

**NAME: Bhushan Malpani**

**DIV:D15C**

**ROLL NO:33**

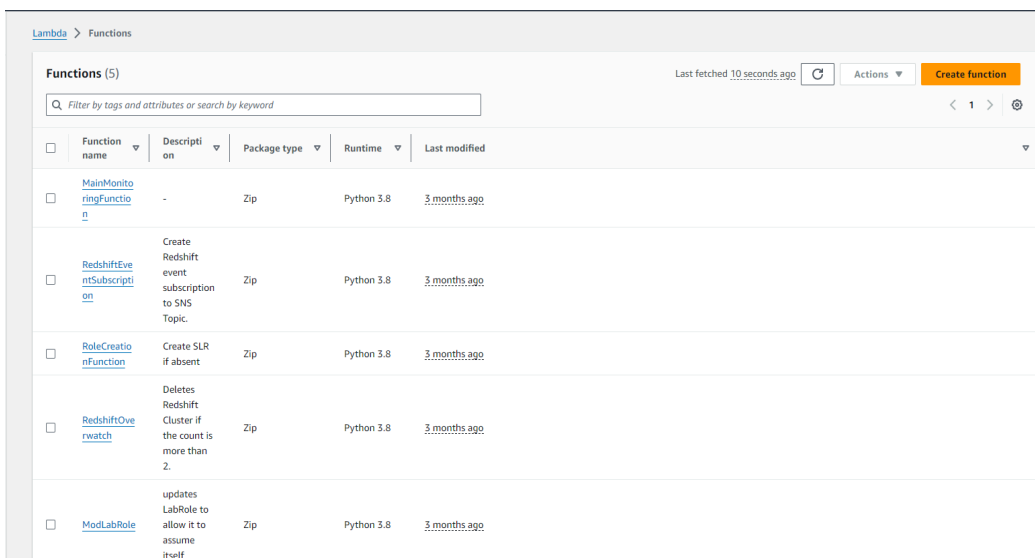
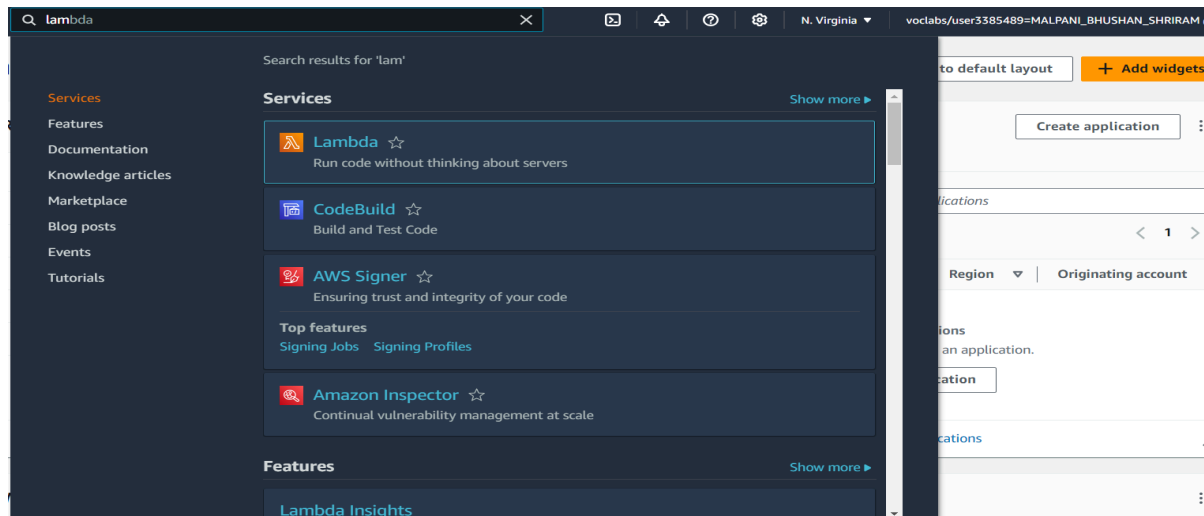
**Aim:** To understand AWS Lambda, its workflow, various functions and create your first Lambda functions using Python / Java / Nodejs.

