

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

**Theory:**

**AWS Lambda**

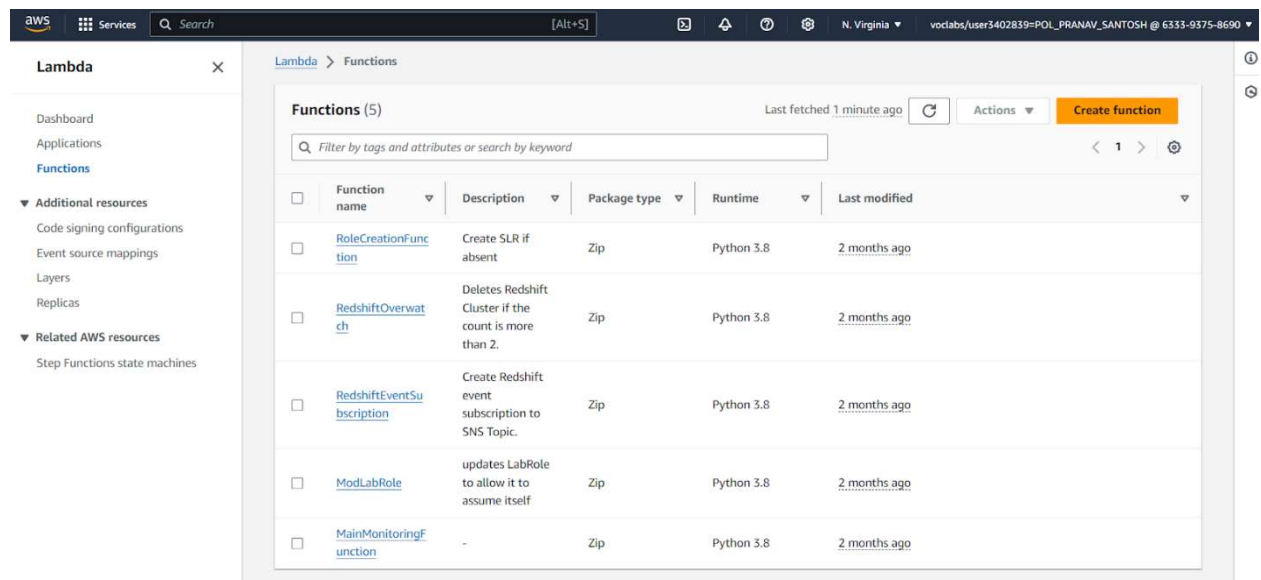
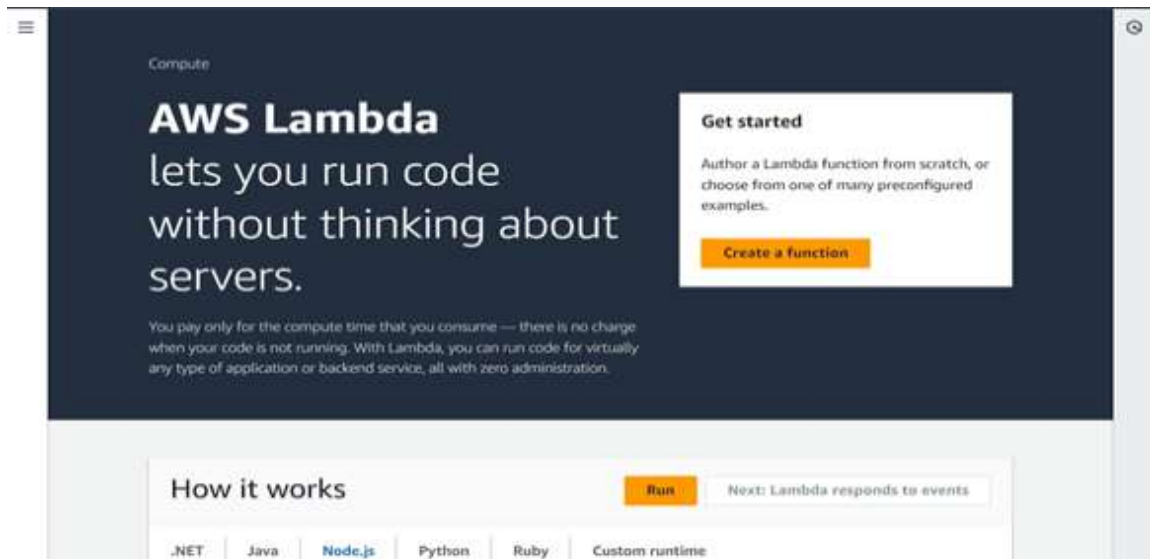
- AWS Lambda is a serverless computing service provided by Amazon Web Services (AWS). Users of AWS Lambda create functions, self-contained applications written in one of the supported languages and runtimes, and upload them to AWS Lambda, which executes those functions in an efficient and flexible manner.
- The Lambda functions can perform any kind of computing task, from serving web pages and processing streams of data to calling APIs and integrating with other AWS services. The concept of “serverless” computing refers to not needing to maintain your own servers to run these functions.
- AWS Lambda is a fully managed service that takes care of all the infrastructure for you. And so “serverless” doesn’t mean that there are no servers involved: it just means that the servers, the operating systems, the network layer and the rest of the infrastructure have already been taken care of so that you can focus on writing application code.

