

Advanced DevOps Lab

Experiment 12

Roll No.	24
Name	Iyer Sreekesh Subramanian
Class	D15-A
Subject	Advanced DevOps Lab

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

Theory:

AWS Lambda

AWS Lambda is a serverless computing service provided by Amazon Web Services (AWS). Users of AWS Lambda create functions, self-contained applications written in one of the supported languages and runtimes, and upload them to AWS Lambda, which executes those functions in an efficient and flexible manner.

The Lambda functions can perform any kind of computing task, from serving web pages and processing streams of data to calling APIs and integrating with other AWS services.

The concept of “serverless” computing refers to not needing to maintain your own servers to run these functions. AWS Lambda is a fully managed service that takes care of all the infrastructure for you. And so “serverless” doesn’t mean that there are no servers involved: it just means that the servers, the operating systems, the network layer and the rest of the infrastructure have already been taken care of so that you can focus on writing application code.

Features of AWS Lambda

- AWS Lambda easily scales the infrastructure without any additional configuration. It reduces the operational work involved.
- It offers multiple options like AWS S3, CloudWatch, DynamoDB, API Gateway, Kinesis, CodeCommit, and many more to trigger an event.
- You don’t need to invest upfront. You pay only for the memory used by the lambda function and minimal cost on the number of requests hence cost-efficient.
- AWS Lambda is secure. It uses AWS IAM to define all the roles and security policies.
- It offers fault tolerance for both services running the code and the function. You do not have to worry about the application down.

Steps to create a Lambda function that reacts to uploads in an S3 Bucket

1. Open up the IAM Console and under Roles, choose the Role we previously created for the Python Lambda Function.

The screenshot shows the AWS IAM console 'Roles (12)' page. The left sidebar contains navigation links for Identity and Access Management (IAM), Access management, Access reports, and Service control policies (SCPs). The main content area displays a table of roles. The role 'myPythonLambdaFunction-role-3k7e4ws3' is highlighted with a red box. Below the table, there are tabs for Permissions, Trust relationships, Tags, Access Advisor, and Revoke sessions. The 'Permissions' tab is active, showing 'Permissions policies (3 policies applied)' and an 'Attach policies' button.

Role name	Trusted entities	Last acti...
[redacted]	AWS Service: ec2	9 days ago
[redacted]	AWS Service: elasticbeanstalk	9 days ago
[redacted]	AWS Service: codepipeline	9 days ago
AWSServiceRoleForAmazonElasticFileSystem	AWS Service: elasticfilesystem (Service-Linked Role)	72 days ago
AWSServiceRoleForAutoScaling	AWS Service: autoscaling (Service-Linked Role)	9 days ago
AWSServiceRoleForAWSCloud9	AWS Service: cloud9 (Service-Linked Role)	42 days ago
AWSServiceRoleForBackup	AWS Service: backup (Service-Linked Role)	14 hours ago
AWSServiceRoleForElasticLoadBalancing	AWS Service: elasticloadbalancing (Service-Linked Role)	9 days ago
AWSServiceRoleForSupport	AWS Service: support (Service-Linked Role)	-
AWSServiceRoleForTrustedAdvisor	AWS Service: trustedadvisor (Service-Linked Role)	-
[redacted]	AWS Service: ec2	73 days ago
myPythonLambdaFunction-role-3k7e4ws3	AWS Service: lambda	1 hour ago

Under Attach Policies, add S3-ReadOnly and CloudWatchFull permissions to this role.

The screenshot shows the 'Create policy' dialog in the AWS IAM console. The 'Filter policies' search bar is set to 'S3Read'. The 'AmazonS3ReadOnlyAccess' policy is selected. Below this, another 'Create policy' dialog is shown with the 'Filter policies' search bar set to 'CloudwatchFull'. The 'CloudWatchFullAccess' policy is selected.

