Guided Lab: Creating an AWS Lambda Function that Interacts with an Amazon DynamoDB Table

Description

AWS Lambda is a serverless compute service that allows you to run code without provisioning or managing servers. DynamoDB is a fast and flexible NoSQL database service for all applications that need consistent, single-digit millisecond latency at any scale.

We will integrate these services to build a simple backend that handles basic HTTP operations—PUT, GET, POST, and DELETE—on a DynamoDB table. These HTTP methods correspond to common operations for interacting with a backend service:

- **GET**: Retrieve data from the database.
- **POST**: Create a new item in the database.
- **PUT**: Update an existing item in the database.
- **DELETE**: Remove an item from the database.

Prerequisites

This lab assumes you have a basic understanding of JavaScript (Node.js) and Familiarity with AWS Lambda functions and Dynamo DB (NoSQL Database).

If you find any gaps in your knowledge, consider taking the following lab:

- Creating an Amazon DynamoDB table
- Creating a NodeJS Function in AWS Lambda

Objectives

By the end of this lab, you will:

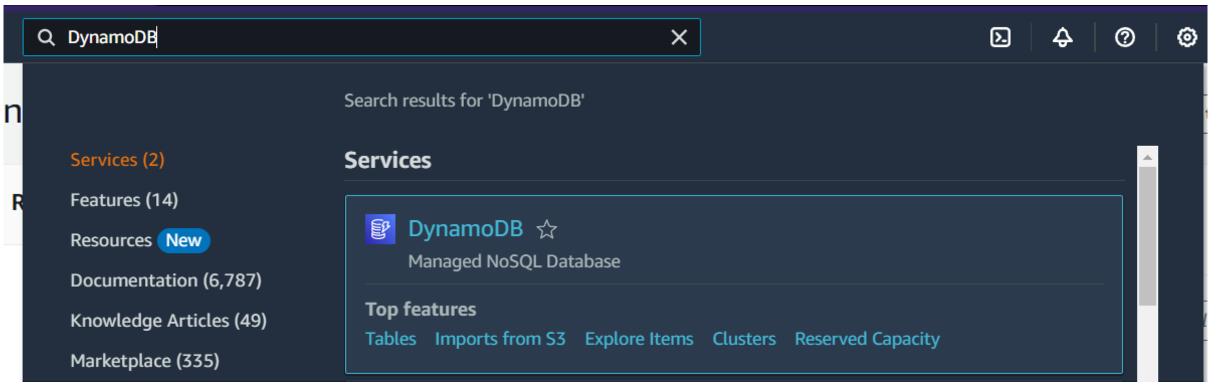
- Build a simple backend service using Lambda and DynamoDB.
- Implement basic HTTP methods (GET, POST, PUT, DELETE) to interact with a DynamoDB table.
- Test the Lambda function with predefined test events to validate its functionality.
- Gain hands-on experience with serverless architecture and NoSQL databases on AWS.

Subscribe to access AWS PlayCloud Labs

Lab Steps

Create the DynamoDB Table

1. Navigate to the DynamoDB service in the AWS Management Console.


DynamoDB

Search results for 'DynamoDB'

Services (2)
Features (14)
Resources New
Documentation (6,787)
Knowledge Articles (49)
Marketplace (335)

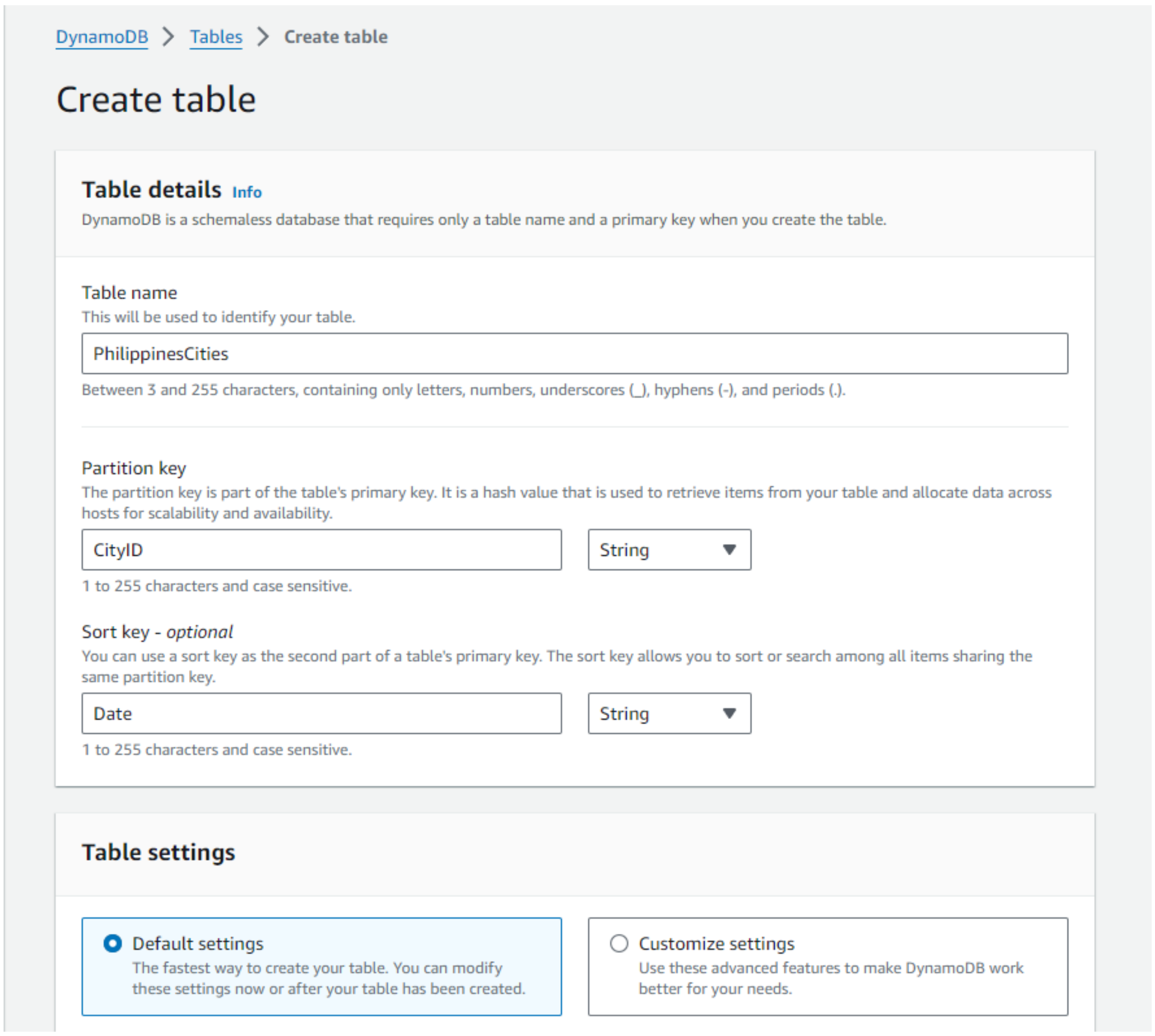
Services

DynamoDB
Managed NoSQL Database

Top features
Tables Imports from S3 Explore Items Clusters Reserved Capacity

2. Create a new table with the following configurations:

- Table name: **PhilippinesCities**.
- Primary key: CityID (String)
- Sort key: Date (String)
- Table settings: Select Default settings


DynamoDB > Tables > Create table

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.
PhilippinesCities
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.
CityID | 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.
Date | String
1 to 255 characters and case sensitive.

Table settings

[✓ Default settings]
The fastest way to create your table. You can modify these settings now or after your table has been created.

