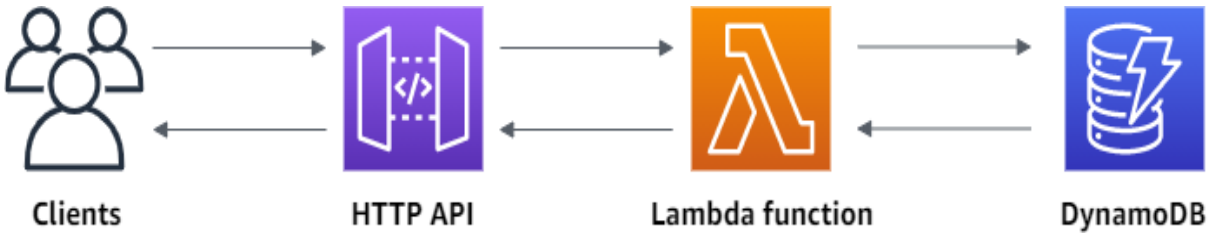
Build a CRUD API with Lambda and DynamoDB

Lifecycle of the Serverless API -



In this project, we create a serverless API that creates, reads, updates, and deletes items from a DynamoDB table we have created.

First, We create a DynamoDB table using the AWS DynamoDB console. Then you create a Lambda function using the AWS Lambda console. Next, we create an HTTP API using the API Gateway console. Lastly, we test the API we have created.

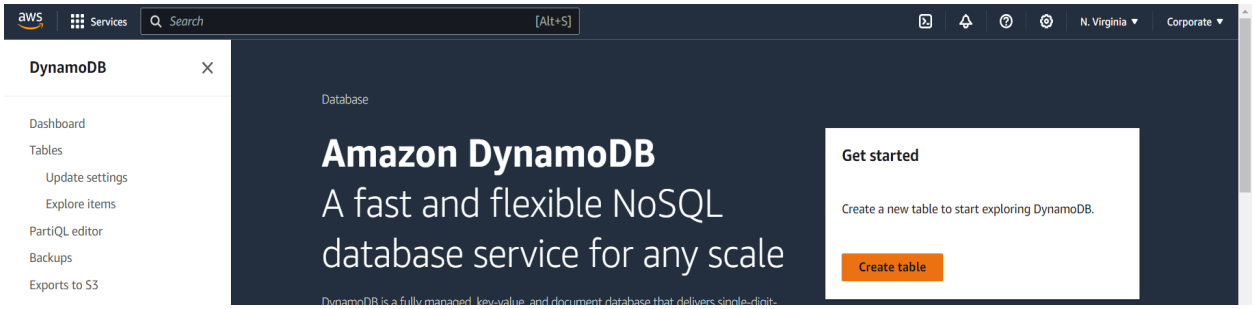
When we invoke the HTTP API, the API Gateway routes the request to our Lambda function. The Lambda function then interacts with DynamoDB, and returns a response to the API Gateway. The API Gateway then returns a response to the client/customer.

Step 1 : Create a DynamoDB table

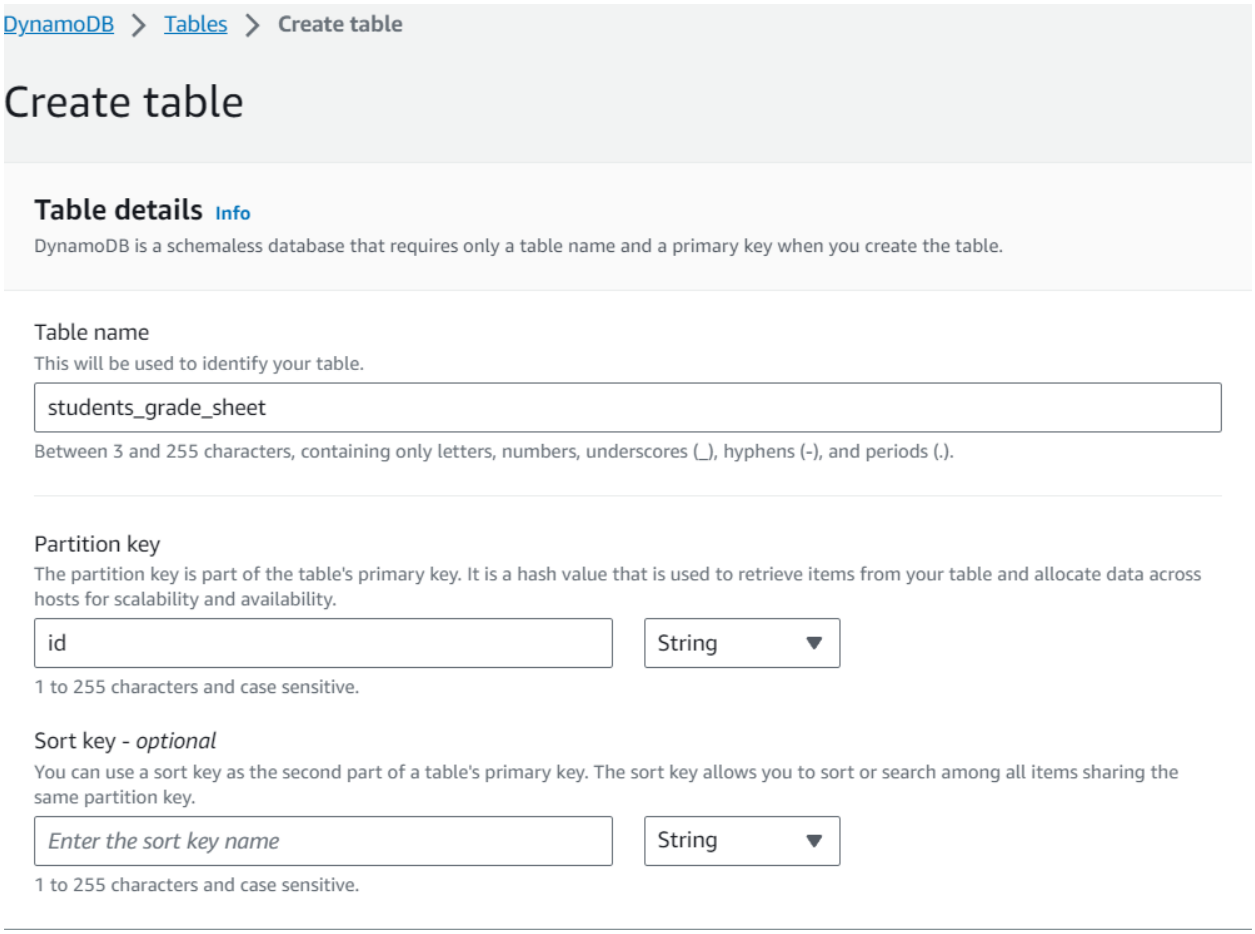
- You use a **DynamoDB table** to store/retrieve data for the **API**.
- Each item has a **unique ID**, which we use as the **partition key** for the table.

Steps to create table-

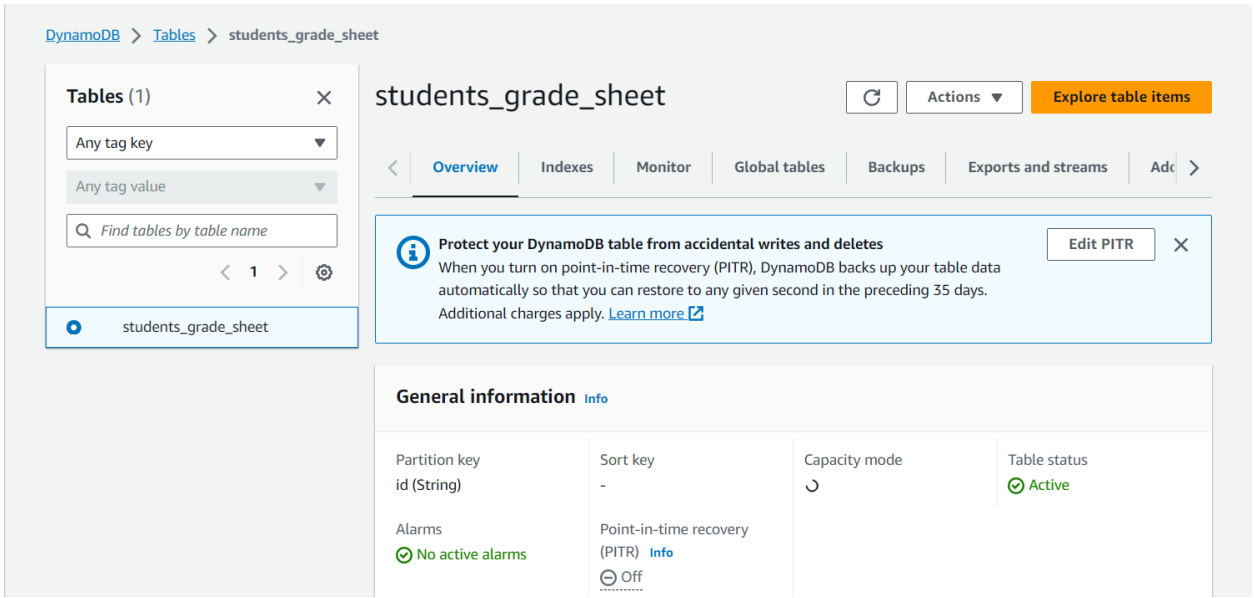
1. Login to the **AWS DynamoDB console** and click on **Create table**.



