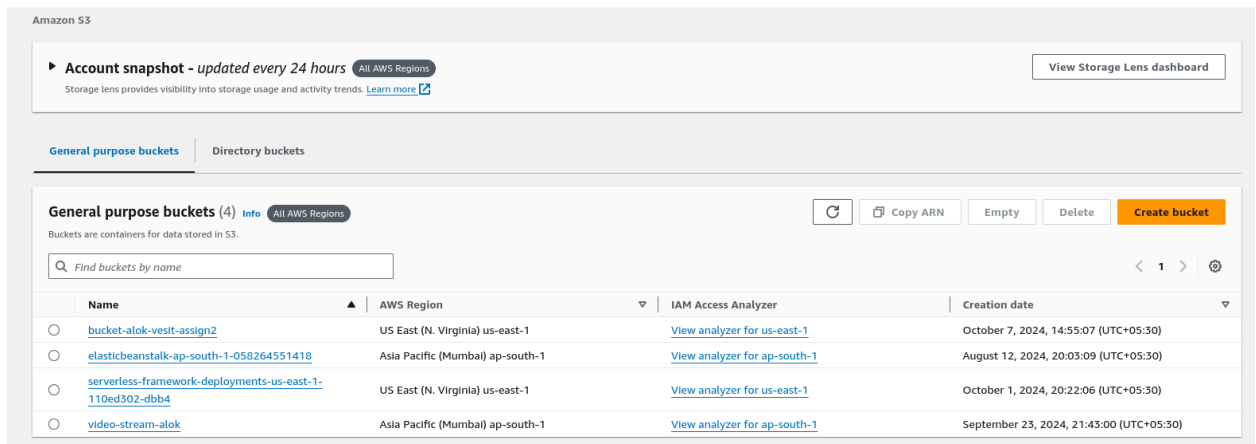


## EXPERIMENT NO.12

**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

### Steps To create the lambda function:

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



2. Now Give a name to the Bucket, select general purpose project and deselect the Block public access and keep other this to default.

## Create bucket [Info](#)

Buckets are containers for data stored in S3.

### General configuration

AWS Region

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

Bucket type [Info](#)

☒ General purpose

Recommended for most use cases and access patterns. General purpose buckets are the original S3 bucket type. They allow a mix of storage classes that redundantly store objects across multiple Availability Zones.

☐ Directory

Recommended for low-latency use cases. These buckets use only the S3 Express One Zone storage class, which provides faster processing of data within a single Availability Zone.

Bucket name [Info](#)

alok-exp-12-lambda

Bucket name must be unique within the global namespace and follow the bucket naming rules. [See rules for bucket naming](#) [↗](#)

#### Copy settings from existing bucket - *optional*

Only the bucket settings in the following configuration are copied.

[Choose bucket](#)

Format: s3://bucket/prefix

### Object Ownership [Info](#)

Control ownership of objects written to this bucket from other AWS accounts and the use of access control lists (ACLs). Object ownership determines who can specify access to objects.

☒ ACLs disabled (recommended)

All objects in this bucket are owned by this account. Access to this bucket and its objects is specified using only policies.

☐ ACLs enabled

Objects in this bucket can be owned by other AWS accounts. Access to this bucket and its objects can be specified using ACLs.

Object Ownership

Bucket owner enforced

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.

Bucket Versioning

Versioning is a means of keeping multiple variants of an object in the same bucket. You can use versioning to preserve, retrieve, and restore every version of every object stored in your Amazon S3 bucket. With versioning, you can easily recover from both unintended user actions and application failures. [Learn more](#)

Bucket Versioning

☒ Disable

☐ Enable

Tags - optional (0)

Successfully created bucket "alok-exp-12-lambda"

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

Amazon S3 > Buckets

Account snapshot - updated every 24 hours

Storage lens provides visibility into storage usage and activity trends. [Learn more](#)

[View Storage Lens dashboard](#)

General purpose buckets

Directory buckets

General purpose buckets (5)

Buckets are containers for data stored in S3.

