Name: Krushikesh Shelar **2024-2025** **Roll No:51** **Div:D15-C**

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

Theory:

Overview of AWS Lambda

AWS Lambda is a serverless computing service that allows you to run code without managing servers. It automatically scales in response to incoming requests or events, ensuring efficient resource use. You are billed only for the time your code is executed, making it a budget-friendly option for on-demand applications.

How AWS Lambda Works

1. **Function Development**: Begin by writing your code and designating a handler (the entry point). You can use the AWS Management Console, CLI, or upload a package.
2. **Trigger Setup**: Determine the events that will trigger your function, such as uploads to S3 or changes in a DynamoDB table.
3. **Execution**: Upon triggering, Lambda runs your function and handles the necessary logic while automatically scaling to accommodate the volume of events.
4. **Scaling and Concurrency**: Lambda dynamically scales by creating additional instances of your function for simultaneous requests. You can also configure reserved concurrency to manage peak traffic.
5. **Monitoring and Logging**: Integrated with Amazon CloudWatch, Lambda provides logging and monitoring capabilities. You can view logs for each execution, helping you track performance and troubleshoot issues.

Programming Languages Supported

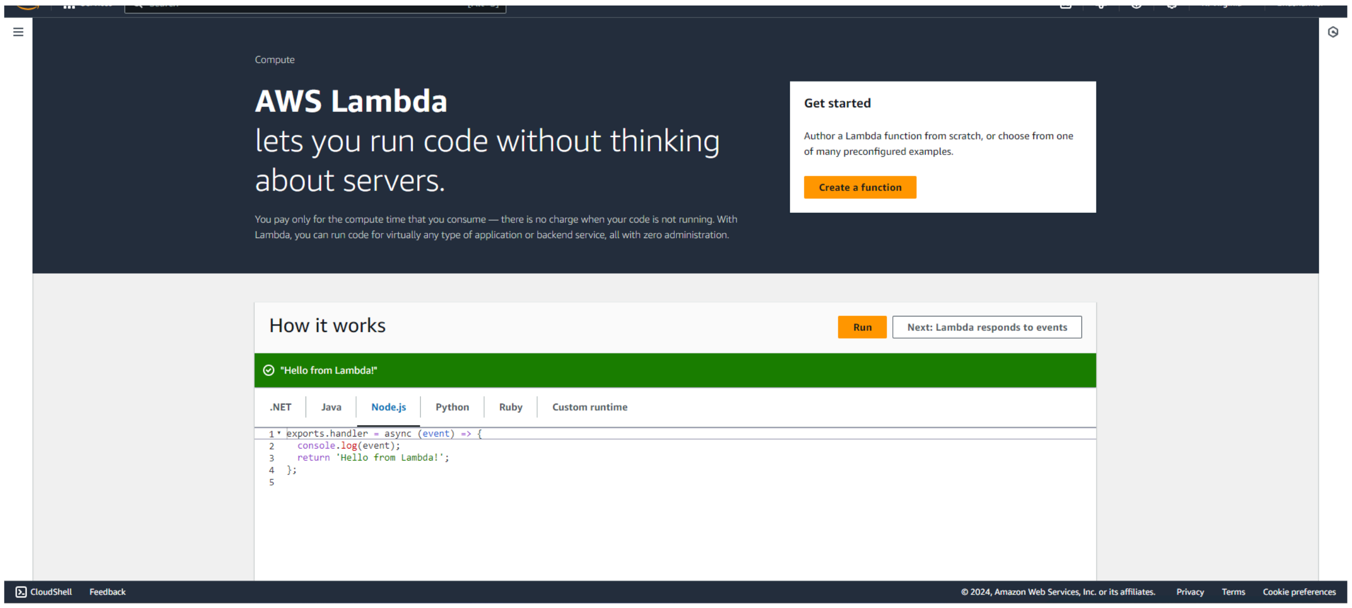
- **Python**: Excellent for quick development and lightweight tasks due to its extensive standard library.
- **Java**: Suitable for more demanding, compute-intensive applications, though it may have longer cold start times.
- **Node.js**: Well-suited for I/O-heavy applications like APIs, offering fast startup times and efficient resource use.