2. Enter the details for the **DynamoDB table** as mentioned in the below image.



3. Once the **DynamoDB table** has been created we can see the overview of the table as seen in the below image.

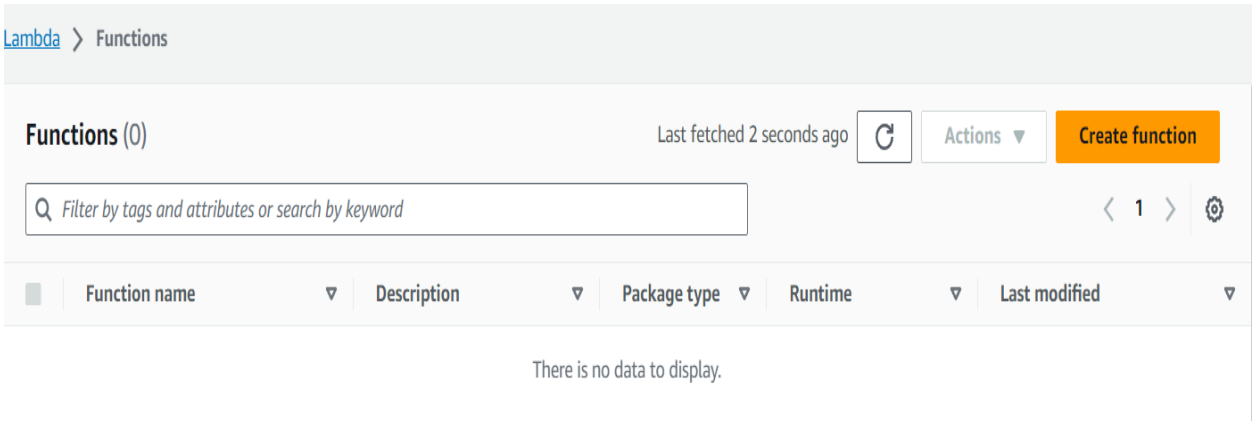


Step 2: Create a Lambda function

We create a Lambda function for the backend of our API. This Lambda function creates, reads, updates, and deletes items from the DynamoDB table.

Steps to create Lambda function -

1. Sign in to the **Lambda console** and Choose **Create function**.



2. For Function name, enter **students_grade_sheet_function**

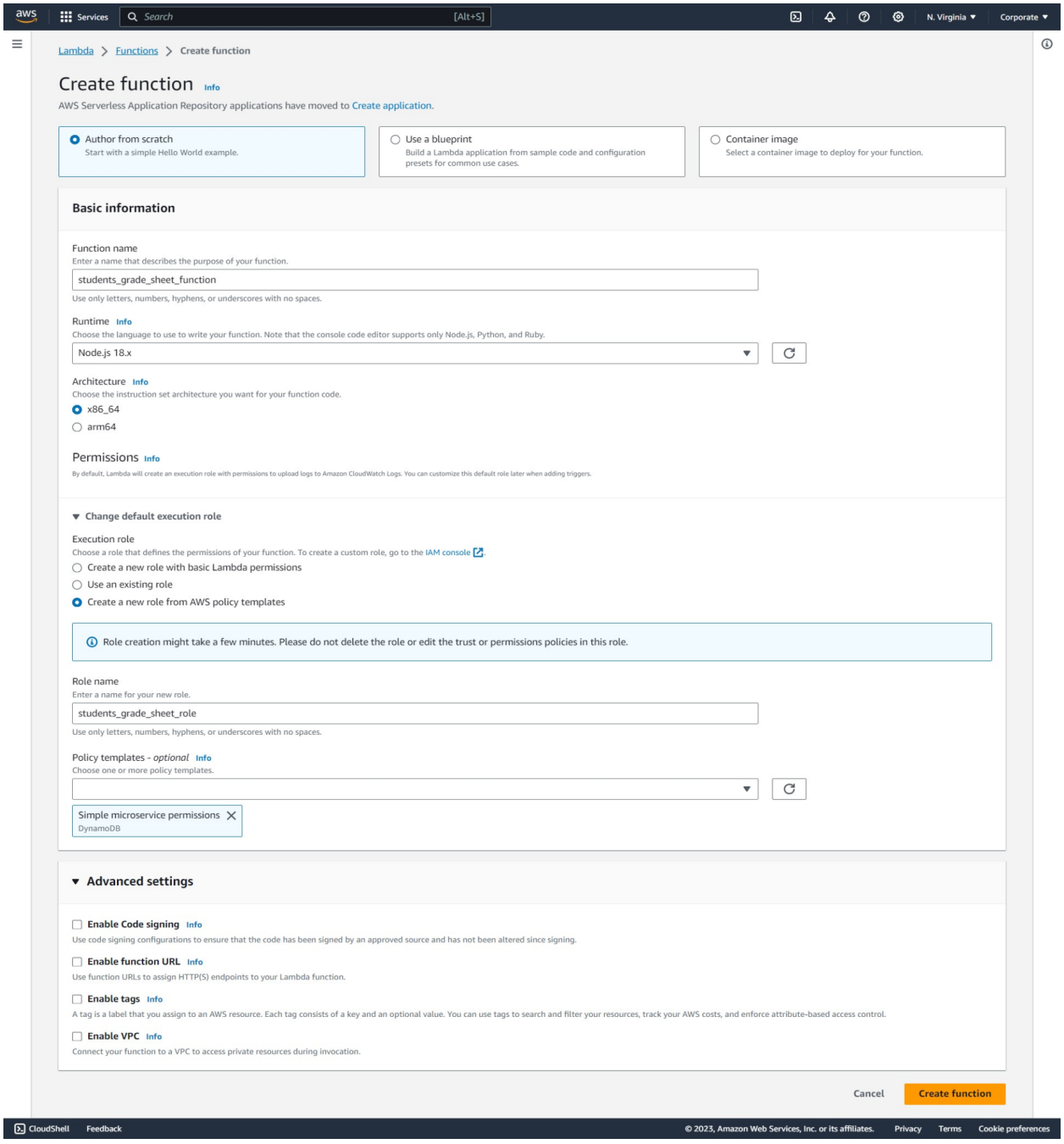
3. Under **Permissions** choose **Change default execution role**.

4. Select Create a new role from **AWS policy templates**.

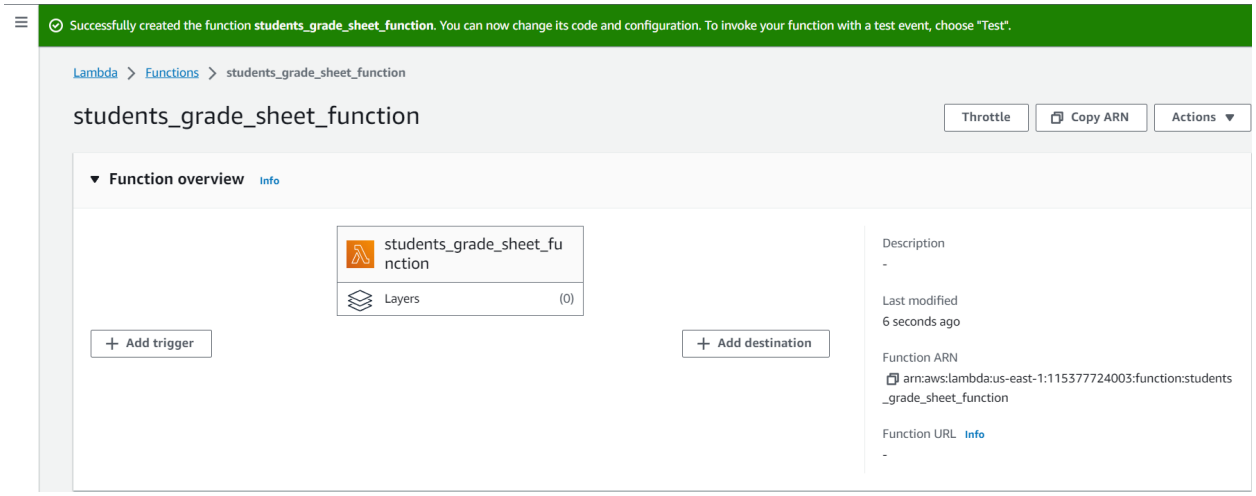
5. For Role name, enter **students_grade_sheet_role**.

6. For Policy templates, choose **Simple microservice permissions**. This policy grants the **Lambda function** permission to interact with **DynamoDB**.

