Name: **Shreyash Kamat** Div/ Roll no: **D15C/ 22**

# Experiment 11

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

## Theory:

### AWS Lambda

A fully managed, serverless computing service where you run code without provisioning or managing servers. Lambda automatically scales your application based on the number of incoming requests or events, ensuring efficient resource utilization. You are only charged for the time your code is running, with no upfront cost, making it cost-effective for on-demand workloads.

### Lambda Workflow

- **Create a Function**: Write the function code and define its handler (entry point). You can use the AWS Console, CLI, or upload a deployment package.
- **Set Event Sources**: Define how the function is triggered (e.g., when an object is uploaded to S3 or a DynamoDB table is updated).
- **Execution**: When triggered, Lambda runs your function, executes the logic, and automatically scales to handle the incoming event volume.
- **Scaling and Concurrency**: Lambda scales automatically by launching more instances of the function to handle simultaneous invocations. There are also options for configuring **reserved concurrency** to manage traffic.
- **Monitoring and Logging**: Lambda integrates with Amazon CloudWatch for logging and monitoring. Logs for each invocation are sent to CloudWatch, allowing you to track performance and troubleshoot errors.

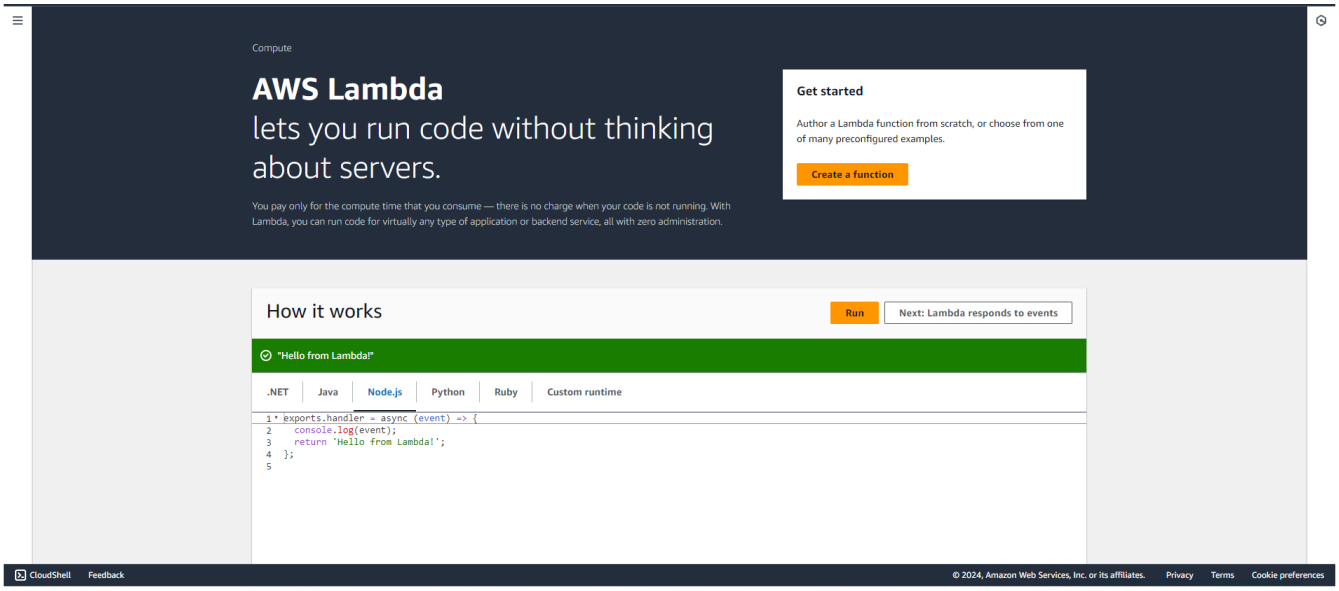
### AWS Lambda Functions

- **Python**: Great for quick development with its rich standard library and support for lightweight tasks.
- **Java**: Typically used for more complex, compute-intensive tasks. While it's robust, cold start times can be higher.
- **Node.js**: Excellent for I/O-bound tasks like handling APIs or streaming data, with fast startup times and efficient memory usage.