Create policy

Filter policies

Policy name

☒ AmazonS3ReadOnlyAccess

Create policy

Filter policies

Policy name

☒ CloudWatchFullAccess

2. Open up AWS Lambda and create a new Python function

The screenshot shows the 'Create function' page in the AWS Lambda console. The 'Author from scratch' option is selected. The 'Function name' is 'myS3EventLogger'. The 'Runtime' is set to 'Python 3.9'. The 'Architecture' is 'x86_64'. Under 'Permissions', the 'Change default execution role' section shows 'Use an existing role' selected, with the role 'service-role/myPythonLambdaFunction-role-3k7e4ws3' chosen from the dropdown. The 'Create function' button is visible at the bottom right.

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

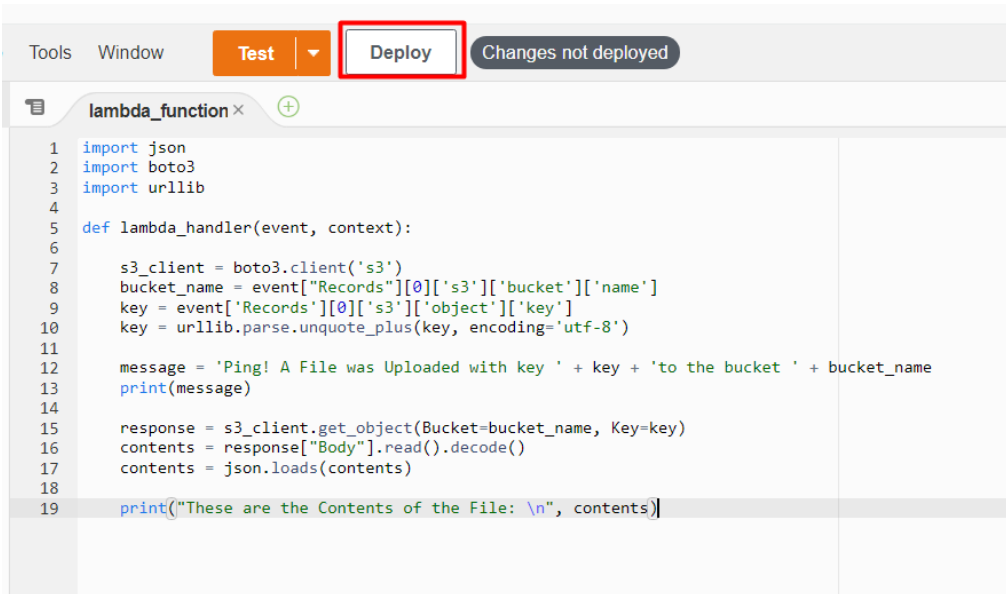
This screenshot shows the 'Execution role' section. The 'Use an existing role' radio button is selected. The 'Existing role' dropdown menu shows 'service-role/myPythonLambdaFunction-role-3k7e4ws3'. A link to view the role on the IAM console is provided below the dropdown.

3. The function is up and running.

The screenshot shows the 'Function overview' page for 'myS3EventLogger'. A green banner at the top states 'Successfully created the function myS3EventLogger. You can now change its code and configuration. To invoke your function with a test event, choose "Test".' The function overview includes a card for 'myS3EventLogger' with a 'Layers' section showing '(0)'. There are buttons for '+ Add trigger' and '+ Add destination'. On the right, the 'Description' is '-', 'Last modified' is '4 seconds ago', and the 'Function ARN' is 'arn:aws:lambda:ap-south-1:050055370753:fu entLogger'.

