

## **Part 02 - Amazon API Gateway**

# API Driven Development

An **API** is an interface.

An interface essentially defines how you can interact with a **resource**.

An interface has defined **actions**, defined **inputs**, and defined **outputs**.

OUTPUT



INPUT

The beauty of this interface is that it **abstracts** what is going on in the kitchen from the consumer.

Interfaces allow you flexibility when it comes to programming.

You can define how you want your clients or consumers to interact with your service.

You define the **actions**, **inputs**, and **outputs**.

From there, you can implement it however you want behind the scenes,  
and as long as you are fulfilling your end of the bargain and  
the outputs are what they should be given the input,  
no one using the interface needs to know or care about the implementation.

One way we use APIs is to **expose services** to one another.

If I have a program that is composed of five different microservices, they would all communicate via a technology-agnostic communication protocol.

Like using **RESTful HTTP-based calls**.

As long as the service can speak HTTP, it doesn't matter what each service is implemented in.



Maybe one service is best written in Python,  
but another service is best written in C++.

Doesn't matter because you are enforcing the use of interfaces through your APIs.

Note: When the API doesn't change, everything works nicely.

Once your APIs are defined and shared with clients,  
it's important that you do what you can to maintain **backwards compatibility** with changes.

This is why it is so important to make sure you are spending time designing your APIs up front and exploring what the actions, inputs, and outputs should be.

Following the practice of **designing the API first**  
is called **API-driven development**,  
where the first artifact created out of building a new service is the API.

Once the API is created, the front-end, or clients, using the API and the actual implementation of the API can be built in parallel.

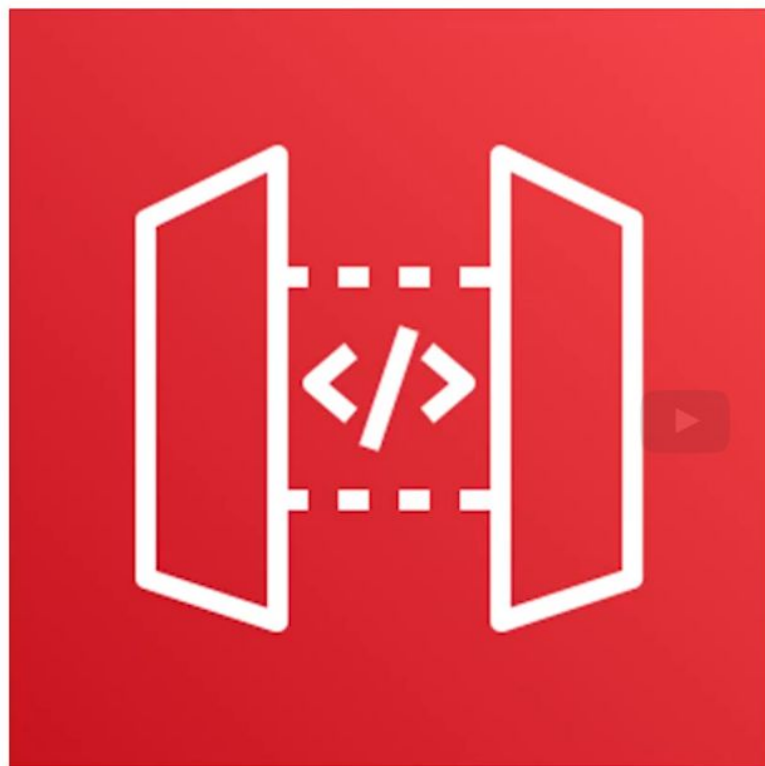
At the end, when both pieces are done, they should be able to communicate seamlessly as long as the back-end implementation adhere to the API that was first designed and it didn't get changed.

To wrap it up here,  
building APIs is like introducing a contract on  
how you allow others to interact with your code.

**What is API Gateway?**



Let's talk about the tools that help to manage and take advantage of APIs.



Amazon API Gateway

**API Gateway** is an AWS service  
for creating, publishing, maintaining, monitoring, and securing  
REST, HTTP, and WebSocket APIs at any scale.

App developers can create APIs that access  
AWS or other web services as well as data stored in the AWS Cloud.

While an API developer can gain the ability to create and deploy an API, enabling the necessary functionality in API Gateway, an app developer can build a functioning application to call AWS services by invoking the created APIs.

While we'll go more into detail on the use of API Gateway later,  
let's talk more about what the service provides.

This service can be used to create **HTTP**, **REST** and **WebSocket** APIs.

**WebSocket APIs** operate using lower-level protocols based on sockets and ports and requires the use of IP addresses and port information.

It is bidirectional, stateful, vertically scalable, and is ideally used for real-time scenarios such as a chat application as part of a game.

**REST** (Representational State Transfer) is an architectural style for designing networked applications that utilize the HTTP protocol.

Unlike WebSocket APIs which rely on lower-level protocols, REST operates over standard web protocols and is **stateless**.

It focuses on **resources**, which are uniquely identified by URLs, and uses standard HTTP methods (GET, POST, PUT, DELETE) to perform actions on these resources.

This makes it a robust choice for applications that primarily involve creating, retrieving, updating, and deleting data.

RESTful services are widely used for tasks like fetching web pages, accessing web APIs, and manipulating data in web and mobile applications.



**HTTP** operates on a request-response model.

A client sends an HTTP request to a server,  
which processes the request and sends back an HTTP response.

We focus on **REST APIs**,  
which are fast, stateless, standard, horizontally scalable and dependable.

## Other features provided by API Gateway:

- Canary deployments for safety and rolling out changes
- AWS CloudTrail integration for logging
- Amazon CloudWatch integration for monitoring
- Support for custom domain names to throttling of requests
- Direct integration with other AWS services including AWS WAF, AWS X-Ray and AWS Lambda.

A **method** represents a client-facing interface by which the client calls the API to access backend resources and refers to the particular HTTP verb used with a request.

Note: The most common HTTP verbs are:

- GET: Used to retrieve data from a server.
- POST: Used to send data to be processed by a server.
- PUT: Used to update a resource on the server.
- DELETE: Used to remove a resource from the server.
- And others like PATCH, OPTIONS, etc.

API Gateway provides multiple **endpoint types** that you can utilize.

An API endpoint type refers to the host name of the API.

It can be **edge** optimized, **regional** or **private** depending on where the majority of your API traffic originates.

Edge optimized API endpoints are best for geographically distributed clients.

Requests are routed to the nearest Amazon CloudFront point of presence.

This is the default type for API Gateway REST APIs.

A **regional API** endpoint is intended for clients in the same region.

When a client running an Amazon EC2 instance calls an API in the same region or when an API is intended to serve a small number of clients with high demands, a regional API endpoint reduces connection overhead.

**Private API** endpoints are a great way to provide a client secure access to resources inside of an Amazon virtual private cloud.

Private APIs are isolated from the public internet and they are only accessed using VPC endpoints for API Gateway that have been granted access.



# Dragon API: API Gateway Terminology

In this section, we're going to build the first method of our dragon API.

To start, we'll be going through the console  
and exploring some of the different pieces you can configure, as well as what they mean.

# AWS Management Console

## AWS services

### Find Services

You can enter names, keywords or acronyms.

Example: Relational Database Service, database, RDS

### Recently visited services



API Gateway



S3



CloudWatch



Cloud9



IAM

### All services

## Build a solution

Get started with simple wizards and automated workflows.

[Launch a virtual machine](#)

[Build a web app](#)

[Build using virtual servers](#)

## Stay connected to your AWS resources on-the-go



Download the AWS Console Mobile App to your iOS or Android mobile device. [Learn more](#)

## Explore AWS

### Amazon Redshift RA3 Nodes

Scale your compute and storage independently and lower your costs.

[Learn more](#)

### S3 Intelligent-Tiering

Optimize costs automatically with Amazon S3.

[Get started](#)

API Gateway

console.aws.amazon.com/apigateway/main/apis?region=us-east-1

Apps Onboarding Training AWS Sandbox Travel AWS Resources DynamoDB Containers Workshops Imported CD workshops GitHub - aws-sam...

aws Services Resource Groups buildingmodernapps @ 3022-1... N. Virginia Support

## API Gateway

Networking & Content Delivery

# Amazon API Gateway

## create, maintain, and secure APIs at any scale

Amazon API Gateway helps developers to create and manage APIs to back-end systems running on Amazon EC2, AWS Lambda, or any publicly addressable web service. With Amazon API Gateway, you can generate custom client SDKs for your APIs, to connect your back-end systems to mobile, web, and server applications or services.

### Choose an API type

**HTTP API**

Build low-latency and cost-effective REST APIs with built-in features such as OIDC and OAuth2, and native CORS support.

Works with the following:  
Lambda, HTTP backends

Import Build

You can build an HTTP API.

This is designed to offer REST-base HTTP API's at low latency and low cost.

HTTP API's are used to proxy backend resources and are supposed to be simple and fast.

This is great for when you want API Gateway to simply take in a request, authorize it and pass it on to the backend resource like a Lambda function or an HTTP endpoint.

### Choose an API type

**HTTP API**  
Build low-latency and cost-effective REST APIs with built-in features such as OIDC and OAuth2, and native CORS support.  
Works with the following:  
Lambda, HTTP backends  
[Import](#) [Build](#)

**WebSocket API**  
Build a WebSocket API using persistent connections for real-time use cases such as chat applications or dashboards.  
Works with the following:  
Lambda, HTTP, AWS Services  
[Build](#)

**REST API**

WebSockets unlike HTTP, is a stateful communications protocol.

WebSocket API's allow for support for applications that need real time data or real time communication.



**API Gateway** [Close]

- APIs
- Custom domain names
- VPC links

[Import] [Build]

### WebSocket API

Build a WebSocket API using persistent connections for real-time use cases such as chat applications or dashboards.

Works with the following:  
Lambda, HTTP, AWS Services

[Build]

### REST API

Develop a REST API where you gain complete control over the request and response along with API management capabilities.

Works with the following:  
Lambda, HTTP, AWS Services

[Import] [Build]

### REST API Private



REST API's with API Gateway uses HTTPS and is stateless.

This is very similar to HTTP API's, but it offers some different functionality.

REST API's allow you to have full control over the response and requests between your client and API Gateway.

You can apply what are called **models** and **mappings** to validate and transform requests and responses.

We will talk about this more in upcoming sections.



## API Gateway



APIs

Custom domain names

VPC links

Works with the following:  
Lambda, HTTP, AWS Services

[Build](#)

## REST API

Develop a REST API where you gain complete control over the request and response along with API management capabilities.

Works with the following:  
Lambda, HTTP, AWS Services

[Import](#)[Build](#)REST API Private

Create a REST API that is only accessible from within a VPC.

Works with the following:  
Lambda, HTTP, AWS Services

[Import](#)[Build](#)

Private API is the same thing as a normal REST API,  
but it allows to set up an API that can only be accessed from a VPC, creating a private API.

This is useful for internal APIs  
that you do not want to expose to outside clients.



## API Gateway



APIs

Custom domain names

VPC links

cases such as chat applications or dashboards.

Works with the following:  
Lambda, HTTP, AWS Services

[Build](#)

### REST API

Develop a REST API where you gain complete control over the request and response along with API management capabilities.

Works with the following:  
Lambda, HTTP, AWS Services

[Import](#)[Build](#)

### REST API Private

Create a REST API that is only accessible from within a VPC.

Works with the following:  
Lambda, HTTP, AWS Services

[Import](#)[Build](#)

We're going to select the public REST API,  
because we have some use cases that require  
we validate and transform incoming requests before they reach the backend.

### Create your first API

Welcome to Amazon API Gateway. To create your first API, we have pre-populated the import form with a Pet Store API defined using Swagger 2.0. To get started, close this modal and select **Import** in the Create API form.

**OK**

## Choose the protocol

Select whether you would like to create a REST API or a WebSocket

REST  WebSocket

## Create new API

In Amazon API Gateway, a REST API refers to a collection of resources

New API  Import from Swagger or Open API 3  Example API

## Example API

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

```
1 {
2   "swagger": "2.0",
3   "info": {
4     "description": "Your first API with Amazon API Gateway. This is a sample API that integrates via HTTP with our demo Pet Store endpoints",
5     "title": "PetStore"
6   },
7   "schemes": [
8     "https"
9   ],
10  "paths": {
11    "/": {
12      "get": {
13        "tags": [
14          "pets"
15        ],
16        "description": "PetStore HTML web page containing API usage information",
```

API Gateway x +

console.aws.amazon.com/apigateway/home?region=us-east-1#/apis/create

Apps Onboarding Training AWS Sandbox Travel AWS Resources DynamoDB Containers Workshops Imported CD workshops GitHub - aws-sam...

aws Services Resource Groups buildingmodernapps @ 3022-1... N. Virginia Supp

Amazon API Gateway APIs > Create Show all hints

## Choose the protocol

Select whether you would like to create a REST API or a WebSocket API.

REST  WebSocket

## Create new API

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

New API  Import from Swagger or Open API 3  Example API

## Settings

Choose a friendly name and description for your API.

API name*	<input type="text" value="My API"/>
Description	<input type="text"/>
Endpoint Type	<input type="text" value="Regional"/> ⓘ

\* Required



API Gateway x +

console.aws.amazon.com/apigateway/home?region=us-east-1#/apis/create

Apps Onboarding Training AWS Sandbox Travel AWS Resources DynamoDB Containers Workshops Imported CD workshops GitHub - aws-sam...

aws Services Resource Groups buildingmodernapps @ 3022-1... N. Virginia Support

Amazon API Gateway APIs > Create Show all hints ?

## Choose the protocol

Select whether you would like to create a REST API or a WebSocket API.

REST  WebSocket

## Create new API

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

New API  Import from Swagger or Open API 3  Example API

## Settings

Choose a friendly name and description for your API.

