1. Introduction

This document explains how to build a **RESTful API** using **AWS Lambda**, **API Gateway**, and **DynamoDB**. The process includes:

- **Creating a DynamoDB table** to store data.
- **Creating a Lambda function** to perform create, read, update, and delete operations on the DynamoDB data.
- **Connecting Lambda to API Gateway** to provide an HTTP endpoint.
- **Testing the API** using **cURL**.

---

2. Creating a DynamoDB Table

2.1. Open the DynamoDB Console:

- Click **Create Table**.

2.2. Table Settings:

- **Table Name:** `http-crud-tutorial-items`
- **Partition Key:** `id` (type: String).

2.3. Create the Table:

- Click **Create Table** to create the table.

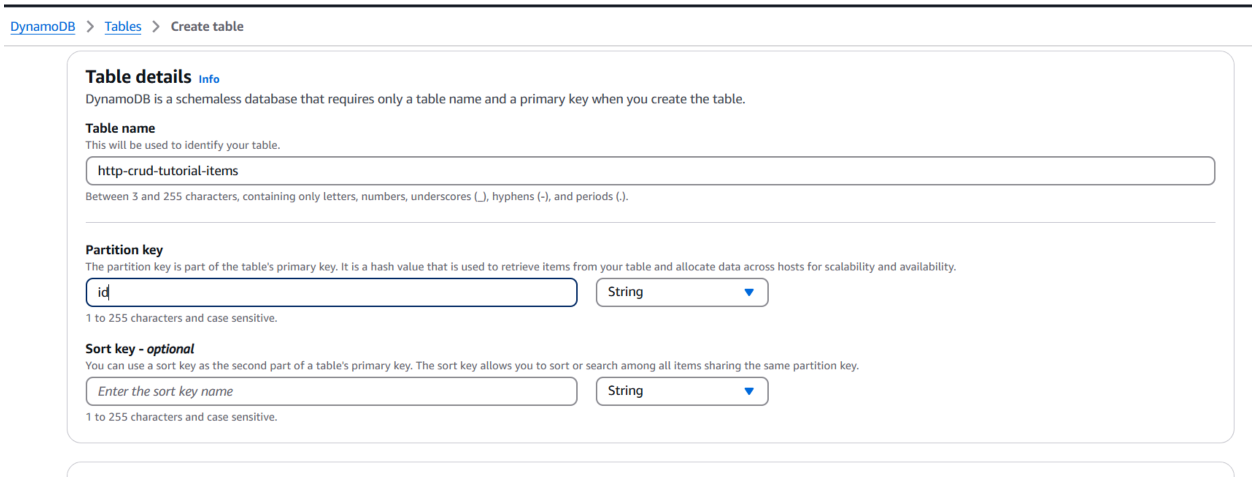


Table details Info

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.

http-crud-tutorial-items

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.

id | String

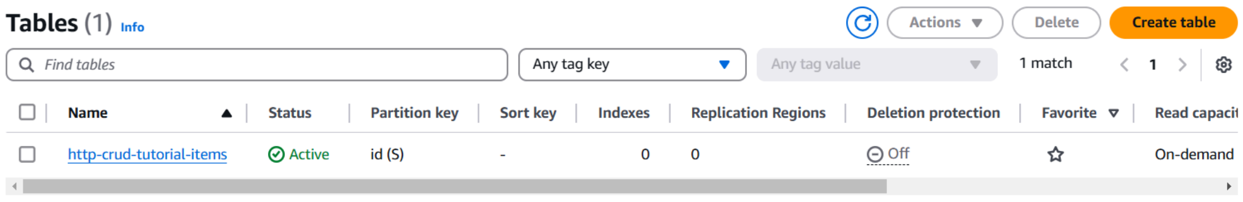
1 to 255 characters and case sensitive.

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.

Enter the sort key name | String

1 to 255 characters and case sensitive.


