Case Study: 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.

Case Study Overview:

In this case study, we explore the implementation of a serverless logging solution using AWS Lambda and Amazon S3. The primary goal is to create a Lambda function that triggers automatically when a text file is uploaded to an S3 bucket. This Lambda function reads the content of the uploaded file and logs it for further processing or monitoring. Instead of Cloud9 EC2 is used for the development purpose.

Key Feature and Application

The unique feature of this case study is the seamless integration between AWS Lambda and Amazon S3, allowing for real-time processing of data as soon as it becomes available. This capability enables automatic logging and monitoring of uploaded files without manual intervention. Applications can be followed as:

- **Data Ingestion and Monitoring**: Automatically logging and processing data files (e.g., CSVs, logs, configuration files) as they are uploaded to S3.
- **File Content Validation**: Implementing checks on file contents upon upload to ensure they meet certain criteria before further processing.
- **Event-Driven Workflows**: Triggering additional workflows or notifications based on file uploads, such as sending alerts to stakeholders or updating dashboards.

Third-Year Project Integration

In my third-year project, **Vaccination Track**, which focuses on tracking vaccination data and appointments, there is a significant relationship with the concepts explored in this case study. The Vaccination Track project involves handling parent and child data, including vaccination records and schedules, which can be efficiently managed using a serverless architecture.

For example, when a new child data is uploaded to an S3 bucket, a Lambda function can automatically check the information and update the database or send notifications to the right people. This approach can also help me set up reminders for parents when their vaccination appointments are coming up, ensuring they don't miss any important dates. Additionally, by adding a logging feature, I can keep track of any changes made to vaccination records, creating an audit trail that is important for meeting compliance standards and maintaining accurate records.

STEPS By Step Explanation:

1. Go to the AWS and launch an EC2 instance and choose the following parameters

- AMI: Choose Amazon Linux 2.
- Instance Type: Select t2.micro (eligible for free tier).
- 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): Optional, allows browser access.
            - HTTPS (TCP port 443): Optional, for secure traffic.
        - Outbound Rules:
            - Allow all outbound traffic (default).

Name and tags Info

Name

serverless

Add additional tags

Application and OS Images (Amazon Machine Image) Info

An AMI is a template that contains the software configuration (operating system, application server, and applications) required to launch your instance. Search or Browse for AMIs if you don't see what you are looking for below

Search our full catalog including 1000s of application and OS images

Quick Start

Amazon Linux | macOS | Ubuntu | Windows | Red Hat | SUSE Li

Browse more AMIs

Including AMIs from AWS, Marketplace and the Community

Amazon Machine Image (AMI)

Amazon Linux 2 AMI (HVM) - Kernel 5.10, SSD Volume Type — Free tier eligible

ami-0ddc798b3f1a5117e (64-bit (x86)) / ami-05f16f3539e999b77 (64-bit (Arm))

Virtualization: hvm ENA enabled: true Root device type: ebs

Key pair name - *required*

casestudy

Create new key pair

Network settings Info

Edit

Network Info

vpc-08b20e70c636d4d1f

Subnet Info

No preference (Default subnet in any availability zone)

Auto-assign public IP Info

Enable

Additional charges apply when outside of free tier allowance

Firewall (security groups) Info

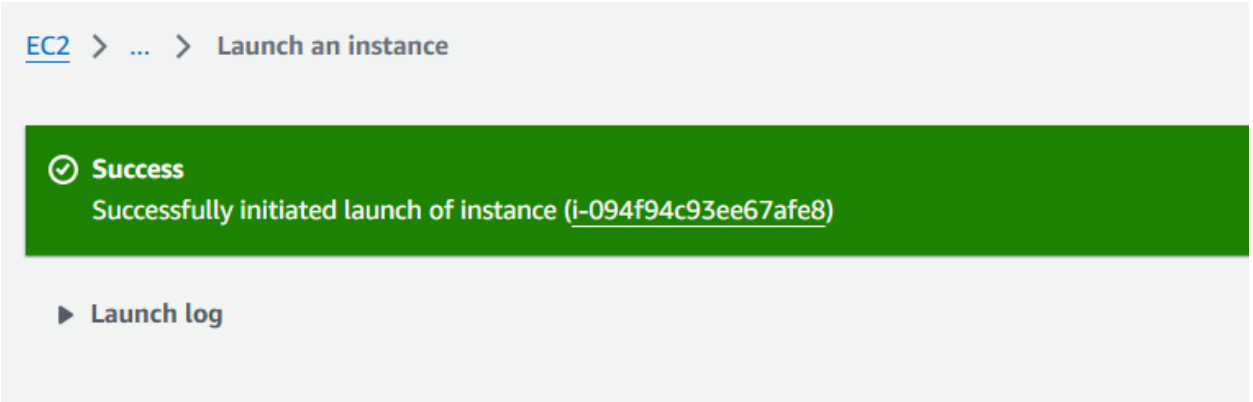
A security group is a set of firewall rules that control the traffic for your instance. Add rules to allow specific traffic to reach your instance.