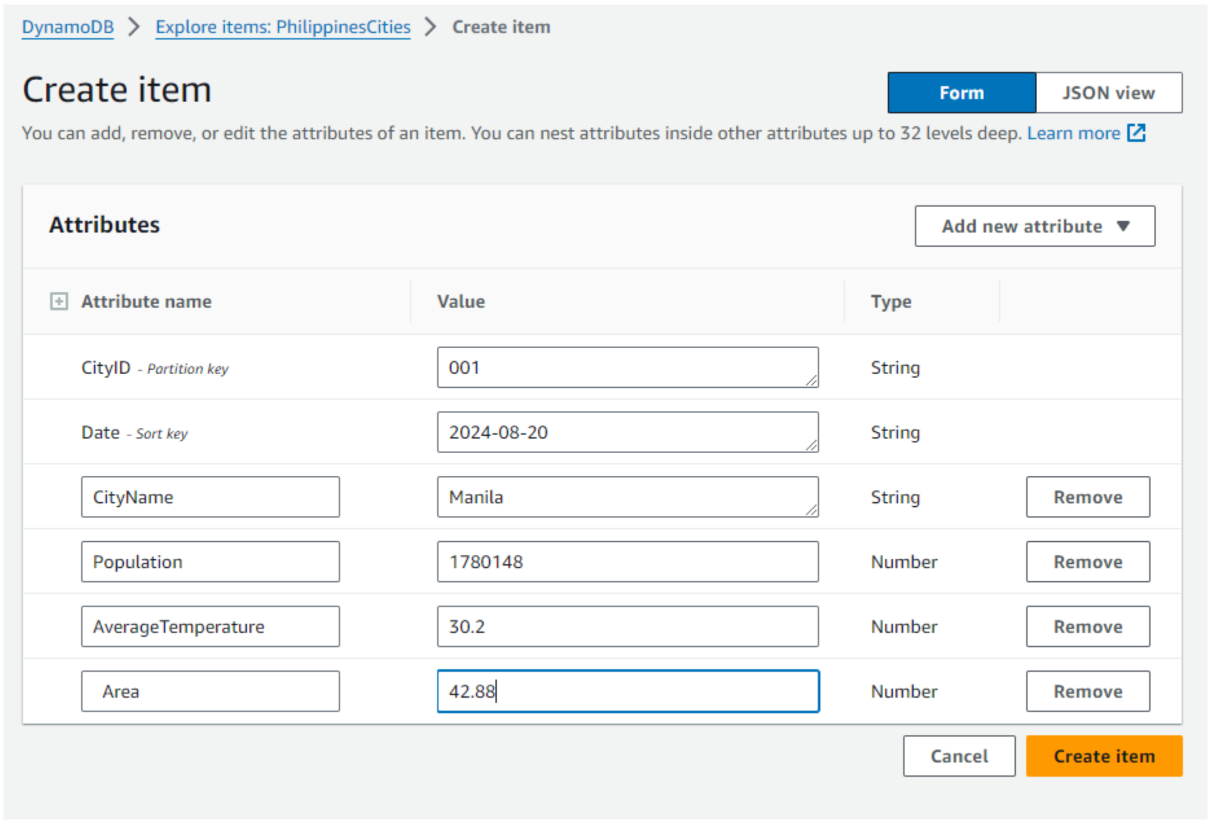
Customize settings
Use these advanced features to make DynamoDB work better for your needs.

- Click **Create table**

3. Ensure the table status is set to Active before proceeding to the next step.

4. Create the following items for the **PhilippinesCities** Table:

- **Item 1:**
  - CityID: **"001" (String)**
  - Date: **"2024-08-20" (String)**
  - CityName: **"Manila" (String)**
  - Population: **1780148 (Number)**
  - AverageTemperature: **30.2 (Number)**
  - Area: **42.88 (Number)**



Create item

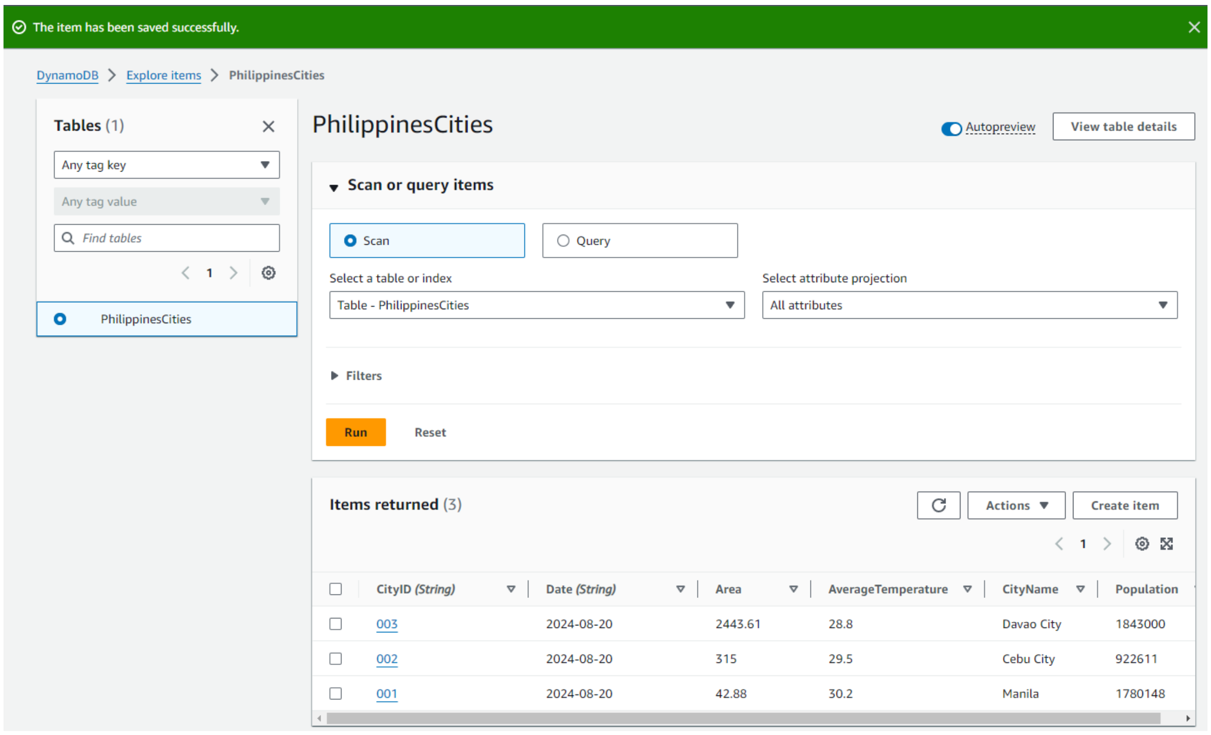
You can add, remove, or edit the attributes of an item. You can nest attributes inside other attributes up to 32 levels deep. Learn more

Attributes

Attribute name	Value	Type	
CityID - *Partition key*	001	String	
Date - *Sort key*	2024-08-20	String	
CityName	Manila	String	Remove
Population	1780148	Number	Remove
AverageTemperature	30.2	Number	Remove
Area	42.88	Number	Remove

- **Item 2:**
  - CityID: **"002" (String)**
  - Date: **"2024-08-20" (String)**
  - CityName: **"Cebu City" (String)**
  - Population: **922611 (Number)**
  - AverageTemperature: **29.5 (Number)**
  - Area: **315.00 (Number)**
- **Item 3:**

- CityID: **“003” (String)**
- Date: **“2024-08-20” (String)**
- CityName: **“Davao City” (String)**
- Population: **1843000 (Number)**
- AverageTemperature: **28.8 (Number)**
- Area: **2443.61 (Number)**



The item has been saved successfully.

DynamoDB > Explore items > PhilippinesCities

PhilippinesCities

Scan or query items

Select a table or index: Table - PhilippinesCities

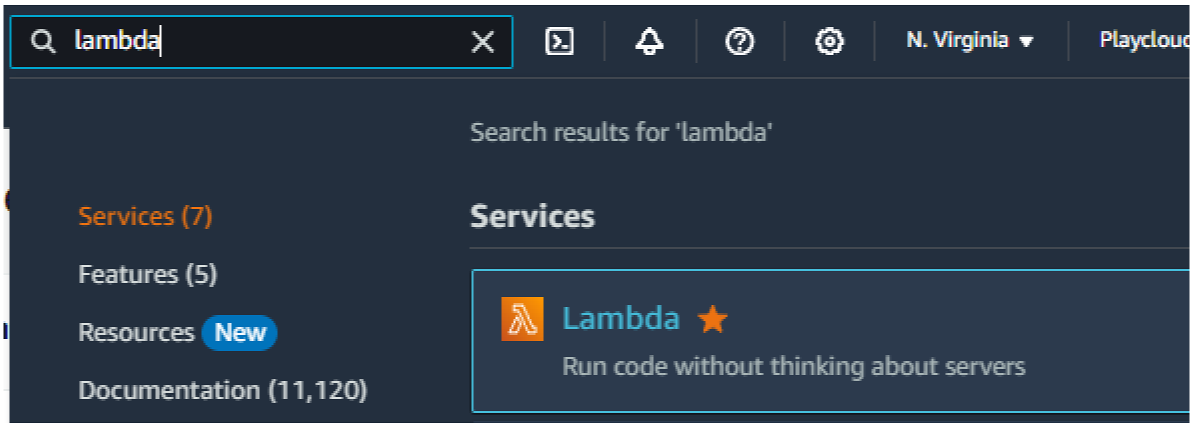
Select attribute projection: All attributes

Items returned (3)

	CityID (String)	Date (String)	Area	AverageTemperature	CityName	Population
☐	003	2024-08-20	2443.61	28.8	Davao City	1843000
☐	002	2024-08-20	315	29.5	Cebu City	922611
☐	001	2024-08-20	42.88	30.2	Manila	1780148

