

## Adv. DevOps

### Experiment No. 12

\* Aim - To create a Lambda function which will log "An image has been added" once you add an object to specific object in s3.

\* Theory - What are features of Lambda function?  
The following key features help you develop lambda applications that are scalable, secure & easily extensible -

#### 1. Concurrency and scaling controls.

Using concurrency setting to ensure that your production applications are highly available & highly responsive. Lambda means the infrastructure that runs code and scales automatically in response to incoming requests.

#### 2. Function URL's -

Lambda offers built-in HTTP(s) endpoint support through function URL's. With function URL's you can assign a dedicated HTTP endpoint to your Lambda function. When your function URL is configured you can use it to invoke your function through a web browser, curl, Postman or any HTTP client. You can also add a function URL to an existing function.

#### 3. Event source mappings -

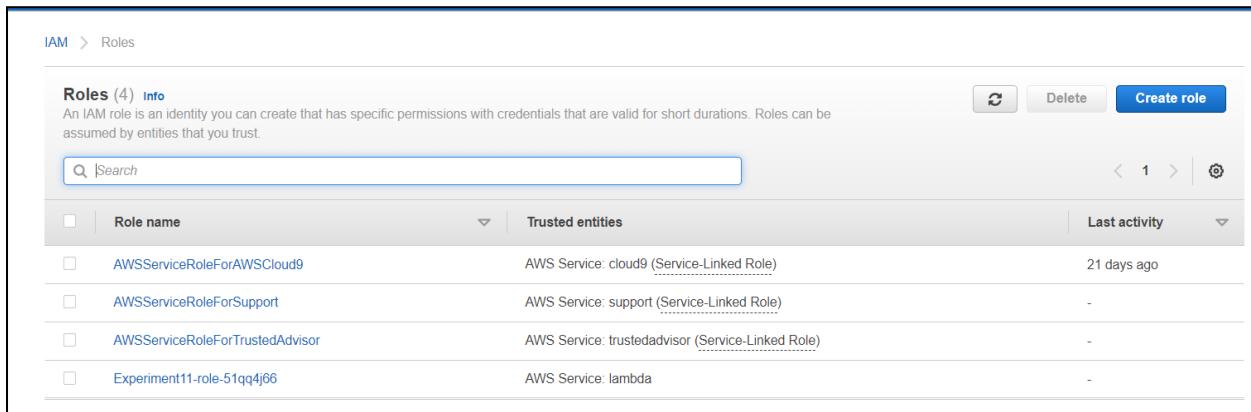
Event source mappings maintain a local queue

of unprocessed items and handle retries if the function returns an error. or you can configure an event source mapping to customize batching behaviour & error handling or to send a record of items that fail to a destination.

4. Testing and deployment tools -  
Lambda supports deploying code as it is or as container images. You can rich tools ecosystem for authorising, building and deploying your lambda functions using AWS.

## Implementation:

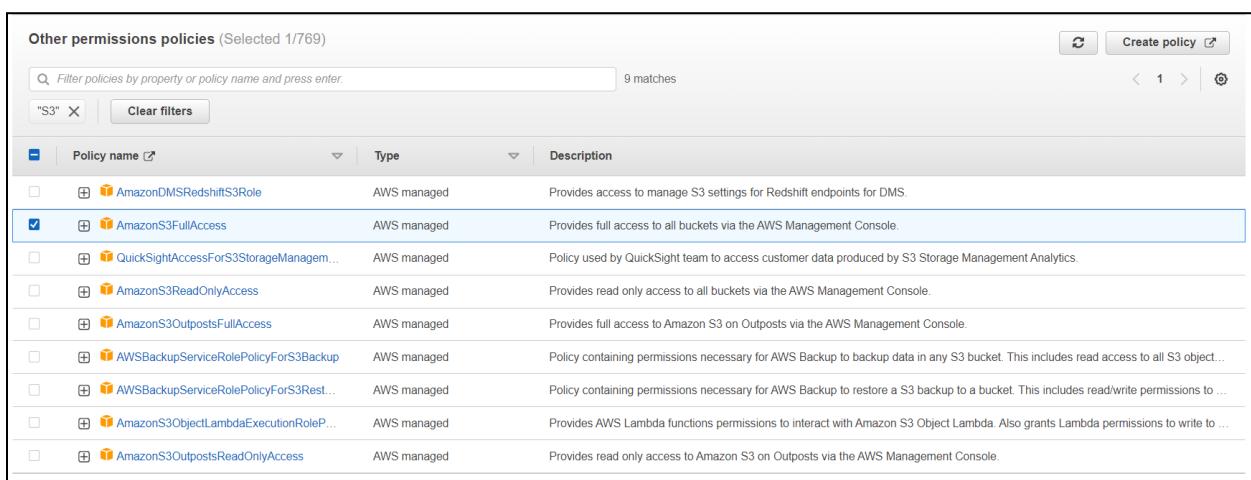
**Step 1:** Open up the IAM Console and under Roles, choose the Role we previously created for the Python Lambda Function (You can find your role name configuration of your Lambda function).