- (●) Create security group
- ( ) Select existing security group

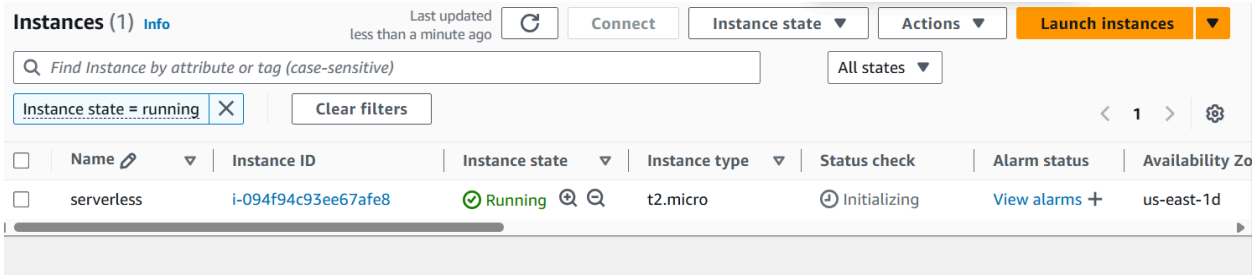
We'll create a new security group called '**launch-wizard-1**' with the following rules:

- [x] Allow SSH traffic from — Anywhere 0.0.0.0/0
  Helps you connect to your instance
- [x] Allow HTTPS traffic from the internet
  To set up an endpoint, for example when creating a web server
- [x] Allow HTTP traffic from the internet
  To set up an endpoint, for example when creating a web server

Click on "Launch Instance".



2. Launch the instance and wait for it to be ready.



3. Connect to the EC2 instance via SSH:

ssh -i <your-key.pem> ec2-user@<your-ec2-public-dns>
(To launch through your terminal first go to the folder where your key is downloaded and then run this command)

EC2 Instance Connect | Session Manager | SSH client | EC2 serial console

Instance ID

i-094f94c93ee67afe8 (serverless)

1. Open an SSH client.
2. Locate your private key file. The key used to launch this instance is casestudy.pem
3. Run this command, if necessary, to ensure your key is not publicly viewable.
   chmod 400 "casestudy.pem"
4. Connect to your instance using its Public DNS:
   ec2-44-202-123-227.compute-1.amazonaws.com

Example:

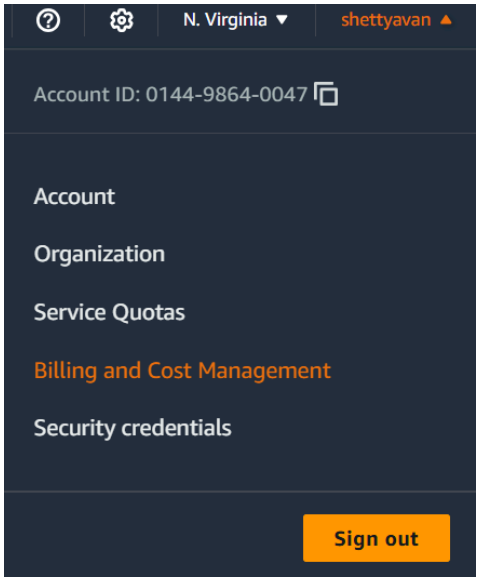
ssh -i "casestudy.pem" ec2-user@ec2-44-202-123-227.compute-1.amazonaws.com

Note: In most cases, the guessed username is correct. However, read your AMI usage instructions to check if the AMI owner has changed the default AMI username.

```
C:\Users\Avan>cd Downloads

C:\Users\Avan\Downloads>ssh -i "casestudy.pem" ec2-user@ec2-44-202-123-227.compute-1.amazonaws.com
The authenticity of host 'ec2-44-202-123-227.compute-1.amazonaws.com (44.202.123.227)' can't be established.
ED25519 key fingerprint is SHA256:uOB0CLYfPbQ+s9G32ODfaNkkNtop0qnrsL2NPgrxKO8.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'ec2-44-202-123-227.compute-1.amazonaws.com' (ED25519) to the list of known hosts.
       #_
   ~\_  ####_        Amazon Linux 2023
  ~~  \_#####\
  ~~     \###|
  ~~       \#/ ___   https://aws.amazon.com/linux/amazon-linux-2023
   ~~       V~' '->
    ~~~         /
      ~~._.   _/
         _/ _/
       _/m/'
[ec2-user@ip-172-31-85-199 ~]$
```

