



Creating a CRUD HTTP API with Lambda and DynamoDB



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 function.

[⌘ 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"

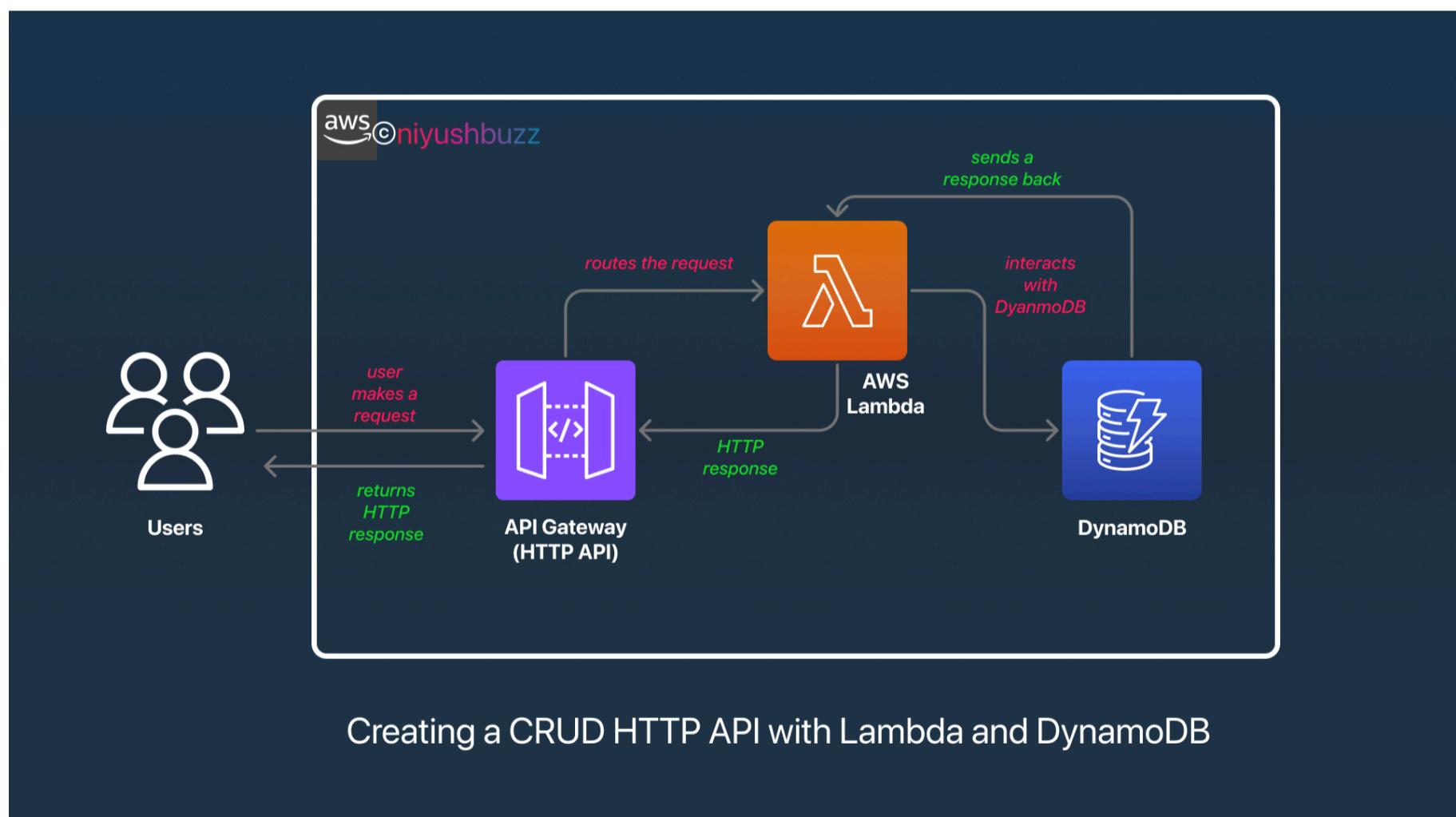
▼ What this project aims?

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



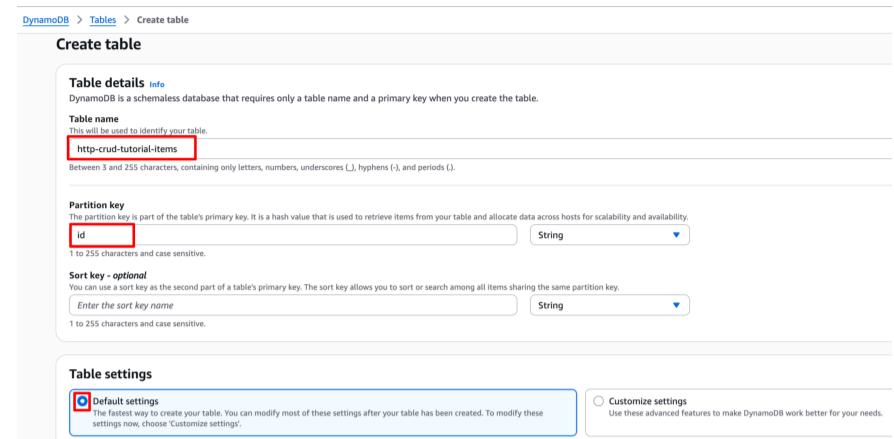
⌘ Step 1 : Creating DynamoDB Table

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.

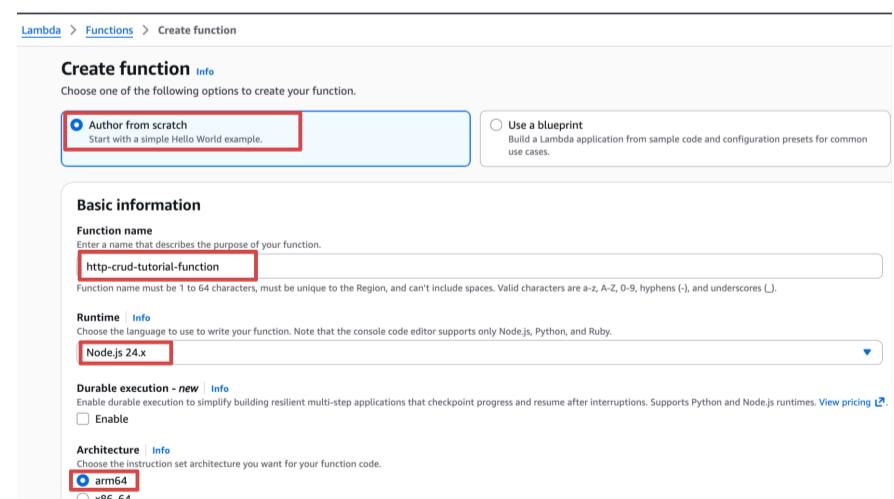
The function uses **events from API Gateway** to determine how to interact with DynamoDB. For this project we will only use one lambda function.

