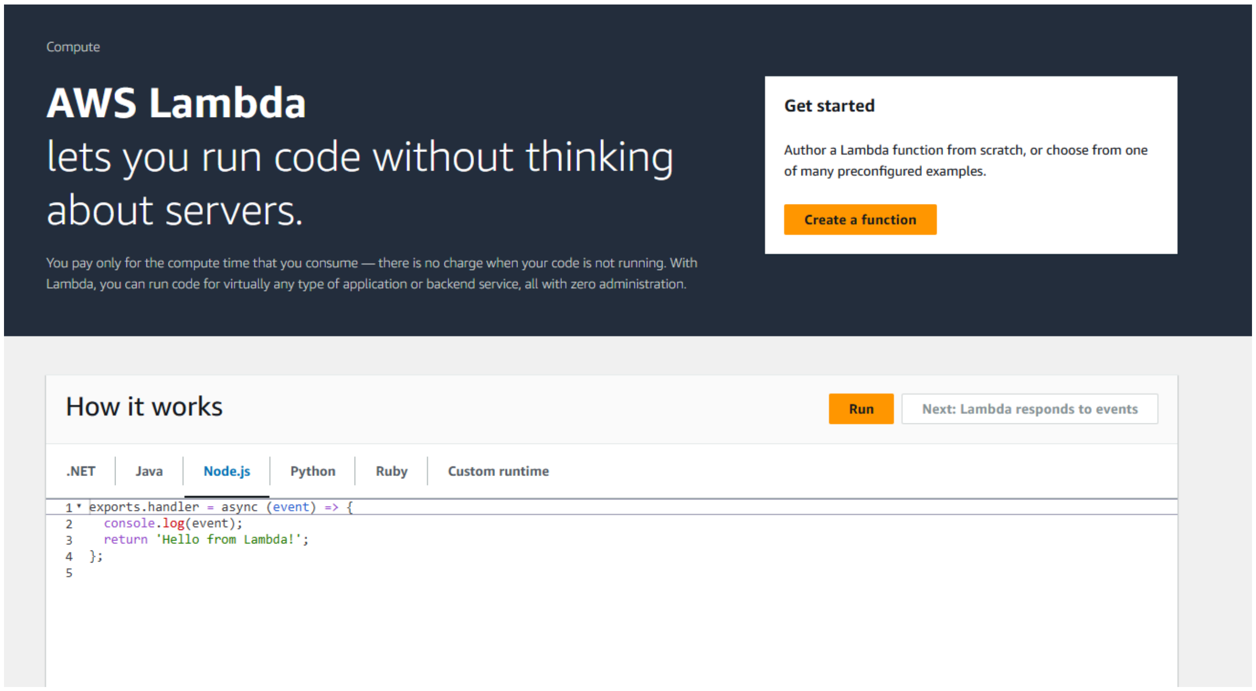
Adv_devops _11 **Avan Shetty** **D15C-52**

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

Steps:

1. Go to AWS ACADEMY.
2. **Create the lambda function:**

Firstly, Search lambda, then Open lambda and then click on create function button.


Compute

AWS Lambda
lets you run code without thinking about servers.

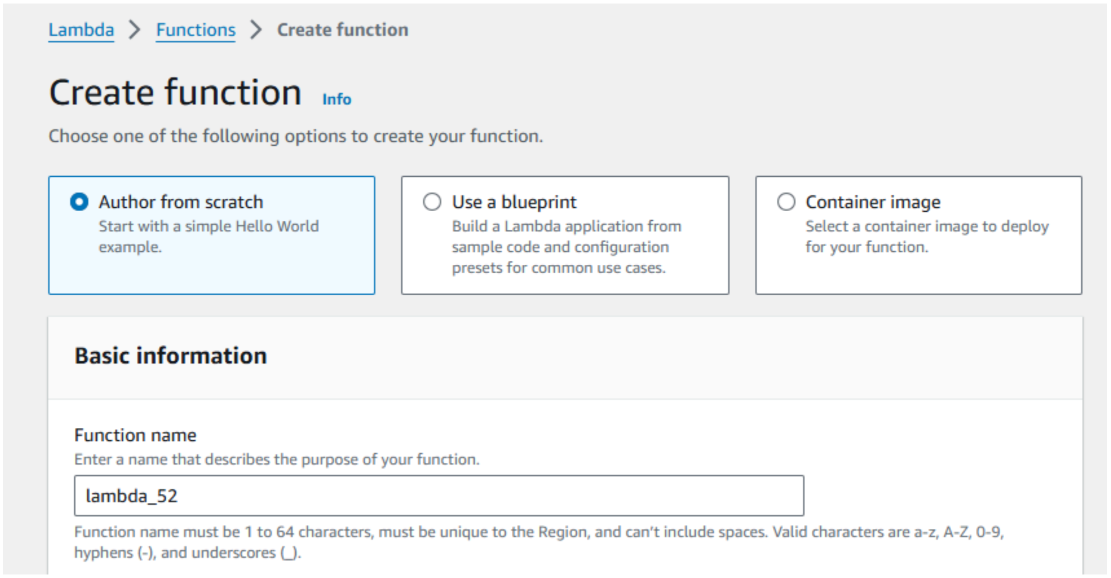
You pay only for the compute time that you consume — there is no charge when your code is not running. With Lambda, you can run code for virtually any type of application or backend service, all with zero administration.

