

## **Advance DevOps Lab - Practical 11**

### **Varun Gupta - D15 B - Roll no. 20**

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

#### **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.

##### **Packaging Functions -**

Lambda functions need to be packaged and sent to AWS. This is usually a process of compressing the function and all its dependencies and uploading it to an S3 bucket. And letting AWS know that you want to use this package when a specific event takes place. To help us with this process we use the Serverless Stack Framework (SST).

## **Execution Model -**

The container (and the resources used by it that runs our function) is managed completely by AWS. It is brought up when an event takes place and is turned off if it is not being used. If additional requests are made while the original event is being served, a new container is brought up to serve a request. This means that if we are undergoing a usage spike, the cloud provider simply creates multiple instances of the container with our function to serve those requests. This has some interesting implications. Firstly, our functions are effectively stateless. Secondly, each request (or event) is served by a single instance of a Lambda function. This means that you are not going to be handling concurrent requests in your code. AWS brings up a container whenever there is a new request. It does make some optimizations here. It will hang on to the container for a few minutes (5 - 15 mins depending on the load) so it can respond to subsequent requests without a cold start.

## **Stateless Functions**

The above execution model makes Lambda functions effectively stateless. This means that every time your Lambda function is triggered by an event it is invoked in a completely new environment. You don't have access to the execution context of the previous event. However, due to the optimization noted above, the actual Lambda function is invoked only once per container instantiation. Recall that our functions are run inside containers. So when a function is first invoked, all the code in our handler function gets executed and the handler function gets invoked. If the container is still available for subsequent requests, your function will get invoked and not the code around it.

## **Common Use Cases for Lambda -**

Due to Lambda's architecture, it can deliver great benefits over traditional cloud computing setups for applications where:

1. Individual tasks run for a short time;
2. Each task is generally self-contained;
3. There is a large difference between the lowest and highest levels in the workload of the application.

Some of the most common use cases for AWS Lambda that fit these criteria are:

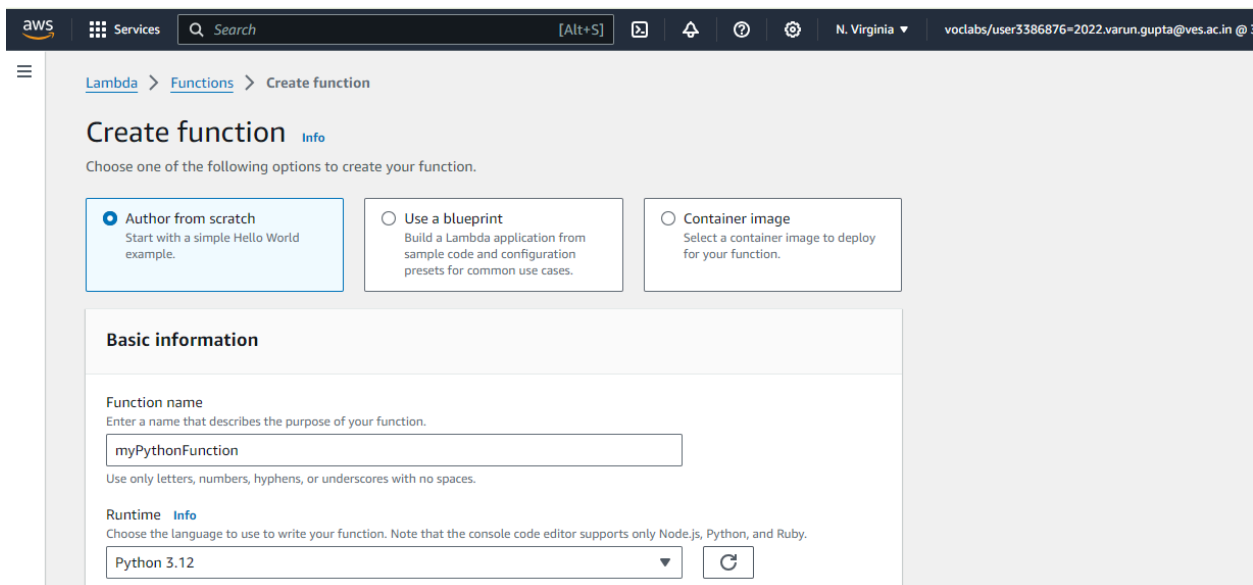
**Scalable APIs.** When building APIs using AWS Lambda, one execution of a Lambda function can serve a single HTTP request. Different parts of the API can be routed to different Lambda functions via Amazon API Gateway. AWS Lambda automatically scales individual functions according to the demand for them, so different parts of your

API can scale differently according to current usage levels. This allows for cost-effective and flexible API setups.