"Best Practice : create separate functions for each route"

To create a Lambda function.

- 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 role from 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({});

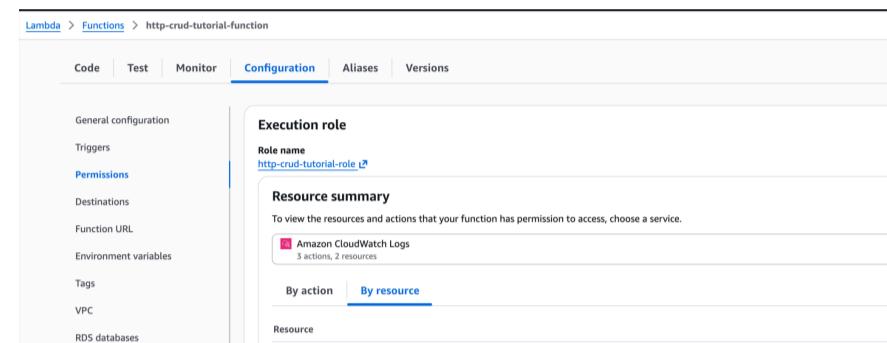
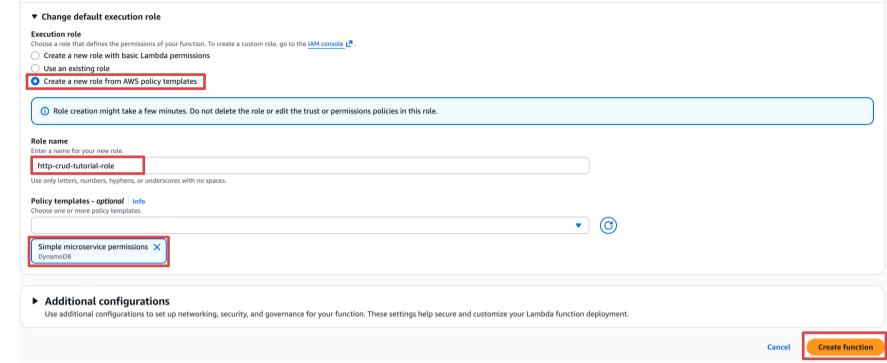
const dynamo = DynamoDBDocumentClient.from(client);

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

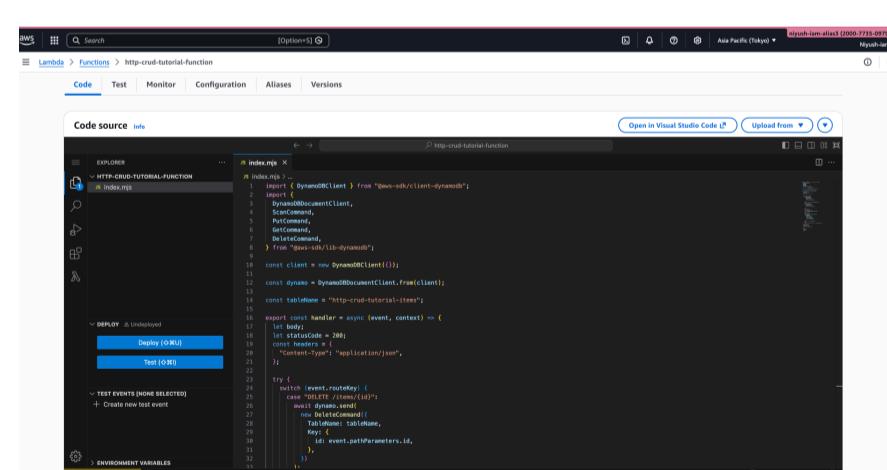
export const handler = async (event, context) => {
  let body;
  let statusCode = 200;
  const headers = {
    "Content-Type": "application/json",
  };

  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.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 "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 ${event.pathParameters.i
d}`;
  break;
default:
  throw new Error(`Unsupported route: "${event.r
outeKey}"`);
}
} catch (err) {
  statusCode = 400;
  body = err.message;
} finally {
  body = JSON.stringify(body);
}

