How to build a web application with front end and backend components using AWS technology

This document outlines the steps required to build a web application using technology from AWS. Specifically, we will be using

- DynamoDB to store the data in the back end
- Lambda functions in the middle to read and write data
- API Gateway features to access the lambda functions

For this example, I will be setting up a database that stores data from the Ontario Government web site concerning Covid-19. The same approach could be used for any type of data.

The steps involved are

1. Create a table in DynamoDB for the data
2. Create the Lambda functions
3. Create the API Gateway

We are going to do things iteratively. We will do steps 1-3 and then repeat steps 2-3 for the addition of a second lambda function

Assumptions

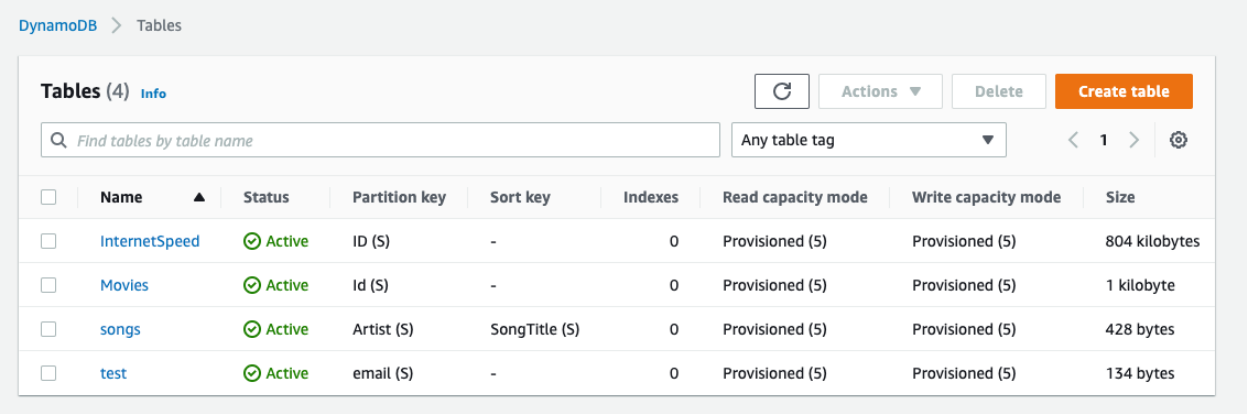
For this, I assume:

- You have access to AWS and some familiarity with it
- You will be setting things up in US-east-1
- You have access to the curl command

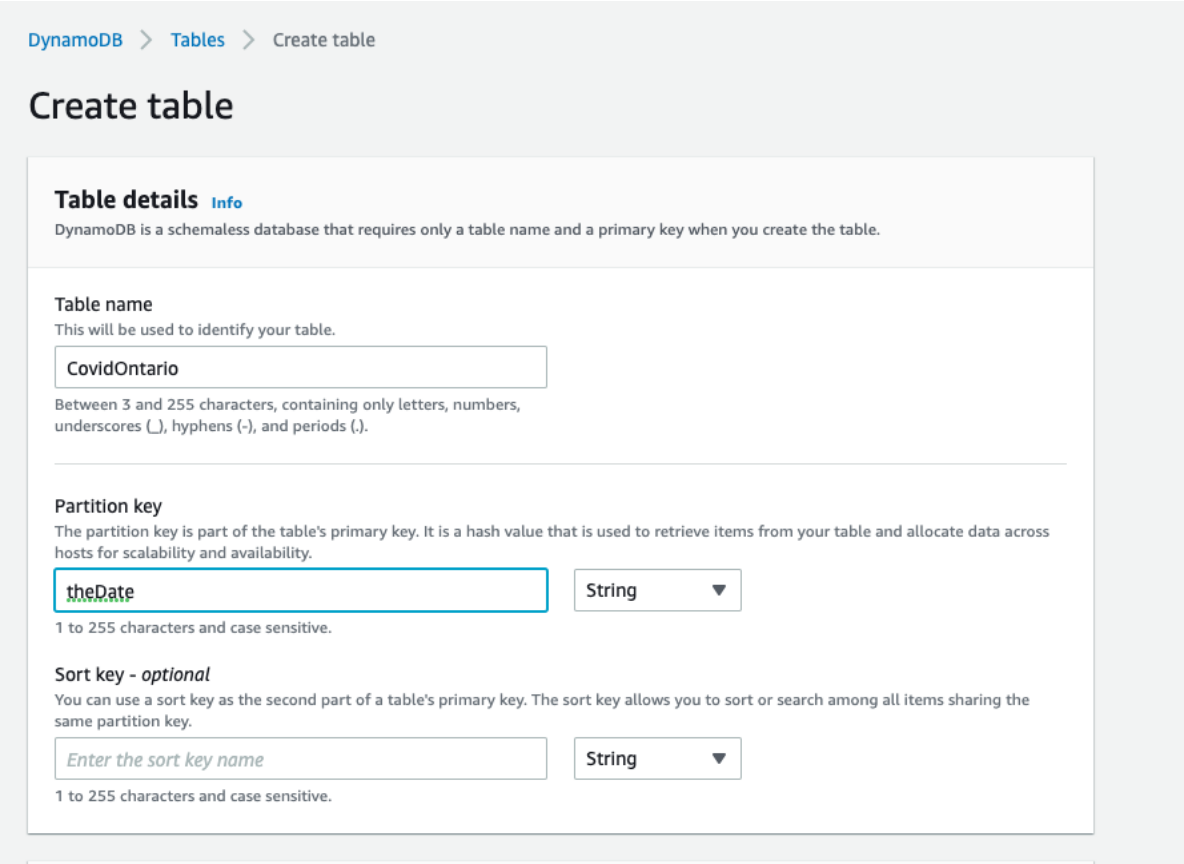
Create a table in DynamoDB

To work with Dynamo, go here: https://us-east-1.console.aws.amazon.com/dynamodbv2/home?region=us-east-1#tables

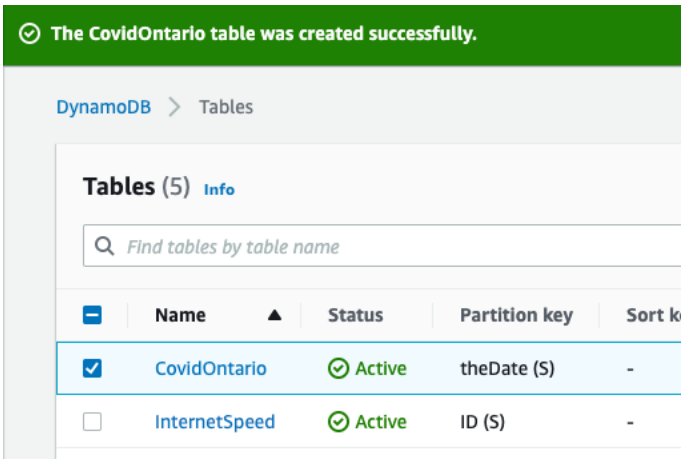
Click the button: Create Table (this is a screen grab of my account. I already have some tables)



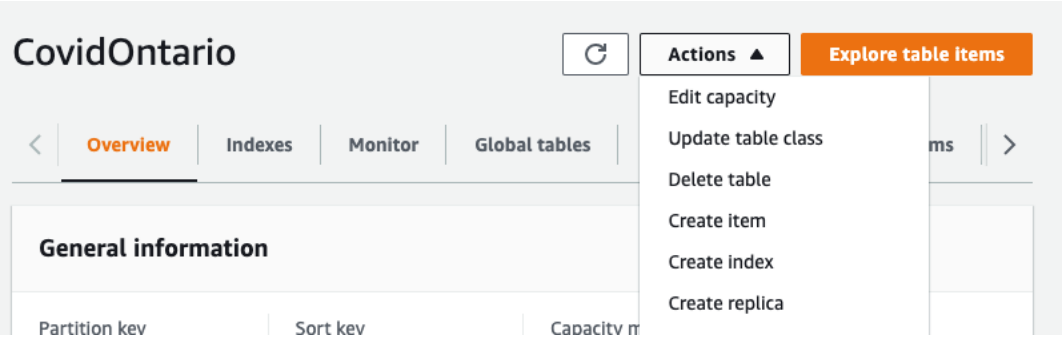
You will see the screen below. Fill it in this and then scroll down and click on Create Table button



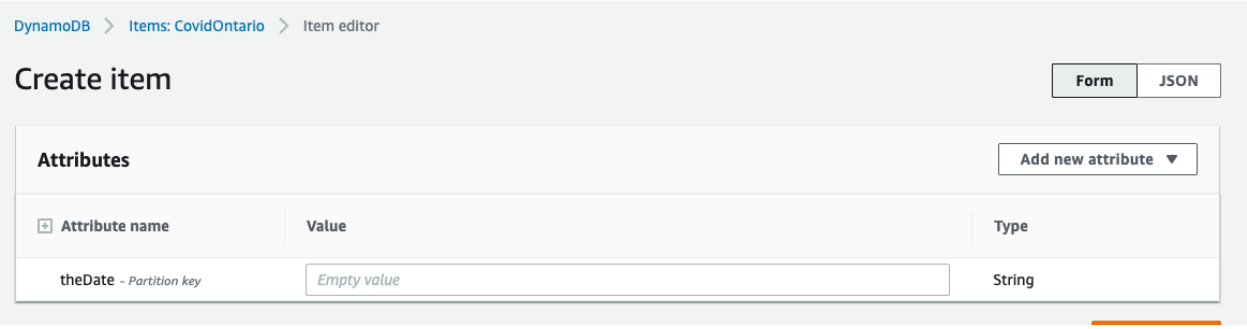
Wait until you see the following. Congrats, you have your table.



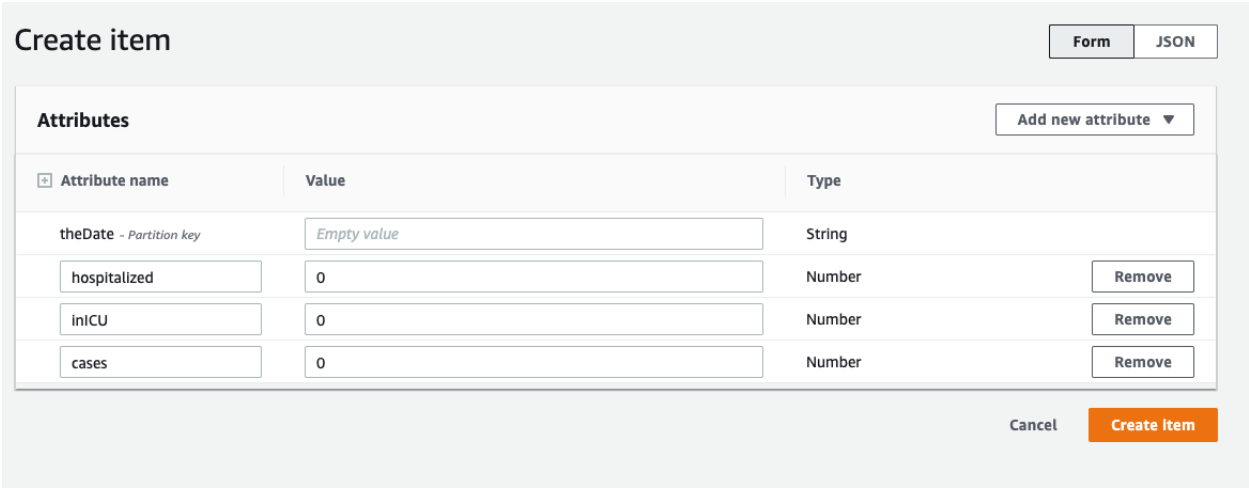
Now click on CovidOntario, then click on Actions > Create item



You want to be in Form mode, not Json mode. Now you are going to add attributes under theDate: hospitalized, inICU, cases. Each will be a number.



It should look like this. Now click Create Item. You will have to give theDate a value. Use 0.

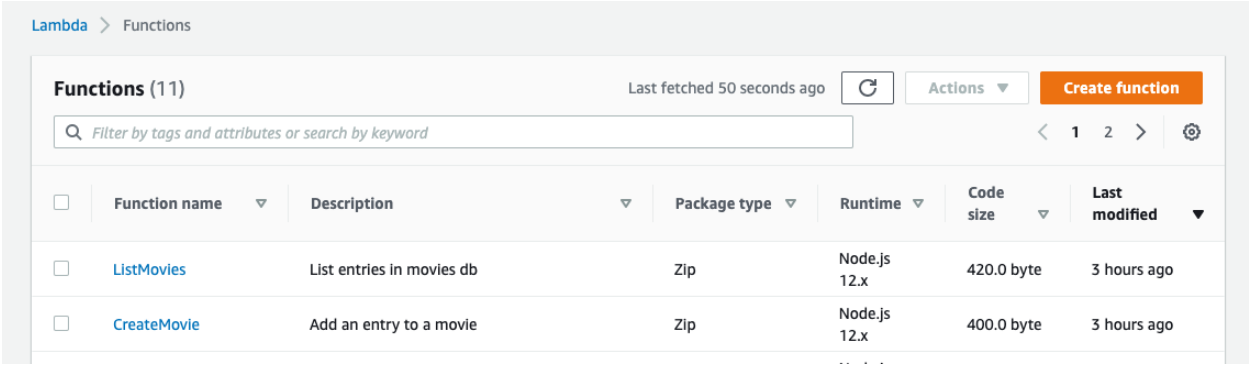


Now that we have a DynamoDB table with one item in it (you can delete it later). Let's make a Lambda function to talk to it.

