

## Guided Lab: Creating a NodeJS Function in AWS Lambda

### Description

**AWS Lambda** is a serverless computing service provided by Amazon Web Services (AWS) that allows you to run code without having to manage or provision servers. With AWS Lambda, you can execute code in response to events, such as changes to data in an Amazon S3 bucket or an update to a DynamoDB table, without the need for server management. AWS Lambda automatically scales your application by running code in response to each trigger and only charges you for the compute time you consume.

**Node.js** is a runtime environment that allows you to execute JavaScript code on the server side, outside of a web browser. It is built on the V8 JavaScript engine, which is used in Google Chrome, and it provides an event-driven, non-blocking I/O model, making it lightweight and efficient for building scalable network applications.

In this lab, you will learn how to create, test, and modify a NodeJS function in AWS Lambda using a pre-built template. AWS Lambda is a serverless compute service that lets you run code without provisioning or managing servers. This lab focuses on using the “Hello world function” blueprint, which provides a basic starting point for NodeJS functions.

### Prerequisites

This lab assumes you have basic understanding of AWS Lambda and NodeJS programming language.

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

- Creating an AWS Lambda function

### Objectives

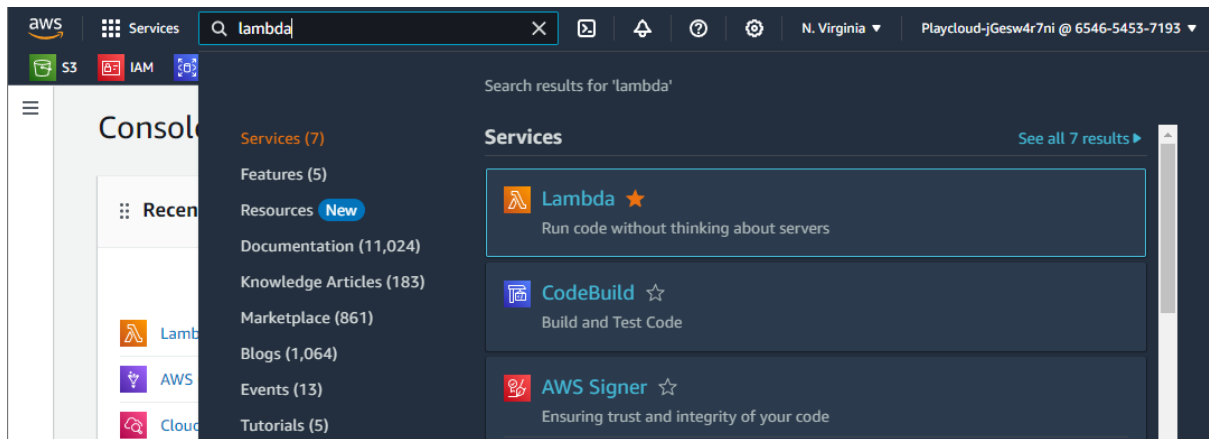
By the end of this lab, you will be able to:

- Create a serverless function using AWS Lambda and a pre-built NodeJS template.
- Configure and run test events with custom data inputs.
- Understand how to modify and deploy code changes in AWS Lambda.

### Lab Steps

#### CREATE AN AWS LAMBDA FUNCTION

1. Navigate to AWS Lambda Console



2. Create Function using the following configurations:

- Select **Use a blueprint**
- Blueprint name : **Hello world function nodejs18.x**
- Function name: **TamarawLambda**
- **Execution role:**
  - Select Use an Existing Role: **PlayCloud-Sandbox**

## 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 [Info](#)

Blueprint name

Hello world function

A starter AWS Lambda function.

nodejs18.x ▼

Function name

Enter a name that describes the purpose of your function.

TamarawLambda

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

Runtime

nodejs18.x

Architecture

x86\_64

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.

