

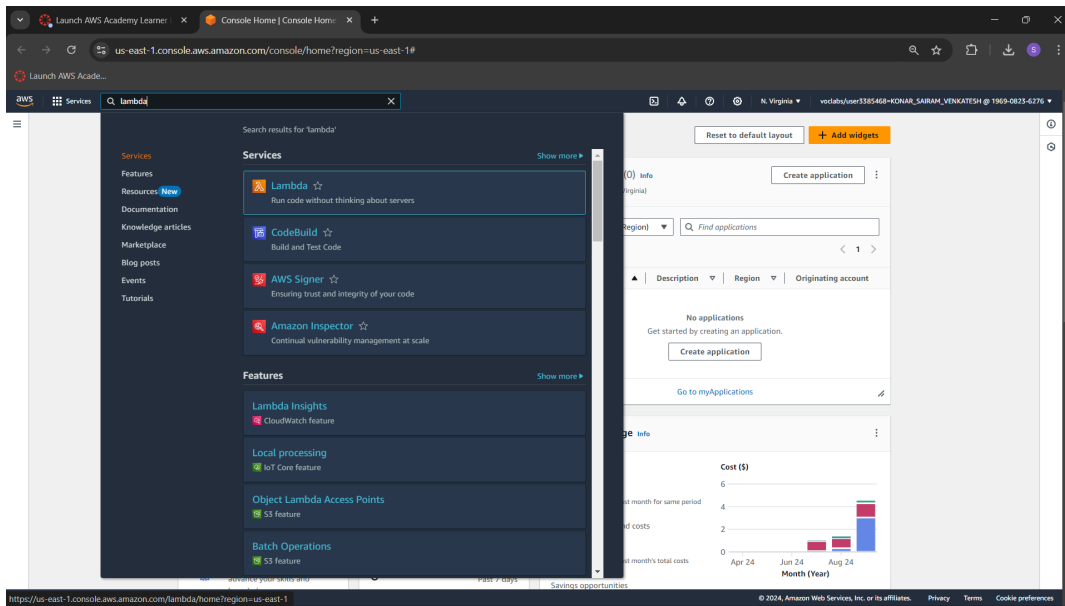
Aim: To understand AWS Lambda, its workflow, various functions and create your first Lambda functions using Python / Java / Nodejs.

Prerequisites:

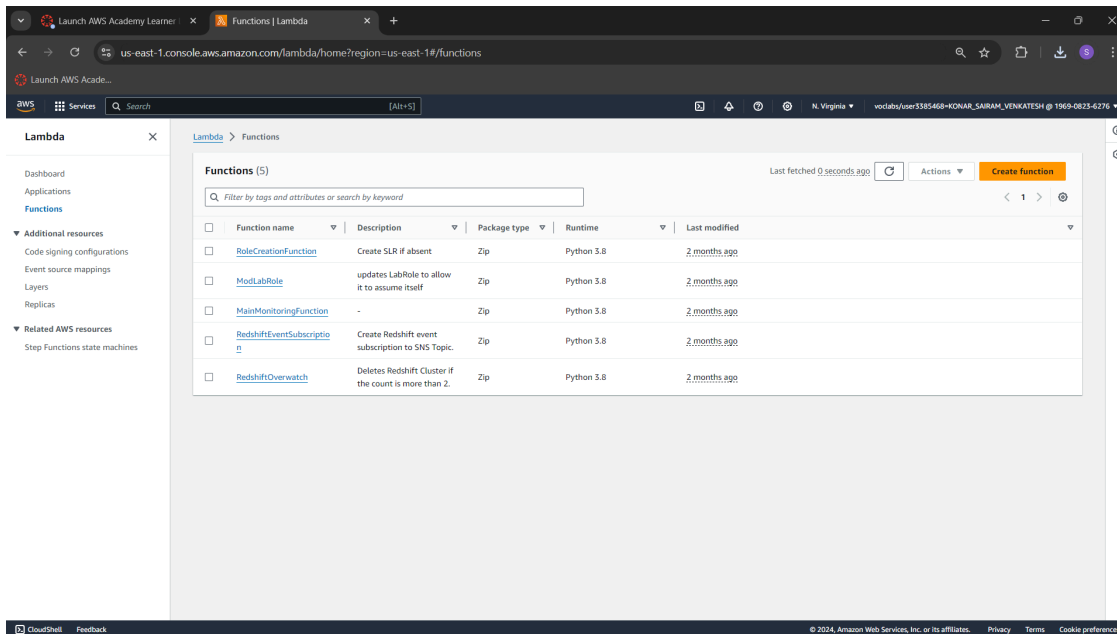
- 1) AWS account (academy recommended)

