Rujuta Medhi D15A-28

EXPERIMENT:11

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.

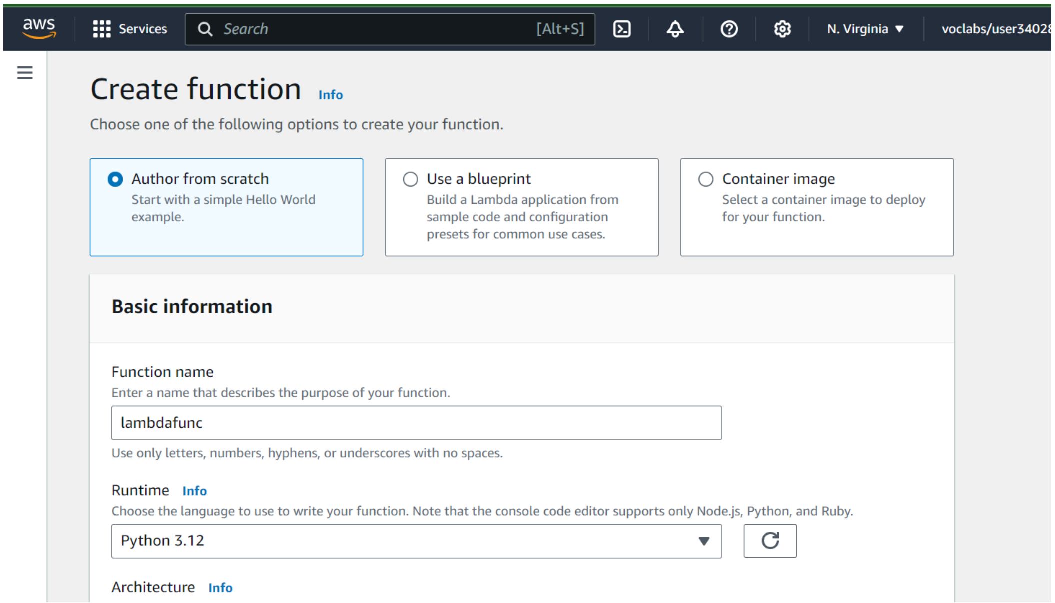
The concept of "serverless" computing refers to not needing to maintain your own servers to run these functions. AWS Lambda is a fully managed service that takes care of all the infrastructure for you. And so "serverless" doesn't mean that there are no servers involved: it just means that the servers, 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.

Implementation:
Steps to create an AWS Lambda function

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



Create function Info

Choose one of the following options to create your function.

● Author from scratch
Start with a simple Hello World example.

○ Use a blueprint
Build a Lambda application from sample code and configuration presets for common use cases.

○ Container image
Select a container image to deploy for your function.

