Amazon API Gateway

Developer Guide

Amazon API Gateway: Developer Guide

Table of Contents

What Is Amazon API Gateway?

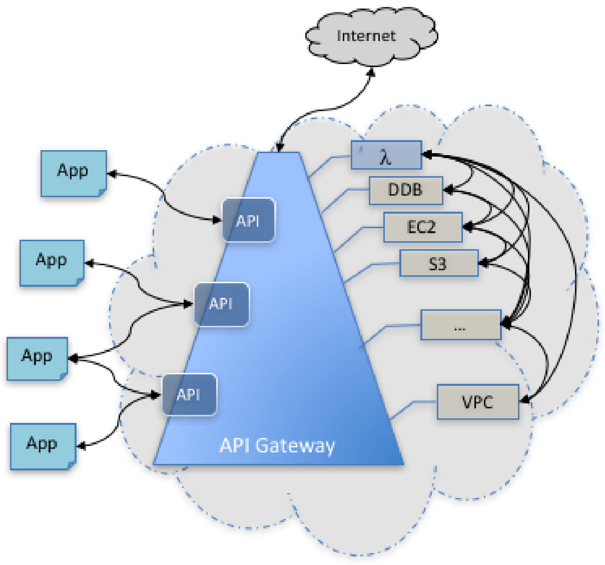
Amazon API Gateway is an AWS service that enables developers to create, publish, maintain, monitor, and secure APIs at any scale. You can create APIs that access AWS or other web services, as well as data stored in the AWS Cloud.

Topics

Gateway to AWS Cloud and Beyond

API Gateway can be considered a backplane in the cloud to connect AWS services and other public or private web sites. It provides consistent RESTful application programming interfaces (APIs) for mobile and web applications to access AWS services.

The following diagram shows API Gateway architecture.



In practical terms, API Gateway lets you create, configure, and host a RESTful API to enable applications to access the AWS Cloud. For example, an application using API Gateway can upload a user's annual income and expense data to Amazon S3 or Amazon DynamoDB, process the data in AWS Lambda to compute tax owed, and file a tax return via the IRS website.

As shown in the architecture diagram, an app (or client) gains programmatic access to AWS services or a website on the internet through one or more APIs, which are hosted in API Gateway. The app is at the API's frontend. The integrated AWS services and websites are located at the API's backend.

With Amazon API Gateway, you can provide your users with an integrated and consistent developer experience for building AWS cloud-based applications.

Part of AWS Serverless Infrastructure

Together with AWS Lambda, API Gateway forms the app-facing part of the AWS serverless infrastructure. For an app to call publicly available AWS services, you can use Lambda to interact with the required services and expose the Lambda functions through API methods in API Gateway. AWS Lambda runs the code on a highly available computing infrastructure. It performs the necessary execution and administration of the computing resources. To enable the serverless applications, API Gateway supports streamlined proxy integrations (p. 111) with AWS Lambda and HTTP endpoints.

Developer Experiences

There are two kinds of developers who use API Gateway: an app developer and an API developer. The app developer builds a functioning application to call AWS services through API Gateway. The API developer creates and deploys an API to enable the required functionality in API Gateway. The API developer must be an IAM user in the AWS account that owns the API. The app developer does not need to have an AWS account, provided that the API does not require IAM permissions.

Create and Manage an API Gateway API

An API developer works with the API Gateway service component, named `apigateway`, to create, configure, and deploy an API. Each API includes a set of resources and methods. A resource is a logical entity that an app can access through a resource path.

For example, `/incomes` is the path of the resource representing the income of the app user. A resource can have one or more operations that are defined by appropriate HTTP verbs such as GET, POST, PUT, PATCH, and DELETE. A combination of a resource path and an operation identify a method of the API. For example, the `POST /incomes` method adds an income earned by the caller, and the `GET /expenses` method queries the reported expenses incurred by the caller.

A method corresponds to a REST API request submitted by the user of your API and the corresponding response returned to the user. The app does not need to know where the requested data is stored and fetched from. The API interfaces with the backend using the integration request or integration response.

For example, with DynamoDB as the backend, the API developer sets up the integration request to forward the incoming method request to the chosen backend. The setup includes specifications of an appropriate DynamoDB action, required IAM role and policies, and required input data transformation. The backend returns the result to API Gateway as an integration response. To route the integration response to an appropriate method response (of a given HTTP status code) to the client, you can configure the integration response to map required response parameters from integration to method. You then translate the output data format of the backend to that of the frontend, if necessary. API Gateway enables you to define a schema or model for the payload to facilitate setting up the body mapping template.

As an API developer, you can create and manage an API by using the API Gateway management console, described in Getting Started (p. 6), or by calling the API Gateway REST API (p. 449). There are several ways to call this API. They include using the AWS Command-Line Interface (CLI), or by using an AWS SDK. You can also use a REST API client, such as Postman, to make raw API calls. In addition, you can enable API creation with AWS CloudFormation templates or API Gateway Extensions to Swagger (p. 358). For a list of regions where API Gateway is available, as well as the associated control service endpoints, see Regions and Endpoints.

Calling an API Gateway API

An app developer works with the API Gateway service component, named `execute-api`, to invoke an API that was created or deployed in API Gateway. The underlying programming entities are exposed by the created API. There are several ways to call such an API. You can use the API Gateway console to test invoking the API. You can use a REST API client, such as CURL or Postman, or an SDK generated by API Gateway for the API to invoke the API.

Be aware of the differences between the `apigateway` and `execute-api` API Gateway service components. Reference the appropriate service component name when you select one while, for example, setting IAM permission policies for building or calling an API.

Benefits of API Gateway

API Gateway helps you deliver robust, secure, and scalable mobile and web application backends. API Gateway allows you 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, you can create and operate APIs for backend services. For example, you don't need to develop and maintain 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 backend 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 be provided by a publicly accessible endpoint that API Gateway proxies call, or it can be entirely run as a Lambda function.

Amazon API Gateway Concepts

API Gateway

API Gateway is an AWS service that supports the following:

1. Creating, deploying, and managing a RESTful application programming interface (API) to expose backend HTTP endpoints, AWS Lambda functions, or other AWS services.
2. Invoking exposed API methods through the frontend HTTP endpoints.

API Gateway API

A collection of resources and methods that are integrated with backend HTTP endpoints, Lambda functions, or other AWS services. The collection can be deployed in one or more stages. API methods are invoked through frontend HTTP endpoints that you can associate with a registered custom domain name. Permissions to invoke a method are granted using IAM roles and policies or API Gateway custom authorizers. An API can present a certificate to be authenticated by the backend. 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 that 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. The entity 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 that is generated by API Gateway on behalf of an API owner or imported from an external source such as a CSV file. The string is used to identify an app developer of the API. An API owner can use API keys to permit or deny access of specific 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 a client to invoke, the deployment must be associated with one or more stages. A stage is a logical reference to a lifecycle status of your API (for example, '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 backend 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 backend.

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 backend 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 frontend and an HTTP backend. For the Lambda proxy integration, API Gateway sends the entire request as an input to a backend Lambda function. API Gateway then transforms the Lambda function output to a frontend 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 frontend data format to the backend data format. Or the scripts can transform a response body from the backend data format to the frontend data format. Mapping templates are specified in the integration request or integration response. They can reference data made available at run time as context and stage variables. An identity transformation is referred to as a passthrough, in which a payload is passed as-is from the client to the backend for a request and from the backend to the client for a response.

Model

Data schema specifying the data structure of a request or response payload. It is required for generating a strongly typed SDK of an API. It is also used to validate payload. A model is 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. You can use a usage plan to configure throttling and quota limits, which are 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

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 backend, API Gateway must have permissions to assume the roles required to call the backend 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

- Next Step (p. 8)

Sign Up for AWS

If you do not have an AWS account, use the following procedure to create one.

To sign up for AWS

1. Open https://aws.amazon.com/ and choose **Create an AWS Account**.
2. Follow the online instructions.

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 or 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. 214).

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. 214).

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. 9).

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

- Map Response Payload (p. 33)

Create an API Gateway API from an Example

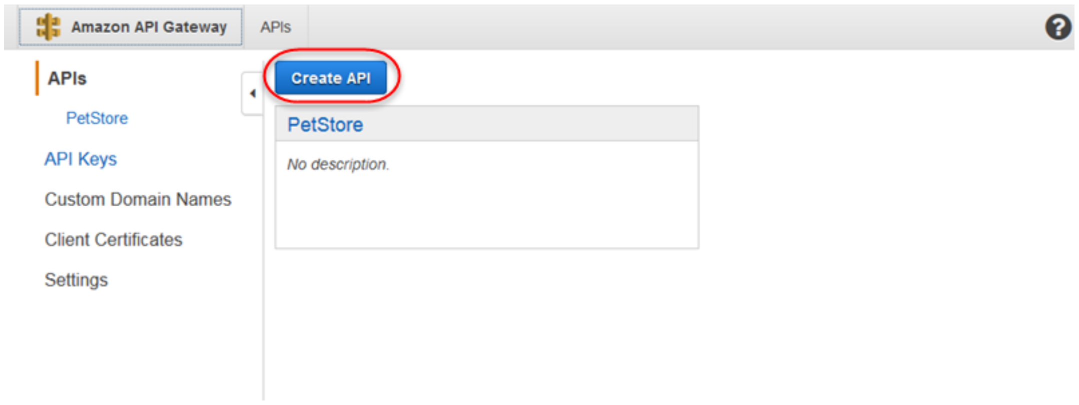
The Amazon API Gateway console now provides an option for you to create an API Gateway API by example, with helpful hints provided along the way. If you are new to API Gateway, you may find it useful as a learning tool. The following steps walk you through using this create-by-example option to create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

1. Do one of the following:

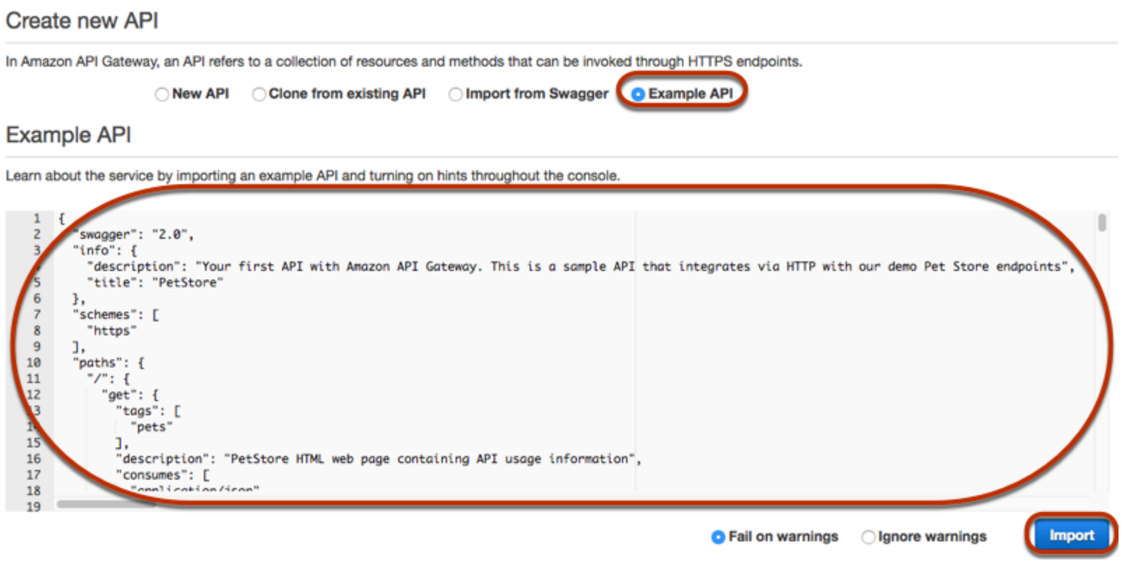
   a. For the first API in your account, choose **Get Started** from the API Gateway console welcome page.

      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:



2. 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.



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]

Example API

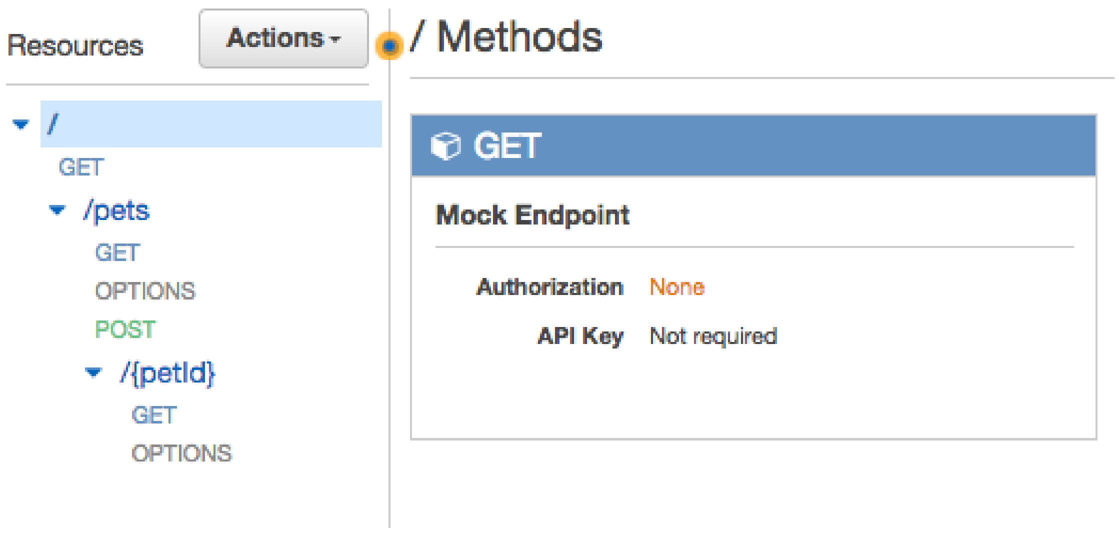
Learn about the service by importing an example API and turning on hints throughout the console.

```
{
  "swagger": "2.0",
  "info": {
    "description": "Your first API with Amazon API Gateway. This is a sample API that integrates via HTTP with our demo Pet Store endpoints",
    "title": "PetStore"
  },
  "schemes": [
    "https"
  ],
  "paths": {
    "/": {
      "get": {
        "tags": [
          "pets"
        ],
        "description": "PetStore HTML web page containing API usage information",
        "consumes": [
          "application/json"
```

[✓ Fail on warnings] Ignore warnings Import

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

3. The newly created API is shown as follows:



Resources Actions

/
GET
/pets
GET
OPTIONS
POST
/{petId}
GET
OPTIONS

/ Methods

GET

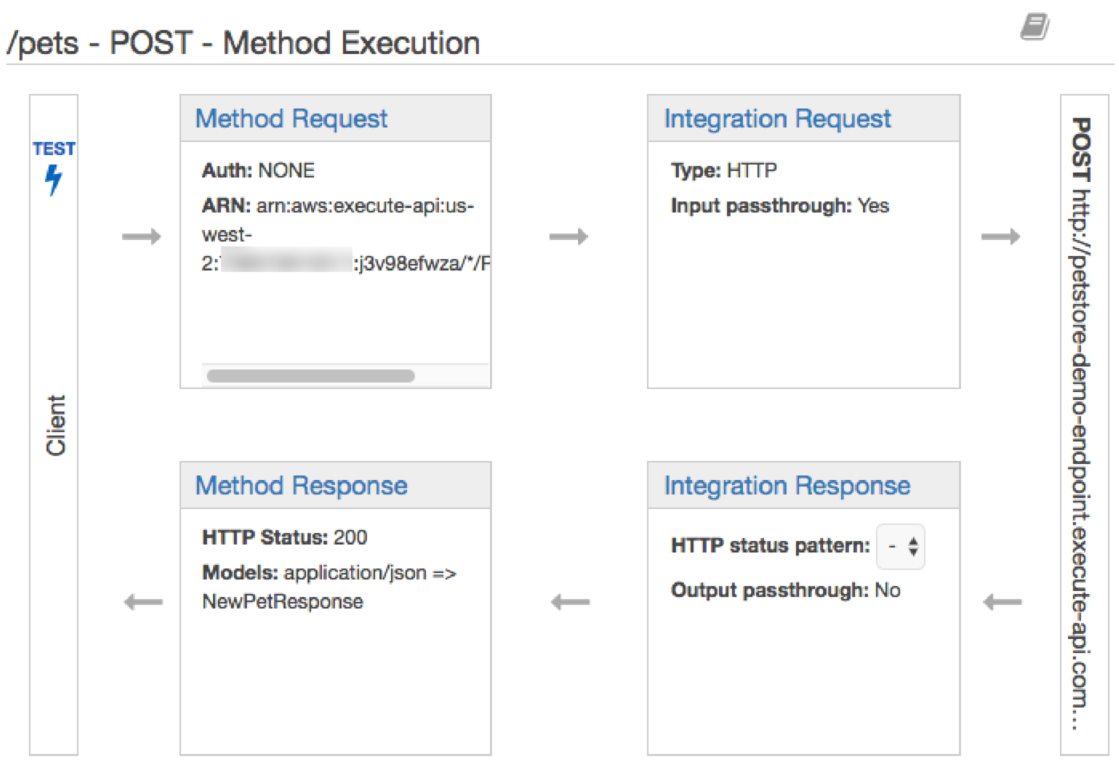
Mock Endpoint

Authorization None

API Key Not required

The **Resources** pane shows the structure of the created 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. Here, we choose the `POST /pets` method as an illustration:



/pets - POST - Method Execution

Method Request
Auth: NONE
ARN: arn:aws:execute-api:us-west-2:[redacted]:j3v98efwza/*/F

Integration Request
Type: HTTP
Input passthrough: Yes

Method Response
HTTP Status: 200
Models: application/json => NewPetResponse

Integration Response
HTTP status pattern: -
Output passthrough: No

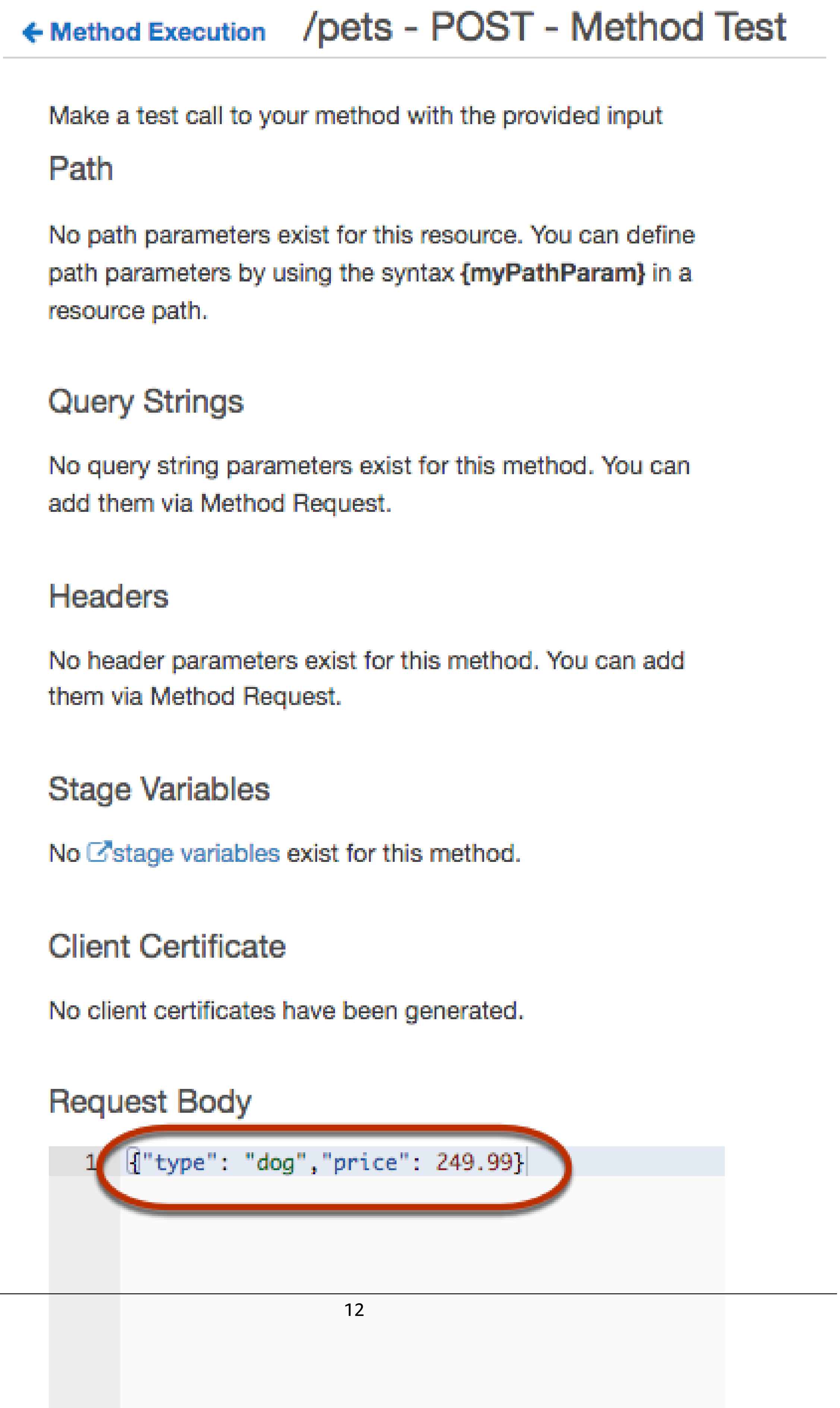
The resulting **Method Execution** pane presents a logical view of the chosen (`POST /pets`) method's structure and behaviors: **Method Request** and **Method Response** are the API's interface with the API's frontend (a client), whereas **Integration Request** and **Integration Response** are the API's interface with the backend (`http://petstore-demo-endpoint.execute-api.com/pets`). A client uses the API to access a backend feature through the **Method Request**. API Gateway translates the client request, if necessary, into the form acceptable to the backend in **Integration Request** before forwarding the incoming request to the backend. The transformed request is known as the integration request . Similarly, the backend returns the response to API Gateway in **Integration Response**. API Gateway then routes it to **Method Response** before sending it to the client. Again, if necessary, API Gateway can map the backend response data to a form expected by the client.

For the `POST` method on an API resource, the method request payload can be passed through to the integration request without modification, if the method request's payload is of the same format as the integration request's payload.

The `GET /` method request uses the `MOCK` integration type and is not tied to any real backend endpoint. The corresponding **Integration Response** is set up to return a static HTML page. When the method is called, the API Gateway simply accepts the request and immediately returns the configured integration response to the client by way of **Method Response**. You can use the mock integration to test an API without requiring a backend endpoint. You can also use it to serve a local response, generated from a response body-mapping template.

As an API developer, you control the behaviors of your API's frontend interactions by configuring the method request and a method response. You control the behaviors of your API's backend interactions by setting up the integration request and integration response. These 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. 8). 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. For example, to test the `POST /pets` method, enter the following `{"type": "dog","price": 249.99}` payload into the **Request Body** before choosing the **Test** button.



← Method Execution /pets - 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

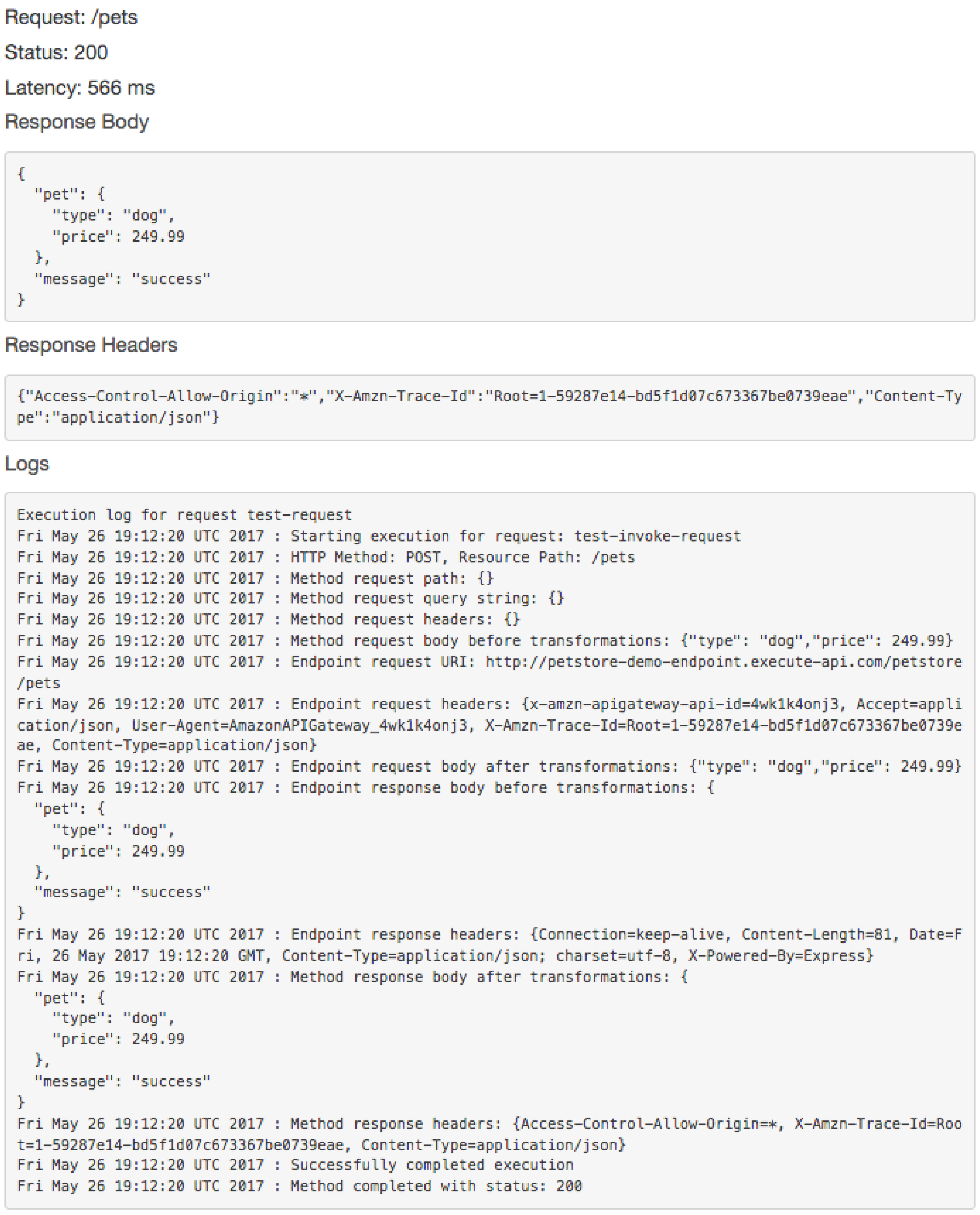
No client certificates have been generated.

Request Body

```
{"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: /pets

Status: 200

Latency: 566 ms

Response Body

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

Response Headers

```
{"Access-Control-Allow-Origin":"*","X-Amzn-Trace-Id":"Root=1-59287e14-bd5f1d07c673367be0739eae","Content-Ty
pe":"application/json"}
```

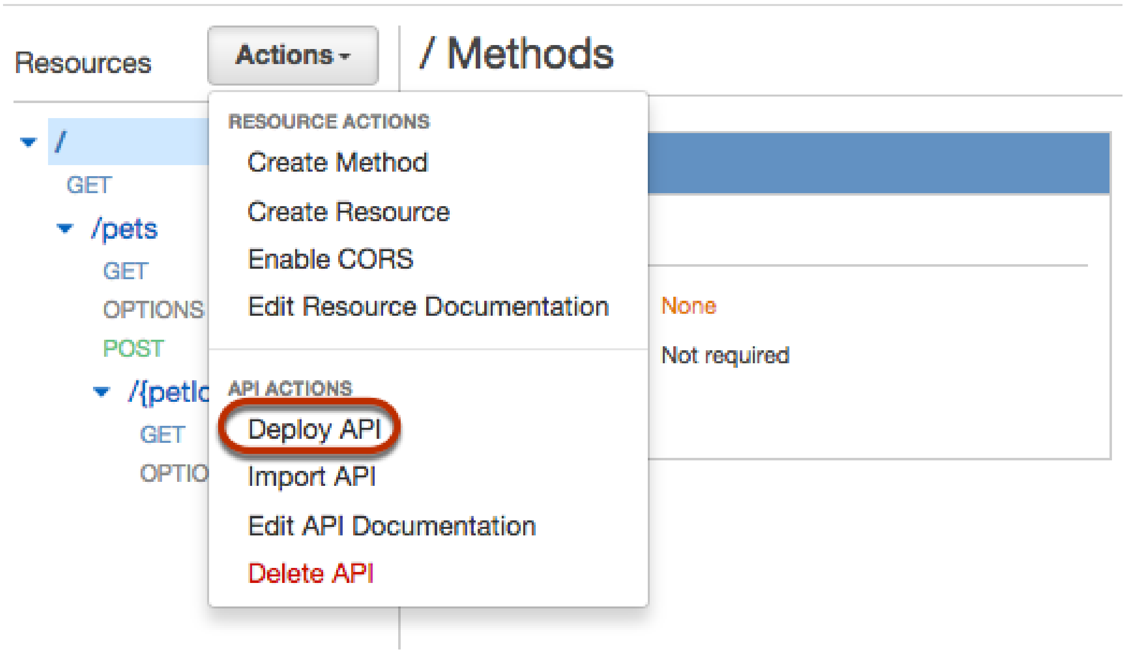
Logs

```
Execution log for request test-request
Fri May 26 19:12:20 UTC 2017 : Starting execution for request: test-invoke-request
Fri May 26 19:12:20 UTC 2017 : HTTP Method: POST, Resource Path: /pets
Fri May 26 19:12:20 UTC 2017 : Method request path: {}
Fri May 26 19:12:20 UTC 2017 : Method request query string: {}
Fri May 26 19:12:20 UTC 2017 : Method request headers: {}
Fri May 26 19:12:20 UTC 2017 : Method request body before transformations: {"type": "dog","price": 249.99}
Fri May 26 19:12:20 UTC 2017 : Endpoint request URI: http://petstore-demo-endpoint.execute-api.com/petstore
/pets
Fri May 26 19:12:20 UTC 2017 : Endpoint request headers: {x-amzn-apigateway-api-id=4wk1k4onj3, Accept=appli
cation/json, User-Agent=AmazonAPIGateway_4wk1k4onj3, X-Amzn-Trace-Id=Root=1-59287e14-bd5f1d07c673367be0739e
ae, Content-Type=application/json}
Fri May 26 19:12:20 UTC 2017 : Endpoint request body after transformations: {"type": "dog","price": 249.99}
Fri May 26 19:12:20 UTC 2017 : Endpoint response body before transformations: {
  "pet": {
    "type": "dog",
    "price": 249.99
  },
  "message": "success"
}
Fri May 26 19:12:20 UTC 2017 : Endpoint response headers: {Connection=keep-alive, Content-Length=81, Date=F
ri, 26 May 2017 19:12:20 GMT, Content-Type=application/json; charset=utf-8, X-Powered-By=Express}
Fri May 26 19:12:20 UTC 2017 : Method response body after transformations: {
  "pet": {
    "type": "dog",
    "price": 249.99
  },
  "message": "success"
}
Fri May 26 19:12:20 UTC 2017 : Method response headers: {Access-Control-Allow-Origin=*, X-Amzn-Trace-Id=Roo
t=1-59287e14-bd5f1d07c673367be0739eae, Content-Type=application/json}
Fri May 26 19:12:20 UTC 2017 : Successfully completed execution
Fri May 26 19:12:20 UTC 2017 : Method completed with status: 200
```

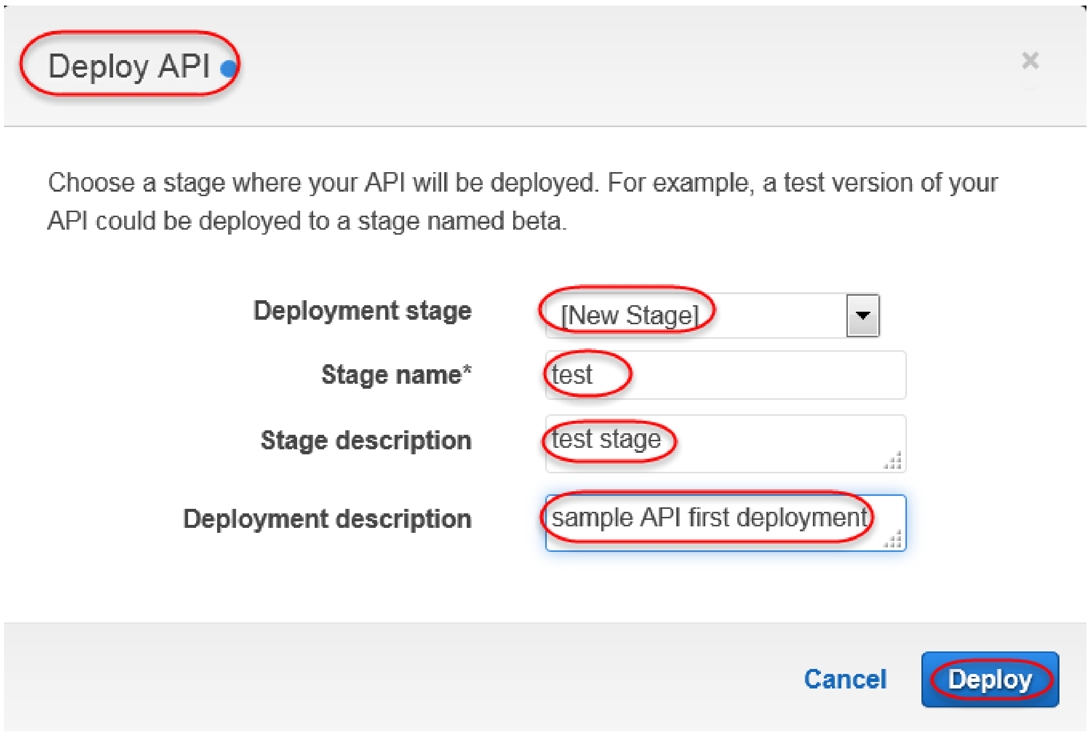
The **Logs** entry of the output shows the state changes from the method request to the integration request and from the integration response to the method response. This can be useful for troubleshooting any mapping errors that cause the request to fail. In this example, no mapping is applied: the method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



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	test stage
Deployment description	sample API first deployment

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

8. On **Stage Editor**, follow the **Invoke URL** link to submit the `GET /` method request in a browser. A successful response return the result, generated from the mapping template in the integration response.
9. In the **Stages** navigation pane, expand the **test** stage, select **GET** on `/pets/{petId}`, and then copy the **Invoke URL** value of `https://`*`api-id`*`.execute-api.`*`region`*`.amazonaws.com/test/pets/{petId}`. `{petId}` stands for a path variable.

   Paste the **Invoke URL** value (obtained in the previous step) into the address bar of a browser, replacing `{petId}` by, for example, `1`, and press Enter to submit the request. A 200 OK response should return with the following JSON payload:

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

   Invoking the API method as shown is possible because its **Authorization** type is set to `NONE`. If the `AWS_IAM` authorization were used, you would sign the request using the Signature Version 4 protocols. For an example of such a request, see Build an API Gateway API to Expose an HTTP Endpoint (p. 8).

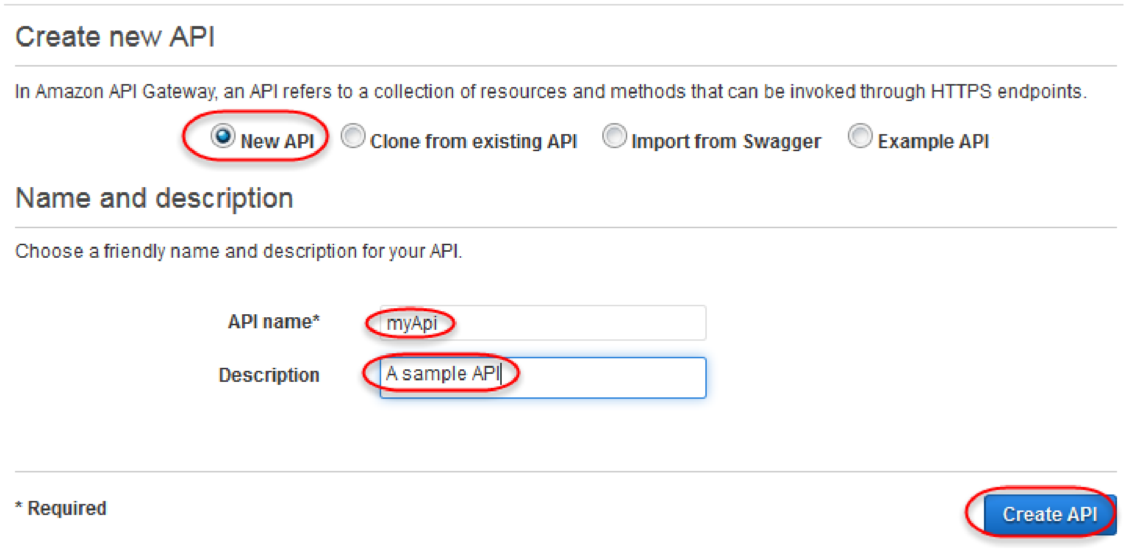
See Also

Use Custom Authorizers (p. 232), Deploying an API (p. 284)

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

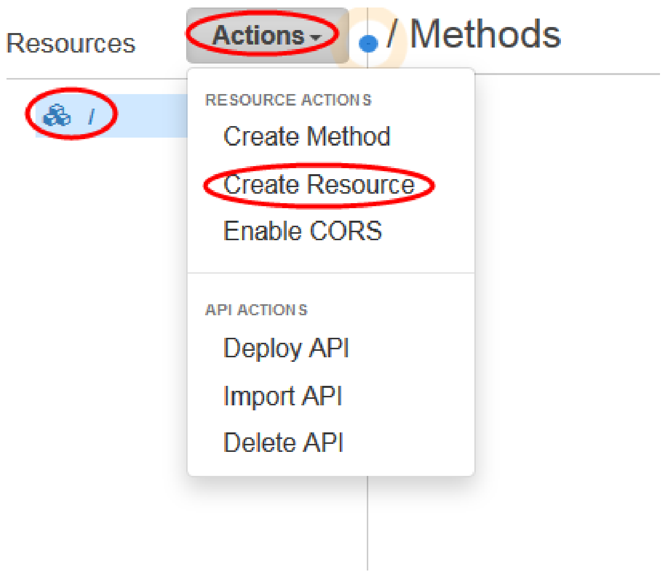
Choose a friendly name and description for your API.

API name*	myApi
Description	A sample API

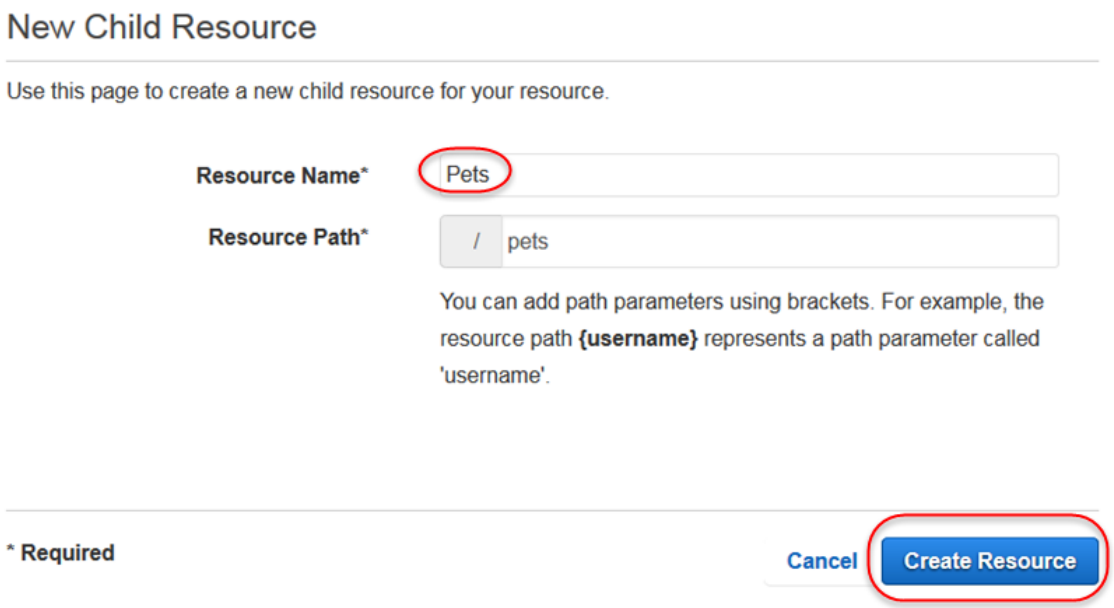
* Required

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.

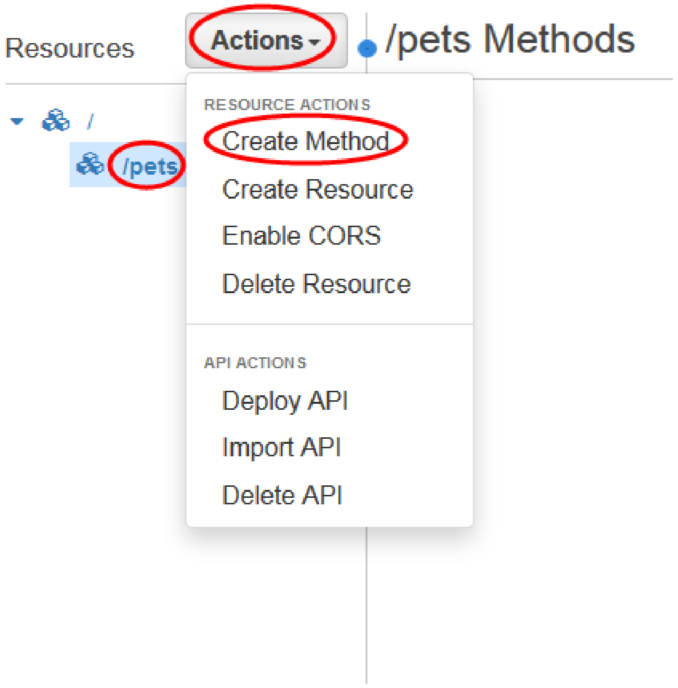
Resource Name* Pets

Resource Path* / pets

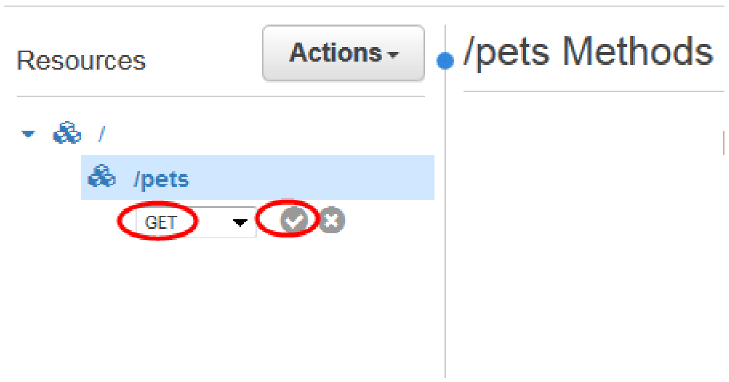
You can add path parameters using brackets. For example, the resource path **{username}** represents a path parameter called 'username'.

* **Required**

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 check mark icon to finish creating the method.



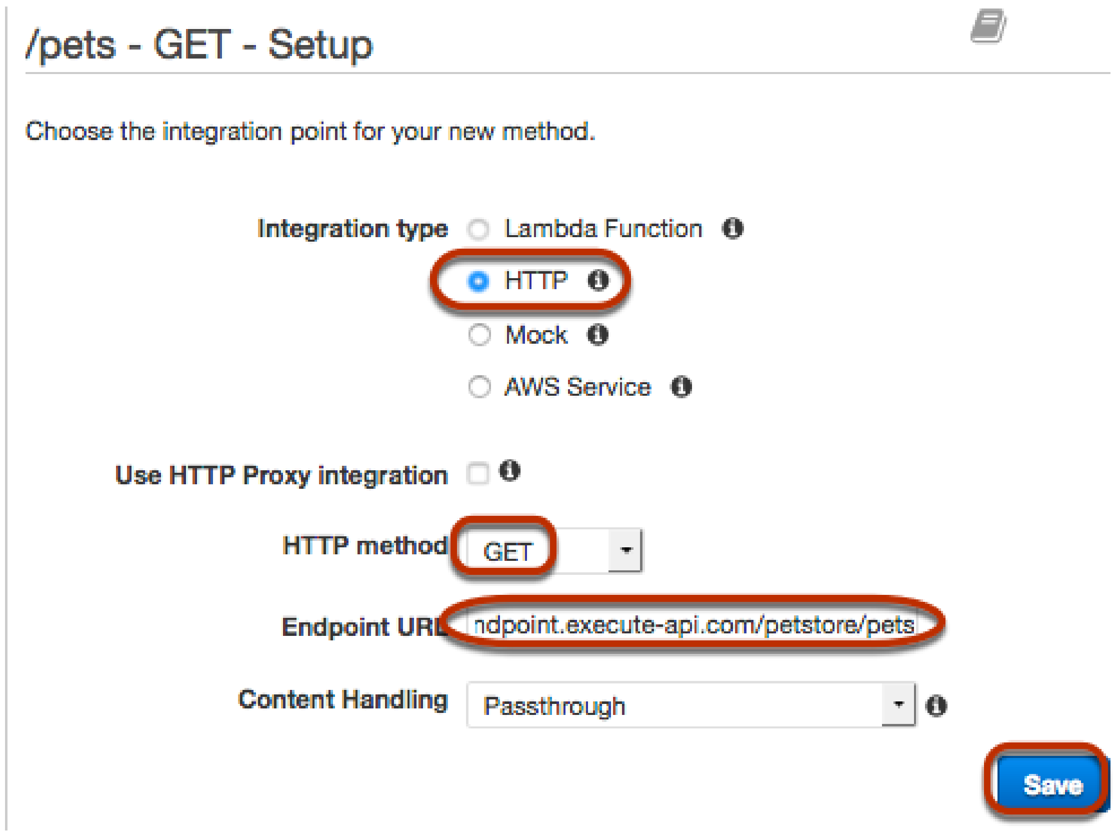
Note
Other options for an API method include:

- **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 backend. 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, leave all other settings as default, and then choose **Save**.

   Note
   For the integration request's **HTTP method**, you must choose one supported by the backend. 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

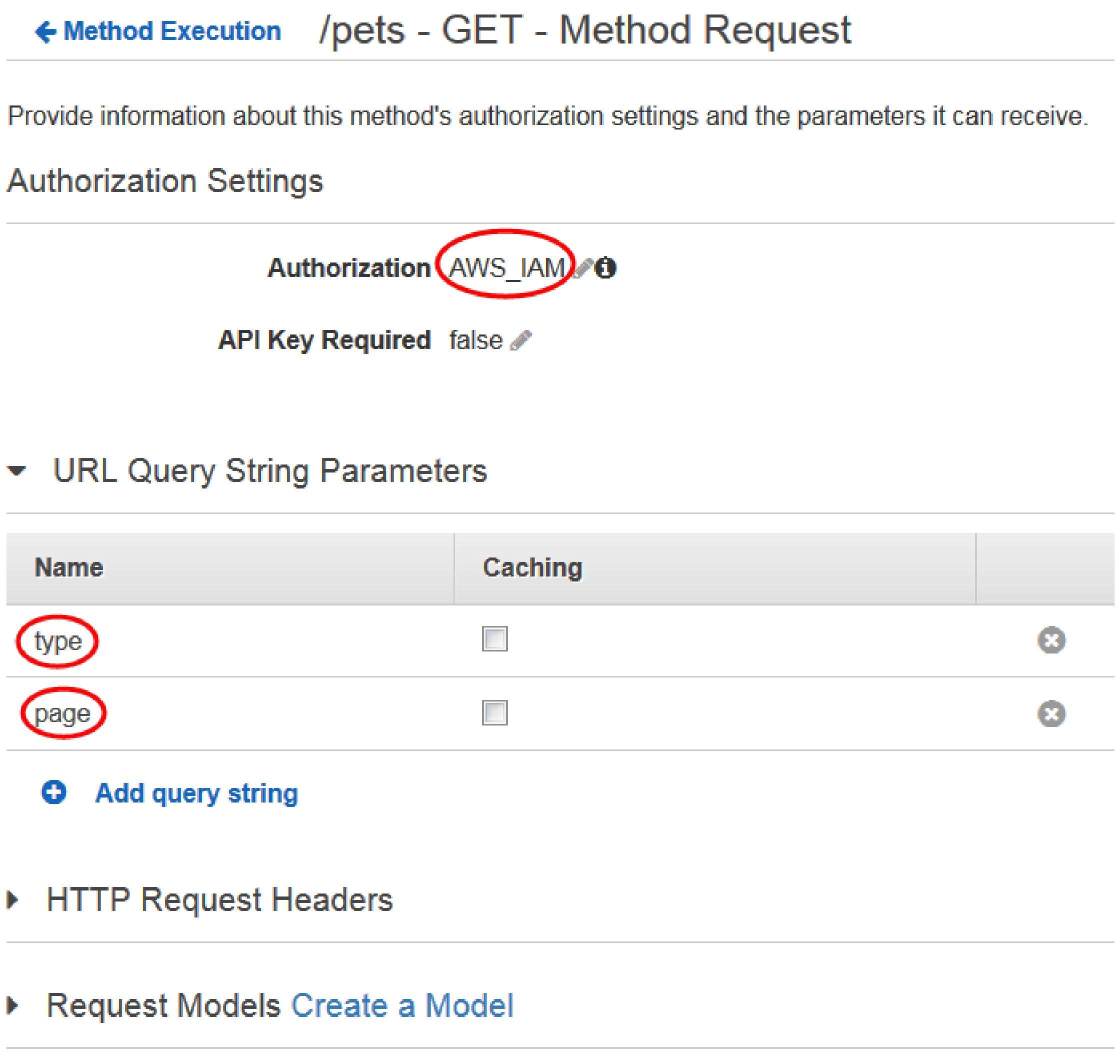
Endpoint URL: ndpoint.execute-api.com/petstore/pets

Content Handling: Passthrough

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 check mark icon to save the newly added query string parameters.



/pets - GET - Method Request

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

Authorization Settings

Authorization AWS_IAM

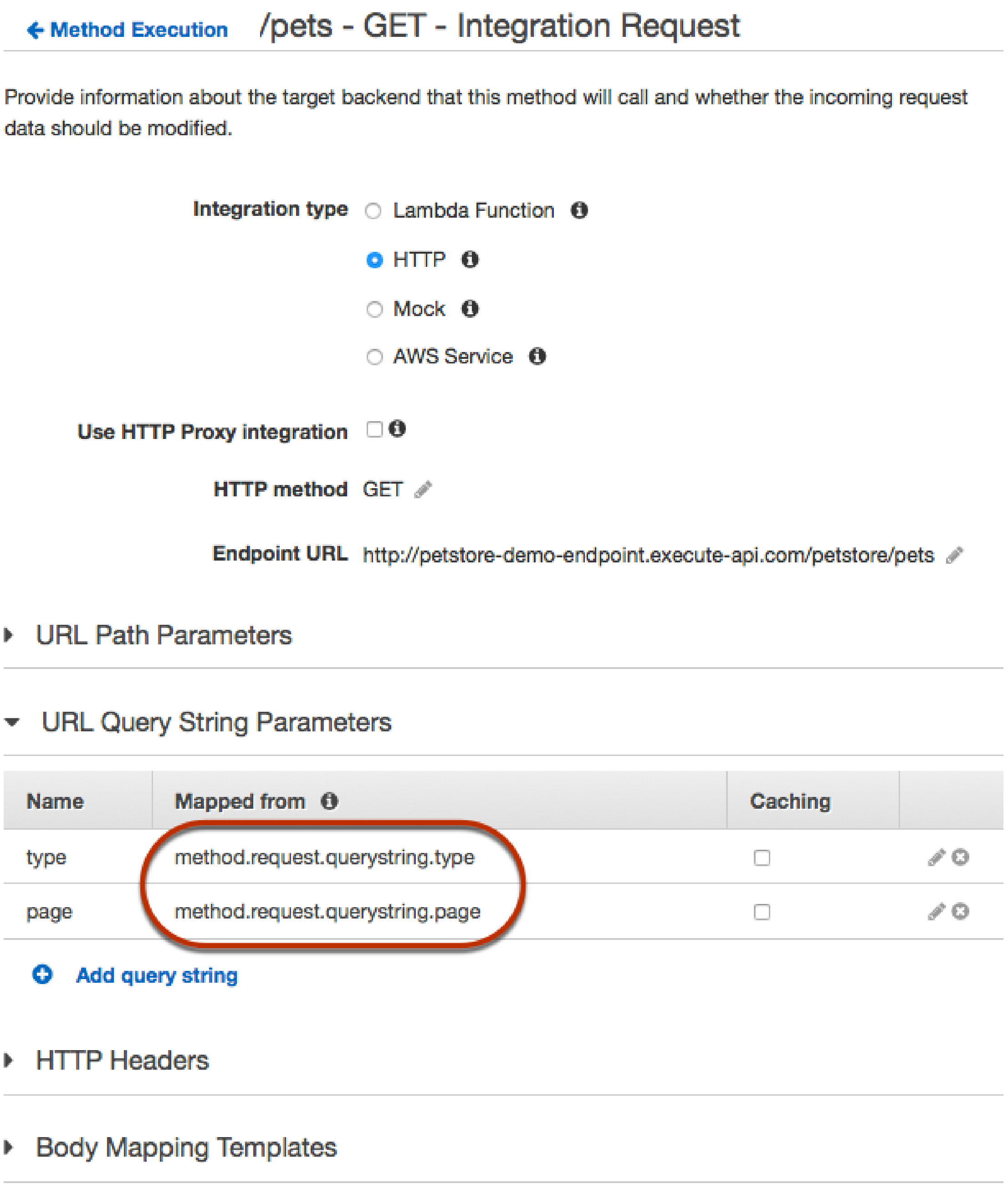
API Key Required false

URL Query String Parameters

Name	Caching	
type	☐	
page	☐	

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 backend. 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.



/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 ☐

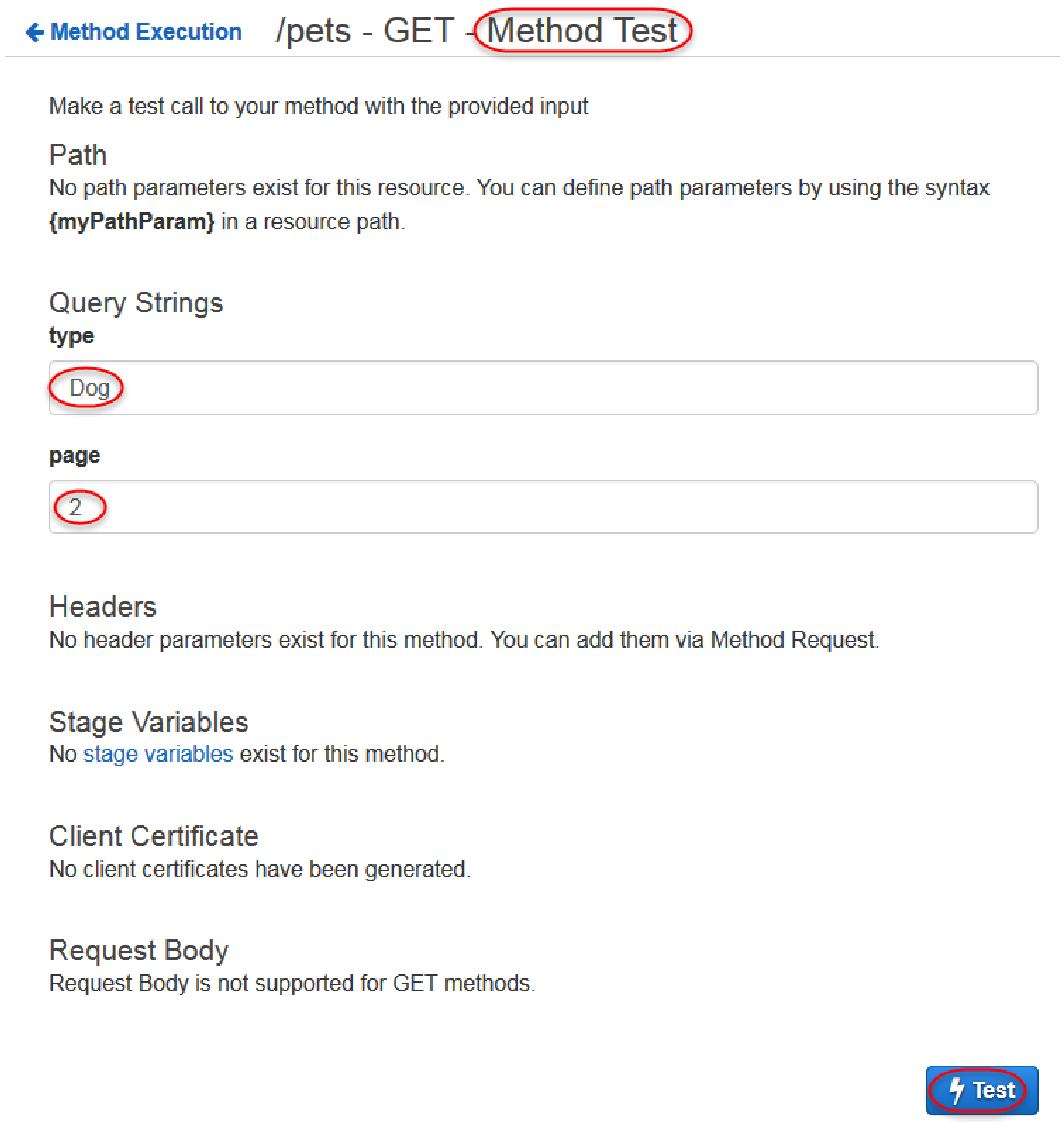
HTTP method GET

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

Name	Mapped from	Caching
type	method.request.querystring.type	☐
page	method.request.querystring.page	☐

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** on the **Method Execution** pane for the `GET /pets` method. 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
Dog

page
2

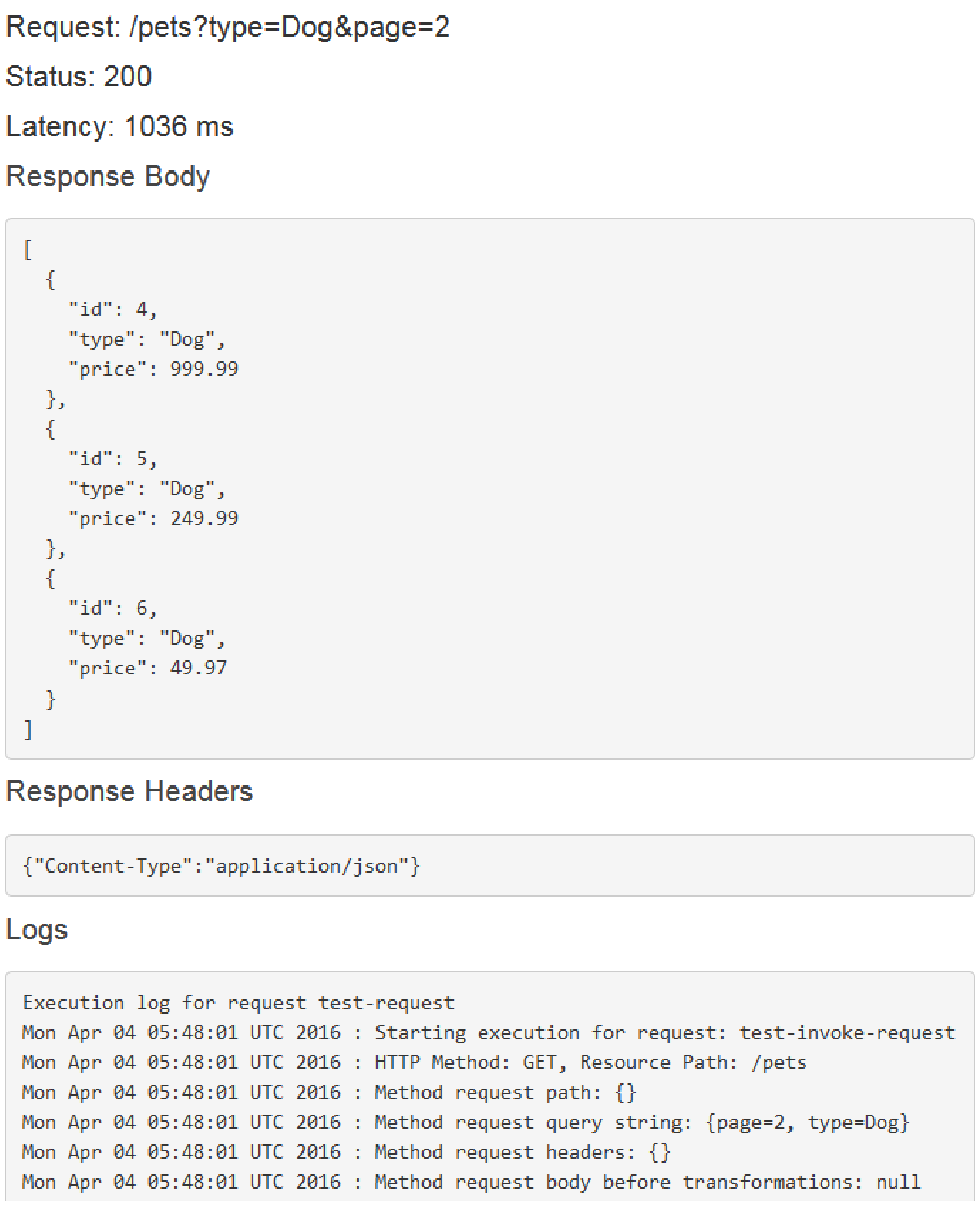
Headers
No header parameters exist for this method. You can add them via Method Request.

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

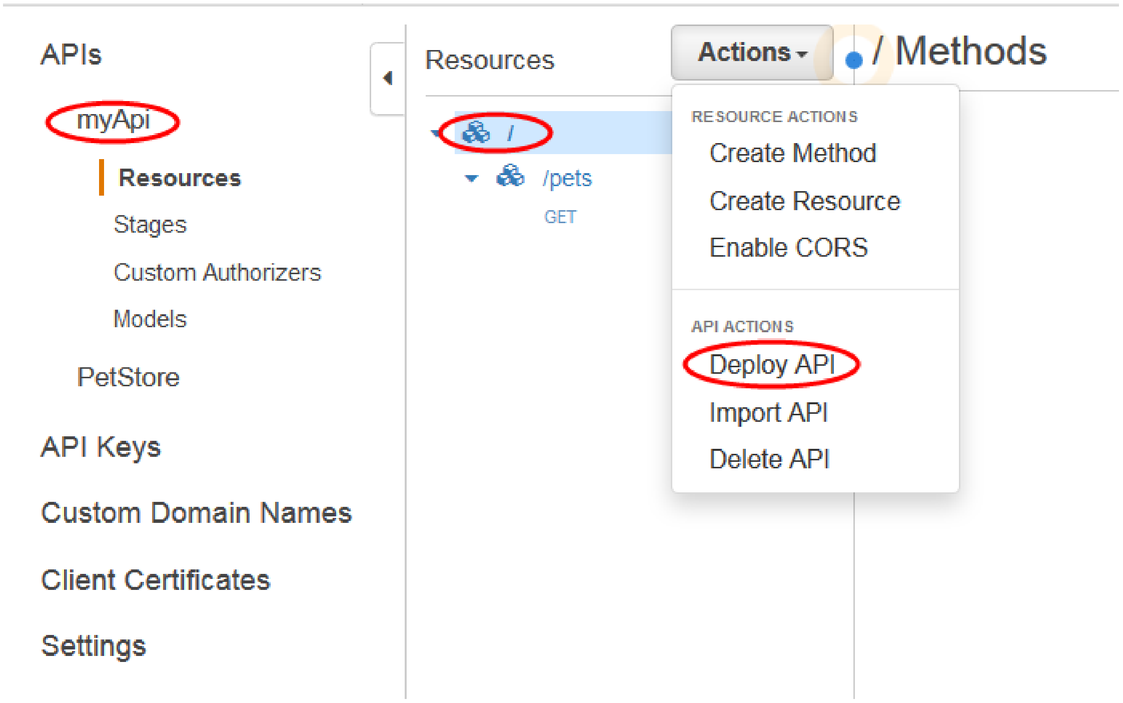
```
{"Content-Type":"application/json"}
```

Logs

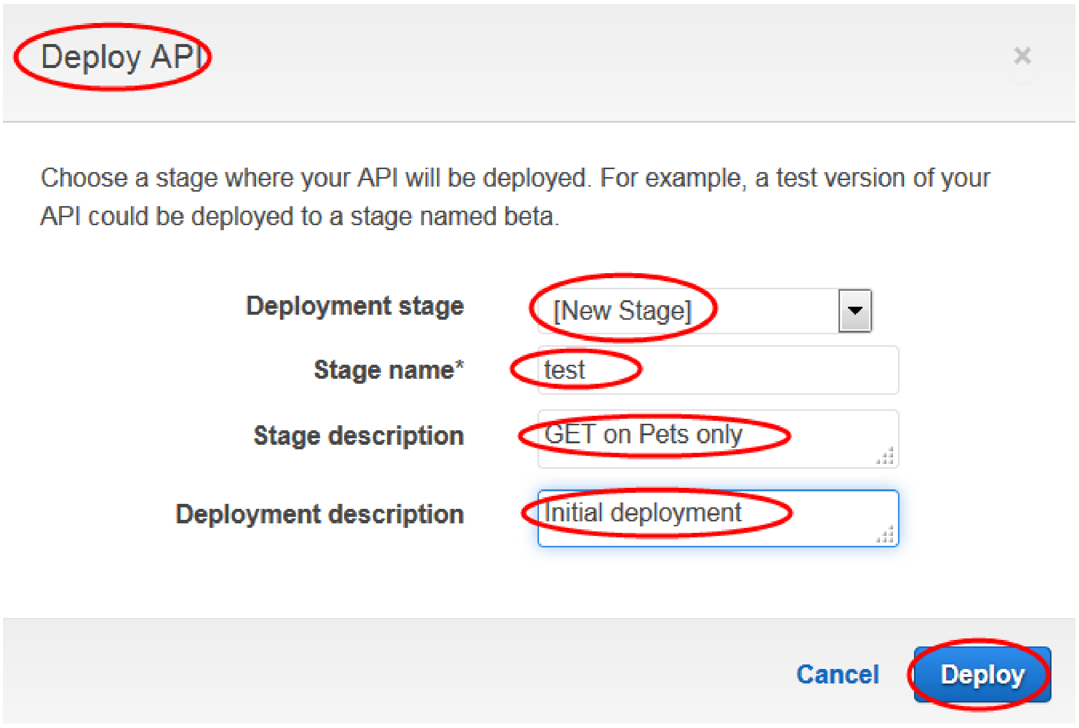
```
Execution log for request test-request
Mon Apr 04 05:48:01 UTC 2016 : Starting execution for request: test-invoke-request
Mon Apr 04 05:48:01 UTC 2016 : HTTP Method: GET, Resource Path: /pets
Mon Apr 04 05:48:01 UTC 2016 : Method request path: {}
Mon Apr 04 05:48:01 UTC 2016 : Method request query 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

Once deployed, you can obtain the invocation URLs (**Invoke URL**) of the API's endpoints.

If the GET method supported anonymous access, (for example, if the method's authorization type were set to `NONE`) you could double-click the **Invoke URL** link to invoke the method in your default browser. If needed, you could also append necessary query string parameters to the invocation URL. With the `AWS_IAM` authorization type described here, you must sign the request with an access key ID and the corresponding secret key of an IAM user of your AWS account. To do this, you must use a client that supports the Signature Version 4 (SigV4) protocols. An example of such a client is an app that uses one of the AWS SDKs or the Postman Chrome browser extension. To call a POST, PUT, or PATCH method, you also need to use such a client to handle the payload.

To invoke this API method in the Postman, 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 **Type** option under the **Authorization** tab, and then specify the following required properties before sending the request:

- For **AccessKey**, type the caller's AWS access key, as provisioned from AWS IAM.
- For **SecretKey**, type the caller's AWS secret key, as provisioned from AWS IAM when the access key was first created.
- For **AWS Region**, type the API-hosting AWS Region, as specified in the invocation URL.
- For **Service Name**, type `execute-api`, for the API Gateway execution service.

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 backend website, pass the payload without modifications through the integration request to the backend HTTP endpoint.

Topics

Note
Pay attention to the casing used in the steps of this walkthrough. Typing a lowercase letter instead of an uppercase letter (or vice versa) can cause errors later in the walkthrough.

Prerequisites

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

1. Complete the steps in Get Ready to Use API Gateway (p. 6), 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. 8) 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`**.
3. For **Resource Path**, accept the default of **/petstorewalkthrough**, and then choose **Create Resource**.

To create the second resource

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

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 and Test the Methods

In this step, you will integrate the methods with the backend 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 check mark 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 check mark 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 check mark 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 check mark 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. Choose **Add path**.
7. For **Name**, type **`id`**.
8. 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`.
9. Choose the check mark icon to finish creating the URL path parameter.
10. Choose **Method Execution**, and in the **Client** box, choose **TEST**. In the **Path** area, for **petId**, type **`1`**.
11. 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 3: 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 4: 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. 33).

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. 45) and the Map Request Parameters (p. 25). 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 Create Models and Mapping Templates for Request and Response Payloads (p. 119).

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

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

1. Complete the steps in Get Ready to Use API Gateway (p. 6), 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. 8).
3. Create two resources: `petstorewalkthrough` and `pets`. For more information, see Create Resources (p. 27) in the Map Request Parameters (p. 25).
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. 46) in the Build an API to Expose a Lambda Function (p. 45).

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/lambdaflattensome**, 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. 38) 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, callback) {
  callback(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 input to the Lambda function. `context` contains calling context. `callback` returns the result (for node.js 4.3 and later). 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. 45).
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/lambdaflattensome** node.
2. In **Method Execution**, choose **Method Response**. And then choose the arrow next to **200** to expand the section.
3. 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.
4. Go back to **Method Execution**. Choose **Integration Response**, and then choose the arrow next to **200**.
5. In the **Body Mapping Templates** section, choose **application/json** under **Content-Type**.
6. For **Generate template**, choose **PetsModelFlattenSome** to display the mapping script template for the output of this method.
7. 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
]
```

8. 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 `lambdaflattensome`.
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.

Warning
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 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. 66).

Build an API to Expose a Lambda Function

AWS Lambda provides an easy way to build backends 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 for Lambda Functions (p. 376).

Note
To create a Lambda function with the Lambda Proxy Integration type, follow the instructions given in Create an API with Lambda Proxy Integration through a Proxy Resource (p. 58) and make sure that the Lambda function returns the output according to the required output format (p. 119).

Topics

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. 6). 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. 16) .

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. 16).

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 for an API Gateway API, see Use IAM Permissions (p. 214). For more information about Lambda see the 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 `nodejs`. 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.

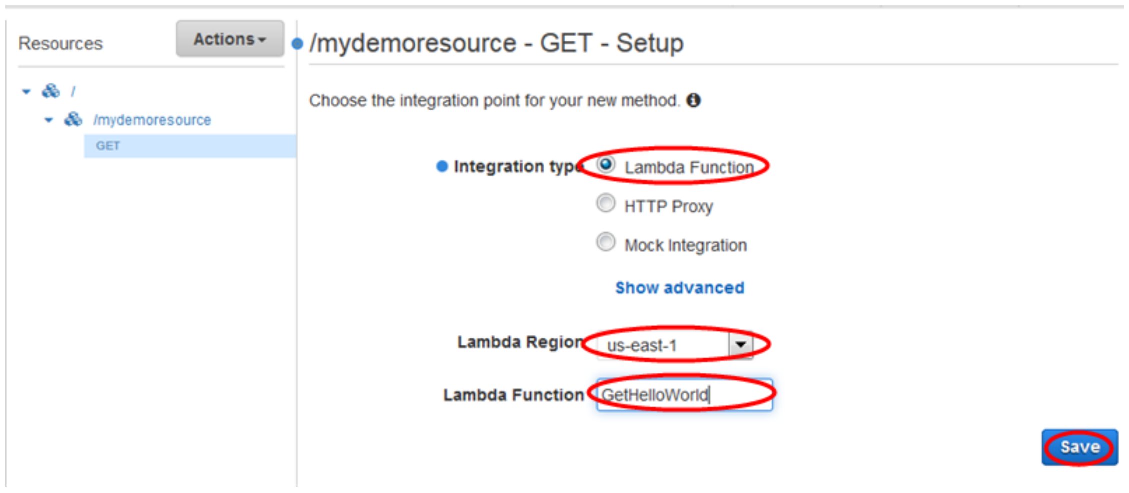
Step 5: Create and Test a GET Method

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*.



Choose the integration point for your new method.

Integration type: [✓ Lambda Function] HTTP Proxy Mock Integration

Lambda Region: us-east-1

Lambda Function: GetHelloWorld

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 request payload by default and then return the response payload 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 backend:

```
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: RequestReponse` 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 for Lambda Functions (p. 376).

