

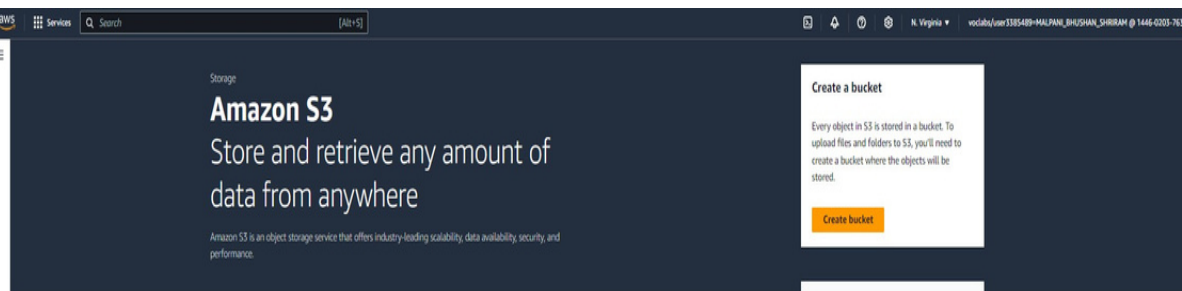
Aim: To create a Lambda function which will log “An Image has been added” once you add an object to a specific bucket in S3.

Prerequisites:

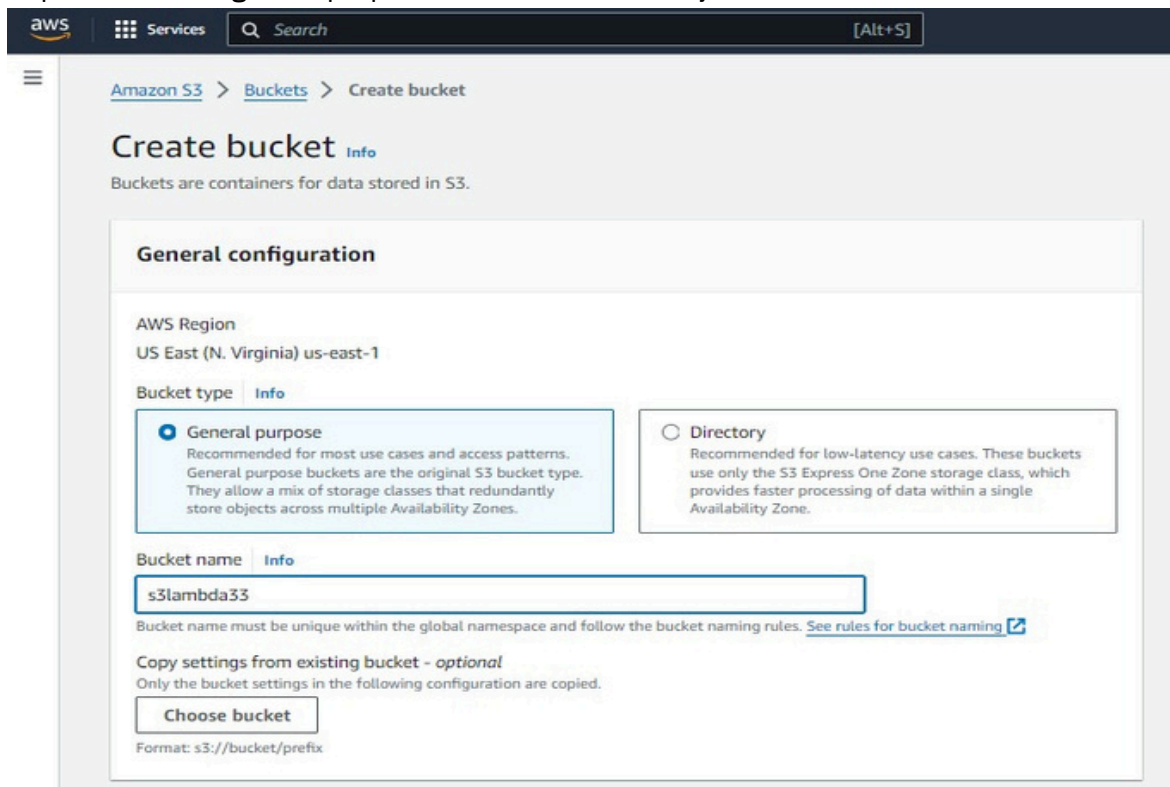
- 1) AWSaccount(academypreferable)
- 2) Lambdafunction(createdinthepreviousexperiment).

