Exercise 3.1 Creating an AWS Lambda Function

Exercise 3.1: Creating a Lambda function

Currently we have our website communicating with a hard coded API mock.

We need to start adding functionality to the API. Such as having a function run when then API is hit that calls a database for weather information.

However to introduce you to the concept of server-less functions and the plumbing that runs it we shall start simple and create the worlds simplest function that returns. You guessed it, a hard coded value back through API Gateway all the way to the browser.

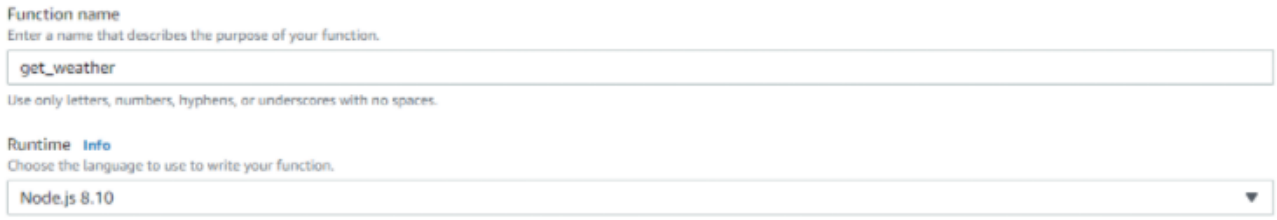
We need to create a server-less function that regardless of what city name you send, it will reply with a fixed temperature and echo back the city name.

This is essentially a Lambda Mock, to replace the API Gateway mock.

This is a useful learning step before making the Lambda function do anything intelligent. Even just creating a simple Lambda mock will introduce a few new services to you. Such as CloudWatch logs for debugging, and IAM for establishing both invocation permissions (what can trigger it [API gateway]) and the execution permission. Basically what Lambda can write to [CloudWatch logs]).

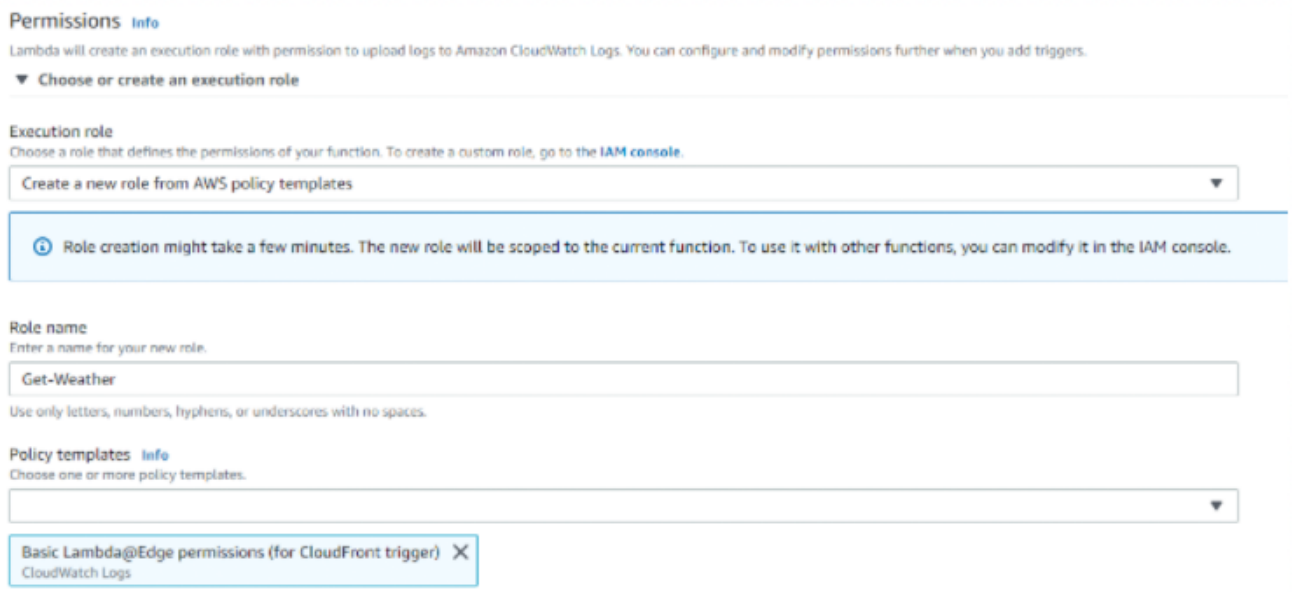
1. Steps for create a simple Lambda function (our new mock)

- Sign in to the AWS Management Console and in the **Find Services** search box type lambda and choose **Lambda**.
- Make sure you are in the **N. Virginia** region at the top right.
- Click **Create function**.
- Select **Author from scratch**.
  - Name the function `get_weather`.
  - Select **Node.js 8.10** from the **Runtime** list.