Copy ARN

Empty

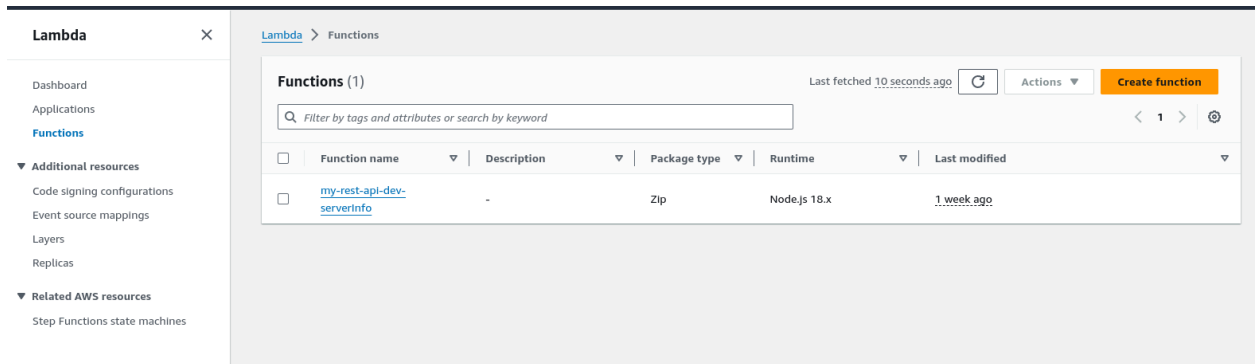
Delete

Create bucket

Name	AWS Region	IAM Access Analyzer	Creation date
<input type="radio"/> alok-exp-12-lambda	US East (N. Virginia) us-east-1	<a href="#">View analyzer for us-east-1</a>	October 10, 2024, 18:21:38 (UTC+05:30)
<input type="radio"/> bucket-alok-visit-assign2	US East (N. Virginia) us-east-1	<a href="#">View analyzer for us-east-1</a>	October 7, 2024, 14:55:07 (UTC+05:30)
<input type="radio"/> elasticbeanstalk-ap-south-1-058264551418	Asia Pacific (Mumbai) ap-south-1	<a href="#">View analyzer for ap-south-1</a>	August 12, 2024, 20:03:09 (UTC+05:30)
<input type="radio"/> serverless-framework-deployments-us-east-1-110ed302-dbb4	US East (N. Virginia) us-east-1	<a href="#">View analyzer for us-east-1</a>	October 1, 2024, 20:22:06 (UTC+05:30)
<input type="radio"/> video-stream-alok	Asia Pacific (Mumbai) ap-south-1	<a href="#">View analyzer for ap-south-1</a>	September 23, 2024, 21:43:00 (UTC+05:30)

Bucket created successfully

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



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 Execution role to Create a new role with basic Lambda permissions.

☒ **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 (\_).

**Runtime** [Info](#)  
Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.  

▼

**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](#).  

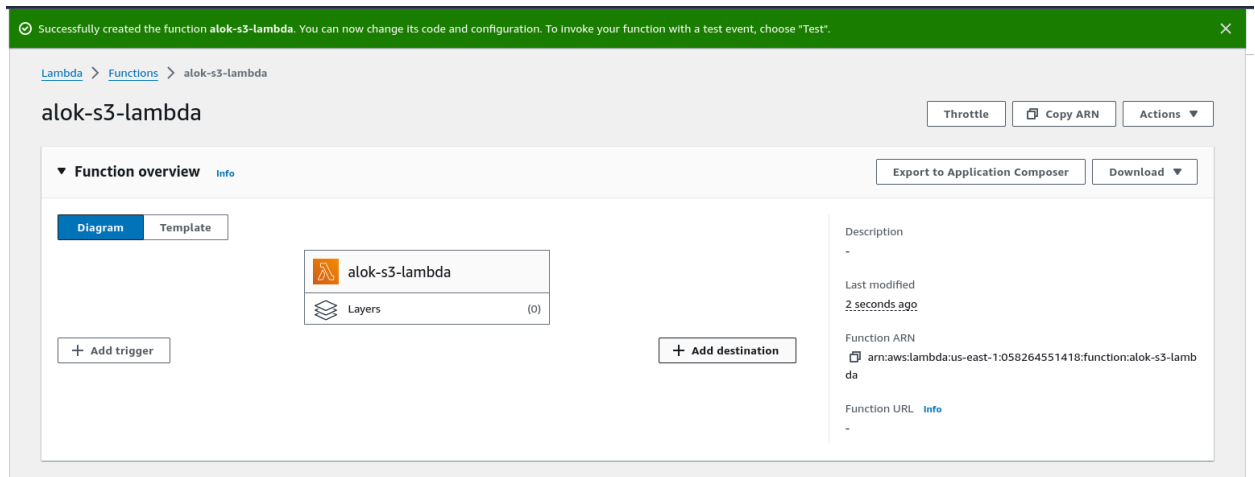
☒ Create a new role with basic Lambda permissions  
☐ Use an existing role  
☐ Create a new role from AWS policy templates