Create the Lambda Function

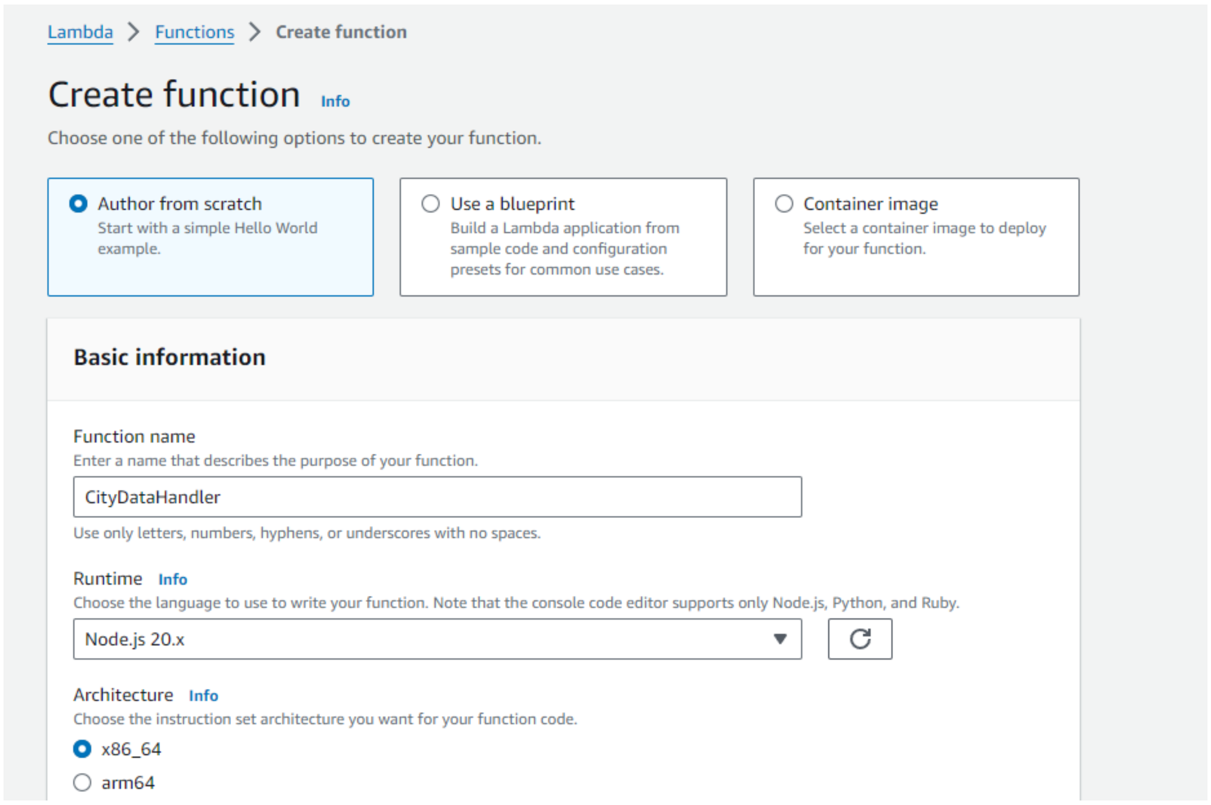
1. Navigate to the Lambda service in the AWS Management Console.



2. Create Function using the following confgurations:

- Choose Author from scratch.
- Function name: CityDataHandler

- Select Node.js 20.x as the runtime.


Lambda > Functions > Create function

Create function Info

Choose one of the following options to create your function.

- [✓ 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.

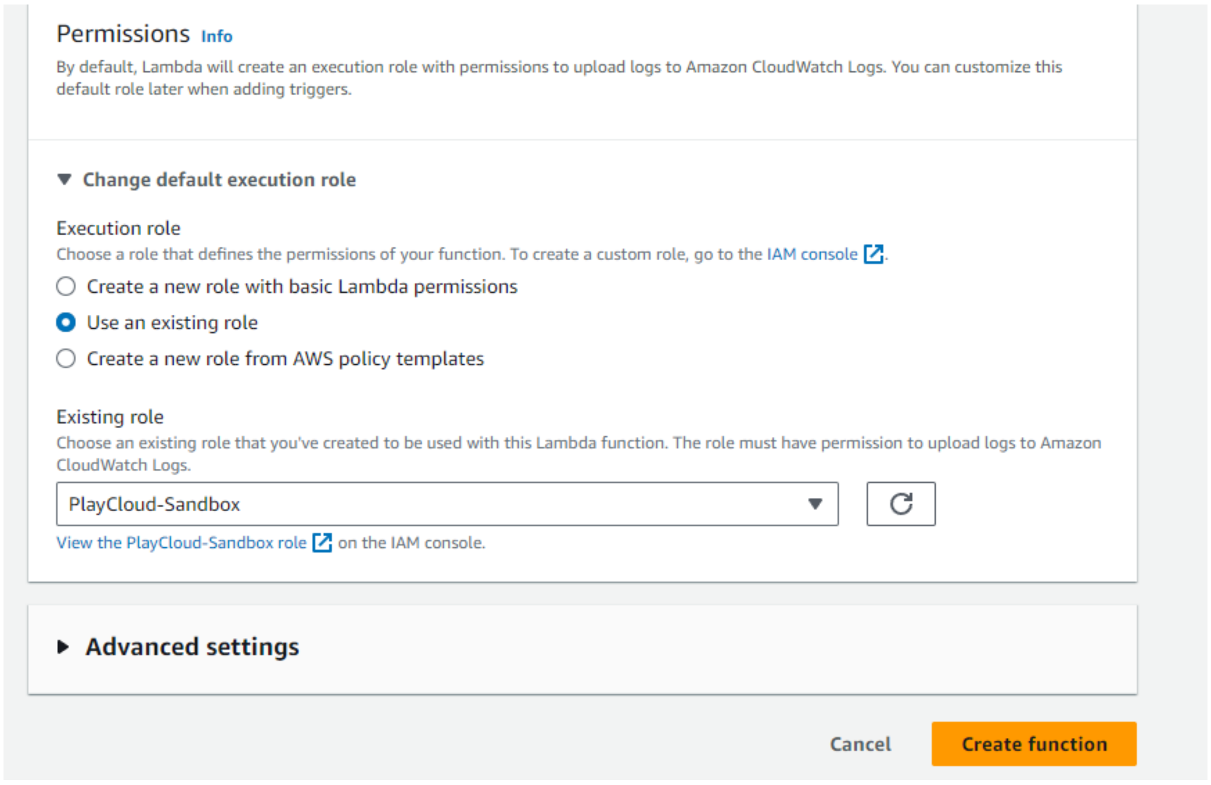
Basic information

Function name
Enter a name that describes the purpose of your function.
CityDataHandler
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 20.x

Architecture Info
Choose the instruction set architecture you want for your function code.
[✓ x86_64]
arm64

- **Execution role:**
    - Select Use an Existing Role: PlayCloud-Sanbox


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

Existing role
Choose an existing role that you've created to be used with this Lambda function. The role must have permission to upload logs to Amazon CloudWatch Logs.
PlayCloud-Sandbox
View the PlayCloud-Sandbox role on the IAM console.

▶ Advanced settings

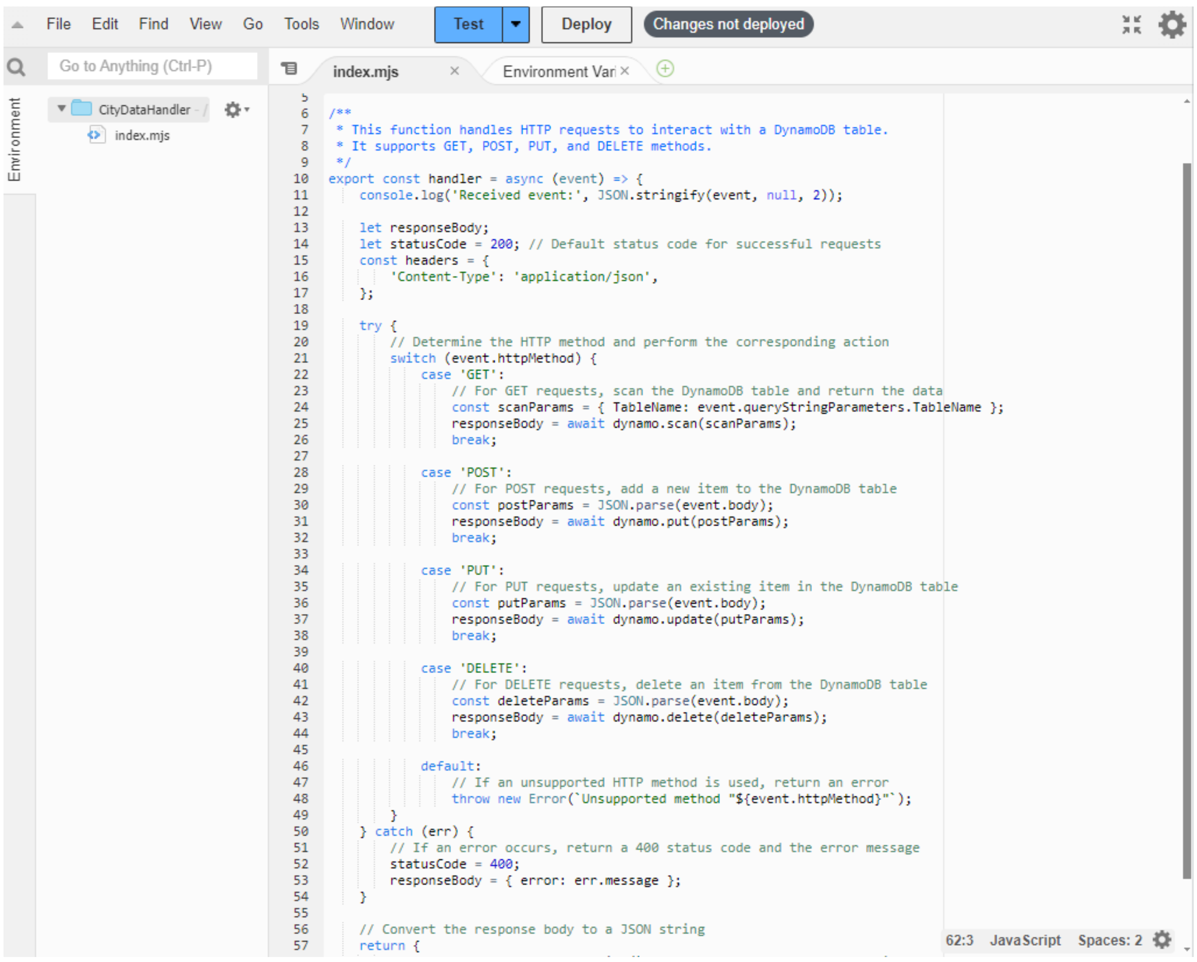
- Click on **Create Function**

***Note**: We are using the **old console editor** for this lab. You can switch to the **new editor** as you desire; the process remains the same, but the interface may look slightly different.*