4. Make the following changes to the function and click on the deploy button.

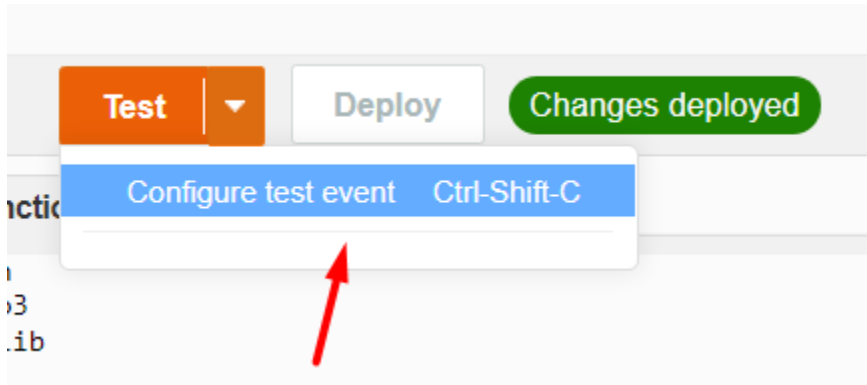
This code basically logs a message and logs the contents of a JSON file which is uploaded to an S3 Bucket.



The screenshot shows the AWS Lambda console interface. At the top, there are tabs for 'Tools' and 'Window'. Below these are two buttons: 'Test' (orange) and 'Deploy' (white with a red border). To the right of the 'Deploy' button is a status indicator that says 'Changes not deployed'. Below the buttons is a code editor window titled 'lambda_function'. The code in the editor is as follows:

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

5. Click on Test and choose the 'S3 Put' Template.



Configure test event

A function can have up to 10 test events. The events are persisted so you can switch to another computer or web browser and test your function with the same events.

☒ Create new test event
☐ Edit saved test events

Event template

hello-world

AWS

Recognition S3 Request

S3 Delete

S3 Put

Event name

MyEventName

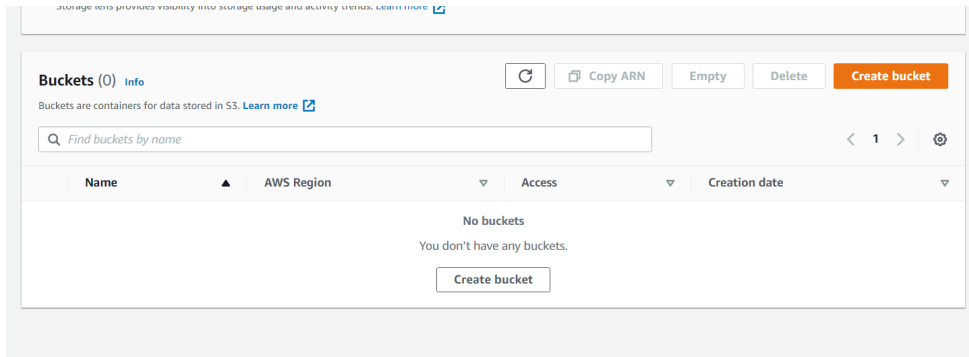
```

1 {
2   "Records": [
3     {
4       "eventVersion": "2.0",
5       "eventSource": "aws:s3",
6       "awsRegion": "ap-south-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/mnopqrstuvwxyzABCDEFGH"
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",
31          "size": 1024,

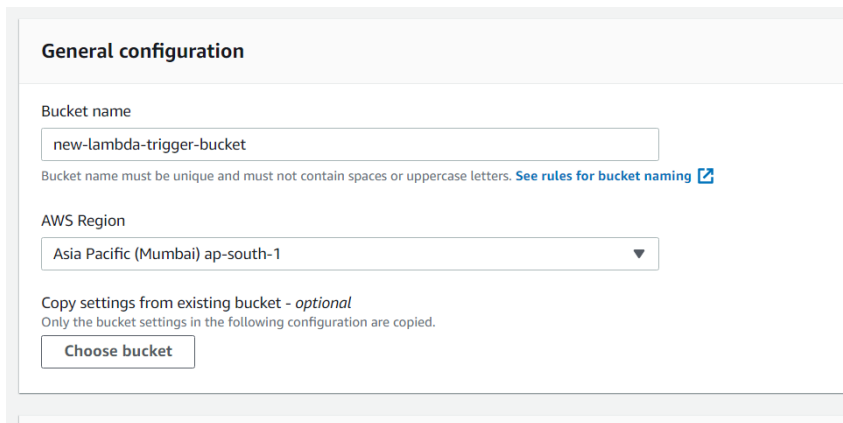
```

Here, inside the region, you may want to change the region to the AZ in which you've created your function and bucket. This doesn't seem mandatory but you might as well do it.