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

```

⌘ Step 3: Creating an HTTP API.

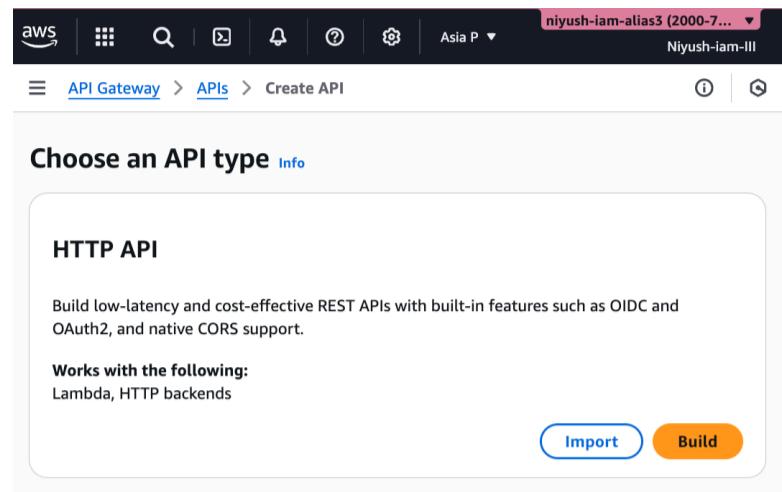
The HTTP API provides HTTP endpoint for Lambda function. We'll create and configure routes and integrations to connect API to lambda function.

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**



Configure API

API details

API name: http-crud-tutorial-api

IP address type: IPv4

Integrations: (0) info

Add integration

Cancel Review and create Next

Review and create

API name and integrations

API name: http-crud-tutorial-api

IP address type: IPv4

Integrations: No integrations configured.

Routes

Routes: No routes configured.

Stages

Stages: \$default (Auto-deploy: enabled)

Cancel Previous 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

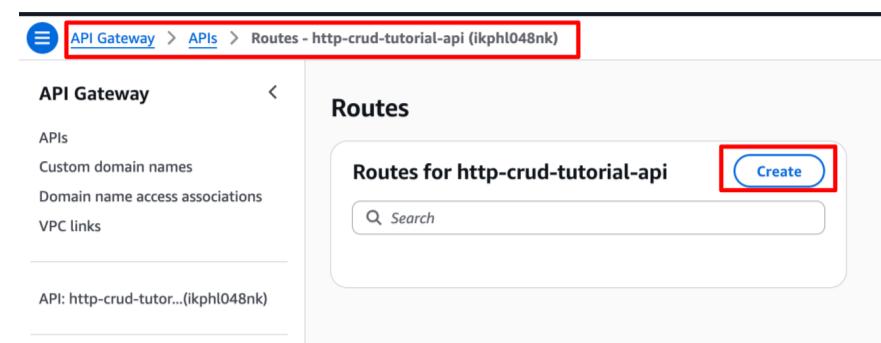
- Head over to [API Gateway](#) ⇒ [API](#)

- **Routes** ⇒ **Create**

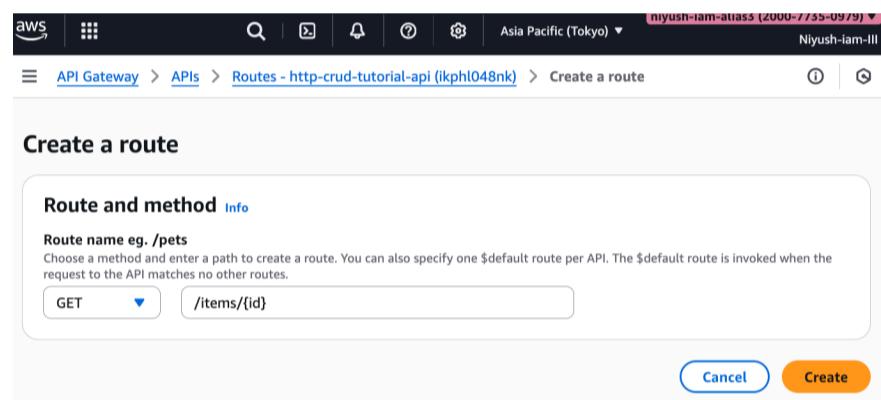
- **Method : GET**

- 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`

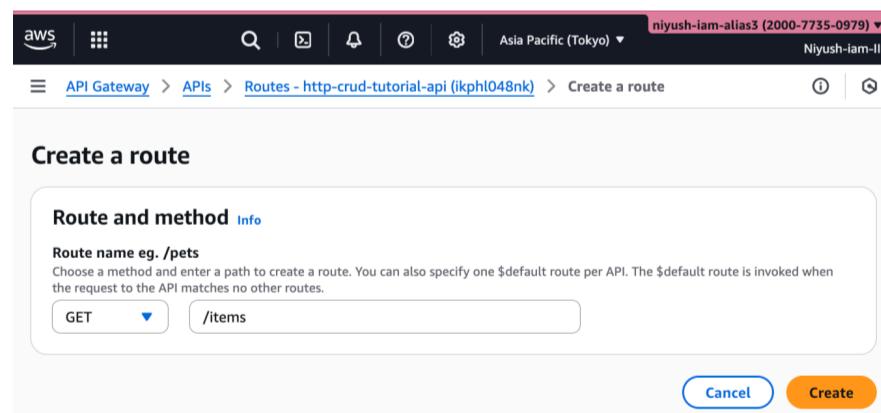
▼ Creating Routes



