😊 Build CRUD Microservice with HTTP API and Lambda

Overview:

In this hands-on lab, we'll create a serverless product API using AWS services such as **API Gateway**, **Lambda**, and **DynamoDB**. The API will allow us to perform **CRUD** (Create, Read, Update, Delete) operations on product data.

Architecture:

1. **HTTP API (API Gateway)** will handle incoming requests (GET, POST, DELETE) from the client.
2. **Lambda function** will process the request, interact with **DynamoDB** to perform the CRUD operations, and return a response to the client via API Gateway.
3. **DynamoDB** will store the product data.

API Operations:

- **GET /products**: Retrieve all products.
- **GET /products/{id}**: Retrieve a product by its ID.
- **POST /products**: Create a new product.
- **DELETE /products/{id}**: Delete a product by its ID.

Steps:

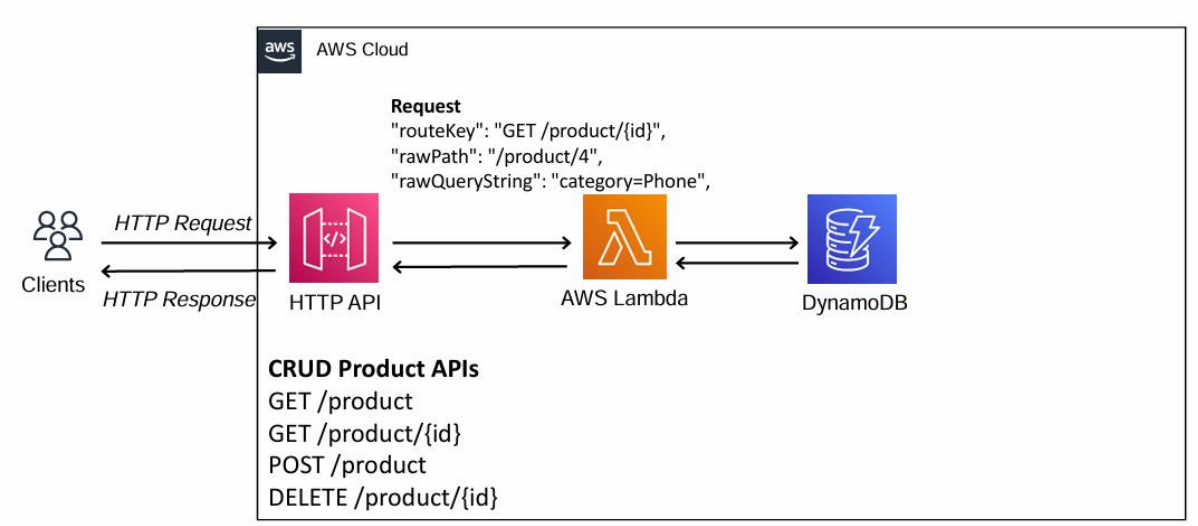
1. **Create Lambda Function**:
    - Open the AWS Lambda Management Console.
    - Create a new Lambda function named ProductFunction.
    - Select the runtime (e.g., Node.js, Python) and set permissions.
    - Initially, deploy the function with a simple "Hello from Lambda" response to test the setup.
2. **Log Incoming Requests**:
    - Modify the Lambda function to log incoming event data (JSON) to understand the request format sent by API Gateway. This will help us process incoming requests correctly.
3. **Create HTTP API in API Gateway**:
    - Set up an HTTP API in API Gateway to handle requests.
    - Configure the routes (GET, POST, DELETE) to trigger the Lambda function.
4. **DynamoDB Integration**:

- The Lambda function will interact with DynamoDB to store and retrieve product data.
- We'll set up the necessary DynamoDB tables to store product information (e.g., ProductID, ProductName, etc.).

5. **Deploy and Test**:
   - Deploy the Lambda function with the necessary code to handle CRUD operations.
   - Test the API endpoints by sending requests (GET, POST, DELETE) and ensuring the Lambda function processes them and interacts with DynamoDB correctly.

Conclusion:

By the end of this lab, you'll have a fully working serverless API that can handle product data CRUD operations, all running on AWS using **API Gateway**, **Lambda**, and **DynamoDB**.



CRUD stands for the four basic operations that can be performed on data in a database or application. The acronym stands for:

1. **Create**: Add new data or records.
2. **Read**: Retrieve or view existing data.
3. **Update**: Modify or update existing data.
4. **Delete**: Remove existing data.

These operations are fundamental for interacting with databases and are commonly used in building web applications, APIs, and systems that manage data.

Example:

In the context of a **Product API** (as described in your lab):

- **Create**: Add a new product to the database.

- **Read**: Get the details of a product, either all products or a specific product by ID.
- **Update**: Modify the details of an existing product (e.g., change the price or description).
- **Delete**: Remove a product from the database.

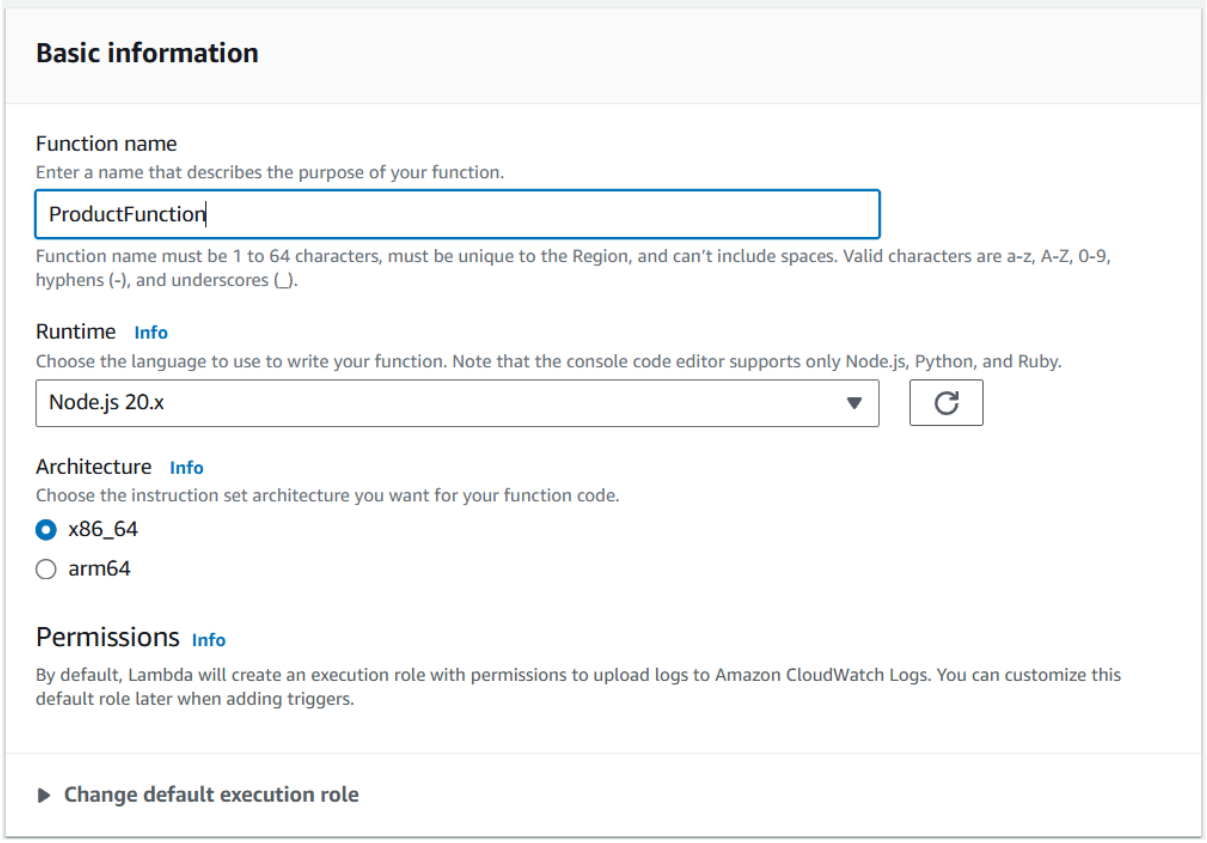
CRUD operations are typically mapped to HTTP methods in web APIs:

