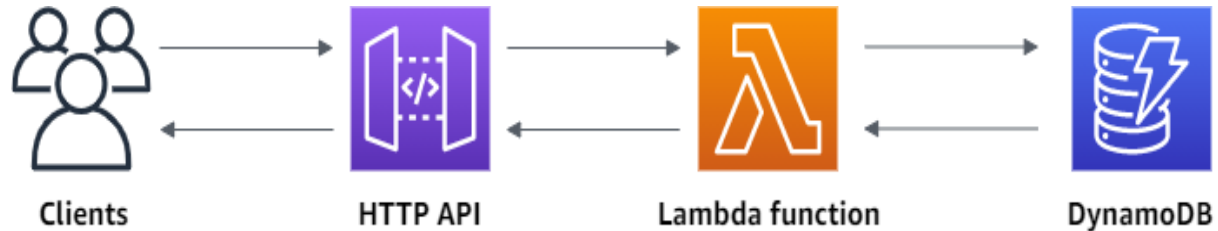


# Build a CRUD API with Lambda and DynamoDB

## 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.

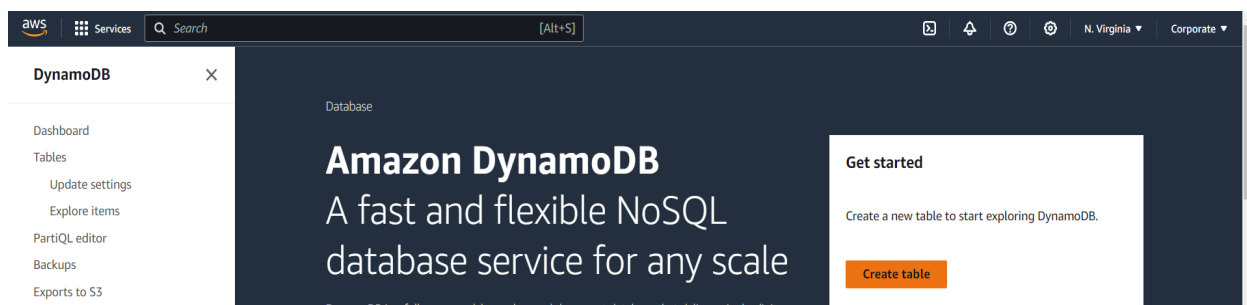
When we invoke the HTTP API, the API Gateway routes the request to our Lambda function. The Lambda function then interacts with DynamoDB, and returns a response to the API Gateway. The API Gateway then returns a response to the client/customer.

## 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.

# Create table

## Table details [Info](#)

DynamoDB is a schemaless database that requires only a table name and a primary key when you create the table.

### Table name

This will be used to identify your table.

Between 3 and 255 characters, containing only letters, numbers, underscores (\_), hyphens (-), and periods (.).

### Partition key

The partition key is part of the table's primary key. It is a hash value that is used to retrieve items from your table and allocate data across hosts for scalability and availability.

String ▼

1 to 255 characters and case sensitive.

### Sort key - optional

You can use a sort key as the second part of a table's primary key. The sort key allows you to sort or search among all items sharing the same partition key.

String ▼

1 to 255 characters and case sensitive.

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

The screenshot shows the AWS DynamoDB console interface for the 'students\_grade\_sheet' table. On the left, a sidebar lists the table. The main area has a top navigation bar with tabs for Overview, Indexes, Monitor, Global tables, Backups, Exports and streams, and Actions. Below the tabs, a blue informational banner advises on protecting the table from accidental writes and deletes by enabling Point-in-time recovery (PITR). The 'General information' section displays the table's configuration: Partition key 'id (String)', Sort key '-', Capacity mode 'On', and Table status 'Active'. It also shows 'No active alarms' and 'Point-in-time recovery (PITR)' is 'Off'.

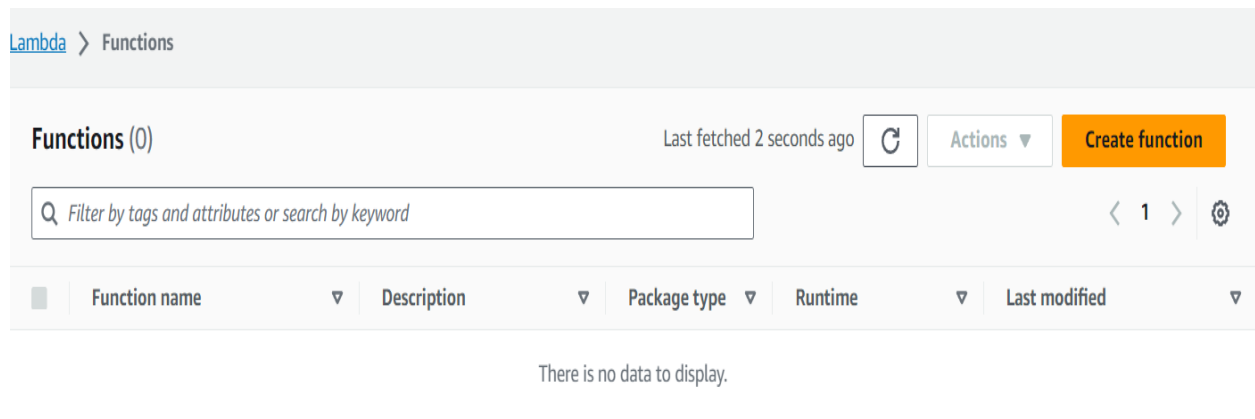
General information <a href="#">Info</a>			
Partition key id (String)	Sort key -	Capacity mode On	Table status Active
Alarms No active alarms	Point-in-time recovery (PITR) Off		

## 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**.

aws

Services

Search

[Alt+S]

N. Virginia

Corporate

Lambda > Functions > Create function

Create function

info

AWS Serverless Application Repository applications have moved to [Create application](#).