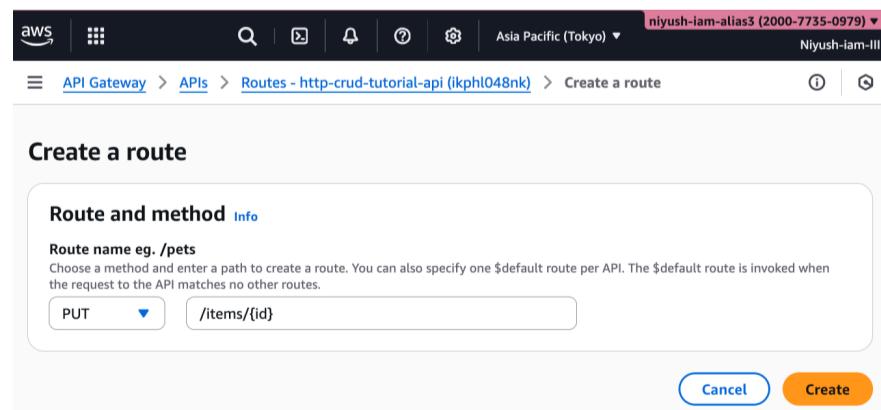
▼ GET /items/{id}



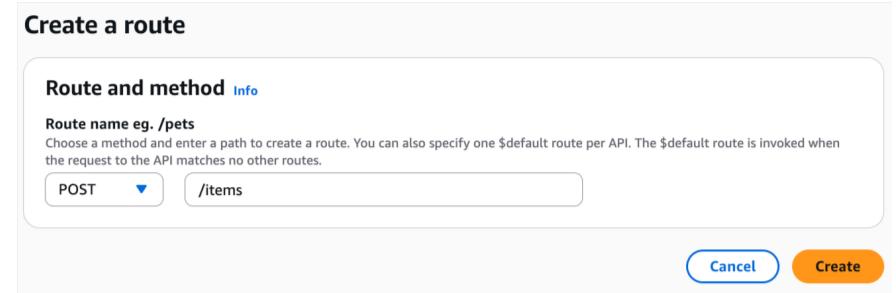
▼ Get /items



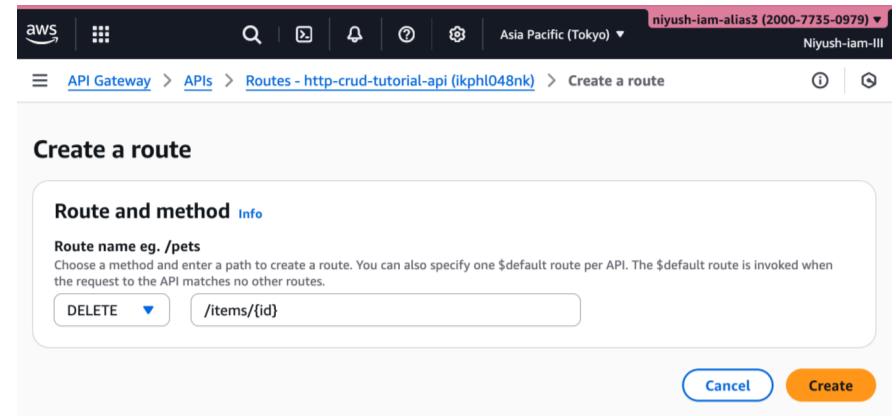
▼ PUT /items/{id}



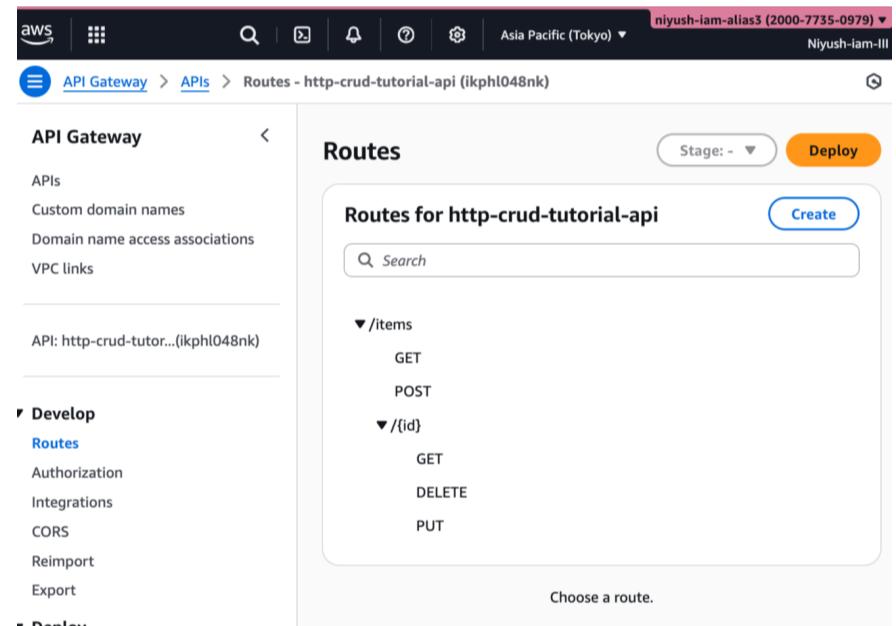
▼ POST /items



▼ DELETE /items/{id}



▼ Final Routes Path



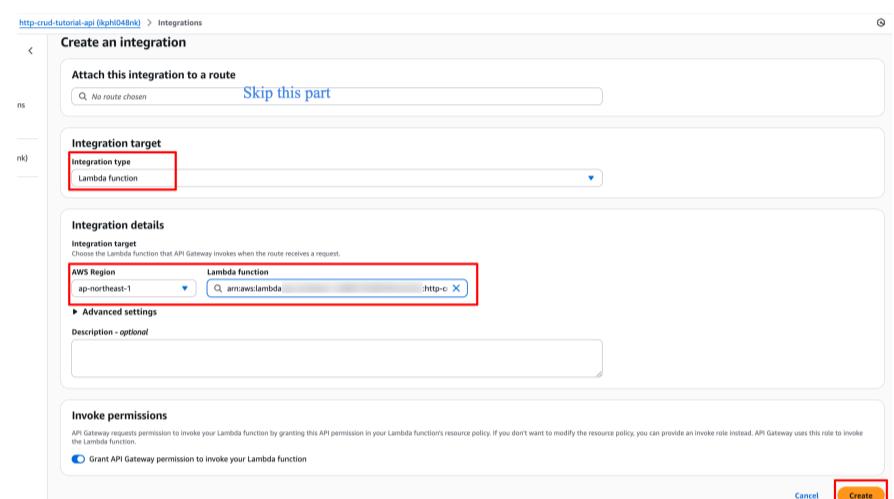
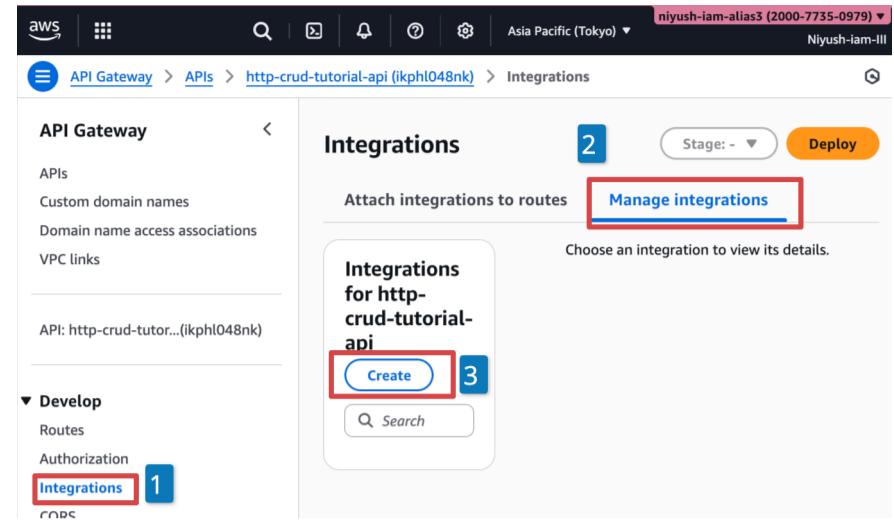
⌘ Step 5 : Creating an Integration.

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

- 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](#)

▼ Integration of API



⌘ 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.

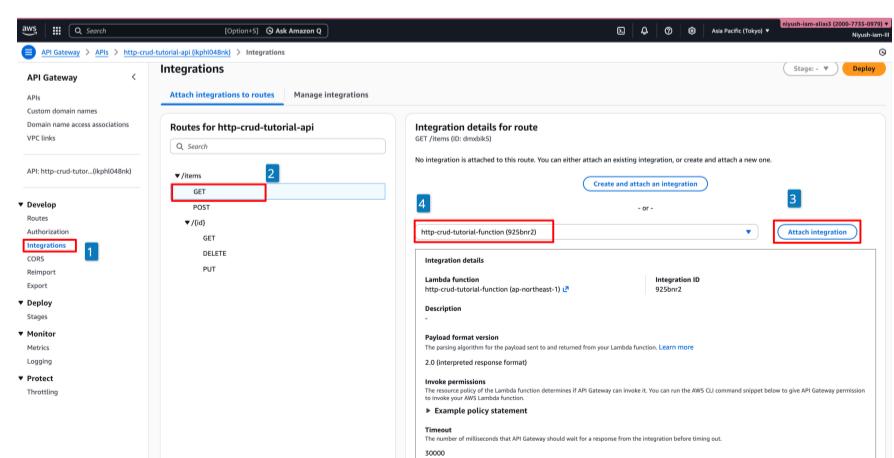
To attach integrations to routes

- 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

Integrations

Attach integrations to routes

Manage integrations

Routes for http-crud-tutorial-api

Search

▼ /items

GET AWS Lambda

POST AWS Lambda

▼ /{id}

GET AWS Lambda

DELETE AWS Lambda

PUT AWS Lambda

⌘ 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

