

**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.

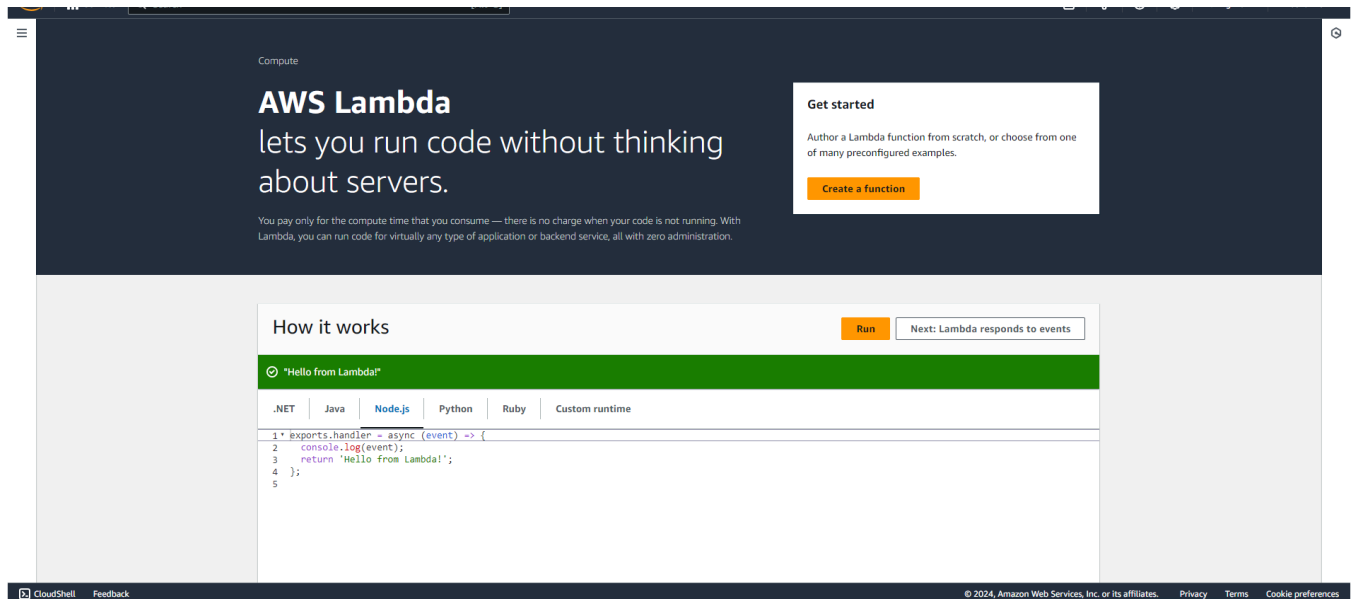
### AWS Lambda Functions

- **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 Account. 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 Existing(Lab Role) role to Create a new role with basic Lambda permissions.

The screenshot shows the 'Create function' wizard in the AWS Lambda console. The 'Author from scratch' option is selected. The 'Basic information' section includes:

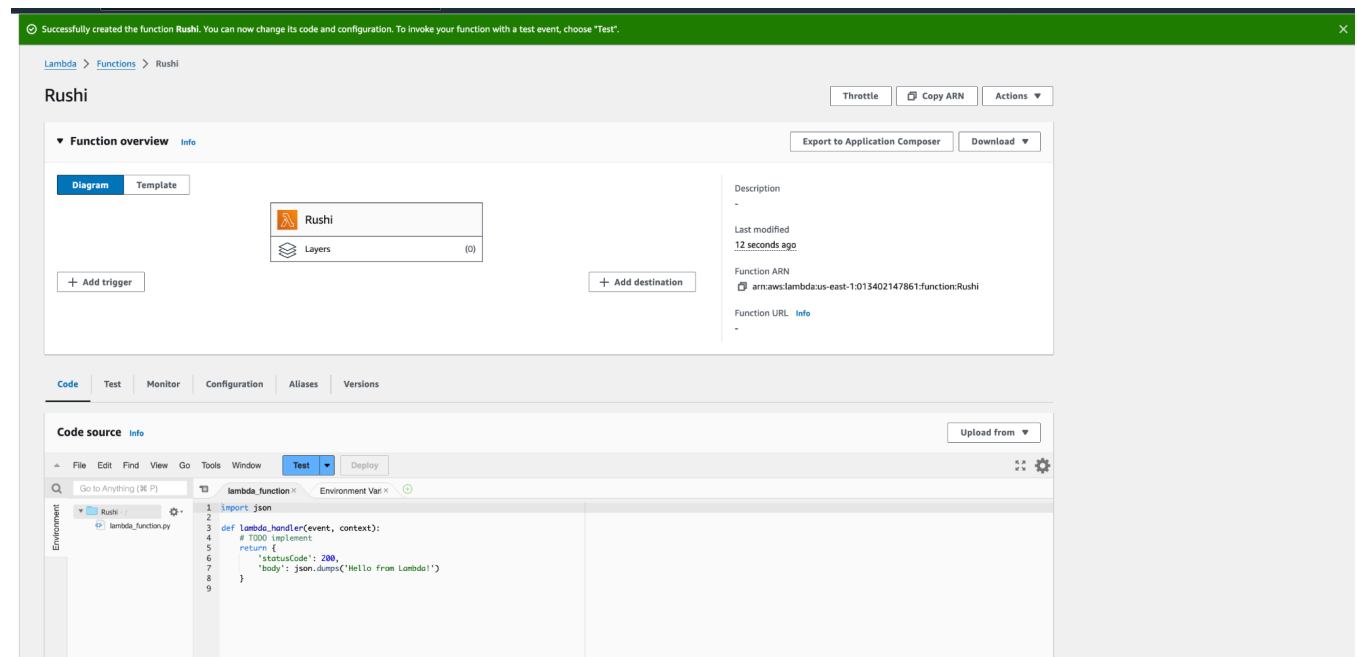
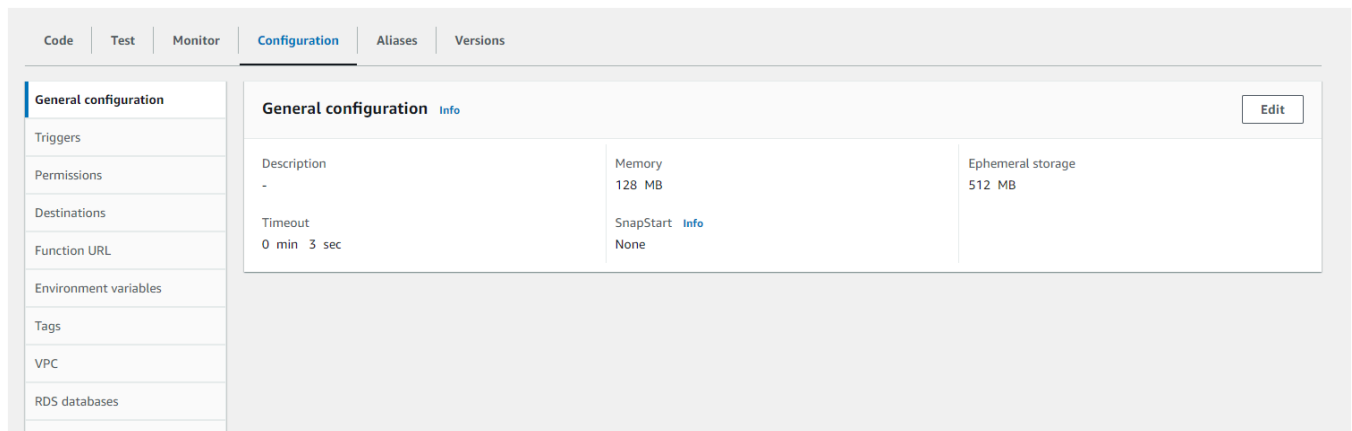
- Function name:** 'Rushi' (with a note: 'Enter a name that describes the purpose of your function. Use only letters, numbers, hyphens, or underscores with no spaces.')
- Runtime:** 'Python 3.12' (with a note: 'Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.')
- Architecture:** 'x86\_64' (with a note: 'Choose the instruction set architecture you want for your function code.')