**Prerequisites:**

- 1) AWS account (academy recommended)

## Step 1: Set up AWS Lambda Function

- 1) Search for Lambda in the services tab. Click on it once found.



- 2) Click on create functions.
- 3) Give a name to your Lambda function. Select the runtime as Node.js 20.x (You can also use python). Select the architecture as x86\_64. Set the default execution role as LabRole if you are doing this on your academy account. (Use an existing role → LabRole)

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** Info  
Enter a name that describes the purpose of your function.

Function name must be 1 to 64 characters, must be unique to the Region, and can't include spaces. Valid characters are a-z, A-Z, 0-9, hyphens (-), and underscores (\_).

**Runtime** Info  
Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.

**Architecture** Info  
Choose the instruction set architecture you want for your function code.

☒ x86\_64  
☐ arm64

**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**

- 4) Once the function is created, click on the name of the function (myLambda27 in my case).

Lambda > Functions

Functions (6) Last fetched 3 minutes ago Actions ▼ Create function

<input type="checkbox"/>	Function name ▼	Description ▼	Package type ▼	Runtime ▼	Last modified ▼
<input type="checkbox"/>	<a href="#">MainMonitoringFunction</a>	-	Zip	Python 3.8	3 months ago
<input type="checkbox"/>	<a href="#">RedshiftEventSubscription</a>	Create Redshift event subscription to SNS Topic.	Zip	Python 3.8	3 months ago
<input type="checkbox"/>	<a href="#">RoleCreationFunction</a>	Create SLR if absent	Zip	Python 3.8	3 months ago
<input type="checkbox"/>	<a href="#">RedshiftOverwatch</a>	Deletes Redshift Cluster if the count is more than 2.	Zip	Python 3.8	3 months ago
<input type="checkbox"/>	<a href="#">ModLabRole</a>	updates LabRole to allow it to assume itself	Zip	Python 3.8	3 months ago
<input type="checkbox"/>	<a href="#">bhushan33</a>	-	Zip	Node.js 20.x	32 seconds ago

5) This is the dashboard of our lambda function.

The screenshot shows the AWS Lambda console dashboard for a function named 'bhushan33'. At the top, there's a breadcrumb trail: 'Lambda > Functions > bhushan33'. The function name 'bhushan33' is prominently displayed. To the right of the name are buttons for 'Throttle', 'Copy ARN', and 'Actions'. Below the name, there's a 'Function overview' section with tabs for 'Diagram' and 'Template'. The 'Diagram' tab is active, showing a visual representation of the function with a box labeled 'bhushan33' and a 'Layers' section indicating '(0)' layers. To the right of the diagram, there's a 'Description' section with fields for 'Description', 'Last modified' (showing '1 minute ago'), 'Function ARN' (showing 'arn:aws:lambda:us-east-1:144602037631:function:bhushan33'), and 'Function URL' (with a link to 'Info'). Below the overview section, there's a horizontal navigation bar with tabs: 'Code', 'Test', 'Monitor', 'Configuration', 'Aliases', and 'Versions'. The 'Code' tab is selected, showing the 'Code source' section. It includes a search bar, a file explorer on the left showing the 'bhushan33' directory with an 'index.mjs' file, and a code editor displaying the default JavaScript code for the function. The code is as follows:

```
1 export const handler = async (event) => {
2   // TODO implement
3   const response = {
4     statusCode: 200,
5     body: JSON.stringify('Hello from Lambda!'),
6   };
7   return response;
8 };
9
```

6) This function has the following default code, which is used to print “Hello form Lambda!”.

This screenshot is a closer view of the 'Code source' section from the previous image. It shows the same code editor with the default JavaScript code for the 'bhushan33' function. The code is a single file named 'index.mjs'. The code is as follows:

```
1 export const handler = async (event) => {
2   // TODO implement
3   const response = {
4     statusCode: 200,
5     body: JSON.stringify('Hello from Lambda!'),
6   };
7   return response;
8 };
9
```

At the bottom right of the editor, there's a status bar showing '1:1 JavaScript Spaces: 2'.

