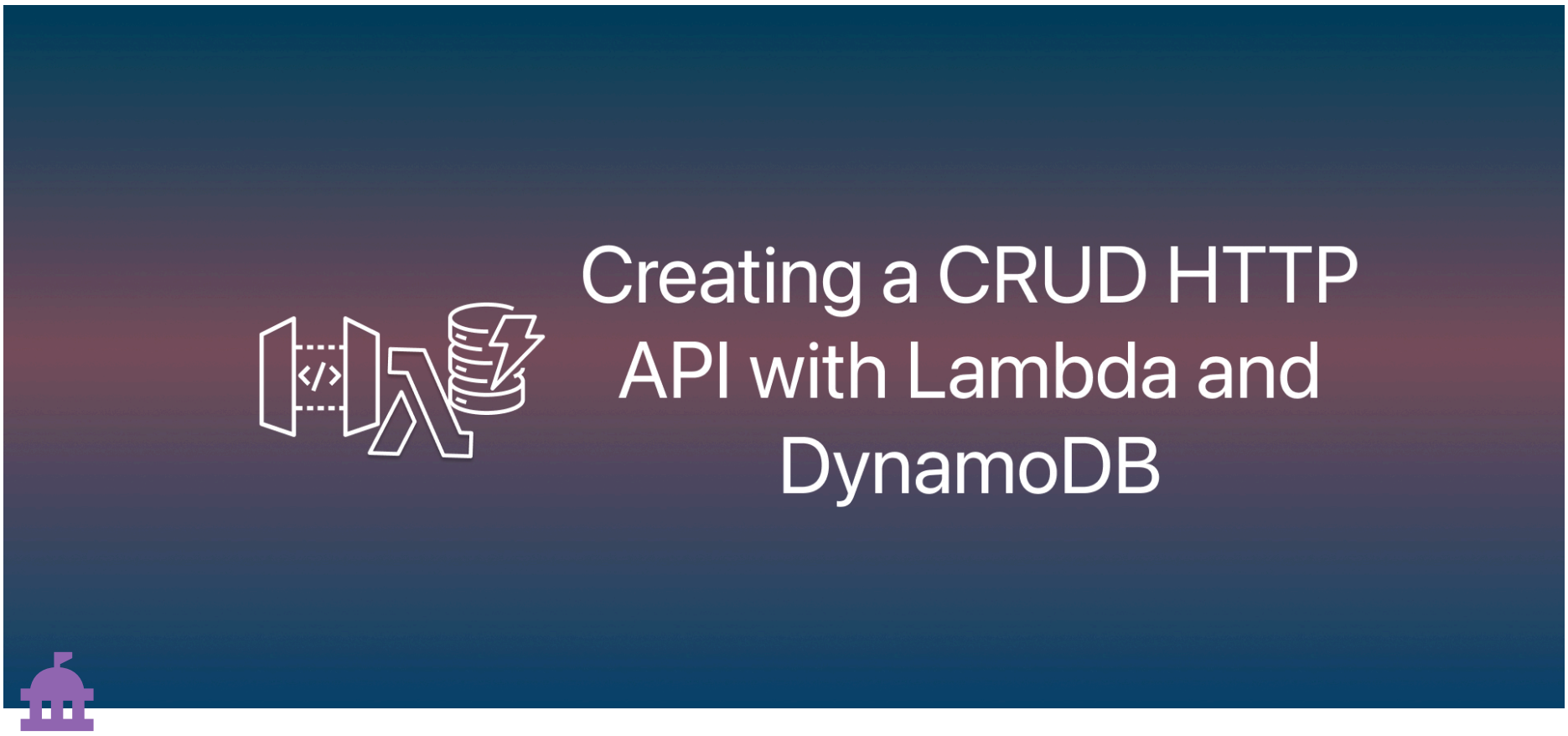


Creating a CRUD HTTP API with Lambda and DynamoDB

Created by	Ⓝ Niyush Bjr
Created time	@24 December 2025 22:02
level	Intermediate
Description	**Creating a CRUD HTTP API with Lambda and DynamoDB**
AWS Services	API Gateway, DynamoDB, Lambda
Other tools	AWS CLI, Postman
Category	Application-Integration, Networking

What this project aims?
Architecture
⌘ Step 1 : Creating DynamoDB Table
⌘ Step 2 : Creating a Lambda function.
To create a Lambda fucntion.
⌘ Step 3: Creating an HTTP API.
To create an HTTP API
⌘Step 4 : Creating routes for the API
To create routes
⌘ Step 5 : Creating an Integration.
⌘ Step 6 : Attaching integration to routes.
⌘ Step 7 : Test your API.
Test 1: To create or update an Item. (POST)
Test 2: To get all Item. (GET)
Test 3: To get a single Item. (GET /items/{id})
Test 4: To update a single Item. (PUT /items/{id})
Test 5 : Delete an item
⌘ Step 8 : Clean up Time.
To Delete DynamoDB Table.
To Delete HTTP API.
To Delete Lambda Function.
To delete a Lambda function's log group
To delete a Lambda function's execution role

▼ About API in general

HTTP is the protocol.
REST is a design style built on top of HTTP.
If the API cares about resources, it's REST. If it only cares about routes, it's HTTP.

- Every REST API is an HTTP API
- Not every HTTP API is REST

People say "REST API" when they really mean:

> "An API over HTTP"

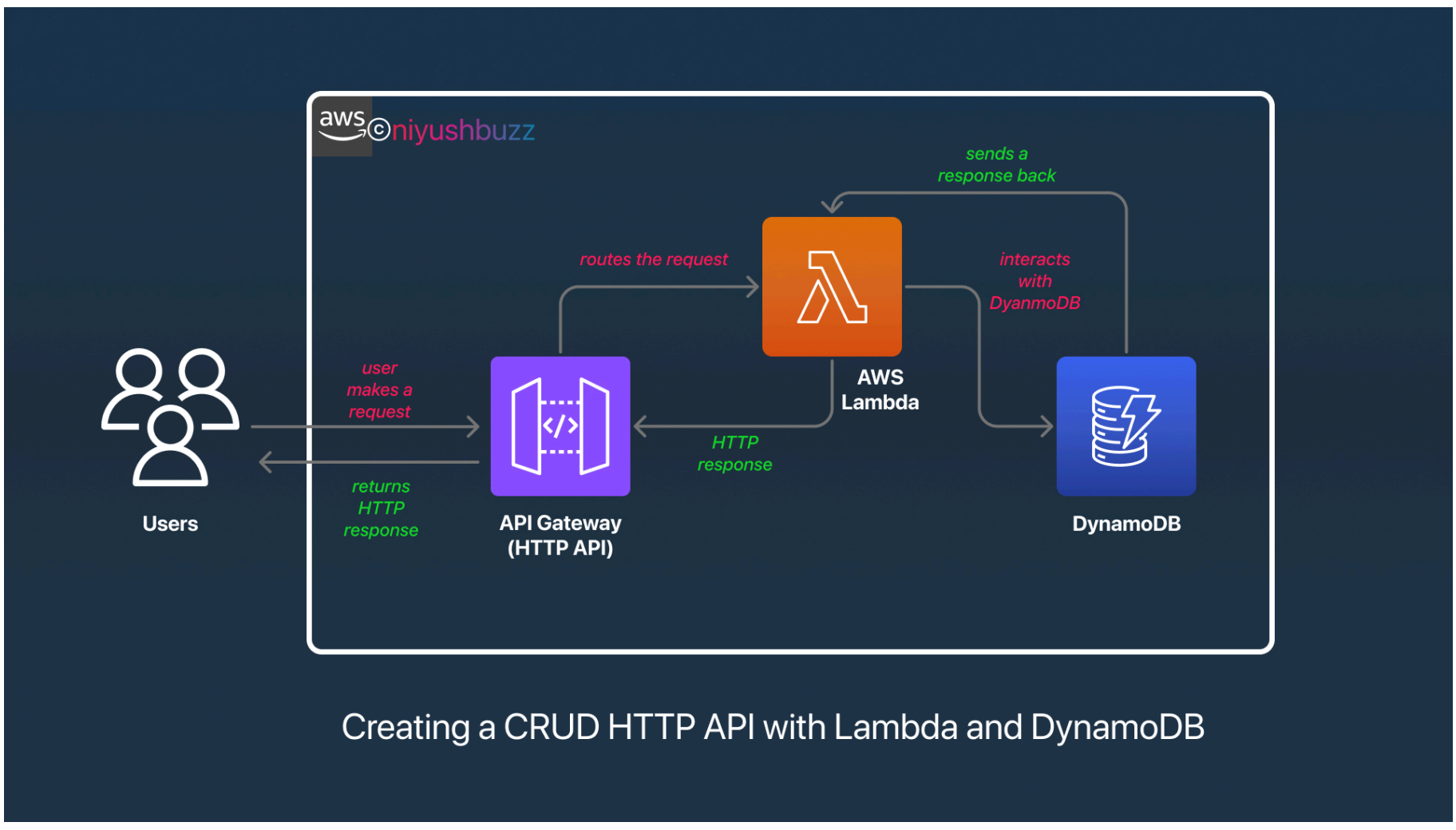
▼ What this project aims?

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

When the HTTP API is invoked, API Gateway routes the request to Lambda function. Then the lambda function interacts with DyanmoDB, and returns a response to API Gateway.

API Gateway then returns a response to Client.