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.

Warning
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 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.

Build an API Gateway API as an HTTP or Lambda Proxy

As an API developer, you can configure a proxy resource in an API Gateway API to enable a client to access an HTTP backend 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 Set up Proxy Integration (p. 111).

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. 6).

Topics

Create and Test an API with HTTP Proxy Integration

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

The following procedure walks you through the steps to create and test an API with a proxy resource for an HTTP backend using the API Gateway console. The HTTP backend is the `PetStore` website (`http://petstore-demo-endpoint.execute-api.com/`) from Build an API Gateway API to Expose an HTTP Endpoint (p. 8).

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

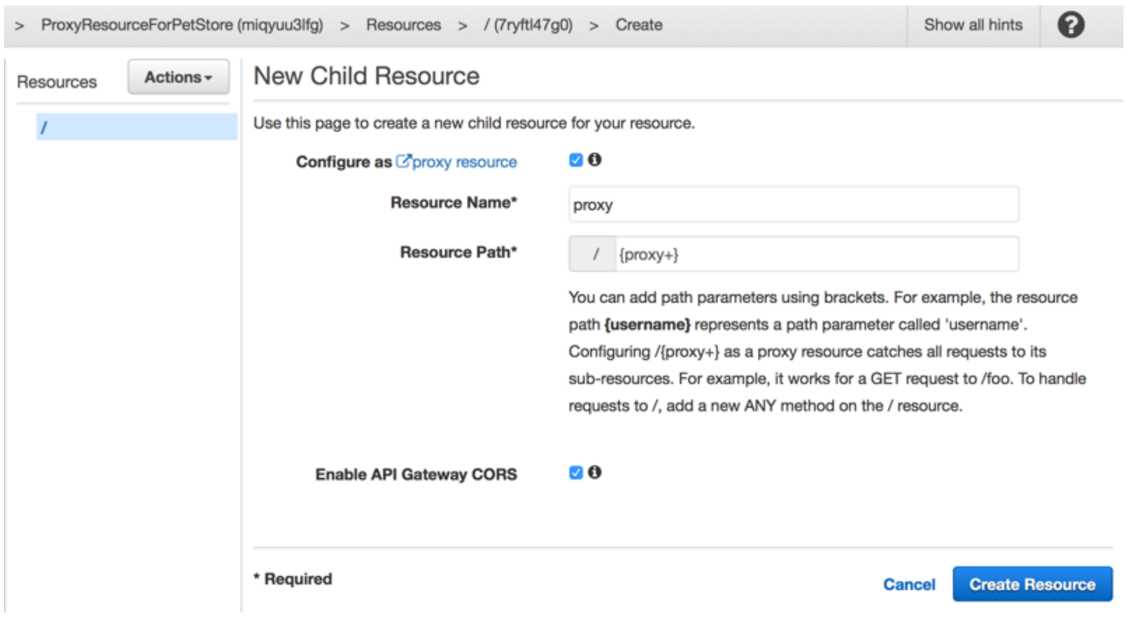
1. Sign in to the API Gateway console at https://console.aws.amazon.com/apigateway.
2. To create an API, choose **Create new API** (for creating the first API) or **Create API** (for creating any subsequent API). Next, do the following:

   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 ✓

Resource Name* proxy

Resource Path* / {proxy+}

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 its sub-resources. For example, it works for a GET request to /foo. To handle requests to /, add a new ANY method on the / resource.

Enable API Gateway CORS ✓

* Required

Cancel Create Resource

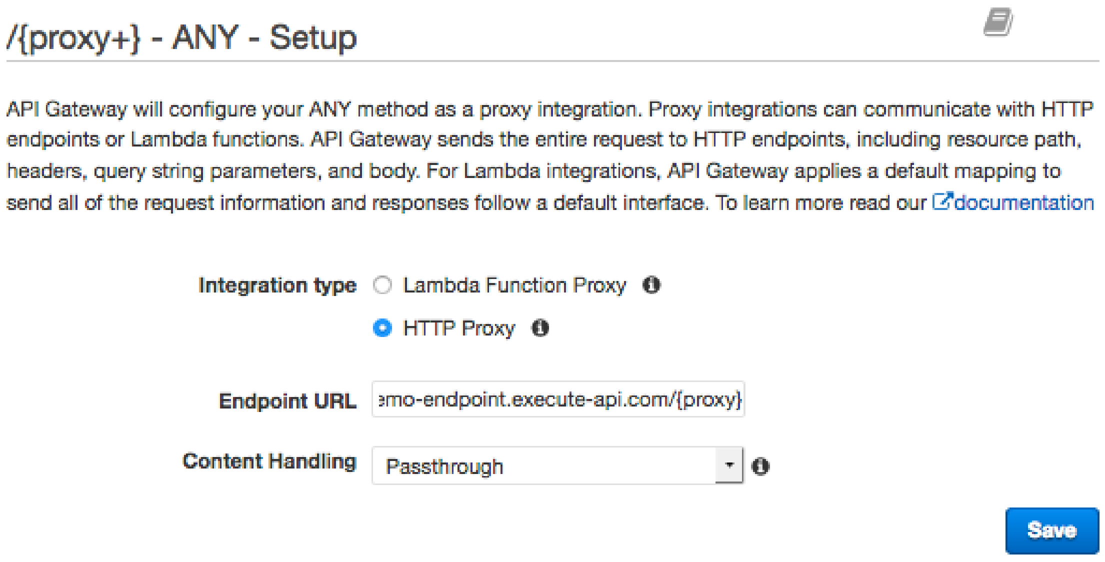
4. To set up the `ANY` method for integration with the HTTP back end, do the following:

   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, headers, query string parameters, and body. For Lambda integrations, API Gateway applies a default mapping to send all of the request information and responses follow a default interface. To learn more read our documentation

Integration type ○ Lambda Function Proxy [✓ HTTP Proxy]

Endpoint URL ɘmo-endpoint.execute-api.com/{proxy}

Content Handling Passthrough

Save

Test an API with HTTP Proxy Integration through Proxy Resource

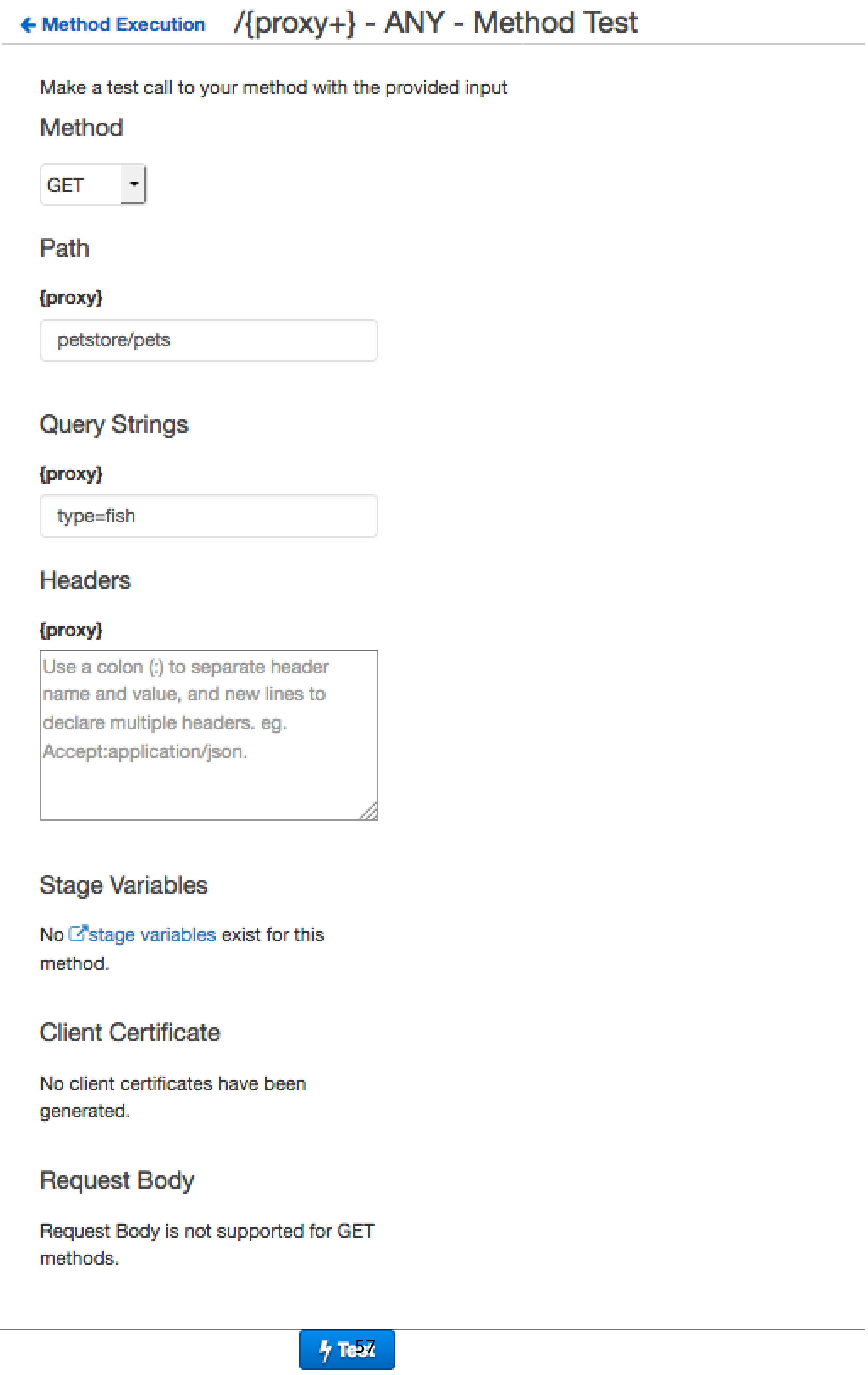
In the API we just created, the API's proxy resource path of `{proxy+}` becomes the placeholder of any of the backend endpoints under `http://petstore-demo-endpoint.execute-api.com/`. For example, it can be `petstore`, `petstore/pets`, and `petstore/pets/{petId}`. The `ANY` method serves as a placeholder 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.



/{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

{proxy}

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

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.

Because the backend 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 backend 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 backend 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

A Lambda Function in Node.js 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. 117) and Output Format of a Lambda Function for Proxy Integration (p. 119).

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

exports.handler = function(event, context, callback) {
    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))
    callback(null, response);
};
```

When used with a proxy resource in API Gateway, the input parameter of `event` contains an API request marshalled as a JSON object by API Gateway. This input can include the request's HTTP method (`httpMethod`), path (`path` and `pathParameters`), query parameters (`queryStringParameters`), headers (`headers`), and applicable payload (`body`) as well as the context (`requestContext`) and stage variables (`stageVariables`).

This example Lambda function parses the `event` parameter to retrieve the query string parameters of `name` and `httpStatus`.

The function then returns a greeting to the named user in the `message` property of the `responseBody` object. To show the details of the incoming request as marshalled by API Gateway, the function also returns the incoming `event` object as the `input` property of the response body.

Finally, upon exiting, the function returns a JSON object, containing the required `statusCode` and any applicable `headers` and `body`, for API Gateway to return it as an HTTP response to the client.

A Lambda Function in Java for Proxy Integration

The following Lambda function in Java is a "Hello, World!" application, similar to its Node.js counterpart (p. 59). The function shows how to parse the input event that is passed through as an InputStream object and 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. It also shows how to parse the `context` object to get the `LambdaLogger`. The example also demonstrates how to format the output of the Lambda function for API Gateway in Java to return the results in an OutputStream object as an HTTP response. For more information about the Lambda proxy integration input and output formats, see Input Format of a Lambda Function for Proxy Integration (p. 117) and Output Format of a Lambda Function for Proxy Integration (p. 119).

```
package examples;

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.io.BufferedReader;
import java.io.Writer;

import com.amazonaws.services.lambda.runtime.RequestStreamHandler;
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.LambdaLogger;


import org.json.simple.JSONObject;
import org.json.simple.JSONArray;
import org.json.simple.parser.ParseException;
import org.json.simple.parser.JSONParser;



public class ProxyWithStream implements RequestStreamHandler {
    JSONParser parser = new JSONParser();


    public void handleRequest(InputStream inputStream, OutputStream outputStream, Context
 context) throws IOException {

        LambdaLogger logger = context.getLogger();
        logger.log("Loading Java Lambda handler of ProxyWithStream");


        BufferedReader reader = new BufferedReader(new InputStreamReader(inputStream));
        JSONObject responseJson = new JSONObject();
        String name = "World";
        String responseCode = "200";

        try {
            JSONObject event = (JSONObject)parser.parse(reader);
            if (event.get("queryStringParameters") != null) {
                JSONObject qps = (JSONObject)event.get("queryStringParameters");
                if ( qps.get("name") != null) {
                    name = (String)qps.get("name");
                }
                if (qps.get("httpStatus") != null) {
                    responseCode = qps.get("httpStatus").toString();
                }
            }


            JSONObject responseBody = new JSONObject();
            responseBody.put("input", event.toJSONString());
            responseBody.put("message", "Hello " + name + "!");

            JSONObject headerJson = new JSONObject();
            headerJson.put("x-custom-response-header", "my custom response header value");

            responseJson.put("statusCode", responseCode);
            responseJson.put("headers", headerJson);
            responseJson.put("body", responseBody.toString());

        } catch(ParseException pex) {
            responseJson.put("statusCode", "400");
            responseJson.put("exception", pex);
        }

        logger.log(responseJson.toJSONString());
        OutputStreamWriter writer = new OutputStreamWriter(outputStream, "UTF-8");
        writer.write(responseJson.toJSONString());
        writer.close();
    }
}
```

When used with a proxy resource in API Gateway, the input stream contains an API request serialized as a JSON string by API Gateway. The input data can include the request's HTTP method (`httpMethod`), path (`path` and `pathParameters`), query parameters (`queryStringParameters`), headers (`headers`), applicable payload (`body`), the context (`requestContext`), and stage variables (`stageVariables`).

This example Lambda function parses the `inputStream` parameter to retrieve the query string parameters of `name` and `httpStatus`. For logging, it retrieves the `LambdaLogger` object from the incoming `context` object.

The function then returns a greeting to the named user in the `message` property of the `responseBody` object. To show the details of the incoming request as marshalled by API Gateway, the function also returns the input data (`event`) in the response body.

Finally, upon exiting, the function returns a JSON string, containing the required `statusCode` and any applicable `headers` and `body`, for API Gateway to return it as an HTTP response to the client.

To create this function in the Lambda console, you must create a deployment package before uploading the package into Lambda. To create a deployment package, follow these instructions.

Create a Backend for Lambda Proxy Integration

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

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

1. Sign in to the Lambda console at https://console.aws.amazon.com/lambda.
2. From the upper-right corner, choose an available region for the Lambda function.
3. From the main navigation pane, choose **Functions**.
4. Choose one of the following procedures to create a Lambda function in Node.js or Java. You can adapt the instructions for other languages and platforms.

    a. To create the Node.js Lambda function, choose **Create a Lambda function** or **Get Started Now** to create your first Lambda function in a region, and then do the following.

        i. On the **Select blueprint** page, choose **Skip**.

        ii. On the **Configure triggers** page, choose **Next**.

        iii. 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**.

        iv. 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.

        v. 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.

        vi. Keep the default values for **Advanced settings**.

        vii. Choose **Next**.

        viii. In the **Review** pane, choose **Create function**.

    For this tutorial, use LambdaForSimpleProxy (p. 59) as the function name and choose the standard **Simple Microservice permissions** policy templates to create the required Lambda execution role.

    b. To create a Java Lambda function, choose **Create a Lambda function** or **Get Started Now** to create your first Lambda function in a region, and then do the following.

        i. On the **Select blueprint** page, choose **Skip**.

        ii. On the **Configure triggers** page, choose **Next**.

        iii. 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 **Java 8**.

        iv. Under **Lambda function code**, do the following:

            - Choose a **Code entry type** for uploading a deployment package from a local drive or Amazon S3.

            - Choose **Upload** next to **Function package**.

- Navigate to the deployment package location and select the package.

v. Under **Lambda function handler and role**, do the following:

   - Type your function handler of the *package_name*.*function_name*::*method_name* format in **Handler**.

   - Choose an existing or create a new IAM role for the function execution.

vi. Keep the default values for **Advanced settings**.

vii. Choose **Next**.

viii. In the **Review** pane, choose **Create function**.

For this tutorial, use `examples` as the Java package name, use ProxyWithStreams (p. 60) as the Java class name, use `handleRequest` as the Java function name, and use `examples.ProxyWithStreams::handleRequest` as the Lambda function handler name. Choose the standard **Simple Microservice permissions** policy templates to create the required Lambda execution role.

Note
Note the region where you created the Lambda function. You will need it 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 backend 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 backend 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": "nBsWBOrSHMgnaROZJK1wGCZ9PcRcSpq_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"}
    }
}
```

Note
The `requestContext` is a map of key-value pairs and corresponds to the $context (p. 153) variable. Its outcome is API-dependent. API Gateway may add new keys to the map. For more information, see Input Format of a Lambda Function for Proxy Integration (p. 117).

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, 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, Kinesis, enabling your client to access the backend 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 Samples and Tutorials (p. 376).

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. 45), 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

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

1. Complete the steps in Get Ready to Use API Gateway (p. 6).

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. 8).
4. Deploy the API at least once to a stage named `test`. For more information, see Deploy the API (p. 51) in the Build an API to Expose a Lambda Function (p. 45).
5. Complete the rest of the steps in the Build an API to Expose a Lambda Function (p. 45).
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 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.

Warning
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 IAM console at https://console.aws.amazon.com/iam/.
2. In the **Details** area, click **Roles**.
3. Select **APIGatewayAWSProxyExecRole**, and then choose **Role Actions**, **Delete Role**. When prompted, choose **Yes, Delete**.
4. In the **Details** area, choose **Policies**.
5. Select **APIGatewayAWSProxyExecPolicy**, 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. 395), Create an API for Lambda Functions (p. 376) or Create an API as an Kinesis Proxy (p. 419).

Creating an API in Amazon API Gateway

In general, creating an API in API Gateway involves the following steps:

1. Create a `RestApi` instance.
2. Append any child `Resource`, if needed, to the root resource (`/`),
3. Add a `Method` instance to represent a client-facing method on the `Resource`.
4. Set up the `Integration` to forward the method request to the integrated backend and to route the response from the backend to the method.

To help your customers understand your API, you can provide documentation for the API, as part of the API creation. To enable this, add a `DocumentationPart` resource for a supported API entity.

To control how clients call an API, you can require AWS credentials, using a custom authorizer or Amazon Cognito user pool. To meter the use of your API, set up `usage plans` to throttle API requests.

Topics

Create a Simple API in API Gateway

For illustration purposes, we will use a simplified PetStore API that exposes only the `GET /` method. This method is integrated with the endpoint of `http://petstore-demo-endpoint.execute-api.com/`

`petstore/pets` and handles `200 OK` responses. The discussions focus on the essential programming tasks for creating an API in API Gateway, taking advantage of default settings. More complex details of API creation are discussed elsewhere in this chapter.

Topics

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. 8).

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. 9).

Alternatively, you can create an API by using the API Gateway Import API (p. 210) feature to upload an external API definition, such as one expressed in the Swagger 2.0 with the API Gateway Extensions to Swagger (p. 358). The example provided in Create an API Gateway API from an Example (p. 9) uses the Import API feature.

Create an API Using the AWS SDK for API Gateway

For more information using an AWS SDK, see AWS SDKs.

Create an API with the API Gateway REST API

To create an API using the API Gateway REST API, you work with the `RestApi`, `Resource`, `Method`, `MethodResponse`, `Integration`, and `IntegrationResponse` entities. The following procedures show how to work with these API entities to create the simple PetStore API.

To create the simple PetStore API

1. To create the `RestApi` for the API, call the `restapi:create` action, as shown in the following:

```
POST /restapis HTTP/1.1
Host: apigateway.us-west-2.amazonaws.com
Content-Type: application/x-amz-json-1.0
X-Amz-Date: 20170511T214723Z
Authorization: AWS4-HMAC-SHA256 Credential={ACCESS-KEY-ID}/20170511/us-west-2/
apigateway/aws4_request, SignedHeaders=content-length;content-type;host;x-amz-date,
 Signature=d0abd98a2a06199531c2916b162ede9f63a247032cdc8e4d077216446d13103c
Cache-Control: no-cache
Postman-Token: 0889d2b5-e507-6aab-f222-ab9548dbacaa

{
    "name": "Simple PetStore (REST API)",
    "description": "A sample API Gateway API created using the REST API."
}
```

   If successful, a `201 Created` response is returned with a payload similar to the following:

```
{
  "createdDate": "2017-05-11T21:47:24Z",
  "description": "A sample API Gateway API created using the REST API.",
  "id": "x7hyqq0ik7",
  "name": "Simple PetStore (REST API)"
}
```

Note the `id` value of the newly created `RestApi`. You use this value in subsequent operations on this API. A newly created `RestApi` comes with the root resource (`/`) of the API. You need to specify the `id` value of this root resource to append a child resource, and to add a method on the root resource or on a child resource. To get this identifier, follow the next step.

2. To get the root resource identifier, call restapi:resources action, as shown in the following:

```
GET /restapis/x7hyqq0ik7/resources HTTP/1.1
Host: apigateway.us-west-2.amazonaws.com
Content-Type: application/x-amz-json-1.0
X-Amz-Date: 20170511T215738Z
Authorization: AWS4-HMAC-SHA256 Credential={ACCESS-KEY-ID}/20170511/us-
west-2/apigateway/aws4_request, SignedHeaders=content-type;host;x-amz-date,
 Signature=76c24ef91d835b85313142bf75545c4ac4c212067e8188ee6a127c21dae09e29
Cache-Control: no-cache
Postman-Token: ed8fb586-8d67-8db7-d229-5a88b4620037
```

If the request is successful, a `200 OK` response is returned with a payload similar to the following:

```
{
  "_embedded": {
    "item": {
      "_links": {
        "self": {
          "href": "/restapis/x7hyqq0ik7/resources/0f72nvvnkd"
        },
        "method:by-http-method": {
          "href": "/restapis/x7hyqq0ik7/resources/0f72nvvnkd/methods/{http_method}",
          "templated": true
        },
        "method:put": {
          "href": "/restapis/x7hyqq0ik7/resources/0f72nvvnkd/methods/{http_method}",
          "templated": true
        },
        "resource:create-child": {
          "href": "/restapis/x7hyqq0ik7/resources/0f72nvvnkd"
        },
        "resource:update": {
          "href": "/restapis/x7hyqq0ik7/resources/0f72nvvnkd"
        }
      },
      "id": "0f72nvvnkd",
      "path": "/"
    }
  }
}
```

Note that the root resource is identified by the `/` path in the returned response payload.

3. To add a `GET` method to the root resource, call the `method:put` action, as shown in the following:

```
PUT /restapis/x7hyqq0ik7/resources/0f72nvvnkd/methods/GET HTTP/1.1
Host: apigateway.us-west-2.amazonaws.com
Content-Type: application/x-amz-json-1.0
```

```
X-Amz-Date: 20170512T000729Z
Authorization: AWS4-HMAC-SHA256 Credential={ACCESS-KEY-ID}/20170512/us-west-2/
apigateway/aws4_request, SignedHeaders=content-length;content-type;host;x-amz-date,
 Signature=8a7093411c97b0aa90f4b1890475d93cf20aa3732089da61e6deb410fbc6037d
Cache-Control: no-cache
Postman-Token: 48abcd2f-c357-9e44-669e-d8d813f876ca

{
    "authorizationType": "NONE"
}
```

Setting the `authorizationType` to `NONE` makes the API callable by anyone. This is consistent with the access permissions on the backend PetStore website. For other backends or for other application scenarios you may need to use a different authorization type.

The successful request returns a `201 Created` response with a payload similar to the following:

```
{
  "_links": {
    "curies": [
      ...
    ],
    "self": {
      "href": "/restapis/x7hyqq0ik7/resources/0f72nvvnkd/methods/GET",
      "name": "GET",
      "title": "GET"
    },
    "integration:put": {
      "href": "/restapis/x7hyqq0ik7/resources/0f72nvvnkd/methods/GET/integration"
    },
    "method:delete": {
      "href": "/restapis/x7hyqq0ik7/resources/0f72nvvnkd/methods/GET"
    },
    "method:update": {
      "href": "/restapis/x7hyqq0ik7/resources/0f72nvvnkd/methods/GET"
    },
    "methodresponse:put": {
      "href": "/restapis/x7hyqq0ik7/resources/0f72nvvnkd/methods/GET/responses/
{status_code}",
      "templated": true
    }
  },
  "apiKeyRequired": false,
  "authorizationType": "NONE",
  "httpMethod": "GET"
}
```

The `Method` resource that was just created represents the client-facing method request. Depending on API requirements, you can set up header, path, and query parameters on a method request. For `POST`, `PUT`, `PATCH`, and other methods taking payloads, you can define a model for the payload in the method request. For more information about these settings, see Set up Method and Integration (p. 81).

4. To set up the `200 OK` response for the `GET /` method, call the method:put action, as shown in the following:

```
PUT /restapis/x7hyqq0ik7/resources/0f72nvvnkd/methods/GET/responses/200 HTTP/1.1
Host: apigateway.us-west-2.amazonaws.com
Content-Type: application/x-amz-json-1.0
X-Amz-Date: 20170512T003943Z
```

```
Authorization: AWS4-HMAC-SHA256 Credential={ACCESS-KEY-ID}/20170512/us-west-2/
apigateway/aws4_request, SignedHeaders=content-length;content-type;host;x-amz-date,
 Signature=229ef4cfba4bbe41132f36c027f0ae4449bb741671875075a4b216e9b778233e
Cache-Control: no-cache
Postman-Token: 268fcf18-92e4-dfea-821a-ebf4e1d0edfd

{}
```

The successful response returns a `201 Created` status code and a payload similar to the following:

```
{
  "_links": {
    "curies": {
      "href": "http://docs.aws.amazon.com/apigateway/latest/developerguide/restapi-
method-response-{rel}.html",
      "name": "methodresponse",
      "templated": true
    },
    "self": {
      "href": "/restapis/x7hyqq0ik7/resources/0f72nvvnkd/methods/GET/responses/200",
      "title": "200"
    },
    "methodresponse:delete": {
      "href": "/restapis/x7hyqq0ik7/resources/0f72nvvnkd/methods/GET/responses/200"
    },
    "methodresponse:update": {
      "href": "/restapis/x7hyqq0ik7/resources/0f72nvvnkd/methods/GET/responses/200"
    }
  },
  "statusCode": "200"
}
```

The `Method` request and `MethodRepsonse` represent the client-facing interface of the `GET /` method of the API. For the interface to function properly, we set up its `Integration` and `IntegrationResponse` to forward the incoming request from the client to the backend, and to route the returning response from the backend to the client.

5. To set up the `Integration` request with the PetStore backend, call the integration:put action, as shown in the following:

```
PUT /restapis/x7hyqq0ik7/resources/0f72nvvnkd/methods/GET/integration HTTP/1.1
Host: apigateway.us-west-2.amazonaws.com
Content-Type: application/x-amz-json-1.0
X-Amz-Date: 20170512T002249Z
Authorization: AWS4-HMAC-SHA256 Credential={ACCESS-KEY-ID}/20170512/us-west-2/
apigateway/aws4_request, SignedHeaders=content-length;content-type;host;x-amz-date,
 Signature=10359971a8c54862a47e39d6a6e4b6e62c263e9a2b785b47b40c426c0aa61c19
Cache-Control: no-cache
Postman-Token: 7f7bbb22-afdb-09ed-e22a-8c13f8e91ee7

{
  "type" : "HTTP",
  "httpMethod" : "GET",
  "uri" : "http://petstore-demo-endpoint.execute-api.com/petstore/pets"
}
```

In the request payload, `uri` points to the backend endpoint associated with the method. The `type` refers to the integration type. For the specified HTTP `uri`, the value of `type` must be `HTTP`. The `httpMethod` property sets the HTTP verb as required by the backend, which may be different from the method request HTTP verb set in the `integration:put` action URL.

The successful response returns a `201 Created` status code with a payload similar to the following:

```
{
  "_links": {
    "curies": [
      ...
    ],
    "self": {
      "href": "/restapis/x7hyqq0ik7/resources/0f72nvvnkd/methods/GET/integration"
    },
    "integration:delete": {
      "href": "/restapis/x7hyqq0ik7/resources/0f72nvvnkd/methods/GET/integration"
    },
    "integration:update": {
      "href": "/restapis/x7hyqq0ik7/resources/0f72nvvnkd/methods/GET/integration"
    },
    "integrationresponse:put": {
      "href": "/restapis/x7hyqq0ik7/resources/0f72nvvnkd/methods/GET/integration/
responses/{status_code}",
      "templated": true
    }
  },
  "cacheKeyParameters": [],
  "cacheNamespace": "0f72nvvnkd",
  "httpMethod": "GET",
  "passthroughBehavior": "WHEN_NO_MATCH",
  "type": "HTTP",
  "uri": "http://petstore-demo-endpoint.execute-api.com/petstore/pets"
}
```

6. To set up the `IntegrationResponse`, call the integrationresponse:put action, as shown in the following:

```
PUT /restapis/x7hyqq0ik7/resources/0f72nvvnkd/methods/GET/integration/responses/200
 HTTP/1.1
Host: apigateway.us-west-2.amazonaws.com
Content-Type: application/x-amz-json-1.0
X-Amz-Date: 20170512T004542Z
Authorization: AWS4-HMAC-SHA256 Credential={ACCESS-KEY-ID}/20170512/us-west-2/
apigateway/aws4_request, SignedHeaders=content-length;content-type;host;x-amz-date,
 Signature=545ab6ea151f72c52161ee856ee621f136d717e40a02743cd8fe3638895794b1
Cache-Control: no-cache
Postman-Token: d29cabea-5232-7021-88ad-f1e950f55a99

{}
```

The successful response returns the `201 Created` status code with a payload similar to the following:

```
{
  "_links": {
    "curies": {
      "href": "http://docs.aws.amazon.com/apigateway/latest/developerguide/restapi-
integration-response-{rel}.html",
      "name": "integrationresponse",
      "templated": true
    },
    "self": {
      "href": "/restapis/x7hyqq0ik7/resources/0f72nvvnkd/methods/GET/integration/
responses/200",
      "title": "200"
    },
    "integrationresponse:delete": {
      "href": "/restapis/x7hyqq0ik7/resources/0f72nvvnkd/methods/GET/integration/
responses/200"
```

```
    },
    "integrationresponse:update": {
      "href": "/restapis/x7hyqq0ik7/resources/0f72nvvnkd/methods/GET/integration/
responses/200"
    }
  },
  "statusCode": "200"
}
```

We now have successfully created a simple PetStore API.

After deploying the API to, for example, a `test` stage (see deployment:create), you can test this API by typing the `https://x7hyqq0ik7.execute-api.us-west-2.amazonaws.com/test` URL in a browser. Substitute the `RestApi` identifier (`x7hyqq0ik7`) with the identifier of your API. The expected output should be as follows:

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

Create an API Using AWS CLI Commands

To create an API using the AWS CLI, you work with the `create-rest-api`, `create-resource` or `get-resources`, `put-method`, `put-method-response`, `put-integration`, and `put-integration-response` commands. The following procedures show how to work with these AWS CLI commands to create the simple PetStore API.

To create a simple PetStore API using AWS CLI

1. Call the `create-rest-api` command to create a `RestApi`.

```
aws apigateway create-rest-api --name 'Simple PetStore (AWS CLI)' --region us-west-2
```

The following is the output of this command:

```
{
    "name": "Simple PetStore (AWS CLI)",
    "id": "vaz7da96z6",
    "createdDate": 1494572809
}
```

2. Call the `get-resources` command to retrieve the root resource identifier of the `RestApi`.

```
aws apigateway get-resources --rest-api-id vaz7da96z6 --region us-west-2
```

The following is the output of this command:

```
{
    "items": [
        {
            "path": "/",
            "id": "begaltmsm8"
        }
    ]
}
```

3. Call the `put-method` command to create a `Method` request of `GET /` with open access.

```
aws apigateway put-method --rest-api-id vaz7da96z6 --resource-id begaltmsm8 --http-
method GET --authorization-type "NONE" --region us-west-2
```

The following is the output of this command:

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

4. Call the `put-method-response` command to create a `MethodResponse` of the `GET /` method.

```
aws apigateway put-method-response --rest-api-id vaz7da96z6 --resource-id begaltmsm8 --
http-method GET --status-code 200  --region us-west-2
```

The following is the output of this command:

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

5. Call the `put-integration` command to create an `Integration` with a specified HTTP endpoint for the `GET /` method.

```
aws apigateway put-integration --rest-api-id vaz7da96z6 --resource-id begaltmsm8 --
http-method GET --type HTTP --integration-http-method GET --uri 'http://petstore-demo-
endpoint.execute-api.com/petstore/pets'  --region us-west-2
```

The following is the output of this command:

```
{
    "httpMethod": "GET",
    "passthroughBehavior": "WHEN_NO_MATCH",
    "cacheKeyParameters": [],
    "type": "HTTP",
    "uri": "http://petstore-demo-endpoint.execute-api.com/petstore/pets",
    "cacheNamespace": "begaltmsm8"
}
```

6. Call the `put-integration-response` command to create an `IntegrationResponse` of the `GET /` method integrated with an HTTP backend.

```
aws apigateway put-integration-response --rest-api-id vaz7da96z6 --resource-id
 begaltmsm8 --http-method GET --status-code 200 --selection-pattern ""  --region us-
west-2
```

The following is the output of this command:

```
{
    "selectionPattern": "",
    "statusCode": "200"
}
```

Deploy the API to a `stage` stage, for example, by calling `create-deployment`:

```
aws apigateway create-deployment --rest-api-id vaz7da96z6 --stage-name test --stage-
description 'Test stage' --description 'First deployment'
```

You can test this API by typing the `https://vaz7da96z6.execute-api.us-west-2.amazonaws.com/test` URL in a browser, and substituting `vaz7da96z6` with the identifier of your API. The expected output should be as follows:

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

Create an API by Importing Swagger Definitions

You can create an API in API Gateway by specifying Swagger definitions of appropriate API Gateway API entities and importing the Swagger definitions into API Gateway.

The following Swagger definitions describe the simple API, exposing only the `GET /` method integrated with an HTTP endpoint of the PetStore website in the backend, and returning a `200 OK` response.

```
{
  "swagger": "2.0",
  "info": {
    "title": "Simple PetStore (Swagger)"
  },
  "schemes": [
    "https"
  ],
  "paths": {
    "/": {
      "get": {
        "responses": {
          "200": {
            "description": "200 response"
          }
        },
        "x-amazon-apigateway-integration": {
          "responses": {
            "default": {
              "statusCode": "200"
            }
          },
          "uri": "http://petstore-demo-endpoint.execute-api.com/petstore/pets",
          "passthroughBehavior": "when_no_match",
          "httpMethod": "GET",
          "type": "http"
        }
      }
    }
  }
}
```

The following procedure describes how to import these Swagger definitions into API Gateway using the API Gateway console.

To import the simple API Swagger definitions using the API Gateway console

1. Sign in to the API Gateway console.
2. Choose **Create API**.
3. Choose **Import from Swagger**.
4. If you saved the preceding Swagger definitions in a file, choose **Select Swagger File**. You can also copy the Swagger definitions and paste them into the import text editor.
5. Choose **Import** to finish importing the Swagger definitions.

To import the Swagger definitions using the API Gateway REST API, call the restapi:import action, supplying the preceding Swagger definitions as the payload. For more information, see the example in the Remarks section of the `restapi:import` topic.

To import the Swagger definitions using the AWS CLI, save the Swagger definitions into a file and then run the following command, assuming that you use the `us-west-2` region and the absolute Swagger file path is `file:///path/to/API_Swagger_template.json`:

```
aws apigateway import-rest-api --body 'file:///path/to/API_Swagger_template.json' --region
 us-west-2
```

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. 8).
- 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. 46) of the Build an API to Expose a Lambda Function (p. 45).

- 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. 247)

Topics

Set up API Gateway Integration Request

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 backend. 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 backend: enforcing secure communications over HTTPS with the backend and translating data formats between the client and backend. 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. 108).
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. 46) section of the Build an API to Expose a Lambda Function (p. 45); 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 Set up Method Request (p. 84).
   - 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 Set up Data Mappings between Method and Integration (p. 86).

Set up API Gateway Method Request

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 Set up API Gateway Integration Request (p. 82).

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. 46) section of the Build an API to Expose a Lambda Function (p. 45).

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

> **Note**
> You can also enable an API key. For instructions, see Use an API Key when Usage Plans Are Not Enabled (p. 279).

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. 292).
   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. 126).

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.)

Set up Data Mappings between Method and Integration

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 Set up API Gateway Integration Request (p. 82).

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 backend 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 backend 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. 146).

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 Create Models and Mapping Templates for Request and Response Payloads (p. 119).
5. You can map an integration response from the backend 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 backend 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 `callback(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**.

Handle Lambda Errors in API Gateway

When your API is integrated with AWS Lambda with the regular Lambda integration, not the Lambda proxy integration, you must map errors returned by Lambda in the integration response to standard HTTP error responses for your clients. Otherwise, Lambda errors are returned as `200 OK` responses by default and the result is not intuitive for your API users.

There are two types of errors that Lambda can return: standard errors and custom errors. In your API, you must handle these differently.

With the Lambda proxy integration, Lambda is required to return an output of the following format:

```
{
  "isBase64Encoded" : "boolean",
  "statusCode": "number",
  "headers": { ... },
  "body": "JSON string"
}
```

In this output, `statusCode` is typically `4XX` for a client error and `5XX` for a server error. API Gateway handles these errors by mapping the Lambda error to an HTTP error response, according to the specified `statusCode`.

Topics

Handle Standard Lambda Errors in API Gateway

A standard AWS Lambda error has the following format:

```
{
  "errorMessage": "<replaceable>string</replaceable>",
  "errorType": "<replaceable>string</replaceable>",
  "stackTrace": [
    "<replaceable>string</replaceable>",
    ...
  ]
}
```

Here, `errorMessage` is a string expression of the error. The `errorType` is a language-dependent error or exception type. The `stackTrace` is a list of string expressions showing the stack trace leading to the occurrence of the error.

For example, consider the following JavaScript Lambda function (Node.js 4.3 and later).

```
exports.handler = function(event, context, callback) {
    callback(new Error("Malformed input ..."));
};
```

This function returns the following standard Lambda error, containing `Malformed input ...` as the error message:

```
{
  "errorMessage": "Malformed input ...",
  "errorType": "Error",
  "stackTrace": [
    "exports.handler (/var/task/index.js:3:14)"
  ]
}
```

Similarly, consider the following Python Lambda function, which raises an `Exception` with the same `Malformed input ...` error message.

```
def lambda_handler(event, context):
    raise Exception('Malformed input ...')
```

This function returns the following standard Lambda error:

```
{
  "stackTrace": [
    [
      "/var/task/lambda_function.py",
      3,
      "lambda_handler",
      "raise Exception('Malformed input ...')"
    ]
  ],
  "errorType": "Exception",
  "errorMessage": "Malformed input ..."
}
```

Note that the `errorType` and `stackTrace` property values are language-dependent. The standard error also applies to any error object that is an extension of the `Error` object or a subclass of the `Exception` class.

To map the standard Lambda error to a method response, you must first decide on an HTTP status code for a given Lambda error. You then set a regular expression pattern on the `selectionPattern` property of the IntegrationResponse associated with the given HTTP status code. In the API Gateway console, this `selectionPattern` is denoted as **Lambda Error Regex** in the **Integration Response** configuration editor.

For example, to set up a new `selectionPattern` expression, using AWS CLI, call the following put-integration-response command:

```
aws apigateway put-integration-response --rest-api-id z0vprf0mdh --resource-id x3o5ih --
http-method GET --status-code 400 --selection-pattern "Invalid*" --region us-west-2
```

Make sure that you also set up the corresponding error code (`400`) on the method request. Otherwise, API Gateway throws an invalid configuration error response at runtime.

At runtime, API Gateway matches the error message against the pattern of the regular expression on the `selectionPattern` property. When there is a match, API Gateway returns the Lambda error as an HTTP response of the corresponding HTTP status code. If there is no match, API Gateway returns the error as a default response or throws an invalid configuration exception if no default response is configured.

Note
Setting the `selectionPattern` value to `.*` for a given response amounts to resetting this response as the default response. This is because such a selection pattern will match all error messages, including null, i.e., any unspecified error message. The resulting mapping overrides the default mapping.

To update an existing `selectionPattern` value using the API Gateway REST API, call the integrationresponse:update operation to replace the `/selectionPattern` path value with the specified regex expression of the `Malformed*` pattern.

```
PATCH /restapis/z0vprf0mdh/resources/x3o5ih/methods/GET/integration/responses/400 HTTP/1.1
Host: apigateway.us-west-2.amazonaws.com
Content-Type: application/x-amz-json-1.0
X-Amz-Date: 20170513T044831Z
Authorization: AWS4-HMAC-SHA256 Credential={ACCESS-KEY-ID}/20170513/us-west-2/
apigateway/aws4_request, SignedHeaders=content-length;content-type;host;x-amz-date,
 Signature=4b4e4881a727bc999799be5110facfcf7ec81d025afc1a4386cf440b4efc80de
Cache-Control: no-cache
Postman-Token: 04794263-29f2-f336-fa16-5653a7f3281d

{
  "patchOperations" : [ {
    "op" : "replace",
    "path" : "/selectionPattern",
    "value" : "Malformed*"
   } ]
}
```

To set the `selectionPattern` expression using the API Gateway console, type the expression in the **Lambda Error Regex** text box when setting up or updating an integration response of a specified HTTP status code.

Handle Custom Lambda Errors in API Gateway

Instead of the standard error described in the preceding section, AWS Lambda allows you to return a custom error object as JSON string. The error can be any valid JSON object. For example, the following JavaScript Lambda function (Node.js 4.3 or later) returns a custom error:

```
exports.handler = (event, context, callback) => {
    ...
    // Error caught here:
    var myErrorObj = {
        errorType : "InternalServerError",
        httpStatus : 500,
        requestId : context.awsRequestId,
        trace : {
            "function": "abc()",
            "line": 123,
            "file": "abc.js"
        }
    }
    callback(JSON.stringify(myErrorObj));
};
```

You must turn the `myErrorObj` object into a JSON string before calling `callback` to exit the function. Otherwise, the `myErrorObj` is returned as a string of `"[object Object]"`. When a method of your API is integrated with the preceding Lambda function, API Gateway receives an integration response with the following payload:

```
{
    "errorMessage": "{\"errorType\":\"InternalServerError\",\"httpStatus\":500,\"requestId
\":\"e5849002-39a0-11e7-a419-5bb5807c9fb2\",\"trace\":{\"function\":\"abc()\",\"line\":123,
\"file\":\"abc.js\"}}"
}
```

As with any integration response, you can pass through this error response as-is to the method response. Or you can have a body-mapping template to transform the payload into a different format. For example, consider the following body-mapping template for a method response of `500` status code:

```
{
    errorMessage: $input.path('$.errorMessage');
}
```

This template translates the integration response body that contains the custom error JSON string to the following method response body. This method response body contains the custom error JSON object:

```
{
    "errorMessage" : {
        errorType : "InternalServerError",
        httpStatus : 500,
        requestId : context.awsRequestId,
        trace : {
            "function": "abc()",
            "line": 123,
            "file": "abc.js"
        }
    }
};
```

Depending on your API requirements, you may need to pass some or all of the custom error properties as method response header parameters. You can achieve this by applying the custom error mappings from the integration response body to the method response headers.

For example, the following Swagger extension defines a mapping from the `errorMessage.errorType`, `errorMessage.httpStatus`, `errorMessage.trace.function`, and `errorMessage.trace` properties to the `error_type`, `error_status`, `error_trace_function`, and `error_trace` headers, respectively.

```
"x-amazon-apigateway-integration": {
    "responses": {
        "default": {
          "statusCode": "200",
          "responseParameters": {
          "method.response.header.error_trace_function":
 "integration.response.body.errorMessage.trace.function",
            "method.response.header.error_status":
 "integration.response.body.errorMessage.httpStatus",
            "method.response.header.error_type":
 "integration.response.body.errorMessage.errorType",
            "method.response.header.error_trace":
 "integration.response.body.errorMessage.trace"
          },
          ...
        }
    }
}
```

At runtime, API Gateway deserializes the `integration.response.body` parameter when performing header mappings. However, this deserialization applies only to body-to-header mappings for Lambda custom error responses and does not apply to body-to-body mappings using `$input.body`. With these custom-error-body-to-header mappings, the client receives the following headers as part of the method response, provided that the `error_status`, `error_trace`, `error_trace_function`, and `error_type` headers are declared in the method request.

```
"error_status":"500",
"error_trace":"{\"function\":\"abc()\",\"line\":123,\"file\":\"abc.js\"}",
"error_trace_function":"abc()",
"error_type":"InternalServerError"
```

The `errorMessage.trace` property of the integration response body is a complex property. It is mapped to the `error_trace` header as a JSON string.

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. 146) 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

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	`binaryMediaTypes`	`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	`binaryMediaTypes`	`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

Integration response payload	Request `Accept` header	`binaryMediaTypes`	`contentHandling`	Method response payload
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
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 `<img>` 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. 146) 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. 146) on the API.

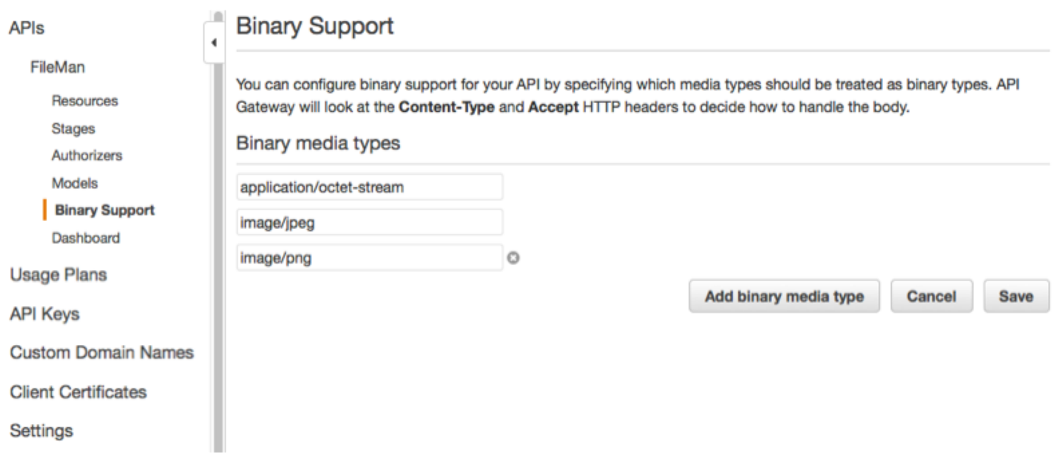
Enable Binary Support Using the API Gateway Console

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. 395).

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**.



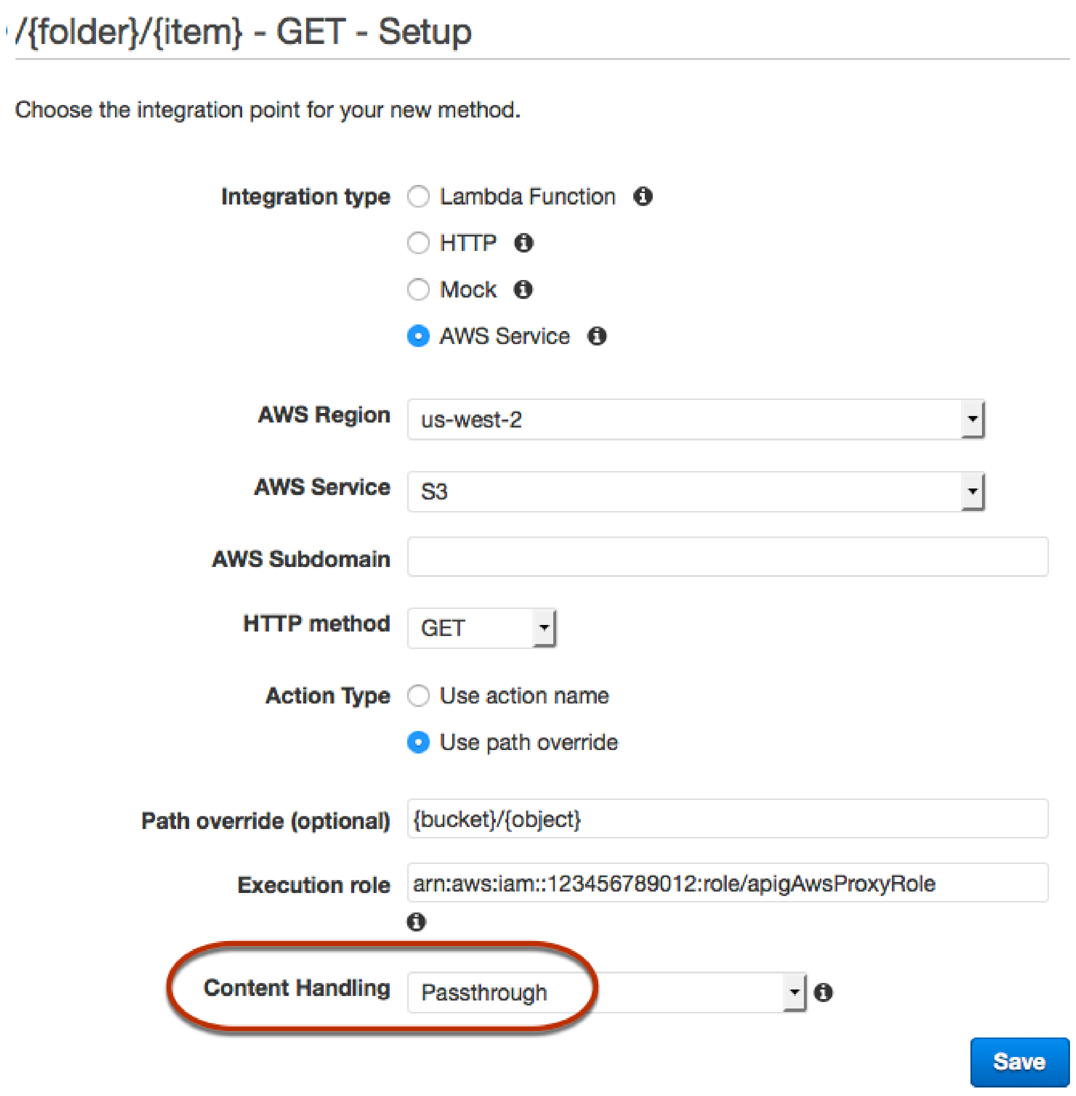
You can configure binary support for your API by specifying which media types should be treated as binary types. API Gateway will look at the **Content-Type** and **Accept** HTTP headers to decide how to handle the body.

Binary media types

application/octet-stream
image/jpeg
image/png

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.

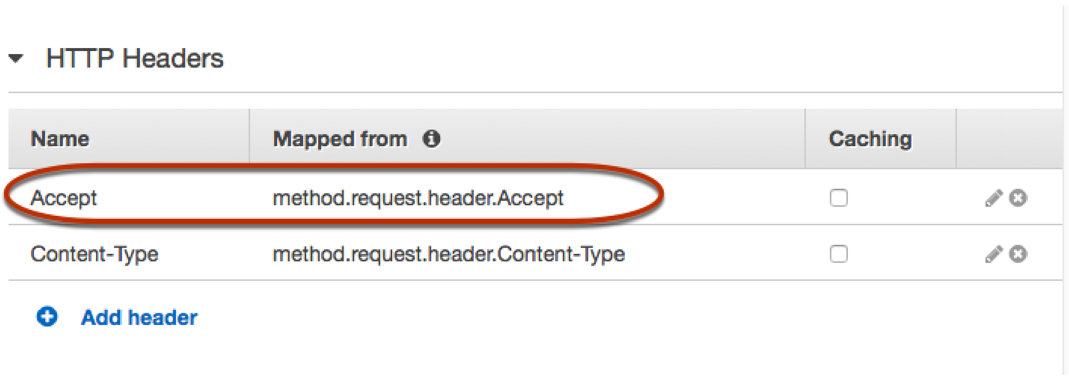

/{folder}/{item} - GET - Setup

Choose the integration point for your new method.

Integration type	Lambda Function / HTTP / Mock / [✓ AWS Service]
AWS Region	us-west-2
AWS Service	S3
AWS Subdomain	
HTTP method	GET
Action Type	Use action name / [✓ Use path override]
Path override (optional)	{bucket}/{object}
Execution role	arn:aws:iam::123456789012:role/apigAwsProxyRole
Content Handling	Passthrough

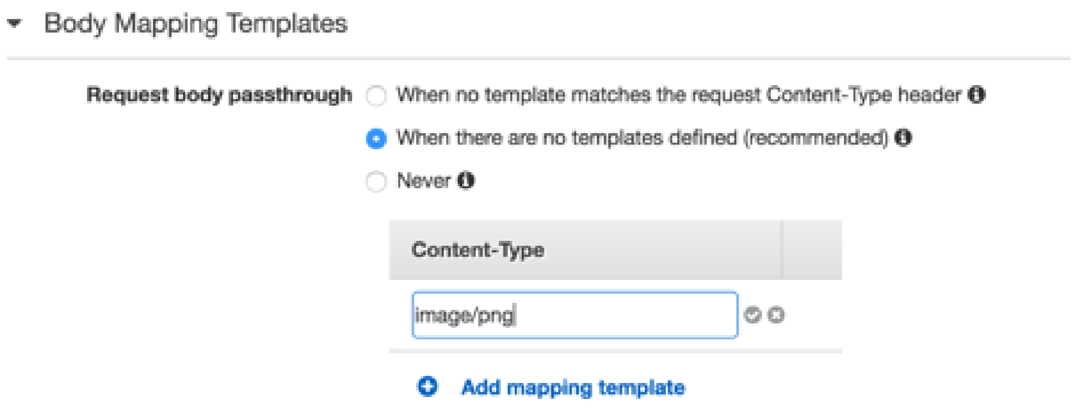
Choose **Passthrough** if you do not want to convert the body when the client and backend accepts the same binary format. Choose **Convert to text (if needed)** to convert the binary body to a Base64-encoded string when, for example, the backend requires that a binary request payload is passed in as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



Name	Mapped from	Caching
Accept	method.request.header.Accept	☐
Content-Type	method.request.header.Content-Type	☐

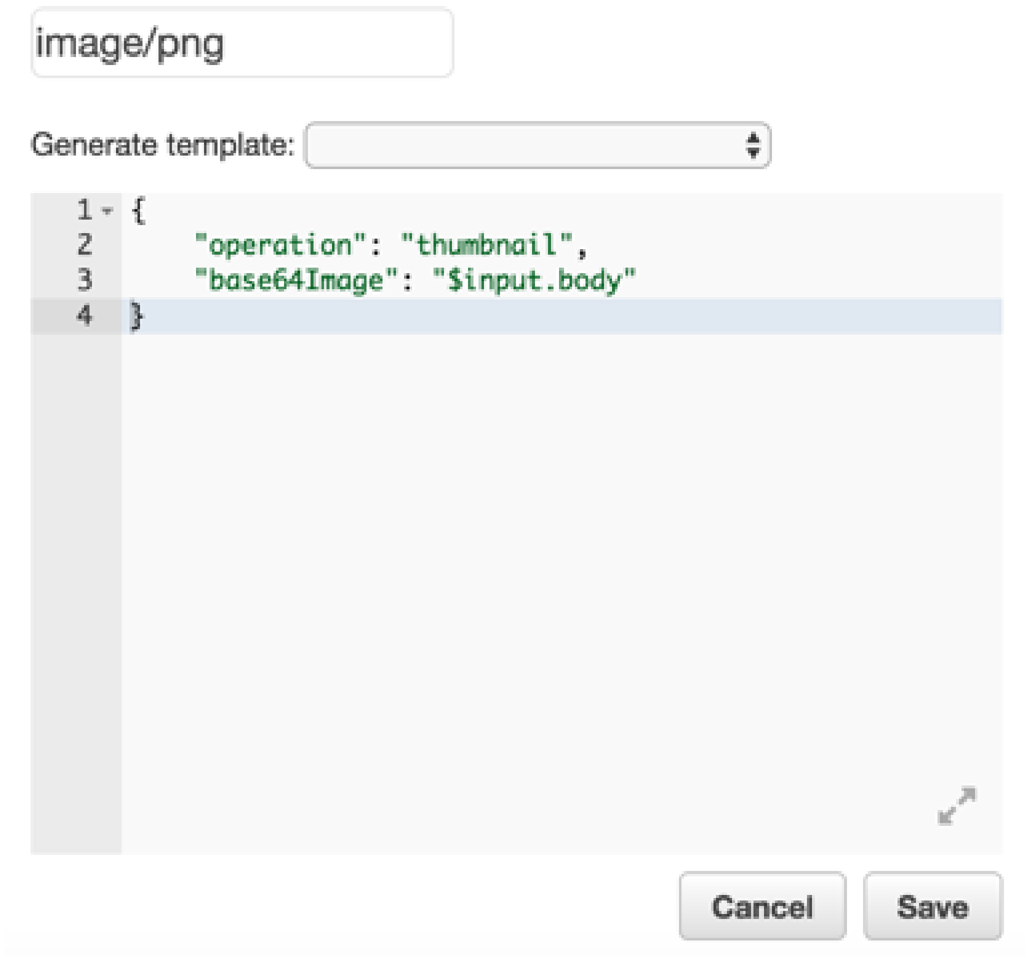
e. Enable the passthrough behavior on the request body.



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

Content-Type
image/png

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



```
{
     "operation": "thumbnail",
     "base64Image": "$input.body"
}
```

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

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 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-stream"
  }
 ]
}
```

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~1octet-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~1octet-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" : "CONTVERT_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~1octet-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" : "CONVERT_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. 362)

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. 366)
- x-amazon-apigateway-integration.response (p. 371)

Examples of Binary Support

The following example demonstrates how to access a binary file in Amazon S3 or 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

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

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 a 200 OK response with an empty payload.

```
{
  "swagger": "2.0",
  "info": {
    "version": "2016-10-21T17:26:28Z",
    "title": "ApiName"
  },
  "host": "abcdefghi.execute-api.us-east-1.amazonaws.com",
  "basePath": "/v1",
  "schemes": [
    "https"
  ],
  "paths": {
    "/s3": {
```

```
    "get": {
      "produces": [
        "application/json"
      ],
      "parameters": [
        {
          "name": "key",
          "in": "query",
          "required": false,
          "type": "string"
        }
      ],
      "responses": {
        "200": {
          "description": "200 response",
          "schema": {
            "$ref": "#/definitions/Empty"
          }
        },
        "500": {
          "description": "500 response"
        }
      },
      "x-amazon-apigateway-integration": {
        "credentials": "arn:aws:iam::123456789012:role/binarySupportRole",
        "responses": {
          "default": {
            "statusCode": "500"
          },
          "2\\d{2}": {
            "statusCode": "200"            }
        },
        "requestParameters": {
          "integration.request.path.key": "method.request.querystring.key"
        },
        "uri": "arn:aws:apigateway:us-west-2:s3:path/{key}",
        "passthroughBehavior": "when_no_match",
        "httpMethod": "GET",
        "type": "aws"
      }
    },
    "put": {
      "produces": [
        "application/json", "application/octet-stream"
      ],
      "parameters": [
        {
          "name": "key",
          "in": "query",
          "required": false,
          "type": "string"
        }
      ],
      "responses": {
        "200": {
          "description": "200 response",
          "schema": {
            "$ref": "#/definitions/Empty"
          }
        },
        "500": {
          "description": "500 response"
        }
      },
      "x-amazon-apigateway-integration": {
        "credentials": "arn:aws:iam::123456789012:role/binarySupportRole",
          "responses": {
            "default": {
              "statusCode": "500"
            },
            "2\\d{2}": {
              "statusCode": "200"
            }
          },
          "requestParameters": {
            "integration.request.path.key": "method.request.querystring.key"
          },
          "uri": "arn:aws:apigateway:us-west-2:s3:path/{key}",
          "passthroughBehavior": "when_no_match",
          "httpMethod": "PUT",
          "type": "aws",
          "contentHandling" : "CONVERT_TO_BINARY"
        }
      }
    }
  },
  "x-amazon-apigateway-binary-media-types" : ["application/octet-stream", "image/jpeg"],
  "definitions": {
    "Empty": {
      "type": "object",
      "title": "Empty Schema"
    }
  }
}
```

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.

Alternatively, to download an image file (`image.jpg`) as a Base64-encoded string, formatted as a JSON property, from Amazon S3, add a response template to the 200 integration response like this, as shown in the bold-faced Swagger definition block below:

```
        "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}"
              }
            }
          },
```

The request to download the image file looks like this:

```
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

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 backend. 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. 9).

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

- You must have the method available in API Gateway. Follow the instructions in Build an API Gateway API to Expose an HTTP Endpoint (p. 8).

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. 110).
    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. 111).
   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 Use the Console to Test a Method (p. 334).
    - 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 an API (p. 333).

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. 25).

```
{
  #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. 25).

```
## 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. 25).

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

Set up API Gateway Proxy Integration

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 backend 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, see Known Issues (p. 453).

Note
When using proxy integration with passthrough, API Gateway will return the default `Content-Type:application/json` header if the content type of a payload is unspecified.

Topics

API Gateway Proxy Resource

API Gateway defines a proxy resource as a place holder for a resource to be specified when the request is submitted. A API Gateway proxy resource has 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 patten 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`.

You can define an API method on a proxy resource and non-proxy resource at the same level of the resource hierarchy of an API. For example, you can expose the `GET` method on the `/{ggg+}` and `/sss` resources under the root resource (`/`)of your API. When a client calls `GET /aaa` where `aaa` is any resource name other than 'sss', the call will go through `GET /{ggg+}` with a proxy integration. In other words, in API Gateway ea method request against a specific resource takes precedence over the similar method request against a generic resource at the same level of the resource hierarchy.

API Gateway Proxy Integration Types

A proxy resource is most powerful when it is integrated with a backend 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 backend HTTP endpoint. With this integration type, API Gateway simply passes the entire request and response between the front end and the backend, subject to certain restrictions and limitations (p. 453).
- 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 backend. 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 backend with a single integration set up. For example, suppose your backend website is organized into multiple branches of tree nodes off the root node (`/site`) as: `/site/`a_0`/`a_1`/.../`a_N, `/site/`b_0`/`b_1`/.../`b_M, etc. If you integrate the `ANY` method on a proxy resource of `/api/{proxy+}` with the backend endpoints with URL paths of `/site/{proxy}`, a single integration request can support any HTTP operations (GET, POST, etc.) on any of `[`a_0`, `a_1`, ..., `a_N`,` `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 backend 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 backend 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 backend 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 (p. 54).

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 frontend:

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

  Integration request sent to the backend:

  ```
  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 backend.
- **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 backend:

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

  Method request initiated from the frontend:

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

  Integration request sent to the backend:

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

  Method request initiated from the frontend:

```
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 backend:

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

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

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 backend.

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

  Method request initiated from the frontend:

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

  Integration request sent to the backend:

```
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 backend 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 frontend:

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

  Integration request sent to the backend:

```
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 backend.

For any client request, if the targeted endpoint URL is invalid or the HTTP verb is valid but not supported, the backend 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 backend (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. 58).

The following Swagger API definition file shows an example of an API with a proxy resource that is integrated with the SimpleLambda4ProxyResource (p. 59) 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. 117). 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. 119).

With the Lambda proxy integration of a proxy resource through the `ANY` method, the single backend 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 backend 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

With the Lambda proxy integration, API Gateway maps the entire client request to the input `event` parameter of the backend Lambda function as follows:

```
{
    "resource": "Resource path",
    "path": "Path parameter",
    "httpMethod": "Incoming request's method name"
    "headers": {Incoming request headers}
    "queryStringParameters": {query string parameters }
    "pathParameters":  {path parameters}
    "stageVariables": {Applicable stage variables}
    "requestContext": {Request context, including authorizer-returned key-value pairs}
    "body": "A JSON string of the request payload."
    "isBase64Encoded": "A boolean flag to indicate if the applicable request payload is
 Base64-encode"
}
```

Let's illustrate this using 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
}
```

The above request produces the following response payload containing the output returned from the backend Lambda function, where `input` was set to the `event` parameter to the Lambda function.

```
{
  "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-PWIJc6thYnZm5P0NMgOUglL1DYtl0gdeJky8tqsg8iS_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": "gy415nuibc"
    },
    "body": "{\r\n\t\"a\": 1\r\n}",
    "isBase64Encoded": false
  }
}
```

In the input to Lambda, the `requestContext` object is a map of key-value pairs, where the key is a property name of the $context (p. 153) variable and the value the property value of the corresponding `$context` variable. API Gateway may add new keys to the map. Depending on the features enabled, the `requestContext` map may vary from API to API. For example, in the example above, `$context.authorizer.*` properties are absent because no custom authroizer is enabled for the API.

Note
API Gateway enacts certain restrictions and limitations when handling methods with either Lambda proxy integration or HTTP proxy integration. For details, see Known Issues (p. 453).

Output Format of a Lambda Function for Proxy Integration

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

```
{
    "isBase64Encoded": true|false,
    "statusCode": httpStatusCode,
    "headers": { "headerName": "headerValue", ... },
    "body": "..."
}
```

where `headers` can be null or unspecified if no extra response headers are to be returned. `body` will be marshalled to the frontend as the method response payload. If `body` is a binary blob, you can encode it as a Base64-encoded string and set `isBase64Encoded` to `true`. Otherwise, you can set it to `false` or leave it unspecified.

If the function output is of a different format, API Gateway will return a `502 Bad Gateway` error response.

In a Lambda function in Node.js, To return a successful response, call `callback(null, {"statusCode": 200, "body": "results"})`. To throw an exception, call `callback(new Error('internal server error'))`. For a client-side error, e.g., a required parameter is missing, you can call `callback(null, {"statusCode": 400, "body": "Missing parameters of ..."})` to return the error without throwing an exception.

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. 284).

To configure access control to your API, see Control Access in API Gateway (p. 214).

Create Models and Mapping Templates for Request and Response 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 backend. Similarly, the backend may return an integration response payload different from the method response payload, as expected by the frontend. API Gateway lets you use mapping templates to map the payload from a method request to the corresponding integration request and from an integration response to the corresponding method response.

A mapping template is a script expressed in Velocity Template Language (VTL) and applied to the payload using JSONPath expressions. The payload can have a data model according to the JSON Schema. You must define the model in order to have API Gateway to generate a SDK or to enable basic request validation for your API. You do not have to define any model to create a mapping template. However, a model can help you create a template because API Gateway will generate a template blueprint based on a provided model.

The section explains how to map the API request and response payload using models and mapping templates.

Topics

Models

In API Gateway, a model defines the data structure of a payload. In API Gateway models are defined using the JSON Schema.

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

Suppose we have an API for managing fruit and vegetable inventory in the produce department of a supermarket. When a manager queries the backend for the current inventory, the server sends back the following response payload:

```
{
  "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 can use the following JSON Schema to define the model for the above data:

```
{
  "$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 model:

- 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. 128) and Output Model (Photos Example) (p. 129) in the Photos Example (p. 127)
- Input Model (News Article Example) (p. 131) and Output Model (News Article Example) (p. 132) in the News Article Example (p. 130)
- Input Model (Sales Invoice Example) (p. 134) and Output Model (Sales Invoice Example) (p. 137) in the Sales Invoice Example (p. 133)
- Input Model (Employee Record Example) (p. 139) and Output Model (Employee Record Example) (p. 141) in the Employee Record Example (p. 138)

To experiment with models in API Gateway, follow the instructions in Map Response Payload (p. 33), specifically Step 1: Create Models (p. 35).

Mapping Templates

When the backend returns the query results (shown in the Models (p. 120) section), the manager of the produce department may be interested in reading them as follows:

```
{
  "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 enable this, we need to provide API Gateway a mapping template to translate the data from the backend format. The following mapping template will do just that.

```
#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
  ]
}
```

Let us now examine some details of the preceding output mapping template:

- The `$inputRoot` variable represents the root object in the original JSON data from the previous section. The variables in an output mapping template map to the original JSON data, not the desired transformed JSON data schema.
- The `choices` array in the output mapping template is mapped from the `bins` array with the root object in the original JSON data (`$inputRoot.bins`).
- In the output mapping template, each of the objects in the `choices` array (represented by `$elem`) are mapped from the corresponding objects in the `bins` array within the root object in the original JSON data.
- In the output mapping template, for each of objects in the `choices` object, the values of the `kind` and `available` objects (represented by `$elem.type` and `$elem.quantity`) are mapped from 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.

The mapping template assumes that the underlying data is of a JSON object. It does not require that a model be defined for the data. As an API developer, you know the data formats at both the front and backends. That knowledge can guide you to define the necessary mappings without ambiguity.

To have an SDK generated for the API, the above data will be returned as a language-specific object. For strongly typed languages, such as Java, Objective-C or Swift, the object corresponds to a user-defined data type (UDT). API Gateway will create such a UDT if you provide it with a data model. For the method response example above, you can define the following payload model in the integration response:

```
{
  "$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 this model, 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).

With this model, you can call an SDK to retrieve the `kind`, `suggestedPrice` and `available` property values by reading the `GroceryStoreOutputModel.kind`, `GroceryStoreOutputModel.suggestedPrice` and `GroceryStoreOutputModel.available` properties, respectively. If no model is provided, API Gateway will use the Empty model to create a default UDT. In this case, you will not be able to read these properties using a strongly-typed SDK.

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

- Input Mapping Template (Photos Example) (p. 128) and Output Mapping Template (Photos Example) (p. 130) in the Photos Example (p. 127)
- Input Mapping Template (News Article Example) (p. 132) and Output Mapping Template (News Article Example) (p. 133) in the News Article Example (p. 130)
- Input Mapping Template (Sales Invoice Example) (p. 135) and Output Mapping Template (Sales Invoice Example) (p. 137) in the Sales Invoice Example (p. 133)
- Input Mapping Template (Employee Record Example) (p. 140) and Output Mapping Template (Employee Record Example) (p. 142) in the Employee Record Example (p. 138)

To experiment with mapping templates in API Gateway, follow the instructions in Map Response Payload (p. 33), specifically Step 5: Set Up and Test the Methods (p. 39).

Tasks for Models and Mapping Templates

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

Create a Model in API Gateway

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

Topics

Prerequisites

- You must have an API available in API Gateway. Follow the instructions in Creating an API (p. 72).

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 Create Models and Mapping Templates for Request and Response Payloads (p. 119).
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

- You must have at least one model in API Gateway. Follow the instructions in Create a Model (p. 126).

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 Create Models and Mapping Templates for Request and Response Payloads (p. 119).

Topics

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 Create Models and Mapping Templates for Request and Response Payloads (p. 119).

Topics

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"
    }#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 Create Models and Mapping Templates for Request and Response Payloads (p. 119).

Topics

- Input Mapping Template (Sales Invoice Example) (p. 135)
- Transformed Data (Sales Invoice Example) (p. 136)
- Output Model (Sales Invoice Example) (p. 137)
- Output Mapping Template (Sales Invoice Example) (p. 137)

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 Create Models and Mapping Templates for Request and Response Payloads (p. 119).

Topics

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 Data Mapping

This section explains how to set up data mappings from an API's method request data, including other data stored in `context` (p. 153), `stage` (p. 158), or `util` (p. 159) 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. 304).

Topics

Map Method Request 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	`method.request.path.PARAM_NAME`
Method request query string	`method.request.querystring.PARAM_NAME`
Method request header	`method.request.header.PARAM_NAME`
Method request body	`method.request.body`
Method request body (JsonPath)	`method.request.body.JSONPath_EXPRESSION`.
Stage variables	`stageVariables.VARIABLE_NAME`
Context variables	`context.VARIABLE_NAME` that must be one of the supported context variables (p. 153).
Static value	`'STATIC_VALUE'`. The `STATIC_VALUE` 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 `methodRequestHeadParam`, 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 Integration Response Data to Method Response Headers

Method response header parameters can be mapped from any integration response header or integration response body, `$context` variables, or static values.

Method response header mapping expressions

Mapped Data Source	Mapping expression
Integration response header	`integration.response.header.`*`PARAM_NAME`*
Integration response body	`integration.response.body`
Integration response body (JsonPath)	`integration.response.body.`*`JSONPath_EXPRESSION`*
Stage variable	`stageVariables.`*`VARIABLE_NAME`*
Context variable	`context.`*`VARIABLE_NAME`* that must be one of the supported context variables (p. 153).
Static value	*`'STATIC_VALUE'`*. The *`STATIC_VALUE`* 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",

}
...
```

Map Request and Response Payloads between Method and Integration

API Gateway uses Velocity Template Language (VTL) engine to process body mapping templates (p. 123) for the integration request and integration response. The mapping templates translate method request payloads to the corresponding integration request payloads and translate integration response bodies to the method response bodies.

