

Case Study :-

Automated Notifications Using SNS

- Concepts Used: AWS Lambda, S3, SNS.
- Problem Statement: "Create a Lambda function that triggers when a new file is uploaded to an S3 bucket and sends an email notification using SNS with details of the uploaded file."
- Tasks:
 - Write a Python Lambda function that triggers on S3 upload events.
 - Extract the file name and size from the event and format a notification message.
 - Use SNS to send the notification to a configured email address.
 - Test by uploading a file to the S3 bucket and verifying that an email is received.

1. Introduction :-**Case Study Overview:**

This case study focuses on automating notifications for file uploads using AWS services. The system is designed to notify users when a new file is uploaded to an S3 bucket by leveraging AWS Lambda and Simple Notification Service (SNS). This automation ensures timely communication of important file uploads without manual intervention.

Key Feature and Application:

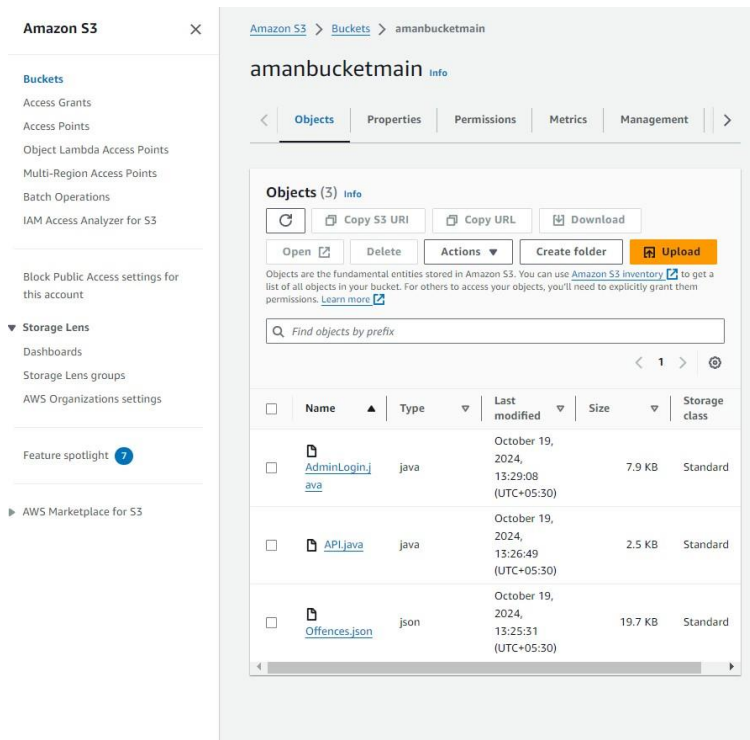
The key feature of this system is the use of an AWS Lambda function that triggers automatically whenever a file is uploaded to the S3 bucket. It extracts details such as the file name and size, formats them into a message, and sends an email notification using AWS SNS. This application is useful for scenarios requiring real-time alerts, such as file monitoring or data processing pipelines.

Third-Year Project Integration (Optional):