Get started

Author a Lambda function from scratch, or choose from one of many preconfigured examples.

```
exports.handler = async (event) => {
  console.log(event);
  return 'Hello from Lambda!';
};
```

3. Now Give a name to your Lambda function,



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.

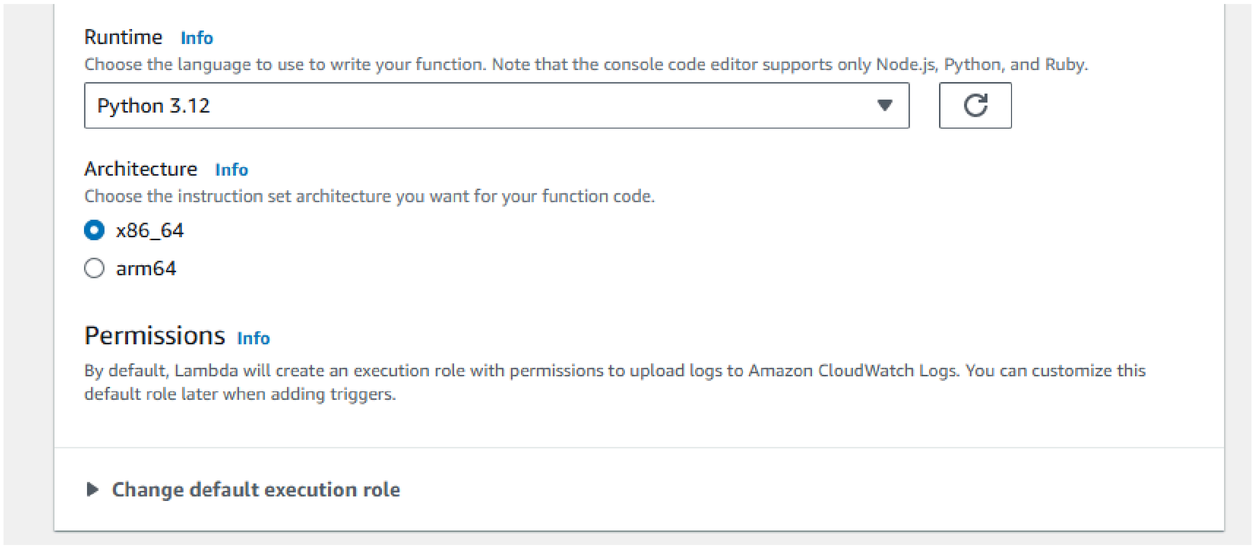
Basic information

Function name: lambda_52

Enter a name that describes the purpose of your function.

Function name must be 1 to 64 characters, must be unique to the Region, and can't include spaces. Valid characters are a-z, A-Z, 0-9, hyphens (-), and underscores (_).

4. 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 Exceution role to Create a new role with basic Lambda permissions.



