

AdvanceDevOps Experiment No: 12

Aim: To create a Lambda function which will log “An Image has been added” once you add an object to a specific bucket in S3

Theory:**AWS Lambda and S3 Integration:**

AWS Lambda allows you to execute code in response to various events, including those triggered by Amazon S3. When an object is added to an S3 bucket, it can trigger a Lambda function to execute, allowing for event-driven processing without managing servers.

Workflow:**1. Create an S3 Bucket:**

- First, create an S3 bucket that will store the objects. This bucket will act as the trigger source for the Lambda function.

2. Create the Lambda Function:

- Set up a new Lambda function using AWS Lambda’s console. You can choose a runtime environment like Python, Node.js, or Java.
- Write code that logs a message like “An Image has been added” when triggered.

3. Set Up Permissions:

- Ensure that the Lambda function has the necessary permissions to access S3. You can do this by attaching an IAM role with policies that allow reading from the bucket and writing logs to CloudWatch.

4. Configure S3 Trigger:

- Link the S3 bucket to the Lambda function by setting up a trigger. Specify that the function should be triggered when an object is created in the bucket (e.g., when an image is uploaded).

5. Test the Setup:

- Upload an object (e.g., an image) to the S3 bucket to test the trigger. The Lambda function should execute and log the message “An Image has been added” in AWS CloudWatch Logs.

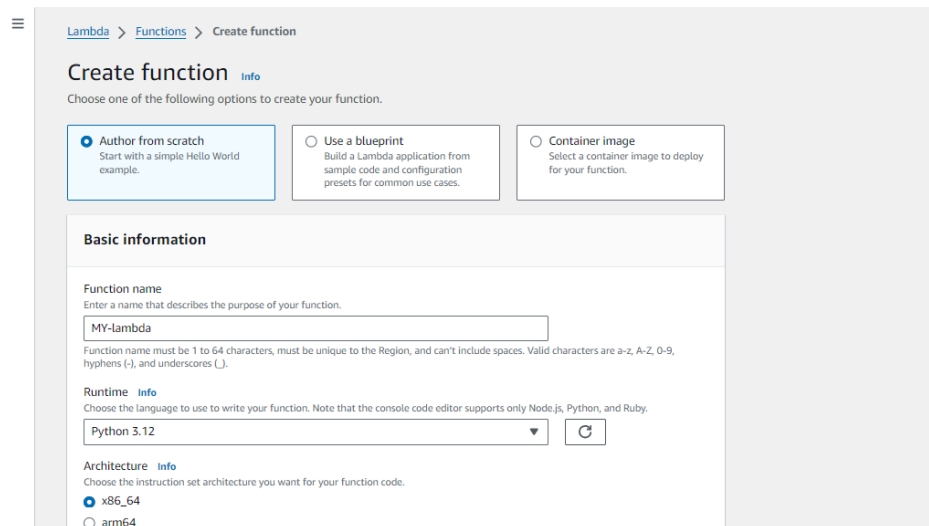
Steps To create the lambda function:

Step 1: Login to your AWS Personal account. Now open S3 from services and click on create S3 bucket.

Step 2: I have used already created bucket v2bucket if you dont have then you can create a basic bucket for this experiment .

Step 3: Open lambda console and click on create function button.