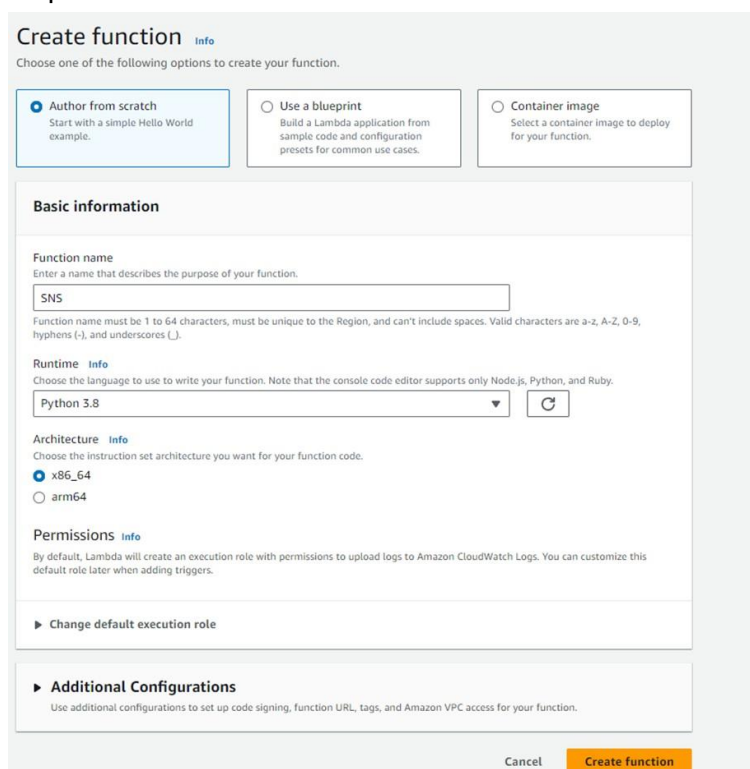
In our third-year project, we implemented a similar feature where email notifications were sent to vaccination centers when a child was registered for a vaccine. This aligns with the current case study, where notifications are automated using AWS services. In both cases, automated notifications play a key role in ensuring timely updates to relevant parties, whether it's notifying the vaccination center about a new registration or alerting users about file uploads in the S3 bucket. This demonstrates the practical application of event-driven notifications in different domains, enhancing communication and operational efficiency.

2. Step-by-Step Explanation :-

Step 1 :- Create a s3 Bucket



Step 2:- Create a lambda function :-



Step 3:- Go to the sns lambda function -> configuration

Lambda > Functions > SNS > Edit basic settings

Edit basic settings

Basic settings [Info](#)

Description - *optional*

Memory [Info](#)
Your function is allocated CPU proportional to the memory configured.
1024 MB
Set memory to between 128 MB and 10240 MB

Ephemeral storage [Info](#)
You can configure up to 10 GB of ephemeral storage (/tmp) for your function. [View pricing](#)
1024 MB
Set ephemeral storage (/tmp) to between 512 MB and 10240 MB.

SnapStart [Info](#)
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](#).
None
Supported runtimes: Java 11, Java 17, Java 21.

Timeout
15 min 0 sec

Execution role
Choose a role that defines the permissions of your function. To create a custom role, go to the [IAM console](#).
☒ 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.
service-role/SNS-role-zzjqxt4
View the SNS-role-zzjqxt4 role [on the IAM console](#).

Step 4: Add the trigger :-

Lambda > Add triggers

Add trigger

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.
s3/amanbucketmain
Bucket region: eu-north-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

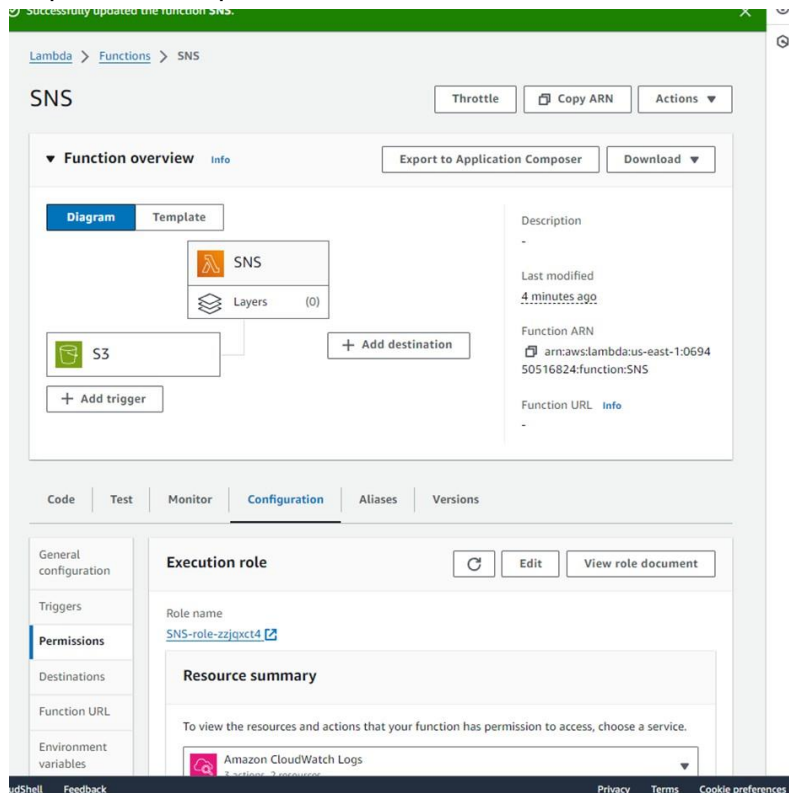
Prefix - *optional*
Enter a single optional prefix to limit the notifications to objects with keys that start with matching characters. Any [special characters](#) must be URL encoded.
e.g. images/