API name*	<input type="text" value="dragons"/>
Description	<input type="text"/>
Endpoint Type	<input type="text" value="Regional"/> ⓘ

\* Required

Create API



APIs

Custom Domain Names

VPC Links

API: dragons

Resources

Stages

Authorizers

Gateway Responses

Models

Resource Policy

Documentation

Settings

Usage Plans

API Keys

Resources

Actions

/Methods



No methods defined for the resource.

On the left hand side is the Navigation, for everything you can configure for an API.

We're going to talk about most of these things,  
but for now though, we're just going to keep it simple.

I want to create an API that allows me  
to submit a request and responds with a 200 OK  
but doesn't actually hit any real backend.

To do that, I need to first create a resource.

- APIs
- Custom Domain Names
- VPC Links
- API: dragons**
- Resources
- Stages
- Authorizers
- Gateway Responses
- Models
- Resource Policy
- Documentation
- Settings
- Usage Plans
- API Keys

Resources Actions /Methods

- RESOURCE ACTIONS
- Create Method
- Create Resource
- Enable CORS
- Edit Resource Documentation
- API ACTIONS
- Deploy API
- Import API
- Edit API Documentation
- Delete API

No methods defined for the resource.

A **resource** is an abstract concept  
that allows you to expose a “*thing*” to be consumed by a client.

For our example, we will be building out the dragons API.  
So the resource is the dragon data.



- APIs
- Custom Domain Names
- VPC Links
- API: dragons
  - Resources
  - Stages
  - Authorizers
  - Gateway Responses
  - Models
  - Resource Policy
  - Documentation
  - Settings
  - Usage Plans
  - API Keys

Resources **Actions** /

### New Child Resource

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

**Configure as**  proxy resource  **i**

**Resource Name\*** dragons

**Resource Path\*** / dragons

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**  **i**

\* Required

[Cancel](#) [Create Resource](#)

Enable CORS on the resource.

This will enable cross origin resource sharing for this API,  
and this will be used for future use.



Show all hid

APIs

Custom Domain Names

VPC Links

API: dragons

Resources

Stages

Authorizers

Gateway Responses

Models

Resource Policy

Documentation

Settings

Usage Plans

API Keys

Resources

Actions

## /dragons Methods

/

/dragons

OPTIONS

### OPTIONS

#### Mock Endpoint

Authorization None

API Key Not required

Resources in API's have **methods** that allow you to interact or submit actions to a resource.

In our dragon example, we will be using HTTP methods like GET and POST.

But you can use whatever HTTP methods you like in your own APIs.

Let's create a method.

- APIs
- Custom Domain Names
- VPC Links

API: dragons

Resources

- Stages
- Authorizers
- Gateway Responses
- Models
- Resource Policy
- Documentation
- Settings
- Usage Plans
- API Keys

Resources

Actions

/dragons Methods

RESOURCE ACTIONS

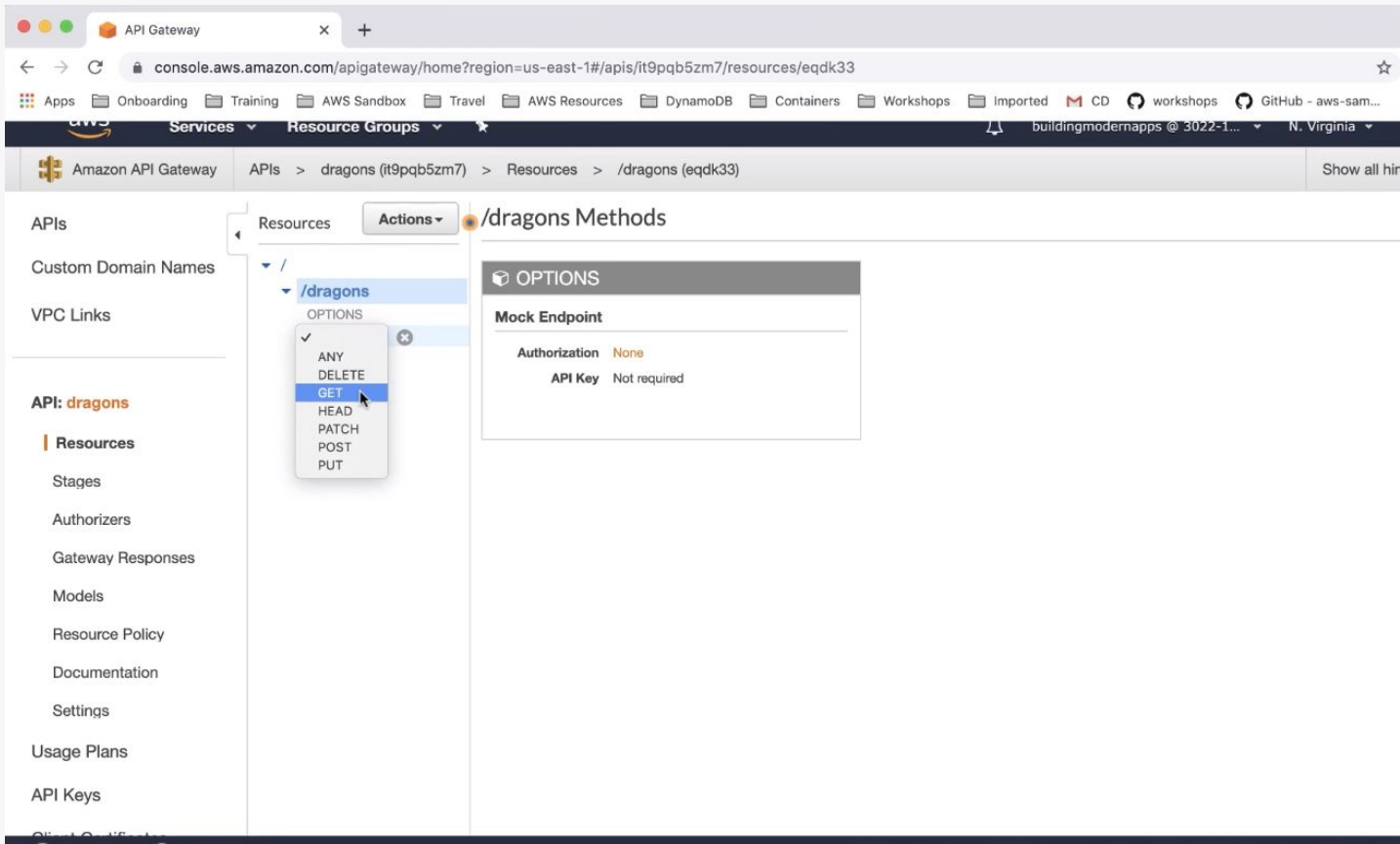
- Create Method
- Create Resource
- Enable CORS
- Edit Resource Documentation
- Delete Resource

API ACTIONS

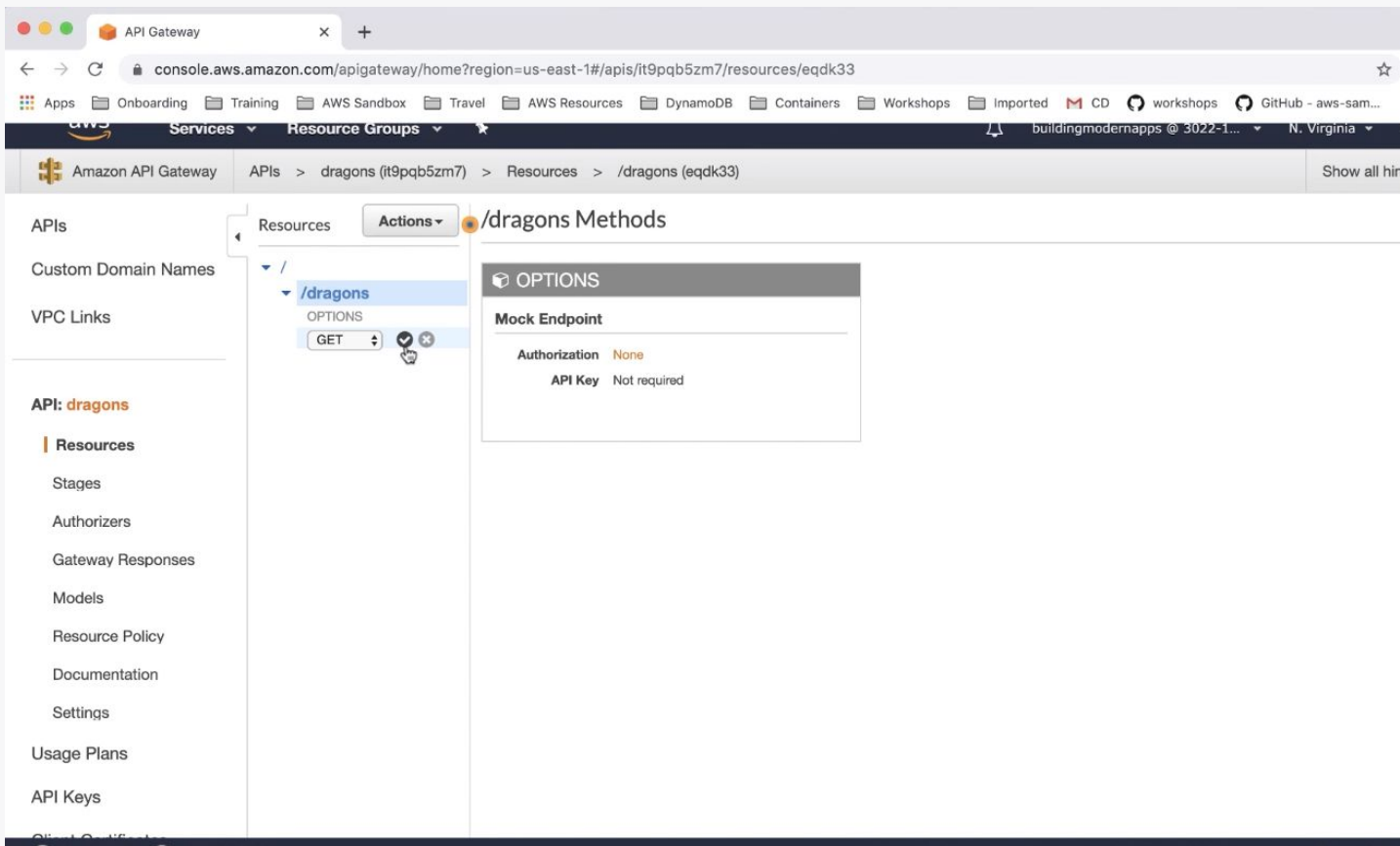
- Deploy API
- Import API
- Edit API Documentation
- Delete API

None

Not required



Select the GET method here



Click the check mark. This brings up a screen for us to configure the integration type.



The screenshot shows the AWS API Gateway console in the 'Resources' section for the API 'dragons (it9pqb5zm7)'. The breadcrumb trail is 'APIs > dragons (it9pqb5zm7) > Resources > /dragons (eqdk33) > GET'. The main heading is '/dragons - GET - Setup'. Below the heading, the instruction reads 'Choose the integration point for your new method.' The 'Integration type' is set to 'Lambda Function'. Below this, there are four radio button options: 'HTTP', 'Mock', 'AWS Service', and 'VPC Link'. The 'Use Lambda Proxy integration' checkbox is unchecked. The 'Lambda Region' is set to 'us-east-1'. The 'Lambda Function' field is empty. A yellow warning box states: 'You do not have any Lambda Functions in us-east-1. Create a Lambda Function in your current account, or provide an Lambda Function Cross-Account Access.' The 'Use Default Timeout' checkbox is checked.

This is where you select what you want your API to sit in front of, and how you want your API to integrate with that backend.

## You can select

- **Lambda function**, where API Gateway is purely a **proxy** for that Lambda function.
- **HTTP**, where you can paste an endpoint that already exists and you want API Gateway to front that.
- **Mock**, where you don't front anything real and instead, you just stub out the API.
- **AWS service**, where you can select another AWS service for API Gateway to front (which we will go into detail on this later).
- **VPC link**, which allows you to expose resources inside of a VPC.

We're following an API driven development process  
and want to design and build the API first.

So we will select Mock as the integration type.

- APIs
- Custom Domain Names
- VPC Links

- API: dragons**
- Resources
  - Stages
  - Authorizers
  - Gateway Responses
  - Models
  - Resource Policy
  - Documentation
  - Settings
  - Usage Plans
  - API Keys

Resources **Actions**

- /
- /dragons
  - GET**
  - OPTIONS

## /dragons - GET - Setup

Choose the integration point for your new method.

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

Save

The Mock integration type allows you to set up the API and interact with it without ever needing to actually hit another service for the backend.

We can hardcode the responses we expect and test it before building and integrating with a real backend.

The screenshot displays the AWS API Gateway console interface. The breadcrumb navigation shows the path: Amazon API Gateway > APIs > dragons (it9pqb5zm7) > Resources > /dragons (eqdk33) > GET. The main content area is titled "/dragons - GET - Method Execution" and features a "TEST" button with a lightning bolt icon. The flow is visualized as follows:

- Client**: A vertical bar on the left representing the client.
- Method Request**: A box containing:
  - Auth: NONE
  - ARN: arn:aws:execute-api:us-east-1:302211264422:it9pqb5zm7/\*/\*/GET/dragons
- Integration Request**: A box containing:
  - Type: MOCK
- Mock Endpoint**: A vertical bar on the right representing the mock endpoint.
- Method Response**: A box containing:
  - HTTP Status: 200
  - Models: application/json => Empty
- Integration Response**: A box containing:
  - HTTP status pattern: - (dropdown)
  - Output passthrough: Yes

Arrows indicate the flow: Client to Method Request, Method Request to Integration Request, Integration Request to Mock Endpoint, and Mock Endpoint to Integration Response, and Integration Response to Method Response. The Method Response then flows back to the Client.