Prerequisites: AWS Personal/Academy Account

Name: Krushikesh Shelar **2024-2025** **Roll No:51** **Div:D15-C**

Steps To create the lambda function:

Step 1: Login to your AWS Personal/Academy Accout.Open lambda and click on create function button.



Compute

AWS Lambda lets you run code without thinking about servers.

You pay only for the compute time that you consume — there is no charge when your code is not running. With Lambda, you can run code for virtually any type of application or backend service, all with zero administration.

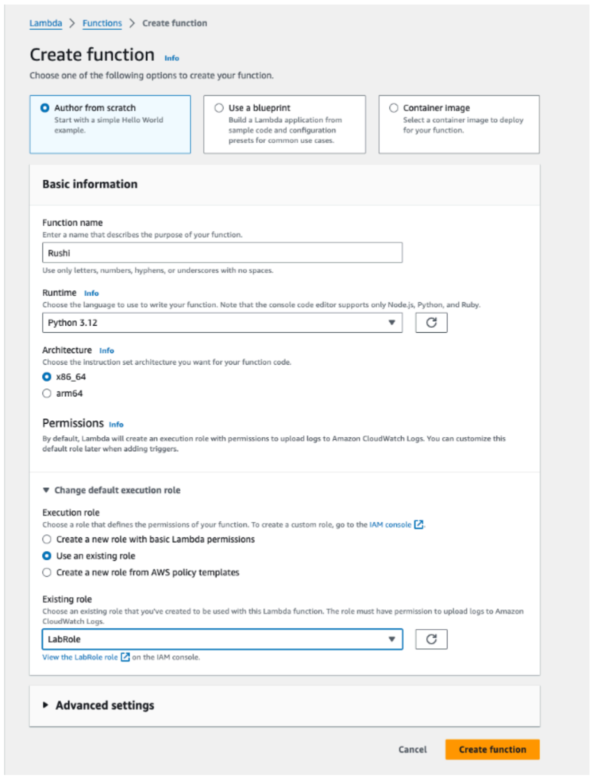
Get started

Author a Lambda function from scratch, or choose from one of many preconfigured examples.

```
exports.handler = async (event) => {
  console.log(event);
  return 'Hello from Lambda!';
};
```

Step 2: Now Give a name to your Lambda function, Select the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby. So will select Python 3.12 , Architecture as x86, and Existing(Lab Role) role to Create a new role with basic Lambda permissions.

Name: Krushikesh Shelar 2024-2025 Roll No:51 Div:D15-C



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: Rushi

Enter a name that describes the purpose of your function.

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

Runtime Info: Python 3.12

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

Architecture Info: [✓ x86_64] arm64

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

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: LabRole

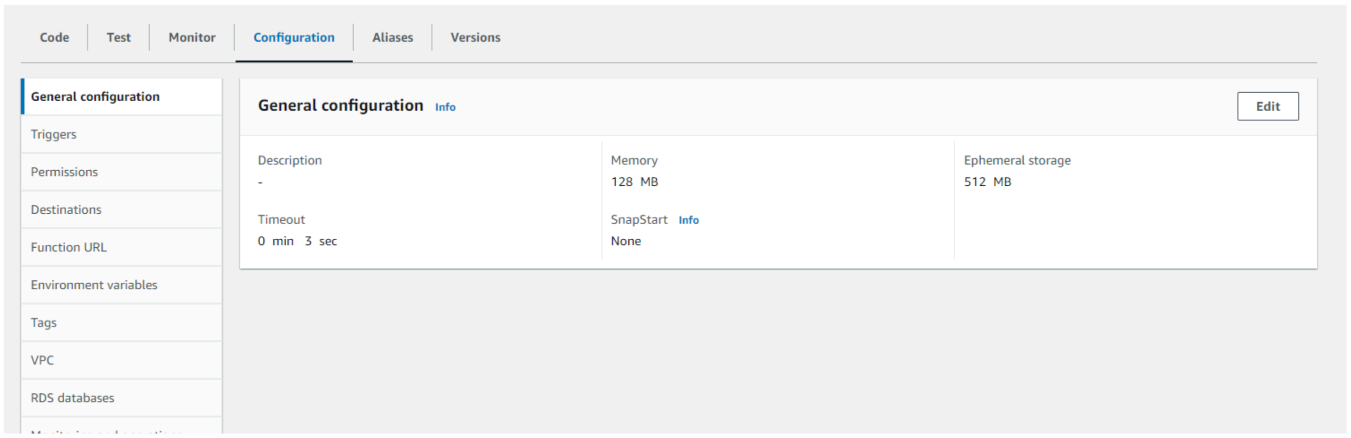
Choose an existing role that you've created to be used with this Lambda function. The role must have permission to upload logs to Amazon CloudWatch Logs.