Data processing. Lambda functions are optimized for event-based data processing. It is easy to integrate AWS Lambda with data sources like Amazon DynamoDB and trigger a Lambda function for specific kinds of data events.

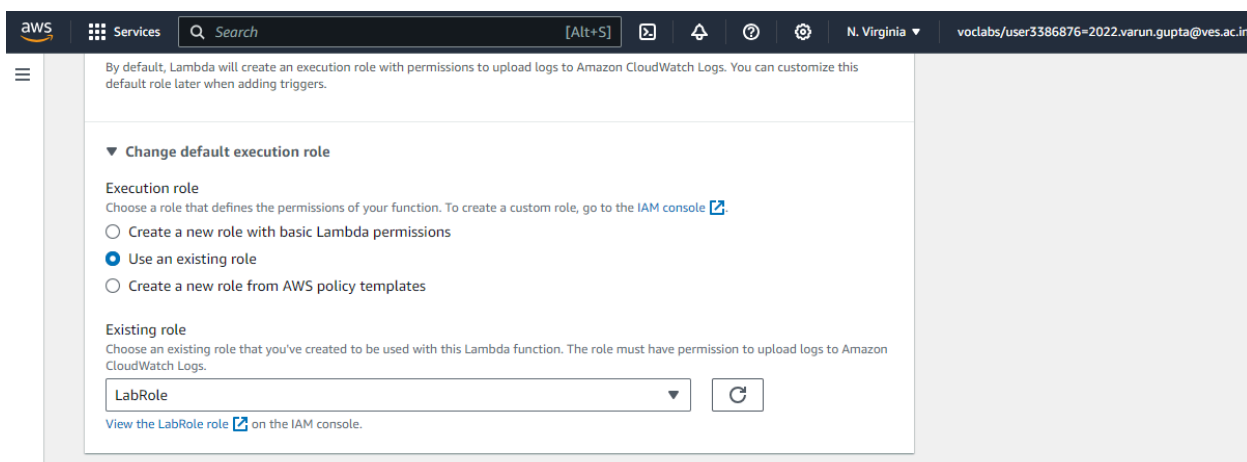
## Steps:

### 1. Open up the Lambda Console and click on the Create button.



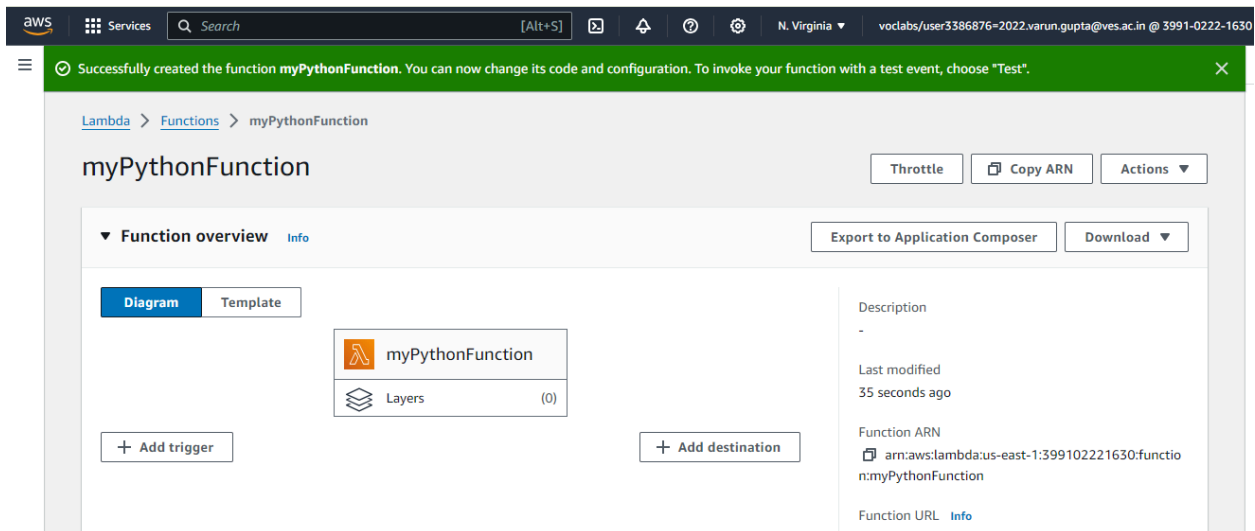
The screenshot shows the AWS Lambda 'Create function' page. At the top, there's a navigation bar with the AWS logo, 'Services', a search bar, and user information. The main heading is 'Create function' with an 'Info' link. Below the heading, it says 'Choose one of the following options to create your function.' There are three radio button options: 'Author from scratch' (selected), 'Use a blueprint', and 'Container image'. The 'Author from scratch' option has a description: 'Start with a simple Hello World example.' Below these options is a section titled 'Basic information'. It contains a 'Function name' field with the value 'myPythonFunction' and a note: 'Enter a name that describes the purpose of your function. Use only letters, numbers, hyphens, or underscores with no spaces.' Below that is a 'Runtime' dropdown menu set to 'Python 3.12' with a refresh button. A note states: 'Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.'

