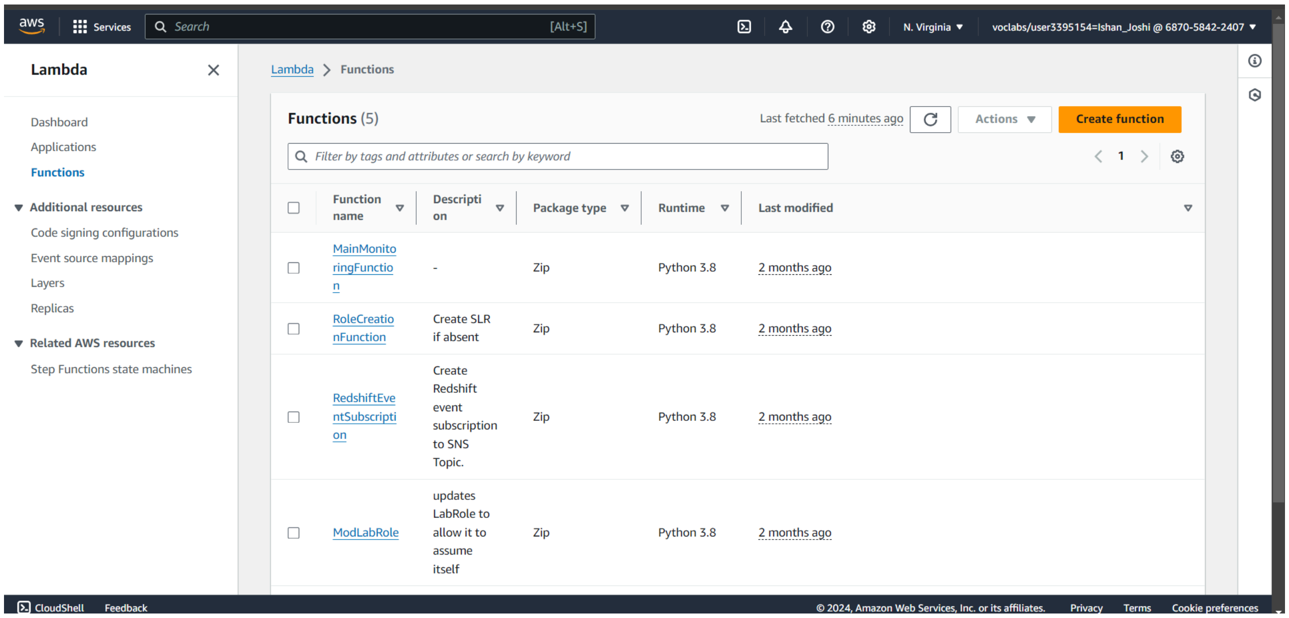
Name-Ishan Kiran Joshi Div-D15C Roll No-21 A.Y.-2024-25

Experiment 11

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

Steps:

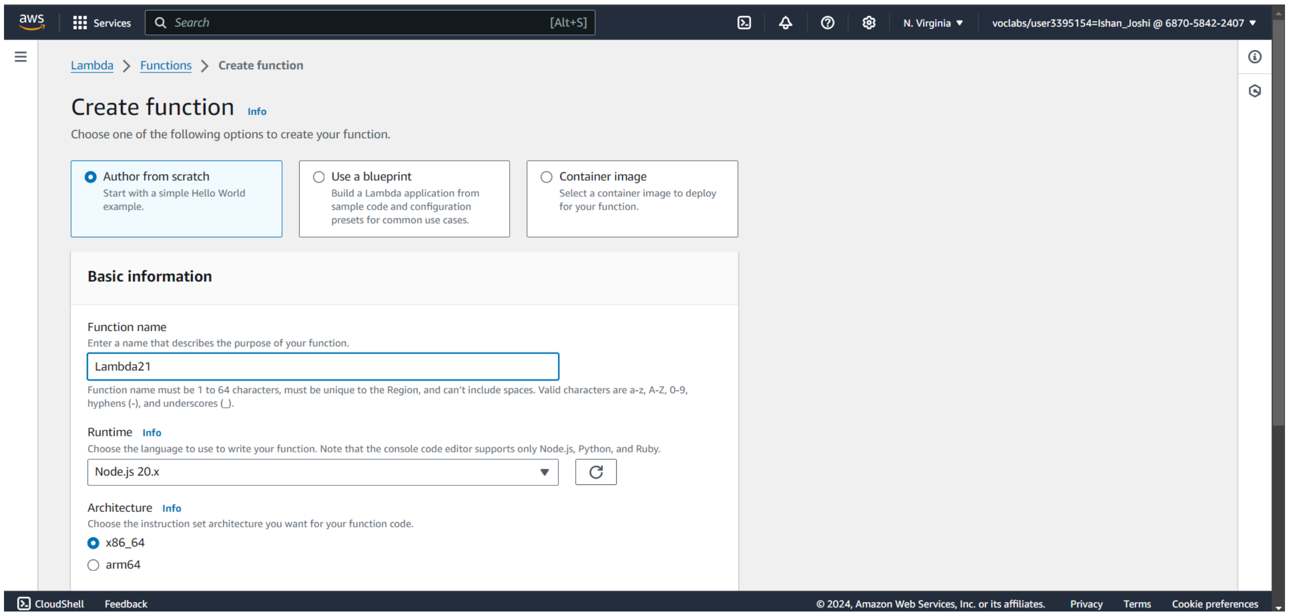
Step 1: On your AWS console, click on 'Lambda' in the services section and click on 'Create function'.