- Click on **Create Function**

### Invoke the Lambda Function Using the Test Button

1. Once your function is created, you'll be directed to the function's dashboard

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

The screenshot shows the AWS Lambda console interface. At the top, there's a 'Function overview' section with tabs for 'Diagram' and 'Template'. Below these, there's a visual representation of the function with the AWS Lambda logo and the name 'TamarawLambda'. To the right, there's a 'Description' section with details: 'A starter AWS Lambda function.', 'Last modified 35 minutes ago', 'Function ARN: arn:aws:lambda:us-east-1:533267:157116:function:TamarawLambda', and 'Function URL: -'. Below the overview, there's a 'Code source' section with tabs for 'Code', 'Test', 'Monitor', 'Configuration', 'Aliases', and 'Versions'. The 'Code source' tab is active, showing a code editor with a file named 'index.mjs'. The code in the editor is as follows:

```
1 console.log('Loading function');
2
3 export const handler = async (event, context) => {
4   //console.log('Received event:', JSON.stringify(event, null, 2));
5   console.log('value1 =', event.key1);
6   console.log('value2 =', event.key2);
7   console.log('value3 =', event.key3);
8   return event.key1; // Echo back the first key value
9   // throw new Error('Something went wrong');
10 };
11
```

2. Take your time to review the code.

```
1 console.log('Loading function');
2
3 export const handler = async (event, context) => {
4   //console.log('Received event:', JSON.stringify(event, null, 2));
5   console.log('value1 =', event.key1);
6   console.log('value2 =', event.key2);
7   console.log('value3 =', event.key3);
8   return event.key1; // Echo back the first key value
9   // throw new Error('Something went wrong');
10 };
11
```

- **console.log('Loading function');** – This logs the message “Loading function” to indicate that the function has started executing.
- **export const handler = async (event, context) => { ... }** – This defines the main function (called the handler) that AWS Lambda will run when triggered. It uses async syntax, which means it can handle asynchronous operations.

- **console.log('value1 =', event.key1);** – This logs the value of key1 from the event data passed to the function.
  - **console.log('value2 =', event.key2);** – This logs the value of key2 from the event data.
  - **console.log('value3 =', event.key3);** – This logs the value of key3 from the event data.
  - **return event.key1;** – This returns the value of key1 from the event, effectively “echoing” it back as the function’s result
- 

3. Click the blue **Test** button at the top-right of the page.

- A Configure test event dialog will pop up, give your test event a name (e.g. test)
- Then, for this initial test, leave the rest as default, save it and...
- Click on **Test** again.

Configure test event

A test event is a JSON object that mocks the structure of requests emitted by AWS services to invoke a Lambda function. Use it to see the function's invocation result.

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

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

hello-world

Event JSON

Format JSON

1 {

2 "key1": "value1",

3 "key2": "value2",

4 "key3": "value3"

