AWS Lambda—the Basics Lambda is a high-scale, provision-free serverless compute offering based on functions. It provides the cloud logic layer for your application. Lambda functions can be triggered by a variety of events that occur on AWS or on supporting third-party services. They enable you to build reactive, event-driven systems. When there are multiple, simultaneous events to respond to, Lambda simply runs more copies of the function in parallel.

Lambda functions scale precisely with the size of the workload, down to the individual request. Thus, the likelihood of having an idle server or container is extremely low. Architectures that use Lambda functions are designed to reduce wasted capacity. Lambda can be described as a type of serverless Function-as-a-Service (FaaS). FaaS is one approach to building event-driven computing systems. It relies on functions as the unit of deployment and execution. Serverless FaaS is a type of FaaS where no virtual machines or containers are present in the programming model and where the vendor provides provision-free scalability and built-in reliability.

Creating lambda functions

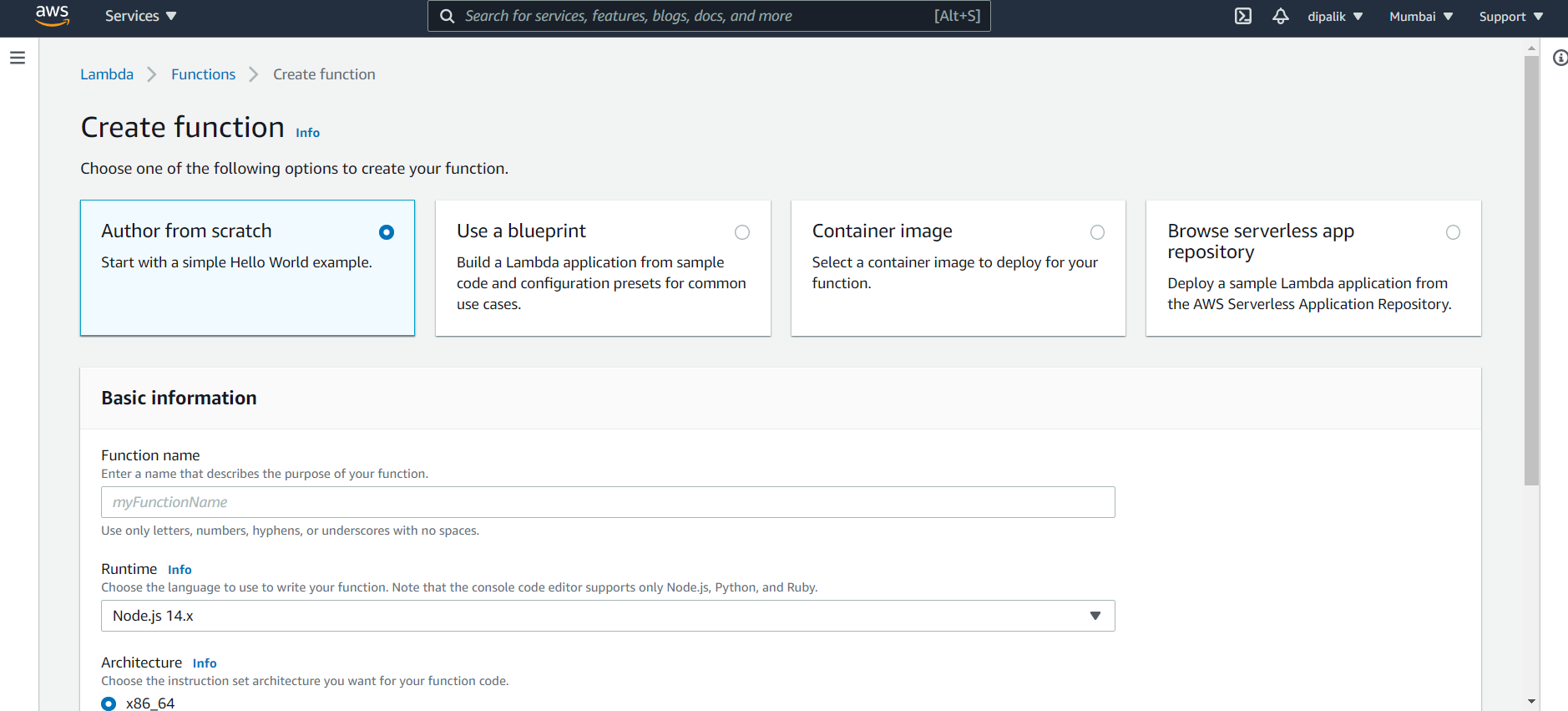
AWS Lambda is a service that allows you to run functions upon certain events, for example, when data is inserted in a DynamoDB table or when a file is uploaded to S3.