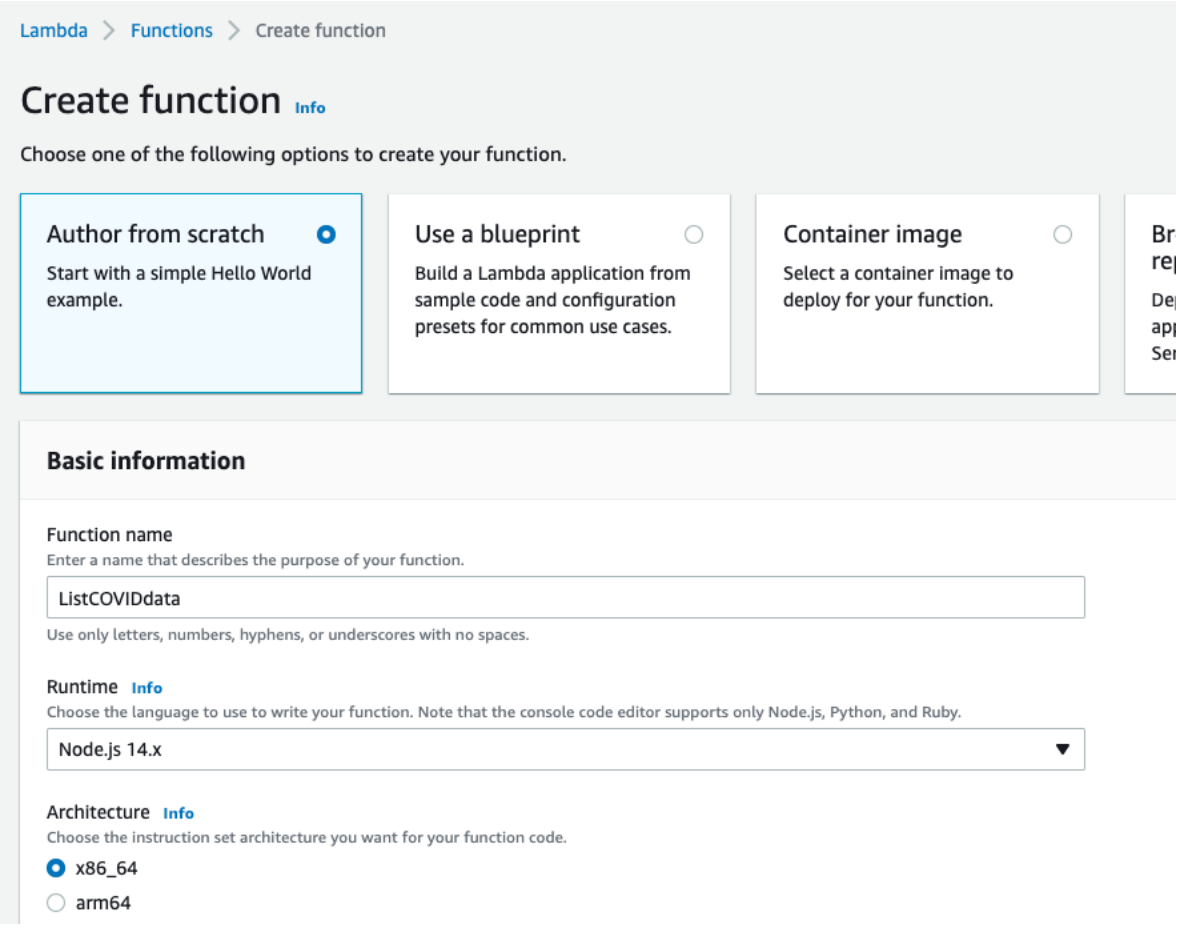
Second, create a lambda function

To create Lambda functions, go here:
https://us-east-1.console.aws.amazon.com/lambda/home?region=us-east-1#/functions

You will see a screen like this. Click on the button: Create function



Select Author from scratch and go with defaults. You will have to give the function a name and choose a runtime like I did for this to work:



The second half of this screen is below:

Architecture Info

Choose the instruction set architecture you want for your function code.

- (●) x86_64
- ( ) arm64

Permissions Info

By default, Lambda will create an execution role with permissions to upload logs to Amazon CloudWatch Logs. You can customize this default role later when adding triggers.

▼ Change default execution role

Execution role

Choose a role that defines the permissions of your function. To create a custom role, go to the IAM console.

- ( ) Create a new role with basic Lambda permissions
- (●) Use an existing role
- ( ) Create a new role from AWS policy templates

Existing role

Choose an existing role that you've created to be used with this Lambda function. The role must have permission to upload logs to Amazon CloudWatch Logs.

service-role/mySimpleMicroservicpermissions

View the mySimpleMicroservicpermissions role on the IAM console.

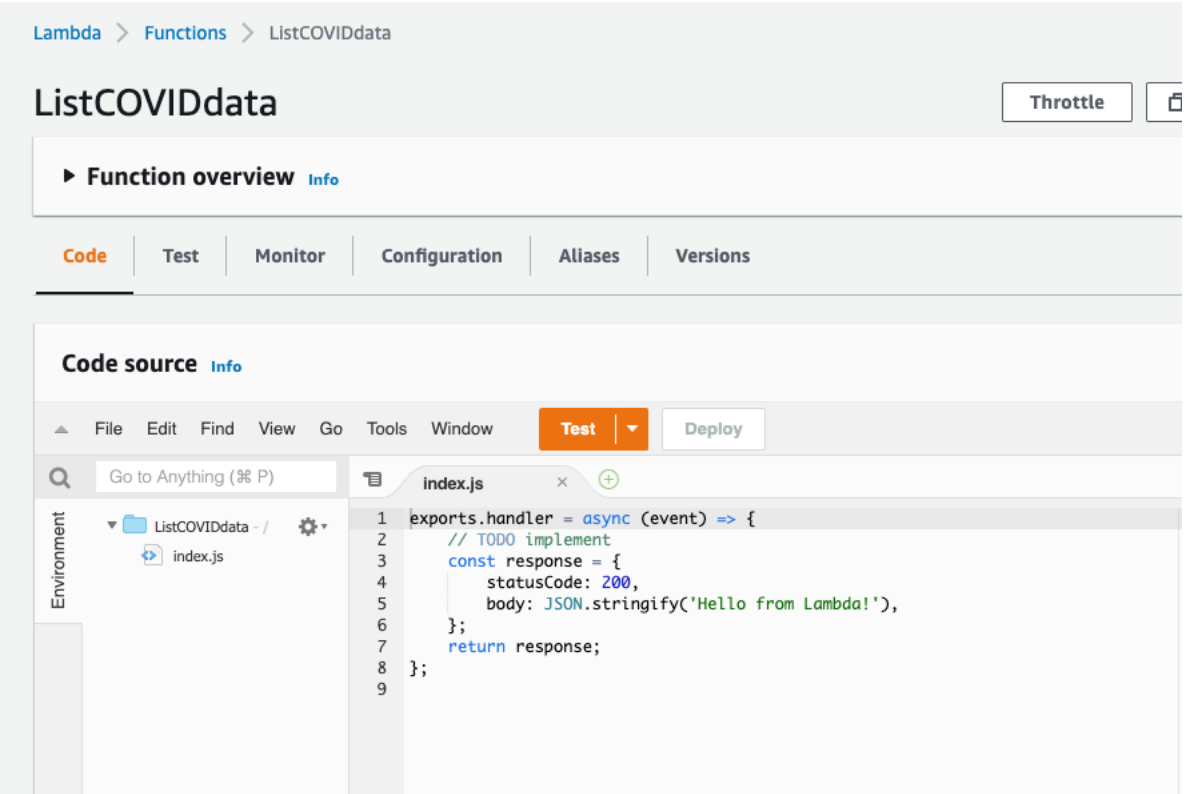
▶ Advanced settings

Cancel | Create function

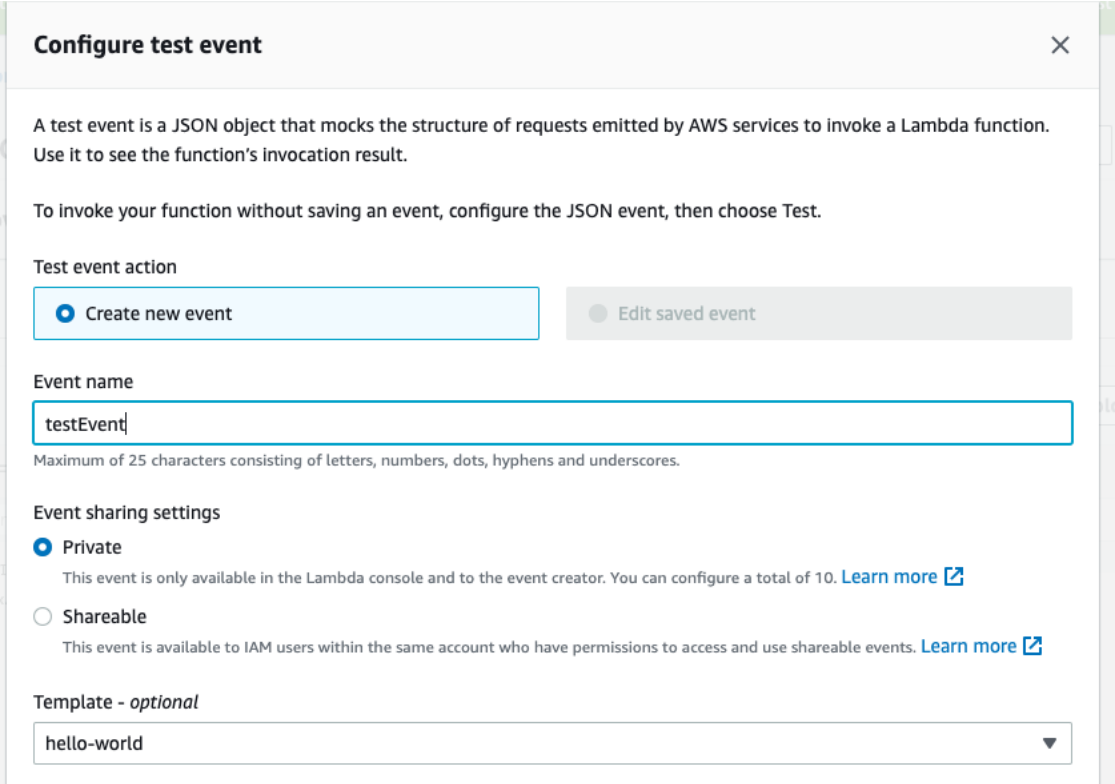
Note in my case I went with an existing role. There's a section on setting up roles at the end of the document. I will show this later.

