 22, 2015
Mock integration of methods	Learn how to mock-integrate an API with Amazon API Gateway (p. 94) . This feature enables developers to generate API responses from API Gateway directly without the need for a final integration back end beforehand.	September 1, 2015
Amazon Cognito Identity support	Amazon API Gateway has expanded the scope of the <code>\$context</code> variable so that it now returns information about Amazon Cognito Identity when requests are signed with Amazon Cognito credentials. In addition, we have added a <code>\$util</code> variable for escaping characters in JavaScript and encoding URLs and strings. For more information, see Request and Response Payload-Mapping Reference (p. 134) .	August 28, 2015

Change	Description	Date Changed
Swagger integration	Use the Swagger import tool on GitHub to import Swagger API definitions into Amazon API Gateway. Learn more about Import and Export API (p. 141) to create and deploy APIs and methods using the import tool. With the Swagger importer tool you can also update existing APIs.	July 21, 2015
Mapping Template Reference	Read about the <code>\$input</code> parameter and its functions in the Request and Response Payload-Mapping Reference (p. 134) .	July 18, 2015
Initial public release	This is the initial public release of the <i>Amazon API Gateway Developer Guide</i> .	July 9, 2015

AWS Glossary

For the latest AWS terminology, see the [AWS Glossary](#) in the *AWS General Reference*.