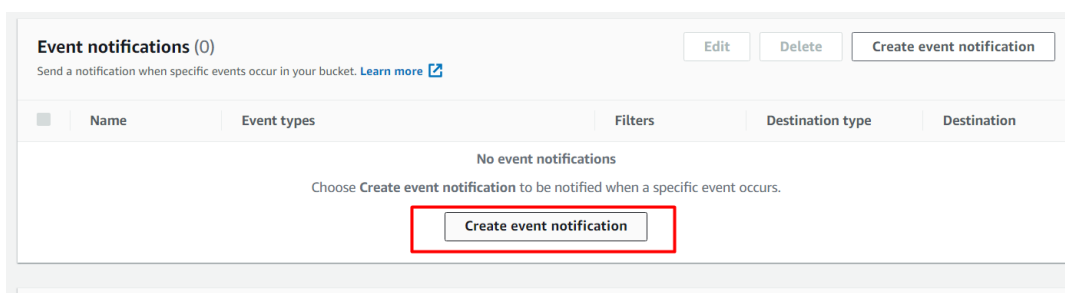
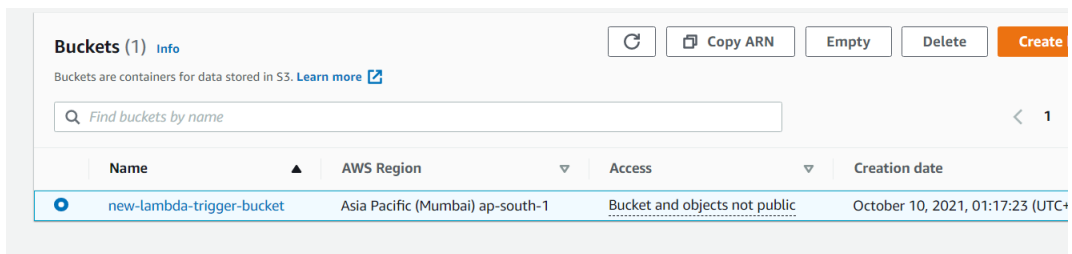
6. Open up the S3 Console and create a new bucket



7. With all general settings, create the bucket in the same region as the function.



8. Click on the created bucket and under properties, look for events.



Click on Create Event Notification.

9. Mention an event name and check Put under event types.

General configuration

Event name
s3PutRequest
Event name can contain up to 255 characters.

Prefix - optional
Limit the notifications to objects with key starting with specified characters.
images/

Suffix - optional
Limit the notifications to objects with key ending with specified characters.
.jpg

Event types
Specify at least one type of event for which you want to receive notifications. [Learn more](#)

- ☐ All object create events
s3:ObjectCreated:*
 - ☒ Put
s3:ObjectCreated:Put
 - ☐ Post
s3:ObjectCreated:Post
 - ☐ Copy
s3:ObjectCreated:Copy
 - ☐ Multipart upload completed
s3:ObjectCreated:CompleteMultipartUpload

You can optionally choose .json under the suffix since the code only accepts JSON.

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**
Send notifications to email, SMS, or an HTTP endpoint.
- ☐ **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**