Functions (5)

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

Step 2: Give your Lambda function a name. Select the language to use to write your function (Node.js is the default and what we will use in this experiment). Keep other options as default.



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.

Lambda21

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.

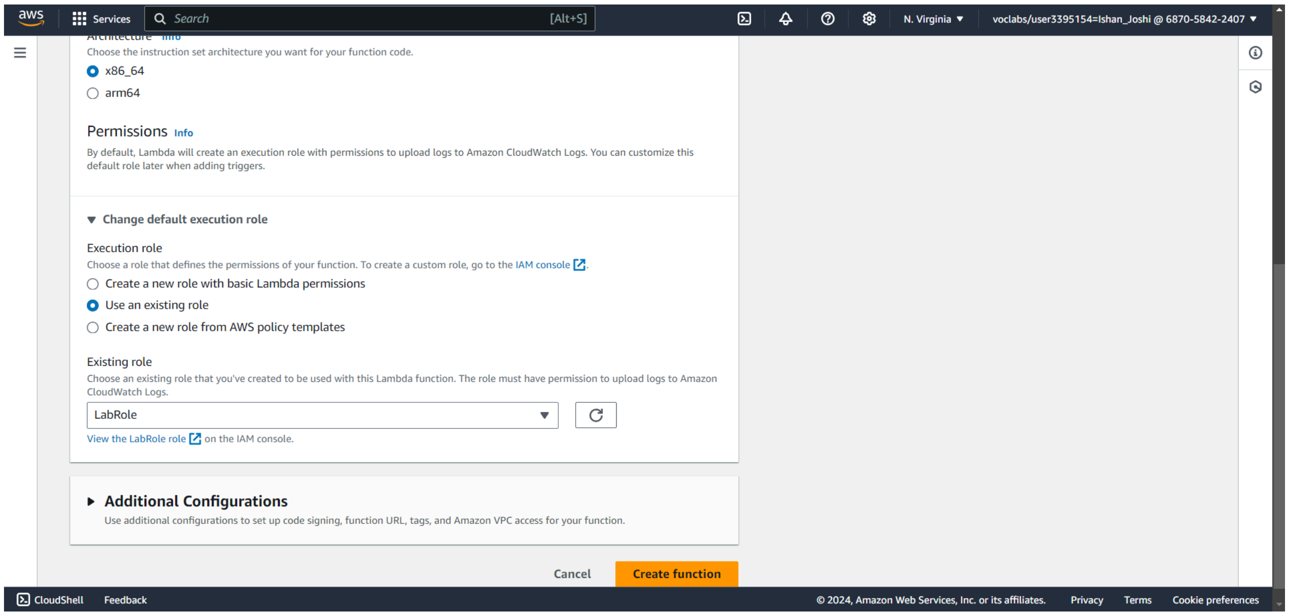
Node.js 20.x

Architecture Info

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

[✓ x86_64] arm64

Services
Search [Alt+S]
N. Virginia
voclabs/user3395154=Ishan_Joshi @ 6870-5842-2407
CloudShell
Feedback
© 2024, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences



