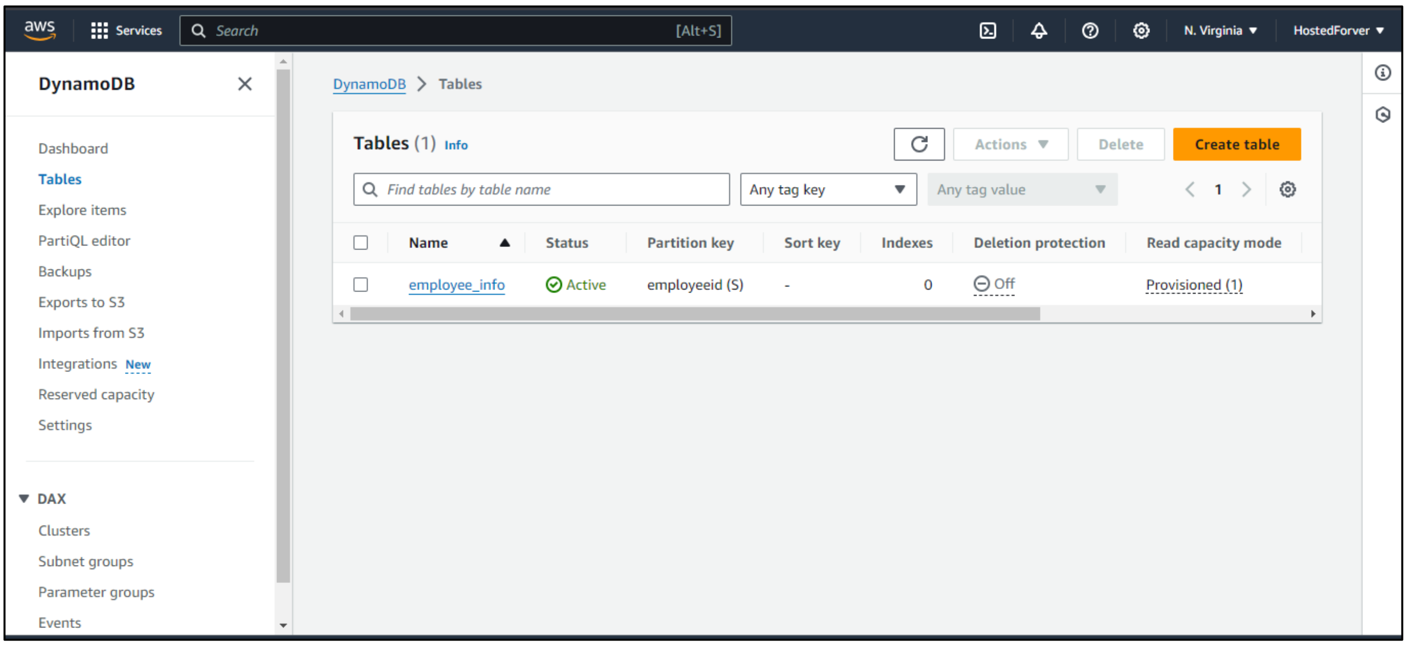
Building a REST API (with CRUD operations) using AWS Lambda, API Gateway, DynamoDB

Whole Process is divided into two parts : Part-1 : Building and connecting all services to each other
Part-2 : Executing CRUD operations on the data.

<u>Part-1 : Creating / Building and connecting all services to each other</u>

Step-1 : Creating a Table in DynamoDB

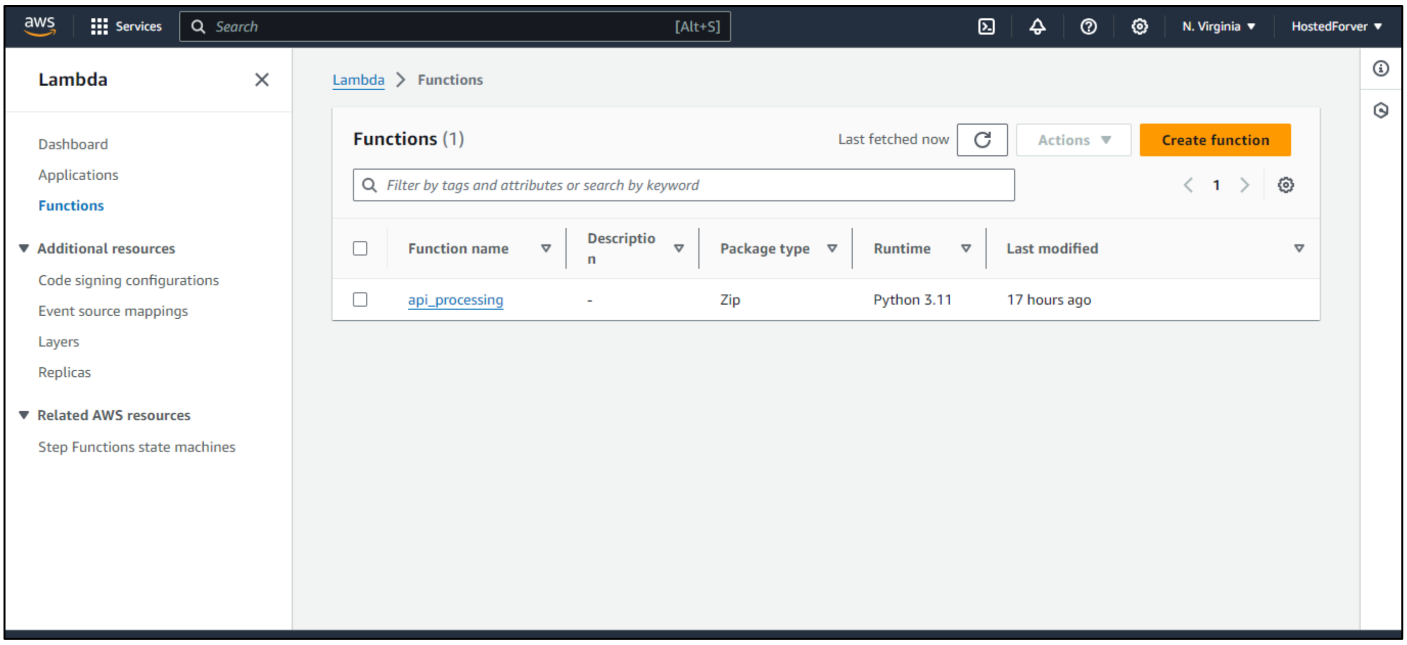
- Navigate to DynamoDB service
- Create a new a table
- Table name : employee_info
- Partition key : Employeeid
- Create Table



Name	Status	Partition key	Sort key	Indexes	Deletion protection	Read capacity mode
employee_info	Active	employeeid (S)	-	0	Off	Provisioned (1)

Step-2 : Creating a function in Lambda using python script

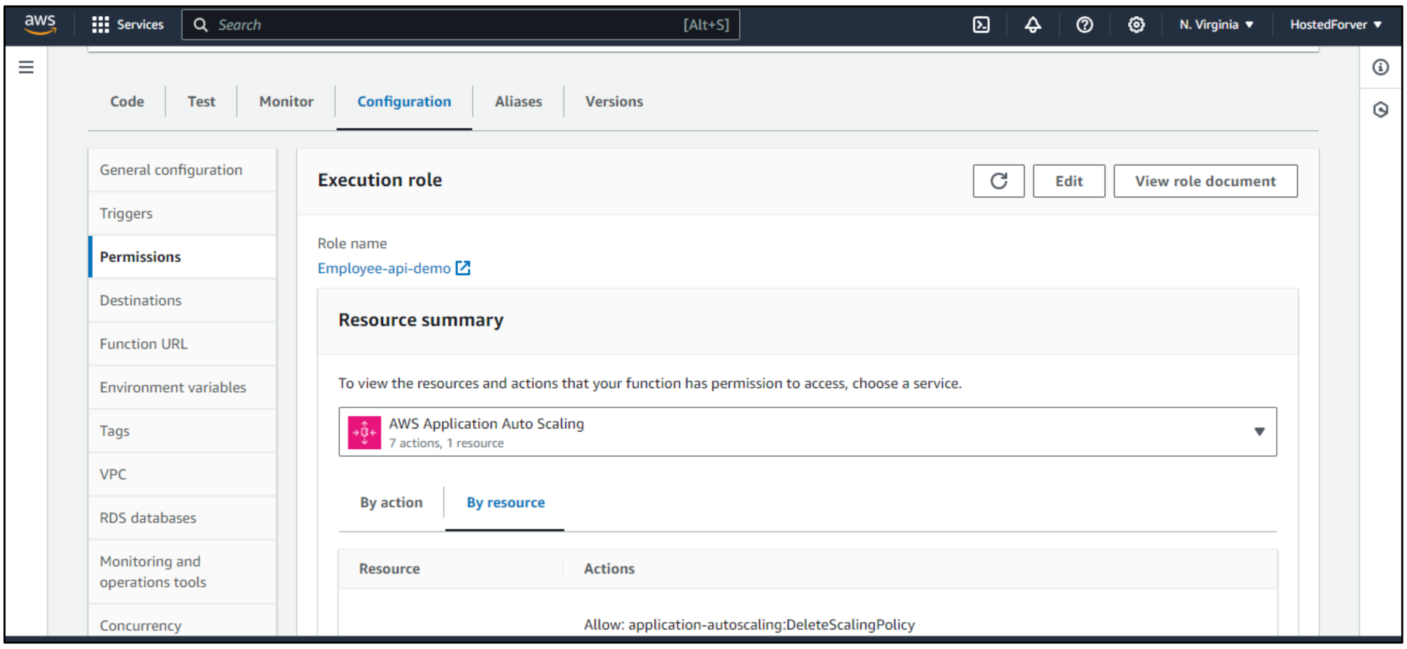
- Navigate to Lambda Service and select Create Function
- Select Author from Scratch
- Function name : api_processing
- Select Python 3.11 runtime
- Scroll down and change default role execution
- Select Create a new role from AWS policy
- Role name : Employee-api-demo
- Create Function



Function name	Description	Package type	Runtime	Last modified
api_processing	-	Zip	Python 3.11	17 hours ago

Step-2 A : Once Lambda function is created, make changes to configuration.