The VTL templates use JSONPath expressions, other parameters such as calling contexts and stage variables, and utility functions to process the JSON data.

If a model is defined to describe the data structure of a payload, API Gateway can use the model to generate a skeletal mapping template for an integration request or integration response. You can use the skeletal template as an aid to customize and expand the mapping VTL script. However, you can create a mapping template from scratch without defining a model for the payload's data structure.

Select a VTL Mapping Template

API Gateway uses the following logic to select a mapping template, in Velocity Template Language (VTL), to map the payload from a method request to the corresponding integration request or to map the payload from an integration response to the corresponding method response.

For a request payload, API Gateway uses the request’s `Content-Type` header value as the key to select the mapping template for the request payload. For a response payload, API Gateway uses the incoming request’s `Accept` header value as the key to select the mapping template.

When the `Content-Type` header is absent in the request, API Gateway assumes that its default value is `application/json`. For such a request, API Gateway uses `application/json` as the default key to select the mapping template, if one is defined. When no template matches this key, API Gateway passes the payload through unmapped if the passthroughBehavior property is set to `WHEN_NO_MATCH` or `WHEN_NO_TEMPLATES`.

When the `Accept` header is not specified in the request, API Gateway assumes that its default value is `application/json`. In this case, API Gateway selects an existing mapping template for `application/json` to map the response payload. If no template is defined for `application/json`, API Gateway selects the first existing template and uses it as the default to map the response payload. Similarly, API Gateway uses the first existing template when the specified `Accept` header value does not match any existing template key. If no template is defined, API Gateway simply passes the response payload through unmapped.

For example, suppose that an API has a `application/json` template defined for a request payload and has a `application/xml` template defined for the response payload. If the client sets the `"Content-Type : application/json"`, and `"Accept : application/xml"` headers in the request, both the request and response payloads will be processed with the corresponding mapping templates. If the `Accept:application/xml` header is absent, the `application/xml` mapping template will be used to map the response payload. To return the response payload unmapped instead, you must set up an empty template for `application/json`.

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.

Integration Passthrough Behaviors

With non proxy integrations, 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 backend without transformation. The process is known as integration passthrough.

For proxy integrations (p. 111), API Gateway passes entire request through to backend and you do not have any option to modify the passthrough behaviors.

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. 86), 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 `application/json`	The request payload is transformed using the template.	The request payload is transformed using the template.	The request payload is transformed using the template.
`application/json`	The request payload is transformed using the template.	The request payload is transformed using the template.	The request payload is transformed using the template.
`application/xml`	The request payload is not transformed and is sent to the backend 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 `application/json`	The request payload is not transformed and is	The request is rejected with an HTTP `415`	The request is rejected with an HTTP `415`

Content-Type header \Selected passthrough option	`WHEN_NO_MATCH`	`WHEN_NO_TEMPLATE`	`NEVER`
	sent to the backend as-is.	`Unsupported Media Type` response.	`Unsupported Media Type` response.
`application/json`	The request payload is not transformed and is sent to the backend 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.
`application/xml`	The request payload is transformed using the template.	The request payload is transformed using the template.	The request payload is transformed using the template.

Map Method Request Data to Customize Gateway Responses

If API Gateway fails to process an incoming request, it returns to the client an error response without forwarding the request to the integration backend. By default, the error response contains a short descriptive error message. For example, if you attempt to call an operation on an undefined API resource, you receive an error response with the `{ "message": "Missing Authentication Token" }` message. If you are new to API Gateway, you may find it difficult to understand what actually went wrong.

For some of the error responses, API Gateway allows customization by API developers to return the responses in different formats. For the `Missing Authentication Token` example, you can add a hint to the original response payload with the possible cause, as in this example: `{"message":"Missing Authentication Token", "hint":"The HTTP method or resources may not be supported."}`.

When your API mediates between an external exchange and the AWS cloud, you use VTL mapping templates for integration request or integration response to map the payload from one format to another. However, the VTL mapping templates work only for valid requests with successful responses. For invalid requests, API Gateway bypasses the integration altogether and returns an error response. You must use the customization to render the error responses in an exchange-compliant format. Here, the customization is rendered in a non-VTL mapping template supporting only simple variable substitutions.

Generalizing the API Gateway-generated error response to any responses generated by API Gateway, we refer to them as *gateway responses*. This distinguishes API Gateway-generated responses from the integration responses. A gateway response mapping template can access `$context` variable values and `$stageVariables` property values, as well as method request parameters, in the form of `method.request.`*`param-position`*`.`*`param-name`*. For more information about `$context` variables, see Accessing the $context Variable (p. 153). For more information about `$stageVariables`, see Accessing the $stageVariables Variable (p. 158). For more information about method request parameters, see Request parameters accessible by a mapping template (p. 143).

API Gateway Responses

An API Gateway response is identified by a response type defined by API Gateway. The response consists of an HTTP status code, a set of additional headers that are specified by parameter mappings, and a payload that is generated by a non-VTL mapping template.

In the API Gateway REST API, a gateway response is represented by the GatewayResponse. In Swagger, a `GatewayResponse` instance is described by the x-amazon-apigateway-gateway-responses.gatewayResponse (p. 363) extension.

To enable a gateway response, you set up a gateway response for a supported response type (p. 148) at the API level. Whenever API Gateway returns a response of the type, the header mappings and payload mapping templates defined in the gateway response are applied to return the mapped results to the API caller.

In the following section, we show how to set up gateway responses using the API Gateway console and the API Gateway REST API.

Gateway Response Types Supported by API Gateway

API Gateway exposes the following gateway responses for customization by API developers.

Gateway response type	Default status code	Description
`DEFAULT_4XX`	Null	The default gateway response for an unspecified response type with the status code of `4XX`. Changing the status code of this fallback gateway response changes the status codes of all other `4XX` responses to the new value. Resetting this status code to null reverts the status codes of all other `4XX` responses to their original values.
`DEFAULT_5XX`	Null	The default gateway response for an unspecified response type with a status code of `5XX`. Changing the status code of this fallback gateway response changes the status codes of all other `5XX` responses to the new value. Resetting this status code to null reverts the status codes of all other `5XX` responses to their original values.
`ACCESS_DENIED`	`403`	The gateway response for authorization failure; for example, when access is denied by a custom or Amazon Cognito authorizer. If the response type is unspecified, this response defaults to the `DEFAULT_4XX` type.
`API_CONFIGURATION_ERROR`	`500`	The gateway response for invalid API configuration, including invalid endpoint address submitted, Base64 decoding failed on binary data when binary support is enacted, or integration response mapping cannot match any template and no default template is configured. If the response type is unspecified, this response

Gateway response type	Default status code	Description
		defaults to the `DEFAULT_5XX` type.
`AUTHORIZER_CONFIGURATION_ERROR`	`500`	The gateway response for failing to connect to a custom or Amazon Cognito authorizer. If the response type is unspecified, this response defaults to the `DEFAULT_5XX` type.
`AUTHORIZER_FAILURE`	`500`	The gateway response when a custom or Amazon Cognito authorizer failed to authenticate the caller. If the response type is unspecified, this response defaults to the `DEFAULT_5XX` type.
`BAD_REQUEST_PARAMETERS`	`400`	The gateway response when the request parameter cannot be validated according to an enabled request validator. If the response type is unspecified, this response defaults to the `DEFAULT_4XX` type.
`BAD_REQUEST_BODY`	`400`	The gateway response when the request body cannot be validated according to an enabled request validator. If the response type is unspecified, this response defaults to the `DEFAULT_4XX` type.
`EXPIRED_TOKEN`	`403`	The gateway response for an AWS authentication token expired error. If the response type is unspecified, this response defaults to the `DEFAULT_4XX` type.
`INTEGRATION_FAILURE`	`504`	The gateway response for an integration failed error. If the response type is unspecified, this response defaults to the `DEFAULT_5XX` type.
`INTEGRATION_TIMEOUT`	`504`	The gateway response for an integration timed out error. If the response type is unspecified, this response defaults to the `DEFAULT_5XX` type.

Gateway response type	Default status code	Description
`INVALID_API_KEY`	`403`	The gateway response for an invalid API key submitted for a method requiring an API key. If the response type is unspecified, this response defaults to the `DEFAULT_4XX` type.
`INVALID_SIGNATURE`	`403`	The gateway response for an invalid AWS signature error. If the response type is unspecified, this response defaults to the `DEFAULT_4XX` type.
`MISSING_AUTHENTICATION_TOKEN`	`403`	The gateway response for a missing authentication token error, including the cases when the client attempts to invoke an unsupported API method or resource. If the response type is unspecified, this response defaults to the `DEFAULT_4XX` type.
`QUOTA_EXCEEDED`	`429`	The gateway response for the usage plan quota exceeded error. If the response type is unspecified, this response defaults to the `DEFAULT_4XX` type.
`REQUEST_TOO_LARGE`	`413`	The gateway response for the request too large error. If the response type is unspecified, this response defaults to the `DEFAULT_4XX` type.
`RESOURCE_NOT_FOUND`	`404`	The gateway response when API Gateway cannot find the specified resource after an API request passes authentication and authorization, except for API key authentication and authorization. If the response type is unspecified, this response defaults to the `DEFAULT_4XX` type.
`THROTTLED`	`429`	The gateway response when usage plan-, method-, stage-, or account-level throttling limits exceeded. If the response type is unspecified, this response defaults to the `DEFAULT_4XX` type.

Gateway response type	Default status code	Description
`UNAUTHORIZED`	`401`	The gateway response when the custom or Amazon Cognito authorizer failed to authenticate the caller.
`UNSUPPORTED_MEDIA_TYPE`	`415`	The gateway response when a payload is of an unsupported media type, if strict passthrough behavior is enabled. If the response type is unspecified, this response defaults to the `DEFAULT_4XX` type.

Use the API Gateway Console to Customize a Gateway Response

To customize a gateway response using the API Gateway console

1. Sign in to the API Gateway console.
2. Choose your existing API or create a new one.
3. Expand the API in the primary navigation pane and choose **Gateway Responses** under the API.
4. In the **Gateway Responses** pane, choose a response type. In this walkthrough, we use **Missing Authentication Token (403)** as an example.
5. You can change the API Gateway-generated **Status Code** to return a different status code that meets your API's requirements.
6. To return custom headers, choose **Add Header** under **Response Headers**. For illustration purposes, we add the following custom headers:

```
Access-Control-Allow-Origin:'a.b.c'
x-request-id:method.request.header.Accept
x-request-path:method.request.path.petId
x-request-query:method.request.querystring.q
```

   In the preceding header mappings, a static domain name (`'a.b.c'`) is mapped to the `Allow-Control-Allow-Origin` header to allow CORS access to the API; the input request header of `x-amazn-RequestId` is mapped to `request-id` in the response; the `petId` path variable of the incoming request is mapped to the `request-path` header in the response; and the `q` query parameter of the original request is mapped to the `request-query` header of the response.
7. Under **Body Mapping Templates**, leave `application/json` for **Content Type** and type the following body mapping template in the **Body Mapping Template** editor:

```
{
     "message":"$context.error.messageString",
     "type": "$context.error.responseType",
     "statusCode": "'404'",
     "stage": "$context.stage",
     "resourcePath": "$context.resourcePath",
     "stageVariables.a": "$stageVariables.a"
}
```

   This example shows how to map `$context` and `$stageVariables` properties to properties of the gateway response body.

8. Choose **Save**.
9. Deploy the API to a new or existing stage.
10. Test it by calling the following CURL command:

```
curl -v https://o81lxisefl.execute-api.us-east-1.amazonaws.com/custErr/pets/5/type?q=1
```

You should get a gateway response similar to the following:

```
> GET /custErr/pets/5/type?q=1 HTTP/1.1
Host: o81lxisefl.execute-api.us-east-1.amazonaws.com
User-Agent: curl/7.51.0
Accept: */*

HTTP/1.1 404 Not Found
Content-Type: application/json
Content-Length: 334
Connection: keep-alive
Date: Tue, 02 May 2017 03:15:47 GMT
x-amzn-RequestId: a2be05a4-2ee5-11e7-bbf2-df131ec50ae6
Access-Control-Allow-Origin: a.b.c
x-amzn-ErrorType: MissingAuthenticationTokenException
header-1: static
x-request-query: 1
x-request-path:
X-Cache: Error from cloudfront
Via: 1.1 441811a054e8d055b893175754efd0c3.cloudfront.net (CloudFront)
X-Amz-Cf-Id: nNDR-fX4csbRoAgtQJ16u0rTDz9FZWT-Mk93KgoxnfzDlTUh3flmzA==

{
     "message":"Missing Authentication Token",
     "type": MISSING_AUTHENTICATION_TOKEN,
     "statusCode": '404',
     "stage": custErr,
     "resourcePath": ,
     "stageVariables.a": a
}
```

The preceding example assumes that the API backend is Pet Store and the API has a stage variable, `a`, defined.

Use the API Gateway REST API to Customize a Gateway Response

Before customizing a gateway response using the API Gateway REST API, you must have already created an API and have obtained its identifier. To retrieve the API identifier, you can follow restapi:gateway-responses link relation and examine the result.

To customize a gateway response using the API Gateway REST API

1. To overwrite an entire GatewayResponse instance, call the gatewayresponse:put action, specifying a desired responseType in the URL path parameter and supplying in the request payload the statusCode, responseParameters and responseTemplates mappings.
2. To update part of a `GatewayResponse` instance, call the gatewayresponse:update action, specifying a desired `responseType` in the URL path parameter and supplying in the request payload desired individual `GatewayResponse` properties, for example, the `responseParameters` or the `responseTemplates` mapping.

Set up Gateway Response Customization in Swagger

You can use the `x-amazon-apigateway-gateway-responses` extension at the API root level to customize gateway responses in Swagger. The following Swagger definition shows an example for customizing the GatewayResponse of the `MISSING_AUTHENTICATION_TOKEN` type.

```
  "x-amazon-apigateway-gateway-responses": {
    "MISSING_AUTHENTICATION_TOKEN": {
      "statusCode": 404,
      "responseParameters": {
        "gatewayresponse.header.x-request-path": "method.input.params.petId",
        "gatewayresponse.header.x-request-query": "method.input.params.q",
        "gatewayresponse.header.Access-Control-Allow-Origin": "'a.b.c'",
        "gatewayresponse.header.x-request-header": "method.input.params.Accept"
      },
      "responseTemplates": {
        "application/json": "{\n     \"message\": $context.error.messageString,\n
    \"type\":  \"$context.error.responseType\",\n     \"stage\":  \"$context.stage
\",\n     \"resourcePath\":  \"$context.resourcePath\",\n     \"stageVariables.a\":
 \"$stageVariables.a\",\n     \"statusCode\": \"'404'\"\n}"
      }
    }
```

In this example, the customization changes the status code from the default (`403`) to `404`. It also adds to the gateway response four header parameters and one body mapping template for the `application/json` media type.

API Gateway Mapping Template Built-in Functions and Variables

Amazon API Gateway defines a set of variables and functions 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

The `$context` variable holds all the contextual information of your API call.

$context Variable Reference

Parameter	Description
`$context.apiId`	The identifier API Gateway assigns to your API.
`$context.authorizer.claims.`*`property`*	A property of the claims returned from the Amazon Cognito user pool after the method caller is successfully authenticated.

Parameter	Description
	Note Calling `$context.authorizer.claims` returns null.
`$context.authorizer.principalId`	The principal user identification associated with the token sent by the client and returned from an API Gateway custom authorizer Lambda function.

`$context.authorizer.`*`property`*	The stringified value of the specified key-value pair of the `context` map returned from an API Gateway custom authorizer Lambda function. For example, if the authorizer returns the following `context` map:
	`"context" : {`
	`  "key": "value",`
	`  "numKey": 1,`
	`  "boolKey": true`
	`}`
	
	calling `$context.authorizer.key` returns the `"value"` string, calling `$context.authorizer.numKey` returns the `"1"` string, and calling `$context.authorizer.boolKey` returns the `"true"` string.

`$context.httpMethod`	The HTTP method used. Valid values include: `DELETE`, `GET`, `HEAD`, `OPTIONS`, `PATCH`, `POST`, and `PUT`.
`$context.error.message`	A string of a customizable API Gateway error message. This variable can only be used for simple variable substitution in a GatewayResponse body-mapping template, which is not processed by the Velocity Template Language engine.
`$context.error.responseType`	A type of GatewayResponse. This variable can only be used for simple variable substitution in a GatewayResponse body-mapping template, which is not processed by the Velocity Template Language engine.
`$context.identity.accountId`	The AWS account ID associated with the request.
`$context.identity.apiKey`	The API owner key associated with your API.
`$context.identity.caller`	The principal identifier of the caller making the request.
`$context.identity.cognitoAuthenticationProvider`	The Amazon Cognito authentication provider used by the caller making the request. Available only if the request was signed with Amazon Cognito credentials. For information related to this and the other Amazon Cognito `$context` variables, see Using Federated Identities in the *Amazon Cognito Developer Guide*.

Parameter	Description
`$context.identity.cognitoAuthenticationType`	The Amazon Cognito authentication type of the caller making the request. Available only if the request was signed with Amazon Cognito credentials.
`$context.identity.cognitoIdentityId`	The Amazon Cognito identity ID of the caller making the request. Available only if the request was signed with Amazon Cognito credentials.
`$context.identity.cognitoIdentityPoolId`	The Amazon Cognito identity pool ID of the caller making the request. Available only if the request was signed with Amazon Cognito credentials.
`$context.identity.sourceIp`	The source IP address of the TCP connection making the request to API Gateway.
`$context.identity.user`	The principal identifier of the user making the request.
`$context.identity.userAgent`	The User Agent of the API caller.
`$context.identity.userArn`	The Amazon Resource Name (ARN) of the effective user identified after authentication.
`$context.requestId`	An automatically generated ID for the API call.
`$context.resourceId`	The identifier API Gateway assigns to your resource.
`$context.resourcePath`	The path to your resource. For more information, see Build an API Gateway API to Expose an HTTP Endpoint (p. 8).
`$context.stage`	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
`$input.body`	Returns the raw payload as a string.
`$input.json(x)`	This function evaluates a JSONPath expression and returns the results as a JSON string. For example, `$input.json('$.pets')` will return a JSON string representing the pets structure. For more information about JSONPath, see JSONPath or JSONPath for Java.
`$input.params()`	Returns a map of all the request parameters of your API call.
`$input.params(x)`	Returns the value of a method request parameter from the path, query string, or header value (in that order) given a parameter name string x.
`$input.path(x)`	Takes a JSONPath expression string (x) and returns an object representation of the result. This allows you to access and manipulate elements of the payload natively in Apache Velocity Template Language (VTL). For example, `$input.path('$.pets').size()` 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 Create Models and Mapping Templates for Request and Response Payloads (p. 119)

Accessing the $stageVariables Variable

The syntax for inserting a stage variable looks like this: `$stageVariables`.

$stageVariables Reference

Syntax	Description
`$stageVariables.<variable_name>`	*<variable_name>* represents a stage variable name.
`$stageVariables['<variable_name>']`	*<variable_name>* represents any stage variable name.

Syntax	Description
`${stageVariables['<variable_name>']}`	*`<variable_name>`* 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
`$util.escapeJavaScript()`	Escapes the characters in a string using JavaScript string rules. **Note** This function will turn any regular single quotes (`'`) into escaped ones (`\'`). However, the escaped single quotes are not valid in JSON. Thus, when the output from this function is used in a JSON property, you must turn any escaped single quotes (`\'`) back to regular single quotes (`'`). This is shown in the following example: `$util.escapeJavaScript(data).replaceAll("\\'","'")`

`$util.parseJson()`	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:
	
	`{"errorMessage":"{\"key1\":\"var1\",\"key2\":{\"arr\":[1,2,3]}}"}`
	
	and use the following mapping template
	
	`#set ($errorMessageObj = $util.parseJson($input.path('$.errorMessage')))`
	`{`
	`   "errorMessageObjKey2ArrVal" : $errorMessageObj.key2.arr[0]`
	`}`
	
	You will get the following output:
	
	`{`
	`   "errorMessageObjKey2ArrVal" : 1`

Function	Description
	`}`
`$util.urlEncode()`	Converts a string into "application/x-www-form-urlencoded" format.
`$util.urlDecode()`	Decodes an "application/x-www-form-urlencoded" string.
`$util.base64Encode()`	Encodes the data into a base64-encoded string.
`$util.base64Decode()`	Decodes the data from a base64-encoded string.

Enable Basic Request Validation for an API in API Gateway

You can configure API Gateway to perform basic validation of an API request before proceeding with the integration request. When the validation fails, API Gateway immediately fails the request, returns a 400 error response to the caller, and publishes the validation results in CloudWatch Logs. This reduces unnecessary calls to the backend. More importantly, it lets you focus on the validation efforts specific to your application.

Topics

Overview of Basic Request Validation in API Gateway

API Gateway can perform the basic validation. This enables you, the API developer, to focus on app-specific deep validation in the backend. For the basic validation, API Gateway verifies either or both of the following conditions:

- The required request parameters in the URI, query string, and headers of an incoming request are included and non-blank.
- The applicable request payload adheres to the configured JSON-Schema request model (p. 126) of the method.

To enable basic validation, you specify validation rules in a request validator, add the validator to the API's map of request validators, and assign the validator to individual API methods.

Note
Request body validation and request body passthrough (p. 146) are two separate issues. When a request payload cannot be validated because no model schema can be matched, you can choose to passthrough or block the original payload. For example, when you enable request validation with a mapping template for the `application/json` media type, you may want to pass an XML payload through to the backend even though the enabled request validation will fail. This may be the case if you expect to support the XML payload on the method in the future. To fail the request with an XML payload, you must explicitly choose the `NEVER` option for the content passthrough behavior.

Set up Basic Request Validation in API Gateway

You can set up request validators in an API's Swagger definition file and then import the Swagger definitions into API Gateway. You can also set them up in the API Gateway console or by calling the API Gateway REST API, AWS CLI, or one of the AWS SDKs. Here, we show how to do this with a Swagger file, in the console, and using the API Gateway REST API.

Topics

Set up Basic Request Validation by Importing API Swagger Definition

The following steps describe how to enable basic request validation by importing a Swagger file.

To enable request validation by importing a Swagger file into API Gateway

1. Declare request validators in Swagger by specifying a set of the x-amazon-apigateway-request-validators.requestValidator (p. 375) objects in the x-amazon-apigateway-request-validators (p. 374) map at the API level. For example, the sample API Swagger file (p. 168) contains the `x-amazon-apigateway-request-validators` map, with the validators' names as the keys.

```
{
      "swagger": "2.0",
      "info": {
        "title": "ReqValidation Sample",
        "version": "1.0.0"
      },
      "schemes": [
        "https"
      ],
      "basePath": "/v1",
      "produces": [
        "application/json"
      ],
      "x-amazon-apigateway-request-validators" : {
        "all" : {
          "validateRequestBody" : true,
          "validateRequestParameters" : true
        },
        "params-only" : {
          "validateRequestBody" : false,
          "validateRequestParameters" : true
        }
      },
      ...
}
```

   You select a validator's name when enabling the validator on the API or on a method, as shown in the next step.

2. To enable a request validator on all methods of an API, specify an x-amazon-apigateway-request-validator (p. 373) property at the API level of the API Swagger definition file. To enable a request validator on an individual method, specify the `x-amazon-apigateway-request-validator` property

at the method level. For example, the following `x-amazon-apigateway-request-validator` property enables the `params-only` validator on all API methods, unless otherwise overridden.

```
{
      "swagger": "2.0",
      "info": {
        "title": “ReqValidation Sample",
        "version": "1.0.0"
      },
      "schemes": [
        "https"
      ],
      "basePath": "/v1",
      "produces": [
        "application/json"
      ],
      ...
      "x-amazon-apigateway-request-validator" : "params-only",
      ...
}
```

To enable a request validator on an individual method, specify the `x-amazon-apigateway-request-validator` property at the method level. For example, the following `x-amazon-apigateway-request-validator` property enables the `all` validator on the `POST /validation` method. This overrides the `params-only` validator that is inherited from the API.

```
{
      "swagger": "2.0",
      "info": {
        "title": “ReqValidation Sample",
        "version": "1.0.0"
      },
      "schemes": [
        "https"
      ],
      "basePath": "/v1",
      "produces": [
        "application/json"
      ],
      ...
      "paths": {
        "/validation": {
          "post": {
            "x-amazon-apigateway-request-validator" : "all",
            ...
          },
          ...
        }
      }
      ...
}
```

3. In API Gateway, create the API with request validators enabled by importing this Sample API Swagger Definition (p. 168):

```
POST /restapis?mode=import&failonwarning=true HTTP/1.1
Content-Type: application/json
Host: apigateway.us-east-1.amazonaws.com
X-Amz-Date: 20170306T234936Z
```

```
Authorization: AWS4-HMAC-SHA256 Credential={access_key_ID}/20170306/us-east-1/
apigateway/aws4_request, SignedHeaders=content-length;content-type;host;x-amz-date,
 Signature={sig4_hash}

Copy the JSON object from this API Swagger Definition (p. 168) and paste it here.
```

4. Deploy the newly created API (`fjd6crafxc`) to a specified stage (`testStage`).

```
POST /restapis/fjd6crafxc/deployments HTTP/1.1
Content-Type: application/json
Host: apigateway.us-east-1.amazonaws.com
X-Amz-Date: 20170306T234936Z
Authorization: AWS4-HMAC-SHA256 Credential={access_key_ID}/20170306/us-east-1/
apigateway/aws4_request, SignedHeaders=content-length;content-type;host;x-amz-date,
 Signature={sig4_hash}

{
    "stageName" : "testStage",
    "stageDescription" : "Test stage",
    "description" : "First deployment",
    "cacheClusterEnabled" : "false"
}
```

For instructions on how to test the request validation using the API Gateway REST API, see Test Basic Request Validation Using the API Gateway REST API (p. 165). For instructions on how to test using the API Gateway console, see Test Basic Request Validation Using the API Gateway Console (p. 168).

Set up Request Validators Using the API Gateway REST API

In the API Gateway REST API, a request validator is represented by a RequestValidator resource. To have an API support the same request validators as the Sample API (p. 168), add to the RequestValidators collection a parameters-only validator with `params-only` as the key, and add a full validator with `all` as its key.

To enable the basic request validation using the API Gateway REST API

We assume that you have an API similar to the sample API (p. 168), but have not set up the request validators. If your API already has request validators enabled, call the appropriate `requestvalidator:update` or `method:put` action instead of `requestvalidator:create` or `method:put`.

1. To set up the `params-only` request validator, call the requestvalidator:create action as follows:

```
POST /restapis/restapi-id/requestvalidators HTTP/1.1
Content-Type: application/json
Host: apigateway.region.amazonaws.com
X-Amz-Date: 20170223T172652Z
Authorization: AWS4-HMAC-SHA256 Credential={access_key_ID}/20170223/region/apigateway/
aws4_request, SignedHeaders=content-type;host;x-amz-date, Signature={sig4_hash}

{
  "name" : "params-only",
  "validateRequestBody" : "false",
  "validateRequestParameters" : "true"
}
```

2. To set up the `all` request validator, call the `requestvalidator:create` action as follows:

```
POST /restapis/restapi-id/requestvalidators HTTP/1.1
Content-Type: application/json
Host: apigateway.region.amazonaws.com
```

```
X-Amz-Date: 20170223T172652Z
Authorization: AWS4-HMAC-SHA256 Credential={access_key_ID}/20170223/region/apigateway/
aws4_request, SignedHeaders=content-type;host;x-amz-date, Signature={sig4_hash}

{
  "name" : "all",
  "validateRequestBody" : "true",
  "validateRequestParameters" : "true"
}
```

If the preceding validator keys already exist in the `RequestValidators` map, call the requestvalidator:update action instead to reset the validation rules.

3. To apply the `all` request validator to the `POST` method, call method:put to enable the specified validator (as identified by the requestValidatorId property) or call method:update to update the enabled validator.

```
PUT /restapis/restapi-id/resources/resource-id/methods/POST HTTP/1.1
Content-Type: application/json
Host: apigateway.region.amazonaws.com
X-Amz-Date: 20170223T172652Z
Authorization: AWS4-HMAC-SHA256 Credential={access_key_ID}/20170223/region/apigateway/
aws4_request, SignedHeaders=content-type;host;x-amz-date, Signature={sig4_hash}

{
  "authorizationType" : "NONE",
  ...,
  "requestValidatorId" : "all"
}
```

Set up Basic Request Validation Using the API Gateway Console

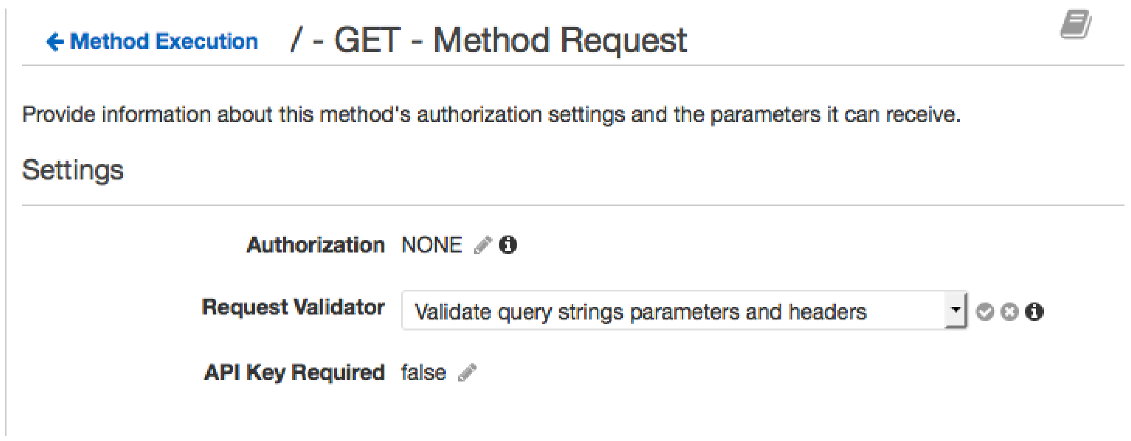
The API Gateway console lets you set up the basic request validation on a method using one of the three validators:

- **Validate body**: This is the body-only validator.
- **Validate query string parameters and headers**: This is the parameters-only validator.
- **Validate body, query string parameters, and headers**: This validator is for both body and parameters validation.

When you choose one of the above validators to enable it on an API method, the API Gateway console will add the validator to the API's RequestValidators map, if the validator has not already been added to the validators map of the API.

To enable a request validator on a method

1. Sign in to the API Gateway console, if not already logged in.
2. Create a new or choose an existing API.
3. Create a new or choose an existing resource of the API.
4. Create a new or choose an existing method the resource.
5. Choose **Method Request**.
6. Choose the pencil icon of **Request Validator** under **Settings**.
7. Choose `Validate body`, `Validate query string parameters and headers` or `Validate body, query string parameters, and headers` from the **Request Validator** drop-down list and then choose the check mark icon to save your choice.



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

Settings

Authorization NONE

Request Validator Validate query strings parameters and headers

API Key Required false

To test and use the request validator in the console, follow the instructions in Test Basic Request Validation Using the API Gateway Console (p. 168).

Test Basic Request Validation in API Gateway

Choose one of the following topics for instructions on testing the basic request validation against the sample API (p. 168).

Topics

Test Basic Request Validation Using the API Gateway REST API

To see the invocation URL of the deployed API, you can export the API from the stage, making sure to include the `Accept: application/json` or `Accept: application/yaml` header:

```
GET /restapis/fjd6crafxc/stages/testStage/exports/swagger?extensions=validators HTTP/1.1
Accept: application/json
Content-Type: application/json
Host: apigateway.us-east-1.amazonaws.com
X-Amz-Date: 20170306T234936Z
Authorization: AWS4-HMAC-SHA256 Credential={access_key_ID}/20170306/us-east-1/
apigateway/aws4_request, SignedHeaders=content-length;content-type;host;x-amz-date,
 Signature={sig4_hash}
```

You can ignore the `?extensions=validators` query parameter, if you do not want to download the Swagger specifications related to the request validation.

To test request validation using the API Gateway REST API calls

1. Call `GET /validation?q1=cat`.

   ```
   GET /testStage/validation?q1=cat HTTP/1.1
   Host: fjd6crafxc.execute-api.us-east-1.amazonaws.com
   Content-Type: application/json
   Accept: application/json
   ```

Because the required parameter of `q1` is set and not blank, the request passes the validation. API Gateway returns the following `200 OK` response:

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

2. Call `GET /validation`.

```
GET /testStage/validation HTTP/1.1
Host: fjd6crafxc.beta.execute-api.us-east-1.amazonaws.com
Content-Type: application/json
Accept: application/json
```

   Because the required parameter of `q1` is not set, the request fails to pass the validation. API Gateway returns the following `400 Bad Request` response:

```
{
    "message": "Missing required request parameters: [q1]"
}
```

3. Call `GET /validation?q1=`.

```
GET /testStage/validation?q1= HTTP/1.1
Host: fjd6crafxc.beta.execute-api.us-east-1.amazonaws.com
Content-Type: application/json
Accept: application/json
```

   Because the required parameter of `q1` is blank, the request fails to pass the validation. API Gateway returns the same `400 Bad Request` response as in the previous example.

4. Call `POST /validation`.

```
POST /testStage/validation HTTP/1.1
Host: fjd6crafxc.beta.execute-api.us-east-1.amazonaws.com
Content-Type: application/json
Accept: application/json
h1: v1

{
    "name" : "Marco",
    "type" : "dog",
    "price" : 260
}
```

Because the required header parameter of `h1` is set and not blank and the payload format adheres to the `RequestDataModel` required properties and associated constraints, the request passes the validation. API Gateway returns the following successful response.

```
{
    "pet": {
      "name": "Marco",
      "type": "dog",
      "price": 260
    },
    "message": "success"
}
```

5. Call `POST /validation`, without specifying the `h1` header or setting its value blank.

```
POST /testStage/validation HTTP/1.1
Host: fjd6crafxc.execute-api.us-east-1.amazonaws.com
Content-Type: application/json
Accept: application/json

{
    "name" : "Marco",
    "type" : "dog",
    "price" : 260
}
```

   Because the required header parameter of `h1` is missing or set to blank, the request fails to pass the validation. API Gateway returns the following `400 Bad Request` response:

```
{
    "message": "Missing required request parameters: [h1]"
}
```

6. Call `POST /validation`, setting the `type` property of the payload to `bird`.

```
POST /testStage/validation HTTP/1.1
Host: fjd6crafxc.execute-api.us-east-1.amazonaws.com
Content-Type: application/json
Accept: application/json
X-Amz-Date: 20170309T000215Z
h1: v1

{
    "name" : "Molly",
    "type" : "bird",
    "price" : 269
}
```

   Because the `type` property value is not a member of the enumeration of `["dog", "cat", "fish"]`, the request fails to pass the validation. API Gateway returns the following `400 Bad Request` response:

```
{
    "message": "Invalid request body"
}
```

   Setting `price` to `501` violates the constraint on the property. This causes the validation to fail and the same `400 Bad Request` response is returned.

Test Basic Request Validation Using the API Gateway Console

The following steps describe how to test basic request validation in the API Gateway console.

To test the request validation on a method using TestInvoke in the API Gateway console

While signed in to the API Gateway console, do the following:

1. Choose **Resources** for the API for which you have configured a request validators map.
2. Choose a method for which you have enabled the request validation with a specified request validator.
3. Under **Method Execution**, in the **Client** box, choose **Test**.
4. Try different values for required request parameter or applicable body, and then choose **Test** to see the response.

When the method call passes validation, you should get expected responses. If validation fails, the following error message returns if the payload is not the correct format:

```
{
  "message": "Invalid request body"
}
```

If the request parameters are not valid, the following error message returns:

```
{
  "message": "Missing required request parameters: [p1]"
}
```

Swagger Definitions of a Sample API with Basic Request Validation

The following Swagger definition defines a sample API with request validation enabled. The API is a subset of the PetStore API. It exposes a `POST` method to add a pet to the `pets` collection and a `GET` method to query pets by a specified type.

There are two request validators declared in the `x-amazon-apigateway-request-validators` map at the API level. The `params-only` validator is enabled on the API and inherited by the `GET` method. This validator allows API Gateway to verify that the required query parameter (`q1`) is included and not blank in the incoming request. The `all` validator is enabled on the `POST` method. This validator verifies that the required header parameter (`h1`) is set and not blank. It also verifies that the payload format adheres to the specified `RequestBodyModel` schema. This model requires that the input JSON object contains the `name`, `type`, and `price` properties. The `name` property can be any string, `type` must be one of the specified enumeration fields (`["dog", "cat", "fish"]`), and `price` must range between 25 and 500. The `id` parameter is not required.

For more information about the behavior of this API, see Enable Basic Request Validation for an API in API Gateway (p. 160).

```

{
  "swagger": "2.0",
  "info": {
    "title": "ReqValidators Sample",
    "version": "1.0.0"
  },
  "schemes": [
    "https"
  ],
  "basePath": "/v1",
  "produces": [
    "application/json"
  ],
  "x-amazon-apigateway-request-validators" : {
    "all" : {
      "validateRequestBody" : true,
      "validateRequestParameters" : true
    },
    "params-only" : {
      "validateRequestBody" : false,
      "validateRequestParameters" : true
    }
  },
  "x-amazon-apigateway-request-validator" : "params-only",
  "paths": {
    "/validation": {
      "post": {
        "x-amazon-apigateway-request-validator" : "all",
        "parameters": [
          {
            "in": "header",
            "name": "h1",
            "required": true
          },
          {
            "in": "body",
            "name": "RequestBodyModel",
            "required": true,
            "schema": {
              "$ref": "#/definitions/RequestBodyModel"
            }
          }
        ],
        "responses": {
          "200": {
            "schema": {
              "type": "array",
              "items": {
                "$ref": "#/definitions/Error"
              }
            },
            "headers" : {
              "test-method-response-header" : {
                "type" : "string"
              }
            }
          }
        },
        "security" : [{
          "api_key" : []
        }],
        "x-amazon-apigateway-auth" : {
          "type" : "none"
        },
        "x-amazon-apigateway-integration" : {
          "type" : "http",
          "uri" : "http://petstore-demo-endpoint.execute-api.com/petstore/pets",
          "httpMethod" : "POST",
          "requestParameters": {
            "integration.request.header.custom_h1": "method.request.header.h1"
          },
          "responses" : {
            "2\\d{2}" : {
              "statusCode" : "200"
            },
            "default" : {
              "statusCode" : "400",
              "responseParameters" : {
                "method.response.header.test-method-response-header" : "'static value'"
              },
              "responseTemplates" : {
                "application/json" : "json 400 response template",
                "application/xml" : "xml 400 response template"
              }
            }
          }
        }
      },
      "get": {
        "parameters": [
          {
            "name": "q1",
            "in": "query",
            "required": true
          }
        ],
        "responses": {
          "200": {
            "schema": {
              "type": "array",
              "items": {
                "$ref": "#/definitions/Error"
              }
            },
            "headers" : {
              "test-method-response-header" : {
                "type" : "string"
              }
            }
          }
        },
        "security" : [{
          "api_key" : []
        }],
        "x-amazon-apigateway-auth" : {
          "type" : "none"
        },
        "x-amazon-apigateway-integration" : {
          "type" : "http",
          "uri" : "http://petstore-demo-endpoint.execute-api.com/petstore/pets",
          "httpMethod" : "GET",
          "requestParameters": {
            "integration.request.querystring.type": "method.request.querystring.q1"
          },
          "responses" : {
            "2\\d{2}" : {
              "statusCode" : "200"
            },
            "default" : {
              "statusCode" : "400",
              "responseParameters" : {
                "method.response.header.test-method-response-header" : "'static value'"
              },
              "responseTemplates" : {
                "application/json" : "json 400 response template",
                "application/xml" : "xml 400 response template"
              }
            }
          }
        }
      }
    }
  },
  "definitions": {
    "RequestBodyModel": {
      "type": "object",
      "properties": {
        "id":    { "type": "integer" },
        "type":  { "type": "string", "enum": ["dog", "cat", "fish"] },
        "name":  { "type": "string" },
        "price": { "type": "number", "minimum": 25, "maximum": 500 }
      },
      "required": ["type", "name", "price"]
    },
    "Error": {
      "type": "object",
      "properties": {

      }
    }
  }
}
```

Document 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

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

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. 173).

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\"info\": {\n\t\t\"description\": \"My first API with Amazon API Gateway.\"\n\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`, `METHOD`, `MODEL`, `REQUEST_BODY` 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` > `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`	`statusCode`	`name`	Remarl
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

Case	path	statusCode	name	Remarks
				specific than Case 1.
3	/pets/petId	*	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.

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	`{ "location": { "type": "API" }, ... }`	`type`	N/A	No
Resource	`{ "location": { "type": "RESOURCE", "path": "resource_path" }, ... }`	`type`	The default value of `path` is `/`.	No
Method	`{ "location": { "type": "METHOD", "path": "resource_path", "method": "http_verb" }, ... }`	`type`	The default values of `path` and `method` are `/` and `*`, respectively.	Yes, matching `path` by prefix and matching `method` of any values.
Query parameter	`{ "location": { "type": "QUERY_PARAMETER", "path": "resource_path", "method": "HTTP_verb", "name": "query_parameter_name"`	`type`	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
	`}, ... }`			
Request body	`{ "location": { "type": "REQUEST", "path": "resource_path", "method": "http_verb" }, ... }`	`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	`{ "location": { "type": "REQUEST_HEADER", "path": "resource_path", "method": "HTTP_verb", "name": "header_name" }, ... }`	`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	`{ "location": { "type": "PATH_PARAMETER", "path": "resource/{path_parameter_name}", "method": "HTTP_verb", "name": "path_parameter_name" }, ... }`	`type`, `name`	The default values of `path` and `method` are `/` and `*`, respectively.	Yes, matching `path` by prefix and matching `method` by exact values.
Response	`{ "location": { "type": "RESPONSE", "path": "resource_path", "method": "http_verb", "statusCode": "status_code" }, ... }`	`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
Response header	`{ "location": { "type": "RESPONSE_HEADER", "path": "resource_path", "method": "http_verb", "statusCode": "status_code", "name": "header_name" }, ... }`	`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	`{ "location": { "type": "RESPONSE_BODY", "path": "resource_path", "method": "http_verb", "statusCode": "status_code" }, ... }`	`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.
Authorizer	`{ "location": { "type": "AUTHORIZER", "name": "authorizer_name" }, ... }`	`type`	N/A	No
Model	`{ "location": { "type": "MODEL", "name": "model_name" }, ... }`	`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

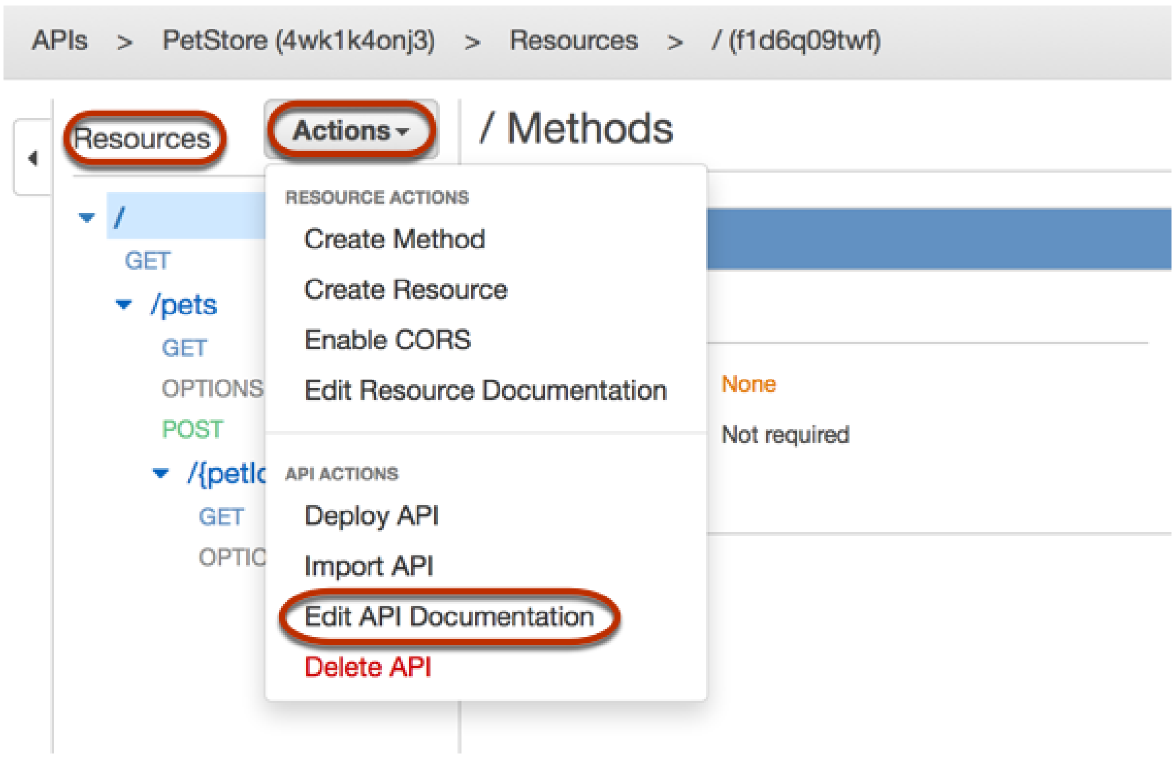
In this section, we describe how to create and maintain documentation parts of an API using the API Gateway console.

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. 9).

Topics

Document the `API` Entity

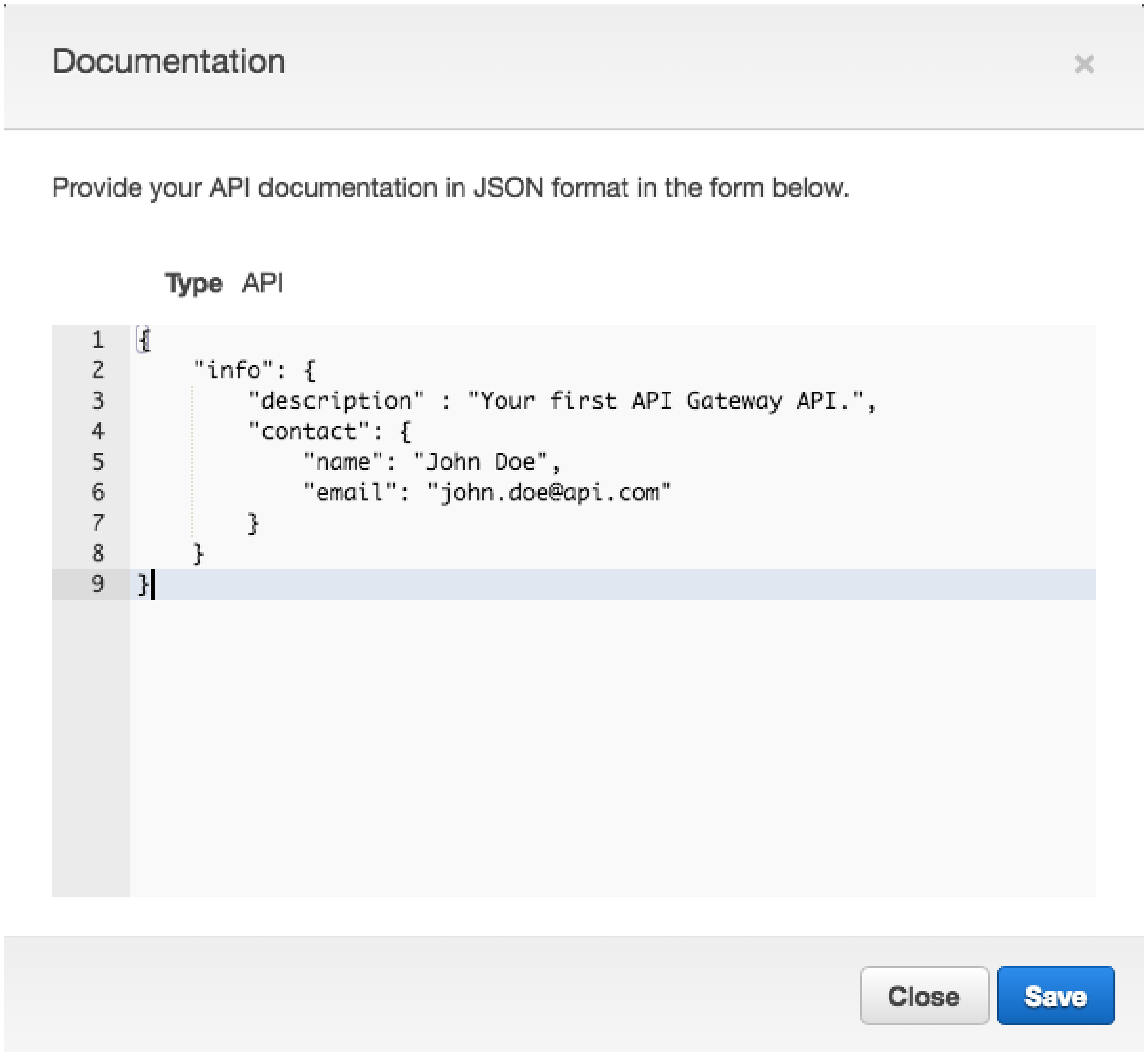
To add a documentation part for the `API` entity, choose **Resources** from the **PetStore** API. Choose the **Actions** → **Edit API Documentation** menu item.



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.

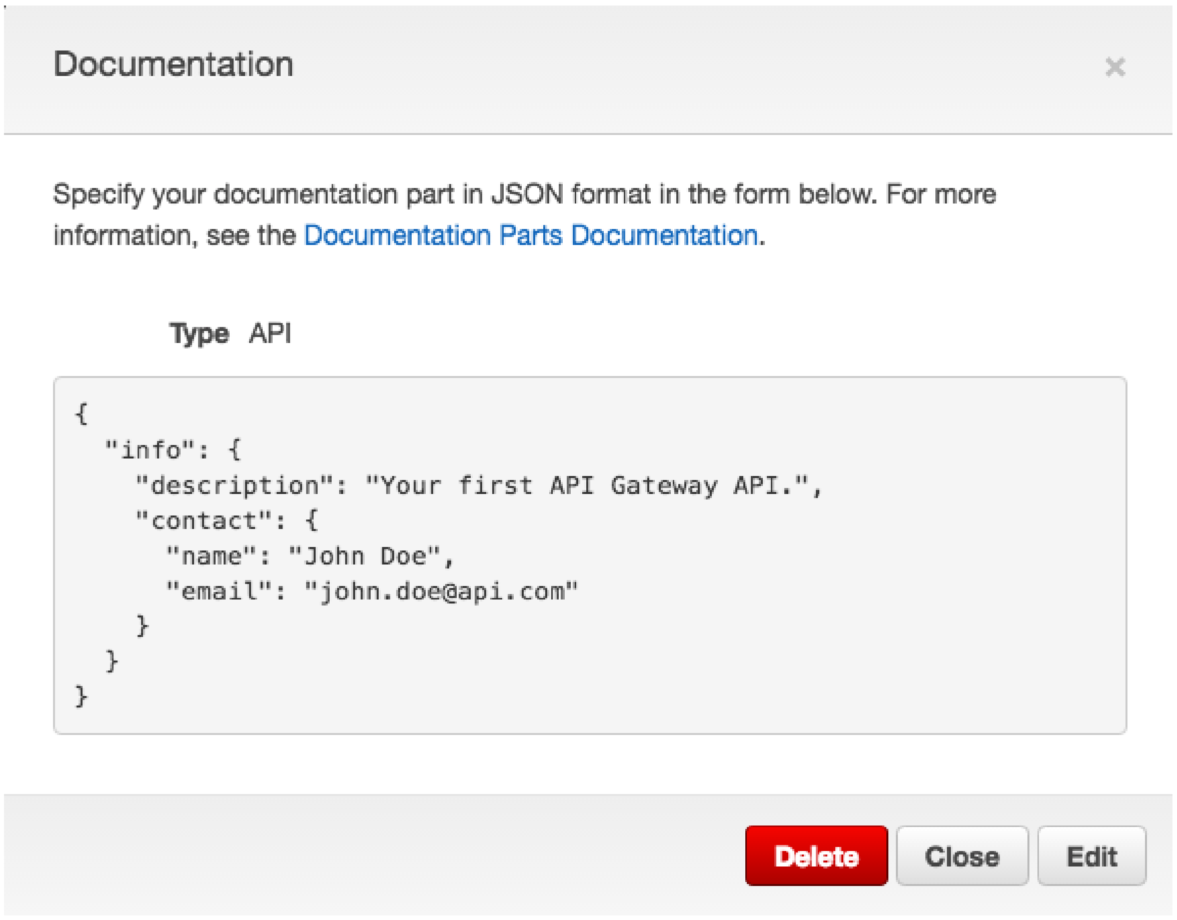


Provide your API documentation in JSON format in the form below.

Type API

```
{
    "info": {
        "description" : "Your first API Gateway API.",
        "contact": {
            "name": "John Doe",
            "email": "john.doe@api.com"
        }
    }
}
```

If a documentation part has already been created, you first get the `properties` map viewer, as shown in the following.



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"
    }
  }
}
```

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 the 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"
}
```

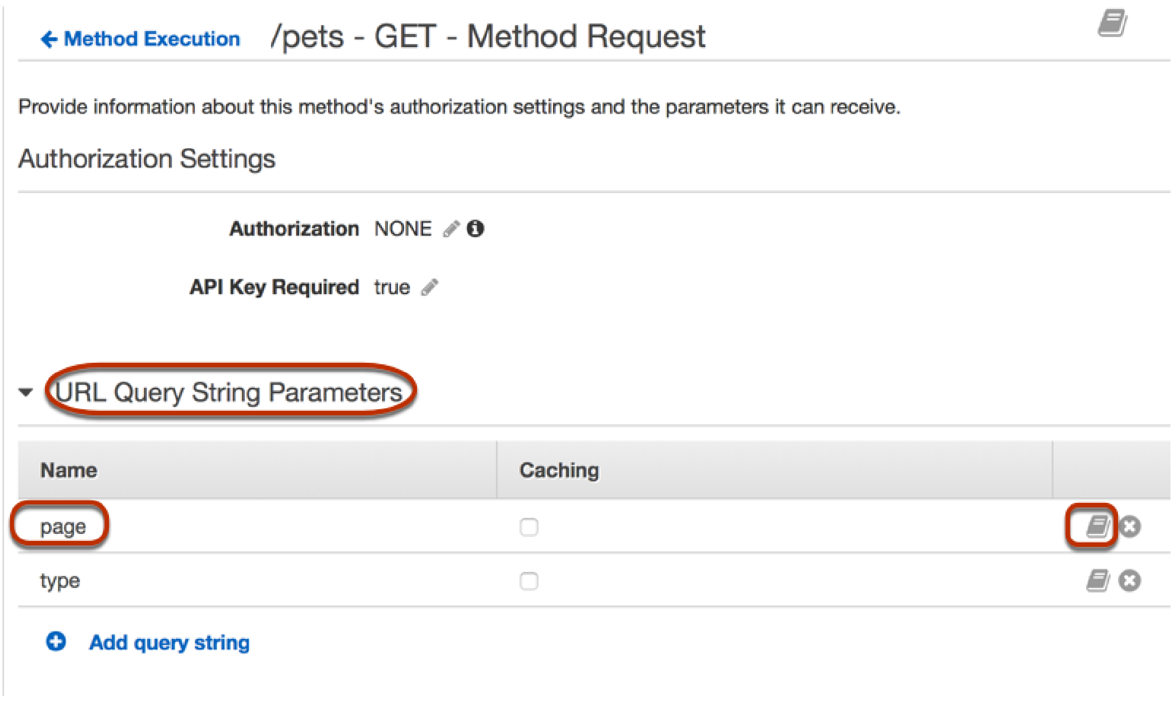
For the existing documentation, choose **Edit** from the **Documentation** viewer. Edit the documentation content in the **Documentation** editor and choose **Save**. Choose **Close**.

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.



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

Authorization Settings

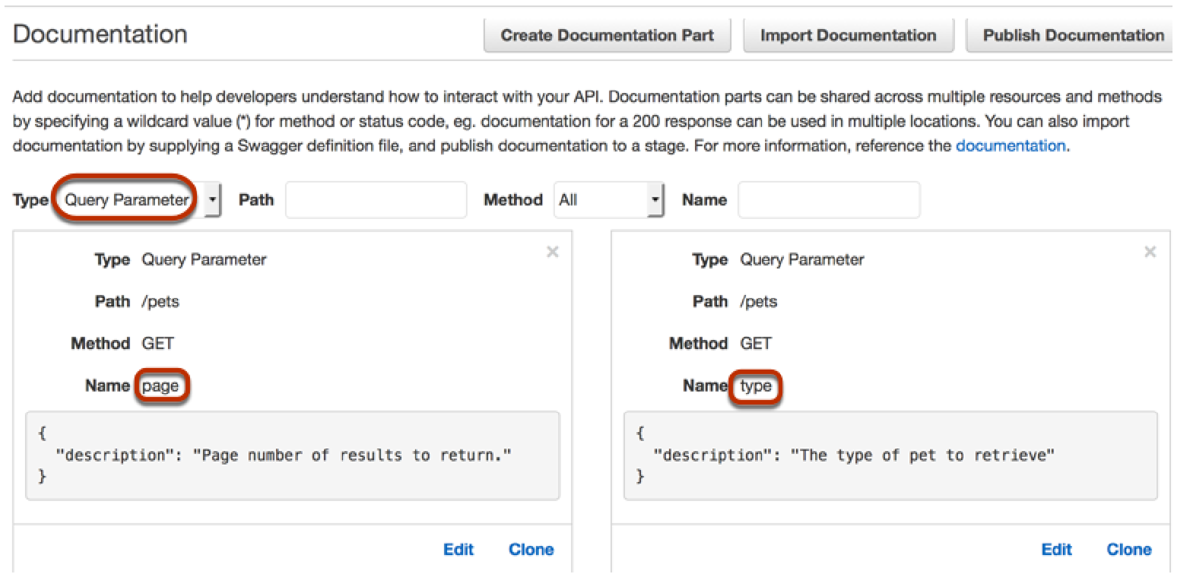
Authorization NONE

API Key Required true

URL Query String Parameters

Name	Caching
page	☐
type	☐

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.



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."
}
```

Type	Query Parameter
Path	/pets
Method	GET
Name	type

```
{
  "description": "The type of pet to retrieve"
}
```

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

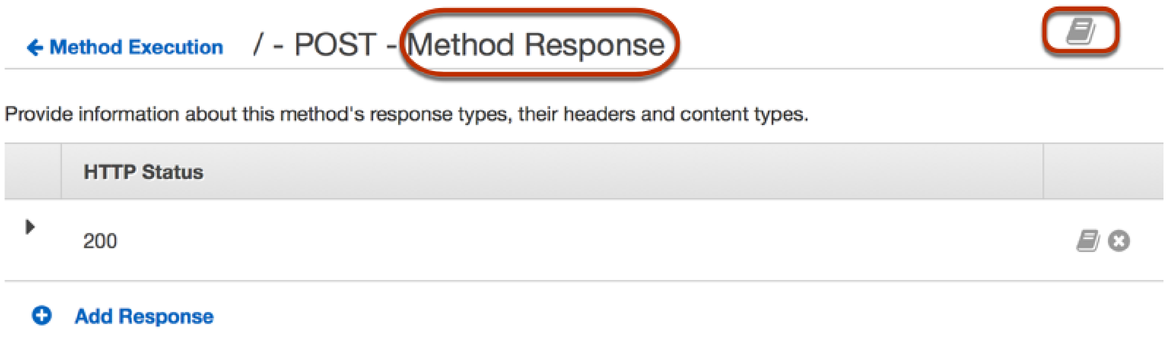
To add or edit documentation for a request body, go to **Method Request** for a method. Choose the book icon for **Request Body** to open the **Documentation** viewer and then editor. Add or modify the properties of the documentation part.

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.


← Method Execution
/ - POST - Method Response
Provide information about this method's response types, their headers and content types.
HTTP Status
200
Add Response

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 *`statusCode`*** 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 *`StatusCode`*** 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 `DocumentPart` 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. 9).

Topics

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: YYYYMMDDTttttttZ
Authorization: AWS4-HMAC-SHA256 Credential=access_key_id/YYYYMMDD/region/
apigateway/aws4_request, SignedHeaders=content-length;content-type;host;x-amz-date,
 Signature=sigv4_secret

{
    "location" : {
         "type" : "API"
    },
    "properties": "{\n\t\"info\": {\n\t\t\"description\" : \"Your first API with Amazon API
 Gateway.\"\n\t}\n}"
}
```

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/4wk1k4onj3/documentation/parts/p76vqo"
    },
    "documentationpart:delete": {
      "href": "/restapis/4wk1k4onj3/documentation/parts/p76vqo"
    },
    "documentationpart:update": {
      "href": "/restapis/4wk1k4onj3/documentation/parts/p76vqo"
    }
  },
  "id": "p76vqo",
  "location": {
    "path": "/",
    "method": null,
    "name": null,
    "statusCode": null,
    "type": "RESOURCE"
  },
  "properties": "{\n\t\"description\" : \"The PetStore root resource.\"\n}"
}
```

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: YYYYMMDDTttttttZ
Authorization: AWS4-HMAC-SHA256 Credential=access_key_id/YYYYMMDD/region/
apigateway/aws4_request, SignedHeaders=content-length;content-type;host;x-amz-date,
 Signature=sigv4_secret

{
    "location" : {
         "type" : "RESOURCE",
         "path" : "/pets"
    },
    "properties": "{\n\t\"description\" : \"A child resource under the root of PetStore.
\"\n}"
}
```

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}"
}
```

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: YYYYMMDDTttttttZ
Authorization: AWS4-HMAC-SHA256 Credential=access_key_id/YYYYMMDD/region/
apigateway/aws4_request, SignedHeaders=content-length;content-type;host;x-amz-date,
 Signature=sigv4_secret

{
    "location" : {
         "type" : "RESOURCE",
         "path" : "/pets/{petId}"
    },
    "properties": "{\n\t\"description\" : \"A child resource specified by the petId path
 parameter.\"\n}"
}
```

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: YYYYMMDDTttttttZ
Authorization: AWS4-HMAC-SHA256 Credential=access_key_id/YYYYMMDD/region/
apigateway/aws4_request, SignedHeaders=content-length;content-type;host;x-amz-date,
 Signature=sigv4_secret

{
    "location" : {
         "type" : "METHOD",
         "path" : "/pets",
         "method" : "GET"
    },
    "properties": "{\n\t\"summary\" : \"List all pets.\"\n}"
}
```

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}"
  } ]
}
```

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}"
}
```

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}"
}
```

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}"
}
```

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: YYYYMMDDTttttttZ
Authorization: AWS4-HMAC-SHA256 Credential=access_key_id/YYYYMMDD/region/
apigateway/aws4_request, SignedHeaders=content-length;content-type;host;x-amz-date,
 Signature=sigv4_secret

{
    "location" : {
         "type" : "PATH_PARAMETER",
         "path" : "/pets/{petId}",
         "method" : "*",
         "name" : "petId"
    },
    "properties": "{\n\t\"description\" : \"The id of the pet to retrieve.\"\n}"
}
```

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: YYYYMMDDTttttttZ
Authorization: AWS4-HMAC-SHA256 Credential=access_key_id/YYYYMMDD/region/
apigateway/aws4_request, SignedHeaders=content-length;content-type;host;x-amz-date,
 Signature=sigv4_secret
```

```
{
    "location" : {
         "type" : "REQUEST_BODY",
         "path" : "/pets",
         "method" : "POST"
    },
    "properties": "{\n\t\"description\" : \"A Pet object to be added to PetStore.\"\n}"
}
```

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/kgmfr1"
    },
    "documentationpart:delete": {
      "href": "/restapis/4wk1k4onj3/documentation/parts/kgmfr1"
    },
    "documentationpart:update": {
      "href": "/restapis/4wk1k4onj3/documentation/parts/kgmfr1"
    }
  },
  "id": "kgmfr1",
  "location": {
    "path": "/pets",
    "method": "POST",
    "name": null,
    "statusCode": null,
    "type": "REQUEST_BODY"
  },
  "properties": "{\n\t\"description\" : \"A Pet object to be added to PetStore.\"\n}"
}
```

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: YYYYMMDDTttttttZ
Authorization: AWS4-HMAC-SHA256 Credential=access_key_id/YYYYMMDD/region/
apigateway/aws4_request, SignedHeaders=content-length;content-type;host;x-amz-date,
 Signature=sigv4_secret

{
    "location" : {
         "type" : "REQUEST_HEADER",
         "path" : "/pets",
         "method" : "GET",
         "name" : "x-my-token"
    },
    "properties": "{\n\t\"description\" : \"A custom token used to authorization the method
 invocation.\"\n}"
}
```

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/h0m3uf"
    },
    "documentationpart:delete": {
      "href": "/restapis/4wk1k4onj3/documentation/parts/h0m3uf"
    },
    "documentationpart:update": {
      "href": "/restapis/4wk1k4onj3/documentation/parts/h0m3uf"
    }
  },
  "id": "h0m3uf",
  "location": {
    "path": "/pets",
    "method": "GET",
    "name": "x-my-token",
    "statusCode": null,
    "type": "REQUEST_HEADER"
  },
  "properties": "{\n\t\"description\" : \"A custom token used to authorization the method
 invocation.\"\n}"
}
```

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: YYYYMMDDTttttttZ
Authorization: AWS4-HMAC-SHA256 Credential=access_key_id/YYYYMMDD/region/
apigateway/aws4_request, SignedHeaders=content-length;content-type;host;x-amz-date,
 Signature=sigv4_secret

{
    "location": {
      "path": "/",
      "method": "*",
      "name": null,
      "statusCode": "200",
      "type": "RESPONSE"
    },
    "properties": "{\n  \"description\" : \"Successful operation.\"\n}"
}
```

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: YYYYMMDDTttttttZ
Authorization: AWS4-HMAC-SHA256 Credential=access_key_id/YYYYMMDD/region/
apigateway/aws4_request, SignedHeaders=content-length;content-type;host;x-amz-date,
 Signature=sigv4_secret

  "location": {
    "path": "/",
    "method": "GET",
    "name": "Content-Type",
    "statusCode": "200",
    "type": "RESPONSE_HEADER"
  },
  "properties": "{\n  \"description\" : \"Media type of request\"\n}"
```

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: YYYYMMDDTttttttZ
Authorization: AWS4-HMAC-SHA256 Credential=access_key_id/YYYYMMDD/region/
apigateway/aws4_request, SignedHeaders=content-length;content-type;host;x-amz-date,
 Signature=sigv4_secret

{
    "location" : {
         "type" : "AUTHORIZER",
         "name" : "myAuthorizer"
    },
    "properties": "{\n\t\"description\" : \"Authorizes invocations of configured methods.
\"\n}"
}
```

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/pw3qw3"
    },
    "documentationpart:delete": {
      "href": "/restapis/4wk1k4onj3/documentation/parts/pw3qw3"
    },
    "documentationpart:update": {
      "href": "/restapis/4wk1k4onj3/documentation/parts/pw3qw3"
    }
  },
  "id": "pw3qw3",
  "location": {
    "path": null,
    "method": null,
    "name": "myAuthorizer",
    "statusCode": null,
    "type": "AUTHORIZER"
  },
  "properties": "{\n\t\"description\" : \"Authorizes invocations of configured methods.
\"\n}"
}
```

Note
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 `DocumentPart` 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: YYYYMMDDTttttttZ
Authorization: AWS4-HMAC-SHA256 Credential=access_key_id/YYYYMMDD/region/
apigateway/aws4_request, SignedHeaders=content-length;content-type;host;x-amz-date,
 Signature=sigv4_secret

{
    "location" : {
         "type" : "MODEL",
         "name" : "Pet"
    },
    "properties": "{\n\t\"description\" : \"Data structure of a Pet object.\"\n}"
}
```

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}"
}
```

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

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: 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
```

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 documenationpart: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: 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
```

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: 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
```

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. 306).

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. 362) 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: YYYYMMDDTttttttZ
Authorization: AWS4-HMAC-SHA256 Credential=access_key_id/YYYYMMDD/region/
apigateway/aws4_request, SignedHeaders=content-length;content-type;host;x-amz-date,
 Signature=sigv4_secret

{
  "swagger": "2.0",
  "info": {
    "description": "description",
    "version": "1",
    "title": "doc"
  },
  "host": "",
  "basePath": "/",
  "schemes": [
    "https"
  ],
  "paths": {
    "/": {
      "get": {
        "description": "Method description.",
        "produces": [
          "application/json"
        ],
        "responses": {
          "200": {
            "description": "200 response",
            "schema": {
              "$ref": "#/definitions/Empty"
            }
          }
        }
      }
    }
  },
  "definitions": {
    "Empty": {
      "type": "object",
      "title": "Empty Schema"
    }
  },
  "x-amazon-apigateway-documentation": {
    "version": "1.0.3",
    "documentationParts": [
      {
        "location": {
          "type": "API"
        },
        "properties": {
          "description": "API description",
          "info": {
            "description": "API info description 4",
            "version": "API info version 3"
          }
        }
      },
      {
        "location": {
          "type": "METHOD",
          "method": "GET"
        },
        "properties": {
          "description": "Method description."
        }
      },
      {
        "location": {
          "type": "MODEL",
          "name": "Empty"
        },
        "properties": {
          "title": "Empty Schema"
        }
      },
      {
        "location": {
          "type": "RESPONSE",
          "method": "GET",
          "statusCode": "200"
        },
        "properties": {
          "description": "200 response"
        }
      }
    ]
  }
}
```

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. 208).
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": "StmtDocPartsAddEditViewDelete",
      "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 Who Can Create and Manage an API Gateway API with IAM Policies (p. 216).

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. 9).

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. 391)
```

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. 391)
```

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. 391)
```

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.

Control 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

Control Access to API Gateway with IAM Permissions

You control access to Amazon API Gateway with IAM permissions by controlling access to the following two API Gateway component processes:

- To create, deploy, and manage an API in API Gateway, you must grant the API developer permissions to perform the required actions supported by the API management component of API Gateway.
- To call a deployed API or to refresh the API caching, you must grant the API caller permissions to perform required IAM actions supported by the API execution component of API Gateway.

The access control for the two processes involves different permissions models, explained next.

API Gateway Permissions Model for Creating and Managing an API

To allow an API developer to create and manage an API in API Gateway, you must create IAM permissions policies that allow a specified API developer to create, update, deploy, view, or delete required API entities. You attach the permissions policy to an IAM user representing the developer, to an IAM group containing the user, or to an IAM role assumed by the user.

In this IAM policy document, the IAM `Resource` element contains a list of API Gateway API entities, including API Gateway resources and API Gateway link-relations. The IAM `Action` element contains the required API Gateway API-managing actions. These actions are declared in the `apigateway:`*`HTTP_VERB`* format, where `apigateway` designates the underlying API management component of API Gateway, and *`HTTP_VERB`* represents HTTP verbs supported by API Gateway.

For more information on how to use this permissions model, see Control Access for Managing an API (p. 216).

API Gateway Permissions Model for Invoking an API

To allow an API caller to invoke the API or refresh its caching, you must create IAM policies that permit a specified API caller to invoke the API method for which the IAM user authentication is enabled. The API developer sets the method's `authorizationType` property to `AWS_IAM` to require that the caller submit the IAM user's access keys to be authenticated. Then, you attach the policy to an IAM user representing the API caller, to an IAM group containing the user, or to an IAM role assumed by the user.

In this IAM permissions policy statement, the IAM `Resource` element contains a list of deployed API methods identified by given HTTP verbs and API Gateway resource paths. The IAM `Action` element contains the required API Gateway API executing actions. These actions include `execute-api:Invoke` or `execute-api:InvalidateCache`, where `execute-api` designates the underlying API execution component of API Gateway.

For more information on how to use this permissions model, see Control Access for Invoking an API (p. 219).

When an API is integrated with an AWS service (for example, AWS Lambda) in the back end, API Gateway must also have permissions to access integrated AWS resources (for example, invoking a Lambda function) on behalf of the API caller. To grant these permissions, create an IAM role of the Amazon API Gateway type. This role contains the following IAM trust policy that declares API Gateway as a trusted entity that is permitted to assume the role:

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

You must also attach to this role IAM permissions policies for calling integrated AWS services. For example, if the backend is Lambda, the IAM permissions policy must include the following permissions policy statement:

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

Note that Lambda supports resource-based access policy, which combines both trust and permissions policies. When integrating an API with a Lambda function using the API Gateway console, you are not asked to set this IAM role explicitly, because the console sets the resource-based permissions on the Lambda function for you, with your consent.

Note

To enact access control to an AWS service, you can use either the caller-based permissions model, where a permissions policy is directly attached to the caller's IAM user or group, or the role-based permission model, where a permissions policy is attached to an IAM role that API Gateway can assume. The permissions policies may differ in the two models. For example, the caller-based policy blocks the access while the role-based policy allows it. You can take advantage of this to require that an IAM user access an AWS service through an API Gateway API only.

Topics

Control Access for Managing an API

In this section, you will learn how to write up IAM policy statements to control who can or cannot create, deploy and update an API in API Gateway. You'll also find the policy statements reference, including the formats of the `Action` and `Resource` fields related to the API managing service.

Control Who Can Create and Manage an API Gateway API with IAM Policies

To control who can or cannot create, deploy and update your API using the API managing service of API Gateway, create an IAM policy document with required permissions as shown in the following policy template:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Permission",
      "Action": [
        "apigateway:HTTP_VERB"
      ],
      "Resource": [
        "arn:aws:apigateway:region::resource1-path",
        "arn:aws:apigateway:region::resource2-path",
        ...
      ]
    }
  ]
}
```

Here, *`Permission`* can be `Allow` or `Deny` to grant or revoke, respectively, the access rights as stipulated by the policy statement. For more information, see AWS IAM permissions.

