

# Serverless File Processing & Monitoring System using AWS

## Abstract

This project demonstrates a simple yet powerful serverless architecture using Amazon Web Services (AWS). The system automatically detects file uploads to an Amazon S3 bucket, processes events using AWS Lambda, logs details to Amazon CloudWatch, and sends real-time notifications through Amazon SNS. The project showcases essential AWS concepts, including serverless computing, event-driven automation, cloud monitoring, and secure resource management.

## 1. Introduction

AWS provides a scalable and cost-effective cloud platform for deploying applications without managing servers. Serverless architecture allows functions to run only when triggered, reducing operational overhead. In this project, a file monitoring system is built using AWS S3, Lambda, CloudWatch, and SNS. Whenever a user uploads a file, the system automatically triggers a workflow: processing, logging, and alerting.

This project aligns with AWS Cloud Practitioner concepts and demonstrates hands-on cloud implementation using AWS core services.

## 2. Aim

To build a fully serverless file processing and monitoring system using AWS services that automatically responds to S3 file uploads and provides real-time visibility through monitoring and notifications.

## 3. Objectives

1. To implement a serverless event-driven workflow using AWS Lambda.
2. To store and manage files using Amazon S3.
3. To enable real-time logs and monitoring using Amazon CloudWatch.
4. To configure Amazon SNS for alert notifications.
5. To demonstrate basic cloud security using IAM roles and permissions.

#### **4. AWS Services Used**

- **Amazon S3 – File storage and event trigger**
- **AWS Lambda – Event-driven compute function**
- **Amazon SNS – Email notifications to the user**
- **Amazon CloudWatch – Logs, monitoring, and dashboards**
- **AWS IAM – Secure access and execution roles**

#### **5. System Architecture**

##### **Workflow:**

1. User uploads a file to Amazon S3
2. S3 sends an event to AWS Lambda
3. Lambda processes the event and logs details
4. Logs appear in CloudWatch
5. Lambda sends notification through Amazon SNS
6. User receives an email alert

#### **6. Implementation Steps**

##### **Step 1 — S3 BUCKET SETUP (Storage Lead)**

###### **Steps to Perform**

###### **1. Create S3 Bucket**

1. Go to **AWS Console → S3**
2. Click **Create bucket**
3. Enter bucket name → serverless-file-bucket-yourname
4. Region → **us-east-1**
5. Keep **Block all public access = ON**
6. Click **Create bucket**

###### **2. Create Folder Structure**

Inside the bucket:

- Create folder: uploads/
- Create folder: processed/

### **3. Upload Test Files**

1. Go to folder uploads/
2. Click **Upload → Add files**
3. Upload test1.txt, sample2.txt

### **4. Verify Bucket**

- Check that files show up
- Confirm bucket name for other members

## **Step 2 — IAM SETUP (Security Lead)**

### **Steps to Perform**

#### **1. Create IAM Role for Lambda**

1. Go to **IAM → Roles → Create role**
2. Select **AWS Service → Lambda**
3. Attach policies:
  - AmazonS3FullAccess
  - CloudWatchLogsFullAccess
  - AmazonSNSFullAccess
4. Name the role:  
**LambdaS3ExecutionRole**

#### **2. Create IAM User (optional for demo)**

- Username → project-user
- Permission → S3FullAccess
- Enable **MFA**
- Download credentials

#### **3. Security Verification**

- Ensure no public access to bucket

- Ensure Lambda uses the created role

### Step 3 — LAMBDA FUNCTION (Automation Lead)

#### Steps to Perform

##### 1. Create Lambda Function

1. Go to **Lambda**
2. Click **Create function**
3. Choose **Author from scratch**
4. Name → FileEventProcessor
5. Runtime → **Python 3.9**
6. Execution role → **Use existing role** → **LambdaS3ExecutionRole**

##### 2. Add S3 Trigger

1. Go to Lambda → **Triggers**
2. Add Trigger → **S3**
3. Select bucket: serverless-file-bucket-yourname
4. Event type: **All object create events**
5. Prefix: uploads/

##### 3. Add Code

###### Paste this code:

```
import json
import boto3
import urllib.parse

sns = boto3.client('sns')
TOPIC_ARN = "YOUR_SNS_TOPIC_ARN"

def lambda_handler(event, context):
    for rec in event['Records']:
        bucket = rec['s3']['bucket']['name']
        key = urllib.parse.unquote_plus(rec['s3']['object']['key'])
```

```

event_name = rec['eventName']

if event_name.startswith("ObjectCreated"):
    message = f"File uploaded: {key} in {bucket}"
elif event_name.startswith("ObjectRemoved"):
    message = f"File deleted: {key} from {bucket}"
else:
    message = f"Other event: {event_name}"

sns.publish(
    TopicArn=TOPIC_ARN,
    Message=message,
    Subject="S3 Notification"
)

return {"statusCode": 200}

```

Click **Deploy**.