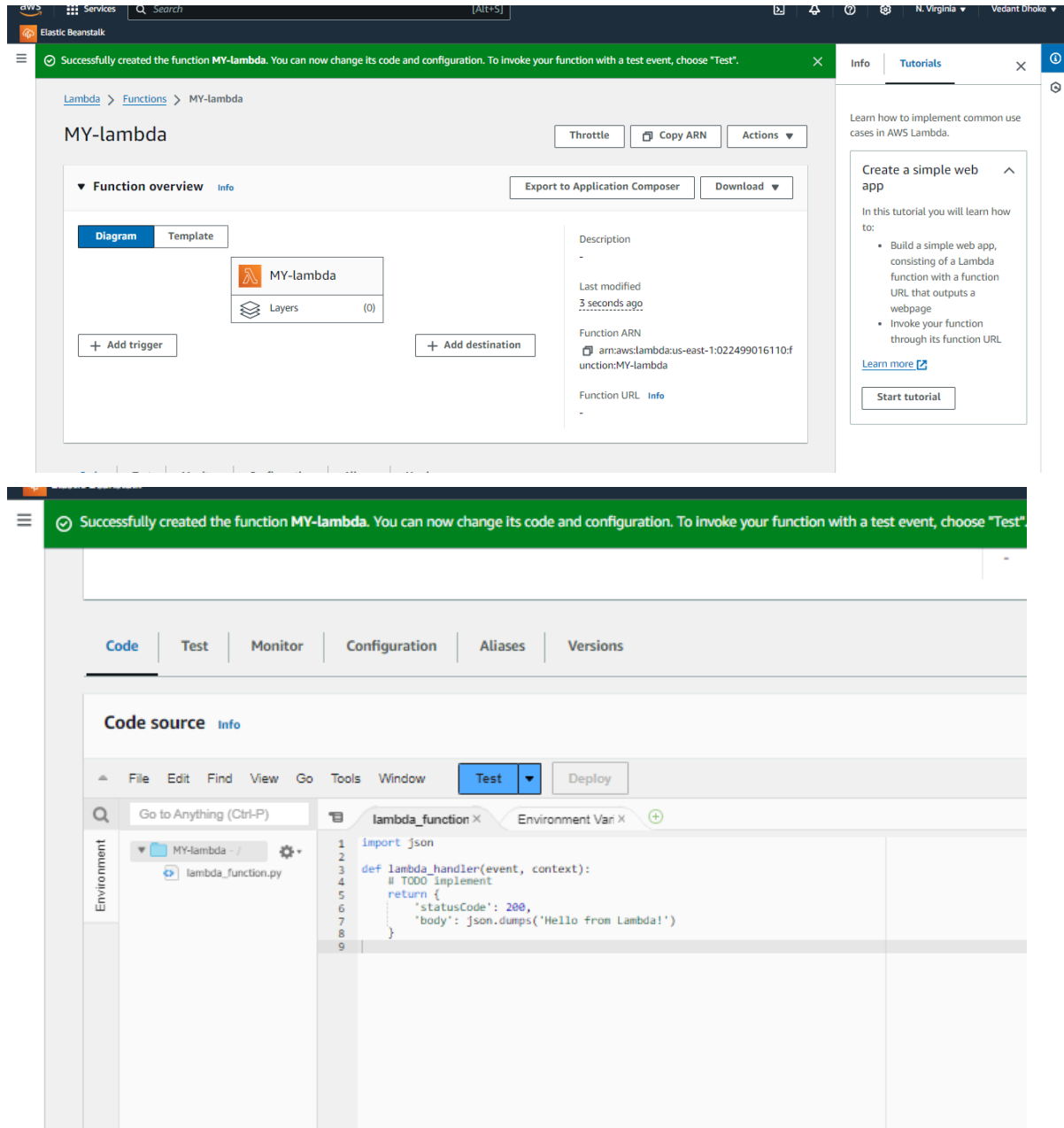
Step 4: Now Give a name to your Lambda function, Select the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby. So will select Python 3.12 , Architecture as x86, and Exceution role to Create a new role with basic Lambda permissions.



The screenshot displays the AWS Lambda 'Create function' interface. At the top, there's a breadcrumb trail: 'Lambda > Functions > Create function'. The main heading is 'Create function' with an 'Info' link. Below this, a prompt says 'Choose one of the following options to create your function.' There are three radio button options: 'Author from scratch' (selected, with a sub-note 'Start with a simple Hello World example.'), 'Use a blueprint' (with a sub-note 'Build a Lambda application from sample code and configuration presets for common use cases.'), and 'Container image' (with a sub-note 'Select a container image to deploy for your function.').

Below the options is the 'Basic information' section. It contains three main fields:

- Function name:** A text input field containing 'MY-lambda'. A sub-note states: '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 (_).'
- Runtime:** A dropdown menu showing 'Python 3.12'. A sub-note says: 'Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.' There is a refresh icon to the right of the dropdown.
- Architecture:** Radio button options for 'x86_64' (selected) and 'arm64'. A sub-note says: 'Choose the instruction set architecture you want for your function code.'



So See or Edit the basic settings go to configuration then click on edit general setting.

The screenshot shows the 'General configuration' tab in the AWS Lambda console. The left sidebar lists various configuration options: General configuration, Triggers, Permissions, Destinations, Function URL, Environment variables, Tags, VPC, and RDS databases. The main panel displays the 'General configuration' for the function 'Bhushan_Lambda'. It includes an 'Edit' button in the top right corner. The configuration details are as follows:

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

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.

The screenshot shows the 'Edit basic settings' page in the AWS Lambda console for the function 'Bhushan_Lambda'. The page includes the following fields and options:

- Description - optional:** A text input field containing 'Basic Settings'.
- Memory:** A dropdown menu set to '128 MB'. Below it, a note states: 'Your function is allocated CPU proportional to the memory configured. Set memory to between 128 MB and 10240 MB.'
- Ephemeral storage:** A dropdown menu set to '512 MB'. Below it, a note states: 'You can configure up to 10 GB of ephemeral storage (/tmp) for your function. View pricing. Set ephemeral storage (/tmp) to between 512 MB and 10240 MB.'
- SnapStart:** A dropdown menu set to 'None'. Below it, a note states: '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.'
- Timeout:** Two input fields for '0 min' and '1 sec'.
- Execution role:** Two radio buttons: 'Use an existing role' (selected) and 'Create a new role from AWS policy templates'.