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)

Launch AWS Academy Learner | x | Create function | Functions | Lab | +

us-east-1.console.aws.amazon.com/lambda/home?region=us-east-1#/create/function

Launch AWS Academy Learner | x | Services | Search | [Alt+S]

Basic information

Function name
Enter a name that describes the purpose of your function.
myLambda27
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

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 [\[Link\]](#).
☐ 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.
LabRole
View the LabRole role [\[Link\]](#) on the IAM console.

CloudShell Feedback

© 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

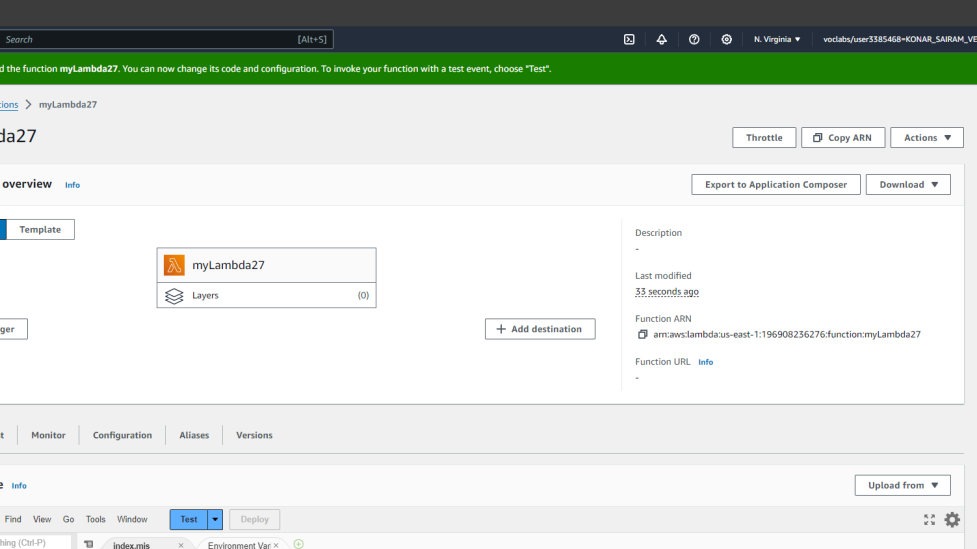
- 4) Once the function is created, click on the name of the function (myLambda27 in my case).

Lambda > Functions

Functions (7) Last fetched 0 seconds ago [Refresh](#) Actions [Create function](#)

Filter by tags and attributes or search by keyword

<input type="checkbox"/>	Function name	Description	Package type	Runtime	Last modified
<input type="checkbox"/>	myLambda27	-	Zip	Node.js 20.x	5 days ago
<input type="checkbox"/>	RoleCreationFunction	Create SLR if absent	Zip	Python 3.8	2 months ago
<input type="checkbox"/>	ModLabRole	updates LabRole to allow it to assume itself	Zip	Python 3.8	2 months ago
<input type="checkbox"/>	myLambda27_12	-	Zip	Python 3.12	5 days ago
<input type="checkbox"/>	MainMonitoringFunction	-	Zip	Python 3.8	2 months ago
<input type="checkbox"/>	RedshiftEventSubscription	Create Redshift event subscription to SNS Topic.	Zip	Python 3.8	2 months ago
<input type="checkbox"/>	RedshiftOverwatch	Deletes Redshift Cluster if the count is more than 2.	Zip	Python 3.8	2 months ago



The screenshot shows the AWS Lambda console for a function named 'myLambda27'. The function is in the 'Test' tab, and the code source is visible. The code is a simple Node.js handler that returns a 200 status code and a 'hello from Lambda!' message.