## Step 3: Set up configurations and test events

- 1) Just above the test code, you would find Configuration, click on it. Then click on Edit.

Code	Test	Monitor	Configuration	Aliases	Versions
------	------	---------	---------------	---------	----------

General configuration

Triggers

Permissions

Destinations

Function URL

General configuration [Info](#)

Edit

Description	Memory	Ephemeral storage
-	128 MB	512 MB
Timeout	SnapStart <a href="#">Info</a>	
0 min 3 sec	None	

- 2) Here, change the Timeout to 1 sec. This is the time for which the function can be running before it is forcibly terminated.

Lambda > Functions > bhushan33 > Edit basic settings

Edit basic settings

Basic settings [Info](#)

Description - optional

Memory [Info](#)  
Your function is allocated CPU proportional to the memory configured.  
128 MB  
Set memory to between 128 MB and 10240 MB

Ephemeral storage [Info](#)  
You can configure up to 10 GB of ephemeral storage (/tmp) for your function. [View pricing](#)  
512 MB  
Set ephemeral storage (/tmp) to between 512 MB and 10240 MB.

SnapStart [Info](#)  
Reduce startup time by having Lambda cache a snapshot of your function after the function has initialized. To evaluate whether your function code is resilient to snapshot operations, review the [SnapStart compatibility considerations](#).  
None  
Supported runtimes: Java 11, Java 17, Java 21.

Timeout  
0 min 1 sec

Execution role  
Choose a role that defines the permissions of your function. To create a custom role, go to the [IAM console](#).  
☒ 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.  
LabRole  
View the LabRole role [on the IAM console](#).

- 3) We can see the executed changes.

Code	Test	Monitor	Configuration	Aliases	Versions
------	------	---------	---------------	---------	----------

General configuration

Triggers

Permissions

Destinations

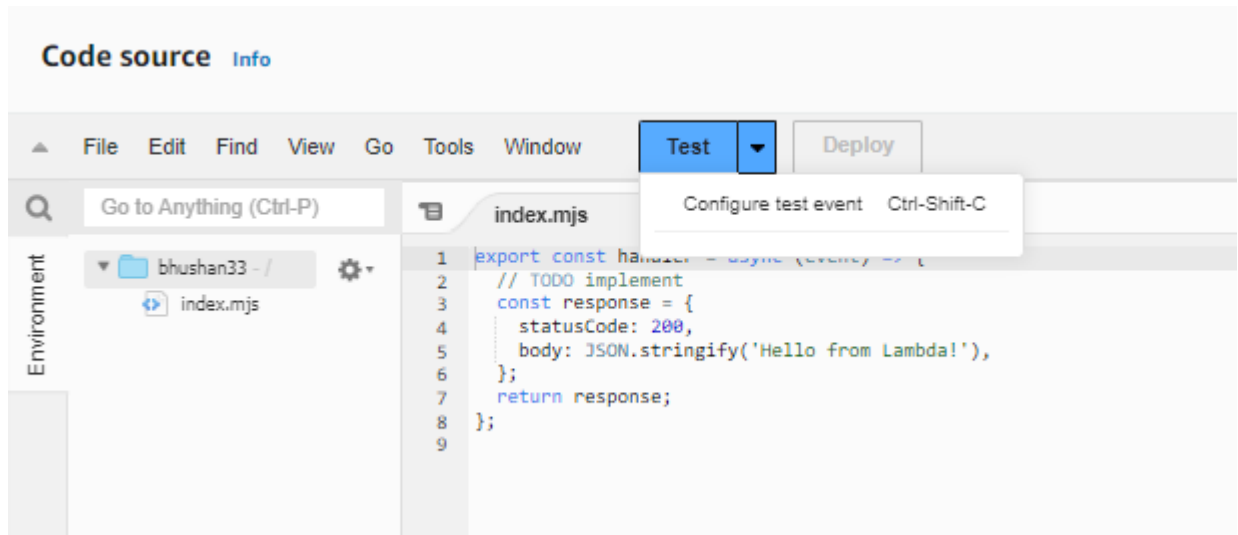
Function URL

General configuration [Info](#)

Edit

Description	Memory	Ephemeral storage
-	128 MB	512 MB
Timeout	SnapStart <a href="#">Info</a>	
0 min 1 sec	None	

- 4) Switch back to the code tab. Click on the dropdown arrow near test. Then select configure test event.