- **POST** → Create
- **GET** → Read
- **PUT/PATCH** → Update
- **DELETE** → Delete

These operations are the building blocks for managing data in most applications.

😄 To begin with the Lab:

1. First, we will create our Lambda function. In your AWS console, navigate to lambda and click on create.
2. Here you need to choose author from scratch and then give a name to your lambda function. Choose your preferred runtime we'll be choosing Node.js.
3. Then keep everything to default and create your lambda function.



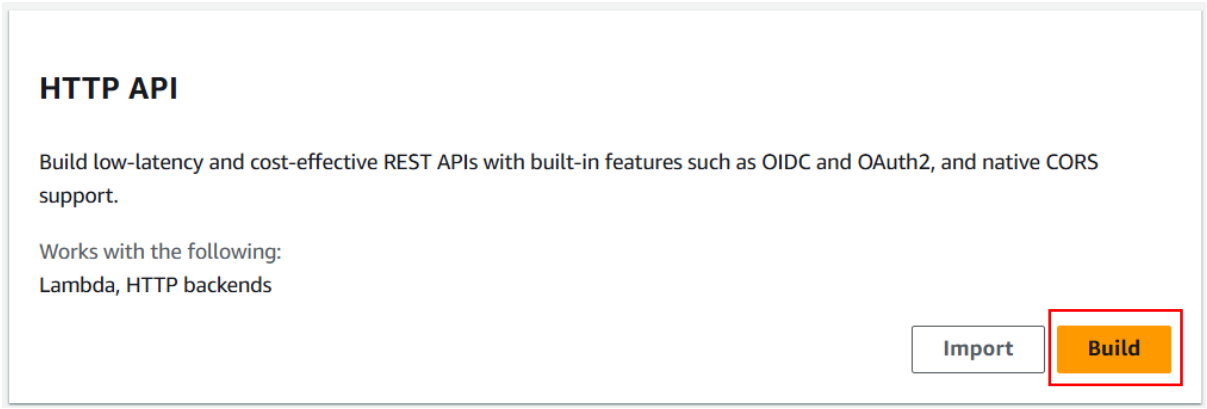
4. Once your function is created, go inside and change the code with the code below. Click on deploy.

```
export const handler = async (event) => {
  // Log the incoming event object (API Gateway event)
  console.log("event:", JSON.stringify(event, undefined, 2));

  // TODO: Implement your logic here (CRUD operations, interacting with DynamoDB, etc.)

  // Return a simple response with a statusCode and body
  const response = {
    statusCode: 200,  // HTTP status code (200 means OK)
    body: JSON.stringify('Hello from Lambda!'),  // Body of the response
  };

  // Return the response object
  return response;
};
```

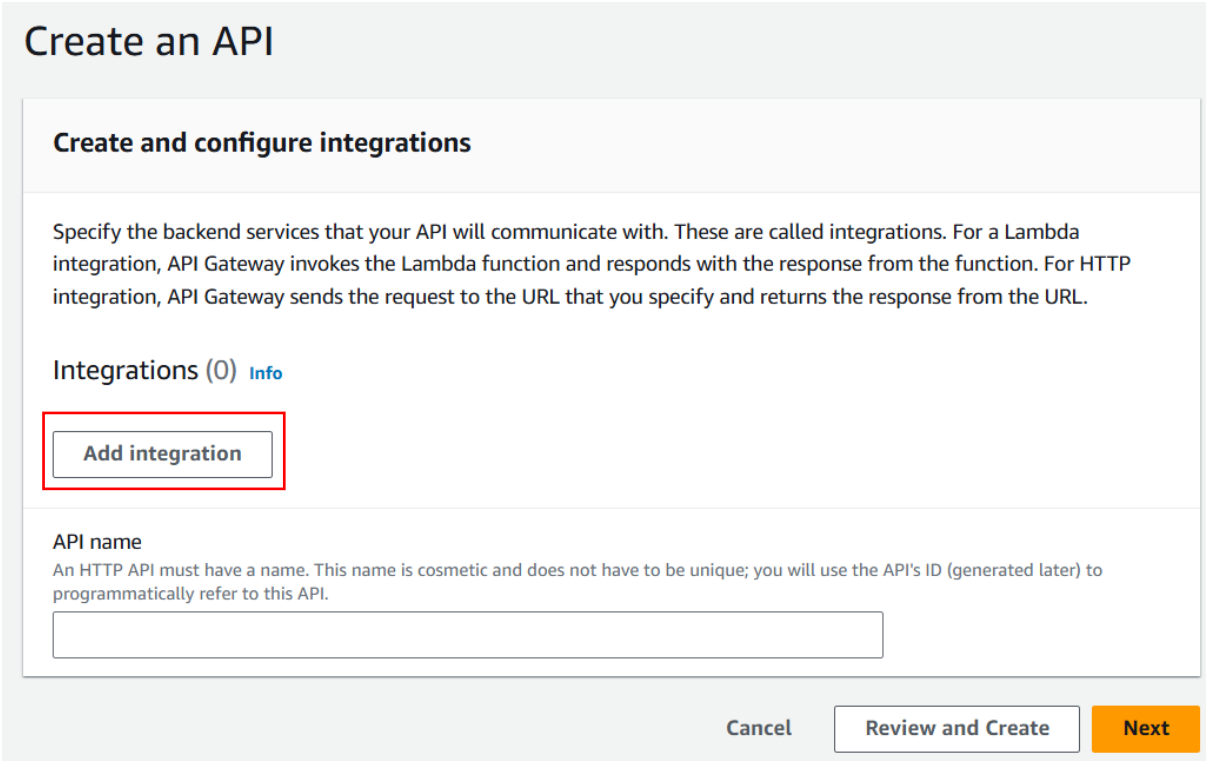
index.mjs

```
export const handler = async (event) => {
  // Log the incoming event object (API Gateway event)
  console.log("event:", JSON.stringify(event, undefined, 2));

  // TODO: Implement your logic here (CRUD operations, interacting with DynamoDB, etc.)

  // Return a simple response with a statusCode and body
  const response = {
    statusCode: 200,  // HTTP status code (200 means OK)
    body: JSON.stringify('Hello from Lambda!'),  // Body of the response
  };

  // Return the response object
  return response;
};
```

