

# Exercise 2.2 Amazon API Gateway

## Exercise 2.2: Creating an API Gateway endpoint and wiring it up to our web site

We have a website sitting on S3 being exclusively delivered via CloudFront, but the problem is that the website is static.

This means when you are interacting with it is "faking it". It is hard-coded in the Javascript to replay with "the temperature in X is 20 degrees".

We need to have our website's JavaScript hit an endpoint in the cloud and return data from that. Perhaps a different temperatures say, 69 degrees. That way when we test it we will know it is hitting our API successfully.

Creating a backend API is easy using Amazon API gateway.

Before we create an API that does anything fancy, such as look up weather data from a database or run function, we want to MOCK our API.

This is a fancy way of saying that will make a request to the API and get a temperature of 60 degrees back regardless of what information we pass to the back end. i.e A hard coded mock API.

Once the mock is wired up and we have tested our website, we can start thinking about putting functionality behind it. We will look at server-less functions next week with John, for now, let's get this mock API built and tested.

### 1. Steps for creating a REST endpoint with API Gateway

- Sign in to the AWS Management Console and in the **Find Services** search box type api and choose **API Gateway**.
- Make sure you are in the **N. Virginia** region at the top right.
- Click **Get Started** and click **OK** to remove the Create Example API pop-up.
  - If you already have APIs click **Create API**
- Choose protocol **REST**.
- Under Create new API choose **New API**.

- For settings use the following:
  - API name: **CatWeather**.
  - Optionally, add a brief description in **Description**.
  - Endpoint Type: **Regional**.

## 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.

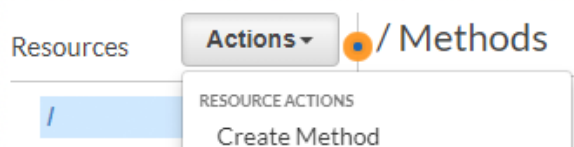
☒ **New API** ☐ **Import from Swagger or Open API 3** ☐ **Example API**

## Settings

Choose a friendly name and description for your API.

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

- Click **Create API**.
- Click **Actions** and **Create Method**.



- Choose **POST** from the list under the / resource and click the **check mark icon**.



- For **Integration type** choose **Mock** and click **Save**.

- Click **TEST**. 

- Under **Request Body** enter in:

```
1  {  
2    "city_str": "VEGAS"  
3  }
```

- Click **Test** and you should get back something similar in the logs section:

Under Response Body:

```
1  no data
```

And under Response Headers:

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

Logs should look similar to:

```
1  HTTP Method: POST, Resource Path: /  
2  Method request path: {}  
3  Method request query string: {}  
4  Method request headers: {}  
5  Method request body before transformations: {  
6    "city_str": "VEGAS"  
7  }  
8  Method response body after transformations:  
9  Method response headers: {Content-Type=application/json}  
10 Successfully completed execution  
11 Method completed with status: 200
```

- Scroll up and click **Method Execution** [← Method Execution](#) to go back to the test settings.
- Click **Integration Response**.
- Click the arrow to open up the response properties.
- Click the drop down arrow for **Mapping Templates**.

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:

[Cancel](#) [Save](#)

Header Mappings

Mapping Templates

Content-Type

[application/json](#)

[Add mapping template](#)

- Click on **application/json** blue hyperlink under **Content-Type**.
- Under **General template** choose **Method Request passthrough**.
- Replace it all with:

```
1 {  
2   "temp_int": 69  
3 }
```

- Click **Save** at the bottom right.

Generate template: Method Request passthrough ▾

```
1 {  
2   "temp_int": 69  
3 }
```

Cancel Save

- **Also** click **Save** at the top right.
- Choose **Method Execution** again.
- Click **Test**.
- Set the request body as

```
1 {  
2   "city_str": "CHICAGO"  
3 }
```

- click **Test** again.
- You should see the following:

Under Response Body:

```
1 {  
2   "temp_int": 69  
3 }
```

Response Headers:

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



Logs should show something similar to:

```
1 Execution log for request bc5f8d95-3f6b-11e9-9e4c-a7f1e746c2c6
2 Starting execution for request: bc5f8d95-3f6b-11e9-9e4c-a7f1e746c2c6
3 HTTP Method: POST, Resource Path: /
4 Method request path: {}
5 Method request query string: {}
6 Method request headers: {}
7 Method request body before transformations:  {
8     "city_str": "CHICAGO"
9 }
10 Method response body after transformations: {
11     "temp_int": 69
12 }
13
14 Method response headers: {Content-Type=application/json}
15 Successfully completed execution
16 Method completed with status: 200
```

Awesome! We have a working MOCK API that returns hard coded data.

Now we need to wire this up to our website, but there's a problem.


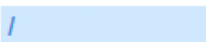
The problem is that our website is being hosted via a CloudFront domain name, and this API (once we publish it) will be available at a different API Gateway domain names, and the browser will bark at you if you try to go across domains.

Basically, if you try to bring in content from a different domain the browser needs to be sure that it is allowed to be pulled into that (cross domain) website.

The way we let the browser know that it is ok for a website of a different domain (CF) to get information from us our API Gateway endpoint is to set up what we call CORS (Cross Origin Resource Sharing).

Luck for us, this is very easy to do via the API Gateway console. For simplicity we will say any website (using the asterix \*) that request information from our endpoint will be allowed to consume our content.

## 2. Steps to enable CORS in the API gateway console.

- Click on the resource `/` (NOT POST)  
- Select **Actions** and **Enable CORS**.
- Select **DEFAULT 4XX** and **DEFAULT 5XX**. Leave the other settings as the default.