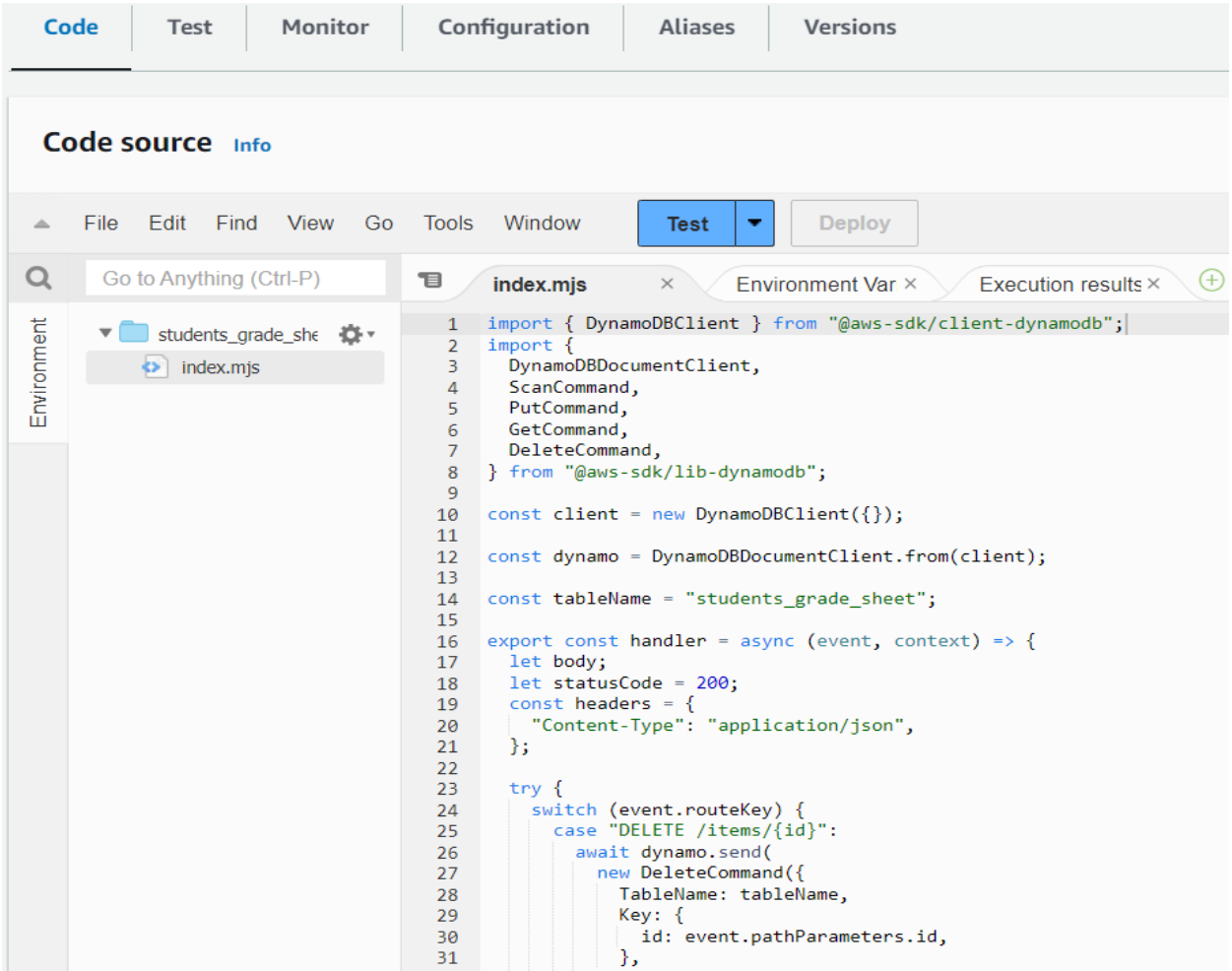
7. Choose **Create function**.



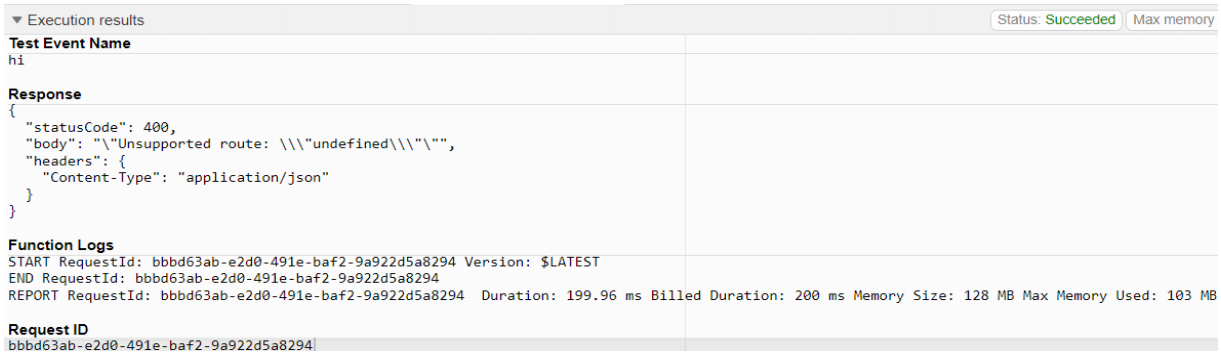
8. Below image we can see our **Lambda function** we have just created.



9. Open **`index.mjs`** in the Lambda function's Code source, and replace its contents with the following code. Choose **Deploy** to update the function.



Test Result of the Lambda Code -




```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import {
  DynamoDBDocumentClient,
  ScanCommand,
  PutCommand,
  GetCommand,
  DeleteCommand,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});

const dynamo = DynamoDBDocumentClient.from(client);

const tableName = "students_grade_sheet";

export const handler = async (event, context) => {
  let body;
  let statusCode = 200;
  const headers = {
    "Content-Type": "application/json",
  };

  try {
    switch (event.routeKey) {
      case "DELETE /items/{id}":
        await dynamo.send(
          new DeleteCommand({
            TableName: tableName,
            Key: {
              id: event.pathParameters.id,
            },
          })
        );
        body = `Deleted item ${event.pathParameters.id}`;
        break;
      case "GET /items/{id}":
        body = await dynamo.send(
          new GetCommand({
            TableName: tableName,
            Key: {
              id: event.pathParameters.id,
            },
          })
        );
        body = body.Item;
        break;
      case "GET /items":
        body = await dynamo.send(
          new ScanCommand({ TableName: tableName })
        );
        body = body.Items;
        break;
      case "PUT /items":
        let requestJSON = JSON.parse(event.body);
        await dynamo.send(
          new PutCommand({
            TableName: tableName,
            Item: {
              id: requestJSON.id,
              marks: requestJSON.marks,
              name: requestJSON.name,
            },
          })
        );
        body = `Put item ${requestJSON.id}`;
        break;
      default:
        throw new Error(`Unsupported route: "${event.routeKey}"`);
    }
  } catch (err) {
    statusCode = 400;
    body = err.message;
  } finally {
    body = JSON.stringify(body);
  }

  return {
    statusCode,
    body,
    headers,
  };
};
```

Above code is to be put into the **index.mjs** file of the **lambda code source**.

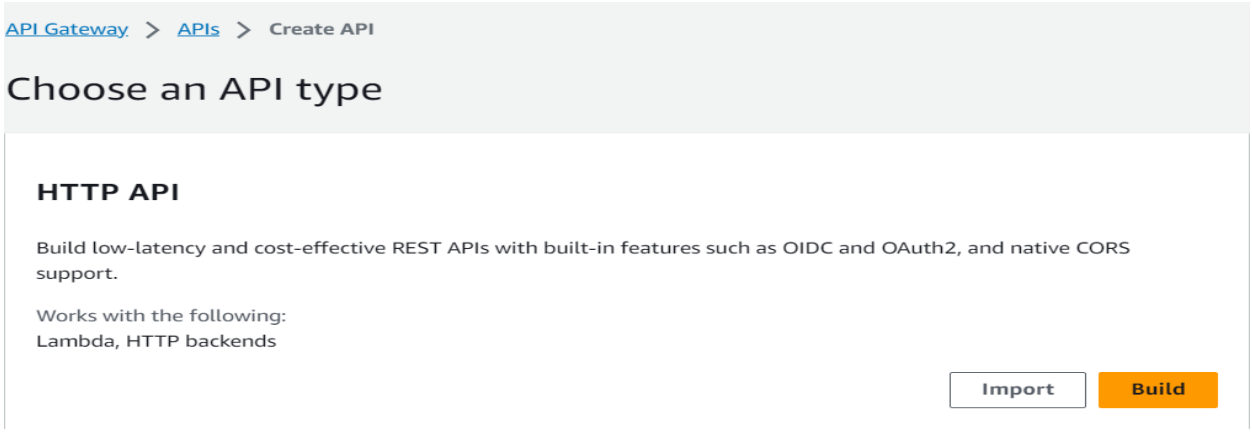
Step 3: Create a HTTP API

The HTTP API provides a HTTP endpoint for your Lambda function. In this step, we create an empty API.

