

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.

[Create a function](#)

How it works [Run](#) [Next: Lambda responds to events](#)

[.NET](#) [Java](#) [Node.js](#) [Python](#) [Ruby](#) [Custom runtime](#)

```
1 * exports.handler = async (event) => {
2   console.log(event);
3   return 'Hello from Lambda!';
4 };
5
```

3. Now Give a name to your Lambda function,

[Lambda](#) > [Functions](#) > [Create 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
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 Execution role to Create a new role with basic Lambda permissions.

The screenshot shows the AWS Lambda console configuration page for a new function. It has three main sections: Runtime, Architecture, and Permissions. The Runtime section has a dropdown menu set to 'Python 3.12' and a refresh button. The Architecture section has two radio buttons: 'x86_64' (selected) and 'arm64'. The Permissions section has a text description and a button labeled 'Change default execution role'.

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.

► Change default execution role

Thus the Lambda function was created successfully.

The screenshot shows the AWS Lambda console 'Function overview' for a function named 'lambda_52'. At the top, a green banner states: 'Successfully created the function lambda_52. You can now change its code and configuration. To invoke your function with a test event, choose "Test".' The overview includes a diagram of the function, a table for layers (currently empty), and a description panel on the right. The description panel shows the function's ARN and URL.

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

lambda_52

Function overview

Diagram Template

lambda_52

Layers (0)

+ Add trigger

+ Add destination

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

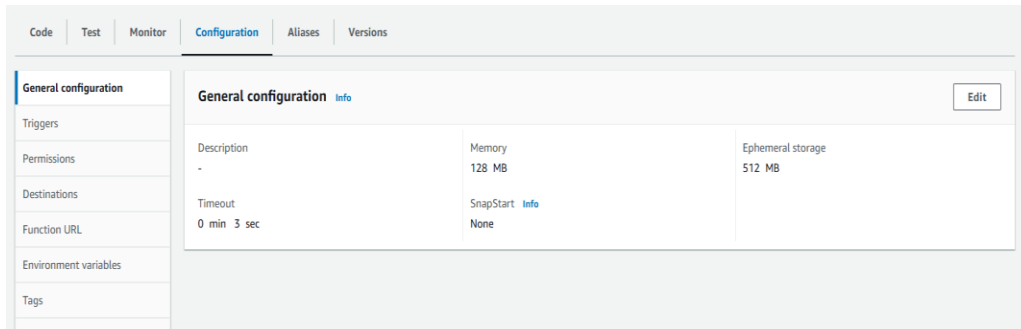
The screenshot shows the 'Code source' section of the AWS Lambda console for the 'lambda_52' function. It features a code editor with a Python script. The script imports the 'json' module and defines a 'lambda_handler' function that returns a JSON response with a status code of 200 and a body of 'Hello from Lambda!'. The interface includes a file explorer on the left, a menu bar at the top, and a 'Test' button.

Code source [Info](#)

Upload from

```
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
```

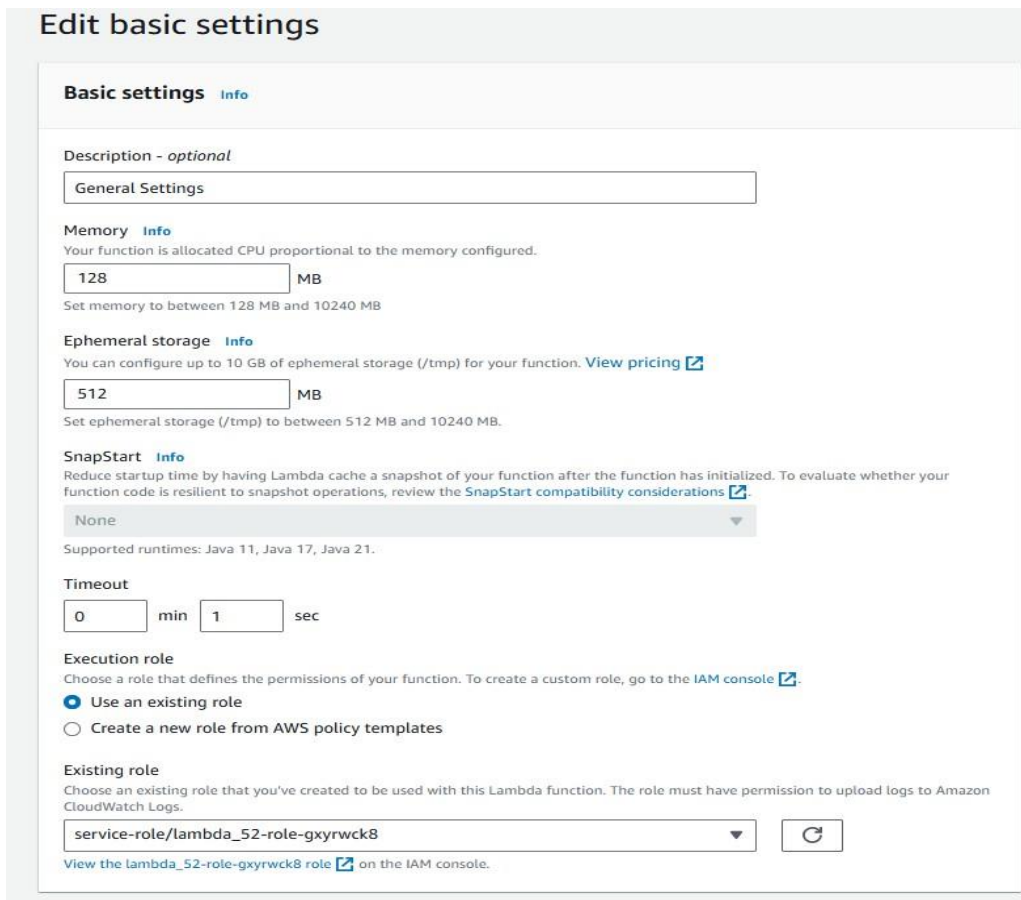
So now to edit the basic settings, go to configuration then click on edit.