 Role creation might take a few minutes. Please do not delete the role or edit the trust or permissions policies in this role.

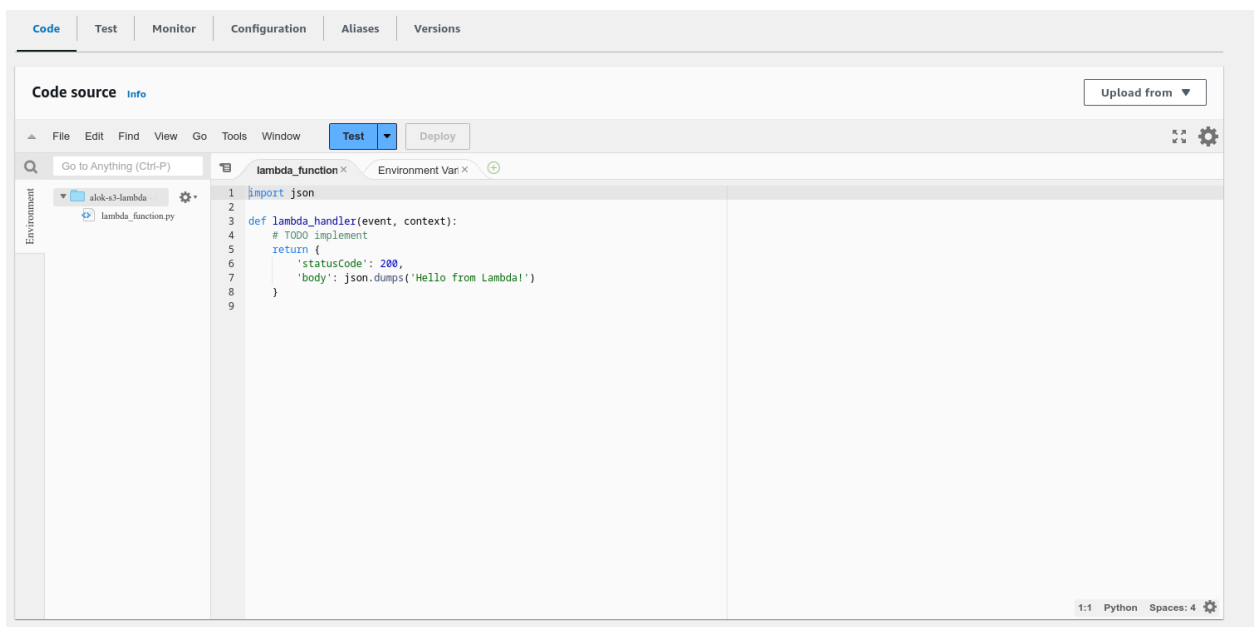
Name: Alok Yadav  
No. 59

Div: D15C

Roll



Lambda function created successfully



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

The screenshot shows the 'Configuration' tab of the AWS Lambda console. On the left is a sidebar with a list of configuration options: General configuration, Triggers, Permissions, Destinations, Function URL, Environment variables, Tags, VPC, RDS databases, Monitoring and operations tools, Concurrency and recursion detection, and Asynchronous invocation. The 'General configuration' option is selected. The main area displays the 'General configuration' details for the function 'alok-s3-lambda'. It includes an 'Edit' button in the top right. The configuration details are as follows:

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

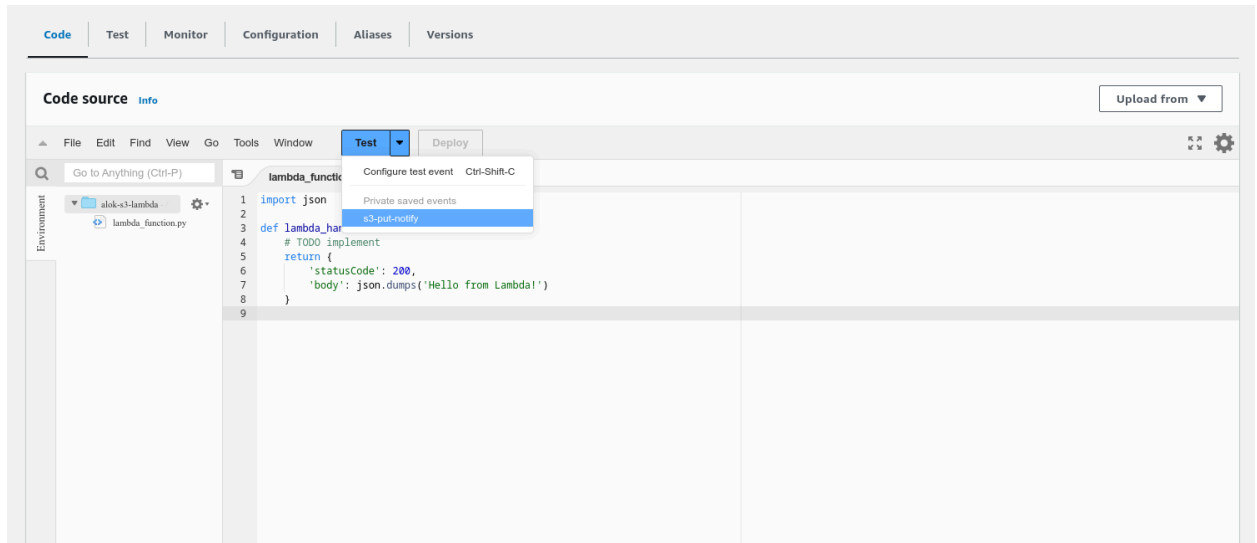
Here, you can enter a description and change Memory and Timeout. I've changed the Timeout period to 1 sec since that is sufficient for now.

- 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.