Step 1: Create a s3 bucket.

- 1) SearchforS3bucketintheservicessearch.Thenclickoncreatebucket.



- 2) Keepthebucketasageneralpurposebucket.Giveanametoyourbucket.



3) Uncheckblockallpublicaccess.

Block Public Access settings for this bucket

Public access is granted to buckets and objects through access control lists (ACLs), bucket policies, access point policies, or all. In order to ensure that public access to this bucket and its objects is blocked, turn on Block all public access. These settings apply only to this bucket and its access points. AWS recommends that you turn on Block all public access, but before applying any of these settings, ensure that your applications will work correctly without public access. If you require some level of public access to this bucket or objects within, you can customize the individual settings below to suit your specific storage use cases. [Learn more](#)

☐ **Block all public access**
Turning this setting on is the same as turning on all four settings below. Each of the following settings are independent of one another.

☐

Block public access to buckets and objects granted through new access control lists (ACLs)
S3 will block public access permissions applied to newly added buckets or objects, and prevent the creation of new public access ACLs for existing buckets and objects. This setting doesn't change any existing permissions that allow public access to S3 resources using ACLs.

☐


Block public access to buckets and objects granted through any access control lists (ACLs)
S3 will ignore all ACLs that grant public access to buckets and objects.

☐

Block public access to buckets and objects granted through new public bucket or access point policies
S3 will block new bucket and access point policies that grant public access to buckets and objects. This setting doesn't change any existing policies that allow public access to S3 resources.

☐

Block public and cross-account access to buckets and objects through any public bucket or access point policies
S3 will ignore public and cross-account access for buckets or access points with policies that grant public access to buckets and objects.



Turning off block all public access might result in this bucket and the objects within becoming public
AWS recommends that you turn on block all public access, unless public access is required for specific and verified use cases such as static website hosting.
☐ I acknowledge that the current settings might result in this bucket and the objects within becoming public.

4) Keeping all other options same, click on create. This would create your bucket. Now click on the name of the bucket.

Successfully created bucket "s3lambda33"
To upload files and folders, or to configure additional bucket settings, choose [View details](#).

Amazon S3 > Buckets

Account snapshot - updated every 24 hours

[View Storage Lens dashboard](#)

General purpose buckets

Directory buckets

General purpose buckets (1) [Info](#) [All AWS Regions](#)

[Copy ARN](#) [Empty](#) [Delete](#) [Create bucket](#)

Find buckets by name

Name

AWS Region

IAM Access Analyzer

Creation date

s3lambda33

US East (N. Virginia) us-east-1

[View analyzer for us-east-1](#)

October 6, 2024, 22:53:44 (UTC+05:30)

5) Here,clickonupload,thenaddfiles.Selectanyimagethatyouwanttouploadinthebucket and click on upload.

s3lambda33 [Info](#)

Objects

Properties

Permissions

Metrics

Management

Access Points

Objects (0) [Info](#)

[Copy S3 URI](#) [Copy URL](#) [Download](#) [Open](#) [Delete](#) [Actions](#) [Create folder](#) [Upload](#)