The screenshot shows the 'Configuration' tab of the AWS Lambda console. On the left is a sidebar with links: General configuration, Triggers, Permissions, Destinations, Function URL, Environment variables, and Tags. The main area is titled 'General configuration' with an 'Edit' button. It contains a table of settings:

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

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



The screenshot shows the 'Edit basic settings' page in the AWS Lambda console. The page title is 'Edit basic settings'. Below it is the 'Basic settings' section with an 'Info' link. The settings are as follows:

- Description - optional:** A text input field containing 'General Settings'.
- Memory:** A text input field containing '128' followed by 'MB'. Below it, a note says 'Set memory to between 128 MB and 10240 MB'.
- Ephemeral storage:** A text input field containing '512' followed by 'MB'. Below it, a note says 'Set ephemeral storage (/tmp) to between 512 MB and 10240 MB'.
- SnapStart:** A dropdown menu set to 'None'. Below it, a note says 'Supported runtimes: Java 11, Java 17, Java 21'.
- Timeout:** Two input fields: '0' for minutes and '1' for seconds, followed by 'sec'.
- Execution role:** Two radio buttons: 'Use an existing role' (selected) and 'Create a new role from AWS policy templates'.
- Existing role:** A dropdown menu showing 'service-role/lambda_52-role-gxyrwck8' and a refresh button.

At the bottom, there is a link: 'View the lambda_52-role-gxyrwck8 role on the IAM console.'

- 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

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](#)

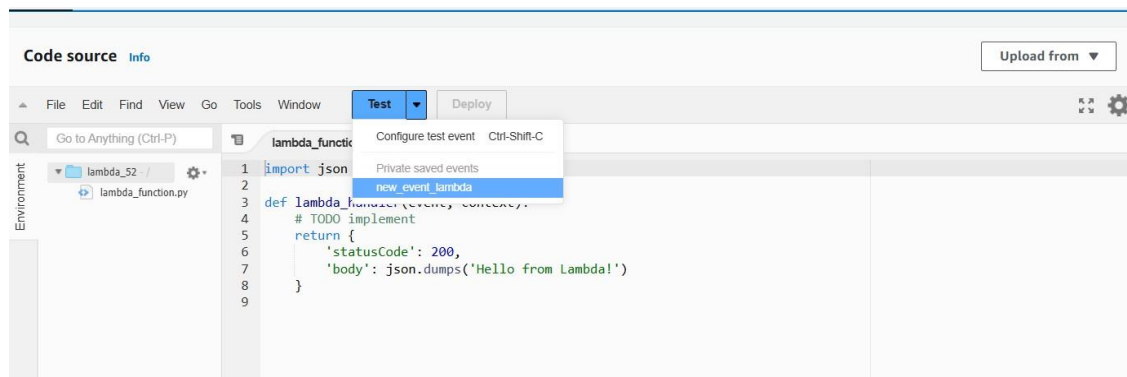
Template - optional

Event JSON Format JSON

```

1 {
2   "key1": "value1",
3   "key2": "value2",
4   "key3": "value3"
5 }
```