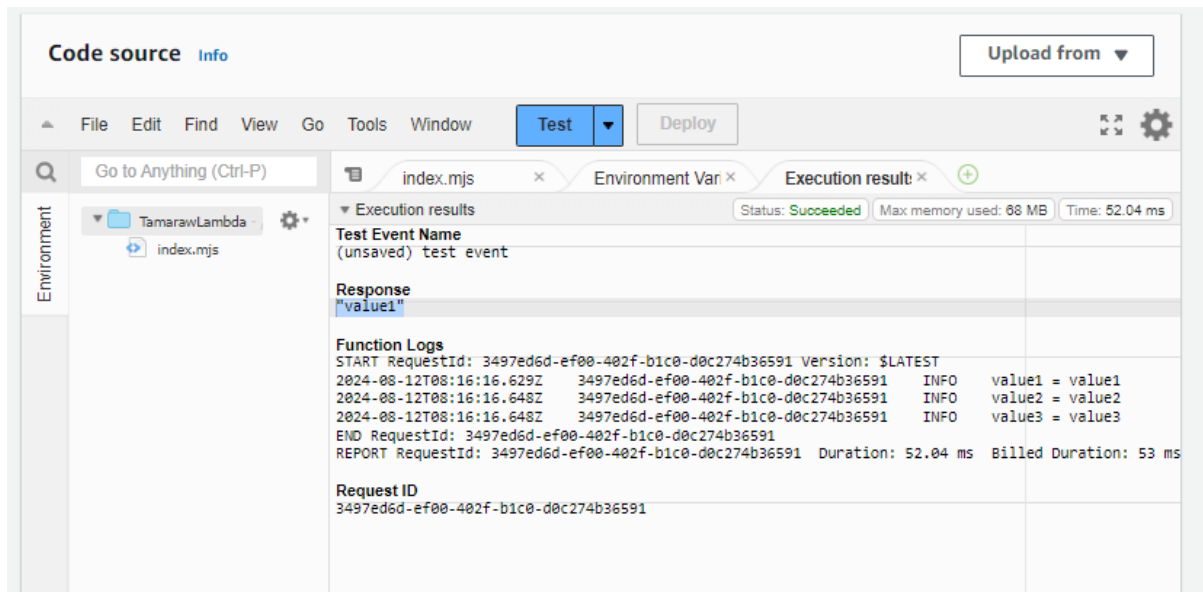
5 }

Cancel

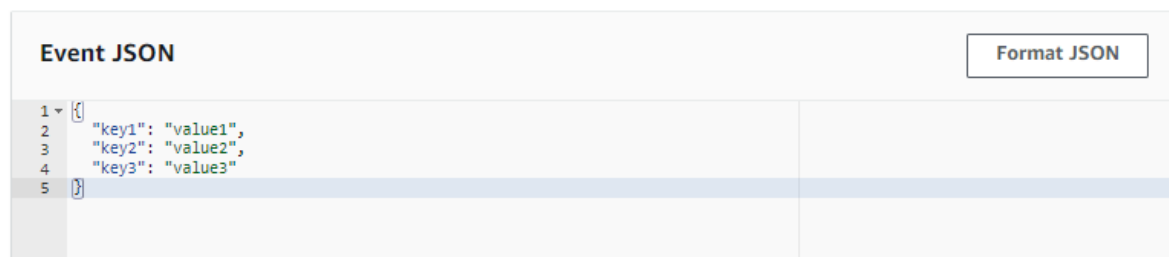
Invoke

Save

4. Review the response:



Notice the response? This is because the default Test Event we triggered is this Event JSON:



An **event** is the input that your Lambda function processes. You can create and save up to 10 test events per function. Saved events are stored in Lambda, meaning they'll be available even if you switch browsers or devices. Unsaved events will only be available during your current session and will be lost if the session ends.

Running a test event in the console triggers your function synchronously. Lambda takes the event (in JSON format), converts it into an object, and passes it to your function's handler method for processing.

5. To modify this Event JSON, navigate to the **Test Tab** beside **Code tab**

**Code** **Test** Monitor Configuration Aliases Versions

**Test event** [Info](#) Save Test

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**

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** Format JSON

```
1 {
2   "key1": "value1",
3   "key2": "value2",
4   "key3": "value3"
5 }
```