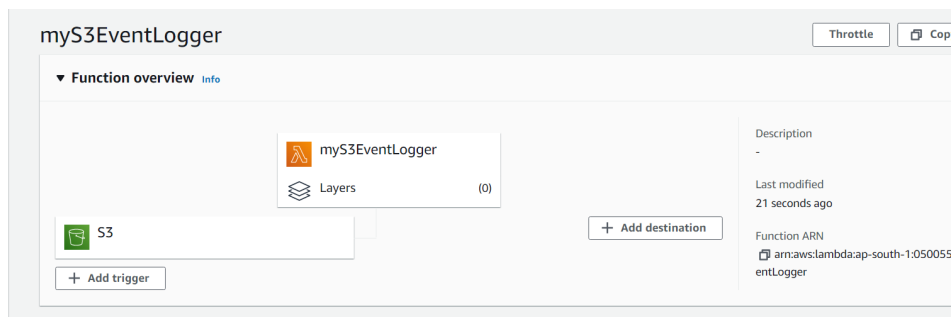
Lambda function

myS3EventLogger

Cancel Save changes

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

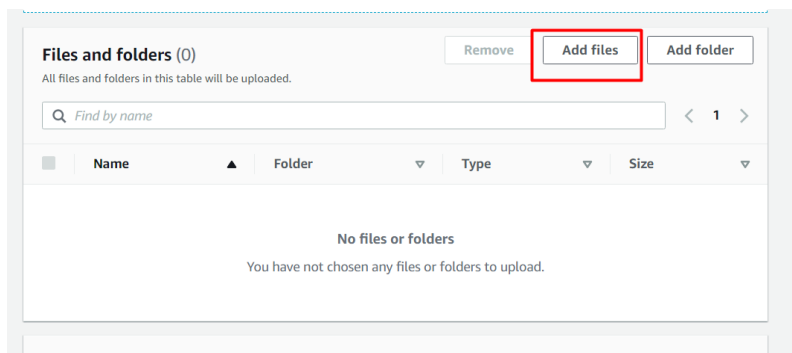
10. Refresh the Lambda function console and you should be able to see an S3 Trigger in the overview.



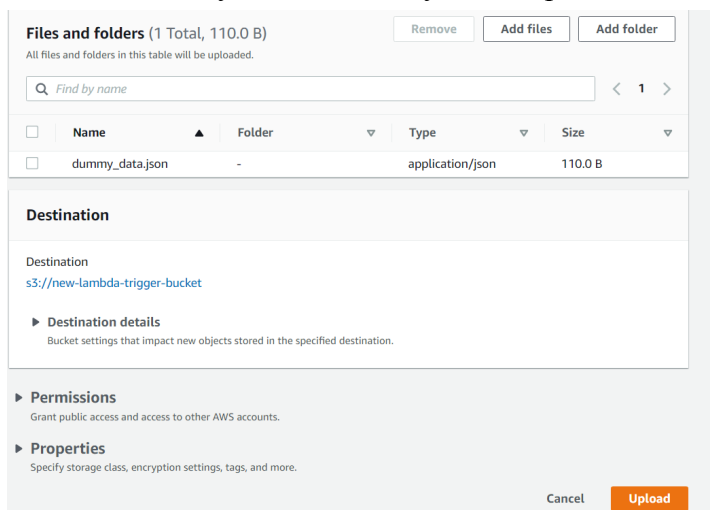
11. Now, create a dummy JSON file locally.

```
dummy_data.json
1  {
2    "id": 49,
3    "name": "Steve Smith",
4    "nationality": "AUS",
5    "runs": 7540,
6    "hs": 239
7  }
8
```

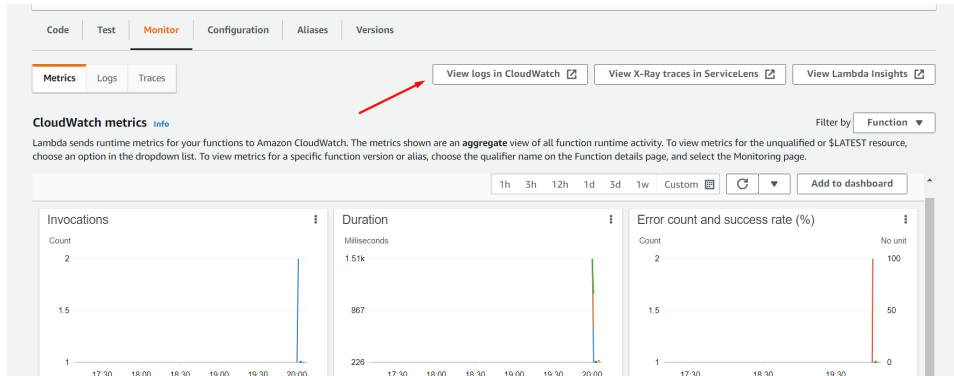
12. Go back to your S3 Bucket and click on Add Files to upload a new file.