The screenshot shows the 'Roles' page in the IAM console. At the top, there is a search bar and navigation buttons. Below the header, a table lists four roles:

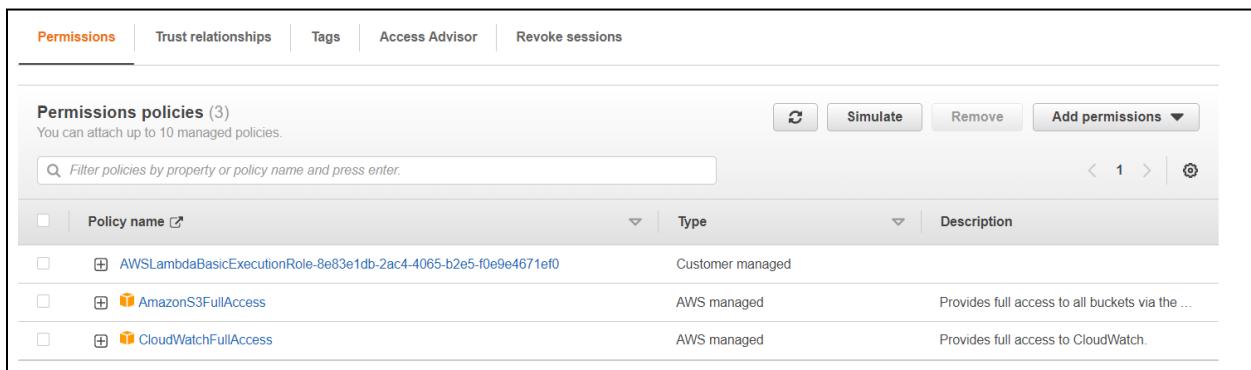
Role name	Trusted entities	Last activity
AWSServiceRoleForAWSCloud9	AWS Service: cloud9 (Service-Linked Role)	21 days ago
AWSServiceRoleForSupport	AWS Service: support (Service-Linked Role)	-
AWSServiceRoleForTrustedAdvisor	AWS Service: trustedadvisor (Service-Linked Role)	-
Experiment11-role-51qq4j66	AWS Service: lambda	-

**Step 2:** Under Attach Policies, add S3-ReadOnly and CloudWatchFull permissions to this role



The screenshot shows the 'Attach Policies' section of the IAM role configuration. It lists various managed policies that can be attached to the role. The 'AmazonS3FullAccess' policy is selected and highlighted.