Step 5: Now Click on the Test tab then select Create a new event, give a name to the event and select Event Sharing to private, and select s3 put template.

Test event Info

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

event-exp_12

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

This event is available to IAM users within the same account who have permissions to access and use shareable events. [Learn more](#)

Template - optional

s3-put

Event JSON [Format JSON](#)

```
2 * "Records": [  
3 {  
4   "eventVersion": "2.0",  
5   "eventSource": "aws:s3",  
6   "awsRegion": "us-east-1",  
7   "eventTime": "1970-01-01T00:00:00.000Z",  
8   "eventName": "ObjectCreated:Put",  
9   "userIdentity": {  
10    "principalId": "EXAMPLE"  
11  },  
12  "requestParameters": {  
13    "sourceIPAddress": "127.0.0.1"  
14  },  
15  "responseElements": {  
16    "x-amz-request-id": "EXAMPLE123456789",  
17    "x-amz-id-2": "EXAMPLE123/5678abcdeghijklmnopqrstuvwxyzABCDEFGHIH"  
18  },  
19  "s3": {  
20    "s3SchemaVersion": "1.0",  
21    "configurationId": "testconfigrule",  
22    "bucket": {  
23      "name": "example-bucket",  
24      "ownerIdentity": {  
25        "principalId": "EXAMPLE"  
26      },  
27      "arn": "arn:aws:s3:::example-bucket"  
28    },  
29    "object": {  
30      "key": "testK2fkey",  
31      "size": 1024,  
32    }  
33  }  
34 }  
35 ]
```

1:1 JSON Spaces: 2

Tutorials

Learn how to implement common use cases in AWS Lambda.

Create a simple web app

In this tutorial you will learn how to:

- Build a simple web app, consisting of a Lambda function with a function URL that outputs a webpage
- Invoke your function through its function URL

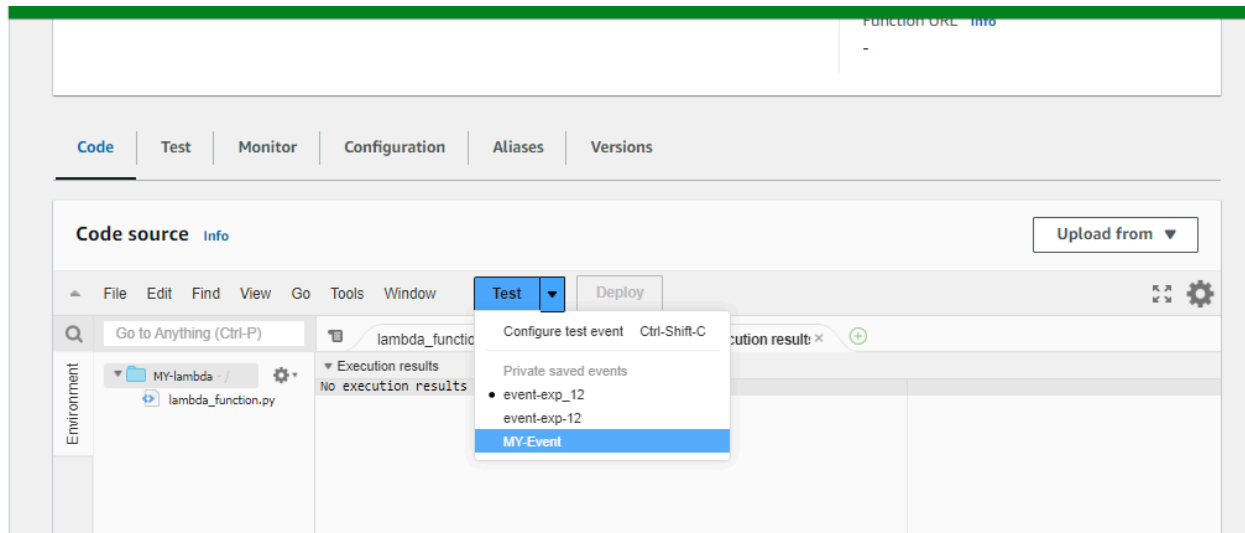
[Learn more](#)

[Start tutorial](#)