**6. Enter TouristDestinationsTest for the Event name**

**Test event** [Info](#) Save Test

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**

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

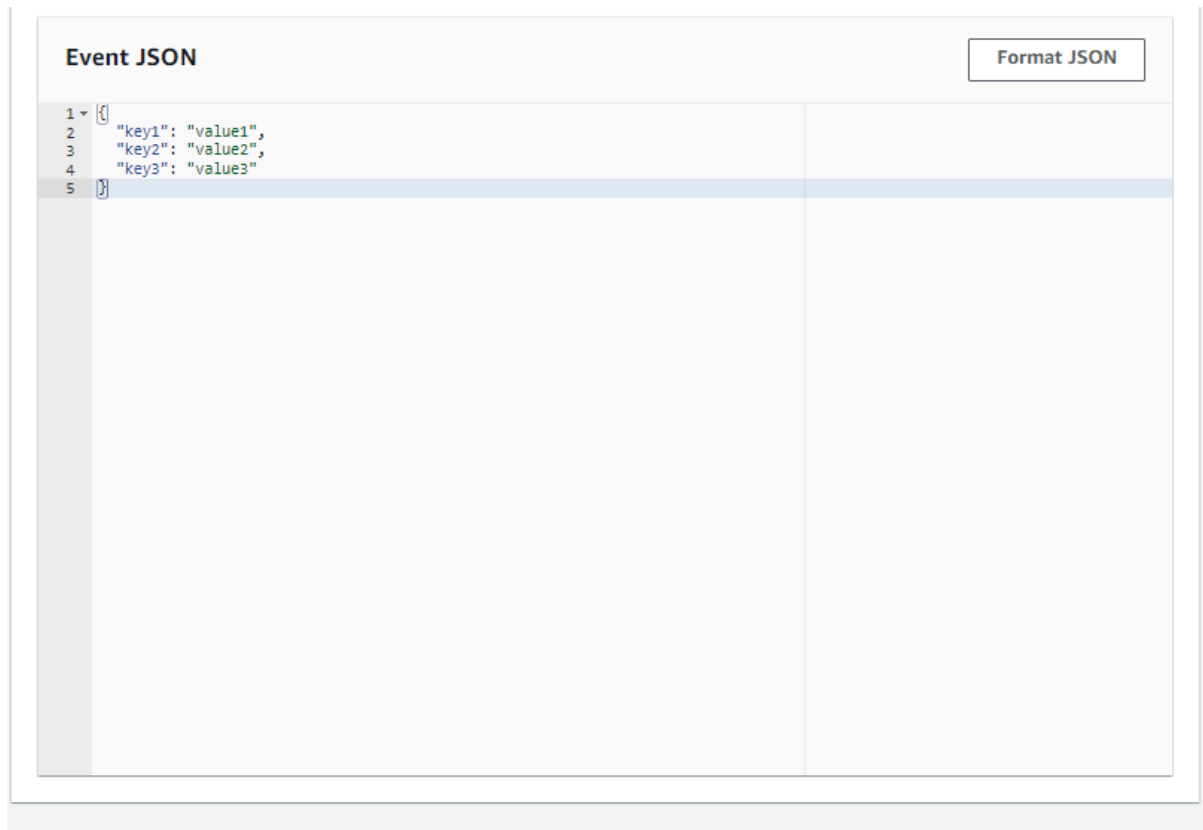
**Event sharing settings**

☒ 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](#)

**7. Scroll down to the Event JSON**





8. Replace the default values with the following:

```
{  
  "key1": "Boracay",  
  "key2": "Palawan",  
  "key3": "Manila"  
}
```

Event JSON

Format JSON

1

{

2

"key1": "Boracay",

3

"key2": "Palawan",

4

"key3": "Manila"

5

}

Cancel

Invoke

Save

- Click on Save

9. Now, click on Test again. Observe the output. The function should return "Boracay" since the original code returns event.key1.

index.mjs

Environment Vari

Execution result

Execution results

Status: Succeeded

Max memory used: 67 MB

Time: 16.09 ms

Test Event Name

TouristDestinationsTest

Response

"Boracay"