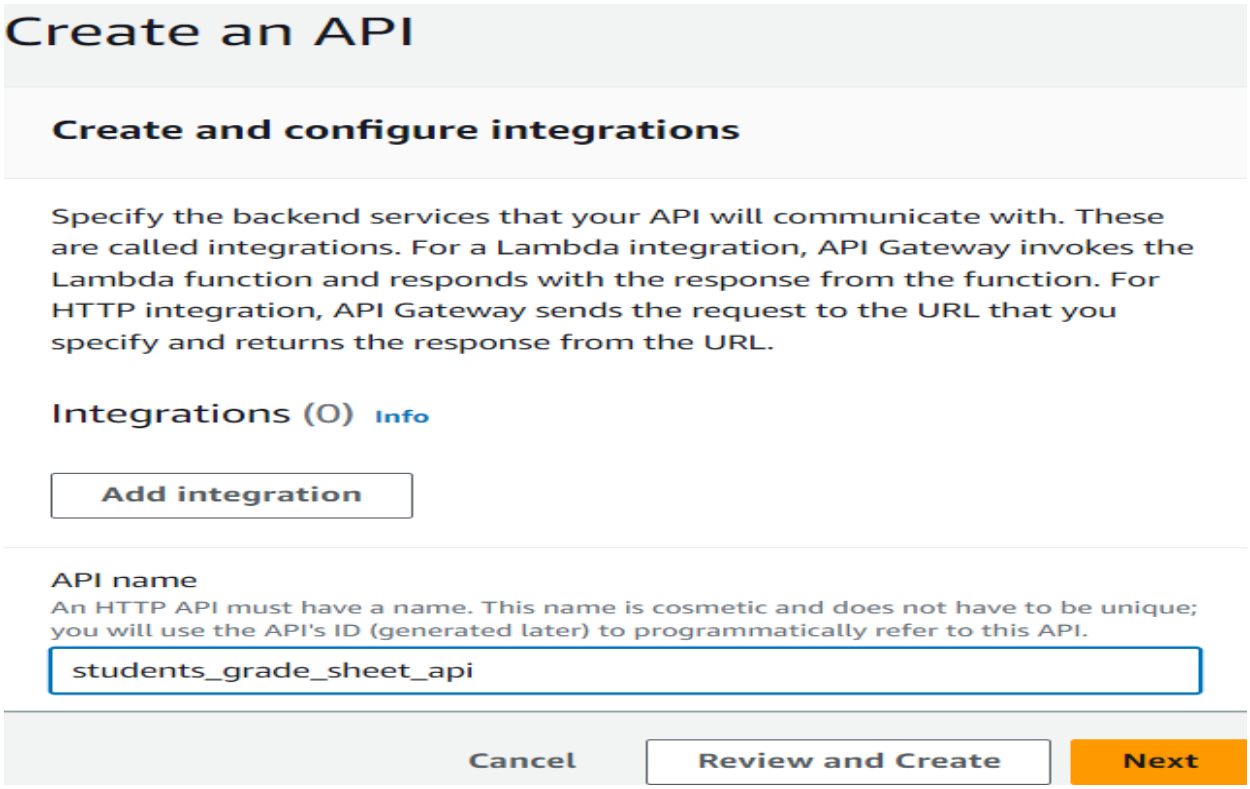
To create an **HTTP API** -

1. Go to **API Gateway**

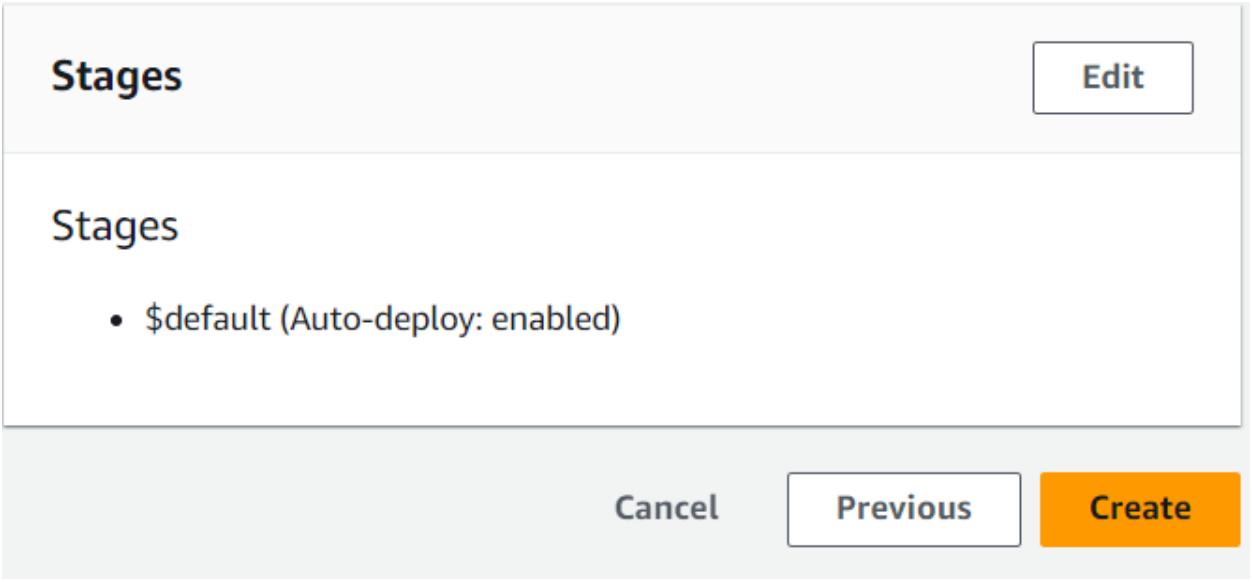
2. Choose **Create API**, and then for **HTTP API**, choose **Build.**



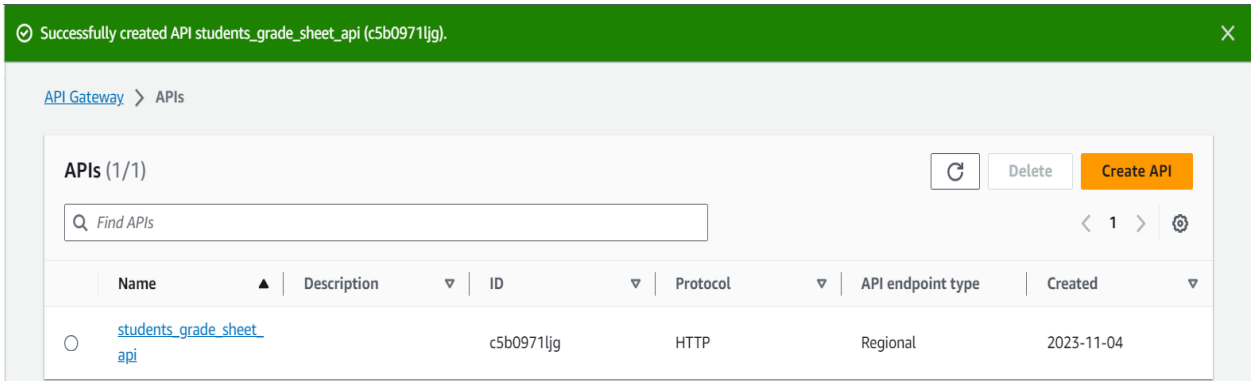
3. For **API name**, enter **students_grade_sheet_api.**



4. Choose Next.
5. For **Configure routes**, choose Next to **skip route creation**. You create routes later.
6. **Review the stage** that **API Gateway** creates for you, and then choose Next.



7. Choose **Create**.



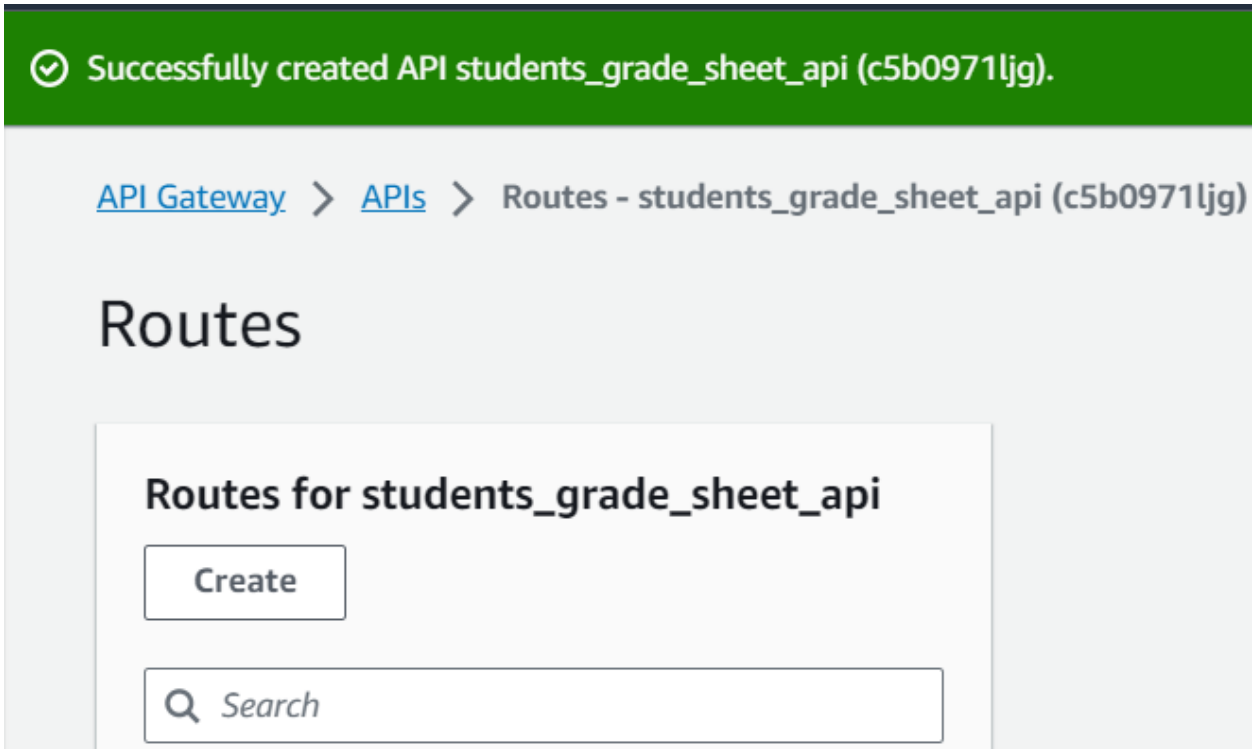
8. Above image shows our successful **HTTP API Creation.**

Step 4: Create routes

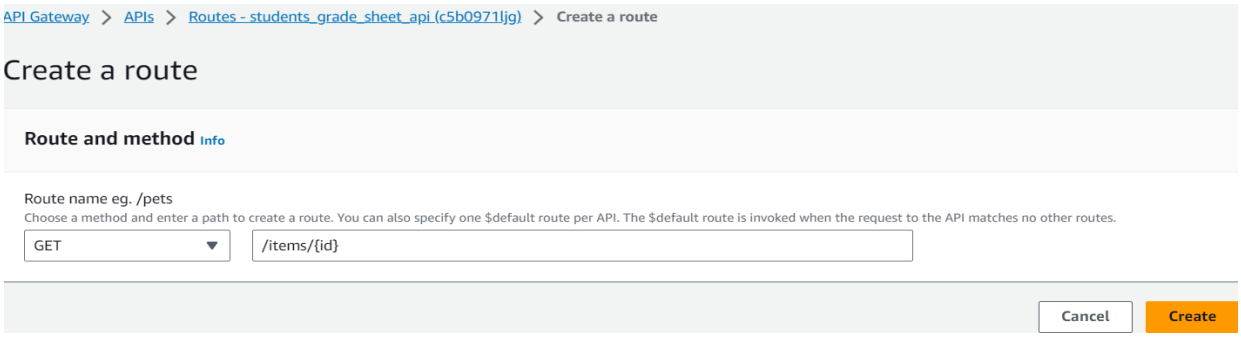
Routes are a way to send incoming API requests to backend resources.
Routes consist of two parts: an **HTTP method** and a **resource path**.