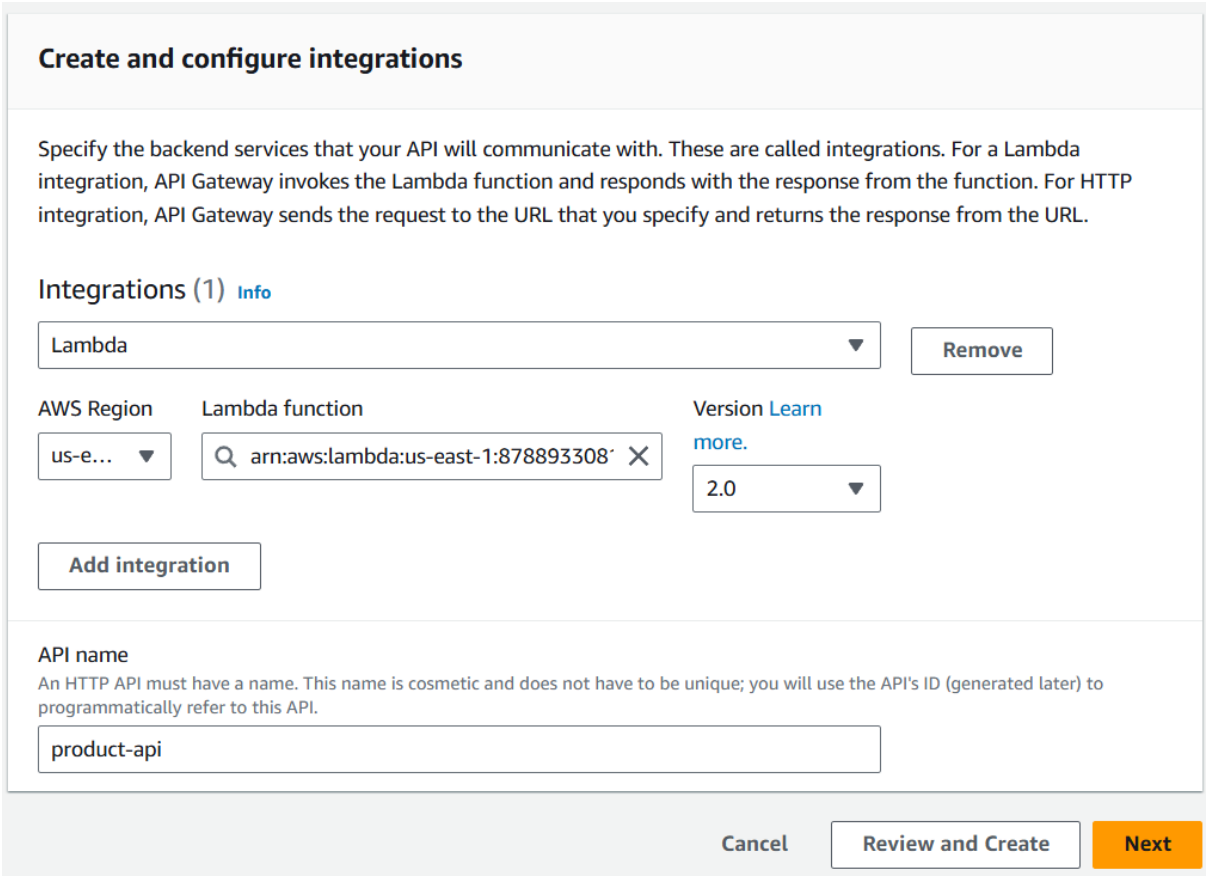
5. Now we are going to move forward and create our HTTP API. For that we need to search and go to the API gateway. From there we need to choose HTTP API and click on Build.



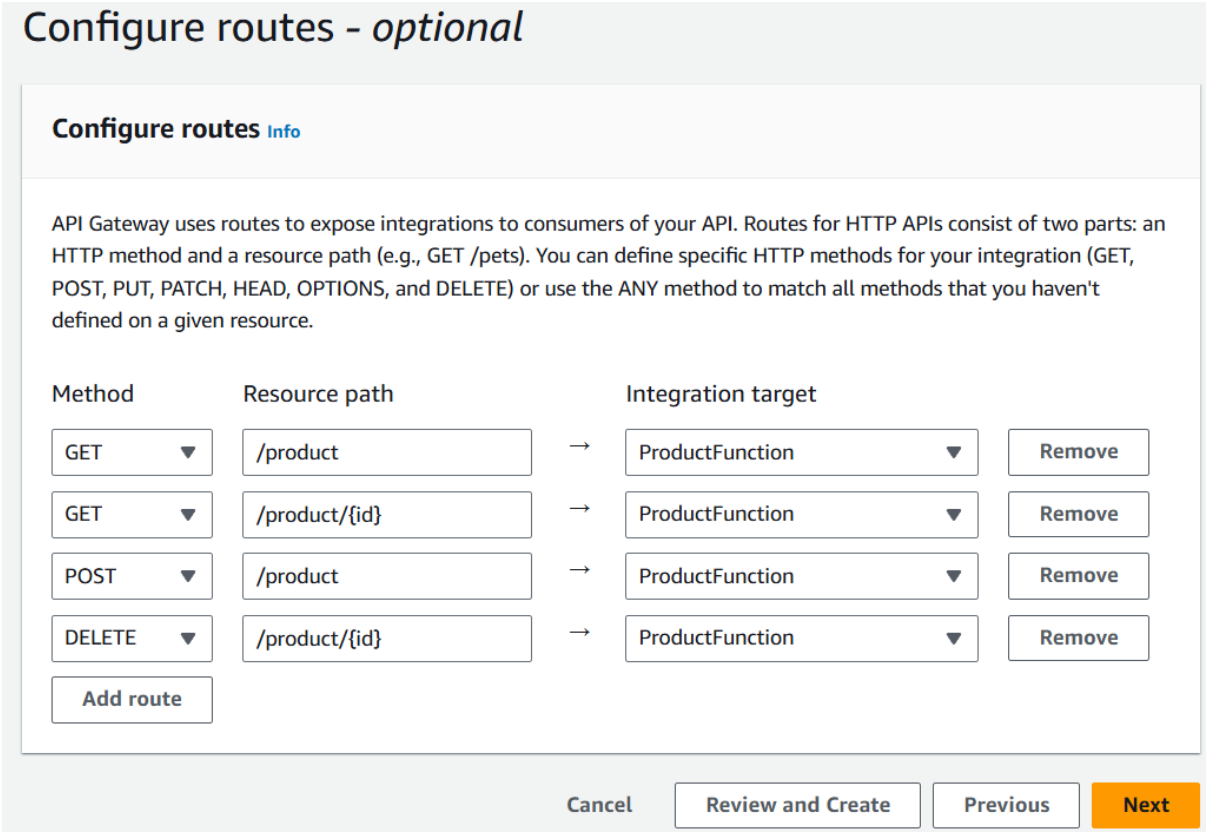
6. From step 1 you need to click on Add Integration.