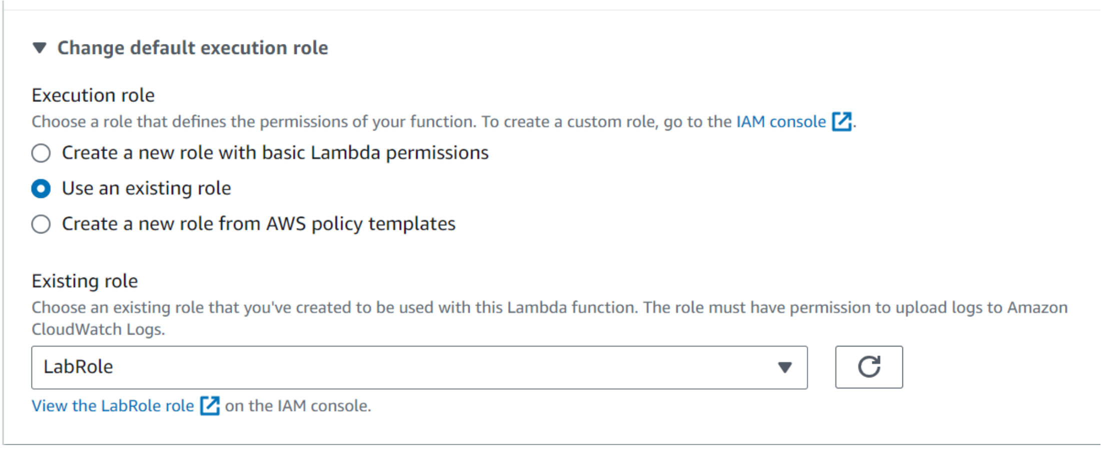
Basic information

Function name
Enter a name that describes the purpose of your function.
lambdafunc
Use only letters, numbers, hyphens, or underscores with no spaces.

Runtime Info
Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.
Python 3.12

Architecture Info

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.



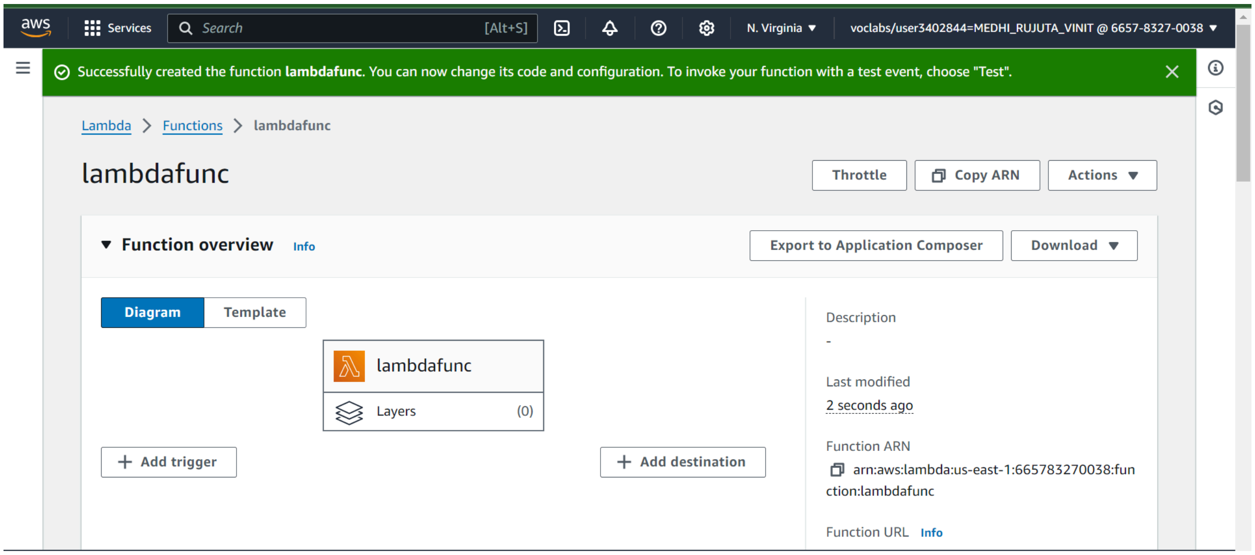
▼ Change default execution role

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

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.