3. Paste the following code in the code editor

```
import { DynamoDB } from '@aws-sdk/client-dynamodb';
import { DynamoDBDocument } from '@aws-sdk/lib-dynamodb';

const dynamo = DynamoDBDocument.from(new DynamoDB());

/**
 * This function handles HTTP requests to interact with a DynamoDB table.
 * It supports GET, POST, PUT, and DELETE methods.
 */
export const handler = async (event) => {
  console.log('Received event:', JSON.stringify(event, null, 2));

  let responseBody;
  let statusCode = 200; // Default status code for successful requests
  const headers = {
    'Content-Type': 'application/json',
  };

  try {
    // Determine the HTTP method and perform the corresponding action
    switch (event.httpMethod) {
      case 'GET':
        // For GET requests, scan the DynamoDB table and return the data
        const scanParams = { TableName: event.queryStringParameters.TableName };
        responseBody = await dynamo.scan(scanParams);
        break;

      case 'POST':
          // For POST requests, add a new item to the DynamoDB table
          const postParams = JSON.parse(event.body);
          responseBody = await dynamo.put(postParams);
          break;

        case 'PUT':
          // For PUT requests, update an existing item in the DynamoDB table
          const putParams = JSON.parse(event.body);
          responseBody = await dynamo.update(putParams);
          break;

        case 'DELETE':
          // For DELETE requests, delete an item from the DynamoDB table
          const deleteParams = JSON.parse(event.body);
          responseBody = await dynamo.delete(deleteParams);
          break;

        default:
          // If an unsupported HTTP method is used, return an error
          throw new Error(`Unsupported method "${event.httpMethod}"`);
    }
  } catch (err) {
    // If an error occurs, return a 400 status code and the error message
    statusCode = 400;
    responseBody = { error: err.message };
  }

  // Convert the response body to a JSON string
  return {
    statusCode: statusCode.toString(), // Convert status code to string
    body: JSON.stringify(responseBody),
    headers,
  };
};
```

4. Take your time to review the code:



```
/**
 * This function handles HTTP requests to interact with a DynamoDB table.
 * It supports GET, POST, PUT, and DELETE methods.
 */
export const handler = async (event) => {
    console.log('Received event:', JSON.stringify(event, null, 2));

    let responseBody;
    let statusCode = 200; // Default status code for successful requests
    const headers = {
        'Content-Type': 'application/json',
    };

    try {
        // Determine the HTTP method and perform the corresponding action
        switch (event.httpMethod) {
            case 'GET':
                // For GET requests, scan the DynamoDB table and return the data
                const scanParams = { TableName: event.queryStringParameters.TableName };
                responseBody = await dynamo.scan(scanParams);
                break;

            case 'POST':
                // For POST requests, add a new item to the DynamoDB table
                const postParams = JSON.parse(event.body);
                responseBody = await dynamo.put(postParams);
                break;

            case 'PUT':
                // For PUT requests, update an existing item in the DynamoDB table
                const putParams = JSON.parse(event.body);
                responseBody = await dynamo.update(putParams);
                break;

            case 'DELETE':
                // For DELETE requests, delete an item from the DynamoDB table
                const deleteParams = JSON.parse(event.body);
                responseBody = await dynamo.delete(deleteParams);
                break;

            default:
                // If an unsupported HTTP method is used, return an error
                throw new Error(`Unsupported method "${event.httpMethod}"`);
        }
    } catch (err) {
        // If an error occurs, return a 400 status code and the error message
        statusCode = 400;
        responseBody = { error: err.message };
    }

    // Convert the response body to a JSON string
    return {
```

Code Breakdown:

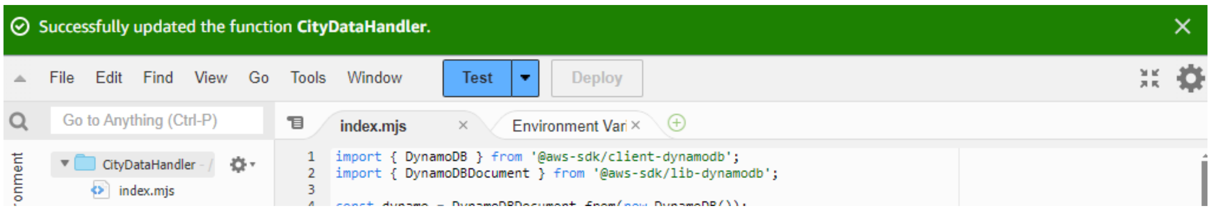
- **Imports:**
    - DynamoDB and DynamoDBDocument are imported from the AWS SDK. These are used to interact with the DynamoDB service. DynamoDBDocument provides a more user-friendly interface for working with DynamoDB operations.
- **Initialization:**
    - The dynamo object is created by wrapping a DynamoDB client with DynamoDBDocument, making it easier to interact with DynamoDB using native JavaScript objects.
- **Handler Function:**

    - The handler function is the main entry point for the Lambda function. It processes incoming HTTP requests and performs the corresponding action on the DynamoDB table based on the HTTP method provided in the request.

- **Request Handling:**
    - The function logs the incoming event for debugging purposes.
    - It initializes responseBody for storing the response and sets the default status code to 200.
    - Based on the httpMethod in the incoming event, the function performs different operations:
        - **GET:** Scans the entire DynamoDB table and returns all items.**POST:** Adds a new item to the table by parsing the request body and using the put method.
        - **PUT:** Updates an existing item in the table using the update method.
        - **DELETE:** Deletes an item from the table using the delete method.
    - If an unsupported HTTP method is used, an error is thrown.
- **Error Handling:**
    - If any error occurs during the operation, the function catches the error, sets the status code to 400, and includes the error message in the response.
- **Response:**
    - The function returns a JSON response containing the status code, response body, and headers.

---

5. Deploy the Lambda Function to save the code.

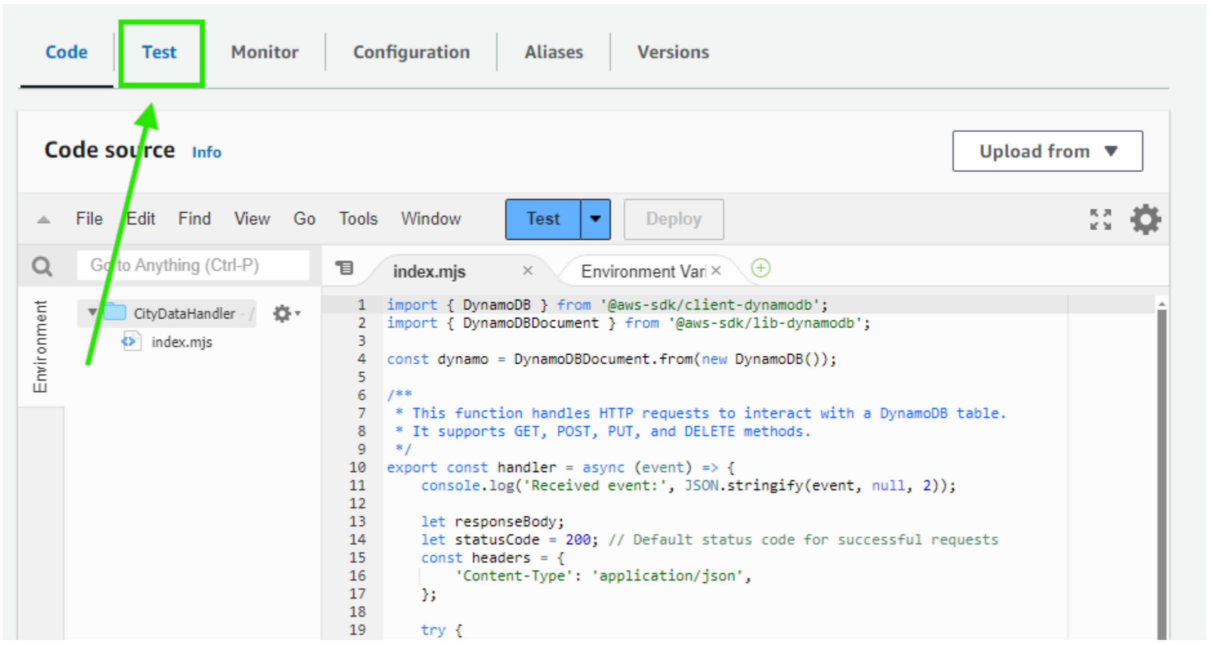


Successfully updated the function **CityDataHandler**.

```
import { DynamoDB } from '@aws-sdk/client-dynamodb';
import { DynamoDBDocument } from '@aws-sdk/lib-dynamodb';