Find objects by prefix

Name

Type

Last modified

Size

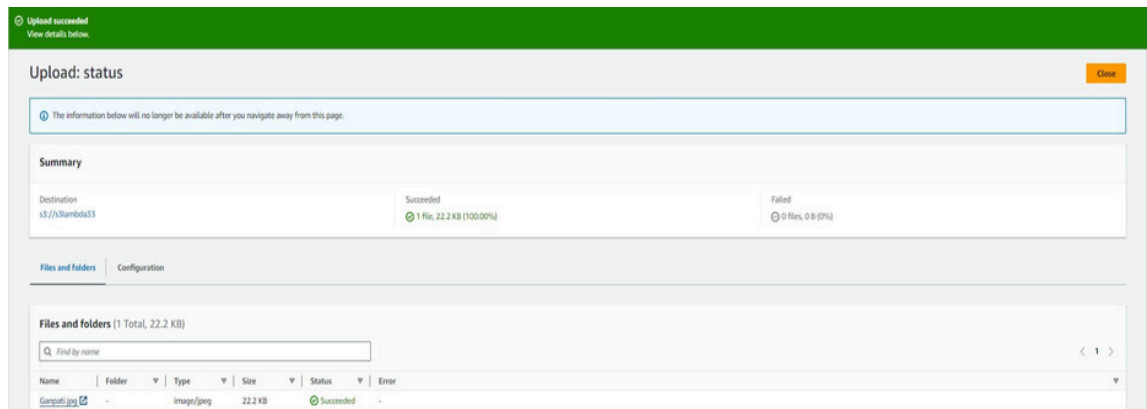
Storage class

No objects

You don't have any objects in this bucket.

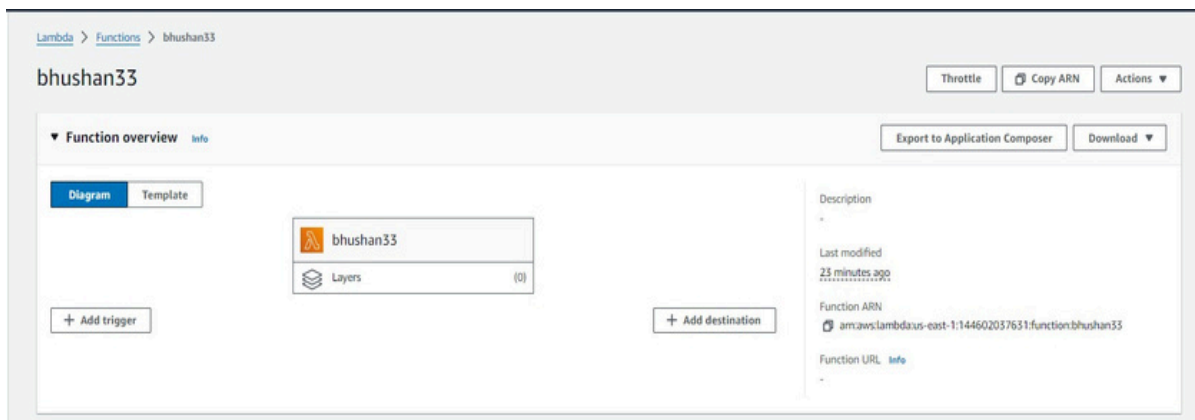
[Upload](#)

6) The image has been uploaded to the bucket.

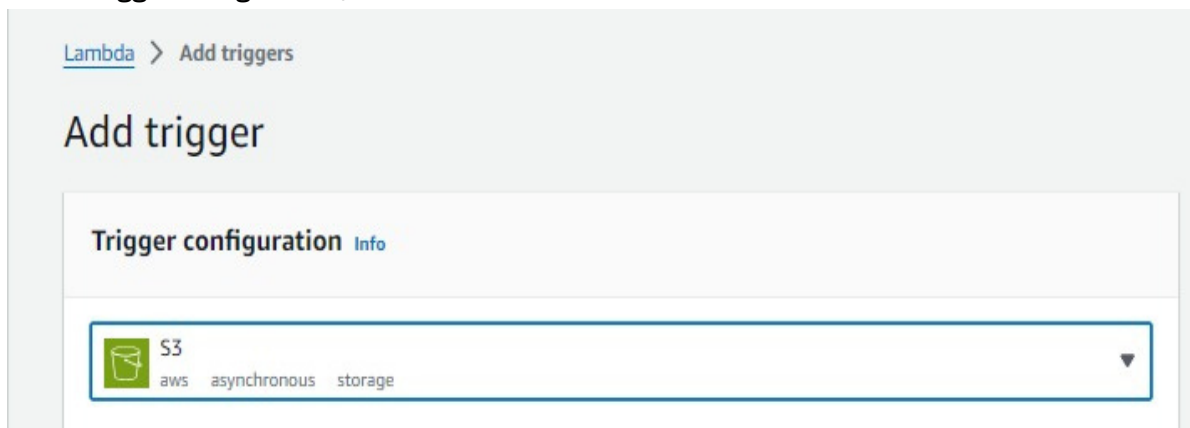


Step 2: Configure Lambda function

1) Go to the lambda function you had created before. (Services → Lambda → Click on name of function). Here, click on add trigger.