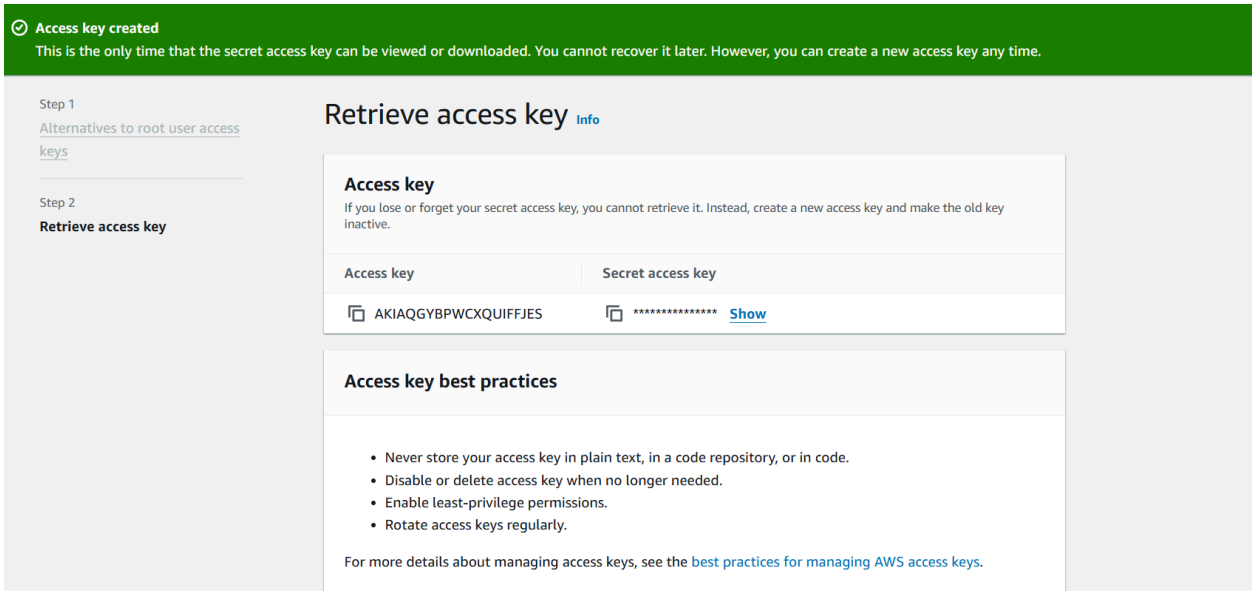
4. Create Access keys for Root user

   a. **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.



b. **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.

   Save these credentials somewhere so that you can access them afterwards



5. Install AWS CLI and Configure EC2

a. Update packages and install AWS CLI:

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

```
[ec2-user@ip-172-31-85-199 ~]$
sudo yum update -y
sudo yum install aws-cli -y
Last metadata expiration check: 0:07:58 ago on Tue Oct 22 14:31:43 2024.
Dependencies resolved.
Nothing to do.
Complete!
Last metadata expiration check: 0:07:59 ago on Tue Oct 22 14:31:43 2024.
Package awscli-2-2.15.30-1.amzn2023.0.1.noarch is already installed.
Dependencies resolved.
Nothing to do.
Complete!
[ec2-user@ip-172-31-85-199 ~]$
```

b. Configure AWS CLI:

aws configure

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

```
[ec2-user@ip-172-31-85-199 ~]$ aws configure
AWS Access Key ID [None]: AKIAQGYBPWCXQUIFFJES
AWS Secret Access Key [None]: [redacted]
Default region name [None]: us-east-1
Default output format [None]: json
[ec2-user@ip-172-31-85-199 ~]$
```

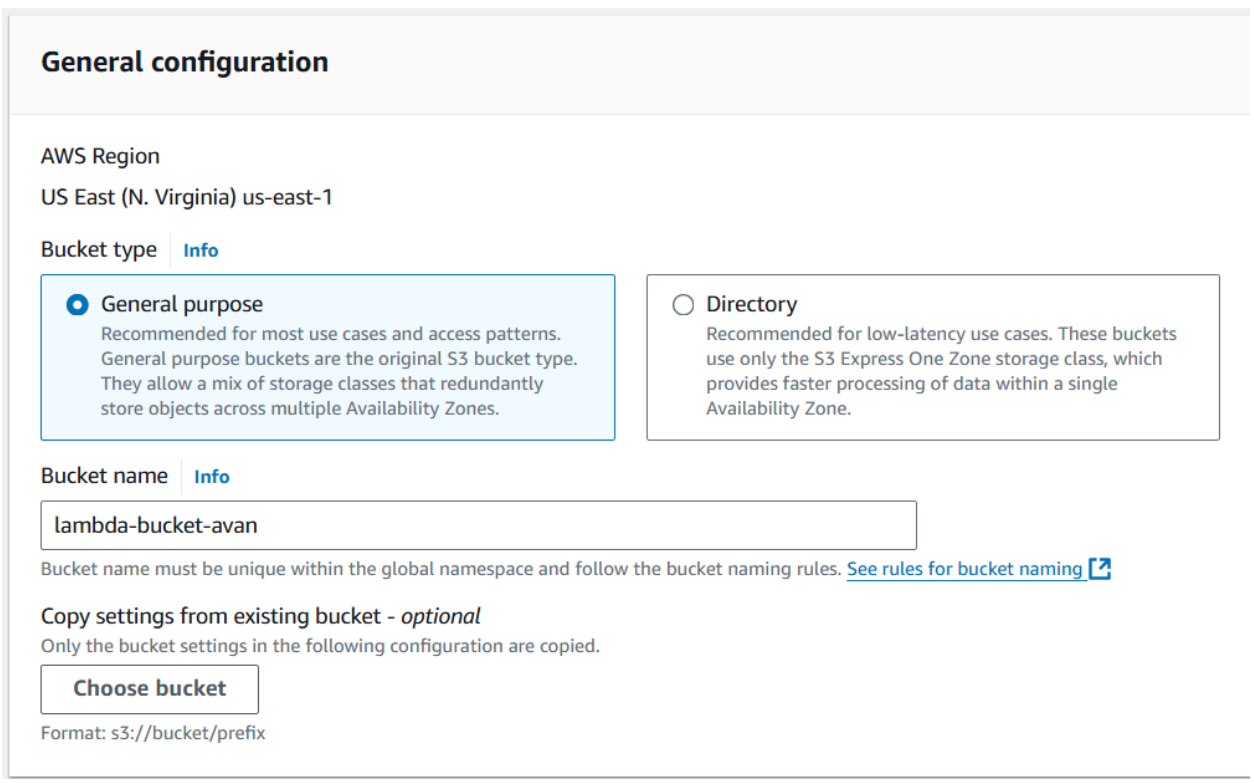
c. **Install Python and pip** (since Lambda uses Python):