The screenshot shows the AWS API Gateway Details page for the 'http-crud-tutorial-api' stage. On the left sidebar, the 'Stages' section is selected, with the '\$default' stage highlighted. In the main content area, the 'Stage details' section displays the 'Invoke URL' field, which is highlighted with a red box and contains the value: <https://62kilicad1.execute-api.ap-northeast-1.amazonaws.com>. Other details shown include the stage name '\$default', creation date (December 29, 2025 12:34 PM), and last update date (December 29, 2025 12:36 PM). The 'Attached deployment' section shows 'Automatic Deployment' is enabled. Deployment ID 'ttcib' and deployment created date (December 29, 2025 12:36 PM) are also listed.

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

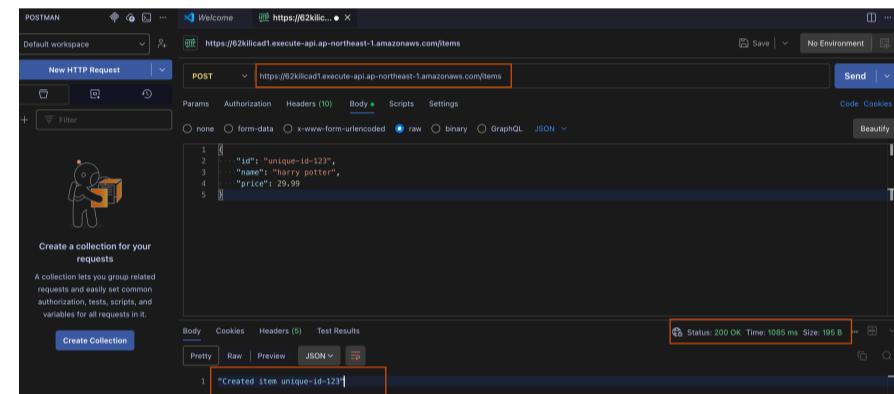
■ Key : [Content-Type](#)

■ Value : [application/json](#)

■ **Set Request Body :** [Body](#) tab ⇒ [raw](#) ⇒ [JSON](#)

■ Send request by clicking the [Send](#) button.

▼ Create using Postman [POST](#)



POST https://62kilicad1.execute-api.ap-northeast-1.amazonaws.com/items

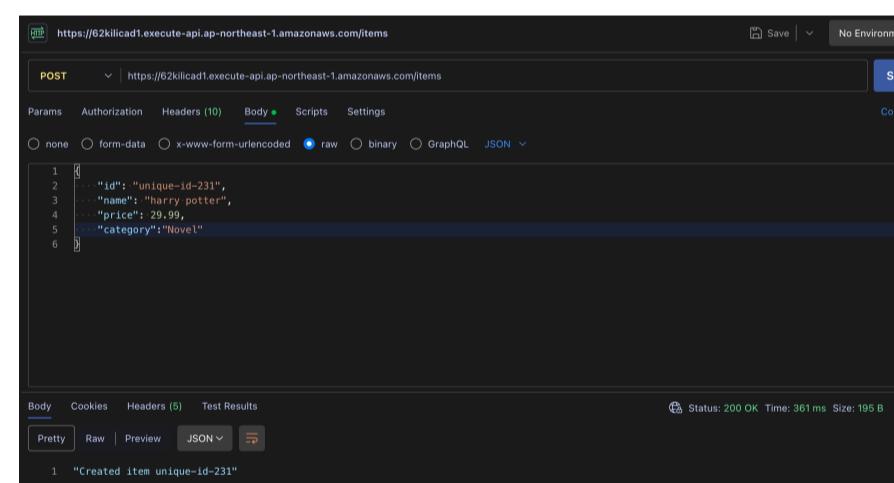
Body (10) Body