Policy name	Type	Description
AmazonDMSRedshiftS3Role	AWS managed	Provides access to manage S3 settings for Redshift endpoints for DMS
<b>AmazonS3FullAccess</b>	AWS managed	Provides full access to all buckets via the AWS Management Console.
QuickSightAccessForS3StorageManagem...	AWS managed	Policy used by QuickSight team to access customer data produced by S3 Storage Management Analytics.
AmazonS3ReadOnlyAccess	AWS managed	Provides read only access to all buckets via the AWS Management Console.
AmazonS3OutpostsFullAccess	AWS managed	Provides full access to Amazon S3 on Outposts via the AWS Management Console.
AWSBackupServiceRolePolicyForS3Backup	AWS managed	Policy containing permissions necessary for AWS Backup to backup data in any S3 bucket. This includes read access to all S3 object...
AWSBackupServiceRolePolicyForS3Rest...	AWS managed	Policy containing permissions necessary for AWS Backup to restore a S3 backup to a bucket. This includes read/write permissions to ...
AmazonS3ObjectLambdaExecutionRoleP...	AWS managed	Provides AWS Lambda functions permissions to interact with Amazon S3 Object Lambda. Also grants Lambda permissions to write to ...
AmazonS3OutpostsReadOnlyAccess	AWS managed	Provides read only access to Amazon S3 on Outposts via the AWS Management Console.



The screenshot shows the 'Permissions' tab of the IAM role configuration. It displays the three policies that have been attached to the role: 'AWSLambdaBasicExecutionRole', 'AmazonS3FullAccess', and 'CloudWatchFullAccess'.

Policy name	Type	Description
AWSLambdaBasicExecutionRole-8e83e1db-2ac4-4065-b2e5-f0e9e4671ef0	Customer managed	
AmazonS3FullAccess	AWS managed	Provides full access to all buckets via the ...
CloudWatchFullAccess	AWS managed	Provides full access to CloudWatch.

### Step 3: Open up AWS Lambda and create a new Python function

**Basic information**

Function name  
Enter a name that describes the purpose of your function.

Use only letters, numbers, hyphens, or underscores with no spaces.

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](#)

Under Execution Role, choose the existing role, then select the one which was previously created and to which we just added permissions.

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.  
  
[View the Experiment11-role-51qq4j66 role on the IAM console.](#)

### Step 4: The function is up and running.

Successfully created the function Experiment12. You can now change its code and configuration. To invoke your function with a test event, choose "Test".

Lambda > Functions > Experiment12

Experiment12

[Function overview](#) [Info](#)

 Experiment12  
 Layers (0)

+ Add trigger [+ Add destination](#)