3. Lambda function is created.



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

Lambda > Functions > lambdafunc

lambdafunc

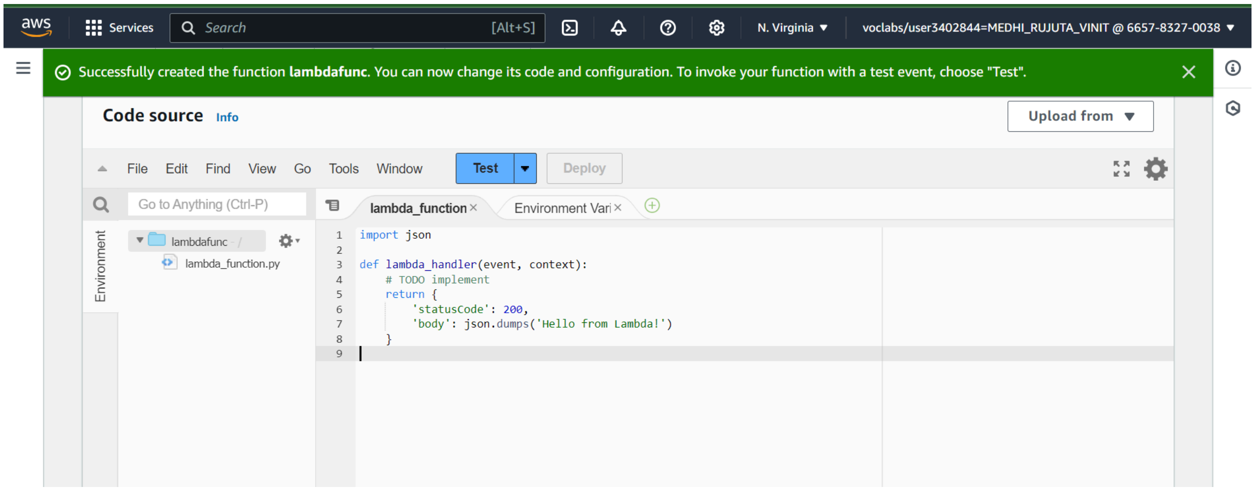
Description
-

Last modified
2 seconds ago

Function ARN
arn:aws:lambda:us-east-1:665783270038:function:lambdafunc

Function URL Info

4. This is the source code



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

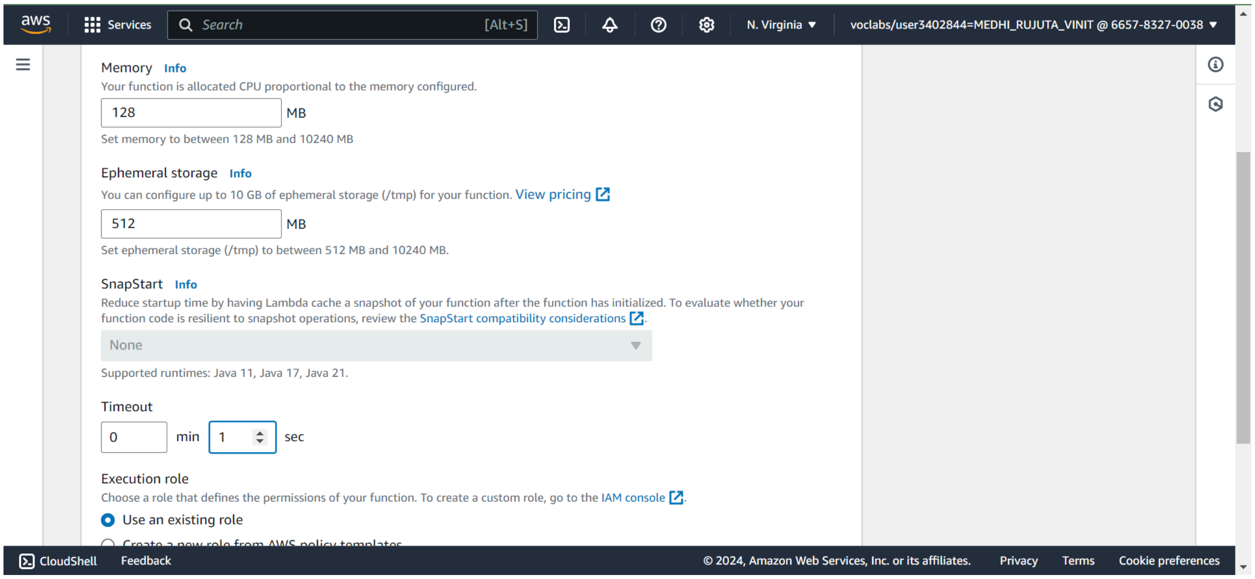
```
import json

def lambda_handler(event, context):
    # TODO implement
    return {
        'statusCode': 200,
        'body': json.dumps('Hello from Lambda!')
    }