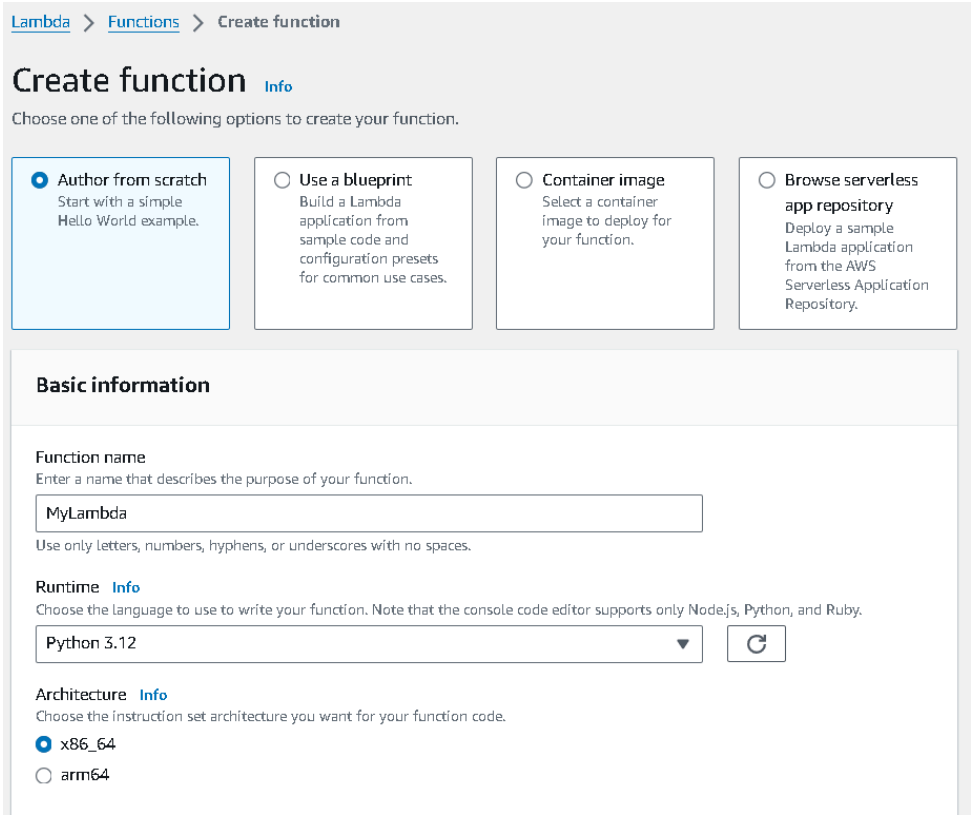
**Prerequisites:** AWS Personal/Academy Account

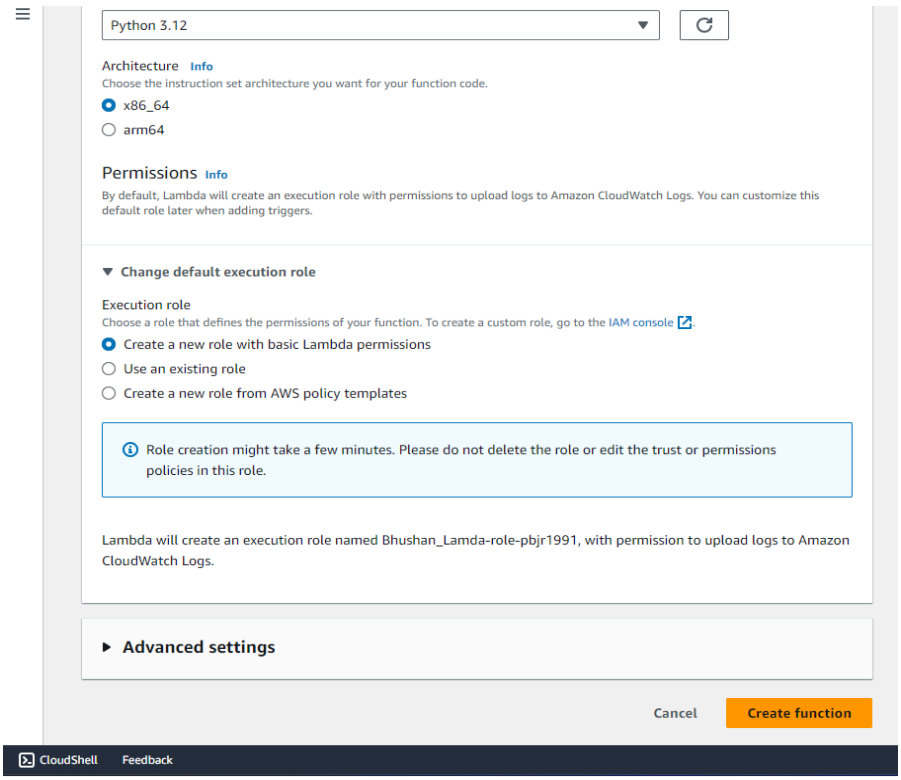
**Steps To create the lambda function:**