2) Under trigger configuration, search for S3 and select it.



Here, select the S3 bucket you created for this experiment. Acknowledge the condition given by AWS. then click on Add. This will add the S3 bucket trigger to your function.

Trigger configuration [Info](#)

 **S3**
aws asynchronous storage

Bucket
Choose or enter the ARN of an S3 bucket that serves as the event source. The bucket must be in the same region as the function.
 ✕ ↺
Bucket region: us-east-1

Event types
Select the events that you want to have trigger the Lambda function. You can optionally set up a prefix or suffix for an event. However, for each bucket, individual events cannot have multiple configurations with overlapping prefixes or suffixes that could match the same object key.

All object create events ✕

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

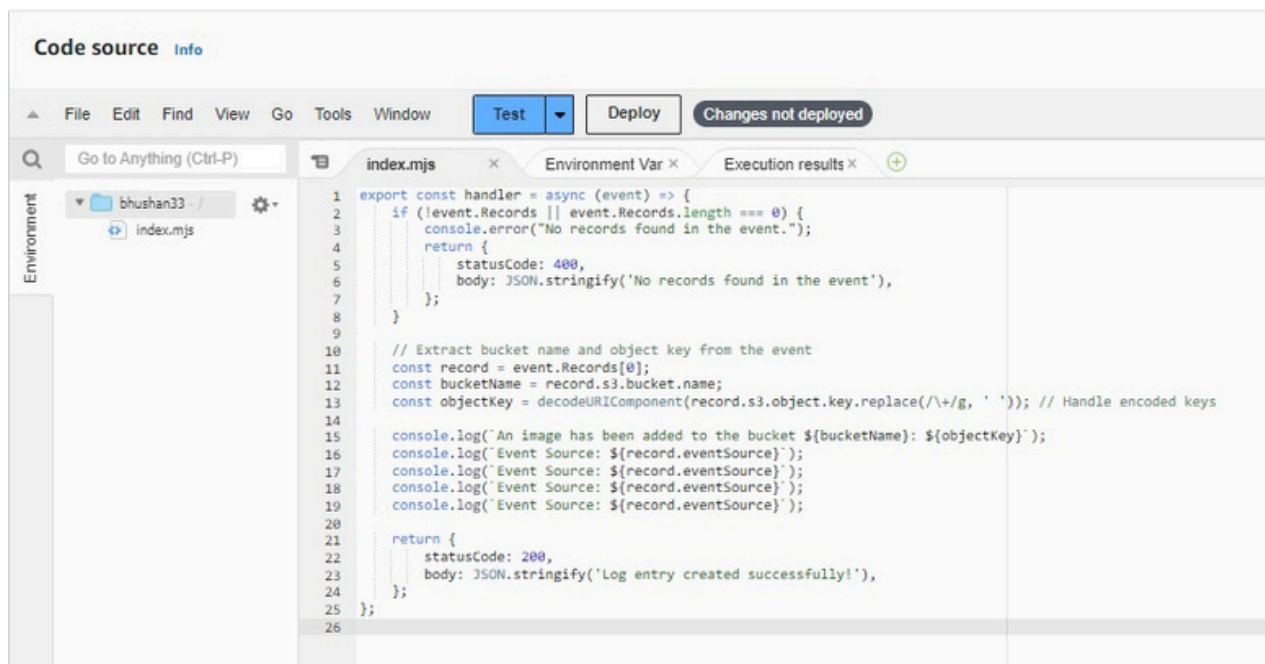
Scroll down to the code section of the function. Add the following javascript code to the code area by replacing the existing code.

```
export const handler = async (event) => { if
  (!event.Records || event.Records.length === 0) {
    console.error("No records found in the event.");
    return { statusCode: 400, body: JSON.stringify('No
      records
        found in the event')
    };
  }
}
```

```
// Extract bucket name and object key from the event
const record = event.Records[0];
const bucketName = record.s3.bucket.name;
const objectKey = decodeURIComponent(record.s3.object.key.replace(/\+/g, ' ')); // Handle encoded keys
```

```
console.log(`An image has been added to the bucket ${bucketName}: ${objectKey}`);
console.log(`Event Source: ${record.eventSource}`);
console.log(`Event Source: ${record.eventSource}`);
console.log(`Event Source: ${record.eventSource}`);
return {
  statusCode: 200,
  body: JSON.stringify('Log entry created successfully!')
};
```