- Navigate to configuration in tab section of code editor
- click on permissions and then click link that appears below role name
- Scroll down to permission policies
- Click on drop-down option name as (Add permissions)
- Select Attach policies
- Once landed on policies page
- Click on the Search box and select the following policies
- DynamoDBFullAccess
- CloudWatchlogsFullAccess
- Once policies selected, sroll down and click on Add permissions.



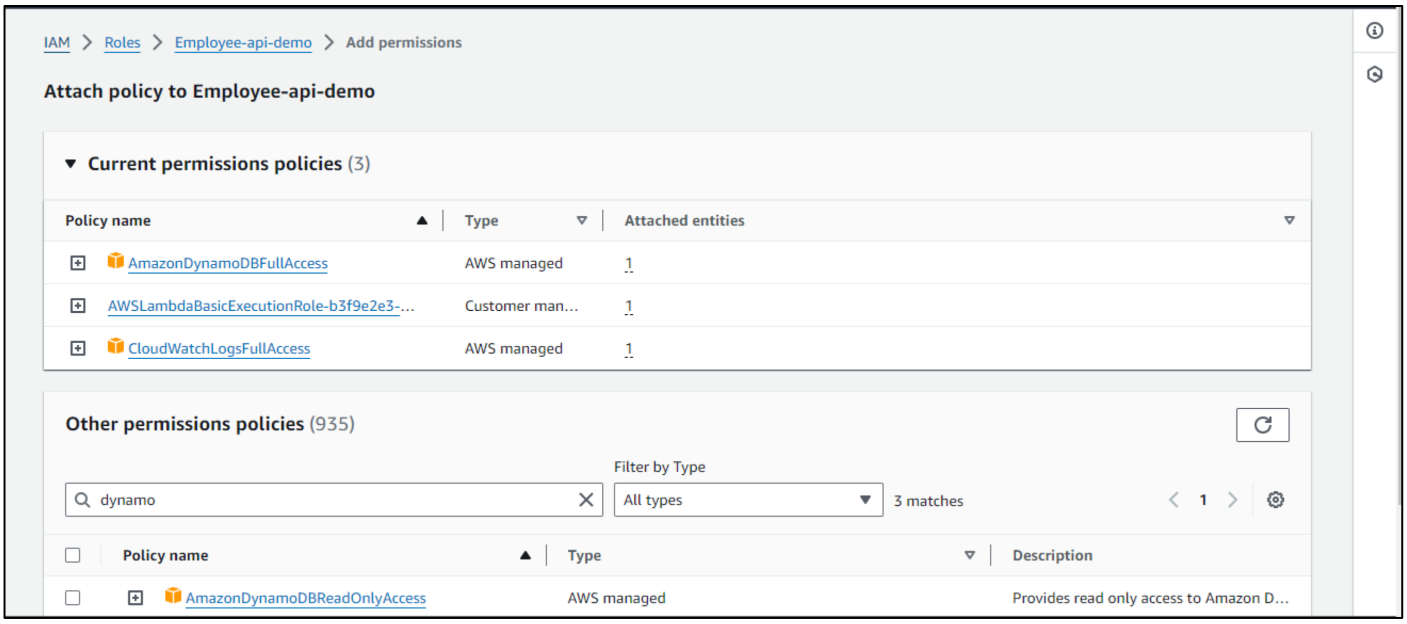
Role name
Employee-api-demo

Resource summary

To view the resources and actions that your function has permission to access, choose a service.

AWS Application Auto Scaling
7 actions, 1 resource

Resource	Actions
	Allow: application-autoscaling:DeleteScalingPolicy


IAM > Roles > Employee-api-demo > Add permissions

Attach policy to Employee-api-demo

Current permissions policies (3)

Policy name	Type	Attached entities
AmazonDynamoDBFullAccess	AWS managed	1
AWSLambdaBasicExecutionRole-b3f9e2e3-...	Customer man...	1
CloudWatchLogsFullAccess	AWS managed	1

Other permissions policies (935)

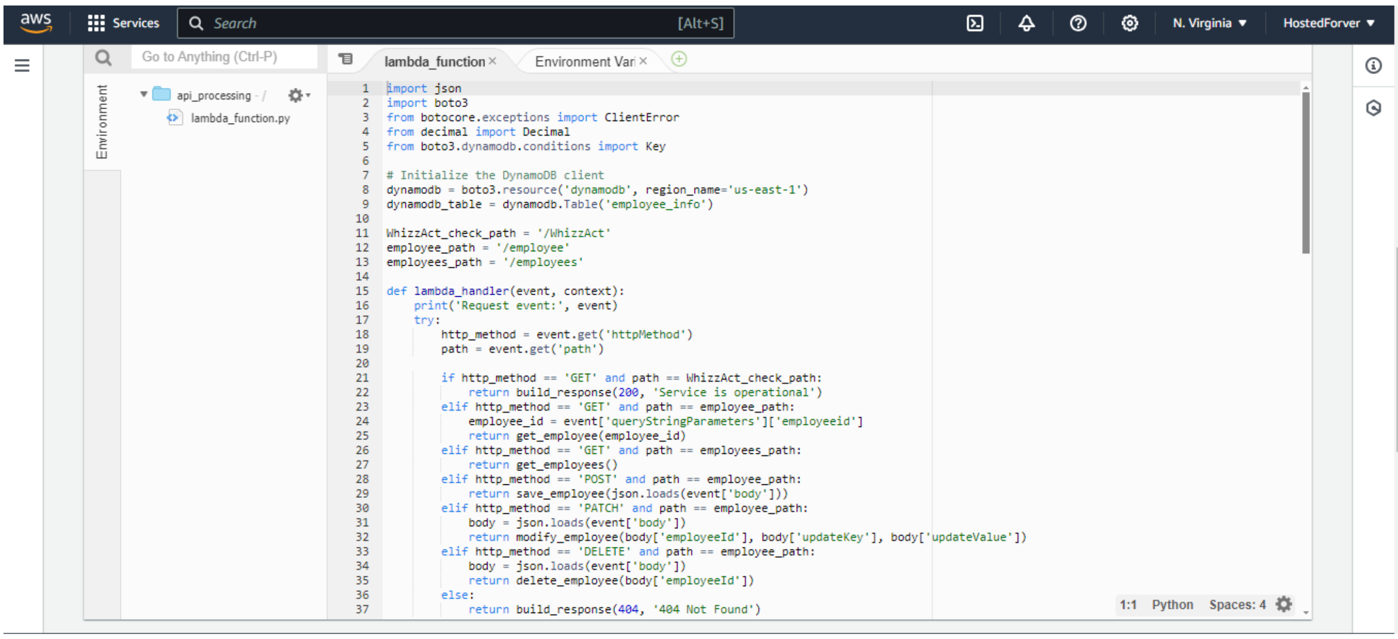
Filter by Type

dynamo — All types — 3 matches

Policy name	Type	Description
AmazonDynamoDBReadOnlyAccess	AWS managed	Provides read only access to Amazon D...

