Adv DevOps Practical 11

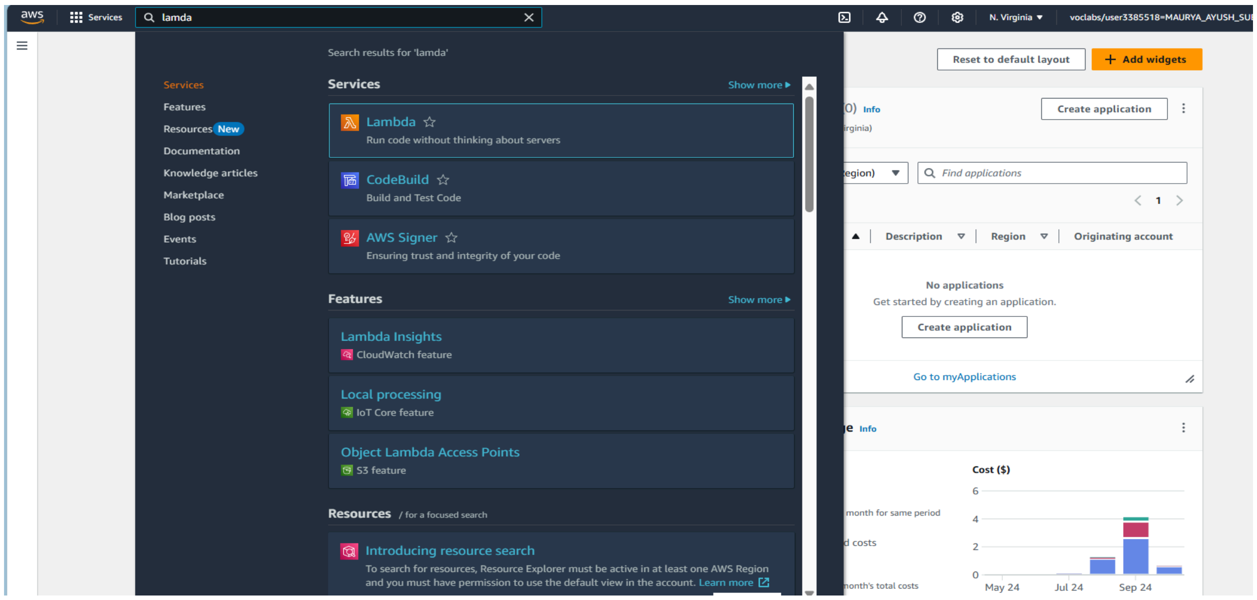
Aim: To understand AWS Lambda, its workflow, various functions and create your first Lambda functions using Python / Java / Nodejs.

Prerequisites:

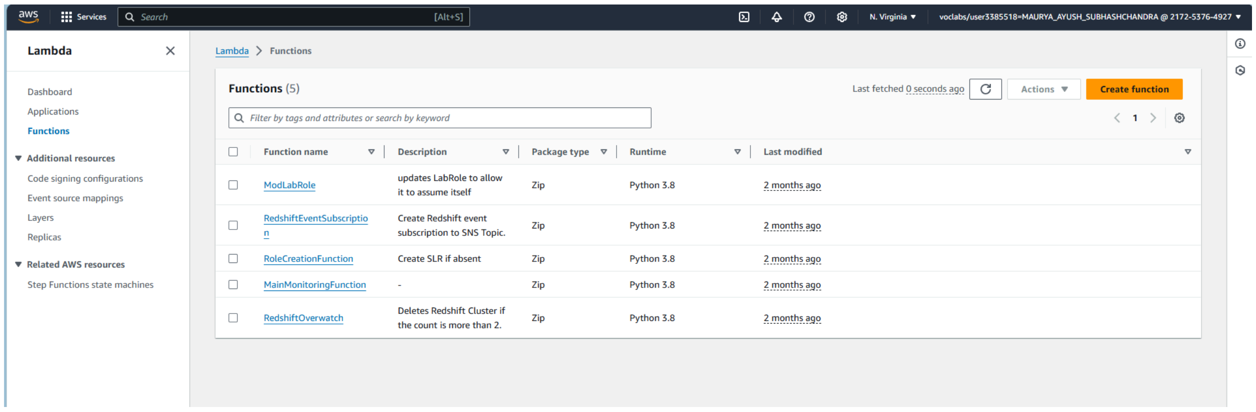
1) AWS account (academy recommended)

Step 1: Set up AWS Lambda Function

1) Search for Lambda in the services tab. Click on it once found.