Function Overview:

- Function name:** myLambda27
- Layers:** (0)
- Description:** -
- Last modified:** 33 seconds ago
- Function ARN:** arn:aws:lambda:us-east-1:196908236276:function:myLambda27
- Function URL:** info

Code source:

```

import { handler as _handler } = require('./index.mjs')

export const handler = async (event) => {
  // TODO: Implement
  const response = {
    statusCode: 200,
    body: JSON.stringify('hello from Lambda!'),
  }
}
  
```

The screenshot shows the AWS Lambda console interface. At the top, a green banner indicates: "Successfully created the function myLambda27. You can now change its code and configuration. To invoke your function with a test event, choose 'Test'." Below this, the "Code source" tab is active, displaying a JavaScript code editor for a file named "index.mjs". The code is as follows:

```

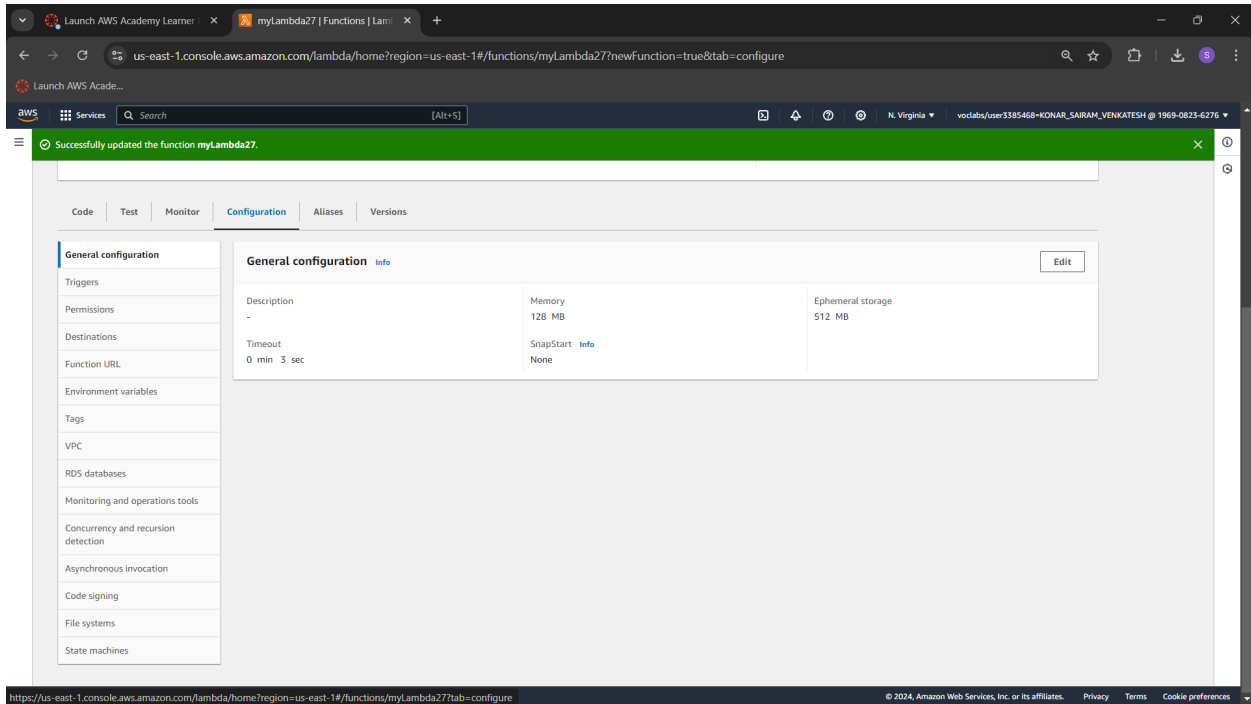
1 export const handler = async (event) => {
2   // 1000 implement
3   const response = {
4     statusCode: 200,
5     body: JSON.stringify('hello from Lambda!'),
6   };
7   return response;
8 };
9