- 5) Here, create a new event, keep the other options default and save the event.

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

lambda33

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

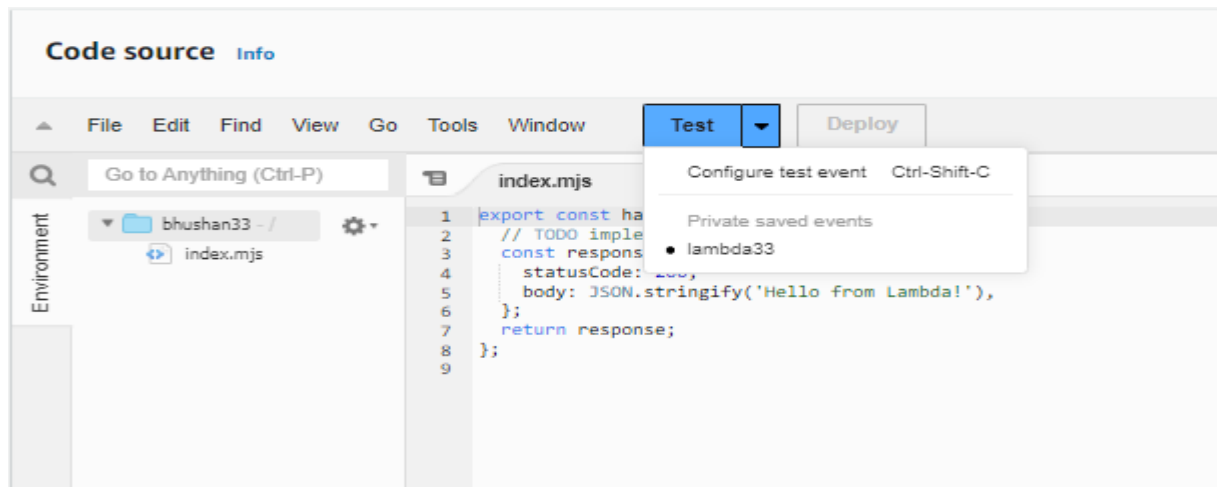
1 \* [
2 "key1": "value1",
3 "key2": "value2",
4 "key3": "value3"
5 ]