☒ Author from scratch

Start with a simple Hello World example.

☐ Use a blueprint

Build a Lambda application from sample code and configuration presets for common use cases.

☐ Container image

Select a container image to deploy for your function.

Basic information

Function name

Enter a name that describes the purpose of your function.

students\_grade\_sheet\_function

Use only letters, numbers, hyphens, or underscores with no spaces.

Runtime

info

Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.

Node.js 18.x

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

Role creation might take a few minutes. Please do not delete the role or edit the trust or permissions policies in this role.

Role name

Enter a name for your new role.

students\_grade\_sheet\_role

Use only letters, numbers, hyphens, or underscores with no spaces.

Policy templates - optional

info

Choose one or more policy templates.

Simple microservice permissions X

DynamoDB

▼ Advanced settings

☐ Enable Code signing

info

Use code signing configurations to ensure that the code has been signed by an approved source and has not been altered since signing.

☐ Enable function URL

info

Use function URLs to assign HTTP(S) endpoints to your Lambda function.

☐ Enable tags

info

A tag is a label that you assign to an AWS resource. Each tag consists of a key and an optional value. You can use tags to search and filter your resources, track your AWS costs, and enforce attribute-based access control.

☐ Enable VPC

info

Connect your function to a VPC to access private resources during invocation.

Cancel

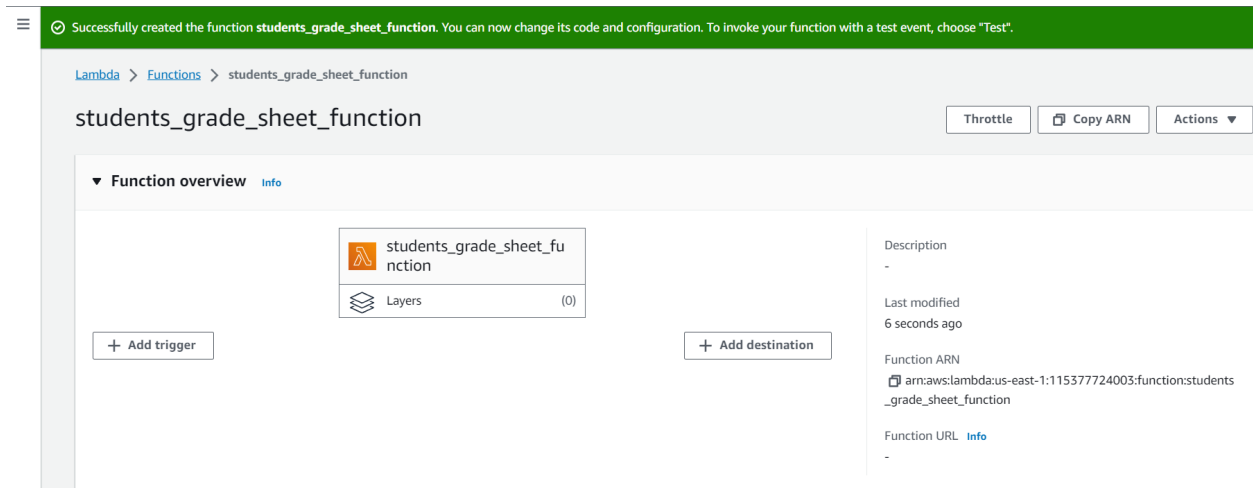
Create function

CloudShell

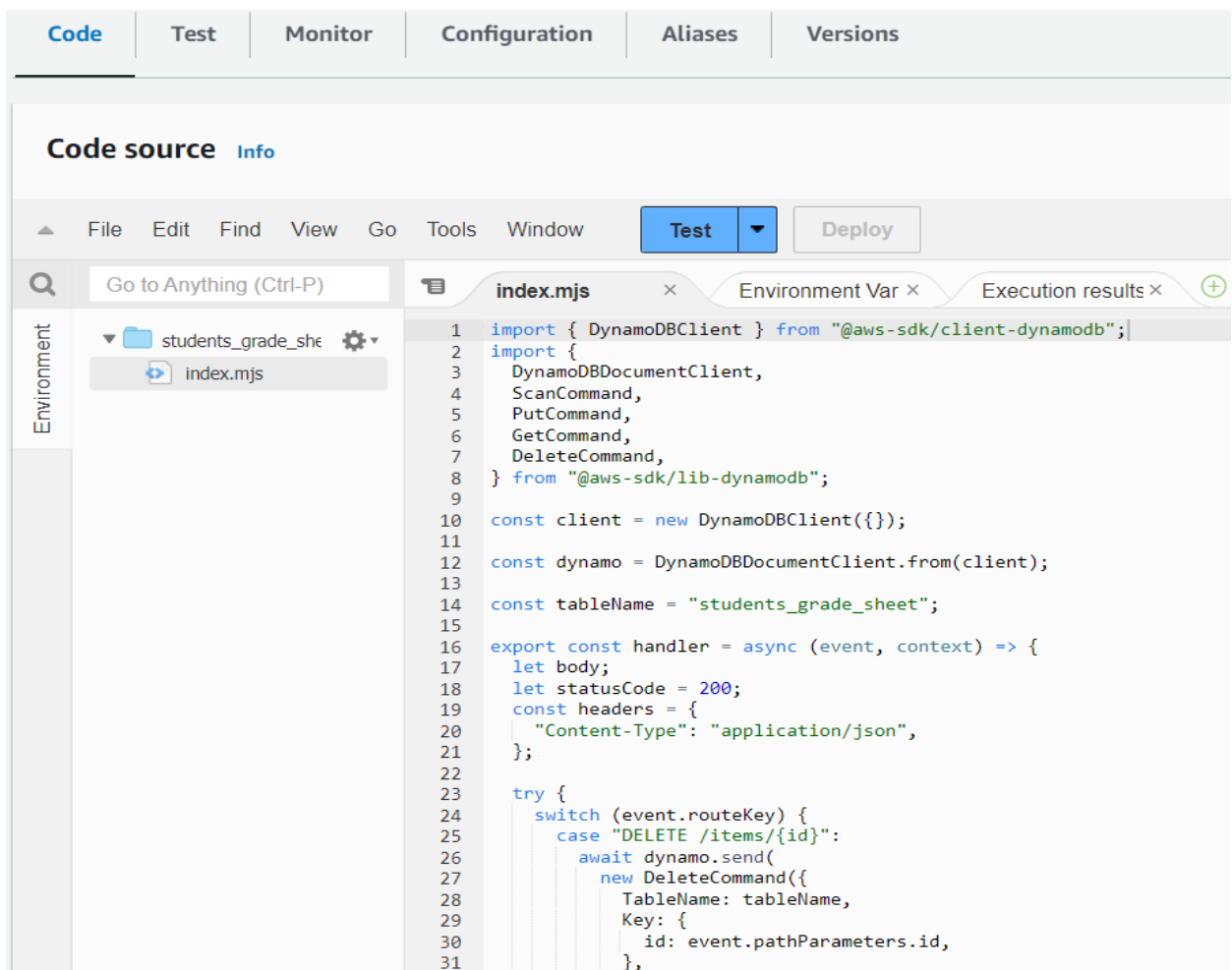
Feedback

© 2023, Amazon Web Services, Inc. or its affiliates. [Privacy](#) [Terms](#) [Cookie preferences](#)