```

Test the Lambda Function Using Test Events

1. Navigate to the Test tab of the AWS Lambda Dashboard.



```
import { DynamoDB } from '@aws-sdk/client-dynamodb';
import { DynamoDBDocument } from '@aws-sdk/lib-dynamodb';

const dynamo = DynamoDBDocument.from(new DynamoDB());

/**
 * This function handles HTTP requests to interact with a DynamoDB table.
 * It supports GET, POST, PUT, and DELETE methods.
 */
export const handler = async (event) => {
    console.log('Received event:', JSON.stringify(event, null, 2));

    let responseBody;
    let statusCode = 200; // Default status code for successful requests
    const headers = {
        'Content-Type': 'application/json',
    };

    try {
```

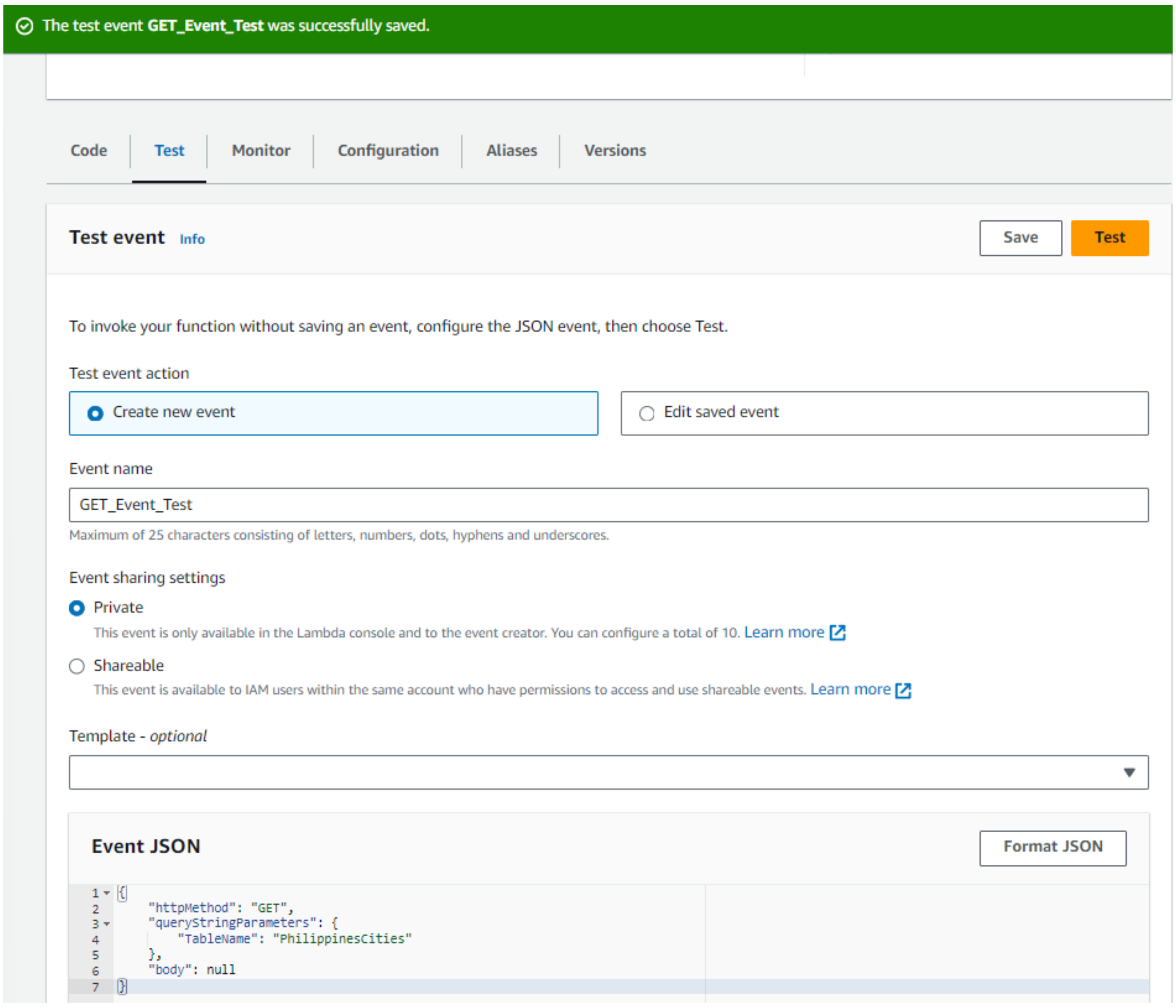
2. Once in the Test tab, click on **Creat new event:**

- Event name: GET_Event_Test
- In the Event JSON, paste the following:

```
{
  "httpMethod": "GET",
  "queryStringParameters": {
    "TableName": "PhilippinesCities"
  },
  "body": null
}
```

This test event triggers a scan of the entire ***PhilippinesCities*** *table and returns all items.*

- Click on **Save**


The test event **GET_Event_Test** was successfully saved.

Test event Info

To invoke your function without saving an event, configure the JSON event, then choose Test.

Test event action

[✓ Create new event] Edit saved event

Event name

GET_Event_Test

Maximum of 25 characters consisting of letters, numbers, dots, hyphens and underscores.

Event sharing settings

[✓ Private]

This event is only available in the Lambda console and to the event creator. You can configure a total of 10. Learn more

Shareable

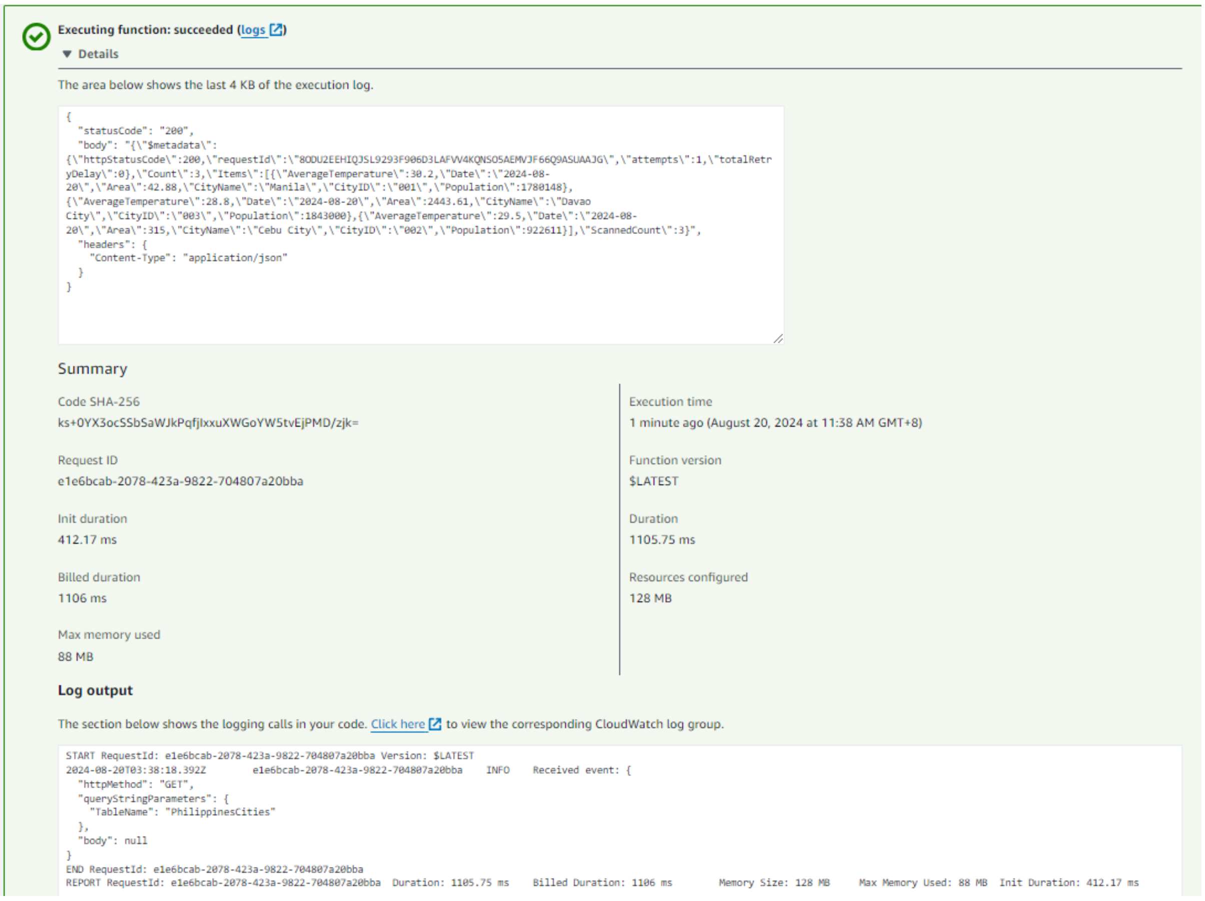
This event is available to IAM users within the same account who have permissions to access and use shareable events. Learn more

Template - *optional*

Event JSON

```
{
    "httpMethod": "GET",
    "queryStringParameters": {
        "TableName": "PhilippinesCities"
    },
    "body": null
}
```

- Then, click on **Test.** Wait for the test event to become successful. Lastly, expand the Details of the Test.