Now we are brought to this screen, which shows the flow of the request and the response with API Gateway.

The screenshot displays the AWS API Gateway console interface. The breadcrumb navigation shows the path: Amazon API Gateway > APIs > dragons (it9pqb5zm7) > Resources > /dragons (eqdk33) > GET. The main content area is titled "/dragons - GET - Method Execution" and features a "TEST" button with a lightning bolt icon. The flow is visualized as follows:

- Client**: A vertical bar on the left representing the client.
- Method Request**: A box containing:
  - Auth: NONE
  - ARN: arn:aws:execute-api:us-east-1:302211264422:it9pqb5zm7/\*/\*/GET/dragThis box is highlighted with a blue and green border.
- Integration Request**: A box containing:
  - Type: MOCK
- Mock Endpoint**: A vertical bar on the right representing the mock endpoint.
- Method Response**: A box containing:
  - HTTP Status: 200
  - Models: application/json => Empty
- Integration Response**: A box containing:
  - HTTP status pattern: - (dropdown)
  - Output passthrough: Yes

Arrows indicate the flow from Client to Method Request, then to Integration Request, then to Mock Endpoint, and finally back to Method Response and Integration Response.

Method request is the first step for API Gateway to accept a request. This is where you can apply authorization and data payload validation.

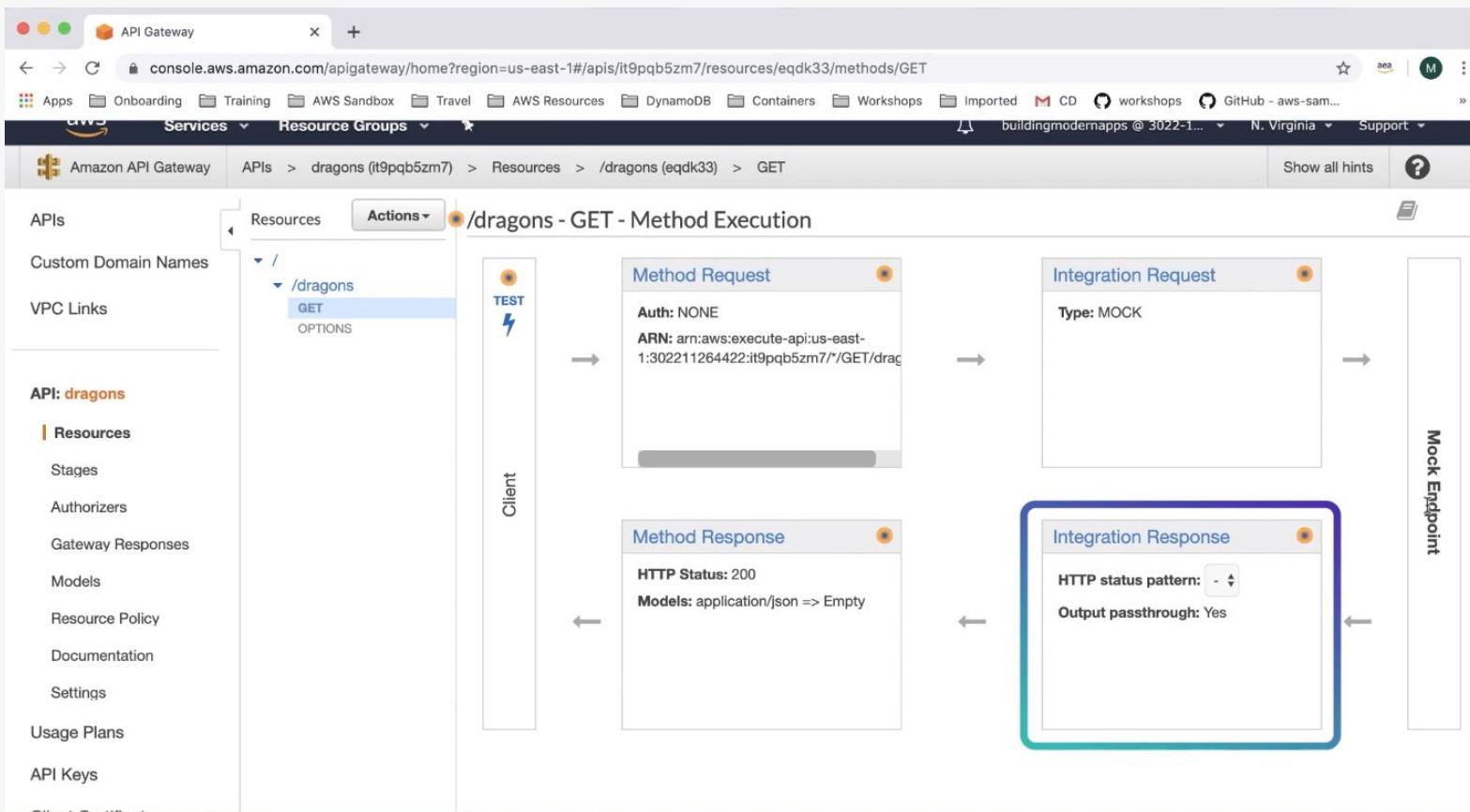
The screenshot displays the AWS API Gateway console interface. The breadcrumb navigation shows the path: Amazon API Gateway > APIs > dragons (it9pqb5zm7) > Resources > /dragons (eqdk33) > GET. The main content area is titled '/dragons - GET - Method Execution'. On the left, a sidebar lists navigation options such as APIs, Custom Domain Names, VPC Links, and API: dragons. The main area shows a flow diagram with four stages: 1. Client (indicated by a lightning bolt icon and 'TEST' label), 2. Method Request (showing 'Auth: NONE' and 'ARN: arn:aws:execute-api:us-east-1:302211264422:it9pqb5zm7/\*/GET/dragons'), 3. Integration Request (highlighted with a blue box, showing 'Type: MOCK'), and 4. Integration Response (showing 'HTTP Status: 200' and 'Models: application/json => Empty'). The flow concludes at a 'Mock Endpoint' on the right.

Next, it is passed to the integration request. This is where you configure what backend service you are fronting with API Gateway, as well as applying any sort of data transformations that you may require.



Once the request has passed the validation and the authorization and the transformation steps, it will then send the request to the backend service.

In our case, since this is a mock endpoint, it will take the request, do nothing and then send a response.



Integration response is an HTTP response encapsulating the backend response. You can configure how your backend service responses map to HTTP responses and apply data transformations at this step.

The screenshot shows the AWS API Gateway console interface. The breadcrumb navigation indicates the path: Amazon API Gateway > APIs > dragons (it9pqb5zm7) > Resources > /dragons (eqdk33) > GET. The main content area displays the "/dragons - GET - Method Execution" flow diagram. On the left, a sidebar lists navigation options such as APIs, Custom Domain Names, VPC Links, and API: dragons. The flow diagram consists of the following components:

- Client:** A vertical bar on the left with a "TEST" button and a lightning bolt icon.
- Method Request:** A box containing:
  - Auth: NONE
  - ARN: arn:aws:execute-api:us-east-1:302211264422:it9pqb5zm7/\*/\*/GET/dragons
- Integration Request:** A box containing:
  - Type: MOCK
- Integration Response:** A box containing:
  - HTTP status pattern: - (dropdown)
  - Output passthrough: Yes
- Method Response:** A box containing:
  - HTTP Status: 200
  - Models: application/json => Empty
- Mock Endpoint:** A vertical bar on the right.

Arrows indicate the flow from Client to Method Request, then to Integration Request, then to Integration Response, and finally to Method Response. A vertical bar labeled "Mock Endpoint" is positioned to the right of the Integration Response box.

Method responses are similar to method requests. They're responsible for validating and fitting responses to models.

The screenshot displays the AWS API Gateway console interface for a specific API method. The breadcrumb navigation shows the path: Amazon API Gateway > APIs > dragons (it9pqb5zm7) > Resources > /dragons (eqdk33) > GET. The main content area is titled "/dragons - GET - Method Execution".

On the left, a sidebar menu lists various API Gateway components, with "API: dragons" selected. Under "Resources", the "/dragons" resource is expanded, and the "GET" method is highlighted. A "TEST" button with a lightning bolt icon is visible next to the method name.

The central diagram illustrates the request and response flow:

- Client**: A vertical box on the left representing the client that initiates the request.
- Method Request**: A box containing details such as "Auth: NONE" and "ARN: arn:aws:execute-api:us-east-1:302211264422:it9pqb5zm7//GET/drag".
- Integration Request**: A box showing "Type: MOCK", indicating that the request is being simulated.
- Method Response**: A box showing "HTTP Status: 200" and "Models: application/json => Empty", representing the response returned to the client.
- Integration Response**: A box showing "HTTP status pattern" and "Output passthrough: Yes", representing the response from the integration.
- Mock Endpoint**: A vertical box on the right representing the endpoint that provides the mock response.

Arrows indicate the flow of data: from the Client to the Method Request, then to the Integration Request, then to the Mock Endpoint, and finally back from the Integration Response to the Method Response, which is then sent back to the Client.

We can actually go ahead and just click on the test.

API Gateway

console.aws.amazon.com/apigateway/home?region=us-east-1#/apis/it9pqb5zm7/resources/eqdk33/methods/GET

Apps Onboarding Training AWS Sandbox Travel AWS Resources DynamoDB Containers Workshops Imported CD workshops GitHub - aws-sam...

**API: dragons**

- Resources
  - Stages
  - Authorizers
  - Gateway Responses
  - Models
  - Resource Policy
  - Documentation
  - Settings
- Usage Plans
- API Keys
- Client Certificates
- Settings

**{myPathParam}** in a resource path.

Query Strings

**{dragons}**

Headers

**{dragons}**

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.

 Test

Leave all blank, and click test



## API: dragons

## Resources

- Stages
- Authorizers
- Gateway Responses
- Models
- Resource Policy
- Documentation
- Settings

## Usage Plans

## API Keys

## Client Certificates

## Settings

{myPathParam} in a resource path.

## Query Strings

## {dragons}

## Headers

## {dragons}

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.

## Response Body

## Response Headers

## Logs

```

Execution log for request f6968935-3c8f-40c7-815c-f59a75b57fde
Fri May 15 21:44:28 UTC 2020 : Starting execution for request: f6968935-3c8f-40c7-815c-f59a75b57fde
Fri May 15 21:44:28 UTC 2020 : HTTP Method: GET, Resource Path: /dragons
Fri May 15 21:44:28 UTC 2020 : Method request path: {}
Fri May 15 21:44:28 UTC 2020 : Method request query string: {}
Fri May 15 21:44:28 UTC 2020 : Method request headers: {}
Fri May 15 21:44:28 UTC 2020 : Method request body before transformations:
Fri May 15 21:44:28 UTC 2020 : Method response body after transformations:
Fri May 15 21:44:28 UTC 2020 : Method response headers: {Content-Type=application/json}
Fri May 15 21:44:28 UTC 2020 : Successfully completed execution
Fri May 15 21:44:28 UTC 2020 : Method completed with status: 200
  
```



Now, we have an API for our dragon resource with the GET method.  
Again, this is mocked up. It's not doing anything real yet in the background,  
but we are going to continue to Mock up our dragon API,  
and then we will back it with the Lambda functions later on in the course.

# Models and Mapping



API Gateway has a lot of great features that allow you to offload some of the burden that might otherwise be on your back end services.

For example, it's oftentimes the back end service that is checking the incoming payload for required fields, ensuring data is not null, or checking that the data is formatted in a particular way, in order for the code to run properly.

API Gateway REST APIs provide a way to validate incoming requests against **models**.

And data can be transformed in shape by using API Gateway **mappings**.

Let's start with models

**Models** in API Gateway define the structure or shape of the payload of the request.

Models are created using JSON schemas,  
which allow you to define the properties of the payload and their types.

When a request comes in, it will be validated against that schema.

And if API Gateway sees that the data cannot be fit into the JSON schema, then API Gateway will return a 400-error response code to the client.

This frees up your back end  
from having to do the sort of basic data validation.

Let's take a look at an example.

```
{  
  "dragonName": "Frank",  
  "description": "This dragon is brand  
                 new, we don't know  
                 much about it yet.",  
  "family": "purple",  
  "city": "Seattle",  
  "country": "USA",  
  "state": "WA",  
  "neighborhood": "Downtown",  
  "reportingPhoneNumber": "15555555555",  
  "confirmationRequired": false  
}
```

This is what a JSON payload for reporting a new dragon would look like.

```
{
```

```
"dragonName": "Frank",
```

```
"description": "This dragon is brand  
new, we don't know  
much about it yet.",
```

```
"family": "purple",
```

```
"city": "Seattle",
```

```
"country": "USA",
```

```
"state": "WA",
```

```
"neighborhood": "Downtown",
```

```
"reportingPhoneNumber": "1555555555",
```

```
"confirmationRequired": false
```

```
}
```

```
{  
  "$schema": "http://json-schema.org/draft-07/schema#",  
  "title": "Dragon",  
  "type": "object",  
  "properties": {  
    "dragonName": {  
      "type": "string"  
    },  
    "description": {  
      "type": "string"  
    },  
    "family": {  
      "type": "string"  
    },  
    "city": {  
      "type": "string"  
    },  
    "country": {  
      "type": "string"  
    },  
    "state": {  
      "type": "string"  
    },  
    "neighborhood": {  
      "type": "string"  
    },  
    "reportingPhoneNumber": {  
      "type": "string"  
    },  
    "confirmationRequired": {  
      "type": "boolean"  
    }  
  }  
}
```

And this is what the JSON schema for that payload would look like.