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

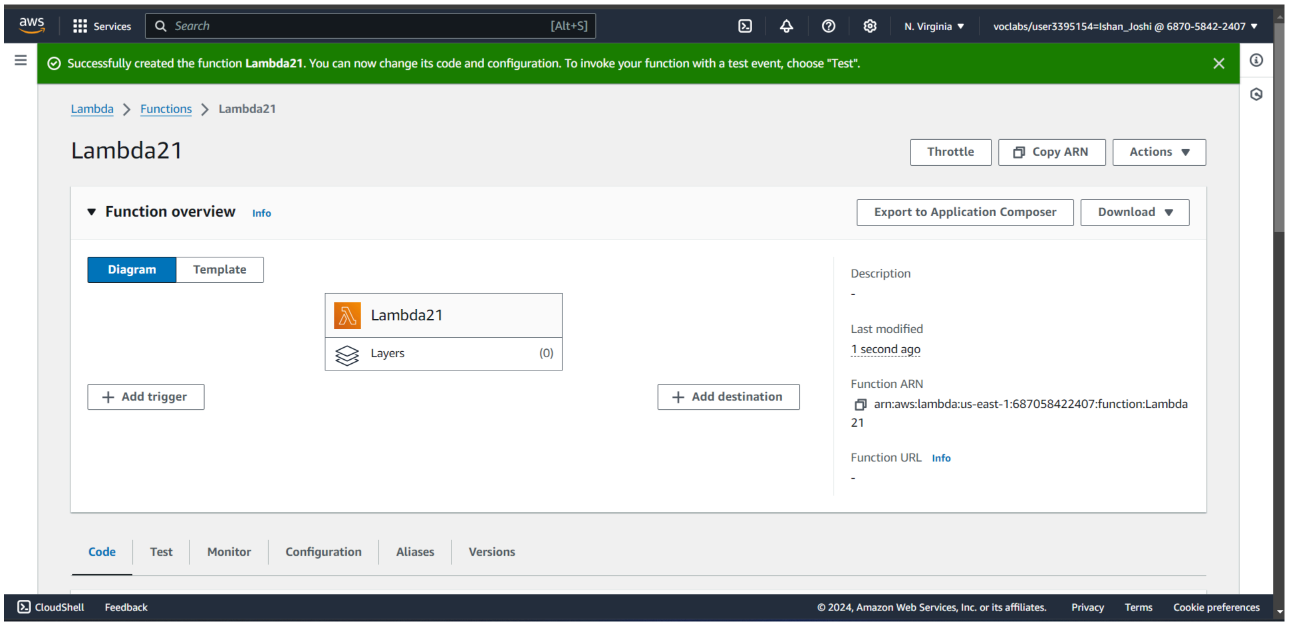
View the LabRole role on the IAM console.

▶ **Additional Configurations**

Use additional configurations to set up code signing, function URL, tags, and Amazon VPC access for your function.

Under ‘Execution role’, choose ‘Use an existing role’ and then choose LabRole. Then, click on ‘Create function’.

Your Lambda function gets created.



Successfully created the function **Lambda21**. You can now change its code and configuration. To invoke your function with a test event, choose "Test".