CloudShell Feedback

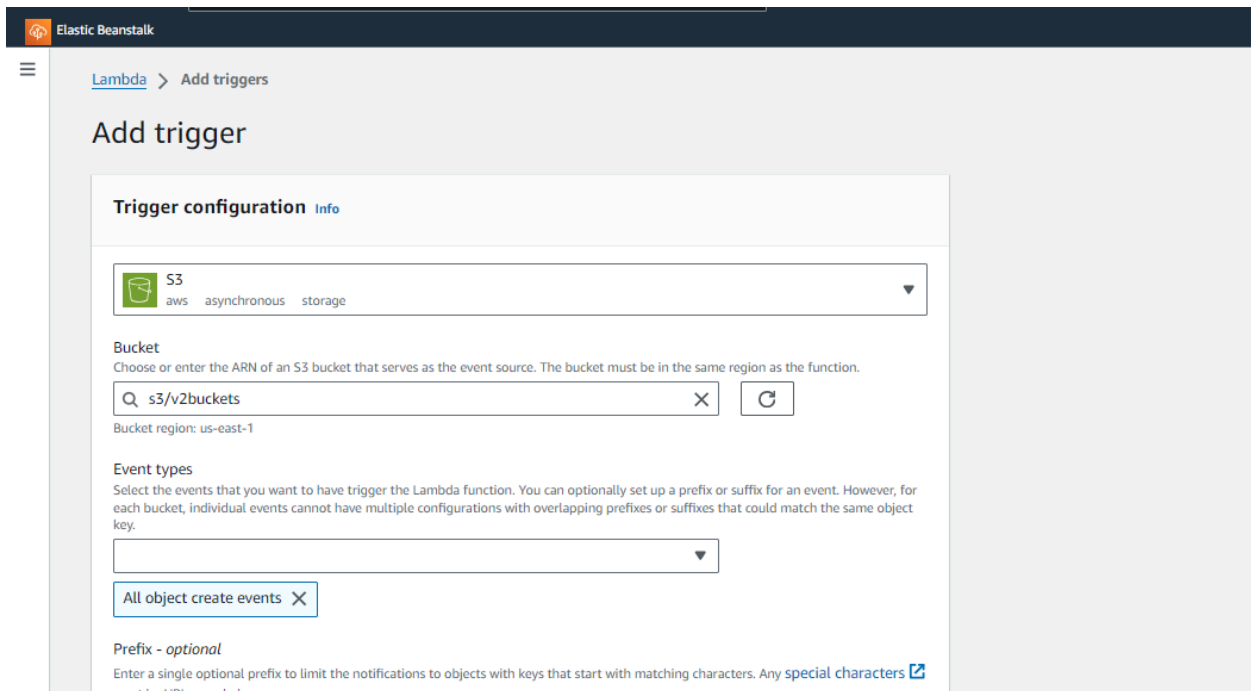
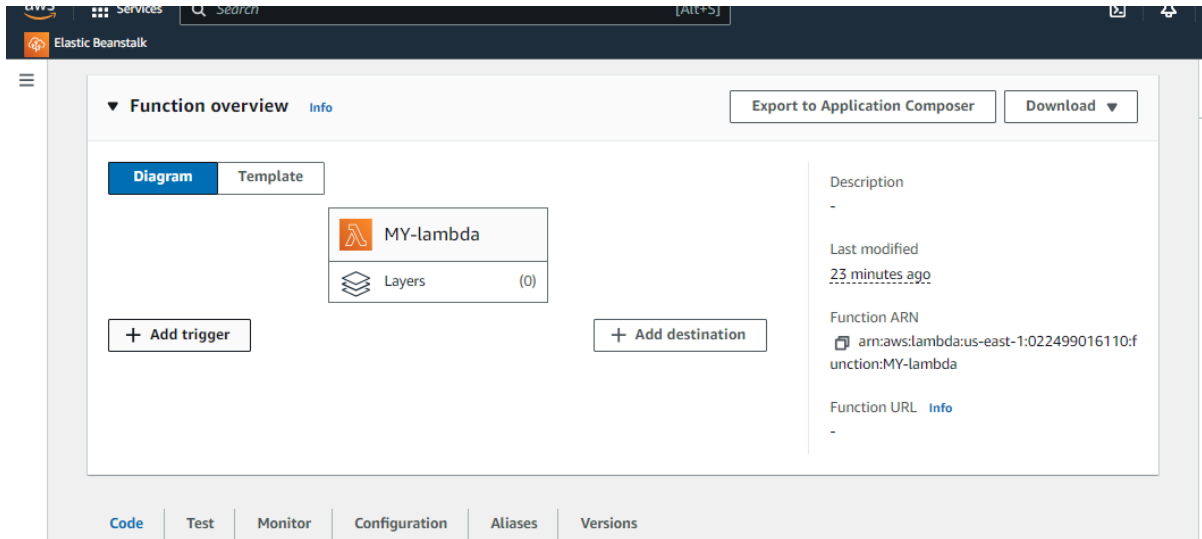
© 2024, Amazon Web Services, Inc. or its affiliates. [Privacy](#) [Terms](#) [Cookie preferences](#)