HTTP_VERB can be any of the API Gateway-supported HTTP verbs (p. 218). `*` can be used to denote any of the HTTP verbs.

`Resource` contains a list of ARNs of the affected API entities, including RestApi, Resource, Method, Integration, DocumentationPart, Model, Authorizer, UsagePlan, etc. For more information, see Resource Format of Permissions for Managing API in API Gateway (p. 218).

By combining different policy statements, you can customize the access permissions for individual users, groups or roles to access selected API entities and to perform specified actions against those entities. For example, you can create the following policy to grant your documentation team the permissions to create, publish, update and delete the documentation parts of a specified API as well as to view the API entities.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "apigateway:GET"
      ],
      "Resource": [
        "arn:aws:apigateway:region::/restapis/api-id/*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "apigateway:POST",
        "apigateway:PATCH",
        "apigateway:DELETE"
      ],
      "Resource": [
        "arn:aws:apigateway:region::/restapis/api-id/documentation/*"
      ]
    }
  ]
}
```

For your API core development team who is responsible for all operations, you can create the following IAM policy to grant the team much broader access permissions.

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

Statement Reference of IAM Policies for Managing API in API Gateway

The following information describes the `Action` and `Resource` element format used in an IAM policy statement to grant or revoke permissions for managing API Gateway API entities.

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*.

Control Access for Invoking an API

In this section you will learn how to write up IAM policy statements to control who can or cannot call a deployed API in API Gateway. Here, you will also find the policy statement reference, including the formats of `Action` and `Resource` fields related to the API execution service.

Control Who Can Call an API Gateway API Method with IAM Policies

To control who can or cannot call a deployed API with IAM permissions, create an IAM policy document with required permissions. A template for such a policy document is shown as follows.

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

Here, *Permission* is to be replaced by `Allow` or `Deny` depending on whether you want to grant or revoke the included permissions. *Execution-operation* is to be replaced by the operations supported by the API execution service. *METHOD_HTTP_VERB* stands for a HTTP verb supported by the specified resources. *Resource-path* is the placeholder for the URL path of a deployed API Resource instance supporting the said *METHOD_HTTP-VERB*. For more information, see Statement Reference of IAM Policies for Executing API in API Gateway (p. 220).

> **Note**
> For IAM policies to be effective, you must have enabled IAM authentication on API methods by setting `AWS_IAM` for the methods' authorizationType property. Failing to do so will make these API methods effectively public accessible.
> When AWS identity and access management is enabled on a specific resource, IAM users from different AWS accounts cannot access that resource, because API Gateway does not currently support cross-account authentication.

For example, to grant a user the permission to view a list of pets exposed by a specified API, but to deny the user the permission to add a pet to the list, you could create the following policy statement:

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

For a developer team testing APIs, you can create the following policy statement to allow the team to call any method on any resource of any API by any developer in the `test` stage.

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

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 Managing 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 Who Can Create and Manage an API Gateway API with IAM Policies (p. 216).

Topics

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 /respais` 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 for Invoking an API (p. 219).

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/*/*/petstorewalkthrough/pets"
      ]
    }
  ]
}
```

Create and Attach a Policy to an IAM User

To enable a user to call the API managing service or the API execution service, you must create an IAM policy for an IAM user, which controls access to the API Gateway entities, and then attach the policy to the IAM user. The following steps describe how to create your IAM policy.

To create your own IAM policy

1. Sign in to the 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 is easy for you to refer to later. Optionally, type descriptive text in **Description**.
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 other statements 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 does not have permissions unless there is an explicit corresponding `Allow` statement.

6. To enable the policy for a user, choose **Users**.
7. Choose the IAM user to whom you want to attach the policy.

You have just created an IAM policy. It won't have any effect until you attach it to an IAM user, to an IAM group containing the user, or to an IAM role assumed by the user.

To attach an IAM policy to an IAM user

1. For the chosen user, choose the **Permissions** tab, and then choose **Attach Policy**.
2. Under **Grant permissions**, choose **Attach existing policies directly**.
3. Choose the policy document just created from the displayed list and then choose **Next: Review**.
4. Under **Permissions summary**, choose **Add permissions**.

Alternatively, you can add the user to an IAM group, if the user is not already a member, and attach the policy document to the group so that the attached policies are applicable to all group members. It is helpful to manage and update policy configurations on a group of IAM users. In the following, we highlight how to attach the policy to an IAM group, assuming that you have already created the group and added the user to the group.

To attach an IAM policy document to an IAM group

1. Choose **Groups** from the main navigation pane.
2. Choose the **Permissions** tab under the chosen group.
3. Choose **Attach policy**.
4. Choose the policy document that you previously created, and then choose **Attach policy**.

For API Gateway to call other AWS services on your behalf, create an IAM role of the **Amazon API Gateway** type.

To create an Amazon API Gateway type of role

1. Choose **Roles** from the main navigation pane.
2. Choose **Create New Role**.
3. Type a name for **Role name** and then choose **Next Step**.
4. Under **Select Role Type**, in **AWS Service Roles**, choose **Select** next to **Amazon API Gateway**.
5. Choose an available managed IAM permissions policy, for example, **AmazonAPIGatewayPushToCloudWatchLog** if you want API Gateway to log metrics in CloudWatch, under **Attach Policy** and then choose **Next Step**.
6. Under **Trusted Entities**, verify that **apigateway.amazonaws.com** is listed as an entry, and then choose **Create Role**.
7. In the newly created role, choose the **Permissions** tab and then choose **Attach Policy**.
8. Choose the previously created custom IAM policy document and then choose **Attach Policy**.

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:'`*`request-originating server addresses`*`'` header in at least its 200 response. You can replace the static value of specific *`request-originating server addresses`* with `*` to indicate any servers. However, you should be careful of enabling such a broad support and do so only when you fully understand the consequences.

With Lambda, AWS or HTTP integrations, you can leverage API Gateway to set up the required headers using the method response and integration response. For Lambda or HTTP proxy integrations (p. 111), you can still set up the required OPTIONS response headers in API Gateway. However, you must rely on the back end to return the `Access-Control-Allow-Origin` headers because the integration response is disabled for the proxy integration.

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 an API (p. 210).

Topics

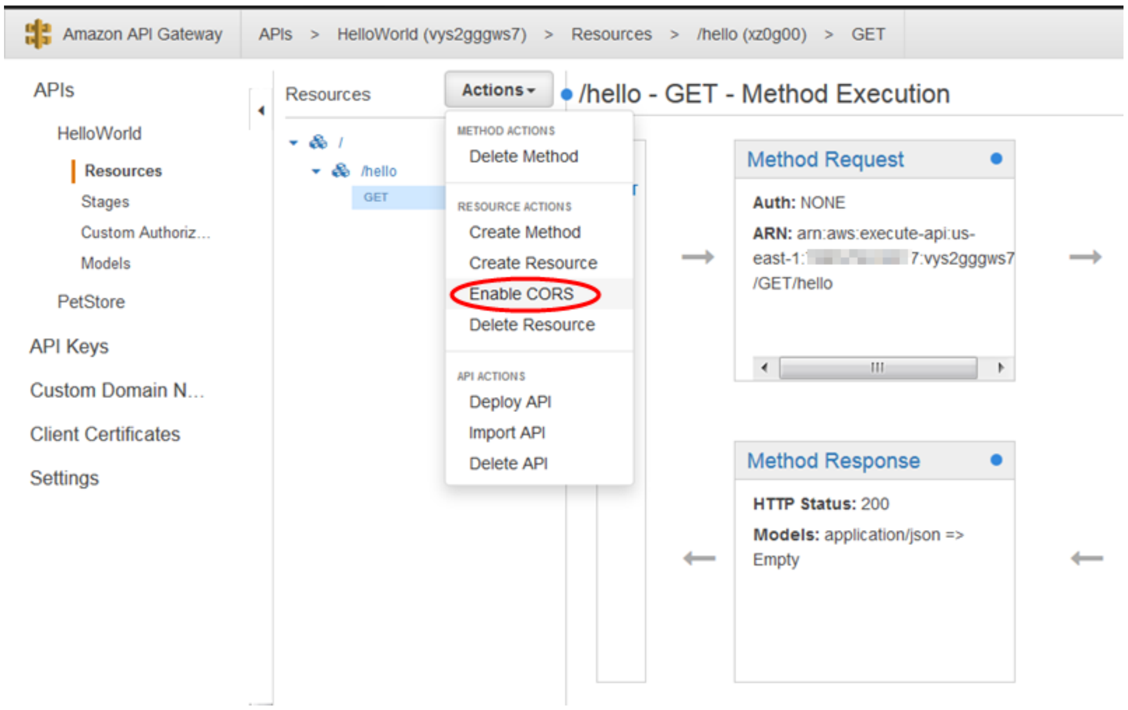
Prerequisites

- 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. 8).

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.



Auth: NONE

ARN: arn:aws:execute-api:us-east-1:[redacted]7:vys2gggws7/GET/hello

HTTP Status: 200

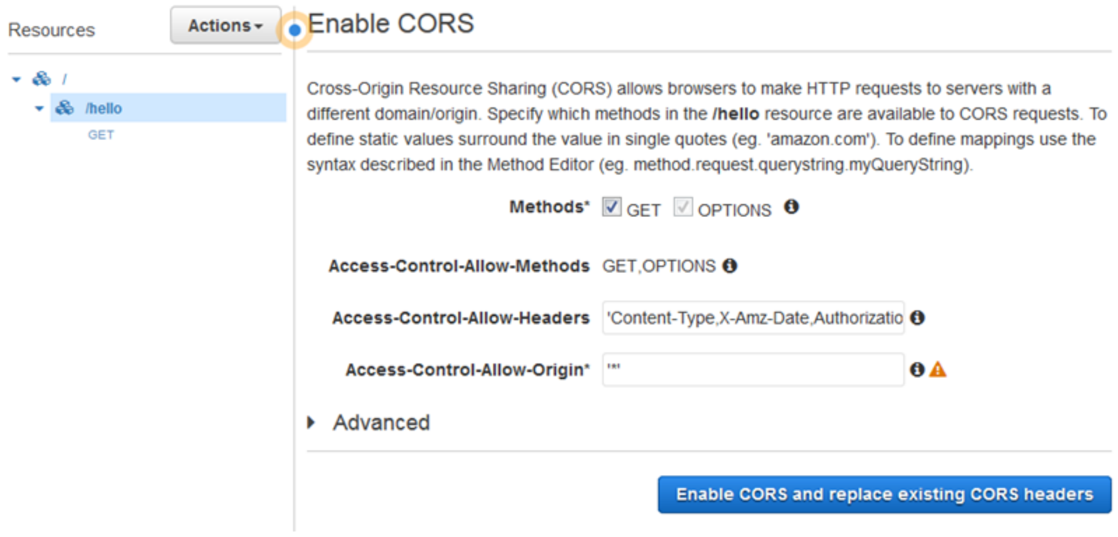
Models: application/json => Empty

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**.

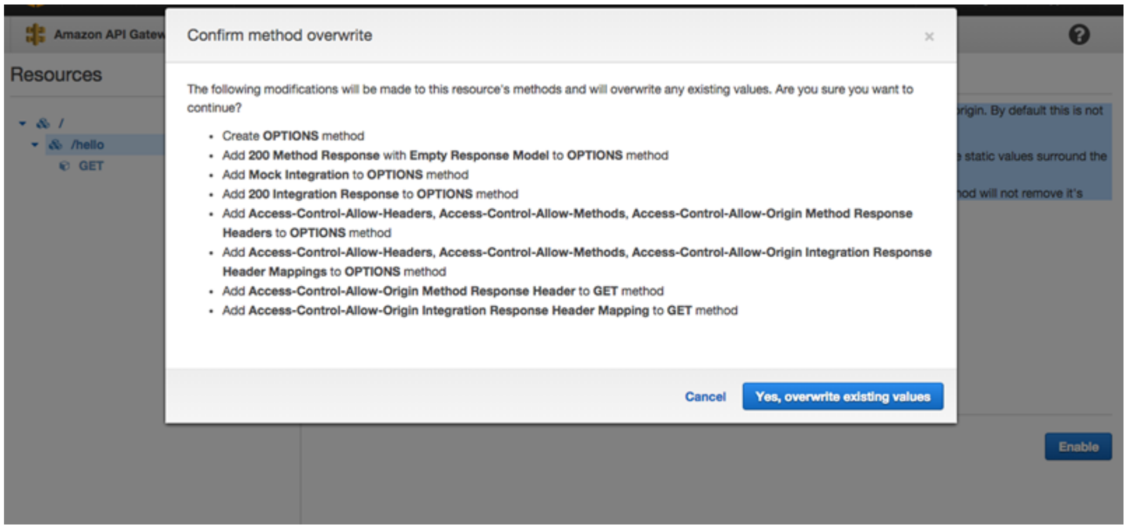


Cross-Origin Resource Sharing (CORS) allows browsers to make HTTP requests to servers with a different domain/origin. Specify which methods in the **/hello** resource are available to CORS requests. To define static values surround the value in single quotes (eg. 'amazon.com'). To define mappings use the syntax described in the Method Editor (eg. method.request.querystring.myQueryString).

Methods*	✓ GET ✓ OPTIONS
Access-Control-Allow-Methods	GET,OPTIONS
Access-Control-Allow-Headers	'Content-Type,X-Amz-Date,Authorizatio
Access-Control-Allow-Origin*	'*'

> **Note**
> When applying the above instructions to the `ANY` method in a proxy integration, any applicable CORS headers will not be set. Instead, you rely on the integration backend to return the applicable CORS headers, such as `Access-Control-Allow-Origin`

6. In **Confirm method changes**, choose **Yes, overwrite existing values** to confirm the new CORS settings.



Confirm method overwrite

The following modifications will be made to this resource's methods and will overwrite any existing values. Are you sure you want to continue?

- Create **OPTIONS** method
- Add **200 Method Response** with **Empty Response Model** to **OPTIONS** method
- Add **Mock Integration** to **OPTIONS** method
- Add **200 Integration Response** to **OPTIONS** method
- Add **Access-Control-Allow-Headers, Access-Control-Allow-Methods, Access-Control-Allow-Origin Method Response Headers** to **OPTIONS** method
- Add **Access-Control-Allow-Headers, Access-Control-Allow-Methods, Access-Control-Allow-Origin Integration Response Header Mappings** to **OPTIONS** method
- Add **Access-Control-Allow-Origin Method Response Header** to **GET** method
- Add **Access-Control-Allow-Origin Integration Response Header Mapping** to **GET** method

Cancel | Yes, overwrite existing values

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. 210), 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. 108).

```
/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 : "'*'"
```

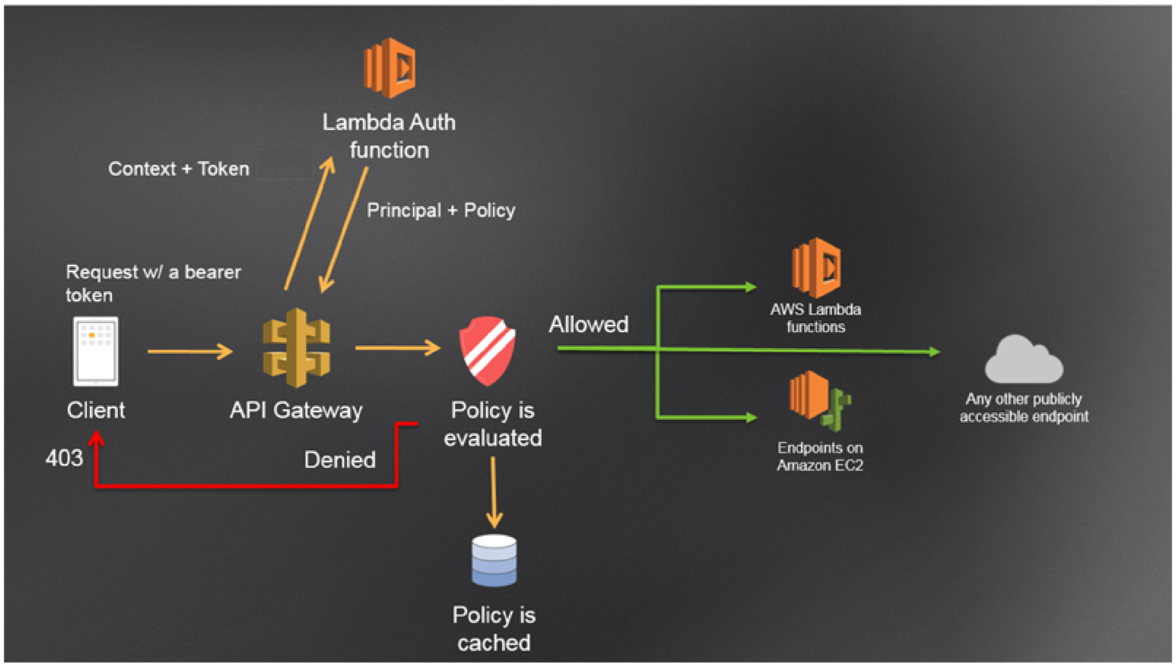
Use API Gateway Custom Authorizers

Topics

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, following this example in awslabs. 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 =  (event, context, callback) => {
    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.toLowerCase()) {
        case 'allow':
            callback(null, generatePolicy('user', 'Allow', event.methodArn));
            break;
        case 'deny':
            callback(null, generatePolicy('user', 'Deny', event.methodArn));
            break;
        case 'unauthorized':
            callback("Unauthorized");   // Return a 401 Unauthorized response
            break;
        default:
            callback("Error: Invalid token");
    }
};

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;
    }

    // Can optionally return a context object of your choosing.
    authResponse.context = {};
    authResponse.context.stringKey = "stringval";
    authResponse.context.numberKey = 123;
    authResponse.context.booleanKey = true;
    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 optionally return a key-value map named `context` to contain additional information. For more information, see Output from an Amazon API Gateway Custom Authorizer (p. 235).

You can then configure the integration request to pass returned `context` map entries to the backend. When the `context` map entries refer to cached credentials, the backend 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.

For the Lambda proxy integration, the `context` object returned from an custom authorizer is passed to the backend Lambda function as part of the input `event`. You can retrieve the `context` key-value pairs in the Lambda function by calling `$event.requestContext.authorizer.`*`key`*. For the preceding custom authorizer example, *`key`* is `stringKey`, `numberKey` or `booleanKey`. Their values are stringified, i.e., `"stringval"`, `"123"` or `"true"`, respectively.

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. 234) and Output from an Amazon API Gateway Custom Authorizer (p. 235).

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 preceeding 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": {
    "stringKey": "value",
    "numberKey": "1",
    "booleanKey": "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. 220).

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 backend. For more information, see Accessing the $context Variable (p. 153).

You can access the `stringKey`, `numberKey`, or `booleanKey` value (i.e., `"value"`, `"1"`, or `"true"`) of the `context` map in a mapping template by calling `$context.authorizer.stringKey`, `$context.authorizer.numberKey`, or `$context.authorizer.booleanKey`, respectively. The returned values are all stringified. Notice that you cannot set a JSON object or array as 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 (`ymy8tbxw7b`) 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:ymy8tbxw7b/*/GET/"
      }
    ]
  }
}
```

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. 378).
   - In **Identity token source**, type the mapping expression for your authorizer's custom header.

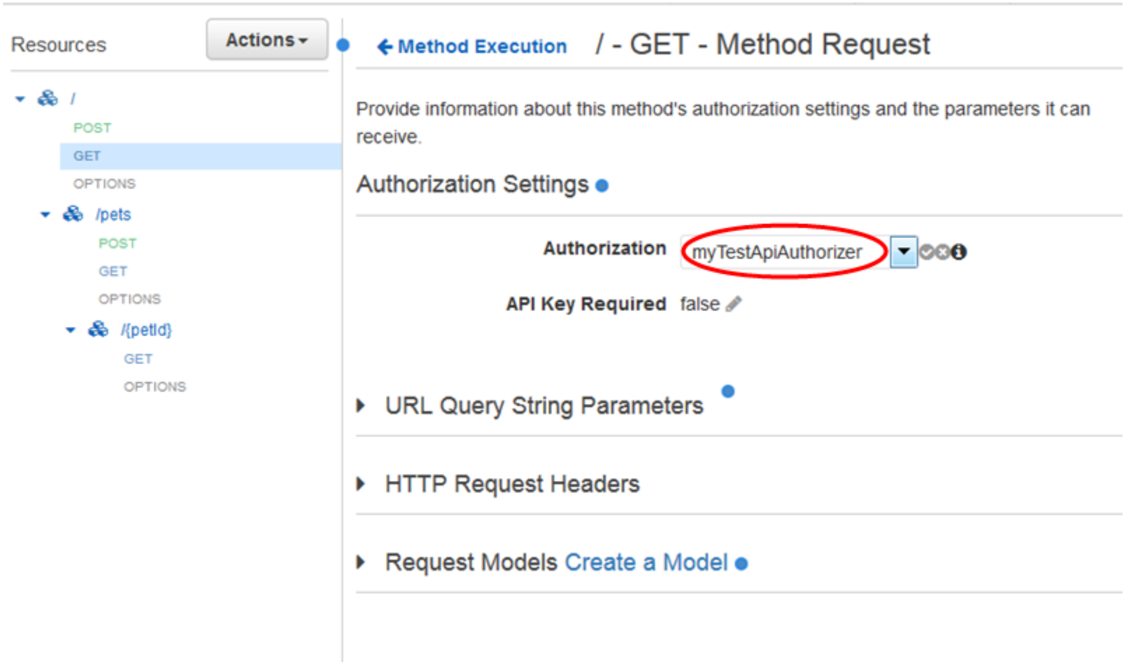
     Note
     The custom header mapping expression is of the `method.request.header.<name>` format, where *<name>* is the name of a custom authorization header submitted as part of the client request. In the following example, this custom header name is `Auth`.
   - In **Token validation expression**, you 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.
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.



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

Authorization Settings

Authorization myTestApiAuthorizer

API Key Required false

4. Optionally, while still on the **Method Request** page, choose **Add header** if you also want to pass the custom authorization header to the backend. 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

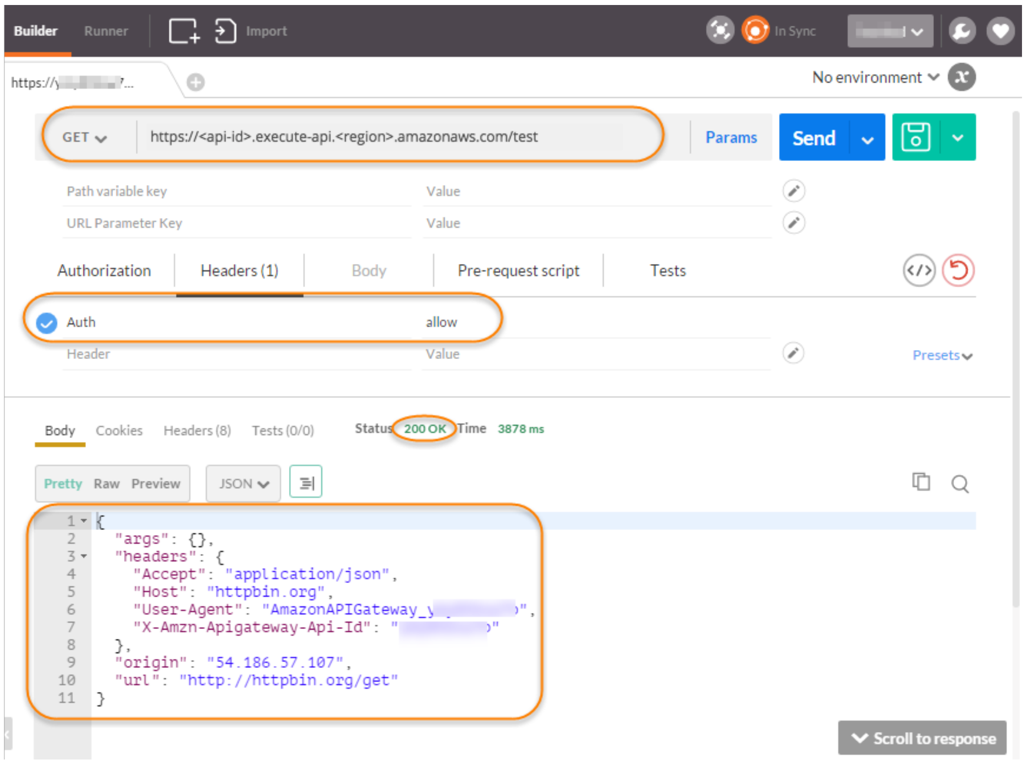
After you configure your API to use the custom authorizer, you or your customers can call the API using the custom authorizer. Because it involves submitting a custom authorization token header in the requests, you need a REST client that supports this. In the following examples, API calls are made using the Postman.

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**, choose the **GET** method and paste the API's **Invoke URL** into the adjacent URL field.

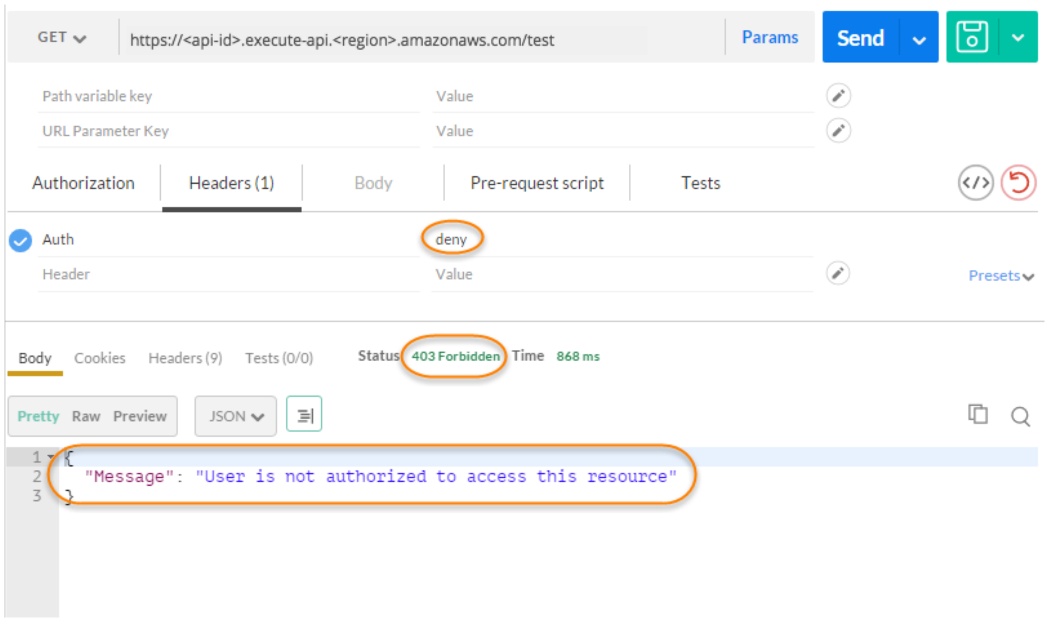
Add the custom authorization token header and set the value to `allow`. Choose **Send**.



```
{
  "args": {},
  "headers": {
    "Accept": "application/json",
    "Host": "httpbin.org",
    "User-Agent": "AmazonAPIGateway_[redacted]",
    "X-Amzn-Apigateway-Api-Id": "[redacted]"
  },
  "origin": "54.186.57.107",
  "url": "http://httpbin.org/get"
}
```

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.

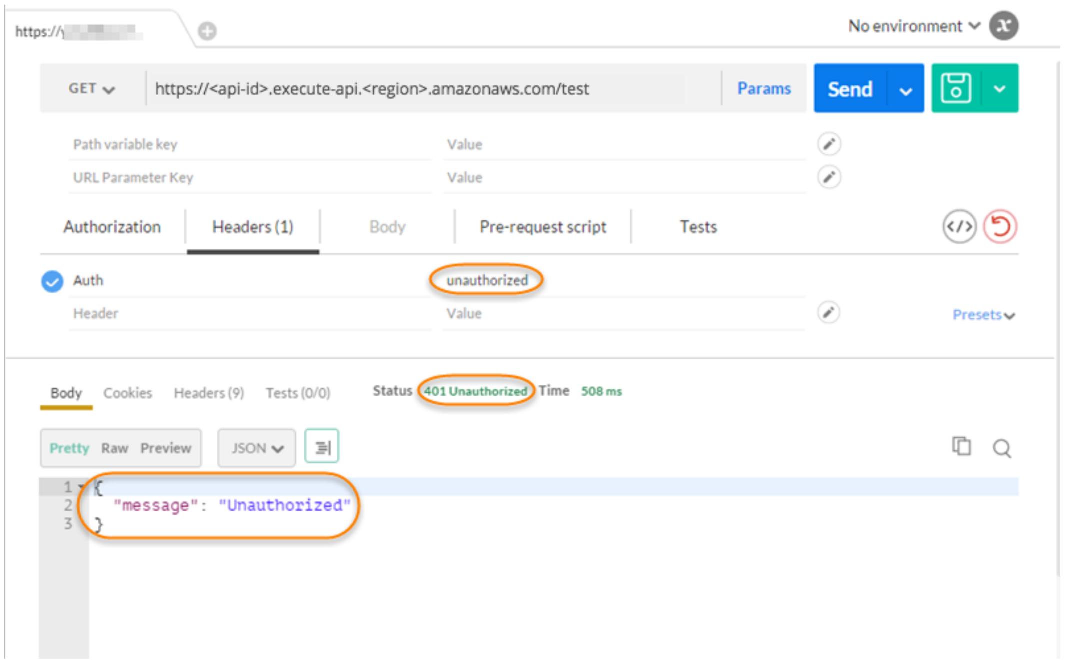
2. Still in Postman, change the custom authorization token header value to `deny`. Choose **Send**.



```
{
  "Message": "User is not authorized to access this resource"
}
```

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**.



https://<api-id>.execute-api.<region>.amazonaws.com/test

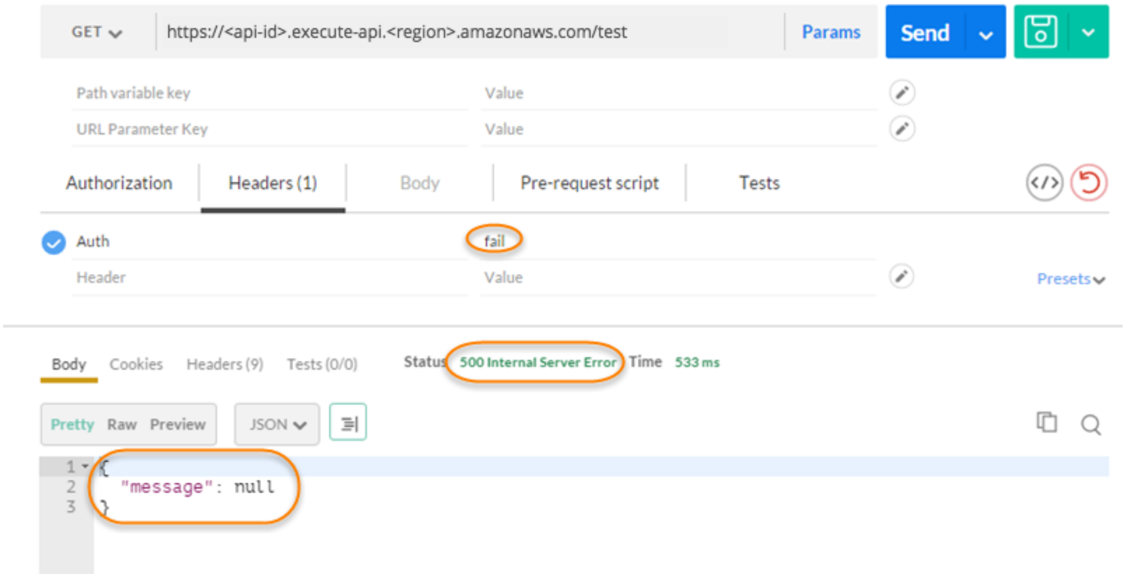
Auth unauthorized

Status 401 Unauthorized Time 508 ms

```
{
  "message": "Unauthorized"
}
```

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**.



https://<api-id>.execute-api.<region>.amazonaws.com/test

Auth fail

Status 500 Internal Server Error Time 533 ms

```
{
  "message": null
}
```

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 User Pools

In addition to using IAM roles and policies (p. 214) or custom authorizers (p. 232), 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.

Note
To get the permissions of an IAM role using Amazon Cognito credentials, use Amazon Cognito Federated Identities to get temporary credentials and set the authorization type to `AWS_IAM` on your API.

A user pool is integrated with an API as a method authorizer that is applicable for any method. 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 backend.

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

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 backend. 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 backend:

```
{
 "context" : {
  "sub" : "$context.authorizer.claims.sub",
  "email" : "$context.authorizer.claims.email"
 }
}
```

   If you have declared a custom claim field when configuring a 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 API Gateway Extensions to Swagger (p. 358).

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 User Pools and Using 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 Backend

You can use API Gateway to generate an SSL certificate and use its public key in the backend to verify that HTTP requests to your backend system are from API Gateway. This allows your HTTP backend to control and accept only requests originating from Amazon API Gateway, even if the back end is publicly accessible.

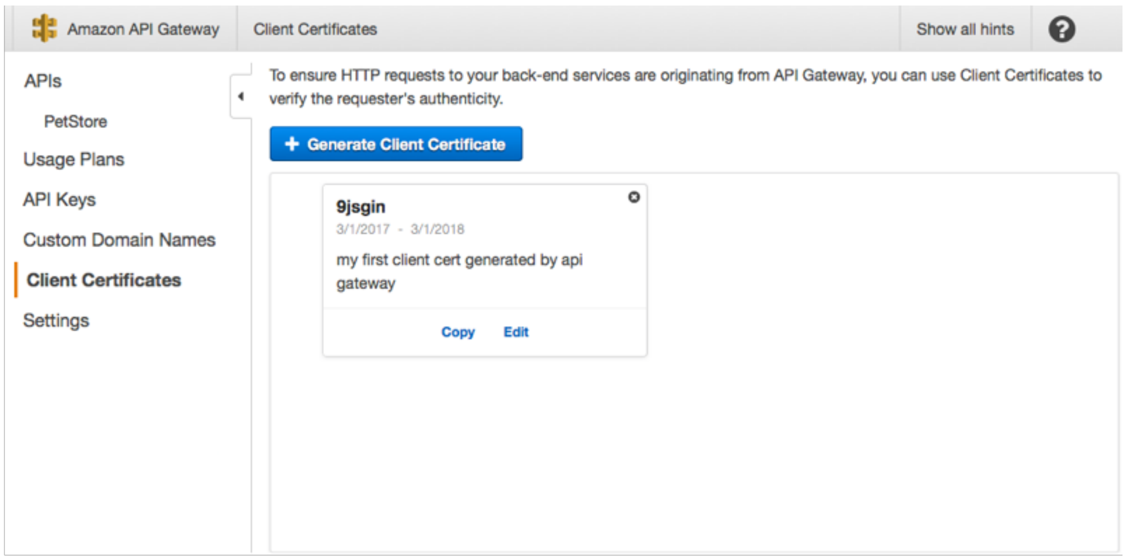
Note
Some backend servers may not support SSL client authentication as API Gateway does and could return an SSL certificate error. For a list of incompatible backends, see Known Issues (p. 453).

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

1. In the main navigation pane, choose **Client Certificates**.
2. From the **Client Certificates** pane, choose **Generate Client Certificate**.
3. Optionally, for **Edit**, choose to add a descriptive title for the generated certificate and choose **Save** to save the description. API Gateway generates a new certificate and returns the new certificate GUID, along with the PEM-encoded public key.



To ensure HTTP requests to your back-end services are originating from API Gateway, you can use Client Certificates to verify the requester's authenticity.

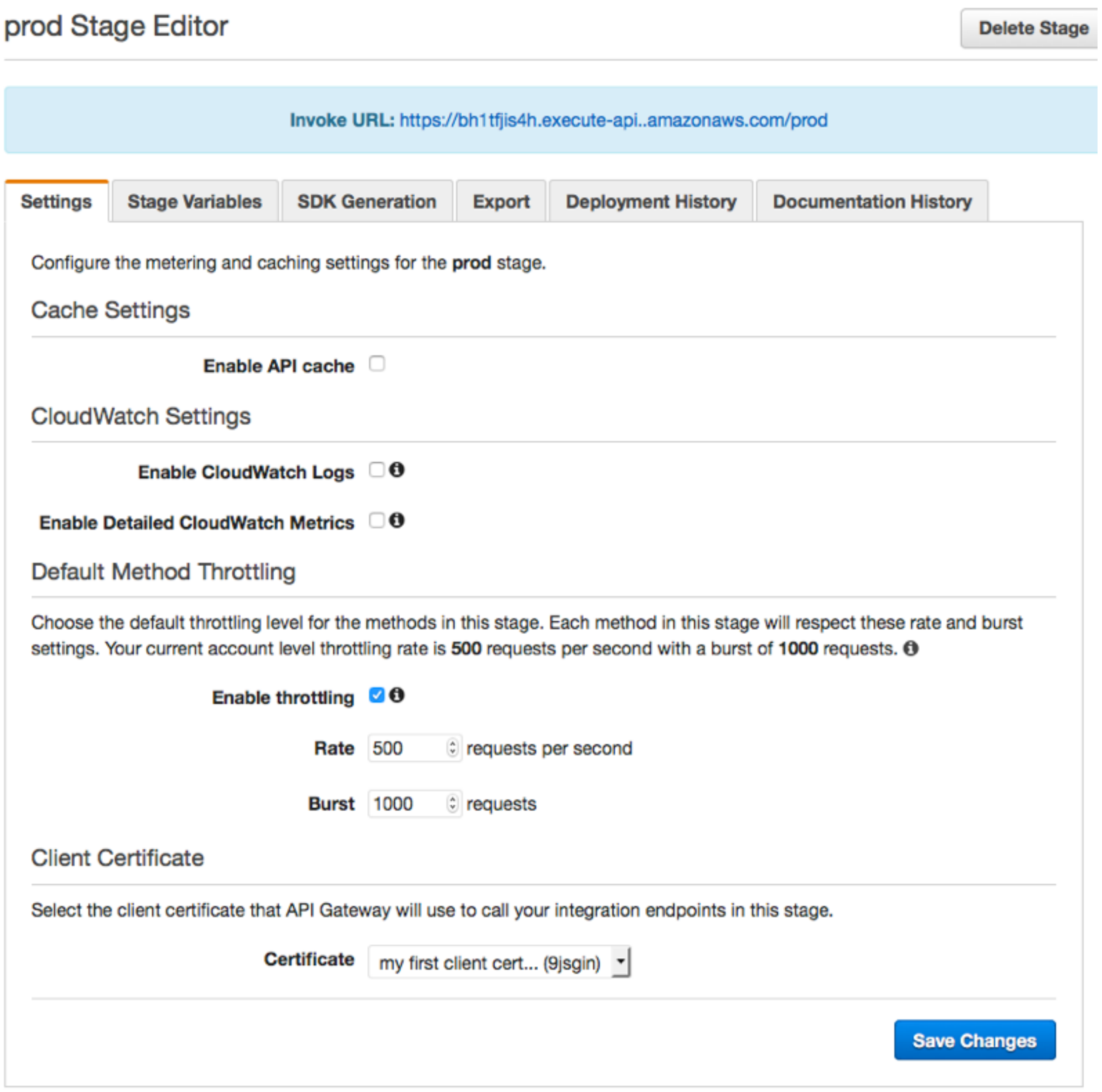
9jsgin
3/1/2017 - 3/1/2018
my first client cert generated by api gateway

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. 244).

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. To save the settings, choose **Save Changes**.



Invoke URL: https://bh1tfjis4h.execute-api..amazonaws.com/prod

Configure the metering and caching settings for the **prod** stage.

Cache Settings

Enable API cache ☐

CloudWatch Settings

Enable CloudWatch Logs ☐

Enable Detailed CloudWatch Metrics ☐

Default Method Throttling

Choose the default throttling level for the methods in this stage. Each method in this stage will respect these rate and burst settings. Your current account level throttling rate is **500** requests per second with a burst of **1000** requests.

Enable throttling ✓

Rate 500 requests per second

Burst 1000 requests

Client Certificate

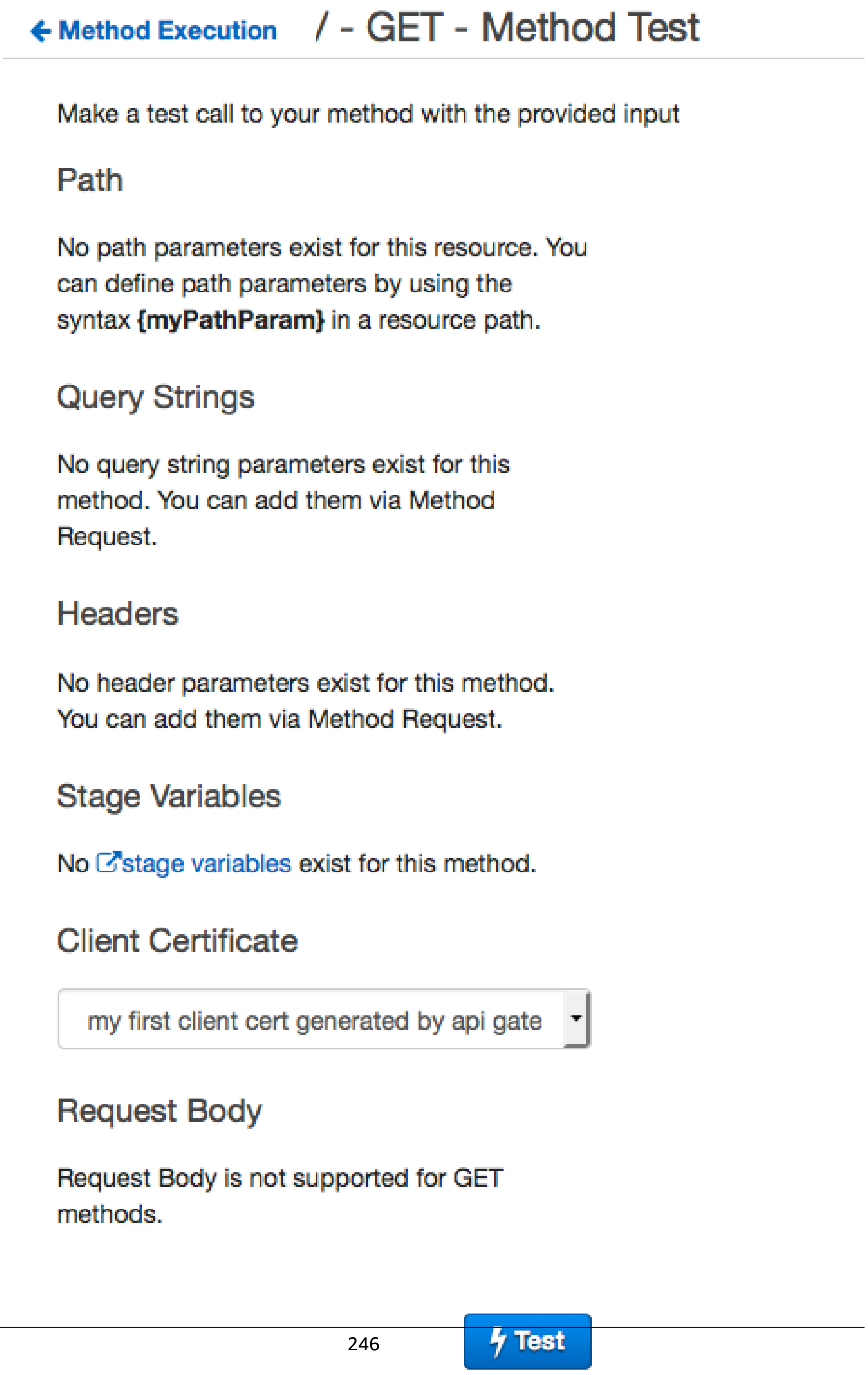
Select the client certificate that API Gateway will use to call your integration endpoints in this stage.

Certificate my first client cert... (9jsgin)

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.



/ - 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

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

my first client cert generated by api gate

Request Body

Request Body is not supported for GET methods.

API Gateway will present the chosen SSL certificate for the HTTP backend to authenticate the API.

Configure Backend to Authenticate API

These instructions assume you have already completed Generate a Client Certificate Using the API Gateway Console (p. 244) and Configure an API to Use SSL Certificates (p. 244).

When receiving HTTPS requests from API Gateway, your backend can authenticate your API using the PEM-encoded certificate generated by API Gateway, provided that the backend 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 backend 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:C4:40:8E
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:66:59:3C:50
Alias name: swisssignplatinumg2ca
  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:14:36
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:D0:52:35:46:95:A3
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:A1:2D
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:FF:3F
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:96:93:C0
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:FE:71
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:D1:22
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:F7:37:5D:73
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:FB:48
Alias name: chunghwaepkirootca
  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:DF:D5
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
  SHA256:
 3A:43:E2:20:FE:7F:3E:A9:65:3D:1E:21:74:2E:AC:2B:75:C2:0F:D8:98:03:05:BC:50:2C:AF:8C:2D:9B:41:A1
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:72:44
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:69:27
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:B3:48
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:FB:CB
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:C7:01:61
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:D5:98
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:A3:89
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:41:25:31:AE
Alias name: verisignc2g1.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:2C:E9
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:15:57
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:ED:50
Alias name: mozillacert119.pem
  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:1B:9E
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:14:03:05
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:39:3D:A3
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:D4:33:1C:99
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:B2:4D
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:A3:32
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:D6:93:55:59:6B
Alias name: equifaxsecureglobalebusinessca1
  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:CE:AE
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:E9:3B:9A
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:46:87
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:FB:37
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:E6:81:0A:36
Alias name: mozillacert84.pem
  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:02:FB
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:01:04
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:D8:81:21:37
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:D0:6C
Alias name: verisignc1g1.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:33:55
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:B6:58
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:B7:F5
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:A7:80:3B:E8
Alias name: swisssignsilverg2ca
  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:81:D5
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:C6:32:5C
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:83:86:0C
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:14:03:05
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:6B:4C
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:F7:37:5D:73
Alias name: soneraclass1ca
  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:32:31:3D
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:B5:35:A0
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:60:42:BB
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:D4:53:1F
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:17:05:25:52
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:B7:DF
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:D8:81:21:37
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:D0:60:16
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:B3:40:8E
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
  SHA256:
 5E:DB:7A:C4:3B:82:A0:6A:87:61:E8:D7:BE:49:79:EB:F2:61:1F:7D:D7:9B:F9:1C:1C:6B:56:6A:21:9E:D7:66
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:72:44
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:7A:F7
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:55:7D
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:E1:84
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:02:85
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:7F:79
Alias name: swisssigngoldg2ca
  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:C8:35:7B:95
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:C5:2A
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:84:88:82:14:23
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:8A:76
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:D9:27
Alias name: starfieldclass2ca
  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:B6:58
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:55:67:D0
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:03:12
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:1B:9E
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:BE:F6
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:80:BD
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:B7:DF
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:9B:7F
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:5F:A7
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:2F:F2
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:B1:18:CF
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:F1:78
Alias name: mozillacert65.pem
  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:73:01:9A
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:0E:1B
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:C4:40:8E
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:B9:12
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:FB:D4
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:B1:C1
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:F0:4C:07
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:54:6A:E4
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:F6:67
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:D3:4C
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:88:73
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:15:57
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:83:74:2E:CB
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:F3:39
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:A8:44:D2
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:C8:24:40:CE
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:FC:32
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:FB:48
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:C4:9E:86
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:52:2B:BD
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:C9:11:DC
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:DA:B5
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:89:32
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
  SHA256:
 8C:4E:DF:D0:43:48:F3:22:96:9E:7E:29:A4:CD:4D:CA:00:46:55:06:1C:16:E1:B0:76:42:2E:F3:42:AD:63:0E
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:E8:4B:3C
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:B3:48
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:96:93:C0
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:81:D5
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:D3:91:B4
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:54:6A:E4
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:E8:4B:3C
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:A9:19:8E:DA
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:B9:12
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:69:9E:F4
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:D9:2E
Alias name: mozillacert28.pem
  MD5:  5C:48:DC:F7:42:72:EC:56:94:6D:1C:CC:71:35:80:75
  SHA1: 66:31:BF:9E:F7:4F:9E:B6:C9:D5:A6:0C:BA:6A:BE:D1:F7:BD:EF:7B
  SHA256:
 0C:2C:D6:3D:F7:80:6F:A3:99:ED:E8:09:11:6B:57:5B:F8:79:89:F0:65:18:F9:80:8C:86:05:03:17:8B:AF:66
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:B4:1B
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:43:4A
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:E9:3B:9A
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:5F:A7
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:8E:4C
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:8C:40
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:AF:B2
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:40:EF
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:B7:F5
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:F2:3C
Alias name: buypassclass3ca
  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:B2:4D
Alias name: mozillacert57.pem
  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:CF:B2
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:A1:BB
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:A1:BB
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:AD:B0
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:D7:60:66
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:72:44
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:E9:39
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:54:6A:E4
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:96:05:61:79:15:06
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:A8:88:63:8B
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:99:02:6B:6C
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:33:CF
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:7F:79
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:E5:8F
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:E4:4E:70
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:68:0A:E3
Alias name: addtrustclass1ca
  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:E9:A7
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:B6:71:6F
Alias name: verisignc1g3.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:55:4B:65
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:E1:B3
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:66:AA
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:F1:78
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:8E:4C
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
  SHA256:
 51:3B:2C:EC:B8:10:D4:CD:E5:DD:85:39:1A:DF:C6:C2:DD:60:D8:7B:B7:36:D2:B5:21:48:4A:A4:7A:0E:BE:F6
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:BA:CA
Alias name: secomscrootca1
  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:99:02:6B:6C
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:E9:A7
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:88:73
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:B7:97:09:91
  SHA256:
 6C:61:DA:C3:A2:DE:F0:31:50:6B:E0:36:D2:A6:FE:40:19:94:FB:D1:3D:F9:C8:D4:66:59:92:74:C4:46:EC:98
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:1F:72
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:D1:04:38:41:13
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:B7:9B:85
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:B1:77
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:60:0D:19
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:F2:6A:EB
Alias name: mozillacert150.pem
  MD5:  C5:E6:7B:BF:06:D0:4F:43:ED:C4:7A:65:8A:FB:6B:19
  SHA1: 33:9B:6B:14:50:24:9B:55:7A:01:87:72:84:D9:E0:2F:C3:D2:D8:E9
  SHA256:
 EF:3C:B4:17:FC:8E:BF:6F:97:87:6C:9E:4E:CE:39:DE:1E:A5:FE:64:91:41:D1:02:8B:7D:11:C0:B2:29:8C:ED
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:10:55:31:6B
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:01:04
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:C1:70:D3
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:07:81:5F:F0
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:A9:19:8E:DA
Alias name: gtecybertrustglobalca
  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:E6:81:0A:36
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:B2:58:01:5D
Alias name: equifaxsecureebusinessca1
  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:F5:96
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:59:95
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:E1:8D:35
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:C4:9E:86
Alias name: soneraclass2ca
  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:D9:27
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:AD:C7
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:96:2B:24
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:C1:4D:17
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:A3:18:5A:E7
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:F2:6A:EB
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:C8:35:7B:95
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:E3:43:08:5C
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:1A:A2
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:BA:CA
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:7F:79
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:96:05:61:79:15:06
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:D8:14:36:24
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:B3:61:60:6A:C3
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:AC:69
Alias name: verisignc1g6.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:34:B9
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:A8:88:63:8B
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:42:16:12
Alias name: mozillacert78.pem
  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:B4:1B
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:E3:38:32:04:1A:A6
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:D8:14:36:24
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:C1:70:D3
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:D0:60:16
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
  SHA256: 
 06:87:26:03:31:A7:24:03:D9:09:F1:05:E6:9B:CF:0D:32:E1:BD:24:93:FF:C6:D9:20:6D:11:BC:D6:77:07:39
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:FB:37
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:C5:3B
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:C5:3B
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:B1:77
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:A9:19:8E:DA
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:80:BD
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:E2:58:61:67:E7
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:68:0A:E3
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:41:A1
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:D3:4C
Alias name: verisignclass1g3ca
  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:55:4B:65
Alias name: mozillacert102.pem
  MD5:  AA:C6:43:2C:5E:2D:CD:C4:34:C0:50:4F:11:02:4F:B6
  SHA1: 96:C9:1B:0B:95:B4:10:98:42:FA:D0:D8:22:79:FE:60:FA:B9:16:83
  SHA256:
 EE:C5:49:6B:98:8C:E9:86:25:B9:34:09:2E:EC:29:08:BE:D0:B0:F3:16:C2:D4:73:0C:84:EA:F1:F3:D3:48:81
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:2F:F2
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:A8:88:63:8B
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:14:03:05
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:DF:D5
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:C7:01:19:61
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:C6:32:5C
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:B8:84:2C
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:F8:07:03
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:BC:E9
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:E7:78:1A:9D
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:52:2B:BD
Alias name: mozillacert142.pem
  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:E1:8D:35
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:FB:9F
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:B7:AD:57:80:81:58:2E:02:07:97:A6:CB:DF:4F:FF:78:CE:83:96:B3:89:37:D7:F5
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:09:45:4D:E9
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:1A:A2
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:AC:44
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:D6:93:55:59:6B
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:B9:5B:0B
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:B3:61:60:6A:C3
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:D0:6C
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:FB:9F
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:9B:7F
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:09:EA
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:DA:B5
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:F6:73
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:A7:3A
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:CB:69
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:C7:01:61
Alias name: comodoaaaca
  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:69:9E:F4
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:F3:78
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:FB:CB
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:A7:3A
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:17:05:25:52
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
  SHA256:
 C4:70:CF:54:7E:23:02:B9:77:FB:29:DD:71:A8:9A:7B:6C:1F:60:77:7B:03:29:F5:60:17:F3:28:BF:4F:6B:E6
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:B1:18:CF
Alias name: amzninternalinfoseccag3
  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:D9:67:6D:68
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:AE:A7
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:A4:74
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:D4:33:1C:99
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:E2:58:61:67:E7
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:E8:4B:3C
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:38:41
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:D3:91:B4
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:A9:11
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:B7:DF
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:FB:D4
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:D7:66
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:CD:43
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:84:88:82:14:23
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:5E:38
```

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. 268). 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

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. 72) and Deploying an API (p. 284), respectively.

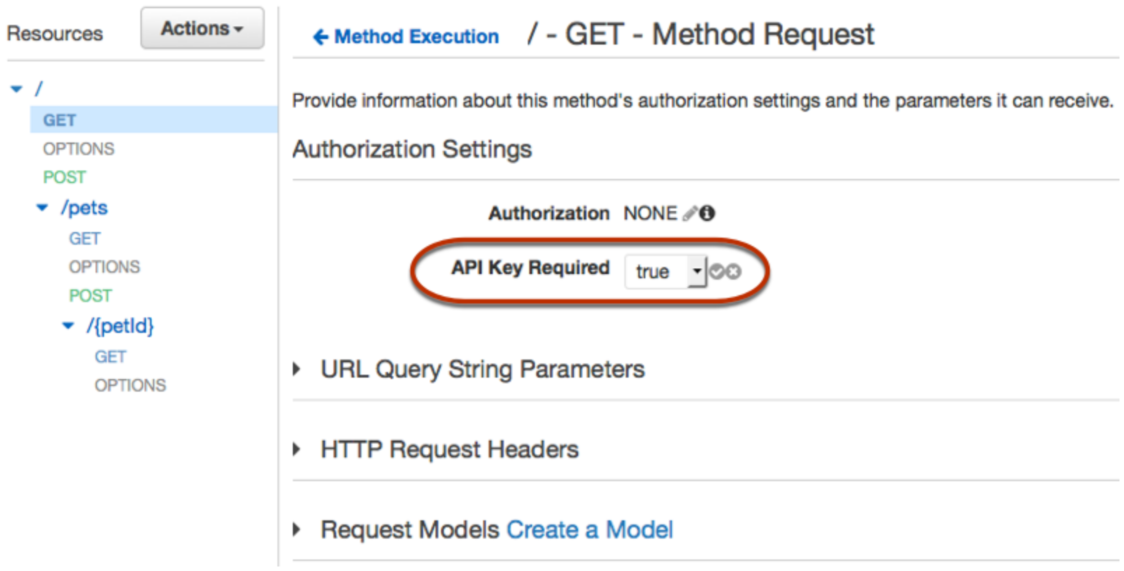
Topics

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.



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

Authorization NONE

API Key Required true

7. Deploy or redeploy the API for the requirement to take effect.

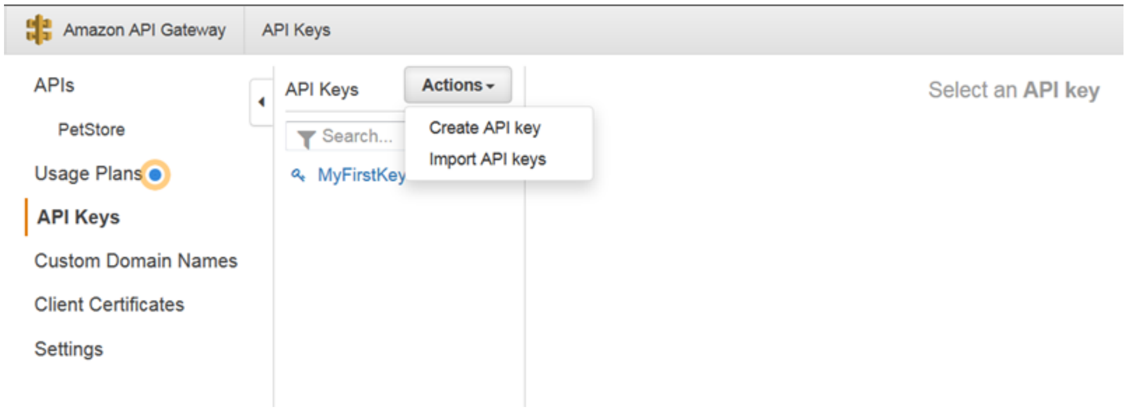
If the **API Key Required** option is set to `false` and you do not execute the steps above, any API key associated with an API stage will not used for the method.

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**.

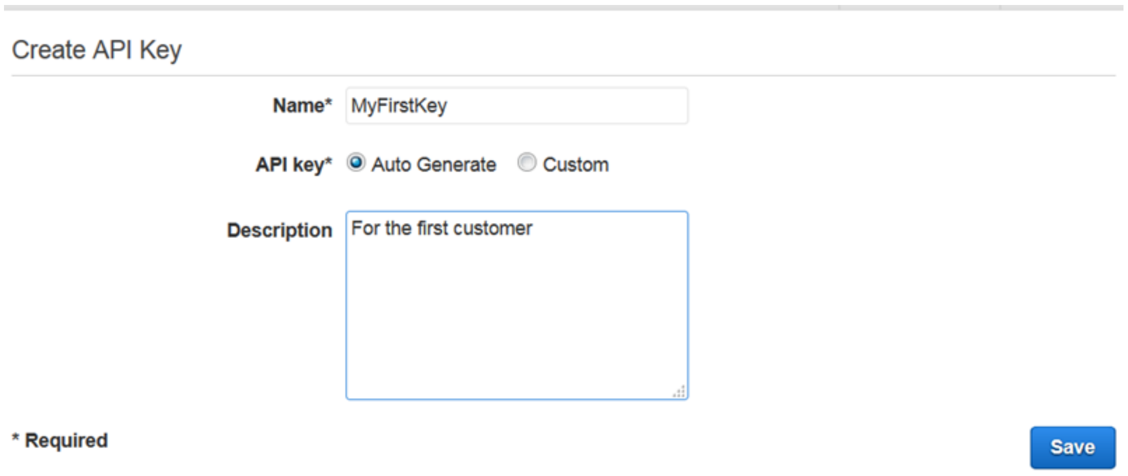


4. In **Create API Key**, do the following:

   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**.



Name* MyFirstKey

API key* [✓ Auto Generate] Custom

Description For the first customer

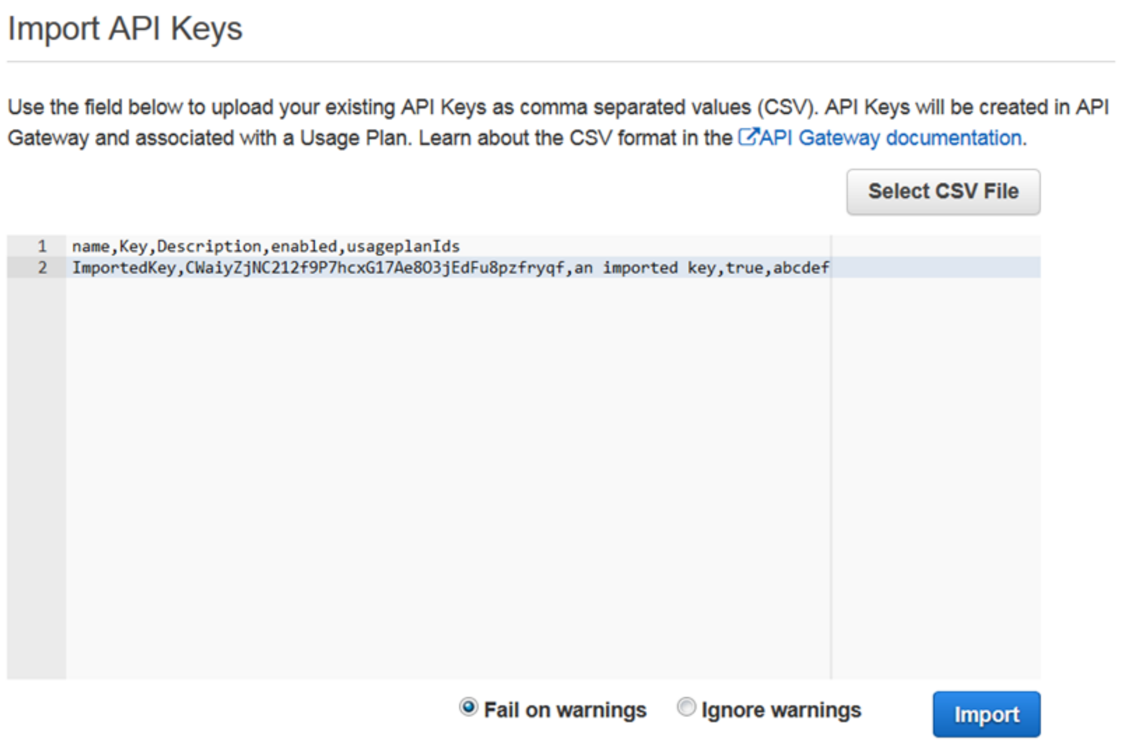
5. Repeat the preceding steps to create more API keys, if needed.

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. 277).


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.

```
name,Key,Description,enabled,usageplanIds
ImportedKey,CWaiyZjNC212f9P7hcxG17Ae803jEdFu8pzfryqf,an imported key,true,abcdef
```

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. 271).

Create, Configure, and Test 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. 268).

This section describes steps needed to create and use a usage plan using the API Gateway console.

Topics

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. 278).

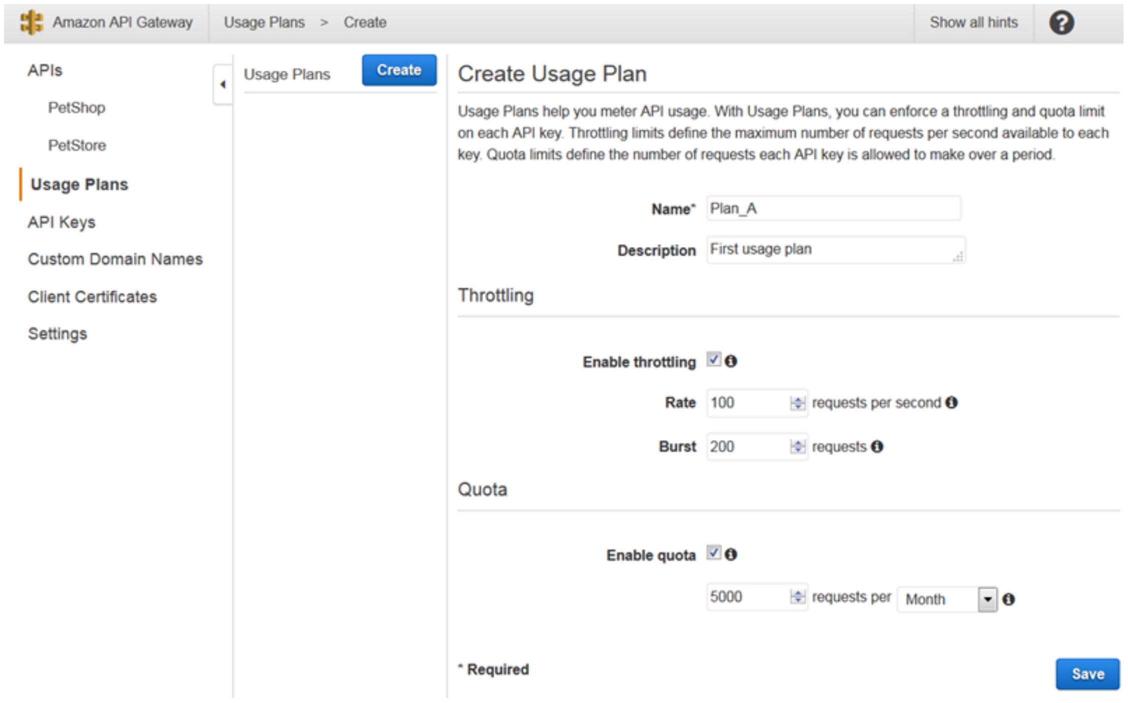
Create Usage Plans

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**.



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

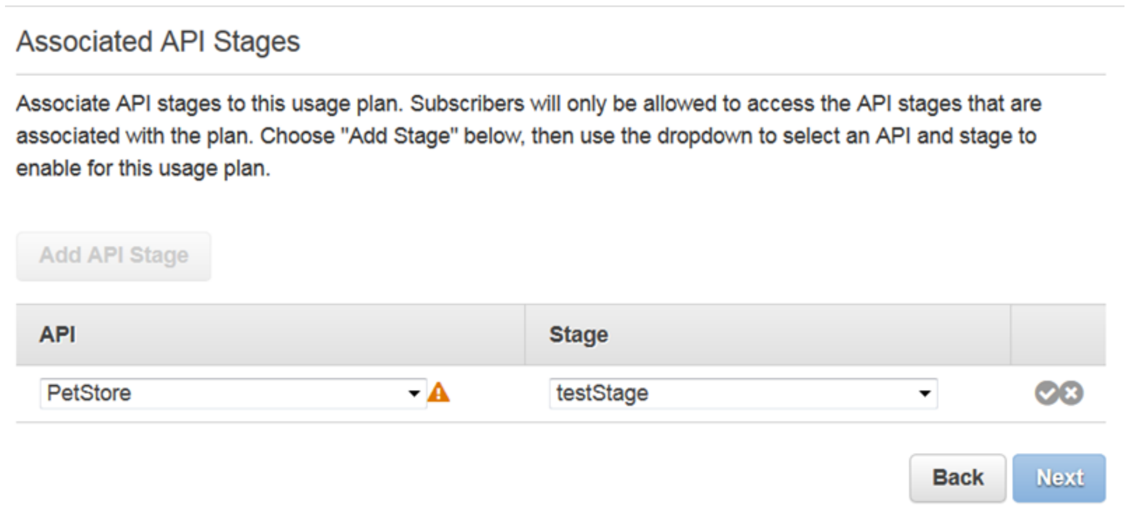
Enable quota ✓

5000 requests per Month

* Required

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**.



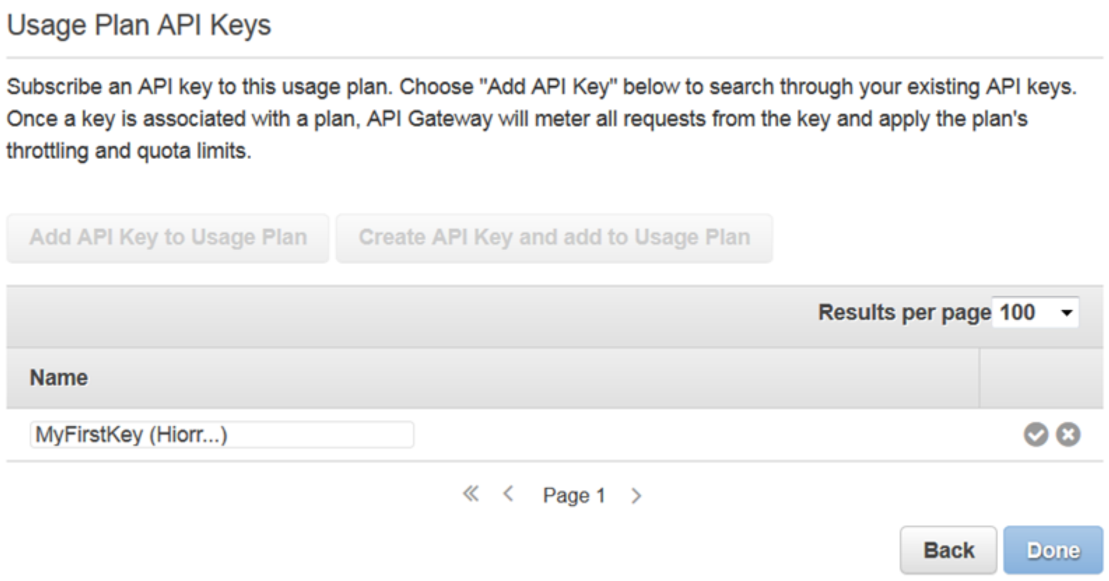
Associated API Stages

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.

API	Stage
PetStore	testStage

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.

Add API Key to Usage Plan
Create API Key and add to Usage Plan
Results per page 100

Name
MyFirstKey (Hiorr...)

   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. 277).

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

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. 277).

With the API key created, you can now proceed to Create, Configure, and Test Usage Plans Using the API Gateway REST API (p. 275).

Create, Configure, and Test Usage Plans Using 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. 275).

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

- Test Usage Plans (p. 277)

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. 271).

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. 9). 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., `xbvxlpijch`) 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 backend. 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.

  Occasionally, when an internal error occurs that makes API Gateway unable to enforce usage plan throttling limits or quotas for the request, API Gateway will serve the request without applying the throttling limits or quotas as specified in the usage plan, but will log an error message of `Usage Plan check failed due to an internal error` in your CloudWatch logs. You can ignore such occasional errors.

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
apikey1234abcdefghij0123456789,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,apikey1234abcdefghij0123456789,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,apikey1234abcdefghij0123456789,"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

Note
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, skip any further discussions of this topic, and follow the instructions in Use API Gateway Usage Plans (p. 267).

If 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. 214), custom authorizers (p. 232) or a Amazon Cognito User Pool (p. 239).

Topics

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. 72).
3. You must have deployed the API in API Gateway at least once. Follow the instructions in Deploying an API (p. 284).