```

5. 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.


aws
Services
Search
[Alt+S]
N. Virginia
voclabs/user3402844=MEDHI_RUJUTA_VINIT @ 6657-8327-0038
CloudShell
Feedback
© 2024, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

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

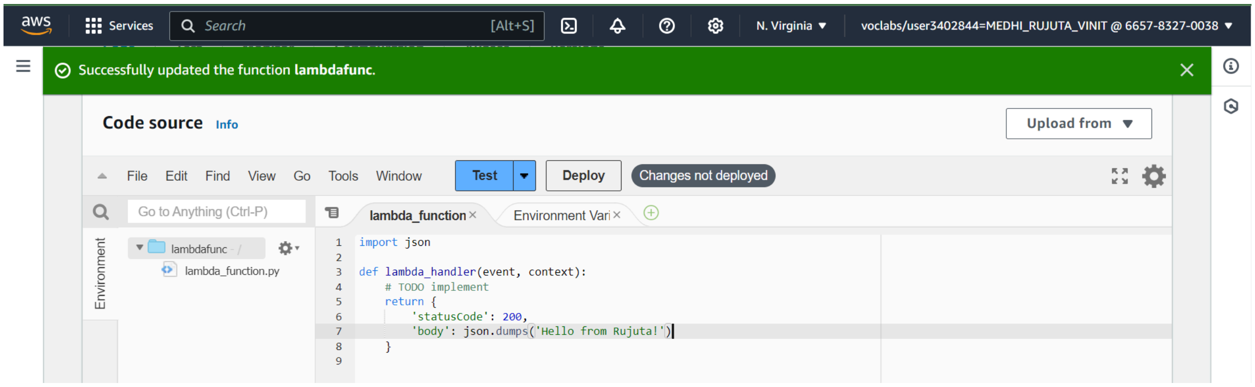
Choose a role that defines the permissions of your function. To create a custom role, go to the IAM console.

● Use an existing role

○ Create a new role from AWS policy templates

6. 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..



Successfully updated the function **lambdafunc**.

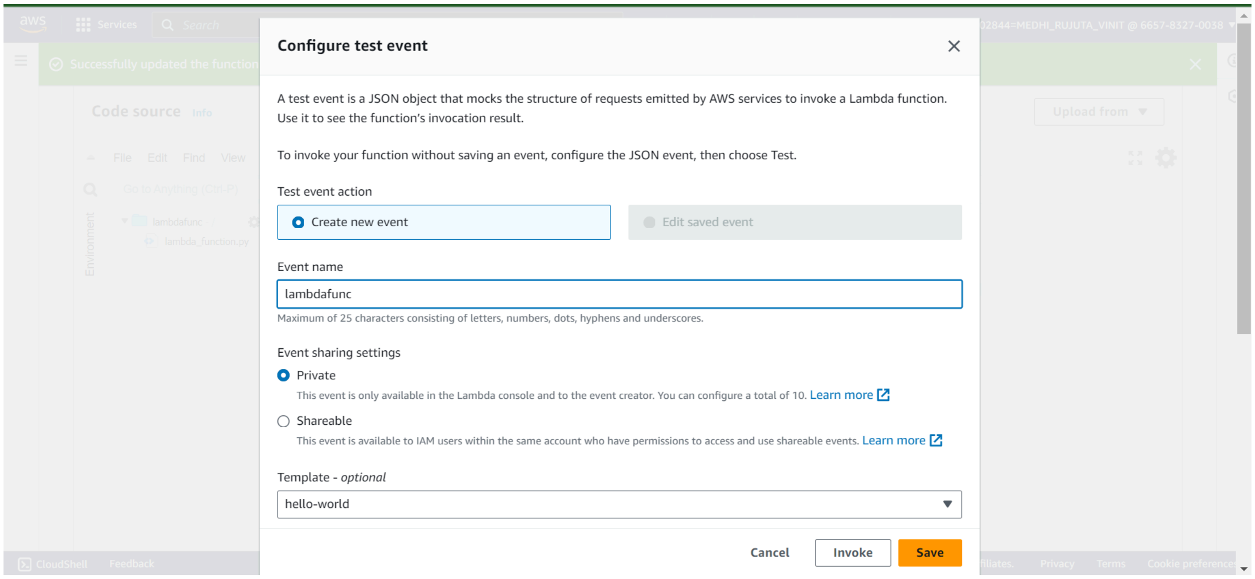
Code source Info

```
import json

def lambda_handler(event, context):
    # TODO implement
    return {
        'statusCode': 200,
        'body': json.dumps('Hello from Rujuta!')
    }