2) Click on create functions.

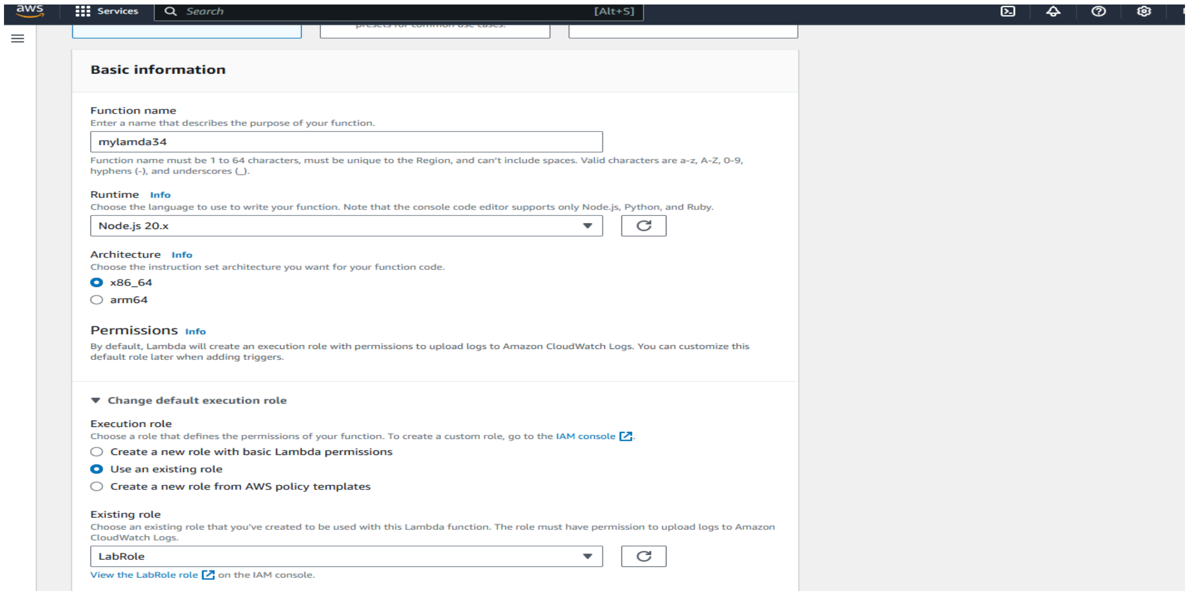


Lambda > Functions

Functions (5)

	Function name	Description	Package type	Runtime	Last modified
☐	ModLabRole	updates LabRole to allow it to assume itself	Zip	Python 3.8	2 months ago
☐	RedshiftEventSubscription	Create Redshift event subscription to SNS Topic.	Zip	Python 3.8	2 months ago
☐	RoleCreationFunction	Create SLR if absent	Zip	Python 3.8	2 months ago
☐	MainMonitoringFunction	-	Zip	Python 3.8	2 months ago
☐	RedshiftOverwatch	Deletes Redshift Cluster if the count is more than 2.	Zip	Python 3.8	2 months ago

3) Give a name to your Lambda function. Select the runtime as Node.js 20.x (You can also use python). Select the architecture as x86_64. Set the default execution role as LabRole if you are doing this on your academy account. (Use an existing role → LabRole)



Basic information

Function name
Enter a name that describes the purpose of your function.
mylamda34
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 (_).

Runtime Info
Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.
Node.js 20.x

Architecture Info
Choose the instruction set architecture you want for your function code.
- [✓ x86_64]
- arm64

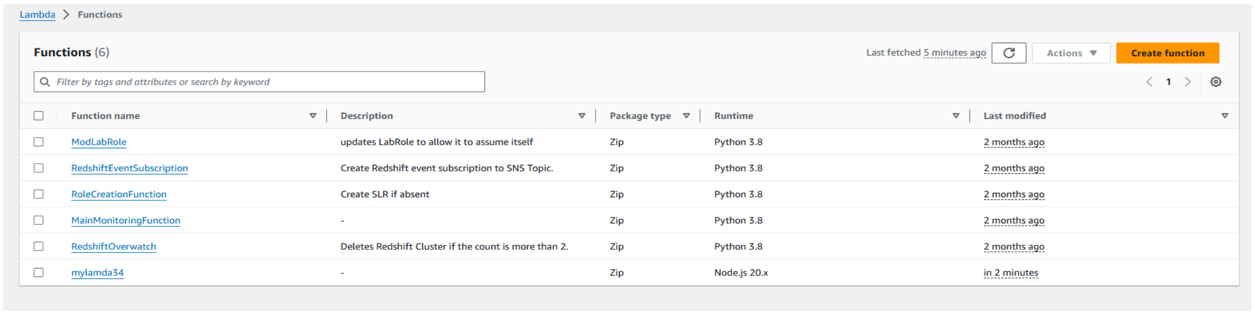
Permissions Info
By default, Lambda will create an execution role with permissions to upload logs to Amazon CloudWatch Logs. You can customize this default role later when adding triggers.