▼ Architecture



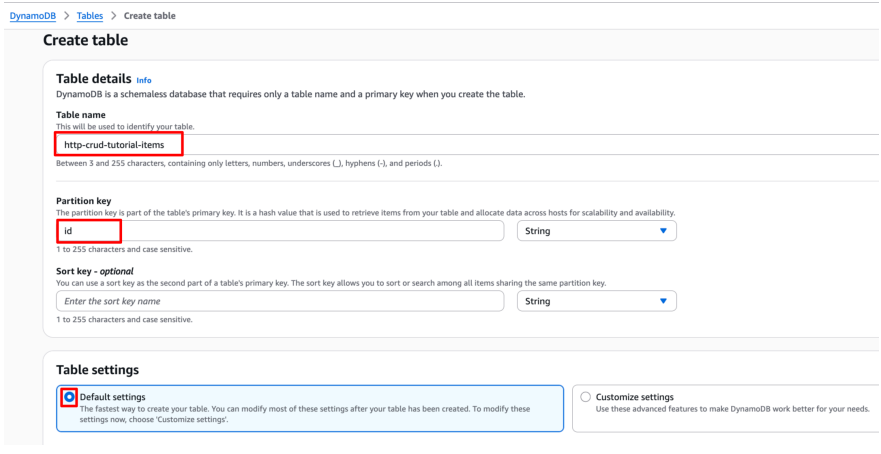
⌘ Step 1 : Creating DynamoDB Table

> we'll be using DynamoDB table to Store data for the API.

- Head to DynamoDB console.

▼ Image :

- Click on **Create Table**.
- **Table Name :** `http-crud-tutorial-items.`
- **Partition Key :** `id`
- Finally click on `Create table` .



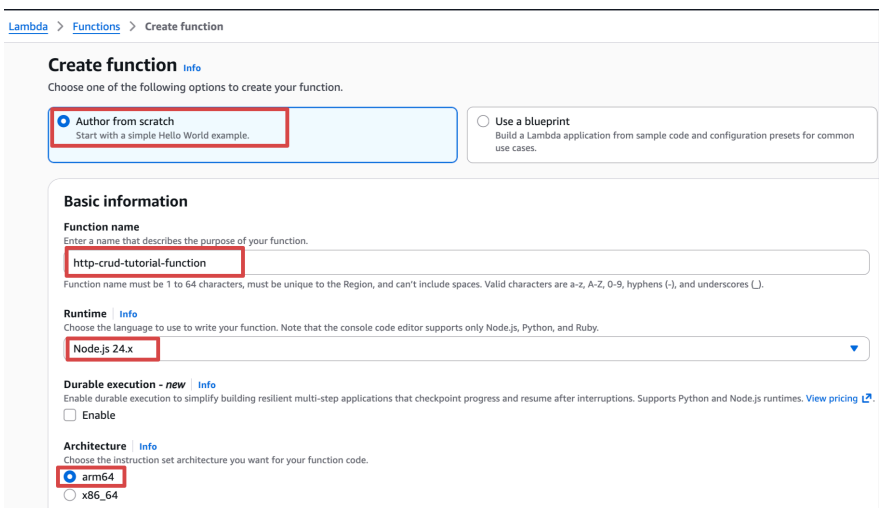
⌘ Step 2 : Creating a Lambda function.

> Now, we will create a Lambda function for the backend of the API. This lambda function is responsible for create, read, update, and delete items from the DynamoDB.
>
> The function uses **events from API Gateway** to determine how to interact with DynamoDB. For this project we will only use one lambda function.
>
> "**Best Practice :** *create separate functions for each route*"

To create a Lambda fucntion.

- Head over to Lambda ⇒ `Create function.`
- **Function name :** `http-crud-tutorial-function`
- **Runtime :** `Python or Node.js`
- **Permissions:** `Change default execution role`
  - `Create a new rolefrom AWS policy templates`
  - **Role name :** `http-crud-tutorial-role`
- **Policy Template :** `Simple microservice permissions` This policy grants the Lambda Function permission to interact with DynamoDB.
- Finally Click on `Create function`
- Once the lambda function is created replace the code with below code and click on `Deploy` to update the function.

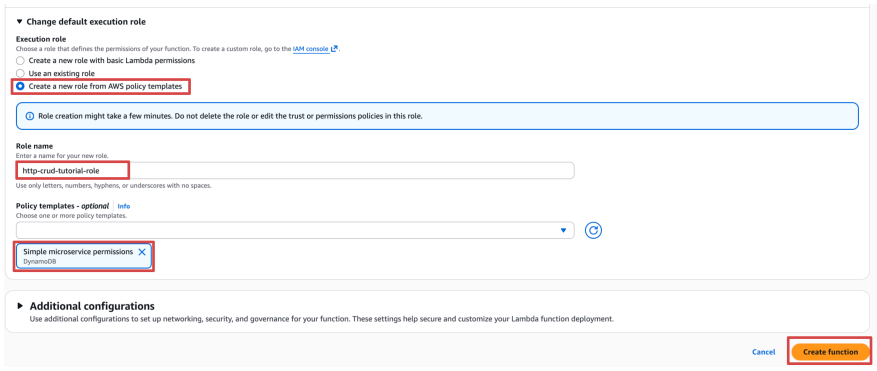
▼ Lambda Function creation

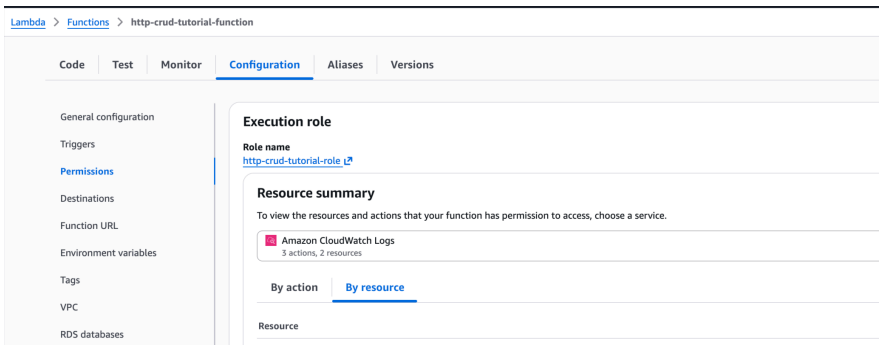


▼ Javascript Code

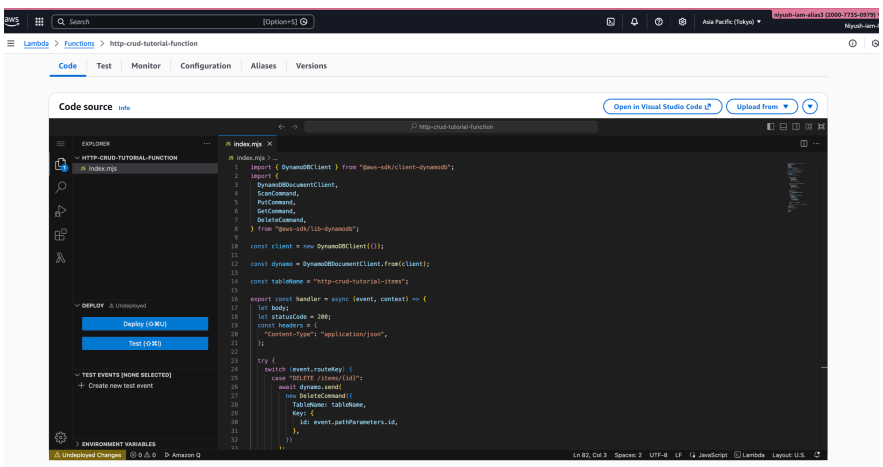
```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import {
```

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

const client = new DynamoDBClient({});

const dynamo = DynamoDBDocumentClient.from(clie
nt);

const tableName = "http-crud-tutorial-items";

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.i
d}`;
        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 "POST /items":
        let requestJSON = JSON.parse(event.body);
        await dynamo.send(
          new PutCommand({
            TableName: tableName,
```





▼ Code and Deploy



```
          Item: {
            id: requestJSON.id,
            price: requestJSON.price,
            name: requestJSON.name,
          },
        })
      );
      body = `Created item ${requestJSON.id}`;
      break;
    case "PUT /items/{id}":
      let updateJSON = JSON.parse(event.body);
      await dynamo.send(
        new PutCommand({
          TableName: tableName,
          Item: {
            id: event.pathParameters.id,
            price: updateJSON.price,
            name: updateJSON.name,
          },
        })
      );
      body = `Updated item ${event.pathParameters.id}`;
      break;
    default:
      throw new Error(`Unsupported route: "${event.routeKey}"`);
  }
} catch (err) {
  statusCode = 400;
  body = err.message;
} finally {
  body = JSON.stringify(body);
}

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

⌘ Step 3: Creating an HTTP API.

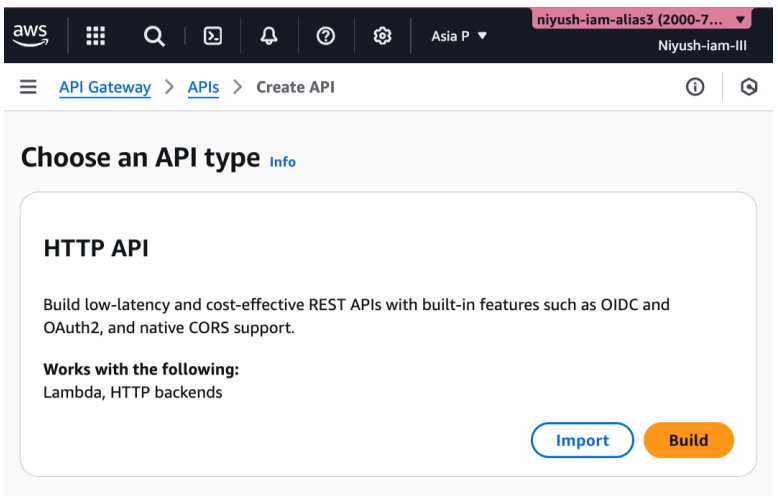
> The HTTP API provides HTTP endpoint for Lambda function. We'll create and configure routes and integrations to connect API to lambda function.