The 'Permissions' section is expanded, showing the 'Change default execution role' options:

- Execution role:** 'Use an existing role' is selected (with a note: 'Choose a role that defines the permissions of your function. To create a custom role, go to the IAM console.').
- Existing role:** 'LabRole' is selected (with a note: '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.').

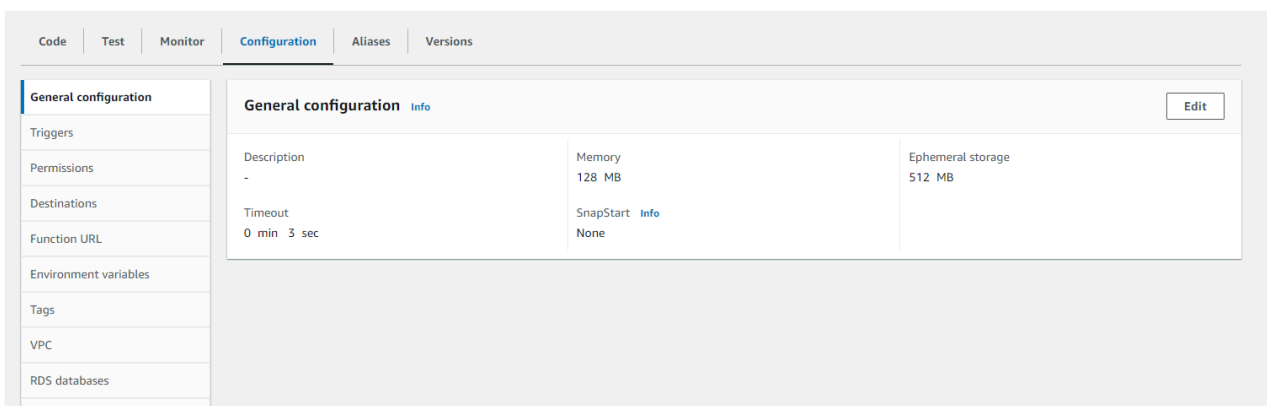
At the bottom, there are 'Cancel' and 'Create Function' buttons.

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.



The screenshot shows the 'Edit basic settings' page for a Lambda function. The 'Basic settings' tab is active. The 'Description - optional' field contains 'Basic Settings'. The 'Memory' is set to 128 MB. The 'Ephemeral storage' is set to 512 MB. The 'SnapStart' is set to 'None'. The 'Timeout' is set to 0 minutes and 1 second. The 'Execution role' is set to 'Use an existing role'. The 'Supported runtimes' are Java 11, Java 17, and Java 21.

**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

Info Tutorials

Learn how to implement common use cases in AWS Lambda.

Create a simple web app  
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.

[Learn more](#)  
Start tutorial

The screenshot shows the 'Test event' page for a function named 'Rushi'. The 'Test' tab is active. The 'Test event action' is set to 'Create new event'. The 'Event name' is 'myevent'. The 'Event sharing settings' are set to 'Private'. The 'Template - optional' is 'hello-world'. The 'Event JSON' is a JSON object with three keys: 'key1', 'key2', and 'key3', each with a corresponding value.

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

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  
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  
Format JSON

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

**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.

The screenshot shows the 'Execution results' page for a function named 'Rushi'. The 'Test' tab is active. The 'Test Event Name' is 'myevent'. The 'Response' is a JSON object with 'statusCode': 200 and 'body': 'Hello from Lambda!'. The 'Function Logs' show the execution details, including the request ID, request ID, and request ID.