▼ Change default execution role

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.

4) Once the function is created, click on the name of the function.

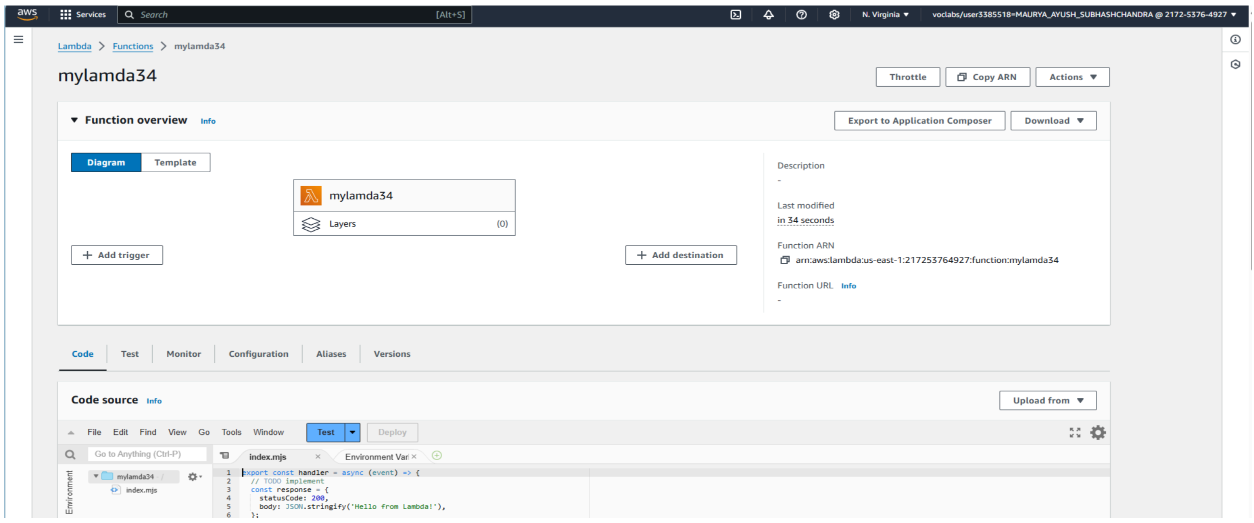


Lambda > Functions

Functions (6)   Last fetched 5 minutes ago   Actions   Create function

Function name	Description	Package type	Runtime	Last modified
ModLabRole	updates LabRole to allow it to assume itself	Zip	Python 3.8	2 months ago
RedshiftEventSubscription	Create Redshift event subscription to SNS Topic.	Zip	Python 3.8	2 months ago
RoleCreationFunction	Create SLR if absent	Zip	Python 3.8	2 months ago
MainMonitoringFunction	-	Zip	Python 3.8	2 months ago
RedshiftOverwatch	Deletes Redshift Cluster if the count is more than 2.	Zip	Python 3.8	2 months ago
mylamda34	-	Zip	Node.js 20.x	in 2 minutes

5) This is the dashboard of our lambda function.



N. Virginia   voclabs/user3385518=MAURYA_AYUSH_SUBHASHCHANDRA @ 2172-5376-4927

Lambda > Functions > mylamda34

mylamda34   Throttle   Copy ARN   Actions

▼ Function overview Info   Export to Application Composer   Download

Diagram | Template

mylamda34
Layers (0)