```
{
  "id": "unique-id-123",
  "name": "harry potter",
  "price": 29.99,
  "category": "Novel"
}
```

Status: 200 OK Time: 1085 ms Size: 195 B

"Created item unique-id-123"

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



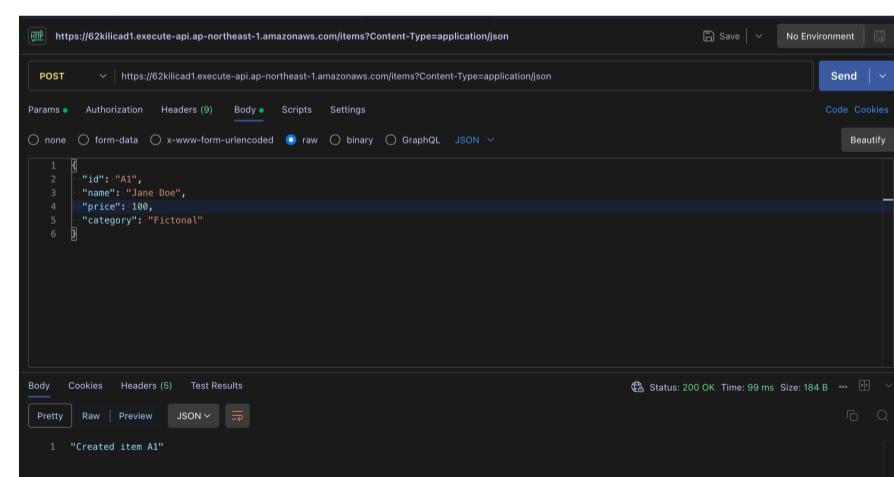
POST https://62kilicad1.execute-api.ap-northeast-1.amazonaws.com/items

Body (10) Body