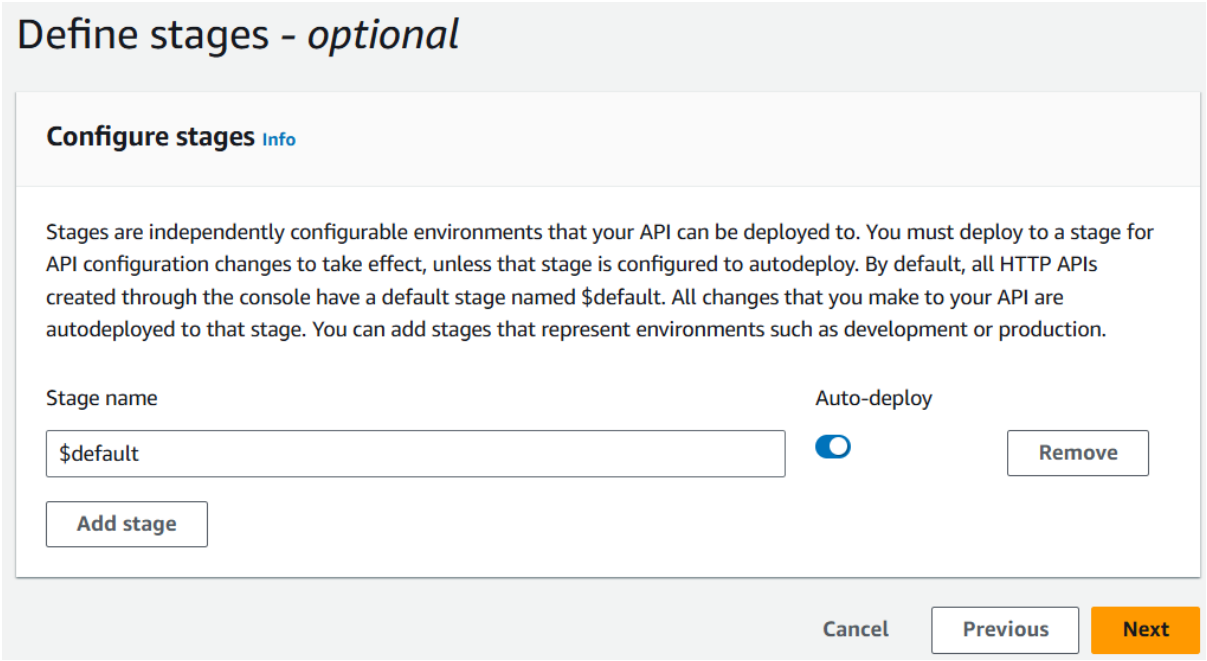
7. Then for the integration we need to choose Lambda and choose our lambda function then give a name to our API and click on Next.



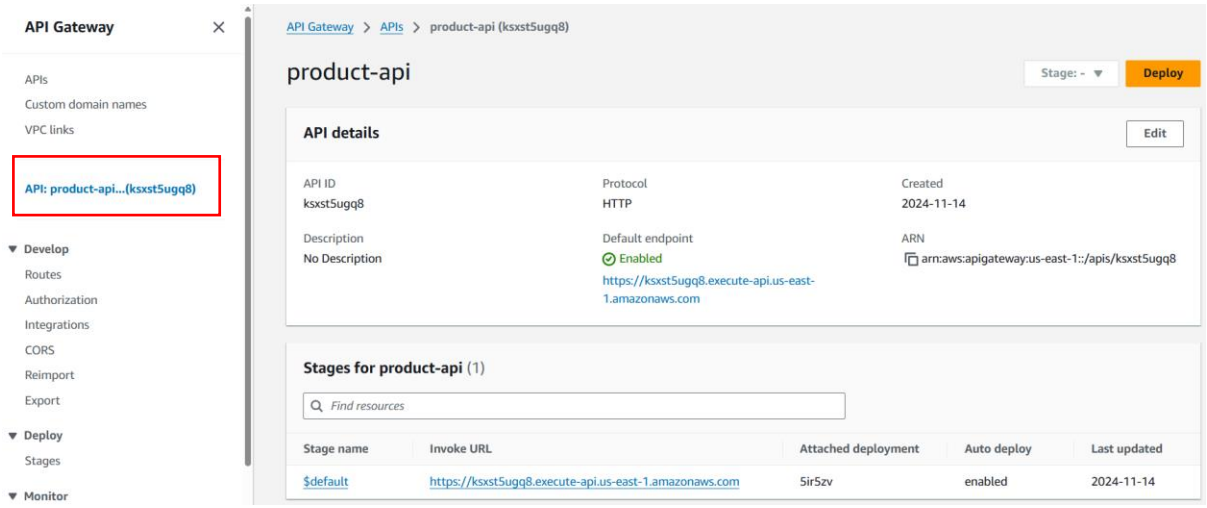
8. In step 2, we are going to configure routes as you can see below in the snapshot, we have to use some methods and you need to use the same methods to configure your routes.