+ Add trigger   + Add destination

Description
-

Last modified
in 34 seconds

Function ARN
arn:aws:lambda:us-east-1:217253764927:function:mylamda34

Function URL Info
-

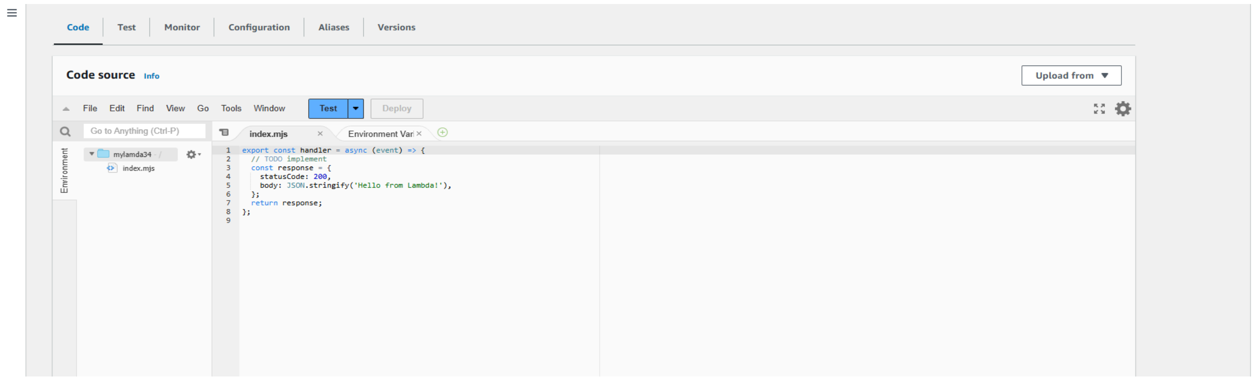
Code | Test | Monitor | Configuration | Aliases | Versions

Code source Info   Upload from

index.mjs

```
export const handler = async (event) => {
  // TODO implement
  const response = {
    statusCode: 200,
    body: JSON.stringify('Hello from Lambda!'),
  };
```

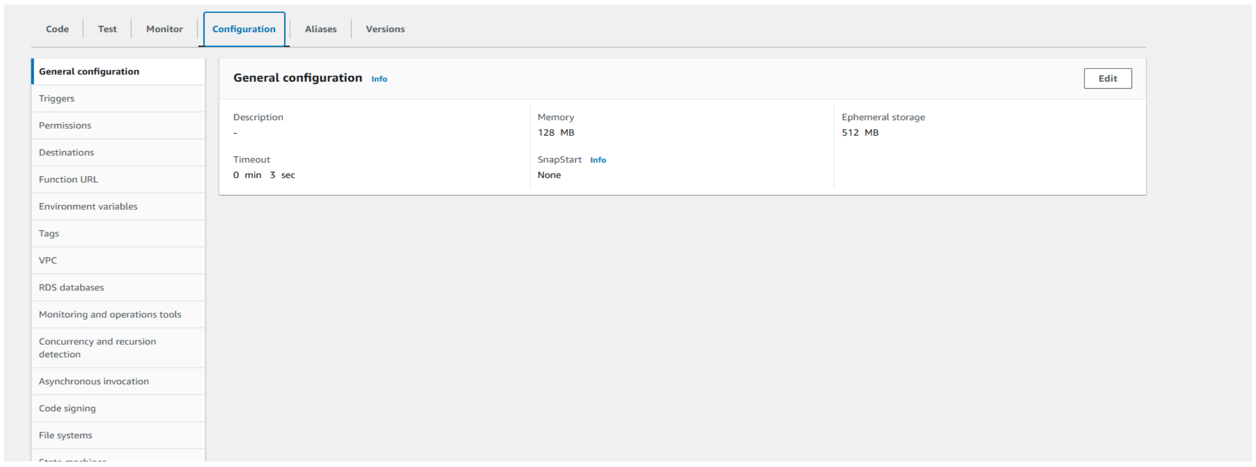
6) This function has the following default code, which is used to print “Hello form Lambda!”.



```
export const handler = async (event) => {
  // TODO implement
  const response = {
    statusCode: 200,
    body: JSON.stringify('Hello from Lambda!'),
  };
  return response;
};
```

Step 2: Set up configurations and test events

1) Just above the test code, you would find Configuration, click on it. Then click on Edit.