In this case, a lambda function will be run whenever a request hits one of the API endpoints you’ll set up in the next section.

We’ll create six lambda functions:

* get-all-restaurants to return all the available restaurants in the database
* get-all-menu to return menu, we can also filter menu for specific restaurant
* get-all-menu-items to show the food items available in the store, we can also filter food items based on the specific menu.
* Create-order to place the food order
* List-all-orders to display the list of orders along with their status, we can also filter orders for a specific customer

Now lets create our first Lambda function for get-all-restauranta, Go to AWS console and search for the Lambda Service, then click on Create Function button, and Select the “Author from Scratch option”



* Function Name : get-all-restaurants
* Runtime: Python 3.8 (or a superior version)
* Role: Create a custom role

Approach 1 – Choose and existing role

Create a new IAM Policy to allow permissions for Lambda to invoke the query on DynamoDB.

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow",

"Action": [

"dynamodb:DeleteItem",

"dynamodb:GetItem",

"dynamodb:PutItem",

"dynamodb:Scan",

"dynamodb:UpdateItem"

],

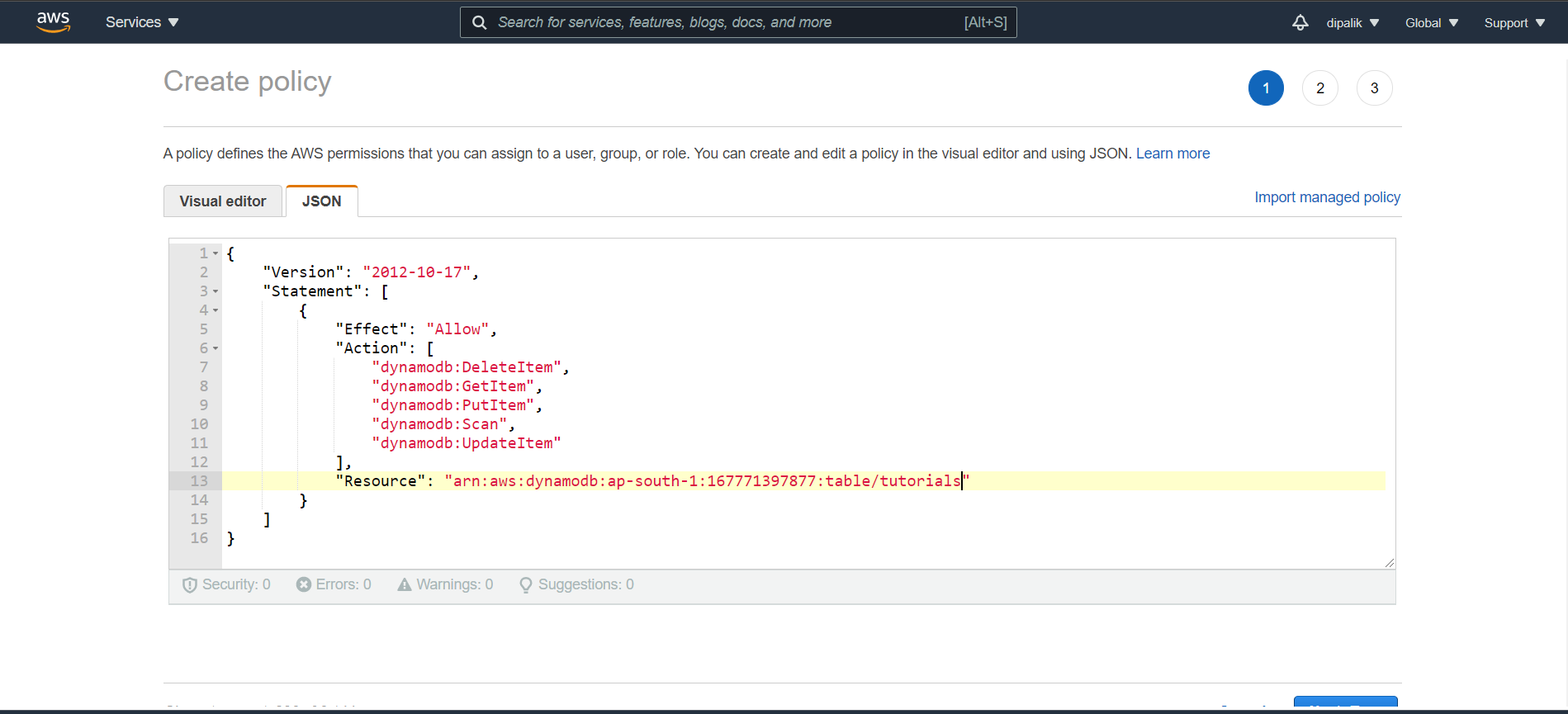
"Resource": "arn:aws:dynamodb:region:accountId:table/\*"