Lambda > Functions > Lambda21

Lambda21

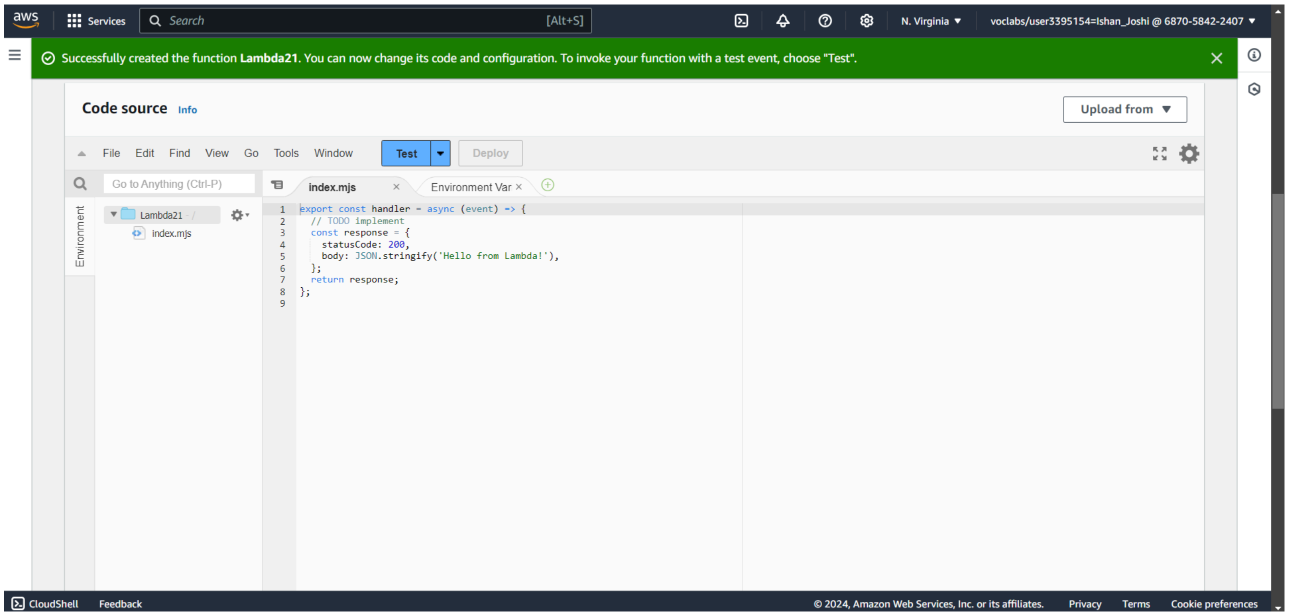
Function overview

Description
-

Last modified
1 second ago

Function ARN
arn:aws:lambda:us-east-1:687058422407:function:Lambda21

Function URL
-

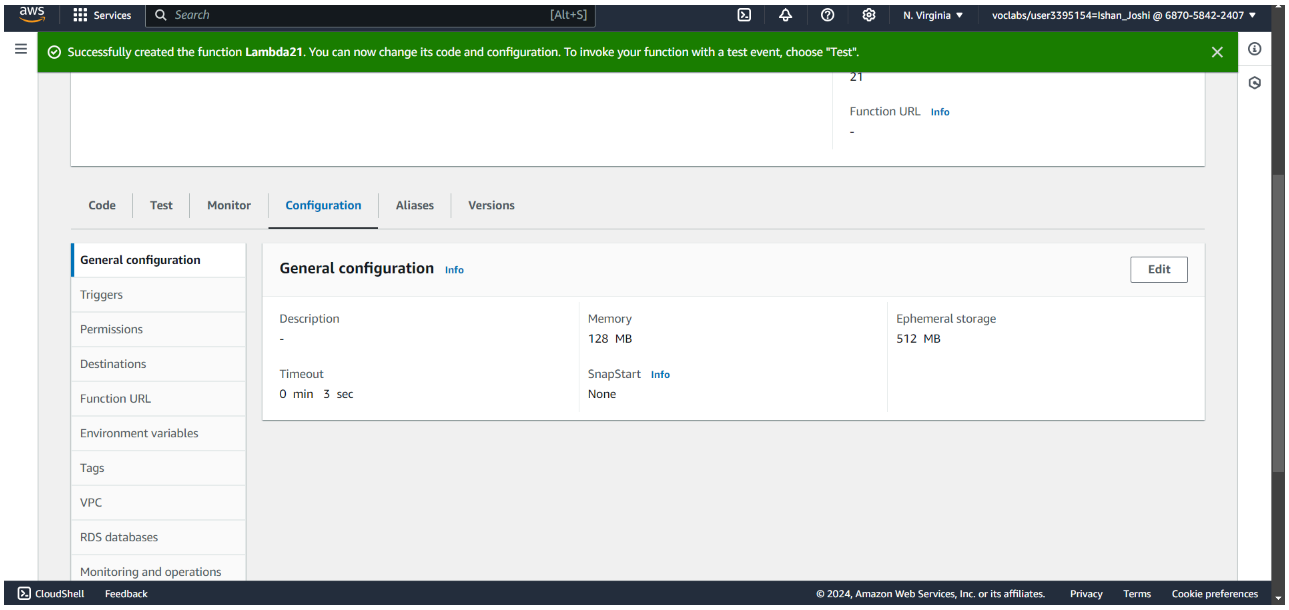


Successfully created the function **Lambda21**. You can now change its code and configuration. To invoke your function with a test event, choose "Test".

Code source

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

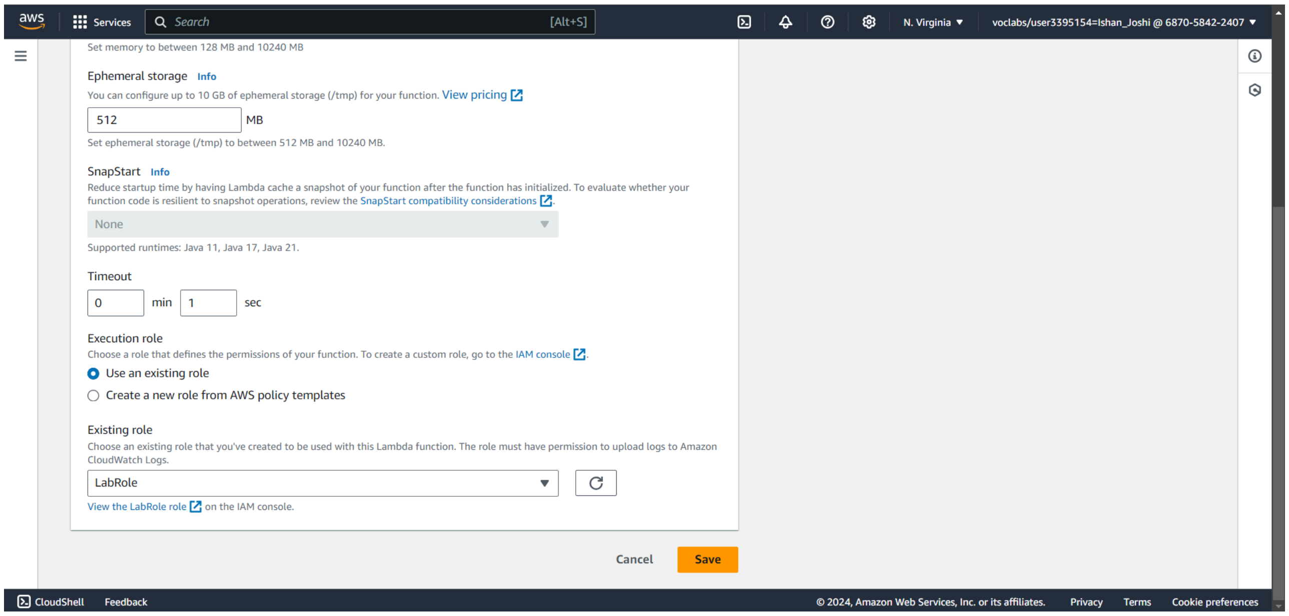
Step 3: The general configuration of the function is visible in the ‘Configuration’ tab. To change the configuration, click on ‘Edit’.