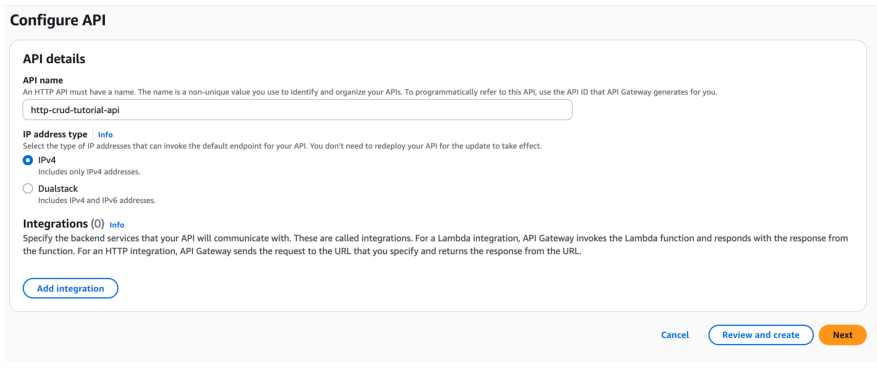
To create an HTTP API

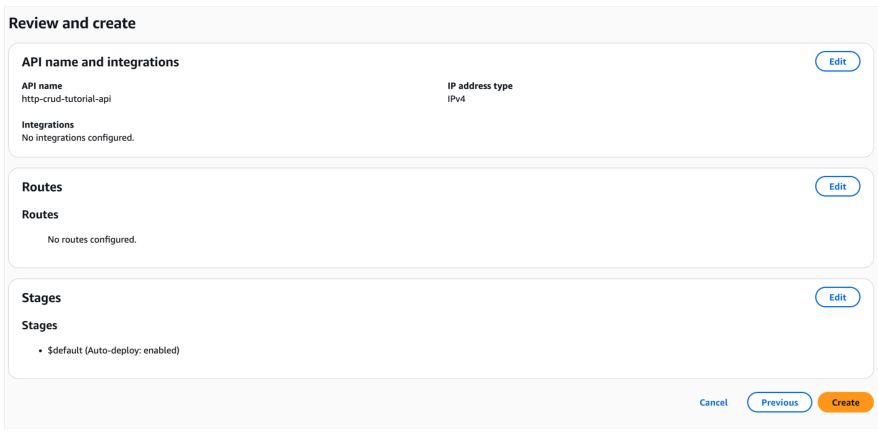
- Head over to API Gateway
- Choose **Create API ⇒ HTTP API ⇒ Build**
- **API Name :** `http-crud-tutorial-api`
- **IP address :** `IPv4` ⇒ `Next`

▼ Creating API

- **Configure routes :** Select **Next** ⇒ skip to route creation.
- Review the stage that API Gateway creates for you, and choose **Next** ⇒ `Create`







⌘Step 4 : Creating routes for the API

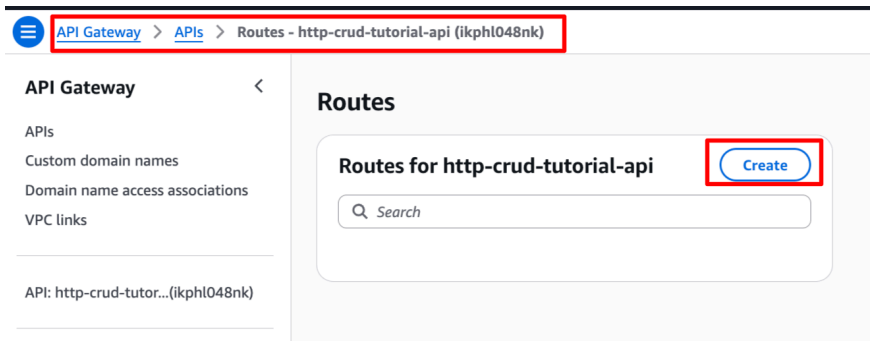
> `Routes` : are the way to send incoming API request to backend resources. Routes consist of two main parts : an HTTP method and a resource path, for example, GET/items. For this example API we create 5 routes.
>
> - GET `/items/{id}`
> - GET `/items`
> - POST `/items`
> - PUT `/items/{id}`
> - DELETE `/items/{id}`

To create routes

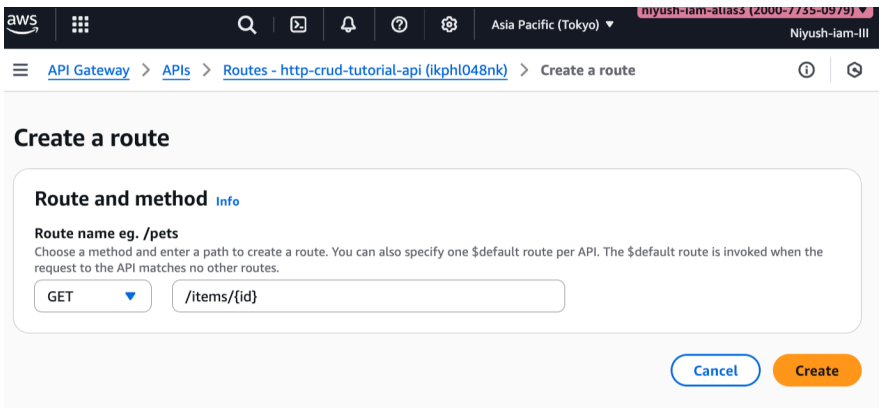
- Head over to API Gateway ⇒ `API`

- `Routes` ⇒ `Create`
- **Method** : GET
- 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.
- Choose `Create` ※ Repeat the same steps for `GET /items, DELETE /items/{id}, and PUT /items/{id}` and `POST /items`