This schema is used to do request validation.



```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "title": "Person Contact Information",
  "type": "object",
  "properties": {
    "firstName": {
      "type": "string",
      "description": "The first name of the person."
    },
    "lastName": {
      "type": "string",
      "description": "The last name of the person."
    },
    "email": {
      "type": "string",
      "format": "email",
      "description": "The email address of the person."
    },
    "phone": {
      "type": "string",
      "pattern": "^\\d{3}-\\d{3}-\\d{4}$",
      "description": "The phone number of the person in the format XXX-XXX-XXXX."
    }
  },
  "required": ["firstName", "lastName", "email"]
}
```

A Contact Information JSON schema using the JSON Schema Draft 7 standard.

Models can be applied to both method requests and method responses.

Note that each method for resource like GET, POST, etc. could have different models.

So these get applied at the method level, not at the resource level.

Now, let's talk about **Mapping**.

It's fairly common to run into situations where your back end services are expecting incoming data in a different format than the client is sending, or vice versa.

Instead of having to change your code to support the clients data structure, wouldn't it be nice if API Gateway could handle that for you?

Luckily, it can. Mappings does this.

**Mappings** are applied to the integration request  
and integration response of your API.

Mappings are written in Velocity Template Language, or VTL.



If you already have defined a model for the method,  
API Gateway can generate a VTL blueprint for the mapping which you can then modify.

A mapping template assumes the data coming in as a JSON object by default, and mappings do support other data types like XML.

Mappings support conditional statements,  
can inject new parameters into the payload,  
can hardcode values, which is needed for mocking,  
map data in complex structures,  
and can even reference data made available at runtime,  
such as context and stage variables which we will cover later.

# Creating a GET API with Mock Integration

In the last section, we created a REST API for the dragon resource and setting it as a mock endpoint.

But, there isn't any data to provide a response.

In this section, we're going to continue that example by adding a method for response with a mock integration in API Gateway.

A mock integration enables your API to return a response for a request directly, without the need for a resource on the backend.

This is a way to develop the API independently from the other parts of your distributed application.

APIs

Custom Domain Names

VPC Links

API: dragons

Resources

Stages

Authorizers

Gateway Responses

Models

Resource Policy

Documentation

Dashboard

Settings

Usage Plans

API Keys

Client Certificates

Settings

Resources

Actions

/dragons - GET - Method Execution

/

/dragons

GET

OPTIONS

POST

TEST

Client

## Method Request

Auth: demo

ARN: arn:aws:execute-api:us-east-1:302211264422:nl18eym14c:/GET/dragons

## Method Response

HTTP Status: 200

Models: application/json =&gt; Empty

## Integration Request

Type: MOCK

## Integration Response

HTTP status pattern: -

Output passthrough: Yes

Having a mock endpoint return a blank response,  
isn't very helpful for testing.

So, we are going to be mocking what a real backend would do.



So now that we have an empty mock backend established,  
I want to set up the response for the integration.

AWS Services Resource Groups

Amazon API Gateway APIs > dragons (nl18eym14c) > Resources > /dragons (o5zw4d) > GET

APIs Resources Actions /dragons - GET - Method Execution

Custom Domain Names

VPC Links

API: dragons

- Resources
- Stages
- Authorizers
- Gateway Responses
- Models
- Resource Policy
- Documentation
- Dashboard
- Settings

Usage Plans

API Keys

Client Certificates

Settings

Client

Method Request

Auth: demo  
ARN: arn:aws:execute-api:us-east-1:302211264422:nl18eym14c/\*/GET/dragons

Integration Request

Type: MOCK

Method Response

HTTP Status: 200  
Models: application/json => Empty

Integration Response

HTTP status pattern: -  
Output passthrough: Yes

Go into the integration response

The screenshot shows the Amazon API Gateway console interface. The breadcrumb navigation at the top reads: Amazon API Gateway > APIs > dragons (nl18eym14c) > Resources > /dragons (o5zw4d) > GET. The left-hand navigation pane includes sections for APIs, Custom Domain Names, VPC Links, and API: dragons. Under the API: dragons section, Resources is selected, and the /dragons resource is expanded to show GET, OPTIONS, and POST methods. The GET method is selected, and the configuration page for the /dragons - GET - Integration Response is displayed. The page title is "Method Execution /dragons - GET - Integration Response".

Instructions: 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:  ⓘ

Content handling:  ⌵ ⓘ

▶ Header Mappings

▶ Mapping Templates

⊕ Add integration response

Expand the 200 method response status that we see already there.

The screenshot shows the AWS API Gateway console interface. The breadcrumb navigation at the top reads: Amazon API Gateway > APIs > dragons (nl18eym14c) > Resources > /dragons (o5zw4d) > GET. The left-hand navigation pane includes sections for APIs, Custom Domain Names, VPC Links, and API: dragons. Under the API: dragons section, Resources is selected, and the GET method is highlighted. The main content area is titled "/dragons - GET - Integration Response". It contains a table with the following data:

HTTP status regex	Method response status	Output model	Default mapping
-	200		Yes

Below the table, there is a section for configuring the response. It includes a text prompt: "Map the output from your HTTP endpoint to the headers and output model of the 200 method response." This is followed by two configuration fields: "HTTP status regex" with a dropdown menu set to "default", and "Content handling" with a dropdown menu set to "Passthrough". Below these are sections for "Header Mappings" and "Mapping Templates". Under "Mapping Templates", there is a "Content-Type" section with a dropdown menu showing "application/json" selected. At the bottom of the page, there are two blue links: "Add mapping template" and "Add integration response".

Select application/json that already exists under the content type.

aws Services Resource Groups Amazon API Gateway APIs > dragons (nl18eym14c) > Resources > /dragons (o5zw4d) > GET

APIs Resources Actions Method Execution /dragons - GET - Integration Response

Custom Domain Names / /dragons GET OPTIONS POST

API: dragons Resources Stages Authorizers Gateway Responses Models Resource Policy Documentation Dashboard Settings Usage Plans API Keys Client Certificates Settings

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

Content handling: Passthrough

Header Mappings

Mapping Templates

Content-Type: application/json

Generate template:

```
1- [{"
2     "description_str": "From the northern fire tribe, Atlas was born from
3     father in combat. He is fearless and does not fear battle.",
4     "dragon_name_str": "Atlas",
5     "family_str": "red",
6     "location_city_str": "anchorage",
7     "location_country_str": "usa",
8     "location_neighborhood_str": "w fireweed ln",
9     "location_state_str": "alaska"
10- },
11- {
12     "description_str": "Protheus is a wise and ancient dragon that serves
13     the sky world. He uses his power to calm those near him.",
14     "dragon_name_str": "Protheus",
15     "family_str": "blue",
16     "location_city_str": "brandon",
17     "location_country_str": "usa",
18     "location_neighborhood_str": "e morgan st",
19     "location state str": "florida"
```

Put your hard-coded response. The data is static, but it gives the appearance of a working API.

aws Services Resource Groups Amazon API Gateway APIs > dragons (nl18eym14c) > Resources > /dragons (o5zw4d) > GET

APIs Resources Actions Method Execution /dragons - GET - Method Test

Custom Domain Names VPC Links

API: dragons Resources Stages Authorizers Gateway Responses Models Resource Policy Documentation Dashboard Settings Usage Plans API Keys Client Certificates Settings

Resources

- /
- /dragons
  - GET
  - OPTIONS
  - POST

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**

**{dragons}**

param1=value1&param2=value2

**Headers**

**{dragons}**

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.

Request: /dragons  
Status: 200  
Latency: 26 ms  
Response Body

```
[
  {
    "description_str": "From the northern fire tribe, Atlas was born fi
fallen father in combat. He is fearless and does not fear battle.",
    "dragon_name_str": "Atlas",
    "family_str": "red",
    "location_city_str": "anchorage",
    "location_country_str": "usa",
    "location_neighborhood_str": "w fireweed ln",
    "location_state_str": "alaska"
  },
  {
    "description_str": "Protheus is a wise and ancient dragon that ser
cil in the sky world. He uses his power to calm those near him.",
    "dragon_name_str": "Protheus",
    "family_str": "blue",
    "location_city_str": "brandon",
    "location_country_str": "usa",
    "location_neighborhood_str": "e morgan st",
    "location_state_str": "florida"
  },
  {
    "description_str": "Xanya is the fire tribe's banished general. She
been wandering ever since.",
    "dragon_name_str": "Xanya",
    "family_str": "red",
    "location_city_str": "las vegas",
    "location_country_str": "usa",
    "location_neighborhood_str": "e clark ave",
    "location_state_str": "nevada"
  }
]
```

Test

And after clicking test, we see the hard-coded data is coming back in the side pane.

What you'll see next is the addition of the functionality for the query parameters, that enable querying the data by dragon family and dragon name.

# Dragon API: Using Mappings



Now that you have an API created for your dragon data and a basic GET method that returns some hard coded data what we need to do now is ensure that our mocked API can account for the fact that when we submit a GET request to our API, we could be trying to list all dragons, list dragons by family, or list dragons by name.

One method will be handling all three of these use cases.

When you submit a request,  
you will include a query parameter on the request, like this:

```
/dragon?dragonName=Atlas
```

```
/dragon?family=red
```

We must modify the mock endpoint to respond with different data if one of these query parameters is present.

To make that happen, what I need to take advantage of is the mappings that were applied to the integration response.

I'm going to modify the integration response to check the query parameters using conditionals in VTL.

The screenshot shows the AWS API Gateway console interface. The breadcrumb navigation indicates the path: Amazon API Gateway > APIs > dragons (it9pqb5zm7) > Resources > /dragons (eqdk33) > GET. The main content area displays the "/dragons - GET - Method Execution" view. On the left, a sidebar lists navigation options: APIs, Custom Domain Names, VPC Links, API: dragons, Resources (selected), Stages, Authorizers, Gateway Responses, Models, Resource Policy, Documentation, Settings, Usage Plans, and API Keys. The main area features a flow diagram with four boxes: "Method Request" (Auth: NONE, ARN: arn:aws:execute-api:us-east-1:302211264422:it9pqb5zm7/\*/\*/GET/dragons), "Integration Request" (Type: MOCK), "Method Response" (HTTP Status: 200, Models: application/json => Empty), and "Integration Response" (HTTP status pattern: -, Output passthrough: Yes). A vertical bar on the left is labeled "Client" and a vertical bar on the right is labeled "Mock Endpoint". Arrows indicate the flow from Client to Method Request, then to Integration Request, then to Integration Response, and finally to Method Response.

Click on the Integration Response.

The screenshot shows the AWS API Gateway console interface. On the left is a navigation sidebar with categories like Resources, Usage Plans, and API Keys. The main area displays the configuration for a specific method. At the top, 'HTTP status regex' is set to 'default' and 'Content handling' is set to 'Passthrough'. Below this, there are sections for 'Header Mappings' and 'Mapping Templates'. In the 'Mapping Templates' section, a table lists the available content types, with 'application/json' selected. A 'Generate template' dropdown is set to 'application/json', and a text area below it shows the JSON response body for this template. The JSON is a list of two objects, each representing a dragon with various attributes like description, name, family, and location.

Resources

- Stages
- Authorizers
- Gateway Responses
- Models
- Resource Policy
- Documentation
- Settings
- Usage Plans
- API Keys
- Client Certificates
- Settings