Description
-

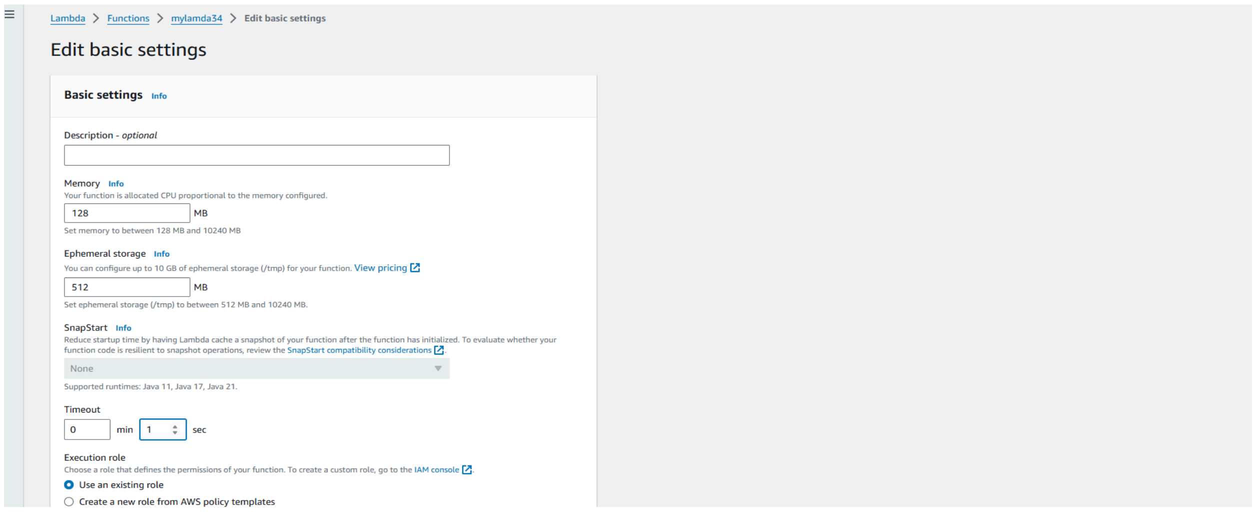
Memory
128 MB

Ephemeral storage
512 MB

Timeout
0 min 3 sec

SnapStart Info
None

2) Here, change the Timeout to 1 sec. This is the time for which the function can be running before it is forcibly terminated.



Description - *optional*

Memory Info
Your function is allocated CPU proportional to the memory configured.
128 MB
Set memory to between 128 MB and 10240 MB

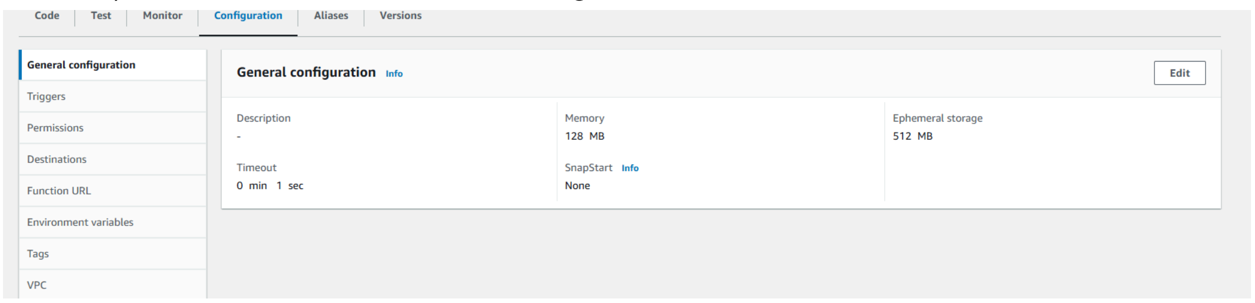
Ephemeral storage Info
You can configure up to 10 GB of ephemeral storage (/tmp) for your function. View pricing
512 MB
Set ephemeral storage (/tmp) to between 512 MB and 10240 MB.

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

3) We can see the executed changes.