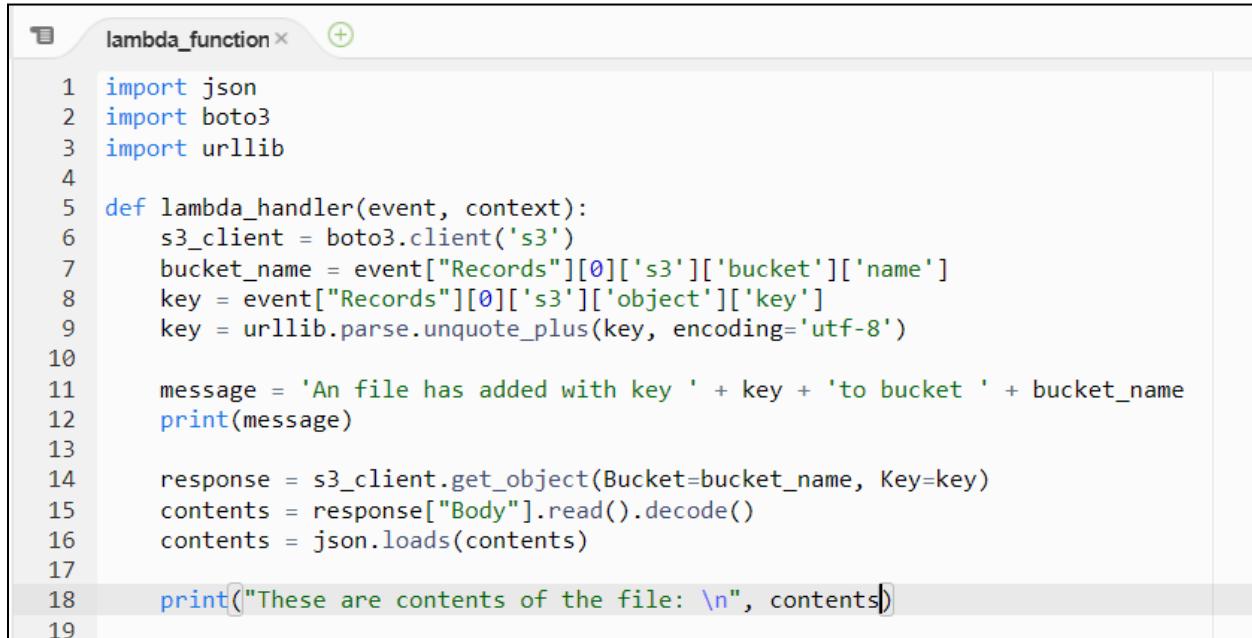
Description  
-

Last modified  
now

Function ARN  
[arn:aws:lambda:us-east-1:811764028225:function:Experiment12](#)

Function URL [Info](#)  
-

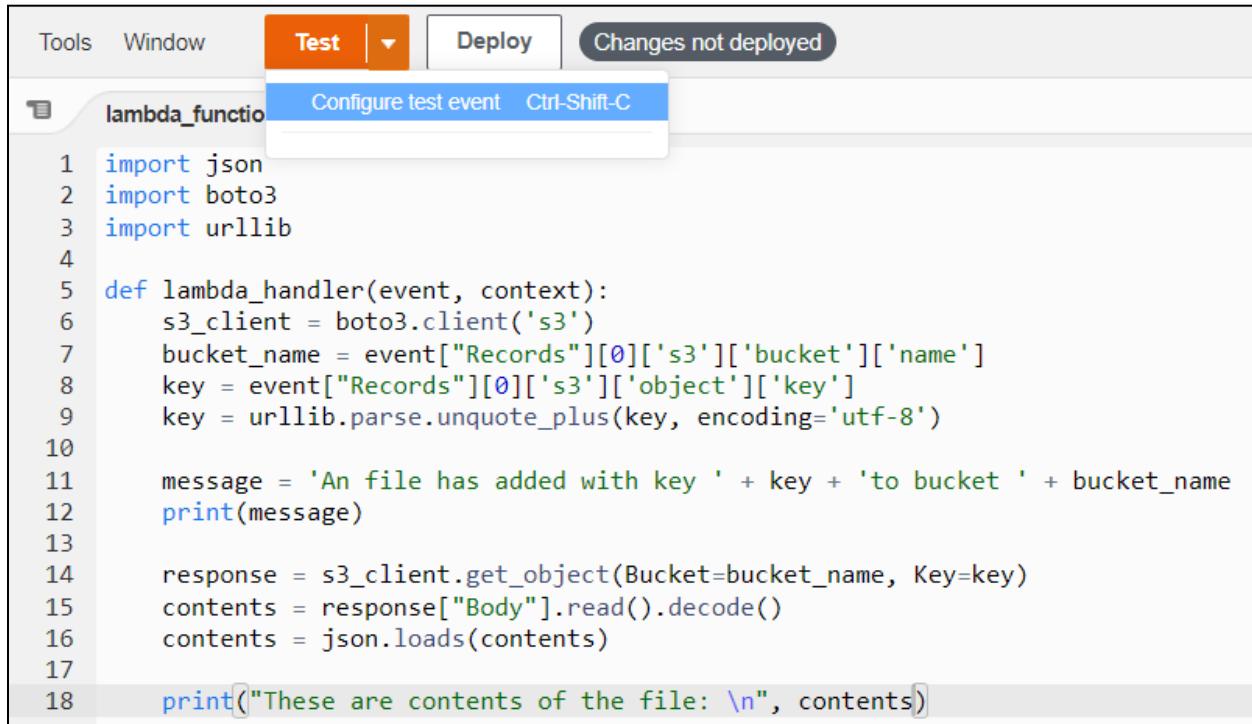
**Step 5:** Make the following changes to the function and click on the deploy button. This code basically logs a message and logs the contents of a JSON file which is uploaded to an S3 Bucket and then deploy the code.