Successfully created the function **Lambda21**. You can now change its code and configuration. To invoke your function with a test event, choose "Test".

General configuration

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

You can change the various parameters of the configuration as per your needs. Here, we can change the ‘Timeout’ period to 1 second as it’s sufficient for our function for now. ‘Timeout’ is the time for which a function can be running before it gets forcibly terminated.


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.

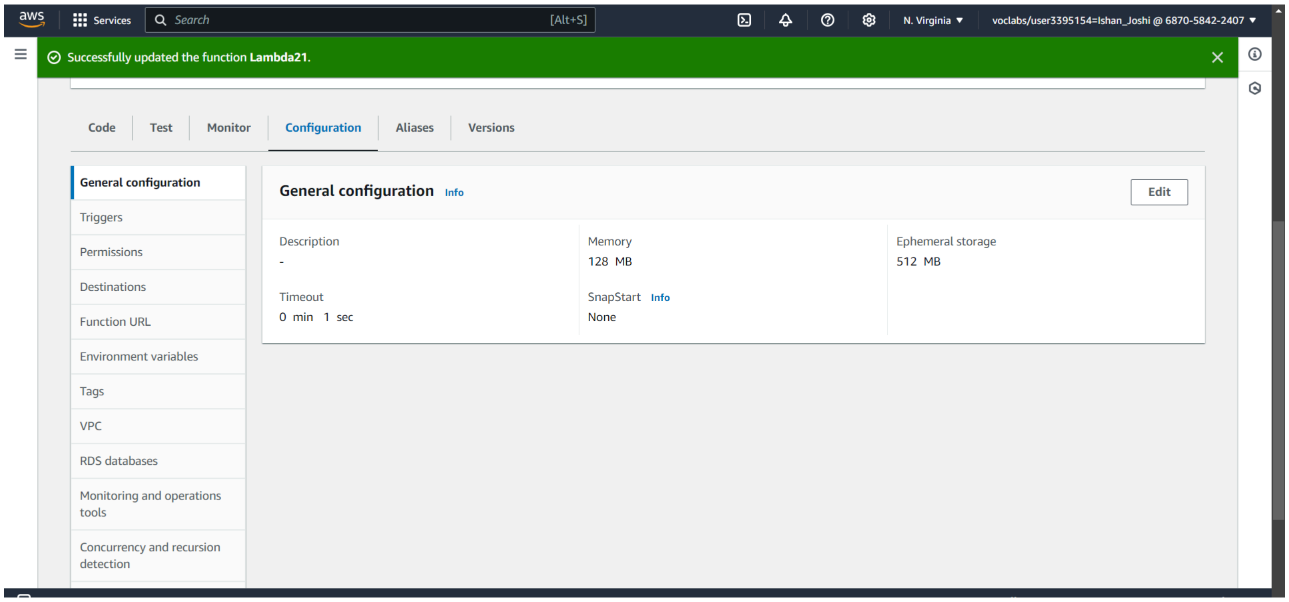
Timeout
0 min 1 sec

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.
LabRole
View the LabRole role on the IAM console.

After making the required changes, click on ‘Save’.

The changes in the general configuration are visible in the function.