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



- 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 -

Execution results		Status: Succeeded	Max memory
<b>Test Event Name</b>			
hi			
<b>Response</b>			
<pre>{   "statusCode": 400,   "body": "\\\"Unsupported route: \\\"\\\"undefined\\\"\\\"\\\"\",   "headers": {     "Content-Type": "application/json"   } }</pre>			
<b>Function Logs</b>			
START RequestId: bbbd63ab-e2d0-491e-baf2-9a922d5a8294 Version: \$LATEST			
END RequestId: bbbd63ab-e2d0-491e-baf2-9a922d5a8294			
REPORT RequestId: bbbd63ab-e2d0-491e-baf2-9a922d5a8294 Duration: 199.96 ms Billed Duration: 200 ms Memory Size: 128 MB Max Memory Used: 103 MB			
<b>Request ID</b>			
bbbd63ab-e2d0-491e-baf2-9a922d5a8294			

```
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**.

[API Gateway](#) > [APIs](#) > Create API

## Choose an API type

### HTTP API

Build low-latency and cost-effective REST APIs with built-in features such as OIDC and OAuth2, and native CORS support.

Works with the following:  
Lambda, HTTP backends

3. For **API name**, enter **students\_grade\_sheet\_api**.

## Create an API

### Create and configure integrations

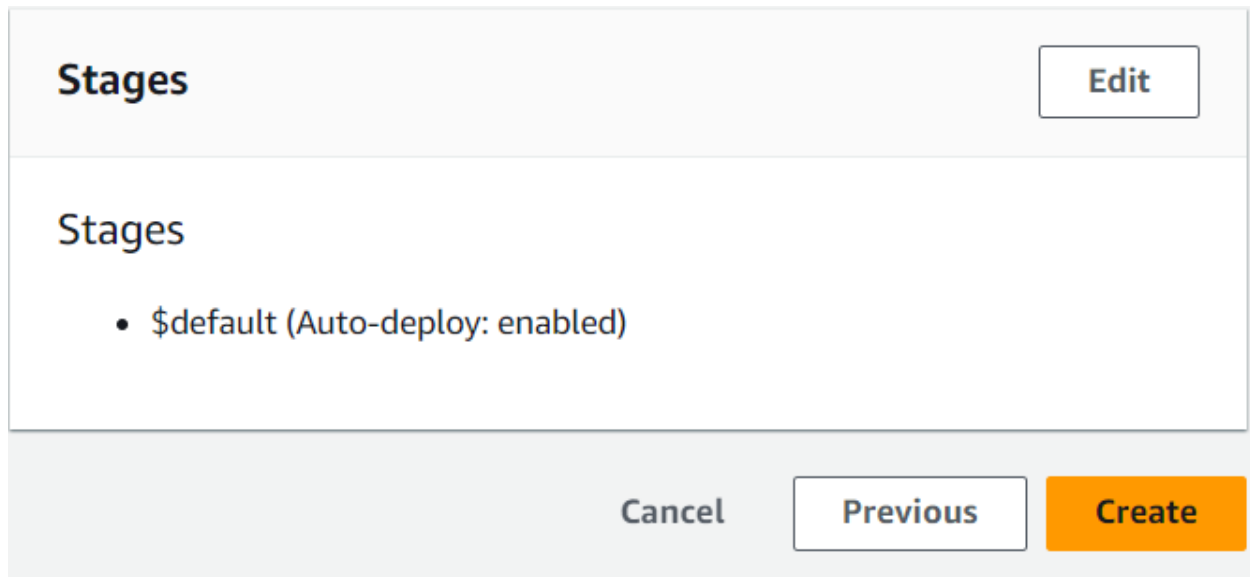