Step-2 B : Last and most important part in Lambda Function

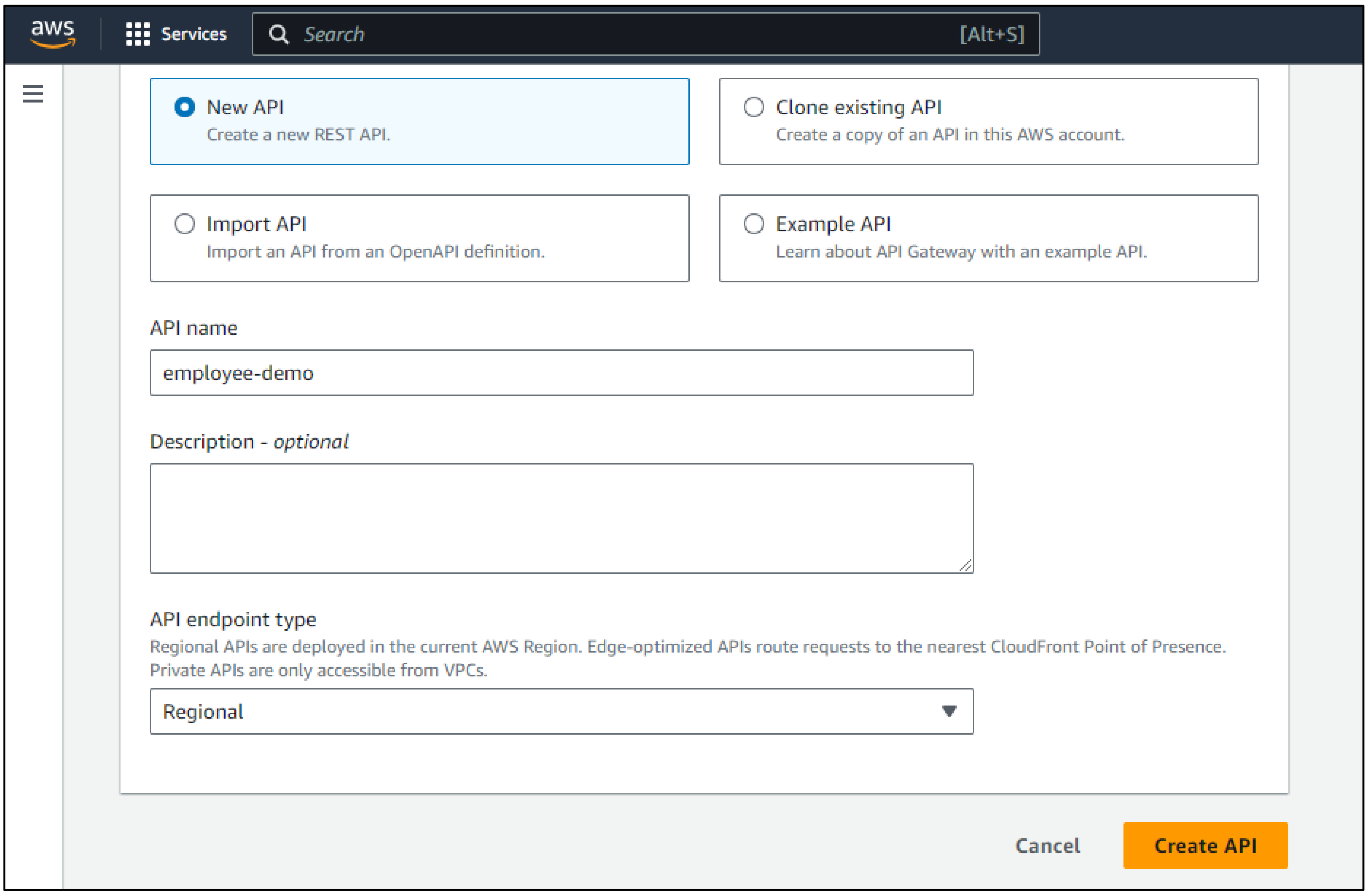
- Write a code that will execute the CRUD operations logic with respect to different Methods
- Navigate to Code section in the Lambda function page
- Copy the given code from the Github link and paste it in the code editor of lambda window
- https://github.com/abhishekpandey3057/AWS-projects/blob/63951d18080a2dee1e8cd5706f14e28a8b4ef041/Lambda%20Python%20Script.txt
- Deploy the code
- Once deployed, Click on Test and Name it as test and Save it



```python
import json
import boto3
from botocore.exceptions import ClientError
from decimal import Decimal
from boto3.dynamodb.conditions import Key

# Initialize the DynamoDB client
dynamodb = boto3.resource('dynamodb', region_name='us-east-1')
dynamodb_table = dynamodb.Table('employee_info')

WhizzAct_check_path = '/WhizzAct'
employee_path = '/employee'
employees_path = '/employees'

def lambda_handler(event, context):
    print('Request event:', event)
    try:
        http_method = event.get('httpMethod')
        path = event.get('path')

        if http_method == 'GET' and path == WhizzAct_check_path:
            return build_response(200, 'Service is operational')
        elif http_method == 'GET' and path == employee_path:
            employee_id = event['queryStringParameters']['employeeid']
            return get_employee(employee_id)
        elif http_method == 'GET' and path == employees_path:
            return get_employees()
        elif http_method == 'POST' and path == employee_path:
            return save_employee(json.loads(event['body']))
        elif http_method == 'PATCH' and path == employee_path:
            body = json.loads(event['body'])
            return modify_employee(body['employeeId'], body['updateKey'], body['updateValue'])
        elif http_method == 'DELETE' and path == employee_path:
            body = json.loads(event['body'])
            return delete_employee(body['employeeId'])
        else:
            return build_response(404, '404 Not Found')
```

Step-3 : Building REST API and connecting the other services to it

- Navigate to API gateway service
- Scroll down and select Build REST API
- Select New API
- API name : Employee-demo
- Select Regional in API endpoint type
- Click on create API



New API [✓] — Create a new REST API.

Clone existing API — Create a copy of an API in this AWS account.

Import API — Import an API from an OpenAPI definition.

Example API — Learn about API Gateway with an example API.

API name: employee-demo

Description - *optional*

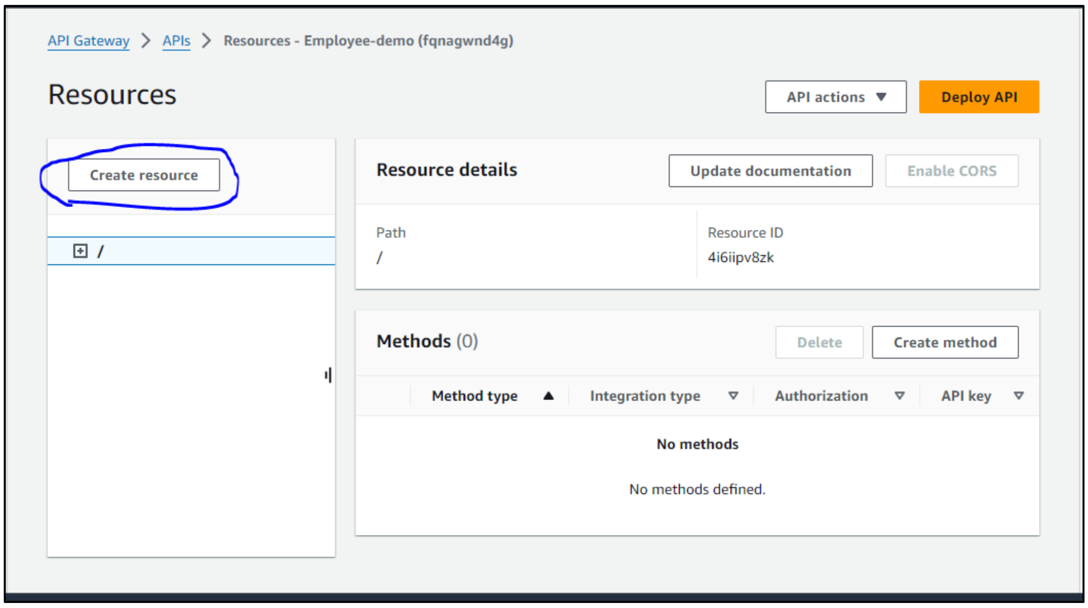
API endpoint type

Regional APIs are deployed in the current AWS Region. Edge-optimized APIs route requests to the nearest CloudFront Point of Presence. Private APIs are only accessible from VPCs.

Regional

Step-3 A : Once API is created, we will add resources and methods in it

- Click on Create resource on the main page



API Gateway > APIs > Resources - Employee-demo (fqnagwnd4g)

Resources

Resource details

Path: /

Resource ID: 4i6iipv8zk

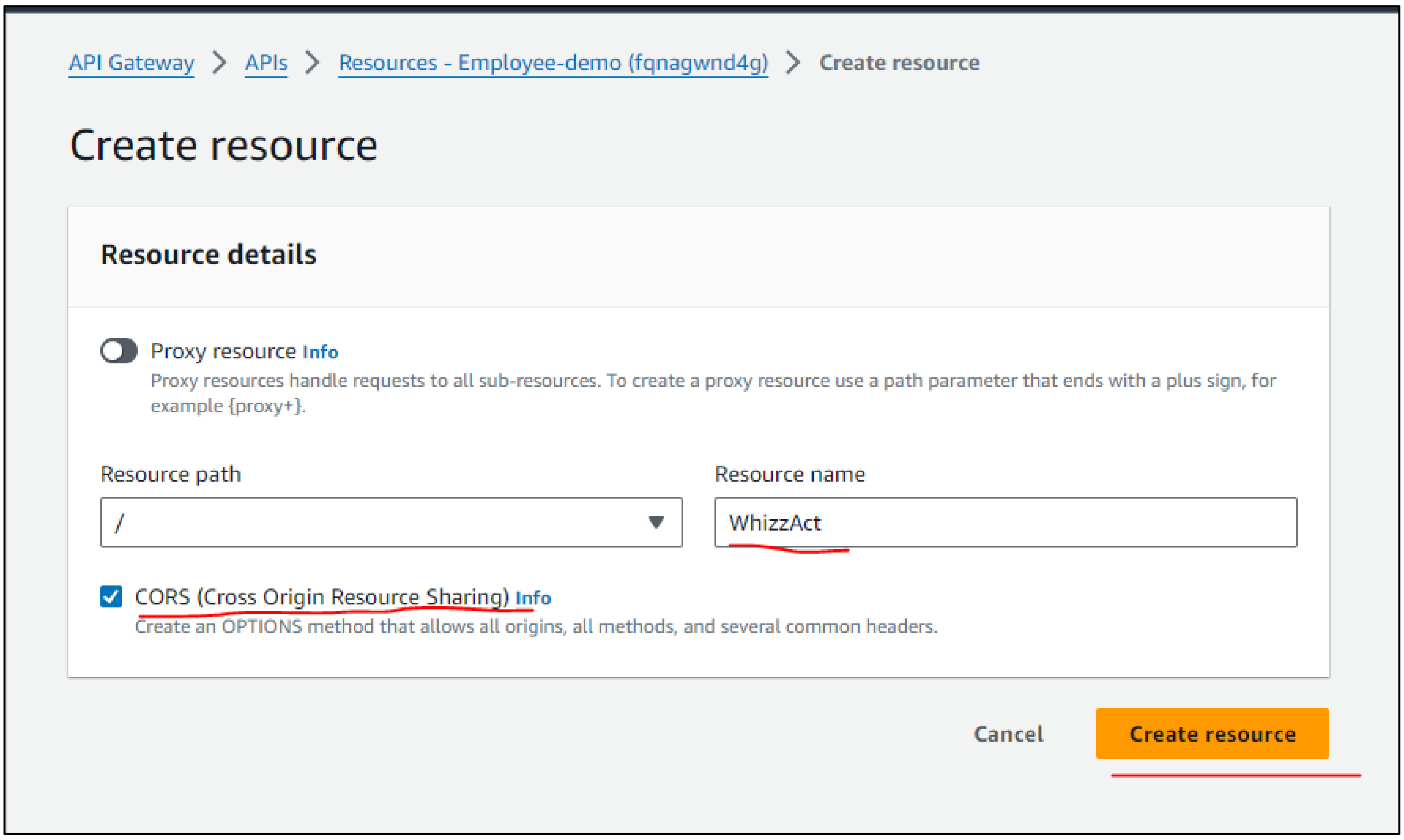
Methods (0)

Method type	Integration type	Authorization	API key

No methods

No methods defined.