HTTP status regex: default

Content handling: Passthrough

Cancel Save

Header Mappings

Mapping Templates

Content-Type
application/json

+ Add mapping template

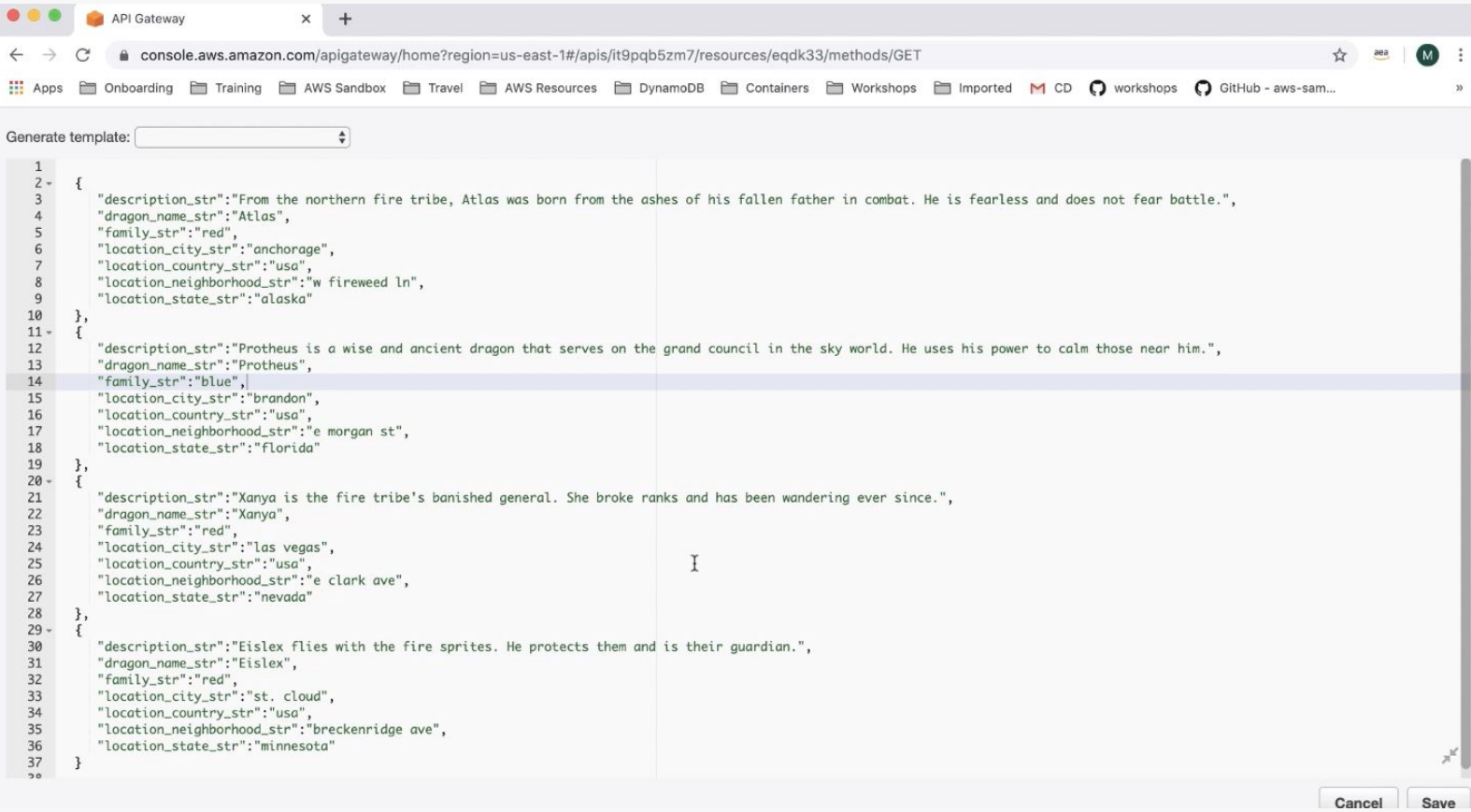
application/json

Generate template:

```
1 {
2   "description_str": "From the northern fire tribe, Atlas was
3     born from the ashes of his fallen father in combat. He
4     is fearless and does not fear battle.",
5   "dragon_name_str": "Atlas",
6   "family_str": "red",
7   "location_city_str": "anchorage",
8   "location_country_str": "usa",
9   "location_neighborhood_str": "w fireweed ln",
10  "location_state_str": "alaska"
11 },
12 {
13   "description_str": "Protheus is a wise and ancient dragon
14     that serves on the grand council in the sky world. He
15     uses his power to calm those near him.",
16   "dragon_name_str": "Protheus",
17   "family_str": "blue",
18   "location city str": "brandon".
19 }
```

Cancel Save

Then navigate to the existing mapping, which is under Mapping Templates, click on application/json.



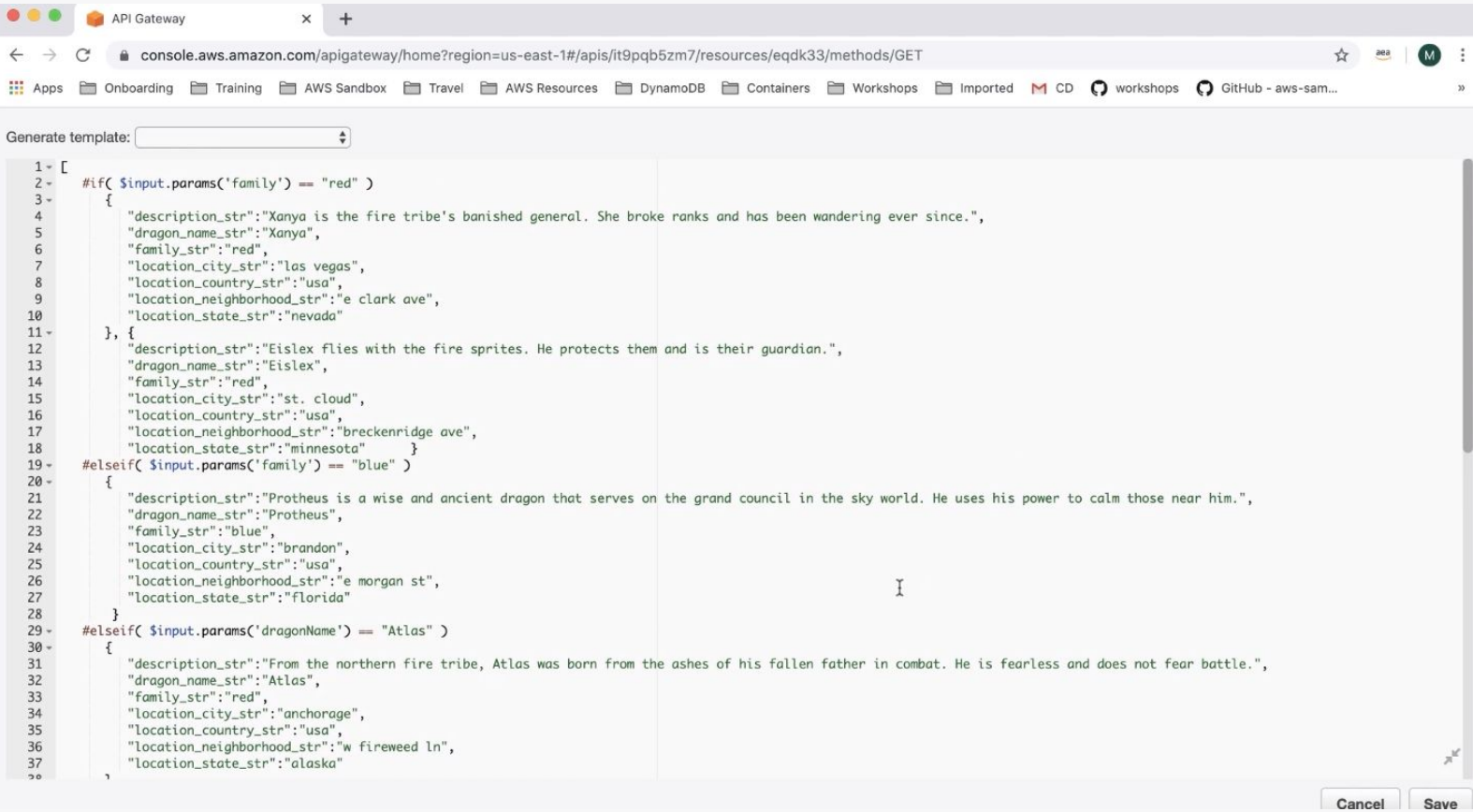
```
1 {
2   {
3     "description_str": "From the northern fire tribe, Atlas was born from the ashes of his fallen father in combat. He is fearless and does not fear battle.",
4     "dragon_name_str": "Atlas",
5     "family_str": "red",
6     "location_city_str": "anchorage",
7     "location_country_str": "usa",
8     "location_neighborhood_str": "w fireweed ln",
9     "location_state_str": "alaska"
10  },
11  {
12    "description_str": "Protheus is a wise and ancient dragon that serves on the grand council in the sky world. He uses his power to calm those near him.",
13    "dragon_name_str": "Protheus",
14    "family_str": "blue",
15    "location_city_str": "brandon",
16    "location_country_str": "usa",
17    "location_neighborhood_str": "e morgan st",
18    "location_state_str": "florida"
19  },
20  {
21    "description_str": "Xanya is the fire tribe's banished general. She broke ranks and has been wandering ever since.",
22    "dragon_name_str": "Xanya",
23    "family_str": "red",
24    "location_city_str": "las vegas",
25    "location_country_str": "usa",
26    "location_neighborhood_str": "e clark ave",
27    "location_state_str": "nevada"
28  },
29  {
30    "description_str": "Eislex flies with the fire sprites. He protects them and is their guardian.",
31    "dragon_name_str": "Eislex",
32    "family_str": "red",
33    "location_city_str": "st. cloud",
34    "location_country_str": "usa",
35    "location_neighborhood_str": "breckenridge ave",
36    "location_state_str": "minnesota"
37  }
38 }
```

This is the existing mapping we have right now.

Mappings are written in VTL.

This means we can use conditionals in the mapping.





The screenshot shows the AWS API Gateway console interface. At the top, the browser address bar displays the URL: `console.aws.amazon.com/apigateway/home?region=us-east-1#/apis/it9pqb5zm7/resources/eqdk33/methods/GET`. Below the address bar, there is a navigation menu with various folders like 'Apps', 'Onboarding', 'Training', 'AWS Sandbox', etc. The main content area is titled 'Generate template:' and contains a code editor with the following JSON template:

```
1 - [
2 -   #if( $input.params('family') == "red" )
3 -   {
4 -     "description_str": "Xanya is the fire tribe's banished general. She broke ranks and has been wandering ever since.",
5 -     "dragon_name_str": "Xanya",
6 -     "family_str": "red",
7 -     "location_city_str": "las vegas",
8 -     "location_country_str": "usa",
9 -     "location_neighborhood_str": "e clark ave",
10 -    "location_state_str": "nevada"
11 -   }, {
12 -     "description_str": "Eislex flies with the fire sprites. He protects them and is their guardian.",
13 -     "dragon_name_str": "Eislex",
14 -     "family_str": "red",
15 -     "location_city_str": "st. cloud",
16 -     "location_country_str": "usa",
17 -     "location_neighborhood_str": "breckenridge ave",
18 -     "location_state_str": "minnesota"
19 -   }
20 -   #elseif( $input.params('family') == "blue" )
21 -   {
22 -     "description_str": "Protheus is a wise and ancient dragon that serves on the grand council in the sky world. He uses his power to calm those near him.",
23 -     "dragon_name_str": "Protheus",
24 -     "family_str": "blue",
25 -     "location_city_str": "brandon",
26 -     "location_country_str": "usa",
27 -     "location_neighborhood_str": "e morgan st",
28 -     "location_state_str": "florida"
29 -   }
30 -   #elseif( $input.params('dragonName') == "Atlas" )
31 -   {
32 -     "description_str": "From the northern fire tribe, Atlas was born from the ashes of his fallen father in combat. He is fearless and does not fear battle.",
33 -     "dragon_name_str": "Atlas",
34 -     "family_str": "red",
35 -     "location_city_str": "anchorage",
36 -     "location_country_str": "usa",
37 -     "location_neighborhood_str": "w fireweed ln",
38 -     "location_state_str": "alaska"
39 -   }
40 - ]
```

At the bottom right of the code editor, there are two buttons: 'Cancel' and 'Save'.

We added some conditionals and it is checking against this `$input.params`.

So what is this `$input`?

API Gateway provides variables that start with a \$ sign that give you access to payload and context information in your mappings.

dragon is the `$input` here:

`/dragon?dragonName=Atlas`

`/dragon?family=red`

Generate template: 

```
1 [
2 #if( $input.params('family') == "red" )
3 {
4     "description_str": "Xanya is the fire tribe's banished general. She broke ranks and has been wandering ever since.",
5     "dragon_name_str": "Xanya",
6     "family_str": "red",
7     "location_city_str": "las vegas",
8     "location_country_str": "usa",
9     "location_neighborhood_str": "e clark ave",
10    "location_state_str": "nevada"
11 }, {
12     "description_str": "Eislex flies with the fire sprites. He protects them and is their guardian.",
13     "dragon_name_str": "Eislex",
14     "family_str": "red",
15     "location_city_str": "st. cloud",
16     "location_country_str": "usa",
17     "location_neighborhood_str": "breckenridge ave",
18     "location_state_str": "minnesota" }
19 #elseif( $input.params('family') == "blue" )
20 {
21     "description_str": "Protheus is a wise and ancient dragon that serves on the grand council in the sky world. He uses his power to calm those near him.",
22     "dragon_name_str": "Protheus",
23     "family_str": "blue",
24     "location_city_str": "brandon",
25     "location_country_str": "usa",
26     "location_neighborhood_str": "e morgan st",
27     "location_state_str": "florida"
28 }
29 #elseif( $input.params('dragonName') == "Atlas" )
30 {
31     "description_str": "From the northern fire tribe, Atlas was born from the ashes of his fallen father in combat. He is fearless and does not fear battle.",
32     "dragon_name_str": "Atlas",
33     "family_str": "red",
34     "location_city_str": "anchorage",
35     "location_country_str": "usa",
36     "location_neighborhood_str": "w fireweed ln",
37     "location_state_str": "alaska"
38 }
```

Cancel

Save

The screenshot shows the AWS API Gateway console interface. The browser address bar displays the URL: `console.aws.amazon.com/apigateway/home?region=us-east-1#/apis/it9pqb5zm7/resources/eqdk33/methods/GET`. The left-hand navigation pane includes sections for Custom Domain Names, VPC Links, and API: dragons. Under the API: dragons section, Resources is selected, and the breadcrumb path is `/ /dragons GET OPTIONS`. The main content area is titled "Make a test call to your method with the provided input" and contains the following sections:

- Path:** No path parameters exist for this resource. You can define path parameters by using the syntax `{myPathParam}` in a resource path.
- Query Strings:** A text input field contains `{dragons} dragonName=Atlas`.
- Headers:** A text input field contains `{dragons} Accept:application/json`. A tooltip is visible over the field with the text: "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:** (partially visible at the bottom)

add in some query parameters to the test

API Gateway console showing the configuration for the **dragons** API resource. The method is GET, and the response body is a JSON object representing a dragon.

**API: dragons**

- Resources
- Stages
- Authorizers
- Gateway Responses
- Models
- Resource Policy
- Documentation
- Settings
- Usage Plans
- API Keys
- Client Certificates
- Settings

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

**Query Strings**

**{dragons}**

dragonName=Atlas

**Headers**

**{dragons}**

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.

**Status: 200**

**Latency: 41 ms**

**Response Body**

```
{
  "description_str": "From the northern fire tribe, Atlas was born from the ashes of his fallen father in combat. He is fearless and does not fear battle.",
  "dragon_name_str": "Atlas",
  "family_str": "red",
  "location_city_str": "anchorage",
  "location_country_str": "usa",
  "location_neighborhood_str": "w fireweed ln",
  "location_state_str": "alaska"
}
```

**Response Headers**

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

**Logs**

```
Execution log for request 886f714a-be7b-4c22-822f-9753d859e3bc
Fri May 15 21:58:29 UTC 2020 : Starting execution for request: 886f714a-be7b-4c22-822f-9753d859e3bc
Fri May 15 21:58:29 UTC 2020 : HTTP Method: GET, Resource Path: /dragons
Fri May 15 21:58:29 UTC 2020 : Method request path: {}
Fri May 15 21:58:29 UTC 2020 : Method request query string:
```

we got back just the Atlas dragon.