View the LabRole role on the IAM console.

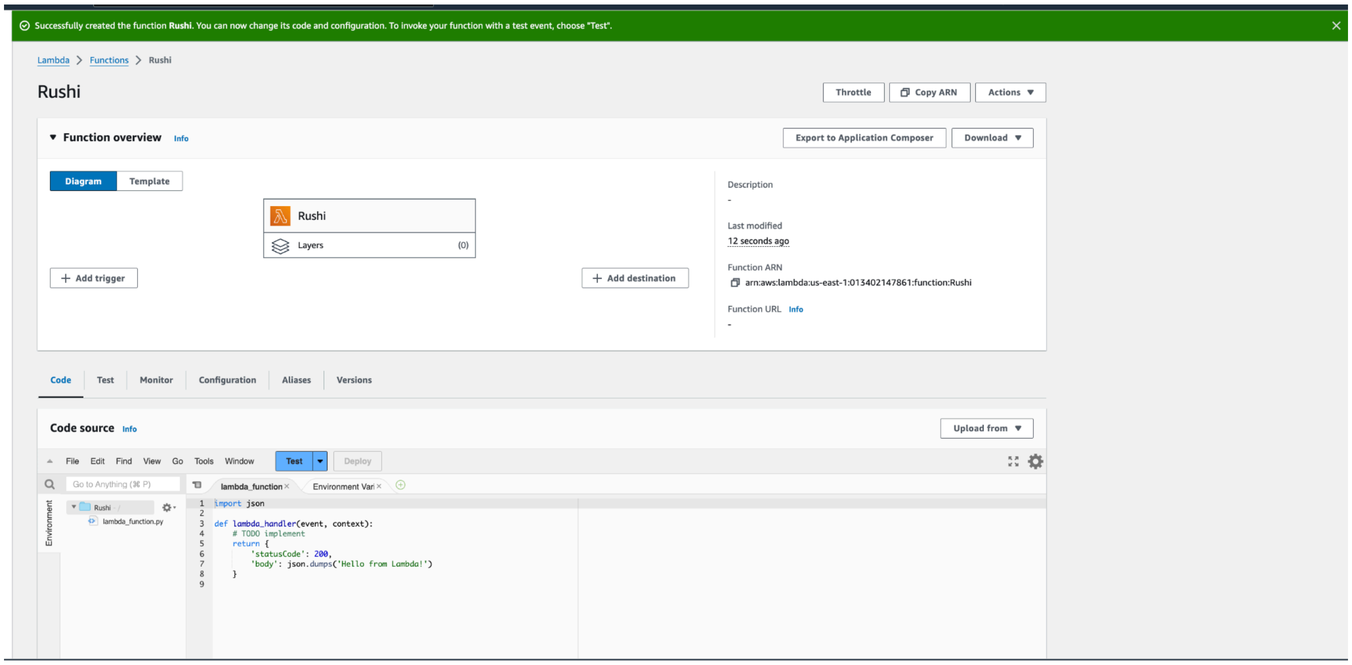
Advanced settings

So See or Edit the basic settings go to configuration then click on edit general setting.

Name: Krushikesh Shelar **2024-2025** **Roll No:51** **Div:D15-C**


General configuration

Description	Memory	Ephemeral storage
-	128 MB	512 MB
Timeout	**SnapStart**	
0 min 3 sec	None	


Successfully created the function **Rushi**. You can now change its code and configuration. To invoke your function with a test event, choose "Test".