Tables (1) Info

Actions | Delete | Create table

Find tables | Any tag key | Any tag value | 1 match | 1

	Name	Status	Partition key	Sort key	Indexes	Replication Regions	Deletion protection	Favorite	Read capacit
☐	http-crud-tutorial-items	Active	id (S)	-	0	0	Off	☆	On-demand

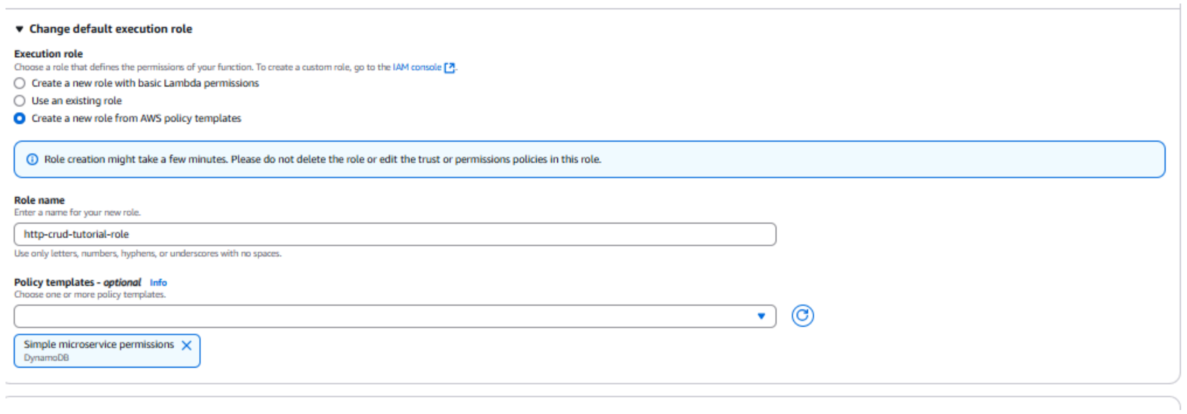
3. Creating the Lambda Function

3.1. Open the Lambda Console

- Click **Create Function**.

3.2. Lambda Settings:

- **Function Name:** `http-crud-tutorial-function`
- **Runtime:** Choose `Node.js` (or Python if preferred).
- Under **Permissions**, choose **Change default execution role**.
- Select **Create a new role from AWS policy templates** and enter the role name `http-crud-tutorial-role`, then choose **Simple microservice permissions**.


▼ 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.

http-crud-tutorial-role

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

Policy templates - *optional* Info

Choose one or more policy templates.

Simple microservice permissions
DynamoDB

-

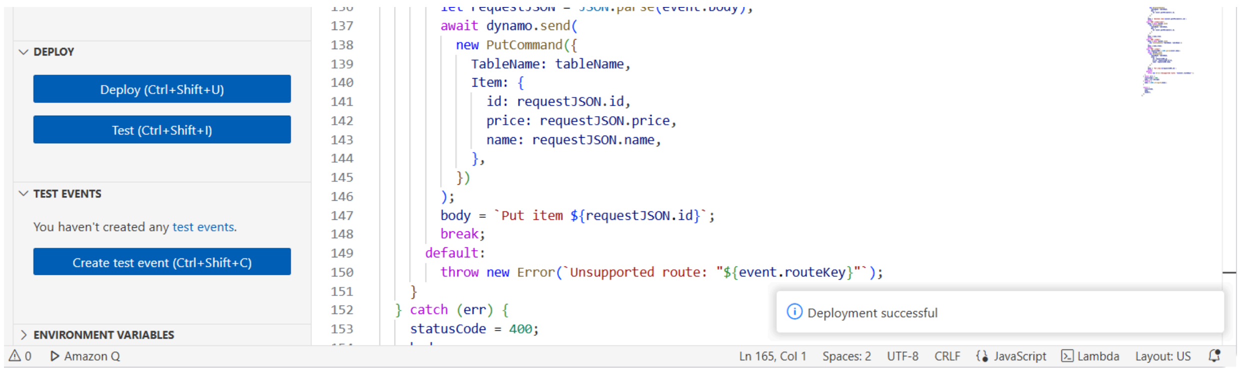
3.3. Add the Code to Lambda:

Replace the default code with the following code to handle operations on DynamoDB:

javascript
Copy 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 "PUT /items":
        let requestJSON = JSON.parse(event.body);
        await dynamo.send(
          new PutCommand({
            TableName: tableName,
            Item: {
              id: requestJSON.id,
              price: requestJSON.price,
              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,
  };
};
```

3.4. Deploy the Code:

- Click **Deploy** to update the Lambda function.



You haven't created any test events.

```
        await dynamo.send(
          new PutCommand({
            TableName: tableName,
            Item: {
              id: requestJSON.id,
              price: requestJSON.price,
              name: requestJSON.name,
            },
          })
        );
        body = `Put item ${requestJSON.id}`;
        break;
      default:
        throw new Error(`Unsupported route: "${event.routeKey}"`);
    }
  } catch (err) {
    statusCode = 400;
```

Deployment successful

---

4. Creating an API using API Gateway

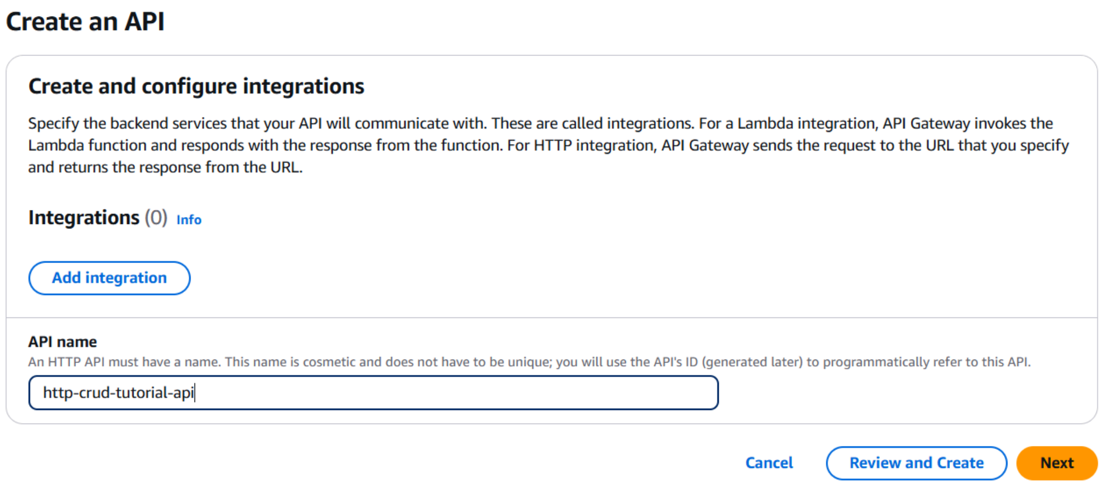
4.1. Open the API Gateway Console:

- Click **Create API**, and then choose **Build** under HTTP API.

4.2. API Settings:

- **API Name:** `http-crud-tutorial-api`.

- Click **Next** and then **Create**.


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.

http-crud-tutorial-api

---

5. Creating Routes

5.1. Create API Routes:

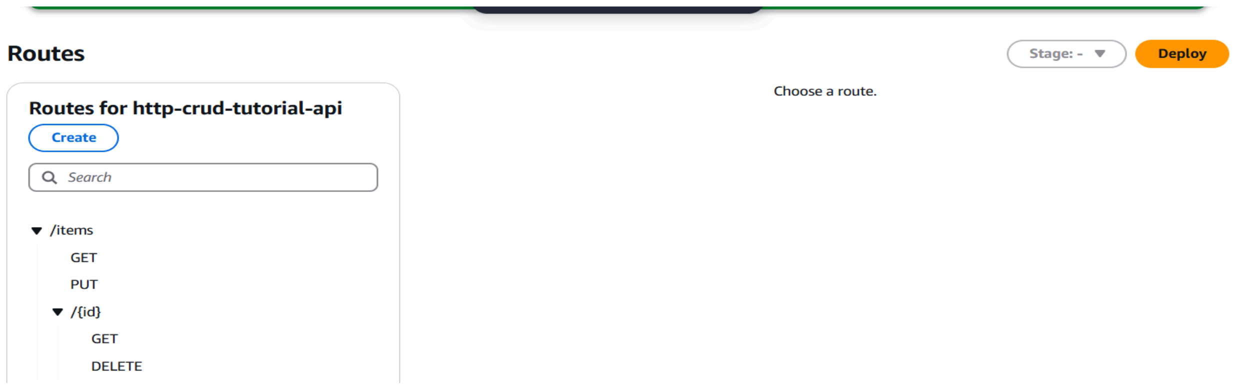
To configure the API and define accessible operations, create four routes:

1. **GET /items/{id}** for reading an item by its `id`.
2. **GET /items** for reading all items.
3. **PUT /items** for creating or updating an item.
4. **DELETE /items/{id}** for deleting an item by its `id`.

5.2. Set Up Routes:

- In the **API Gateway** console, select your created API.

- Choose **Routes** and then click **Create** to create each of the above routes.