File Edit Find View Go Tools Window Test Display

Go to Anything (X P)

Environment  
Rushi  
lambda\_function.py

Execution results  
Status: Succeeded | Max memory used: 32 MB | Time: 2.03 ms

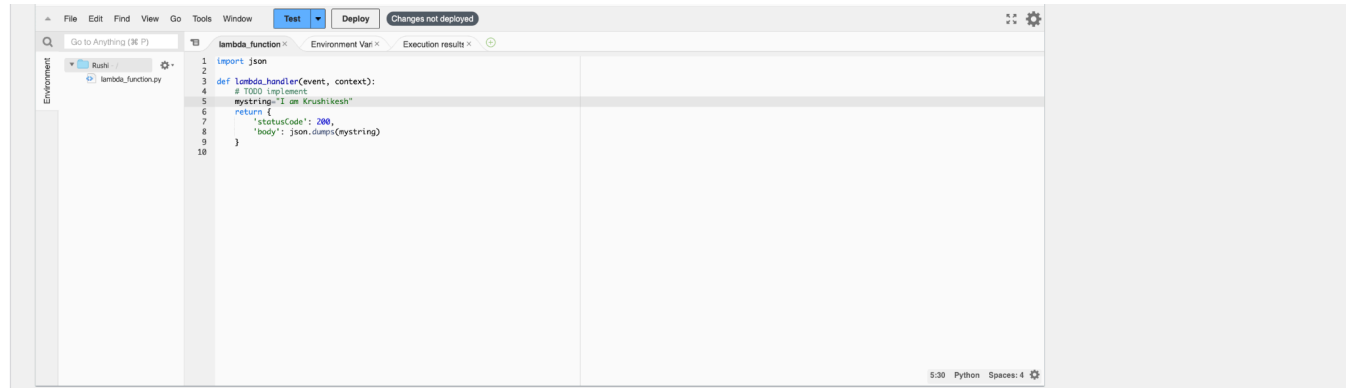
Test Event Name  
myevent

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

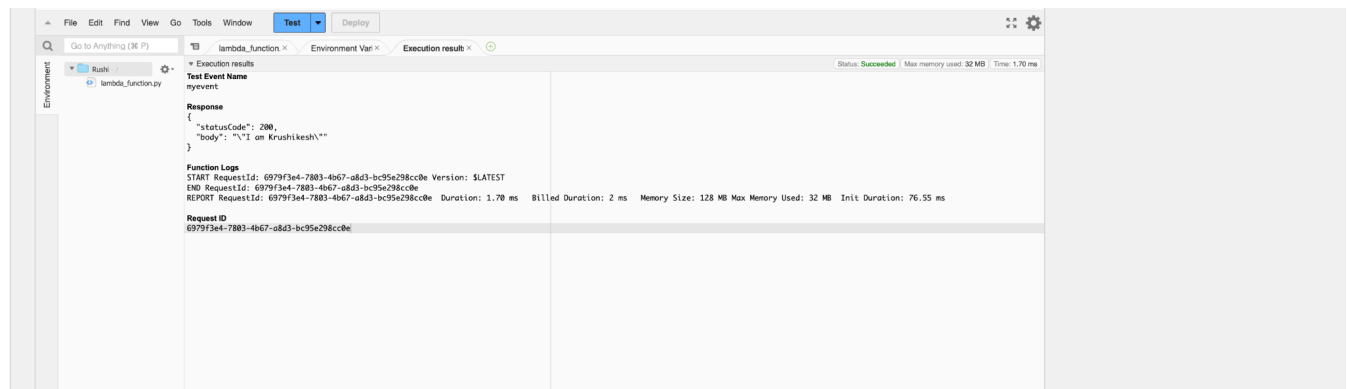
Function Logs  
START RequestId: 246db2c9-72b6-4254-afbb-04e82bea8b88 Version: SLATEST  
END RequestId: 246db2c9-72b6-4254-afbb-04e82bea8b88  
REPORT RequestId: 246db2c9-72b6-4254-afbb-04e82bea8b88 Duration: 2.03 ms Billed Duration: 3 ms Memory Size: 128 MB Max Memory Used: 32 MB Init Duration: 91.88 ms

Request ID  
246db2c9-72b6-4254-afbb-04e82bea8b88

**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.



**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 created an AWS Lambda function and walked through its essential steps. After setting up the function with Python, we configured the basic settings, including adjusting the timeout to 1 second. We then created a test event, deployed the function, and validated the output. Additionally, we modified the Lambda function's code and redeployed it to observe the changes in real-time.

This practical experience demonstrated the simplicity and flexibility of AWS Lambda in creating serverless applications, allowing you to focus on code while AWS manages the infrastructure and scaling.