General configuration Info

Description
-

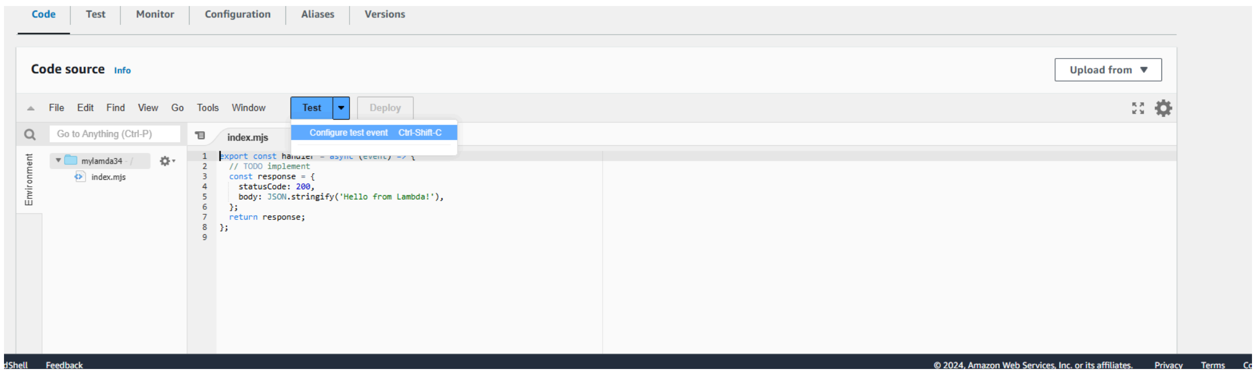
Memory
128 MB

Ephemeral storage
512 MB

Timeout
0 min 1 sec

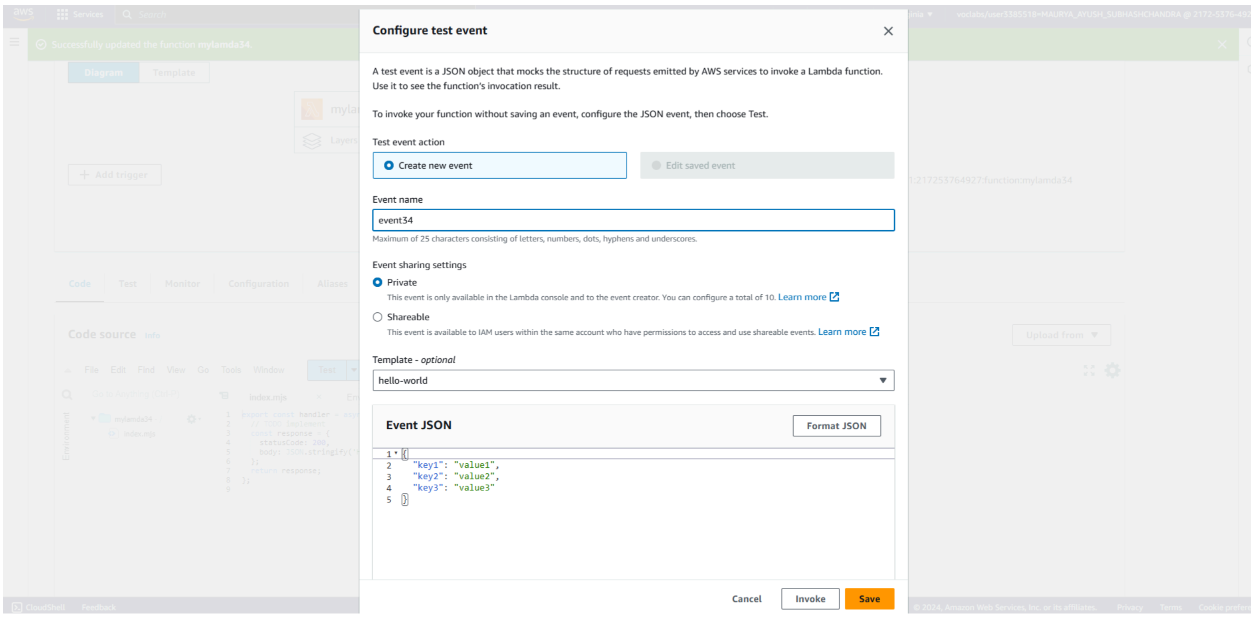
SnapStart Info
None

4) Switch back to the code tab. Click on the dropdown arrow near test. Then select configure test event.



```
export const handler = async (event) => {
  // TODO implement
  const response = {
    statusCode: 200,
    body: JSON.stringify('Hello from Lambda!'),
  };
  return response;
};
```

5) Here, create a new event, keep the other options default and save the event.



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
event34
Maximum of 25 characters consisting of letters, numbers, dots, hyphens and underscores.

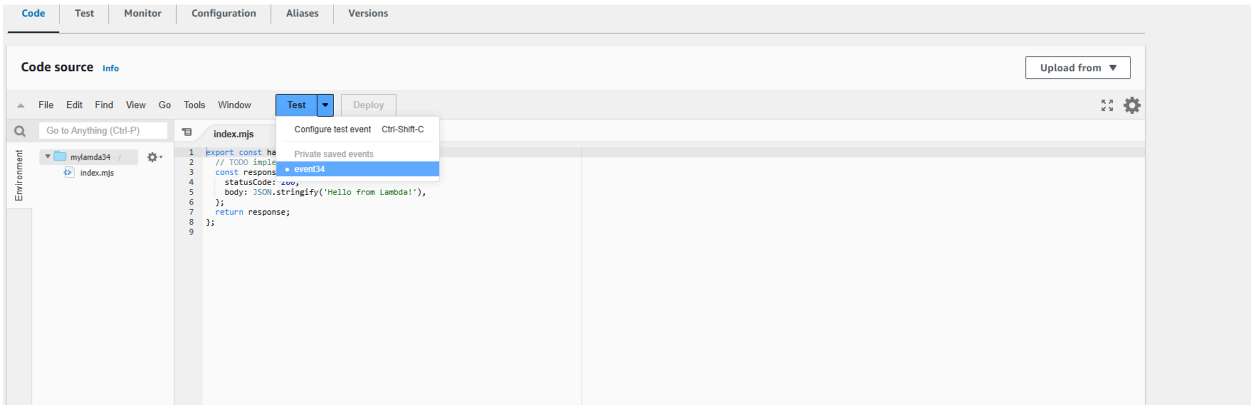
Event sharing settings
[✓ Private]
This event is only available in the Lambda console and to the event creator. You can configure a total of 10. Learn more
Shareable
This event is available to IAM users within the same account who have permissions to access and use shareable events. Learn more