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

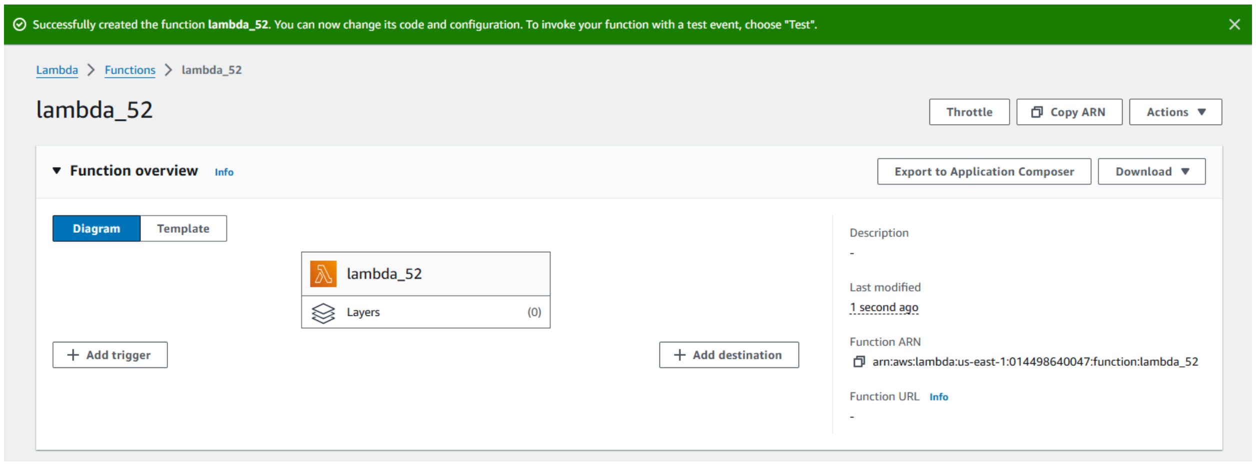
Choose the instruction set architecture you want for your function code.

[✓ x86_64] arm64

Permissions Info

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.

Thus the Lambda function was created successfully.



Successfully created the function **lambda_52**. You can now change its code and configuration. To invoke your function with a test event, choose "Test".