Lambda > Functions > Rushi

Description: -

Last modified: 12 seconds ago

Function ARN: arn:aws:lambda:us-east-1:013402147861:function:Rushi

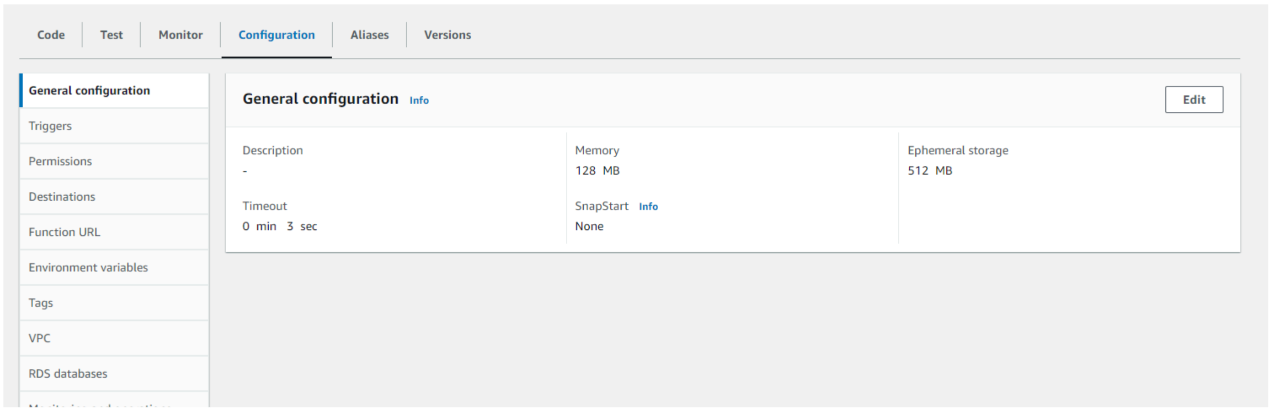
Function URL: -

```
import json

def lambda_handler(event, context):
    # TODO implement
    return {
        'statusCode': 200,
        'body': json.dumps('Hello from Lambda!')
    }
```

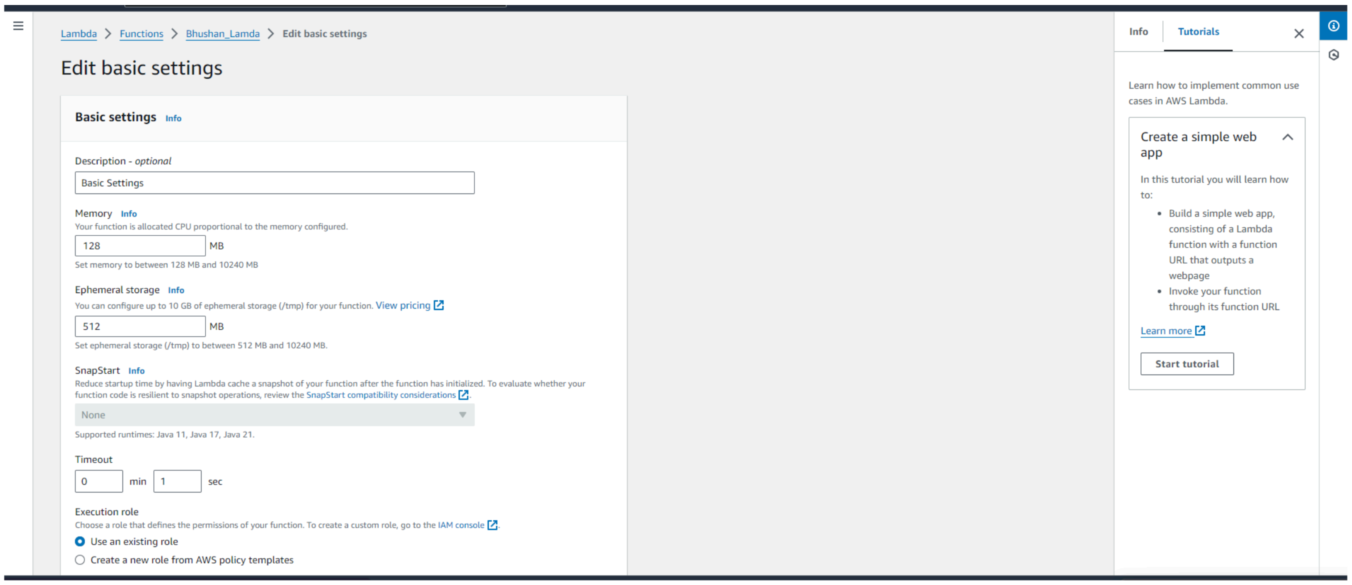
Here, you can enter a description and change Memory and Timeout. I’ve changed the Timeout period to 1 sec since that is sufficient for now.