**Features of AWS Lambda**

- AWS Lambda easily scales the infrastructure without any additional configuration. It reduces the operational work involved.
- It offers multiple options like AWS S3, CloudWatch, DynamoDB, API Gateway, Kinesis, CodeCommit, and many more to trigger an event.
- You don’t need to invest upfront. You pay only for the memory used by the lambda function and minimal cost on the number of requests hence cost-efficient.
- AWS Lambda is secure. It uses AWS IAM to define all the roles and security policies.
- It offers fault tolerance for both services running the code and the function. You do not have to worry about the application down

## Steps to create an AWS Lambda function

1. Open up the Lambda Console and click on the Create button. Be mindful of where you create your functions since Lambda is region-dependent.



## 2. Attach CloudWatch Logs permissions:

- In the "Permissions" step, search for the policy called [AWSLambdaBasicExecutionRole](#).
- Select this policy, which gives your Lambda function permission to write logs to CloudWatch.
- Click **Next**.

### Use case

Allow an AWS service like EC2, Lambda, or others to perform actions in this account.

Service or use case

Lambda ▼

Choose a use case for the specified service.

Use case

☒ **Lambda**  
Allows Lambda functions to call AWS services on your behalf.

[IAM](#) > [Roles](#) > Create role

Step 1  
[Select trusted entity](#)

Step 2  
[Add permissions](#)

Step 3  
**Name, review, and create**

## Name, review, and create

### Role details

**Role name**  
Enter a meaningful name to identify this role.

lambda-user

Maximum 64 characters. Use alphanumeric and '+=, @-./\_{' characters.

**Description**  
Add a short explanation for this role.

Allows Lambda functions to call AWS services on your behalf.