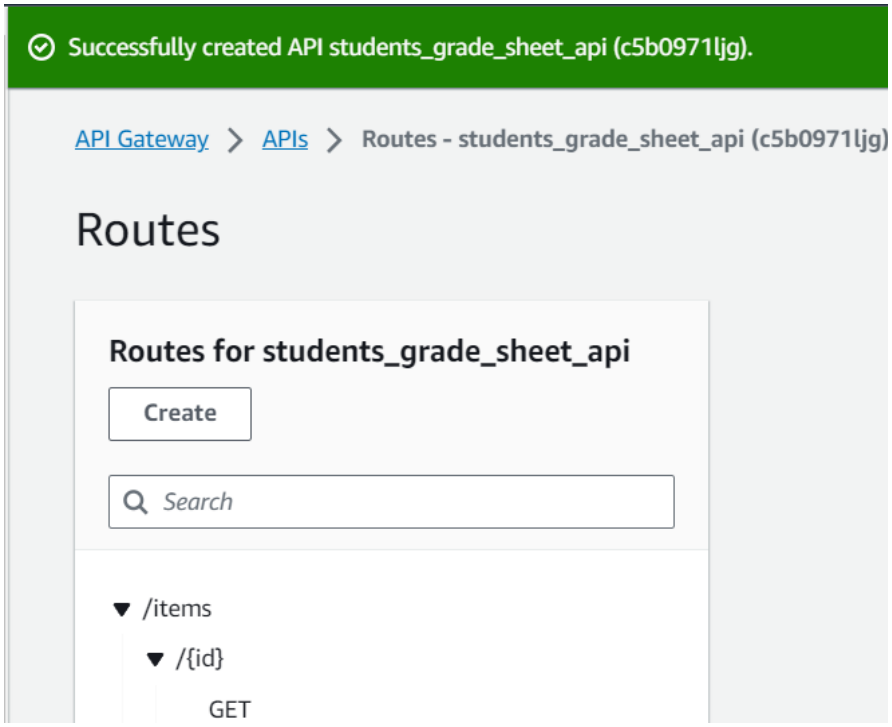
To create Routes -

1. Choose the **API** we have created.
2. Choose **Routes**.
3. Choose **Create**.

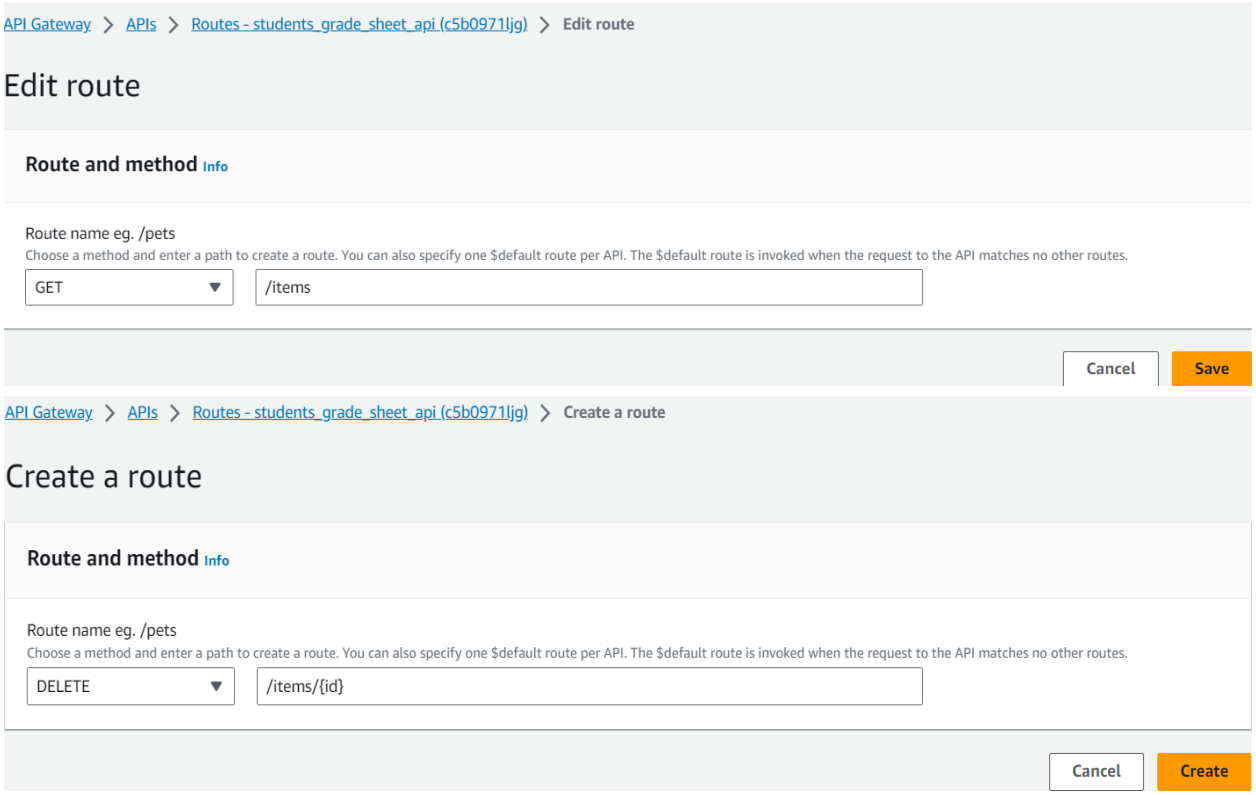


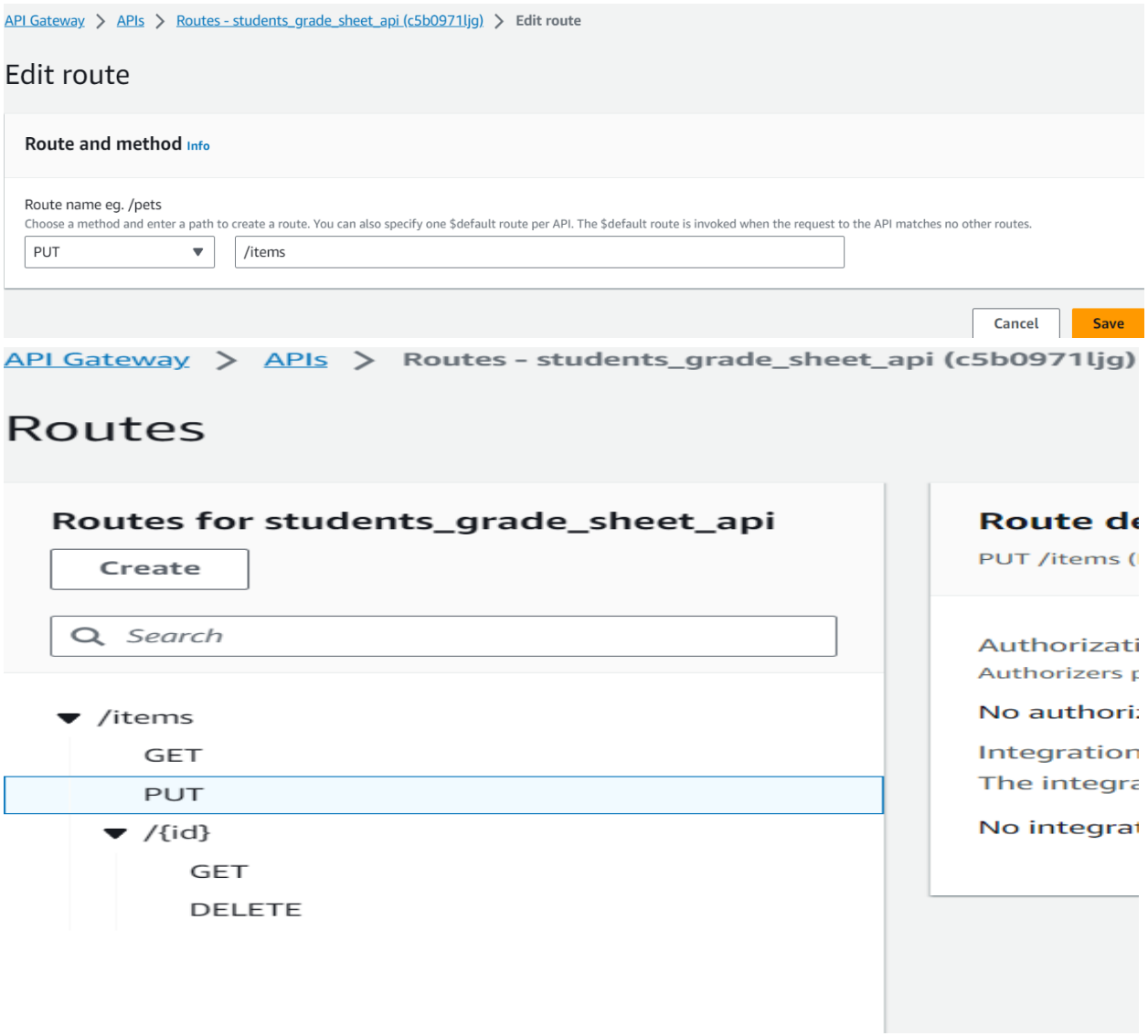
4. For Method, choose **`GET`**.
5. For the path, enter **`/items/{id}`**. The **`{id}`** at the end of the path is a path parameter that API Gateway retrieves from the request path when a client makes a request.
6. Choose **Create**.