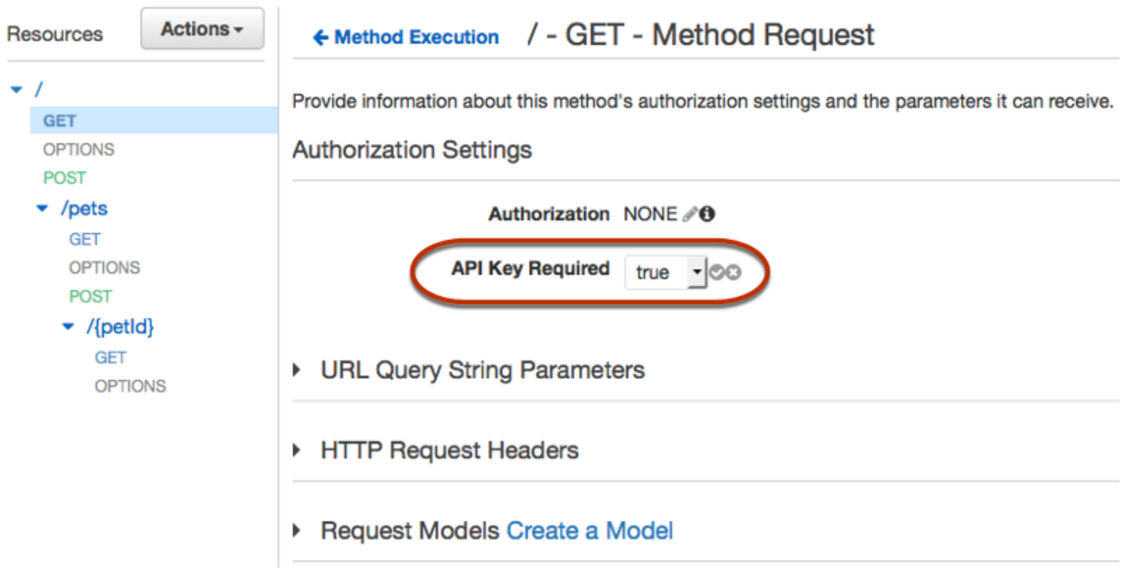
Use an API Key when Usage Plans Are Not Enabled

Note
The following instructions apply only if usage plans are not enabled for your account in the chosen region. To associate an API key with an API stage in a usage plan, follow the instructions in Use API Gateway Usage Plans (p. 267).

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.



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

Authorization NONE

API Key Required true

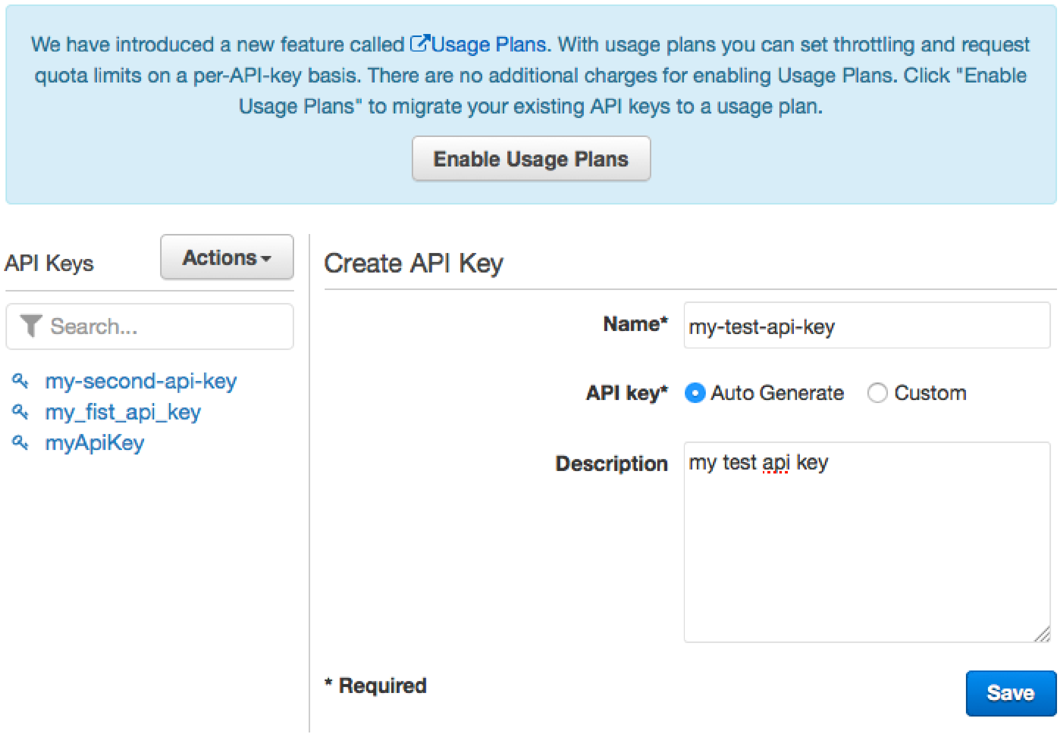
   Note that the steps above configure the API Gateway to enforce using API key on the method. Otherwise, the API key created following the instructions below will not be used for any of such calls.
3. Deploy or redeploy the API for the requirement to take effect.
4. Create an API key:

   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.

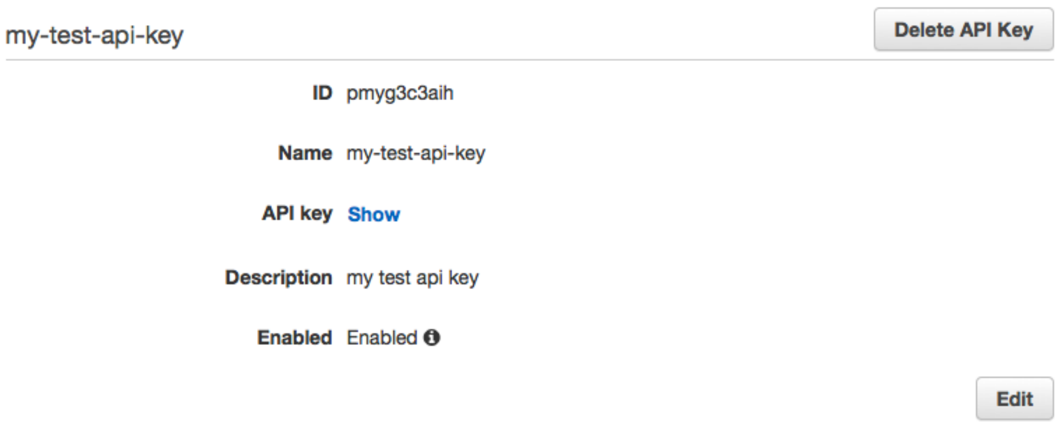
Name* my-test-api-key

API key* [✓ Auto Generate] Custom

Description my test api key

* Required

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.

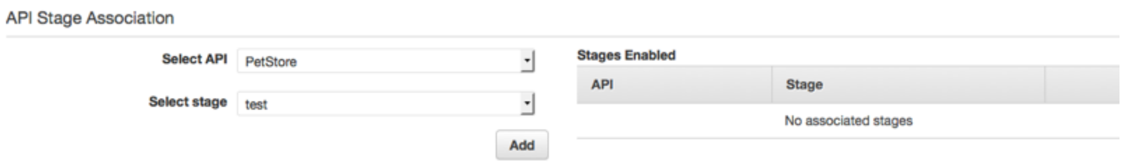


ID	pmyg3c3aih
Name	my-test-api-key
API key	Show
Description	my test api key
Enabled	Enabled

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

   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.



   Note
   The console UI elements shown above are not available if usage plans have 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 `hzYAVO9Sg98nsNh81M84O2kyXVy6K1xwHD8` 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: hzYAVO9Sg98nsNh81M84O2kyXVy6K1xwHD8
```

Maintain an API in Amazon API Gateway

Topics

- View a List of APIs in API Gateway (p. 281)
- Delete an API in API Gateway (p. 282)
- Delete a Resource in API Gateway (p. 282)
- View a Methods List in API Gateway (p. 282)
- Delete a Method in API Gateway (p. 283)

View a List of APIs in API Gateway

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

Topics

- Prerequisites (p. 281)
- View a List of APIs with the API Gateway Console (p. 282)

Prerequisites

- You must have an API available in API Gateway. Follow the instructions in Creating an API (p. 72).

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. 282)
- Delete an API with the API Gateway Console (p. 282)

Prerequisites

- You must have deployed the API at least once. Follow the instructions in Deploying an API (p. 284).

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. 283)

- View a Methods List with the API Gateway Console (p. 283)

Prerequisites

- You must have methods available in API Gateway. Follow the instructions in Build an API Gateway API to Expose an HTTP Endpoint (p. 8).

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 creating your API, you must deploy it to make the API callable for your users. An API deployment represents an API snapshot and becomes callable by the API users when it is associated with a stage. Deploying an API involves creating a deployment and stage and associating the deployment with the stage.

A stage prescribes a unique base URL (of the `https://{restapi-id}.execute-api.{region}amazonaws.com/{stageName}` format) for your users to call the associated API snapshot. Using different stage-deployment combinations, you can enable smooth and robust version control for the API.

For each stage, you can optimize the API performance by adjusting the default account-level request throttling limits and enabling API caching. You can also enable logging API calls to CloudTrail or CloudWatch and select a client certificate for the backend to authenticate the API requests. In addition, you can override stage-level settings for individual methods, define stage variables to pass stage-specific environment contexts to the API integration at run time. At an API stage, you can export the API definitions and generate an SDK for your users to call the API using a supported programming language.

Every time you update an API or its stage settings, you must redeploy the API by creating new snapshot of the API and associating the new deployment with a new or existing stage to make the changes available for your users.

To make an API's default base URL more user-friendly, you can create a custom domain name (e.g., `myapi.mydomain.com`) to replace the default host name of the API. With the custom domain name example, the API's base URL becomes `https://myapi.mydomain.com/stage`.

In this section, we discuss how to perform these deployment-related tasks, using the API Gateway console or calling the API Gateway REST API. To use other tools to do the same, see the documentation of, for example, AWS CLI or an AWS SDK.

To monetize your API deployment, you can leverage the API Gateway integration with AWS Marketplace to vend your API as a Software as a Service (SaaS) product. The instructions are also included in this chapter.

Topics

Deploy API from the API Gateway Console

You must have created an API before deploying it for the first time. For more information see Creating an API (p. 72).

Topics

Deploy an API to a Stage

The API Gateway console lets you deploy an API by creating a deployment and associating it with a new or existing stage.

Note
To associate a stage in API Gateway with a different deployment, see Associate a Stage with a Different Deployment (p. 286) instead.

1. Sign in to the API Gateway console at https://console.aws.amazon.com/apigateway.
2. In the **APIs** navigation pane, choose the API you want to deploy.
3. In the **Resources** navigation pane, choose **Actions**.
4. From the **Actions** drop-down menu, choose **Deploy API**.
5. In the **Deploy API** dialog, choose an entry from the **Deployment stage** dropdown list.
6. If you choose **[New Stage]**, type a name in **Stage name** and optionally provide a description for the stage and deployment in **Stage description** and **Deployment description**. If you choose an existing stage, you may want to provide a description of the new deployment in **Deployment description**.
7. Choose **Deploy** to deploy the API to the specified stage with default stage settings.

Update the Stage Configuration of a Deployment

After an API is deployed, you can modify the stage settings to enable or disable API cache, logging, or request throttling. You can also choose a client certificate for the backend to authenticate API Gateway and set stage variables to pass deployment context to the API integration at run time. For more information, see Update Stage Settings (p. 286).

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.

Set Stage Variables for the Deployment

For a deployment, you can set or modify stage variables to pass deployment-specific data to the API integration at run time. You can do this on the **Stage Variables** tab in the **Stage Editor**. For more information, see instructions in Set up Stage Variable for API Deployment (p. 295).

Associate a Stage with a Different Deployment

Because a deployment represents an API snapshot and a stage defines a path into a snapshot, you can choose different deployment-stage combinations to control how users call into different versions of the API. This is useful, for example, when you want to roll back API state to a previous deployment or to merge a 'private branch' of the API into the public one.

The following procedure shows how to do this using the **Stage Editor** in the API Gateway console. It is assumed that you must have deployed an API more than once.

1. If not already in **Stage Editor**, choose the stage you want to update the deployment from an API's **Stages** option in the **APIs** main navigation pane.
2. On the **Deployment History** tab, choose the option button next to the deployment you want the stage to use.
3. Choose **Change Deployment**.

Manage an API Stage

Topics

Associate an Existing Deployment with a New stage

After the initial deployment, you can add more stages and associate them with existing deployments. You can use the API Gateway console to create and use a new stage or choose an existing stage while deploying an API. In general, you can add a new stage to an API deployment before redeploying the API. To do so using the API Gateway console, follow the instructions below.

1. Sign in to the API Gateway console at https://console.aws.amazon.com/apigateway.
2. From the **APIs** navigation pane, choose **Stages** under an API.
3. From the **Stages** navigation pane, choose **Create**.
4. Under **Create Stage**, type a stage name, e.g., `prod`, for **Stage name**.
5. Optionally, type a stage description for **Stage description**
6. From the **Deployment** drop-down list, choose the date and time of the existing API deployment you want to associate with this stage.
7. Choose **Create**.

Update Stage Settings

After a successful deployment of an API, the stage is populated with default settings. You can use the console or API Gateway REST API to change the stage settings, including API caching and logging. In the following, we show how to do so using the **Stage Editor** of the API Gateway console.

Update Stage Settings Using the API Gateway Console

Make sure that you have already deployed the API and created a stage at least once before proceeding.

1. Sign in to the API Gateway console at https://console.aws.amazon.com/apigateway.
2. In the **APIs** pane, choose the API you want to update the stage settings, and then choose **Stages**.
3. In the **Stages** pane, choose the name of the stage.
4. In the **Stage Editor** pane, choose the **Settings** tab.
5. To enable API caching, select the **Enable API cache** option under the **Cache Settings** section. Then, choose desired options and associated values for **Cache capacity**, **Encrypt cache data**, **Cache time-to-live (TTL)**, as well as the requirements for per-key cache invalidation. For more information about the stage-level cache settings, see Enable API Caching (p. 291).

   Important
   By selecting this option, your AWS account may be charged for API caching.

   Tip
   You can override enabled stage-level cache settings. To do so, expand the stage under the **Stages** secondary navigation pane and choose a method. Then, in the stage editor, choose the **Override for this method** option 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 cache settings, see Enable API Caching (p. 291).

6. To enable Amazon CloudWatch Logs for all of the methods associated with this stage of this API Gateway API, do the following:

   a. Under the **CloudWatch Settings** section, select the **Enable CloudWatch Logs Logs** option.

      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.

   b. 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.
   c. To log full API call request and response information, select **Log full requests/responses data**. No sensitive data will be logged unless the **Log full requests/responses data** option is selected.
   d. To have API Gateway to report to CloudWatch the API metrics of `API calls`, `Latency`, `Integration latency`, `400 errors` and `500 errors`, select the **Enable Detailed CloudWatch Metrics** option. For more information about CloudWatch, see the Amazon CloudWatch User Guide.
   e. Choose **Save Changes**. The new settings will take effect after a new deployment.

      Important
      To 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 so, choose **Settings** from the **APIs** main navigation pane. Then type the ARN of an IAM role in the **CloudWatch log role ARN** text field. 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"
    }
  ]
}
```

For more information about CloudWatch, see the Amazon CloudWatch User Guide.

7. To set the stage-level throttle limit for all of the methods associated with this API, do the following in the **Default Method Throttling** section:

    a. For **Rate**, type the maximum number of stage-level steady-state requests per second that API Gateway can serve without returning a `429 Too Many Requests` response. This stage-level rate limit must not be more than the account-level (p. 289) rate limit as specified in API Gateway Limits for Configuring and Running an API (p. 450).

    b. For **Burst**, type the maximum number of stage-level concurrent requests that API Gateway can serve without returning a `429 Too Many Requests` response. This stage-level burst must not be more than the account-level (p. 289) burst limit as specified in API Gateway Limits for Configuring and Running an API (p. 450).

8. To override the stage-level throttling for individual methods, expand the stage under the **Stages** secondary navigation pane, choose a method of interest, and, back in the stage editor, choose **Override for this method** for **Settings**. In the **Default Method Throttling** area, select appropriate options.

Delete a Stage for an API

When you no longer need a stage, you can delete it to avoid paying for unused resources. In the following, we explain how to use the API Gateway console to delete a stage .

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, but you can recreate the stage and associate it with the same deployment.

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**.

Optimize API Performance

You can optimize API performance by throttling requests to prevent your API from being overwhelmed and by enabling API caching to enhance your API's responsiveness.

Topics

Throttle API Requests for Better Throughput

To prevent your API from being overwhelmed by too many requests, Amazon API Gateway throttles requests to your API using the token bucket algorithm, where a token counts for a request. Specifically, API Gateway sets a limit on a steady-state rate and a burst of request submissions against all APIs in your account. In the token bucket algorithm, the burst is the maximum bucket size.

When request submissions exceed the steady-state request rate and burst limits, API Gateway fails the limit-exceeding requests and returns `429 Too Many Requests` error responses to the client. Upon catching such exceptions, the client can resubmit the failed requests in a rate-limiting fashion, while complying with the API Gateway throttling limits.

As an API developer, you can set the limits for individual API stages or methods to improve overall performance across all APIs in your account. Alternatively, you can enable usage plans (p. 267) to restrict client request submissions to within specified request rates and quotas. This restricts the overall request submissions so that they don't go significantly past the account-level throttling limits.

Account-Level Throttling

By default, API Gateway limits the steady-state request rate to 10,000 requests per second (rps). It limits the burst (that is, the maximum bucket size) to 5,000 requests across all APIs within an AWS account. In API Gateway, the burst limit corresponds to the maximum number of concurrent request submissions that API Gateway can fulfill at any moment without returning `429 Too Many Requests` error responses.

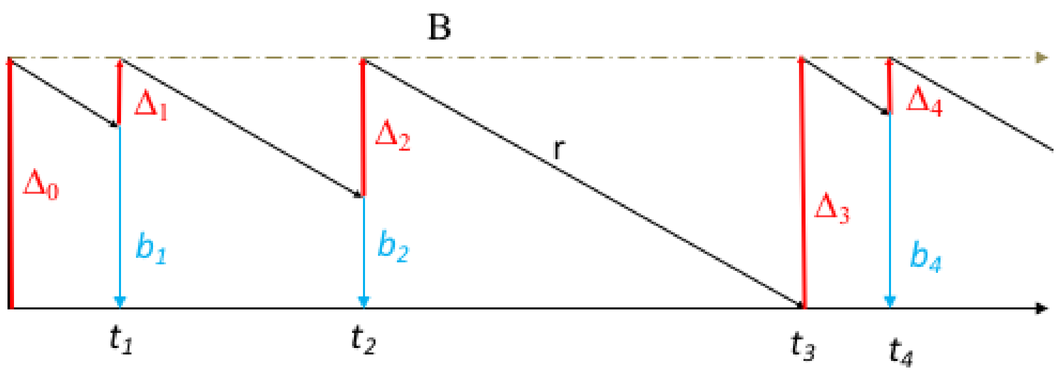
To help understand these throttling limits, here are a few examples, given the default account-level rate and burst limits:

- If a caller submits 10,000 requests in a one second period evenly (for example, 10 requests every millisecond), API Gateway processes all requests without dropping any.
- If the caller sends 10,000 requests in the first millisecond, API Gateway serves 5,000 of those requests and throttles the rest in the one-second period.

- If the caller submits 5,000 requests in the first millisecond and then evenly spreads another 5,000 requests through the remaining 999 milliseconds (for example, about 5 requests every millisecond), API Gateway processes all 10,000 requests in the one-second period without returning `429 Too Many Requests` error responses.
- If the caller submits 5,000 requests in the first millisecond and waits until the 101st millisecond to submit another 5,000 requests, API Gateway processes 6,000 requests and throttles the rest in the one-second period. This is because at the rate of 10,000 rps, API Gateway has served 1,000 requests after the first 100 milliseconds and thus emptied the bucket by the same amount. Of the next spike of 5,000 requests, 1,000 fill the bucket and are queued to be processed. The other 4,000 exceed the bucket capacity and are discarded.
- If the caller submits 5,000 requests in the first millisecond, submits 1,000 requests at the 101st millisecond, and then evenly spreads another 4,000 requests through the remaining 899 milliseconds, API Gateway processes all 10,000 requests in the one-second period without throttling.

More generally, at any given moment, when a bucket contains `b` and the maximum bucket capacity is `B`, the maximum additional tokens that can be added to the bucket is `#=B-b`. This maximum number of additional tokens corresponds to the maximum number of additional concurrent requests that a client can submit without receiving any `429` error responses. In general, `#` varies in time. The value ranges from zero when the bucket is full (that is, `b=B`) to `B` when the bucket is empty (that is, `b=0`). The range depends on the request-processing rate, which is the rate at which tokens are removed from the bucket, and the rate limit rate, which is the rate at which tokens are added to the bucket.

The following schematic shows the general behaviors of `#`, the maximum additional concurrent requests, as a function of time. The schematic assumes that the tokens in the bucket decrease at a combined rate of `r`, starting from an empty bucket.



The account-level rate limit and burst limit can be increased upon request. To request an increase of account-level throttling limits, contact the AWS Support Center. For more information, see API Gateway Limits (p. 450).

Default Method Throttling

As an API owner, you can set the default method throttling to override the account-level request throttling limits for a specific stage or for individual methods in an API. The Default method throttling limits are bounded by the account-level rate limits, even if you set the default method throttling limits higher than the account-level limits.

You can set the default method throttling limits by using the API Gateway console or by calling the API Gateway REST API (p. 449). For instructions on using the console, see Update Stage Settings (p. 286).

Enable API Caching to Enhance Responsiveness

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**.

To configure API caching for a given stage:

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**.

To configure API caching for individual methods using the console:

1. Choose **Stages** of an API from the main navigation pane.
2. Choose a method of the API in the chosen stage, from the secondary navigation pane.
3. Choose **Override for this method** in **Settings**.
4. Choose appropriate settings under the **Cache Settings** section (that is shown only if the stage-level caching is enabled).

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`.

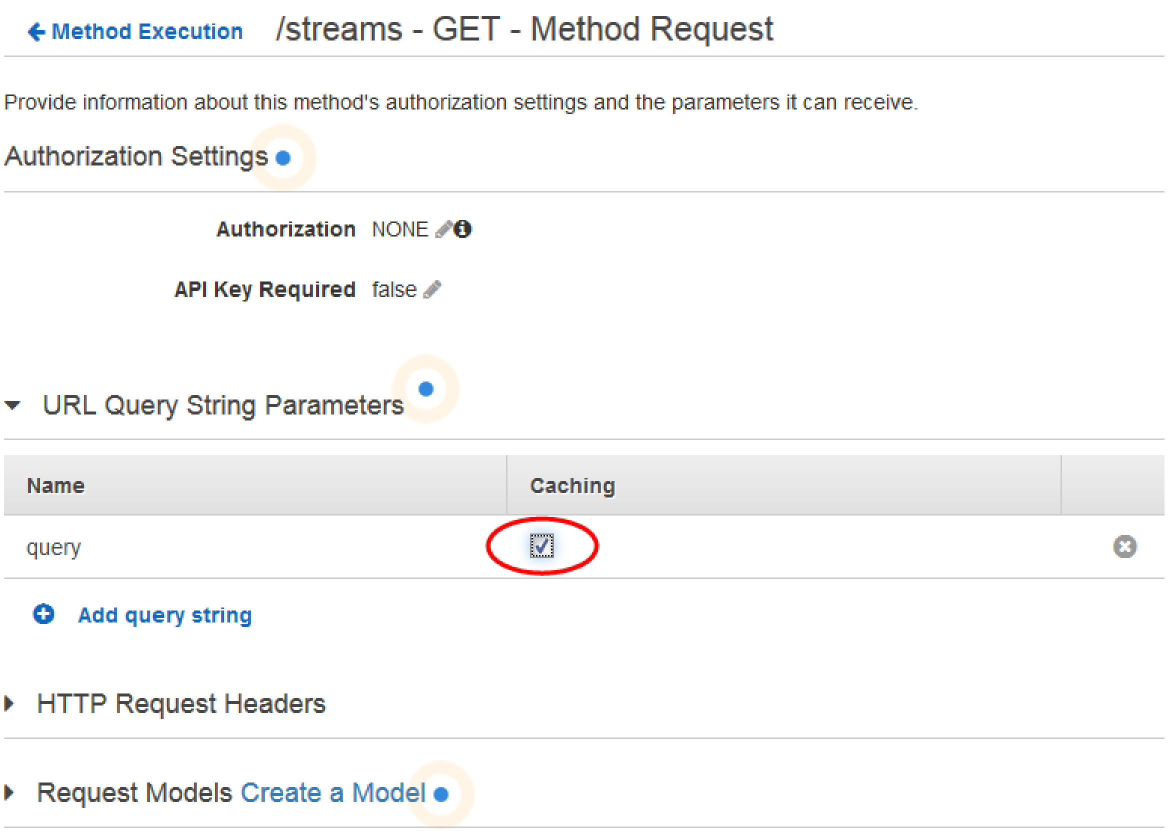
When a method or integration request takes more than one parameter, you can choose to include some or all of the parameters to create the cache key. For example, you can include only the `type` parameter in the cache key for the following request, made in the listed order within a TTL period:

```
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	✓

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 backend 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.

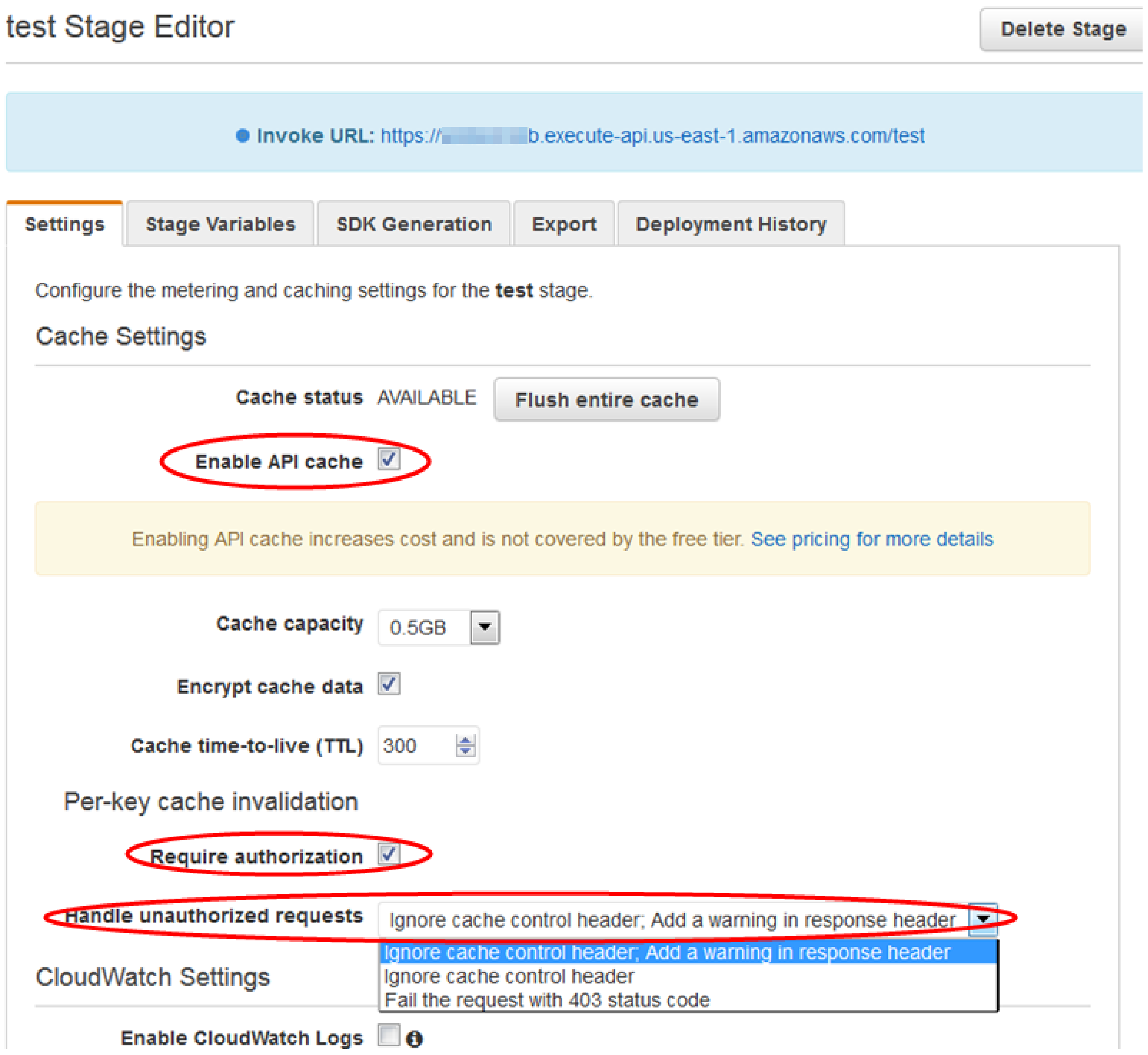
To grant permission for a caller, attach a policy of the following format to an IAM execution role for the user.

```
{
  "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 wild card (*) 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. 214)

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.



test Stage Editor

Delete Stage

● **Invoke URL:** https://[redacted]b.execute-api.us-east-1.amazonaws.com/test

Settings | Stage Variables | SDK Generation | Export | Deployment History

Configure the metering and caching settings for the **test** stage.

Cache Settings

Cache status AVAILABLE Flush entire cache

Enable API cache ✓

Enabling API cache increases cost and is not covered by the free tier. See pricing for more details

Cache capacity 0.5GB

Encrypt cache data ✓

Cache time-to-live (TTL) 300

Per-key cache invalidation

Require authorization ✓

Handle unauthorized requests Ignore cache control header; Add a warning in response header

- Ignore cache control header; Add a warning in response header
- Ignore cache control header
- Fail the request with 403 status code

CloudWatch Settings

Enable CloudWatch Logs ☐

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`.

Set up Stage Variable for API Deployment

ß

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 backend.

For more information about mapping templates, see Mapping Template Built-in Functions and Variables (p. 153).

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 backend endpoints. For example, your API can pass a GET request as an HTTP proxy to the backend web host (for example, `http://example.com`). In this case, the backend 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

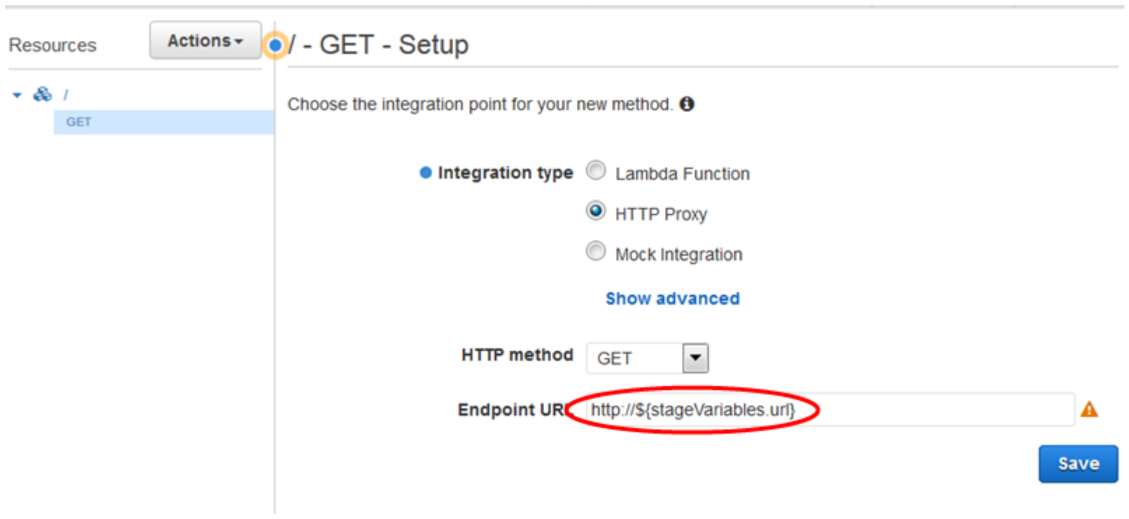
In this tutorial, you will learn how to set stage variables for two deployment stages of a sample API, using the Amazon API Gateway console.

Prerequisites

1. You must have an API available in API Gateway. Follow the instructions in Creating an API (p. 72).
2. You must have deployed the API at least once. Follow the instructions in Deploying an API (p. 284).
3. You must have created the first stage for a deployed API. Follow the instructions in Associate an Existing Deployment with a New stage (p. 286).

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**.



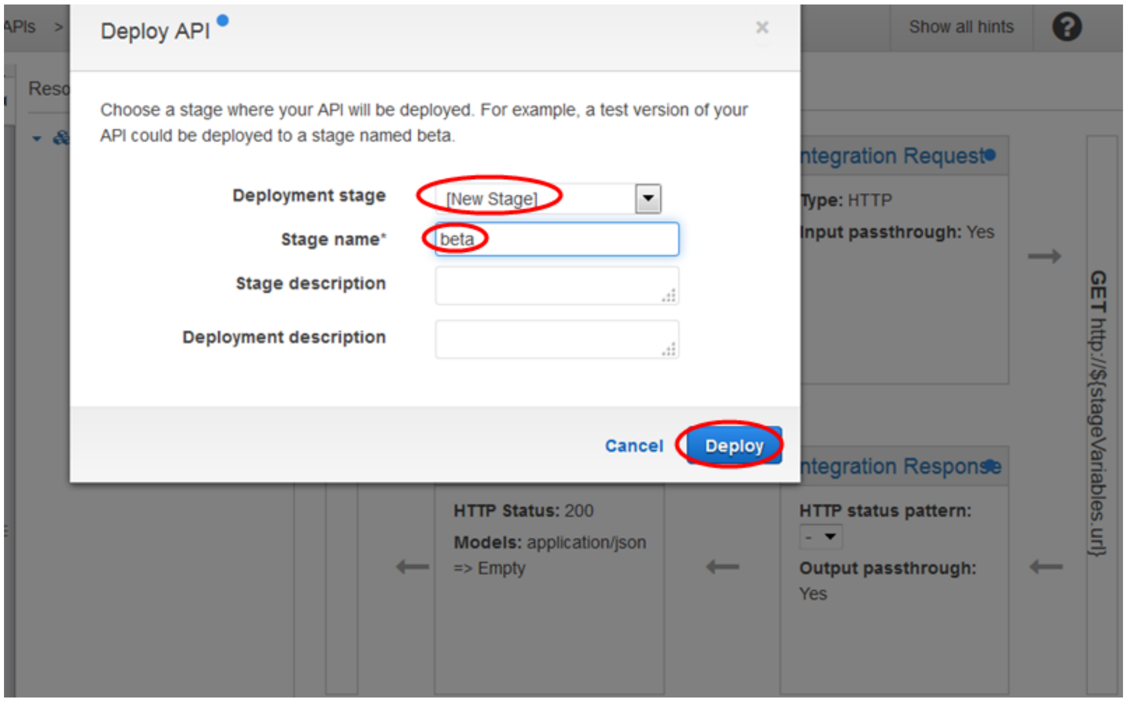
Choose the integration point for your new method.

Integration type: Lambda Function / [✓ HTTP Proxy] / Mock Integration

HTTP method: GET

Endpoint URL: http://${stageVariables.url}

3. Choose **Deploy API**. Choose **New Stage** and enter "`beta`" for **Stage name**. Choose **Deploy**.



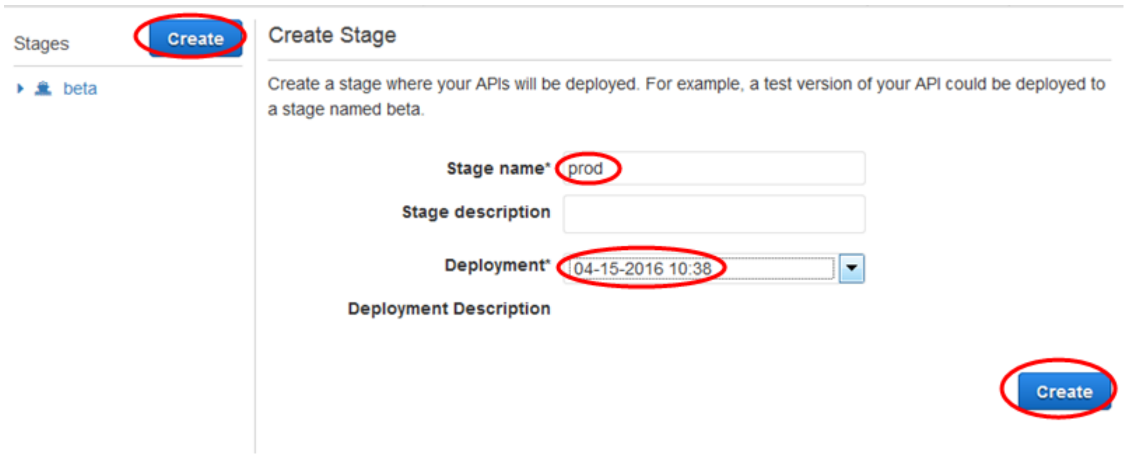
Deploy API

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*: beta
Stage description:
Deployment description:

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.
6. 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.
7. From the **Stages** navigation pane, choose **Create**. For **Stage name**, type **`prod`**. Select a recent deployment from **Deployment** and then choose **Create**.



Create Stage

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

8. 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.

Use Amazon API Gateway Stage Variables

You can use API Gateway stage variables to access the HTTP and Lambda backends for different API deployment stages and to pass stage-specific configuration metadata into an HTTP backend 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. 297).

Access an HTTP endpoint through an API with a stage variable

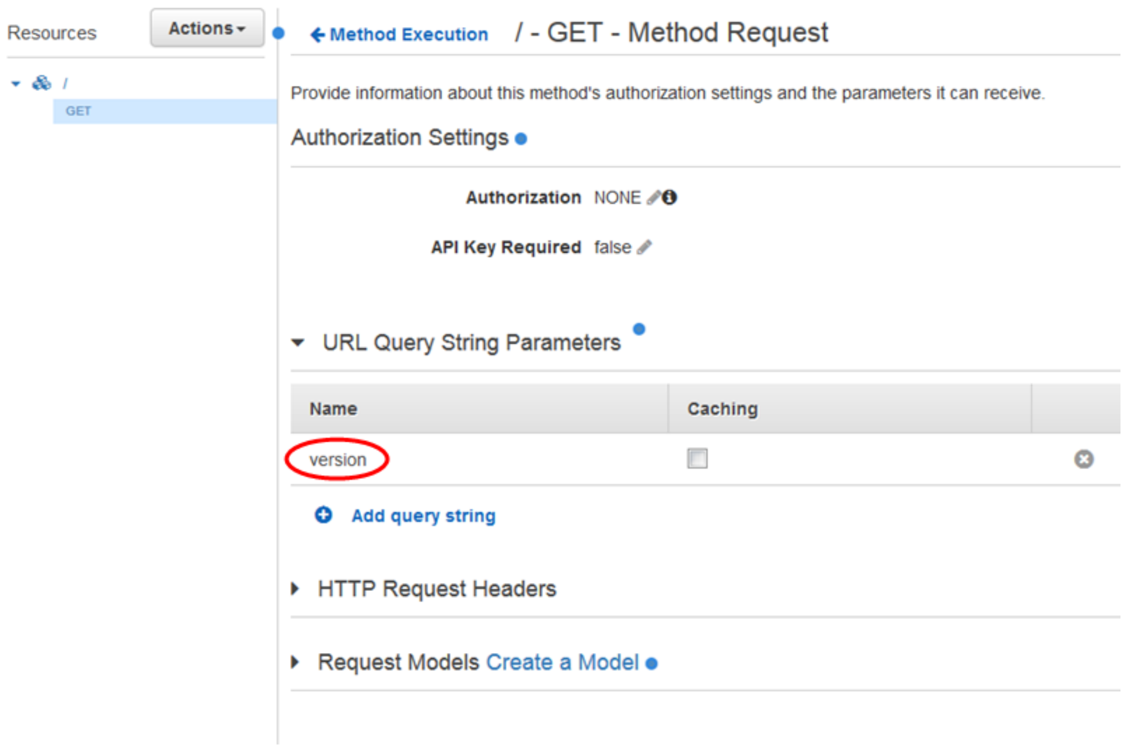
1. In the **Stages** navigation pane, choose **beta**. In **beta Stage Editor**, choose the **Invoke URL** link. This starts the **beta** stage `GET` request on the root resource of the API.

   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 backend 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. 297).

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.



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

Authorization Settings

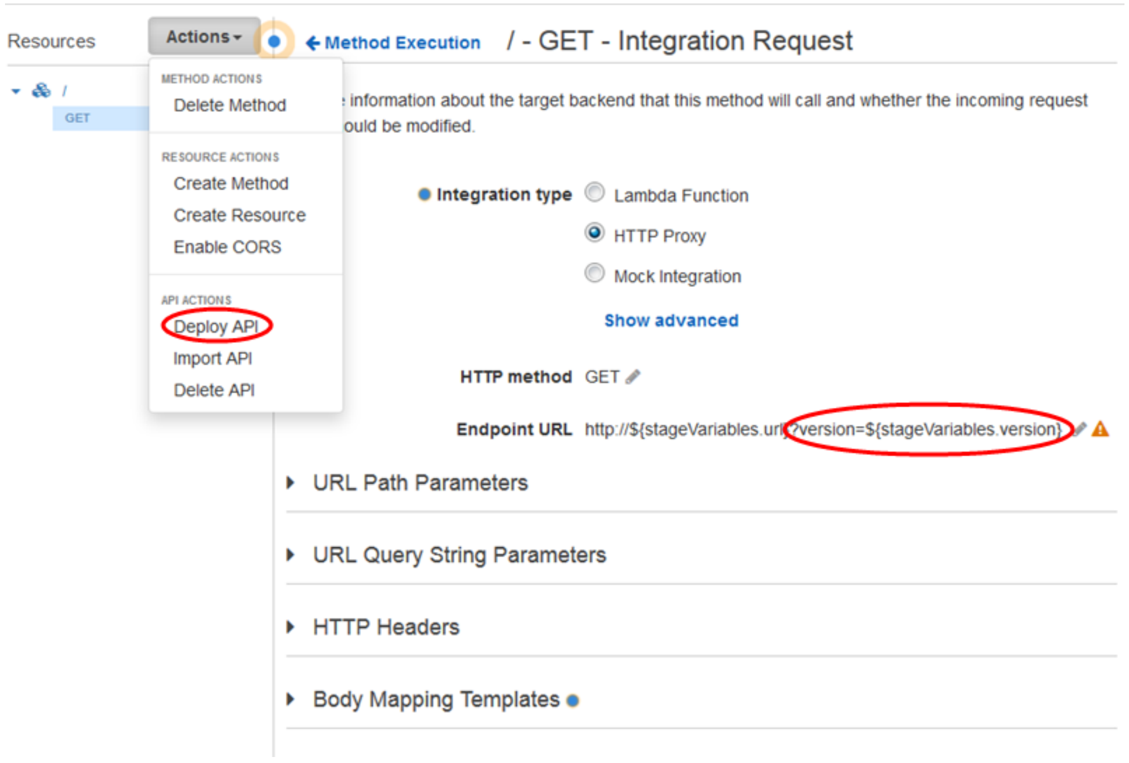
Authorization NONE

API Key Required false

URL Query String Parameters

Name	Caching
version	☐

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.



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

Integration type ○ Lambda Function [✓ HTTP Proxy] ○ Mock Integration

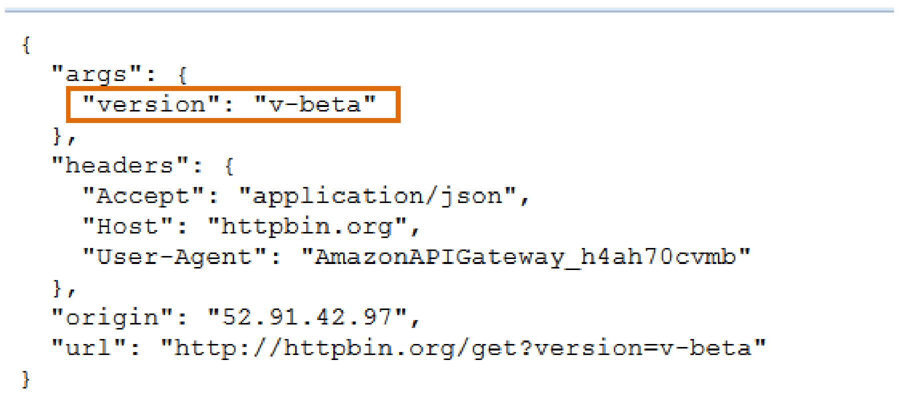
HTTP method GET

Endpoint URL http://${stageVariables.url}?version=${stageVariables.version}

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.

4. The response is shown next. Notice that `v-beta`, assigned to the `version` stage variable, is passed in the backend as the `version` argument.

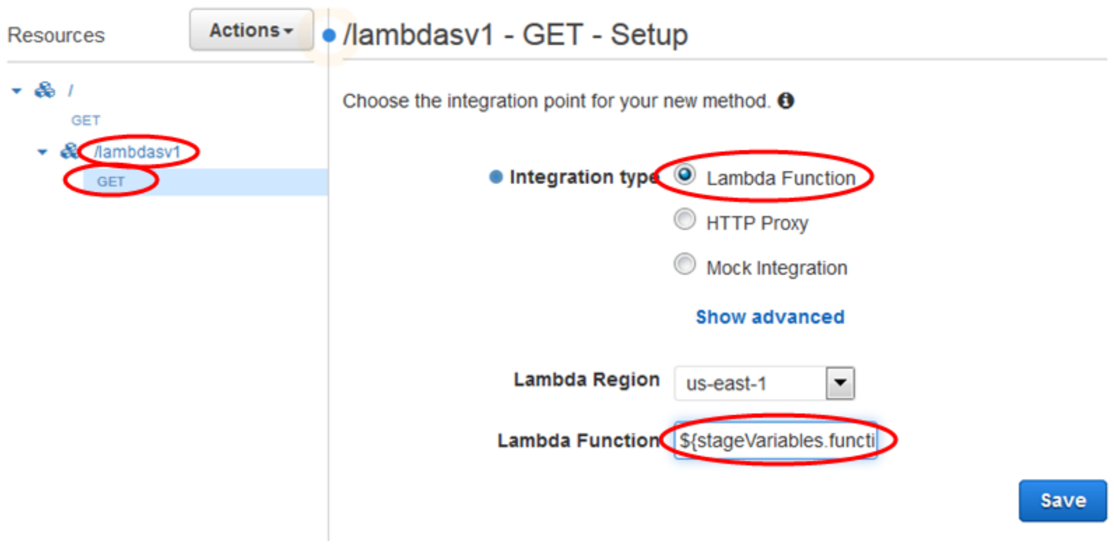


```
{
  "args": {
    "version": "v-beta"
  },
  "headers": {
    "Accept": "application/json",
    "Host": "httpbin.org",
    "User-Agent": "AmazonAPIGateway_h4ah70cvmb"
  },
  "origin": "52.91.42.97",
  "url": "http://httpbin.org/get?version=v-beta"
}
```

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. 297).

1. 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**.



/lambdasv1 - GET - Setup

Choose the integration point for your new method.

Integration type [✓ Lambda Function] HTTP Proxy Mock Integration

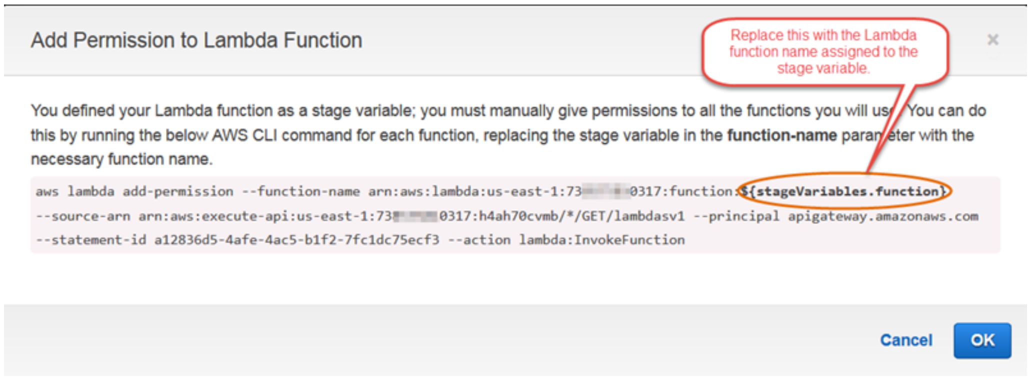
Lambda Region: us-east-1

Lambda Function: ${stageVariables.functi

Tip
When prompted with **Add Permision 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/lambdasv1 --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.


Add Permission to Lambda Function
Replace this with the Lambda function name assigned to the stage variable.
×
Cancel
OK

You defined your Lambda function as a stage variable; you must manually give permissions to all the functions you will use. You can do this by running the below AWS CLI command for each function, replacing the stage variable in the **function-name** parameter with the necessary function name.

```
aws lambda add-permission --function-name arn:aws:lambda:us-east-1:73[redacted]0317:function:${stageVariables.function}
--source-arn arn:aws:execute-api:us-east-1:73[redacted]0317:h4ah70cvmb/*/GET/lambdasv1 --principal apigateway.amazonaws.com
--statement-id a12836d5-4afe-4ac5-b1f2-7fc1dc75ecf3 --action lambda:InvokeFunction
```

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 `/lambdasv1` 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, callback) {
       if (event.version)
           callback(null, 'Hello, World! (' + event.version + ')' );
       else
           callback(null, "Hello, world! (v-unknown)");
   };
   ```

   This implementation results in the following response.

   ```
   "Hello, world! (v-unknown)"
   ```

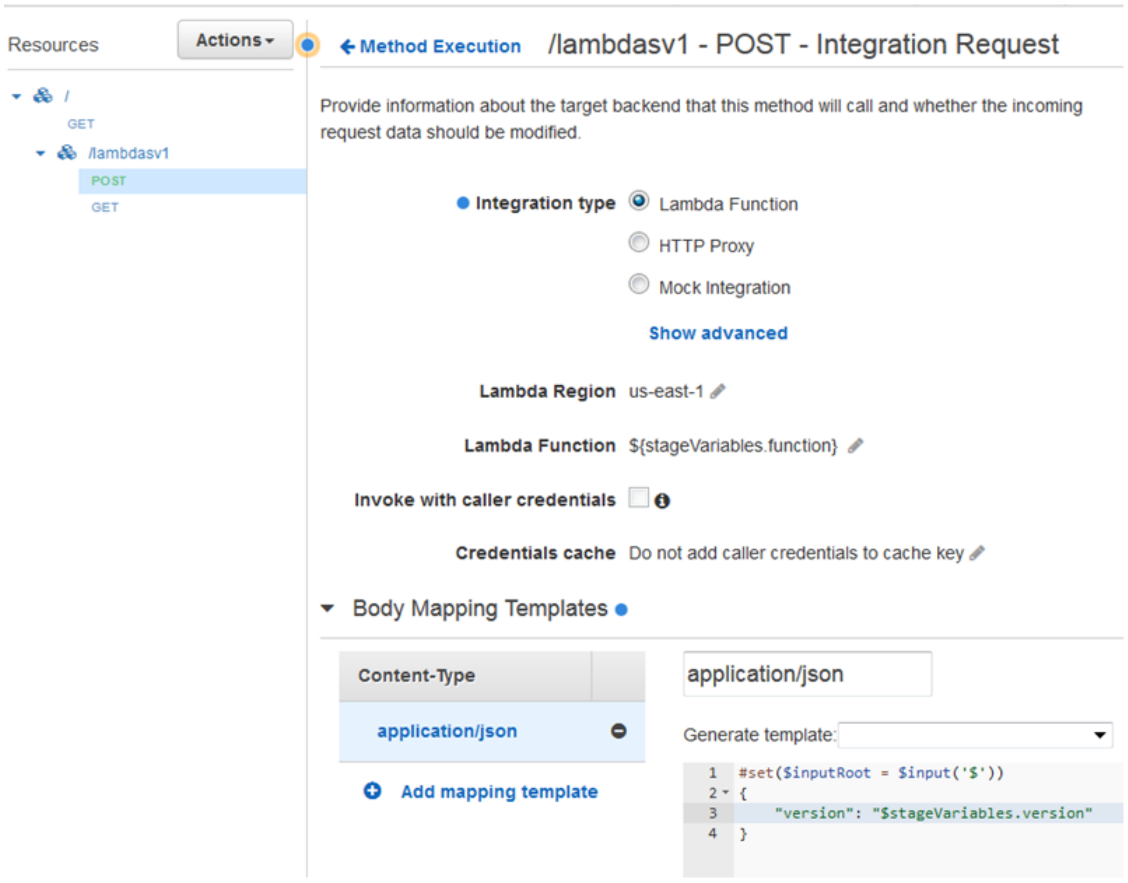
Pass stage-specific metadata to a Lambda function via a stage variable

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. 301).

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.



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

Lambda Region: us-east-1
Lambda Function: ${stageVariables.function}
Invoke with caller credentials: ☐
Credentials cache: Do not add caller credentials to cache key

application/json
Generate template:

```
#set($inputRoot = $input('$'))
{
    "version": "$stageVariables.version"
}
```

   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>}`

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 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

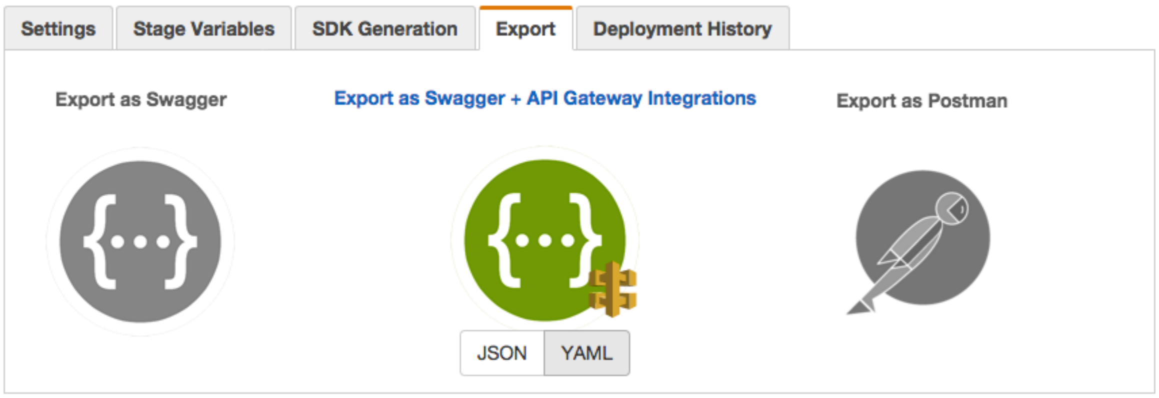
To export and download an API in Swagger definitions with API Gateway integration in YAML format:

```
GET /restapis/<restapi_id>/stages/<stage_name>/exports/swagger?extensions=integrations
Host: apigateway.<region>.amazonaws.com
Accept: application/yaml
```

Export API Using the API Gateway Console

After deploying your API to a stage (p. 285), you can proceed to export the API in the stage to a Swagger file 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.



Generate SDK for an API

To call your API in a platform- or language-specific way, you must generate the platform- or language-specific SDK of the API. Currently, API Gateway supports generating an SDK for an API in Java, JavaScript, Java for Android, and Objective-C or Swift for iOS.

This section explains how to generate an SDK of an API Gateway API and demonstrates how to use the generated SDK in a Java app, a Java for Android apps, Objective-C and Swift for iOS apps, and a JavaScript app.

To facilitate the discussion, we refer to this API Gateway API (p. 310), which exposes this Simple Calculator (p. 309) Lambda function.

Before proceeding, make sure you create or import the API and deploy it at least once in API Gateway. For instructions, see Deploying an API (p. 284).

Topics

Generate SDKs for an API Using the API Gateway Console

To generate a platform- or language-specific SDK for an API in API Gateway, you must first create, test, and deploy the API in a stage. For illustration purposes, we use the Simple Calculator (p. 315) API as an example to generate language-specific or platform-specific SDKs throughout this section. For instructions on how to create, test, and deploy this API, see Create the Simple Calculator API (p. 310).

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 **Java**, specify the following:

   - For **Service Name**, specify the name of your SDK. For example, `SimpleCalcSdk`. This becomes the name of your SDK client class. The name corresponds to the `<name>` tag under `<project>` in the pom.xml file, which is in the SDK's project folder. Do not include hyphens.
   - For **Java Package Name**, specify a package name for your SDK. For example, `examples.aws.apig.simpleCalc.sdk`. This package name is used as the namespace of your SDK library. Do not include hyphens.
   - For **Java Build System**, type `maven` or `gradle` to specify the build system.
   - For **Java Group Id**, type a group identifier for your SDK project. For example, `my-apig-api-examples`. This identifier corresponds to the `<groupId>` tag under `<project>` in the `pom.xml` file, which is in the SDK's project folder.
   - For **Java Artifact Id**, type an artifact identifier for your SDK project. For example, `simple-calc-sdk`. This identifier corresponds to the `<artifactId>` tag under `<project>` in the `pom.xml` file, which is in the SDK's project folder.
   - For **Java Artifact Version**, type a version identifier string. For example, `1.0.0`. This version identifier corresponds to the `<version>` tag under `<project>` in the `pom.xml` file, which is in the SDK's project folder.
   - For **Source Code License Text**, type the license text of your source code, if any.

b. 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`**).

c. 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. 310) 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. 342).

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 **Java** for **Platform**, follow the instructions in Use a Java SDK Generated by API Gateway (p. 335).
   - If you chose **Android** for **Platform**, follow the instructions in Use an Android SDK Generated by API Gateway (p. 339).
   - If you chose **iOS** for **Platform**, follow the instructions in Use iOS SDK Generated by API Gateway in Objective-C or Swift (p. 342).
   - If you chose **JavaScript** for **Platform**, follow the instructions in Use a JavaScript SDK Generated by API Gateway (p. 340).

After updating an API, you must redeploy the API and regenerate the SDK to have the updates included.

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.

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

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.

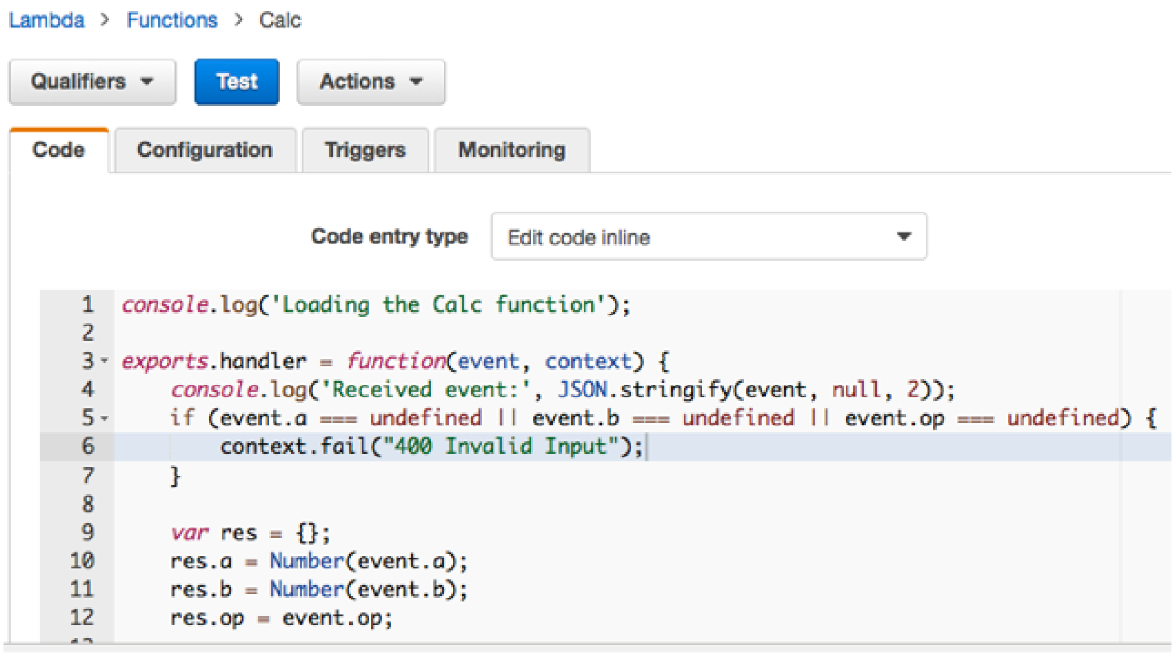
Simple Calculator Lambda Function Implementation

The implementation of the Lambda function is as follows:

```
console.log('Loading the Calc function');

exports.handler = function(event, context, callback) {
    console.log('Received event:', JSON.stringify(event, null, 2));
    if (event.a === undefined || event.b === undefined || event.op === undefined) {
        callback("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)) {
        callback("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:
            callback("400 Invalid Operator");
            break;
    }
    callback(null, 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.

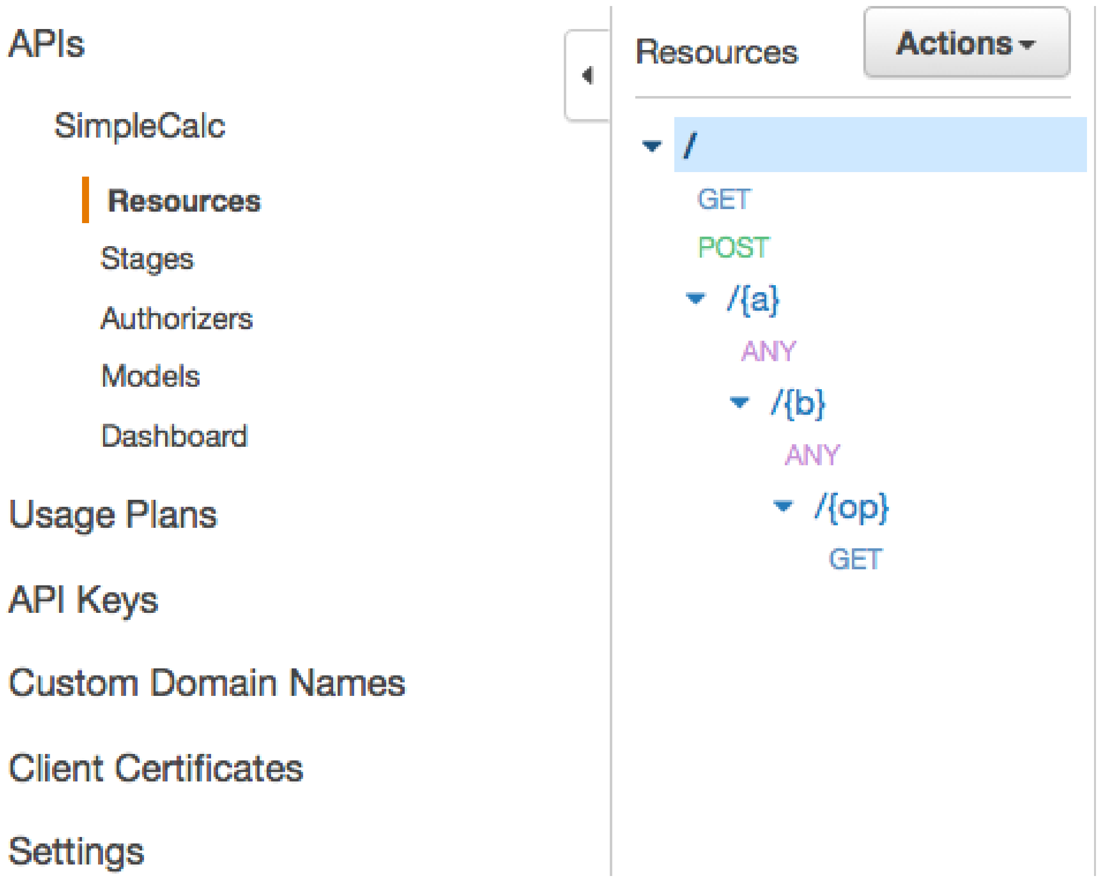


Code entry type: Edit code inline

```
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;
```

Simple Calculator API in API Gateway

Our simple calculator API exposes three methods (GET, POST, GET) to invoke the Simple Calculator Lambda Function (p. 309) (`Calc`). A graphical representation of this API is shown as follows:



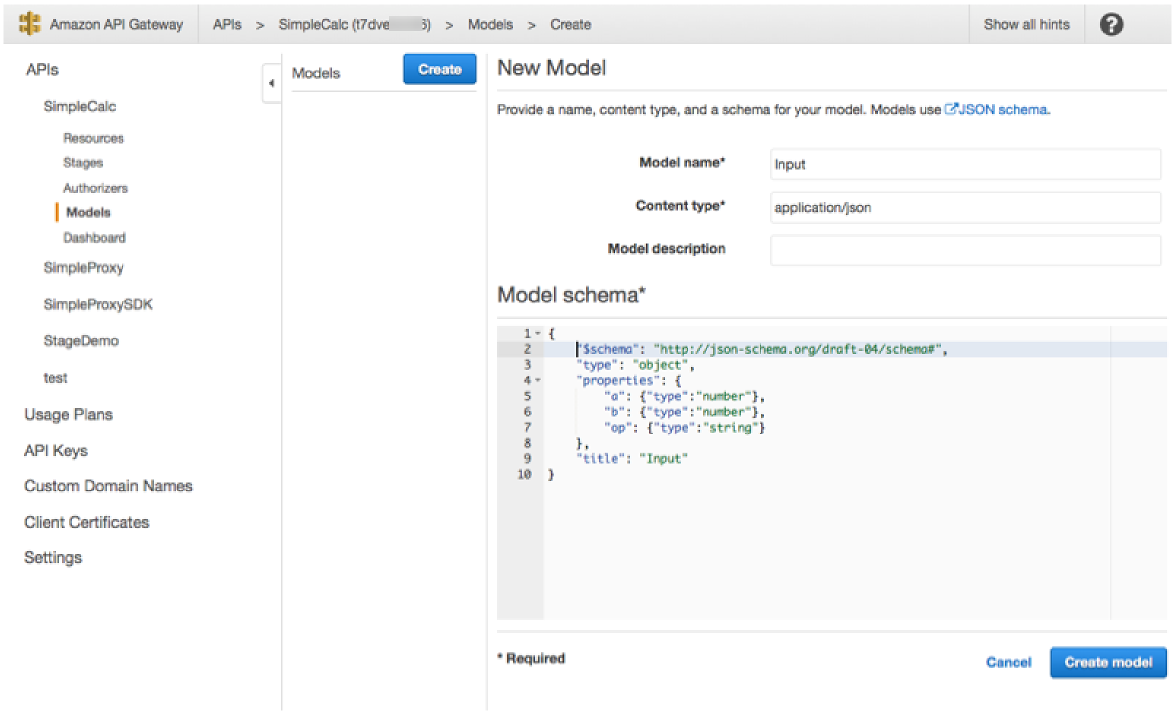
These three methods show different ways to supply the input for the backend Lambda function to perform the same operation:

- 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. 72). If you're new to API Gateway, see Getting Started (p. 6) first.

Create Models for Input and Output

To specify strongly typed input in the SDK, we create an `Input` model for the API:



New Model

Provide a name, content type, and a schema for your model. Models use JSON schema.