Template - *optional*
hello-world

Event JSON

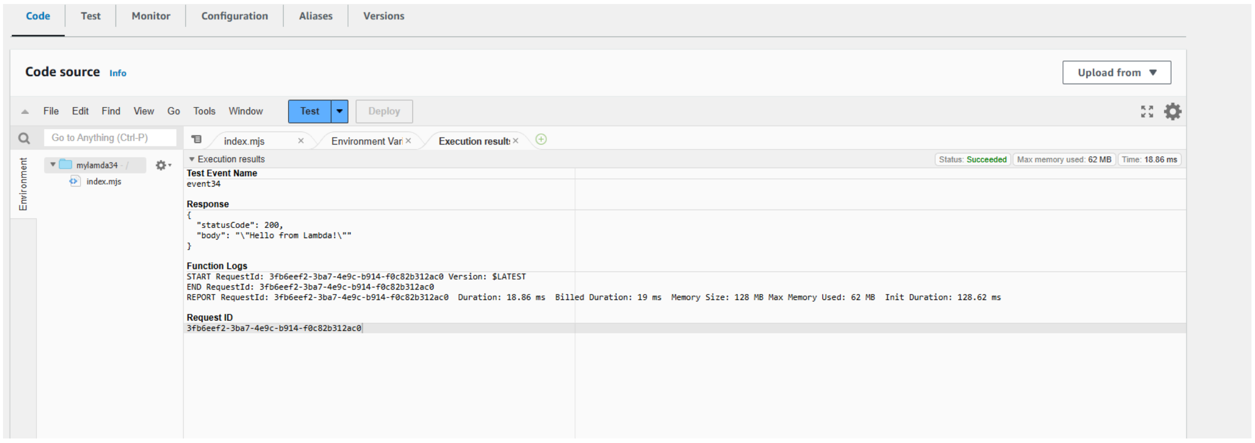
```
{
  "key1": "value1",
  "key2": "value2",
  "key3": "value3"
}
```

6) Now, again click on the dropdown. This time, select the event you have created. Then, click on TEST.



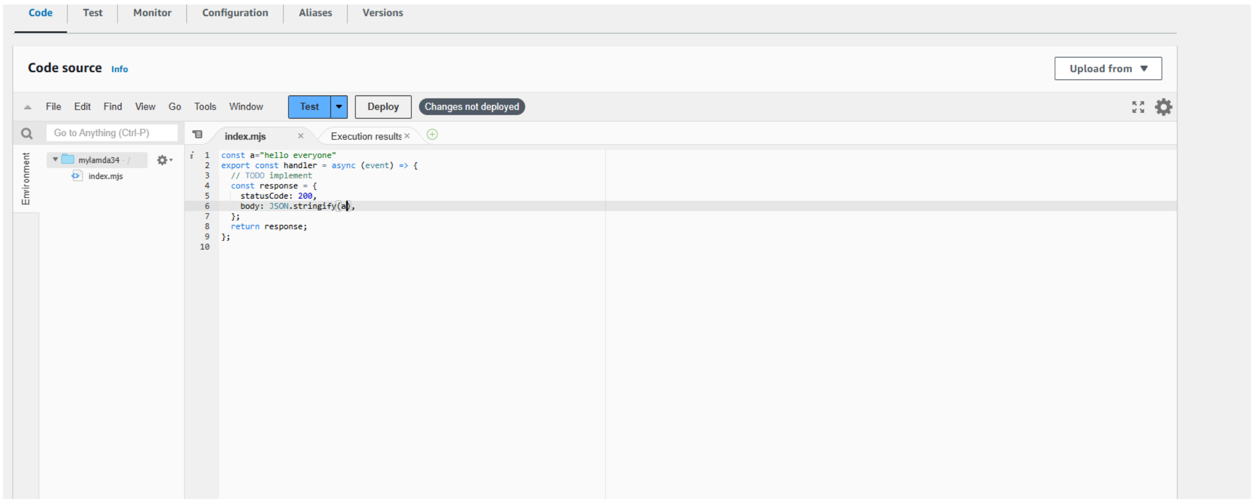
```
export const ha
  // TODO imple
  const respons
    statusCode: 200,
    body: JSON.stringify('Hello from Lambda!'),
  };
  return response;
};
```