```
/items
    GET
    PUT
    /{id}
        GET
        DELETE
```

---

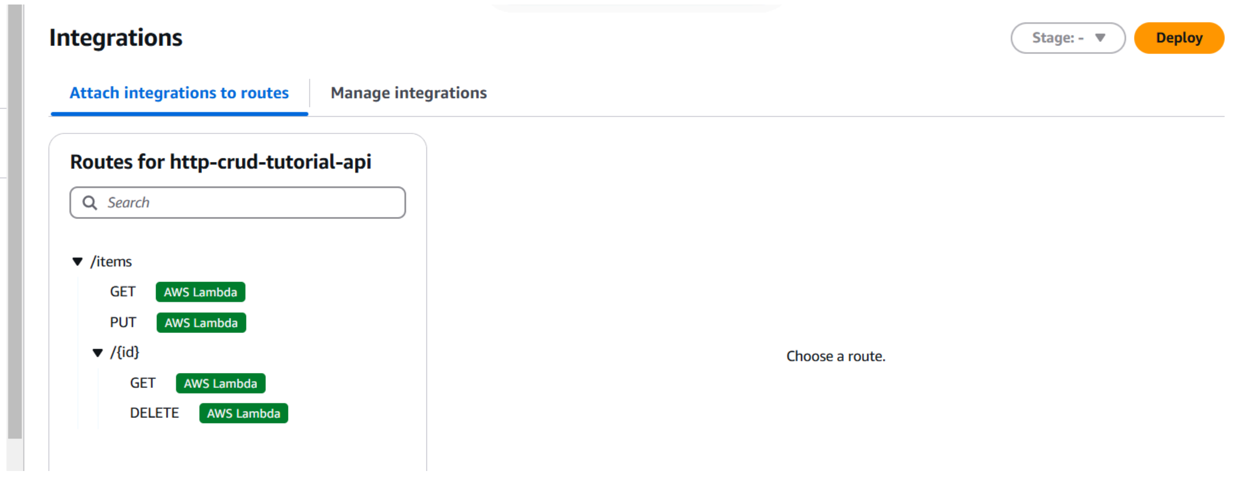
6. Connecting Lambda to API Gateway

6.1. Create an Integration:

- Choose **Integrations** in the **API Gateway** console.
- Click **Create** and choose **Lambda function**.
- Select the Lambda function you created (e.g., `http-crud-tutorial-function`) and click **Create**.

6.2. Attach Integration to Routes:

- Select each route in **Routes**.
- Choose **Attach Integration** and select the **Lambda** function you created.
- Repeat this process for all routes.