API Gateway console showing details for the resource `family=red`. The resource is a GET method with a status of 200 and a latency of 40 ms. The response body is a JSON array of dragon objects.

**API: dragons**

- Resources
  - Stages
  - Authorizers
  - Gateway Responses
  - Models
  - Resource Policy
  - Documentation
  - Settings
- Usage Plans
- API Keys
- Client Certificates
- Settings

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

**Query Strings**

**{dragons}**

family=red

**Headers**

**{dragons}**

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.

**Status: 200**

**Latency: 40 ms**

**Response Body**

```
[
  {
    "description_str": "Xanya is the fire tribe's banished general. She broke ranks and has been wandering ever since.",
    "dragon_name_str": "Xanya",
    "family_str": "red",
    "location_city_str": "las vegas",
    "location_country_str": "usa",
    "location_neighborhood_str": "e clark ave",
    "location_state_str": "nevada"
  },
  {
    "description_str": "Eislex flies with the fire sprite s. He protects them and is their guardian.",
    "dragon_name_str": "Eislex",
    "family_str": "red",
    "location_city_str": "st. cloud",
    "location_country_str": "usa",
    "location_neighborhood_str": "breckenridge ave",
    "location_state_str": "minnesota"
  }
]
```

**Response Headers**

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

family=red

This API is responding as if there is a backend  
but there isn't.

This method is now fully mocked up and we can move on to the next method.

# DragonAPI: Using Models



Let's continue to build out our dragon API  
and move on to adding the method that will handle the reporting a new dragon.

First, add a new POST method to the dragons resource.



Services

Resource Groups



buildingmodernapps @ 3022-1...

N. Virginia

Support



Amazon API Gateway

APIs > dragons (it9pqb5zm7) > Resources > /dragons (eqdk33)

Show all hints



APIs

Custom Domain Names

VPC Links

API: dragons

Resources

Stages

Authorizers

Gateway Responses

Models

Resource Policy

Documentation

Settings

Usage Plans

API Keys

Resources

Actions

/dragons Methods

RESOURCE ACTIONS

- Create Method
- Create Resource
- Enable CORS
- Edit Resource Documentation
- Delete Resource

API ACTIONS

- Deploy API
- Import API
- Edit API Documentation
- Delete API

### OPTIONS

#### Mock Endpoint

Authorization None  
API Key Not required

None  
Not required



Services

Resource Groups



buildingmodernapps @ 3022-1...

N. Virginia

Support



Amazon API Gateway

APIs > dragons (it9pqb5zm7) > Resources > /dragons (eqdk33)

Show all hints



APIs

Custom Domain Names

VPC Links

API: dragons

Resources

Stages

Authorizers

Gateway Responses

Models

Resource Policy

Documentation

Settings

Usage Plans

API Keys

Resources

Actions

### /dragons Methods

/

/dragons

GET

OPTIONS

POST



#### GET

##### Mock Endpoint

Authorization None

API Key Not required

#### OPTIONS

##### Mock Endpoint

Authorization None

API Key Not required



Services

Resource Groups



buildingmodernapps @ 3022-1...

N. Virginia

Support

Amazon API Gateway APis > dragons (it9pqb5zm7) > Resources > /dragons (eqdk33) > POST

Show all hints



APIs

Custom Domain Names

VPC Links

API: dragons

Resources

Stages

Authorizers

Gateway Responses

Models

Resource Policy

Documentation

Settings

Usage Plans

API Keys

Resources

Actions

### /dragons - POST - Setup

- /
- /dragons
  - GET
  - OPTIONS
  - POST

Choose the integration point for your new method.

- Integration type
- Lambda Function
  - HTTP
  - Mock
  - AWS Service
  - VPC Link

Save



Services

Resource Groups



buildingmodernapps @ 3022-1...

N. Virginia

Support

Amazon API Gateway APis > dragons (it9pqb5zm7) > Resources > /dragons (eqdk33) > POST

Show all hints



APIs Resources Actions /dragons - POST - Method Execution

Custom Domain Names

VPC Links

API: dragons

Resources

Stages

Authorizers

Gateway Responses

Models

Resource Policy

Documentation

Settings

Usage Plans

API Keys

Resources

Actions

/dragons

GET

OPTIONS

POST



Client

Method Request

Auth: NONE

ARN: arn:aws:execute-api:us-east-1:302211264422:it9pqb5zm7/\*/POST/dra

Method Response

HTTP Status: 200

Models: application/json => Empty

Integration Request

Type: MOCK

Integration Response

HTTP status pattern: -

Output passthrough: Yes

Mock Endpoint

This method is going to be used for reporting new dragons  
so there will be a payload on the request that has the new dragons information.

One of the features API Gateway has  
is to validate incoming requests based on models.

Let's use that feature with this request.



console.aws.amazon.com/apigateway/home?region=us-east-1#/apis/it9pqb5zm7/resources/eqdk33/methods/POST

Apps Onboarding Training AWS Sandbox Travel AWS Resources DynamoDB Containers Workshops Imported CD workshops GitHub - aws-sam...

APIs Resources Actions /dragons - POST - Method Execution

Custom Domain Names

VPC Links

API: dragons

Resources

- Stages
- Authorizers
- Gateway Responses
- Models
- Resource Policy
- Documentation
- Settings

Usage Plans

API Keys

Client Certificates

Settings

Client

Method Request

Auth: NONE

ARN: arn:aws:execute-api:us-east-1:302211264422:it9pqb5zm7/POST/dra

Integration Request

Type: MOCK

Mock Endpoint

Method Response

HTTP Status: 200

Models: application/json => Empty

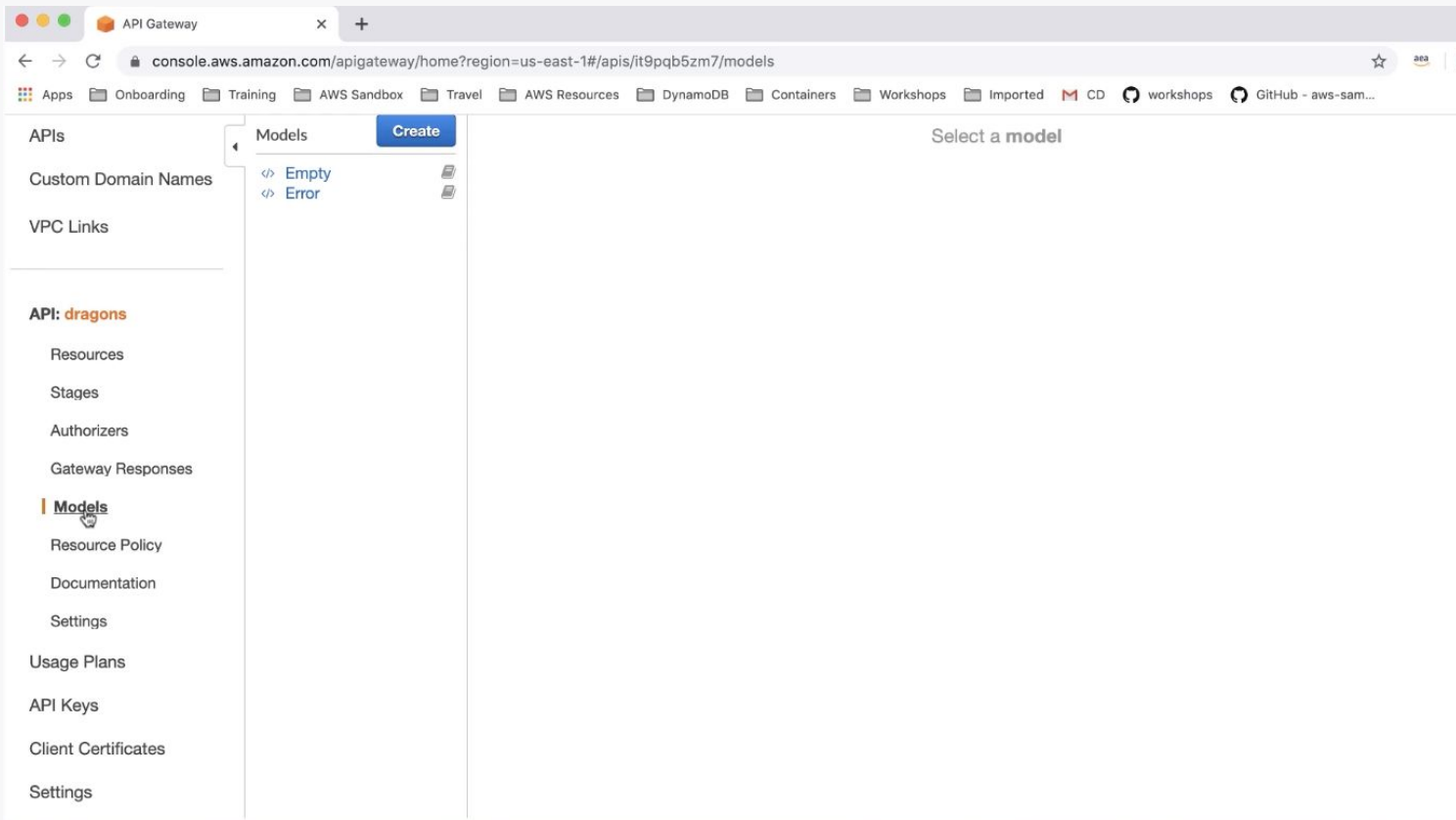
Integration Response

HTTP status pattern: -

Output passthrough: Yes

© 2008 - 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy Policy Terms of Use

We first need to create a model. Click on the models section in the left-hand side.



Click on Create

The screenshot shows the AWS API Gateway console interface. The browser address bar displays the URL: `console.aws.amazon.com/apigateway/home?region=us-east-1#/apis/it9pqb5zm7/models/create`. The left-hand navigation pane includes sections for 'APIs', 'Custom Domain Names', 'VPC Links', and 'API: dragons'. Under 'API: dragons', the 'Models' option is highlighted. The main content area is titled 'New Model' and contains the following elements:

- A heading: 'New Model'
- Instructional text: 'Provide a name, content type, and a schema for your model. Models use [JSON schema](#).'
- Form fields:
  - 'Model name\*': A text input field containing 'My Model'.
  - 'Content type\*': A text input field containing 'eg. application/json'.
  - 'Model description': An empty text input field.
- 'Model schema\*': A large text area for defining the JSON schema, currently empty with a cursor.
- Footer: A '\* Required' label and two buttons: 'Cancel' and 'Create n'.

Models in API Gateway are JSON schemas.

API Gateway console showing the 'New Model' page. The page includes a sidebar with navigation options (APIs, Custom Domain Names, VPC Links, API: dragons, Resources, Stages, Authorizers, Gateway Responses, Models, Resource Policy, Documentation, Settings, Usage Plans, API Keys, Client Certificates) and a main content area for creating a new model. The 'New Model' section contains a 'Create' button, a 'Model name\*' field (My Model), a 'Content type\*' field (eg. application/json), and a 'Model description' field. Below these is a 'Model schema\*' section with a text area containing a JSON schema for a 'Dragon' object. The schema includes properties for 'dragonName', 'description', 'family', 'city', and 'country'. At the bottom right, there are 'Cancel' and 'Create' buttons.

```
1 - {
2   "$schema": "http://json-schema.org/draft-07/schema#",
3   "title": "Dragon",
4   "type": "object",
5   "properties": {
6     "dragonName": {
7       "type": "string"
8     },
9     "description": {
10      "type": "string"
11    },
12    "family": {
13      "type": "string"
14    },
15    "city": {
16      "type": "string"
17    },
18    "country": {
19      "type": "string"
20    }
21  }
22 }
```

I paste in a pre-written JSON schema for the incoming dragon data that we expect to come in for the POST request.

API Gateway console page showing the "New Model" creation process. The browser address bar shows the URL: `console.aws.amazon.com/apigateway/home?region=us-east-1#/apis/it9pqb5zm7/models/create`.

The left sidebar shows the navigation menu with "APIs" selected. Under "APIs", the "API: dragons" is highlighted. The "Models" section is active, showing a "Create" button and a list of models: "Empty" and "Error".

The main content area is titled "New Model" and contains the following fields:

- Model name\***:
- Content type\***:
- Model description**:

The "Model schema\*" section contains a JSON schema definition:

```
1- {
2  "$schema": "http://json-schema.org/draft-07/schema#",
3  "title": "Dragon",
4  "type": "object",
5  "properties": {
6    "dragonName": {
7      "type": "string"
8    },
9    "description": {
10     "type": "string"
11   },
12   "family": {
13     "type": "string"
14   },
15   "city": {
16     "type": "string"
17   },
18   "country": {
19     "type": "string"
20   }
21 }
```

At the bottom right, there are "Cancel" and "Create model" buttons.

Name this dragon, give it the content type application/JSON, and then click create model.

- APIs
- Custom Domain Names
- VPC Links
- API: dragons
  - Resources
  - Stages
  - Authorizers
  - Gateway Responses
  - Models**
  - Resource Policy
  - Documentation
  - Settings
- Usage Plans
- API Keys
- Client Certificates
- Settings

- Models
  - Create
  - dragon
  - Empty
  - Error

## Update Model

Delete Model

Make changes to your model in the form below. Models are declared using [JSON schema](#).