Specify the backend services that your API will communicate with. These are called integrations. For a Lambda integration, API Gateway invokes the Lambda function and responds with the response from the function. For HTTP integration, API Gateway sends the request to the URL that you specify and returns the response from the URL.

**Integrations (0)** [Info](#)

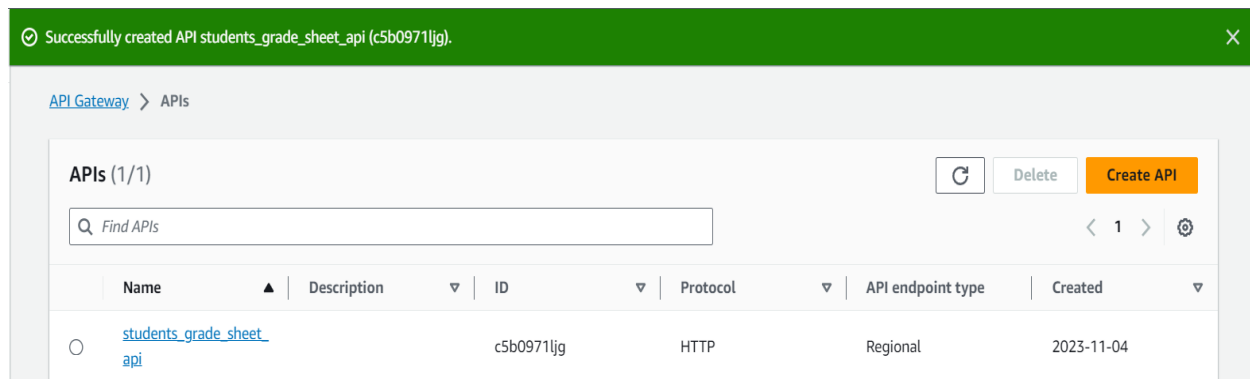
---

**API name**  
An HTTP API must have a name. This name is cosmetic and does not have to be unique; you will use the API's ID (generated later) to programmatically refer to this 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**.

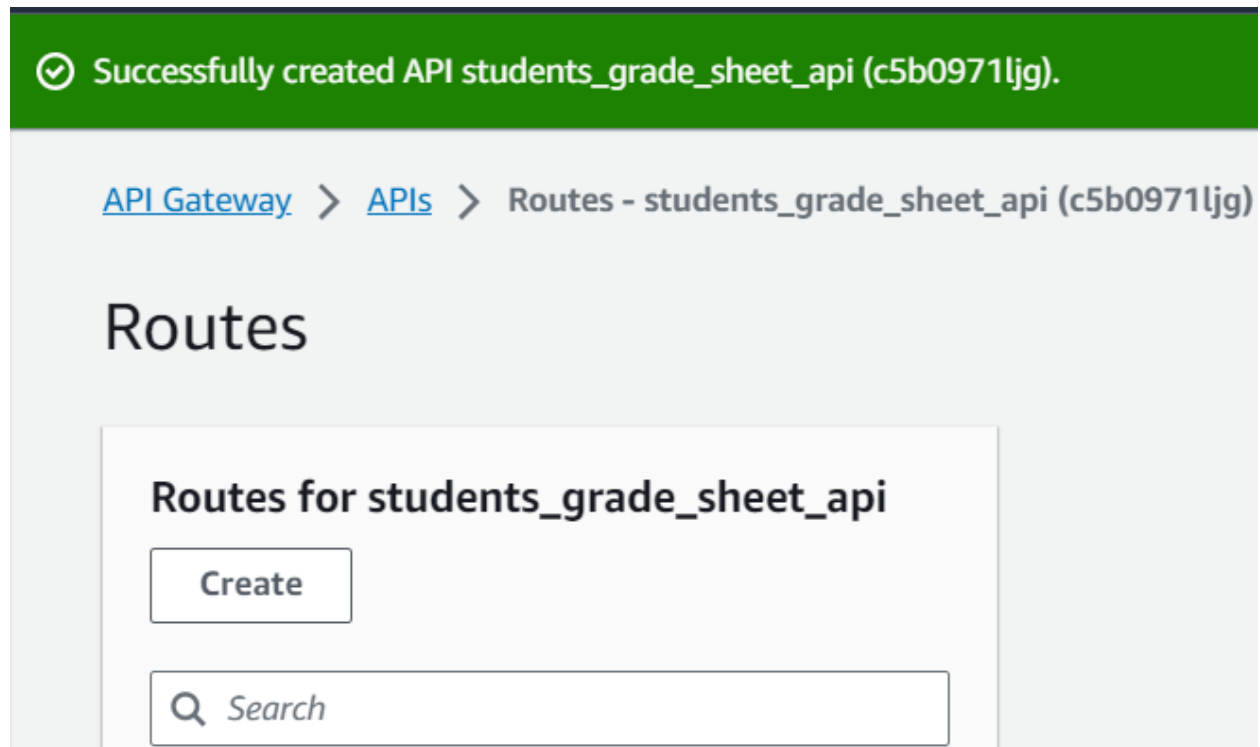
## Step 4: Create routes

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

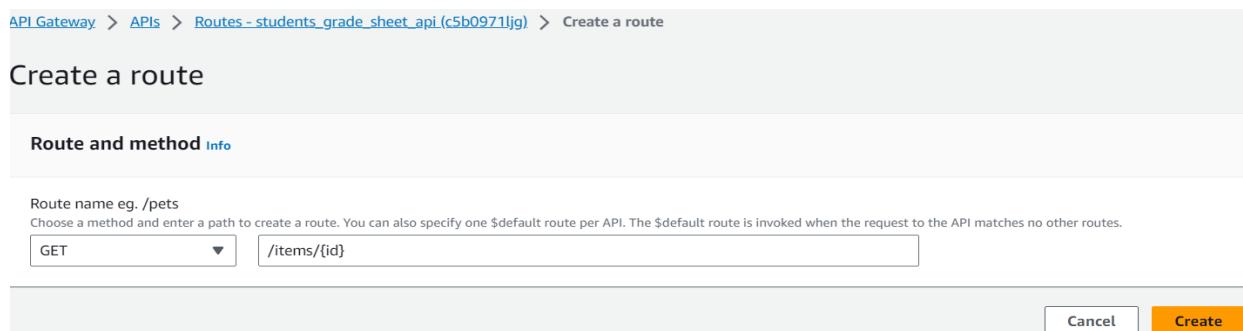
To create Routes -

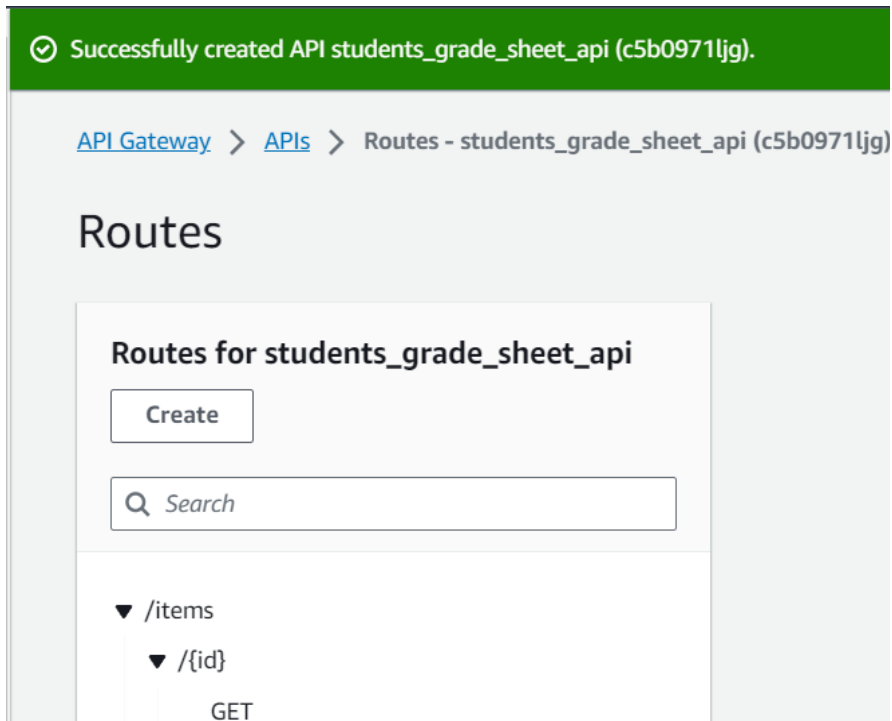


1. Choose the **API** we have created.
2. Choose **Routes**.
3. Choose **Create**.

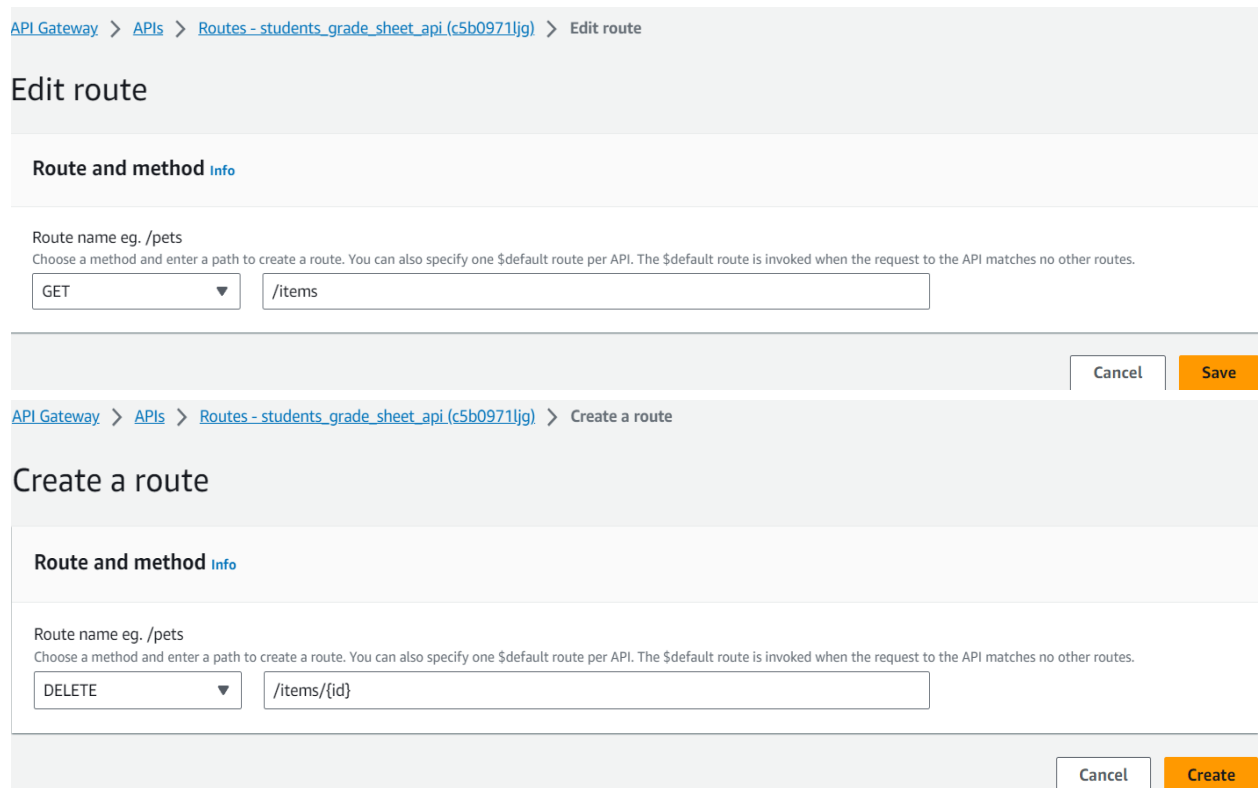


4. For Method, choose **GET**.
5. For the path, enter **/items/{id}**. The **{id}** at the end of the path is a path parameter that API Gateway retrieves from the request path when a client makes a request.
6. Choose **Create**.





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