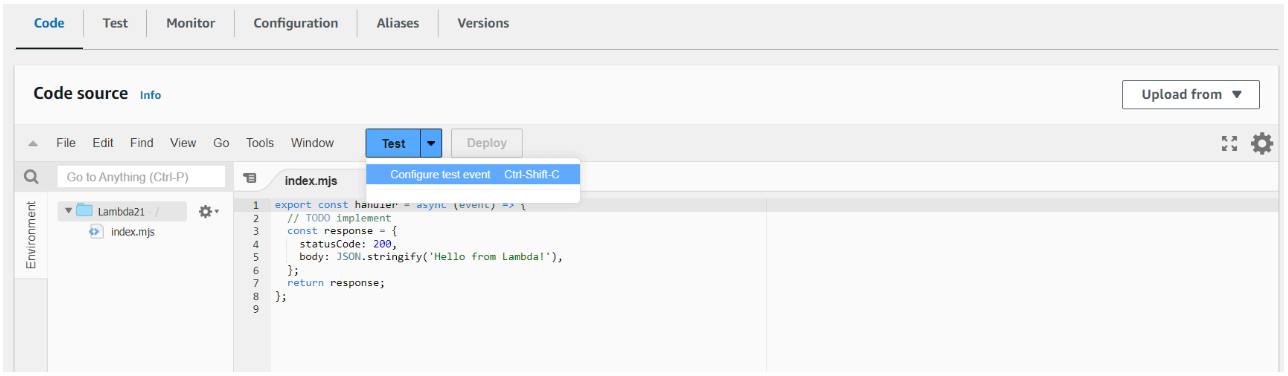


Successfully updated the function Lambda21.

General configuration

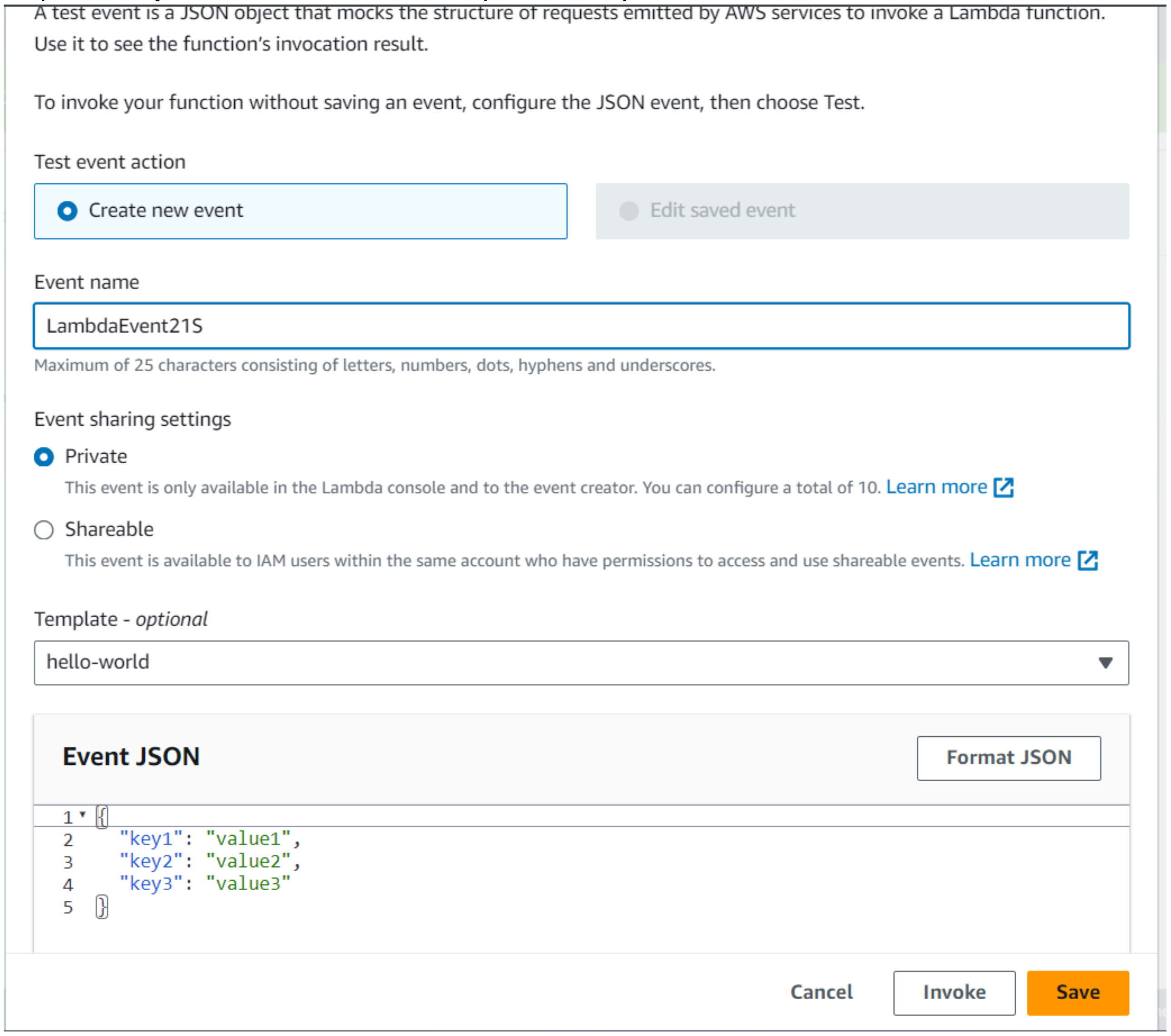
Description	Memory	Ephemeral storage
-	128 MB	512 MB
Timeout	SnapStart	
0 min 1 sec	None	