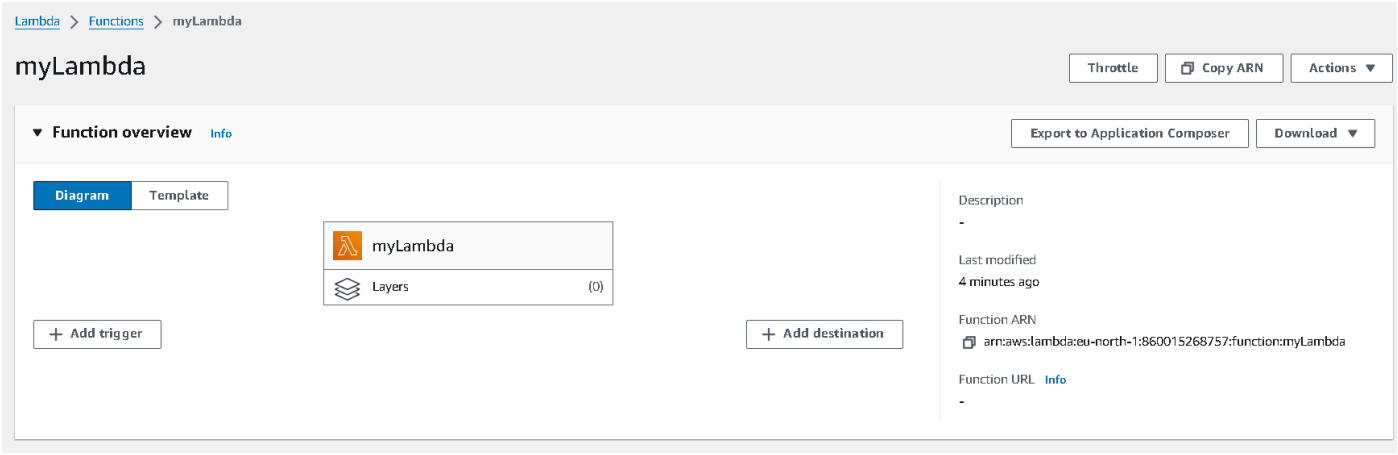
**Step 1:** Login to your AWS Personal/Academy Accout.Open lambda and click on create function button.

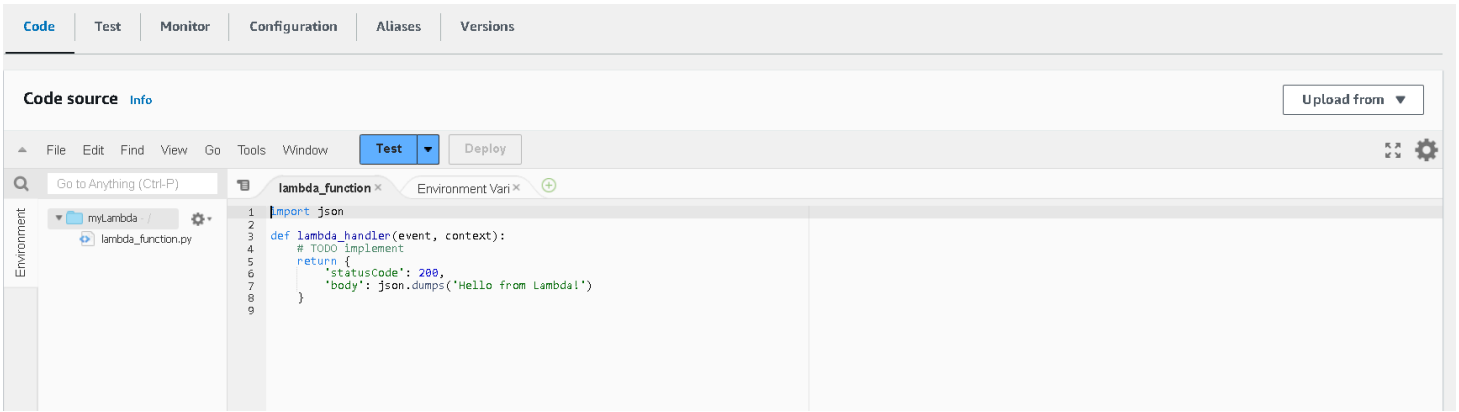



**Step 2:** Now Give a name to your Lambda function, Select the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby. So will select Python 3.12, Architecture as x86, and Execution role to Create a new role with basic Lambda permissions.