This code checks for records in the event, extracts the bucket name and object key, logs the details, and returns a success message if an image is added to the bucket.



Now, click on the dropdown near test, then click on configure test event.

Div :D15C

- 6) Here, select edit saved event. Select the event that you had created before. Under Event JSON, paste the following code.

```
{
  "Records": [
    {
      "eventVersion": "2.0",
      "eventSource": "aws:s3",
      "awsRegion": "us-east-1",
      "eventTime": "1970-01-01T00:00:00.000Z",
      "eventName": "ObjectCreated:Put",
      "userIdentity": {
        "principalId": "EXAMPLE"
      },
      "requestParameters": {
        "sourceIPAddress": "127.0.0.1"
      },
      "responseElements": {
        "x-amz-request-id": "EXAMPLE123456789", "x-amz-id-2":
"EXAMPLE123/5678abcdefghijklambdaisawesome/mnopqrstuvwxyzABCDEFGH"
      },
      "s3": {
        "s3SchemaVersion": "1.0",
        "configurationId": "testConfigRule",

      "bucket": {
        "name": "example-bucket",
        "ownerIdentity": {
          "principalId": "EXAMPLE"
        },
        "arn": "arn:aws:s3:::example-bucket"
      },
      "object": {
```

```

    "key": "test%2Fkey",
    "size": 1024,
    "eTag": "0123456789abcdef0123456789abcdef",
    "sequencer": "0A1B2C3D4E5F678901"
  }
}
}
]
}

```

This JSON structure represents an S3 event notification triggered when an object is uploaded to an S3 bucket. It contains details about the event, including the bucket name (example-bucket), the object key (test/key), and metadata like the object's size, the event source (aws:s3), and the event time.

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, modify the event, then choose Test. Lambda uses the modified event to invoke your function, but does not overwrite the original event until you choose Save changes.

Test event action

☐ Create new event
 ☒ Edit saved event

Event name

lambda33

↕

↺

Delete

Event JSON

Format JSON

```

1 {
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": "EXAMPLE123/5678abcdefghijklambdaisawesome/mnopqrstuvwxyzABCDI
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": "test%2Fkey",