```

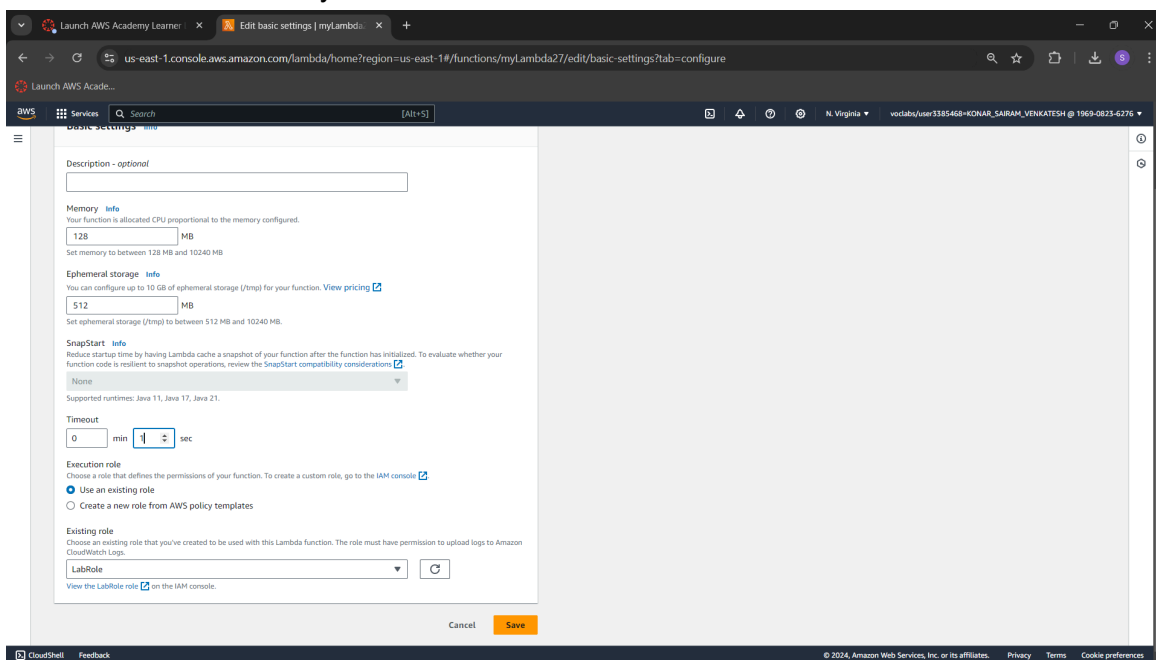
On the left side of the code editor, there is a file explorer showing the project structure with "myLambda27" and "index.mjs". The bottom of the console shows the "Code properties" section, which includes fields for "Package size" (205 bytes), "SHA256 hash", and "Last modified" (43 seconds ago).

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.