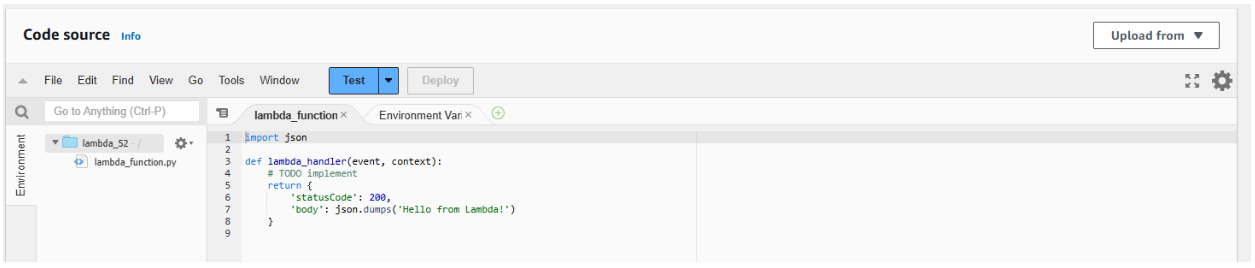
Description: -

Last modified: 1 second ago

Function ARN: arn:aws:lambda:us-east-1:014498640047:function:lambda_52

Function URL Info: -

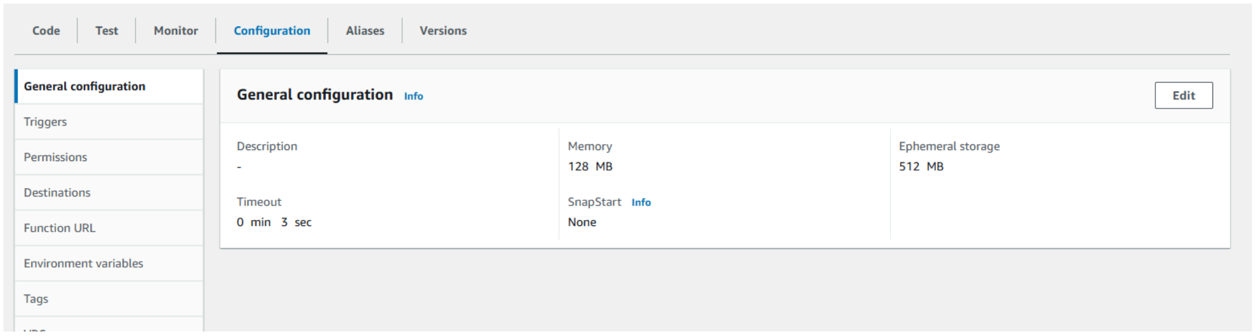
5. Code Source Section



```
import json

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

So now to edit the basic settings, go to configuration then click on edit.



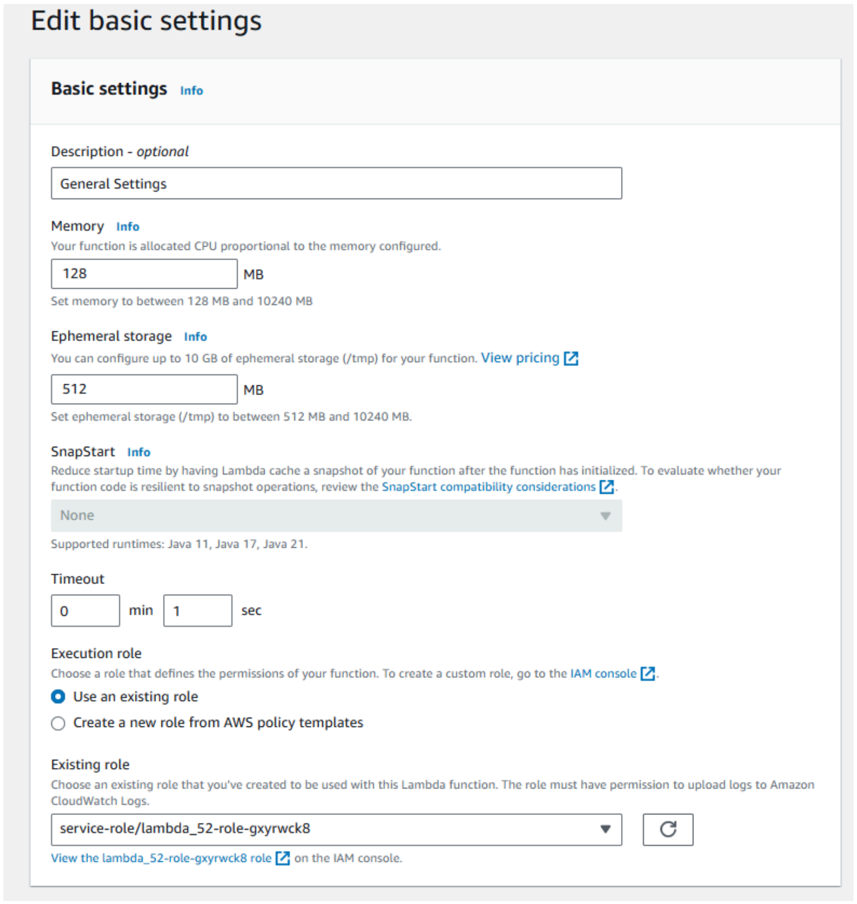
Code | Test | Monitor | Configuration | Aliases | Versions

General configuration
Triggers
Permissions
Destinations
Function URL
Environment variables
Tags

General configuration Info — Edit

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

Now enter a description and change Memory and Timeout.Here, I've changed the Timeout period to 1 sec.



Edit basic settings

Basic settings Info

Description - *optional*: General 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.

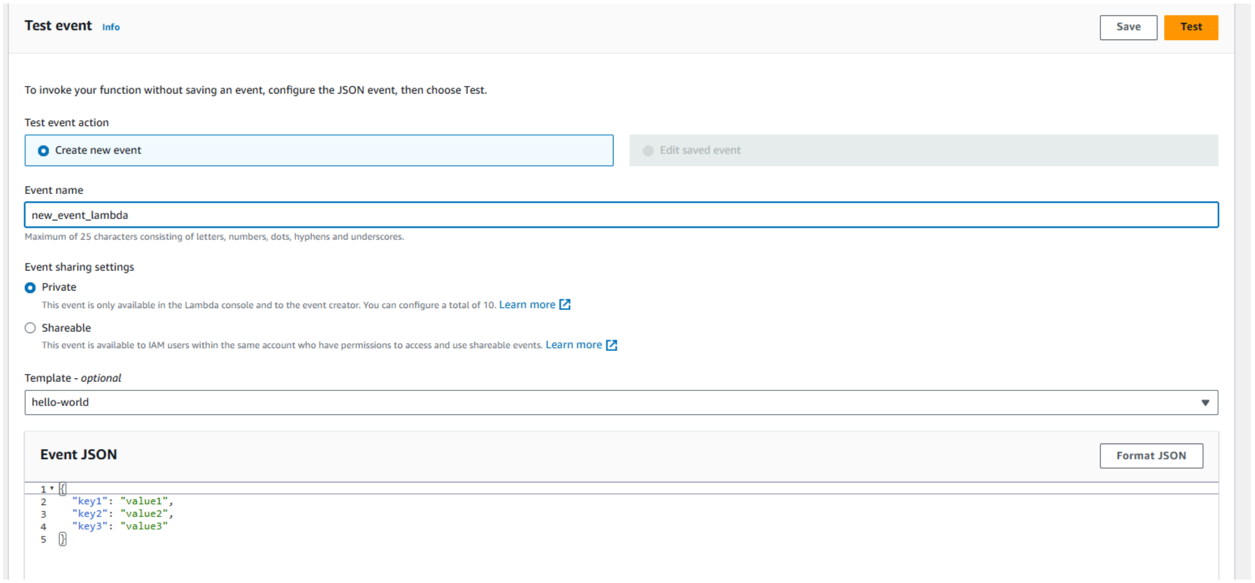
SnapStart Info
Reduce startup time by having Lambda cache a snapshot of your function after the function has initialized. To evaluate whether your function code is resilient to snapshot operations, review the SnapStart compatibility considerations.
None
Supported runtimes: Java 11, Java 17, Java 21.

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

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_52-role-gxyrwck8
View the lambda_52-role-gxyrwck8 role on the IAM console.

6. Now Click on the Test tab then select Create a new event, give a name to the event here i have given name as "new_event_lambda" and then select event sharing to private, and select hello-world template.


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

new_event_lambda

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*

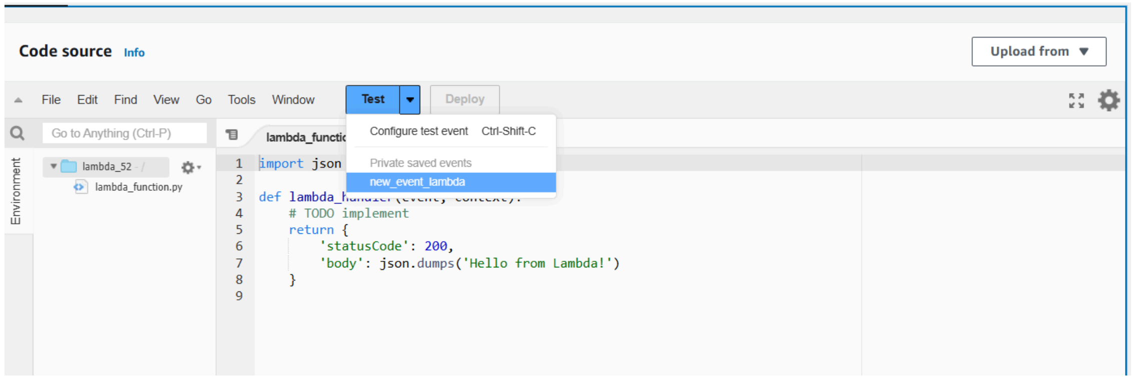
hello-world

Event JSON

Format JSON