Function Logs

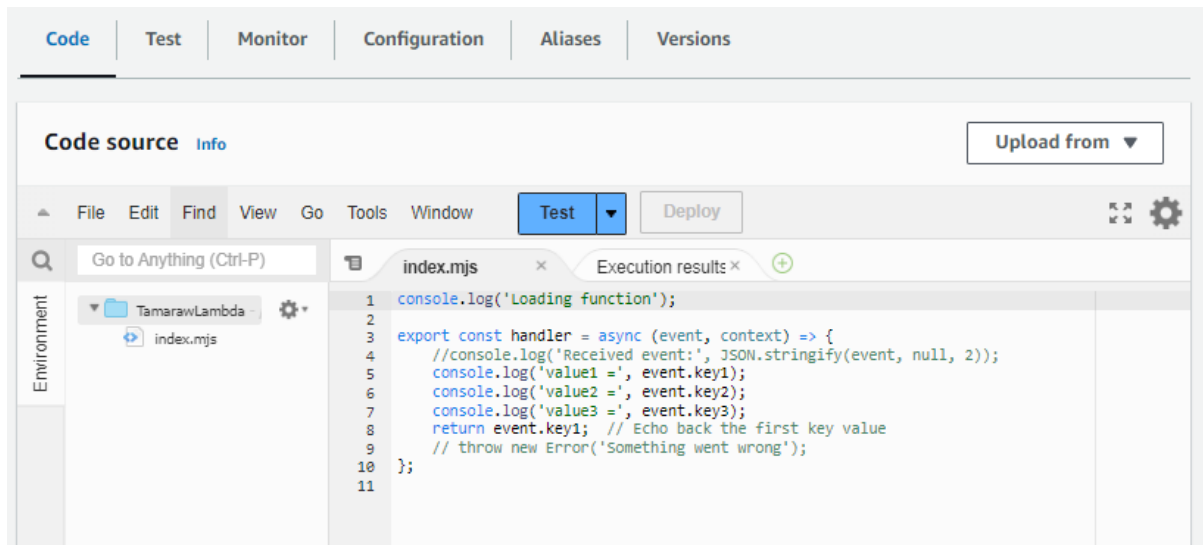
2024-08-12T08:36:19.745Z undefined INFO Loading function  
START RequestId: 7c20ae6f-a1e1-4ebb-811a-9db38505e4b1 Version: \$LATEST  
2024-08-12T08:36:19.751Z 7c20ae6f-a1e1-4ebb-811a-9db38505e4b1 INFO value1 = Boracay  
2024-08-12T08:36:19.751Z 7c20ae6f-a1e1-4ebb-811a-9db38505e4b1 INFO value2 = Palawan  
2024-08-12T08:36:19.763Z 7c20ae6f-a1e1-4ebb-811a-9db38505e4b1 INFO value3 = Manila  
END RequestId: 7c20ae6f-a1e1-4ebb-811a-9db38505e4b1  
REPORT RequestId: 7c20ae6f-a1e1-4ebb-811a-9db38505e4b1 Duration: 16.09 ms Billed Duration: 17 ms