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.



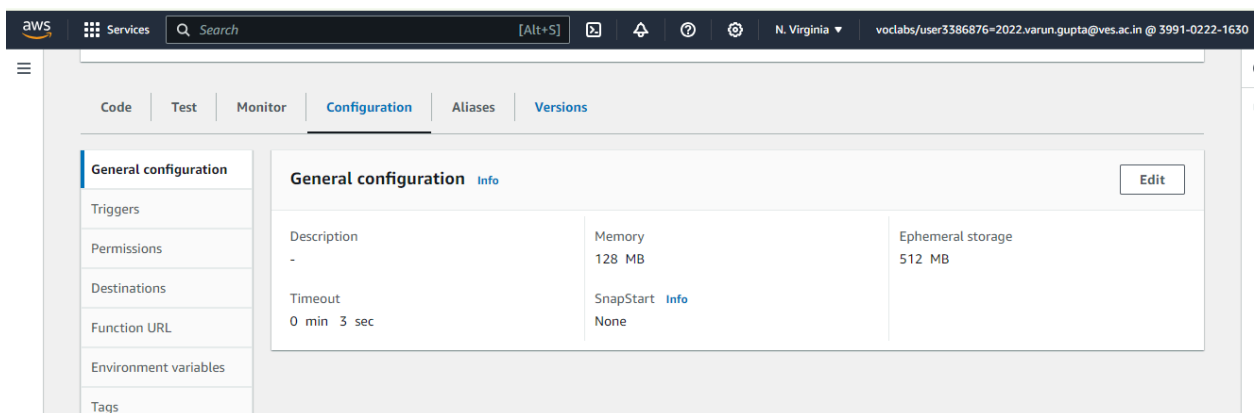
The screenshot shows the 'Change default execution role' section of the AWS Lambda console. It starts with a note: 'By default, Lambda will create an execution role with permissions to upload logs to Amazon CloudWatch Logs. You can customize this default role later when adding triggers.' Below this is a section titled 'Change default execution role'. It has a sub-section 'Execution role' with the instruction: 'Choose a role that defines the permissions of your function. To create a custom role, go to the IAM console.' There are three radio button options: 'Create a new role with basic Lambda permissions', 'Use an existing role' (selected), and 'Create a new role from AWS policy templates'. Below these is a sub-section 'Existing role' with the instruction: '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.' There is a dropdown menu showing 'LabRole' and a refresh button. At the bottom, there is a link: 'View the LabRole role on the IAM console.'

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



4. To change the configuration, open up the Configuration tab and under General Configuration, choose Edit.

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.



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.

LabRole

[View the LabRole role](#) on the IAM console.

Cancel Save

**5. You can make changes to your function inside the code editor. You can also upload a zip file of your function or upload one from an S3 bucket if needed. Press Ctrl + S to save the file and click Deploy to deploy the changes.**

The test event myTestEvent was successfully saved.