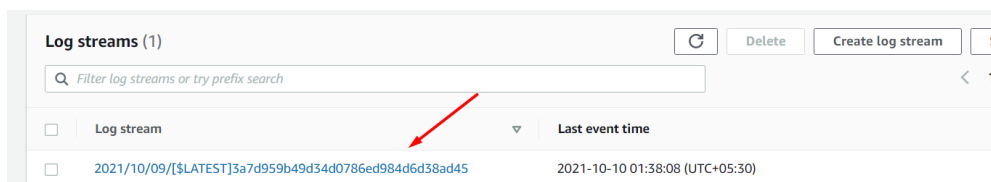
13. Select the dummy data file from your computer and click Upload.



14. Go back to your Lambda function and check the Monitor tab.



Under Metrics, click on View logs in Cloudwatch to check the Function logs



15. Click on this log Stream that was created to view what was logged by your function.

▶	2021-10-10T03:54:24.287+05:30	START RequestId: 5c8f5a94-d6a1-467c-bb5d-17b9b08683f1 Version: \$LATEST
▶	2021-10-10T03:54:25.518+05:30	Ping! A File was Uploaded with key dummy_data.json to the bucket sreeesh-bucket-s3-trigger
▶	2021-10-10T03:54:25.789+05:30	These are the Contents of the File:
▶	2021-10-10T03:54:25.789+05:30	{'id': 49, 'name': 'Steve Smith', 'nationality': 'AUS', 'runs': 7540, 'hs': 239}
▶	2021-10-10T03:54:25.816+05:30	END RequestId: 5c8f5a94-d6a1-467c-bb5d-17b9b08683f1
▶	2021-10-10T03:54:25.816+05:30	REPORT RequestId: 5c8f5a94-d6a1-467c-bb5d-17b9b08683f1 Duration: 1528.79 ms Billed Duration: 1529 ms Memory Size: 128 MB

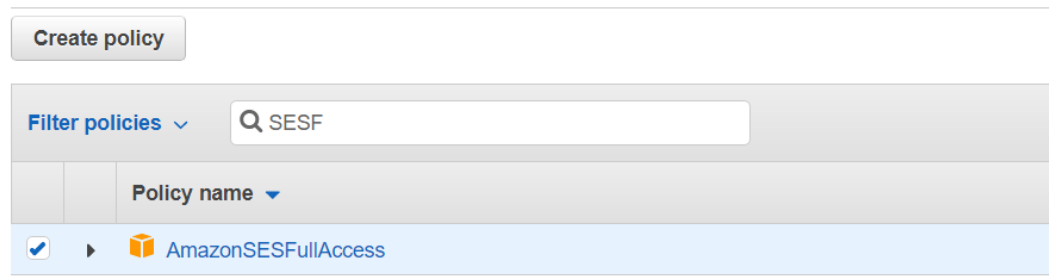
As you can see, our function logged that a file was uploaded with its file name and the bucket to which it was uploaded. It also mentions the contents inside the file as our function was defined to.

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

OPTIONAL

Sending an Email on Bucket additions to Bucket

1. Go to the IAM console and edit the same Lambda Role. This time, add SESFullAccess Permission to the role.



2. Create a new Lambda function in a Python environment. Use the existing role which was previously created.

3. In this function, above the default hello-world TODO, add the following code.

This code is basically to send an email on the creation of an object in the attached S3 Bucket. It sends the bucket name, event and source IP address.

In this code, modify the **Source** and **Destination ToAddresses** to your sender and receiver email addresses. Once done, deploy the function.