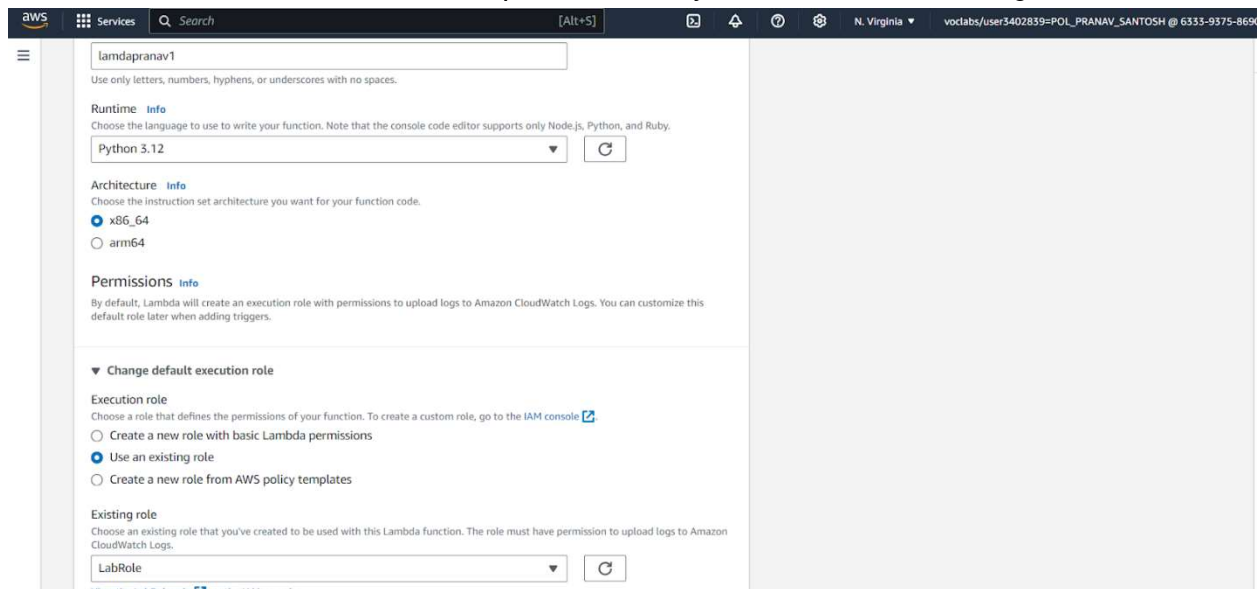
Maximum 1000 characters. Use letters (A-Z and a-z), numbers (0-9), tabs, new lines, or any of the following characters: \_+=, @-./\_{'#\$%^&\*()';:"'`

**Step 1: Select trusted entities**

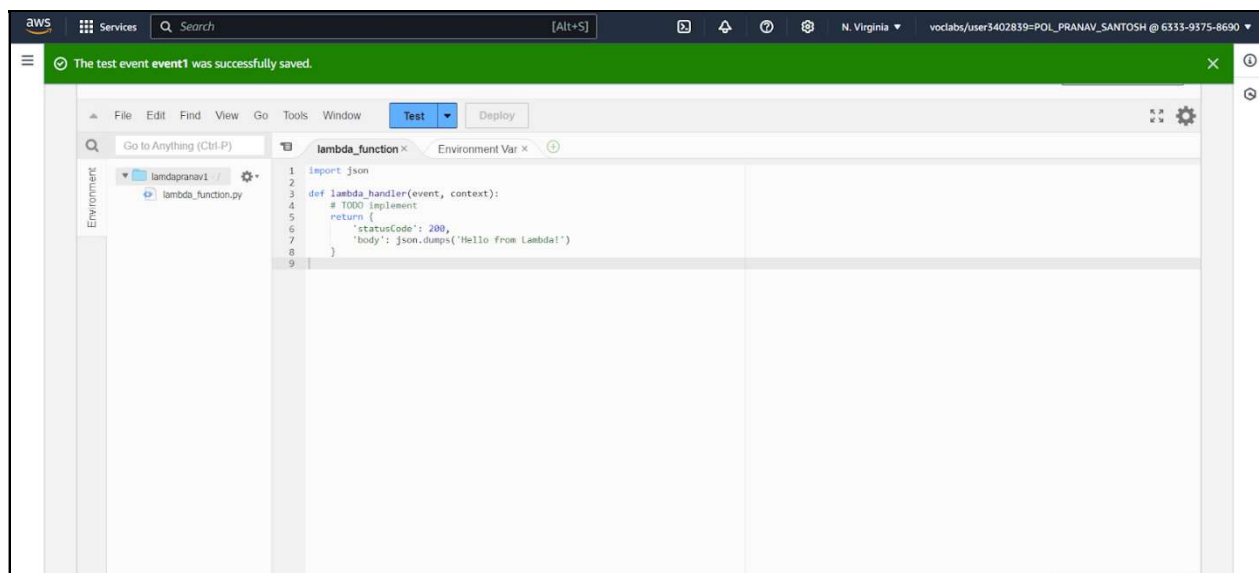
Edit

## 2. Choose to create a function from scratch or use a blueprint, i.e templates defined by AWS for you with all configuration presets required for the most common use cases.

Then, choose a runtime env for your function, under the dropdown, you can see all the options AWS supports, Python, Nodejs, .NET and Java being the most popular ones. After that, choose to create a new role with basic Lambda permissions if you don't have an existing one.