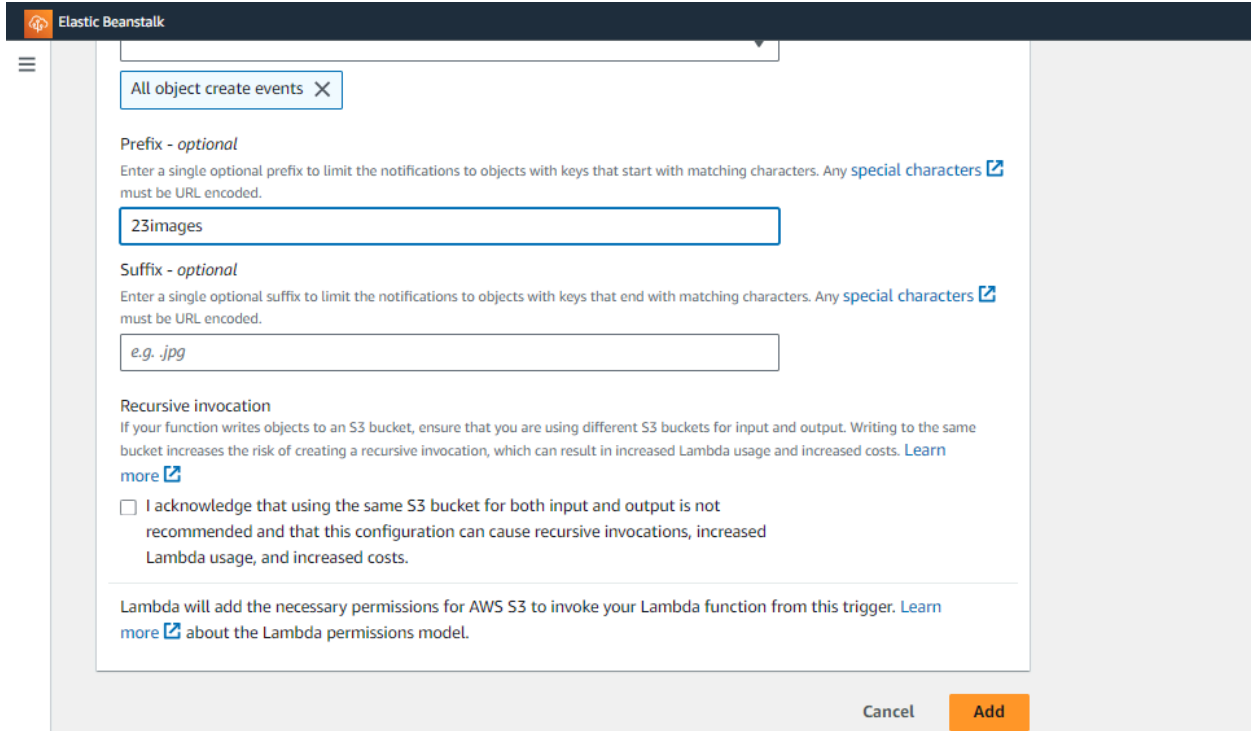
Step 6: Now In Code section select the created event from the dropdown .



Step 7: Now In the Lambda function click on add trigger.

Now select the source as S3 then select the bucket name from the dropdown, keep other things to default and also you can add prefix to





The screenshot shows the 'Add trigger' configuration window in the AWS Elastic Beanstalk console. It includes fields for a prefix (set to '23images') and a suffix (set to '.jpg'). There is a checkbox for 'Recursive invocation' which is currently unchecked. At the bottom, there are 'Cancel' and 'Add' buttons.

Prefix - optional
Enter a single optional prefix to limit the notifications to objects with keys that start with matching characters. Any [special characters](#) must be URL encoded.

Suffix - optional
Enter a single optional suffix to limit the notifications to objects with keys that end with matching characters. Any [special characters](#) must be URL encoded.