#### 4. Test Lambda

- Upload a new file to S3 → uploads/test3.txt
- Check logs in CloudWatch

### Step 4 — CLOUDWATCH MONITORING (Observability Lead)

#### Steps to Perform

##### 1. View Lambda Logs

1. Go to **CloudWatch → Logs**
2. Click log group: /aws/lambda/FileEventProcessor
3. Open latest log stream
4. Verify log:  
“New file uploaded: uploads/test3.txt”

##### 2. Create CloudWatch Dashboard

1. Go to **Dashboards → Create Dashboard**

2. Name: ServerlessMonitoringDashboard

3. Add widgets:

- Lambda Invocations
- Lambda Errors
- S3 Metrics
- SNS Delivery Status

### **3. Create Alarm (Optional)**

1. Go to **Alarms** → **Create alarm**
2. Select Errors metric for Lambda
3. Condition → Errors ≥ 1
4. Notification → SNS topic

## **Step 5: SNS NOTIFICATIONS (Alerting Lead)**

### **Steps to Perform**

#### **1. Create SNS Topic**

1. Go to **SNS** → **Topics** → **Create topic**
2. Type → **Standard**
3. Name → FileUploadAlerts

#### **2. Add Email Subscription**

1. SNS → Subscriptions → Create subscription
2. Protocol → Email
3. Enter team email IDs
4. Each member must click **Confirm Subscription** in email

#### **3. Connect SNS to Lambda**

Replace in code:

```
TopicArn='YOUR_SNS_TOPIC_ARN',
```

With your actual SNS topic ARN.

#### **4. Test Email Alert**

- Upload AlertTest.txt to S3
- All team members should receive email:  
**“New file uploaded: uploads/AlertTest.txt”**

## 7. OUTPUT

- S3 successfully stored uploaded files
- Lambda function triggered automatically
- CloudWatch generated real-time logs
- SNS sent email notifications instantly
- Full serverless workflow executed without manual intervention

## 8. Screenshot:

### Step 1 — S3 BUCKET SETUP (Storage Lead)

The image consists of three vertically stacked screenshots of the AWS S3 console, illustrating the steps to set up an S3 bucket for file sharing.

**Screenshot 1: Summary Page**

This screenshot shows the "Summary" page of the S3 bucket configuration. It displays two sections: "Succeeded" (2 files, 0 B (0%)) and "Failed" (0 files, 0 B (0%)). Below this, the "Files and folders" tab is selected, showing a table with two entries: "SAMPLE 1.txt" and "TEST1.txt", both of which have a status of "Succeeded".

Name	Folder	Type	Size	Status	Error
SAMPLE 1.txt	-	text/plain	0 B	Succeeded	-
TEST1.txt	-	text/plain	0 B	Succeeded	-

**Screenshot 2: Upload Page**

This screenshot shows the "Upload" page where files are being prepared for upload. A message at the top says "Add the files and folders you want to upload to S3. To upload a file larger than 160GB, use the AWS CLI, AWS SDKs or Amazon S3 REST API." Below this is a large dashed box for dragging and dropping files. A "Files and folders" table shows the same two files ("SAMPLE 1.txt" and "TEST1.txt") ready for upload. Buttons for "Remove", "Add files", and "Add folder" are visible.

Name	Folder	Type	Size
SAMPLE 1.txt	-	text/plain	0 B
TEST1.txt	-	text/plain	0 B

**Screenshot 3: Bucket Details Page**

This screenshot shows the "Objects" tab of the bucket details page. A green success message at the top states "Successfully created folder 'processed'." The table lists two objects: "processed/" and "uploads/", both of which are folders. The "Actions" dropdown menu is open, showing options like "Create folder" and "Upload".