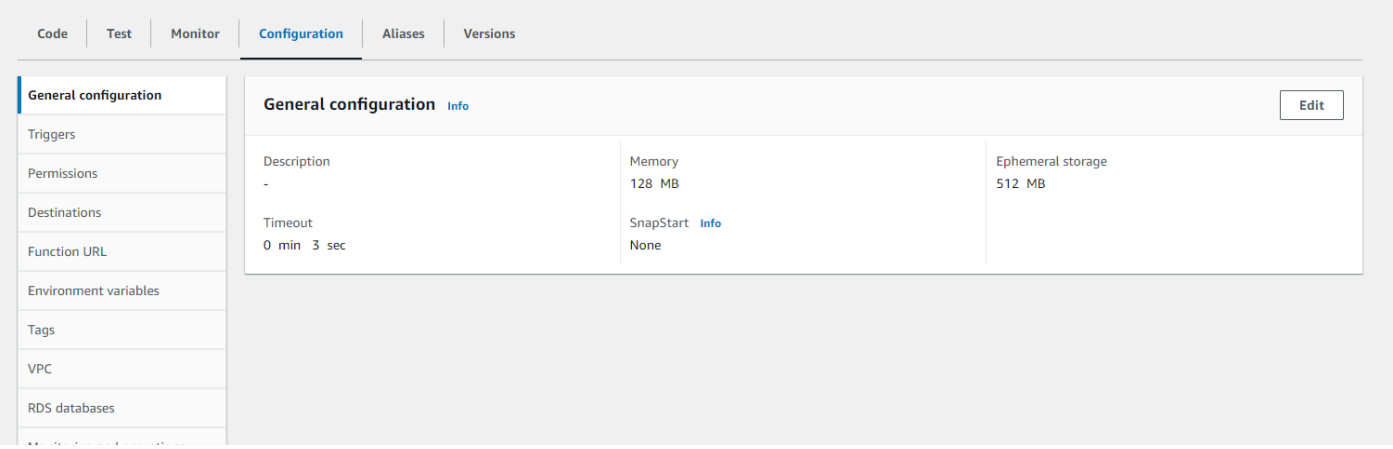



Python 3.12
Architecture Info
Choose the instruction set architecture you want for your function code.
x86_64
arm64
Permissions Info
By default, Lambda will create an execution role with permissions to upload logs to Amazon CloudWatch Logs. You can customize this default role later when adding triggers.
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
Role creation might take a few minutes. Please do not delete the role or edit the trust or permissions policies in this role.
Lambda will create an execution role named Bhushan_Lamda-role-pbjr1991, with permission to upload logs to Amazon CloudWatch Logs.
Advanced settings
Cancel
Create function
CloudShell
Feedback