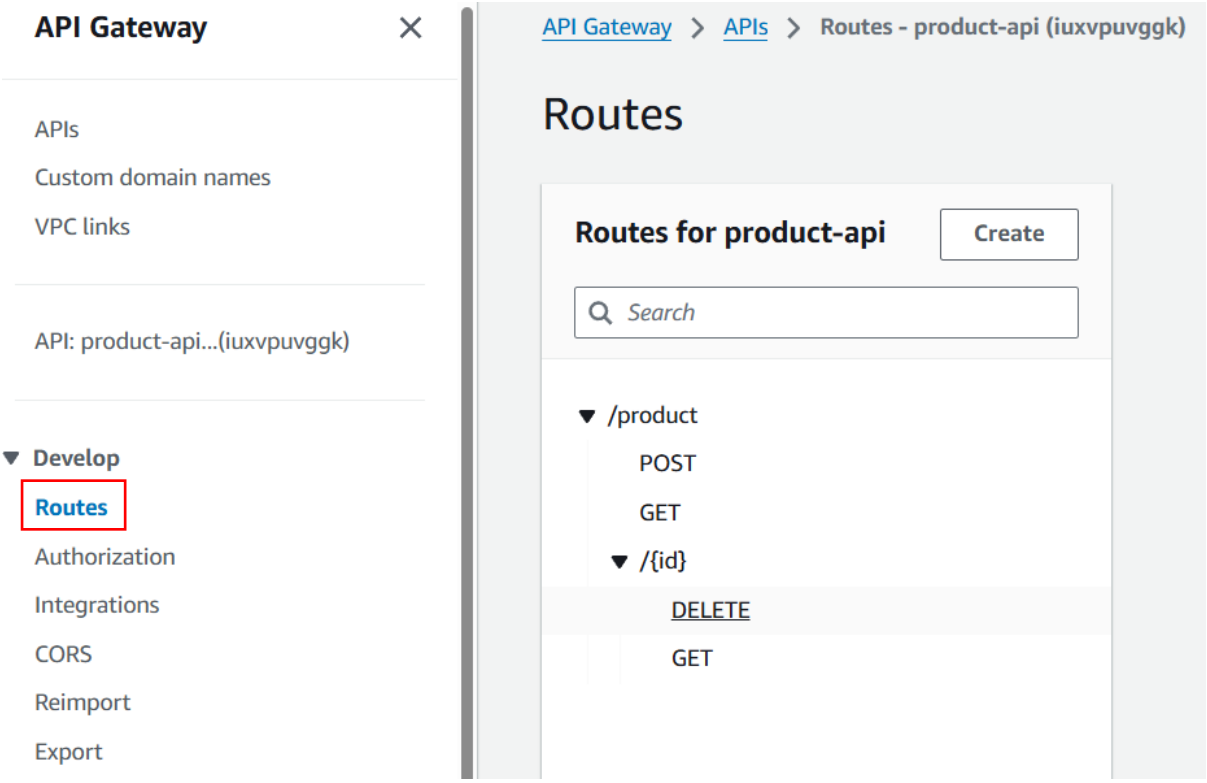
9. Now we are in the default stage and keep the stage name as it is. Move to review page and create your API.



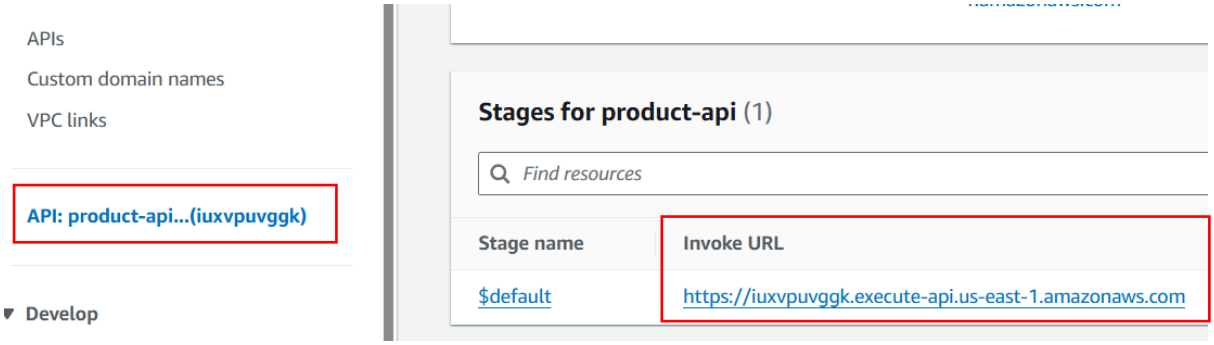
10. Once your API is created open it and you will see the details for it.



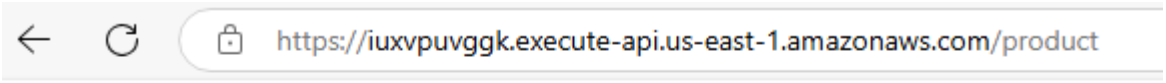
11. Also, you can go to the routes and here you will see that the configuration routes that you defined while creating it.



12. Now it is time to test our API. For that go to the API product and from the stages copy the Invoke URL.

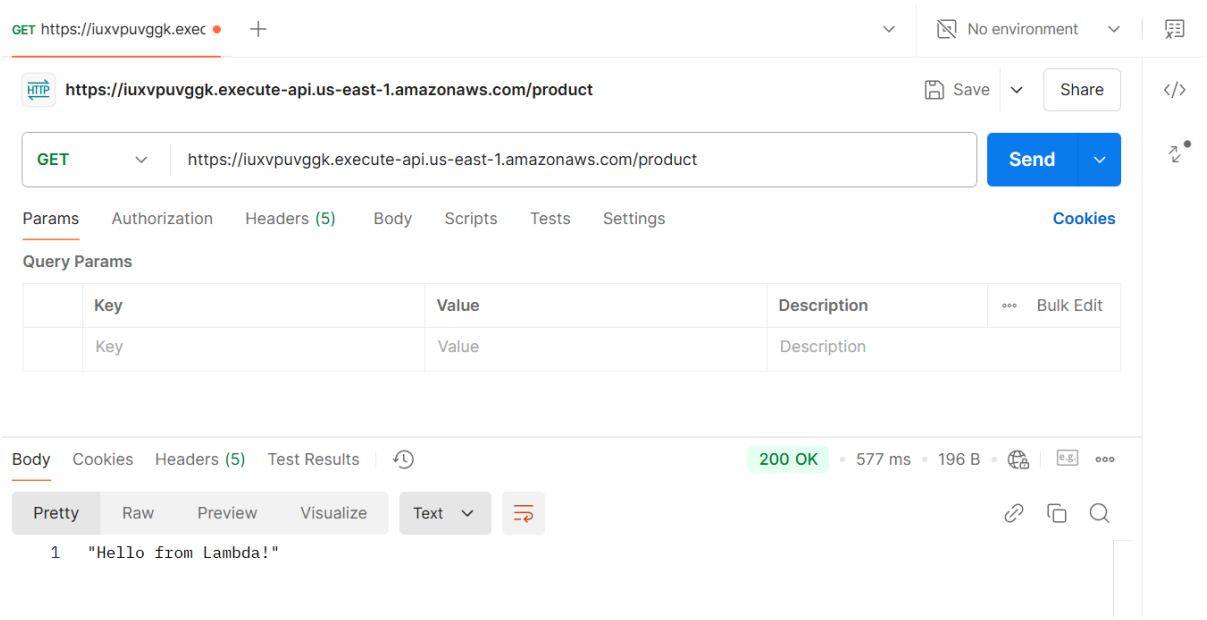


13. You need to append this invoke URL with /product and you will see that you are getting hello from lambda. So, this means that our API response is working properly.



"Hello from Lambda!"