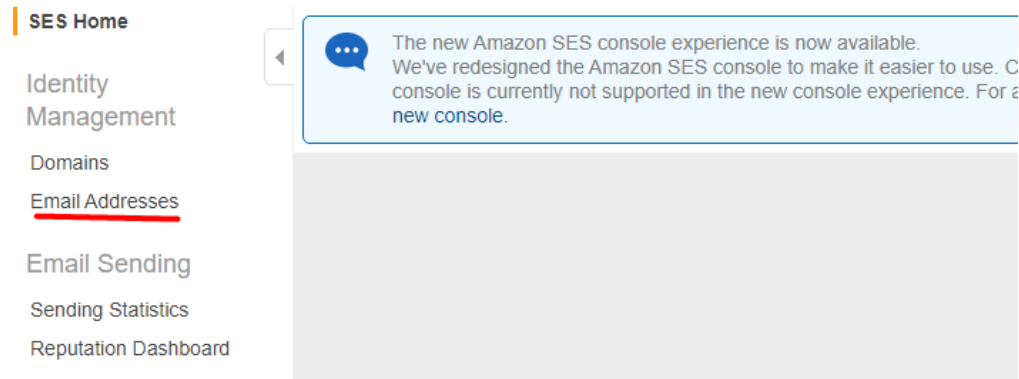
```

1 import json
2 import boto3
3 def lambda_handler(event, context):
4     for a in event["Records"]:
5         action = a["eventName"]
6         ip = a["requestParameters"]["sourceIPAddress"]
7         bucket_name = a["s3"]["bucket"]["name"]
8         object = a["s3"]["object"]["key"]
9         client = boto3.client("ses")
10        subject = str(action) + 'Event From ' + bucket_name
11        body = ""
12        <br>
13        <p>
14        Hey! This e-mail was generated to notify you about the event <strong>{}</strong>.
15        Source IP: {}
16        </p>
17        """format(action, ip)
18        message = {
19            'Subject': {
20                'Data': subject
21            },
22            'Body': {
23                'Html': {
24                    'Data': body
25                }
26            }
27        }
28        response = client.send_email(
29            Source="sreeskeshiyer@gmail.com",
30            Destination={
31                'ToAddresses': [
32                    "redacted@gmail.com"
33                ]
34            },
35            Message=message
36        )
37        # TODO implement

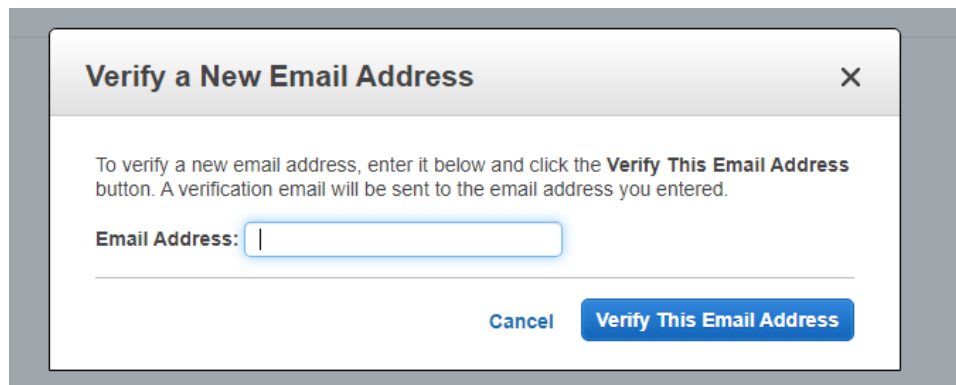
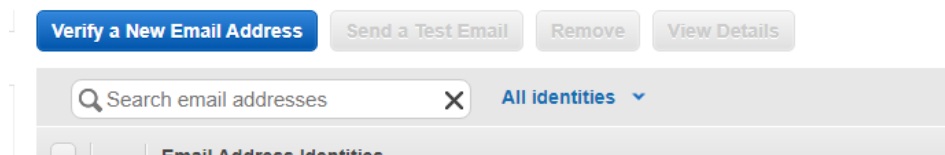
```

```
1 import json
2 import boto3
3 def lambda_handler(event, context):
4     for a in event["Records"]:
5         action = a["eventName"]
6         ip = a["requestParameters"]["sourceIPAddress"]
7         bucket_name = a["s3"]["bucket"]["name"]
8         object = a["s3"]["object"]["key"]
9         client = boto3.client("ses")
10        subject = str(action) + 'Event From ' + bucket_name
11        body = """
12            <br>
13            <p>
14                Hey! This e-mail was generated to notify you about the event <strong>{} </strong>.
15                Source IP: {}
16            </p>
17        """.format(action, ip)
18        message = {
19            'Subject': {
20                'Data': subject
21            },
22            'Body': {
23                'Html': {
24                    'Data': body
25                }
26            }
27        }
28        response = client.send_email(
29            Source="sreekeshiyer@gmail.com",
30            Destination={
31                'ToAddresses': [
32                    "sreekeshiyer@gmail.com"
33                ]
34            },
35            Message=message
36        )
37        # TODO implement
38        return {
39            'statusCode': 200,
40            'body': json.dumps('Hey there! Check your mail I guess..')
41        }
```