}

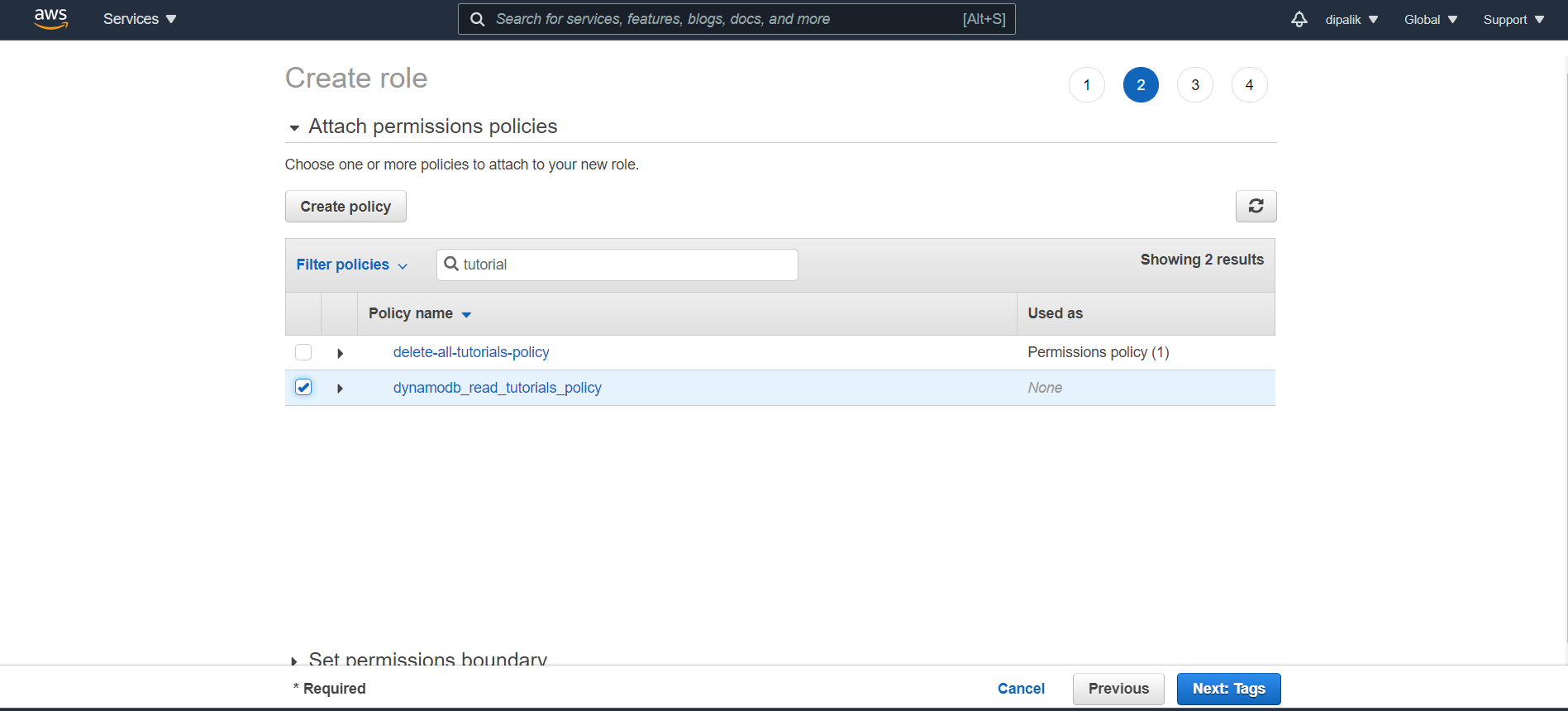
]