The screenshot shows a code editor window titled "lambda\_function". The code is a Python function named "lambda\_handler" that takes "event" and "context" parameters. It uses the boto3 library to interact with an S3 bucket. The code prints a message to the console indicating a new file has been added to the bucket. It then reads the contents of the file from the S3 object and prints them. The code editor interface includes a toolbar with "Tools" and "Window" buttons, and tabs for "Test" and "Deploy".

```
1 import json
2 import boto3
3 import urllib
4
5 def lambda_handler(event, context):
6     s3_client = boto3.client('s3')
7     bucket_name = event["Records"][0]['s3']['bucket']['name']
8     key = event["Records"][0]['s3']['object']['key']
9     key = urllib.parse.unquote_plus(key, encoding='utf-8')
10
11     message = 'An file has added with key ' + key + 'to bucket ' + bucket_name
12     print(message)
13
14     response = s3_client.get_object(Bucket=bucket_name, Key=key)
15     contents = response["Body"].read().decode()
16     contents = json.loads(contents)
17
18     print("These are contents of the file: \n", contents)
19
```

**Step 6:** Click on Test and choose the 'S3 Put' Template.



The screenshot shows the same code editor window as before, but with the "Test" tab highlighted in orange. A dropdown menu is open under the "Test" tab, showing options like "Configure test event" and "Ctrl-Shift-C". The rest of the interface and code are identical to the previous screenshot.

```
1 import json
2 import boto3
3 import urllib
4
5 def lambda_handler(event, context):
6     s3_client = boto3.client('s3')
7     bucket_name = event["Records"][0]['s3']['bucket']['name']
8     key = event["Records"][0]['s3']['object']['key']
9     key = urllib.parse.unquote_plus(key, encoding='utf-8')
10
11     message = 'An file has added with key ' + key + 'to bucket ' + bucket_name
12     print(message)
13
14     response = s3_client.get_object(Bucket=bucket_name, Key=key)
15     contents = response["Body"].read().decode()
16     contents = json.loads(contents)
17
18     print("These are contents of the file: \n", contents)
19
```