API Gateway > APIs > Routes - students\_grade\_sheet\_api (c5b0971ljg) > Edit route

## Edit route

**Route and method** [Info](#)

Route name eg. /pets  
Choose a method and enter a path to create a route. You can also specify one \$default route per API. The \$default route is invoked when the request to the API matches no other routes.

PUT

[Cancel](#) [Save](#)

API Gateway > APIs > Routes - students\_grade\_sheet\_api (c5b0971ljg)

## Routes

**Routes for students\_grade\_sheet\_api**

[Create](#)

- ▼ /items
  - GET
  - PUT**
- ▼ /{id}
  - GET
  - DELETE

**Route details**

PUT /items (

Authorization

Authorizers p

No authori:

Integration

The integra

No integrat

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**.

[API Gateway](#) > [APIs](#) > [students\\_grade\\_sheet\\_api\(c5b0971ljg\)](#) > [Integrations](#)

## Integrations

[Attach integrations to routes](#)

[Manage integrations](#)

### Routes for students\_grade\_sheet\_api

▼ /items

GET

PUT

▼ /{id}

GET

DELETE

### Integration details

PUT /items (ID: g2fnln)

You have no integrations

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

## Integrations

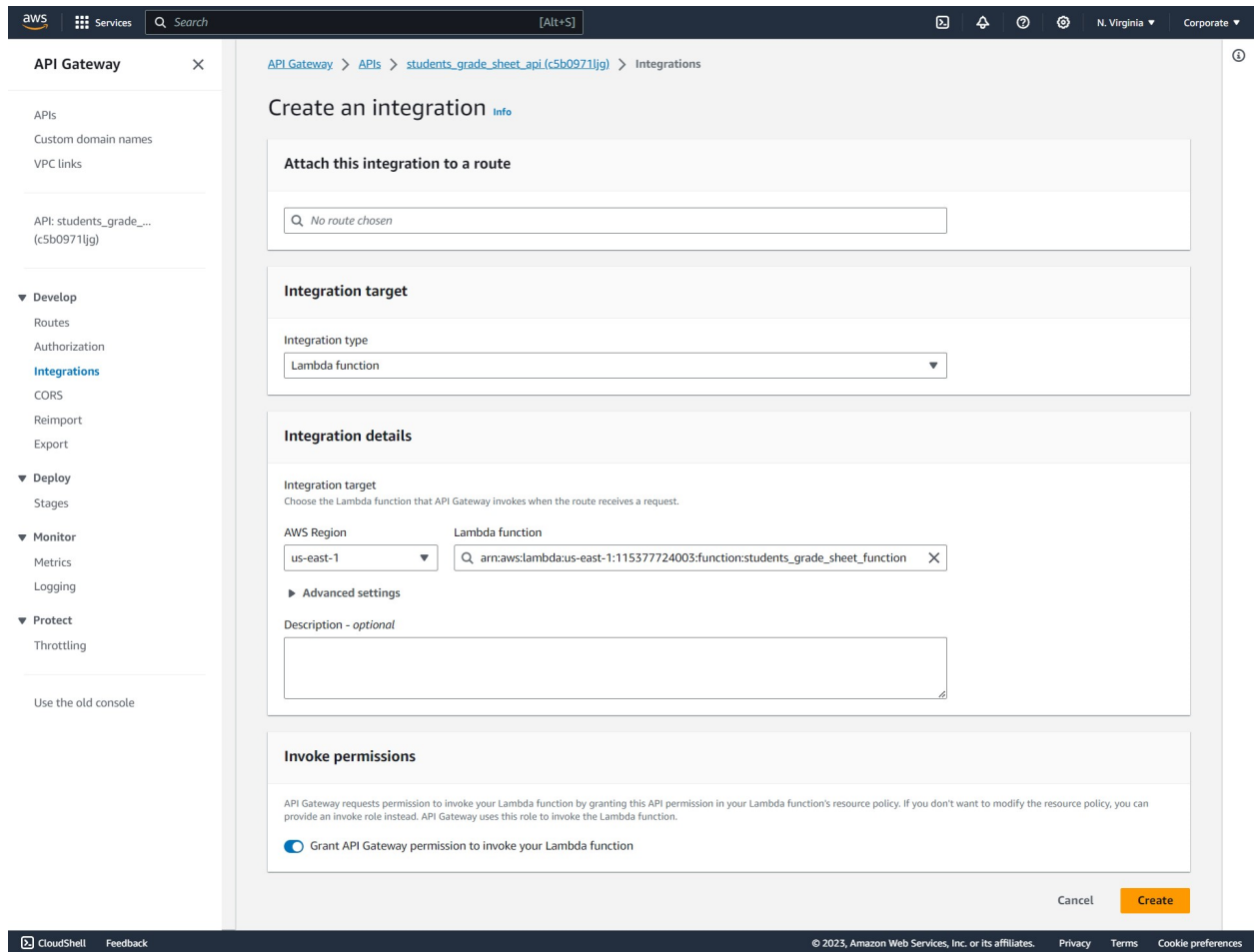
[Attach integrations to routes](#)

[Manage integrations](#)

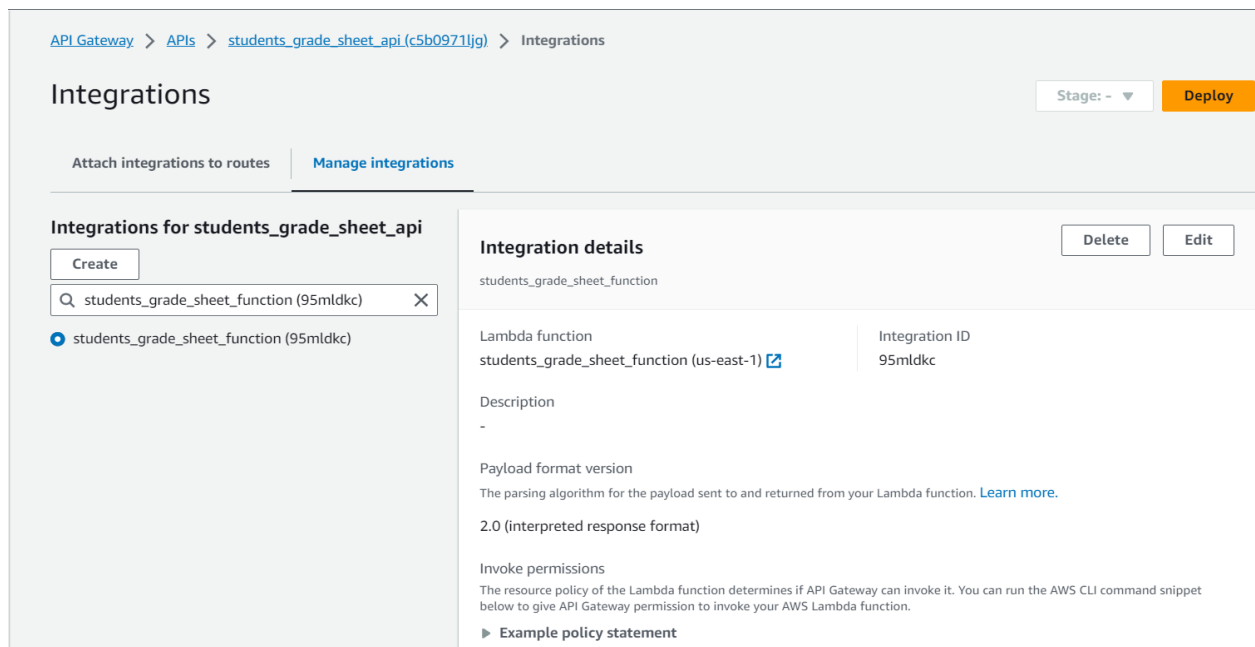