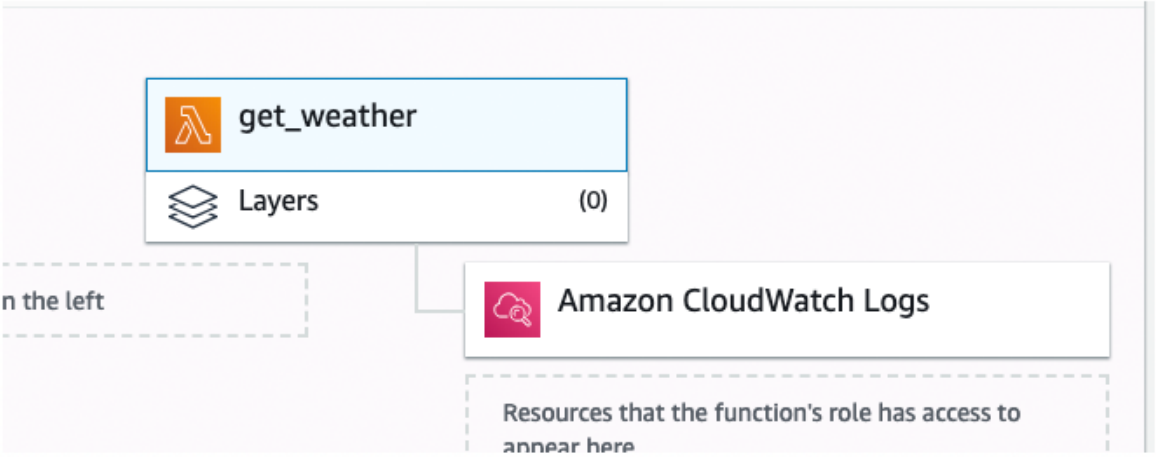
  - For **Execution role** choose **Create a new role from AWS policy templates**.
  - For **Role name** name the role `Get-Weather`.
  - From the **Policy templates** list scroll down and choose **Basic Lambda@Edge permissions (For CloudFront trigger)**.

We use the **Basic Lambda@Edge permission** because it is a very managed simple policy that allows us to write to CloudWatch Logs, which is all we really need right now, in terms of executions permissions at least.



- Click **Create function**.
- This will give us permission to write to CloudWatch logs as well as create a log group.

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



- Scroll down to the inline code editor.
- Paste the following code into the **index.js** tab replacing the existing code:

```
function handler(event, context, callback){
    var
        city_str = event.city_str,
        response = {
            city_str: city_str,
            temp_int: 74
        };
    console.log(response);
    callback(null, response);
 }
 exports.handler = handler;
```

This code simply takes the city as a string and will echo it back along with a 74 temperature, as a Number then exits successfully.

You will notice we console log out the response too. This is so you can view this information in CloudWatch logs, if for example you needed to debug it.

API Gateway can interact with this function, essentially acting as an intermediary between the clients browser and your Lambda function.

- No need to change any of the defaults.
- Click **Save**.

Again, later on we will make this Lambda function smarter, for now a mock is fine to get all the plumbing in place and to be able to test it all.

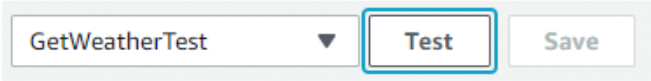
Talking of testing, let's create a test case for our Lambda function and make sure it works as intended if we send it a random city.