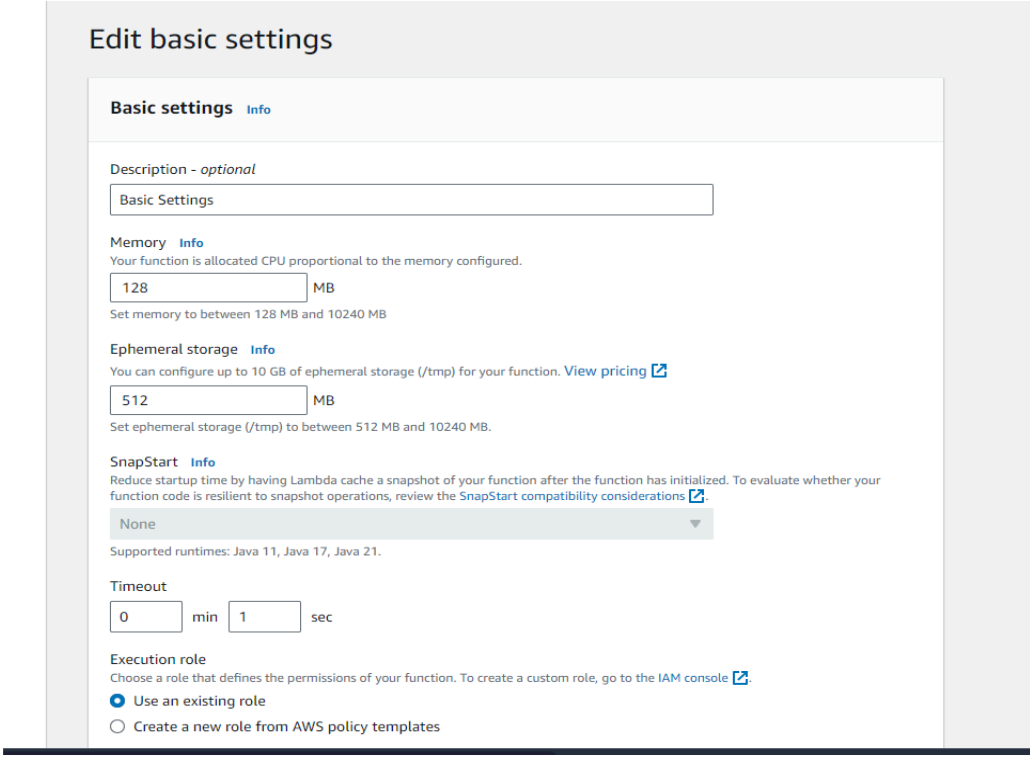

Lambda > Functions > myLambda
myLambda
Throttle
Copy ARN
Actions
Function overview Info
Export to Application Composer
Download
Diagram
Template
myLambda
Layers (0)
Add trigger
Add destination
Description
-
Last modified
4 minutes ago
Function ARN
arn:aws:lambda:eu-north-1:860015268757:function:myLambda
Function URL Info
-


Code
Test
Monitor
Configuration
Aliases
Versions
Code source Info
Upload from
File Edit Find View Go Tools Window
Test
Deploy
Go to Anything (Ctrl-P)
lambda_function
Environment Vari
Environment
myLambda
lambda_function.py
import json
def lambda_handler(event, context):
# TODO implement
return {
'statusCode': 200,
'body': json.dumps('Hello from Lambda!')
}

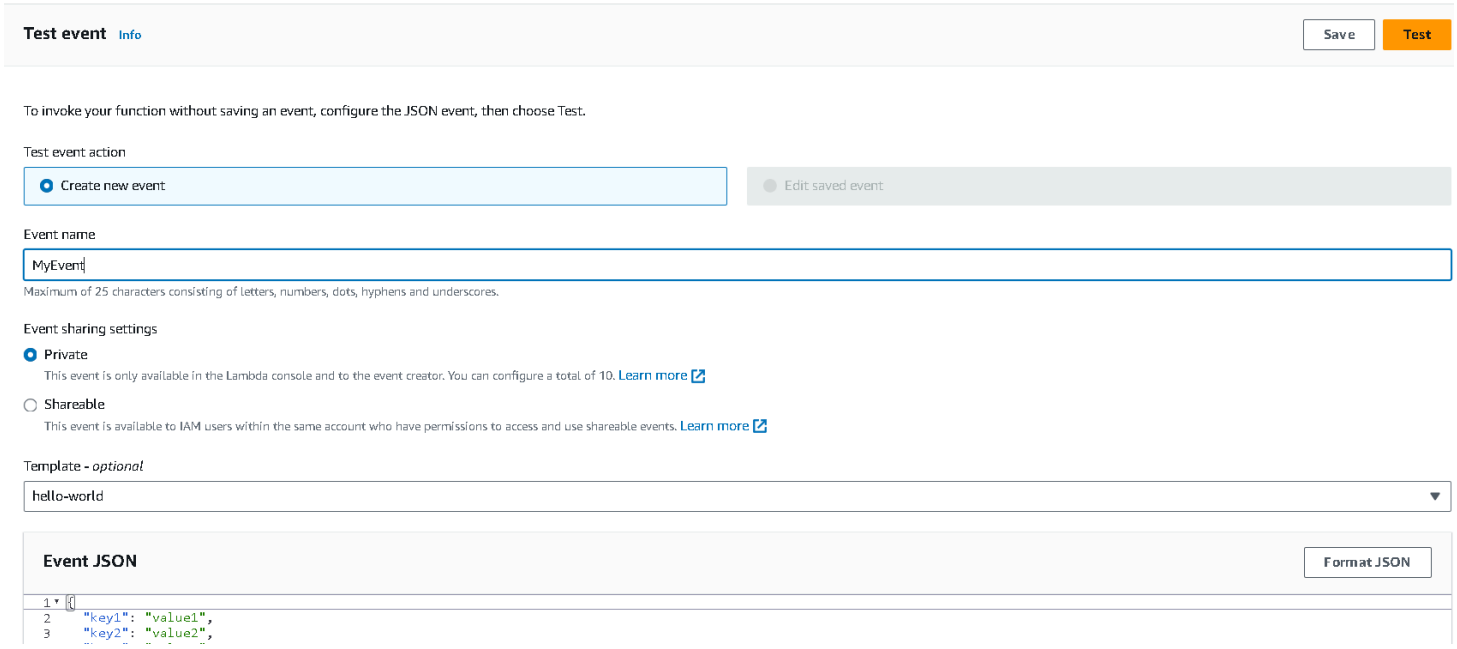
So See or Edit the basic settings go to configuration then click on edit general setting.