The screenshot shows the 'Test event' configuration page in the AWS Lambda console. At the top, a green banner indicates 'Successfully updated the function alok-s3-lambda'. The page has 'Save' and 'Test' buttons in the top right. Below the header, there is a note: 'To invoke your function without saving an event, configure the JSON event, then choose Test.' The 'Test event action' section has two options: 'Create new event' (selected) and 'Edit saved event'. The 'Event name' field contains 's3-put-notify' with a note: 'Maximum of 25 characters consisting of letters, numbers, dots, hyphens and underscores.' The 'Event sharing settings' section has two radio buttons: 'Private' (selected) and 'Shareable'. Below 'Private' is a note: 'This event is only available in the Lambda console and to the event creator. You can configure a total of 10. [Learn more](#)'. Below 'Shareable' is a note: 'This event is available to IAM users within the same account who have permissions to access and use shareable events. [Learn more](#)'. The 'Template - optional' dropdown menu is set to 's3-put'. The 'Event JSON' section has a 'Format JSON' button and a text area containing the following 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    }
13  ]
14 }
```

- Now In Code section select the created event from the dropdown .




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 image.



### 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**

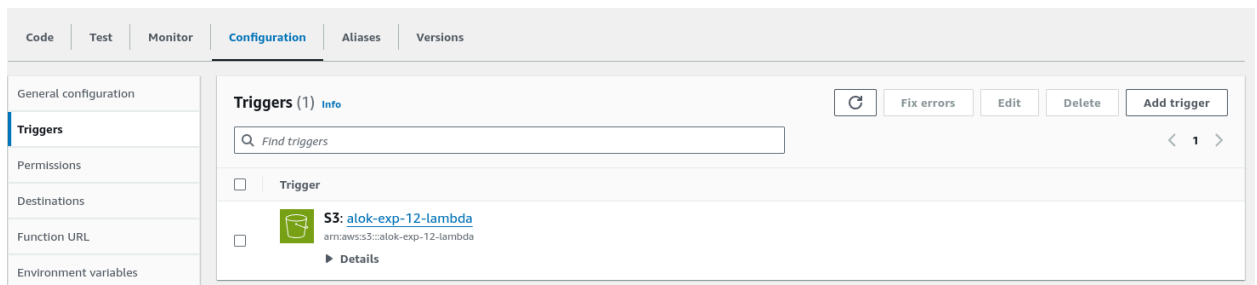
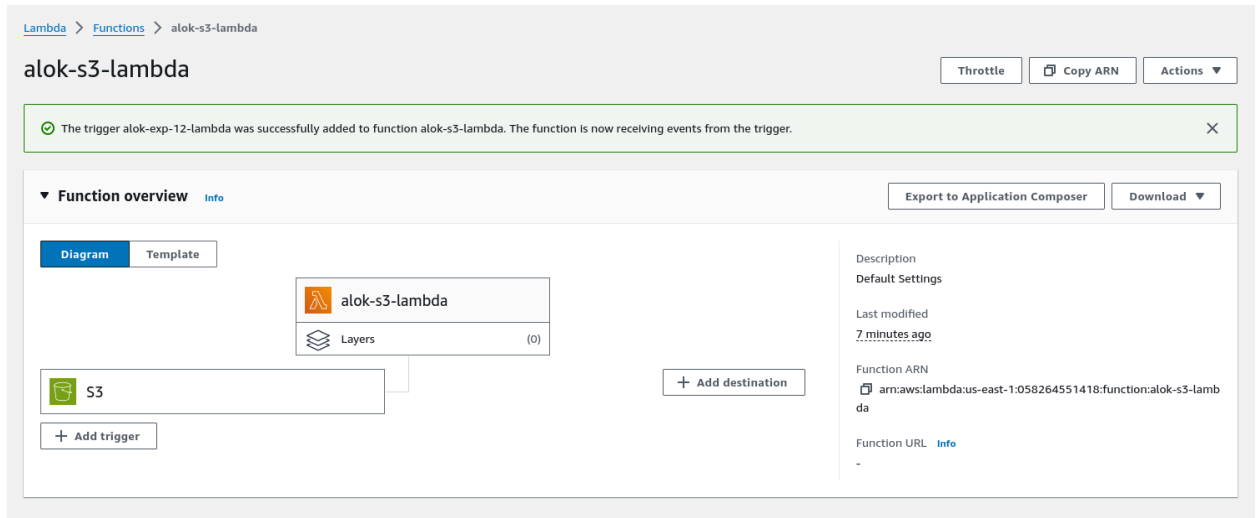
Enter a single optional suffix to limit the notifications to objects with keys that end with matching characters. Any [special characters](#) must be URL encoded.

**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.



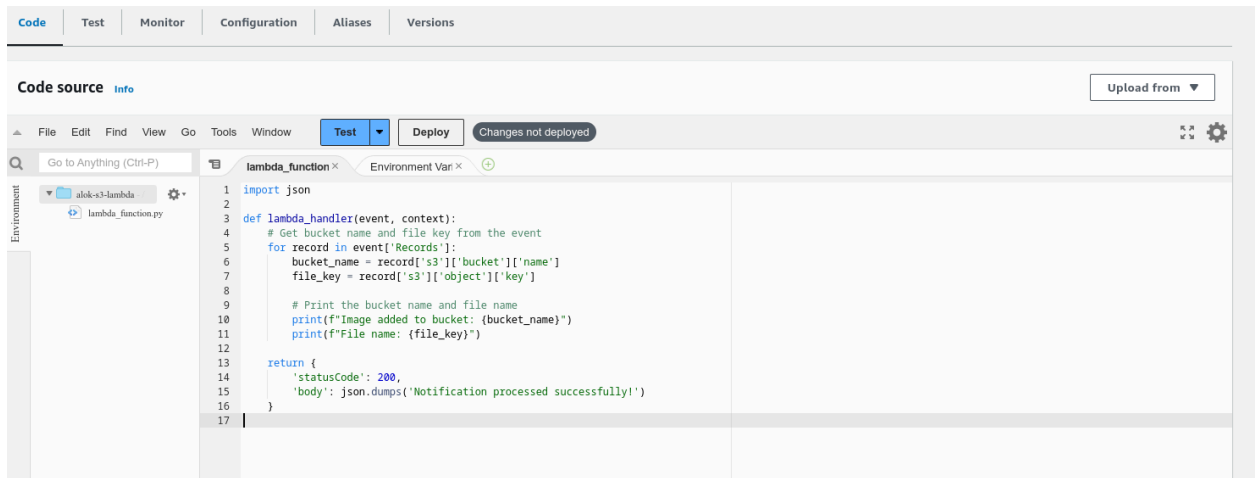
8. Now Write code that logs a message like “Image added to bucket” when triggered. Save the file and click on deploy.