Model name* Input

Content type* application/json

Model description

Model schema*

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "type": "object",
    "properties": {
        "a": {"type":"number"},
        "b": {"type":"number"},
        "op": {"type":"string"}
    },
    "title": "Input"
}
```

Similarly, to describe the response body data type, we create the following models in the API Gateway:

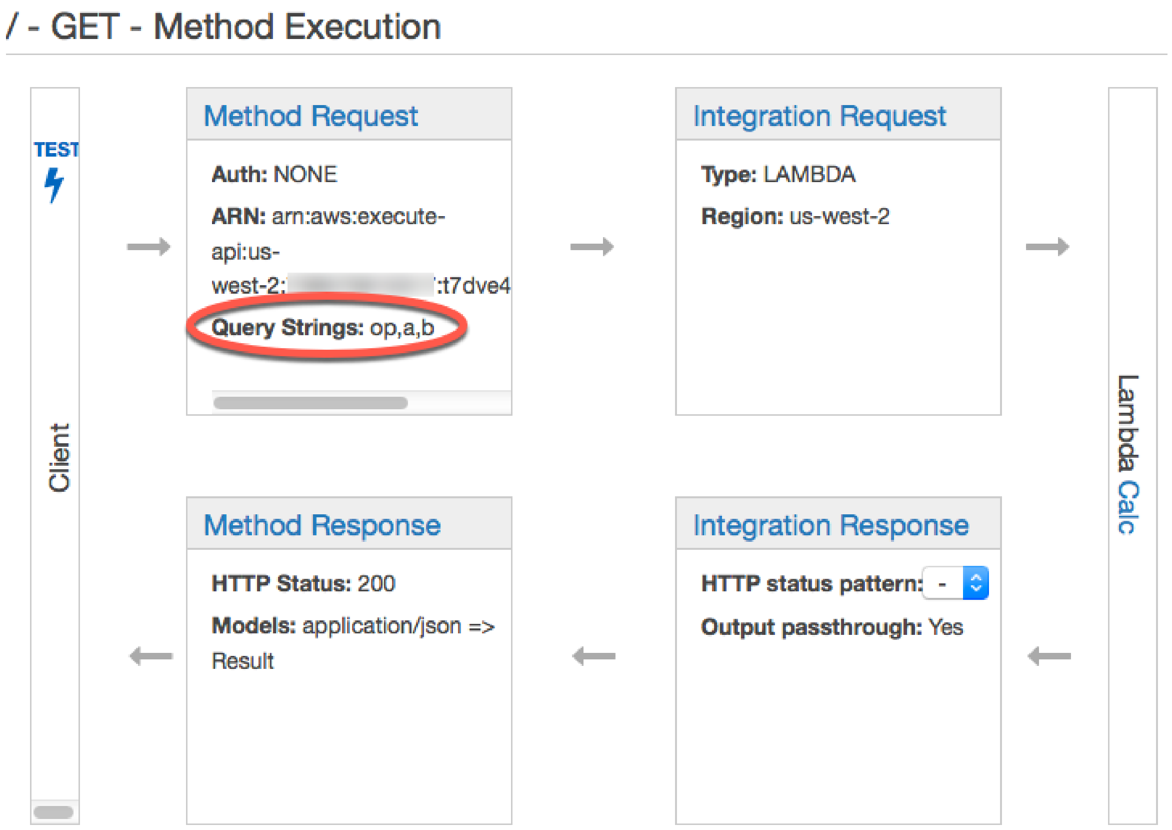
```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "type": "object",
    "properties": {
        "c": {"type":"number"}
    },
    "title": "Output"
}
```

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

Method Request

Auth: NONE

ARN: arn:aws:execute-api:us-west-2:[redacted]:t7dve4

Query Strings: op,a,b

Integration Request

Type: LAMBDA

Region: us-west-2

Method Response

HTTP Status: 200

Models: application/json => Result

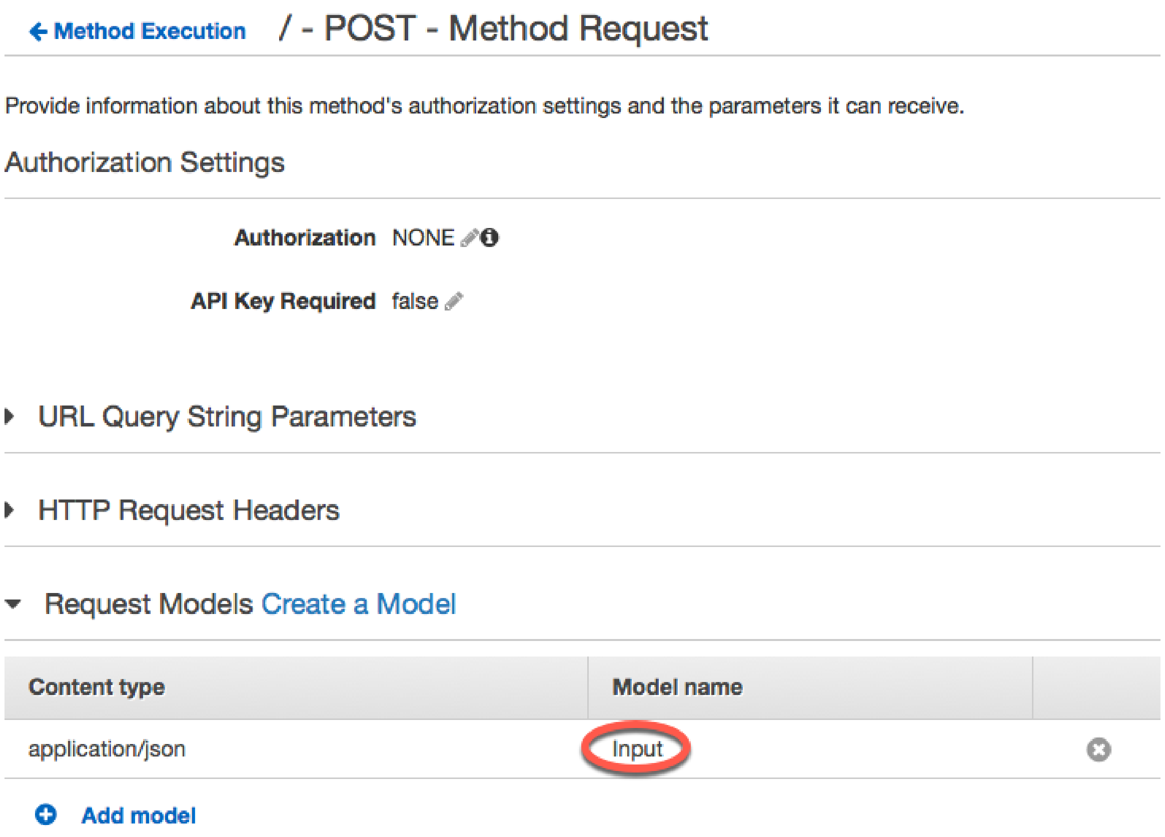
Integration Response

HTTP status pattern: -

Output passthrough: Yes

Set Up Data Model for the Payload as Input to the Backend

For the `POST /` method, we create the `Input` model and add it to the method request to define the shape of input data.



/ - POST - Method Request

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

Authorization Settings

Authorization NONE

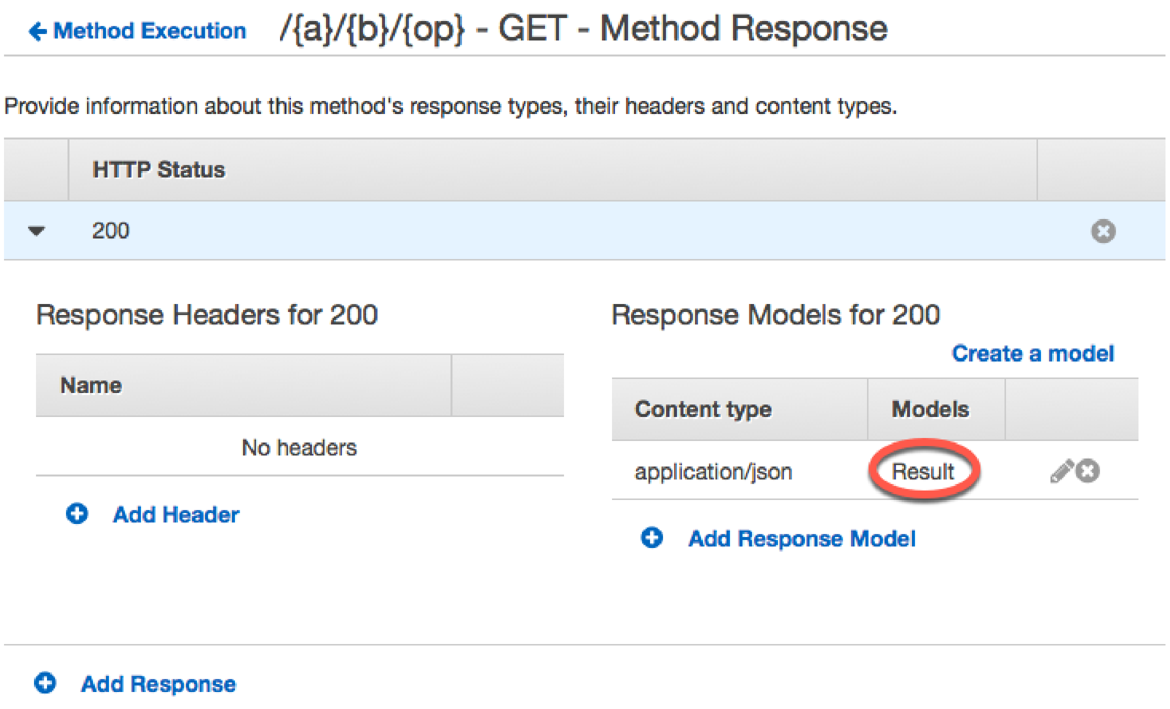
API Key Required false

Content type	Model name
application/json	Input

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 Backend

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.



Provide information about this method's response types, their headers and content types.

HTTP Status
200

Response Headers for 200

Name
No headers

Response Models for 200

Content type	Models
application/json	Result

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. 315).

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. 309) 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  \"a\" :
 $input.params('a'),\n  \"b\" : $input.params('b'),\n  \"op\" : \"$input.params('op')\"\n}"
          },
          "uri": "arn:aws:apigateway:us-west-2:lambda:path/2015-03-31/functions/
arn:aws:lambda:us-west-2:123456789012:function:Calc/invocations",
          "passthroughBehavior": "when_no_templates",
          "httpMethod": "POST",
          "responses": {
            "default": {
              "statusCode": "200",
              "responseTemplates": {
                "application/json": "#set($inputRoot = $input.path('$'))\n{\n  \"input\" :
 {\n    \"a\" : $inputRoot.a,\n    \"b\" : $inputRoot.b,\n    \"op\" : \"$inputRoot.op\"\n
  },\n  \"output\" : {\n    \"c\" : $inputRoot.c\n  }\n}"
              }
            }
          },
          "type": "aws"
        }
      },
      "post": {
        "consumes": [
          "application/json"
        ],
        "produces": [
          "application/json"
        ],
        "parameters": [
          {
            "in": "body",
            "name": "Input",
            "required": true,
            "schema": {
              "$ref": "#/definitions/Input"
            }
          }
        ],
        "responses": {
          "200": {
            "description": "200 response",
            "schema": {
              "$ref": "#/definitions/Result"
            }
          }
        },
        "x-amazon-apigateway-integration": {
          "uri": "arn:aws:apigateway:us-west-2:lambda:path/2015-03-31/functions/
arn:aws:lambda:us-west-2:123456789012:function:Calc/invocations",
          "passthroughBehavior": "when_no_match",
          "httpMethod": "POST",
          "responses": {
            "default": {
              "statusCode": "200",
              "responseTemplates": {
                "application/json": "#set($inputRoot = $input.path('$'))\n{\n  \"input\" :
 {\n    \"a\" : $inputRoot.a,\n    \"b\" : $inputRoot.b,\n    \"op\" : \"$inputRoot.op\"\n
  },\n  \"output\" : {\n    \"c\" : $inputRoot.c\n  }\n}"
              }
            }
          },
          "type": "aws"
        }
      }
    },
    "/{a}": {
      "x-amazon-apigateway-any-method": {
        "consumes": [
          "application/json"
        ],
        "produces": [
          "application/json"
        ],
        "parameters": [
          {
            "name": "a",
            "in": "path",
            "required": true,
            "type": "string"
          }
        ],
        "responses": {
          "404": {
            "description": "404 response"
          }
        },
        "x-amazon-apigateway-integration": {
          "requestTemplates": {
            "application/json": "{\"statusCode\": 200}"
          },
          "passthroughBehavior": "when_no_match",
          "responses": {
            "default": {
              "statusCode": "404",
              "responseTemplates": {
                "application/json": "{ \"Message\" : \"Can't $context.httpMethod
 $context.resourcePath\" }"
              }
            }
          },
          "type": "mock"
        }
      }
    },
    "/{a}/{b}": {
      "x-amazon-apigateway-any-method": {
        "consumes": [
          "application/json"
        ],
        "produces": [
          "application/json"
        ],
        "parameters": [
          {
            "name": "a",
            "in": "path",
            "required": true,
            "type": "string"
          },
          {
            "name": "b",
            "in": "path",
            "required": true,
            "type": "string"
          }
        ],
        "responses": {
          "404": {
            "description": "404 response"
          }
        },
        "x-amazon-apigateway-integration": {
          "requestTemplates": {
            "application/json": "{\"statusCode\": 200}"
          },
          "passthroughBehavior": "when_no_match",
          "responses": {
            "default": {
              "statusCode": "404",
              "responseTemplates": {
                "application/json": "{ \"Message\" : \"Can't $context.httpMethod
$context.resourcePath\" }"
              }
            }
          },
          "type": "mock"
        }
      }
    },
    "/{a}/{b}/{op}": {
      "get": {
        "consumes": [
          "application/json"
        ],
        "produces": [
          "application/json"
        ],
        "parameters": [
          {
            "name": "a",
            "in": "path",
            "required": true,
            "type": "string"
          },
          {
            "name": "b",
            "in": "path",
            "required": true,
            "type": "string"
          },
          {
            "name": "op",
            "in": "path",
            "required": true,
            "type": "string"
          }
        ],
        "responses": {
          "200": {
            "description": "200 response",
            "schema": {
              "$ref": "#/definitions/Result"
            }
          }
        },
        "x-amazon-apigateway-integration": {
          "requestTemplates": {
            "application/json": "#set($inputRoot = $input.path('$'))\n{\n  \"a\" :
 $input.params('a'),\n  \"b\" : $input.params('b'),\n  \"op\" : \"$input.params('op')\"\n}"
          },
          "uri": "arn:aws:apigateway:us-west-2:lambda:path/2015-03-31/functions/
arn:aws:lambda:us-west-2:123456789012:function:Calc/invocations",
          "passthroughBehavior": "when_no_templates",
          "httpMethod": "POST",
          "responses": {
            "default": {
              "statusCode": "200",
              "responseTemplates": {
                "application/json": "#set($inputRoot = $input.path('$'))\n{\n  \"input\" :
 {\n    \"a\" : $inputRoot.a,\n    \"b\" : $inputRoot.b,\n    \"op\" : \"$inputRoot.op\"\n
  },\n  \"output\" : {\n    \"c\" : $inputRoot.c\n  }\n}"
              }
            }
          },
          "type": "aws"
        }
      }
    }
  },
  "definitions": {
    "Input": {
      "type": "object",
      "properties": {
        "a": {
          "type": "number"
        },
        "b": {
          "type": "number"
        },
        "op": {
          "type": "string"
        }
      },
      "title": "Input"
    },
    "Output": {
      "type": "object",
      "properties": {
        "c": {
          "type": "number"
        }
      },
      "title": "Output"
    },
    "Result": {
      "type": "object",
      "properties": {
        "input": {
          "$ref": "#/definitions/Input"
        },
        "output": {
          "$ref": "#/definitions/Output"
        }
      },
      "title": "Result"
    }
  }
}
```

Set up Custom Domain Name for API Host Name

After deploying your API, you (and your customers) can invoke the API using the default root URL of the following format:

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

Here, the host name of the URL (i.e., `api-id.execute-api.region.amazonaws.com`) can be difficult to recall and not user-friendly. 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.haymuto.com`) as the API's host name and choose a base path (e.g., `myservice`) to present an alternative URL of the API of the following format:

```
https://api.haymuto.com/myservice
```

The base path can be an empty string. In this case, the API's root URL is the same as the custom domain (e.g., `https://api.haymuto.com`.) In this case, the custom domain name can support only one API.

For every API you create, API Gateway sets up an Amazon CloudFront distribution for the API. Requests with the default API URL are routed through the corresponding CloudFront distribution. Similarly, for every custom domain name, API Gateway sets up a CloudFront distribution. An API request with the custom domain name is routed through the custom domain name's CloudFront distribution.

Note
The CloudFront distribution created by API Gateway is owned by a region-specific account affiliated with API Gateway. When tracing operations to create and update such a CloudFront distribution in CloudWatch logs, you must use the API Gateway account Id. For more information, see Log Custom Domain Name Creation in CloudTrail (p. 326).

Tip
To set up a custom domain name or to update its certificate, you must have a permission to update CloudFront distributions. You can do so by attaching the following IAM policy statement an IAM user, group or role in your account:

```
{
    "Version": "2012-10-17",
    "Statement": [
         {
            "Sid": "AllowCloudFrontUpdateDistribution",
            "Effect": "Allow",
            "Action": [
                "cloudfront:updateDistribution"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

API Gateway supports custom domain names 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*.

To set up a custom domain name as your API's host name, you, as the API owner, must provide an SSL/TLS certificate for the custom domain name. To provide this certificate, you can request AWS Certificate Manager (ACM) to generate a new certificate in ACM or to import into ACM a custom certificate issued by a third-party certificate authority. To import an SSL/TLS certificate, you must provide the PEM-formatted SSL/TLS certificate body, its private key, and the certificate chain for the custom domain name. Each certificate stored in ACM is identified by its ARN. To use an AWS-managed certificate for a domain name, you simply reference its ARN.

ACM makes it straightforward to set up and use a custom domain name for an API: create in or import into ACM a certificate for the given domain name, set up the domain name in API Gateway with the ARN of the certificate provided by ACM, and map a base path under the custom domain name to a deployed stage of the API. With certificates issued by ACM, you do not have to worry about exposing any sensitive certificate details, such as the private key.

Note
API Gateway does not support self-signed SSL/TLS certificates because these certificates are not supported by CloudFront.

You must have a registered Internet domain name in order to set up custom domain names for your APIs. If needed, you can register an Internet domain using Amazon Route 53 or using a third-party domain registrar of your choice. An API's custom domain name can be the name of a subdomain or the root domain (aka, zone apex) of a registered Internet domain.

After a custom domain name is created 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. The mapping ensures that API requests against the custom domain name are routed through the correct CloudFront distribution.

Note
A custom domain name associated with an API is region- and account-specific. Moving a custom domain name between regions or AWS accounts involves deleting the 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.

This section describes how to use ACM to create an SSL/TLS certificate for a custom domain name, to set up the custom domain name for an API, to associate a specific API with a base path under the custom domain name, and to renew (aka rotate) an expiring certificate that was imported into ACM for the custom domain name.

Topics

Get Certificates Ready in AWS Certificate Manager

Before setting up a custom domain name for an API, you must have an SSL/TLS certificate ready in AWS Certificate Manager. The following steps describe how to get this done. For more information, see the AWS Certificate Manager User Guide.

> **Note**
> To use an ACM Certificate with API Gateway, you must request or import the certificate in the US East (N. Virginia) (`us-east-1`) Region.

To get a certificate for a given domain name issued by or imported into ACM

1. Register your Internet domain; e.g., `*myDomain*.com`. You can use either Amazon Route 53 or a third-party accredited domain registrar. For a list of such registrar, see Accredited Registrar Directory at the ICANN website.
2. To create in or import into ACM an SSL/TLS certificate for a domain name, do one of the following:

   To request a certificate provided by ACM for a domain name

   1. Sign in to the AWS Certificate Manager console.
   2. Choose **Request a certificate**.
   3. Type a custom domain name for your API; e.g., `api.example.com`, in **Domain name**.
   4. Optionally, choose **Add another name to this certificate**.
   5. Choose **Review and request**.
   6. Choose **Confirm and request**.
   7. For a valid request, a registered owner of the Internet domain must consent to the request before ACM issues the certificate.

   To import into ACM a certificate for a domain name

   1. Get a PEM-encoded SSL/TLS certificate for your custom domain name from a certificate authority. For a partial list of such CAs, see the Mozilla Included CA List

      a. Generate a private key for the certificate and save the 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/TLS 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 certificate authority you use. Ask your certificate authority 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
```

2. Upload the certificate to AWS Certificate Manager:

   a. Sign in to the AWS Certificate Manager console.

   b. Choose **Import a 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+KgAwIBAgIQJ1XxJ8Pl++gOfQtj0IBoqDANBgkqhkiG9w0BAQUFADBB
   ...
   az8Cg1aicxLBQ7EaWIhhgEXAMPLE
   -----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
   ...
   1qGvJ3u04vdnzaYN5WoyN5LFckrlA71+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/arxbx1BJCHBvlEPNnsdw8EXAMPLE
   -----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-----
   ```

   f. Choose **Review and import**.

3. After the certificate is successfully created or imported, make note of the certificate ARN. You need it when setting up the custom domain name, next.

Set Up a Custom Domain Name for an API Gateway API

The following procedure describes how to set up a custom domain name for an API using the API Gateway console.

To set up a custom domain name using the API Gateway console

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 Custom Domain Name** next.
4. a. Under **New Custom Domain Name**, type your domain name (for example, `api.example.com`) in **Domain Name**.

   Note
   Do not use the wildcard character (i.e., `*`) for your custom domain names. API Gateway does not support it, even though the API Gateway console (or the AWS CLI) accepts it and can map it to a CloudFront distribution. However, you can use wildcard certificates.

   b. Choose a certificate from the **ACM Certificate** list.

   c. Choose **Add mapping** under **Base Path Mappings** to set a base path (**Path**) for a deployed API in a given stage (selected from the **Destination** dropdown lists.) You can also set the base path mapping after the custom domain name is created. For more information, see Configure Base Path Mapping of an API with a Custom Domain Name as its Host Name (p. 327).

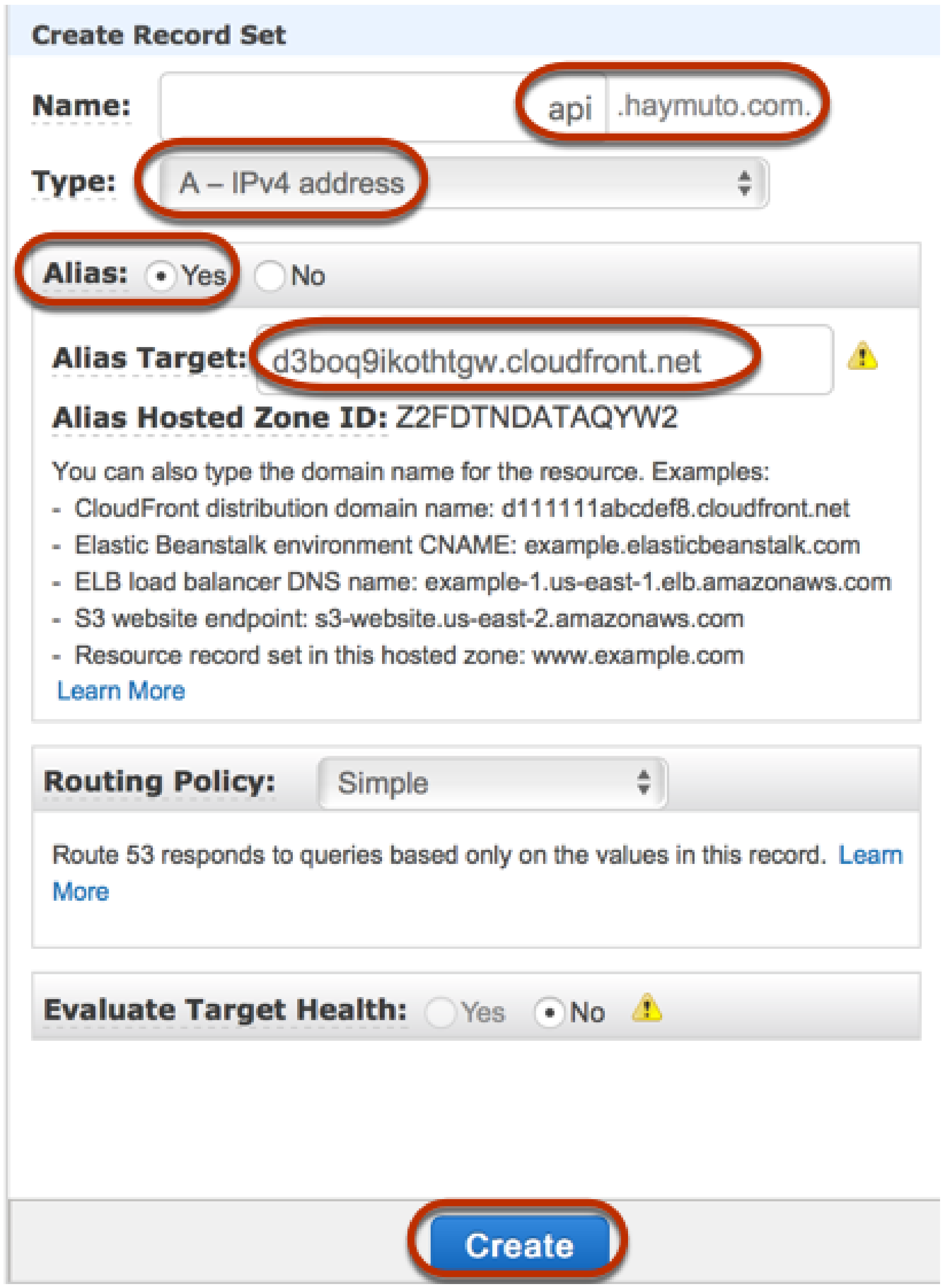
   d. Choose **Save**.
5. After the custom domain name is created, the console displays the associated CloudFront distribution domain name, in the form of *`distribution-id`*`.cloudfront.net`, along with the certificate ARN. Note the CloudFront distribution domain name shown in the output. You need it in the next step to set the custom domain's CNAME value or A-record alias target in your DNS.

   Note
   The newly created custom domain name takes about 40 minutes to be ready. In the meantime, you can configure the DNS record alias to map the custom domain name to the associated CloudFront distribution domain name and to set up the base path mapping for the custom domain name while the custom domain name is being initialized.
6. In this step, we use Amazon Route 53 as an example DNS provider to show how to set up an A-record alias for your Internet domain to map the custom domain name to the associated CloudFront distribution name. The instructions can be adapted to other DNS providers.

   a. Sign in to the Amazon Route 53 console.

   b. Create an `A-IPv4 address` record set for your custom domain (e.g., `api.haymuto.com`). An A-record maps a custom domain name to an IP4 address.

   c. Choose **Yes** for **Alias**, type the CloudFront domain name (e.g., `d3boq9ikothtgw.cloudfront.net`) in **Alias Target**, and then choose **Create**. The A-record alias here maps your custom domain name to the specified CloudFront domain name that is itself mapped to an IP4 address.



Create Record Set

Name: api .haymuto.com.

Type: A – IPv4 address

Alias: [✓ Yes] No

Alias Target: d3boq9ikothtgw.cloudfront.net

Alias Hosted Zone ID: Z2FDTNDATAQYW2

You can also type the domain name for the resource. Examples:
- CloudFront distribution domain name: d111111abcdef8.cloudfront.net
- Elastic Beanstalk environment CNAME: example.elasticbeanstalk.com
- ELB load balancer DNS name: example-1.us-east-1.elb.amazonaws.com
- S3 website endpoint: s3-website.us-east-2.amazonaws.com
- Resource record set in this hosted zone: www.example.com

Learn More

Routing Policy: Simple

Route 53 responds to queries based only on the values in this record. Learn More

Evaluate Target Health: Yes [✓ No]

Tip
The **Alias Hosted Zone ID** identifies the hosted zone of the specified **Alias Target**. The Amazon Route 53 console automatically fills in the value when you enter a valid domain name for **Alias Target**. To create an A-record alias without using the Amazon Route 53 console, such as when you use the AWS CLI, you must specified the required hosted zone

Id. For any CloudFront distribution domain name, the hosted zone Id value is always `Z2FDTNDATAQYW2`, as documented in AWS Regions and Endpoints for CloudFront.

For most DNS providers, a 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 in **Domain Name** (for example, `api.haymuto.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.haymuto.com`). A zone apex is also commonly known as the root domain of your organization. For a zone apex you need to use an A-record alias, provided that is supported by your DNS provider.

With Amazon Route 53 you can create an A record alias for your custom domain name and specify the CloudFront distribution domain name as the alias target, as shown above. 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*.

Use of A-record aliases also eliminates exposure of the underlying CloudFront distribution domain name because the domain name mapping takes place solely within Amazon Route 53. For these reasons, we recommend that you use Amazon Route 53 A-record alias whenever possible.

In addition to using the API Gateway console, you can use the API Gateway REST API, AWS CLI or one of the AWS SDKs to set up the custom domain name for your APIs. As an illustration, the following procedure outlines the steps to do so using the REST API calls.

To set up a custom domain name using the API Gateway REST API

1. Call domainname:create, specifying the custom domain name and the ARN of a certificate stored in AWS Certificate Manager.

   The successful API call returns a `201 Created` response containing the certificate ARN as well as the associated CloudFront distribution name in its payload.
2. Note the CloudFront distribution domain name shown in the output. You need it in the next step to set the custom domain's CNAME value or A-record alias target in your DNS.
3. Follow Step 6 of the previous procedure to set up an A-record alias to map the custom domain name to its CloudFront distribution name.

For code examples of this REST API call, see domainname:create.

Log Custom Domain Name Creation in CloudTrail

When CloudTrail is enabled for logging API Gateway calls made by your account, API Gateway logs the associated CloudFront distribution updates when a custom domain name is created or updated for an API. Because these CloudFront distributions are owned by API Gateway, each of these reported CloudFront distributions is identified by one of the following region-specific API Gateway account IDs, instead of the API owner's account ID.

Region-specific API Gateway account IDs of CloudFront distributions associated with a custom domain name

Region	Account ID
us-east-1	392220576650
us-east-2	718770453195

Region	Account ID
us-west-1	968246515281
us-west-2	109351309407
eu-west-1	631144002099
eu-west-2	544388816663
eu-central-1	474240146802
ap-northeast-1	969236854626
ap-northeast-2	020402002396
ap-southeast-1	195145609632
ap-southeast-2	798376113853

Configure Base Path Mapping of an API with a Custom Domain Name as its Host Name

You can use a single custom domain name as the host name of multiple APIs. You achieve this by configuring the base path mappings on the custom domain name. With the base path mappings, an API under the custom domain is accessible through the combination of the custom domain name and the associated base path.

For example, if you created an API named `PetStore` and another API named `PetShop` and set up a custom domain name of `api.haymuto.com` in API Gateway, you can set the `PetStore` API's URL as `https://api.haymuto.com` or `https://api.haymuto.com/myPetStore`. The `PetStore` API is associated with the base path of an empty string or `myStore` under the custom domain name of `api.haymuto.com`. Similarly, you can assign a base path of `yourPetStore` for the `PetShop` API. The URL of `https://api.haymuto.com/yourPetStore` is then the root URL of the `PetShop` API.

Before setting the base path for an API, complete the steps in Set Up a Custom Domain Name for an API Gateway API (p. 324).

To set the base path for API mappings using the API Gateway console

1. Choose a custom domain name from the list of available **Custom Domain Names** list under your account.
2. Choose **Show Base Path Mappings** or **Edit**.
3. Choose **Add mapping**.
4. (Optional) Type a base path name for **Path**, choose an API from **Destination**, and then choose a stage.

   Note
   The **Destination** list shows the deployed APIs under your account.
5. Choose **Save** to finish setting up the base path mapping for the API.

Note
To delete a mapping after you create it, next to the mapping that you want to delete, choose the trash icon.

In addition, you can call the API Gateway REST API, AWS CLI, or one of the AWS SDKs to set up the base path mapping of an API with a custom domain name as its host name. As an illustration, the following procedure outlines the steps to do so using the REST API calls.

To set up the base path mapping of an API using the API Gateway REST API

- Call basepathmapping:create on a specific custom domain name, specifying the `basePath`, `restApiId`, and a deployment `stage` property in the request payload.

  The successful API call returns a `201 Created` response.

For code examples of the REST API call, see basepathmapping:create.

Rotate a Certificate Imported into ACM

ACM automatically handles renewal of certificates it issues. You do not need to rotate any ACM-issued certificates for your custom domain names. CloudFront handles it on your behalf.

However, if you import a certificate into ACM and use it for a custom domain name, you must rotate the certificate before it expires. This involves importing a new third-party certificate for the domain name and rotate the existing certificate to the new one. You need to repeat the process when the newly imported certificate expires. Alternatively, you can request ACM to issue a new certificate for the domain name and rotate the existing one to the new ACM-issued certificate. After that, you can leave ACM and CloudFront to handle the certificate rotation for you automatically. To create or import a new ACM Certificate, follow the steps to request or import a new ACM Certificate (p. 322) for the specified domain name.

To rotate a certificate for a domain name, you can use the API Gateway console, the API Gateway REST API, AWS CLI, or one of the AWS SDKs.

To rotate an expiring certificate imported into ACM using the API Gateway console

1. Request or import a certificate in ACM.
2. Go back to the API Gateway console.
3. Choose **Custom Domain Names** from the API Gateway console main navigation pane.
4. Select the custom domain name of your choice, under the **Custom Domain Names** pane.
5. Choose **Edit**.
6. Choose the desired certificate from the **ACM Certificate** dropdown list.
7. Choose **Save** to begin rotating the certificate for the custom domain name.

   Note
   It takes about 40 minutes for the process to finish. After the rotation is done, you can choose the two-way arrow icon next to **ACM Certificate** to roll back to the original certificate.

To illustrate how to programmatically rotate an imported certificate for a custom domain name, we outline the steps using the API Gateway REST API.

Rotate an imported certificate using the API Gateway REST API

- Call domainname:update action, specifying the ARN of the new ACM Certificate for the specified domain name.

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.

The following examples compare and contrast a set of default URLs and corresponding custom URLs of two APIs (`udxjef` and `qf3duz`) in a specified region (`us-east-1`), and of a given custom domain name (`apis.example.com`).

Root URLs of APIs with default and custom domain names

API ID	Stage	Default URL	Base path	Custom URL
udxjef	pro	https://udxjef.execute-api.us-east-1.amazonaws.com/pro	/petstore	https://apis.example.com/petstore
udxjef	tst	https://udxjef.execute-api.us-east-1.amazonaws.com/tst	/petdepot	https://apis.example.com/petdepot
qf3duz	dev	https://qf3duz.execute-api.us-east-1.amazonaws.com/dev	/bookstore	https://apis.example.com/bookstore
qf3duz	tst	https://qf3duz.execute-api.us-east-1.amazonaws.com/tst	/bookstand	https://apis.example.com/bookstand

API Gateway supports custom domain names for an API by using Server Name Indication (SNI). You can invoke the API with a custom domain name 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.

Sell Your 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. 267).

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

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"
    }]
}
```

Calling an API in Amazon API Gateway

Calling a deployed API involves submitting requests to the API Gateway component service for API execution, known as `execute-api`. The root URL of such requests is of the following format:

```
https://{restapi_id}.execute-api.{region}.amazonaws.com/{stage_name}/
```

where *{restapi_id}* is the API identifier, *{region}* is the API deployed region, and *{stage_name}* is the stage name of an API deployment.

You can find this root URL in the given **Stage Editor**. It is listed as the **Invoke URL** at the top. If the API's root resource exposes a GET method without requiring user authentication, you can call the method by clicking the **Invoke URL** link. You can also construct this root URL by combining the `host` and `basePath` fields of an exported Swagger definition file of the API.

If an API permits anonymous access, you can use any web browser to invoke any GET-method calls by copying and pasting an appropriate invocation 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 app to test a successfully deployed API, without writing a script or a client.

Note
Query string parameter values in an invocation URL cannot contain `%%`.

Topics

You must have already deployed the API in API Gateway. Follow the instructions in Deploying an API (p. 284).

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. 282).

To call the method directly from the API Gateway console, see Use the Console to Test a Method (p. 334).

For more options, contact the API owner.

Use the API Gateway Console to Test a Method

Use the API Gateway console to test a method.

Topics

Prerequisites

- You must specify the settings for the methods you want to test. Follow the instructions in Set up Method and Integration (p. 81).

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 Call an API

Use the Postman app 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. 237).

Use a Java SDK Generated by API Gateway

In this section, we outline the steps to use a Java SDK generated by API Gateway, by using the Simple Calculator (p. 315) API as an example. Before proceeding, you must complete the steps in Generate SDKs for an API Using the API Gateway Console (p. 307).

To install and use a Java 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 (must be version 3.5 or later).
3. Download and install the JDK (must be version 1.8 or later).
4. Set the `JAVA_HOME` environment variable.
5. Go to the unzipped SDK folder where the pom.xml file is located. This folder is `generated-code` by default. 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. Type the following command to create a client project stub to call the API using the installed SDK library.

   ```
   mvn -B archetype:generate \
       -DarchetypeGroupdId=org.apache.maven.archetypes \
       -DgroupId=examples.aws.apig.simpleCalc.sdk.app \
       -DartifactId=SimpleCalc-sdkClient
   ```

   Note
   The separator `\` in the preceding command is included for readability. The whole command should be on a single line without the separator.

   This command creates an application stub. The application stub contains a `pom.xml` file and an `src` folder under the project's root directory (*`SimpleCalc-sdkClient`* in the preceding command). Initially, there are two source files: `src/main/java/{package-path}/App.java` and `src/test/java/{package-path}/AppTest.java`. In this example, *`{package-path}`* is `examples/aws/apig/simpleCalc/sdk/app`. This package path is derived from the `DarchetypeGroupdId` value. You can use the `App.java` file as a template for your client application, and you can add others in the same folder if needed. You can use the `AppTest.java` file as a unit test template for your application, and you can add other test code files to the same test folder as needed.
7. Update the package dependencies in the generated `pom.xml` file to the following, substituting your project's `groupId`, `artifactId`, `version`, and `name` properties, if necessary:

   ```
   <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/
   XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://
   maven.apache.org/maven-v4_0_0.xsd">
     <modelVersion>4.0.0</modelVersion>
     <groupId>examples.aws.apig.simpleCalc.sdk.app</groupId>
     <artifactId>SimpleCalc-sdkClient</artifactId>
     <packaging>jar</packaging>
     <version>1.0-SNAPSHOT</version>
     <name>SimpleCalc-sdkClient</name>
     <url>http://maven.apache.org</url>

      <dependencies>
         <dependency>
             <groupId>com.amazonaws</groupId>
             <artifactId>aws-java-sdk-core</artifactId>
             <version>1.11.69</version>
         </dependency>
         <dependency>
             <groupId>my-apig-api-examples</groupId>
             <artifactId>simple-calc-sdk</artifactId>
             <version>1.0.0</version>
         </dependency>

       <dependency>
         <groupId>junit</groupId>
         <artifactId>junit</artifactId>
         <version>4.12</version>
   ```

```
      <scope>test</scope>
    </dependency>

    <dependency>
        <groupId>commons-io</groupId>
        <artifactId>commons-io</artifactId>
        <version>2.5</version>
    </dependency>
  </dependencies>

  <build>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>3.5.1</version>
        <configuration>
          <source>1.8</source>
          <target>1.8</target>
        </configuration>
      </plugin>
    </plugins>
  </build>
</project>
```

8. Next, we show how to call the API using the SDK by calling the `getABOp(GetABOpRequest req)`, `getApiRoot(GetApiRootRequest req)`, and `postApiRoot(PostApiRootRequest req)` methods of the SDK. These methods correspond to the `GET /{a}/{b}/{op}`, `GET /?a={x}&b={y}&op={operator}`, and `POST /` methods, with a payload of `{"a": x, "b": y, "op": "operator"}` API requests, respectively.

   Update the `App.java` file as follows:

```
package examples.aws.apig.simpleCalc.sdk.app;

import java.io.IOException;

import com.amazonaws.opensdk.config.ConnectionConfiguration;
import com.amazonaws.opensdk.config.TimeoutConfiguration;

import examples.aws.apig.simpleCalc.sdk.*;
import examples.aws.apig.simpleCalc.sdk.model.*;
import examples.aws.apig.simpleCalc.sdk.SimpleCalcSdk.*;

public class App
{
    SimpleCalcSdk sdkClient;

    public App() {
        initSdk();
    }

    // The configuration settings are for illustration purposes and may not be a
 recommended best practice.
    private void initSdk() {
        sdkClient = SimpleCalcSdk.builder()
              .connectionConfiguration(
                  new ConnectionConfiguration()
                        .maxConnections(100)
                        .connectionMaxIdleMillis(1000))
              .timeoutConfiguration(
                  new TimeoutConfiguration()
                        .httpRequestTimeout(3000)
                        .totalExecutionTimeout(10000)
                        .socketTimeout(2000))
        .build();

    }
    // Calling shutdown is not necessary unless you want to exert explicit control of
 this resource.
    public void shutdown() {
        sdkClient.shutdown();
    }

    // GetABOpResult getABOp(GetABOpRequest getABOpRequest)
    public Output getResultWithPathParameters(String x, String y, String operator) {
     operator = operator.equals("+") ? "add" : operator;
     operator = operator.equals("/") ? "div" : operator;

        GetABOpResult abopResult = sdkClient.getABOp(new
 GetABOpRequest().a(x).b(y).op(operator));
        return abopResult.getResult().getOutput();
    }

    public Output getResultWithQueryParameters(String a, String b, String op) {
        GetApiRootResult rootResult = sdkClient.getApiRoot(new
 GetApiRootRequest().a(a).b(b).op(op));
        return rootResult.getResult().getOutput();
    }

    public Output geResultByPostInputBody(Double x, Double y, String o) {
     PostApiRootResult postResult = sdkClient.postApiRoot(
      new PostApiRootRequest().input(new Input().a(x).b(y).op(o)));
     return postResult.getResult().getOutput();
    }

    public static void main( String[] args )
    {
        System.out.println( "Simple calc" );
        // to begin
        App calc = new App();

        // call the SimpleCalc API
        Output res = calc.getResultWithPathParameters("1", "2", "-");
        System.out.printf("GET /1/2/-: %s\n", res.getC());

        // Use the type query parameter
        res = calc.getResultWithQueryParameters("1", "2", "+");
        System.out.printf("GET /?a=1&b=2&op=+: %s\n", res.getC());

        // Call POST with an Input body.
        res = calc.geResultByPostInputBody(1.0, 2.0, "*");
        System.out.printf("PUT /\n\n{\"a\":1, \"b\":2,\"op\":\"*\"}\n %s\n",
 res.getC());


    }
}
```

In the preceding example, the configuration settings used to instantiate the SDK client are for illustration purposes and are not necessarily recommended best practice. Also, calling `sdkClient.shutdown()` is optional, especially if you need precise control on when to free up resources.

We have shown the essential patterns to call an API using a Java SDK. You can extend the instructions to calling other API methods.

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 Generate SDKs for an API Using the API Gateway Console (p. 307).

Note
The generated SDK is not compatible with Android 4.4 and earlier. For more information, see Known Issues (p. 453).

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 a JavaScript SDK Generated by API Gateway

Note
These instructions assume you have already completed the instructions in Generate SDKs for an API Using the API Gateway Console (p. 307).

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. 228).
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 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. 310) 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 Generated iOS SDK (Objective-C) to Call API

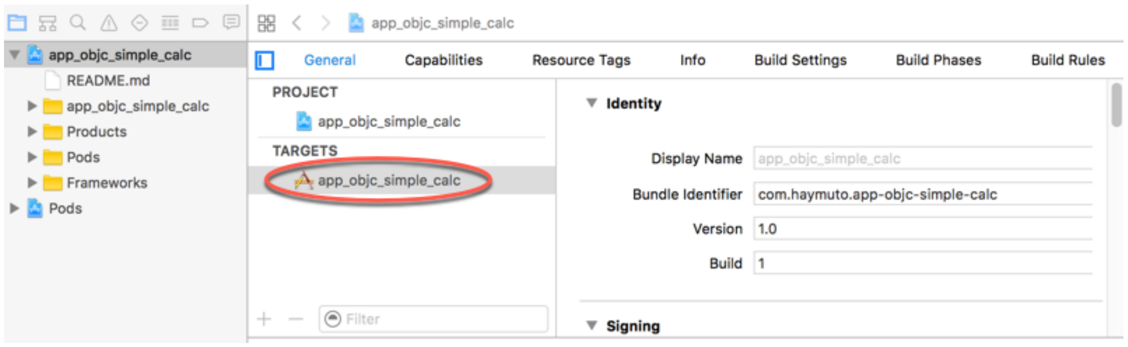
Before beginning the following procedure, you must complete the steps in Generate SDKs for an API Using the API Gateway Console (p. 307) for iOS in Objective-C and download the .zip file of the generated SDK.

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. 310), 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`.



Display Name: app_objc_simple_calc
Bundle Identifier: com.haymuto.app-objc-simple-calc
Version: 1.0
Build: 1

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'
      ```

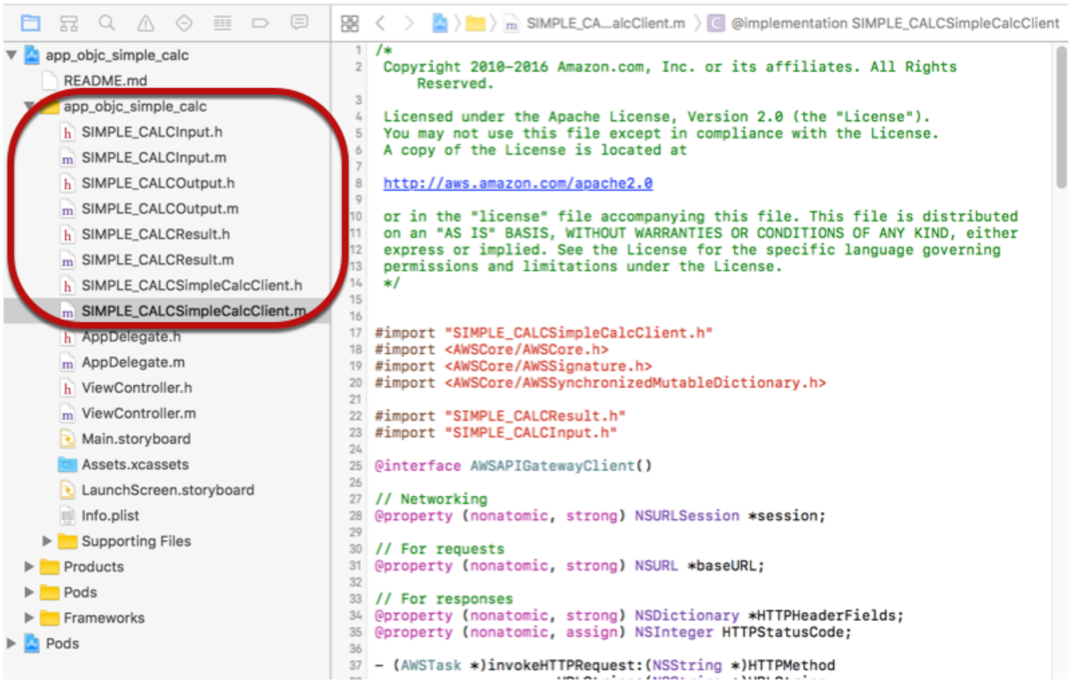
   3. Open a terminal window and run the following command:

      ```
      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.



```
/*
 Copyright 2010-2016 Amazon.com, Inc. or its affiliates. All Rights
     Reserved.

 Licensed under the Apache License, Version 2.0 (the "License").
 You may not use this file except in compliance with the License.
 A copy of the License is located at

 http://aws.amazon.com/apache2.0

 or in the "license" file accompanying this file. This file is distributed
 on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
 express or implied. See the License for the specific language governing
 permissions and limitations under the License.
 */


#import "SIMPLE_CALCSimpleCalcClient.h"
#import <AWSCore/AWSCore.h>
#import <AWSCore/AWSSignature.h>
#import <AWSCore/AWSSynchronizedMutableDictionary.h>

#import "SIMPLE_CALCResult.h"
#import "SIMPLE_CALCInput.h"

@interface AWSAPIGatewayClient()

// Networking
@property (nonatomic, strong) NSURLSession *session;

// For requests
@property (nonatomic, strong) NSURL *baseURL;

// For responses
@property (nonatomic, strong) NSDictionary *HTTPHeaderFields;
@property (nonatomic, assign) NSInteger HTTPStatusCode;

- (AWSTask *)invokeHTTPRequest:(NSString *)HTTPMethod
```

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. 310) 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. 309) in the backend does not handle URL encoded path variables.

Use Generated iOS SDK (Swift) to Call API

(

Before beginning the following procedure, you must complete the steps in Generate SDKs for an API Using the API Gateway Console (p. 307) 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

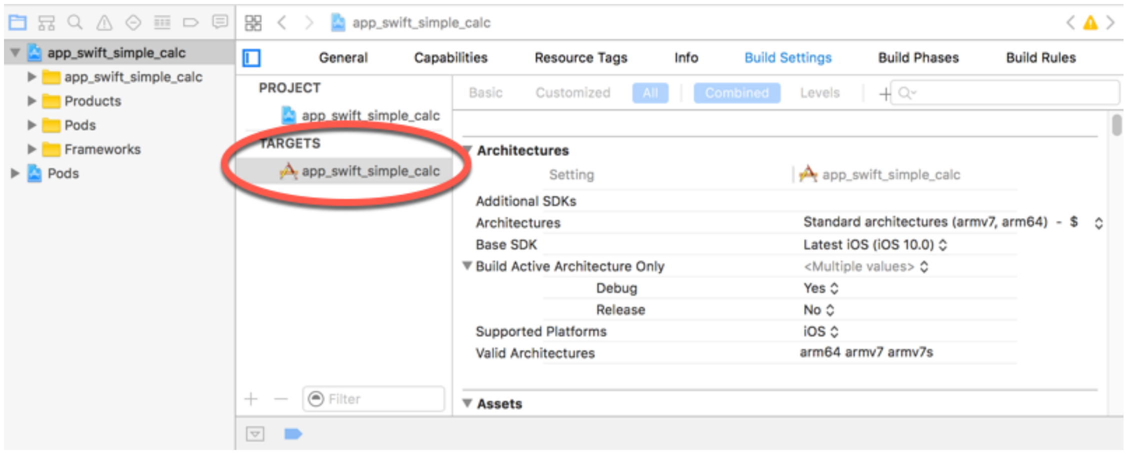
The following procedure describes how to install the SDK.

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. 310), 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`.



Architectures	
Setting	app_swift_simple_calc
Additional SDKs	
Architectures	Standard architectures (armv7, arm64) - $
Base SDK	Latest iOS (iOS 10.0)
Build Active Architecture Only	<Multiple values>
Debug	Yes
Release	No
Supported Platforms	iOS
Valid Architectures	arm64 armv7 armv7s

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'
   ```

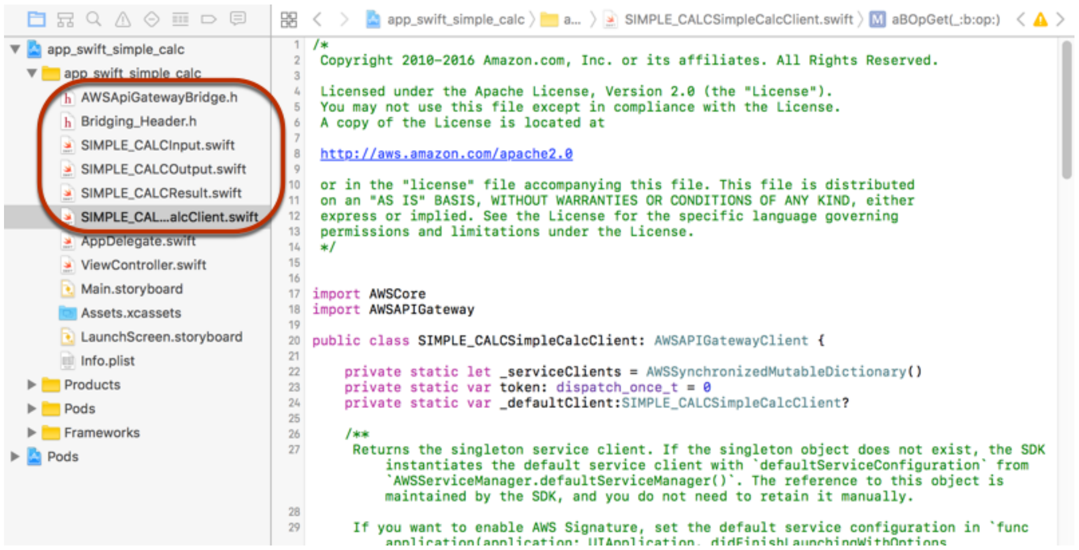
   3. Open a terminal window and run the following command in the app directory:

   ```
   pod install
   ```

   This installs the API Gateway component and any dependent AWS Mobile SDK components into the app's project.

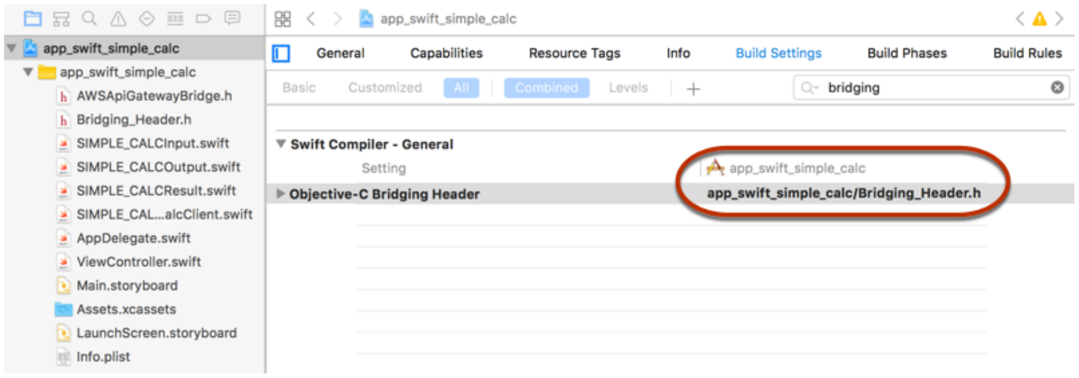
   4. Close the Xcode project and then open the `*.xcworkspace` file to relaunch Xcode.

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.



```
/*
 Copyright 2010-2016 Amazon.com, Inc. or its affiliates. All Rights Reserved.

 Licensed under the Apache License, Version 2.0 (the "License").
 You may not use this file except in compliance with the License.
 A copy of the License is located at

 http://aws.amazon.com/apache2.0

 or in the "license" file accompanying this file. This file is distributed
 on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
 express or implied. See the License for the specific language governing
 permissions and limitations under the License.
 */


import AWSCore
import AWSAPIGateway

public class SIMPLE_CALCSimpleCalcClient: AWSAPIGatewayClient {

    private static let _serviceClients = AWSSynchronizedMutableDictionary()
    private static var token: dispatch_once_t = 0
    private static var _defaultClient:SIMPLE_CALCSimpleCalcClient?

    /**
     Returns the singleton service client. If the singleton object does not exist, the SDK
         instantiates the default service client with `defaultServiceConfiguration` from
         `AWSServiceManager.defaultServiceManager()`. The reference to this object is
         maintained by the SDK, and you do not need to retain it manually.

     If you want to enable AWS Signature, set the default service configuration in `func
         application(application: UIApplication, didFinishLaunchingWithOptions
```

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:

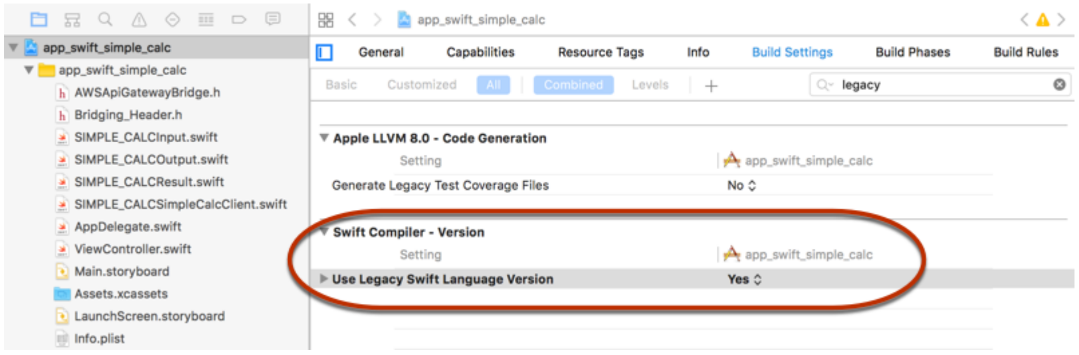


Swift Compiler - General

Setting	app_swift_simple_calc
Objective-C Bridging Header	app_swift_simple_calc/Bridging_Header.h

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:



Apple LLVM 8.0 - Code Generation

Setting	app_swift_simple_calc
Generate Legacy Test Coverage Files	No

Swift Compiler - Version

Setting	app_swift_simple_calc
Use Legacy Swift Language Version	Yes

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. 310) 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.USEast1,
    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 div 2 = 0.5`. Here, `div` is used in place of `/` because the simple Lambda function (p. 309) in the backend does not handle URL encoded path variables.

Trace API Calls with CloudWatch Logs

Topics

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. 286) 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. 449). 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 backend.
- 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

The metrics and dimensions that API Gateway sends to Amazon CloudWatch are listed below. For more information, see Monitor API execution with Amazon CloudWatch (p. 353).

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

Metric	**Description**
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. The latency includes the integration latency and other API Gateway overhead. 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 stage:update action of the API Gateway REST API to update the `metricsEnabled` property to `true`. Enabling such metrics will incur additional charges to your account. For pricing information, see Amazon CloudWatch Pricing.
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

1. You must have an API created in API Gateway. Follow the instructions in Creating an API (p. 72).
2. You must have the API deployed at least once. Follow the instructions in Deploying an API (p. 284).
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 Update Stage Settings (p. 286). 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 Update Stage Settings Using the API Gateway Console (p. 287).

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. 353).
- **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.

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. 305).

Topics

- x-amazon-apigateway-request-validators.requestValidator Object (p. 375)

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. 366)	Specifies the integration of the method with the backend. This is an extended property of the Swagger Operation object. The integration can be of type `AWS`, `AWS_PROXY`, `HTTP`, `HTTP_PROXY`, or `MOCK`.

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, `"arn:aws:apigateway:us-east-1:lambda:path/2015-03-31/functions/arn:aws:lambda:us-east-1:account-id:function:auth_function_name/invocations"`
authorizerCredentials	`string`	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 `DocumentationPart` 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-gateway-responses Object

Defines the gateway responses for an API as a string-to-GatewayResponse map of key-value pairs.

Properties

Property Name	Type	Description
responseType	x-amazon-apigateway-gateway-responses.gatewayResponse (p. 363)	A `GatewayResponse` for the specified *responseType*.

x-amazon-apigateway-gateway-responses Example

The following API Gateway extension to Swagger example defines a GatewayResponses map containing two GatewayResponse instances, one for the `DEFAULT_4XX` type and another for the `INVALID_API_KEY` type.

```
"{
  x-amazon-apigateway-gateway-responses": {
    "DEFAULT_4XX": {
      "responseParameters": {
        "gatewayresponse.header.Access-Control-Allow-Origin": "'domain.com'"
      },
      "responseTemplates": {
        "application/json": "{\"message\": test 4xx b }"
      }
    },
    "INVALID_API_KEY": {
      "statusCode": "429",
      "responseTemplates": {
        "application/json": "{\"message\": test forbidden }"
      }
    }
  }
}
```

x-amazon-apigateway-gateway-responses.gatewayResponse Object

Defines a gateway response of a given response type, including the status code, any applicable response parameters, or response templates.

Properties

Property Name	Type	Description
responseParameters	x-amazon-apigateway-gateway-responses.responseParameters (p. 364)	Specifies the GatewayResponse parameters, namely the header parameters. The parameter values can take any incoming request parameter (p. 143) value or a static custom value.
responseTemplates	x-amazon-apigateway-gateway-responses.responseTemplates (p. 365)	Specifies the mapping templates of the gateway response. The templates are not processed by the VTL engine.
statusCode	`string`	An HTTP status code for the gateway response.

x-amazon-apigateway-gateway-responses.gatewayResponse Example

The following example of the API Gateway extension to Swagger defines a GatewayResponse to customize the `INVALID_API_KEY` response to return the status code of `456`, the incoming request's `api-key` header value, and `"Bad api-key"` message.

```
"INVALID_API_KEY": {
  "statusCode": "456",
  "responseParameters": {
    "gatewayresponse.header.api-key": "method.request.header.api-key"
  },
  "responseTemplates": {
    "application/json": "{\"message\": \"Bad api-key\" }"
  }
}
```

x-amazon-apigateway-gateway-responses.responseParameters Object

Defines a string-to-string map of key-value pairs to generate gateway response parameters from the incoming request parameters or using literal strings.

Properties

Property Name	Type	Description
`gatewayresponse.`*`param-position`*`.`*`param-name`*	`string`	*`param-position`* can be `header`, `path` or `querystring`. For more information, see Map Method Request Data to Integration Request Parameters (p. 143).

x-amazon-apigateway-gateway-responses.repsonseParameters Example

The following Swagger extensions example shows a GatewayResponse response parameter mapping expression to enable CORS support for resources on the `*.example.domain` domains.

```
    "responseParameters": {
      "gatewayresponse.header.Access-Control-Allow-Origin": '*.example.domain',
      "gatewayresponse.header.from-request-header" : method.request.header.Accept,
      "gatewayresponse.header.from-request-path" : method.request.path.petId,
      "gatewayresponse.header.from-request-query" : method.request.querystring.qname
    }
```

x-amazon-apigateway-gateway-responses.responseTemplates Object

Defines GatewayResponse mapping templates, as a string-to-string map of key-value pairs, for a given gateway response. For each key-value pair, the key is the content type; for example, "application/json", and the value is a stringified mapping template for simple variable substitutions. A `GatewayResponse` mapping template is not processed by the Velocity Template Language (VTL) engine.

Properties

Property Name	Type	Description
content-type	`string`	A `GatewayResponse` body mapping template supporting only simple variable substitution to customize a gateway response body.

x-amazon-apigateway-gateway-responses.responseTemplates Example

The following Swagger extensions example shows a GatewayResponse mapping template to customize an API Gateway-generated error response into an app-specific format.

```
    "responseTemplates": {
      "application/json": "{ \"message\": $context.error.messageString, \"type\":
$context.error.responseType, \"statusCode\": '488' }
    }
```

The following Swagger extensions example shows a GatewayResponse mapping template to override an API Gateway-generated error response with a static error message.

```
    "responseTemplates": {
      "application/json": "{ \"message\": 'API-specific errors' }" }
    }
```

x-amazon-apigateway-integration Object

Specifies details of the backend 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
cacheKeyParameters	An array of `string`	A list of request parameters whose values are to be cached.
cacheNamespace	`string`	An API-specific tag group of related cached parameters.
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.
contentHandling	`string`	Request payload encoding conversion types. Valid values are 1) `CONVERT_TO_TEXT`, 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) `CONVERT_TO_BINARY`, for converting a text payload into Base64-decoded blob or passing through a binary payload natively without modification.
httpMethod	`string`	The HTTP method used in the integration request. For Lambda function invocations, the value must be `POST`.
passthroughBehavior	`string`	Specifies how a request payload of unmapped content

Property Name	Type	Description
		type is passed through the integration request without modification. Supported values are `when_no_templates`, `when_no_match`, and `never`. For more information, see Integration.passthroughBehavior.
requestParameters	x-amazon-apigateway-integration.requestParameters (p. 369)	Specifies mappings from method request parameters to integration request parameters. Supported request parameters are `querystring`, `path`, `header`, and `body`.
requestTemplates	x-amazon-apigateway-integration.requestTemplates (p. 368)	Mapping templates for a request payload of specified MIME types.
responses	x-amazon-apigateway-integration.responses (p. 370)	Defines the method's responses and specifies desired parameter mappings or payload mappings from integration responses to method responses.
type	`string`	The type of integration with the specified backend. The valid value is `http` (for integration with an HTTP backend) or `aws` (for integration with AWS Lambda functions or other AWS services, such as Amazon DynamoDB, Amazon Simple Notification Service or Amazon Simple Queue Service).
uri	`string`	The endpoint URI of the backend. For integrations of the `aws` type, this is an ARN value. For the HTTP integration, this is the URL of the HTTP endpoint including the `https` or `http` scheme.

x-amazon-apigateway-integration Example

The following example integrates an API's `POST` method with a Lambda function in the backend. 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.

Properties

Property Name	Type	Description
MIME type	string	An example of the MIME type is `application/json`. For information about creating a

Property Name	Type	Description
		mapping template, see Mapping Templates (p. 123).

x-amazon-apigateway-integration.requestTemplates Example

The following example sets mapping templates for a request payload of the `application/json` and `application/xml` MIME types.

```
"requestTemplates" : {
    "application/json" : "#set ($root=$input.path('$')) { \"stage\": \"$root.name\",
 \"user-id\": \"$root.key\" }",
    "application/xml" : "#set ($root=$input.path('$')) <stage>$root.name</stage> "
}
```

x-amazon-apigateway-integration.requestParameters Object

Specifies mappings from named method request parameters to integration request parameters. The method request parameters must be defined before being referenced.

Properties

Property Name	Type	Description
`integration.request.<param-type>.<param-name>`	`string`	The value must be a predefined method request parameter of the `method.request.<param-type>.<param-name>` format, where `<param-type>` can be `querystring`, `path`, `header`, or `body`. For the `body` parameter, the `<param-name>` is a JSON path expression without the `$.` prefix.

`x-amazon-apigateway-integration.requestParameters` Example

The following request parameter mappings example translates a method request's query (`version`), header (`x-user-id`) and path (`service`) parameters to the integration request's query (`stage`), header (`x-userid`), and path parameters (`op`), respectively.

```
"requestParameters" : {
    "integration.request.querystring.stage" : "method.request.querystring.version",
    "integration.request.header.x-userid" : "method.request.header.x-user-id",
    "integration.request.path.op" : "method.request.path.service"
},
```

x-amazon-apigateway-integration.responses Object

Defines the method's responses and specifies parameter mappings or payload mappings from integration responses to method responses.

Properties

Property Name	Type	Description
`Response status pattern`	x-amazon-apigateway-integration.response (p. 371)	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 `errorMessage` field of the error information object returned by AWS Lambda as a failure response body when the Lambda function execution throws an exception. **Note** The *`Response status pattern`* 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.

`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. 372)	Specifies MIME type-specific mapping templates for the response's payload.
responseParameters	x-amazon-apigateway-integration.responseParameters (p. 373)	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) `CONVERT_TO_TEXT`, 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) `CONVERT_TO_BINARY`, for converting a text payload into Base64-decoded blob or passing

Property Name	Type	Description
		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 backend. 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
`MIME type`	`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. 123). An example of the *`MIME type`* is `application/json`.

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
`method.response.header.`*`<param-name>`*	`string`	The named parameter value can be derived from the `header` and `body` 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"
}
```

x-amazon-apigateway-request-validator Property

Specifies a request validator, by referencing a *`request_validator_name`* of the x-amazon-apigateway-request-validators (p. 374) map, to enable request validation on the containing API or a method. The value of this extension is a JSON string.

This extension can be specified at the API level or at the method level. The API-level validator applies to all of the methods unless it is overridden by the method-level validator.

`x-amazon-apigateway-request-validator` Example

The following example applies the `basic` request validator at the API level while applying the `parameter-only` request validator on the `POST /validation` request.

```
{
  "swagger": "2.0",
  "x-amazon-apigateway-request-validators" : {
    "basic" : {
      "validateRequestBody" : true,
      "validateRequestParameters" : true
    },
    "params-only" : {
      "validateRequestBody" : false,
      "validateRequestParameters" : true
    }
  },
  "x-amazon-apigateway-request-validator" : "basic",
  "paths": {
    "/validation": {
      "post": {
        "x-amazon-apigateway-request-validator" : "params-only",
       ...
     }
}
```

x-amazon-apigateway-request-validators Object

Defines the supported request validators for the containing API as a map between a validator name and the associated request validation rules. This extension applies to an API.

Properties

Property Name	Type	Description
request_validator_name	x-amazon-apigateway-request-validators.requestValidator (p. 375)	Specifies the validation rules consisting of the named validator. For example: `"basic" : { "validateRequestBody" : true, "validateRequestParameters" : true },` To apply this validator to a specific method, reference the validator name (`basic`) as the value of the x-amazon-apigateway-request-validator (p. 373) property.

`x-amazon-apigateway-request-validators` Example

The following example shows a set of request validators for an API as a map between a validator name and the associated request validation rules.

```
{
  "swagger": "2.0",
  ...
  "x-amazon-apigateway-request-validators" : {
    "basic" : {
      "validateRequestBody" : true,
      "validateRequestParameters" : true
    },
    "params-only" : {
      "validateRequestBody" : false,
      "validateRequestParameters" : true
    }
  },
  ...
}
```

x-amazon-apigateway-request-validators.requestValidator Object

Specifies the validation rules of a request validator as part of the x-amazon-apigateway-request-validators (p. 374) map definition.

Properties

Property Name	Type	Description
`validateRequestBody`	`Boolean`	Specifies whether to validate the request body (`true`) or not (`false`).
`validateRequestParameters`	`Boolean`	Specifies whether to validate the required request parameters (`true`) or not (`false`).

`x-amazon-apigateway-request-validators.requestValidator` Example

The following example shows a parameter-only request validator:

```
"params-only": {
    "validateRequestBody" : false,
    "validateRequestParameters" : true
}
```

Samples and Tutorials

The following tutorials provide hands-on exercises to help you learn about API Gateway.

Topics

Create an API Gateway API for AWS Lambda Functions

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.

In the Getting Started (p. 45) section, you learned how to use the API Gateway console to build an API to expose a Lambda function. There, the console let you choose `Lambda Function` for **Integration type**, among other options of `HTTP`, `Mock` and `AWS Service`. The `Lambda Function` option is a special case of the `AWS Service` integration type and simplifies the integration set-up for you with default settings. For example, with the former, the console automatically adds the required resource-based permissions for invoking the Lambda function. With the latter, you have more control, but more responsibilities to set up the integration, including creating and specifying an IAM role containing appropriate permissions. For the both options, the underlying Integration.type is `AWS` in the API Gateway REST API and its Swagger definition file.

In this section, we walk you through the steps to integrate an API with a Lambda function using the `AWS Service` and `Lambda Function` integration types. To support asynchronous invocation of the Lambda function, 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.

You 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 backend Lambda function through three query string parameters (`operand1`, `operand2`, and `operator`). You will set up a mapping template to map these 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. You can pass the method request payload through to the integration request, if the client input conforms to the input model. Alternatively, you 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. You will need to provide a mapping template to translate the path parameters of the method request into an integration request payload as the Lambda function input and to translate the output from the integration responses to the method response.

In this tutorial, we will cover the following topics:

- 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. We will enable a request validator to ensure that the client submit all the required query string parameters before API Gateway calling the Lambda function.
- Expose POST on the `/calc` resource to invoke the Lambda function, supplying the input in the payload. We will enable a request validator to ensure that the client submitted the valid request payload before API Gateway call the Lambda function.
- Expose GET on the `/calc/{operand1}/{operand2}/{operator}` resource to invoke the Lambda function, specifying the input in the path parameters. We also explain how to define a `Result` schema to model the method response body so that any strongly typed SDK of the API can access the method response data through properties defined in the `Result` schema.

You can inspect the sample API in its Swagger definition file (p. 391). You can also import the API Swagger definitions to API Gateway, following the instructions given in Import an API (p. 210).

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 https://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

For API Gateway to invoke a Lambda function, the API must have a permission to call the Lambda's InvokeFunction action. This means that, 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 Backend

The following procedure outlines the steps to create a Lambda function using the Lambda console.

1. Go to the Lambda console.
2. Choose **Create a Lambda function**.
3. Choose the **Blank Function** blueprint for the runtime of Node.js 4.3 or later.
4. Follow the on-screen instructions to the **Lambda function code** editor.
5. Copy the following Lambda function and paste it into the code editor in the Lambda console.

```
console.log('Loading the Calc function');

exports.handler = function(event, context, callback) {
    console.log('Received event:', JSON.stringify(event, null, 2));
    if (event.a === undefined || event.b === undefined || event.op === undefined) {
        callback("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)) {
        callback("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:
            callback("400 Invalid Operator");
            break;
    }
    callback(null, res);
};
```

6. Choose an existing or create a new IAM role
7. Follow the on-screen instructions to finish creating the function.

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 in the next step.

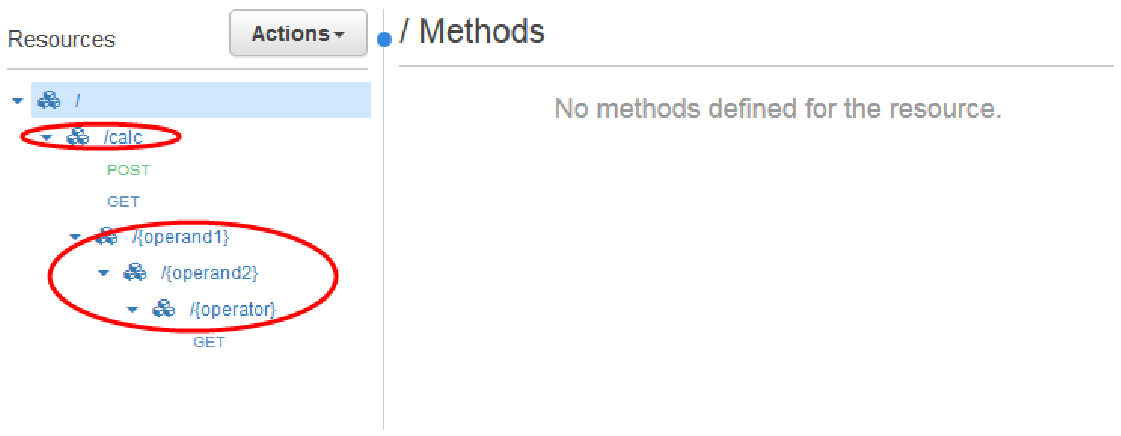
Create API Resources for the Lambda Function

The following procedure describes how to create API resources for the Lambda function. As an illustration, we use multiple API resources and methods to enable different API behaviors for calling the same function.

To create API resources for Lambda functions

1. In the API Gateway console, create an API named **LambdaGate**.
2. Create the **/calc** resource off the API's root. We will expose the GET and POST methods on this resource for the client to invoke the backend Lambda function. The caller must supply 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.

   We will also create the **/calc/{operand1}/{operand2}/{operator}** resource subtree to expose the GET method to invoke the Lambda function. The caller must supply the required input by specifying the three path parameters (**operand1**, **operand2**, and **operator**).



Create a GET Method with Query Parameters to Call the Lambda Function

By creating a GET method with query parameters to call the Lambda function, we can let the API user to do the calculations via any browser. This can be useful especially if the API allows open access.

To set up the GET method with query strings to invoke the Lambda function

1. In the API Gateway console, choose the API's **/calc** resource under **Resources**.
2. Choose **Create Method**, from the **Actions** drop-down menu, to create the GET method.
3. In the ensuing **Set up** pane,

    a. Choose `AWS Service` for **Integration type**.

    b. Choose the region (e.g., `us-west-2`) where you created the Lambda function for **AWS Region**.

    c. Choose `Lambda` for **AWS Service**.

    d. Leave **AWS Subdomain** blank because our Lambda function is not hosted on any of AWS subdomain.

    e. Choose `POST` for **HTTP method**. Lambda requires that the `POST` request be used to invoke any Lambda function. This examples shows that the HTTP method in a frontend method request can be different from the integration request in the backend.

    f. Choose `Use path override` for **Action Type**. This option allows us to specify the ARN of the Invoke action to execute our `Calc` function.

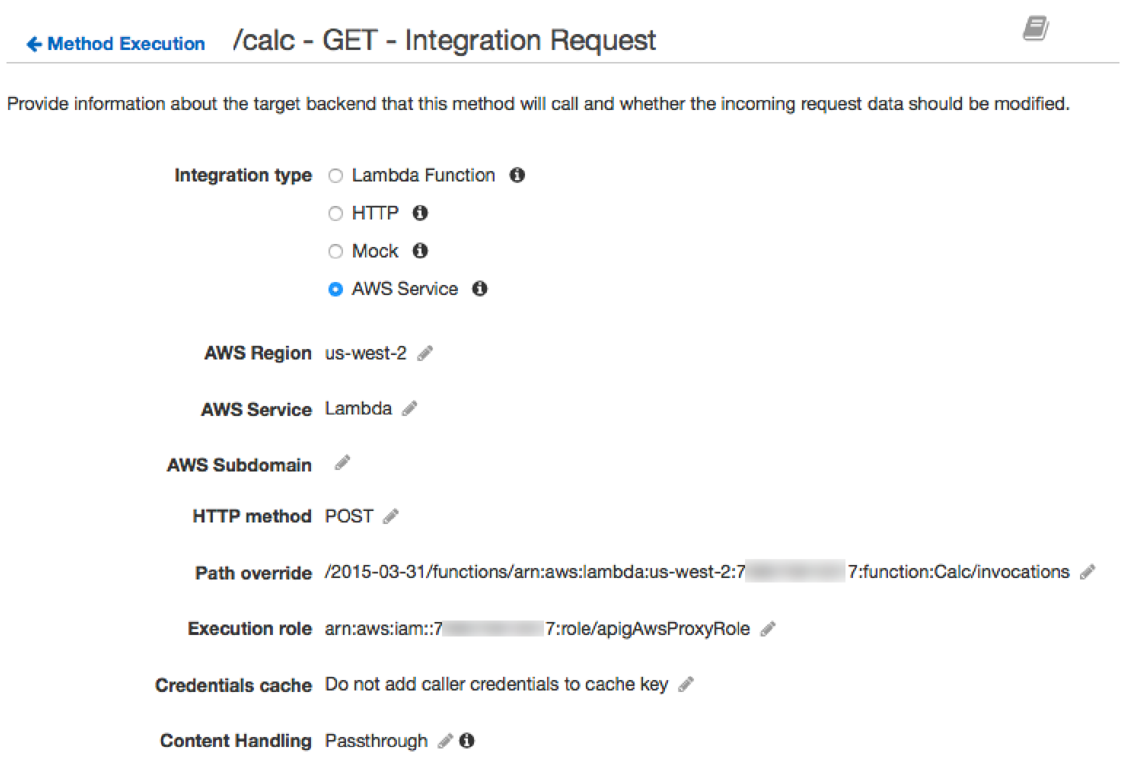
    g. Type `/2015-03-31/functions/arn:aws:lambda:`*`region`*`:`*`account-id`*`:function:Calc/invocations` in **Path override**.

    h. Specify the ARN of an IAM role for **Execution role**. You can find the ARN of the role in the IAM console. The role must contain the necessary permissions for the caller to call the `Calc` function and for API Gateway to assume the role of the caller.

    i. Leave the `Passthrough` as the default of **Content Handling**, because we will not deal with any binary data.

    j. Choose the **Save** to finish setting up the `GET /calc` method.