Step 4: In the ‘Code source’ section, click on the arrow next to the ‘Test’ button and click on ‘Configure test event’.



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

Step 5: Give your test event a name, keep all other options as default and click on ‘Save’.



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

LambdaEvent21S

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*

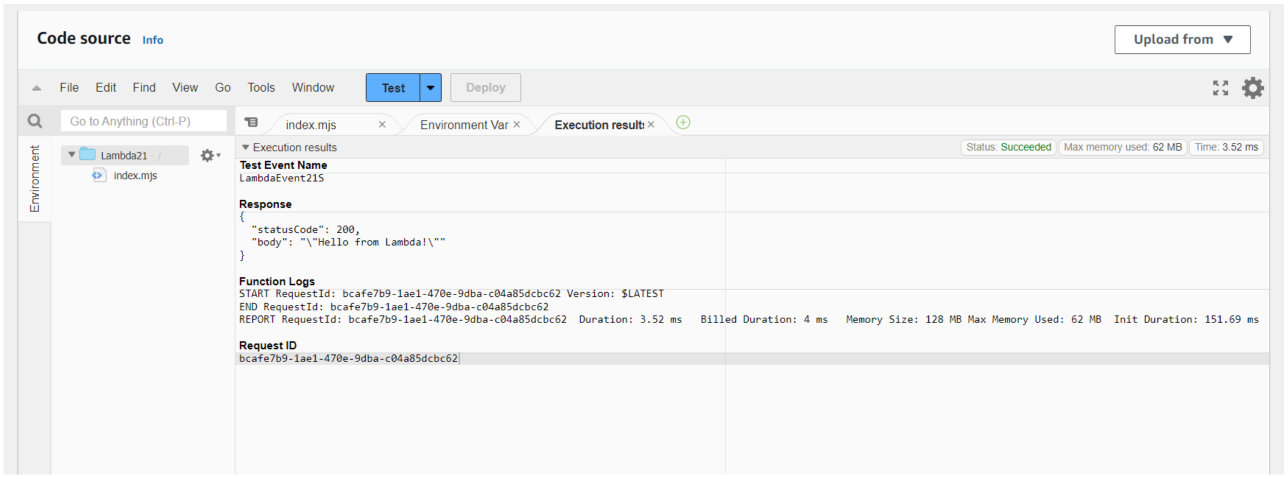
hello-world

Event JSON

Format JSON

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

Step 6: Click on the ‘Test’ button. The following output appears.



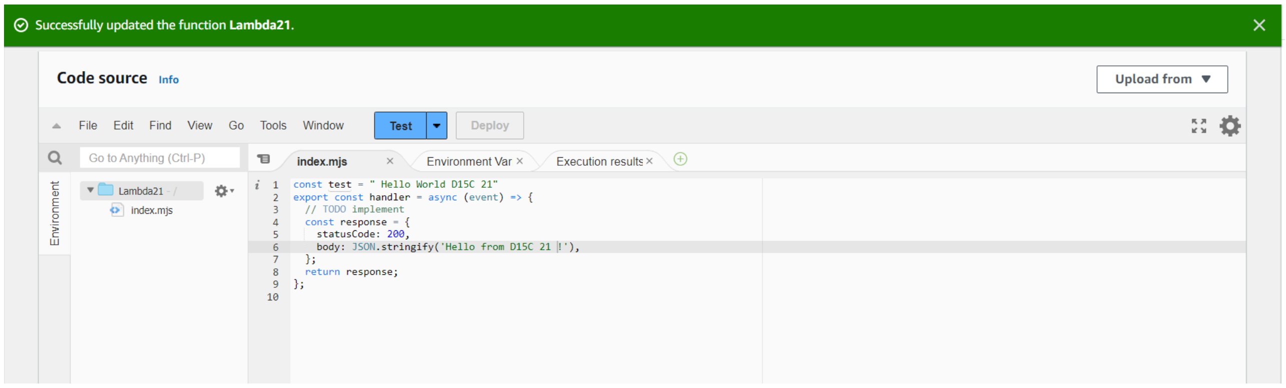
Test Event Name
```
LambdaEvent21S
```

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

Function Logs
```
START RequestId: bcafe7b9-1ae1-470e-9dba-c04a85dcbc62 Version: $LATEST
END RequestId: bcafe7b9-1ae1-470e-9dba-c04a85dcbc62
REPORT RequestId: bcafe7b9-1ae1-470e-9dba-c04a85dcbc62  Duration: 3.52 ms   Billed Duration: 4 ms   Memory Size: 128 MB Max Memory Used: 62 MB  Init Duration: 151.69 ms
```