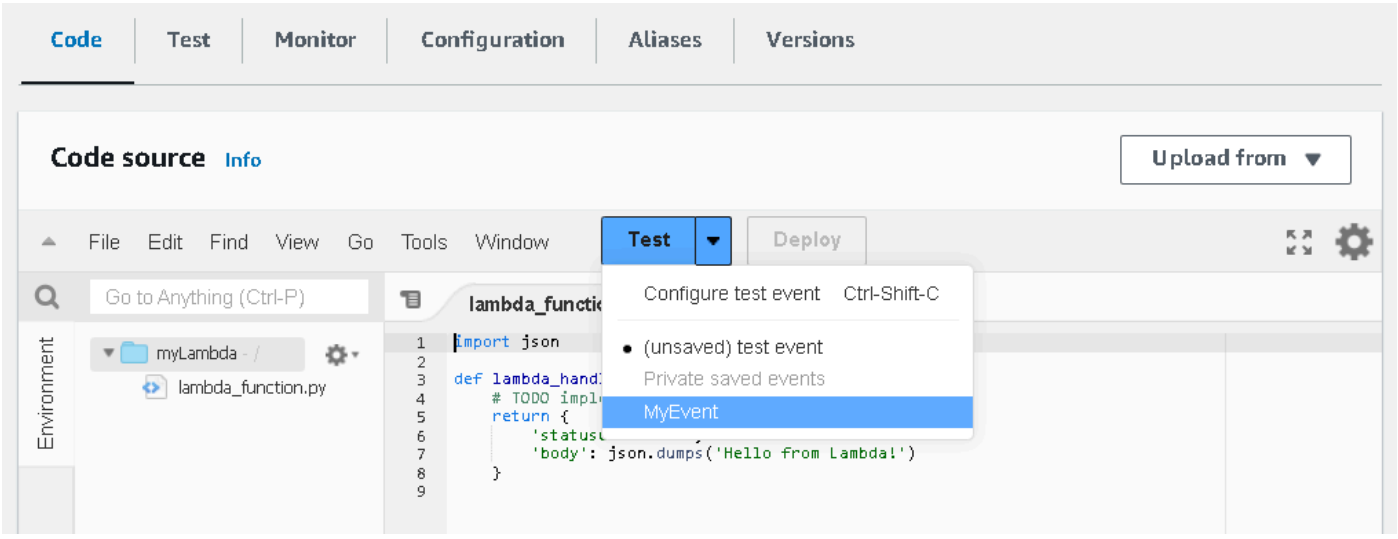

Here, you can enter a description and change Memory and Timeout. I've changed the Timeout period to 1 sec since that is sufficient for now.