7. Repeat steps 3-6 for **`GET /items`**, **`DELETE /items/{id}`** and **`PUT /items`**.





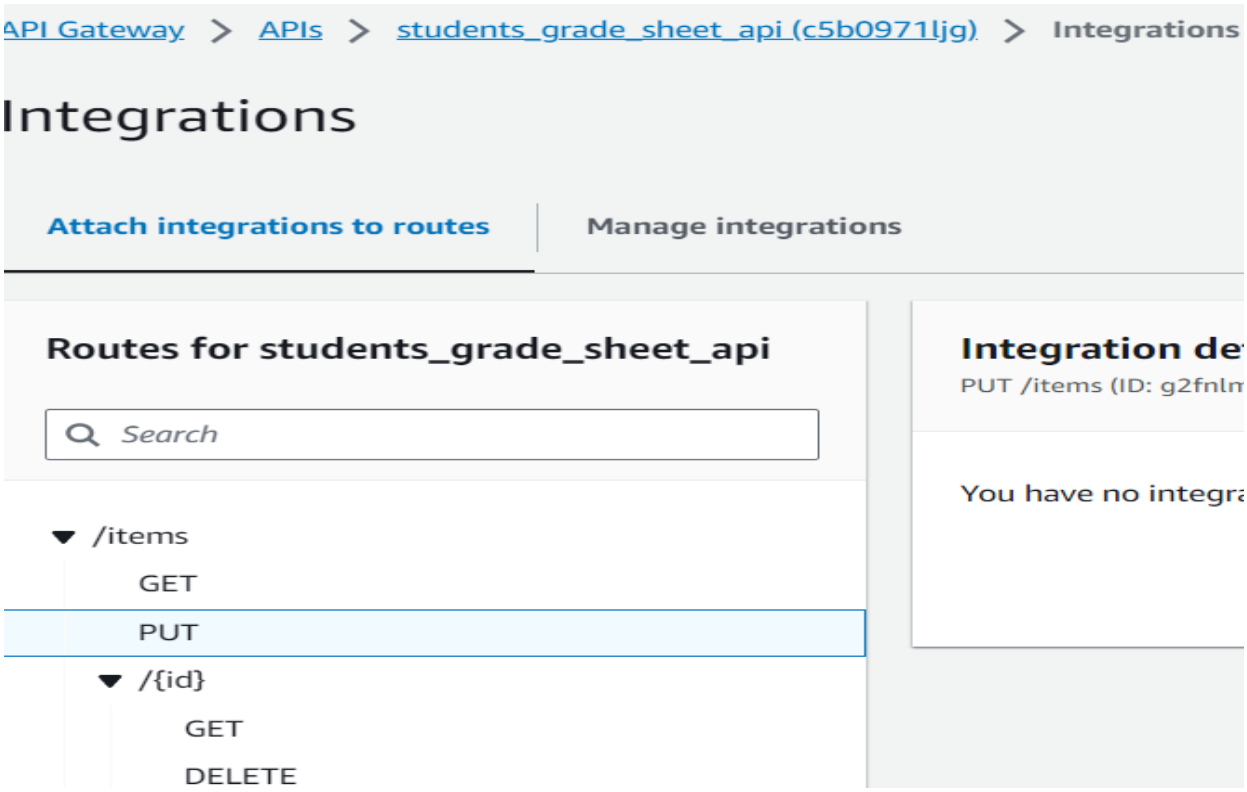
8. Above **Routes** have been created by us inside the **HTTP API**.

Step 5: Create an integration

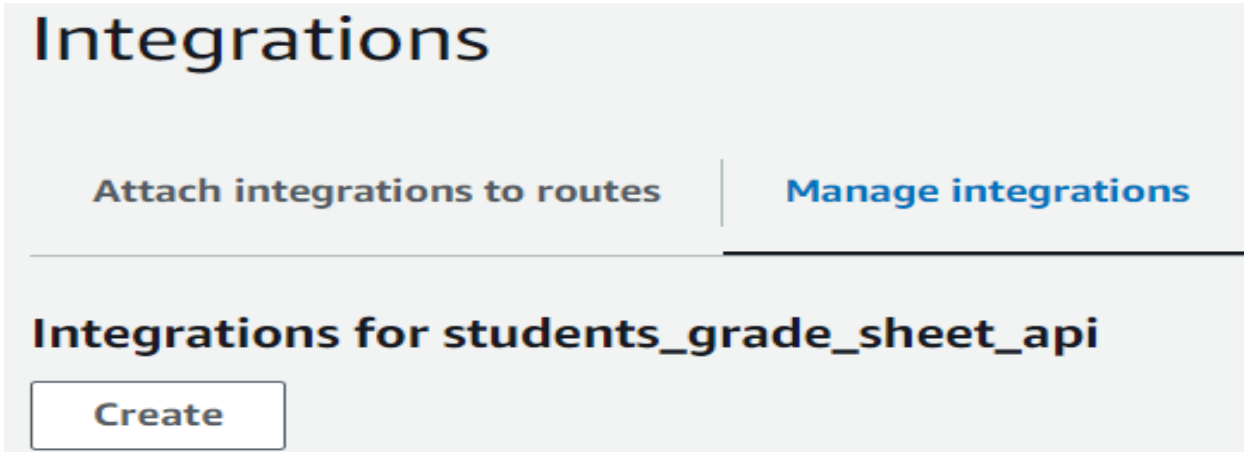
We create an integration to connect a route to the backend resources. For our API, we create one Lambda integration that we use for all routes.

To create an integration -

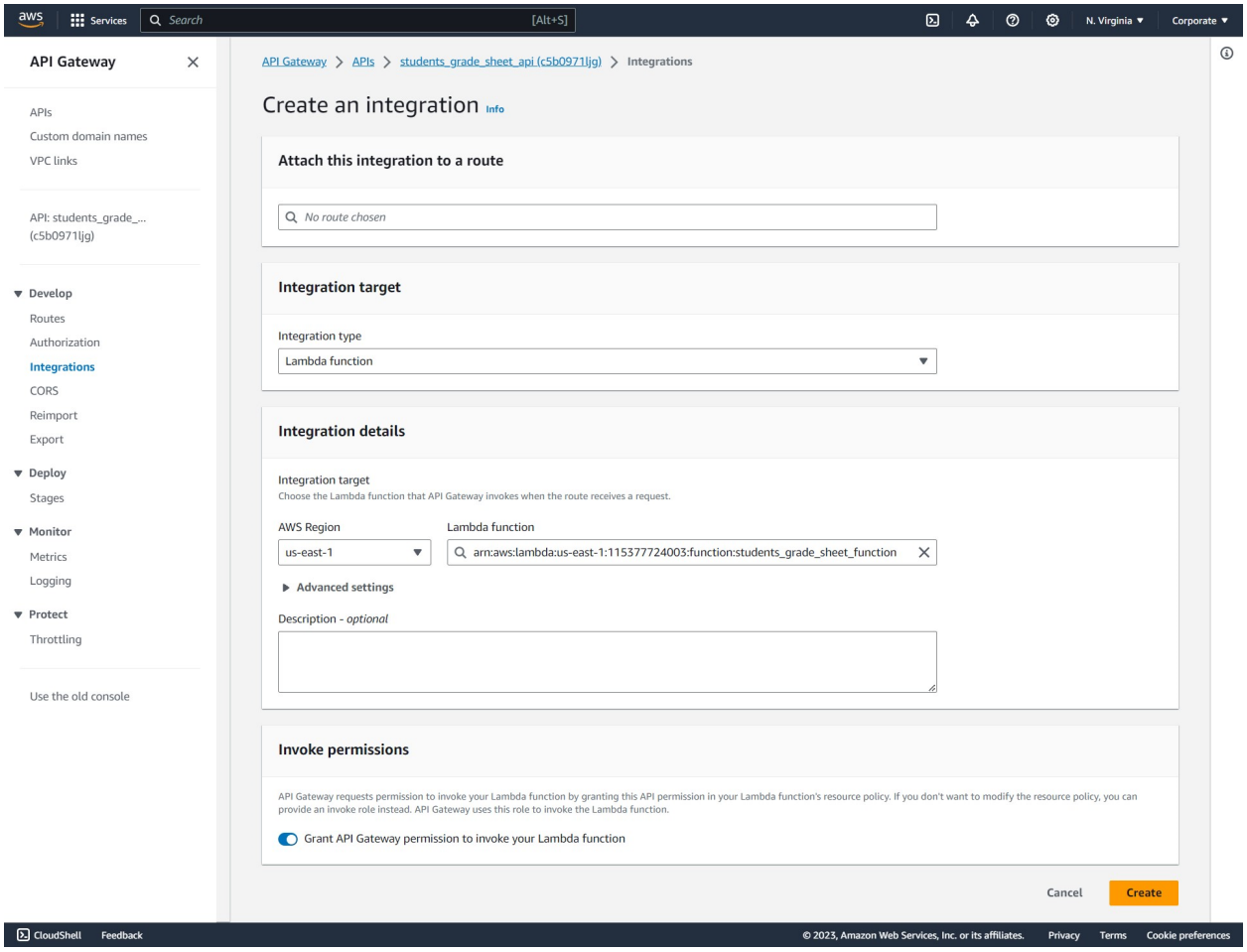
1. Choose your **API**.
2. On the left side pane, under the **Develop** dropdown - **Choose Integrations**.