Name	Type	Last modified	Size	Storage class
processed/	Folder	-	-	-
uploads/	Folder	-	-	-

Screenshot of the AWS S3 console showing the General purpose buckets list. A single bucket named "serverless-file-bucket-sharing" is listed under the "General purpose buckets" tab.

Name	AWS Region	Creation date
serverless-file-bucket-sharing	US East (N. Virginia) us-east-1	November 24, 2025, 15:07:24 (UTC+05:30)

The sidebar on the left shows various S3 management options like General purpose buckets, Directory buckets, Table buckets, etc.

Screenshot of the "Create bucket" wizard step 1: General configuration. The "General purpose" bucket type is selected.

**Bucket Key**  
Using an S3 Bucket Key for SSE-KMS reduces encryption costs by lowering calls to AWS KMS. S3 Bucket Keys aren't supported for DSSE-KMS. [Learn more](#)

Enable

#### Advanced settings

After creating the bucket, you can upload files and folders to the bucket, and configure additional bucket settings.

[Cancel](#) [Create bucket](#)

Screenshot of the "Create bucket" wizard step 2: General configuration. The "General purpose" bucket type is selected.

**Bucket name** [Info](#)  
serverless-file-bucket-sharing

Bucket names must be 3 to 63 characters and unique within the global namespace. Bucket names must also begin and end with a letter or number. Valid characters are a-z, 0-9, periods (.), and hyphens (-). [Learn more](#)

**Copy settings from existing bucket - optional**  
Only the bucket settings in the following configuration are copied.  
[Choose bucket](#)

## Step 2 — IAM SETUP (Security Lead)

The screenshots below show the AWS IAM setup process for creating a Lambda execution role.

**Step 1: Select trusted entity**

**Step 2: Name, review, and create**

**Step 3: Select trusted entities**

## Step 3 — LAMBDA FUNCTION (Automation Lead)

The screenshot shows the 'Create function' configuration page for AWS Lambda. In the 'Execution role' section, 'Use an existing role' is selected, and 'LambdaS3ExecutionRole' is chosen from the dropdown. Below this, under 'Additional configurations', there is a note about setting up networking, security, and governance.

**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.  
LambdaS3ExecutionRole

**Additional configurations**  
Use additional configurations to set up networking, security, and governance for your function. These settings help secure and customize your Lambda function deployment.

**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/serverless-file-bucket-sharing

**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.  
uploads/

**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.

Lambda will add the necessary permissions for AWS S3 to invoke your Lambda function from this trigger. [Learn more](#) about the Lambda permissions model.

**Add**

Screenshot of the AWS Lambda Function Overview page for 'FileEventProcessor'.

**Function overview**

Description: -

Last modified: 21 seconds ago

Function ARN: arn:aws:lambda:us-east-1:03768848471:function:FileEventProcessor

Function URL: -

Triggers:

- S3 (Selected)
- + Add trigger

Actions: Throttle, Copy ARN, Actions ▾

**Code Editor**

File: lambda\_function.py

```

1 import json
2
3 def lambda_handler(event, context):
4     bucket_name = event['Records'][0]['s3']['bucket']['name']
5     file_name = event['Records'][0]['s3']['object']['key']
6
7     print(f"New note uploaded: {file_name} in bucket: {bucket_name}")
8
9     return {
10         'statusCode': 200,
11         'body': json.dumps('Note upload logged successfully!')
12     }
  
```

Deploy: Deploy (Ctrl+Shift+U), Test (Ctrl+Shift+I)

**CloudWatch Log Events**

Log events:

Timestamp	Message
2025-11-24T09:55:34.308Z	INIT_START Runtime Version: python:3.9.v125 Runtime Version ARN: arn:aws:lambda:us-east-1::run...
2025-11-24T09:55:34.389Z	START RequestId: 5bbfba02-d75a-49d5-a223-3a043e9ce0ed Version: \$LATEST
2025-11-24T09:55:34.390Z	New note uploaded: uploads/SAMPLE+2.txt in bucket: serverless-file-bucket-sharing
2025-11-24T09:55:34.391Z	END RequestId: 5bbfba02-d75a-49d5-a223-3a043e9ce0ed
2025-11-24T09:55:34.391Z	REPORT RequestId: 5bbfba02-d75a-49d5-a223-3a043e9ce0ed Duration: 1.34 ms Billed Duration: 79 ms...