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

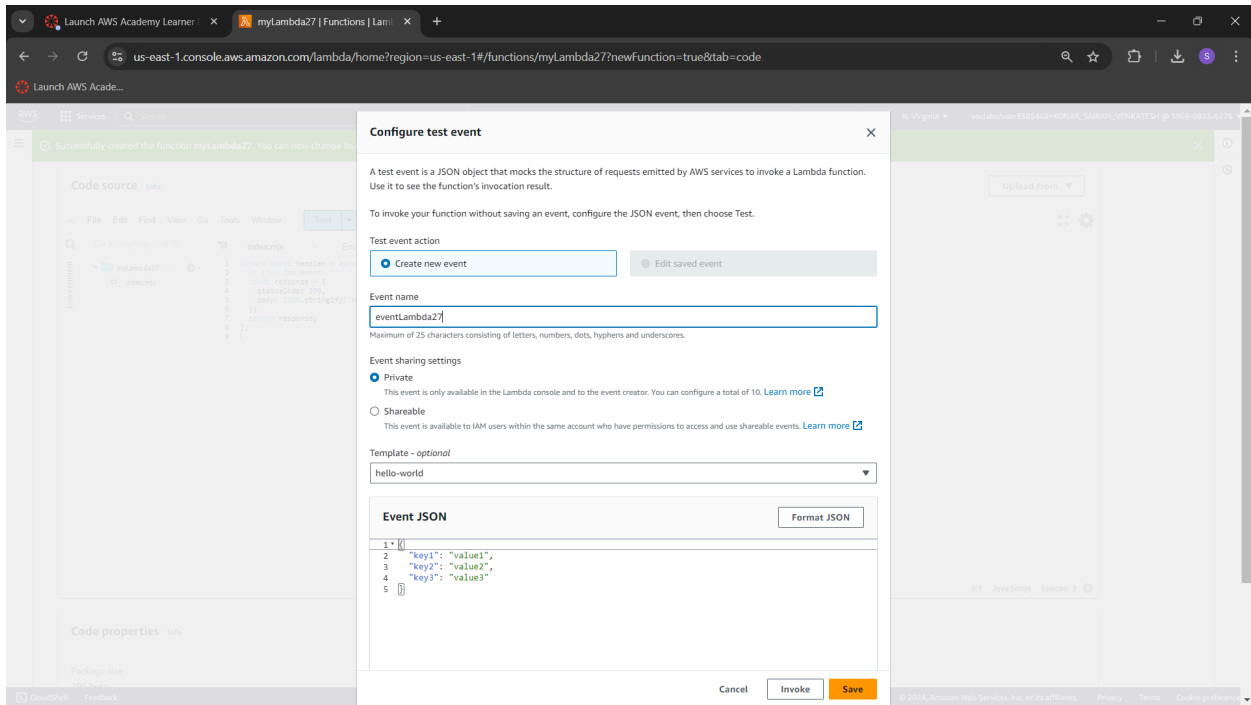


The screenshot shows the AWS Lambda console for a function named 'myLambda27' in the 'us-east-1' region. The 'Configuration' tab is selected, displaying the 'General configuration' section. The configuration table shows the following details:

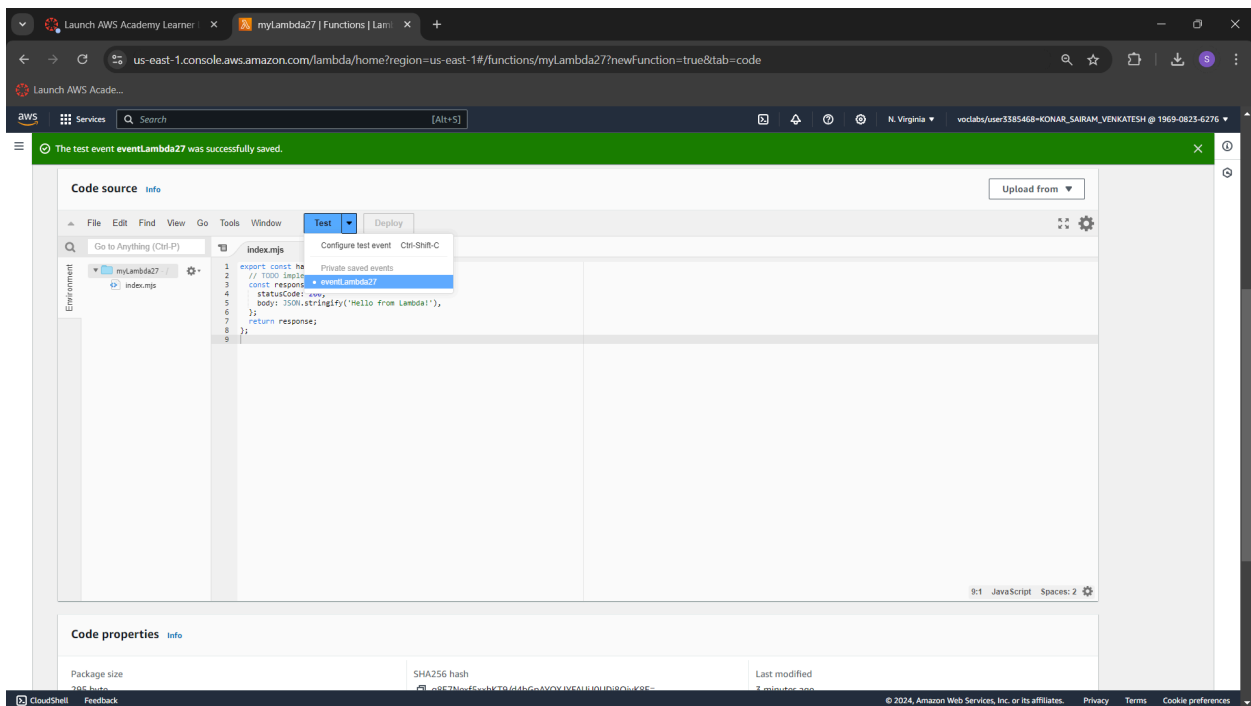
General configuration Info		
Description	Memory	Ephemeral storage
-	128 MB	512 MB
Timeout	SnapStart Info	
0 min 1 sec	None	

The left sidebar lists various configuration options: Triggers, Permissions, Destinations, Function URL, Environment variables, Tags, VPC, RDS databases, Monitoring and operations tools, Concurrency and recursion detection, Asynchronous invocation, Code signing, File systems, and State machines. The top navigation bar shows the function name and region, and the bottom status bar displays the URL and copyright information.

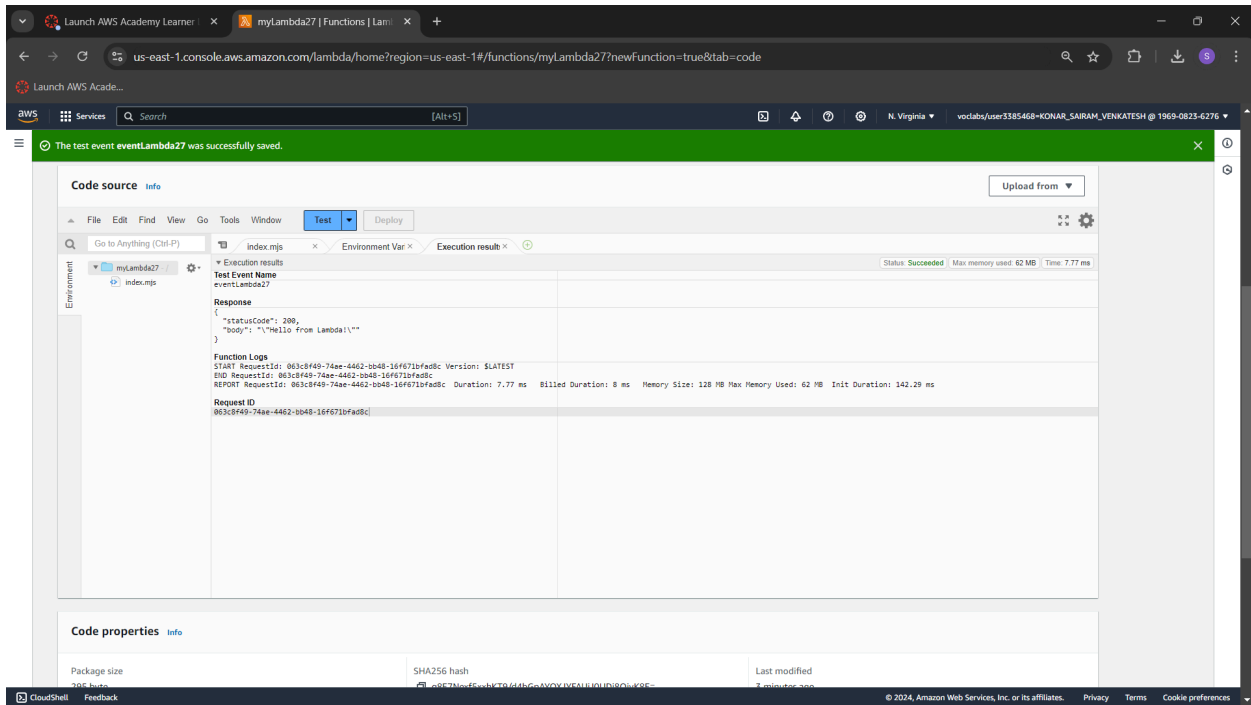
5) Here, create a new event, keep the other options default and save the event.