- Resource name : WhizzAct
- Scroll down and make sure that CORS checkbox is enabled
- Click on create Resource



API Gateway > APIs > Resources - Employee-demo (fqnagwnd4g) > Create resource

Create resource

Resource details

Proxy resource Info
Proxy resources handle requests to all sub-resources. To create a proxy resource use a path parameter that ends with a plus sign, for example {proxy+}.

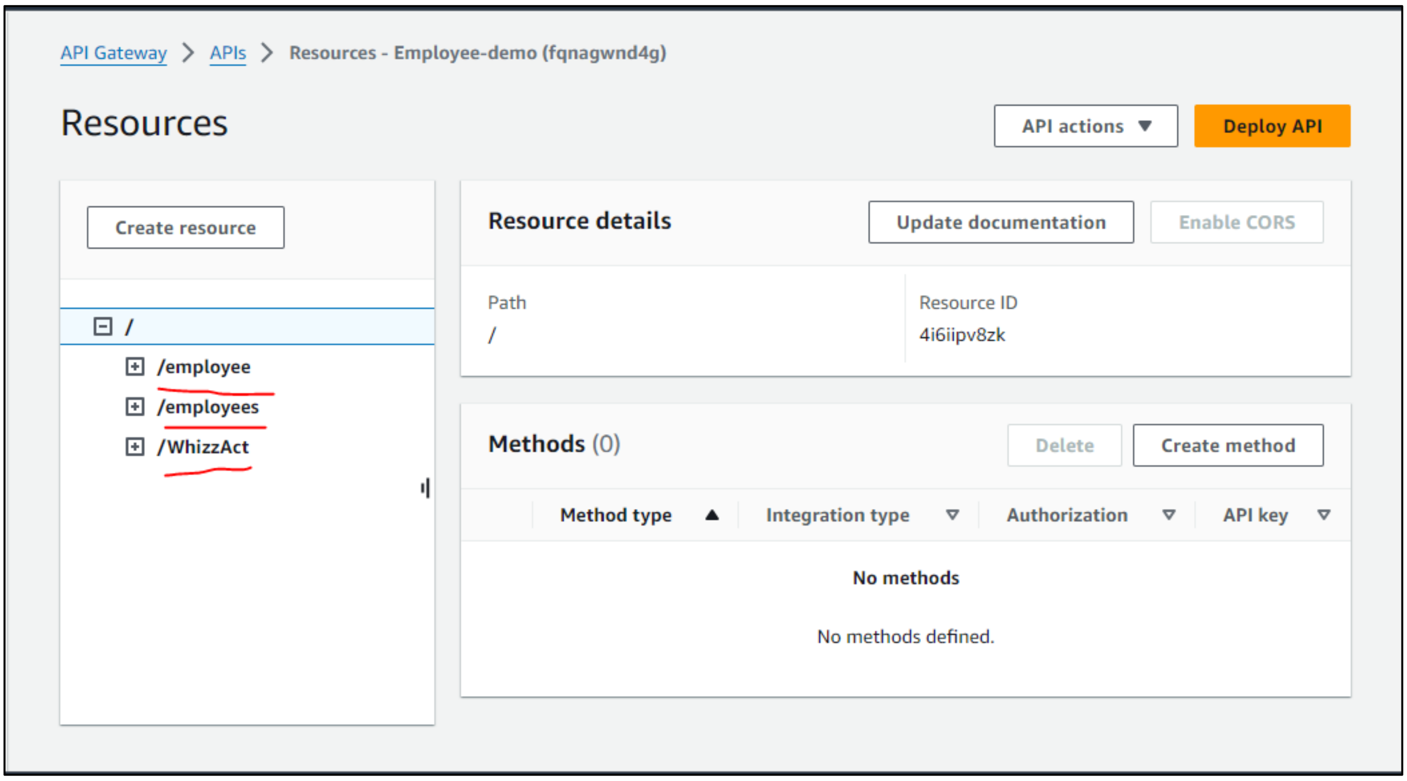
Resource path: /

Resource name: WhizzAct

☑ CORS (Cross Origin Resource Sharing) Info
Create an OPTIONS method that allows all origins, all methods, and several common headers.

Cancel | Create resource

Step-3 B : Repeat the above process and create the 2 more resource named as Employee and Employees



API Gateway > APIs > Resources - Employee-demo (fqnagwnd4g)

Resources

API actions | Deploy API

Create resource

- /
  - /employee
  - /employees
  - /WhizzAct

Resource details | Update documentation | Enable CORS

Path: /

Resource ID: 4i6iipv8zk

Methods (0) | Delete | Create method

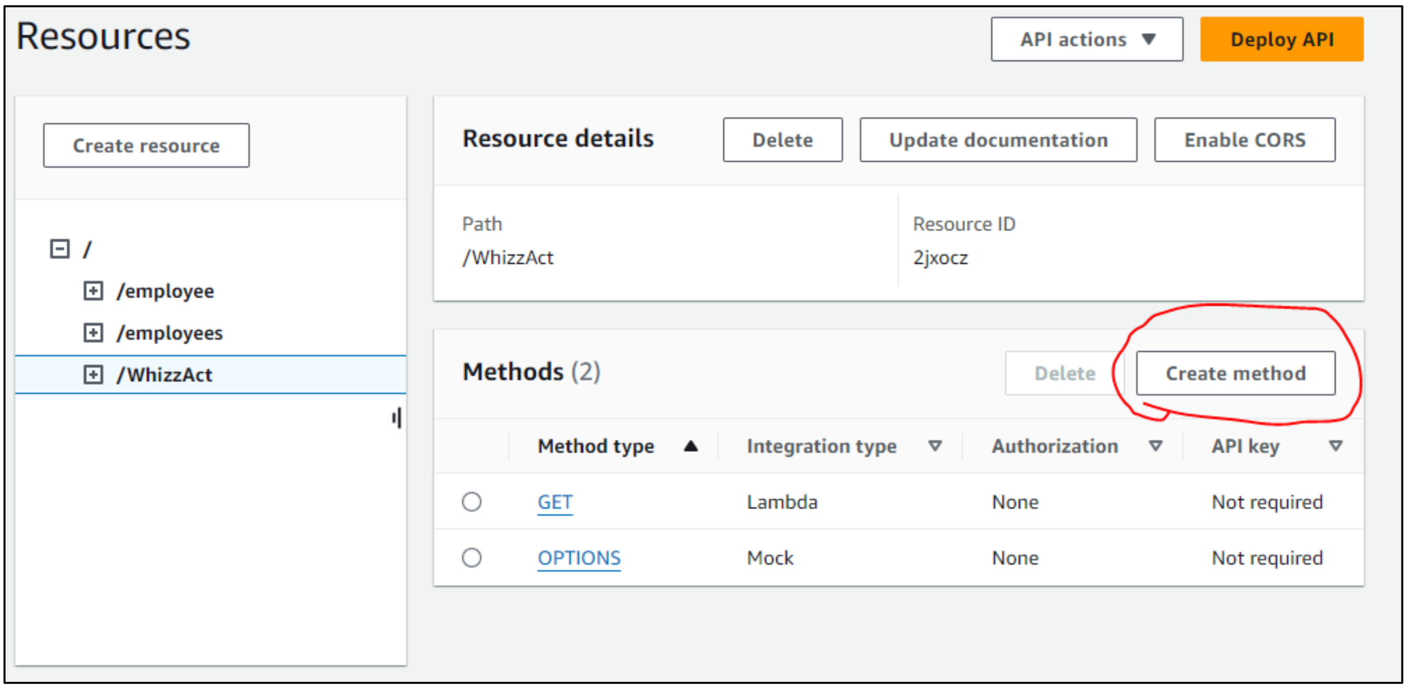
Method type	Integration type	Authorization	API key

No methods

No methods defined.