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



8. Now you will see the following output.

lambda_function x Environment Vari x **Execution results** x

▼ 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

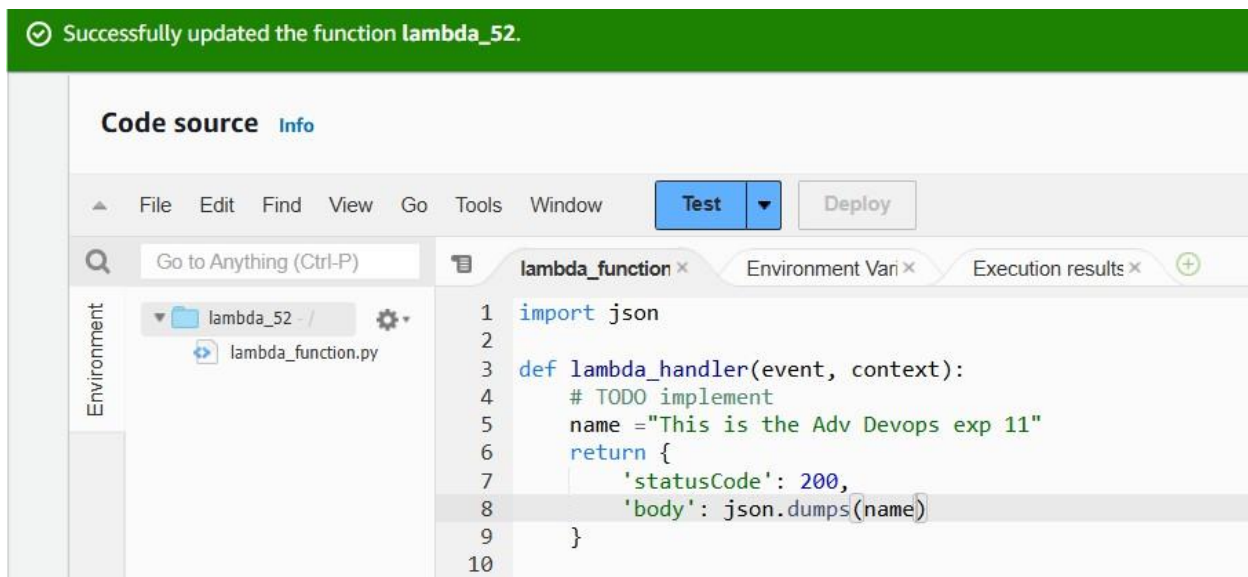
assigned a string to it.



The screenshot shows the AWS Lambda console interface. At the top, there are three tabs: 'lambda_function', 'Environment Vari', and 'Execution results'. The 'lambda_function' tab is active. Below the tabs, the file path '/lambda_function.py' is highlighted. The code is as follows:

```
2
3 def lambda_handler(event, context):
4     # TODO implement
5     name = "This is the Adv Devops exp 11"
6     return {
7         'statusCode': 200,
8         'body': json.dumps(name)
9     }
10
```

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

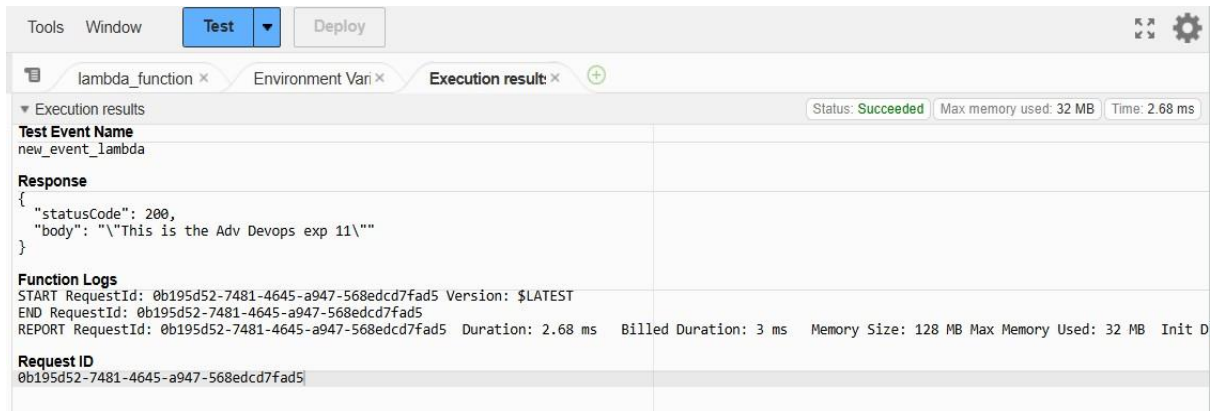


The screenshot shows the AWS Lambda console interface after a successful update. A green banner at the top reads: 'Successfully updated the function lambda_52.' Below the banner, the 'Code source' tab is active. The code is as follows:

```
1 import json
2
3 def lambda_handler(event, context):
4     # TODO implement
5     name = "This is the Adv Devops exp 11"
6     return {
7         'statusCode': 200,
8         'body': json.dumps(name)
9     }
10
```

The left sidebar shows the 'Environment' section with a folder named 'lambda_52' and a file named 'lambda_function.py'.

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.



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.