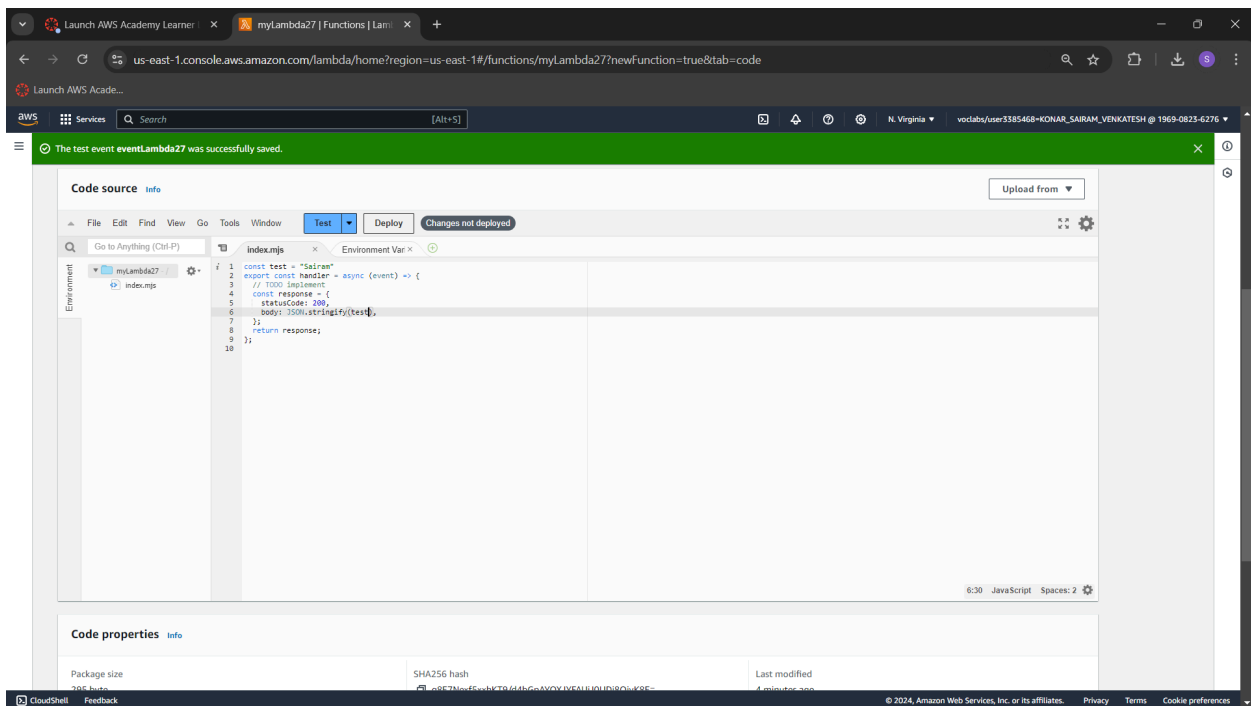
6) Now, again click on the dropdown. This time, select the event you have created. Then, click on TEST.