## 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

Test

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

S3

AWS

Rekognition S3 Request

S3 Delete

S3 Put

s3-put

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"
```

**Step 7:** Open up the S3 Console and create a new bucket.

The screenshot shows the AWS S3 Management Console. On the left, there's a sidebar with various navigation options: Buckets, Access Points, Object Lambda Access Points, Multi-Region Access Points, Batch Operations, Access analyzer for S3, Block Public Access settings for this account, Storage Lens (Dashboards, AWS Organizations settings), Feature spotlight, and AWS Marketplace for S3. The main content area has a banner at the top with a message about improving the console and a 'Provide feedback' link. Below that, another banner encourages replicating data to save costs, with a 'Get started' button. The central part of the screen is titled 'Buckets' and shows an 'Account snapshot' with a note about storage usage and activity trends. A search bar and a table for managing buckets are present. The table has columns for Name, AWS Region, Access, and Creation date. It displays a message 'No buckets' and a 'Create bucket' button. At the bottom of the page, there's a feedback section, a search bar, and a footer with copyright information and links to Privacy, Terms, and Cookie preferences.

**Step 8:** With all general settings, create the bucket in the same region as the function

This screenshot shows the 'General configuration' step of a Lambda function creation wizard. It includes fields for 'Bucket name' (containing 'advdevopsexp12'), 'AWS Region' (set to 'US East (N. Virginia) us-east-1'), and a 'Choose bucket' button for optional copy settings from existing buckets. The background shows the rest of the Lambda function configuration interface.

**Step 9:** Click on the created bucket and under properties, look for events.

Buckets (1)		Info	Copy ARN	Empty	Delete	Create bucket
Buckets are containers for data stored in S3. <a href="#">Learn more</a>						
Name	AWS Region	Access	Creation date			
advdevopsexp12	US East (N. Virginia) us-east-1	Bucket and objects not public	October 4, 2022, 12:07:53 (UTC+05:30)			

Event notifications (0)				
<a href="#">Edit</a> <a href="#">Delete</a> <a href="#">Create event notification</a>				
Name	Event types	Filters	Destination type	Destination
No event notifications				
Choose <a href="#">Create event notification</a> to be notified when a specific event occurs.				
<a href="#">Create event notification</a>				
<b>Amazon EventBridge</b>				
For additional capabilities, use Amazon EventBridge to build event-driven applications at scale using S3 event notifications. <a href="#">Learn more</a> or see <a href="#">EventBridge pricing</a>				
<a href="#">Edit</a>				

Click on Create Event Notification.

**Step 10:** Mention an event name and check Put under event types.

General configuration	
Event name	<input type="text" value="S3PutRequest"/>
Event name can contain up to 255 characters.	
Prefix - <i>optional</i>	<input type="text" value="images/"/>
Limit the notifications to objects with key starting with specified characters.	
Suffix - <i>optional</i>	<input type="text" value=".jpg"/>
Limit the notifications to objects with key ending with specified characters.	
Event types	
Specify at least one event for which you want to receive notifications. For each group, you can choose an event type for all events, or you can choose one or more individual events.	
Object creation	
<input type="checkbox"/> All object create events s3:ObjectCreated:*	<input checked="" type="checkbox"/> Put s3:ObjectCreated:Put

Choose Lambda function as destination and choose your lambda function and save the changes.

### Destination

**Before Amazon S3 can publish messages to a destination, you must grant the Amazon S3 principal the necessary permissions to call the relevant API to publish messages to an SNS topic, an SQS queue, or a Lambda function. [Learn more](#)**

**Destination**  
Choose a destination to publish the event. [Learn more](#)

**Lambda function**  
Run a Lambda function script based on S3 events.

**SNS topic**  
Fanout messages to systems for parallel processing or directly to people.

**SQS queue**  
Send notifications to an SQS queue to be read by a server.

**Specify Lambda function**

**Choose from your Lambda functions**

**Enter Lambda function ARN**

**Lambda function**

Experiment12

**Cancel** **Save changes**

**Step 11:** Refresh the Lambda function console and you should be able to see an S3 Trigger in the overview.

Lambda > Functions > Experiment12

Experiment12

**Function overview** [Info](#)

 Experiment12  Layers (0)	<b>Description</b> -
 S3 <a href="#">+ Add trigger</a>	<b>Last modified</b> in 23 seconds
	<b>Function ARN</b> <a href="#">arn:aws:lambda:us-east-1:811764028225:function:Experiment12</a>
	<b>Function URL</b> <a href="#">Info</a> -

**Step 12:** Now, create a dummy JSON file locally