Gateway Responses for *CatWeather API* ☒ DEFAULT 4XX ☒ DEFAULT 5XX ⓘ

Methods ☒ POST ☒ OPTIONS ⓘ

Access-Control-Allow-Methods POST, OPTIONS ⓘ

Access-Control-Allow-Headers 'Content-Type,X-Amz-Date,Authorizatio ⓘ

Access-Control-Allow-Origin\*  ⓘ

▶ Advanced

- Click **Enable CORS** and replace existing CORS headers.
- Click **Yes, replace existing values** on the **Confirm method changes** pop-up.
- Click **Actions** and choose **Deploy API** under API Actions.
- On the **Deploy API** settings pop-up select:
  - Deployment stage **[New Stage]**
  - Stage name `test`
  - Stage description `test`
  - Click **Deploy**.
- Copy down the **Invoke URL**. e.g <https://j5c1lgdjf6.execute-api.us-east-1.amazonaws.com/test>
- This is a post request so IF you simply visit that URL in the browser you will get this error:

```
1 {  
2   "message": "Missing Authentication Token"  
3 }
```

Now our mock API is tested, CORS is enabled and fully deployed. We need to have our website call that endpoint to get weather data (always 69 degrees as it is a MOCK remember).

### 3. Steps to wire up your mock API to your website

You don't have to write any code for this next bit. All you need to do is change the `API_GATEWAY_URL_STR` value in the website's `/scripts/config.js` file. Save it, and re-upload it back to S3, CloudFront will pick up the changes to S3 automatically.

- On your local machine, open the folder where you unzipped the file and edit the file `s3website/scripts/config.js`
- Replace the `var API_GATEWAY_URL_STR` with the **Invoke URL** you copied from step 2.
- It will look like this:

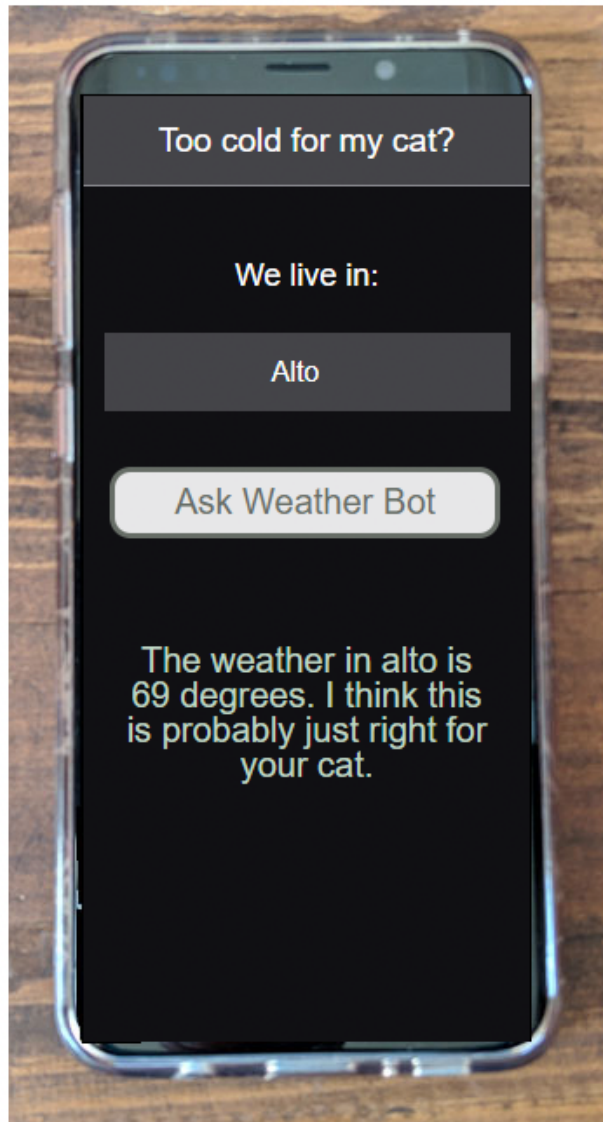
```
1  var API_GATEWAY_URL_STR = null;
```

- Use your own deployed API gateway URL (above). Like in this example:

```
1  var API_GATEWAY_URL_STR = "https://Your-Invoke-URL.execute-api.us-east-1  
    .amazonaws.com/test";
```

- Save the file and upload it back to S3 into the **scripts** folder, remembering to set `Cache-Control` to `max age=0`. (*Refer to week 1 if you don't know how to upload items to S3*)
- Browse to your website (i.e. your CloudFront distribution URL)
- Choose a city, and press **Ask Weather Bot**.





- The weather should always return **69 degrees** no matter which city is chosen.

We now have a live MOCK API interacting with our website. 😊

Next week we will look at wiring up our API Gateway to a function (Lambda), and will use IAM, and Cloudwatch.

See you next week!

## Exercise goal checklist

---

1. ~~Create a simple chatbot using the lex console.~~
2. ~~Upload our website to S3.~~
3. ~~Create a content delivery network and lock down S3.~~
4. ~~Build an API gateway mock with CORS.~~
5. Build a Lambda mock, use IAM, push logs to CloudWatch.
6. Create and seed a database with weather data.
7. Enhance the lambda, so it can query the database.
8. Play with your new text based data driven application.
9. Create a LEX proxy using Lambda.
10. Enhance API gateway to use the LEX proxy.
11. Play with your new voice web application.