Cancel

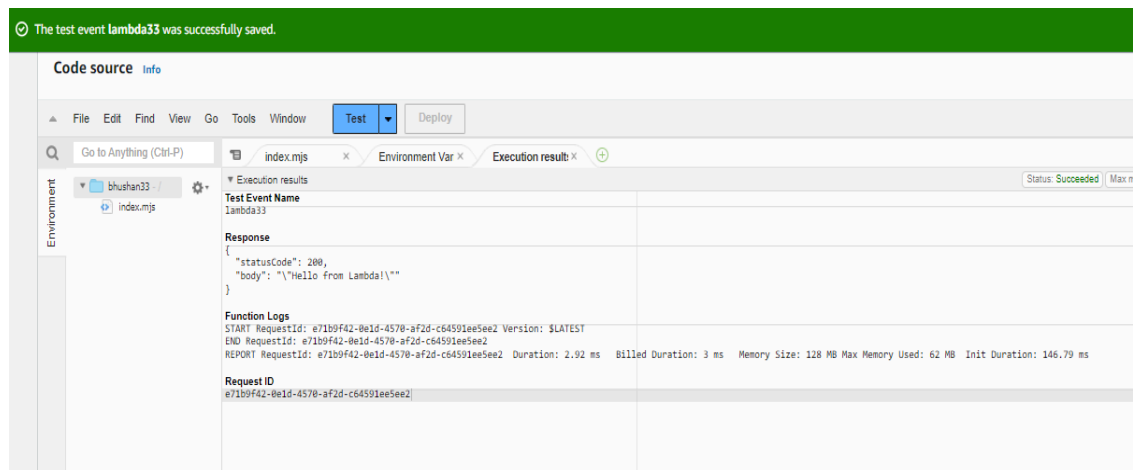
Invoke

Save

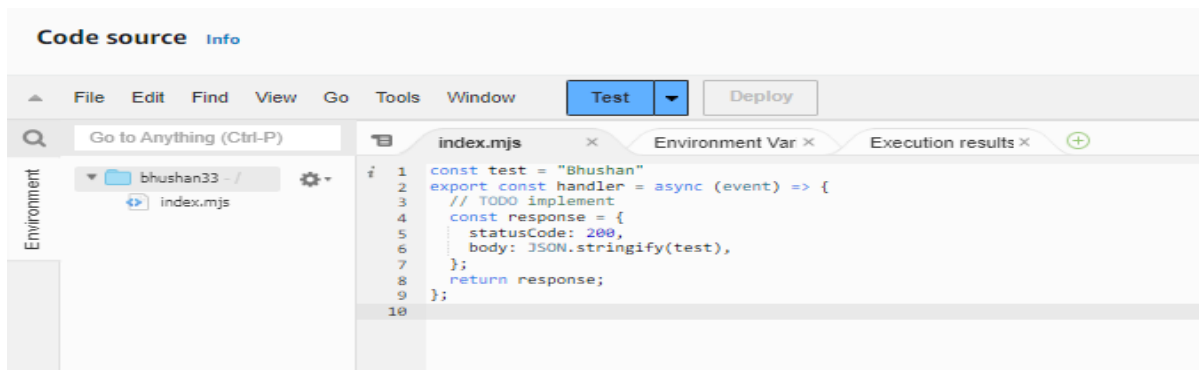
- 6) Now, again click on the dropdown. This time, select the event you have created. Then, click on TEST.



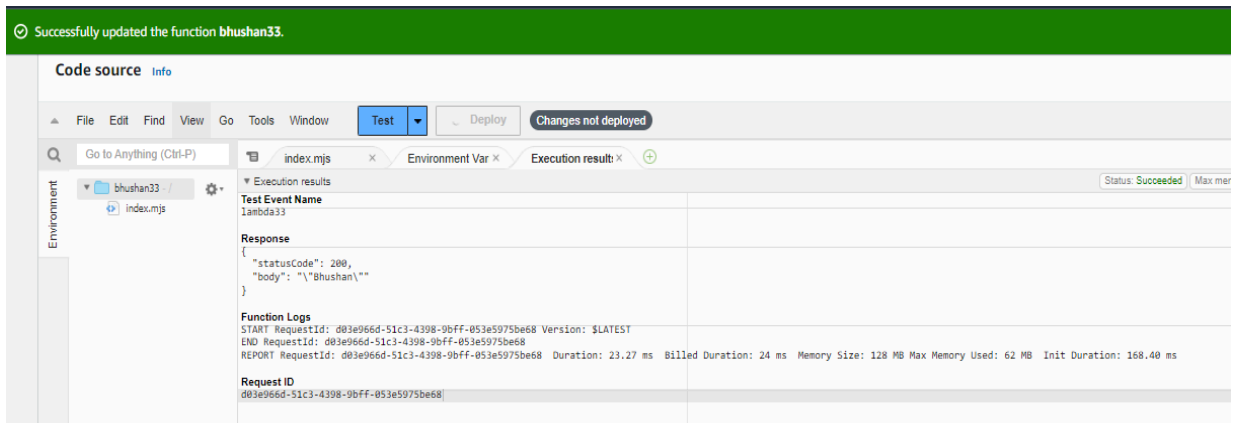
- 7) We can see the expected output for the sample code.



- 8) For a test, declare a string and call it in line 6. After making the changes click on deploy.



9) Run the test. We can see that the string we declared has come in the output.



## Conclusion:

In this experiment, we effectively investigated the AWS Lambda service by developing and configuring Lambda functions using Python, Java, or Node.js. We discovered how to establish a Lambda function, tweak its settings (like modifying the timeout duration), and evaluate the function with personalized events. Throughout this experience, we noted how Lambda manages executions, including timeout handling and producing expected results according to modifications in the code.