```
/items
    GET     AWS Lambda
    PUT     AWS Lambda
    /{id}
        GET     AWS Lambda
        DELETE  AWS Lambda
```

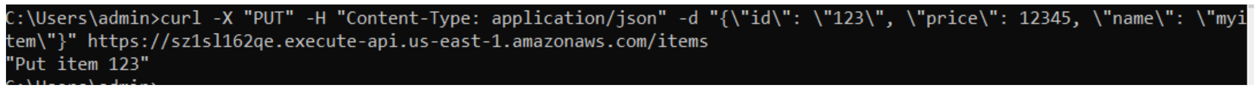
---

7. Testing the API using cURL

7.1. Use cURL to Test the Routes:

Create or Update an Item:
bash
Copy code

```
curl -X "PUT" -H "Content-Type: application/json" -d "{\"id\":
\"123\", \"price\": 12345, \"name\": \"myitem\"}"
https://sz1sl162qe.execute-api.us-east-1.amazonaws.com/items
```

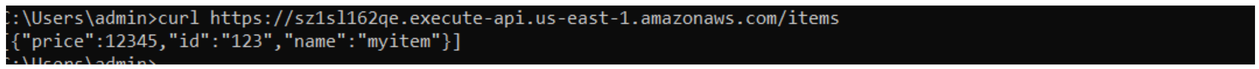


```
C:\Users\admin>curl -X "PUT" -H "Content-Type: application/json" -d "{\"id\": \"123\", \"price\": 12345, \"name\": \"myi
tem\"}" https://sz1sl162qe.execute-api.us-east-1.amazonaws.com/items
"Put item 123"
```

•

Get All Items:
bash
Copy code

```
curl https://sz1sl162qe.execute-api.us-east-1.amazonaws.com/items
```

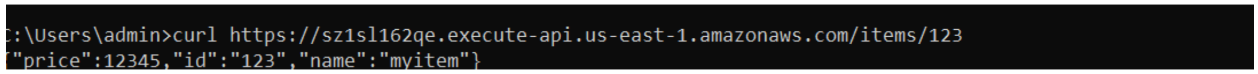


```
:\Users\admin>curl https://sz1sl162qe.execute-api.us-east-1.amazonaws.com/items
[{"price":12345,"id":"123","name":"myitem"}]
```

•

Get a Specific Item:
bash
Copy code

```
curl https://sz1sl162qe.execute-api.us-east-1.amazonaws.com/items/123
```

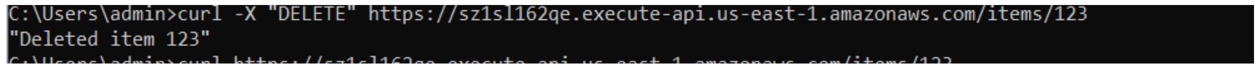


```
:\Users\admin>curl https://sz1sl162qe.execute-api.us-east-1.amazonaws.com/items/123
{"price":12345,"id":"123","name":"myitem"}
```

•

Delete an Item:
bash
Copy code

```
curl -X "DELETE"
https://sz1sl162qe.execute-api.us-east-1.amazonaws.com/items/123
```



```
C:\Users\admin>curl -X "DELETE" https://sz1sl162qe.execute-api.us-east-1.amazonaws.com/items/123
"Deleted item 123"
```

•

8. Clean Up

8.1. Delete Resources:

To prevent unnecessary costs, delete the resources you created:

- **Delete the DynamoDB Table**:
  - Open dynamoDB, select your table, and click **Delete Table**.
- **Delete the HTTP API**:
  - Go to the api gateway, select the API, and click **Actions**, then **Delete**.
- **Delete the Lambda Function**:
  - Visit the lambda, select your function, click **Actions**, and choose **Delete**.
- **Delete the Lambda Log Group**:
  - In cloudwatch, find the log group (/aws/lambda/http-crud-tutorial-function) and delete it.