

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.

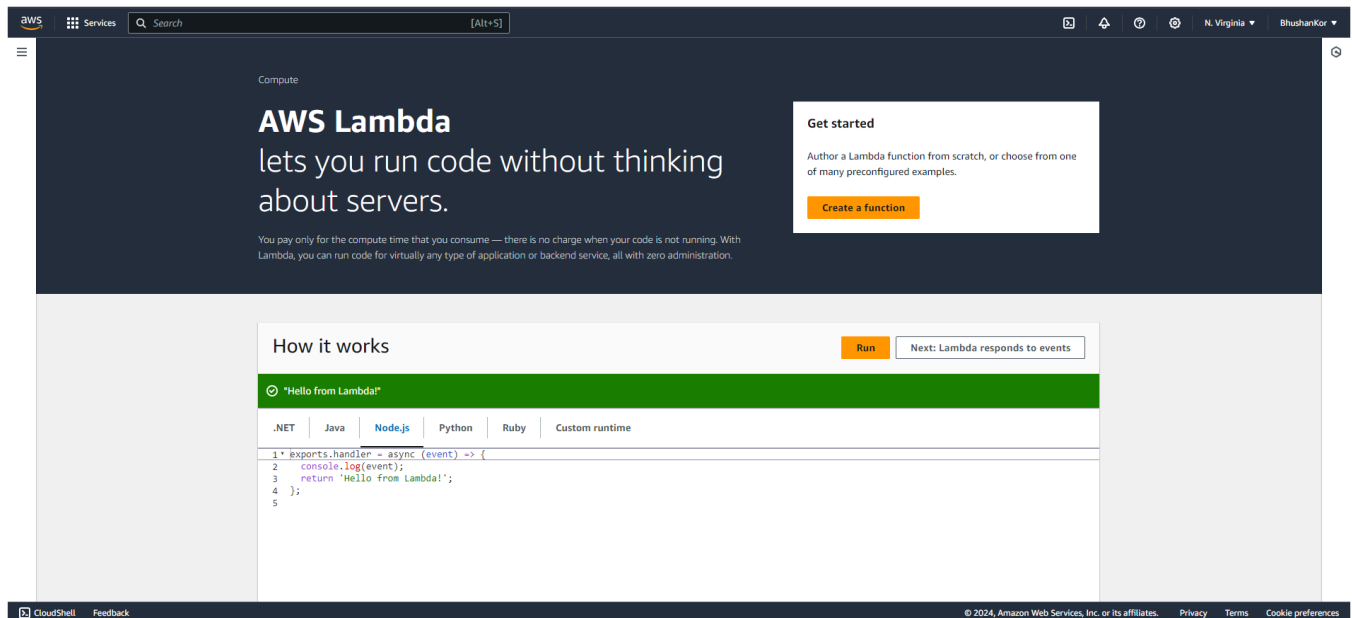
AWS Lambda Functions

- **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 Execution role to Create a new role with basic Lambda permissions.

The screenshot shows the 'Create function' wizard in the AWS Lambda console. It starts with three options: 'Author from scratch' (selected), 'Use a blueprint', and 'Container image'. Below these is the 'Basic information' section. It includes a 'Function name' field with the value 'lambda_lab11', a 'Runtime' dropdown set to 'Python 3.12', and an 'Architecture' dropdown set to 'x86_64'. The 'Runtime' section has a note that the console code editor supports only Node.js, Python, and Ruby. The 'Architecture' section has a note to choose the instruction set architecture for the function code.

▼ 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 `lambda_lab11-role-6rec818x`, with permission to upload logs to Amazon CloudWatch Logs.

► **Additional Configurations**

Use additional configurations to set up code signing, function URL, tags, and Amazon VPC access for your function.

Cancel **Create function**

☑ Successfully created the function `lambda_lab11`. You can now change its code and configuration. To invoke your function with a test event, choose "Test".

Lambda > Functions > lambda_lab11

lambda_lab11

Throttle Copy ARN Actions ▼

▼ Function overview Info

Export to Application Composer Download ▼

Diagram Template

lambda_lab11

Layers (0)

+ Add trigger

+ Add destination

Description

-

Last modified

1 second ago

Function ARN

arn:aws:lambda:us-east-1:017820672175:function:lambda_lab11

Function URL [Info](#)

-

☑ Successfully created the function `lambda_lab11`. You can now change its code and configuration. To invoke your function with a test event, choose "Test".