2. Steps to create a test case for our Lambda Mock

- Click the dropdown that says **"select a test event"**
- Choose **configure test events**
- Keep the default **Create new test event** selected.
- For **Event template** leave it as **Hello World**.
- For **Event name** type in **GetWeatherTest**.
- Paste the following into the inline code editor:

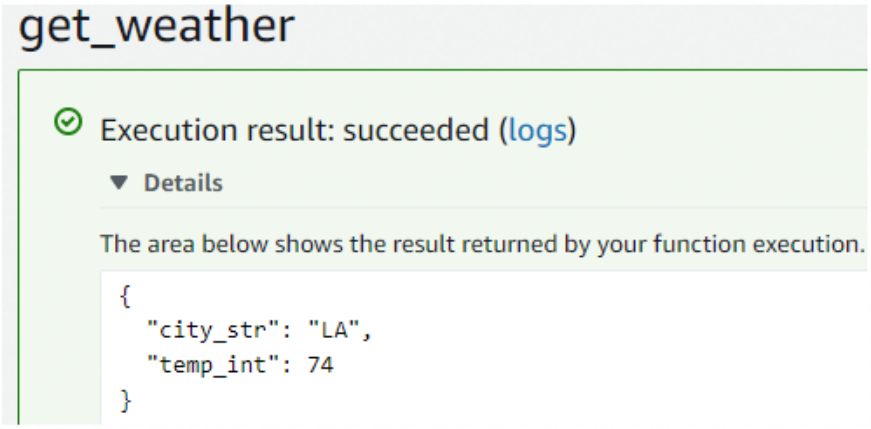
```
{
  "city_str": "LA"
}
```

- Click **Create**.
- The test case should now be saved at the top right.

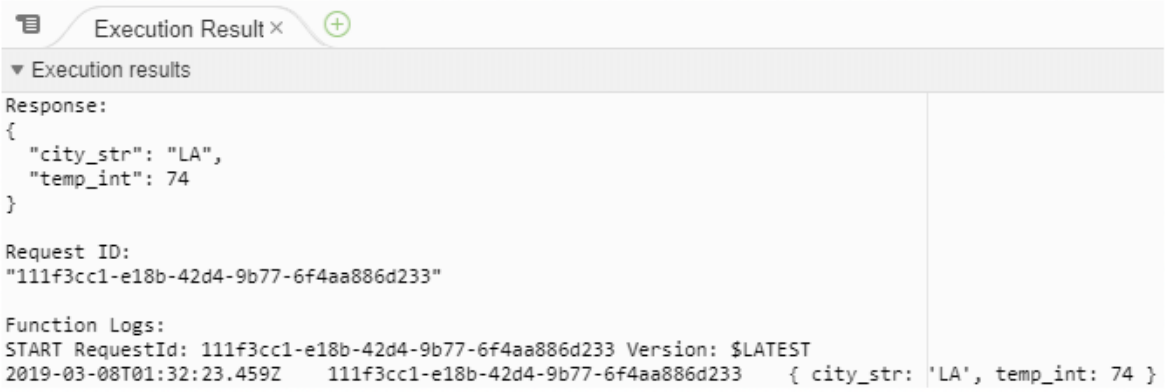


- Click **Test**.

You should now see at the top the execution succeeded:



You can also view this at the bottom of the inline code editor under the **Execution Result** tab.



You could edit the `city_str` with a new city if you like, but at this point, it should be workinbg as expected.

Currently our website will still return 69 degrees because the API Gateway that it is pointing to is a hard coded mock. All we need to do is go into the API gateway console and replace the hard coded text mock that we created in week 2 with this new lambda mock.

We only need to make a small change to our API's configuration in the console, and then we should be able to test our website and get 74 degrees returned instead of 69.

3. Steps to replace the API mock endpoint with the `get_weather` Lambda function

- Click **Services** and search for API, and then select **API Gateway**.
- Select the **CatWeather** API under **APIs**.
- Under resources choose **POST** 

and click **Integration Request**.
- For the **Integration type** change **Mock** to **Lambda Function**.
- ⚠️ ENSURE **Use Lambda Proxy Integration** is **de-selected**.
- Choose the **Lambda Region**: `us-east-1`.
- Type in the name of the Lambda function `get_weather`. (Once you type in `g` you should be able to select **get_weather** from the list).
- Leave **Use Default Timeout** checked.
- Click **Save**.
- At the **Switch to Lambda integration** pop-up. Click **OK**.
- It will tell you that you are adding **Permissions to Lambda Function**. Click **OK**.