Click on Create Function

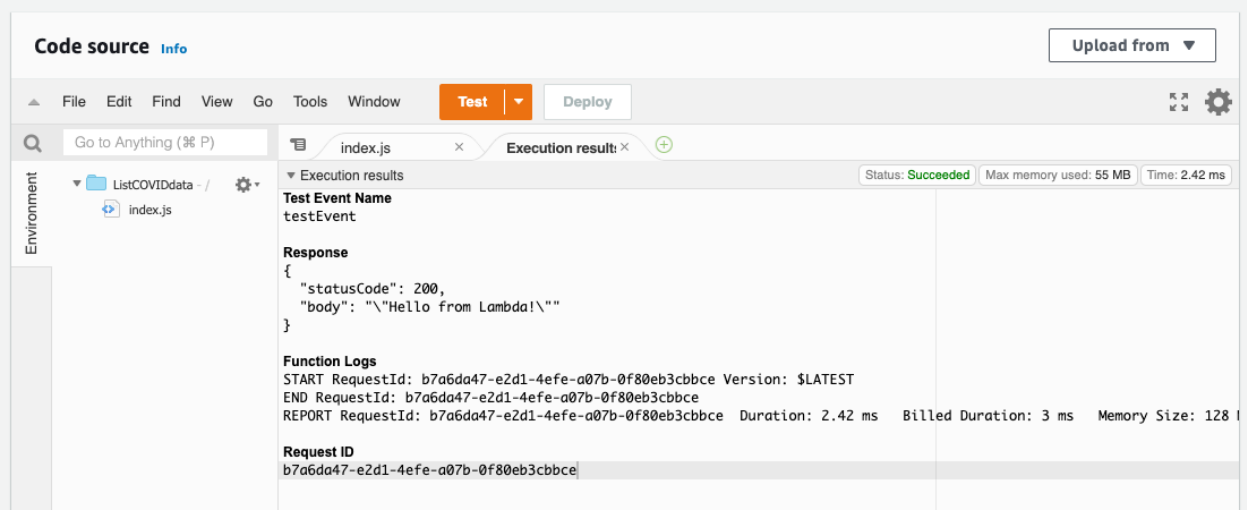
This is what you get:



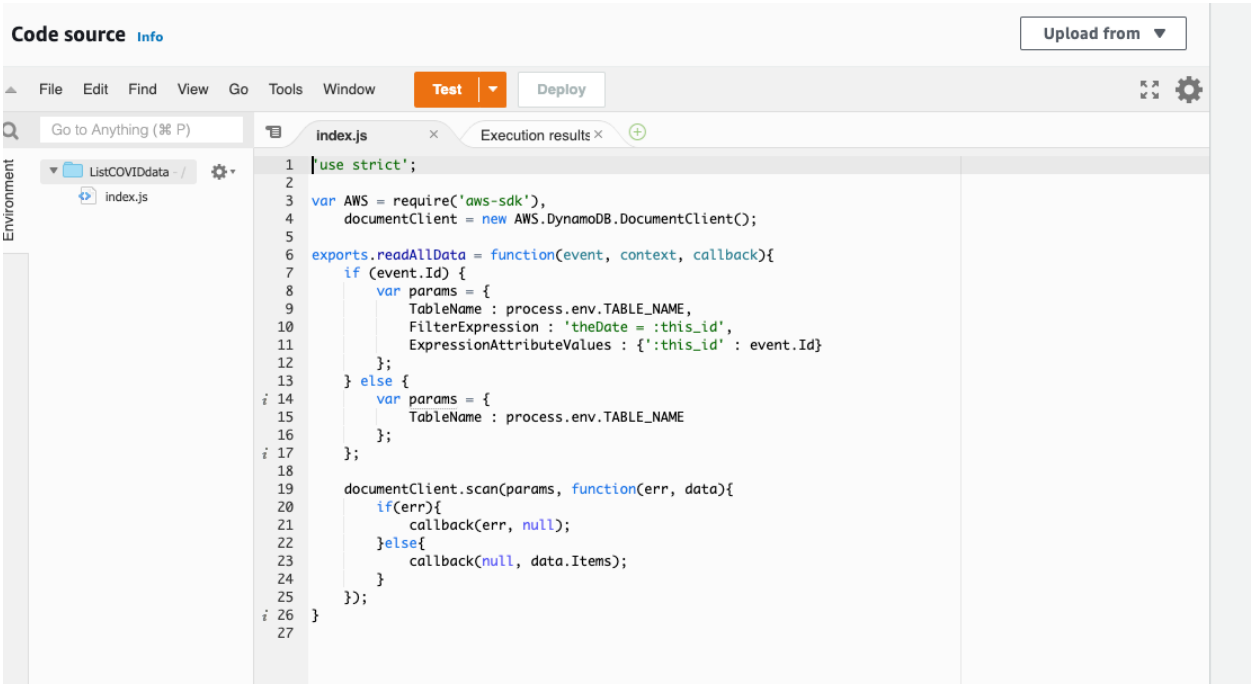
Click Test to Test it. You will have to fill information on the Test Event:



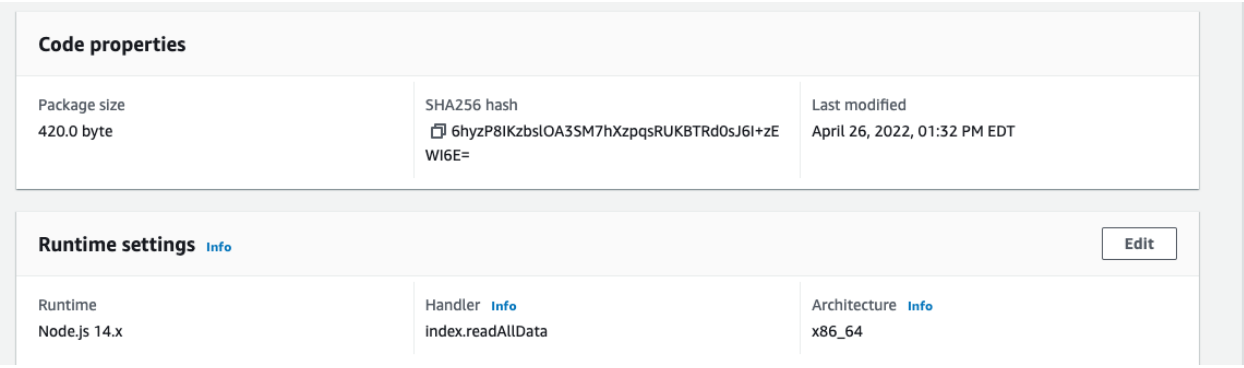
And save it. Then run Test again. You should get this.



You now want to replace the code with the code below. You can find it in this repo.
After you replace the code, make sure you click Deploy

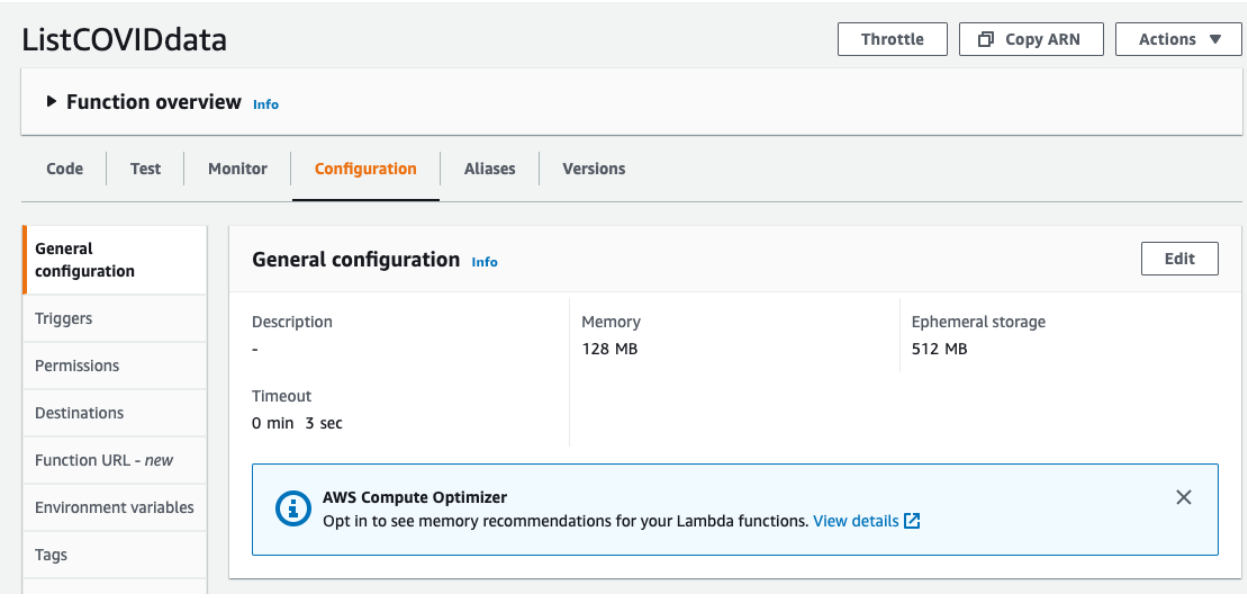


Important thing to note. See how it says “exports.readAllData”? Scroll down this screen



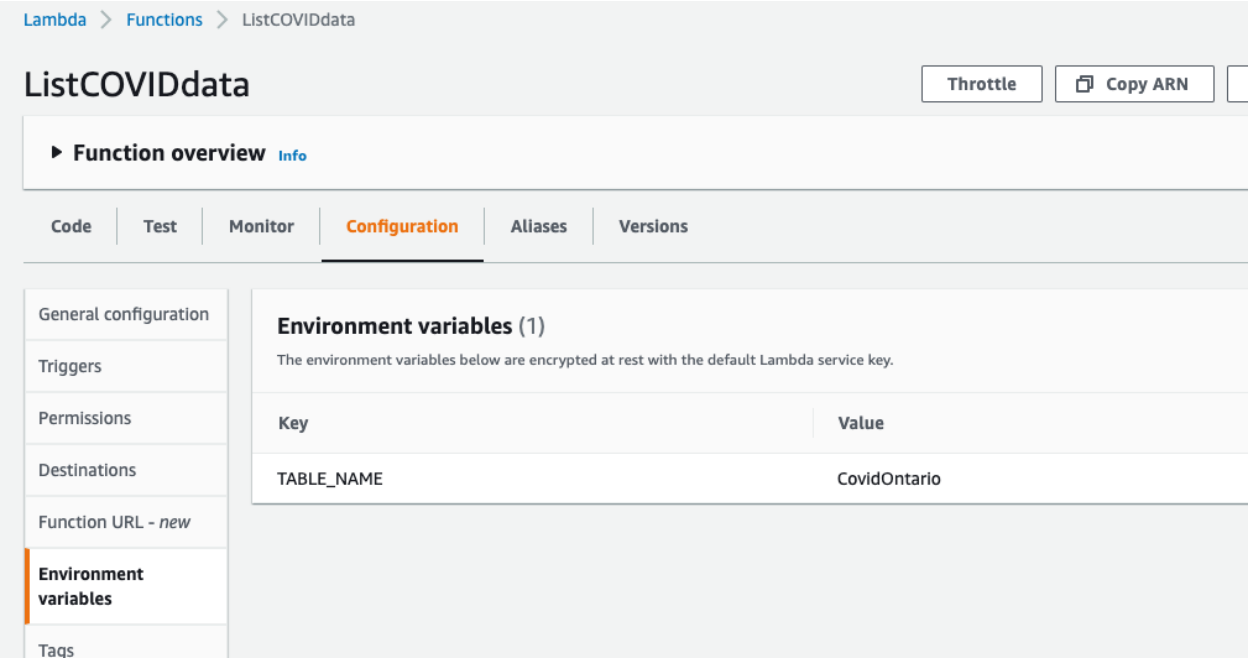
See the Handler? It needs to read: index.readAllData

Now you need to change the configuration of the function. Right now the Description is blank.



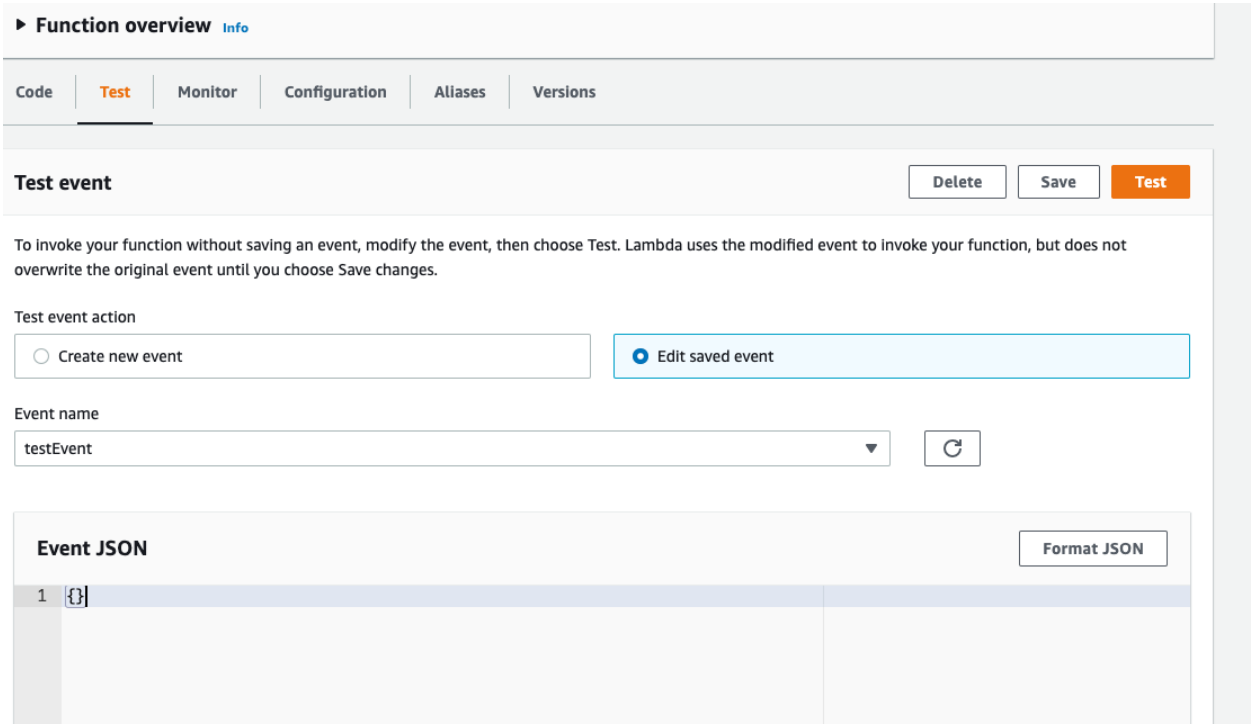
Click on the Edit button and change the Description and Save it.