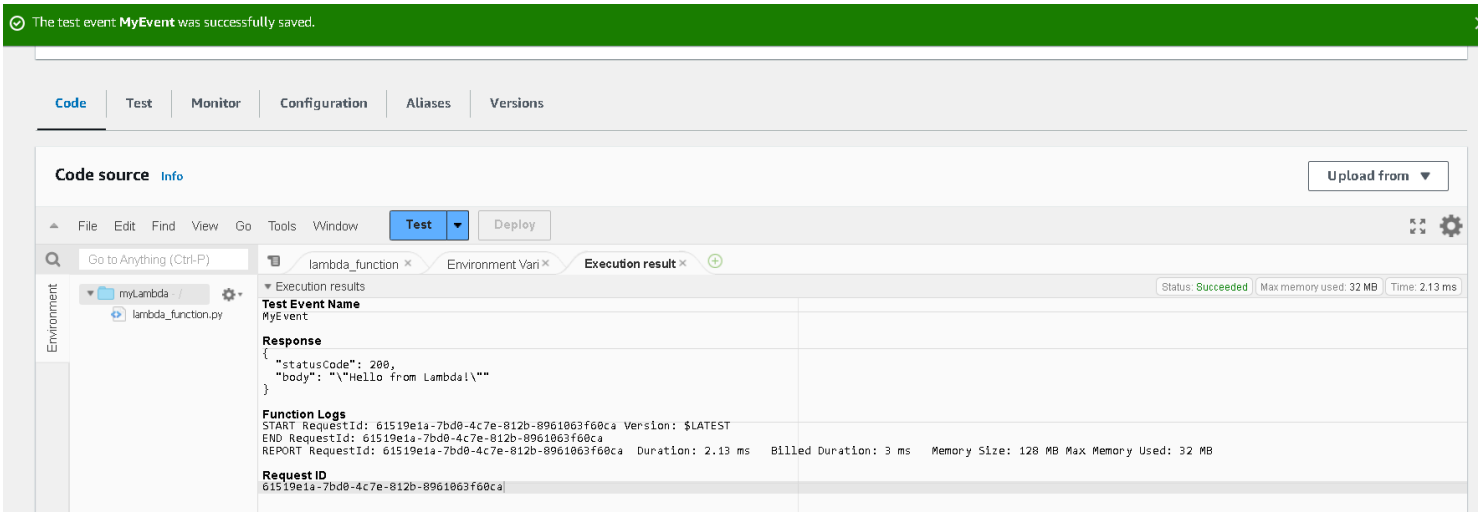


**Step 3:** Now Click on the Test tab then select Create a new event, give a name to the event and select Event Sharing to private, and select hello-world template.



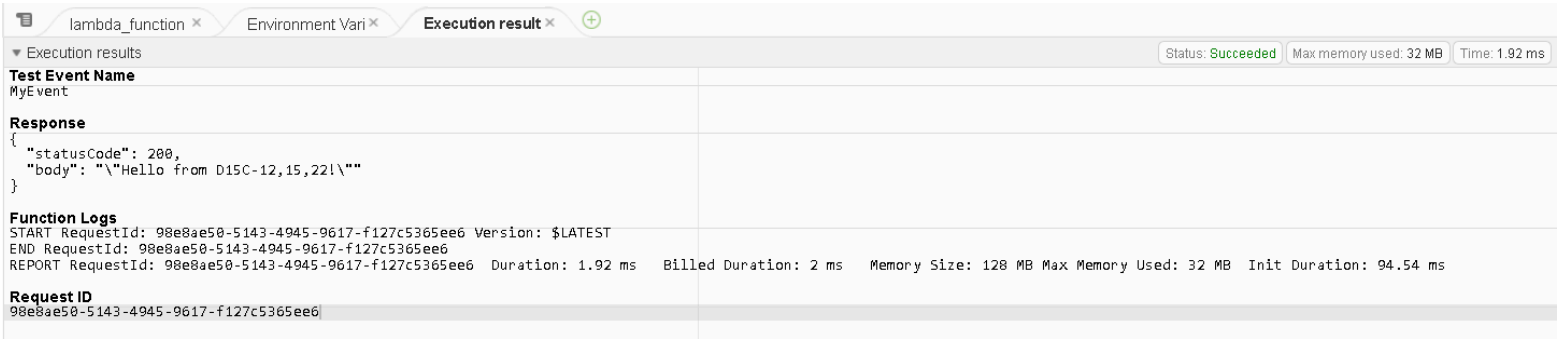

**Step 4:** Now In the Code section select the created event from the dropdown of test then click on test . You will see the below output.






**Step 5:** You can edit your lambda function code. I have changed the code to display the new String.


**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.




## Conclusion:

In this experiment, we successfully implemented an AWS Lambda function, covering all the key steps involved. Starting with the function's setup in Python, we configured essential settings such as adjusting the timeout to 1 second. A test event was then created, followed by deploying the function and verifying its output. We also made code modifications to the Lambda function, redeployed it, and observed the real-time effects of these changes. This hands-on experience highlighted the ease and adaptability of AWS Lambda for building serverless applications, enabling developers to concentrate on writing code while AWS handles infrastructure and scalability.