7) We can see the expected output for the sample code.



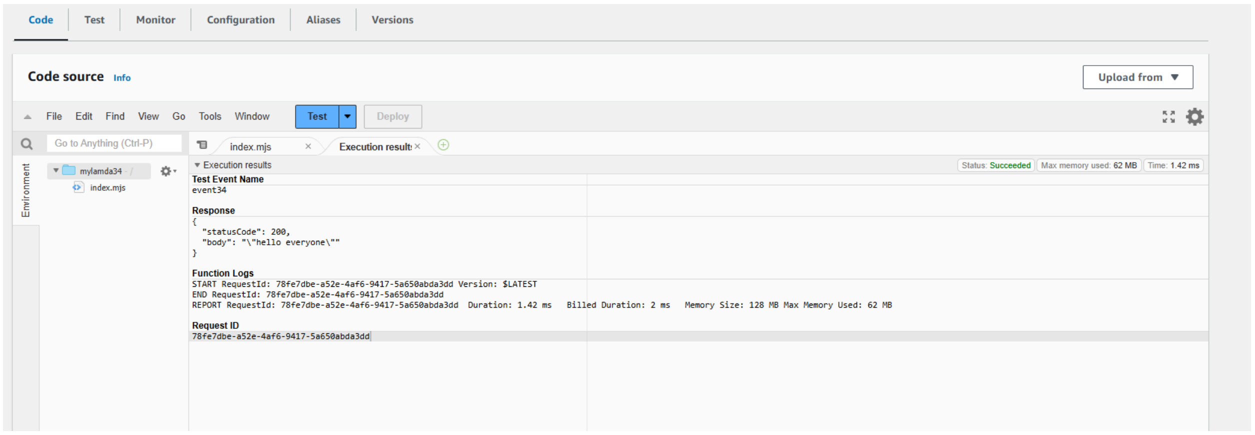
```
Test Event Name
event34

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

Function Logs
START RequestId: 3fb6eef2-3ba7-4e9c-b914-f0c82b312ac0 Version: $LATEST
END RequestId: 3fb6eef2-3ba7-4e9c-b914-f0c82b312ac0
REPORT RequestId: 3fb6eef2-3ba7-4e9c-b914-f0c82b312ac0  Duration: 18.86 ms  Billed Duration: 19 ms  Memory Size: 128 MB Max Memory Used: 62 MB  Init Duration: 128.62 ms

Request ID
3fb6eef2-3ba7-4e9c-b914-f0c82b312ac0
```

8) For a test, declare a string and call it in line 6. After making the changes click on deploy.



```
1  const a="hello everyone"
2  export const handler = async (event) => {
3    // TODO implement
4    const response = {
5      statusCode: 200,
6      body: JSON.stringify(a),
7    };
8    return response;
9  };
10
```

9) Run the test. We can see that the string we declared has come in the output.



Test Event Name
```
event34
```

Response
```
{
  "statusCode": 200,
  "body": "\"hello everyone\""
}
```

Function Logs
```
START RequestId: 78fe7dbe-a52e-4af6-9417-5a650abda3dd Version: $LATEST
END RequestId: 78fe7dbe-a52e-4af6-9417-5a650abda3dd
REPORT RequestId: 78fe7dbe-a52e-4af6-9417-5a650abda3dd  Duration: 1.42 ms   Billed Duration: 2 ms   Memory Size: 128 MB Max Memory Used: 62 MB
```

Request ID
```
78fe7dbe-a52e-4af6-9417-5a650abda3dd
```

Conclusion:

In this experiment, we explored AWS Lambda by creating and configuring Lambda functions using Node.js. We learned to set up a function, adjust configurations like timeout settings, and test it with custom events. This hands-on experience provided us with foundational skills in serverless computing, enabling us to develop scalable applications efficiently. Moving forward, we can investigate integrating AWS Lambda with other services to enhance our serverless applications.