Executing function: succeeded (logs)

▼ Details

The area below shows the last 4 KB of the execution log.

```
{
  "statusCode": "200",
  "body": "{\"$metadata\":
{\"httpStatusCode\":200,\"requestId\":\"8ODU2EEHIQJSL9293F906D3LAFVV4KQNSO5AEMVJF66Q9ASUAAJG\",\"attempts\":1,\"totalRetr
yDelay\":0},\"Count\":3,\"Items\":[{\"AverageTemperature\":30.2,\"Date\":\"2024-08-
20\",\"Area\":42.88,\"CityName\":\"Manila\",\"CityID\":\"001\",\"Population\":1780148},
{\"AverageTemperature\":28.8,\"Date\":\"2024-08-20\",\"Area\":2443.61,\"CityName\":\"Davao
City\",\"CityID\":\"003\",\"Population\":1843000},{\"AverageTemperature\":29.5,\"Date\":\"2024-08-
20\",\"Area\":315,\"CityName\":\"Cebu City\",\"CityID\":\"002\",\"Population\":922611}],\"ScannedCount\":3}",
  "headers": {
    "Content-Type": "application/json"
  }
}
```

Summary

Code SHA-256 ks+0YX3ocSSbSaWJkPqfjlxxuXWGoYW5tvEjPMD/zjk=	Execution time 1 minute ago (August 20, 2024 at 11:38 AM GMT+8)
Request ID e1e6bcab-2078-423a-9822-704807a20bba	Function version $LATEST
Init duration 412.17 ms	Duration 1105.75 ms
Billed duration 1106 ms	Resources configured 128 MB
Max memory used 88 MB	

Log output

The section below shows the logging calls in your code. Click here to view the corresponding CloudWatch log group.

```
START RequestId: e1e6bcab-2078-423a-9822-704807a20bba Version: $LATEST
2024-08-20T03:38:18.392Z	e1e6bcab-2078-423a-9822-704807a20bba	INFO	Received event: {
  "httpMethod": "GET",
  "queryStringParameters": {
    "TableName": "PhilippinesCities"
  },
  "body": null
}
END RequestId: e1e6bcab-2078-423a-9822-704807a20bba
REPORT RequestId: e1e6bcab-2078-423a-9822-704807a20bba  Duration: 1105.75 ms	Billed Duration: 1106 ms	Memory Size: 128 MB	Max Memory Used: 88 MB  Init Duration: 412.17 ms
```

- Take your time to review the result of the test. It should return the Table items of the **PhilippinesCities** Dynamo DB table

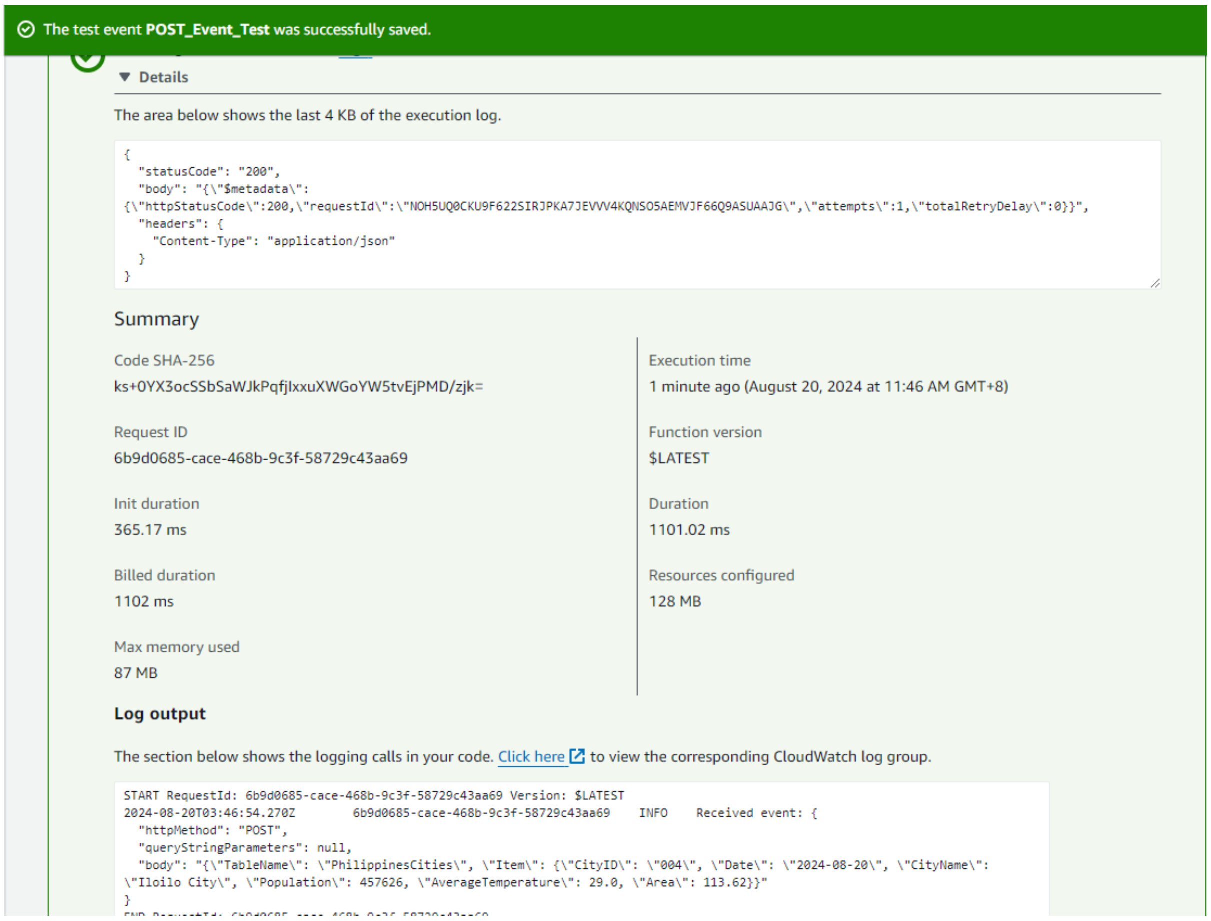
3. Now lets create another Test Event using the following details:

- Event name: POST_Event_Test
- In the Event JSON, paste the following:

```
{
  "httpMethod": "POST",
  "queryStringParameters": null,
  "body": "{\"TableName\": \"PhilippinesCities\", \"Item\": {\"CityID\": \"004\", \"Date\": \"2024-08-20\", \"CityName\": \"Iloilo City\", \"Population\": 457626, \"AverageTemperature\": 29.0, \"Area\": 113.62}}"
}
```

*This test event adds a new item (Iloilo City) to the **PhilippinesCities** table.*

- Click on **Save**
- Click on **Test**



The test event **POST_Event_Test** was successfully saved.

▼ Details

The area below shows the last 4 KB of the execution log.

```
{
  "statusCode": "200",
  "body": "{\"$metadata\":
{\"httpStatusCode\":200,\"requestId\":\"NOH5UQ0CKU9F622SIRJPKA7JEVVV4KQNSO5AEMVJF66Q9ASUAAJG\",\"attempts\":1,\"totalRetryDelay\":0}}",
  "headers": {
    "Content-Type": "application/json"
  }
}
```

Summary

Code SHA-256
ks+0YX3ocSSbSaWJkPqfjIxxuXWGoYW5tvEjPMD/zjk=

Request ID
6b9d0685-cace-468b-9c3f-58729c43aa69

Init duration
365.17 ms

Billed duration
1102 ms

Max memory used
87 MB

Execution time
1 minute ago (August 20, 2024 at 11:46 AM GMT+8)

Function version
$LATEST

Duration
1101.02 ms

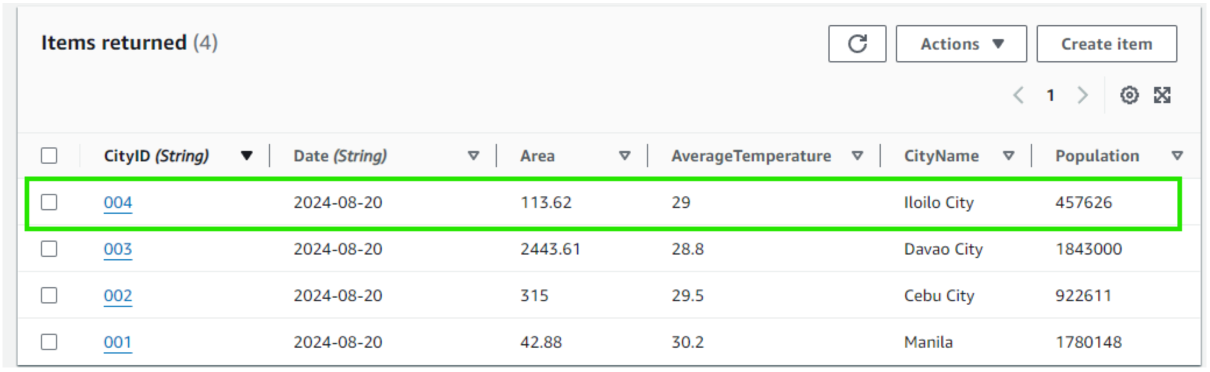
Resources configured
128 MB

Log output

The section below shows the logging calls in your code. Click here to view the corresponding CloudWatch log group.

```
START RequestId: 6b9d0685-cace-468b-9c3f-58729c43aa69 Version: $LATEST
2024-08-20T03:46:54.270Z	6b9d0685-cace-468b-9c3f-58729c43aa69	INFO	Received event: {
  "httpMethod": "POST",
  "queryStringParameters": null,
  "body": "{\"TableName\": \"PhilippinesCities\", \"Item\": {\"CityID\": \"004\", \"Date\": \"2024-08-20\", \"CityName\":
\"Iloilo City\", \"Population\": 457626, \"AverageTemperature\": 29.0, \"Area\": 113.62}}"
}
```

- Navigate back to the Dynamo DB Table **PhilippinesCities,** then on the on the explore items. The newly added item (Iloilo City) should be visible in the items table.