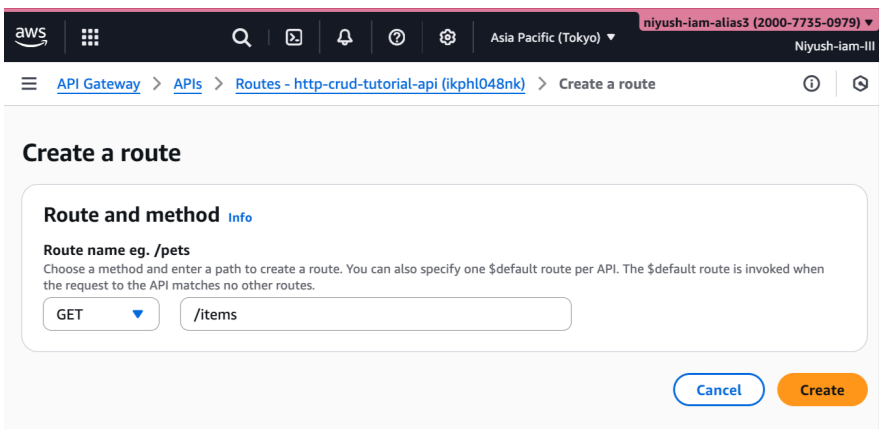
▼ Creating Routes



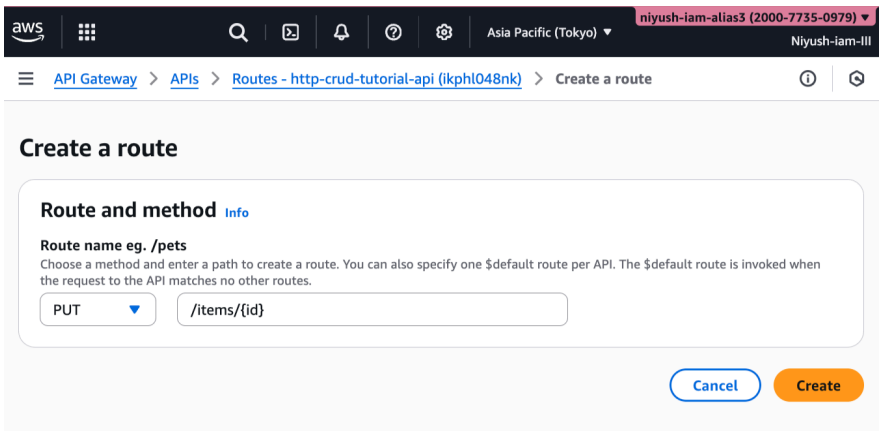
▼ GET /items/{id}



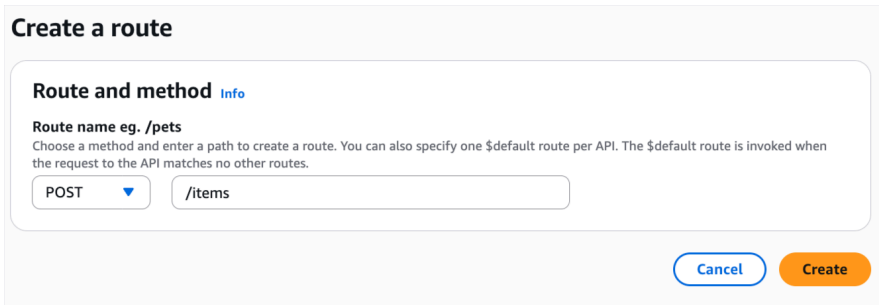
▼ Get /items



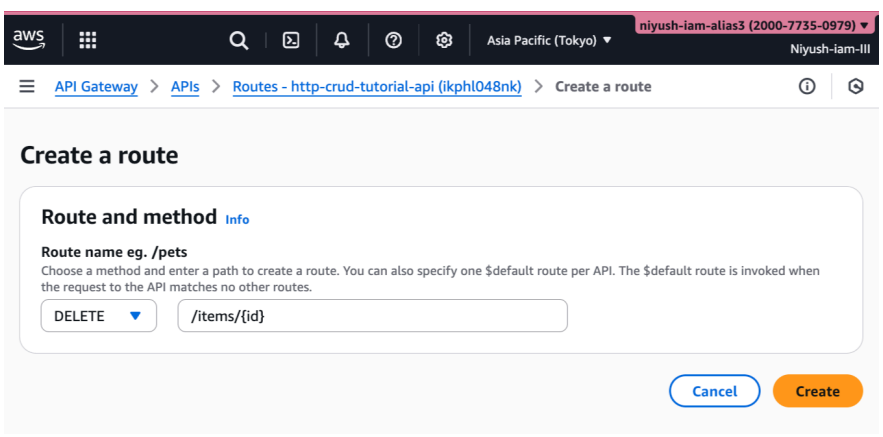
▼ PUT /items/{id}



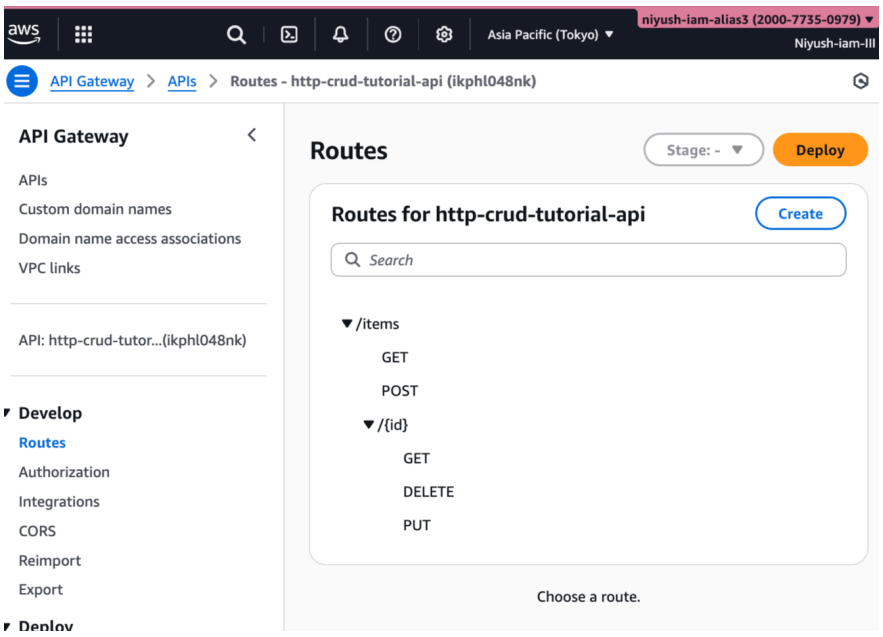
▼ POST /items



▼ DELETE /items/{id}



▼ Final Routes Path



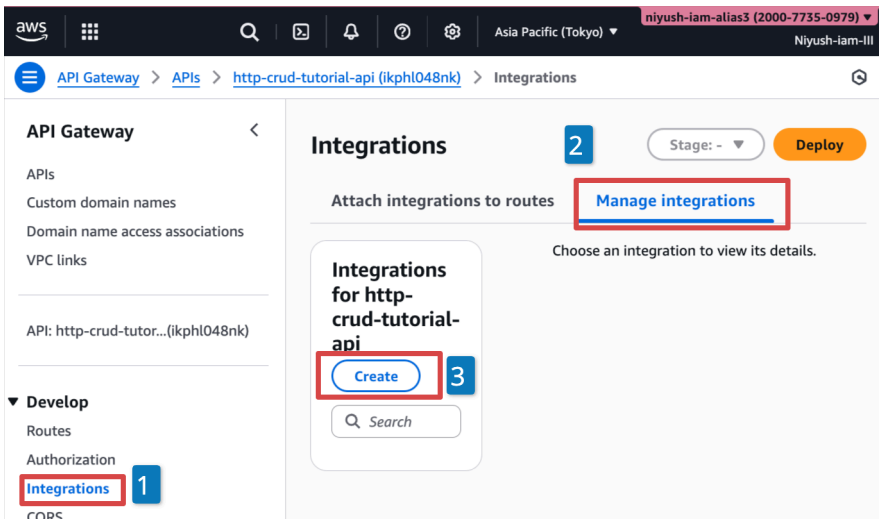
⌘ Step 5 : Creating an Integration.

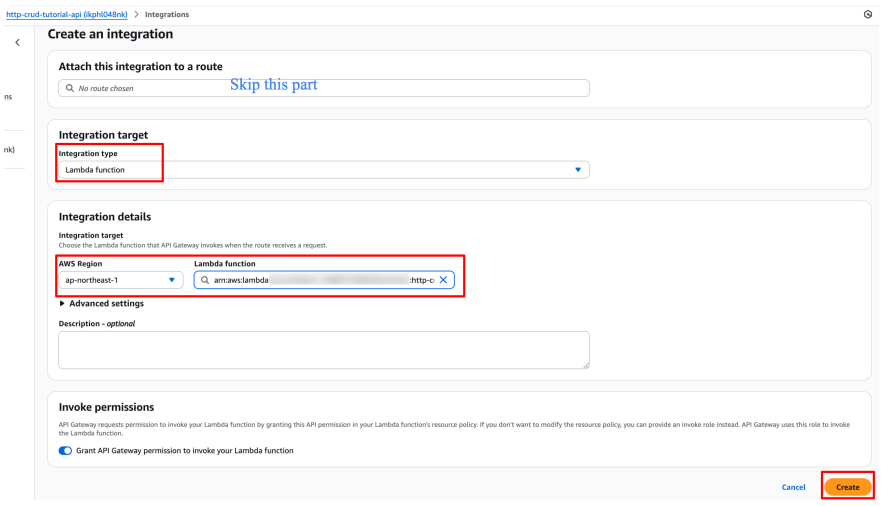
> Integration is used to connect a route to backend resources. In this case, one Lambda integration created will be used for all routes.

To create an integration

▼ Integration of API

- Head over to API Gateway ⇒ `API`
- **Integrations** ⇒ `Manage Integrations` ⇒ `Create`
- Skip *Attach this integration to a route*
- **Integration Type :** select `Lambda function`
- **Lambda function** : `http-crud-tutorial-function`
- Finally click on `Create`





⌘ Step 6 : Attaching integration to routes.

> For this project, we can use same Lambda integration for all routes. Once the integrated to all of the APIs routes, Lambda function is invoked when a client calls any of the routes.

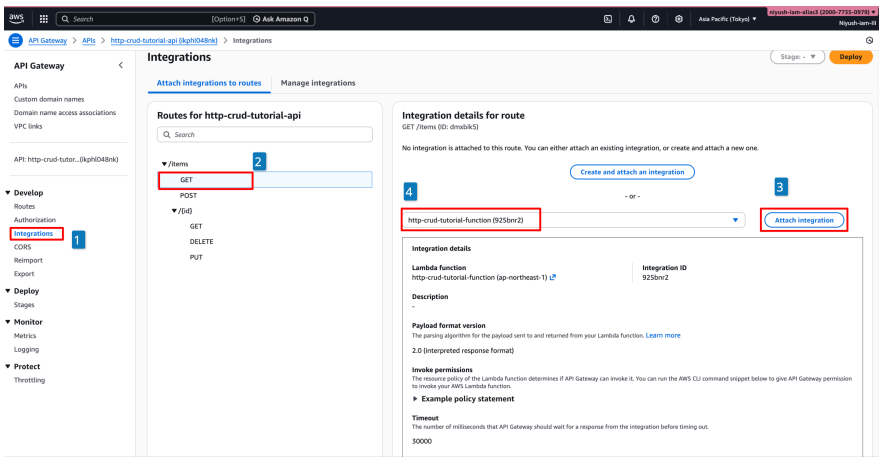
To attach integrations to routes

- Head over to API Gateway ⇒ `API`
- `Integrations` ⇒ Select a `route`
- under **Choose an existing integration**, select `http-crud-tutorial-function` ⇒ `Attach integration`
- Repeat the steps for all routes.