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

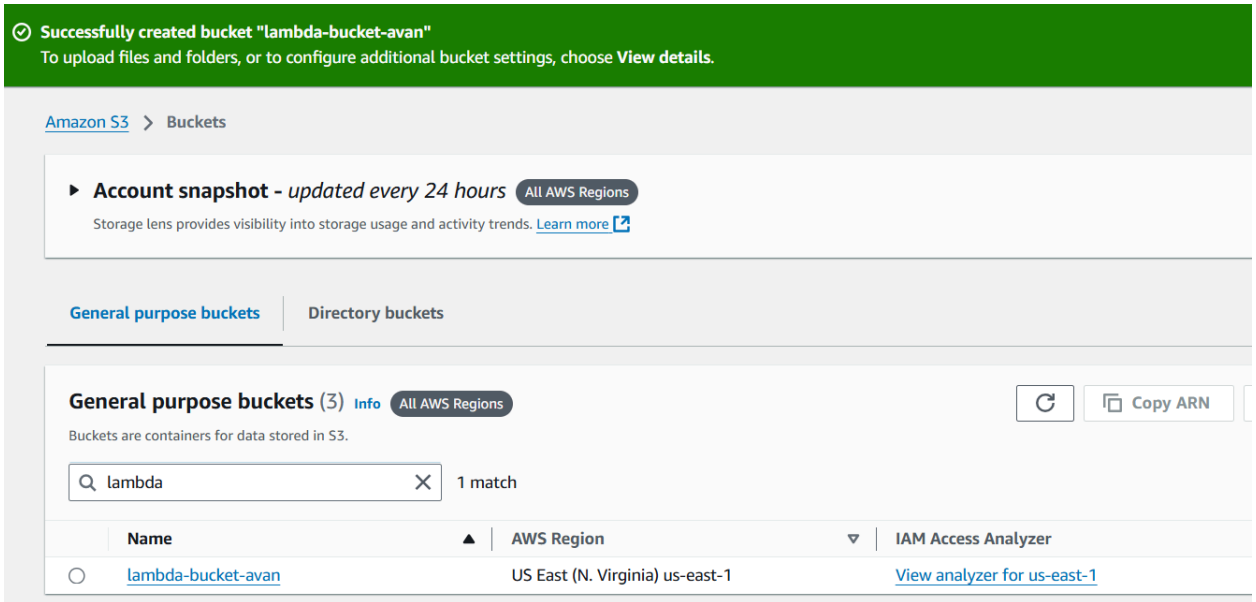
```
[ec2-user@ip-172-31-85-199 ~]$ sudo yum install python3 -y
sudo yum install python3-pip -y
Last metadata expiration check: 0:15:03 ago on Tue Oct 22 14:31:43 2024.
Package python3-3.9.16-1.amzn2023.0.9.x86_64 is already installed.
Dependencies resolved.
Nothing to do.
Complete!
Last metadata expiration check: 0:15:04 ago on Tue Oct 22 14:31:43 2024.
Dependencies resolved.
================================================================================================================
 Package                        Architecture        Version                                Repository                Size
================================================================================================================
Installing:
 python3-pip                    noarch              21.3.1-2.amzn2023.0.8                  amazonlinux              1.8 M
Installing weak dependencies:
 libxcrypt-compat               x86_64              4.4.33-7.amzn2023                      amazonlinux               92 k

Transaction Summary
================================================================================================================
Install  2 Packages

Total download size: 1.9 M
Installed size: 11 M
Downloading Packages:
(1/2): libxcrypt-compat-4.4.33-7.amzn2023.x86_64.rpm                                  1.8 MB/s |  92 kB     00:00
(2/2): python3-pip-21.3.1-2.amzn2023.0.8.noarch.rpm                                    22 MB/s | 1.8 MB     00:00
----------------------------------------------------------------------------------------------------------------
Total                                                                                  15 MB/s | 1.9 MB     00:00
Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
Running transaction
  Preparing        :                                                                                            1/1
  Installing       : libxcrypt-compat-4.4.33-7.amzn2023.x86_64                                                  1/2
  Installing       : python3-pip-21.3.1-2.amzn2023.0.8.noarch                                                   2/2
  Running scriptlet: python3-pip-21.3.1-2.amzn2023.0.8.noarch                                                   2/2
  Verifying        : libxcrypt-compat-4.4.33-7.amzn2023.x86_64                                                  1/2
  Verifying        : python3-pip-21.3.1-2.amzn2023.0.8.noarch                                                   2/2

Installed:
  libxcrypt-compat-4.4.33-7.amzn2023.x86_64                  python3-pip-21.3.1-2.amzn2023.0.8.noarch
```