No newer events at this moment. Auto retrying... Pause

## Step 4 — CLOUDWATCH MONITORING (Observability Lead)

The screenshot shows the AWS CloudWatch Metrics interface. At the top, there is a search bar and navigation links for CloudWatch Dashboards. A modal window titled "Create new dashboard" is open, showing a "Dashboard name" input field with "ServerlessMonitoringDashboard" typed in. Below the input field is a note: "Valid characters in dashboard names include '0-9A-Za-z-\_-'." There are "Cancel" and "Create dashboard" buttons at the bottom of the modal.

Below the modal, the main dashboard view is visible. It features a "Add metric graph" section with a "Metrics" tab selected. Under "Metrics", there is a table titled "Graphed metrics (1)". The table lists three metrics from the "FileEventProcessor" function: "AsyncEventAge", "AsyncEventsDropped", and "Errors". The "Errors" row has a checked checkbox and is highlighted with a blue border. Other rows have unchecked checkboxes. The table includes columns for "Metric name", "Unit", and "Alarms".

At the bottom of the "Add metric graph" section, there are "Cancel" and "Create widget" buttons. The main dashboard below this section shows a single metric graph titled "Errors" with a count of 1. The graph has a Y-axis labeled "Count" with values 0, 0.5, and 1. The X-axis shows a timeline from 07:05 to 10:05. The graph area is light gray with a blue line representing the error count over time.

## Step 5: SNS NOTIFICATIONS (Alerting Lead)

The screenshots illustrate the process of setting up SNS notifications for file uploads.

**Screenshot 1: Create topic**

**Details**

Type: **Standard** (selected)

- FIFO (first-in, first-out)
  - Strictly-preserved message ordering
  - Exactly-once message delivery
  - Subscription protocols: SQS, Lambda, Data Firehose, HTTP, SMS, email, mobile application endpoints
- Standard
  - Best-effort message ordering
  - At-least once message delivery
  - Subscription protocols: SQS, Lambda, Data Firehose, HTTP, SMS, email, mobile application endpoints

Name: FileUploadAlerts

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.

**Screenshot 2: Create subscription (Email)**

**Details**

Topic ARN: arn:aws:sns:us-east-1:037688848471:FileUploadAlerts

Protocol: Email

Endpoint: An email address that can receive notifications from Amazon SNS.  
jupg24mca23173@jainuniversity.ac.in

After your subscription is created, you must confirm it. Info

**Screenshot 3: Create subscription (Email)**

**Details**

Topic ARN: arn:aws:sns:us-east-1:037688848471:FileUploadAlerts

Protocol: Email

Endpoint: An email address that can receive notifications from Amazon SNS.  
santhosh.govindan0932@gmail.com

After your subscription is created, you must confirm it. Info

Screenshot of the AWS Lambda Functions console showing the successful update of the function FileEventProcessor.