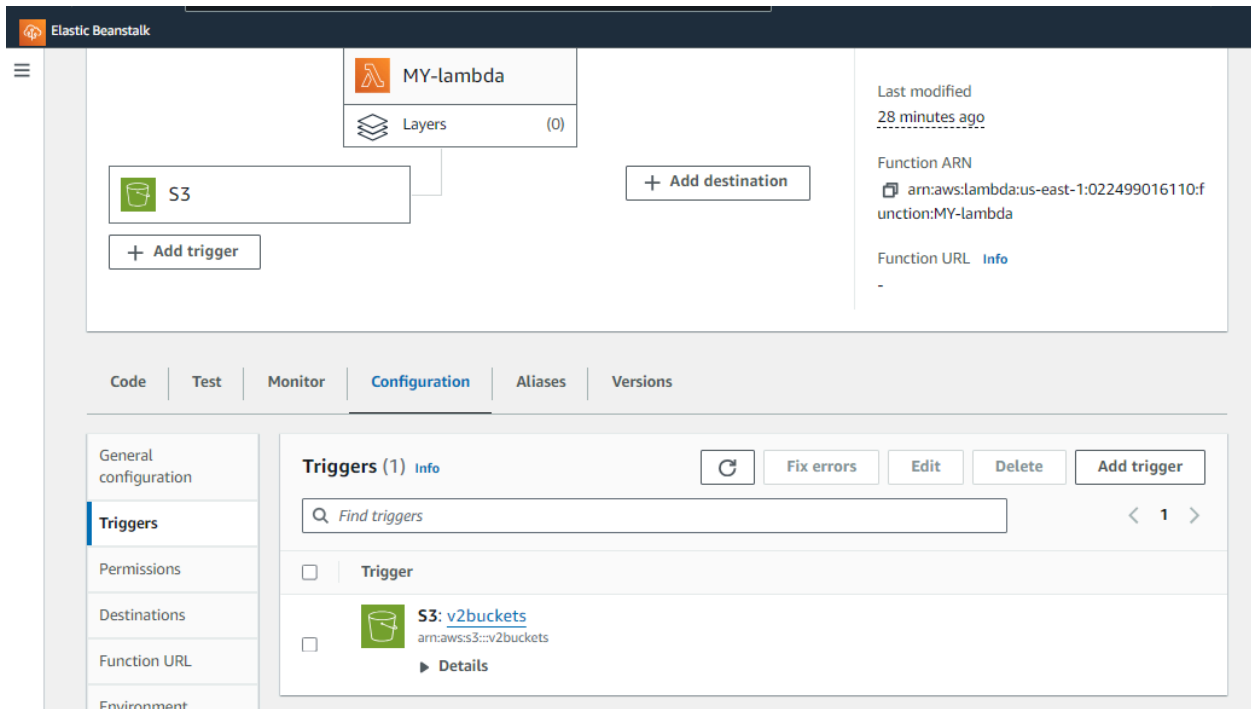
Recursive invocation
If your function writes objects to an S3 bucket, ensure that you are using different S3 buckets for input and output. Writing to the same bucket increases the risk of creating a recursive invocation, which can result in increased Lambda usage and increased costs. [Learn more](#)

☐ I acknowledge that using the same S3 bucket for both input and output is not recommended and that this configuration can cause recursive invocations, increased Lambda usage, and increased costs.

Lambda will add the necessary permissions for AWS S3 to invoke your Lambda function from this trigger. [Learn more](#) about the Lambda permissions model.

Cancel Add

Step 8: Now Write code that logs a message like “An Image has been added” when triggered. Save the file and click on deploy.



The screenshot shows the 'Configuration' tab for the 'MY-lambda' function in the AWS Elastic Beanstalk console. It displays the 'Triggers' section with a search bar and a list of triggers. The first trigger is 'S3: v2buckets' with the ARN 'arn:aws:s3::v2buckets'. There are buttons for 'Add trigger', 'Fix errors', 'Edit', and 'Delete'.

MY-lambda
Layers (0)

S3 + Add trigger

+ Add destination

Last modified 28 minutes ago

Function ARN
arn:aws:lambda:us-east-1:022499016110:function:MY-lambda

Function URL [Info](#)

Code Test Monitor **Configuration** Aliases Versions

General configuration
Triggers
Permissions
Destinations
Function URL
Environment

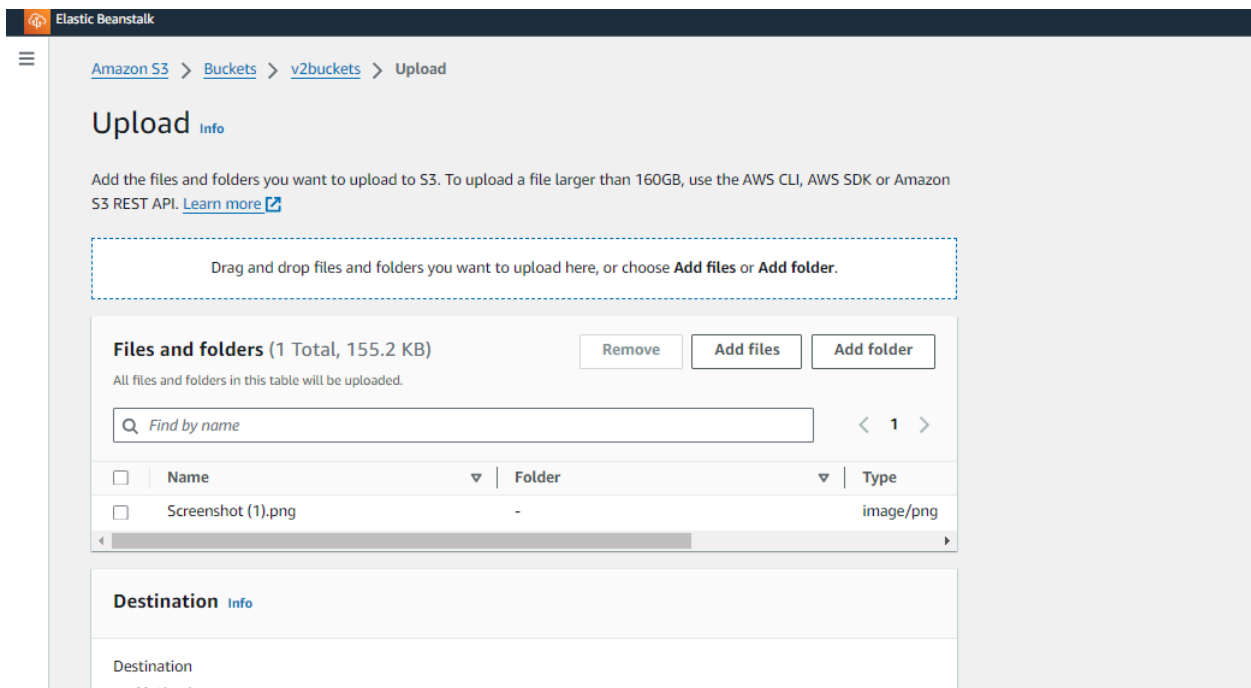
Triggers (1) Info [Info](#) [Refresh](#) [Fix errors](#) [Edit](#) [Delete](#) [Add trigger](#)