4. Open up the SES Console and click on Manage Email Addresses.



5. Choose Verify Email Address and verify both sender and receiver email addresses



Click on the verification links you are sent and verify the emails.

Email Address Identities	Verification Status
▶ [redacted]@gmail.com	verified
▶ sreekeshiyer@gmail.com	verified

- Now, open up the S3 Console, create a new bucket as you did previously and add an event notification inside events and attach it to your Lambda function.

Buckets (1) [Info](#)

Buckets are containers for data stored in S3. [Learn more](#)

	Name ▲	AWS Region
<input type="radio"/>	s3-trigger-email	Asia Pacific (Mumbai) ap-south-1

Event notifications (1)

Send a notification when specific events occur in your bucket. [Learn more](#)

<input type="checkbox"/>	Name	Event types	Filters
<input type="checkbox"/>	s3-put	Put	-

- Once that's done, upload any file to your S3 Bucket. I'll upload the same dummy JSON file again.

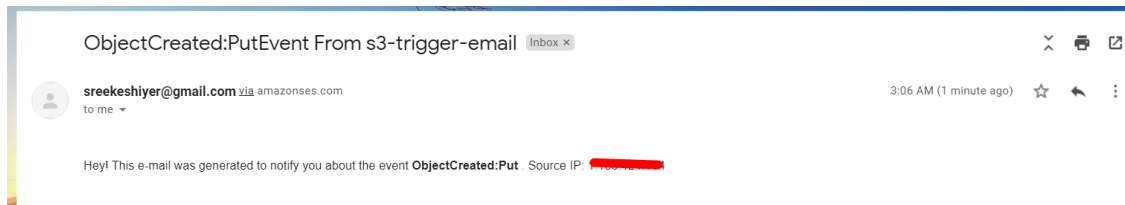
Objects (1)

Objects are the fundamental entities stored in Amazon S3. You can use [Amazon S3 inventory](#) to get a list of all objects in your bucket.

☐ Show versions

<input type="checkbox"/>	Name ▲	Type ▼	Last modified
<input type="checkbox"/>	dummy_data.json	json	October 10, 2021, 03:06:55 (U

8. Check your **ToAddress** email. You'll receive an email from the Source Address via Amazon SES.



In this way, we successfully created a function in AWS Lambda that sends an email on uploading an object to an S3 Bucket using Amazon SES.

Recommended Cleanup

Once done with the experiment, it is recommended to delete all resources which have been created and used by us to avoid charges in AWS.

Here is a list of things you may delete:

1. AWS Lambda Function
2. Amazon S3 Storage Bucket
3. Amazon SES Verified Emails
4. AWS Cloudwatch Logs (Optional, won't affect bills)
5. AWS IAM Role (the one which was created for the function, again, won't affect bills)

Conclusion:

In this experiment, we learned how to create an AWS Lambda function to log every time an object is added to an S3 Bucket.