6. Create and S3 Bucket

a. In the AWS Management Console, go to **S3** and Create bucket:



Once created you will be notified with the message “launched successfully”



7. Create the **Lambda Function** code.

a. On your EC2 instance, create the Python file for the Lambda function:

nano lambda_code.py

```
[ec2-user@ip-172-31-85-199 ~]$ nano lambda_code.py
```

b. **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')
    }
```

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

```
  GNU nano 5.8                                          lambda_code.py
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')
    }
```

8. Deploy the Lambda function from EC2

- ➢ AWS Lambda expects a specific file where your Lambda function is defined. By default, it looks for a file named lambda_function.py unless otherwise specified in the Handler configuration.
- ➢ Update the Handler configuration to point to the correct file and function name. For example, if your file is named **lambda_code.py** and the handler function inside is handler, your Handler in the AWS Lambda configuration should be **lambda_code.handler**
- ➢ It can be done even in the terminal :
  *aws lambda update-function-configuration *

*--function-name file-logger-S3 *
--handler lambda_code.lambda_handler

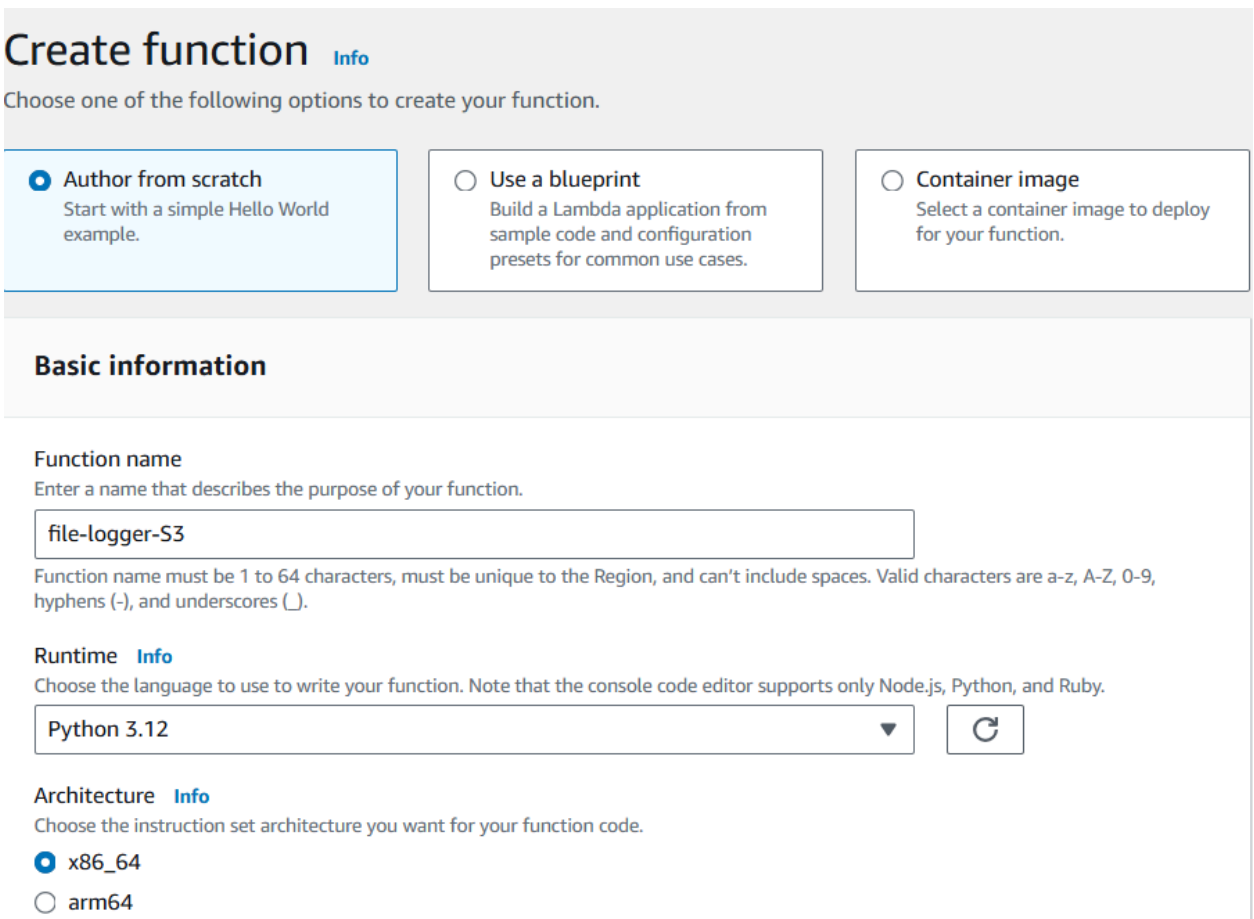
a. Package the Lambda function:

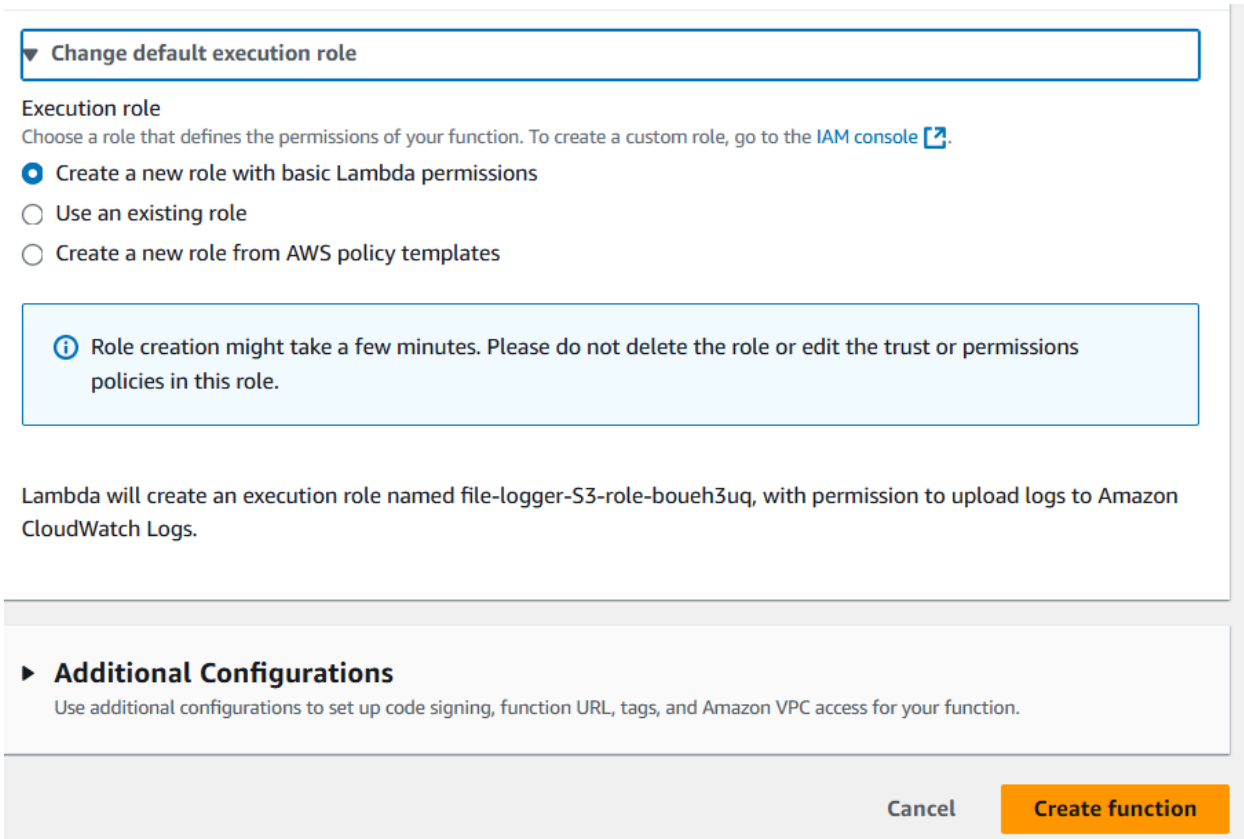
zip function.zip lambda_code.py