Request ID

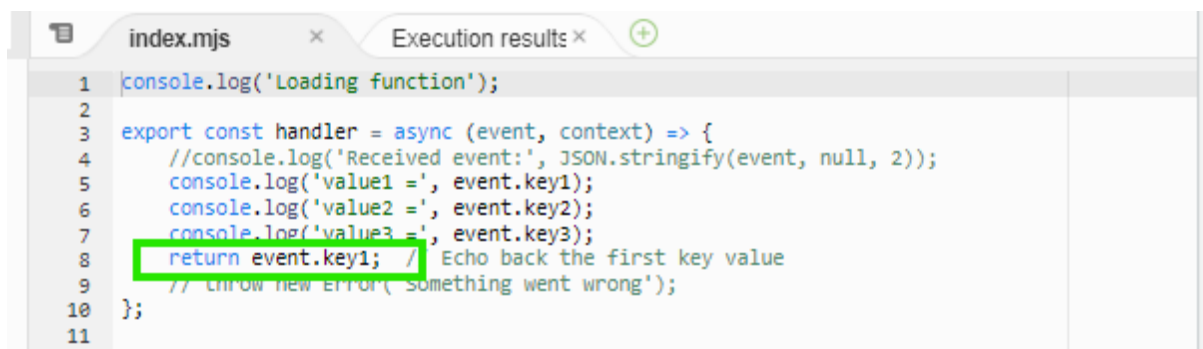
7c20ae6f-a1e1-4ebb-811a-9db38505e4b1

## Modify the Function Code

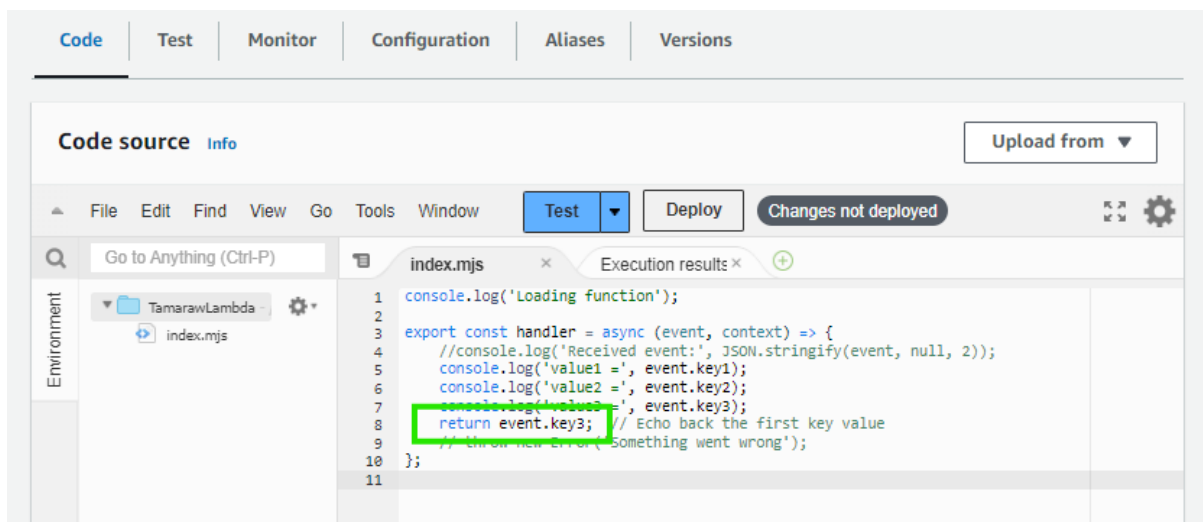
- Navigate to the **Code tab** or **Code source** section.