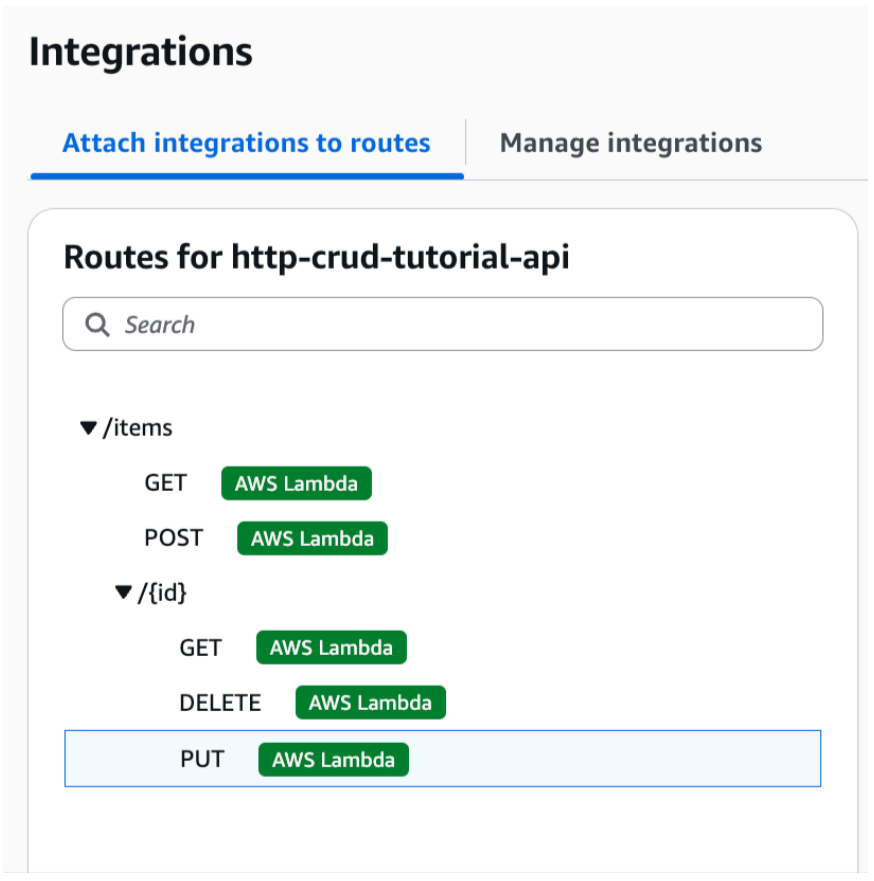
The final product should look something like this.

▼ attaching integration to route

FOR GET



▼ Final Product : Integration to Routes



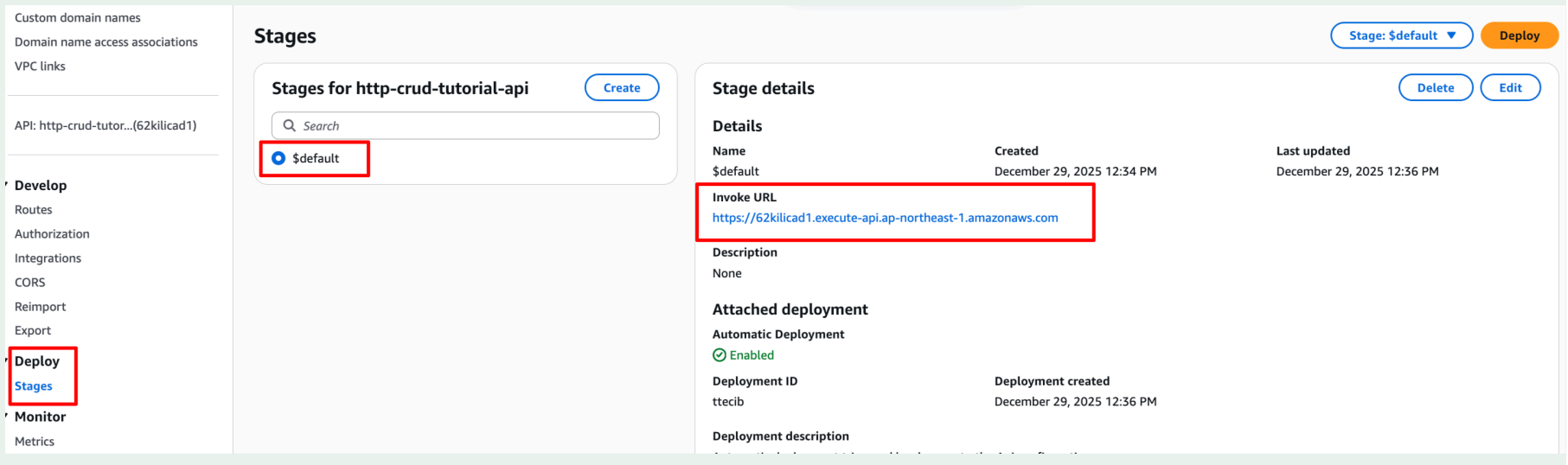
⌘ Step 7 : Test your API.

- POST /items to create a new item (server assigns ID).
- PUT /items/{id} to fully replace an existing item (client provides full item data, including its ID in the URL).
- PATCH /items/{id} to partially update an existing item (client provides only changed fields).

- To get the URL to invoke API Gateway
- Choose your API.

> Note your API's invoke URL. It appears under Invoke URL on the Details page.

▼ Invoke URL



Test 1: To create or update an Item. (POST)

Test 1a : CREATE item using `curl`

```
curl -X "POST" \
     -H "Content-Type: application/json" \
     -d '{
           "name": "New Awesome Gadget",
           "description": "A cutting-edge device for modern li
fe.",
           "price": 99.99,
           "category": "Electronics"
         }' \
     https://YOUR-INVOKE-URL.amazonaws.com/items`
```

Test 1b : CREATE item using `Postman`

■ Open `Postman` ⇒ `New Request`

■ **HTTP Method** : `POST` ⇒

■ **URL** : `https://YOUR-INVOKE-URL.amazonaws.com/items`

The target URL for the request.

■ **Headers** : `Headers` tab ⇒ Add a new header

■ Key : `Content-Type`

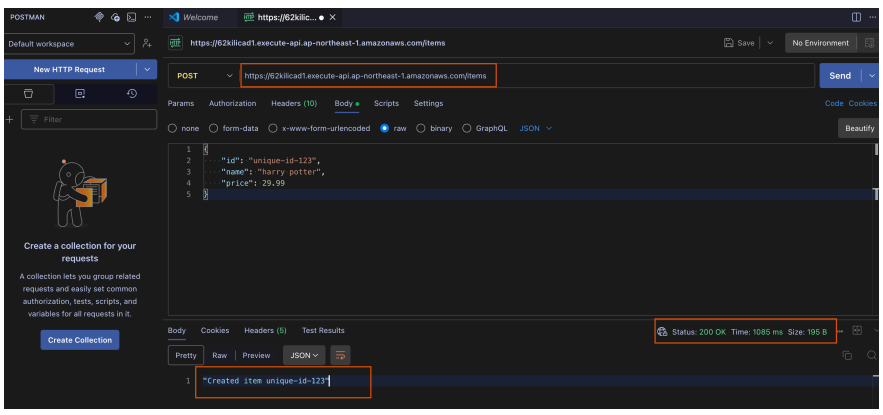
■ Value : `application/json`

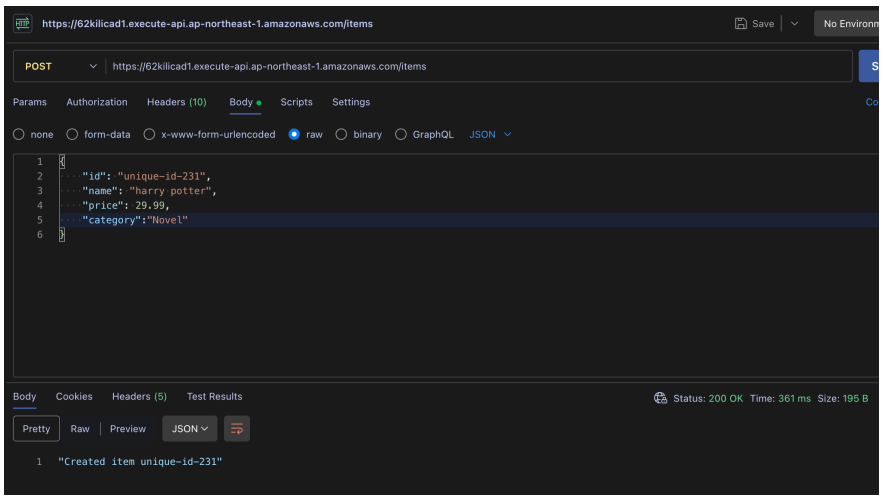
■ **Set Request Body** : `Body` tab ⇒ `raw` ⇒ `JSON`

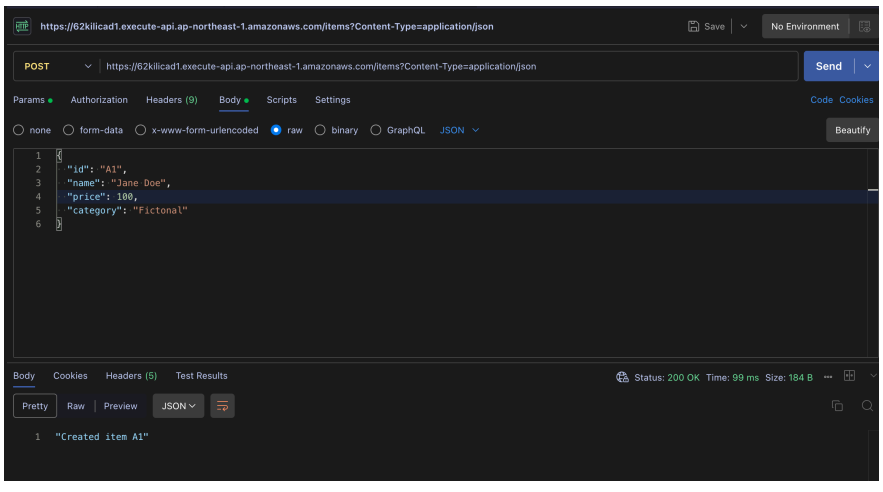
■ Send request by clicking the `Send` button.

```
{
   "name": "New Awesome Item",
   "description": "This is a brand new item.",
   "price": 29.99,
   "category": "Electronics"
}
```