```

Cancel

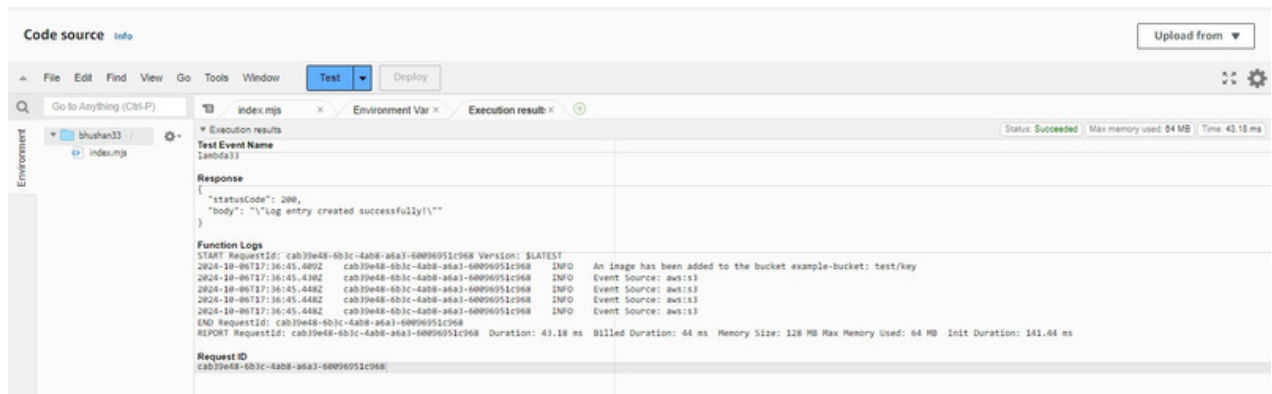
Invoke

Save

Save the changes. Then deploy the code changes by clicking on deploy.

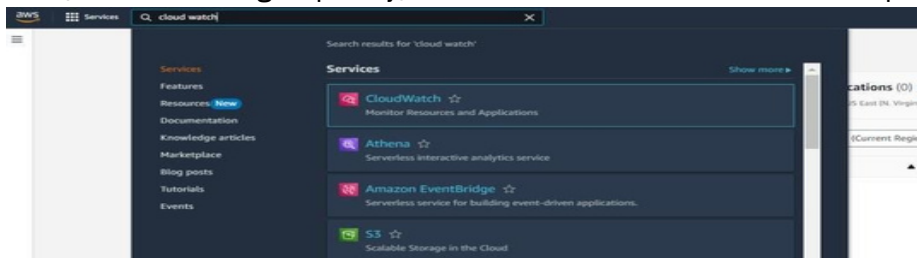
7) After deploying, click on Test. The console output shows that 'an image has been added to the bucket'

The JSON response shows that the log entry was created successfully.

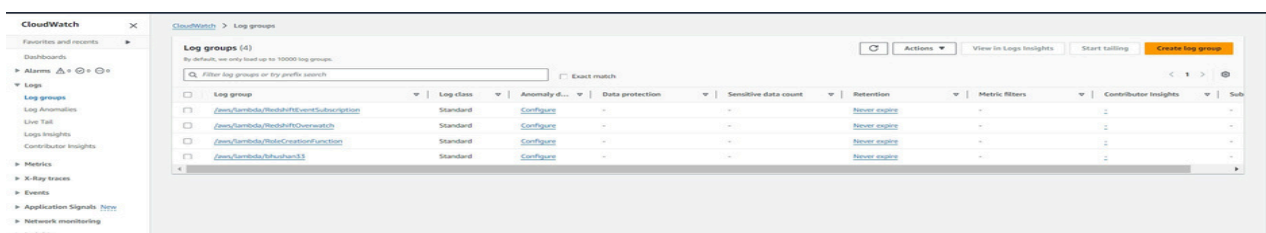


Step 3: Check the logs

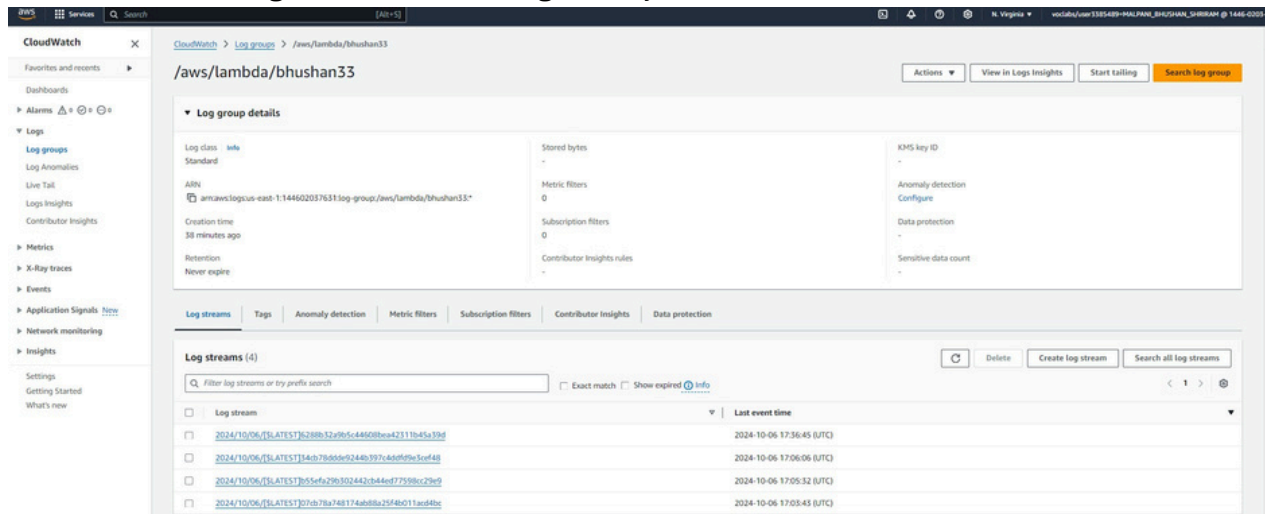
1) To check the log explicitly, search for CloudWatch on services and open it in a new tab.