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

Integration type
- (●) Lambda Function
- ( ) HTTP
- ( ) Mock
- ( ) AWS Service
- ( ) VPC Link

Use Lambda Proxy integration [ ]

Lambda Region us-east-1

Lambda Function get_weather

Execution role

Invoke with caller credentials [ ]

Credentials cache Do not add caller credentials to cache key

Use Default Timeout [x]

4. Test

- Click **Method Execution** at the top: ← Method Execution
- Click **Test**: TEST
- Paste the following into the **Request Body**:

```
{
    "city_str": "SEATTLE"
}
```

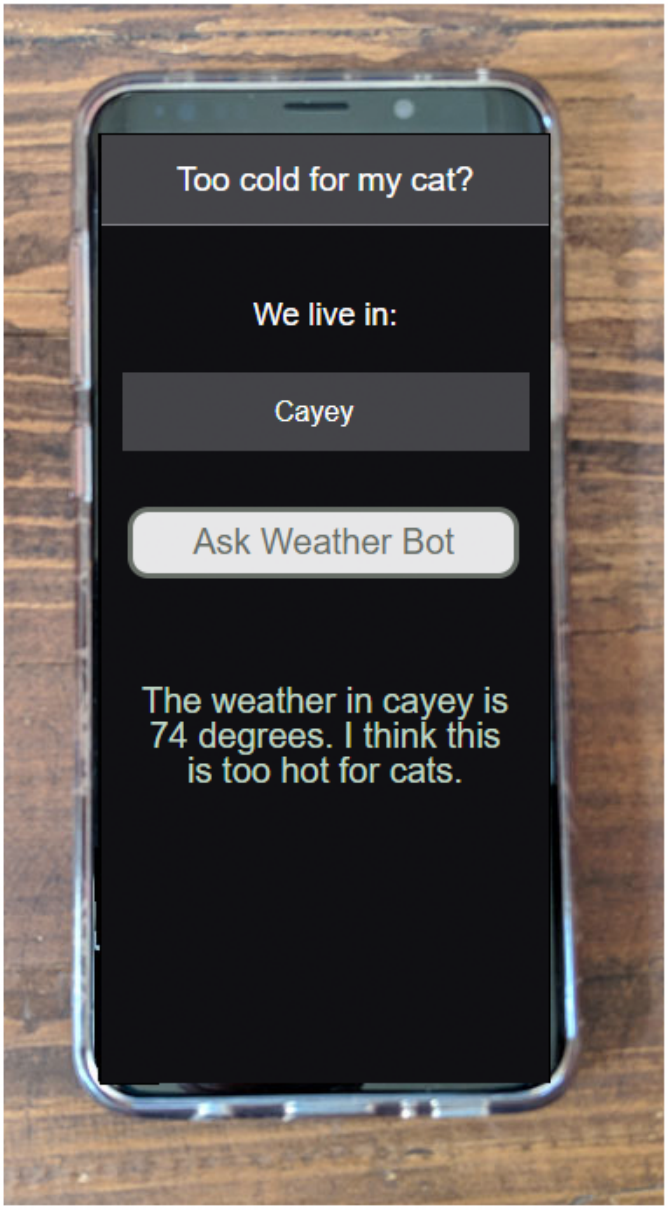
- Click **Test**.
- Under **Response Body** we can see the desired result:

```
{
  "city_str": "SEATTLE",
  "temp_int": 74
}
```

- We can also see in the **Logs** section that the request was indeed sent to our Lambda function: Similar to:

```
Sending request to https://lambda.us-east-1.amazonaws.com/2015-03-31/functions
    /arn:aws:lambda:us-east-1:179741345863:function:get_weather/invocations
Method response body after transformations: {"city_str":"SEATTLE","temp_int":74}
```

- We now need to re-enable **CORS** by going to /
- Click **Actions** and **Enable CORS**. Again select **DEFAULT 4XX** and **DEFAULT 5XX**.
- Click **Enable CORS and replace existing CORS headers**.
- Click **Yes, replace existing values** on the **Confirm method changes** pop-up.
- Click **Actions** again and **Deploy API**.
- Set **Deployment stage** to **test**.
- Click **Deploy**.
- We can now browse back to our CloudFront website.
- Choose a city, and click **Ask Weather Bot**.
- You should now see the update from Lambda.



Every city will now return 74 degrees.

Awesome you now have a server-less website that hits a functional (albeit a bit a bit dumb) backend.

We will now work to make it smarter by asking a database for real weather information based on city.

Of course we will need a database for that, so we will do that next.

We are doing well on our checklist tho'!

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 Lamba.
10. Enhance API gateway to use the LEX proxy.
11. Play with your new voice web application.