▼ Create using Postman `POST`





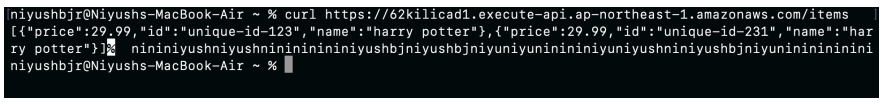


Test 2: To get all Item. (GET)

Test 2a : GET all using `curl`

```
curl https://YOUR-INVOKE-URL.amazonaws.com/items
```

▼ GET items using CURL

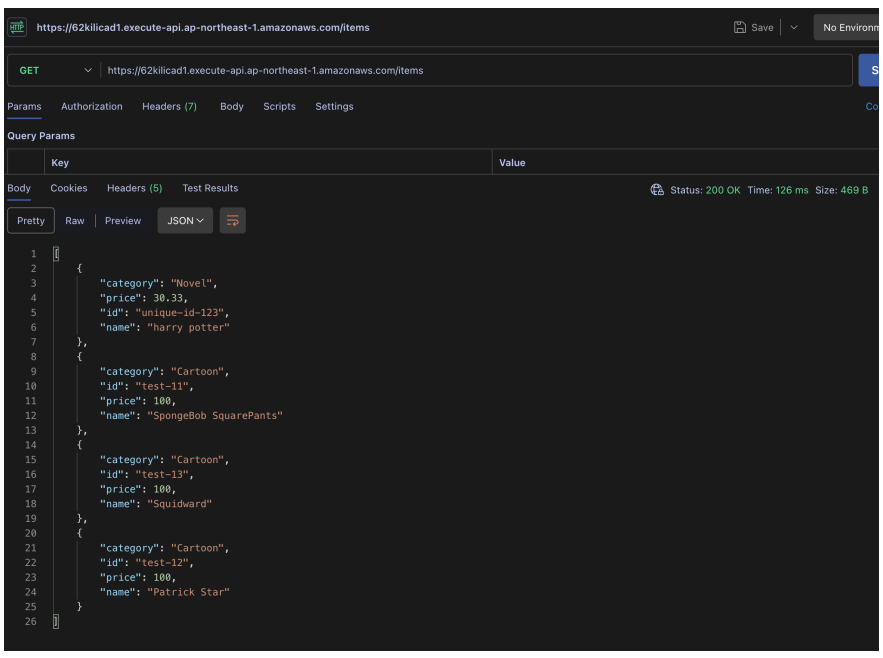


Test 2b : GET all using `Postman`

- Open `Postman` ⇒ `New Request`
- **HTTP Method** : `GET`
- **URL** : `https://YOUR-INVOKE-URL.amazonaws.com/items`
- Send request by clicking the `Send` button.

> No headers or body are typically needed for a simple GET request unless your API requires specific authentication or other headers.

▼ GET with post man



Test 3: To get a single Item. (GET /items/{id})

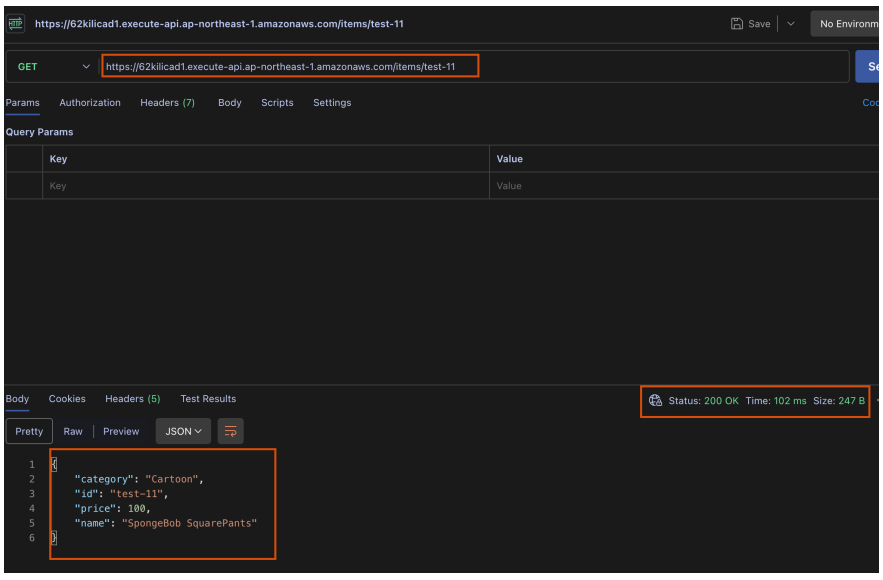
Test 3a : GET one item using `curl`

```
curl https://YOUR-INVOKE-URL.amazonaws.com/items/{id}
```

▼ GET single item using CURL

Test 3b : GET one item using `Postman`

▼ GET single item using POSTMAN



■ Open Postman ⇒ New Request

■ **HTTP Method** : GET

■ **URL** : https://YOUR-INVOKE-URL.amazonaws.com/items/{id}

■ **Headers** : (None typically required)

■ **Body** : None

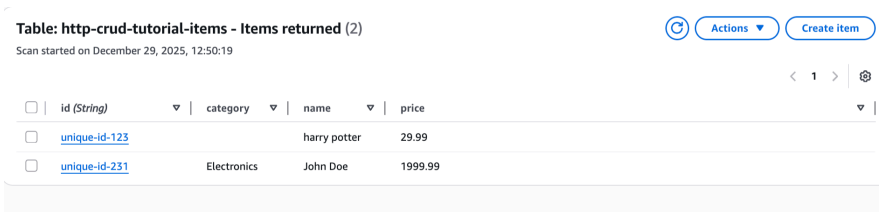
■ Click Send

Test 4: To update a single Item. (PUT /items/{id})

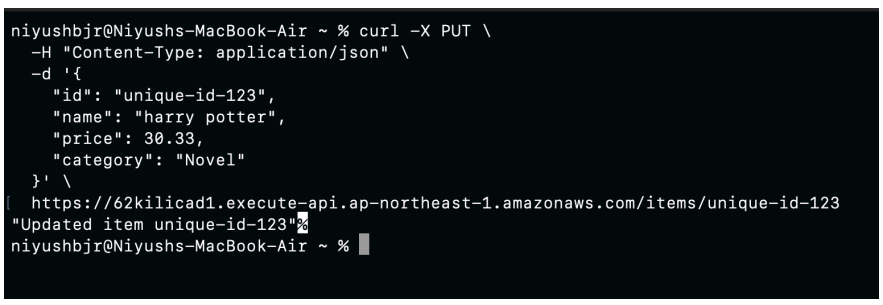
Test 4a : UPDATE using curl

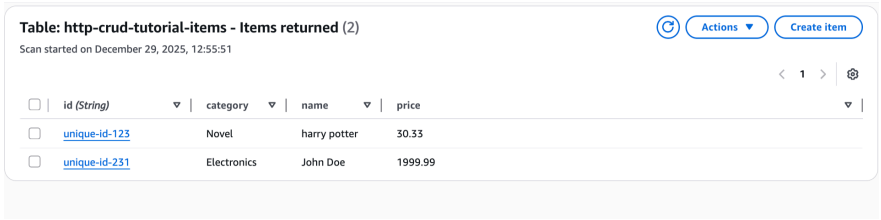
```
curl -X "PUT" \
    -H "Content-Type: application/json" \
    -d '{
         "id": "THE ACTUAL ID",
         "name": "Laptop Pro (Updated Model)",
         "description": "Latest generation high-performanc
e laptop.",
         "price": 1999.99,
         "category": "Electronics"
       }' \
 https://YOUR-INVOKE-URL.amazonaws.com/items/{id}
```