After the setup succeeds, the configuration should look as follows:



/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, Mock, [✓ AWS Service]

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[redacted]7:function:Calc/invocations

Execution role arn:aws:iam::7[redacted]7:role/apigAwsProxyRole

Credentials cache Do not add caller credentials to cache key

Content Handling Passthrough

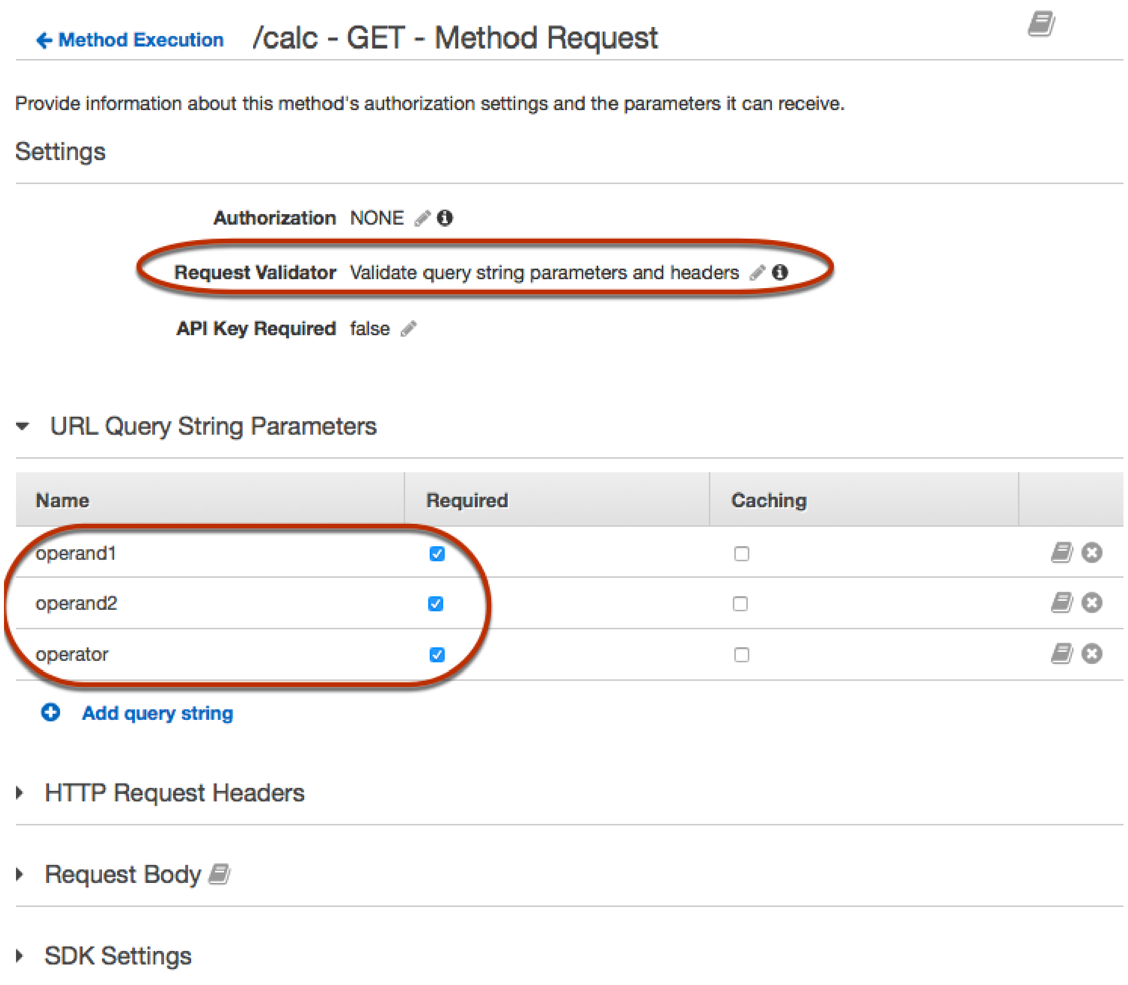
You can also add, in **Integration Request**, the `X-Amz-Invocation-Type: Event | RequestReponse | DryRun` header to have the action invoked asynchronously, as request and response, or as a test run, respectively. If the header is not specified, the action will be invoked as request and response.

4. Go to **Method Request** to set up query parameters for the **GET** method on **/calc** to receive input to the backend Lambda function.

   a. Choose the pencil icon next to **Request Validator** to select `Validate query string parameters and headers`. This setting will cause an error message to return to state the required parameters are missing if the client does not specify them. You will not get charged for the call to the backend.

   b. Expand the **URL Query String Parameters** section. Choose **Add query string** to add the **operand1**, **operand2**, and **operator** query string parameters. Check the **Required** option for each parameter to ensure that they are validated.

   The configuration now looks as follows:



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

Settings

Authorization NONE

Request Validator Validate query string parameters and headers

API Key Required false

URL Query String Parameters

Name	Required	Caching
operand1	✓	☐
operand2	✓	☐
operator	✓	☐

5. Go back to **Integration Request** to set up the mapping template to translate the client-supplied query strings to the integration request payload as required by the `Calc` function.

   a. Expand the **Body Mapping Templates** section.

b. Choose `When no template matches the request Content-Type header` for **Request body passthrough**.

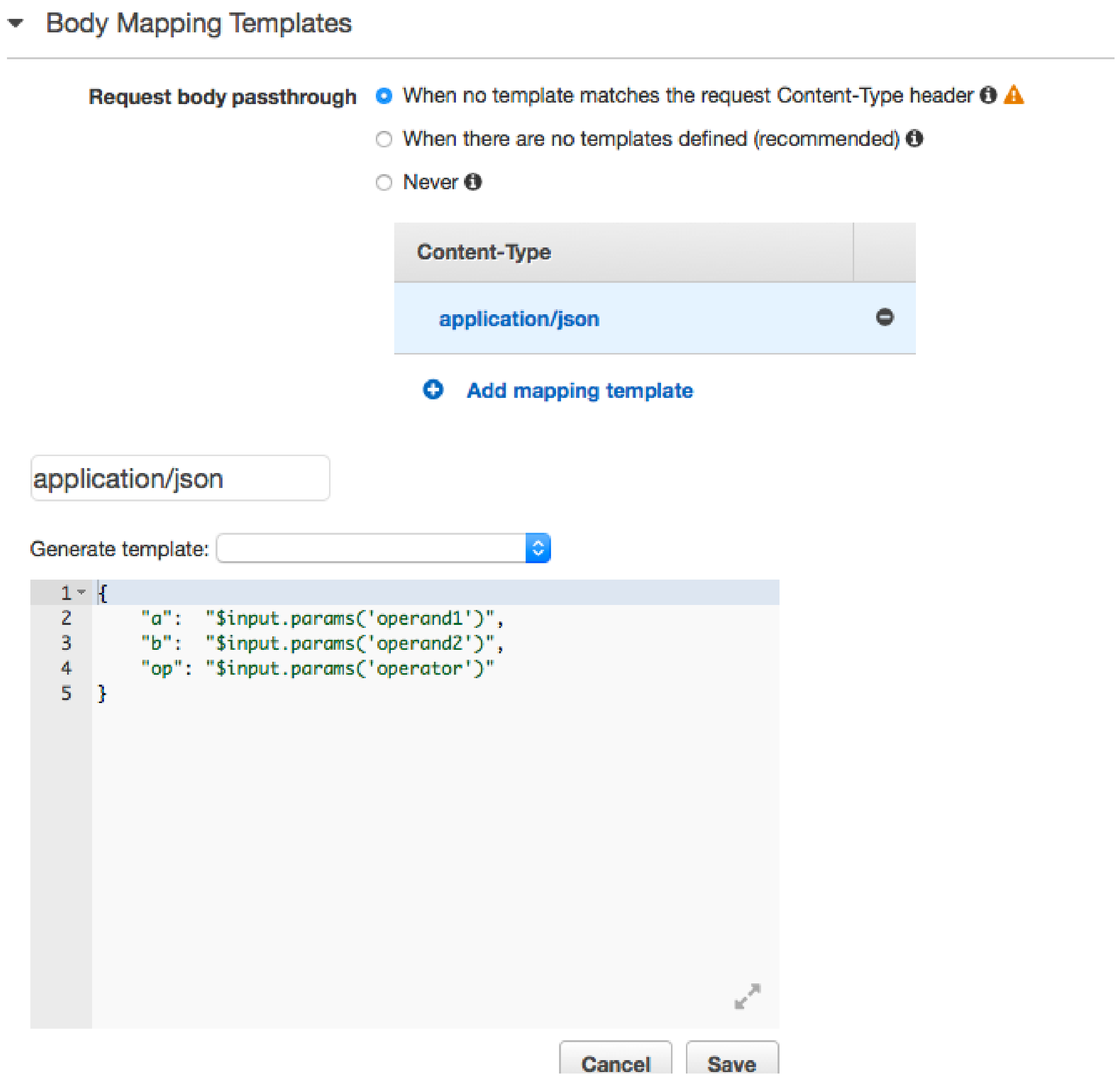
c. If `application/json` is not shown under **Content-Type**, choose **Add mapping template** to add it.

d. And then type and save the following mapping script in the mapping template editor:

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

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 backend Lambda function. The transformed JSON object will be included as the integration request payload.

The final settings of this step is shown as follows:


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
application/json

application/json

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

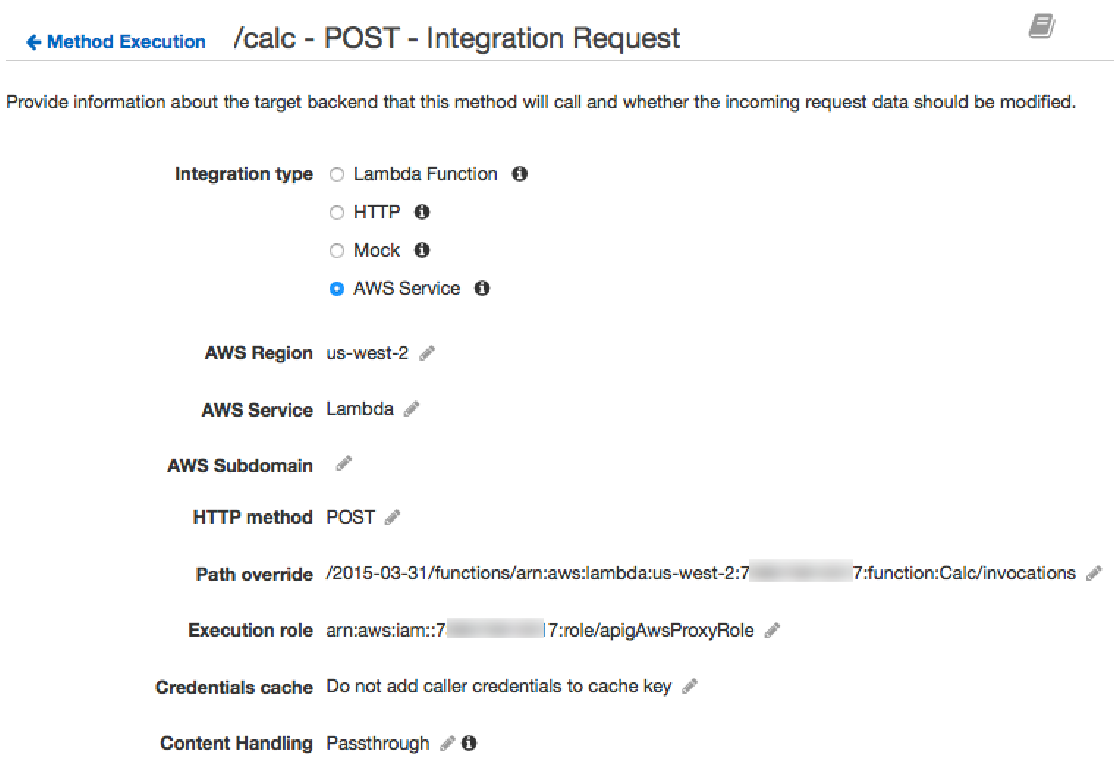
6. You can now choose **Test** to verify that the GET method on the **/calc** resource has been properly set up to invoke the Lambda function.

Create a POST Method with a JSON Payload to Call the Lambda Function

By creating a POST method with a JSON payload to call the LAM function, we expect the client to submit the necessary input to the backend function in the request body. To ensure that the client uploads the correct input data, we will enable request validation on the payload.

To set up the POST method with a JSON payload to invoke a Lambda function

1. Go to the API Gateway console and choose the API created previously.
2. Highlight the **/calc** resource from **Resources** pane.
3. Choose **Create Method** from the **Actions** menu to create the **POST /calc** method.
4. In the method's **Set Up** panel, configure this POST method with the following integration settings. For more information, follow the discussions in Create a GET Method with Query Parameters to Call the Lambda Function (p. 380).



/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, Mock, [✓ AWS Service]

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[redacted]7:function:Calc/invocations

Execution role: arn:aws:iam::7[redacted]7:role/apigAwsProxyRole

Credentials cache: Do not add caller credentials to cache key

Content Handling: Passthrough

5. Choose **Models** under the API from the API Gateway console's primary navigation pane to create data models for the method's input and output:

   a. Choose **Create** in the **Models** pane. Type `Input` in **Model name**, type `application/json` in **Content type**, and type the following schema definition in **Model schema**:

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

This model describes the input data structure and will be used to validate the incoming request body.

b. Choose **Create** in the **Models** pane. Type `Output` in **Model name**, type `application/json` in **Content type**, and type the following schema definition in **Model schema**:

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

This model describes the data structure of the calculated output from the backend. It can be used to map the integration response data to a different model. This tutorial relies on the passthrough behavior and does not use this model.

c. Choose **Create** in the **Models** pane. Type `Result` in **Model name**, type `application/json` in **Content type**, and type the following schema definition in **Model schema**:

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

This model describes the data structure of the returned response data. It references both the `Input` and `Output` schemas defined in the specified API (*`restapi-id`*). Again, this model is not used in this tutorial because it leverages the passthrough behavior.

6. In the **Method Request** configuration settings, do the following to enable request validation on the incoming request body:

   a. Choose the pencil icon next to **Request Validator** to choose `Validate body`.
   b. Expand the **Request Body** section, choose **Add model**
   c. Type `application/json` in the **Content-Type** input field and choose `Input` from the drop-down list in the **Model name** column.

7. 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 this method as an asynchronous call, you can add an `InvocationType` header in the method request and map it to the `X-Amz-Invocation-Type` header in the integration request with either a static value of `'Event'` or the header mapping expression of `method.request.header.InvocationType`. For the latter, the client must include the `InvocationType:Event` header in the method request. The asynchronous call will return an empty response, instead.

Create a GET Method with Path Parameters to Call the Lambda Function

In this section, we create a GET method on a resource specified by a sequence of path parameters to call the backend Lambda function. The path parameter values specify the input data to the Lambda function. We will define a mapping template to map the incoming path parameter values to the required integration request payload.

In addition, we will use the simple LAM integration feature provided by the ABP console to set up the method. As you can see, this console-provided feature provides much more streamlined user experiences.

To set up the GET method with URL path parameters to call the Lambda function

1. Go to the API Gateway console.
2. Highlight the **/calc/{operand1}/{operand2}/{operator}** resource on the **Resources** tree of the previously created API
3. Choose **Create Method** from the **Actions** drop-down menu, choose `GET`.
4. In the
   <guilable>Setup</guilable>
   pane, choose `Lambda Function` for **Integration type**, to use the streamlined setup process enabled by the console.
5. Choose a region (e.g., `us-west-2`) for **Lambda Region**. This is the region where the Lambda function is hosted.
6. Choose an existing Lambda function (e.g., `Calc`) for **Lambda Function**.
7. Choose **Save** and then choose **OK** to consent to **Add Permissions to Lambda Function**.
8. Choose **Integration Request** to set up body mapping template.

   a. Expand the **Body Mapping Templates** section.
   b. Choose **Add mapping template**.
   c. Type `application/json` for **Content-Type** and then choose the check mark icon to open the template editor.
   d. Choose **Yes, secure this integration** to proceed.

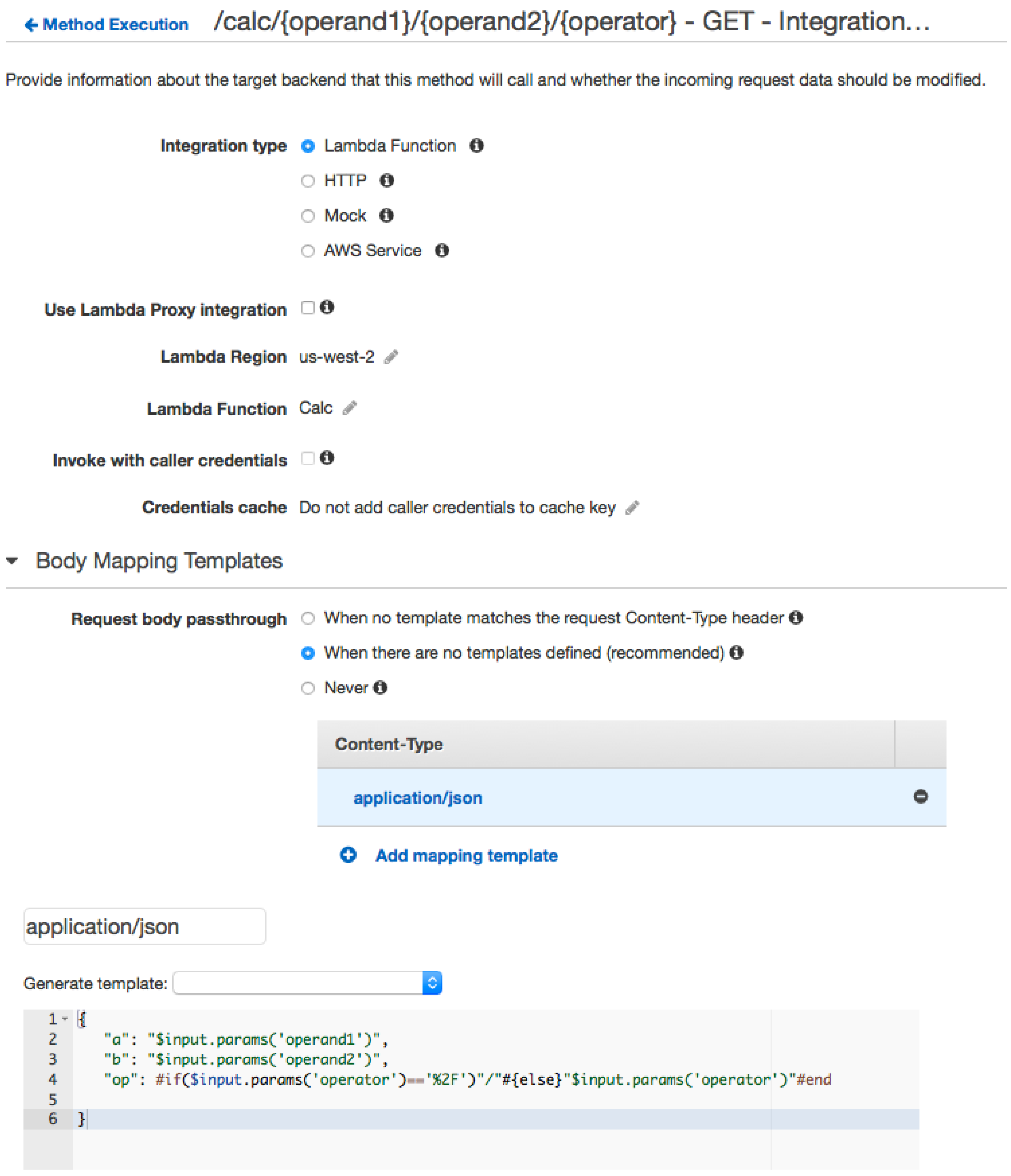
e. Type the following mapping script to the template editor:

```
{
   "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 translates the `%2F` into '`/` before passing it to the Lambda function.

f. Save the mapping template.

When the method is set up correctly, the settings should look similar to the following:



/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 Mock AWS Service

Use Lambda Proxy integration ☐

Lambda Region us-west-2

Lambda Function Calc

Invoke with caller credentials ☐

Credentials cache Do not add caller credentials to cache key

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
application/json

application/json

Generate template:

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

}
```

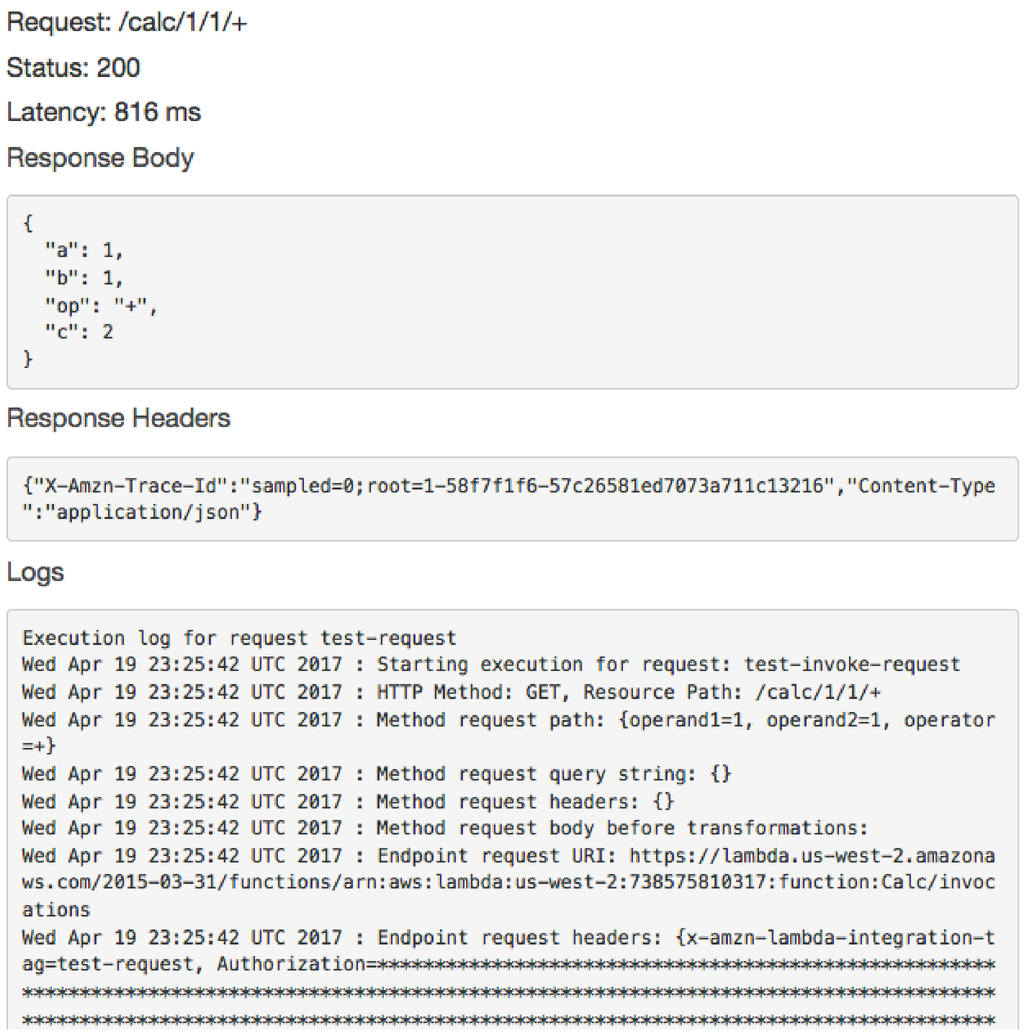
9. Now, let us test the API using the TestInvoke feature of the console.

   a. Choose **Test** from **Method Execution**.

   b. Type `1`, `2` and `+` in **{operand1}**, **{operand2}** and **{operator}** fields, respectively.

   c. Choose **Test**.

   d. The result will be shown similar to the following:



Request: /calc/1/1/+

Status: 200

Latency: 816 ms

Response Body

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

Response Headers

```
{"X-Amzn-Trace-Id":"sampled=0;root=1-58f7f1f6-57c26581ed7073a711c13216","Content-Type
":"application/json"}
```

Logs

```
Execution log for request test-request
Wed Apr 19 23:25:42 UTC 2017 : Starting execution for request: test-invoke-request
Wed Apr 19 23:25:42 UTC 2017 : HTTP Method: GET, Resource Path: /calc/1/1/+
Wed Apr 19 23:25:42 UTC 2017 : Method request path: {operand1=1, operand2=1, operator
=+}
Wed Apr 19 23:25:42 UTC 2017 : Method request query string: {}
Wed Apr 19 23:25:42 UTC 2017 : Method request headers: {}
Wed Apr 19 23:25:42 UTC 2017 : Method request body before transformations:
Wed Apr 19 23:25:42 UTC 2017 : Endpoint request URI: https://lambda.us-west-2.amazona
ws.com/2015-03-31/functions/arn:aws:lambda:us-west-2:738575810317:function:Calc/invoc
ations
Wed Apr 19 23:25:42 UTC 2017 : Endpoint request headers: {x-amzn-lambda-integration-t
ag=test-request, Authorization=*************************************************************
*************************************************************************************
*************************************************************************************
```

This test result shows the original output from the backend Lambda function, as passed through the integration response without mapping, because we have not configured any mapping template. Next, we model the data structure of the method response payload after the `Result` schema.

10. By default, the method response body is assigned an Empty model. This will cause the integration response body passed through without mapping. However, when you generate an SDK for one of the strongly-type languages, such as Java or Objective-C, your SDK users will receive an empty object as the result. To ensure both the REST client and SDK clients receive the desired result, you must model the response data using a predefined schema. Here, we demonstrate how to define a model for the method response body and to construct a mapping template to transform the integration response body to the method response body.

    a. In **/calc/{operand1}/{operand2}/{operator} - GET - Method Execution**, choose **Method Response**.

    b. Expand the **200** response,

    c. Under **Response Body for 200** section. If no model has been assigned for the method response,

    d. Choose the pencil icon next to the model for the `application/json` content type.

    e. Choose a predefined model from the **Models** drop-down list. For this tutorial, this is `Result`.

f. Save the model choice.

Note
If no model is defined for the content type of `application/json`, choose **Add Response Model** and follow the on-screen instructions to add the model.

Setting the model for the method response body ensure that the response data will be cast into the `Result` object of a given SDK. For this, we also need to make sure that the integration response data is mapped accordingly, which we discuss next.

11. To return the backend result according to the specified schema,

    a. Choose **Integration Response** and expand the 200 method response entry.

    b. Expand the **Body Mapping Templates** section.

    c. Choose or add `application/json` to the **Content-Type** list.

    d. Choose `Result` from the **Generate template** drop-down list to bring up the `Result` template blueprint.

    e. Change the template blueprint as follows:

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

    f. Choose **Save**.

    g. To test the mapping template, choose **Test** in **Method Execution** and type `1` `2` and `+` in the **operand1**, **operand2** and **operator** input fields, respectively. The integration response from the Lambda function is now mapped to a `Result` object:

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

12. To make the API accessible beyond Test Invoke in the API Gateway console, choose **Deploy API** from the **Actions** drop-down menu. Make sure to repeat deploying the API whenever you finish adding, modifying or deleting a resource or method, updating any data mapping, updating the stage settings. Otherwise, new features or updates will not be available.

Swagger Definitions of Sample API Integrated with a Lambda Function

```
{
  "swagger": "2.0",
  "info": {
    "version": "2017-04-20T04:08:08Z",
    "title": "LambdaGate"
  },
  "host": "uojnr9hd57.execute-api.us-east-1.amazonaws.com",
  "basePath": "/test",
  "schemes": [
    "https"
  ],
  "paths": {
    "/calc": {
      "get": {
        "consumes": [
          "application/json"
        ],
        "produces": [
          "application/json"
        ],
        "parameters": [
          {
            "name": "operand2",
            "in": "query",
            "required": true,
            "type": "string"
          },
          {
            "name": "operator",
            "in": "query",
            "required": true,
            "type": "string"
          },
          {
            "name": "operand1",
            "in": "query",
            "required": true,
            "type": "string"
          }
        ],
        "responses": {
          "200": {
            "description": "200 response",
            "schema": {
              "$ref": "#/definitions/Result"
            },
            "headers": {
              "operand_1": {
                "type": "string"
              },
              "operand_2": {
                "type": "string"
              },
              "operator": {
                "type": "string"
              }
            }
          }
        },
        "x-amazon-apigateway-request-validator": "Validate query string parameters and
 headers",
        "x-amazon-apigateway-integration": {
          "credentials": "arn:aws:iam::123456789012:role/apigAwsProxyRole",
          "responses": {
            "default": {
              "statusCode": "200",
              "responseParameters": {
                "method.response.header.operator": "integration.response.body.op",
                "method.response.header.operand_2": "integration.response.body.b",
                "method.response.header.operand_1": "integration.response.body.a"
              },
              "responseTemplates": {
                "application/json": "#set($res = $input.path('$'))\n{\n    \"result\":
 \"$res.a, $res.b, $res.op => $res.c\",\n  \"a\" : \"$res.a\",\n  \"b\" : \"$res.b\",\n
 \"op\" : \"$res.op\",\n  \"c\" : \"$res.c\"\n}"
              }
            }
          },
          "uri": "arn:aws:apigateway:us-west-2:lambda:path//2015-03-31/functions/
arn:aws:lambda:us-west-2:123456789012:function:Calc/invocations",
          "passthroughBehavior": "when_no_match",
          "httpMethod": "POST",
          "requestTemplates": {
            "application/json": "{\n    \"a\":  \"$input.params('operand1')\",\n    \"b\":
 \"$input.params('operand2')\", \n    \"op\": \"$input.params('operator')\"   \n}"
          },
          "type": "aws"
        }
      },
      "post": {
        "consumes": [
          "application/json"
        ],
        "produces": [
          "application/json"
        ],
        "parameters": [
          {
            "name": "InvocationType",
            "in": "header",
            "required": false,
            "type": "string"
          },
          {
            "in": "body",
            "name": "Input",
            "required": true,
            "schema": {
              "$ref": "#/definitions/Input"
            }
          }
        ],
        "responses": {
          "200": {
            "description": "200 response",
            "schema": {
              "$ref": "#/definitions/Result"
            }
          }
        },
        "x-amazon-apigateway-request-validator": "Validate body",
        "x-amazon-apigateway-integration": {
          "credentials": "arn:aws:iam::123456789012:role/apigAwsProxyRole",
          "responses": {
            "default": {
              "statusCode": "200",
              "responseTemplates": {
                "application/json": "#set($inputRoot = $input.path('$'))\n{\n  \"a\" :
 $inputRoot.a,\n  \"b\" : $inputRoot.b,\n  \"op\" : $inputRoot.op,\n  \"c\" : $inputRoot.c
\n}"
              }
            }
          },
          "uri": "arn:aws:apigateway:us-west-2:lambda:path//2015-03-31/functions/
arn:aws:lambda:us-west-2:123456789012:function:Calc/invocations",
          "passthroughBehavior": "when_no_templates",
          "httpMethod": "POST",
          "type": "aws"
        }
      }
    },
    "/calc/{operand1}/{operand2}/{operator}": {
      "get": {
        "consumes": [
          "application/json"
        ],
        "produces": [
          "application/json"
        ],
        "parameters": [
          {
            "name": "operand2",
            "in": "path",
            "required": true,
            "type": "string"
          },
          {
            "name": "operator",
            "in": "path",
            "required": true,
            "type": "string"
          },
          {
            "name": "operand1",
            "in": "path",
            "required": true,
            "type": "string"
          }
        ],
        "responses": {
          "200": {
            "description": "200 response",
            "schema": {
              "$ref": "#/definitions/Result"
            }
          }
        },
        "x-amazon-apigateway-integration": {
          "responses": {
            "default": {
              "statusCode": "200",
              "responseTemplates": {
                "application/json": "#set($inputRoot = $input.path('$'))\n{\n  \"input\" :
 {\n    \"a\" : $inputRoot.a,\n    \"b\" : $inputRoot.b,\n    \"op\" : \"$inputRoot.op\"\n
  },\n  \"output\" : {\n    \"c\" : $inputRoot.c\n  }\n}"
              }
            }
          },
          "uri": "arn:aws:apigateway:us-west-2:lambda:path/2015-03-31/functions/
arn:aws:lambda:us-west-2:123456789012:function:Calc/invocations",
          "passthroughBehavior": "when_no_templates",
          "httpMethod": "POST",
          "requestTemplates": {
            "application/json": "{\n    \"a\": \"$input.params('operand1')\",\n    \"b\":
 \"$input.params('operand2')\",\n    \"op\": #if($input.params('operator')=='%2F')\"/
\"#{else}\"$input.params('operator')\"#end\n    \n}"
          },
          "contentHandling": "CONVERT_TO_TEXT",
          "type": "aws"
        }
      }
    }
  },
  "definitions": {
    "Input": {
      "type": "object",
      "required": [
        "a",
        "b",
        "op"
      ],
      "properties": {
        "a": {
          "type": "number"
        },
        "b": {
          "type": "number"
        },
        "op": {
          "type": "string",
          "description": "binary op of ['+', 'add', '-', 'sub', '*', 'mul', '%2F', 'div']"
        }
      },
      "title": "Input"
    },
    "Output": {
      "type": "object",
      "properties": {
        "c": {
          "type": "number"
        }
      },
      "title": "Output"
    },
    "Result": {
      "type": "object",
      "properties": {
        "input": {
          "$ref": "#/definitions/Input"
        },
        "output": {
          "$ref": "#/definitions/Output"
        }
      },
      "title": "Result"
    }
  },
  "x-amazon-apigateway-request-validators": {
    "Validate body": {
      "validateRequestParameters": false,
      "validateRequestBody": true
    },
    "Validate query string parameters and headers": {
      "validateRequestParameters": true,
      "validateRequestBody": false
    }
  }
```

```
}
```

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. 409). For instructions on how to import an API using the Swagger definition, see Import an API (p. 210).

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 https://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

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"
    }
  ]
}
```

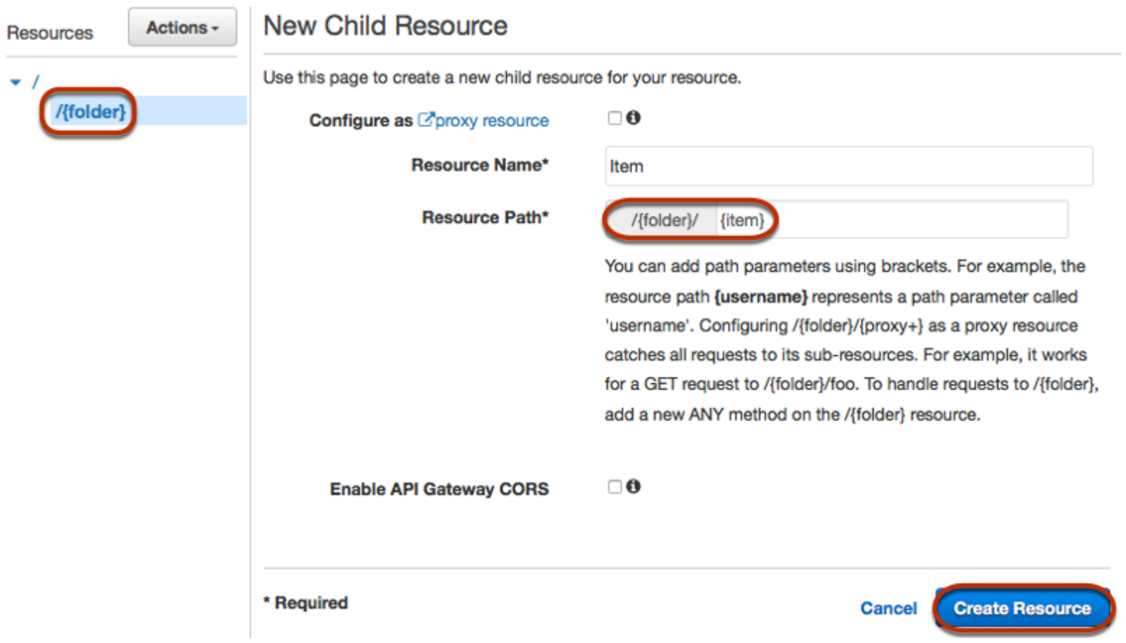
When using the IAM console to create the role, choose the **Amazon API Gateway** role type to ensure that this trust policy is automatically included.

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}**.
3. For the API's **Folder** resource, create an **Item** child resource. Set the required **Resource Path** as **/{item}**.



New Child Resource

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

Configure as proxy resource

Resource Name* Item

Resource Path* /{folder}/ {item}

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

Enable API Gateway CORS

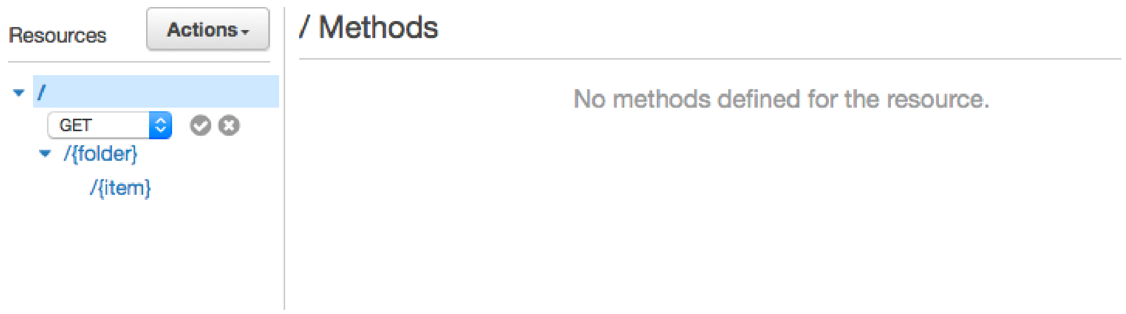
* Required

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

1. Choose **Create method** on the root node (`/`) from the **Actions** drop-down menu at the top-right corner of the **Resources** panel.
2. Choose the `GET` from the drop-down list of HTTP verbs, and choose the check-mark icon to start creating the method.



No methods defined for the resource.

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**.
10. Choose **Save** to finish setting up this method.

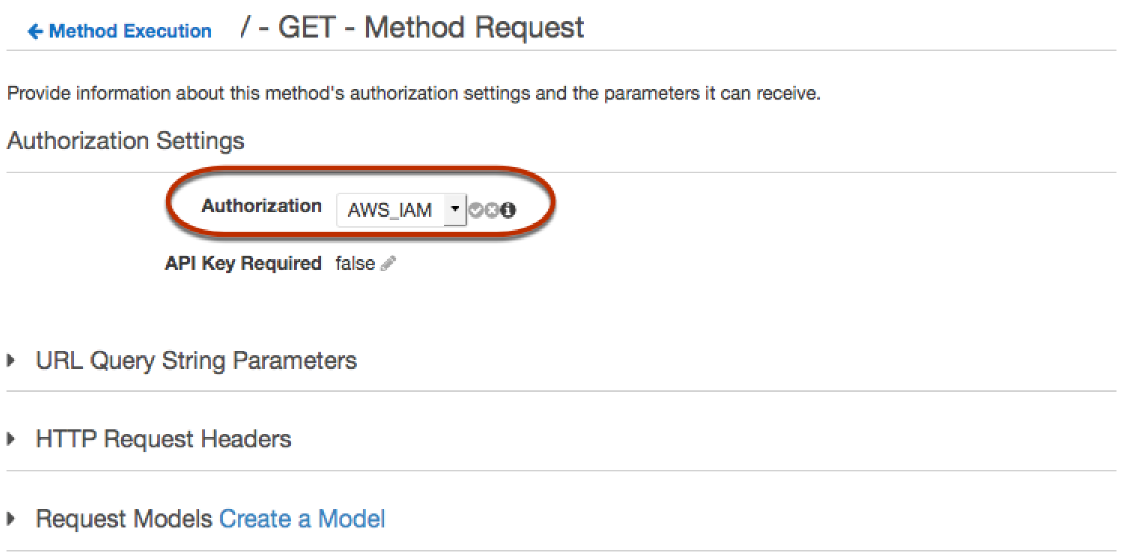
This setup integrates the frontend `GET https://your-api-host/stage/` request with the backend `GET https://your-s3-host/`.

> **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.



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

Authorization Settings

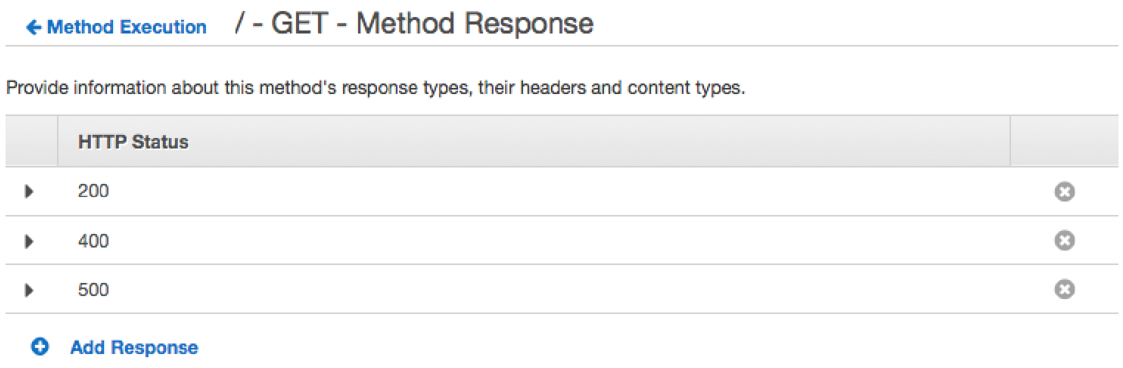
Authorization AWS_IAM

API Key Required false

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 backend 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

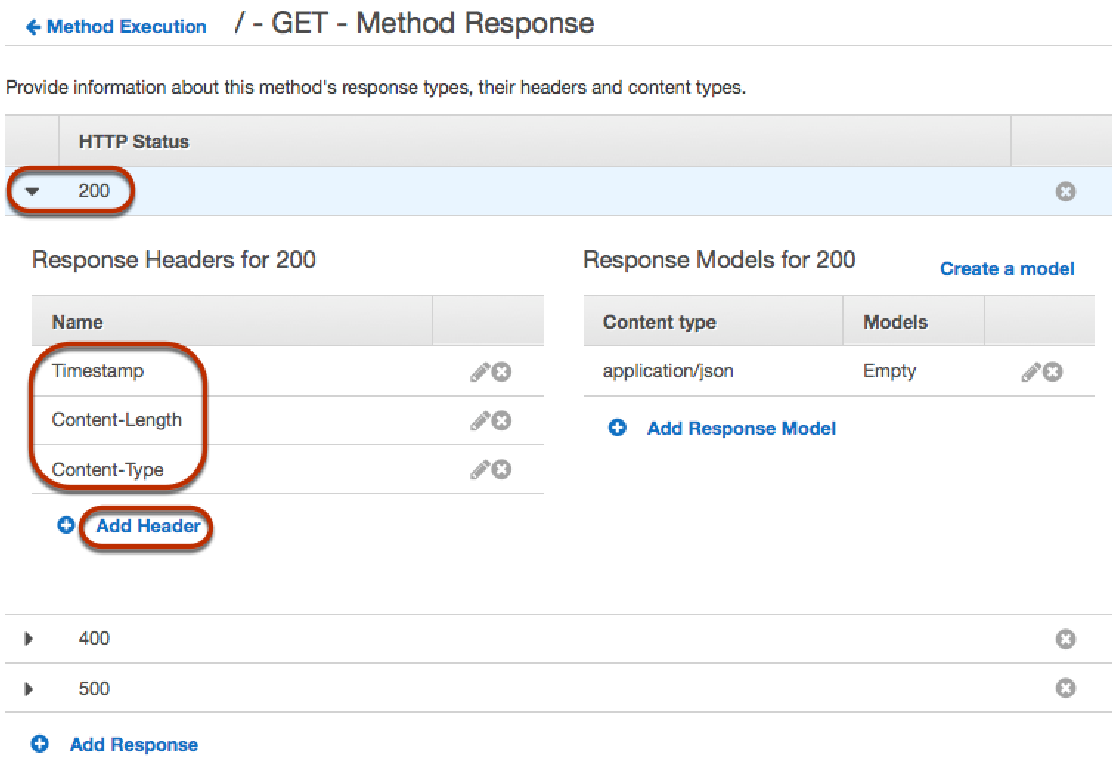
1. From the **Method Execution** pane, choose the **Method Response** box. The API Gateway declares the 200 response by default.
2. Choose **Add response**, enter **`400`** in the input text box, and choose the check-mark to finish the declaration.
3. Repeat the above step to declare the 500 response 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

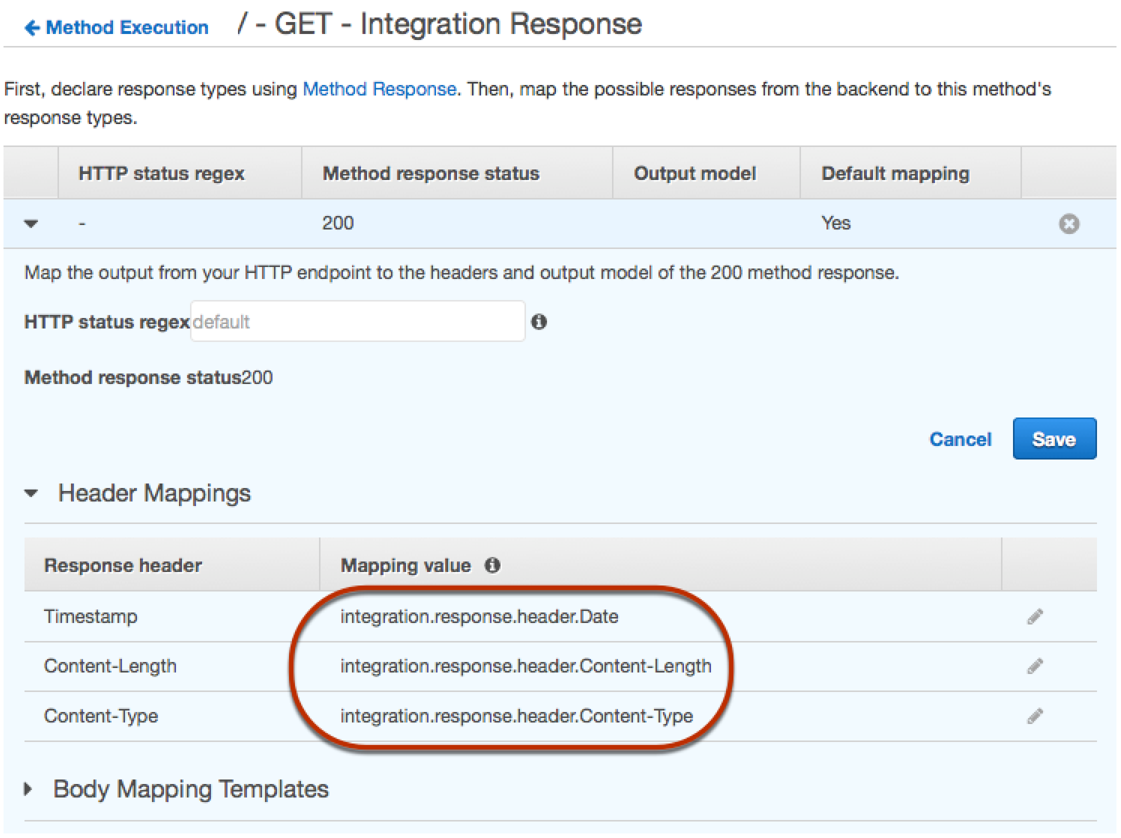
Because the successful integration response from Amazon S3 returns the bucket list as an XML payload and the default method response from API Gateway returns a JSON payload, we must map the backend Content-Type header parameter value to the frontend counterpart. Otherwise, the client will receive `application/json` 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.



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: default

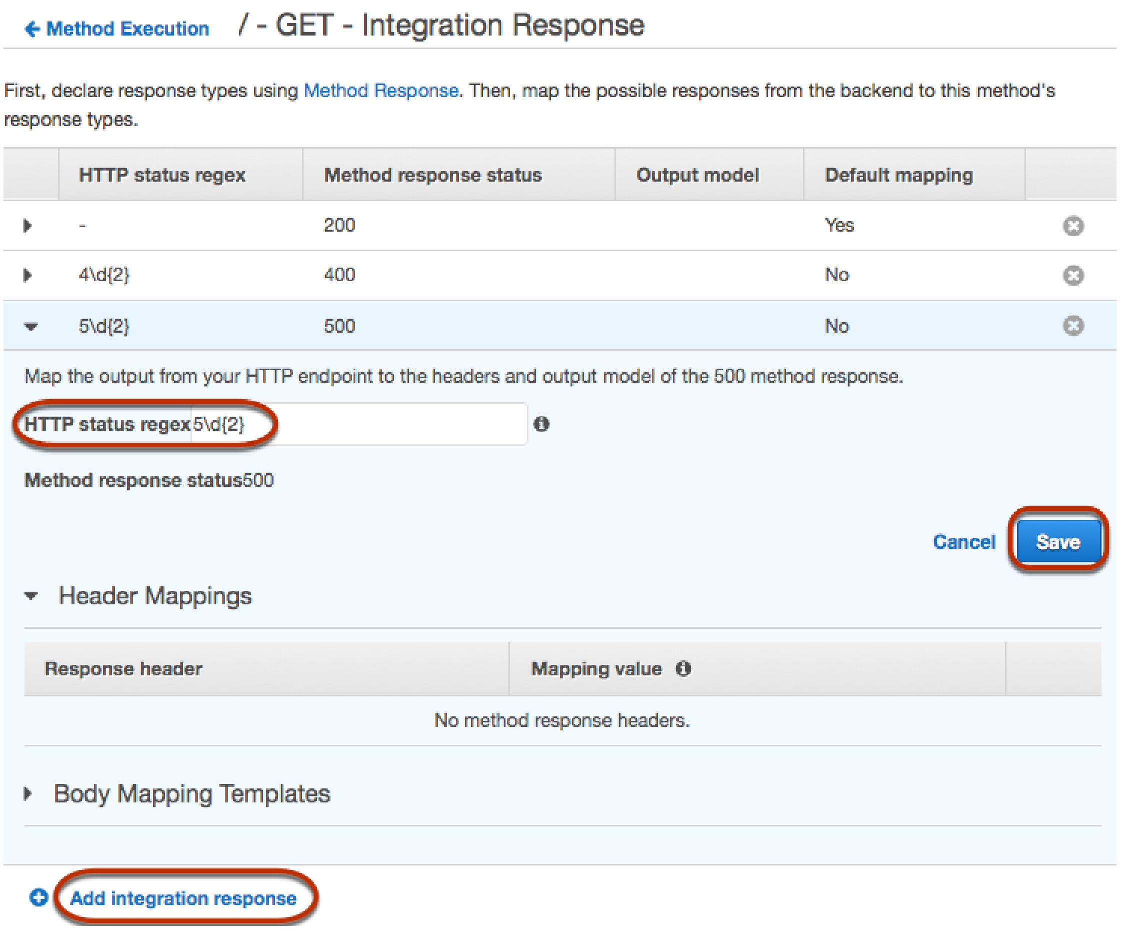
Method response status: 200

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

With the above header mappings, API Gateway will translate the `Date` header from the backend 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.



/ - 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.

HTTP status regex 5\d{2}

Method response status 500

Header Mappings

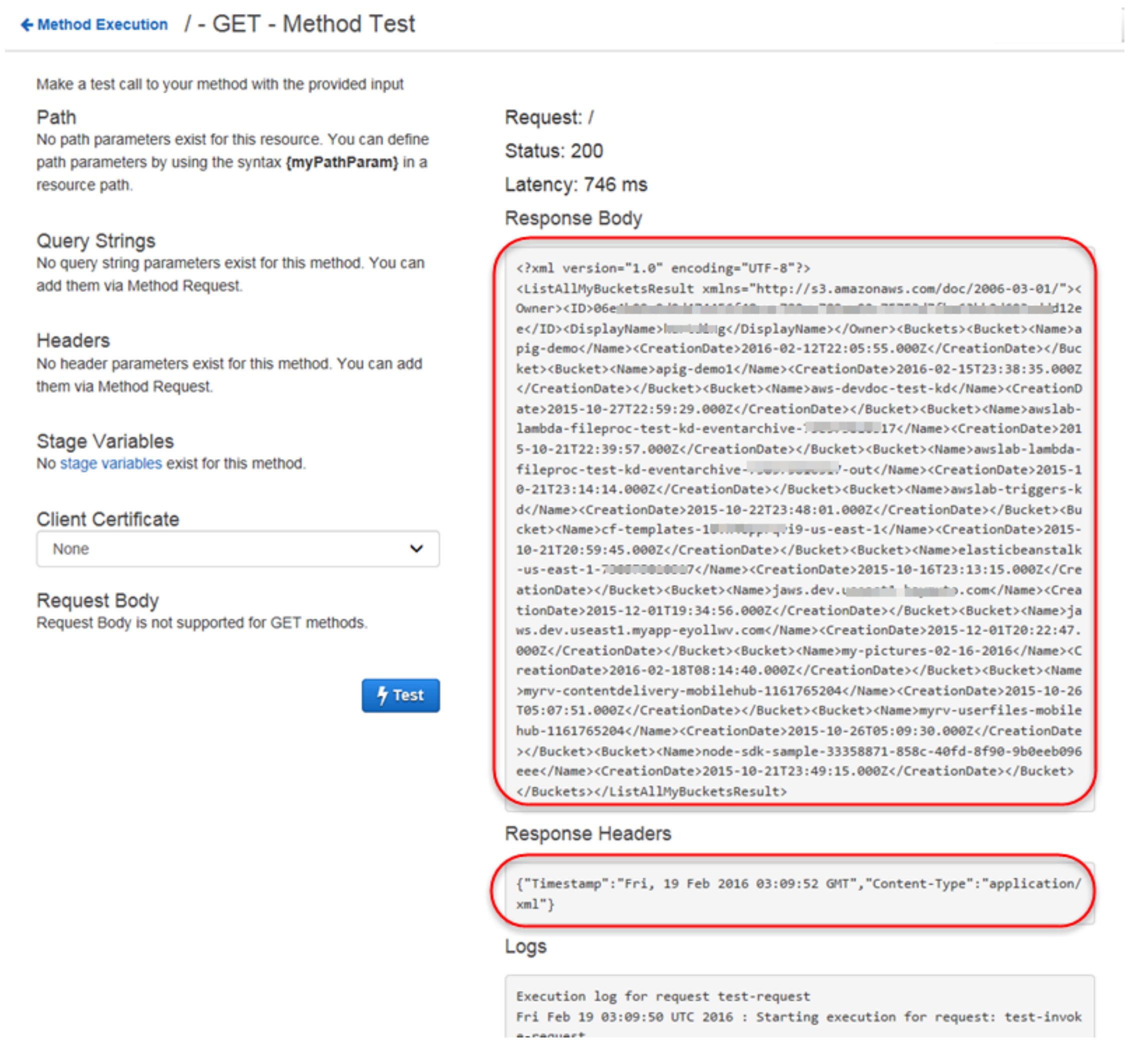
Response header	Mapping value
No method response headers.	

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.



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
None

Request Body
Request Body is not supported for GET methods.

Request: /
Status: 200
Latency: 746 ms

Response Body

```
<?xml version="1.0" encoding="UTF-8"?>
<ListAllMyBucketsResult xmlns="http://s3.amazonaws.com/doc/2006-03-01/"><
Owner><ID>06e[redacted]d12e
e</ID><DisplayName>[redacted]g</DisplayName></Owner><Buckets><Bucket><Name>a
pig-demo</Name><CreationDate>2016-02-12T22:05:55.000Z</CreationDate></Buc
ket><Bucket><Name>apig-demo1</Name><CreationDate>2016-02-15T23:38:35.000Z
</CreationDate></Bucket><Bucket><Name>aws-devdoc-test-kd</Name><CreationD
ate>2015-10-27T22:59:29.000Z</CreationDate></Bucket><Bucket><Name>awslab-
lambda-fileproc-test-kd-eventarchive-[redacted]17</Name><CreationDate>201
5-10-21T22:39:57.000Z</CreationDate></Bucket><Bucket><Name>awslab-lambda-
fileproc-test-kd-eventarchive-[redacted]-out</Name><CreationDate>2015-1
0-21T23:14:14.000Z</CreationDate></Bucket><Bucket><Name>awslab-triggers-k
d</Name><CreationDate>2015-10-22T23:48:01.000Z</CreationDate></Bucket><Bu
cket><Name>cf-templates-1[redacted]i9-us-east-1</Name><CreationDate>2015-
10-21T20:59:45.000Z</CreationDate></Bucket><Bucket><Name>elasticbeanstalk
-us-east-1-7[redacted]7</Name><CreationDate>2015-10-16T23:13:15.000Z</Cre
ationDate></Bucket><Bucket><Name>jaws.dev.u[redacted].com</Name><Crea
tionDate>2015-12-01T19:34:56.000Z</CreationDate></Bucket><Bucket><Name>ja
ws.dev.useast1.myapp-eyollwv.com</Name><CreationDate>2015-12-01T20:22:47.
000Z</CreationDate></Bucket><Bucket><Name>my-pictures-02-16-2016</Name><C
reationDate>2016-02-18T08:14:40.000Z</CreationDate></Bucket><Bucket><Name
>myrv-contentdelivery-mobilehub-1161765204</Name><CreationDate>2015-10-26
T05:07:51.000Z</CreationDate></Bucket><Bucket><Name>myrv-userfiles-mobile
hub-1161765204</Name><CreationDate>2015-10-26T05:09:30.000Z</CreationDate
></Bucket><Bucket><Name>node-sdk-sample-33358871-858c-40fd-8f90-9b0eeb096
eee</Name><CreationDate>2015-10-21T23:49:15.000Z</CreationDate></Bucket>
</Buckets></ListAllMyBucketsResult>
```

Response Headers

```
{"Timestamp":"Fri, 19 Feb 2016 03:09:52 GMT","Content-Type":"application/
xml"}
```

Logs

```
Execution log for request test-request
Fri Feb 19 03:09:50 UTC 2016 : Starting execution for request: test-invok
```

Note
To use the API Gateway console to test the API as an Amazon S3 proxy, make sure that the targeted S3 bucket is from a different region from the API's region. Otherwise, you may get a 500 Internal Server Error response. This limitation does not apply to any deployed API.

Expose API Methods to Access an Amazon S3 Bucket

To work with an Amazon S3 bucket, we expose the GET, PUT, and DELETE methods on the /{folder} resource to list objects in a bucket, create a new bucket, and delete an existing bucket. The instructions are similar to those prescribed in Expose an API Method to List the Caller's Amazon S3 Buckets (p. 398). In the following discussions, we outline the general tasks and highlight relevant differences.

To expose GET, PUT and DELETE methods on a folder resource

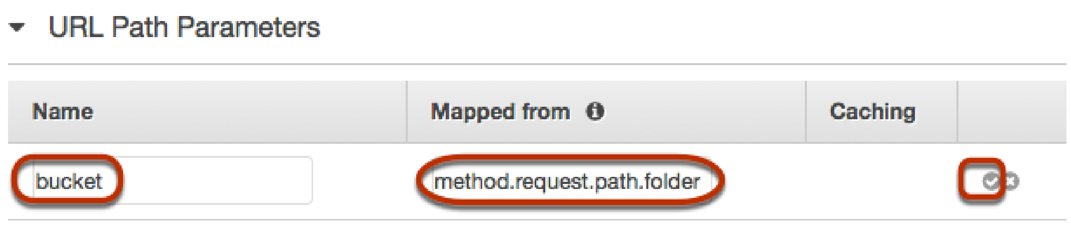
1. On the **/{folder}** node from the **Resources** tree, create the DELETE, GET and PUT methods, one at a time.
2. Set up the initial integration of each created method with its corresponding Amazon S3 endpoints. The following screen shot illustrates this setting for the `PUT /{folder}` method. For the `DELETE /`

`{folder}` and `GET /{folder}` method, replace the `PUT` value of **HTTP method** by `DELETE` and `GET`, respectively.

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}`:

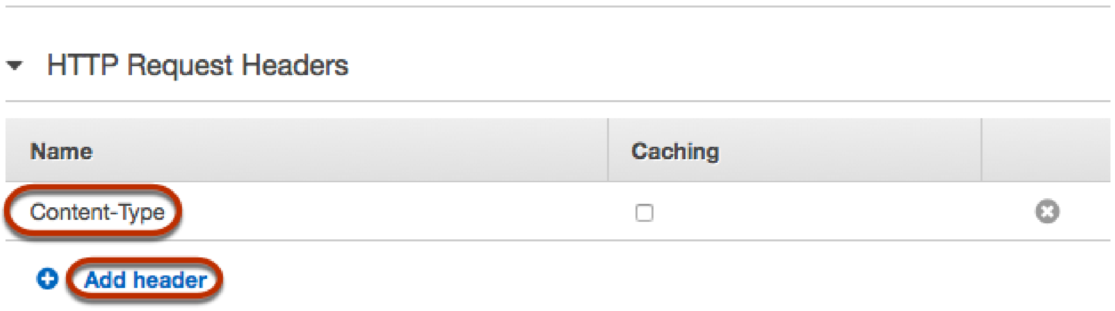
   a. Choose **Method Execution** and then **Integration Request**.
   b. Expand **URL Path Parameters** and choose **Add path**
   c. Type `bucket` in the **Name** column and `method.request.path.folder` in the **Mapped from** column. Choose the check-mark icon to save the mapping.



URL Path Parameters

Name	Mapped from	Caching
bucket	method.request.path.folder	

4. In **Method Request**, add the `Content-Type` to the **HTTP Request Headers** section.

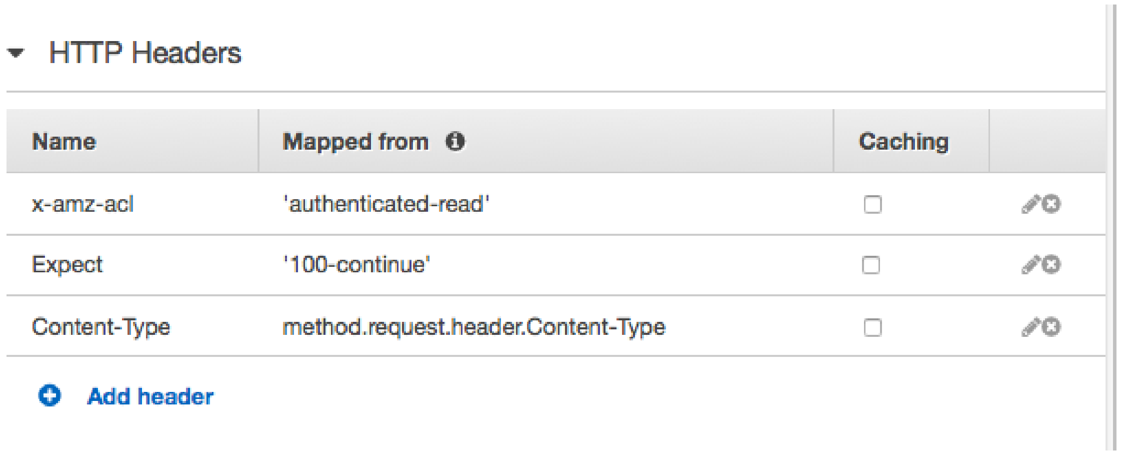


HTTP Request Headers

Name	Caching
Content-Type	☐

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. 398).



HTTP Headers

Name	Mapped from	Caching
x-amz-acl	'authenticated-read'	☐
Expect	'100-continue'	☐
Content-Type	method.request.header.Content-Type	☐

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.

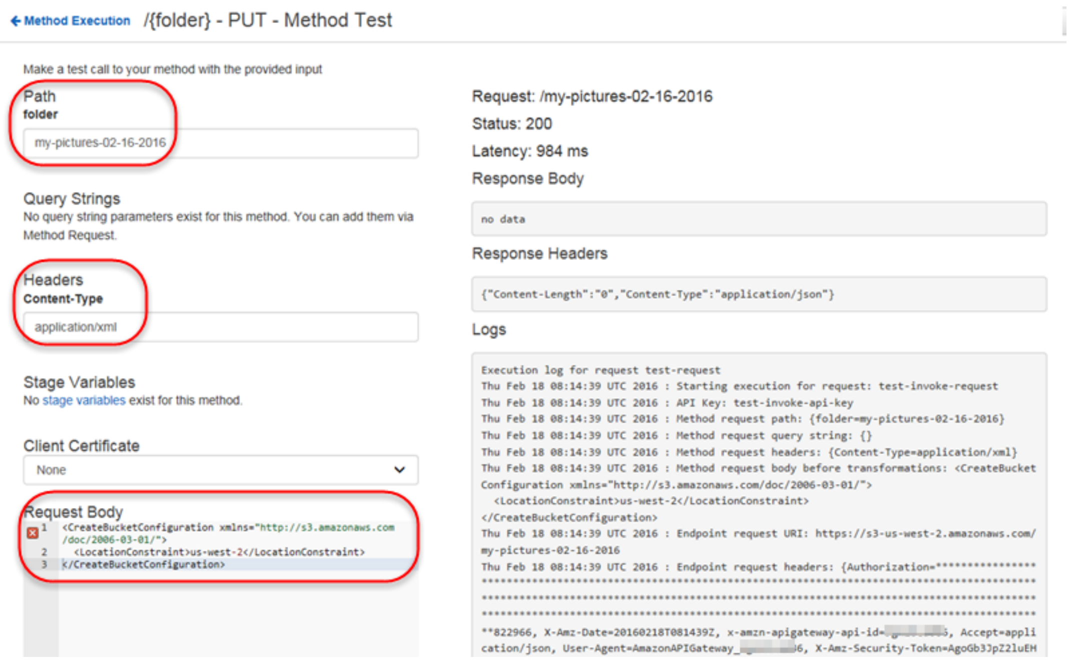
6. To test the `PUT` method, choose **Test** in the **Client** box from **Method Execution**, and enter the following as input to the testing:

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



Make a test call to your method with the provided input

Path
folder: my-pictures-02-16-2016

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

Headers
Content-Type: application/xml

Stage Variables
No stage variables exist for this method.

Client Certificate: None

Request Body

```
<CreateBucketConfiguration xmlns="http://s3.amazonaws.com
/doc/2006-03-01/">
  <LocationConstraint>us-west-2</LocationConstraint>
</CreateBucketConfiguration>
```

Request: /my-pictures-02-16-2016
Status: 200
Latency: 984 ms

Response Body

```
no data
```

Response Headers

```
{"Content-Length":"0","Content-Type":"application/json"}
```

Logs

```
Execution log for request test-request
Thu Feb 18 08:14:39 UTC 2016 : Starting execution for request: test-invoke-request
Thu Feb 18 08:14:39 UTC 2016 : API Key: test-invoke-api-key
Thu Feb 18 08:14:39 UTC 2016 : Method request path: {folder=my-pictures-02-16-2016}
Thu Feb 18 08:14:39 UTC 2016 : Method request query string: {}
Thu Feb 18 08:14:39 UTC 2016 : Method request headers: {Content-Type=application/xml}
Thu Feb 18 08:14:39 UTC 2016 : Method request body before transformations: <CreateBucket
Configuration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <LocationConstraint>us-west-2</LocationConstraint>
</CreateBucketConfiguration>
Thu Feb 18 08:14:39 UTC 2016 : Endpoint request URI: https://s3-us-west-2.amazonaws.com/
my-pictures-02-16-2016
Thu Feb 18 08:14:39 UTC 2016 : Endpoint request headers: {Authorization=***************
********************************************************************************
********************************************************************************
********************************************************************************
**822966, X-Amz-Date=20160218T081439Z, x-amzn-apigateway-api-id=[redacted], Accept=appli
cation/json, User-Agent=AmazonAPIGateway_[redacted], X-Amz-Security-Token=AgoGb3JpZ2luEH
```

7. Repeat the preceding steps to create and configure the GET and DELETE method on the API's **/{folder}** resource.

The above examples illustrate how to create a new bucket in the specified region, to view the list of objects in the bucket, and to delete the bucket. Other Amazon S3 bucket operations allow you work with the metadata or properties of the bucket. For example, you can set up your API to call the Amazon S3's PUT /?notification action to set up notifications on the bucket, to call PUT /?acl to set an access control list on the bucket, etc. The API set up is similar, except for that you must append appropriate query parameters to the Amazon S3 endpoint URLs. At run time, you must provide the appropriate XML payload to the method request. The same can be said about supporting the other GET and DELETE operations on a Amazon S3 bucket. For more information on possible &S3; actions on a bucket, see Amazon S3 Operations on Buckets.

Expose API Methods to Access an Amazon S3 Object in a Bucket

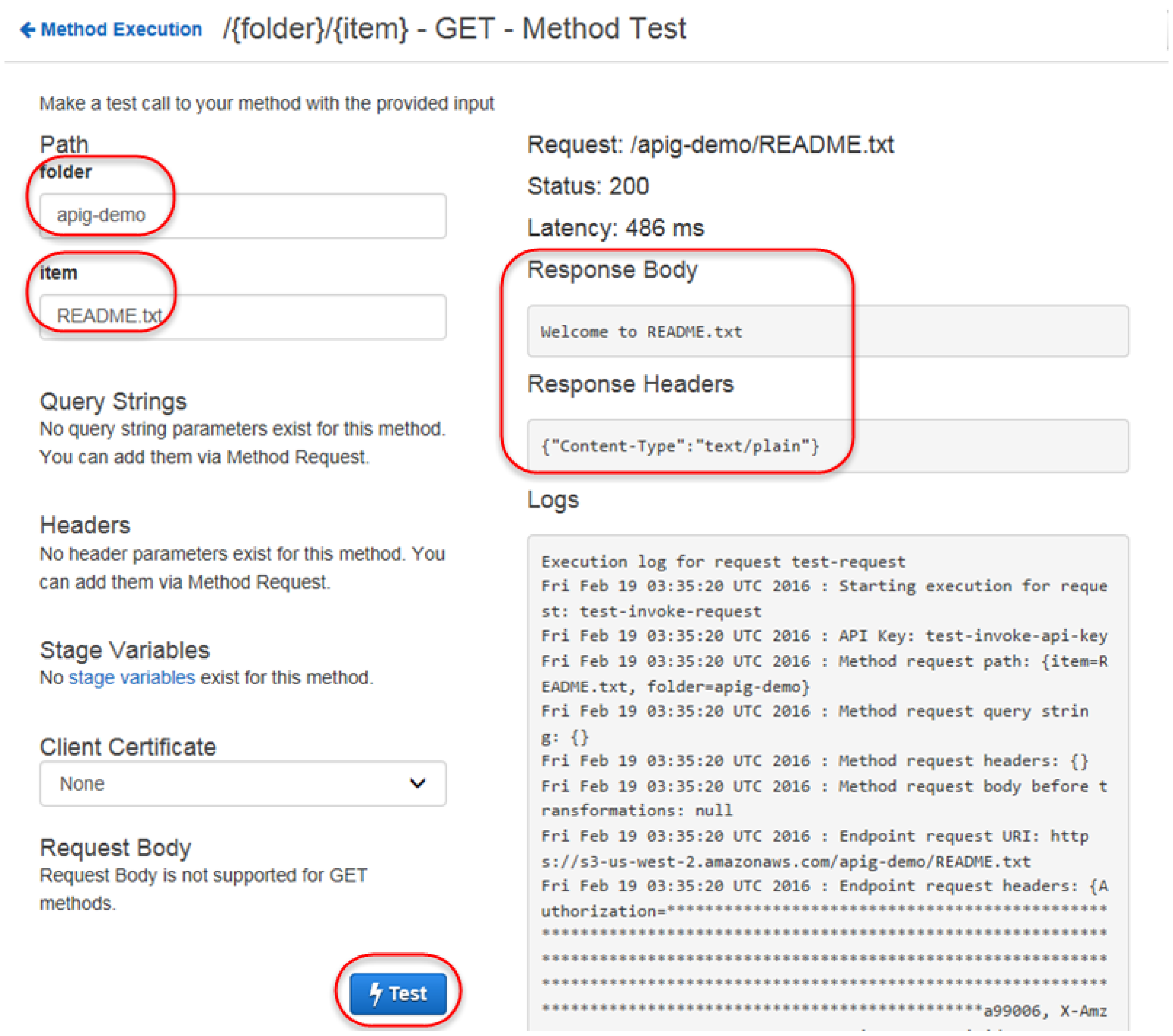
Amazon S3 supports GET, DELETE, HEAD, OPTIONS, POST and PUT actions to access and manage objects in a given bucket. For the complete list of supported actions, see Amazon S3 Operations on Objects.

In this tutorial, we expose the PUT Object operation, the GET Object operation, HEAD Object operation, and the DELETE Object operation through the API methods of `PUT /{folder}/{item}`, `GET /{folder}/{item}`, `HEAD /{folder}/{item}` and `DELETE /{folder}/{item}`, respectively.

The API setups for the PUT, GET and DELETE methods on `/{folder}/{item}` are the similar to those on `/{folder}`, as prescribed in Expose API Methods to Access an Amazon S3 Bucket (p. 403). One major difference is that the additional path parameter of `{object}` is appended to the method request URL and this path parameter is mapped to the Amazon S3 endpoint URL path parameter of `{item}` in the backend.

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).



/{folder}/{item} - GET - Method Test

Make a test call to your method with the provided input

Path

folder: apig-demo

item: README.txt

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: None

Request Body

Request Body is not supported for GET methods.

Request: /apig-demo/README.txt

Status: 200

Latency: 486 ms

Response Body

```
Welcome to README.txt
```

Response Headers

```
{"Content-Type":"text/plain"}
```

Logs

```
Execution log for request test-request
Fri Feb 19 03:35:20 UTC 2016 : Starting execution for reque
st: 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=R
EADME.txt, folder=apig-demo}
Fri Feb 19 03:35:20 UTC 2016 : Method request query strin
g: {}
Fri Feb 19 03:35:20 UTC 2016 : Method request headers: {}
Fri Feb 19 03:35:20 UTC 2016 : Method request body before t
ransformations: null
Fri Feb 19 03:35:20 UTC 2016 : Endpoint request URI: http
s://s3-us-west-2.amazonaws.com/apig-demo/README.txt
Fri Feb 19 03:35:20 UTC 2016 : Endpoint request headers: {A
uthorization=************************************************
************************************************************
************************************************************
************************************************************
**********************************************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.
4. To add a bucket named `apig-demo-5` to your Amazon S3 account in the *`{region}`* region:

Note
Be sure that the bucket name must be globally unique.

a. 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`

b. 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>{region}</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.{region}.amazonaws.com
Content-Type: application/xml
X-Amz-Date: 20161015T062647Z
Authorization: AWS4-HMAC-SHA256 Credential=access-key-id/20161015/{region}/execute-
api/aws4_request, SignedHeaders=content-length;content-type;host;x-amz-date,
 Signature=ccadb877bdb0d395ca38cc47e18a0d76bb5eaf17007d11e40bf6fb63d28c705b
Cache-Control: no-cache
Postman-Token: 6135d315-9cc4-8af8-1757-90871d00847e

Hello, World!
```

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

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.{region}.amazonaws.com
Content-Type: application/xml
X-Amz-Date: 20161015T064324Z
Authorization: AWS4-HMAC-SHA256 Credential=access-key-id/20161015/{region}/
execute-api/aws4_request, SignedHeaders=content-type;host;x-amz-date,
 Signature=4ac9bd4574a14e01568134fd16814534d9951649d3a22b3b0db9f1f5cd4dd0ac
Cache-Control: no-cache
```

```
Postman-Token: 9c43020a-966f-61e1-81af-4c49ad8d1392
```

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...603addd12ee</ID>
            <DisplayName>user-name</DisplayName>
        </Owner>
        <StorageClass>STANDARD</StorageClass>
    </Contents>
</ListBucketResult>
```

Note
To upload or download an image, you need to set content handling to CONVERT_TO_BINARY.

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.{region}.amazonaws.com",
  "basePath": "/S3",
  "schemes": [
    "https"
  ],
  "paths": {
    "/": {
      "get": {
        "produces": [
          "application/json"
        ],
        "responses": {
          "200": {
            "description": "200 response",
            "schema": {
              "$ref": "#/definitions/Empty"
            },
            "headers": {
              "Content-Length": {
                "type": "string"
              },
              "Timestamp": {
                "type": "string"
              },
              "Content-Type": {
                "type": "string"
              }
            }
          },
          "400": {
            "description": "400 response"
          },
          "500": {
            "description": "500 response"
          }
        },
        "security": [
          {
            "sigv4": []
          }
        ],
        "x-amazon-apigateway-integration": {
          "credentials": "arn:aws:iam::<replaceable>123456789012</replaceable>:role/
apigAwsProxyRole",
          "responses": {
            "4\\d{2}": {
              "statusCode": "400"
            },
            "default": {
              "statusCode": "200",
              "responseParameters": {
                "method.response.header.Content-Type":
 "integration.response.header.Content-Type",
                "method.response.header.Content-Length":
 "integration.response.header.Content-Length",
                "method.response.header.Timestamp": "integration.response.header.Date"
              }
            },
            "5\\d{2}": {
              "statusCode": "500"
            }
          },
          "uri": "arn:aws:apigateway:us-west-2:s3:path//",
          "passthroughBehavior": "when_no_match",
          "httpMethod": "GET",
          "type": "aws"
        }
      }
    },
    "/{folder}": {
      "get": {
        "produces": [
          "application/json"
        ],
        "parameters": [
          {
            "name": "folder",
            "in": "path",
            "required": true,
            "type": "string"
          }
        ],
        "responses": {
          "200": {
            "description": "200 response",
            "schema": {
              "$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": {
                "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.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": {
                "method.response.header.Content-Type":
 "integration.response.header.Content-Type",
                "method.response.header.Date": "integration.response.header.Date"
              }
            },
            "5\\d{2}": {
              "statusCode": "500"
            }
          },
          "requestParameters": {
            "integration.request.path.bucket": "method.request.path.folder"
          },
          "uri": "arn:aws:apigateway:us-west-2:s3:path/{bucket}",
          "passthroughBehavior": "when_no_match",
          "httpMethod": "DELETE",
          "type": "aws"
        }
      }
    },
    "/{folder}/{item}": {
      "get": {
        "produces": [
          "application/json"
        ],
        "parameters": [
          {
            "name": "item",
            "in": "path",
            "required": true,
            "type": "string"
          },
          {
            "name": "folder",
            "in": "path",
            "required": true,
            "type": "string"
          }
        ],
        "responses": {
          "200": {
            "description": "200 response",
            "schema": {
              "$ref": "#/definitions/Empty"
            },
            "headers": {
              "content-type": {
                "type": "string"
              },
              "Content-Type": {
                "type": "string"
              }
            }
          },
          "400": {
            "description": "400 response"
          },
          "500": {
            "description": "500 response"
          }
        },
        "security": [
          {
            "sigv4": []
          }
        ],
        "x-amazon-apigateway-integration": {
          "credentials": "arn:aws:iam::<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 Kinesis Proxy

This section describes how to create and configure an API Gateway API with an integration of the `AWS` type to access Kinesis.

> **Note**
> To integrate your API Gateway API with Kinesis, you must choose a region where both the API Gateway and Kinesis services are available. For region availability, see Regions and Endpoints.

For the purpose of illustration, we create an example API to enable a client to do the following:

1. List the user's available streams in 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 Kinesis
2. The `CreateStream`, `DescribeStream`, or `DeleteStream` action
3. The `GetRecords` or `PutRecords` (including `PutRecord`) action in Kinesis

Specifically, we 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 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 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 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 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 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 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 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 Kinesis. This helper method must be supplied to the `ListStreams` action in Kinesis.

You can apply the instructions presented here to other Kinesis actions. For the complete list of the 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. 210). 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 https://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 Kinesis

To allow the API to invoke 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 Kinesis, you can use the AmazonKinesisReadOnlyAccess policy that allows the `Get*`, `List*`, and `Describe*` actions in Kinesis to be invoked.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kinesis:Get*",
                "kinesis:List*",
                "kinesis:Describe*"
            ],
            "Resource": "*"
        }
    ]
}
```

This policy is available as an IAM-provisioned managed policy and its ARN is `arn:aws:iam::aws:policy/AmazonKinesisReadOnlyAccess`.