The Lambda function code (lambda\_function.py) is displayed:

```
def lambda_handler(event, context):
    key = event['Records'][0]['s3']['bucket']['name']
    file_name = event['Records'][0]['s3']['object']['key']

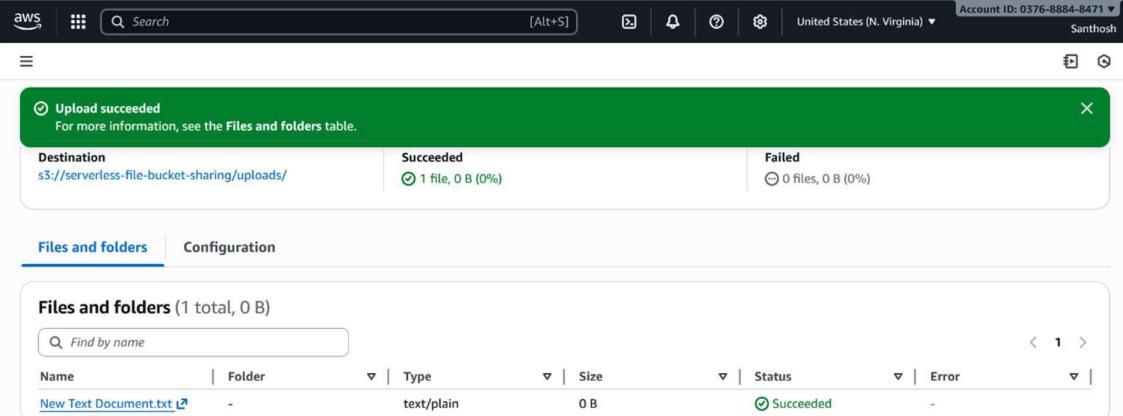
    message = f"New file uploaded: {file_name} in bucket: {key}"
    print(message)

    sns.publish(
        TopicArn='arn:aws:sns:us-east-1:037688848471:FileUploadAlerts',
        Message=message,
        Subject='New File Upload Alert'
    )

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

A green success message at the top right of the editor area states: "Successfully updated the function FileEventProcessor."

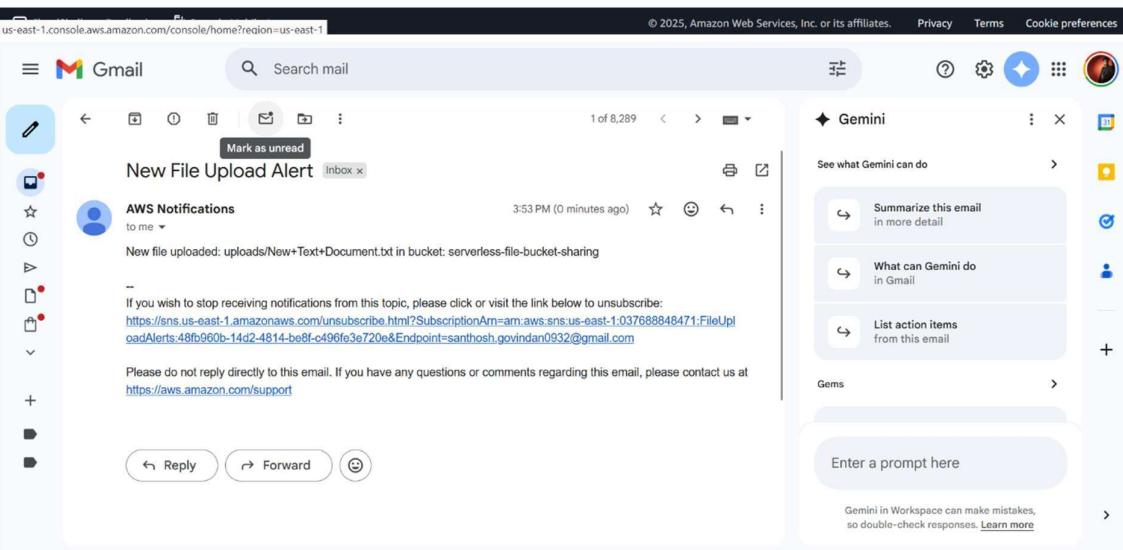
## Final verification:



The screenshot shows the AWS Lambda console interface. A green success message at the top states "Upload succeeded" and "For more information, see the Files and folders table." Below this, a table summarizes the upload results:

Destination	Succeeded	Failed
s3://serverless-file-bucket-sharing/uploads/	1 file, 0 B (0%)	0 files, 0 B (0%)

Below the table, there are two tabs: "Files and folders" (selected) and "Configuration". Under "Files and folders", it says "(1 total, 0 B)" and lists a single file: "New Text Document.txt" (text/plain, 0 B, Succeeded).

The screenshot shows the Gmail inbox. An email from "AWS Notifications" titled "New File Upload Alert" is selected. The email body contains the following text:

New file uploaded: uploads/New+Text+Document.txt in bucket: serverless-file-bucket-sharing

If you wish to stop receiving notifications from this topic, please click or visit the link below to unsubscribe:  
<https://sns.us-east-1.amazonaws.com/unsubscribe.html?SubscriptionArn=arn:aws:sns:us-east-1:037688848471:FileUploadAlerts:48fb960b-14d2-4814-be8f-c496fe3e720&Endpoint=santhosh.govindan0932@gmail.com>

Please do not reply directly to this email. If you have any questions or comments regarding this email, please contact us at  
<https://aws.amazon.com/support>

To the right of the email, there is a Gemini AI sidebar with options like "Summarize this email", "What can Gemini do in Gmail", and "List action items from this email".

## 9. Conclusion

This project demonstrates a complete serverless event-driven workflow using AWS cloud services. With S3 for storage, Lambda for processing, CloudWatch for monitoring, and SNS for notifications, the system showcases a practical real-world cloud solution. The implementation proves how AWS serverless computing can simplify development and automate workflows with minimal setup and zero server management.