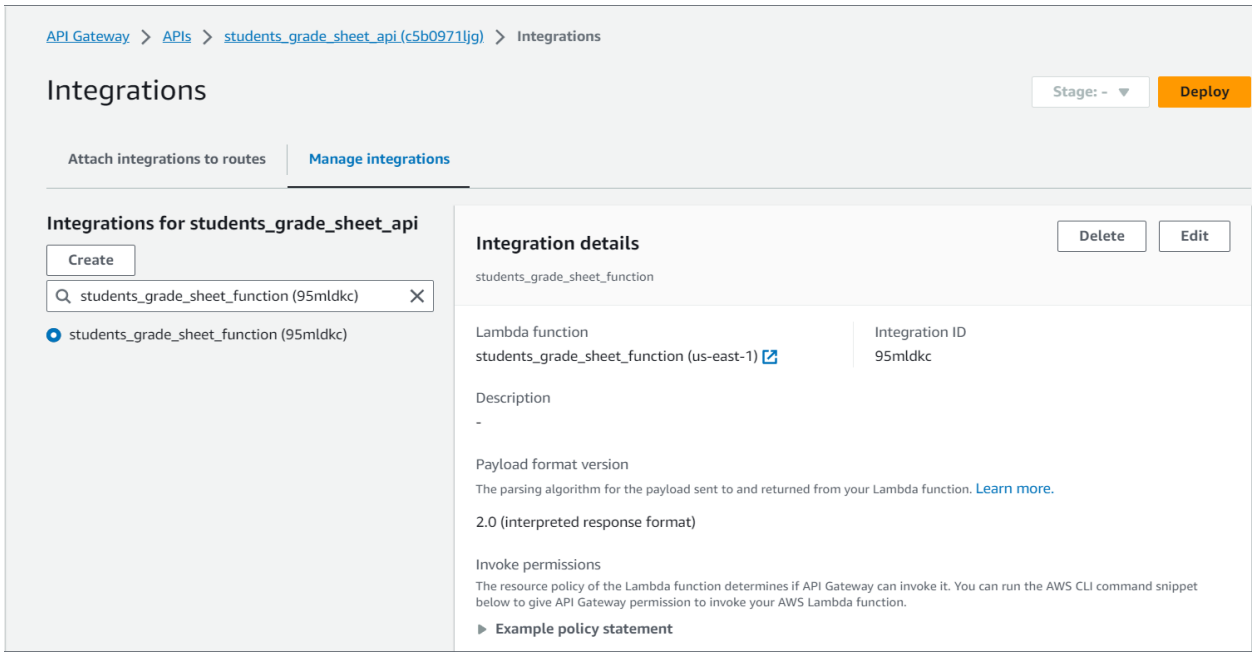
3. Choose **Manage integrations** and then choose Create.



4. Skip Attach this integration to a route. We will perform this in a later step.
5. For **Integration type**, choose **Lambda function**.
6. For **Lambda function**, enter **students_grade_sheet_function**.



7. Choose **Create**.

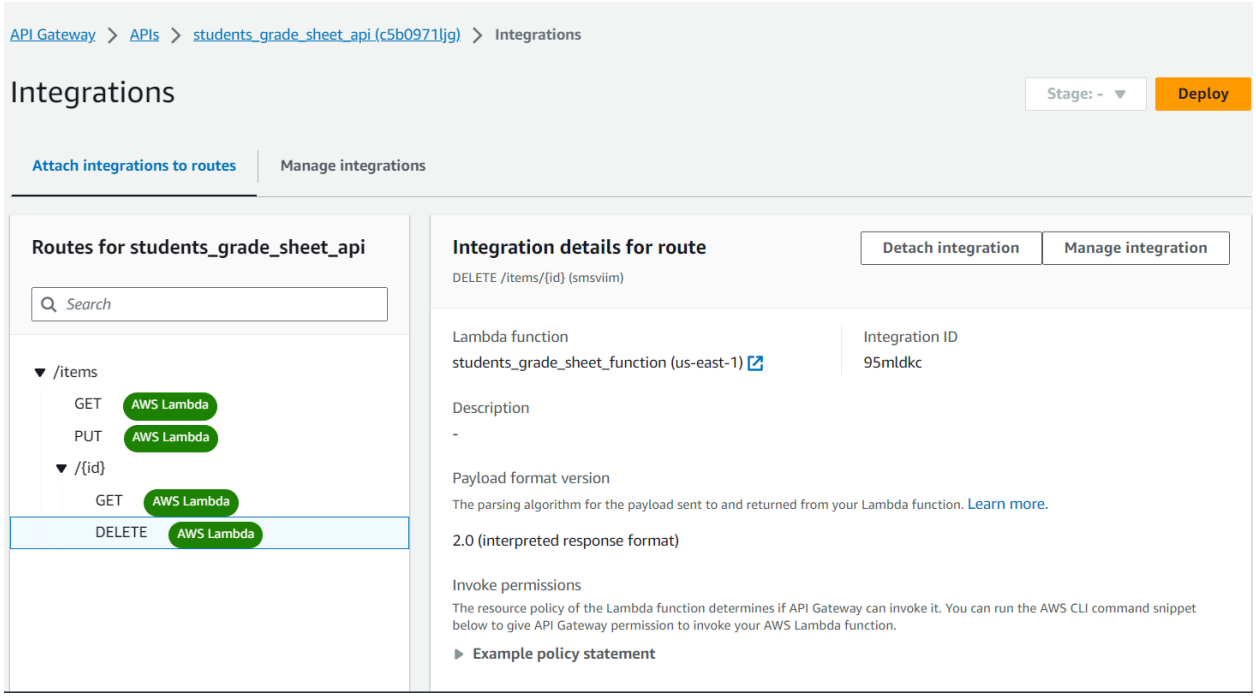


Step 6: Attach your integration to routes

For our HTTP API, we use the same Lambda integration for all routes. After we attach the integration to all of the API's routes, our Lambda function is invoked whenever a client calls any of our routes.

To attach **Integrations to routes** -

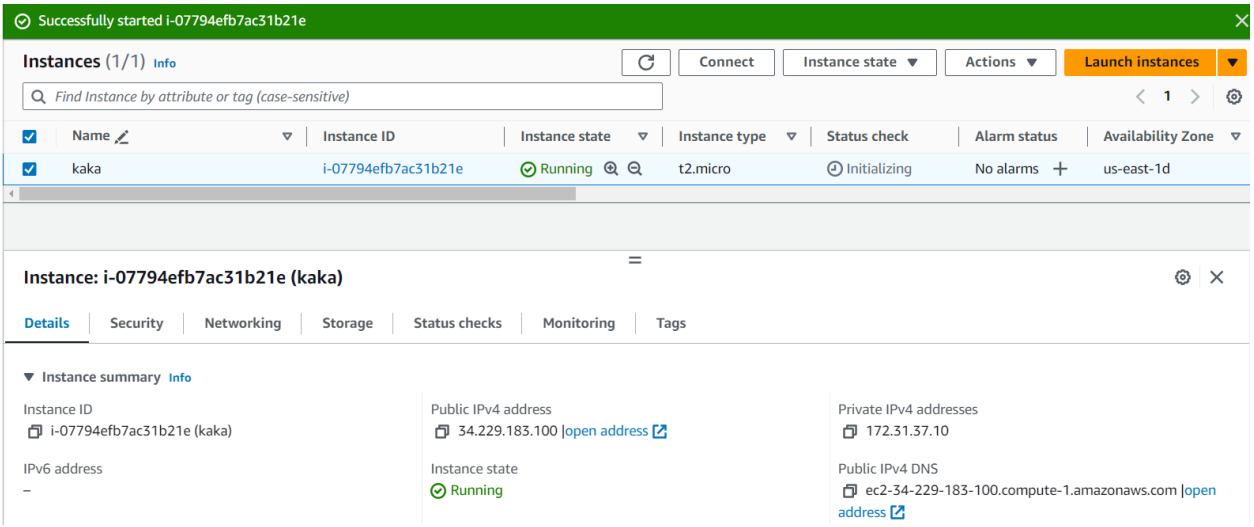
1. On the left side pane, under the **Develop** dropdown - Choose **Integrations**.
2. Choose route (**PUT**).
3. Under **Choose an existing integration**, choose - **students_grade_sheet_function** (the Lambda function we have previously created)
4. Choose **Attach integration**.
5. Repeat steps 2-4 for all routes.



All above routes are now integrated into our AWS Lambda function.
Now that we have a **HTTP API** with proper routes and integrations, we can finally test our CRUD API.

Step 7: Test your API

Create an EC2 instance in AWS and login into the instance through MobaXterm software.