```
{
    "key1": "value1",
    "key2": "value2",
    "key3": "value3"
}
```

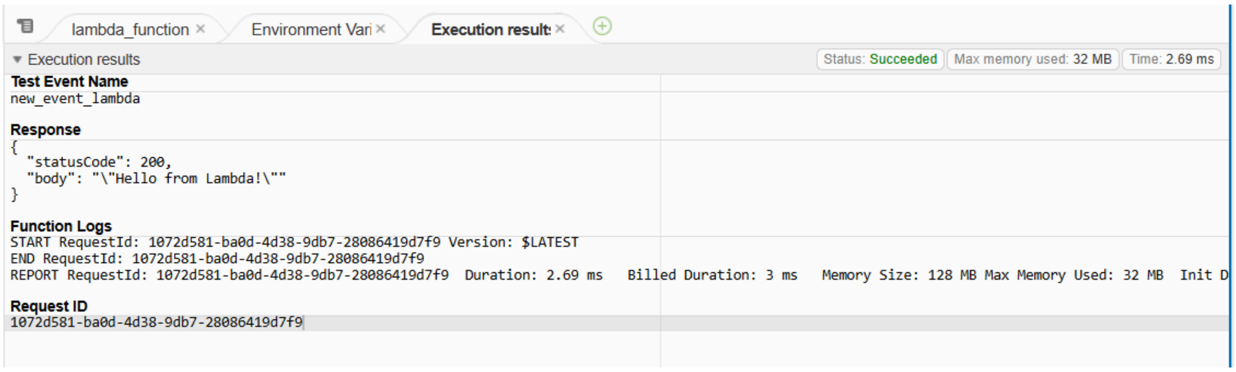
7. **Testing & Deployment:** Now In Code section select the created event (our_event) from the dropdown of test ,then click on test .