```
{} dummy.json X  
C: > Users > Ishika Devare > OneDrive > Desktop > {} dummy.json  
1 {  
2   "firstName": "Ishika",  
3   "lastName": "Devare",  
4   "gender": "Female",  
5   "age": 20  
6 }
```

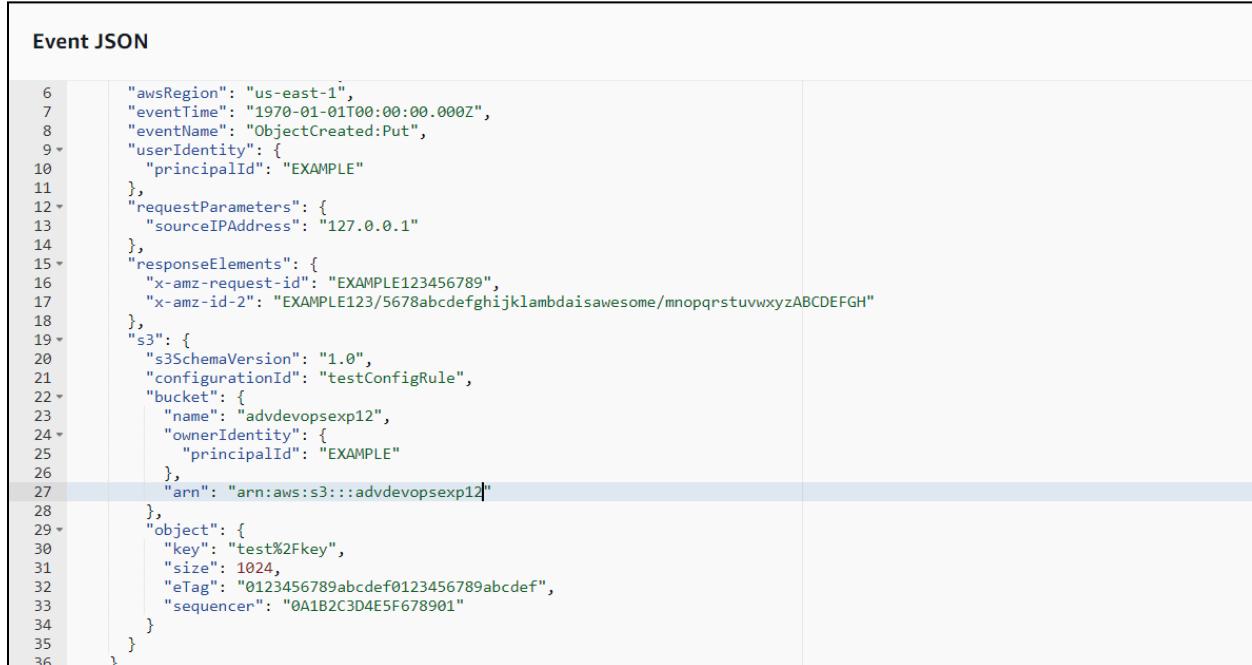
**Step 13:** Go back to your S3 Bucket and click on Add Files to upload a new file.

The screenshot shows the AWS S3 'Upload' interface. The top navigation bar includes 'Amazon S3 > Buckets > advdevopsexp12 > Upload'. Below this, the title 'Upload' has an 'Info' link. A large central area is labeled 'Drag and drop files and folders you want to upload here, or choose Add files, or Add folders.' Below this is a table titled 'Files and folders (1 Total, 110.0 B)'. The table lists one item: 'dummy.json' (application/json, 110.0 B). There are buttons for 'Remove', 'Add files', and 'Add folder'. At the bottom right of the table are navigation arrows and a page number '1'.

**Step 14:** Select the dummy data file from your computer and click Upload.

The screenshot shows the AWS S3 'Upload: status' interface. The top bar indicates 'Upload succeeded' with a green icon. Below this is a summary table with three columns: 'Destination' (s3://advdevopsexp12), 'Succeeded' (1 file, 110.0 B (100.00%)), and 'Failed' (0 files, 0 B (0%)). A note at the top says 'The information below will no longer be available after you navigate away from this page.'

**Step 15:** After this make the necessary changes in the Test configuration file which we created previously by replacing the Bucket Name and the ARN of Bucket.

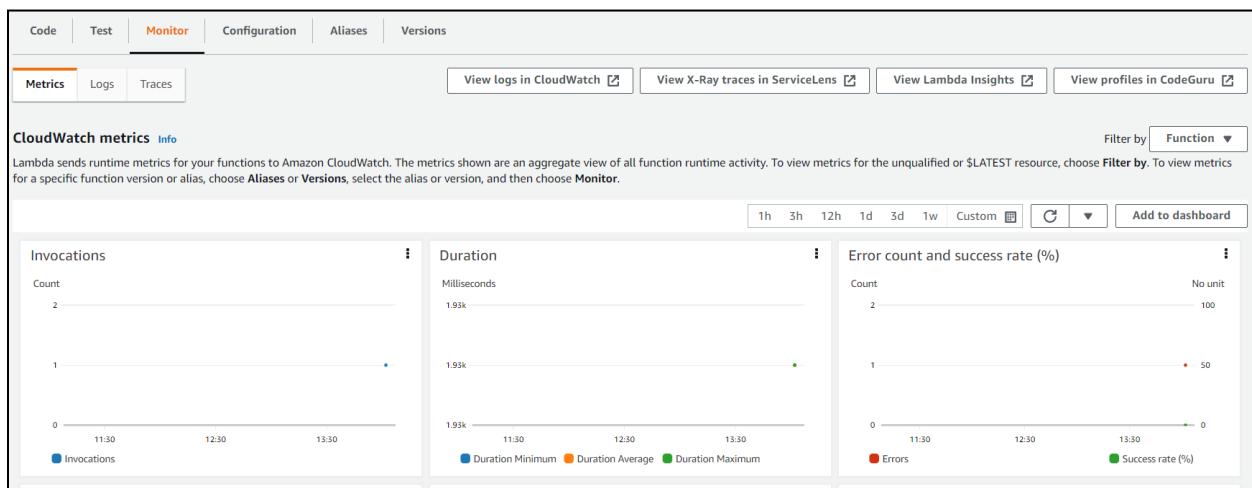