Then go to Environment variables and edit them so it looks like this

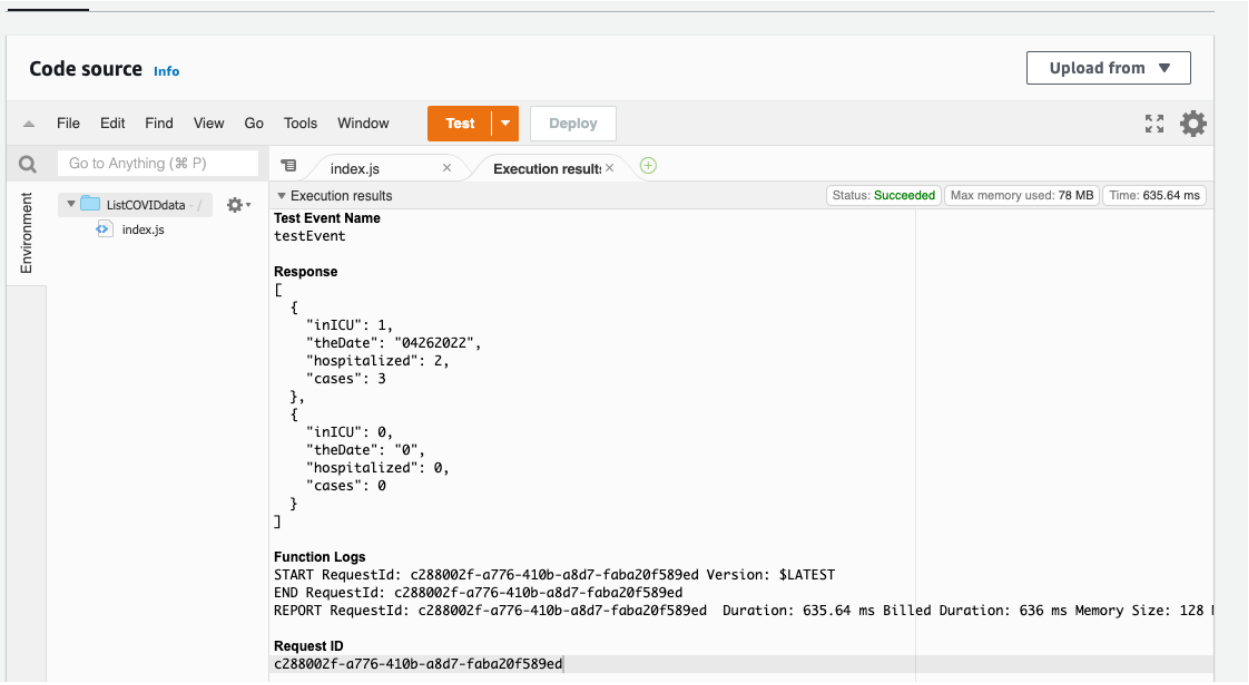


The case is very important here for both the Key and the Value.

Change the test to this. You can see the Event JSON is simply {} and the event name is testEvent.



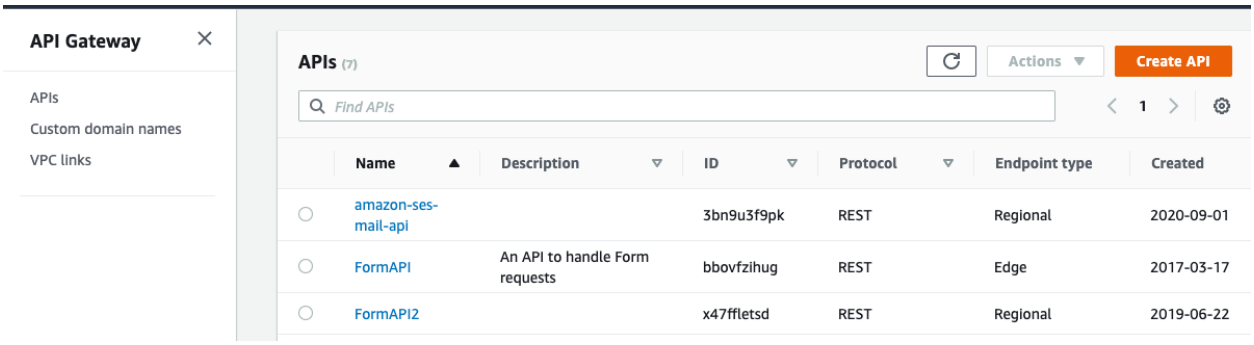
Now run the test. You should see get back a JSON response with all the data in the CovidOntario table. Nice!



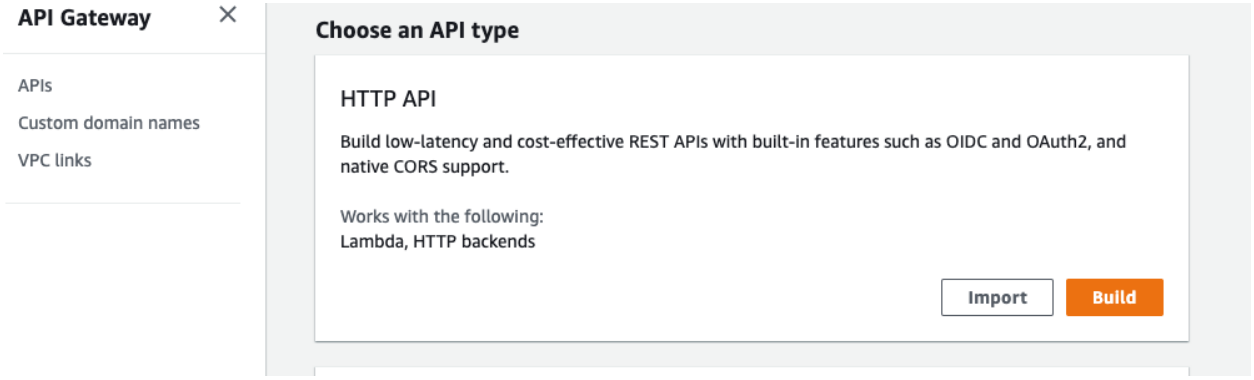
Third, create an API to call the function. Go here:

To create an API, go here:
https://us-east-1.console.aws.amazon.com/apigateway/main/apis?region=us-east-1

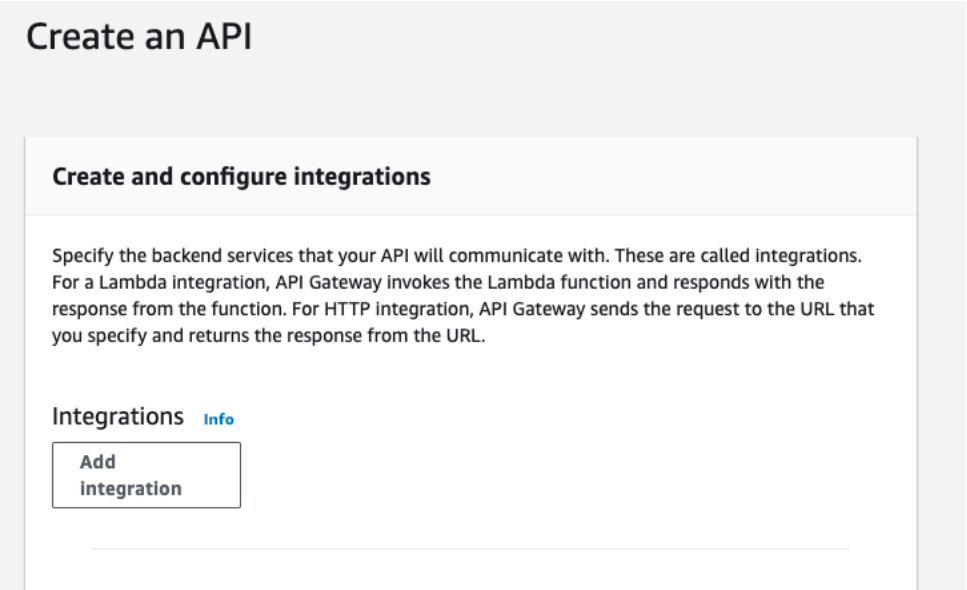
Click on the button Create API



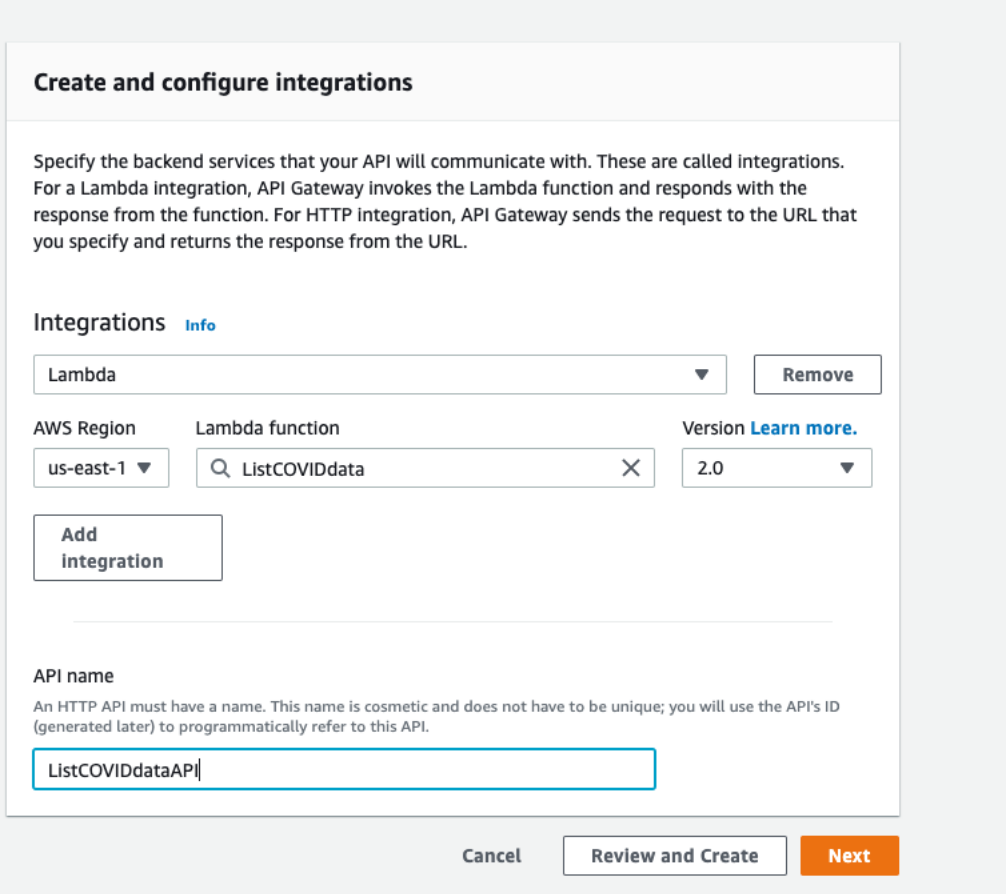
Let's create a simple one for now with a type of HTTP API:



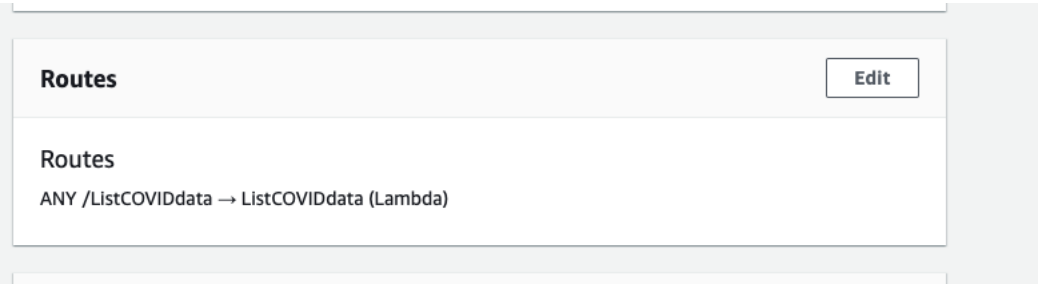
Click on button, Add integration:



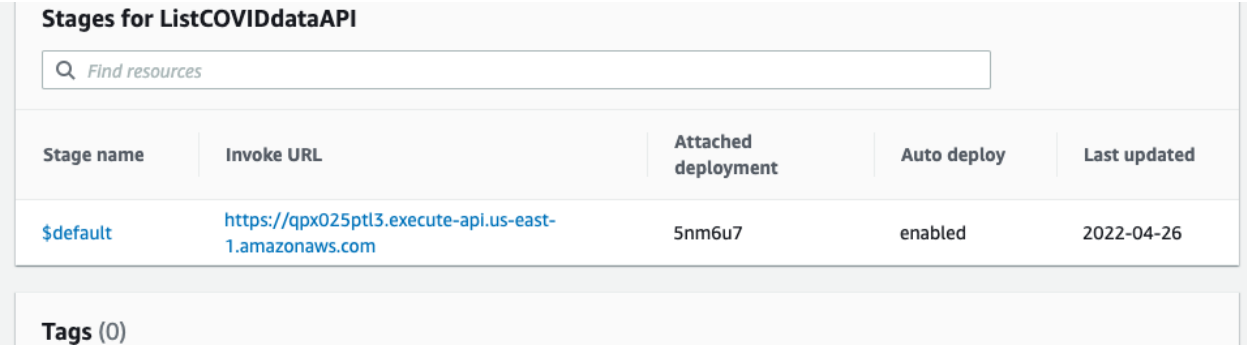
Fille out the data. You want to link to your lambda function, as follows:



Click Review and Create. Note the Route. You will need that later to call the Lambda function via the API.



and then Create then Deploy you API:



Note the invoke URL.