```
import json
```

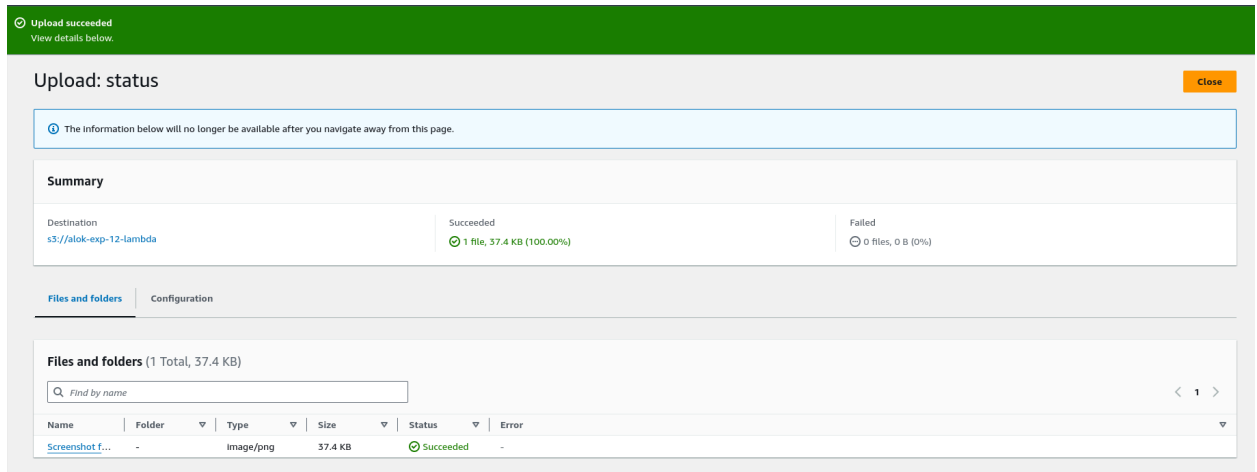
```
def lambda_handler(event, context):
    # Get bucket name and file key from the event
    for record in event['Records']:
        bucket_name = record['s3']['bucket']['name']
        file_key = record['s3']['object']['key']

    # Print the bucket name and file name
    print(f"Image added to bucket: {bucket_name}")
    print(f"File name: {file_key}")