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_lab11 - /

lambda_function.py

```
1 import json
2
3 def lambda_handler(event, context):
4     # TODO: Implement
5     return {
6         'statusCode': 200,
7         'body': json.dumps('Hello from Lambda!')
8     }
9
```

1:1 Python Spaces: 4

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

The screenshot shows the AWS Lambda Configuration page with the 'Configuration' tab selected. The left sidebar lists various configuration options, with 'General configuration' highlighted. The main content area displays the 'General configuration' settings, including Description, Memory, Ephemeral storage, Timeout, SnapStart, and an 'Edit' button.

General configuration Info		
Description	Memory	Ephemeral storage
-	128 MB	512 MB
Timeout	SnapStart Info	
0 min 3 sec	None	

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.

The screenshot shows the 'Basic settings' dialog box for an AWS Lambda function. It includes fields for Description, Memory, Ephemeral storage, SnapStart, Timeout, and Execution role. The Timeout is set to 1 second, and the Execution role is 'service-role/lambda_lab11-role-6rec818x'.

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.
service-role/lambda_lab11-role-6rec818x

[View the lambda_lab11-role-6rec818x role](#) on the IAM console.

Cancel Save

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 'Test' tab in the AWS Lambda console. At the top, there are tabs for 'Code', 'Test', 'Monitor', 'Configuration', 'Aliases', and 'Versions'. Below these, the 'Test event' section has 'Save' and 'Test' buttons. A message states: 'To invoke your function without saving an event, configure the JSON event, then choose Test.' Under 'Test event action', 'Create new event' is selected. The 'Event name' field contains 'lab11Event'. Below it, a note says 'Maximum of 25 characters consisting of letters, numbers, dots, hyphens and underscores.' Under 'Event sharing settings', 'Private' is selected with a note: 'This event is only available in the Lambda console and to the event creator. You can configure a total of 10. [Learn more](#)'. 'Shareable' is also an option. The 'Template - optional' dropdown is set to 'hello-world'. At the bottom, the 'Event JSON' section has a 'Format JSON' button and a text area containing a JSON object:

```
{ 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 'Code' tab in the AWS Lambda console. At the top, there are tabs for 'Code', 'Test', 'Monitor', 'Configuration', 'Aliases', and 'Versions'. Below these, the 'Code source' section has an 'Upload from' button. A menu is open over the 'Test' button, showing 'Configure test event' and 'Private saved events' with a list containing 'lab11Event'. The 'Execution results' section shows the following details:
Test Event Name: lab11Event
Status: Succeeded | Max memory used: 32 MB | Time: 2.07 ms
Response:

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


Function Logs:

```
START RequestId: d6c1eb5a-4c72-4c85-8694-d3c221c14fee Version: $LATEST  END RequestId: d6c1eb5a-4c72-4c85-8694-d3c221c14fee  REPORT RequestId: d6c1eb5a-4c72-4c85-8694-d3c221c14fee Duration: 2.07 ms Billed Duration: 3 ms Memory Size: 128 MB Max Memory Used: 32 MB Init Duration: 94.30 ms
```


Request ID: d6c1eb5a-4c72-4c85-8694-d3c221c14fee

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.



The screenshot shows the AWS Lambda console with the 'lambda_function' tab selected. The code editor displays the following Python code:

```
1 import json
2
3 def lambda_handler(event, context):
4     # TODO implement
5     return {
6         'statusCode': 200,
7         'body': json.dumps('Brijesh from D15C - Roll No. 48!')}
8
9
```

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.



The screenshot shows the 'Execution results' tab for the 'lambda_function'. The status is 'Succeeded'. The test event name is 'lab11Event'. The response is:

```
{
  "statusCode": 200,
  "body": "\"Brijesh from D15C - Roll No. 48!\""
}
```

The function logs show the following details:

```
START RequestId: bf7e8aa5-7e14-413d-8c72-64169e52dac0 Version: $LATEST
END RequestId: bf7e8aa5-7e14-413d-8c72-64169e52dac0
REPORT RequestId: bf7e8aa5-7e14-413d-8c72-64169e52dac0 Duration: 2.37 ms Billed Duration: 3 ms Memory Size: 128 MB Max Memory Used: 32 MB Init Duration: 113.23 ms
```

The request ID is 'bf7e8aa5-7e14-413d-8c72-64169e52dac0'.

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.