2) Here, Click on Logs → Log Groups. Select the log that has the lambda function name you just ran.

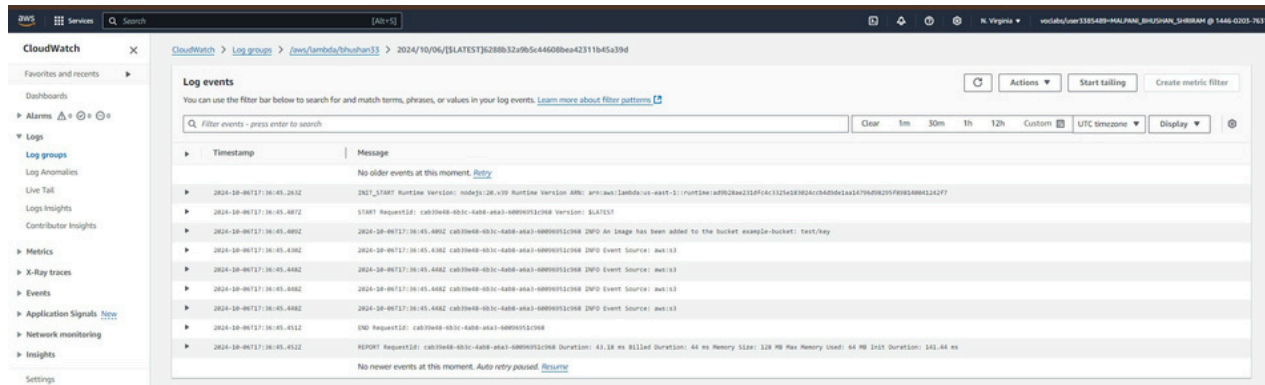


3) Here, under Log streams, select the log stream you want to check.



The screenshot shows the AWS CloudWatch console. The left sidebar contains navigation options like Dashboards, Alarms, Log groups, Live Tail, Logs Insights, Contributor Insights, Metrics, X-Ray traces, Events, Application Signals, Network monitoring, and Insights. The main content area is titled '/aws/lambda/bhushan33'. Under 'Log group details', it shows the log class 'info' and standard retention. The 'Log streams' tab is active, displaying a list of log streams. The first log stream is selected, and its details are shown on the right, including stored bytes, metric filters, subscription filters, contributor insights rules, and KMS key ID.

4) Here again, we can see that 'An image has been added to the bucket'.



The screenshot shows the AWS CloudWatch console. The left sidebar is the same as the previous screenshot. The main content area is titled '/aws/lambda/bhushan33 > 2024/10/06/\$LATEST'. The 'Log events' tab is active, displaying a list of log events. The first log event is selected, and its details are shown on the right, including the timestamp, message, and event source.

Conclusion:

In this experiment, we developed and deployed a Lambda function designed to respond to file uploads in an S3 bucket. The function was triggered automatically whenever a new object was added to the bucket, illustrating how AWS services can efficiently automate workflows. The Lambda function extracted and logged key details from the event, such as the bucket's name and the object's key. We tested this by uploading a sample file, and upon reviewing the logs in CloudWatch, we confirmed that the function executed successfully, capturing the upload event. This experiment demonstrated the powerful synergy between AWS Lambda and S3, enabling seamless, event-driven automation.