```
{
  "id": "unique-id-231",
  "name": "harry potter",
  "price": 29.99,
  "category": "Novel"
}
```

Status: 200 OK Time: 361 ms Size: 195 B

"Created item unique-id-231"



POST https://62kilicad1.execute-api.ap-northeast-1.amazonaws.com/items?Content-Type=application/json

Body (9) Body

```
{
  "id": "A1",
  "name": "Jane Doe",
  "price": 100,
  "category": "Fictional"
}
```

Status: 200 OK Time: 99 ms Size: 184 B

"Created item A1"

Test 2: To get all Item. (GET)

Test 2a : GET all using curl

▼ GET items using CURL

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

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.

- i 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

The screenshot shows a browser-based API testing interface. At the top, the URL is https://62kilicad1.execute-api.ap-northeast-1.amazonaws.com/items. The status bar indicates "Save" and "No Environment". Below the URL, the method is set to "GET" and the full URL is displayed again. The navigation bar includes "Params", "Authorization", "Headers (7)", "Body", "Scripts", and "Settings". The "Headers" tab is selected, showing "Content-Type: application/json" and "User-Agent: Postman/8.0.10". The "Body" tab is active, showing the response body as JSON. The response body contains a list of four items:

```
[{"id": "unique-id-123", "name": "Harry Potter", "category": "Novel", "price": 30.33}, {"id": "test-11", "name": "SpongeBob SquarePants", "category": "Cartoon", "price": 100}, {"id": "test-13", "name": "Squidward", "category": "Cartoon", "price": 100}, {"id": "test-12", "name": "Patrick Star", "category": "Cartoon", "price": 100}]
```

The interface also shows "Status: 200 OK", "Time: 126 ms", and "Size: 469 B". Below the body, there are tabs for "Pretty", "Raw", "Preview", and "JSON".

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

Test 3a : GET one item using curl

▼ GET single item using CURL

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

```
niyushbjr@Niyuhs-MacBook-Air ~ % curl https://62kilicad1.execute-api.ap-northeast-1.amazonaws.com/items/unique-id-123  
{"price":30.33,"id":"unique-id-123","name":"harry potter","category":"Novel"}  
niyushbjr@Niyuhs-MacBook-Air ~ %
```

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-performance laptop.",
  "price": 1999.99,
  "category": "Electronics"
}' \
https://YOUR-INVOKE-URL.amazonaws.com/items/{id}
```

▼ Updating the item with CURL

Table: http-crud-tutorial-items - Items returned (2)					
	id (String)	category	name	price	
	unique-id-123		harry potter	29.99	
	unique-id-231	Electronics	John Doe	1999.99	

lets' update the category of unique ID [unique-id-123](#)

```
niyushbjr@Niyushs-MacBook-Air ~ % curl -X PUT \
-H "Content-Type: application/json" \
-d '{
  "id": "unique-id-123",
  "name": "harry potter",
  "price": 30.33,
  "category": "Novel"
}' \
https://62kilicad1.execute-api.ap-northeast-1.amazonaws.com/items/unique-id-123
"Updated item unique-id-123"
niyushbjr@Niyushs-MacBook-Air ~ %
```

Table: http-crud-tutorial-items - Items returned (2)					
	id (String)	category	name	price	
	unique-id-123	Novel	harry potter	30.33	
	unique-id-231	Electronics	John Doe	1999.99	

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 lapt  
  op.",  
  "price": 1999.99,  
  "category": "Electronics"  
}
```

▼ PUT with POSTMAN

https://62kilicad1.execute-api.ap-northeast-1.amazonaws.com/items/unique-id-231

PUT https://62kilicad1.execute-api.ap-northeast-1.amazonaws.com/items/unique-id-231

Params Authorization Headers (10) Body Scripts Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1
2  "id": "unique-id-231", //
3  "name": "John Doe",
4  "description": "Latest generation high-performance laptop.",
5  "price": 1999.99,
6  "category": "Electronics"
7
```

Body Cookies Headers (5) Test Results

Pretty Raw Preview JSON

Status: 200 OK Time: 332 ms Size: 195 B

1 "Updated item unique-id-231"

Table: http-crud-tutorial-items - Items returned (2)				
Scan started on December 29, 2025, 12:50:19				
	id (String)	category	name	price
	unique-id-123		harry potter	29.99
	unique-id-231	Electronics	John Doe	1999.99

part 2 updating with POSTMAN [PUT/items/{id}](#)

Table: http-crud-tutorial-items - Items returned (3)				
Scan started on December 29, 2025, 12:59:06				
	id (String)	category	name	price
	unique-id-123	Novel	harry potter	30.33
	A1	Fictional	Jane Doe	100
	unique-id-231	Electronics	John Doe	1999.99

lets' update the name and category

https://62kilicad1.execute-api.ap-northeast-1.amazonaws.com/items/A1?Content-Type=application/json

PUT https://62kilicad1.execute-api.ap-northeast-1.amazonaws.com/items/A1?Content-Type=application/json

Params Authorization Headers (9) Body Scripts Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1
2  "id": "unique-id-124", // ID in body should match URL for consistency
3  "name": "Niyush Bajracharya",
4  "price": 9999999999999999,
5  "category": "Fictional"
6
```

Body Cookies Headers (5) Test Results

Pretty Raw Preview JSON

Status: 200 OK Time: 376 ms Size: 184 B

1 "Updated item A1"

Completed · Items returned: 3 · Items scanned: 3 · Efficiency: 100% · RCU consumed: 2				
Table: http-crud-tutorial-items - Items returned (3)				
Scan started on December 29, 2025, 13:05:14				
	id (String)	category	name	price
	unique-id-123	Novel	harry potter	30.33
	A1	Fictional	Niyush Bajracharya	9999999999999999
	unique-id-231	Electronics	John Doe	1999.99

Test 5 : Delete an item

Test 5a : DELETE using curl

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

▼ Deleting with curl