2. Locate **Line 8** in the code, which currently reads:

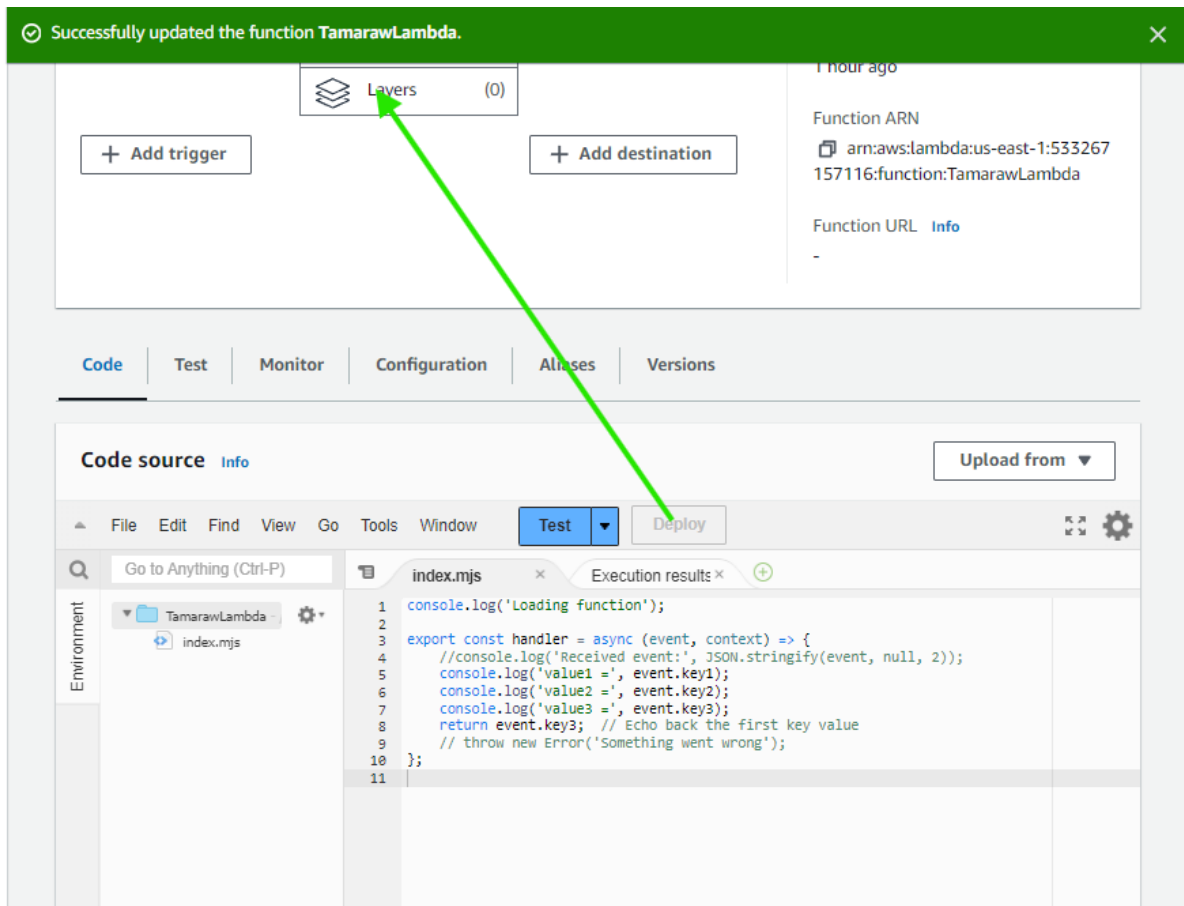


3. Modify this line to return a different key (e.g., event.key3)

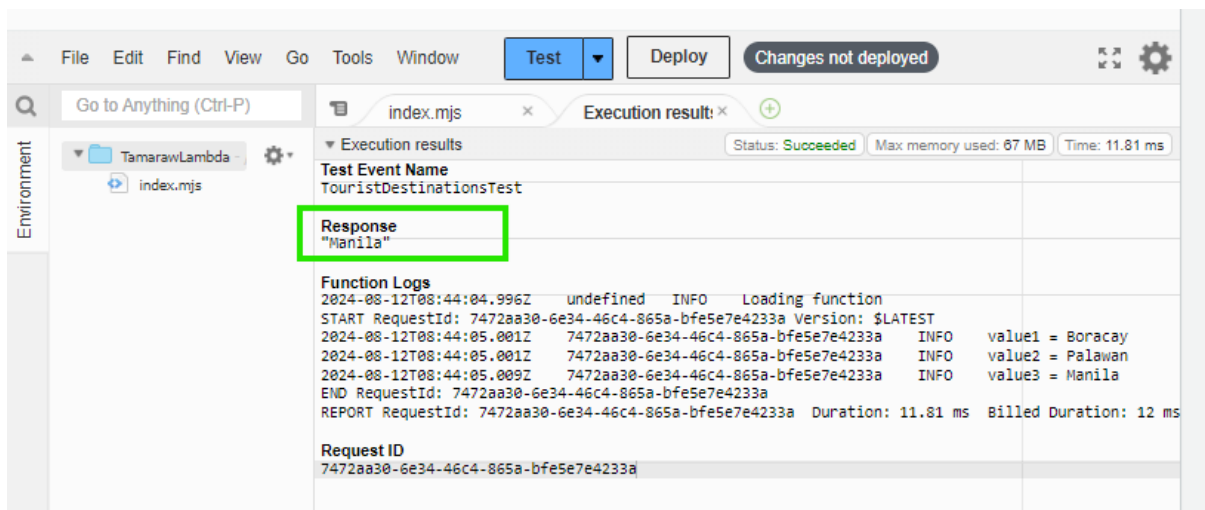


4. Click on **Deploy**

*When testing your Lambda function, it's important to remember that any code changes must be deployed before you can test and see the changes in action.*



5. Click on Test again. Now, the function should return "Manila" as the output, reflecting the change made to event.key3.



That's It! congratulations! You've successfully created a NodeJS function in AWS Lambda using a blueprint, configured test events with Filipino tourist destinations, and modified the function's code to demonstrate the impact of code changes after deployment.

This lab serves as an introductory guide to using NodeJS in AWS Lambda, providing you with the foundational skills needed to build and deploy serverless functions. As you progress, you can explore more advanced features and capabilities of AWS Lambda to further enhance your serverless applications. Happy learning!