```
[ec2-user@ip-172-31-85-199 ~]$ zip function.zip lambda_code.py
  adding: lambda_code.py (deflated 47%)
```

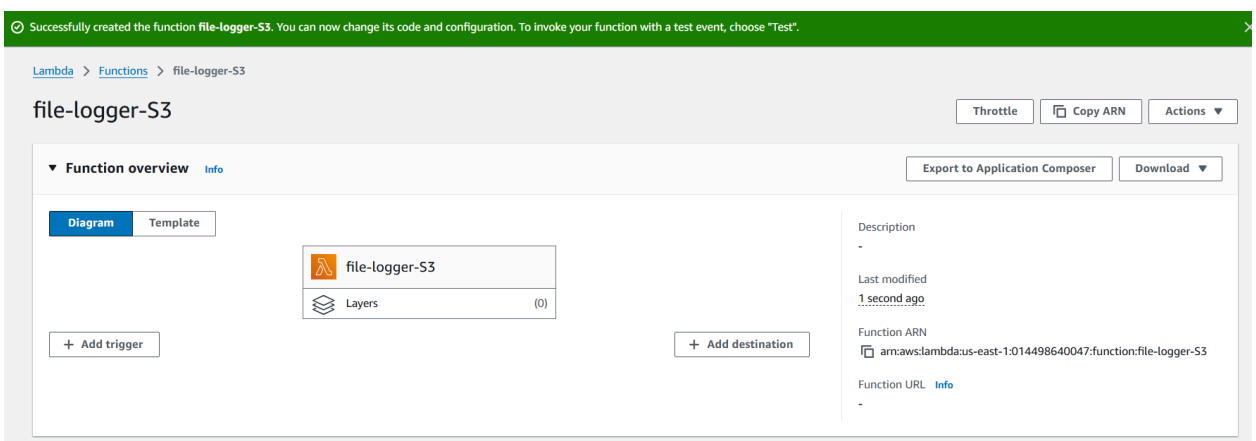
b. **Create a Lambda function in AWS Console**:

- Go to **AWS Lambda** and then **Create Function** and choose the following settings.
- ➢ Choose **Author from Scratch**:
- ➢ **Function Name**: file-logger-S3
- ➢ **Runtime**: Python 3.12
- ➢ **Execution Role**: Select "Create a new role with basic Lambda permissions."





➢ Once you create function a success message is displayed



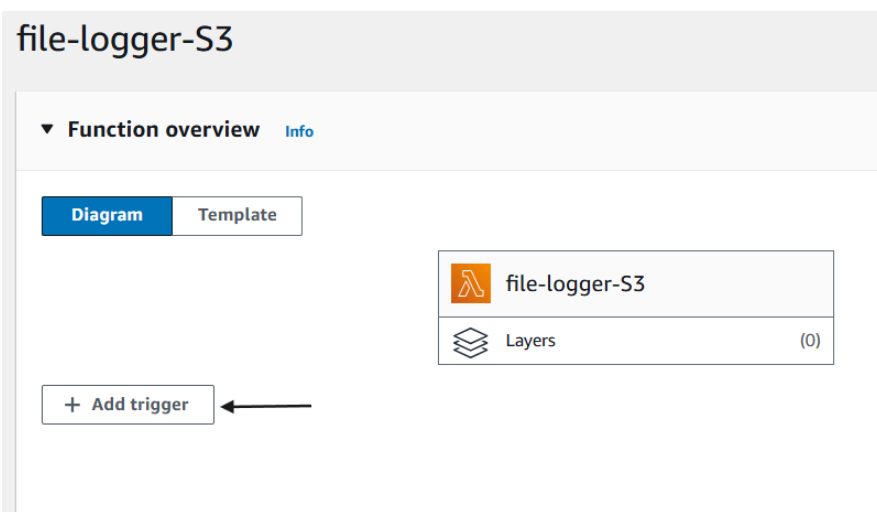
c. Upload the function code from EC2 using the AWS CLI:

aws lambda update-function-code --function-name file-logger-S3 --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+47OgIq4Fn8ufxx4Cuc=",
    "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"
    }
```