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.
e.g. .jpg

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.

Step 5 :- After Configuration your Code look

Step 6 :- Go the permission



Step 7:- when you click on this link role name you will see the trust relationships do the changes in services

"Service": [

```
"lambda.amazonaws.com",  
"glue.amazonaws.com",  
"rds.amazonaws.com",  
"textract.amazonaws.com",  
"events.amazonaws.com",  
"transfer.amazonaws.com"
```

]

Step 8:- Save those changes

[IAM](#) > [Roles](#) > [SNS-role-zzjqxt4](#) > Edit trust policy

Edit trust policy

```
1 {
2   "Version": "2012-10-17",
3   "Statement": [
4     {
5       "Effect": "Allow",
6       "Principal": {
7         "Service": [
8           "lambda.amazonaws.com",
9           "glue.amazonaws.com",
10          "rds.amazonaws.com",
11          "textract.amazonaws.com",
12          "events.amazonaws.com",
13          "transfer.amazonaws.com"
14        ]
15      },
16      "Action": "sts:AssumeRole"
17    }
18  ]
19 }
```

Edit statement

Select a statement

Select an existing statement in the p
add a new statement.

+ Add new statement

Step 9:- Now go the Amazon SNS create topic

New Feature

Amazon SNS now supports in-place message archiving and replay for FIFO topics. [Learn more](#)

Amazon SNS

>

Topics

>

Create topic

Create topic

Details

Type [Info](#)

Topic type cannot be modified after topic is created

☐ FIFO (first-in, first-out)

- Strictly-preserved message ordering
- Exactly-once message delivery
- High throughput, up to 300 publishes/second
- Subscription protocols: SQS

☒ Standard

- Best-effort message ordering
- At-least once message delivery
- Highest throughput in publishes/second
- Subscription protocols: SQS, Lambda, HTTP, SMS, email, mobile application endpoints

Name

MyTopic

Maximum 256 characters. Can include alphanumeric characters, hyphens (-) and underscores (_).

Display name - *optional* [Info](#)

To use this topic with SMS subscriptions, enter a display name. Only the first 10 characters are displayed in an SMS message.

My Topic

Maximum 100 characters.

► Encryption - *optional*

Amazon SNS provides in-transit encryption by default. Enabling server-side encryption adds at-rest encryption to your topic.

▼ Access policy - *optional* [Info](#)

This policy defines who can access your topic. By default, only the topic owner can publish or subscribe to the topic.

Step 10 :- Now edit access policy and delivery status logging

▼ Access policy - *optional* [Info](#)

This policy defines who can access your topic. By default, only the topic owner can publish or subscribe to the topic.

Choose method

☒ Basic

Use simple criteria to define a basic access policy.

☐ Advanced

Use a JSON object to define an advanced access policy.

Publishers

Specify who can publish messages to the topic.

Everyone

Anybody can publish

Subscribers

Specify who can subscribe to this topic.

Everyone

Any AWS account can subscribe to the topic

JSON preview

```
"Principal": {
  "AWS": "*"
},
"Action": [
  "SNS:Publish",
  "SNS:RemovePermission",
  "SNS:SetTopicAttributes",
  "SNS:DeleteTopic",
  "SNS:ListSubscriptionsByTopic",
  "SNS:GetTopicAttributes",
  "SNS:AddPermission",
  "SNS:Subscribe"
],
"Resource": "*"
}
```

▼ Delivery status logging - optional [info](#)

These settings configure the logging of message delivery status to CloudWatch Logs.