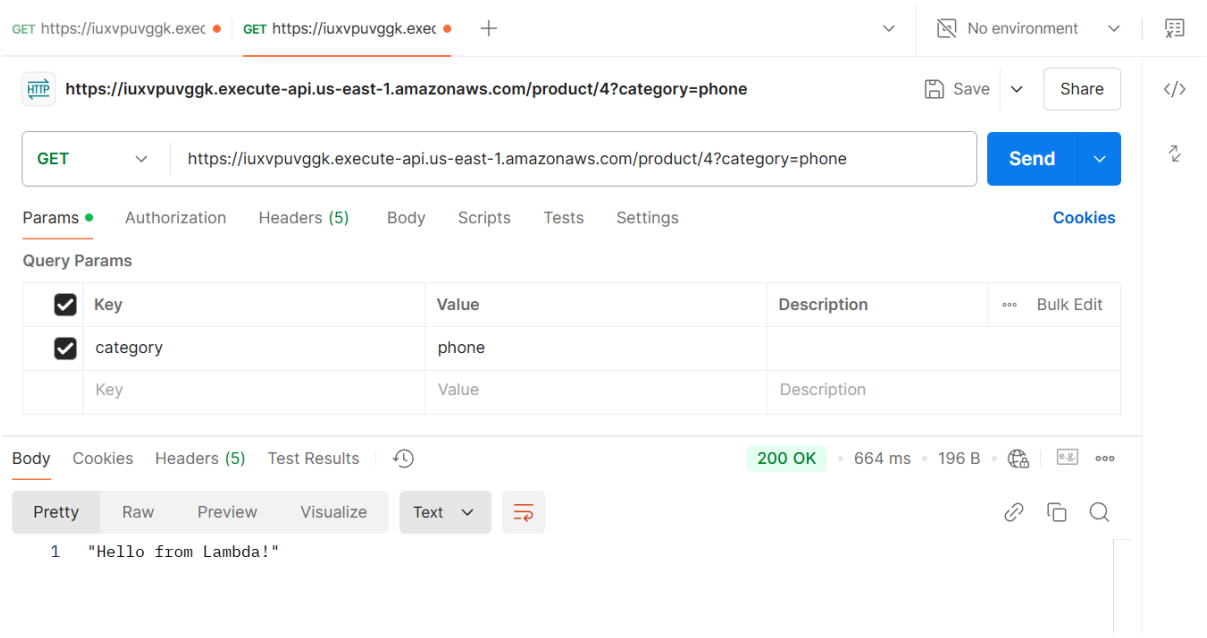
14. Now we are going to test our lambda function for all the test events. So, to do that open the Post Man tool on your laptop.
15. Then you need to paste the invoke URL using the GET method in your postman workspace and click on Send. You will see that you get a hello from lambda which means that it succeeded.



16. Now you can go to CloudWatch to open the log stream for your lambda function and here you will see the logs generated when you invoked it using the GET method in postman.

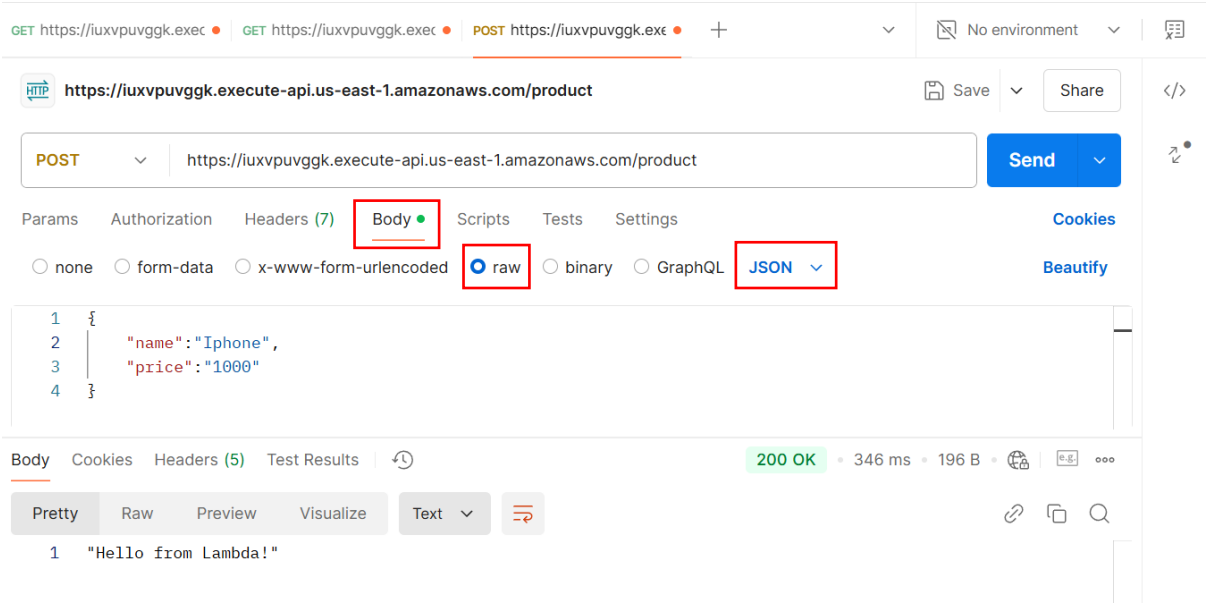
```
▼ 2024-11-14T16:28:55.853+05:30 2024-11-14T10:58:55.853Z 1e643321-3ce8-4a13-a2fb-71f7fccdcfa2 INFO event:

2024-11-14T10:58:55.853Z 1e643321-3ce8-4a13-a2fb-71f7fccdcfa2 INFO event:
{
    "version": "2.0",
    "routeKey": "GET /product",
    "rawPath": "/product",
    "rawQueryString": "",
    "headers": {
        "accept": "*/*",
        "accept-encoding": "gzip, deflate, br",
        "cache-control": "no-cache",
        "content-length": "0",
        "host": "iuxvpuvggk.execute-api.us-east-1.amazonaws.com",
        "postman-token": "823e107e-022f-4429-acbd-0f0e702be6d9",
        "user-agent": "PostmanRuntime/7.42.0",
        "x-amzn-trace-id": "Root=1-6735d7ef-1ffc5c6d79a73c1231feaa3a",
        "x-forwarded-for": "54.86.50.139",
        "x-forwarded-port": "443",
        "x-forwarded-proto": "https"
    },
    "requestContext": {
        "accountId": "878893308172",
        "apiId": "iuxvpuvggk",
        "domainName": "iuxvpuvggk.execute-api.us-east-1.amazonaws.com",
        "domainPrefix": "iuxvpuvggk",
        "http": {
            "method": "GET",
            "path": "/product",
            "protocol": "HTTP/1.1",
            "sourceIp": "54.86.50.139",
            "userAgent": "PostmanRuntime/7.42.0"
```


17. Then we created a new request for the GET method again in Postman but this time we appended the URL with an ID which is 4 here and then added a key and a value then click on Send.
18. As expected, you can see that we got the response.