Step 3: Now Click on the Test tab then select Create a new event, give a name to the event and select Event Sharing to private, and select hello-world template.


General configuration

Description	Memory	Ephemeral storage
-	128 MB	512 MB
Timeout	**SnapStart**	
0 min 3 sec	None	

Name: Krushikesh Shelar **2024-2025** **Roll No:51** **Div:D15-C**



Edit basic settings

Basic settings Info

Description - *optional*
Basic Settings

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.

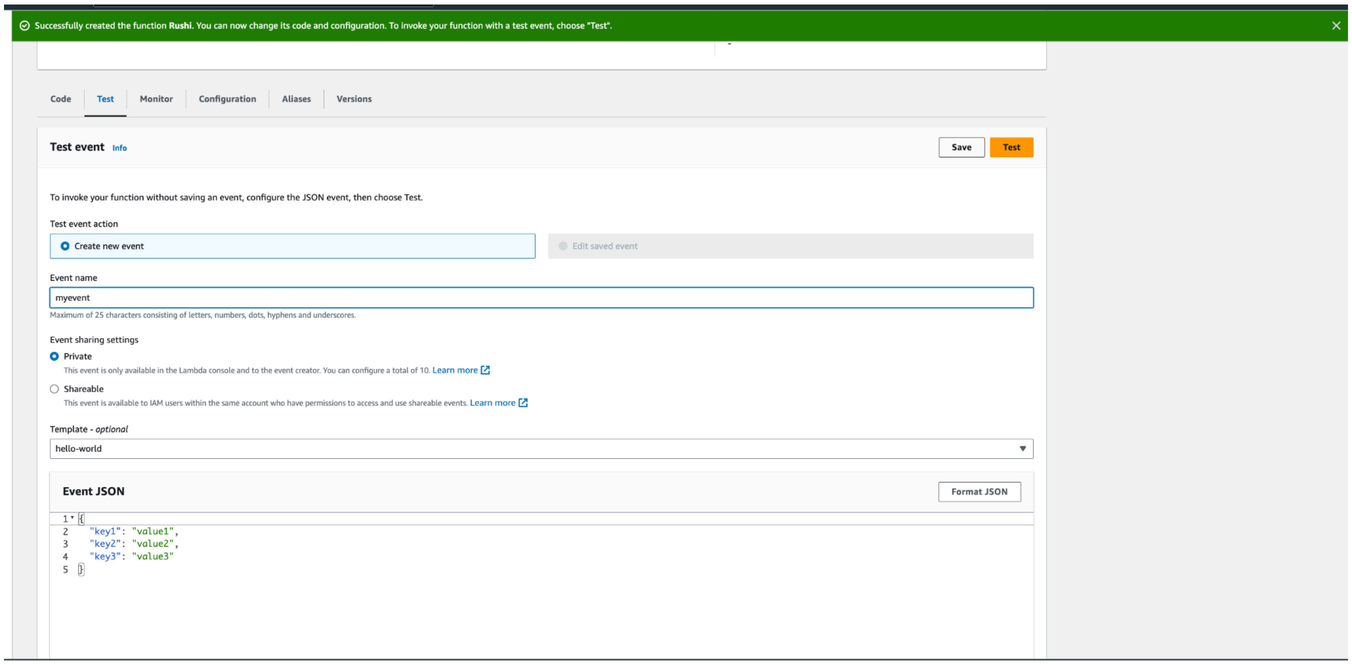
SnapStart Info
Reduce startup time by having Lambda cache a snapshot of your function after the function has initialized. To evaluate whether your function code is resilient to snapshot operations, review the SnapStart compatibility considerations.
None
Supported runtimes: Java 11, Java 17, Java 21.

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

Learn how to implement common use cases in AWS Lambda.