Log delivery status for these protocols

- ☐ AWS Lambda
- ☐ Amazon SQS
- ☐ HTTP/S
- ☐ Platform application endpoint
- ☐ Amazon Kinesis Data Firehose

Success sample rate

The percentage of successful message deliveries to log.

%

IAM roles

Amazon SNS requires permission to write logs to CloudWatch Logs. You can use separate roles for successful and failed message deliveries.

Service role [info](#)

☐ Use existing service role
Choose an existing service role from your account

☒ Create new service role
Create a new service role in your account

[Create new roles](#)

IAM role for successful deliveries

-

IAM role for failed deliveries

-

✔ Topic MyTopic created successfully.

You can create subscriptions and send messages to them from this topic.

[Publish message](#)

[Amazon SNS](#) > [Topics](#) > MyTopic

MyTopic

[Edit](#)

[Delete](#)

[Publish message](#)

Details

Name	Display name
MyTopic	-
ARN	Topic owner
arn:aws:sns:us-east-1:069450516824:MyTopic	069450516824
Type	
Standard	

< [Subscriptions](#) | [Access policy](#) | [Data protection policy](#) | [Delivery](#) >

Subscriptions (0)

[Edit](#) [Delete](#) [Request confirmation](#) [Confirm subscription](#)

[Create subscription](#)

Step 11 :- Go to subscription in sns choose the topic then copy that topic arn paste in code in lambda function

```
import boto3
```

```
topic_arn = "arn:aws:sns:us-east-1:069450516824:PratikTopic"
```

```
def send_sns(message, subject):
```

```
    try:
```

```
        client = boto3.client("sns")
```

```
        result = client.publish(TopicArn=topic_arn, Message=message, Subject=subject)
```

```
        if result['ResponseMetadata']['HTTPStatusCode'] == 200:
```

```
            print(result)
```

```
            print("Notification sent successfully..!!!")
```

```
            return True
```

```
    except Exception as e:
```

```
        print("Error occurred while publishing notification: ", e)
```

```
        return False
```

```
def lambda_handler(event, context):
```

```
    print("Event collected: {}".format(event))
```

```
    for record in event['Records']:
```

```
        # Extract bucket name and key
```

```
        s3_bucket = record['s3']['bucket']['name']
```

```
        s3_key = record['s3']['object']['key']
```

```
        s3_size = record['s3']['object']['size']
```

```
        print("Bucket name: {}".format(s3_bucket))
```

```
        print("File key: {}".format(s3_key))
```

```
        print("File size: {} bytes".format(s3_size))
```

```
        from_path = "s3://{}/{}".format(s3_bucket, s3_key)
```

```
        # Construct message with file name and size
```

```
        message = "A file has been uploaded at {}\nFilename: {}\nFile size: {} bytes".format(from_path, s3_key, s3_size)
```

```
        subject = "S3 File Upload Notification"
```

```
        # Send SNS notification
```

```
        SNSResult = send_sns(message, subject)
```