19. Similarly, you can go to CloudWatch and view the latest log events.

20. In the end we created a Post method and as you can see, we got the response from lambda. As you can see in the snapshot for Post you have to choose body then choose raw and change it to JSON.



21. If you go and check your CloudWatch logs you will see an event Post method too.

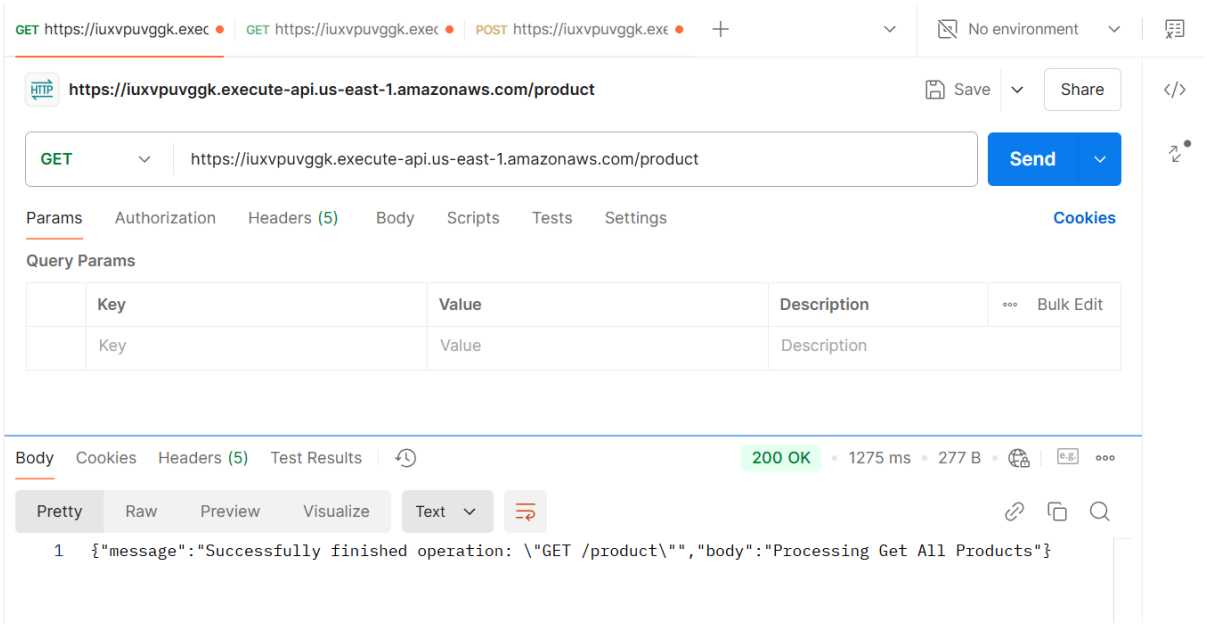
```
    },
    "requestContext": {
        "accountId": "878893308172",
        "apiId": "iuxvpuvggk",
        "domainName": "iuxvpuvggk.execute-api.us-east-1.amazonaws.com",
        "domainPrefix": "iuxvpuvggk",
        "http": {
            "method": "POST",
            "path": "/product",
            "protocol": "HTTP/1.1",
            "sourceIp": "54.86.50.139",
            "userAgent": "PostmanRuntime/7.42.0"
        },
        "requestId": "BO8Y7guBIAMEMGA=",
        "routeKey": "POST /product",
        "stage": "$default",
        "time": "14/Nov/2024:11:10:23 +0000",
        "timeEpoch": 1731582623179
    },
    "body": "{\r\n    \"name\":\"Iphone\",\r\n    \"price\":\"1000\"\r\n}",
    "isBase64Encoded": false
}
```

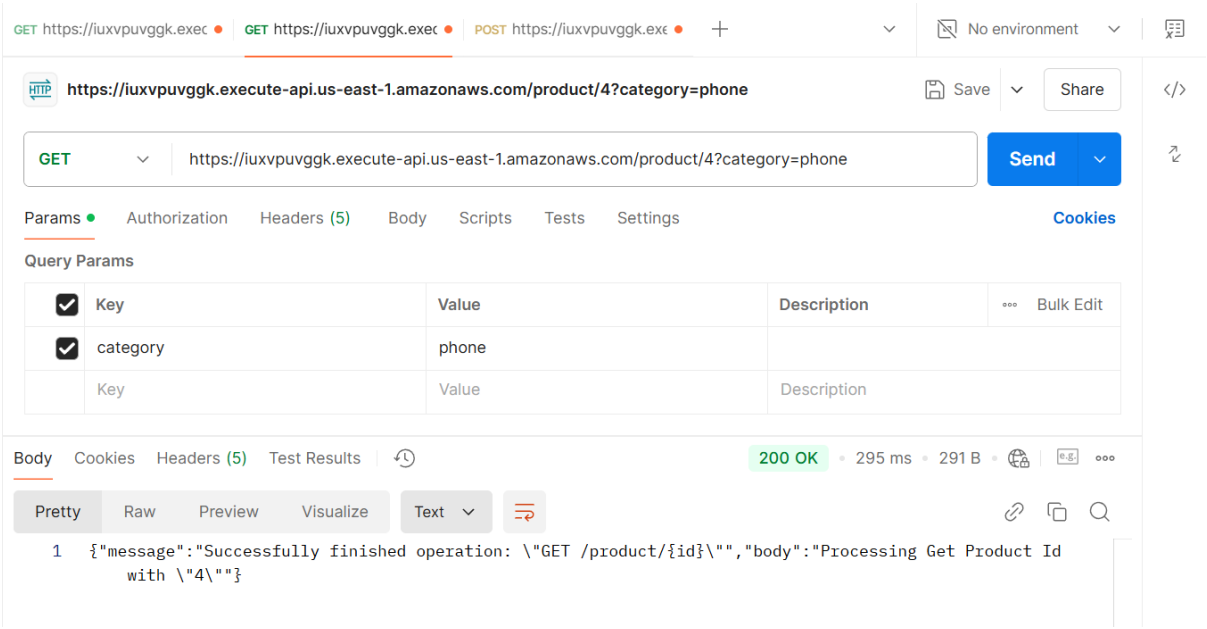

22. Now we are going to update the function code with the code given below and run the methods again in Postman.
23. So, go to your lambda function and update code then deploy it.