**Code source** [Info](#)

Upload from

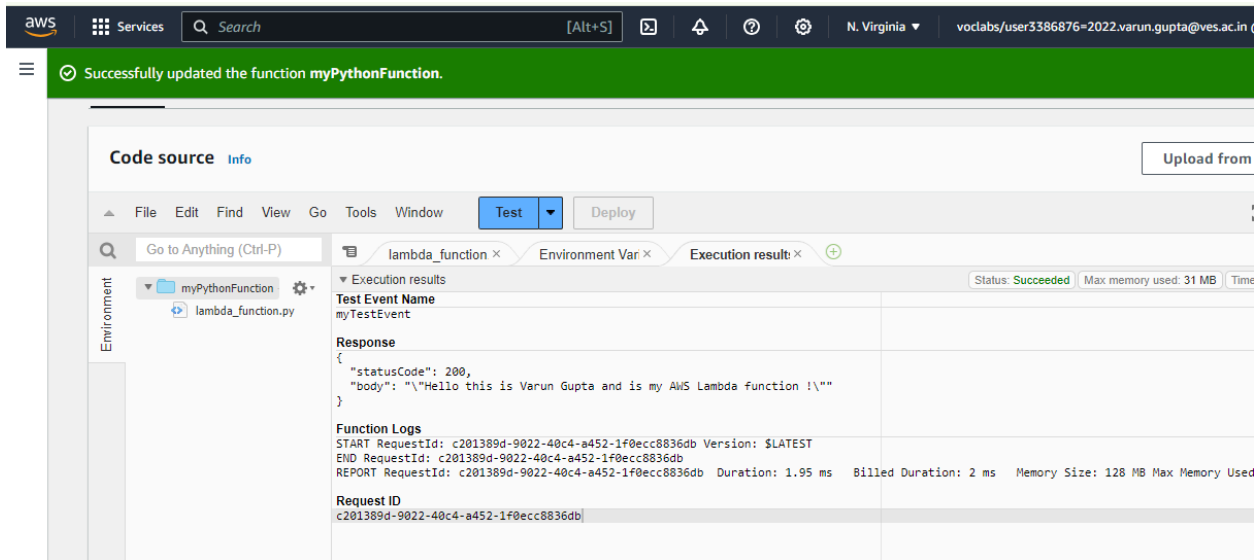
File Edit Find View Go Tools Window Test Deploy Changes not deployed

Go to Anything (Ctrl-P)

Environment

- myPythonFunction
- 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 this is Varun Gupta and is my AWS Lambda function !')}
8
9
```



6. Click on Test and you can change the configuration, like so. If you do not have anything in the request body, it is important to specify two curly braces as valid JSON, so make sure they are there.

### Configure test event

A test event is a JSON object that mocks the structure of requests emitted by AWS services to invoke a Lambda function. Use it to see the function's invocation result.

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

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

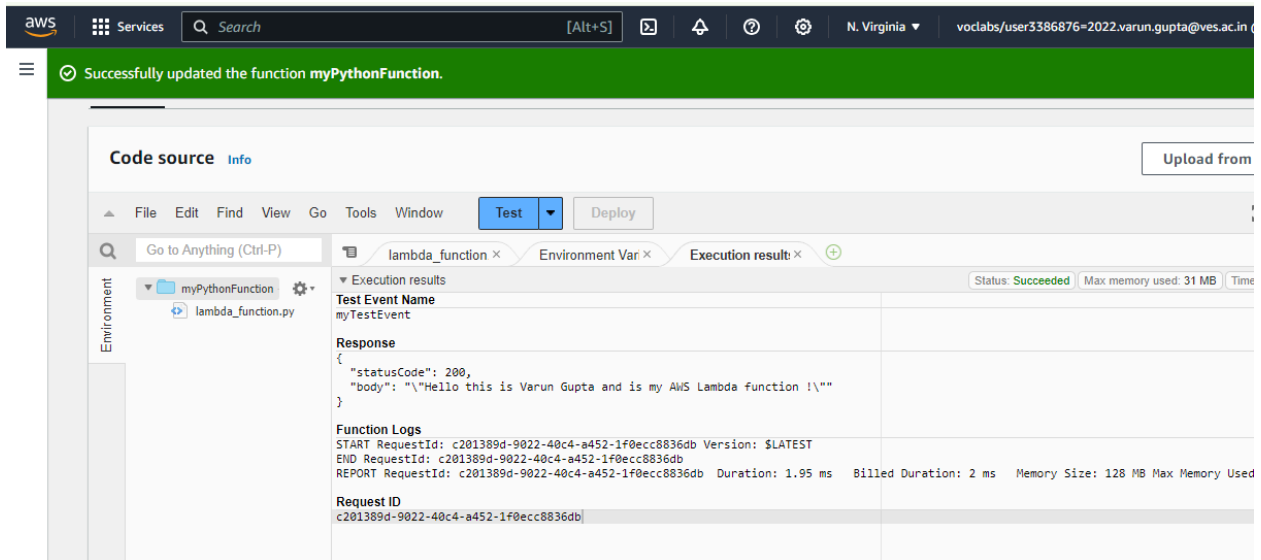
hello-world

Cancel

Invoke

Save

7. Now click on Test and you should be able to see the results.



## Conclusion:

In this experiment, we learned about AWS Lambda and using it to create, deploy and test serverless functions in the Cloud.