Step-4 : Once all resources has been created, now we will add method in every resource

- To add method select WhizzAct resource
- Navigate to Create Method in right side of the screen
- Select Method type = GET / Integration Type = Lambda Function
- Enable Proxy Integration
- Scroll down and under Lambda Function select Lambda function which we have created in Lambda Service (api_processing)
- Scroll down and Click Create method



Path
/WhizzAct

Resource ID
2jxocz

Method type	Integration type	Authorization	API key
GET	Lambda	None	Not required
OPTIONS	Mock	None	Not required

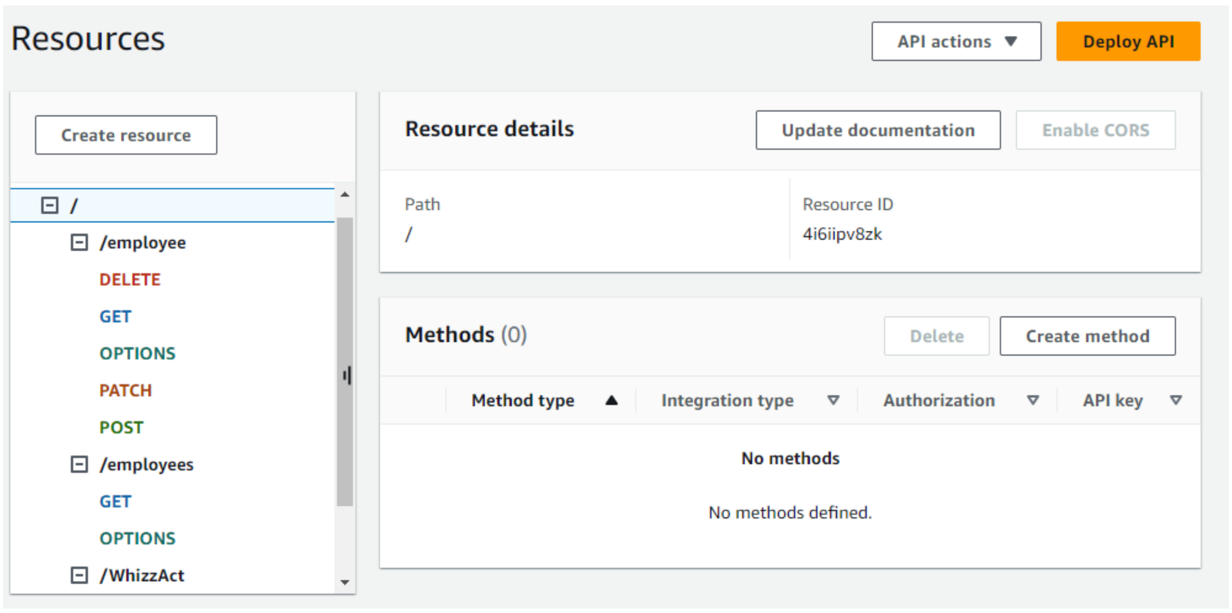
Step-4 A : Creating method in Employees resource

- Click on employees resource on the left side of the screen
- Repeat the Step-4 process for adding GET method in employee resource

Step-4 B : Creating methods in employee resource

- Click on employee resource
- Now repeat the Step-4 process in employee resource
- Remember, add GET, PATCH, POST, DELETE methods in same way (Step-4) under employee resource.

Once all the resource and methods has been created, your main screen should look like this



Resource details

Path: /

Resource ID: 4i6iipv8zk

/
- /employee
  - DELETE
  - GET
  - OPTIONS
  - PATCH
  - POST
- /employees
  - GET
  - OPTIONS
- /WhizzAct

Methods (0)

Method type	Integration type	Authorization	API key

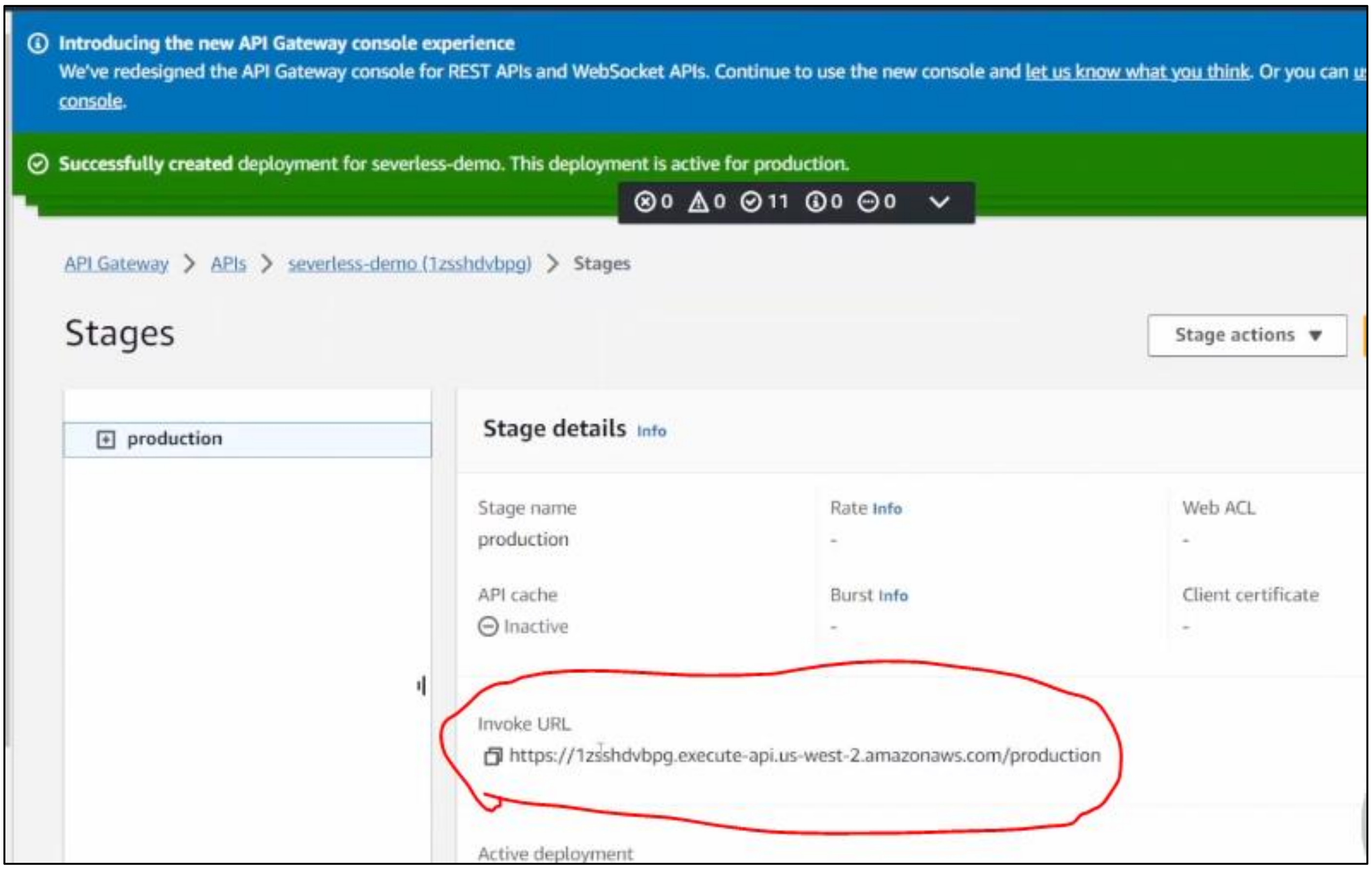
No methods

No methods defined.

Step-4 C : Click on Deploy API

- Click on stage menu and select New Stage
- Name the stage as Production
- Click on Deploy

Once API is deployed, Scroll down and copy the Invoke URl that is generated for later.



Introducing the new API Gateway console experience

We've redesigned the API Gateway console for REST APIs and WebSocket APIs. Continue to use the new console and let us know what you think. Or you can u console.

Successfully created deployment for severless-demo. This deployment is active for production.

Stage details Info

Stage name: production

Rate Info: -

Web ACL: -

API cache: Inactive

Burst Info: -

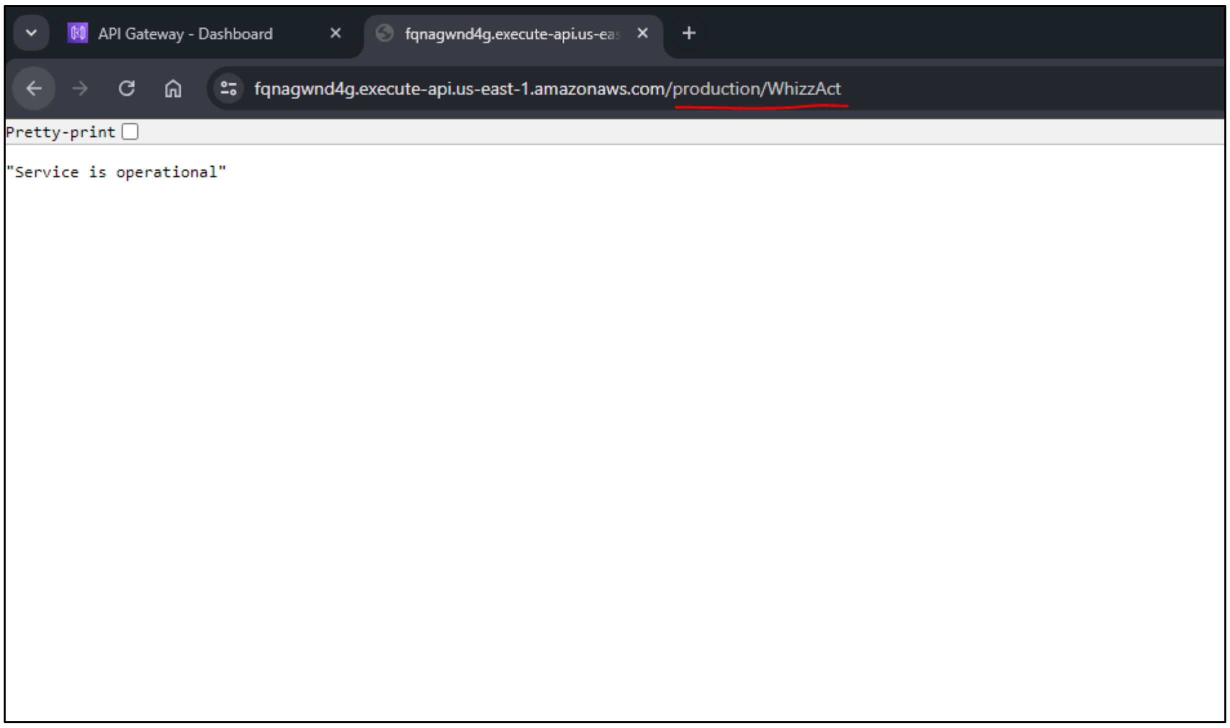
Client certificate: -

Invoke URL: https://1zsshdvbpg.execute-api.us-west-2.amazonaws.com/production

Active deployment

Step-4 D : To check if our API is deployed properly or not, follow the below steps