To enable read-write actions in Kinesis, you can use the AmazonKinesisFullAccess policy.

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

This policy is also available as an IAM-provisioned managed policy. 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 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 Kinesis

1. In the API Gateway console, choose **Create API**.
2. Choose **New API**.
3. In **API name**, type **`KinesisProxy`**. Leave the default values in the other fields.
4. Type a description in **Description**, if you like.
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 Kinesis

Kinesis supports the `ListStreams` action with the following REST API call:

```
POST /?Action=ListStreams HTTP/1.1
Host: kinesis.<region>.<domain>
Content-Length: <PayloadSizeBytes>
User-Agent: <UserAgentString>
Content-Type: application/x-amz-json-1.1
Authorization: <AuthParams>
X-Amz-Date: <Date>

{
   ...
}
```

In the above REST API request, the action is specified in the `Action` query parameter. Alternatively, you can specify the action in a `X-Amz-Target` header, instead:

```
POST / HTTP/1.1
Host: kinesis.<region>.<domain>
Content-Length: <PayloadSizeBytes>
User-Agent: <UserAgentString>
Content-Type: application/x-amz-json-1.1
Authorization: <AuthParams>
X-Amz-Date: <Date>
X-Amz-Target: Kinesis_20131202.ListStreams
{
   ...
}
```

In this tutorial, we use the query parameter to specify action.

To expose an Kinesis action in the API, add a `/streams` resource to the API's root. Then set a `GET` method on the resource and integrate the method with the `ListStreams` action of Kinesis.

The following procedure describes how to list Kinesis streams by using the API Gateway console.

To list Kinesis streams by using the API Gateway console

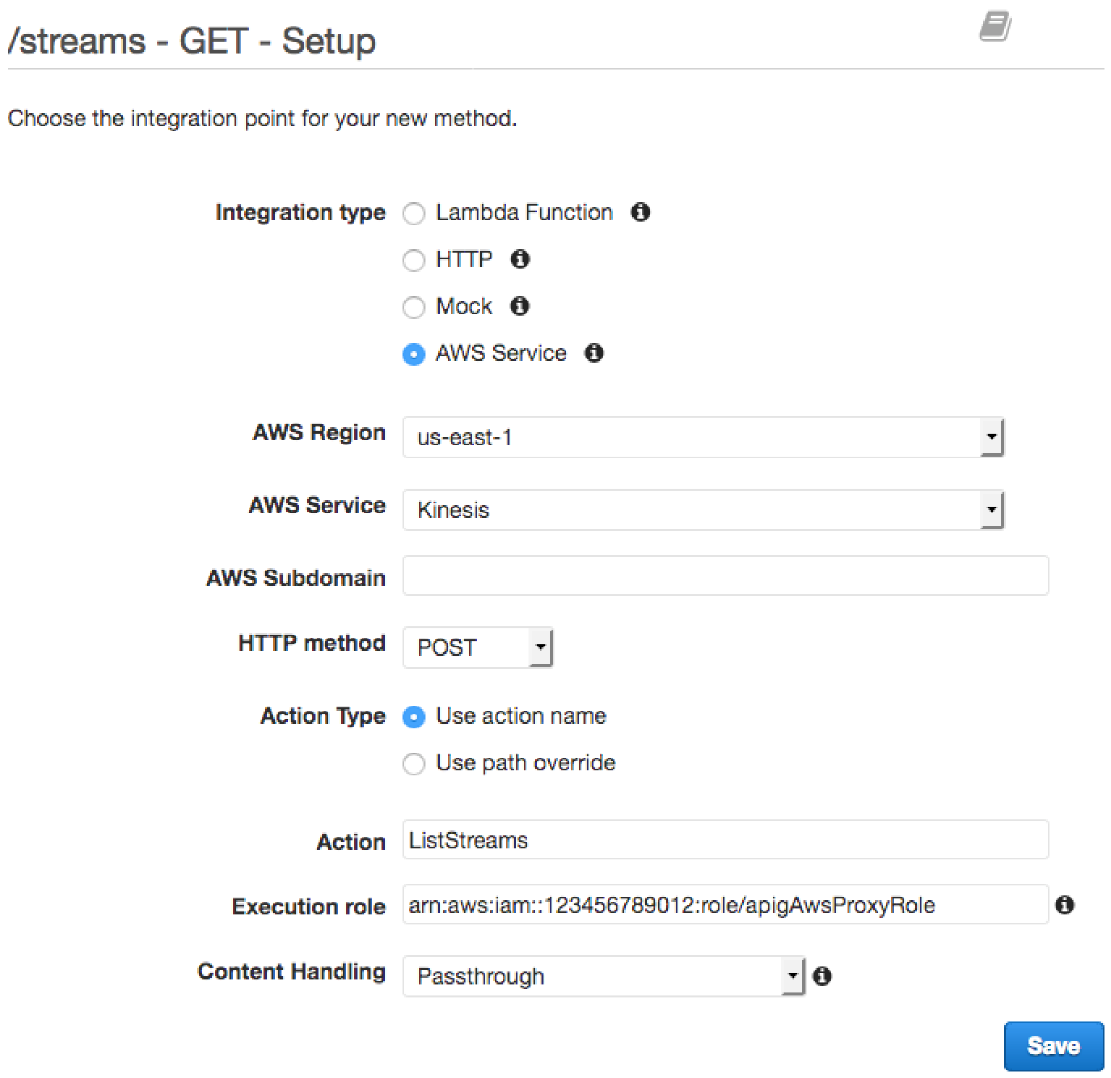
1. Select the API root resource. In **Actions**, choose **Create Resource**.

   In **Resource Name**, type `Streams`, leave **Resource Path** and other fields as the default, and choose **Create Resource**.
2. Choose the `/Streams` resource. From **Actions**, choose **Create Method**, choose **GET** from the list, and then choose the check mark icon to finish creating the method.

   Note
   The HTTP verb for a method invoked by a client may differ from the HTTP verb for an integration required by the backend. We chose `GET` here, because listing streams is intuitively a READ operation.
3. In the method's **Setup** pane, choose **AWS Service**.

   a. For **AWS Region**, choose a region (e.g., **us-east-1**).
   b. For **AWS Service**, choose **Kinesis**.
   c. Leave **AWS Subdomain** blank.
   d. For **HTTP method**, choose **POST**.

      Note
      We chose `POST` here because Kinesis requires that the `ListStreams` action be invoked with it.
   e. For **Action Type**, choose **Use action name**.
   f. For **Action**, type **`ListStreams`**.
   g. For **Execution role**, type the ARN for your execution role.
   h. Leave the default of **Passthrough** for **Content Handling**.
   i. Choose **Save** to finish the initial setup of the method.



/streams - GET - Setup

Choose the integration point for your new method.

Integration type	Lambda Function / HTTP / Mock / [✓ AWS Service]
AWS Region	us-east-1
AWS Service	Kinesis
AWS Subdomain	
HTTP method	POST
Action Type	[✓ Use action name] / Use path override
Action	ListStreams
Execution role	arn:aws:iam::123456789012:role/apigAwsProxyRole
Content Handling	Passthrough

4. Still 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 check mark icon to save the setting.

   We used a request parameter mapping to set the `Content-Type` header to the static value of `'application/x-amz-json-1.1'` to inform Kinesis that the input is of a specific version of JSON.

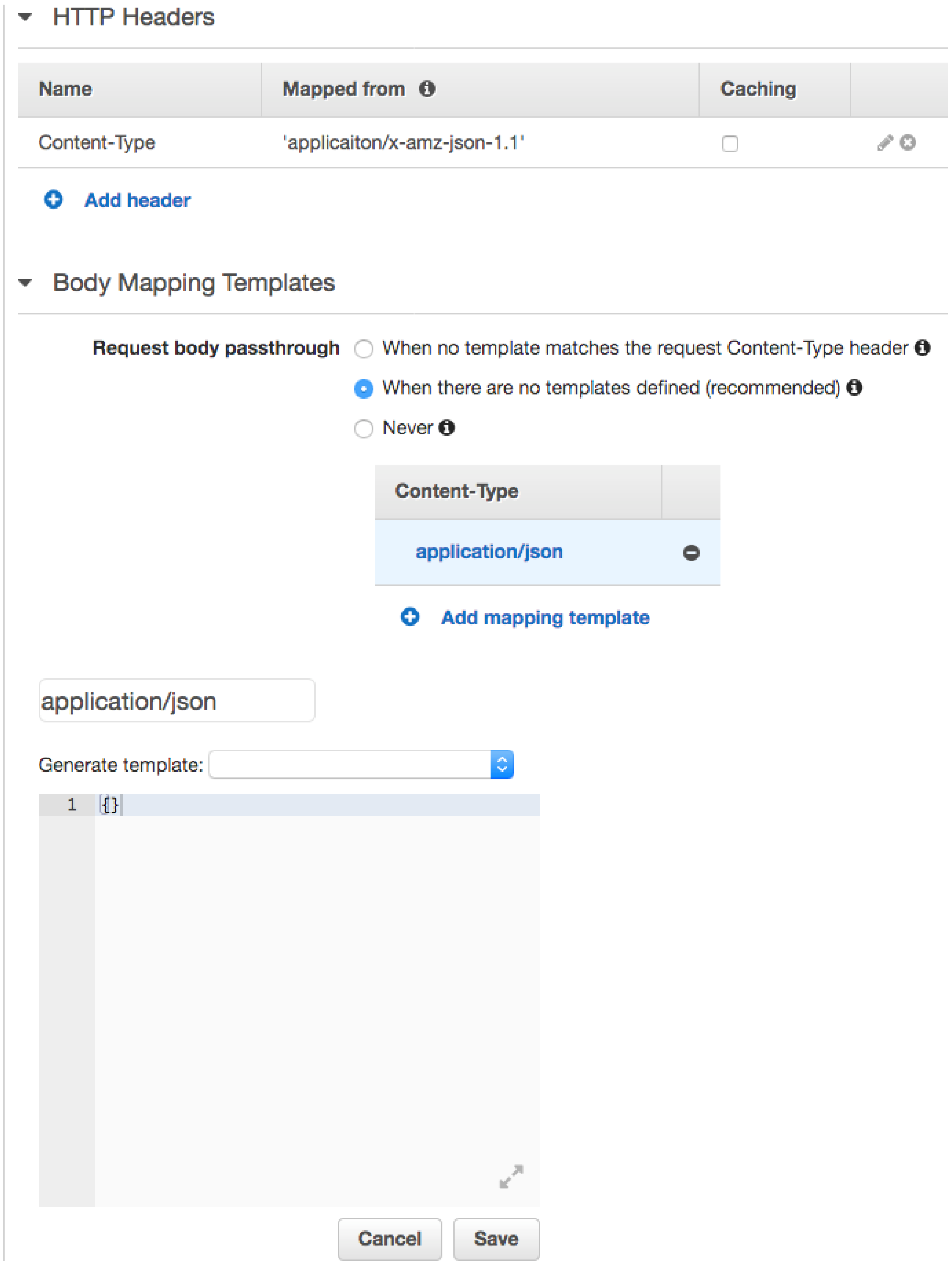
5. Expand the **Body Mapping Templates** section:

   a. Choose **Add mapping template**.
   b. For **Content-Type**, type `application/json`.
   c. Choose the check mark icon to save the **Content-Type** setting. Choose **Yes, secure this integration** in **Change passthrough behavior**.
   d. Type `{}` in the template editor.
   e. Choose the **Save** button 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.



Name	Mapped from	Caching
Content-Type	'applicaiton/x-amz-json-1.1'	☐

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

Content-Type
application/json

application/json

Generate template:

```
{}
```

6. Test the GET method on the Streams resource to invoke the `ListStreams` action in 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 already created two streams named "myStream" and "yourStream" in Kinesis, the successful test returns a 200 OK response containing the following payload:

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

Create, Describe, and Delete a Stream in Kinesis

Creating, describing, and deleting a stream in Kinesis involves making the following 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"
}
```

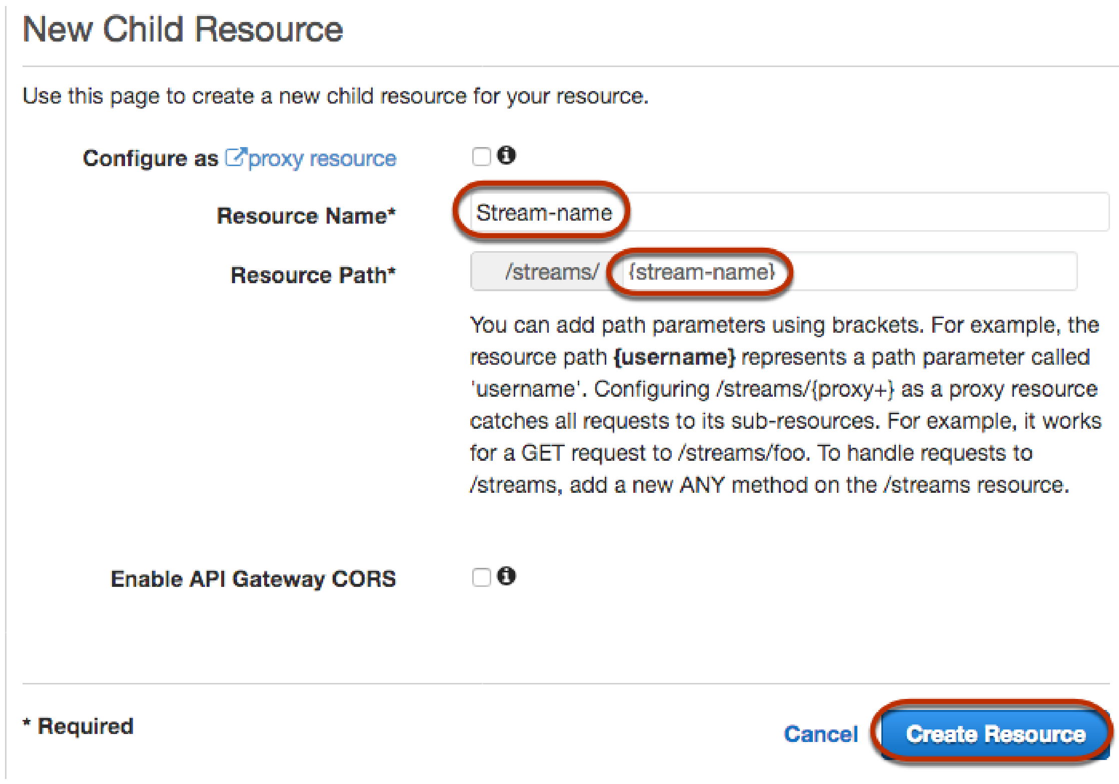
```
POST /?Action=DeleteStream HTTP/1.1
Host: kinesis.region.domain
...
Content-Type: application/x-amz-json-1.1
Content-Length: PayloadSizeBytes

{
    "StreamName":"string"
}
```

We can build the 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 create our API somewhat differently.

We expose the `GET`, `POST`, and `Delete` HTTP methods on a to-be-named `Stream` resource. We use the `{stream-name}` path variable as the placeholder of the stream resource and integrate these API methods with the 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. We provide mapping templates to transform the data to the required integration request payload.

To configure and test the GET method on a stream resource

1. Create a child resource with the `{stream-name}` path variable under the previously created `/streams` resource.



New Child Resource

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

Configure as proxy resource ☐

Resource Name* Stream-name

Resource Path* /streams/ {stream-name}

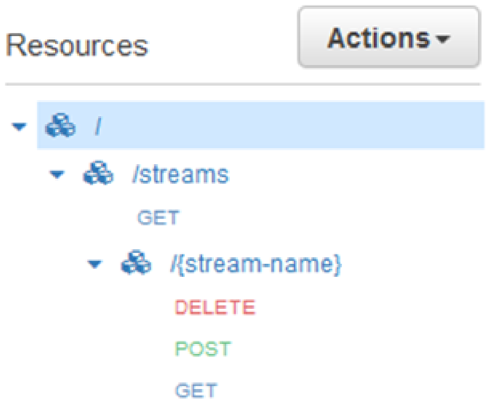
You can add path parameters using brackets. For example, the resource path **{username}** represents a path parameter called 'username'. Configuring /streams/{proxy+} as a proxy resource catches all requests to its sub-resources. For example, it works for a GET request to /streams/foo. To handle requests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

Enable API Gateway CORS ☐

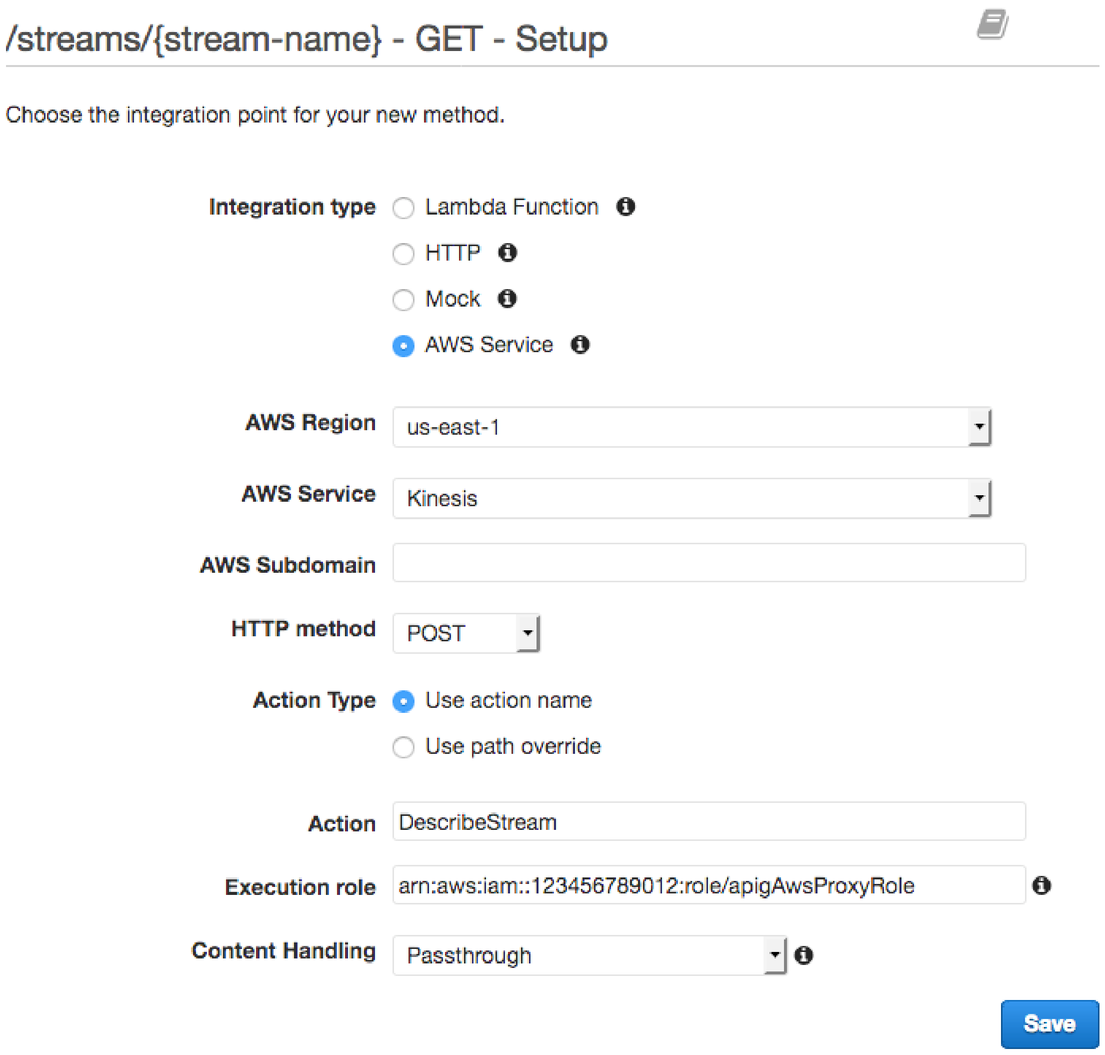
* Required

2. Add the `POST`, `GET`, and `DELETE` HTTP verbs to this resource.

   After the methods are created on the resource, the structure of the API looks like the following:



3. Set up the `GET /streams/{stream-name}` method to call the `POST /?Action=DescribeStream` action in Kinesis, as shown in the following.


/streams/{stream-name} - GET - Setup

Choose the integration point for your new method.

Integration type	○ Lambda Function ○ HTTP ○ Mock [✓ AWS Service]
AWS Region	us-east-1
AWS Service	Kinesis
AWS Subdomain	
HTTP method	POST
Action Type	[✓ Use action name] ○ Use path override
Action	DescribeStream
Execution role	arn:aws:iam::123456789012:role/apigAwsProxyRole
Content Handling	Passthrough

4. Add the following `Content-Type` header mapping to the integration request:

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

The task follows the same procedure to set up the request parameter mapping for the `GET /streams` method.

5. Add the following body mapping template to map data from the `GET /streams/{stream-name}` method request to the `POST /?Action=DescribeStream` integration request:

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

This mapping template generates the required integration request payload for the `DescribeStream` action of Kinesis from the method request's `stream-name` path parameter value.

6. Test the `GET /stream/{stream-name}` method to invoke the `DescribeStream` action in 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 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": "272225893536750770770699685945414569164"
        },
        "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 /streams/{stream-name}` method to call `POST /?Action=CreateStream` action in Kinesis. The task follows the same procedure to set up the `GET /streams/{stream-name}` method provided that you replace the `DescribeStream` action by `CreateStream`.
2. Add the following `Content-Type` header mapping to the integration request:

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

   The task follows the same procedure to set up the request parameter mapping for the `GET /streams` method.
3. Add the following body mapping template to map data from the `POST /streams/{stream-name}` method request to the `POST /?Action=CreateStream` integration request:

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

   In the preceding mapping template, we set `ShardCount` to a fixed value of 5 if the client does not specify a value in the method request payload.
4. Test the `POST /streams/{stream-name}` method to create a named stream in 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 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 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 /streams/{stream-name}` method to integrate with the `POST /?Action=DeleteStream` action in Kinesis. The task follows the same procedure to set up the `GET /streams/{stream-name}` method provided that you replace the `DescribeStream` action by `DeleteStream`.
2. Add the following `Content-Type` header mapping to the integration request:

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

   The task follows the same procedure to set up the request parameter mapping for the `GET /streams` method.
3. Add the following body mapping template to map data from the `DELETE /streams/{stream-name}` method request to the corresponding integration request of `POST /?Action=DeleteStream`:

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

   This mapping template generates the required input for the `DELETE /streams/{stream-name}` action from the client-supplied URL path name of `stream-name`.
4. Test the DELETE method to delete a named stream in 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 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 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 Kinesis

After you create a stream in 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 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 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 Kinesis action to obtain a shard iterator:

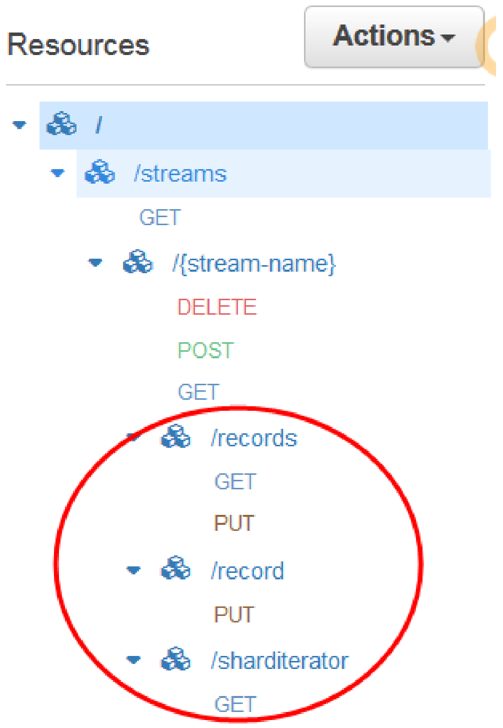
```
POST /?Action=GetShardIterator HTTP/1.1
Host: kinesis.region.domain
Authorization: AWS4-HMAC-SHA256 Credential=..., ...
...
Content-Type: application/x-amz-json-1.1
Content-Length: PayloadSizeBytes
```

```
{
    "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`).

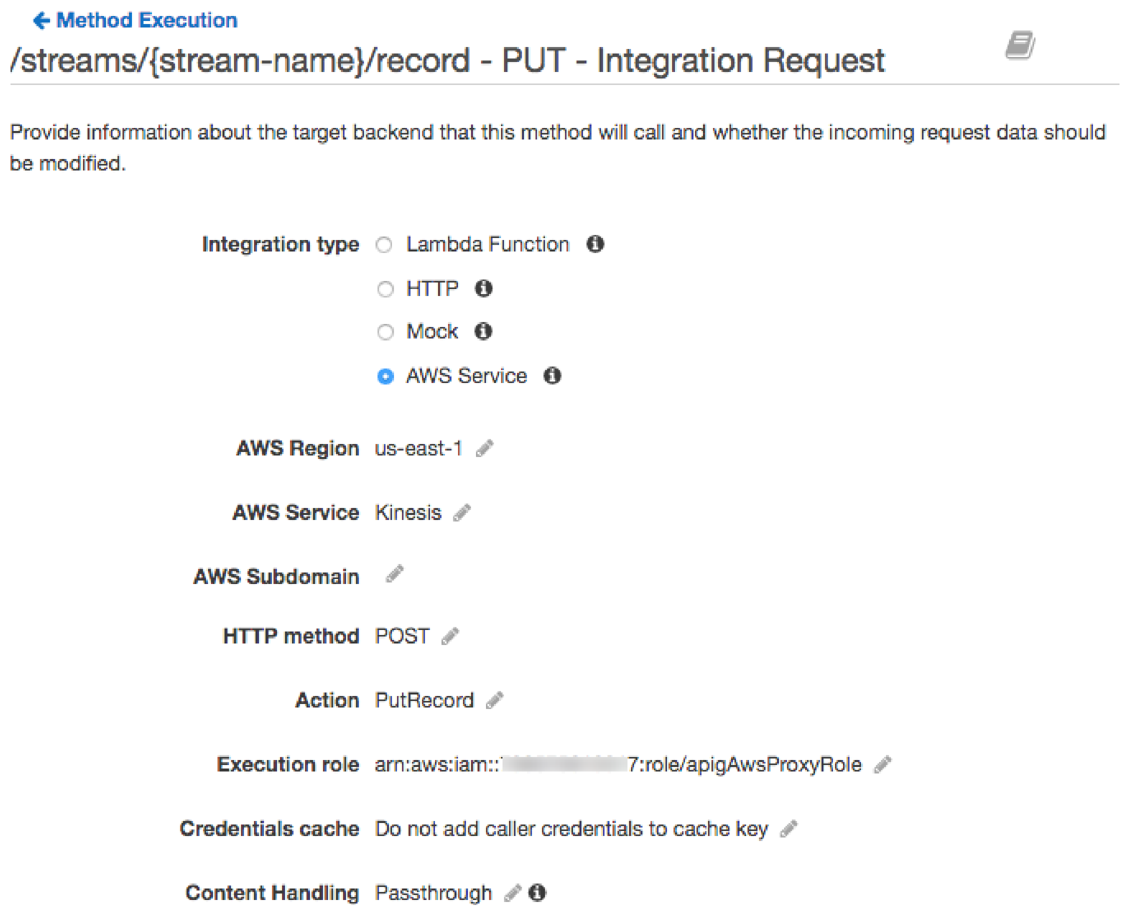
The following figure shows the API structure of resources after the methods are created:



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 set up and test the `PUT /streams/{stream-name}/record` method to invoke `PutRecord` in Kinesis:

1. Set up the PUT method, as shown in the following:



← 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 / Mock / [✓ AWS Service]
AWS Region	us-east-1
AWS Service	Kinesis
AWS Subdomain	
HTTP method	POST
Action	PutRecord
Execution role	arn:aws:iam::[redacted]7:role/apigAwsProxyRole
Credentials cache	Do not add caller credentials to cache key
Content Handling	Passthrough

2. Add the following request parameter mapping to set the `Content-Type` header to an AWS-compliant version of JSON in the integration request:

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

   The task follows the same procedure to set up the request parameter mapping for the `GET /streams` method.

3. Add the following body mapping template to map data from the `PUT /streams/{stream-name}/record` method request to the corresponding integration request of `POST /?Action=PutRecord`:

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

   This 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:

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

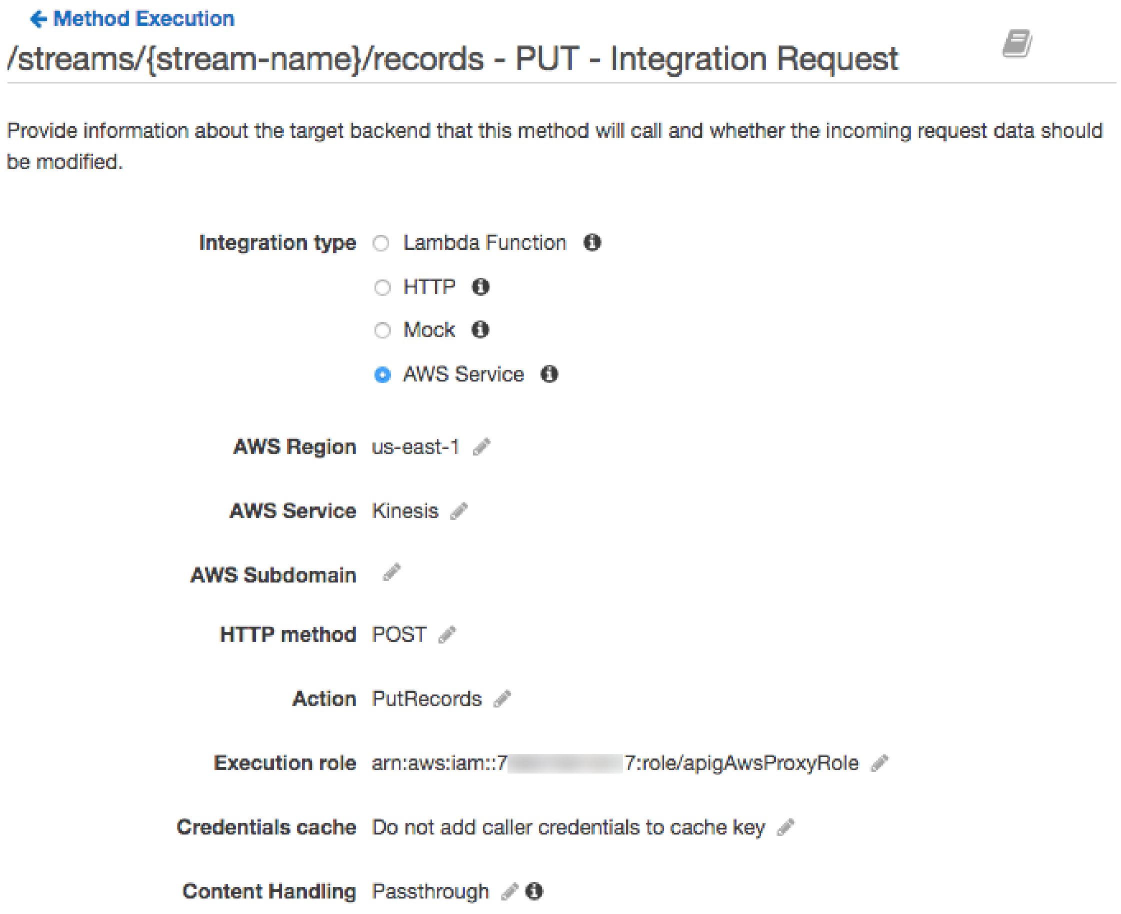
You can create a model to include this schema and use the model to facilitate generating the mapping template. However, you can generate a mapping template without using any model.

4. To test the `PUT /streams/{stream-name}/record` method, set the `stream-name` path variable to the name of an existing stream, supply a payload of the required 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 set up and test the `PUT /streams/{stream-name}/records` method to invoke `PutRecords` in Kinesis

1. Set up the `PUT /streams/{stream-name}/records` 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 ○ Mock [● AWS Service]
AWS Region	us-east-1
AWS Service	Kinesis
AWS Subdomain	
HTTP method	POST
Action	PutRecords
Execution role	arn:aws:iam::7[redacted]7:role/apigAwsProxyRole
Credentials cache	Do not add caller credentials to cache key
Content Handling	Passthrough

2. Add the following request parameter mapping to set the `Content-Type` header to an AWS-compliant version of JSON in the integration request:

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

   The task follows the same procedure to set up the request parameter mapping for the `GET /streams` method.

3. Add the following body mapping template to map data from the `PUT /streams/{stream-name}/records` method request to the corresponding integration request of `POST /?Action=PutRecords` :

```
{
    "StreamName": "$input.params('stream-name')",
    "Records": [
       #foreach($elem in $input.path('$.records'))
          {
            "Data": "$util.base64Encode($elem.data)",
            "PartitionKey": "$elem.partition-key"
          }#if($foreach.hasNext),#end
        #end
    ]
}
```

   This mapping template assumes that the method request payload can be modelled by the 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" }
        }
      }
    }
  }
}
```

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.

In this tutorial, we used two slightly different payload formats to illustrate that an API developer can choose to expose the backend data format to the client or hide it from the client. One format is for the `PUT /streams/{stream-name}/records` method (above). Another format is used for the `PUT /streams/{stream-name}/record` method (in the previous procedure). In production environment, you should keep both formats consistent.

4. To test the `PUT /streams/{stream-name}/records` method, set the `stream-name` path variable to an existing stream, supply the following payload, and submit the method request.

```

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

```

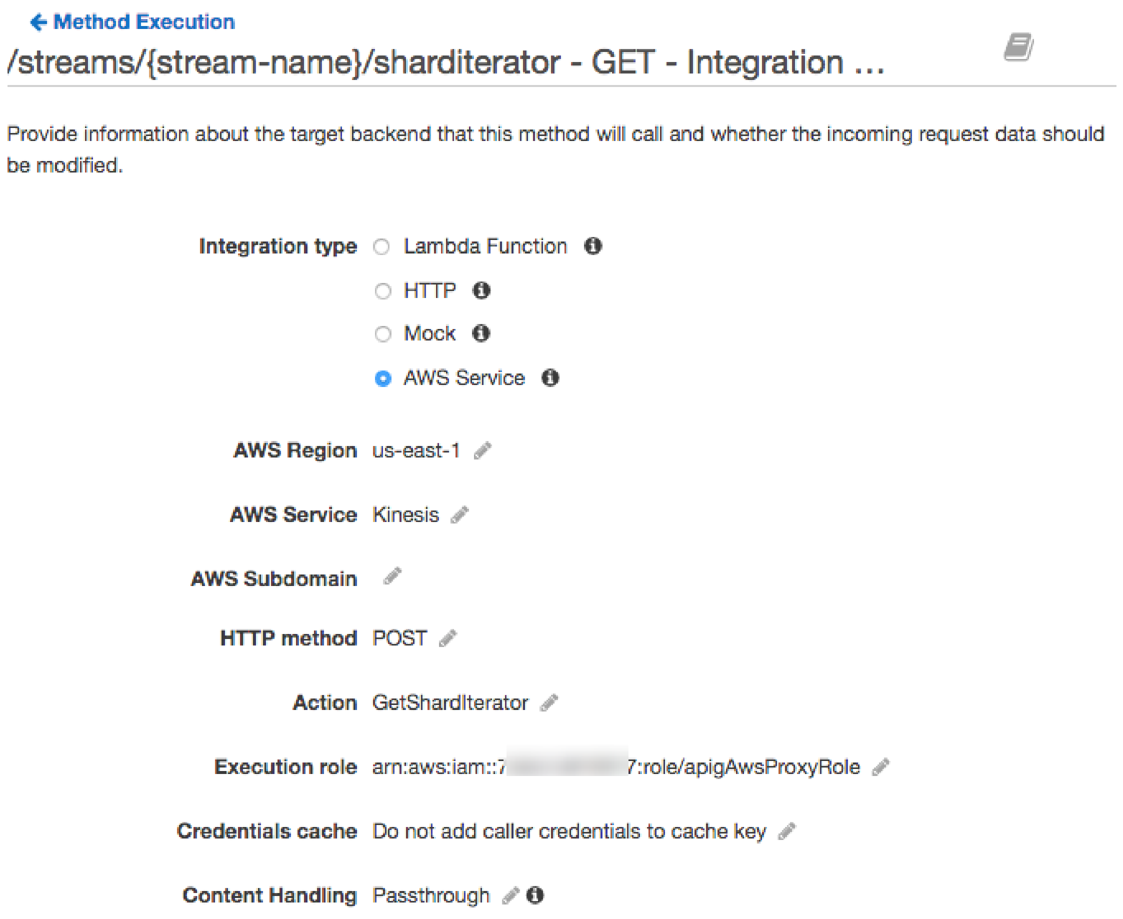
The successful result is a 200 OK response with a payload similar to the following output:

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

To set up and test the `GET /streams/{stream-name}/sharditerator` method invoke `GetShardIterator` in Kinesis

The `GET /streams/{stream-name}/sharditerator` method is a helper method to acquire a required shard iterator before calling the `GET /streams/{stream-name}/records` method.

1. Set up integration for the `GET /streams/{stream-name}/sharditerator` method, as shown in the following:



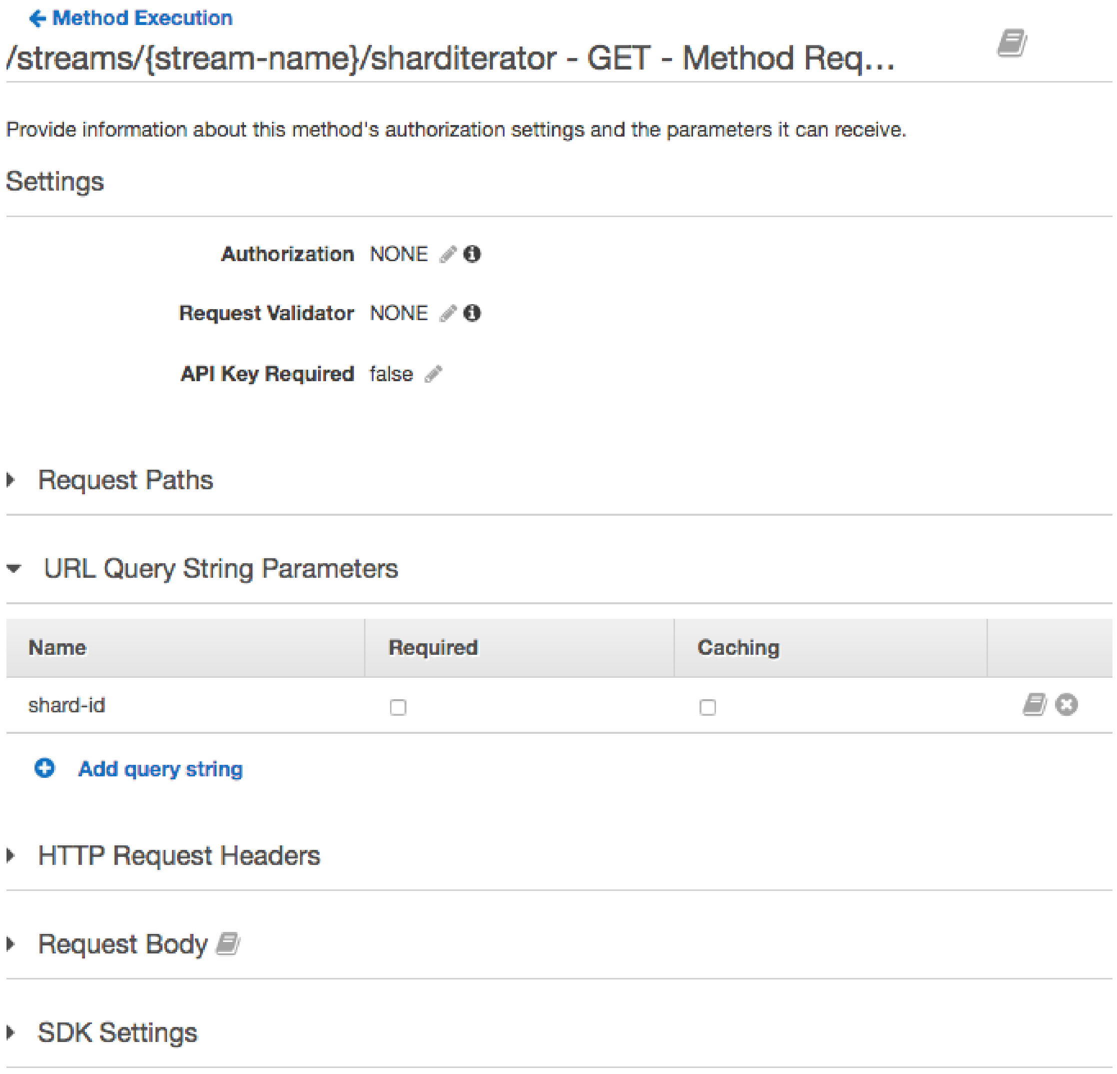
← Method Execution

/streams/{stream-name}/sharditerator - 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 [✓ AWS Service] HTTP Mock
AWS Region	us-east-1
AWS Service	Kinesis
AWS Subdomain	
HTTP method	POST
Action	GetShardIterator
Execution role	arn:aws:iam::7[redacted]7:role/apigAwsProxyRole
Credentials cache	Do not add caller credentials to cache key
Content Handling	Passthrough

2. 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:



/streams/{stream-name}/sharditerator - GET - Method Req...

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

Settings

Authorization	NONE
Request Validator	NONE
API Key Required	false

Name	Required	Caching	
shard-id	☐	☐	

In the following body-mapping template, we set the `shard-id` query parameter value to the `ShardId` property value of the JSON payload as the input to the `GetShardIterator` action in Kinesis.

3. Configure the body mapping template to generate the required input (`ShardId` and `StreamName`) to the `GetShardIterator` action from the `shard-id` and `stream-name` parameters of the method request. In addition, the mapping template also sets `ShardIteratorType` to `TRIM_HORIZON` as a default.

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

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**.

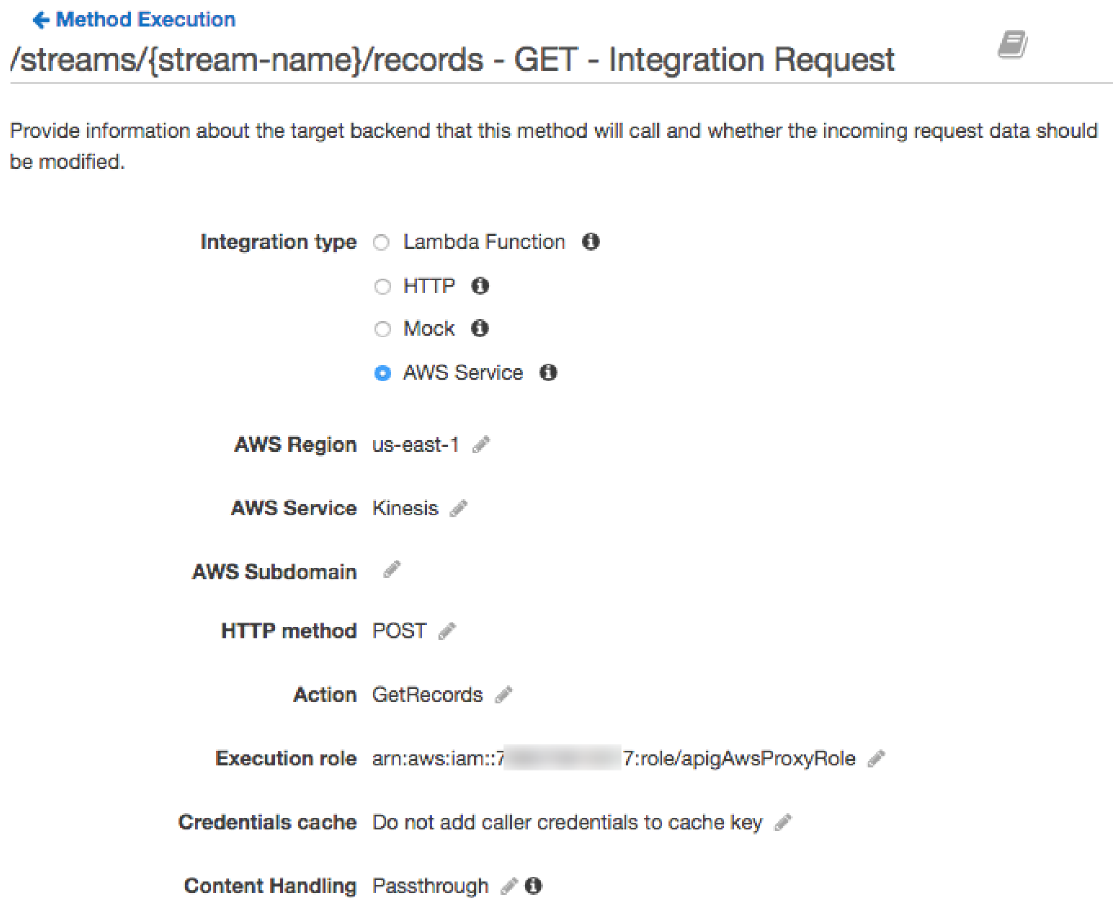
   The successful response payload is similar to the following output:

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

Make note of the `ShardIterator` value. You need it to get records from a stream.

To configure and test the `GET /streams/{stream-name}/records` method to invoke the `GetRecords` action in Kinesis

1. Set up the `GET /streams/{stream-name}/records` method, as shown in the following:

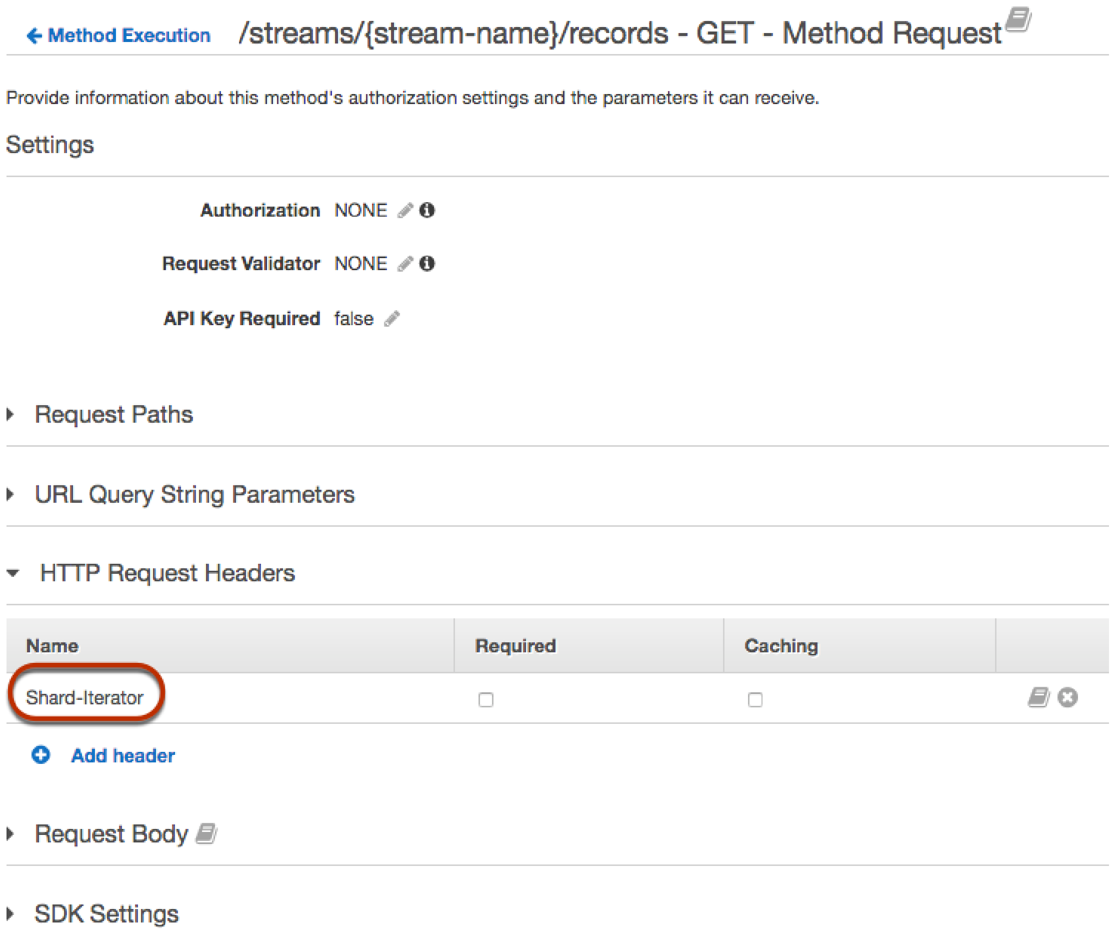


/streams/{stream-name}/records - GET - Integration Request

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]
AWS Region	us-east-1
AWS Service	Kinesis
AWS Subdomain	
HTTP method	POST
Action	GetRecords
Execution role	arn:aws:iam::7[redacted]7:role/apigAwsProxyRole
Credentials cache	Do not add caller credentials to cache key
Content Handling	Passthrough

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:



/streams/{stream-name}/records - GET - Method Request

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

Settings

Authorization NONE

Request Validator NONE

API Key Required false

Name	Required	Caching
Shard-Iterator	☐	☐

3. Set up the following mapping template to map the `Shard-Iterator` header parameter value to the `ShardIterator` property value of the JSON payload for the `GetRecords` action in Kinesis.

```
{
    "ShardIterator": "$input.params('Shard-Iterator')"
}
```

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 /streams/{stream-name}/sharditerator` method (above), and choose **Test**.

   The successful response payload is similar to the following output:

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

Swagger Definitions of a Sample API as an Kinesis Proxy

The following shows the Swagger Definitions of the sample API as an Kinesis proxy used for this tutorial.

```
{
  "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": {
              "$ref": "#/definitions/Empty"
            }
          }
        },
        "x-amazon-apigateway-integration": {
          "credentials": "arn:aws:iam::123456789012:role/apigAwsProxyRole",
          "responses": {
            "default": {
              "statusCode": "200"
            }
          },
          "requestTemplates": {
            "application/json": "{\n}"
          },
          "uri": "arn:aws:apigateway:us-east-1:kinesis:action/ListStreams",
          "httpMethod": "POST",
          "requestParameters": {
            "integration.request.header.Content-Type": "'application/x-amz-json-1.1'"
          },
          "type": "aws"
        }
      }
    },
    "/streams/{stream-name}": {
      "get": {
        "consumes": [
          "application/json"
        ],
        "produces": [
          "application/json"
        ],
        "parameters": [
          {
            "name": "stream-name",
            "in": "path",
            "required": true,
            "type": "string"
          }
        ],
        "responses": {
          "200": {
            "description": "200 response",
            "schema": {
              "$ref": "#/definitions/Empty"
            }
          }
        },
        "x-amazon-apigateway-integration": {
          "credentials": "arn:aws:iam::123456789012:role/apigAwsProxyRole",
          "responses": {
            "default": {
              "statusCode": "200"
            }
          },
          "requestTemplates": {
            "application/json": "{\n    \"StreamName\": \"$input.params('stream-
name')\"\n}"
          },
          "uri": "arn:aws:apigateway:us-east-1:kinesis:action/DescribeStream",
          "httpMethod": "POST",
          "type": "aws"
        }
      },
      "post": {
        "consumes": [
          "application/json"
        ],
        "produces": [
          "application/json"
        ],
        "parameters": [
          {
            "name": "stream-name",
            "in": "path",
            "required": true,
            "type": "string"
          }
        ],
        "responses": {
          "200": {
            "description": "200 response",
            "schema": {
              "$ref": "#/definitions/Empty"
            }
          }
        },
        "x-amazon-apigateway-integration": {
          "credentials": "arn:aws:iam::123456789012:role/apigAwsProxyRole",
          "responses": {
            "default": {
              "statusCode": "200"
            }
          },
          "requestTemplates": {
            "application/json": "{\n    \"ShardCount\": 5,\n    \"StreamName\":
 \"$input.params('stream-name')\"\n}"
          },
          "uri": "arn:aws:apigateway:us-east-1:kinesis:action/CreateStream",
          "httpMethod": "POST",
          "requestParameters": {
            "integration.request.header.Content-Type": "'application/x-amz-json-1.1'"
          },
          "type": "aws"
        }
      },
      "delete": {
        "consumes": [
          "application/json"
        ],
        "produces": [
          "application/json"
        ],
        "parameters": [
          {
            "name": "stream-name",
            "in": "path",
            "required": true,
            "type": "string"
          }
        ],
        "responses": {
          "200": {
            "description": "200 response",
            "schema": {
              "$ref": "#/definitions/Empty"
            },
            "headers": {
              "Content-Type": {
                "type": "string"
              }
            }
          },
          "400": {
            "description": "400 response",
            "headers": {
              "Content-Type": {
                "type": "string"
              }
            }
          },
          "500": {
            "description": "500 response",
            "headers": {
              "Content-Type": {
                "type": "string"
              }
            }
          }
        },
        "x-amazon-apigateway-integration": {
          "credentials": "arn:aws:iam::123456789012:role/apigAwsProxyRole",
          "responses": {
            "4\\d{2}": {
              "statusCode": "400",
              "responseParameters": {
                "method.response.header.Content-Type":
"integration.response.header.Content-Type"
              }
            },
            "default": {
              "statusCode": "200",
              "responseParameters": {
                "method.response.header.Content-Type":
"integration.response.header.Content-Type"
              }
            },
            "5\\d{2}": {
              "statusCode": "500",
              "responseParameters": {
                "method.response.header.Content-Type":
 "integration.response.header.Content-Type"
              }
            }
          },
          "requestTemplates": {
            "application/json": "{\n    \"StreamName\": \"$input.params('stream-
name')\"\n}"
          },
          "uri": "arn:aws:apigateway:us-east-1:kinesis:action/DeleteStream",
          "httpMethod": "POST",
          "requestParameters": {
            "integration.request.header.Content-Type": "'application/x-amz-json-1.1'"
          },
          "type": "aws"
        }
      }
    },
    "/streams/{stream-name}/record": {
      "put": {
        "consumes": [
          "application/json"
        ],
        "produces": [
          "application/json"
        ],
        "parameters": [
          {
            "name": "stream-name",
            "in": "path",
            "required": true,
            "type": "string"
          }
        ],
        "responses": {
          "200": {
            "description": "200 response",
            "schema": {
              "$ref": "#/definitions/Empty"
            }
          }
        },
        "x-amazon-apigateway-integration": {
          "credentials": "arn:aws:iam::123456789012:role/apigAwsProxyRole",
          "responses": {
            "default": {
              "statusCode": "200"
            }
          },
          "requestTemplates": {
            "application/json": "{\n    \"StreamName\": \"$input.params('stream-name')\",
\n    \"Data\": \"$util.base64Encode($input.json('$.Data'))\",\n    \"PartitionKey\":
 \"$input.path('$.PartitionKey')\"\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"
          }
        ],
        "responses": {
          "200": {
            "description": "200 response",
            "schema": {
              "$ref": "#/definitions/Empty"
            }
          }
        },
        "x-amazon-apigateway-integration": {
          "credentials": "arn:aws:iam::123456789012:role/apigAwsProxyRole",
          "responses": {
            "default": {
              "statusCode": "200"
            }
          },
          "requestTemplates": {
            "application/json": "{\n    \"ShardIterator\": \"$input.params('Shard-
Iterator')\"\n}"
          },
          "uri": "arn:aws:apigateway:us-east-1:kinesis:action/GetRecords",
          "httpMethod": "POST",
          "requestParameters": {
            "integration.request.header.Content-Type": "'application/x-amz-json-1.1'"
          },
          "type": "aws"
        }
      },
      "put": {
        "consumes": [
          "application/json",
          "application/x-amz-json-1.1"
        ],
        "produces": [
          "application/json"
        ],
        "parameters": [
          {
            "name": "Content-Type",
            "in": "header",
            "required": false,
            "type": "string"
          },
          {
            "name": "stream-name",
            "in": "path",
            "required": true,
            "type": "string"
          },
          {
            "in": "body",
            "name": "PutRecordsMethodRequestPayload",
            "required": true,
            "schema": {
              "$ref": "#/definitions/PutRecordsMethodRequestPayload"
            }
          }
        ],
        "responses": {
          "200": {
            "description": "200 response",
            "schema": {
              "$ref": "#/definitions/Empty"
            }
          }
        },
        "x-amazon-apigateway-integration": {
          "credentials": "arn:aws:iam::123456789012:role/apigAwsProxyRole",
          "responses": {
            "default": {
              "statusCode": "200"
            }
          },
          "requestTemplates": {
            "application/json": "{\n    \"StreamName\": \"$input.params('stream-name')\",
\n    \"Records\": [\n       #foreach($elem in $input.path('$.records'))\n          {\n
           \"Data\": \"$util.base64Encode($elem.data)\",\n            \"PartitionKey\":
 \"$elem.partition-key\"\n          }#if($foreach.hasNext),#end\n        #end\n    ]\n}",
            "application/x-amz-json-1.1": "#set($inputRoot = $input.path('$'))\n{\n
 \"StreamName\": \"$input.params('stream-name')\",\n  \"records\" : [\n    #foreach($elem
 in $inputRoot.records)\n    {\n        \"Data\" : \"$elem.data\",\n        \"PartitionKey
\" : \"$elem.partition-key\"\n    }#if($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, Pricing and Known Issues

Topics

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	10000 request per second (rps) with an additional burst capacity provided by the token bucket algorithm, using a maximum bucket capacity of 5000 requests.	Yes
APIs per account	60	Yes
API keys per account	500	Yes

Resource or Operation	**Default Limit**	**Can Be Increased**
Client certificates per account	60	Yes
Custom authorizers per API	10	Yes
Documentation parts per API	2000	Yes
Resources per API	300	Yes
Stages per API	10	Yes
Usage plans per account	300	Yes
Usage plans per API key	10	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 `#foreach ... #end` loop in mapping templates	1000	No
ARN length of a method with authorization	1600 bytes	No

For restapi:import or restapi:put, the maximum size of the API definition file is 2MB.

All of the per API limits can only be increased on specific APIs.

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 minute per account	No
DeleteRestApi	2 requests per minute per account	No
CreateDeployment	3 requests per minute per account	No
UpdateAccount	3 requests per minute per account	No
GetResources	150 requests per minute per account	No
CreateResource	300 requests per minute per account	No
DeleteResource	300 requests per minute per account	No
CreateDomainName	2 requests per minute per account	No
UpdateUsagePlan	3 requests per minute 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 requests.	No

API Gateway Pricing

For general API Gateway region-specific pricing information, see Amazon API Gateway Pricing.

The following lists the exceptions of the general pricing scheme:

- API caching in Amazon API Gateway is not eligible for the AWS Free Tier.
- Calling methods with the authorization type of `AWS_IAM`, `CUSTOM`, and `COGNITO_USER_POOLS` are not charged for authorization and authentication failures.
- Calling methods requiring API keys are not charged when API keys are missing or invalid.
- API Gateway-throttled requests are not charged when the request rate or burst exceed the pre-configured limits.
- Usage plan-throttled requests are not charged when rate limits or quota exceed the pre-configured limits.

Known Issues

- The plain text pipe character (`|`) is not supported for any request URL query string and must be URL-encoded.
- Paths of `/ping` and `/sping` are reserved for the service health check. Use of these for API root-level resources with custom domains will fail to produce the expected result.
- 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 backend, the intermediate certificate is missing from the certificate chain, or any other unrecognizable certificate-related exceptions thrown by the backend.
- The following backends may not support SSL client authentication in a compatible way with API Gateway:
  - NGINX
- API Gateway supports most of the Swagger specification, with the following exceptions:
  - API Gateway models are defined using JSON Schema, instead of the JSON schema used by Swagger.
  - The `additionalProperties` field is not supported in Models.
  - the `allOff` field is not supported in Models.
  - The `discriminator` parameter is not supported in any schema object.
  - The `example` tag is not supported.
  - The `readOnly` field is not supported.
  - Response definitions of the `"500": {"$ref": "#/responses/UnexpectedError"}` form is not supported in the Swagger document root. To work around this, replace the reference by the inline schema.
  - Numbers of the `Int32` or `Int64` type is not supported. An example is shown as follows:

```
"elementId": {
    "description": "Working Element Id",
    "format": "int32",
    "type": "number"
}
```

  - Decimal number format type (`"format": "decimal"`) is not supported in a schema definition.
  - In method responses, schema definition must be of an object type and cannot be of primitive types. For example, `"schema": { "type": "string"}` is not supported. However, you can work around this using the following object type:

```
"schema": {
     "$ref": "#/definitions/StringResponse"
            }

 "definitions": {
    "StringResponse": {
      "type": "string"
    }
  }
```

- 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`
- The Android SDK of an API generated by API Gateway uses the `java.net.HttpURLConnection` class. This class will throw an unhandled exception, on devices running Android 4.4 and earlier, for a 401 response resulted from remapping of the `WWW-Authenticate` header to `X-Amzn-Remapped-WWW-Authenticate`.

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:** June 6, 2017

Change	Description	Date Changed
Customizing gateway responses	Customize API Gateway-generated gateway responses to API requests that failed to reach the integration backend. A customized gateway message can provide the caller with API-specific custom error messages, including returning needed CORS headers, or can transform the gateway response data to a format of an external exchange. For more information, see Map Method Request Data to Customize Gateway Responses (p. 147).	June 6, 2017
Mapping Lambda custom error properties to method response headers	Map individual custom error properties returned from Lambda to the method response header parameters using the `integration.response.body` parameter, relying API Gateway to deserialize the stringified custom error object at run time. For more information, see Handle Custom Lambda Errors in API Gateway (p. 90).	June 6, 2017
Throttling limits increase	Increase the account-level steady-state request rate limit to 10,000 requests per second (rps) and the bust limit to 5000 concurrent requests. For more information, see Throttle API Requests for Better Throughput (p. 289).	June 6, 2017
Validating method requests	Configure basic request validators on the API level or method levels so that API Gateway can validate incoming requests. API Gateway verifies that required parameters are set and not blank, and verifies that the format of applicable payloads conforms to the configured model. For more information, see Enable Basic Request Validation for an API in API Gateway (p. 160).	April 11, 2017
Integrating with ACM	Use ACM Certificates for your API's custom domain names. You can create a certificate in AWS Certificate Manager or import	March 9, 2017

Change	Description	Date Changed
	an existing PEM-formatted certificate into ACM. You then refer to the certificate's ARN when setting a custom domain name for your APIs. For more information, see Set up Custom Domain Name for API Host Name (p. 320).	
Generating and calling a Java SDK of an API	Let API Gateway generate the Java SDK for your API and use the SDK to call the API in your Java client. For more information, see Use a Java SDK Generated by API Gateway (p. 335).	January 13, 2017
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 Sell Your API as SaaS (p. 329).	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 `DocumentationPart` 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 Document an API (p. 171).	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 `context` map and an IAM policy. For more information, see Output from an Amazon API Gateway Custom Authorizer (p. 235).	December 1, 2016
Supporting binary payloads	Set binaryMediaTypes on your API to support binary payloads of a request or response. Set the `contentHandling` property on an Integration or IntegrationResponse to specify whether to handle a binary payload as the native binary blob, as a Base64-enocded string, or as a passthrough without modifications. For more information, see Enable Support for Binary Payloads in API Gateway (p. 92).	November 17, 2016
Enabling a proxy integration with an HTTP or Lambda backend through a proxy resource of an API.	Create a proxy resource with a greedy path parameter of the form `{proxy+}` and the catch-all `ANY` method. The proxy resource is integrated with an HTTP or Lambda backend using the HTTP or Lambda proxy integration, respectively. For more information, see Set up Proxy Integration (p. 111).	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. 267).	August 11, 2016

Change	Description	Date Changed
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 User Pools (p. 239).	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. 353).	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 Rotate a Certificate Imported into ACM (p. 328).	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. 9) and Build an API Gateway API to Expose an HTTP Endpoint (p. 8).	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. 210).	April 5, 2016
Exposing the `$input.body` variable to access the raw payload as string and the `$util.parseJson()` function to turn a JSON string into a JSON object in a mapping template.	For more information about `$input.body` and `$util.parseJson()`, see Mapping Template Built-in Functions and Variables (p. 153).	April 5, 2016
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. 293) and Invalidate an API Gateway Cache Entry (p. 293). Improve the console experience for managing API request throttling. For more information, see Throttle API Requests for Better Throughput (p. 289).	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 Custom Authorizers (p. 232).	February 11, 2016

Change	Description	Date Changed
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 an API (p. 210) and Export an API (p. 305).	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 Data Mapping (p. 142).	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 Set up Stage Variable for API Deployment (p. 295).	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. 228).	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. 243).	September 22, 2015
Mock integration of methods	Learn how to mock-integrate an API with Amazon API Gateway (p. 108). This feature enables developers to generate API responses from API Gateway directly without the need for a final integration backend beforehand.	September 1, 2015
Amazon Cognito Identity support	Amazon API Gateway has expanded the scope of the `$context` variable so that it now returns information about Amazon Cognito Identity when requests are signed with Amazon Cognito credentials. In addition, we have added a `$util` variable for escaping characters in JavaScript and encoding URLs and strings. For more information, see Mapping Template Built-in Functions and Variables (p. 153).	August 28, 2015
Swagger integration	Use the Swagger import tool on GitHub to import Swagger API definitions into Amazon API Gateway. Learn more about API Gateway Extensions to Swagger (p. 358) 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 `$input` parameter and its functions in the Mapping Template Built-in Functions and Variables (p. 153).	July 18, 2015
Initial public release	This is the initial public release of the *API Gateway Developer Guide*.	July 9, 2015

AWS Glossary

For the latest AWS terminology, see the AWS Glossary in the *AWS General Reference*.