In this tutorial you will learn how to:
- Build a simple web app, consisting of a Lambda function with a function URL that outputs a webpage
- Invoke your function through its function URL



Successfully created the function Rushi. You can now change its code and configuration. To invoke your function with a test event, choose "Test".

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

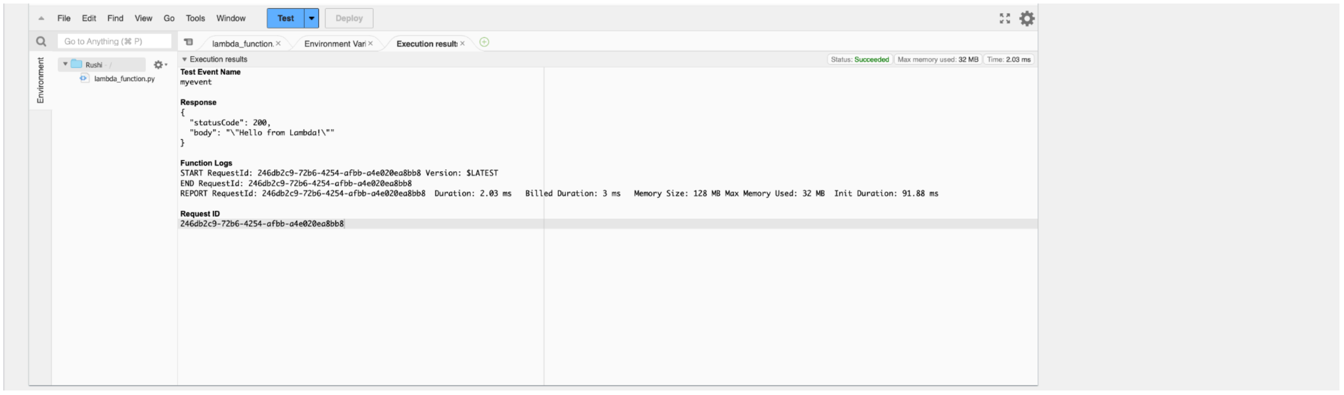
Event sharing settings
● Private
This event is only available in the Lambda console and to the event creator. You can configure a total of 10. Learn more
○ Shareable
This event is available to IAM users within the same account who have permissions to access and use shareable events. Learn more

Template - *optional*
hello-world

Event JSON

```
{
  "key1": "value1",
  "key2": "value2",
  "key3": "value3"
}
```

Step 4: Now In Code section select the created event from the dropdown of test then click on test . You will see the below output.



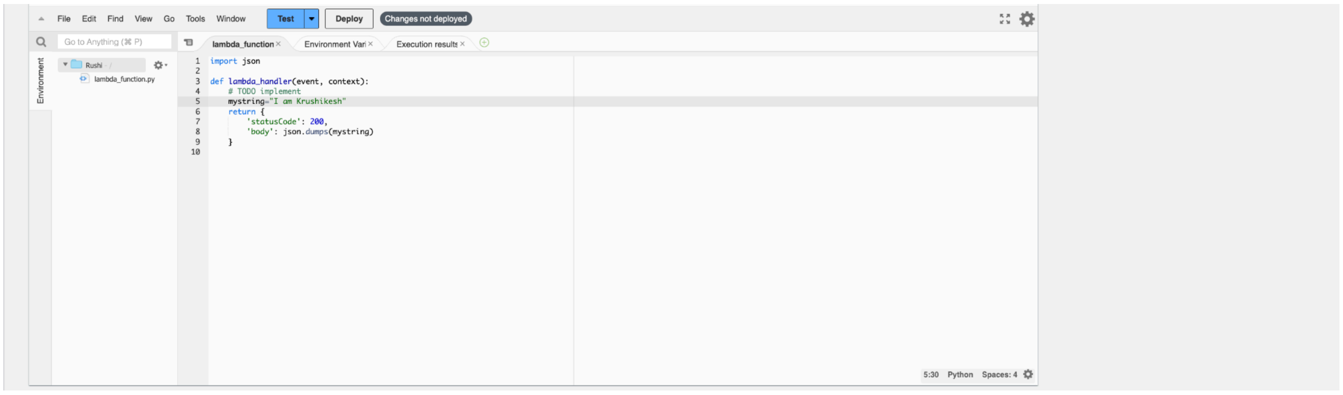
```
Test Event Name
myevent

Response
{
  "statusCode": 200,
  "body": "\"Hello from Lambda!\""
}

Function Logs
START RequestId: 246db2c9-72b6-4254-afbb-a4e020ea8bb8 Version: $LATEST
END RequestId: 246db2c9-72b6-4254-afbb-a4e020ea8bb8
REPORT RequestId: 246db2c9-72b6-4254-afbb-a4e020ea8bb8  Duration: 2.03 ms   Billed Duration: 3 ms   Memory Size: 128 MB Max Memory Used: 32 MB  Init Duration: 91.88 ms

Request ID
246db2c9-72b6-4254-afbb-a4e020ea8bb8
```