- Paste the Copied invoke URl to new browser tab
- add /WhizzAct at end of the URl after production and hit enter
- If the output on the screen shows "Service is operational" then your link is working and all the connection has been established properly


API Gateway - Dashboard
fqnagwnd4g.execute-api.us-ea
fqnagwnd4g.execute-api.us-east-1.amazonaws.com/production/WhizzAct
Pretty-print

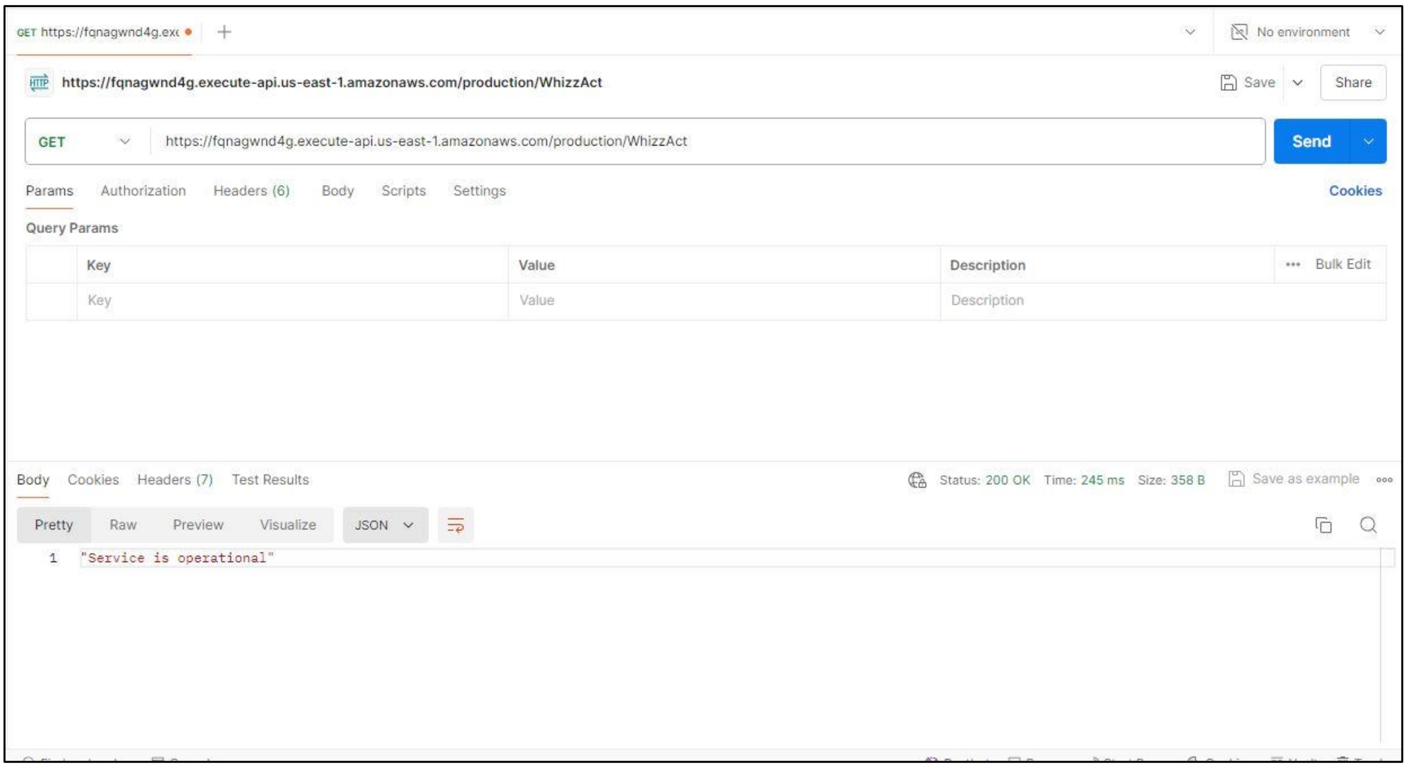
```
"Service is operational"
```

Part-2 : Executing Create, Read, Update, Delete (CRUD) operations on the data.

Step-1 : Open POSTMAN console / Web version

- Paste the link that you have copied in last step (Invoke URL link)
- add /WhizzAct at the end of the URL
- select GET method from left-side drop down menu
- Click on send button

In the terminal, you will see the output like below and it will display "Service is Operational"



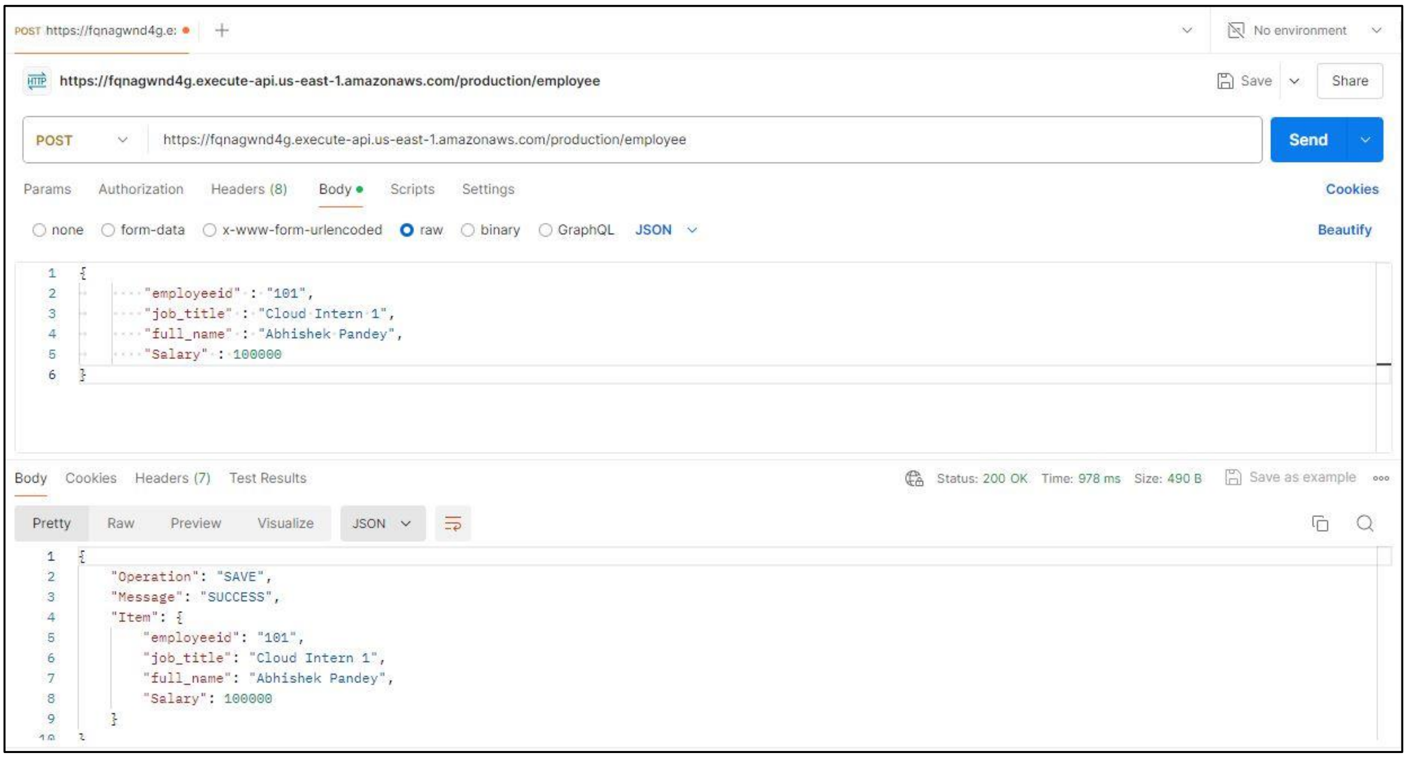
```
"Service is operational"
```

Step-2 : Now we have to add employees by using CREATE METHOD

- Paste the copied Invoke URL link in the same POSTMAN console
- select POST (Create) method from left-side drop down menu
- add /employee at the end of the URL after /production

