

## **AdvDevOps Case Study 12: Serverless Logging with S3 and Lambda**

- **Concepts Used:** AWS Lambda, S3, and AWS Cloud9.
- **Problem Statement:** "Set up a Lambda function using AWS Cloud9 that triggers when a text file is uploaded to an S3 bucket. The Lambda function should read the file's content and log it."
- **Tasks:**
  - Create a Lambda function in Python using AWS Cloud9.
  - Configure an S3 bucket as the trigger for the Lambda function.
  - Upload a text file to the S3 bucket and verify that the Lambda function logs the content.

### **Note\*\***

AWS **Cloud9** has been **discontinued**, so we will now use **EC2** for our development environment.

### **Overview:**

This case study sets up a serverless logging system using AWS Lambda and S3. A Lambda function is triggered when a text file is uploaded to an S3 bucket, logs the content, and stores it in CloudWatch for real-time monitoring. EC2 is used for development instead of Cloud9.

### **Key Features:**

- **Serverless Automation:** Lambda auto-triggers on file uploads to S3.
- **Real-Time File Logging:** Logs content instantly in CloudWatch.
- **Cost-Efficient:** Pay only for Lambda execution time.
- **EC2 Development:** Lambda code is deployed via EC2.

### **Applications:**

- **Automated File Processing:** Ideal for handling log or data uploads.
- **Real-Time Monitoring:** Immediate insights with CloudWatch.
- **Scalable & Cost-Effective:** Flexible solution across multiple industries.

### **Real-Life Applications:**

- **Document Processing:** Automatically log and track uploaded documents for compliance.
- **IT Log Monitoring:** Capture system logs in real-time for issue detection.
- **Data Ingestion:** Automate processing of uploaded datasets for analytics.

## Step-by-Step Explanation:

### 1. Launch an EC2 Instance and Connect via SSH.

Settings for the EC2 instance:

- AMI: Choose Amazon Linux 2.
- Instance Type: Select t2.micro.
- Key Pair: Create a new key pair (or select an existing one). You'll need this for SSH access.
- Network Settings:
  - Choose default VPC.
  - Security Group: Create a new security group:
    - Inbound Rules:
      - SSH (TCP port 22): Allow from your IP.
      - HTTP (TCP port 80): allows browser access.
      - HTTPS (TCP port 443): for secure traffic.
    - Outbound Rules:
      - Allow all outbound traffic (default).

### 2. Create Access keys for Root user

2.1 Access the Root User Security Credentials:

- In the top-right corner of AWS Management Console, click on your account name or email address, and then click Security Credentials from the dropdown menu.

2.2 Manage Root Access Keys:

- Scroll down to the Access keys for the root account section.
- If you don't have any existing access keys, click on Create New Access Key.
  - This will generate an Access Key ID and a Secret Access Key for your root user.
- Download the keys or copy them immediately. You won't be able to see the Secret Access Key again after closing this page.


#### Access key

If you lose or forget your secret access key, you cannot retrieve it. Instead, create a new access key and make the old key inactive.

##### Access key

 AKIA4QPHHSOK5AD6PEO4

##### Secret access key

 \*\*\*\*\* [Show](#)

### 3. Install AWS CLI and Configure EC2

#### 3.1 Update packages and install AWS CLI:

```
sudo yum update -y
sudo yum install aws-cli -y
```

#### 3.2 Configure AWS CLI:

```
aws configure
```

Enter your:

- AWS Access Key ID
- AWS Secret Access Key
- Region (e.g., us-east-1)
- Output format: json

```
[ec2-user@ip-172-31-33-47 ~]$ aws configure
AWS Access Key ID [None]: AKIA4QPHHSOK5AD6PE04
AWS Secret Access Key [None]: 
Default region name [None]: us-east-1
Default output format [None]: json
[ec2-user@ip-172-31-33-47 ~]$
```

#### 3.3 Install Python and pip:

```
sudo yum install python3 -y
sudo yum install python3-pip -y
```

### 4. Create a S3 Bucket

Keep the region same as your AWS Configuration (e.g., us-east-1).

Keep other settings default.

General purpose buckets (1) <a href="#">Info</a> <a href="#">All AWS Regions</a>						Copy ARN	Empty	Delete	Create bucket
Buckets are containers for data stored in S3.									
<input type="text" value="Find buckets by name"/>					< 1 >				
	Name	AWS Region	IAM Access Analyzer	Creation date					
	<a href="#">dev12-lambda-s3-bucket</a>	Europe (Stockholm) eu-north-1	<a href="#">View analyzer for eu-north-1</a>	October 20, 2024, 17:25:16 (UTC+05:30)					

## 5. Create the Lambda Function code.

5.1 On your EC2 instance, create the Python Lambda function code:

```
nano lambda_function.py
```

5.2 Write the following Lambda function to read the uploaded file from S3:

```
import json
import boto3

s3 = boto3.client('s3')

def lambda_handler(event, context):
    # Get the bucket name and the uploaded file's key
    bucket_name = event['Records'][0]['s3']['bucket']['name']
    file_key = event['Records'][0]['s3']['object']['key']

    # Fetch the file from S3
    file_obj = s3.get_object(Bucket=bucket_name, Key=file_key)
    file_content = file_obj['Body'].read().decode('utf-8')

    # Log the content of the file
    print(f"File Content from {file_key}:")
    print(file_content)

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

5.3 Press Ctrl+X, then Y, and hit Enter to save the file.

## 6. Deploy the Lambda function from EC2

### 6.1 Package the Lambda function:

```
zip function.zip lambda_function.py
```

### 6.2 Create a Lambda function in AWS Console:

Choose Author from Scratch:

- Function Name: S3TextFileLogger
- Runtime: Python 3.12
- Execution Role: Select "Create a new role with basic Lambda permissions."



### 6.3 Upload the function code from EC2 using the AWS CLI:

```
aws lambda update-function-code --function-name S3TextFileLogger --zip-file fileb://function.zip
```

```
[ec2-user@ip-172-31-33-47 ~]$ aws lambda update-function-code --function-name S3TextFileLogger --zip-file fileb://function.zip
{
  "FunctionName": "S3TextFileLogger",
  "FunctionArn": "arn:aws:lambda:eu-north-1:860015268757:function:S3TextFileLogger",
  "Runtime": "python3.12",
  "Role": "arn:aws:iam::860015268757:role/service-role/S3TextFileLogger-role-l6qnx3qp",
  "Handler": "lambda_function.lambda_handler",
  "CodeSize": 524,
  "Description": "",
  "Timeout": 3,
  "MemorySize": 128,
  "LastModified": "2024-10-20T12:13:22.000+0000",
  "CodeSha256": "r3bhnxP0TGjIFMX9rKsiULVb+470gIq4Fn8ufxx4Cuc=",
  "Version": "$LATEST",
  "TracingConfig": {
    "Mode": "PassThrough"
  },
  "RevisionId": "b5c0eee4-ec69-482f-9836-35d63c7752e8",
  "State": "Active",
  "LastUpdateStatus": "InProgress",
  "LastUpdateStatusReason": "The function is being created.",
  "LastUpdateStatusReasonCode": "Creating",
  "PackageType": "Zip",
  "Architectures": [
    "x86_64"
  ],
  "EphemeralStorage": {
    "Size": 512
  },
  "SnapStart": {
    "ApplyOn": "None",
    "OptimizationStatus": "Off"
  },
  "RuntimeVersionConfig": {
    "RuntimeVersionArn": "arn:aws:lambda:eu-north-1::runtime:188d9ca2e2714ff5637bd2bbe06ceb81ec3bc408a0f277dab104c14cd814b081"
  },
  "LoggingConfig": {
    "LogFormat": "Text",
    "LogGroup": "/aws/Lambda/S3TextFileLogger"
  }
}
```

## 7. Configure S3 as the Trigger


7.1 In Lambda console, go to the Function Overview section and click Add Trigger.

7.2 Choose S3 as the trigger:

- Select your bucket (lambda-s3-trigger-bucket).
- Event type: Choose All object create events.

### 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.  
 × ↺  
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 ×

## 8. Upload a File and Test

Upload a text file to your S3 bucket:

- Go to S3 > your bucket > Upload.
- Upload a .txt file with some content (e.g., hello.txt)

**Files and folders** (1 Total, 41.0 B)  
All files and folders in this table will be uploaded.

Remove Add files Add folder

< 1 >

<input type="checkbox"/>	Name	Folder
<input type="checkbox"/>	hello.txt	-

The Lambda function will automatically run when the file is uploaded.

## 9. Check Logs in CloudWatch

9.1 In the AWS Console, go to CloudWatch > Logs.

9.2 Under Log Groups, find the log group for your Lambda function (/aws/lambda/S3TextFileLogger).

9.3 Open the latest log stream to see the file content logged by the Lambda function.

Log events

Actions

Start tailing

Create metric filter

You can use the filter bar below to search for and match terms, phrases, or values in your log events. [Learn more about filter patterns](#)

Q

Filter events - press enter to search

Clear

1m

30m

1h

12h

Custom

UTC timezone

Display

▶

Timestamp

Message

No older events at this moment. [Retry](#)

▶

2024-10-20T13:23:45.499Z

INIT\_START Runtime Version: python:3.12.v36 Runtime Version ARN: arn:aws:lambda:eu-north-1::runtime:188d9ca2e2714ff5637bd2bbe06ceb81ec3bc408a0f277dab104c14cd814b081

▶

2024-10-20T13:23:45.993Z

START RequestId: 3f9d75cd-5faf-4284-a301-2c8f843fad2c Version: \$LATEST

▶

2024-10-20T13:23:46.557Z

File Content from hello.txt:

▶

2024-10-20T13:23:46.557Z

Hello from Dev Gaonkar, Rollno-12, D15C.

▶

2024-10-20T13:23:46.578Z

END RequestId: 3f9d75cd-5faf-4284-a301-2c8f843fad2c

▶

2024-10-20T13:23:46.579Z

REPORT RequestId: 3f9d75cd-5faf-4284-a301-2c8f843fad2c Duration: 585.22 ms Billed Duration: 586 ms Memory Size: 128 MB Max Memory Used: 83 MB Init Duration: 491.24 ms

No newer events at this moment. Auto retry paused. [Resume](#)

2024-10-20T13:23:46.557Z

File Content from hello.txt:

2024-10-20T13:23:46.557Z

Hello from Dev Gaonkar, Rollno-12, D15C.

### Conclusion:

In this experiment, I gained a deeper understanding of serverless architecture by working with AWS Lambda and S3. I learned how to automate processes like file logging and real-time monitoring without managing servers. Setting up Lambda triggers for S3 uploads demonstrated the efficiency and scalability of cloud solutions. I also found EC2 to be a suitable alternative for development after the switch from Cloud9. This hands-on experience highlighted the simplicity and cost-effectiveness of integrating AWS services for real-time applications.