```
import json

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

8. Now you will see the following output.



Execution results Status: Succeeded Max memory used: 32 MB Time: 2.69 ms

Test Event Name
new_event_lambda

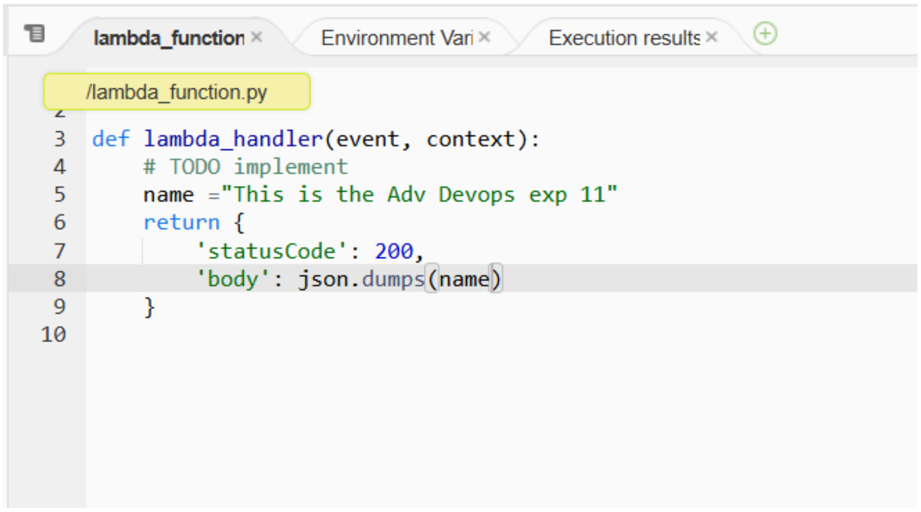
Response
```
{
  "statusCode": 200,
  "body": "\"Hello from Lambda!\""
}
```

Function Logs
```
START RequestId: 1072d581-ba0d-4d38-9db7-28086419d7f9 Version: $LATEST
END RequestId: 1072d581-ba0d-4d38-9db7-28086419d7f9
REPORT RequestId: 1072d581-ba0d-4d38-9db7-28086419d7f9  Duration: 2.69 ms   Billed Duration: 3 ms   Memory Size: 128 MB Max Memory Used: 32 MB  Init D
```

Request ID
1072d581-ba0d-4d38-9db7-28086419d7f9

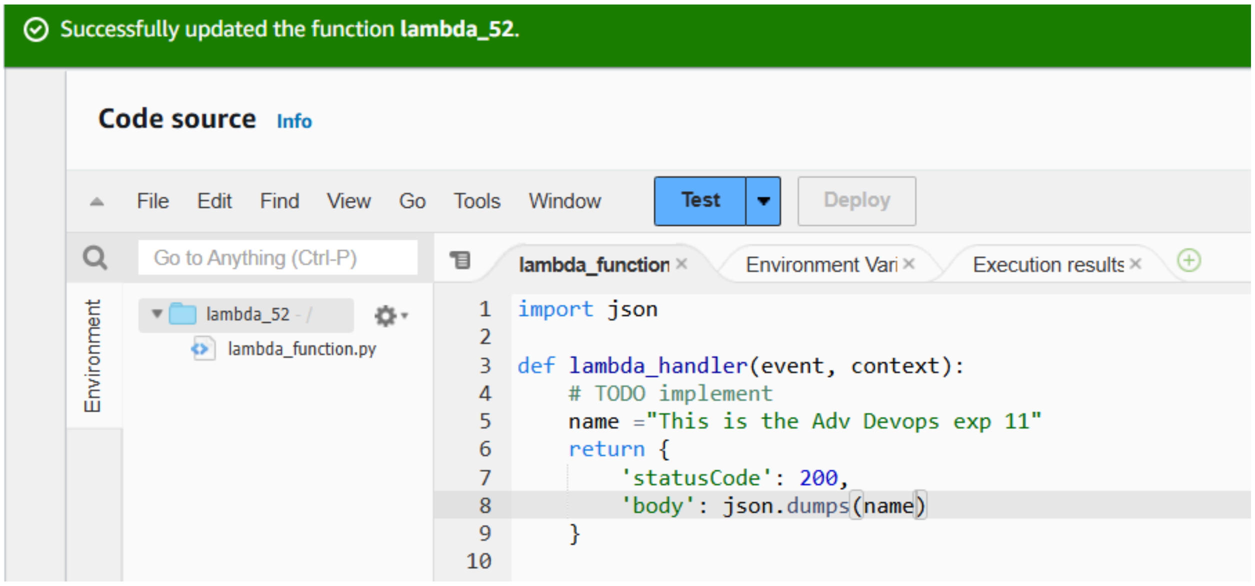
9. **Editing the given code into our own by adding a string or you can add anything**