- below the URL dialog box, select body tab and under body tab select raw button
- beside raw radio button, there will be dropdown for script language
- select JSON and there will be code editor window below that
- write the following code in JSON to add employee details to the database table (employee_info)

-
  ```
  {
     "employeeid" : "101",
     "job_title" : "Cloud Intern 1",
     "full_name" : "Abhishek Pandey",
     "Salary" : 100000
   }
  ```
- Click on Send button and you will see below output with message



https://fqnagwnd4g.execute-api.us-east-1.amazonaws.com/production/employee

POST https://fqnagwnd4g.execute-api.us-east-1.amazonaws.com/production/employee

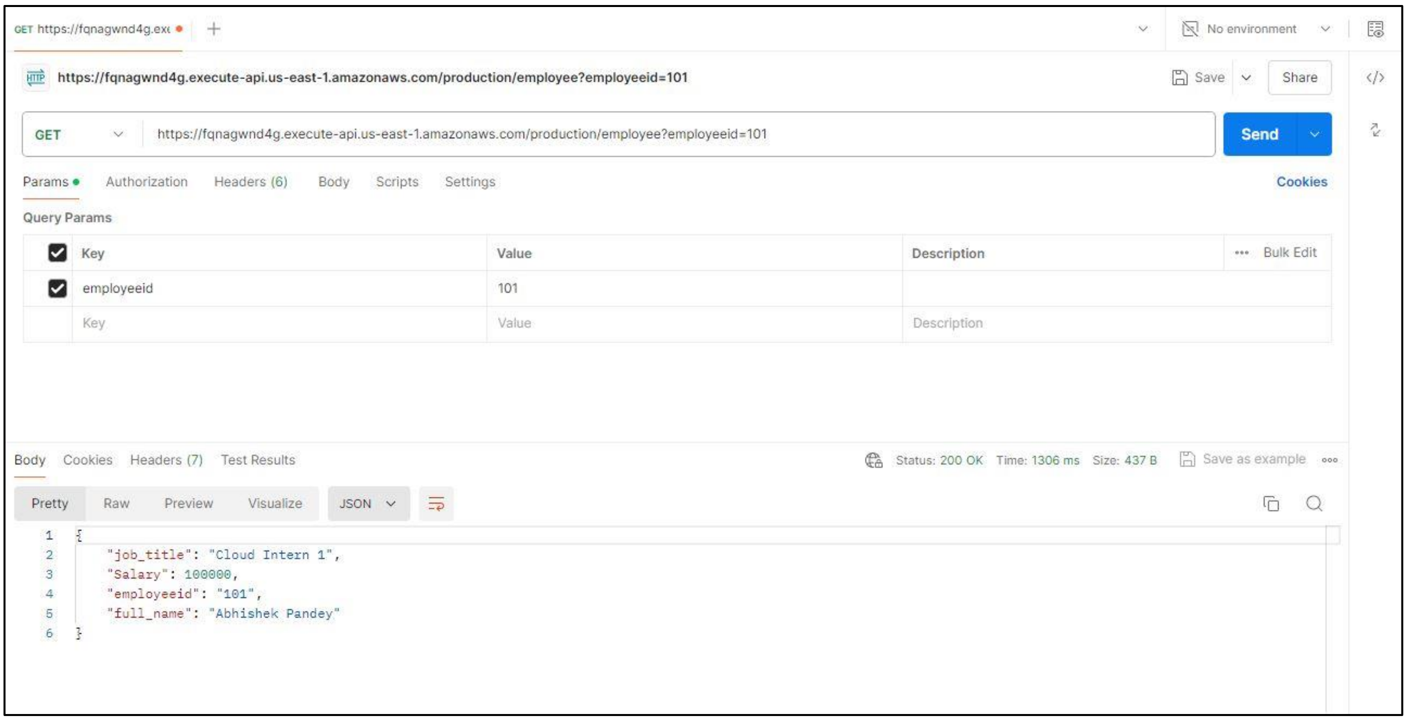
```
{
    "employeeid" : "101",
    "job_title" : "Cloud Intern 1",
    "full_name" : "Abhishek Pandey",
    "Salary" : 100000
}
```

Status: 200 OK Time: 978 ms Size: 490 B

```
{
    "Operation": "SAVE",
    "Message": "SUCCESS",
    "Item": {
        "employeeid": "101",
        "job_title": "Cloud Intern 1",
        "full_name": "Abhishek Pandey",
        "Salary": 100000
    }
}
```

Step-3 : Now we have to read(get) employee details by using GET METHOD

- Paste the link that you have copied in Step-4 (Invoke URL link)
- add /employee?employee=101 at the end of the URL after /production
- select GET method from left-side drop down menu
- Click on send button
- You will see the following output as given below and it indicates that employee details is been created successfully



https://fqnagwnd4g.execute-api.us-east-1.amazonaws.com/production/employee?employeeid=101

GET https://fqnagwnd4g.execute-api.us-east-1.amazonaws.com/production/employee?employeeid=101

Query Params

Key	Value	Description
✓ employeeid	101	
Key	Value	Description

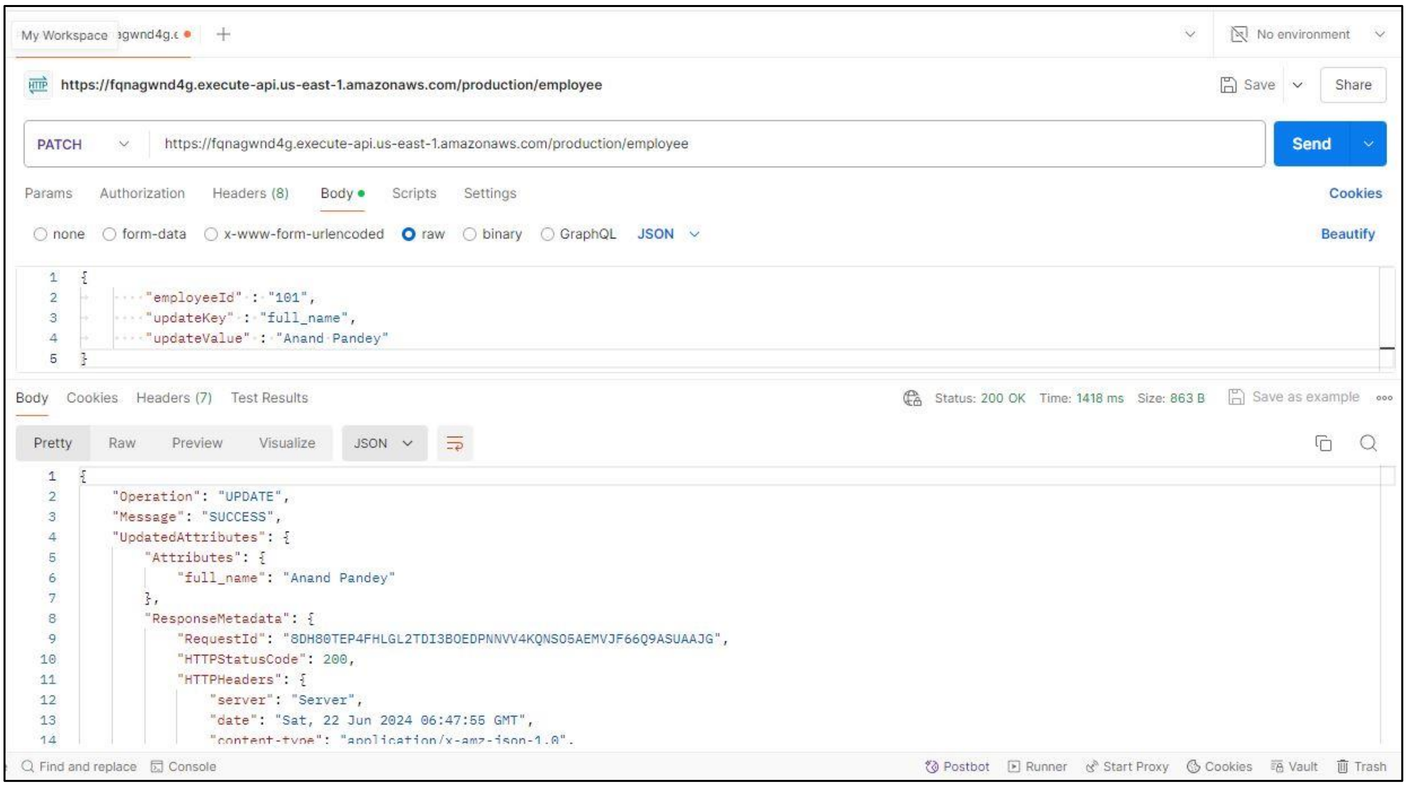
Status: 200 OK Time: 1306 ms Size: 437 B

```
{
    "job_title": "Cloud Intern 1",
    "Salary": 100000,
    "employeeid": "101",
    "full_name": "Abhishek Pandey"
}
```

Step-4 : Now we have to update employee details by using PATCH METHOD

- Paste the link that you have copied in Step-4 (Invoke URL link)
- add /employee at the end of the URL after /production
- select PATCH method from left-side drop down menu
- below the URL dialog box, select body tab and under body tab select raw button
- beside raw radio button, there will be dropdown for script language
- select JSON and there will be code editor window below that
- write the following code in JSON to update employee details to the database table (employee_info)

-
  ```
  {
    "employeeId" : "101",
    "updateKey" : "full_name",
    "updateValue" : "Anand Pandey"
  }
  ```
- Click on Send button and you will see below output with message



PATCH https://fqnagwnd4g.execute-api.us-east-1.amazonaws.com/production/employee

```
{
    "employeeId" : "101",
    "updateKey" : "full_name",
    "updateValue" : "Anand Pandey"
}
```