## 2. This process will take a while to finish and after that, you'll get a message that your function was successfully created



## Edit Basic Settings:

- On the function's **Configuration** tab, locate the **Basic settings** section

The screenshot shows the AWS Lambda console's Configuration tab for a function. The left sidebar contains a menu with options: General configuration, Triggers, Permissions, Destinations, Function URL, Environment variables, Tags, and VPC. The main content area is titled 'General configuration' with an 'Info' link and an 'Edit' button. It displays the following settings:

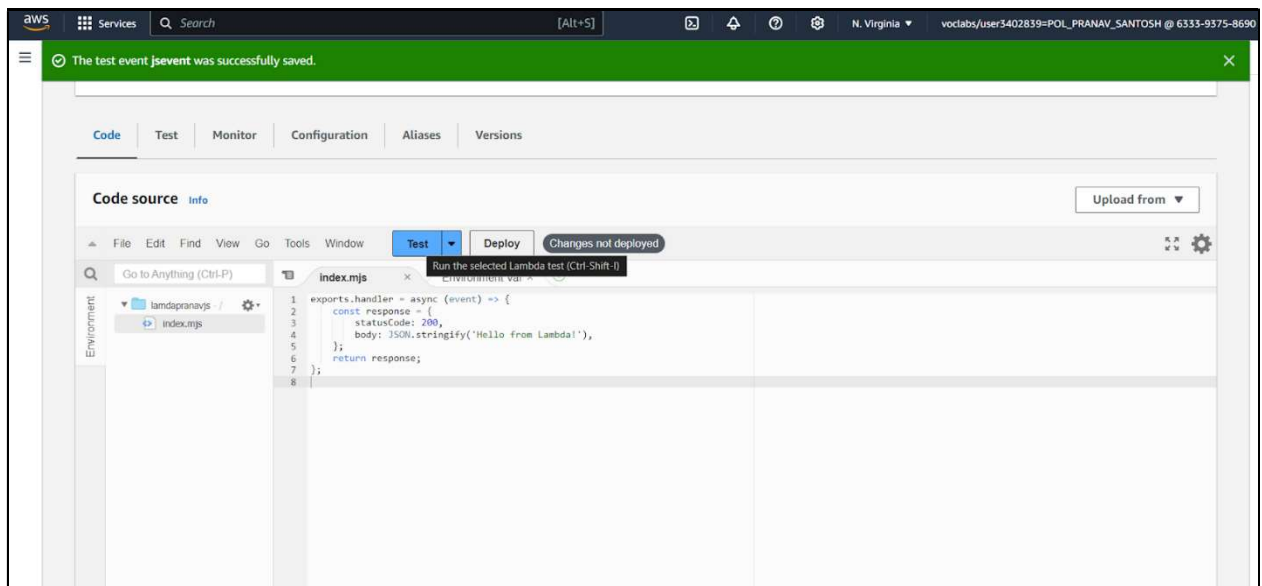
| Setting           | Value       |
|-------------------|-------------|
| Description       | -           |
| Memory            | 128 MB      |
| Ephemeral storage | 512 MB      |
| Timeout           | 0 min 3 sec |
| SnapStart         | None        |

## Configuring test event which triggers when the function is tested

The screenshot shows the 'Configure test event' dialog box. It includes the following sections:

- Test event action:** Radio buttons for 'Create new event' (selected) and 'Edit saved event'.
- Event name:** A text input field containing 'event1'. Below it, a note states: 'Maximum of 25 characters consisting of letters, numbers, dots, hyphens and underscores.'
- Event sharing settings:** Radio buttons for 'Private' (selected) and 'Shareable'. Descriptions and 'Learn more' links are provided for each.
- Template - optional:** A dropdown menu currently showing 'hello-world'.
- Event JSON:** A large text area for entering the event JSON, with a 'Format JSON' button to the right.

At the bottom of the dialog are 'Cancel', 'Invoke', and 'Save' buttons.



## Conclusion:

AWS Lambda is a serverless computing service that allows you to run code without managing servers, making it highly scalable, cost-effective, and easy to use. It automatically manages the compute resources, executes your code in response to specific events such as API calls, file uploads, or database updates, and scales based on the demand.