Items returned (4)

CityID (String)	Date (String)	Area	AverageTemperature	CityName	Population
004	2024-08-20	113.62	29	Iloilo City	457626
003	2024-08-20	2443.61	28.8	Davao City	1843000
002	2024-08-20	315	29.5	Cebu City	922611
001	2024-08-20	42.88	30.2	Manila	1780148

4. Take note of the Average Temperature of CityID **001** CityName **Manila.**

5. Navigate back to the Lambda Function **CityDataHandler.** Then, to the **Test tab.** Create this time the **PUT Request Test:**

- Event name: PUT_Event_Test
- In the Event JSON, paste the following:

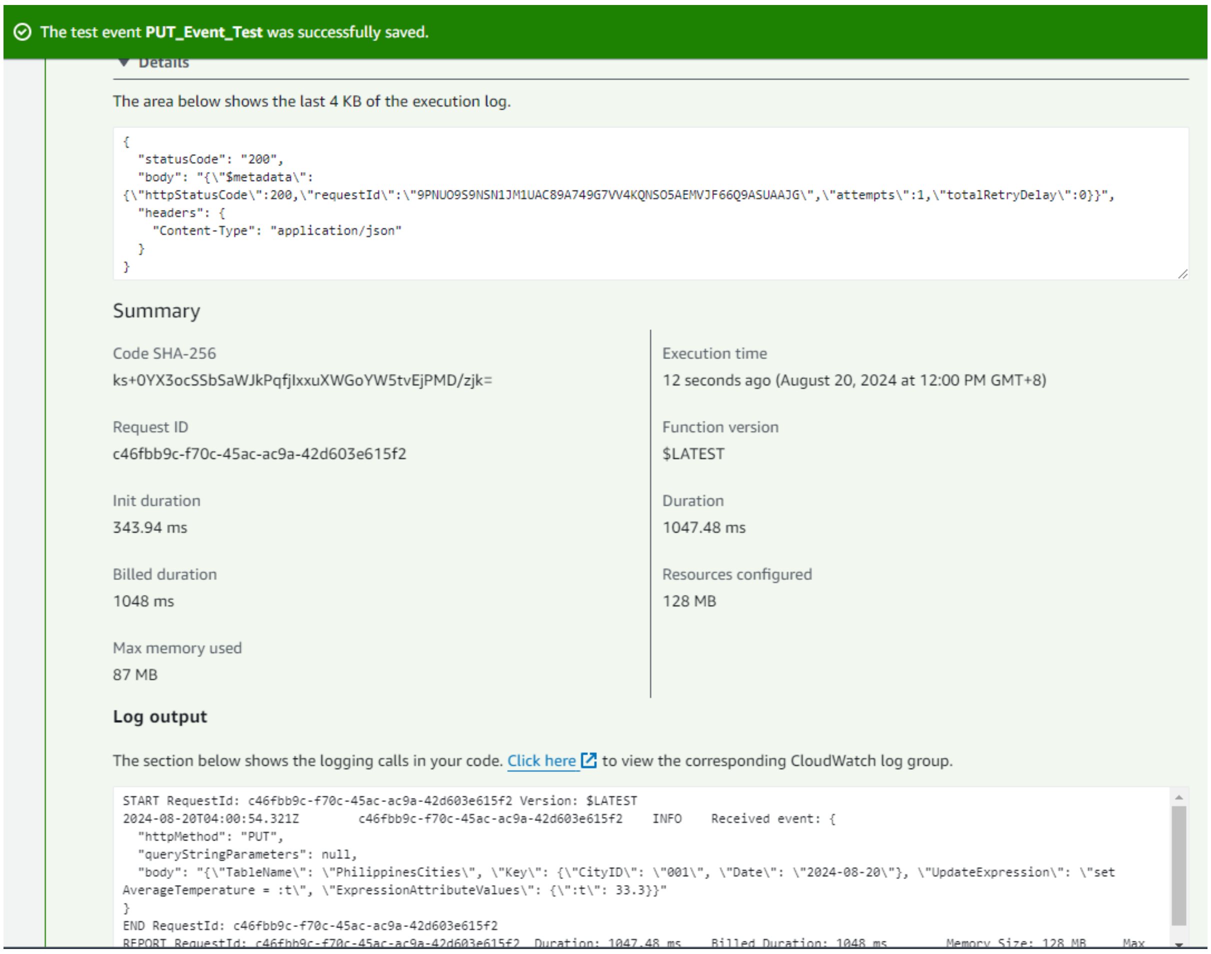
```
{
  "httpMethod": "PUT",
  "queryStringParameters": null,
```

"body": "{\"TableName\": \"PhilippinesCities\", \"Key\": {\"CityID\": \"001\", \"Date\": \"2024-08-20\"}, \"UpdateExpression\": \"set AverageTemperature = :t\", \"ExpressionAttributeValues\": {\":t\": 33.3}}"

}

*This test event updates the **AverageTemperature** of Manila in the **PhilippinesCities** table.*

- Click on **Save** and then **Test.**



The test event **PUT_Event_Test** was successfully saved.

▼ Details

The area below shows the last 4 KB of the execution log.

```
{
  "statusCode": "200",
  "body": "{\"$metadata\":
{\"httpStatusCode\":200,\"requestId\":\"9PNUO9S9NSN1JM1UAC89A749G7VV4KQNSO5AEMVJF66Q9ASUAAJG\",\"attempts\":1,\"totalRetryDelay\":0}}",
  "headers": {
    "Content-Type": "application/json"
  }
}
```

Summary

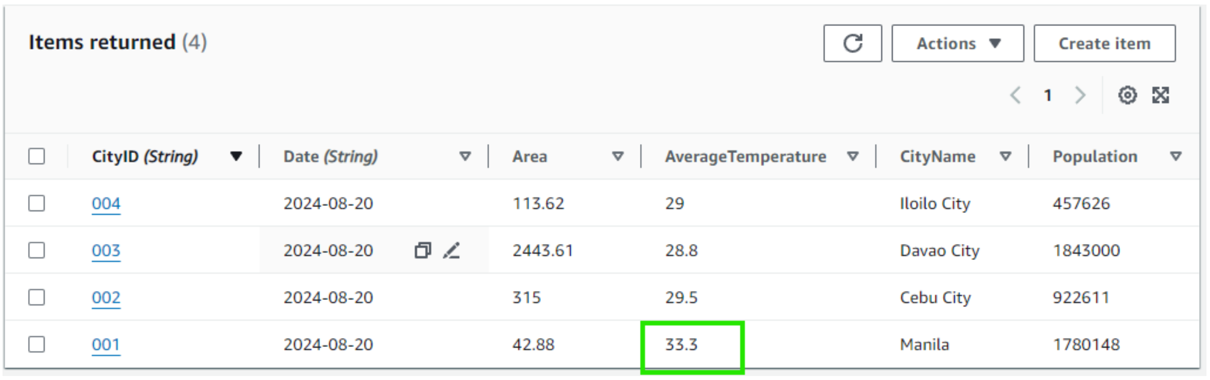
Code SHA-256	Execution time
ks+0YX3ocSSbSaWJkPqfjlxxuXWGoYW5tvEjPMD/zjk=	12 seconds ago (August 20, 2024 at 12:00 PM GMT+8)
Request ID	Function version
c46fbb9c-f70c-45ac-ac9a-42d603e615f2	$LATEST
Init duration	Duration
343.94 ms	1047.48 ms
Billed duration	Resources configured
1048 ms	128 MB
Max memory used	
87 MB	

Log output

The section below shows the logging calls in your code. Click here to view the corresponding CloudWatch log group.

```
START RequestId: c46fbb9c-f70c-45ac-ac9a-42d603e615f2 Version: $LATEST
2024-08-20T04:00:54.321Z	c46fbb9c-f70c-45ac-ac9a-42d603e615f2	INFO	Received event: {
  "httpMethod": "PUT",
  "queryStringParameters": null,
  "body": "{\"TableName\": \"PhilippinesCities\", \"Key\": {\"CityID\": \"001\", \"Date\": \"2024-08-20\"}, \"UpdateExpression\": \"set
AverageTemperature = :t\", \"ExpressionAttributeValues\": {\":t\": 33.3}}"
}
END RequestId: c46fbb9c-f70c-45ac-ac9a-42d603e615f2
REPORT RequestId: c46fbb9c-f70c-45ac-ac9a-42d603e615f2	Duration: 1047.48 ms	Billed Duration: 1048 ms	Memory Size: 128 MB	Max
```