Find triggers

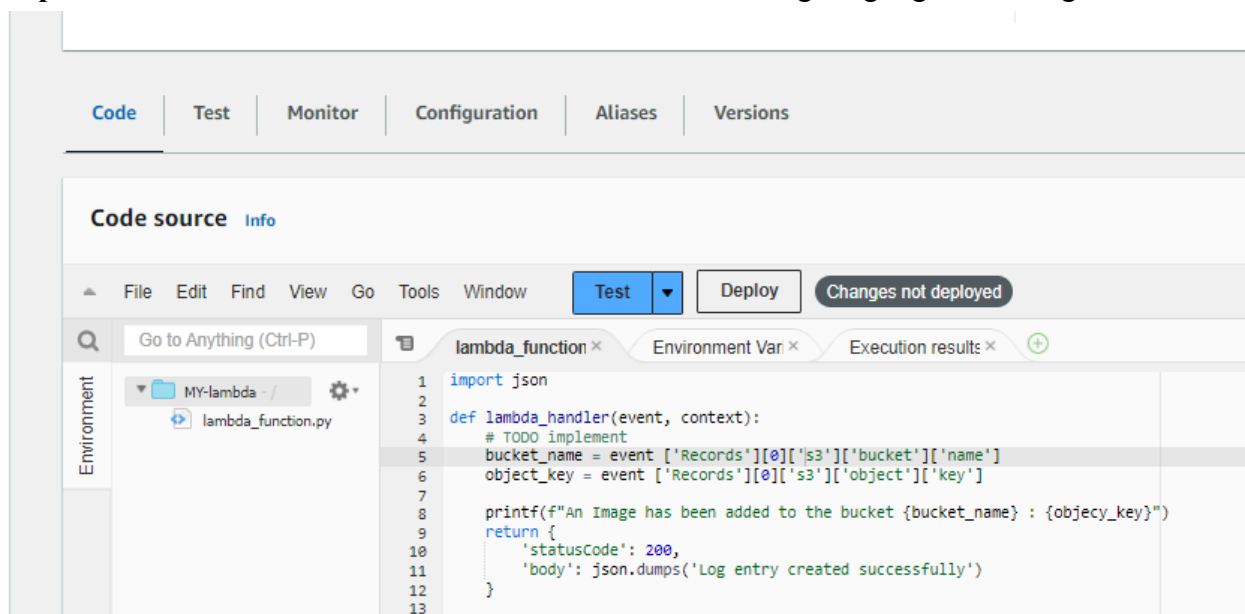
☐ Trigger

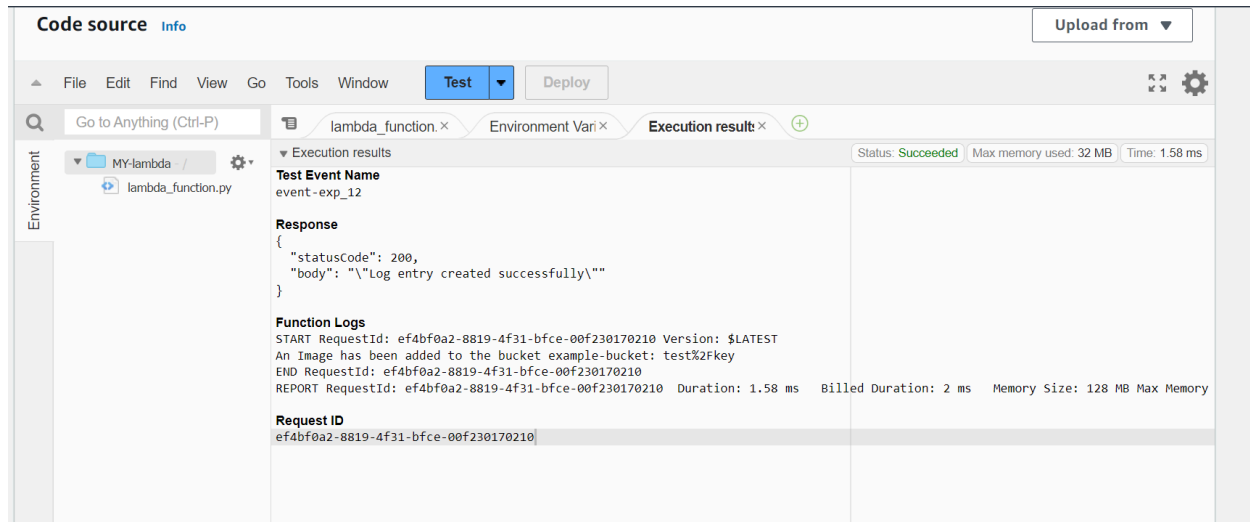
☐ **S3: v2buckets**
arn:aws:s3::v2buckets
[Details](#)

Step 9: Now upload any image to the bucket.

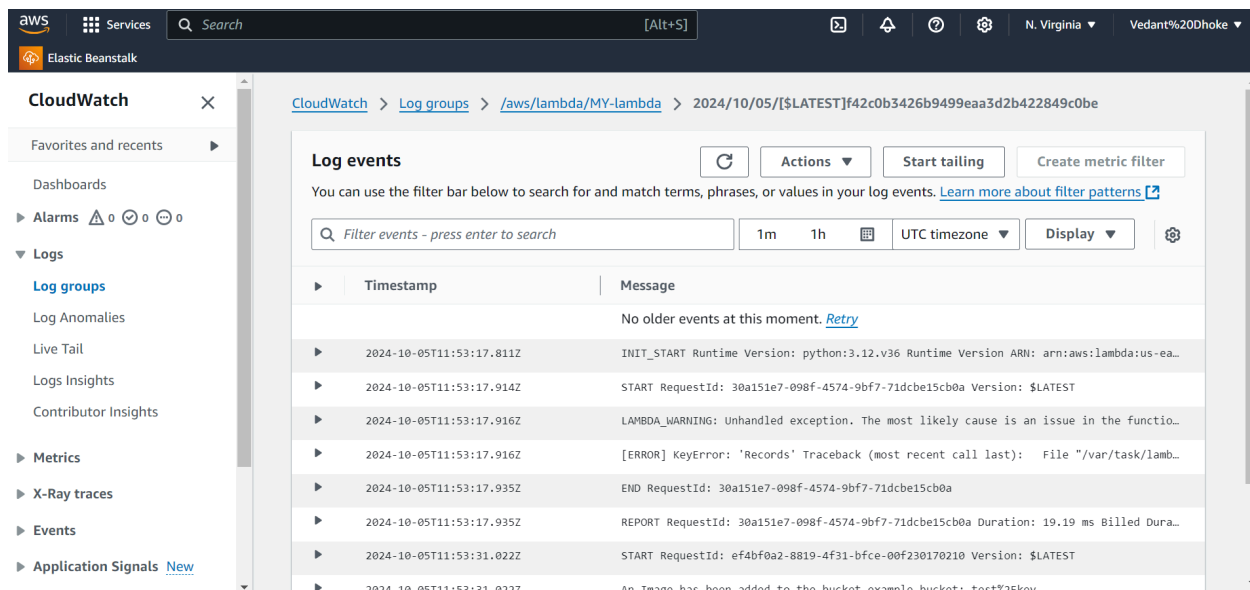


Step 10: Now to click on test in lambda to check whether it is giving log when image is added to S3.





Step 11: Now Lets see the log on Cloud watch.To see it go to monitor section and then click on view cloudwatch logs.



Conclusion:

In this experiment, we successfully created a Lambda function designed to log the message “An Image has been added” upon the addition of an object to a specified S3 bucket. This process deepened our understanding of integrating AWS Lambda with S3 for event-driven applications. However, we encountered several challenges that provided valuable learning experiences:

- **S3 Bucket Configuration:** Setting up the S3 bucket correctly was crucial. We had to ensure that the bucket's public access settings were appropriately configured to allow the Lambda function to trigger without compromising security. Misconfigurations here could have led to access issues.
- **Trigger Setup:** Configuring the S3 bucket as a trigger for the Lambda function required careful attention. Selecting the correct bucket and event type was essential to ensure that the function executed as intended. Any oversight could result in the function not being triggered.
- **Testing and Monitoring:** After uploading images to the S3 bucket, verifying that the Lambda function logged the appropriate messages required us to navigate CloudWatch logs. Initially, we struggled to find the correct log group, which delayed our testing process.