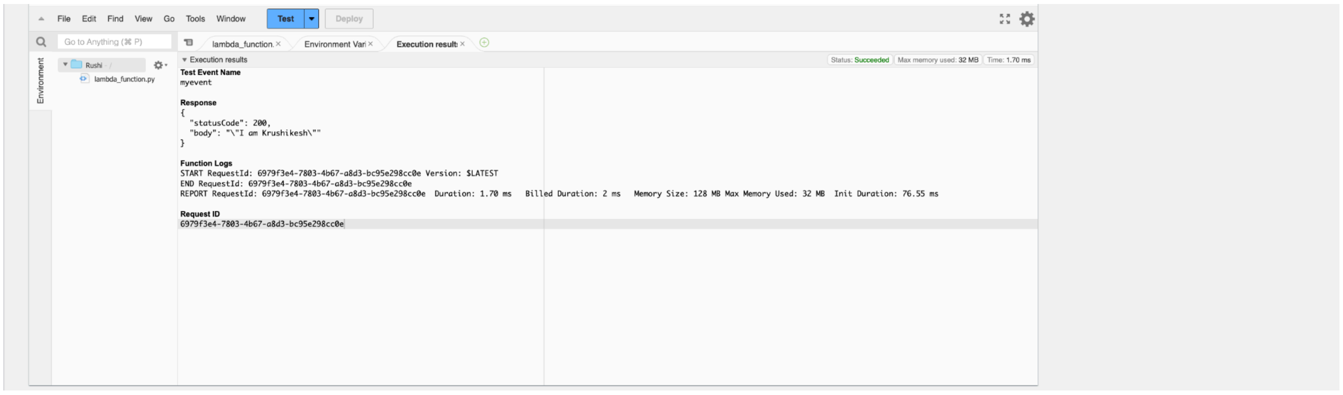
Step 5: You can edit your lambda function code. I have changed the code to display the new String. Now ctrl+s to save and click on deploy to deploy the changes.



```
import json

def lambda_handler(event, context):
    # TODO implement
    mystring="I am Krushikesh"
    return {
        'statusCode': 200,
        'body': json.dumps(mystring)
    }
```

Step 6: Now click on the test and observe the output. We can see the status code 200 and your string output and function logs. On successful deployment.



```
Test Event Name
myevent

Response
{
  "statusCode": 200,
  "body": "\"I am Krushikesh\""
}

Function Logs
START RequestId: 6979f3e4-7803-4b67-a8d3-bc95e298cc0e Version: $LATEST
END RequestId: 6979f3e4-7803-4b67-a8d3-bc95e298cc0e
REPORT RequestId: 6979f3e4-7803-4b67-a8d3-bc95e298cc0e  Duration: 1.70 ms   Billed Duration: 2 ms   Memory Size: 128 MB Max Memory Used: 32 MB  Init Duration: 76.55 ms

Request ID
6979f3e4-7803-4b67-a8d3-bc95e298cc0e
```

Conclusion:

We successfully built an AWS Lambda function and explored the core steps involved. After developing the function in Python, we adjusted the key settings, including a timeout of 1 second. We then generated a test event, deployed the function, and checked the output. Furthermore, we updated the function's code and redeployed it to observe the immediate impact of those changes.

This hands-on session showcased the user-friendly nature and adaptability of AWS Lambda for building serverless applications, allowing us to focus on writing code while AWS efficiently handles infrastructure and scaling concerns.