```

7. 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

lambdafunc

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

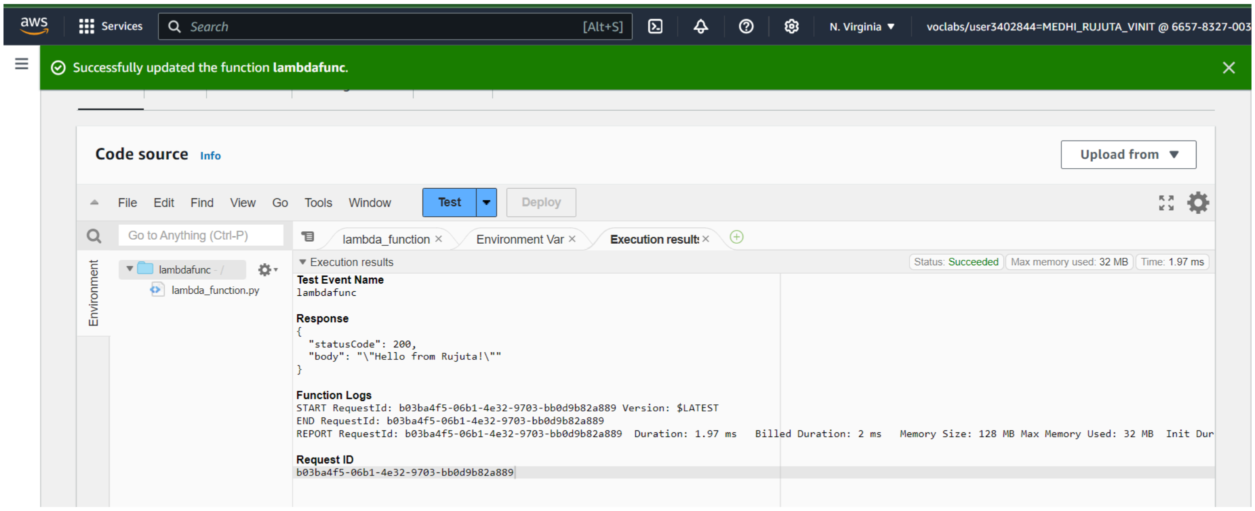
○ Shareable

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

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



Successfully updated the function **lambdafunc**.

Code source Info

Execution results

Status: Succeeded | Max memory used: 32 MB | Time: 1.97 ms

Test Event Name

```
lambdafunc
```

Response

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

Function Logs

```
START RequestId: b03ba4f5-06b1-4e32-9703-bb0d9b82a889 Version: $LATEST
END RequestId: b03ba4f5-06b1-4e32-9703-bb0d9b82a889
REPORT RequestId: b03ba4f5-06b1-4e32-9703-bb0d9b82a889  Duration: 1.97 ms   Billed Duration: 2 ms   Memory Size: 128 MB Max Memory Used: 32 MB  Init Dur
```

Request ID

```
b03ba4f5-06b1-4e32-9703-bb0d9b82a889
```