▼ Updating the item with CURL



lets' update the category of unique ID unique-id-123





Test 4b : UPDATE using Postman

■ Open Postman ⇒ `New Request`

■ **HTTP Method** : `PUT`

■ **URL** : `https://YOUR-INVOKE-URL.amazonaws.com/items/{id}`

■ Headers : `Headers` tab ⇒ Add a new header

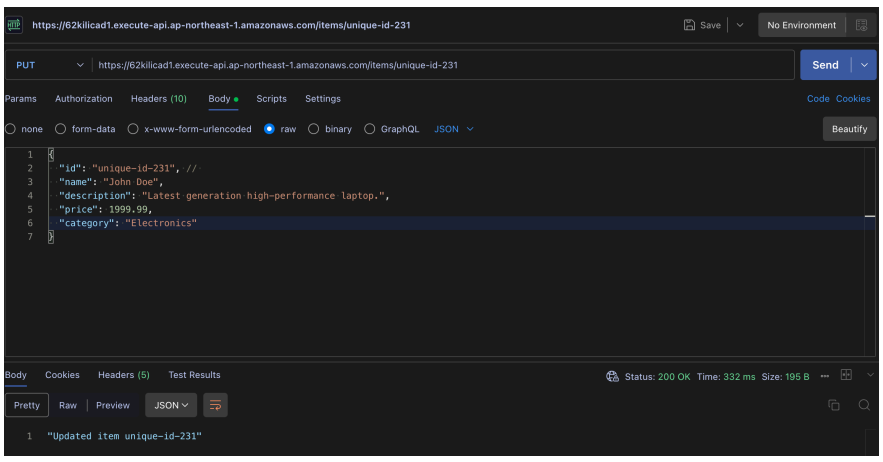
  ■ Key : `Content-Type`

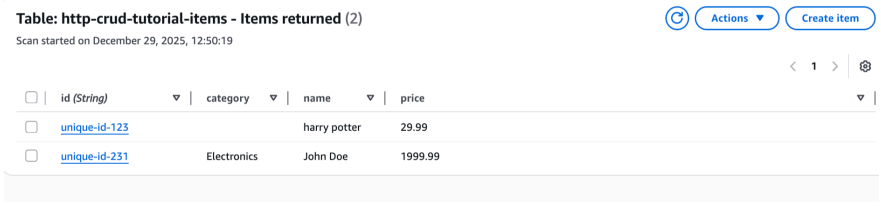
  ■ Value : `application/json`

■ **Set Request Body** : `Body` tab ⇒ `raw` ⇒ `JSON`

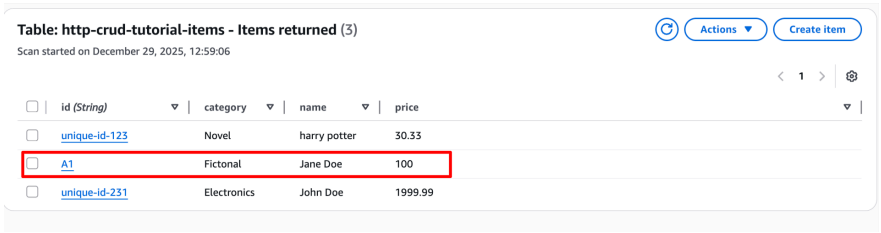
```
{
  "id": "a1b2c3d4-e5f6-7890-1234-567890abcdef", // ID in body should match URL for consistency
  "name": "Laptop Pro (Updated Model)",
  "description": "Latest generation high-performance laptop.",
  "price": 1999.99,
  "category": "Electronics"
}
```

▼ PUT with POSTMAN

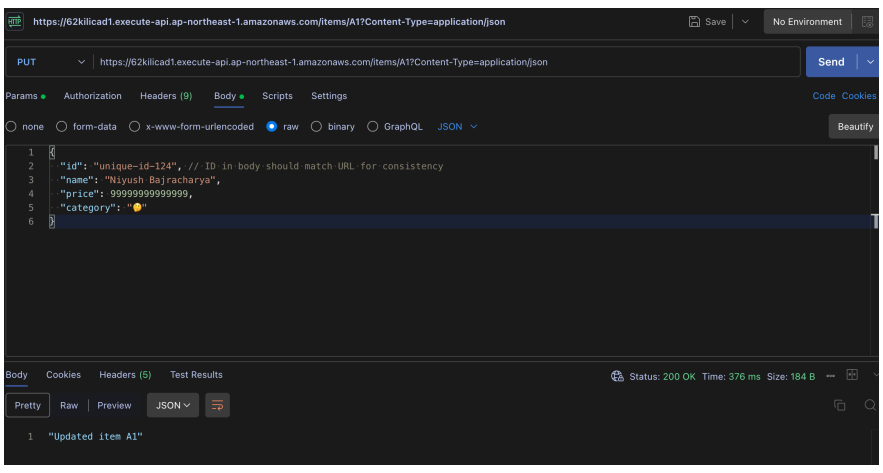


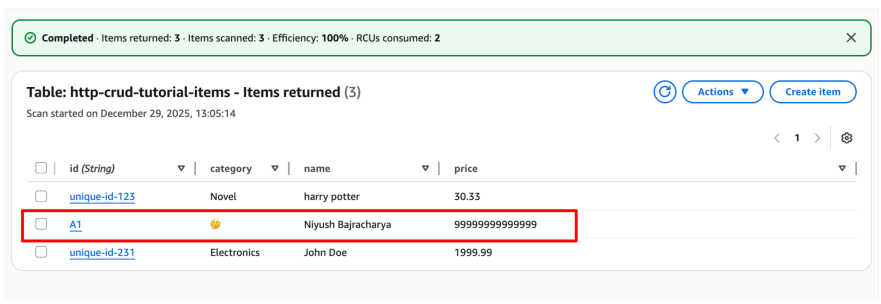


part 2 updating with POSTMAN `PUT/items/{id}`



lets' update the name and category





Test 5 : Delete an item

Test 5a : DELETE using `curl`

```
curl -X "DELETE" https://YOUR-INVOKE-URL.amazonaws.com/items/{id}
```

▼ Deleting with curl

Test 5b : DELETE using `Postman`

■ Open `Postman` ⇒ `New Request`

■ **HTTP Method** : `DELETE`

■ **URL** : `https://YOUR-INVOKE-URL.amazonaws.com/items/{id}`

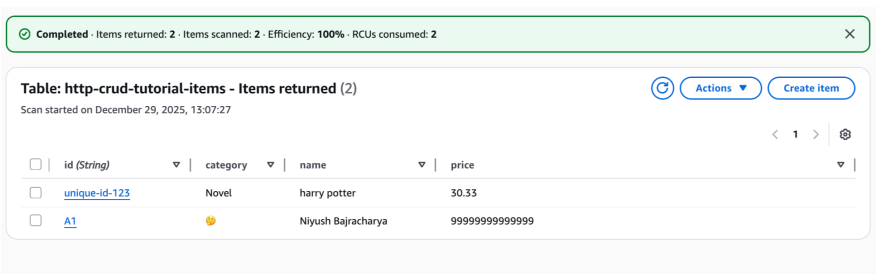
■ **Headers** : `(None typically required)`

■ **Body** : `None`

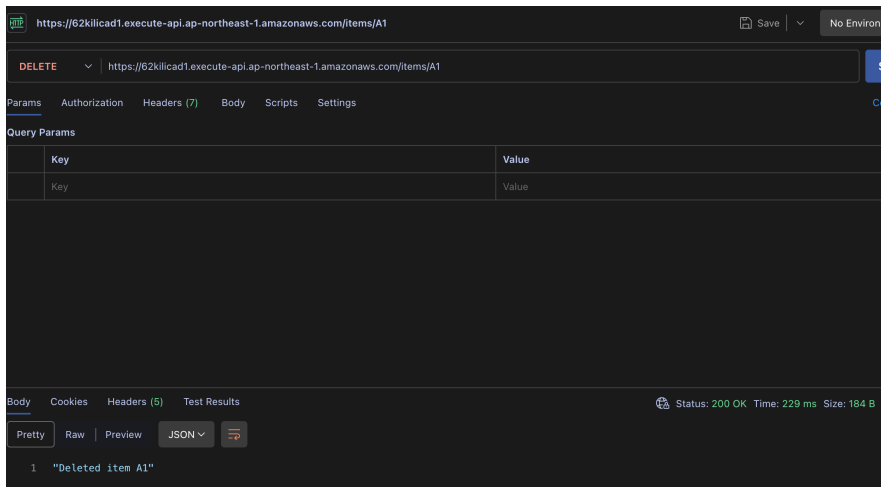
■ Click `Send`

▼ Delete using postman

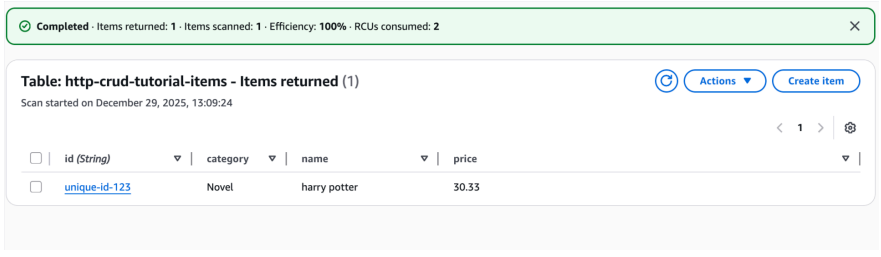
currently the db :



let’s delete the A1 using Postman



Checking the DynamoDB.

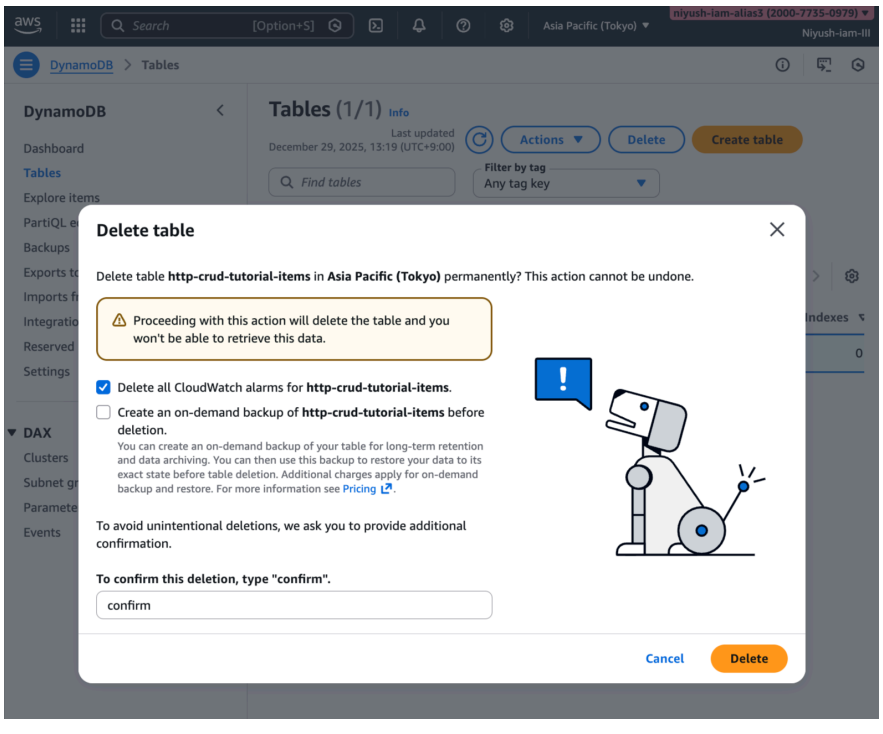


⌘ Step 8 : Clean up Time.