Request ID
```
bcafe7b9-1ae1-470e-9dba-c04a85dcbc62
```

Step 7: You can make changes in the code to observe the difference in the output. Here, we change the code to display a different string as such:-

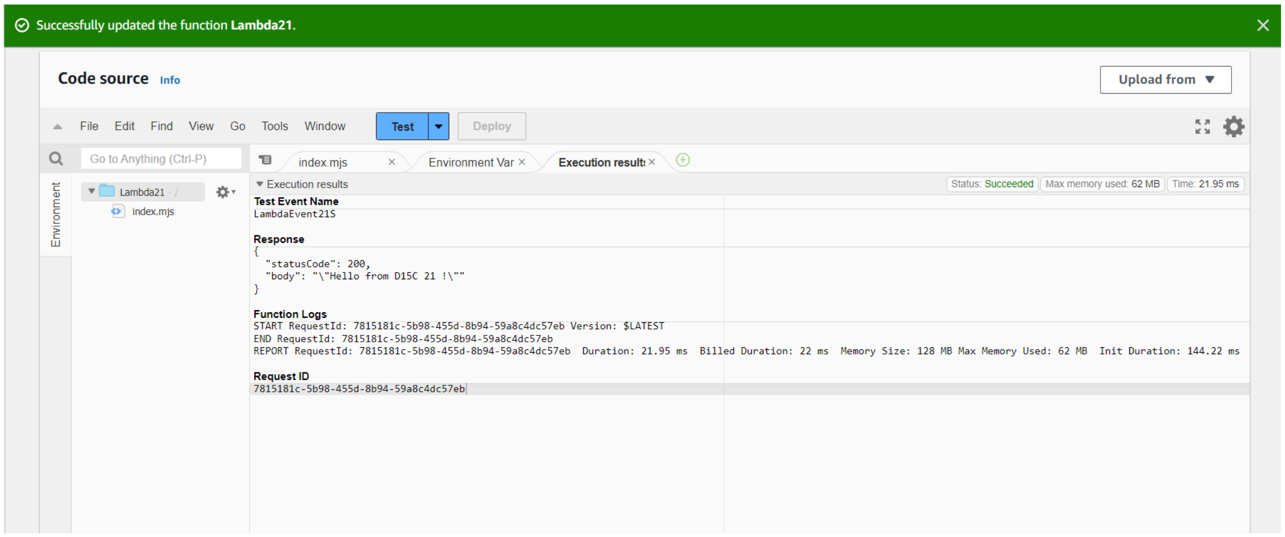


Successfully updated the function **Lambda21**.

```
const test = " Hello World D15C 21"
export const handler = async (event) => {
  // TODO implement
  const response = {
    statusCode: 200,
    body: JSON.stringify('Hello from D15C 21 !'),
  };
  return response;
};
```

Once the changes are made, click on ‘Deploy’.

Step 8: Click on ‘Test’ and observe how the output after the changes differs from the output before the changes.


Successfully updated the function **Lambda21**.

Code source Info

Status: Succeeded | Max memory used: 62 MB | Time: 21.95 ms

Test Event Name
```
LambdaEvent21S
```

Response
```
{
  "statusCode": 200,
  "body": "\"Hello from D15C 21 !\""
}
```

Function Logs
```
START RequestId: 7815181c-5b98-455d-8b94-59a8c4dc57eb Version: $LATEST
END RequestId: 7815181c-5b98-455d-8b94-59a8c4dc57eb
REPORT RequestId: 7815181c-5b98-455d-8b94-59a8c4dc57eb  Duration: 21.95 ms  Billed Duration: 22 ms  Memory Size: 128 MB Max Memory Used: 62 MB  Init Duration: 144.22 ms
```

Request ID
```
7815181c-5b98-455d-8b94-59a8c4dc57eb
```

Conclusion: This experiment provided a comprehensive understanding of AWS Lambda's capabilities. By developing a Node.js-based Lambda function, we explored its configuration options and execution process. Through the creation and execution of test events, we gained insights into how Lambda functions respond to various inputs and how changes in configuration impact their behavior. These findings highlight the flexibility and scalability of Lambda for serverless applications.