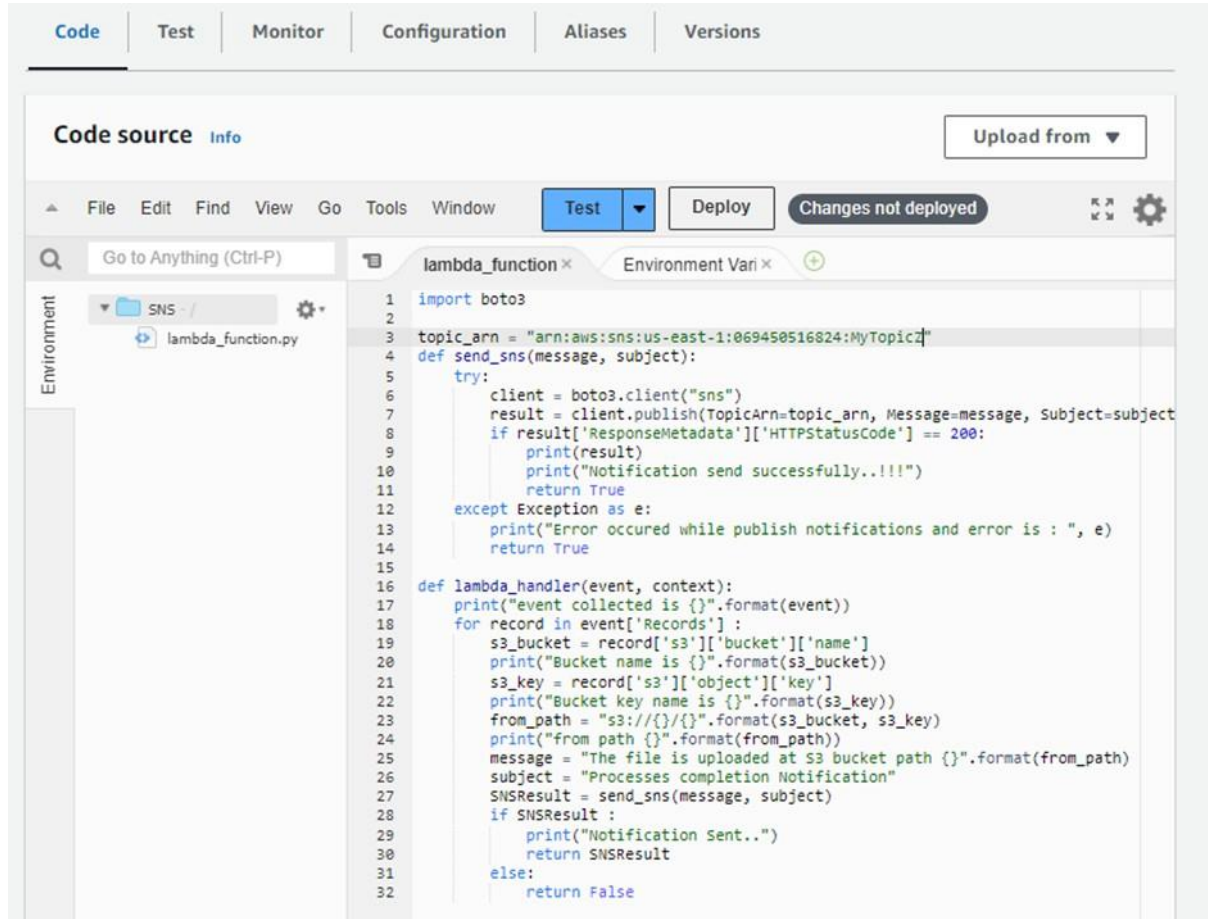
```
        if SNSResult:
```

```
            print("Notification Sent..")
```

```
            return SNSResult
```

```
        else:
```

```
            return False
```

```
1 import boto3
2
3 topic_arn = "arn:aws:sns:us-east-1:069450516824:MyTopic"
4 def send_sns(message, subject):
5     try:
6         client = boto3.client("sns")
7         result = client.publish(TopicArn=topic_arn, Message=message, Subject=subject)
8         if result['ResponseMetadata']['HTTPStatusCode'] == 200:
9             print(result)
10            print("Notification send successfully..!!!")
11            return True
12        except Exception as e:
13            print("Error occurred while publish notifications and error is : ", e)
14            return True
15
16 def lambda_handler(event, context):
17     print("event collected is {}".format(event))
18     for record in event['Records']:
19         s3_bucket = record['s3']['bucket']['name']
20         print("Bucket name is {}".format(s3_bucket))
21         s3_key = record['s3']['object']['key']
22         print("Bucket key name is {}".format(s3_key))
23         from_path = "s3://{}/{}".format(s3_bucket, s3_key)
24         print("from path {}".format(from_path))
25         message = "The file is uploaded at S3 bucket path {}".format(from_path)
26         subject = "Processes completion Notification"
27         SNSResult = send_sns(message, subject)
28         if SNSResult:
29             print("Notification Sent..")
30             return SNSResult
31         else:
32             return False
```

Step 12 :- Now Create a Subscription :

[Amazon SNS](#) > [Subscriptions](#) > Create subscription

Create subscription

Details


Topic ARN

Protocol

The type of endpoint to subscribe

Endpoint

An email address that can receive notifications from Amazon SNS.