To prevent unnecessary cost, delete all the resources at the end.

To Delete DynamoDB Table.

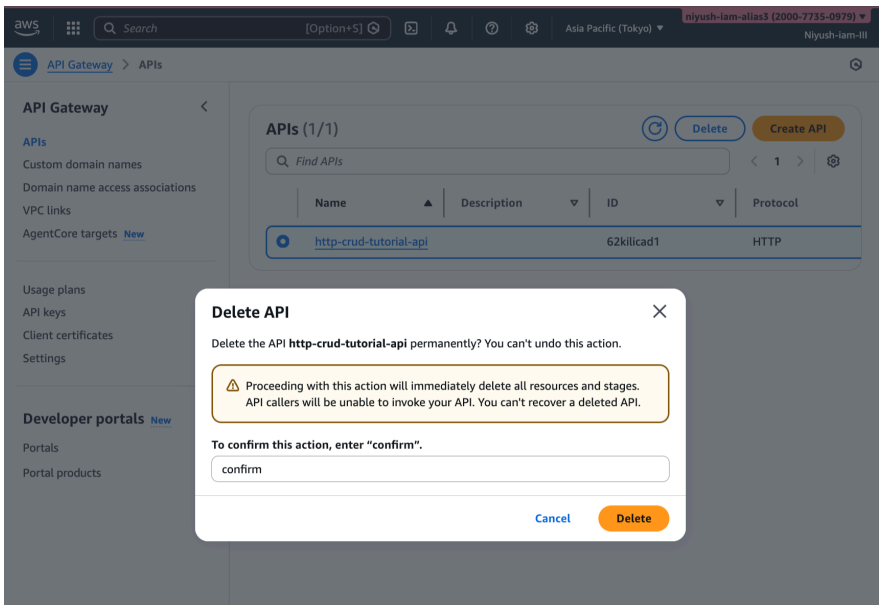
- Open DynamoDB Console ⇛ Select `Table` ⇛ **Delete table** . ▼ Delete the DynamoDB



To Delete HTTP API.

- Open API Gateway
- Select the **API** ⇛ Choose **Actions** ⇛ **Delete**

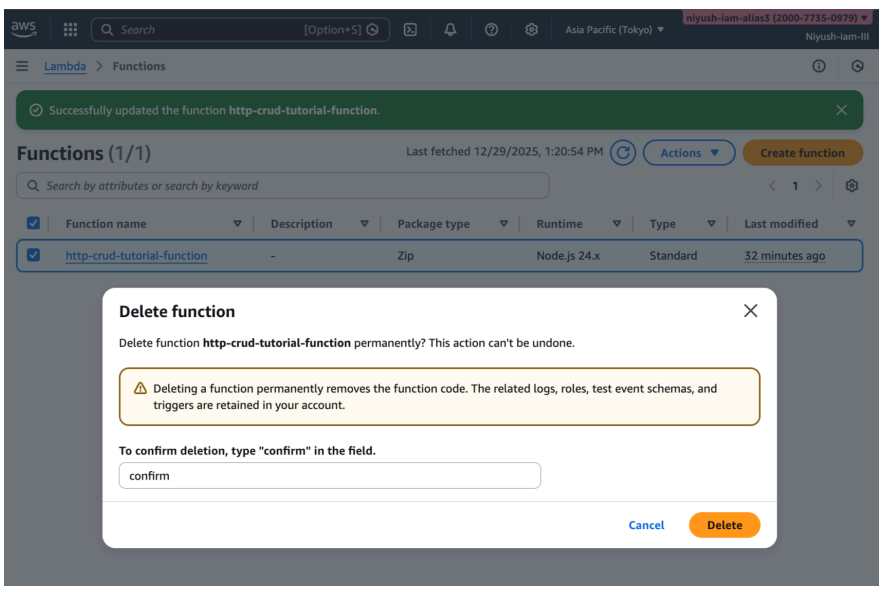
▼ Delete the API Gateway



To Delete Lambda Function.

- Open Lambda
- **Functions** page, select a function ⇒ Choose **Actions** ⇒ **Delete**

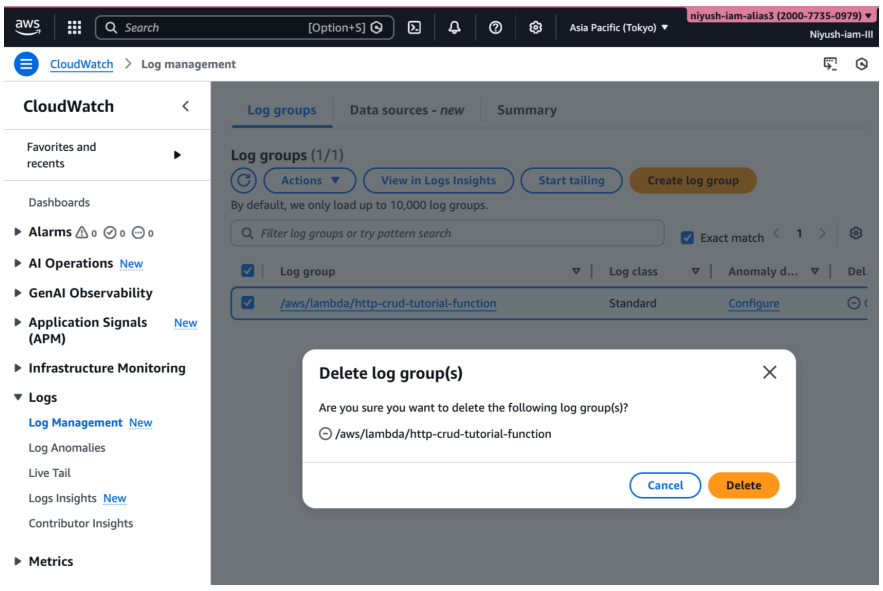
▼ Delete the Lambda Function.



To delete a Lambda function's log group

- Amazon CloudWatch console
- **Log groups** page, select the **function's log group** (/aws/lambda/http-crud-tutorial-function).
- Choose **Actions** ⇒ **Delete Log group**

▼ Delete the Cloudwatch logs

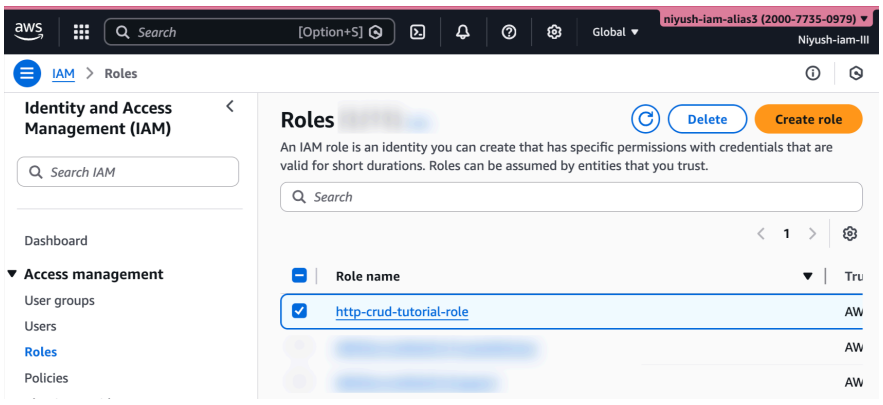


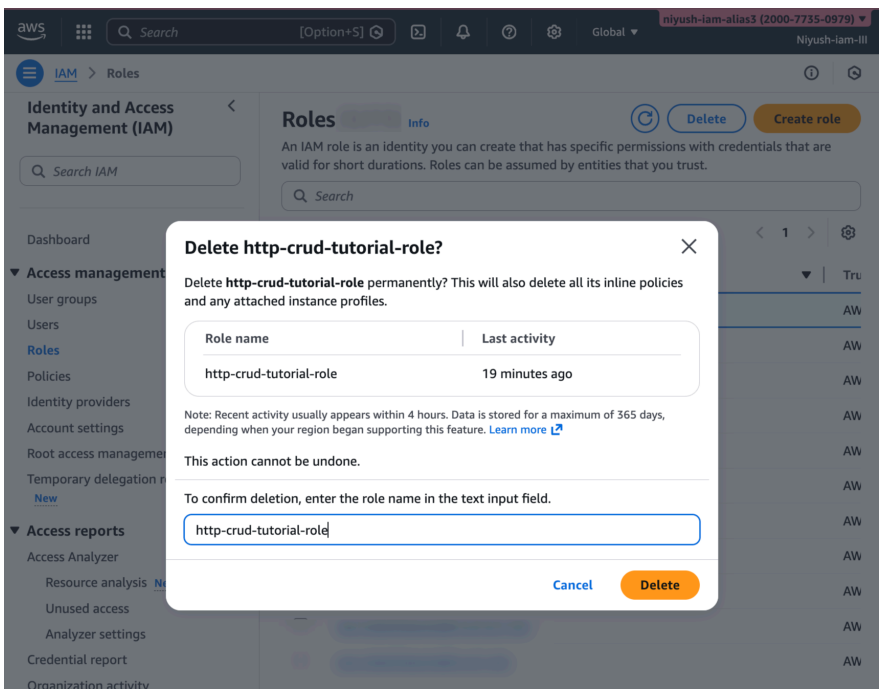
To delete a Lambda function's execution role

- In AWS IAM Role's page.

▼ Delete the IAM role

- Select the function's role `http-crud-tutorial-role`
- **Delete role**





And that’s a wrap!!!!!!!!!!
^_^