}

Specify the ARN of DynamoDB Table created in the previous section. It will allow access on the specific DynamoDB Table only.



Create IAM Role and assign the policy created in the above step. Also add policy to allow lambda to send logs to CloudWatch in the same role.



Approach 2 – Use the default AWS provided Roles

Alternatively, you could choose the option Create a new role from a template(s) and choose the policy template Simple Microservice permissions. This template will give you permissions to read, create, update, and delete items from any table

Enter the following Python code :

import json

import boto3

from boto3.dynamodb.conditions import Key

from boto3.dynamodb.conditions import Attr

def lambda\_handler(event, context):

print("Event json %s" % json.dumps(event))

print("Context %s" % context)

client = boto3.resource('dynamodb')

table = client.Table('foodorder')

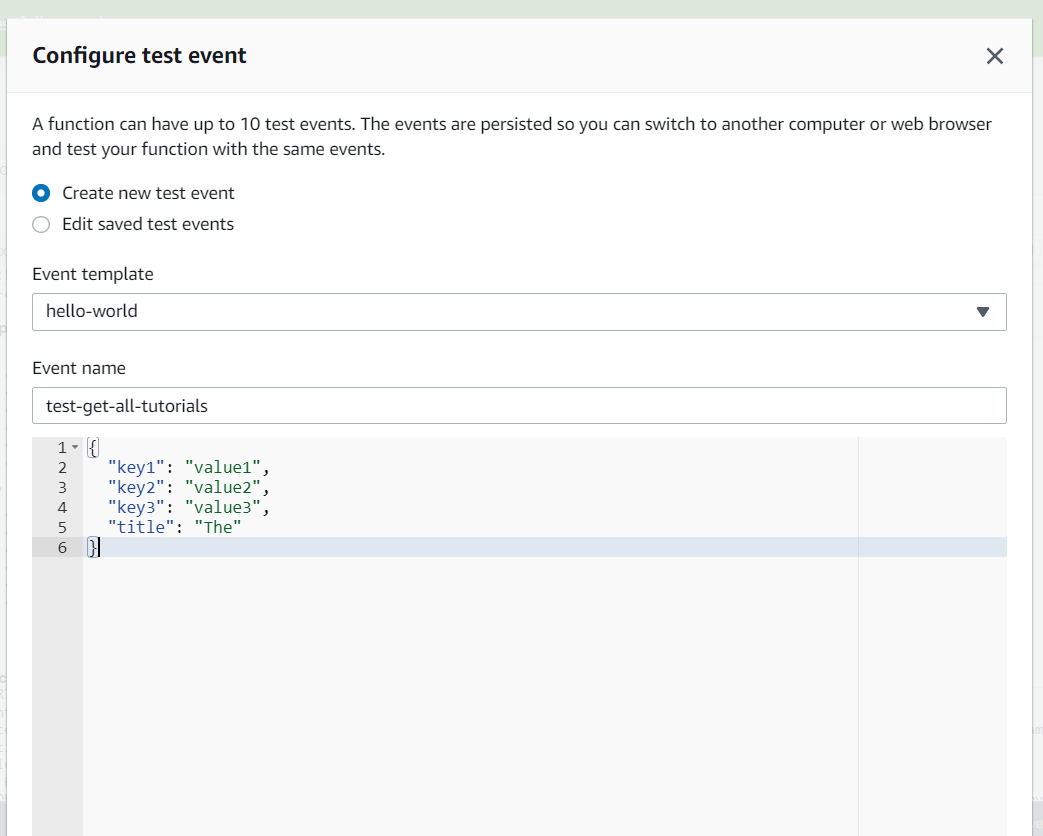
response = table.scan(

FilterExpression = Attr('Type').eq('Restaurant')

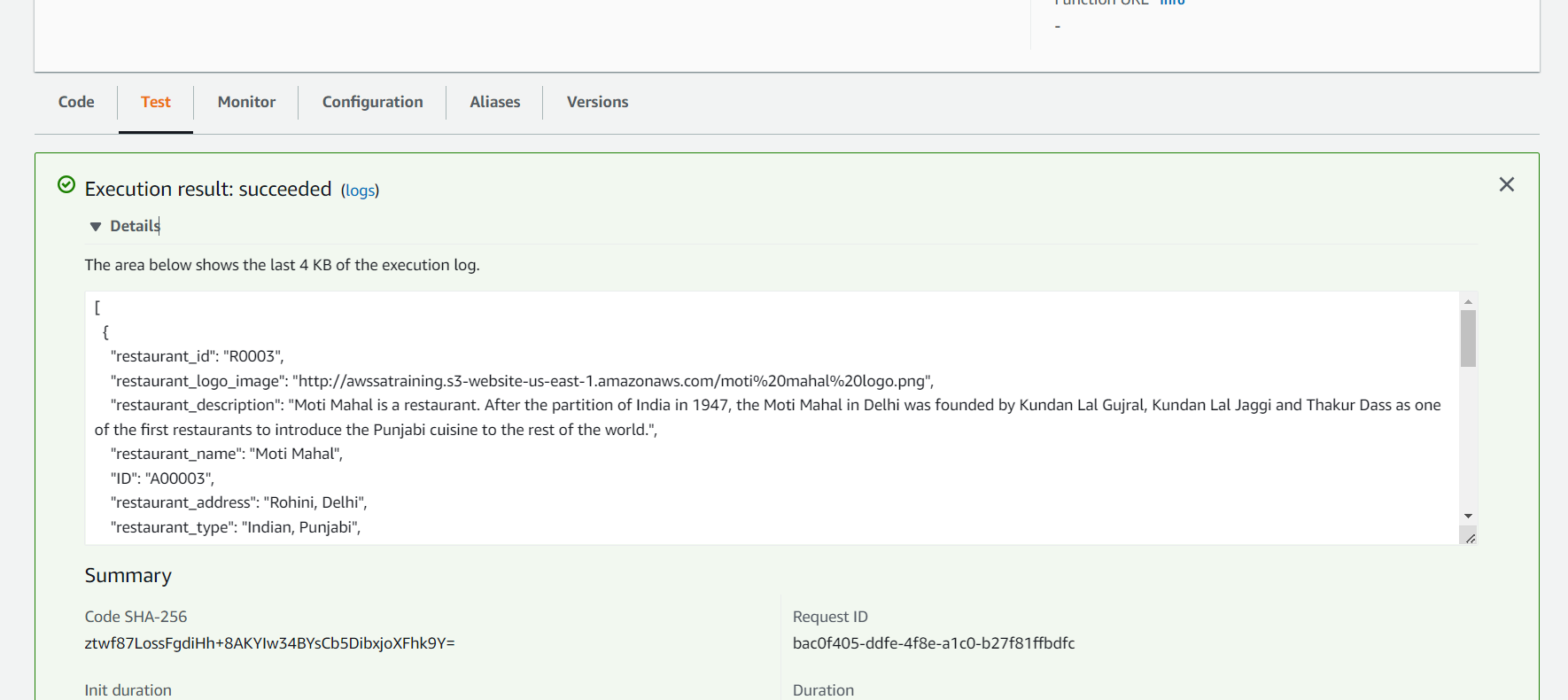
)