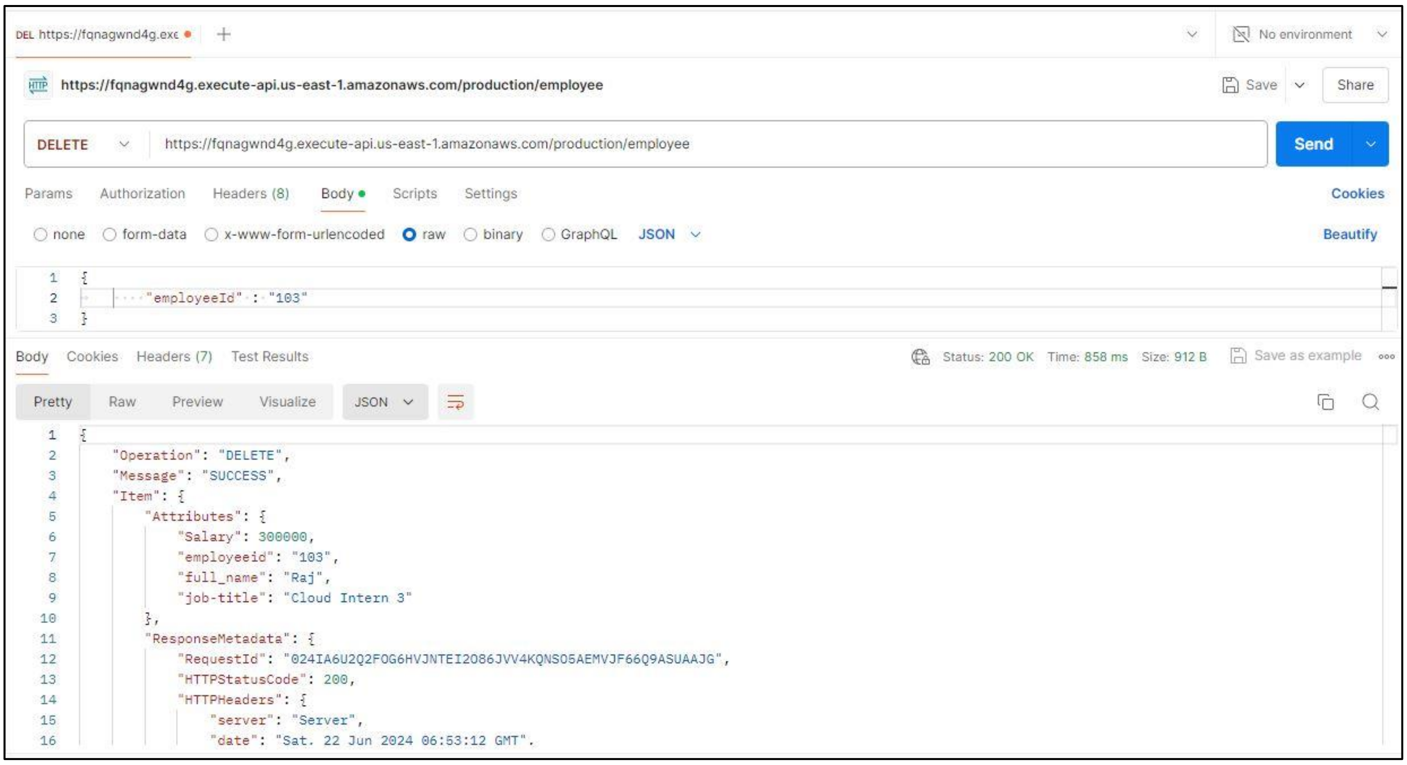
Status: 200 OK Time: 1418 ms Size: 863 B

```
{
    "Operation": "UPDATE",
    "Message": "SUCCESS",
    "UpdatedAttributes": {
        "Attributes": {
            "full_name": "Anand Pandey"
        },
        "ResponseMetadata": {
            "RequestId": "8DH80TEP4FHLGL2TDI3BOEDPNNVV4KQNSO5AEMVJF66Q9ASUAAJG",
            "HTTPStatusCode": 200,
            "HTTPHeaders": {
                "server": "Server",
                "date": "Sat, 22 Jun 2024 06:47:55 GMT",
                "content-type": "application/x-amz-json-1.0",
```

Step-5 : Now we have to delete employee details by using DELETE METHOD

- Paste the link that you have copied in Step-4 (Invoke URL link)
- add /employee at the end of the URL after /production
- select DELETE method from left-side drop down menu
- below the URL dialog box, select body tab and under body tab select raw button
- beside raw radio button, there will be dropdown for script language
- select JSON and there will be code editor window below that
- write the following code in JSON to update employee details to the database table (employee_info)

-
  ```
  {
    "employeeId" : "101",
   }
  ```
- Click on Send button and you will see below output with message



https://fqnagwnd4g.execute-api.us-east-1.amazonaws.com/production/employee

DELETE https://fqnagwnd4g.execute-api.us-east-1.amazonaws.com/production/employee

```
{
    "employeeId" : "103"
}
```

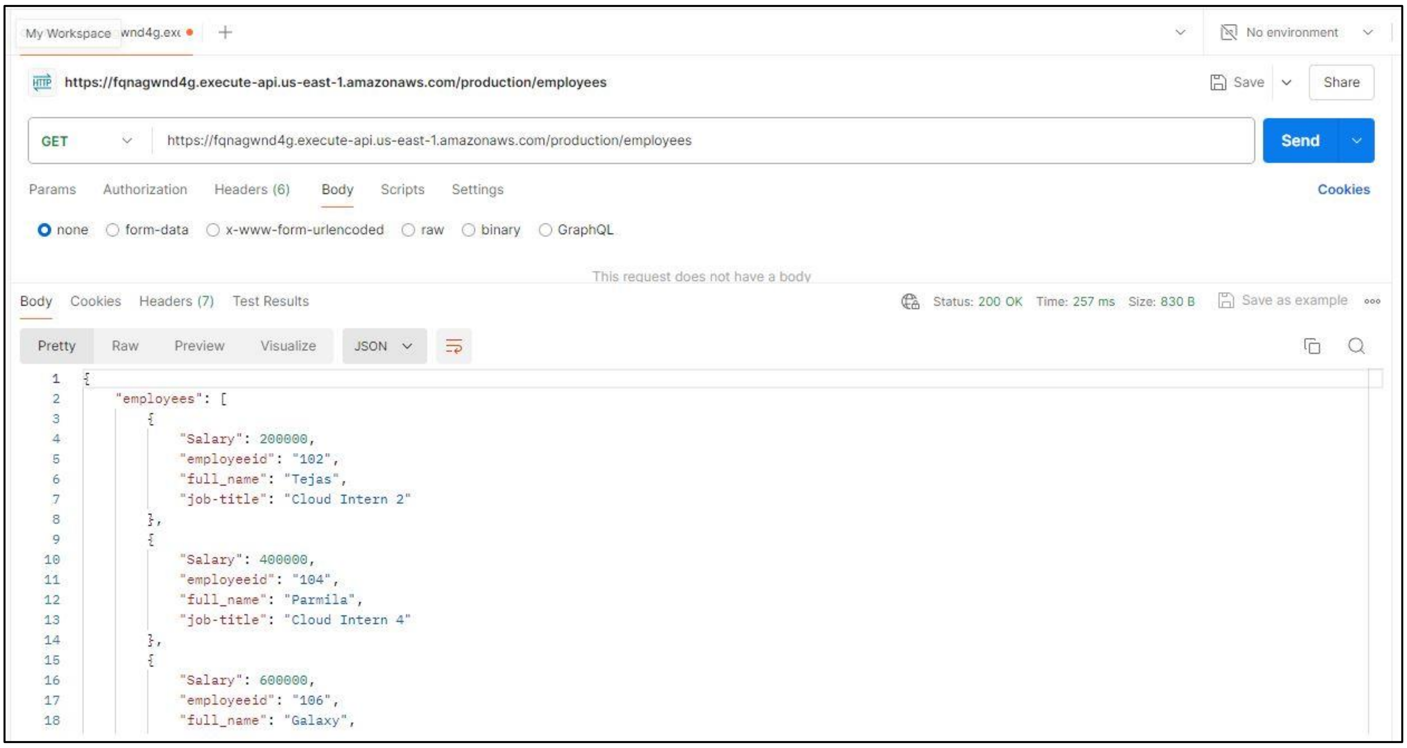
Status: 200 OK Time: 858 ms Size: 912 B

JSON

```
{
    "Operation": "DELETE",
    "Message": "SUCCESS",
    "Item": {
        "Attributes": {
            "Salary": 300000,
            "employeeid": "103",
            "full_name": "Raj",
            "job-title": "Cloud Intern 3"
        },
        "ResponseMetadata": {
            "RequestId": "024IA6U2Q2F0G6HVJNTEI2O86JVV4KQNSO5AEMVJF66Q9ASUAAJG",
            "HTTPStatusCode": 200,
            "HTTPHeaders": {
                "server": "Server",
                "date": "Sat. 22 Jun 2024 06:53:12 GMT",
```

Step-6 : After CRUD operations on employee table, get all employees details at once

- Paste the link that you have copied in Step-4 (Invoke URL link)
- add /employees at the end of the URL after /production
- select GET method from left-side drop down menu
- Click on Send button and you will see below output with message which will display all the employees details



https://fqnagwnd4g.execute-api.us-east-1.amazonaws.com/production/employees

GET https://fqnagwnd4g.execute-api.us-east-1.amazonaws.com/production/employees

Body: none

This request does not have a body

Status: 200 OK Time: 257 ms Size: 830 B

JSON

```
{
    "employees": [
        {
            "Salary": 200000,
            "employeeid": "102",
            "full_name": "Tejas",
            "job-title": "Cloud Intern 2"
        },
        {
            "Salary": 400000,
            "employeeid": "104",
            "full_name": "Parmila",
            "job-title": "Cloud Intern 4"
        },
        {
            "Salary": 600000,
            "employeeid": "106",
            "full_name": "Galaxy",
```

To check whether your above processes is executed correctly or not, follow below steps in every method executed : - Go to AWS console home page

- Navigate to DynamoDb service

- Click on tables tab

- Navigate to the table that you have created in the STEP-1

- Click on explore table items

- Scroll down and refresh, you will see that the details of employee has been created from the above process

All the above process, if performed correctly then we have successfully done the CRUD operations on our Employee_info table database.