- Navigate again to the Dynamo DB table and observe that the Average Temperature for Manila has changed from 30.2 to **33.3**



Items returned (4)

	CityID (String)	Date (String)	Area	AverageTemperature	CityName	Population
☐	004	2024-08-20	113.62	29	Iloilo City	457626
☐	003	2024-08-20	2443.61	28.8	Davao City	1843000
☐	002	2024-08-20	315	29.5	Cebu City	922611
☐	001	2024-08-20	42.88	33.3	Manila	1780148

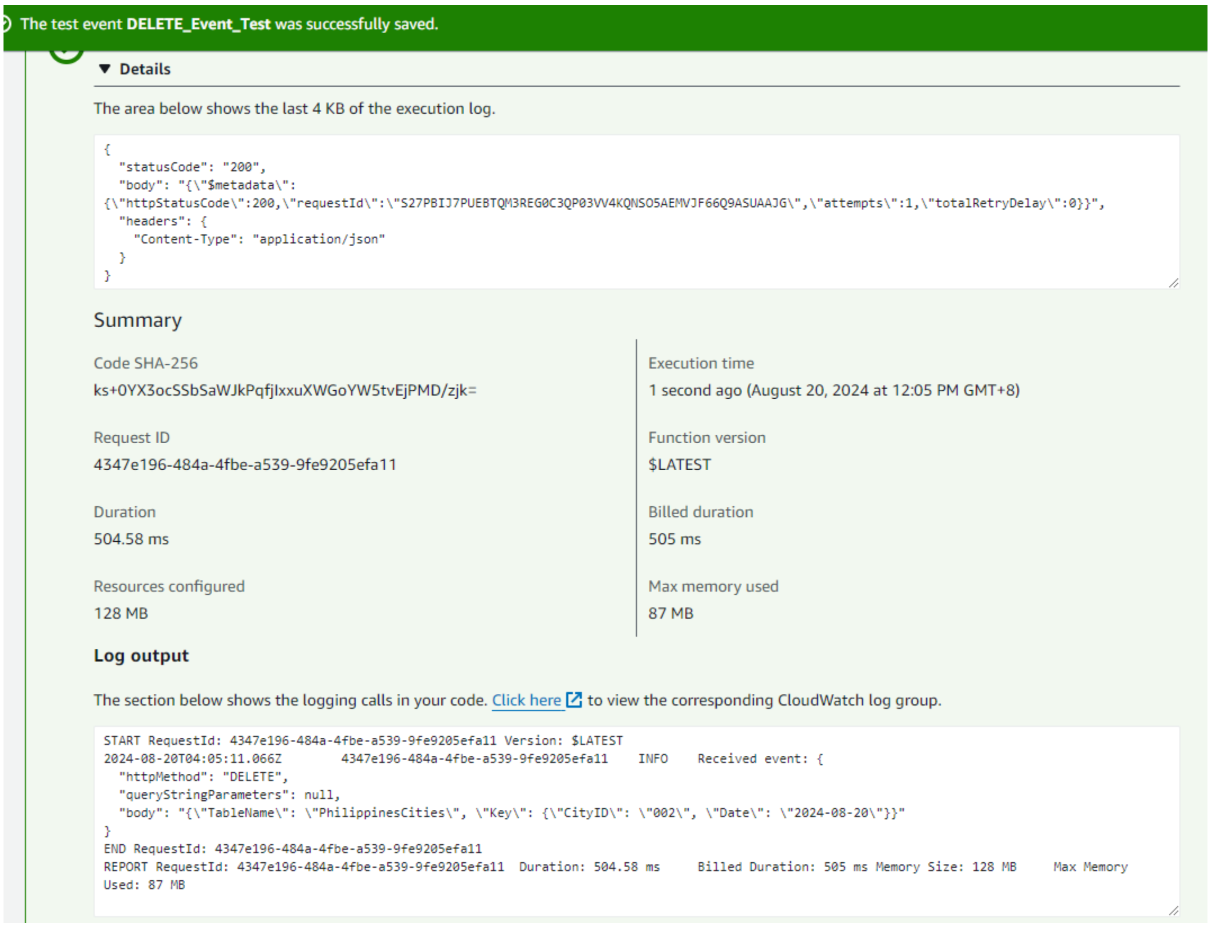
6. For the last test, navigate back to the Lambda Function again and create a new test event:

- Event name: DELETE_Event_Test
- In the Event JSON, paste the following:

```
{
   "httpMethod": "DELETE",
   "queryStringParameters": null,
   "body": "{\"TableName\": \"PhilippinesCities\", \"Key\": {\"CityID\": \"002\", \"Date\": \"2024-08-20\"}}"
}
```

*This test event deletes the item with CityID: "002" (Cebu City) from the **PhilippinesCities** table.*

- Click on **Save** and then **Test.**



The test event **DELETE_Event_Test** was successfully saved.

Details

The area below shows the last 4 KB of the execution log.

```
{
  "statusCode": "200",
  "body": "{\"$metadata\":
{\"httpStatusCode\":200,\"requestId\":\"S27PBIJ7PUEBTQM3REG0C3QP03VV4KQNSO5AEMVJF66Q9ASUAAJG\",\"attempts\":1,\"totalRetryDelay\":0}}",
  "headers": {
    "Content-Type": "application/json"
  }
}
```

Summary

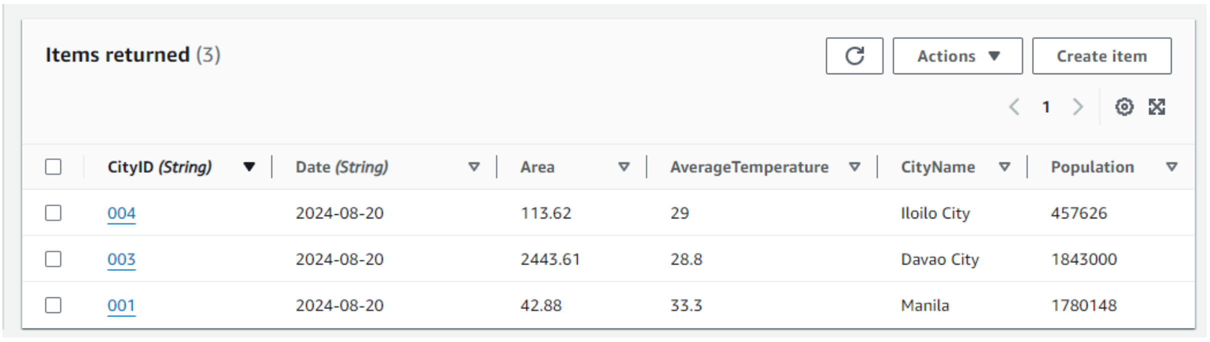
Code SHA-256 ks+0YX3ocSSbSaWJkPqfjIxxuXWGoYW5tvEjPMD/zjk=	Execution time 1 second ago (August 20, 2024 at 12:05 PM GMT+8)
Request ID 4347e196-484a-4fbe-a539-9fe9205efa11	Function version $LATEST
Duration 504.58 ms	Billed duration 505 ms
Resources configured 128 MB	Max memory used 87 MB

Log output

The section below shows the logging calls in your code. Click here to view the corresponding CloudWatch log group.

```
START RequestId: 4347e196-484a-4fbe-a539-9fe9205efa11 Version: $LATEST
2024-08-20T04:05:11.066Z        4347e196-484a-4fbe-a539-9fe9205efa11    INFO    Received event: {
  "httpMethod": "DELETE",
  "queryStringParameters": null,
  "body": "{\"TableName\": \"PhilippinesCities\", \"Key\": {\"CityID\": \"002\", \"Date\": \"2024-08-20\"}}"
}
END RequestId: 4347e196-484a-4fbe-a539-9fe9205efa11
REPORT RequestId: 4347e196-484a-4fbe-a539-9fe9205efa11  Duration: 504.58 ms     Billed Duration: 505 ms Memory Size: 128 MB     Max Memory
Used: 87 MB
```

- Navigate again to the Dynamo DB table and observe the item with CityID: "002" (Cebu City) from the **PhilippinesCities** table was deleted.



	CityID (String)	Date (String)	Area	AverageTemperature	CityName	Population
☐	004	2024-08-20	113.62	29	Iloilo City	457626
☐	003	2024-08-20	2443.61	28.8	Davao City	1843000
☐	001	2024-08-20	42.88	33.3	Manila	1780148

That's it! Congratulations! You've successfully created a serverless backend using AWS Lambda and DynamoDB. By integrating these powerful services, you've learned how to handle basic CRUD operations—GET, POST, PUT, and DELETE—via simulated HTTP requests using test events in Lambda. This lab is a fundamental building block for creating scalable and cost-effective serverless applications.

Throughout the lab, you configured a DynamoDB table to store city-related data and used Lambda functions to interact with the database. You also learned how to test your Lambda function using predefined test events, ensuring that your code performs as expected. Happy learning!