### Integrations for students\_grade\_sheet\_api

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.

The screenshot displays the AWS API Gateway console interface. At the top, the breadcrumb navigation shows 'API Gateway > APIs > students\_grade\_sheet\_api (c5b0971ljg) > Integrations'. Below this, the 'Integrations' header is visible with a 'Stage: -' dropdown and a 'Deploy' button. The main content area is divided into two panes. The left pane, titled 'Routes for students\_grade\_sheet\_api', contains a search bar and a list of routes. The routes are organized into two sections: '/items' and '/{id}'. Under '/items', there are three routes: GET, PUT, and DELETE, each with an 'AWS Lambda' integration. Under '/{id}', there are two routes: GET and DELETE, each with an 'AWS Lambda' integration. The 'DELETE /items/{id}' route is currently selected. The right pane, titled 'Integration details for route', shows the details for the selected route. It includes a 'Detach integration' button and a 'Manage integration' button. The details include the Lambda function 'students\_grade\_sheet\_function (us-east-1)', the integration ID '95mldkc', the description '-', the payload format version '2.0 (interpreted response format)', and the invoke permissions section which includes an example policy statement.

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.

The screenshot shows the AWS Management Console interface for an EC2 instance. At the top, a green banner indicates "Successfully started i-07794efb7ac31b21e". Below this, the "Instances (1/1)" page is displayed. A table lists the instance "kaka" with ID "i-07794efb7ac31b21e", state "Running", type "t2.micro", and availability zone "us-east-1d". The instance is in the "Initializing" status check phase. Below the table, the "Instance: i-07794efb7ac31b21e (kaka)" details are shown. The "Details" tab is active, displaying the instance summary. Key information includes: Instance ID "i-07794efb7ac31b21e (kaka)", Public IPv4 address "34.229.183.100", Private IPv4 addresses "172.31.37.10", and Public IPv4 DNS "ec2-34-229-183-100.compute-1.amazonaws.com". The instance state is confirmed as "Running".

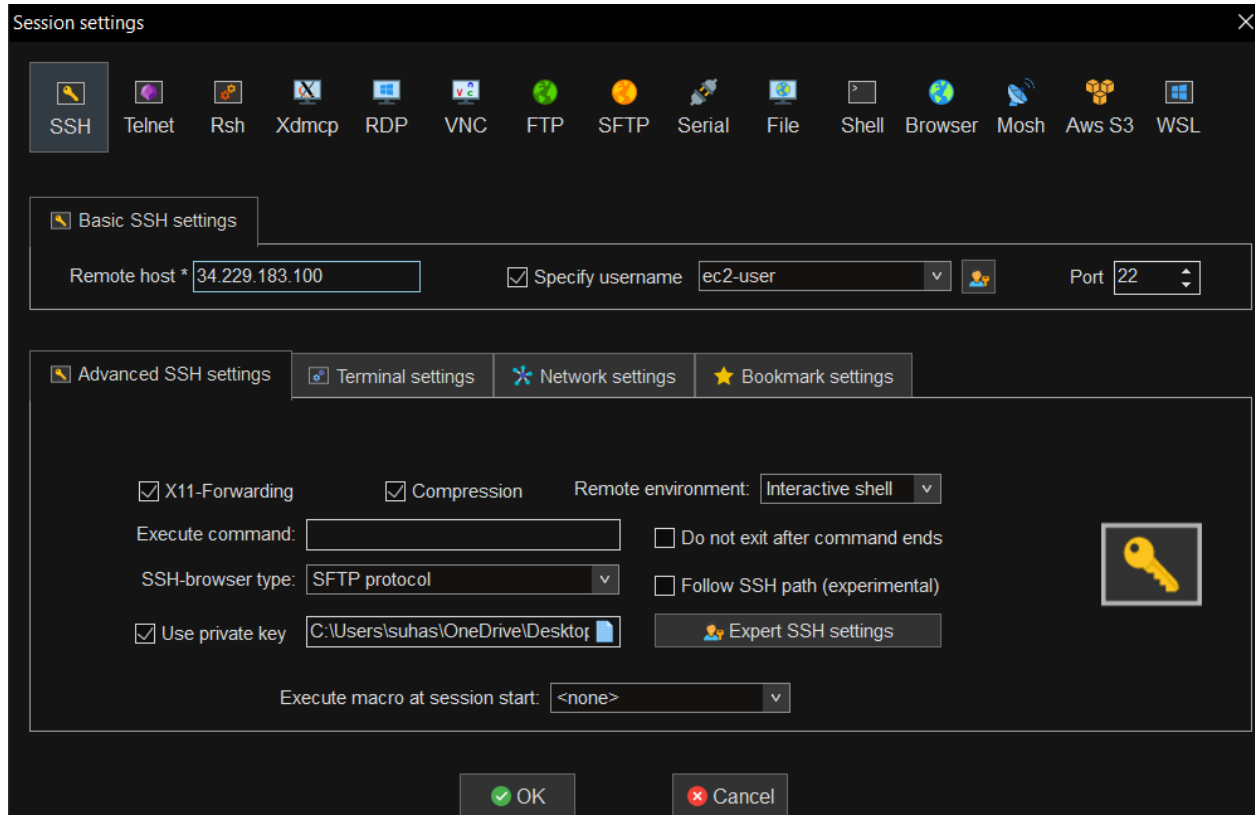
To make sure that our API is working, use **Curl**.

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)

The screenshot shows the AWS API Gateway console. The breadcrumb navigation indicates the path: "API Gateway > APIs > students\_grade\_sheet\_api (c5b0971ljg) > Stages". The "Stages" page is displayed, showing a list of stages for the "students\_grade\_sheet\_api". A "Create" button is visible. Below the "Create" button, a search bar is present. A table lists the stages, with the "default" stage selected. The "default" stage is shown in the "Stage details" pane on the right. The "Stage details" pane includes a "Delete" button and an "Edit" button. The "Details" section for the "default" stage shows the "Name" as "\$default", the "Created" date as "November 4, 2023 11:18 AM", and the "Last updated" date as "November 4, 2023 11:37 AM". The "Invoke URL" is displayed as <https://c5b0971ljg.execute-api.us-east-1.amazonaws.com/>.



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": "suhasi"}' https://c5b0971ljg.execute-api.us-east-1.amazonaws.com/items
```

```
[root@ip-172-31-37-10 ~]# curl -X "PUT" -H "Content-Type: application/json" -d '{"id": "124", "marks": 98, "name": "suhasi"}' https://c5b0971ljg.execute-api.us-east-1.amazonaws.com/items
"Put item 124"[root@ip-172-31-37-10 ~]#
```

```
> 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-172-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 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.

← → ↻ 🏠 🔒 c5b0971ljg.execute-api.us-east-1.amazonaws.com/items

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