```

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": "EXAMPLE123/5678abcdefghijklmnaaaaaaaaaaaaaaaaaaaaaaa"
18 },
19 "s3": {
20     "s3SchemaVersion": "1.0",
21     "configurationId": "testConfigRule",
22     "bucket": {
23         "name": "advdevopsexp12",
24         "ownerIdentity": {
25             "principalId": "EXAMPLE"
26         },
27         "arn": "arn:aws:s3:::advdevopsexp12"
28     },
29     "object": {
30         "key": "test%2Fkey",
31         "size": 1024,
32         "eTag": "0123456789abcdef0123456789abcdef",
33         "sequencer": "0A1B2C3D4E5F678901"
34     }
35 }
36

```

**Step 16:** Go back to your Lambda function , Refresh it and check the Monitor tab.



Under Log streams, click on View logs in Cloudwatch to check the Function logs.

The screenshot shows the AWS CloudWatch Log Stream interface. At the top, there are tabs for 'Log streams' (which is selected), 'Metric filters', 'Subscription filters', 'Contributor Insights', and 'Tags'. Below the tabs, a search bar contains the placeholder 'Filter log streams or try prefix search'. To its right is a checkbox for 'Exact match'. Further right are buttons for 'Delete', 'Create log stream', and 'Search all log streams'. A page navigation bar shows '1' of 1 results. The main area displays a single log stream entry with a checkbox next to it, labeled 'Log stream'. Below the entry is the timestamp '2022-10-04/[\$LATEST]e591265d23fb413a86c523867d7aaa4c' and the last event time '2022-10-04 19:32:57 (UTC+05:30)'.

**Step 17:** Click on this log Stream that was created to view what was logged by your function.

▶	Timestamp	Message
		No older events at this moment. <a href="#">Retry</a>
▶	2022-10-04T19:32:57.069+05:30	START RequestId: 12bd5ecc-073c-47aa-9506-ddd75ef3013d Version: \$LATEST
▶	2022-10-04T19:32:58.654+05:30	An file has added with key test/keyto bucket Experiment12
▶	2022-10-04T19:32:58.993+05:30	[ERROR] NoSuchBucket: An error occurred (NoSuchBucket) when calling the GetObject operation: The specified bucket does not exist Traceback (most recent...
▶	2022-10-04T19:32:59.012+05:30	END RequestId: 12bd5ecc-073c-47aa-9506-ddd75ef3013d
▶	2022-10-04T19:32:59.012+05:30	REPORT RequestId: 12bd5ecc-073c-47aa-9506-ddd75ef3013d Duration: 1925.35 ms Billed Duration: 1926 ms Memory Size: 128 MB Max Memory Used: 72 MB Init Du...

As you can see, our function logged that a file was uploaded with its file name and the bucket to which it was uploaded. It also mentions the contents inside the file as our function was defined to.

Hence, we have successfully created a Python function inside AWS Lambda which logs every time an object is uploaded to an S3 Bucket.

\* Conclusion - Here, we created a python function, inside AWS function which logs everytime when an object is uploaded to an AWS S3 bucket.