```
niyushbjr@Niyushs-MacBook-Air ~ % curl -X "DELETE" https://62kilicad1.execute-api.ap-northeast-1.amazonaws.com/items/unique-id-231
"Deleted item unique-id-231"
niyushbjr@Niyushs-MacBook-Air ~ %
```

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 :

Completed - Items returned: 2 - Items scanned: 2 - Efficiency: 100% - RCU consumed: 2					
Table: http-crud-tutorial-items - Items returned (2)					
Scan started on December 29, 2025, 13:07:27					
	id (String)	category	name	price	
□	unique-id-123	Novel	harry potter	30.33	
□	A1	吁	Niyush Bajracharya	9999999999999999	

let's delete the A1 using Postman

The screenshot shows a Postman request to the endpoint <https://62kilicad1.execute-api.ap-northeast-1.amazonaws.com/items/A1>. The method is set to DELETE. The response status is 200 OK, and the body contains the message "Deleted item A1".

Checking the DynamoDB.

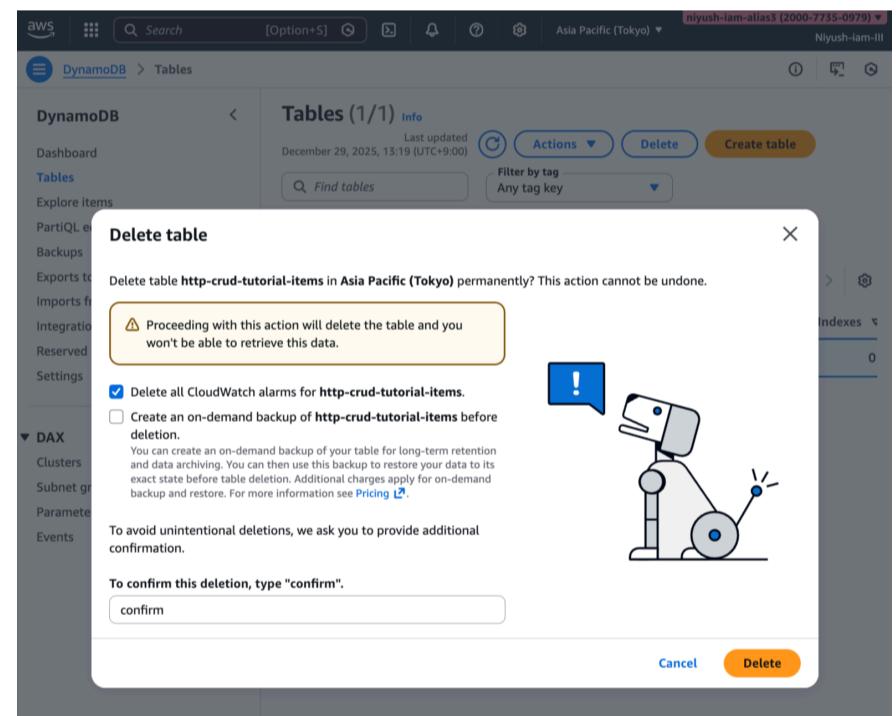
Completed - Items returned: 1 - Items scanned: 1 - Efficiency: 100% - RCUs consumed: 2				
Table: http-crud-tutorial-items - Items returned (1)				
Scan started on December 29, 2025, 13:09:24				
	id (String)	category	name	price
<input type="checkbox"/>	unique-id-123	Novel	harry potter	30.33

⌘ Step 8 : Clean up Time.

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

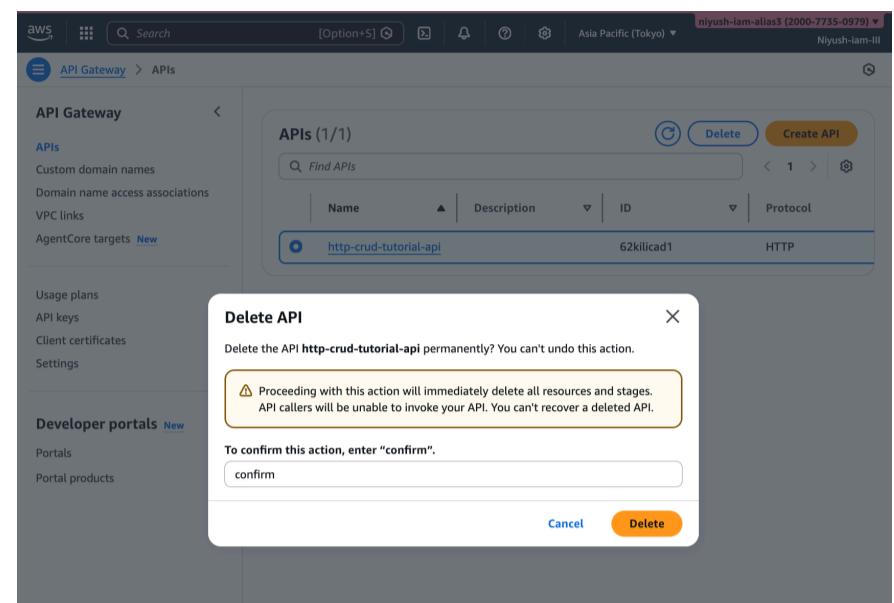
To Delete DynamoDB Table.

- Open [DynamoDB Console](#) ⇒ Select [Table](#) ⇒ **Delete table** . ▼ Delete the DynamoDB



To Delete HTTP API.

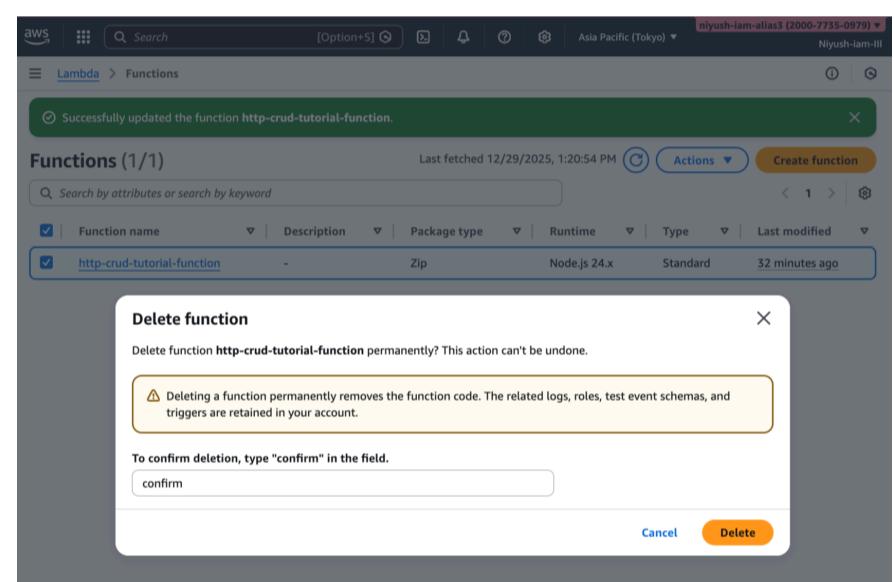
- Open [API Gateway](#)
- Select the **API** ⇒ Choose **Actions** ⇒ **Delete**
- ▼ Delete the API Gateway



To Delete Lambda Function.

- Open [Lambda](#)
- **Functions** page, select a function ⇒ Choose **Actions** ⇒ **Delete**

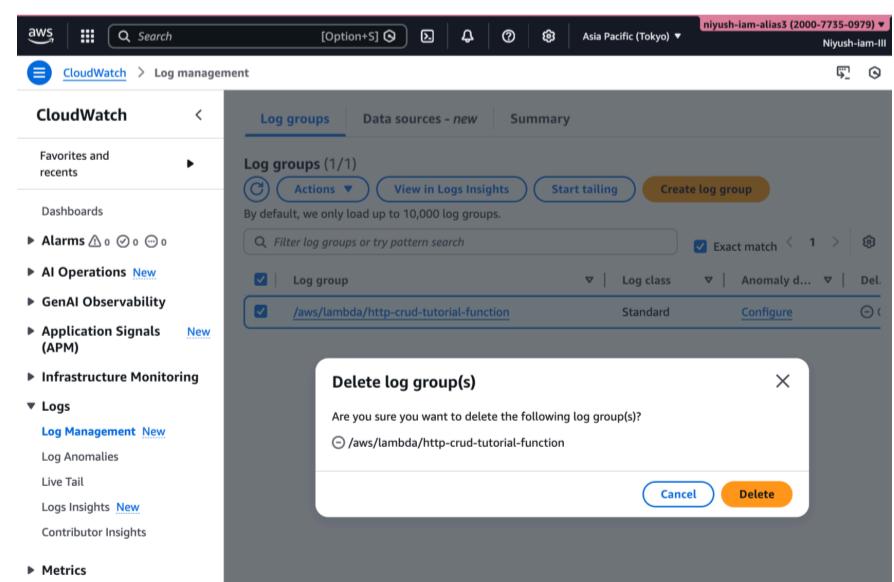
▼ Delete the Lambda Function.



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



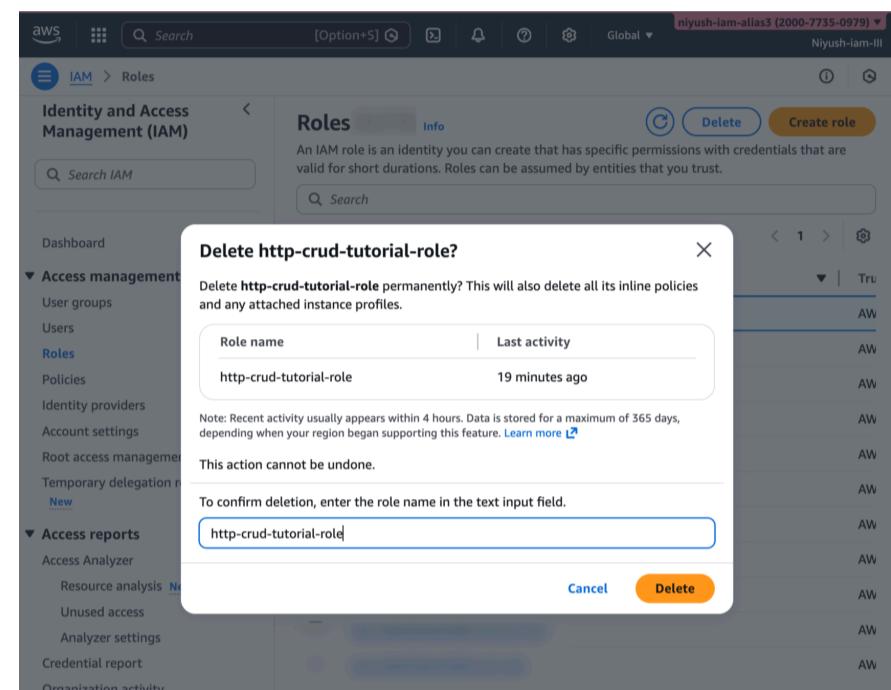
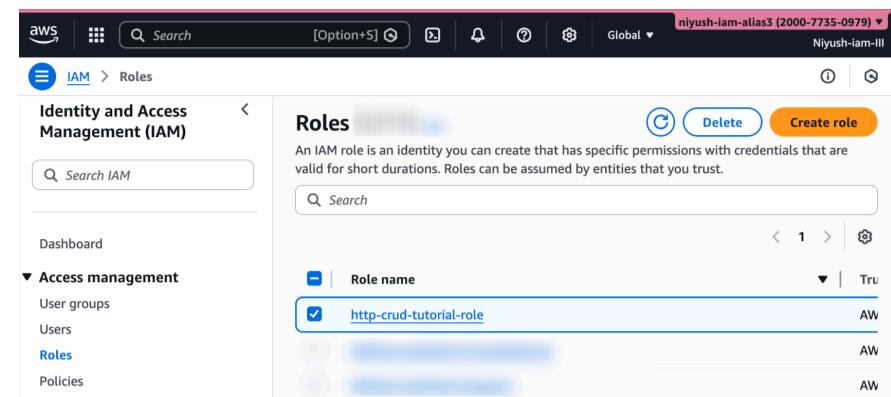
To delete a Lambda function's execution role

- 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!!!!!!!!!

^ ^