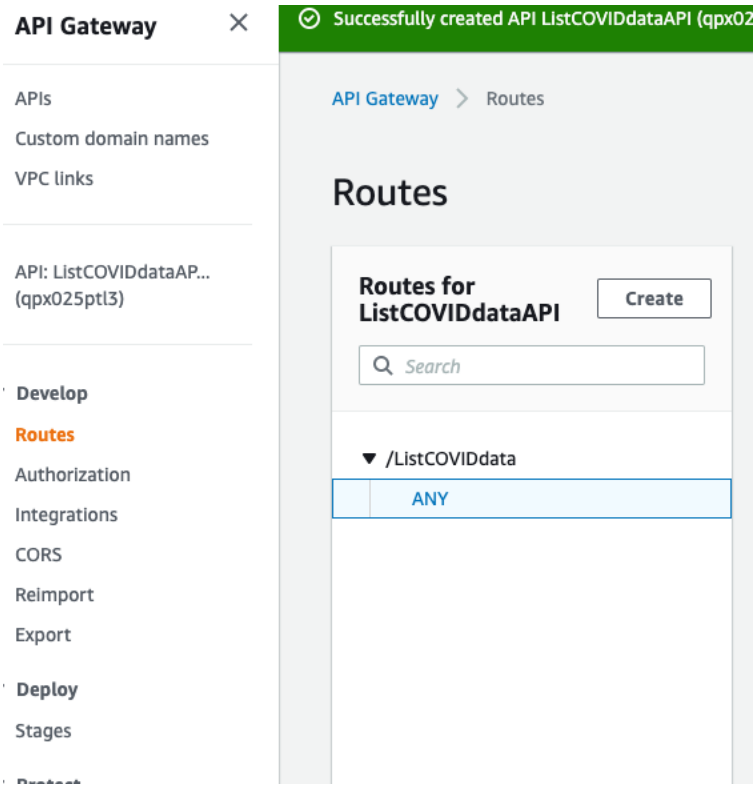
If you combine the invoke URL with the route, you can access the function. For example, you can curl it like this:

```
berniemichalik@Bernies-MacBook-Air charts % curl https://qpx025ptl3.execute-api.us-east-1.amazonaws.com/ListCOVIDdata
[{"inICU":1,"theDate":"04262022","hospitalized":2,"cases":3},{"inICU":0,"theDate":"0","hospitalized":0,"cases":0}]%
berniemichalik@Bernies-MacBook-Air charts %
```

Or you can type that into your browser

You will get back the data in the DynamoDB table in the form of JSON output.

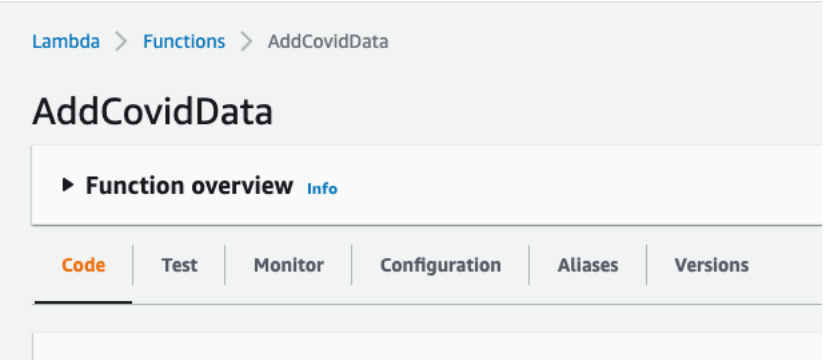
If you forgot the route, you could go here and check it.



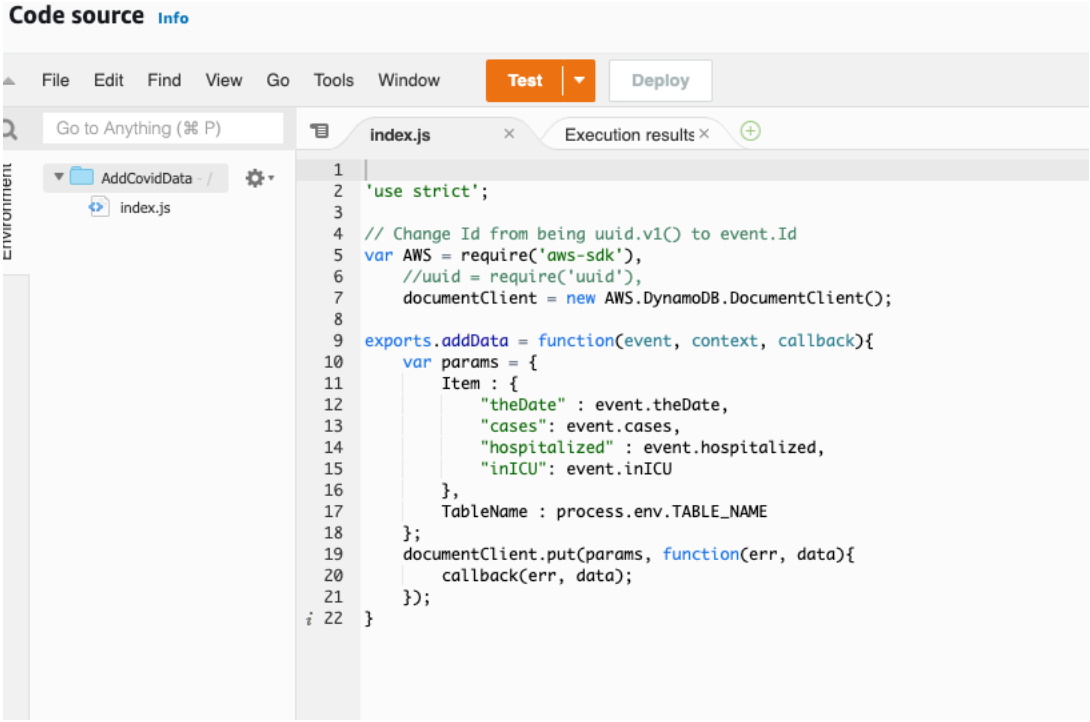
Create a second function to add data to the database

Now that you can list data in the database via our API, you may want to a function to add to it. Here's how you do that.

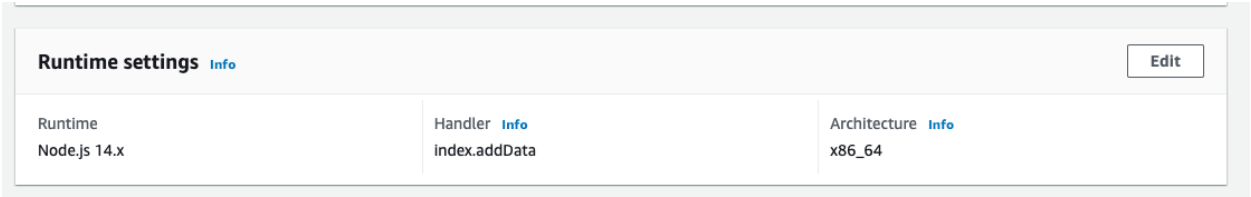
Here is the function name you will use for this example:



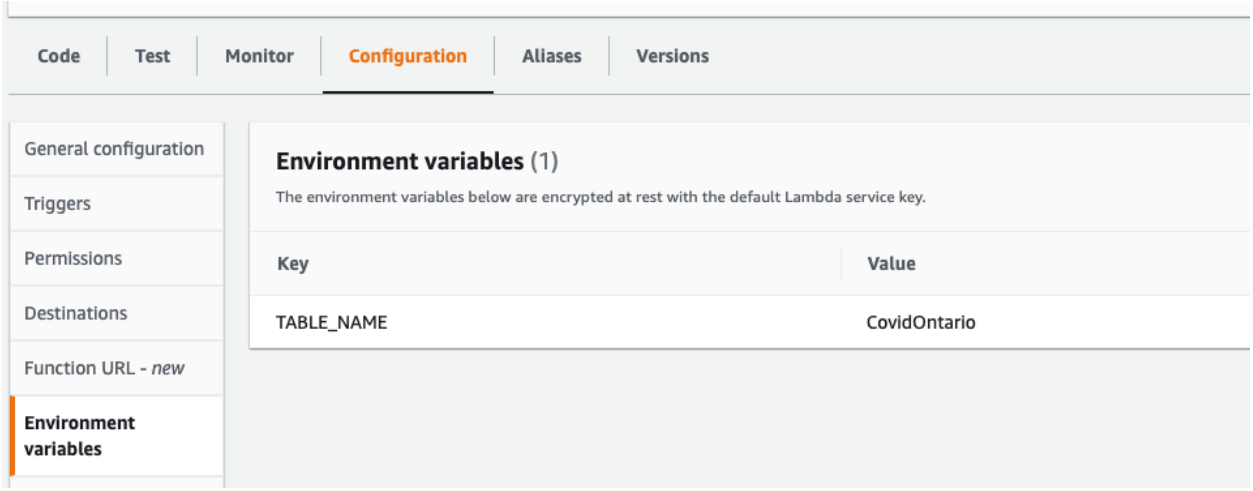
Here is the source code. Note the names of the values in Item



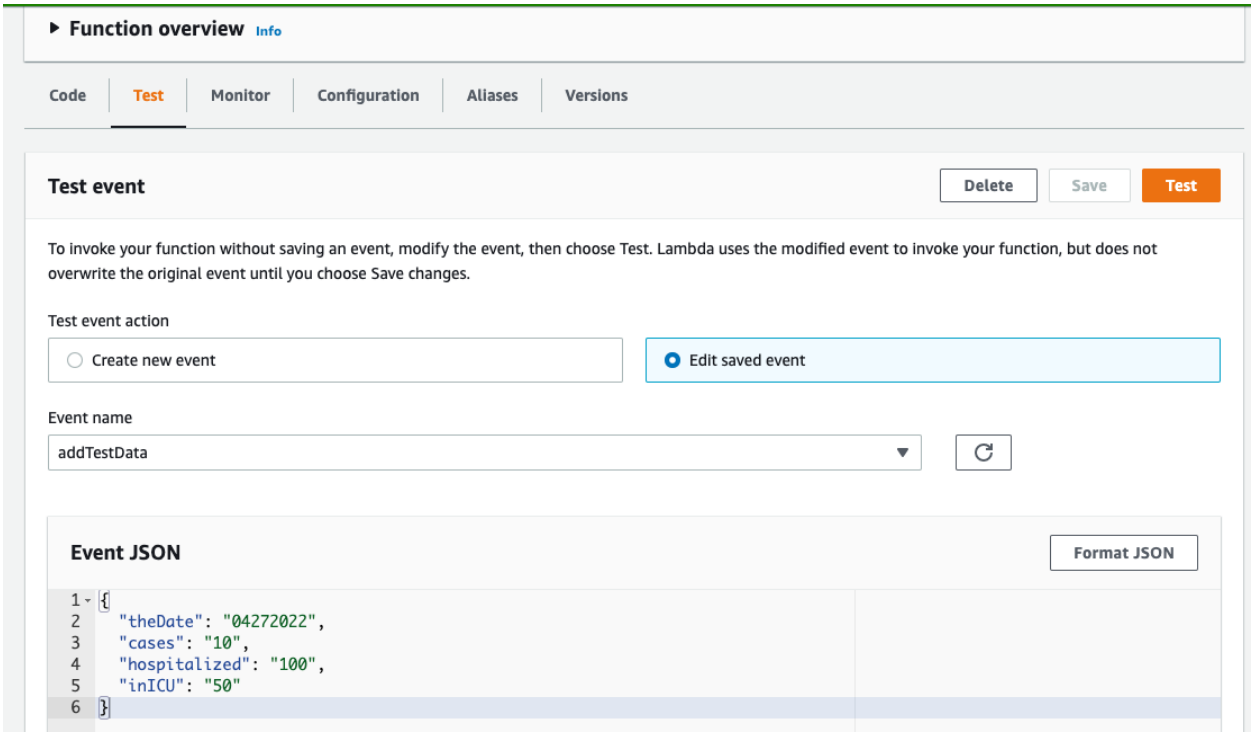
Make sure we set the handler properly:



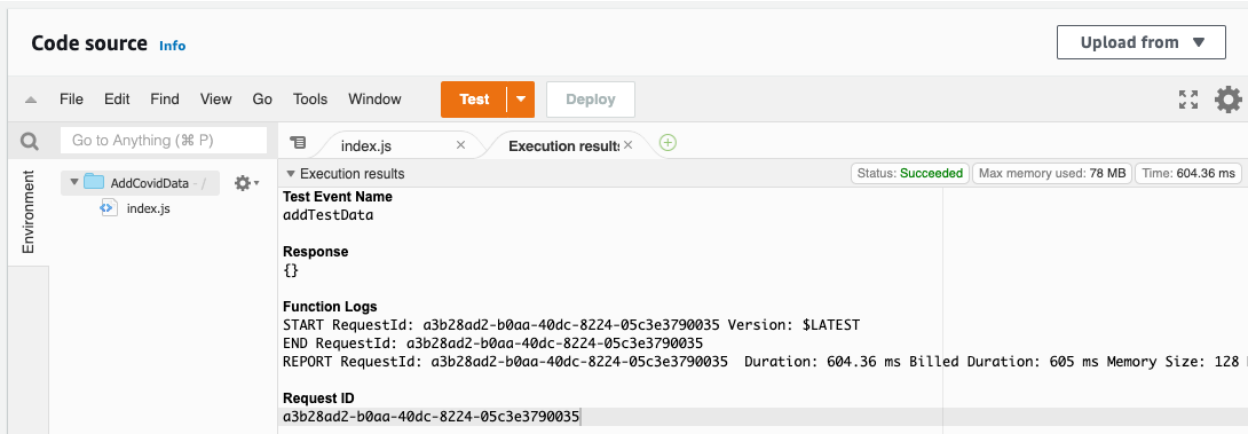
Put the table name in the environment variables:



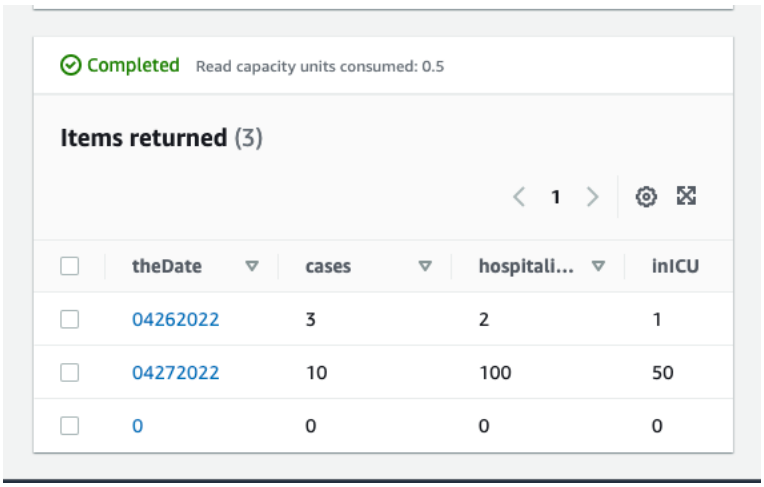
Create a test event. Note the data in the Event JSON. It's no longer {}.



Run the test:



Voila! The test JSON data that you used above is now in the table:

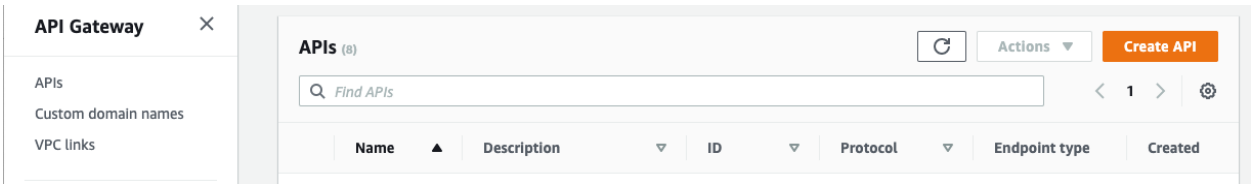


If we curl the data, we now get the new information:

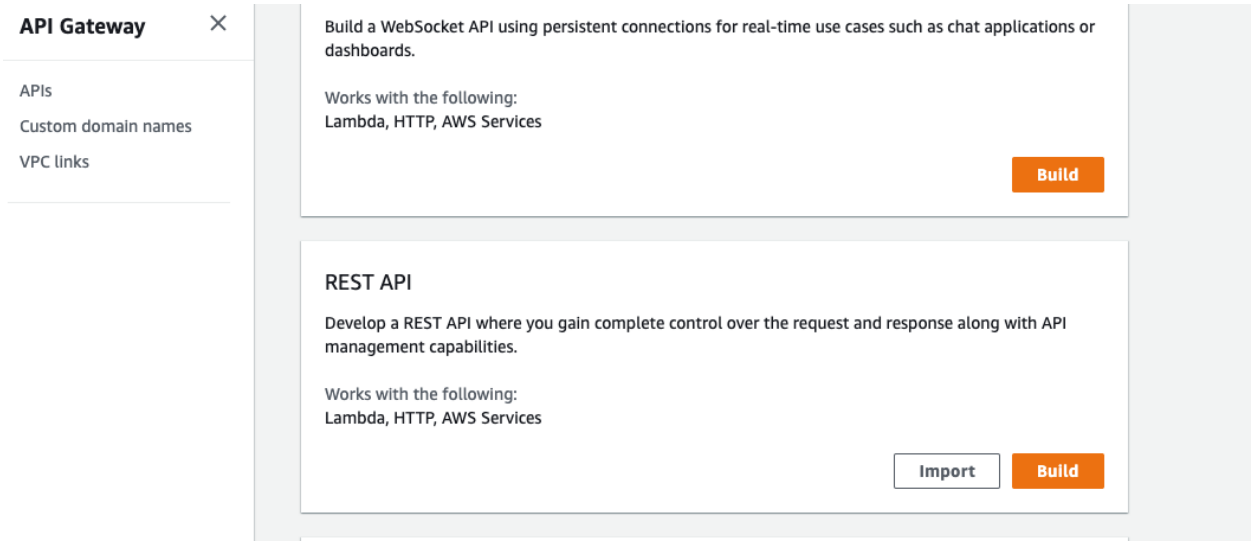
```
berniemichalik@Bernies-MacBook-Air charts % curl https://qpx025ptl3.execute-api.us-east-1.amazonaws.com/ListCOVIDdata
[{"inICU":1,"theDate":"04262022","hospitalized":2,"cases":3},{"inICU":"50","theDate":"04272022","hospitalized":"100","cases":"10"},
{"inICU":0,"theDate":"0","hospitalized":0,"cases":0}]%
berniemichalik@Bernies-MacBook-Air charts %
```

Create a second API to invoke the second function

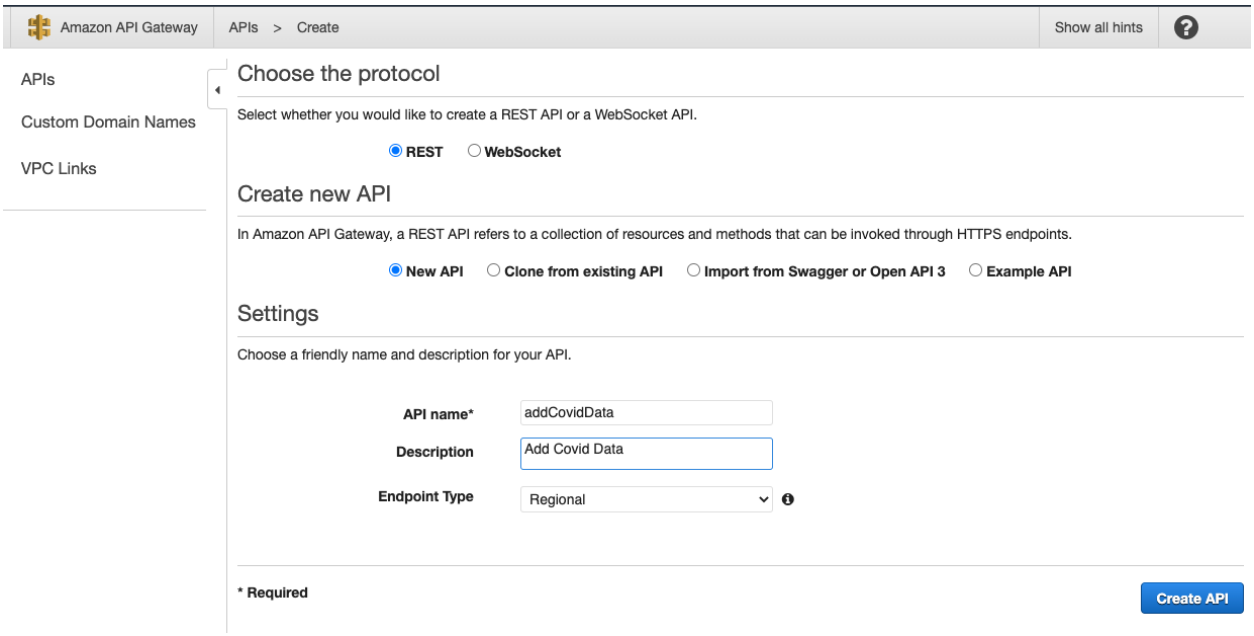
Now that we have the second function, we are going to create a new API to access it. Click Create API:



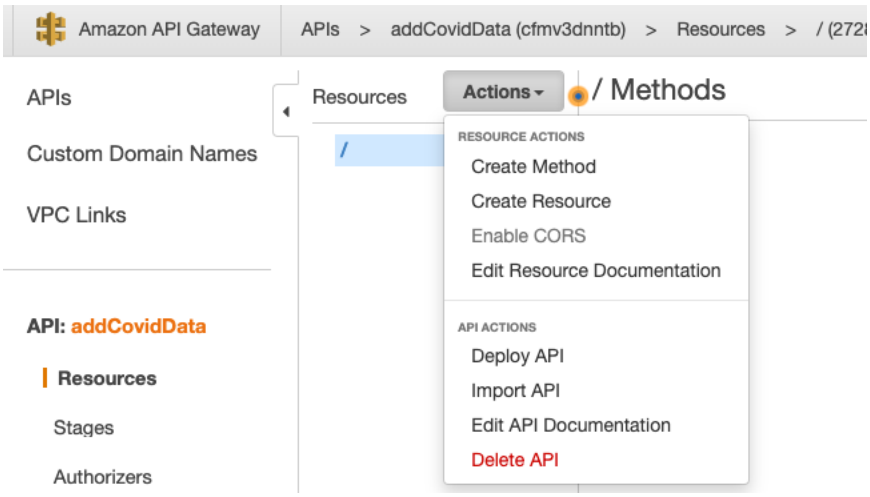
This time you are going to go with REST API, not HTTP API. Click Build in the REST API box:



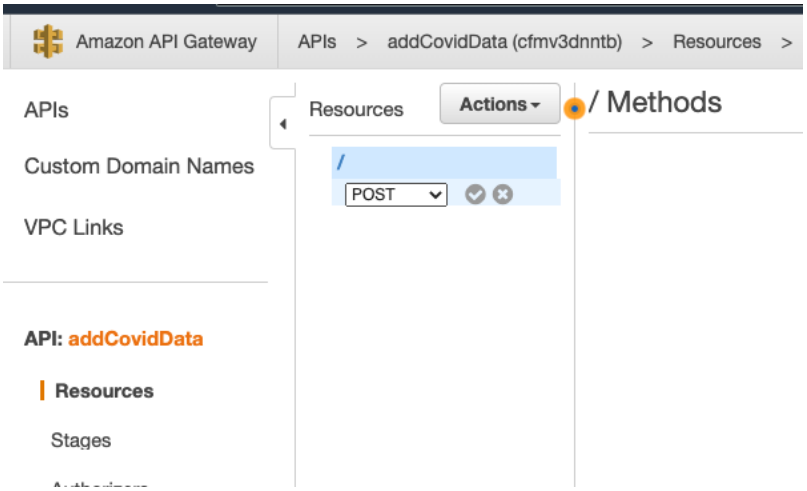
Stick with the defaults, but add the API name, description. Click Create API:



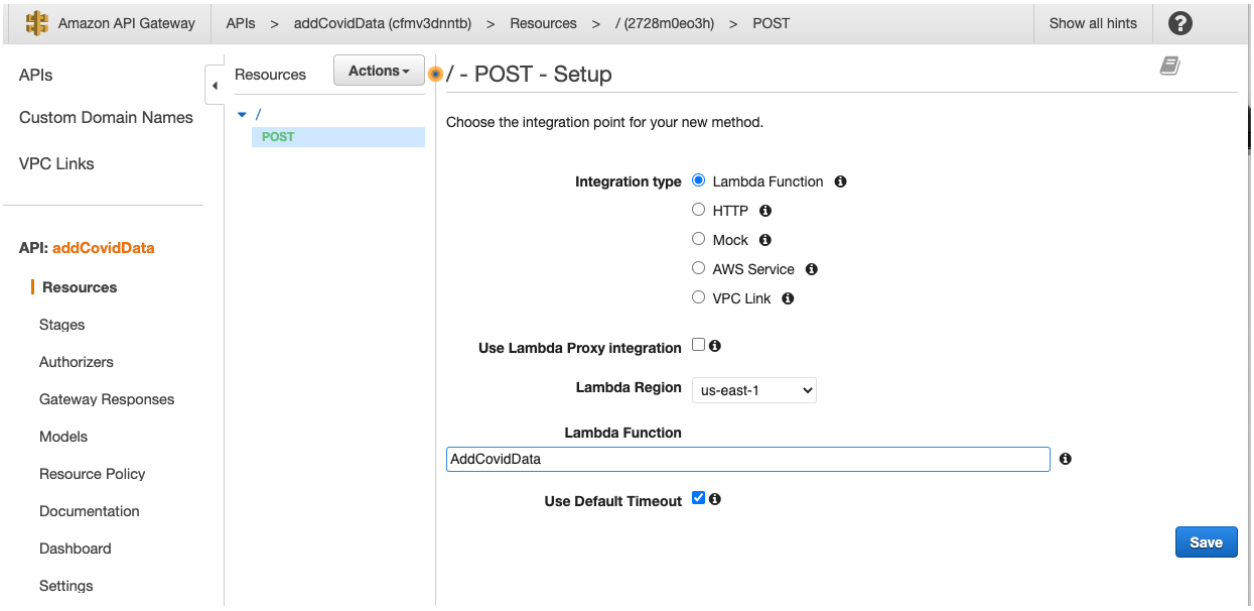
Now you are going to click on Actions > Create Method:



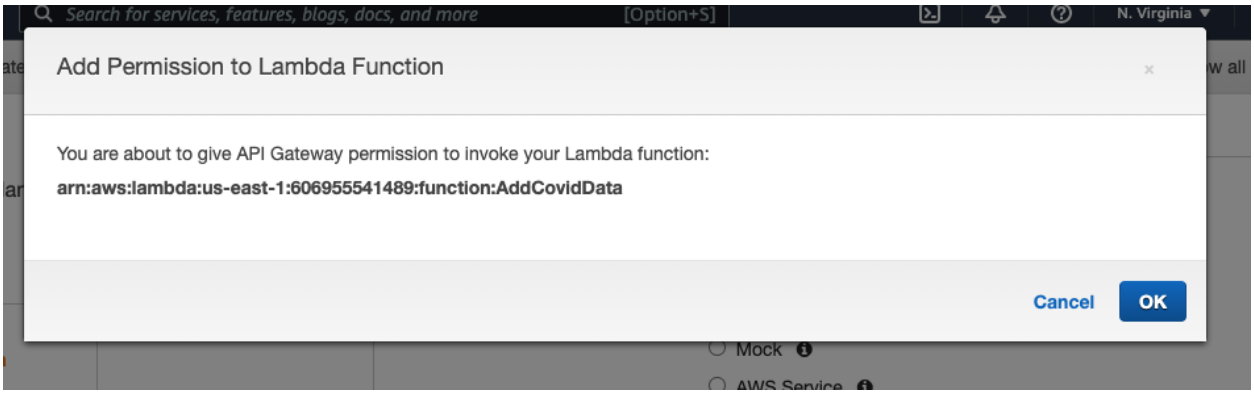
The method we want is POST



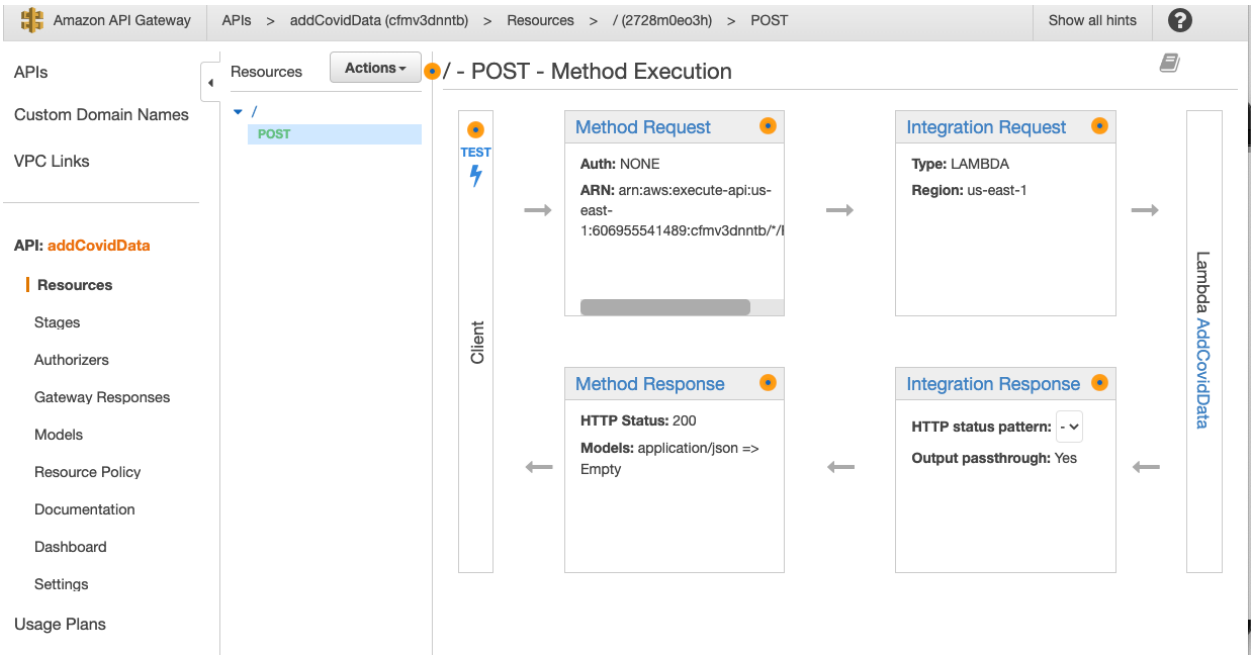
We want to integrate with Lambda Function “AddCovidData”. Make sure the values are as you see here and click Save.



Click OK for this window:



To test your API, click on Test in the box labelled Client (with the lightning bolt):



9 Scroll data and enter some data. (It's similar to the JSON data used elsewhere). Then click on Test:

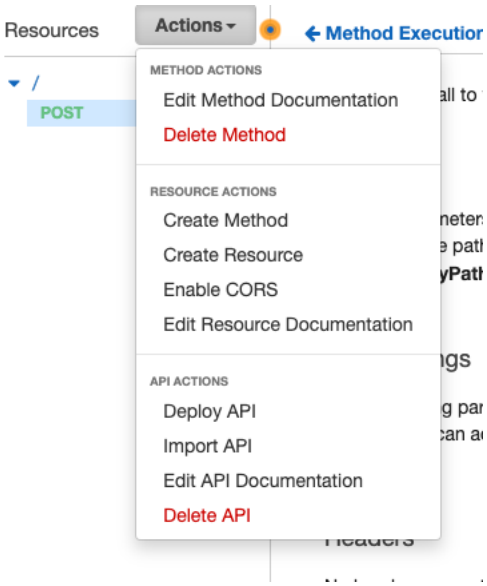
Go check the database. You should see the JSON test data above in your database:

Completed Read capacity units consumed: 0.5

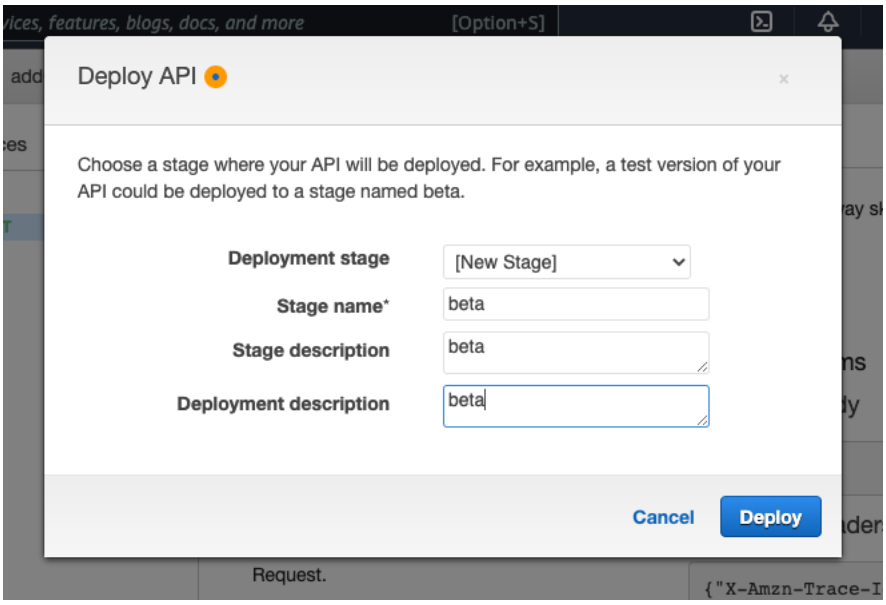
Items returned (6)

theDate	cases	hospitali...	inICU
999	99	19900	51
04262022	3	2	1
04272022	10	100	50
9	10	100	51
0	0	0	0
10	10	100	51

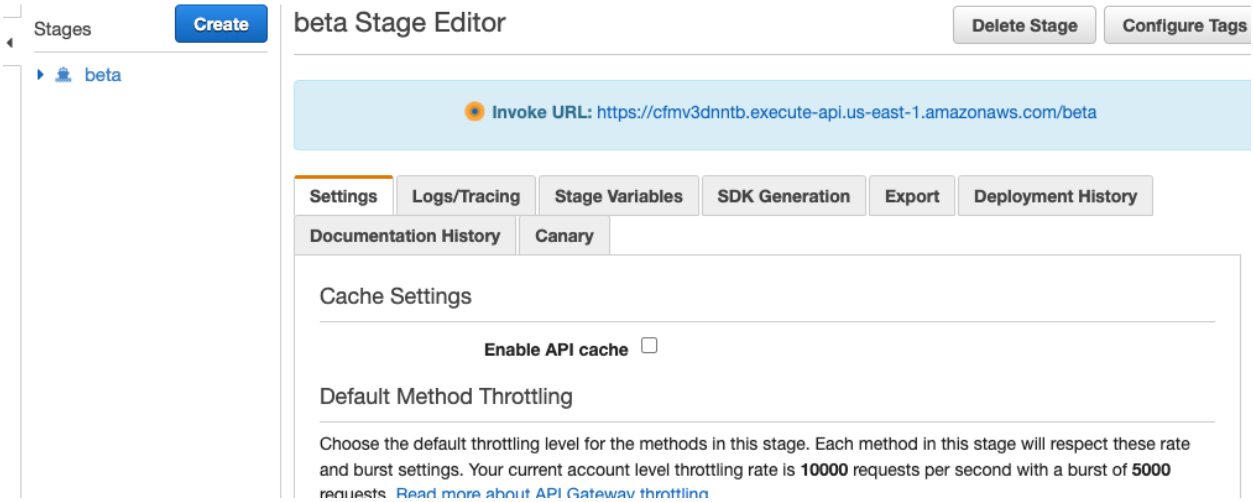
Now go back and click on Actions > Deploy API:



You need to assign a stage. It can be anything “test”, “alpha”, “beta”, “v1” etc. I went with beta:



After I clicked on Deploy, I got this. Copy the Invoke URL:



Now you want to make this curl command. Note the data we are going to pass: it's the JSON data you've used before. Also note the invoke URL with the "beta" in it:

```
curl -X POST -H "Content-Type: application/json"
-d  '{"theDate": "11", "cases": "11", "hospitalized": "11","inICU": "11"}'
https://cfmv3dnntb.execute-api.us-east-1.amazonaws.com/beta
```

Make the curl command a single line and run it. You get this!

```
berniemichalik@Bernies-MacBook-Air charts % curl -X POST -H "Content-Type: application/json" -d  '{"theDate": "11", "cases": "11", 
"hospitalized": "11","inICU": "11"}'  https://cfmv3dnntb.execute-api.us-east-1.amazonaws.com/beta
{}%
berniemichalik@Bernies-MacBook-Air charts %
```

Did it work? Check the database: at the bottom is your new record.