return response['Items']

Now lets Test the lambda function, click on the Test Button and create the test configuration, Create new test event and use hello-world as the Event Template



Save and execute the Test you should see the all-restaurants as follows



Next, we need to create the rest of the functions with their corresponding roles and policies. Of course, you can create one role with a full access policy and use it for all your functions, but remember, that is not the recommended approach. Custom roles that minimize the number of permissions are better in the long run.

You can also create the policies and roles from the Identity and Access Management (IAM) console with the visual editor, first creating a policy with required permission and then creating the role that will contain that policy.

And the code for the corresponding lambda function

* get-all\_menu to all menu, we can also filter menu for specific restaurant using restaurant\_id

import json

import boto3

from boto3.dynamodb.conditions import Key

from boto3.dynamodb.conditions import Attr

def lambda\_handler(event, context):

print("Event json %s" % json.dumps(event))

print("Context %s" % context)

client = boto3.resource('dynamodb')

table = client.Table('foodorder')

restaurant\_id = event['restaurant\_id']

print("Getting restaurant\_id Filter %s" % restaurant\_id)

if not restaurant\_id:

print("Restaurant\_id not passed, display all available Menus")

response = table.scan(

FilterExpression = Attr('Type').eq('Menu')

)

else:

print("Title is NOT empty")

response = table.scan(

FilterExpression = Attr('Type').eq('Menu') & Attr('restaurant\_id').eq(restaurant\_id)

)

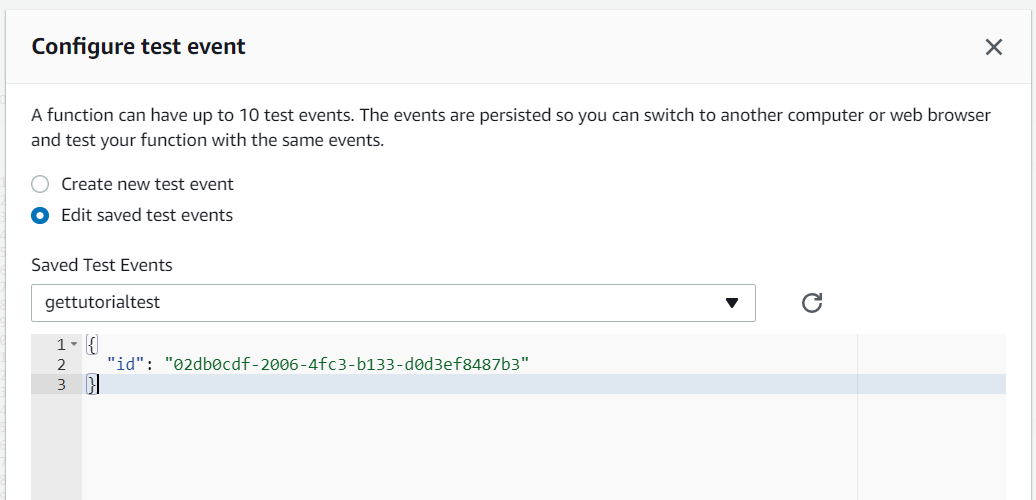
return response['Items']

Configure Test event for get-tutorial

{

"restaurant\_id": "R0001"

}



In the same way you can create all Lambda functions, refer the github location for the source code.

Now let’s expose them to the world via a API Gateway.