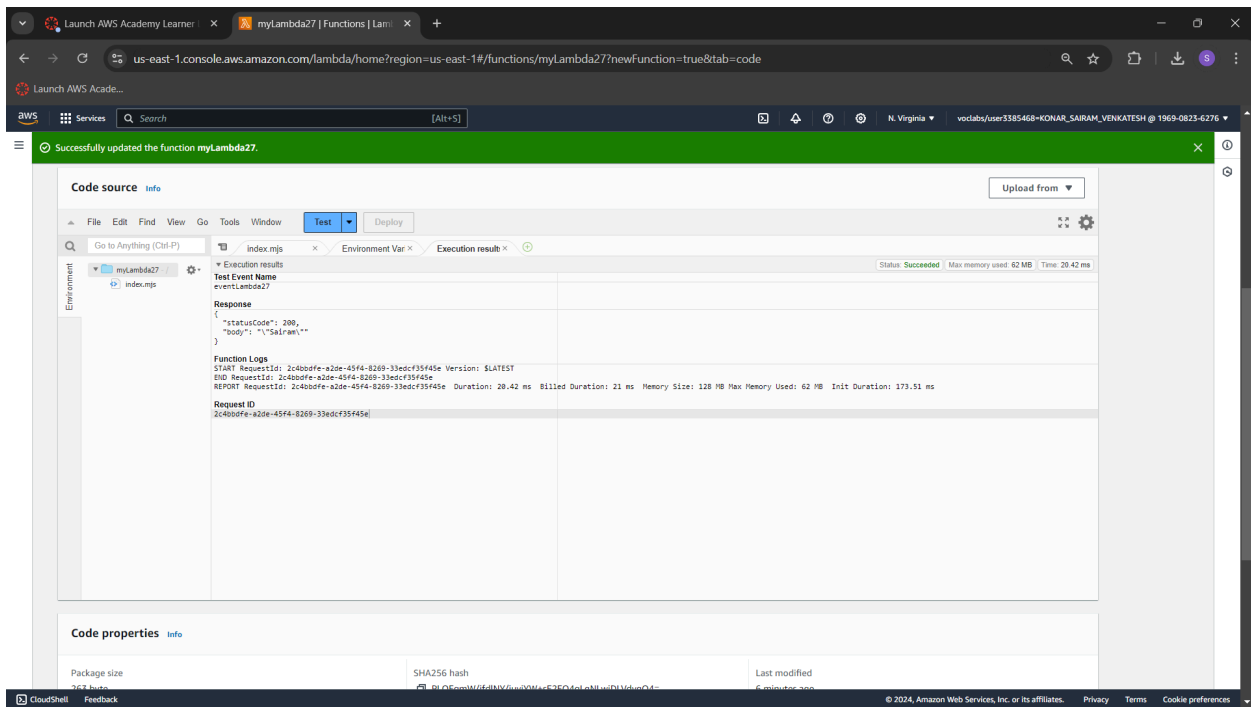
7) We can see the expected output for the sample code.



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 successfully explored the AWS Lambda service by creating and configuring Lambda functions using Python, Java, or Node.js. We learned how to set up a Lambda function, modify its configuration (such as adjusting the timeout), and test the function with custom events. Through this process, we observed how Lambda handles executions, including managing timeouts and returning expected outputs based on the code changes.