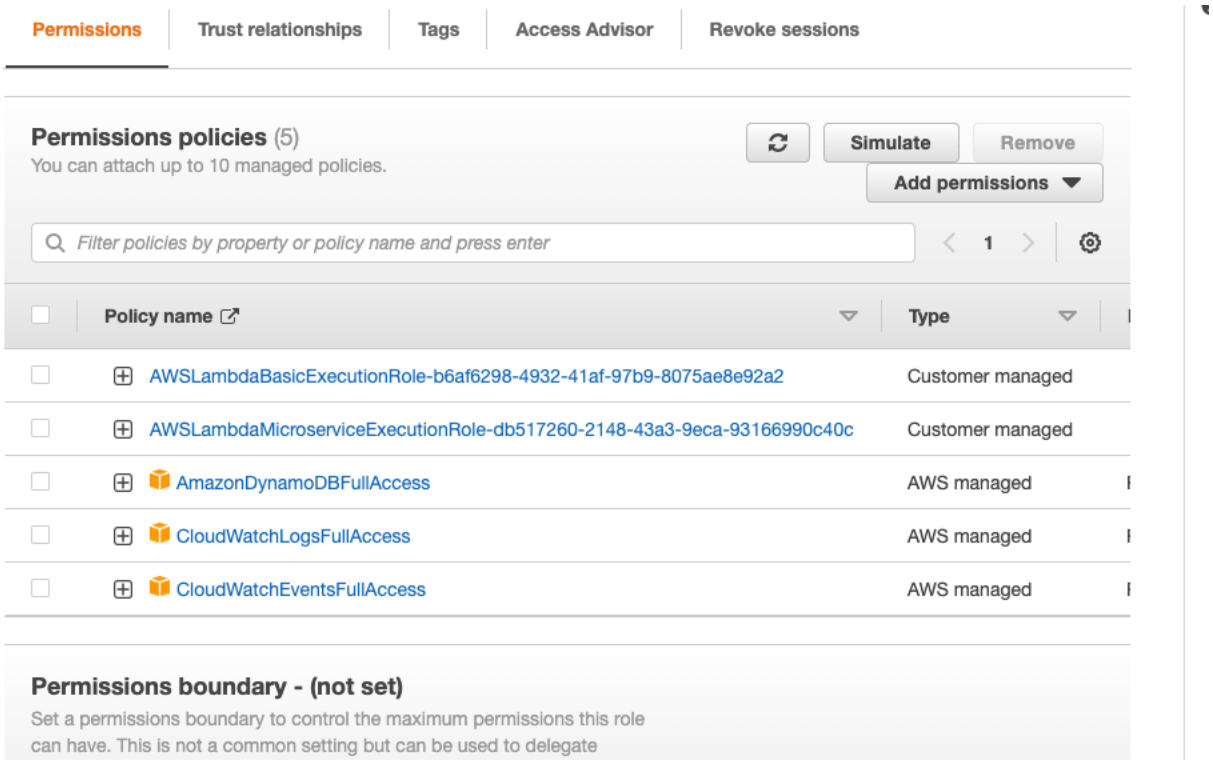
Items returned (7)

1

theDate	cases	hospitali...	inICU
999	99	19900	51
04262022	3	2	1
04272022	10	100	50
9	10	100	51
0	0	0	0
11	11	11	11

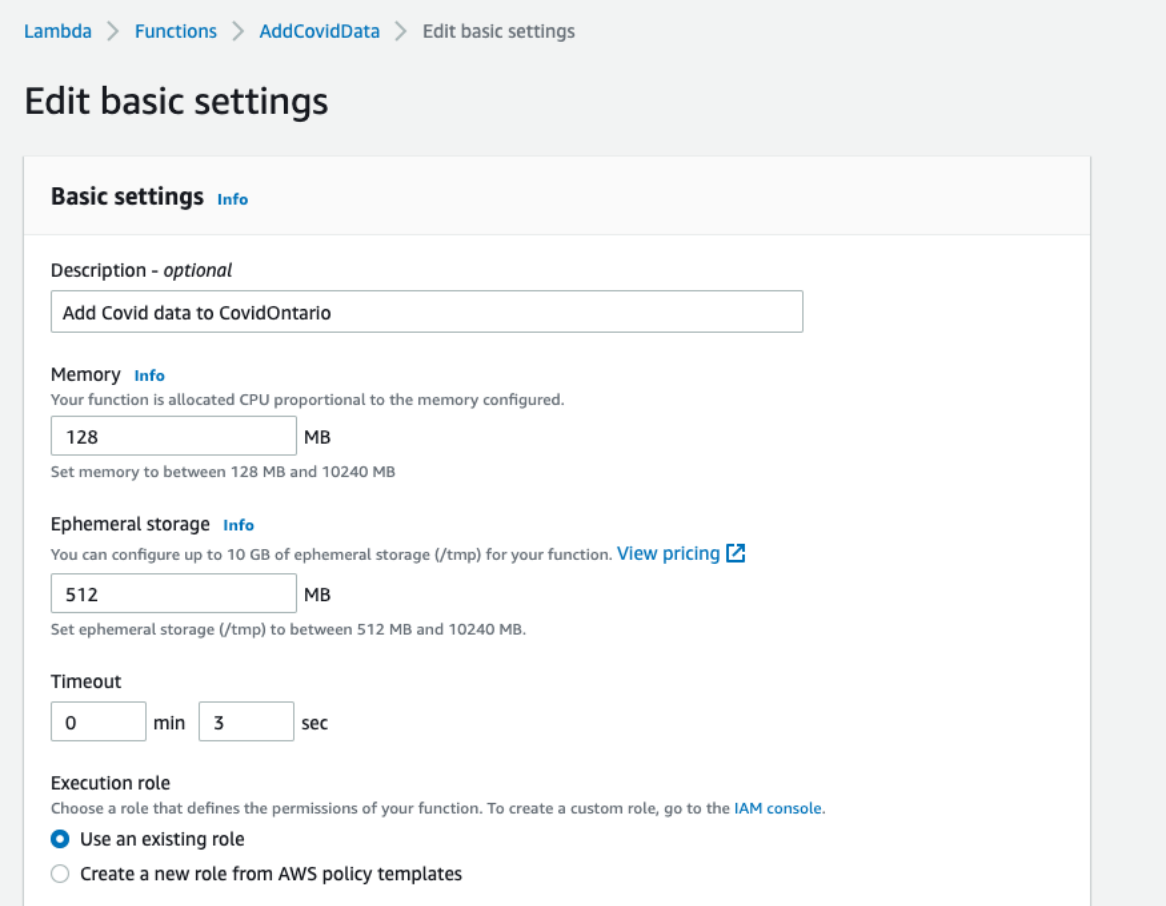
Roles

Most of my Lambda functions work with a role I created with the name: **mySimpleMicroservicpermissions** (I know, it has a typo). It looks like this:

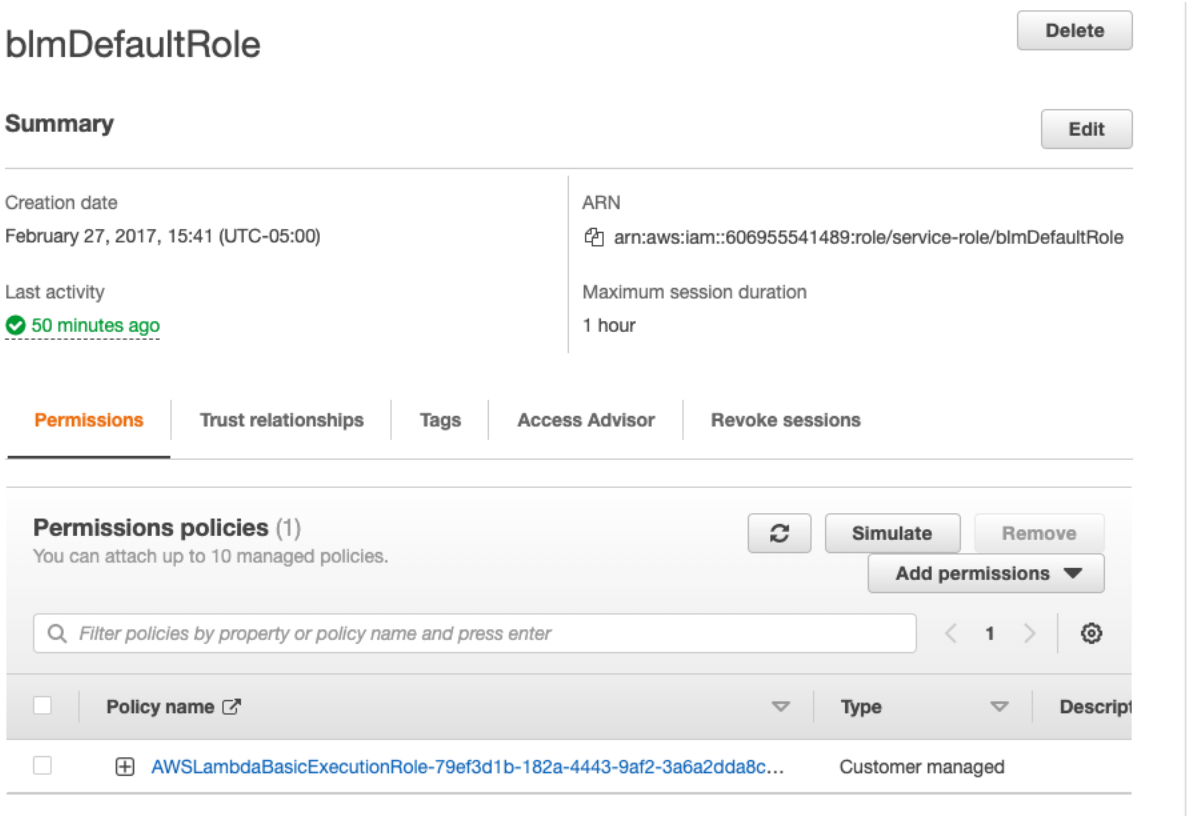


The first two items in the Policy name list were autogenerated. The third I added to write to the DynamoDB tables. And the last two were likely for some logging I was doing. It does the job.

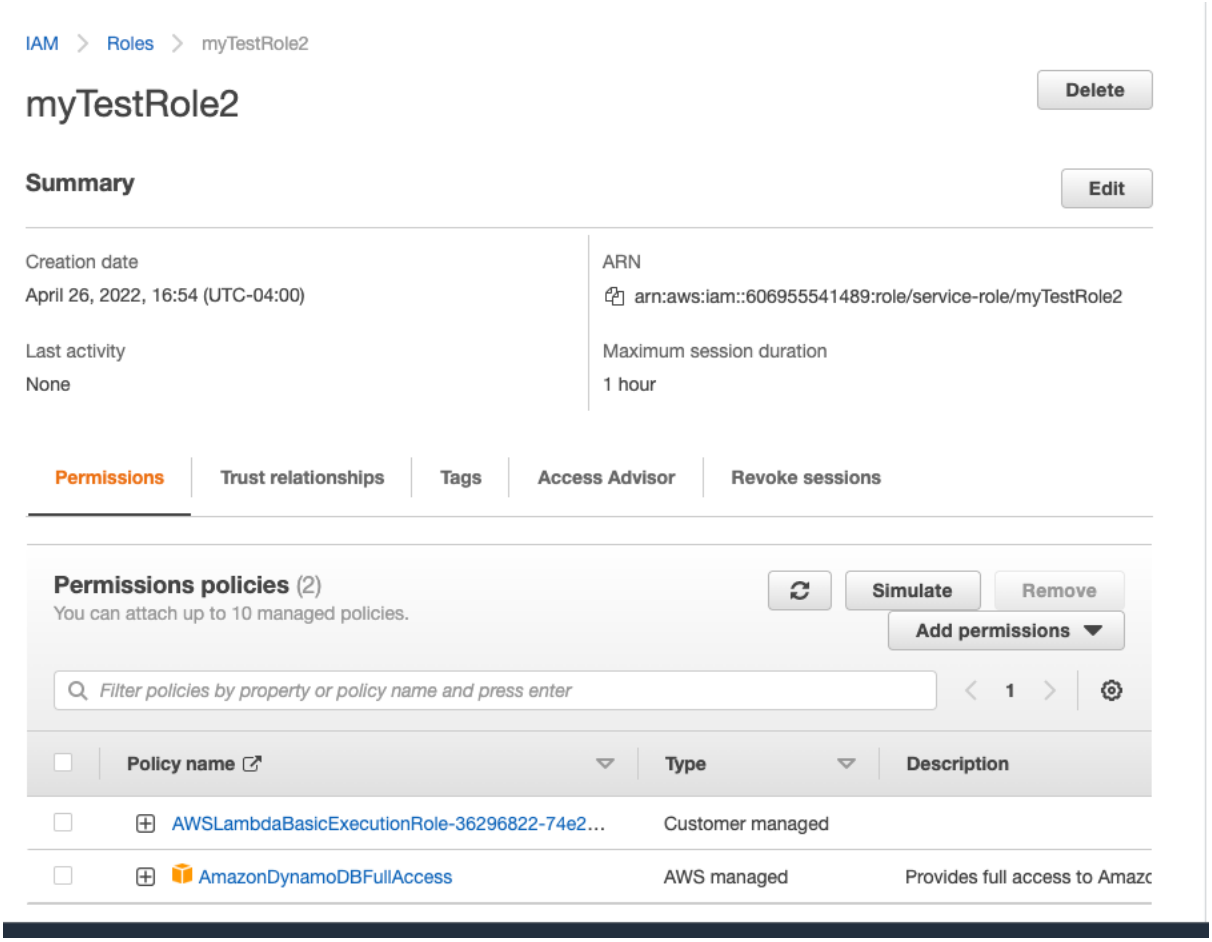
When I create a Lambda function, it will ask me for a role. I use that one. You can make a similar one before you start making lambda functions. Or you can also create your own when you make a lambda function, like this



For example, for one function I created this:



It’s fine for running Lambda functions, but I can’t get it to work with DynamoDB. So, I created this role and it works because I added an additional policy:



When it comes to creating roles, use the KISS principle. Keep the number of roles limited and the number of policies limited until your code works. I have one role that works for all my Lambda code. That may work for you, or you may need more.