Model name dragon

Content type application/json

Model description

### Model schema\*

```
1 - {
2   "$schema": "http://json-schema.org/draft-07/schema#",
3   "title": "Dragon",
4   "type": "object",
5   "properties": {
6     "dragonName": {
7       "type": "string"
8     },
9     "description": {
10      "type": "string"
11    },
12    "family": {
13      "type": "string"
14    },
15    "city": {
16      "type": "string"
17    },
18    "country": {
19      "type": "string"

```

Based on this model,  
if the input deviates from the structure that we created,  
I want API Gateway to reject the request before it ever hits my backend.

The screenshot displays the AWS API Gateway console interface. The browser address bar shows the URL: `console.aws.amazon.com/apigateway/home?region=us-east-1#/apis/it9pqb5zm7/resources/eqdk33/methods/POST`. The left-hand navigation pane includes sections for "APIs", "Custom Domain Names", "VPC Links", and "API: dragons". Under "API: dragons", the "Resources" section is expanded, showing a tree structure with `/dragons` containing methods `GET`, `OPTIONS`, and `POST`. The `POST` method is selected and highlighted.

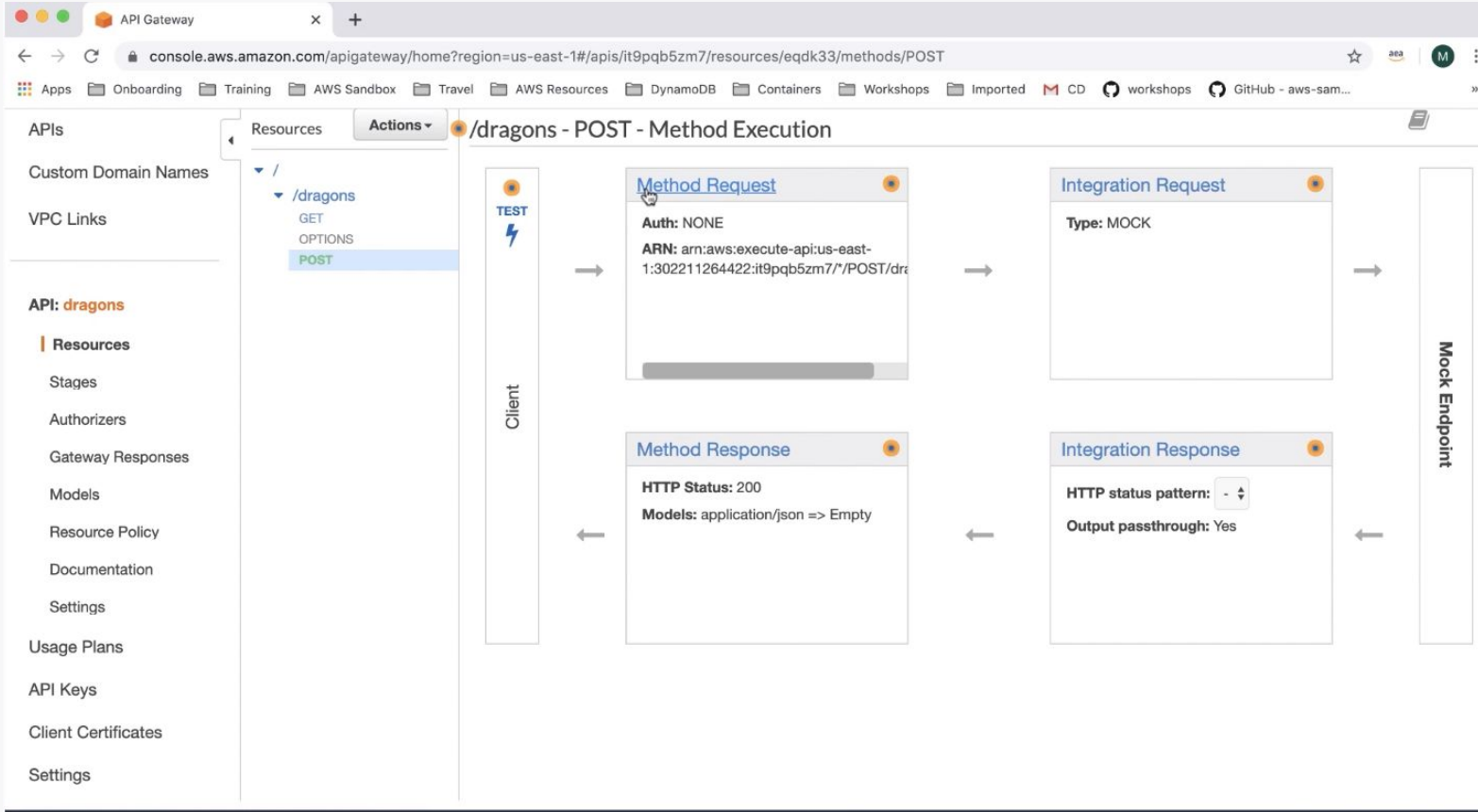
The main content area is titled `/dragons - POST - Method Execution`. It features a flow diagram illustrating the request and response process:

- Client**: A vertical bar on the left representing the client.
- Method Request**: A box containing:
  - Auth:** NONE
  - ARN:** `arn:aws:execute-api:us-east-1:302211264422:it9pqb5zm7/*/POST/dra`
- Integration Request**: A box containing:
  - Type:** MOCK
- Mock Endpoint**: A vertical bar on the right representing the mock endpoint.
- Integration Response**: A box containing:
  - HTTP status pattern:** - (with a dropdown arrow)
  - Output passthrough:** Yes
- Method Response**: A box containing:
  - HTTP Status:** 200
  - Models:** `application/json => Empty`

Arrows indicate the flow: Client → Method Request → Integration Request → Mock Endpoint → Integration Response → Method Response → Client.

Navigate back to the resources and then POST method





The place you apply request validation is on the method request.

- APIs
- Custom Domain Names
- VPC Links
  
- API: dragons
  - Resources
    - GET
    - OPTIONS
    - POST
  - Stages
  - Authorizers
  - Gateway Responses
  - Models
  - Resource Policy
  - Documentation
  - Settings
- Usage Plans
- API Keys
- Client Certificates
- Settings

Resources Actions **Method Execution** /dragons - POST - 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

▶ URL Query String Parameters

▶ HTTP Request Headers

▼ Request Body

Content type	Model name
No Models	

+ Add model

▶ SDK Settings

API Gateway console page showing the configuration for a POST method on the /dragons resource. The page includes a left sidebar with navigation options, a breadcrumb trail, and a main content area with settings for authorization, request validation, and request body.

URL: console.aws.amazon.com/apigateway/home?region=us-east-1#/apis/it9pqb5zm7/resources/eqdk33/methods/POST

APIs

Custom Domain Names

VPC Links

API: dragons

- Resources
- Stages
- Authorizers
- Gateway Responses
- Models
- Resource Policy
- Documentation
- Settings

Usage Plans

API Keys

Client Certificates

Settings

Resources

Actions

Method Execution /dragons - POST - 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

URL Query String Parameters

HTTP Request Headers

Request Body

Content type	Model name
application/json	dragon

SDK Settings

API Gateway console showing the configuration for a POST method. The 'Request Validator' dropdown menu is open, showing options: NONE (selected), Validate body, Validate body, query string parameters, and headers, and Validate query string parameters and headers.

The 'Request Body' section shows a table with content type 'application/json' and model name 'dragon'.

Content type	Model name
application/json	dragon

to actually validate this, select the request validator, select the validate body and then click save.

APIs

Custom Domain Names

VPC Links

API: dragons

Resources

Stages

Authorizers

Gateway Responses

Models

Resource Policy

Documentation

Settings

Usage Plans

API Keys

Client Certificates

Settings

Resources

Actions

[← Method Execution](#) /dragons - POST - Method Request

/

/dragons

GET

OPTIONS

POST

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

## Settings

Authorization NONE

Request Validator Validate body

API Key Required false

## ▶ URL Query String Parameters

## ▶ HTTP Request Headers

## ▼ Request Body

Content type

Model name

application/json

dragon

+ Add model

## ▶ SDK Settings

- Documentation
- Settings
- Usage Plans
- API Keys
- Client Certificates
- Settings

Accept:application/json.

### Stage Variables

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

### Client Certificate

No client certificates have been generated.

### Request Body

```
1
```



API Gateway

console.aws.amazon.com/apigateway/home?region=us-east-1#/apis/it9pqb5zm7/resources/eqdk33/methods/POST

Apps Onboarding Training AWS Sandbox Travel AWS Resources DynamoDB Containers Workshops Imported CD workshops GitHub - aws-sam...

- Gateway Responses
- Models
- Resource Policy
- Documentation
- Settings
- Usage Plans
- API Keys
- Client Certificates
- Settings

### Headers

**{dragons}**

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

1
---

### Response Headers

```
{ "x-amzn-ErrorType": "BadRequestException" }
```

### Logs

```
Execution log for request 1629434d-b656-4ca3-a372-44485a74478
Fri May 15 22:07:52 UTC 2020 : Starting execution for request: 1629434d-b656-4ca3-a372-44485a74478
Fri May 15 22:07:52 UTC 2020 : HTTP Method: POST, Resource Path: /dragons
Fri May 15 22:07:52 UTC 2020 : Method request path: {}
Fri May 15 22:07:52 UTC 2020 : Method request query string: {}
Fri May 15 22:07:52 UTC 2020 : Method request headers: {}
Fri May 15 22:07:52 UTC 2020 : Method request body before transformations:
Fri May 15 22:07:52 UTC 2020 : Request body does not match model schema for content type application/json: [Unknown error parsing request body]
Fri May 15 22:07:52 UTC 2020 : Method completed with status: 400
```

This error is what we would expect here because this is an empty request body

The screenshot displays the AWS API Gateway console interface. The left-hand navigation pane includes sections for Gateway Responses, Models, Resource Policy, Documentation, Settings, Usage Plans, API Keys, Client Certificates, and another Settings section. The main content area is divided into three panels: Headers, Stage Variables, and Request Body. The Headers panel shows a note about header syntax. The Stage Variables panel indicates that no stage variables exist for this method. The Request Body panel displays a JSON object with the following structure:

```
1 - {
2     "dragonName": "Frank",
3     "description": "This dragon is brand
4         new, we don't know much about it
5         yet.",
6     "family": "purple",
7     "city": "Seattle",
8     "country": "USA",
9     "state": "WA",
10    "neighborhood": "Downtown",
11    "reportingPhoneNumber": "1555555555",
12    "confirmationRequired": false
13 }
```

The Logs panel on the right provides a detailed execution log for request ID 923e6680-3d05-4b93-9f2a-92be053a0bb7. The log shows the request method (POST), path (/dragons), and the request body before transformations. The response body after transformations is also shown, matching the request body. The log concludes with the message: "Successfully completed execution".

Successfully completed the execution with a sample request body.



API Gateway is now validating the incoming request against a model and it is responding with the GET method with those mappings that we already had created.

That means that this API is totally mocked up.

So now we can deploy it and interact with it as if it was our real API.

**Publish API**

After creating your API, you must **deploy** it to make it **callable** by your users.  
To deploy an API, you create an **API deployment** and associate it with a **stage**.

A **stage** is a logical reference to a lifecycle state of your API  
(for example, dev, prod, beta, v2).

API stages are identified by the **API ID** and **stage name**.

They're included in the URL that you use to invoke the API.

Every time you update an API,  
you must redeploy the API to an existing stage or to a new stage.

Updating an API includes modifying routes, methods, integrations, authorizers,  
and anything else other than stage settings.

Our API is built and mocked, so now let's deploy it to a stage.

The screenshot shows the AWS API Gateway console interface. The breadcrumb navigation is: APIs > dragons (it9pqb5zm7) > Resources > /dragons (eqdk33). The main content area is titled "/dragons Methods". A dropdown menu labeled "Actions" is open, showing two sections: "RESOURCE ACTIONS" and "API ACTIONS". The "API ACTIONS" section is expanded, and the "Deploy API" option is highlighted with a mouse cursor. Other options in the "API ACTIONS" section include "Import API", "Edit API Documentation", and "Delete API". The "RESOURCE ACTIONS" section includes "Create Method", "Create Resource", "Enable CORS", "Edit Resource Documentation", and "Delete Resource".

API Gateway console interface showing the "Actions" menu for a resource method. The "Deploy API" option is highlighted.