You can edit your lambda function code.Here I have created a new string name “new_string” and assigned a string to it.



```python
def lambda_handler(event, context):
    # TODO implement
    name ="This is the Adv Devops exp 11"
    return {
        'statusCode': 200,
        'body': json.dumps(name)
    }
```

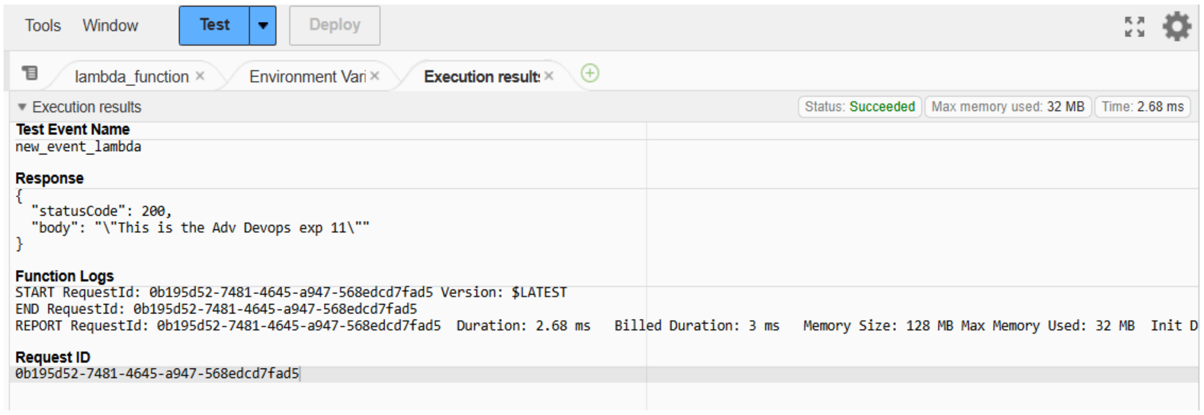
Now save it by ctrl+s and then click finally on deploy to deploy the changes.


Successfully updated the function **lambda_52**.

```python
import json

def lambda_handler(event, context):
    # TODO implement
    name ="This is the Adv Devops exp 11"
    return {
        'statusCode': 200,
        'body': json.dumps(name)
    }
```

10. **Testing and redeploying changes** Now click on the test and observe the output. Thus Output gives status code 200. This deployment is done successfully.



Test Event Name
```
new_event_lambda
```

Response
```
{
  "statusCode": 200,
  "body": "\"This is the Adv Devops exp 11\""
}
```

Function Logs
```
START RequestId: 0b195d52-7481-4645-a947-568edcd7fad5 Version: $LATEST
END RequestId: 0b195d52-7481-4645-a947-568edcd7fad5
REPORT RequestId: 0b195d52-7481-4645-a947-568edcd7fad5  Duration: 2.68 ms   Billed Duration: 3 ms   Memory Size: 128 MB Max Memory Used: 32 MB  Init D
```

Request ID
```
0b195d52-7481-4645-a947-568edcd7fad5
```

Conclusion:

In this experiment, we successfully created an AWS Lambda function using Python as the chosen language. After configuring the basic settings, we adjusted the timeout to 1 second, tested the function, and deployed it. The deployment was successful. We then modified the Lambda function's code and redeployed it to observe real-time changes. This process highlighted the ease of using AWS Lambda for building serverless applications. However, one issue we encountered was that, while we initially added the code source in Python, we could have chosen other supported languages as well.