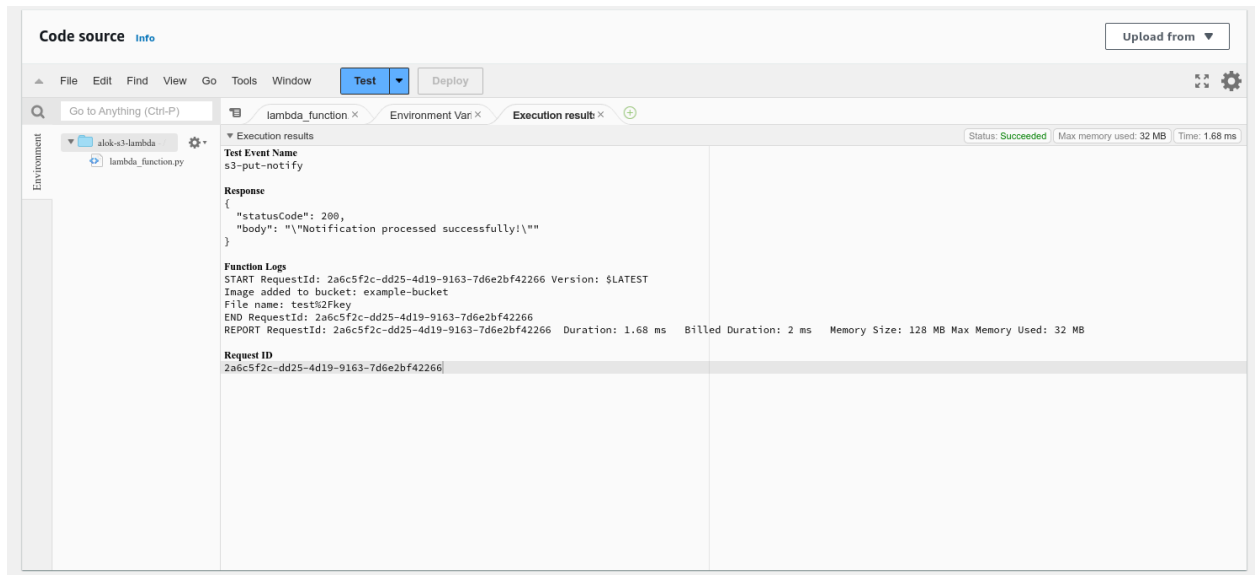
    return {
        'statusCode': 200,
        'body': json.dumps('Notification processed successfully!')
    }
```



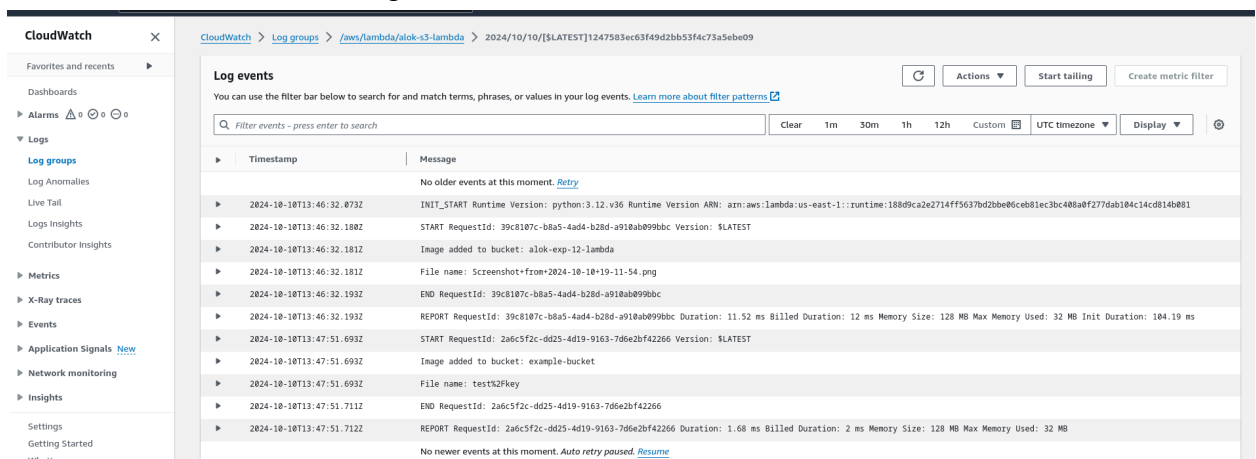
9. Now upload any image to the bucket.



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



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



As we can see our lambda function activity is recorded by Cloudwatch

## Conclusion:

In this experiment, we successfully implemented an AWS Lambda function that logs a message when an image is uploaded to an S3 bucket. A key aspect was selecting the **S3 Object Created (Put)** event template, as failing to do so would result in errors due to an incompatible event structure. The Lambda function was successfully triggered by S3 object uploads, demonstrating the functionality and efficiency of AWS Lambda's event-driven architecture. This experiment highlighted Lambda's ability to seamlessly respond to S3 events and emphasized

Name: Alok Yadav

Div: D15C

Roll

No. 59

the importance of correctly configuring event triggers to avoid common issues with event data.