Navigation: APIs > dragons (it9pqb5zm7) > Resources > /dragons (eqdk33)

Resource: /dragons

Method: /dragons Methods

Actions:

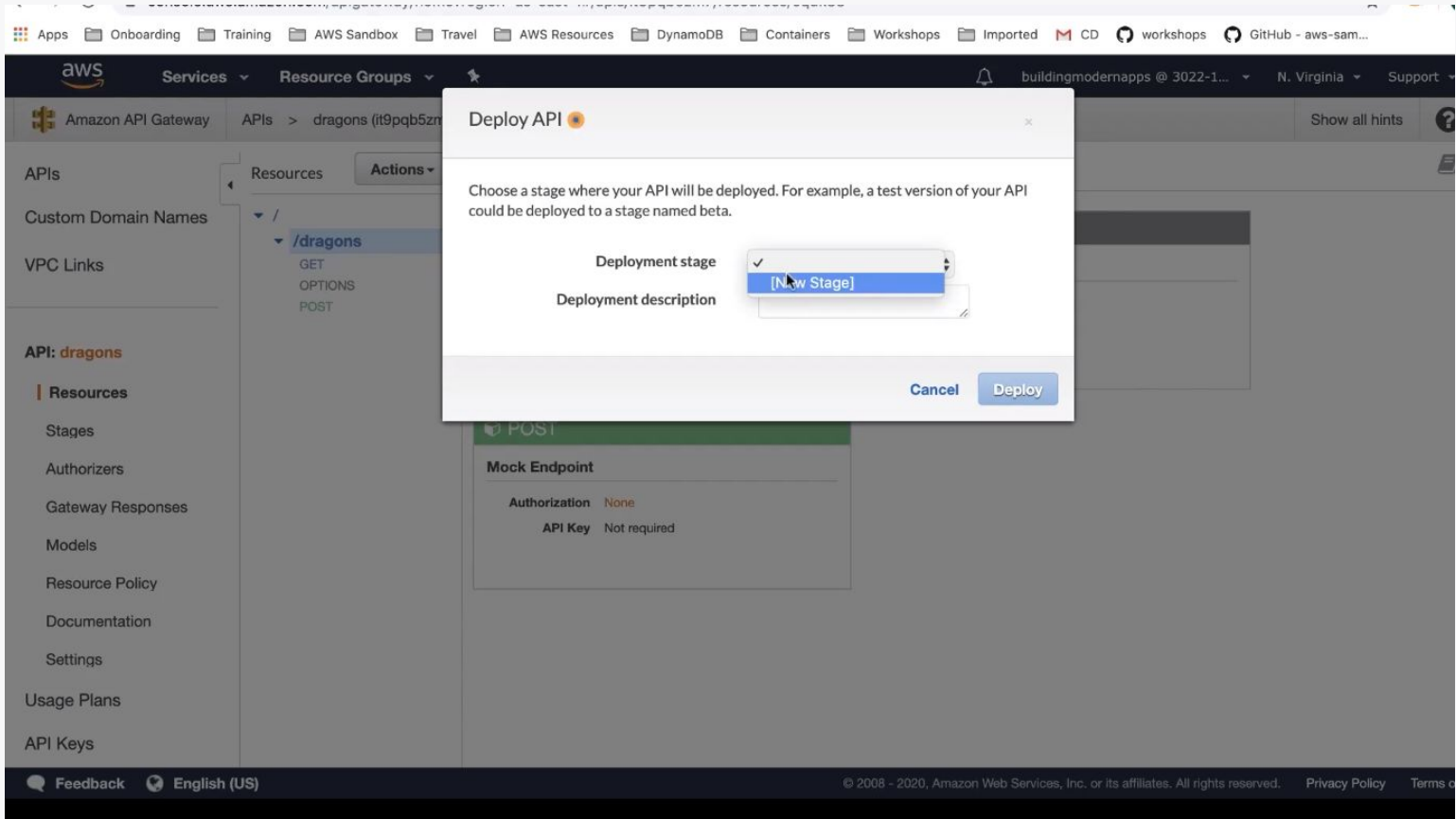
- RESOURCE ACTIONS
  - Create Method
  - Create Resource
  - Enable CORS
  - Edit Resource Documentation
  - Delete Resource
- API ACTIONS
  - Deploy API
  - Import API
  - Edit API Documentation
  - Delete API

Mock Endpoint:

- Authorization: None
- API Key: Not required

API Key: None, Not required

Click actions and the click deploy API.



create a new deployment stage



Apps Onboarding Training AWS Sandbox Travel AWS Resources DynamoDB Containers Workshops Imported CD workshops GitHub - aws-sam...

Services Resource Groups buildingmodernapps @ 3022-1... N. Virginia Support

Amazon API Gateway APIs > dragons (it9pqb5zn)

APIs Resources Actions

Custom Domain Names

VPC Links

API: dragons

Resources

- Stages
- Authorizers
- Gateway Responses
- Models
- Resource Policy
- Documentation
- Settings
- Usage Plans
- API Keys

GET OPTIONS POST

Authorization None

API Key Not required

### 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

Deployment description

Cancel Deploy

Feedback English (US)

© 2008 - 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy Policy Terms of Service

The screenshot shows the AWS API Gateway console interface. The top navigation bar includes the AWS logo, 'Services', 'Resource Groups', and user information. The breadcrumb trail indicates the path: 'APIs > dragons (it9pqb5zm7) > Stages > test'. The left sidebar contains a navigation menu with categories like 'APIs', 'Custom Domain Names', 'VPC Links', and 'API: dragons'. Under 'API: dragons', the 'Stages' option is selected. The main content area is titled 'test Stage Editor' and features a 'Create' button. A prominent light blue banner displays the 'Invoke URL: https://it9pqb5zm7.execute-api.us-east-1.amazonaws.com/test'. Below this, a series of tabs are visible: 'Settings', 'Logs/Tracing', 'Stage Variables', 'SDK Generation', 'Export', 'Deployment History', 'Documentation History', and 'Canary'. The 'Settings' tab is active, showing sections for 'Cache Settings' (with an 'Enable API cache' checkbox), 'Default Method Throttling' (with 'Enable throttling' checked and fields for 'Rate' at 10000 and 'Burst' at 5000 requests per second), and 'Web Application Firewall (WAF)'.

It gives you an invoke URL at the top. This is the endpoint for your API.

Now, we have our API deployed,  
and we have an active endpoint.

Now, let's test it out.

I don't want to use the test features built into the console, but instead.

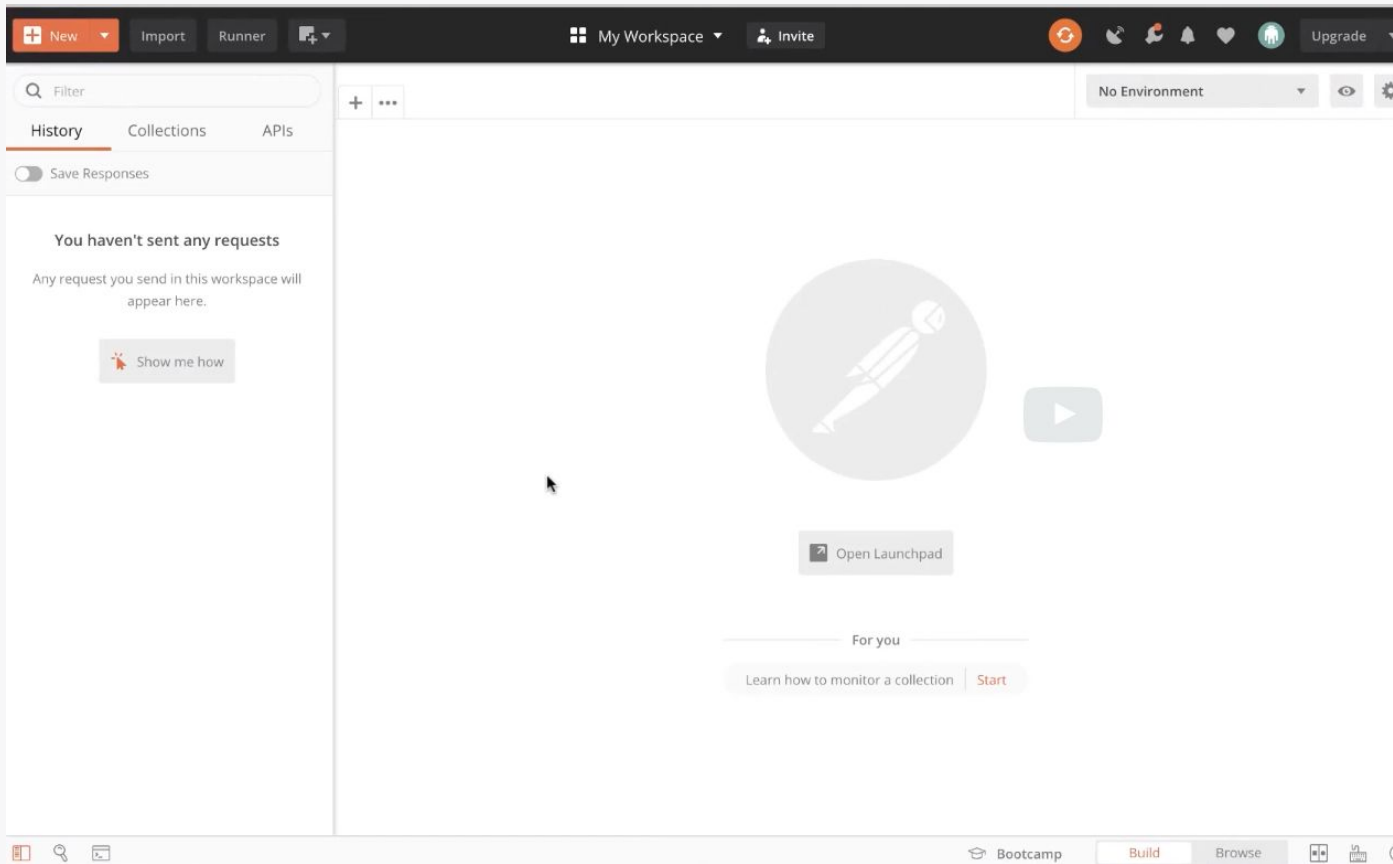
Let's actually hit this endpoint  
from an external entity using a tool called **Postman**.

(You could use cURL on the command line as well)

Postman is not an AWS tool, but rather a tool used very widely in the developer community.

The Postman API client allows you to configure requests, invoke APIs and then view the result.

This is just to show you how to hit your endpoint using a service that is outside of the AWS environment, proving that your API is deployed and ready to go.



Sign into the Postman app

Filter

History Collections APIs

Save Responses

You haven't sent any requests

Any request you send in this workspace will appear here.

Show me how

GET list-dragons

No Environment

list-dragons

Comments 0 Examples 0

GET https://it9pqb5zm7.execute-api.us-east-1.amazonaws.com/test/dragons

Sending... Save

Params Authorization Headers (7) Body Pre-request Script Tests Settings Cookies Code

Query Params

KEY	VALUE	DESCRIPTION	...	Bulk Edit
key	Value	Description		

Response

Sending request...

Cancel

Hit Send to get a response

For you

Learn how to monitor a collection Start



Filter

History Collections APIs

Save Responses Clear all

Today

GET https://it9pqb5zm7.execute-api.us-east-1.amazonaws.com/test/dragons

GET list-dragons No Environment

list-dragons Comments 0 Examples 0

GET https://it9pqb5zm7.execute-api.us-east-1.amazonaws.com/test/dragons Send Save

Params Authorization Headers (7) Body Pre-request Script Tests Settings Cookies Code

Query Params

KEY	VALUE	DESCRIPTION
Key	Value	Description

Body Cookies Headers (7) Test Results Status: 200 OK Time: 59ms Size: 1.81 KB Save Response

Pretty Raw Preview Visualize JSON

```

1  {
2
3    "description_str": "From the northern fire tribe, Atlas was born from the ashes of his fallen father in combat.
4      He is fearless and does not fear battle.",
5    "dragon_name_str": "Atlas",
6    "family_str": "red",
7    "location_city_str": "anchorage",
8    "location_country_str": "usa",
9    "location_neighborhood_str": "w fireweed ln",
10   "location_state_str": "alaska"
11  },
12  {
13    "description_str": "Protheus is a wise and ancient dragon that serves on the grand council in the sky world. He
14      uses his power to calm those near him.",
15    "dragon_name_str": "Protheus",
16    "family_str": "blue",
17    "location_city_str": "brandon",
18    "location_country_str": "usa",

```

The screenshot shows a REST client interface with the following components:

- Header:** Includes navigation buttons like 'New', 'Import', 'Runner', and 'My Workspace', along with an 'Upgrade' button.
- Left Sidebar:** Contains a search filter, 'History', 'Collections', and 'APIs' sections. Under 'Today', three GET requests are listed with their respective URLs.
- Main Panel:** Displays a selected GET request for 'list-dragons' with the URL `https://it9pqb5zm7.execute-api.us-east-1.amazonaws.com/test/dragons?family=blue`. The 'Send' button is highlighted in blue.
- Params Section:** Shows 'Query Params' with a table:

KEY	VALUE	DESCRIPTION	...	Bulk Edit
<input checked="" type="checkbox"/> family	blue			
Key	Value	Description		

Below the table, the 'Body' section shows the response status: 'Status: 200 OK Time: 76ms Size: 685 B'. The response is displayed in 'Pretty' JSON format:

```
1 [{"description_str": "Protheus is a wise and ancient dragon that serves on the grand council in the sky world. He
2   uses his power to calm those near him.",
3   "dragon_name_str": "Protheus",
4   "family_str": "blue",
5   "location_city_str": "brandon",
6   "location_country_str": "usa",
7   "location_neighborhood_str": "e morgan st",
8   "location_state_str": "florida"}]
```

test with query parameter

To download and try out Postman for yourself click here:

<https://www.postman.com/downloads>

## **Exercise 2: Amazon API Gateway**

In this lab, you will continue to build the Dragons application.

First, you will build the REST API that's used to list and add dragons.

You will create the API by using mock integrations.

You can use mock integrations to create a testable API before you write any code for your backend services.

After the API is created, you will deploy an updated version of the web application.

The web application contains the frontend logic to access the GET and POST methods of your `/dragons` resource.

Start from where you stopped at the end of Lab 1,  
and continue to build the application.

## Exercise 2: Amazon API Gateway

Upload a doc with a screenshot for each completed Task as a Lab report in Moodle.



## Task: Deleting all lab resources

### 1. Delete the API Gateway DragonsApp API

- Open the Amazon API Gateway dashboard.
- In the navigation pane, choose APIs.
- Delete the DragonsApp API and confirm the deletion.

### 2. Delete the S3 bucket for the Dragons application.

- Open the Amazon Simple Storage Service (Amazon S3) dashboard.
- Delete the bucket that ends with -dragons-app and confirm the deletion. You must empty the bucket before you delete it.

### 3. Delete the AWS Cloud9 development environment for this project.

- Open the AWS Cloud9 dashboard.
- Delete the Python-DevEnv environment and confirm the deletion.