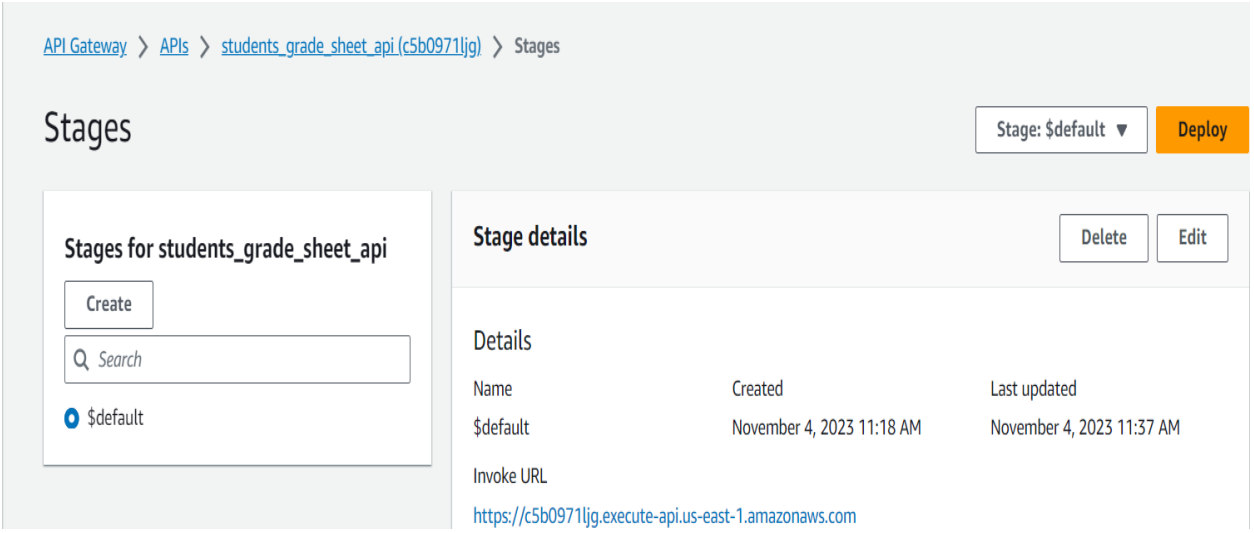


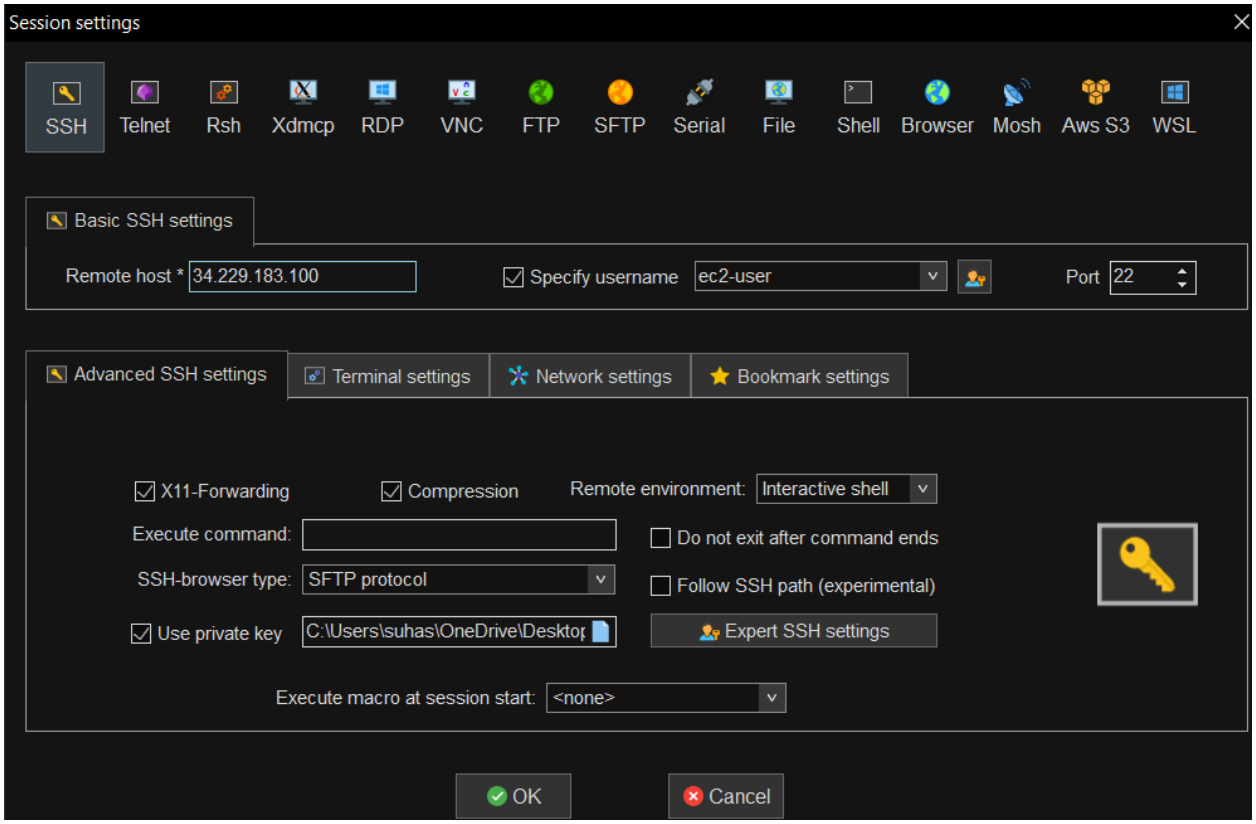
To make sure that our API is working, use **Curl**.

To get the URL to invoke our API -

1. Choose the **API** we have created.
2. Note your API's **Invoke URL**. (On left side pane, Under **Deploy- Stages -** Select our **default stage** and copy the **Invoke URL**)

https://c5b0971ljg.execute-api.us-east-1.amazonaws.com/ (our API Invoke URL)



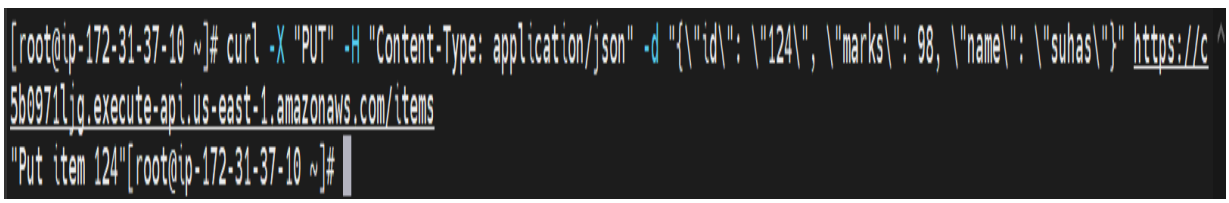


3. We login to our EC2 Instance to test out API calls.

4. To create or update an item -

Use the following command to create or update an item to our DynamoDB table. The command includes a request body with the item's ID, price, and name.

> curl -X "PUT" -H "Content-Type: application/json" -d "{\"id\": \"124\", \"marks\": 98, \"name\": \"suhas\"}" https://c5b0971ljg.execute-api.us-east-1.amazonaws.com/items



> curl -X "PUT" -H "Content-Type: application/json" -d "{\"id\": \"125\", \"marks\":85, \"name\": \"gandhi\"}" https://c5b0971ljg.execute-api.us-east-1.amazonaws.com/items

```
"Put item 124"[root@ip-172curl -X "PUT" -H "Content-Type: application/json" -d "{\"id\": \"125\", \"marks\":85, \"name\": \"gandhi\"}" https://c
5b0971ljg.execute-api.us-east-1.amazonaws.com/items
"Put item 125"[root@ip-172-31-37-10 ~]
```

5. To get all items -

Use the following command to list all items.

> curl https://c5b0971ljg.execute-api.us-east-1.amazonaws.com/items

```
[root@ip-172-31-37-10 ~]# curl https://c5b0971ljg.execute-api.us-east-1.amazonaws.com/items
[{"id":"124","name":"suhas","marks":98},{"id":"125","name":"gandhi","marks":85}][root@ip-172-31-37-10 ~]#
```

6. To delete an item -

Use the following command to delete an item.

> curl -X "DELETE"
https://c5b0971ljg.execute-api.us-east-1.amazonaws.com/items/124

```
[root@ip-172-31-37-10 ~]# curl -X "DELETE" https://c5b0971ljg.execute-api.us-east-1.amazonaws.com/items/124
"Deleted item 124"[root@ip-172-31-37-10 ~]#
```

7. Get all items to verify that the item was deleted -

> curl https://c5b0971ljg.execute-api.us-east-1.amazonaws.com/items/

```
[root@ip-172-31-37-10 ~]# curl https://c5b0971ljg.execute-api.us-east-1.amazonaws.com/items
[{"id":"125","name":"gandhi","marks":85}][root@ip-172-31-37-10 ~]#
```

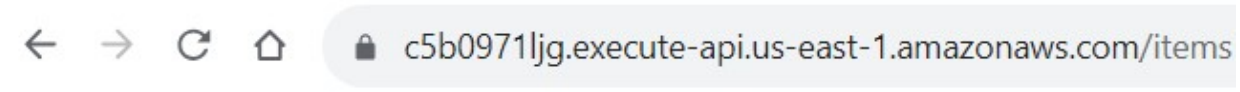
8. To get a specific item -

Use the following command to get an item by its ID.

curl https://c5b0971ljg.execute-api.us-east-1.amazonaws.com/items/125

```
[root@ip-172-31-37-10 ~]# curl https://c5b0971ljg.execute-api.us-east-1.amazonaws.com/items/125
{"id":"125","name":"gandhi","marks":85}[root@ip-172-31-37-10 ~]#
```

9. We can see the contents of our **DynamoDB table** through the **API invoke URL** entered into Browser.



[{"id":"125","name":"gandhi","marks":85}]