 After your subscription is created, you must confirm it. [Info](#)

► **Subscription filter policy - optional** [Info](#)

This policy filters the messages that a subscriber receives.

► **Redrive policy (dead-letter queue) - optional** [Info](#)

Send undeliverable messages to a dead-letter queue.

[Cancel](#)[Create subscription](#)[Amazon SNS](#) > [Subscriptions](#)

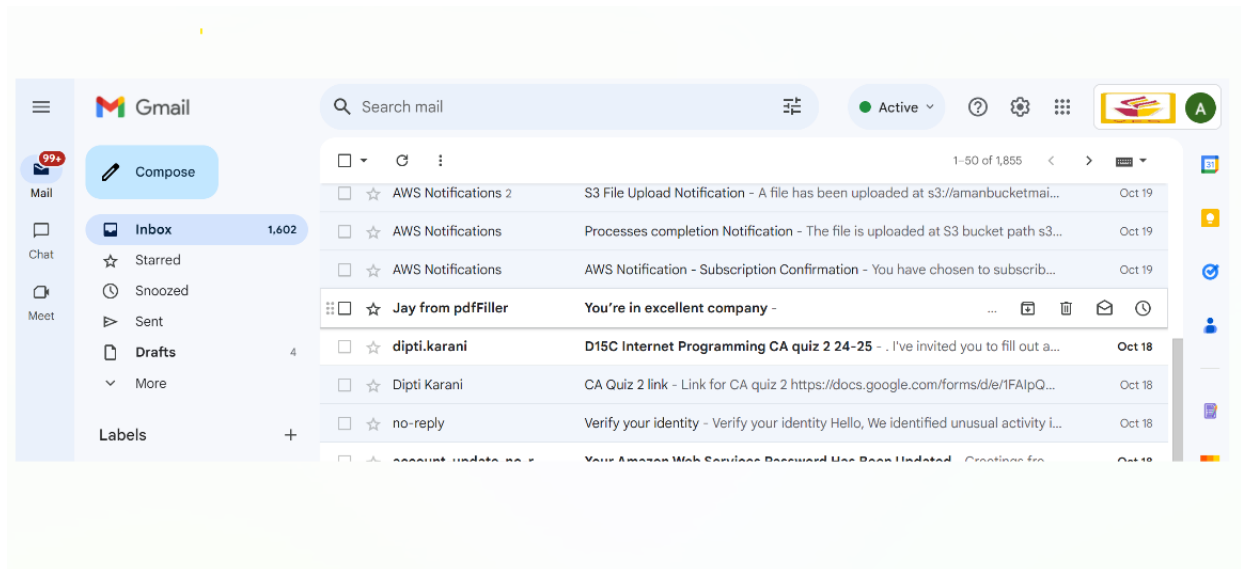
Subscriptions (1)

[Edit](#)[Delete](#)[Request confirmation](#)[Confirm subscription](#)[Create subscription](#)< 1 > 

ID	Endpoint	Status	Protocol	Topic
a5f59895-61eb-...	2022.aman.yada...	 Confirmed	EMAIL	amanT...

Step 13 :- Go to the email and verify it

Step 14 : - Upload files in S3 bucket You will get like this email



Conclusion

In conclusion, this case study showcases the power of AWS services, such as Lambda, S3, and SNS, to automate notifications in a highly efficient manner. By triggering events based on file uploads and delivering real-time alerts via email, the system ensures timely communication without manual intervention. This approach can be adapted to various use cases, including those involving data pipelines, file monitoring, or real-time system notifications. Furthermore, the integration with the third-year project highlights how similar event-driven mechanisms, such as sending email notifications to vaccination centers, can enhance practical applications across different fields, making operations smoother and more efficient.