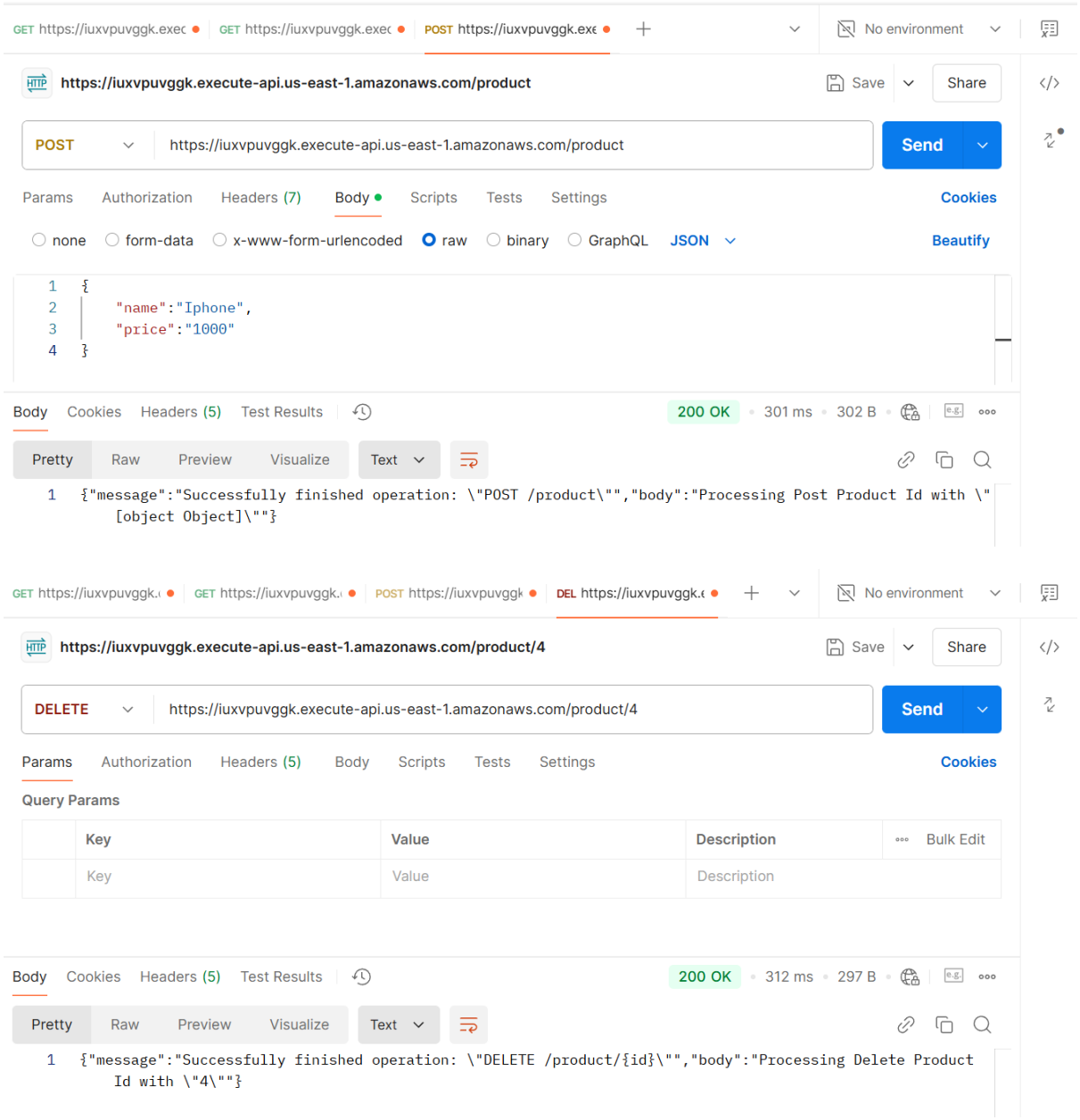
```
// Use ES Module (ESM) syntax to define the handler
export const handler = async (event) => {
  console.log('Received event:', JSON.stringify(event, null, 2));  // Log incoming event for debugging
  let body;

  try {
    // Switch based on the route key to determine the operation
    switch (event.routeKey) {
      case "GET /product":
        body = `Processing Get All Products`;  // Handle GET request to fetch all products
        break;
      case "GET /product/{id}":
        if(event.pathParameters != null) {
          body = `Processing Get Product Id with "${event.pathParameters.id}"`;  // Handle GET request for a specific product by ID
        }
        break;
      case "POST /product":
```

```
            let payload = JSON.parse(event.body);  // Parse incoming JSON body for POST request
            body = `Processing Post Product Id with "${payload}"`;  // Handle POST request to create a new product
            break;
        case "DELETE /product/{id}":
            if(event.pathParameters != null) {
              body = `Processing Delete Product Id with "${event.pathParameters.id}"`;  // Handle DELETE request for a specific product by ID
            }
            break;
        default:
            throw new Error(`Unsupported route: "${event.routeKey}"`);  // If route is not supported, throw an error
    }

    console.log(body);  // Log the body for debugging

    return {
        statusCode: 200,  // Return HTTP 200 OK for successful operations
        body: JSON.stringify({
            message: `Successfully finished operation: "${event.routeKey}"`,
            body: body
        })
    };

 } catch (e) {
    console.error(e);  // Log any error encountered
    return {
        statusCode: 400,  // Return HTTP 400 Bad Request for errors
        body: JSON.stringify({
            message: "Failed to perform operation.",
            errorMsg: e.message  // Return the error message for debugging
        })
    };
 }
};
```

24. Now you need to go to Postman and run the methods again. This time you will get the proper response.


GET https://iuxvpuvggk.exec
GET https://iuxvpuvggk.exec
POST https://iuxvpuvggk.exe
No environment
https://iuxvpuvggk.execute-api.us-east-1.amazonaws.com/product
Save
Share
GET
https://iuxvpuvggk.execute-api.us-east-1.amazonaws.com/product
Send
Params
Authorization
Headers (5)
Body
Scripts
Tests
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
Key
Value
Description
Body
Cookies
Headers (5)
Test Results
200 OK
1275 ms
277 B
Pretty
Raw
Preview
Visualize
Text
1 {"message":"Successfully finished operation: \"GET /product\"","body":"Processing Get All Products"}


GET https://iuxvpuvggk.exec
GET https://iuxvpuvggk.exec
POST https://iuxvpuvggk.exe
No environment
https://iuxvpuvggk.execute-api.us-east-1.amazonaws.com/product/4?category=phone
Save
Share
GET
https://iuxvpuvggk.execute-api.us-east-1.amazonaws.com/product/4?category=phone
Send
Params
Authorization
Headers (5)
Body
Scripts
Tests
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
category
phone
Key
Value
Description
Body
Cookies
Headers (5)
Test Results
200 OK
295 ms
291 B
Pretty
Raw
Preview
Visualize
Text
1 {"message":"Successfully finished operation: \"GET /product/{id}\"","body":"Processing Get Product Id with \"4\""}



25. From the above snapshots we can say that our CRUD API is running successfully.
26. Once you are done with the lab delete all the resources. Your Lambda function and your HTTP API.