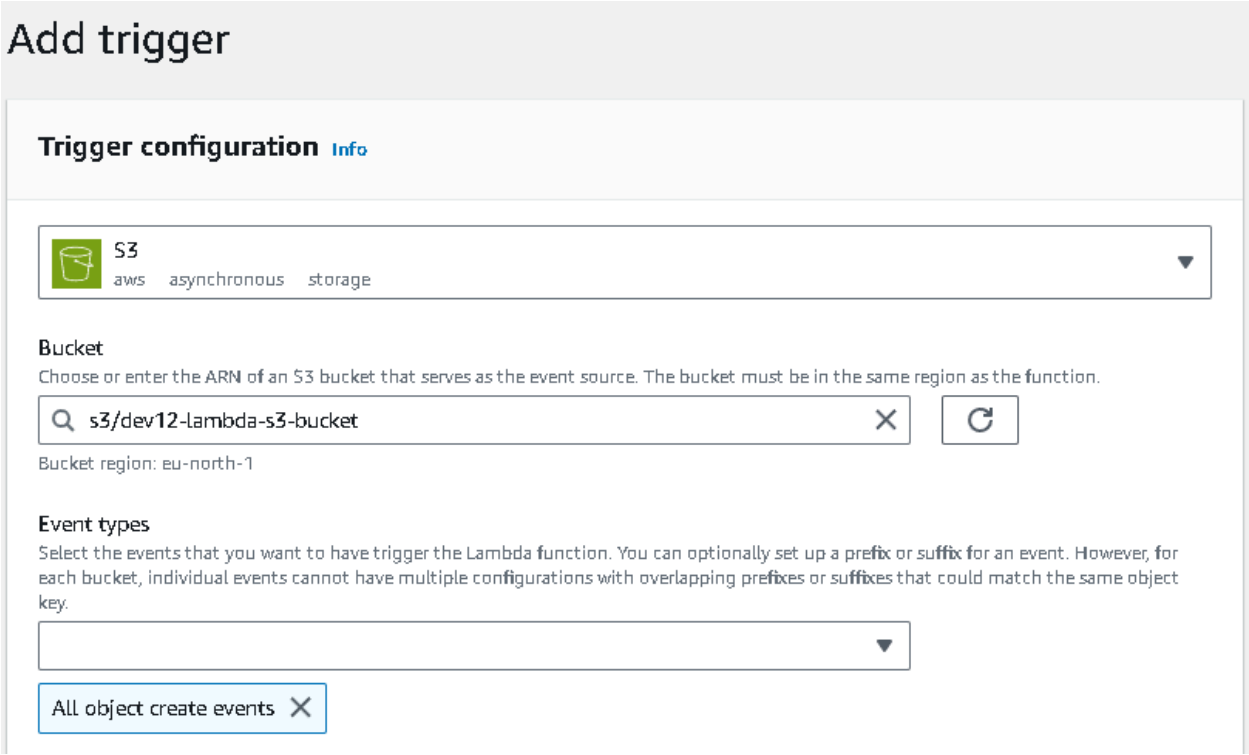
9. Configure S3 as the Trigger

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

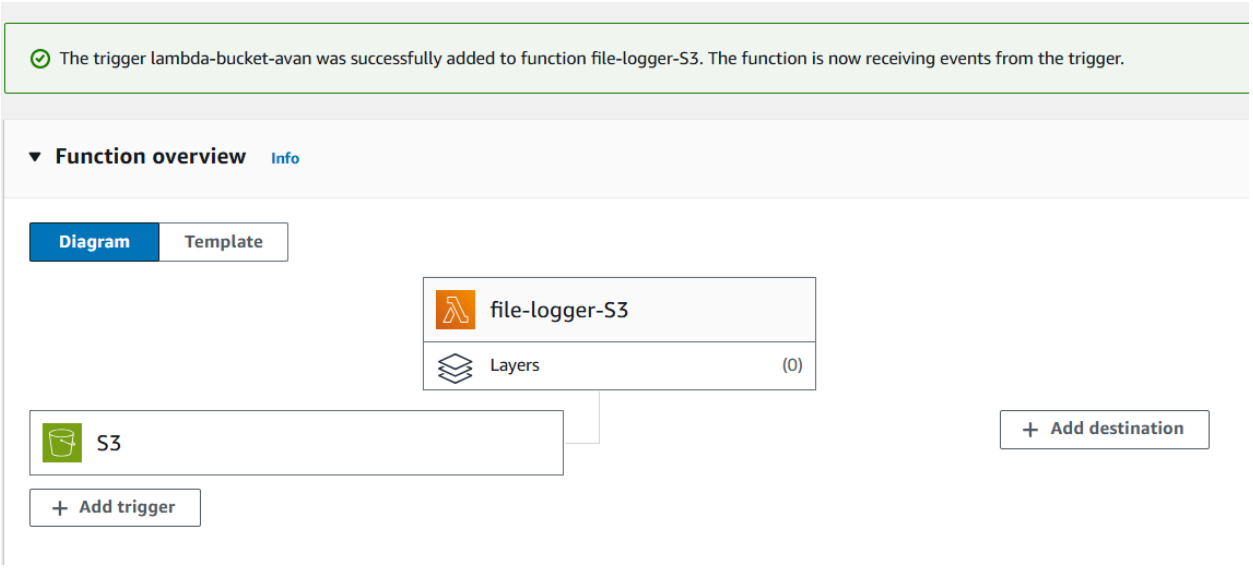


b. Choose **S3** as the trigger:

- ➢ Select your bucket (lambda-bucket-avan).
- ➢ **Event type**: Choose All object create events.



c. Click **Add** to enable the trigger.



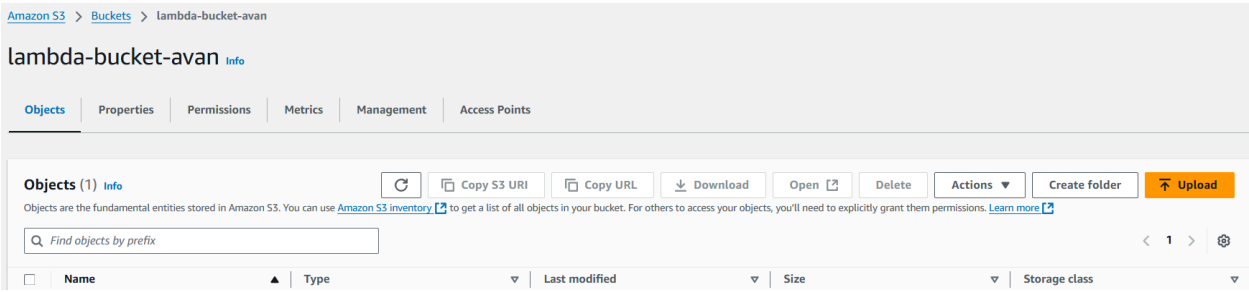
10. Upload a File and Test

a. Create a text/csv/json file in your local host with some content.

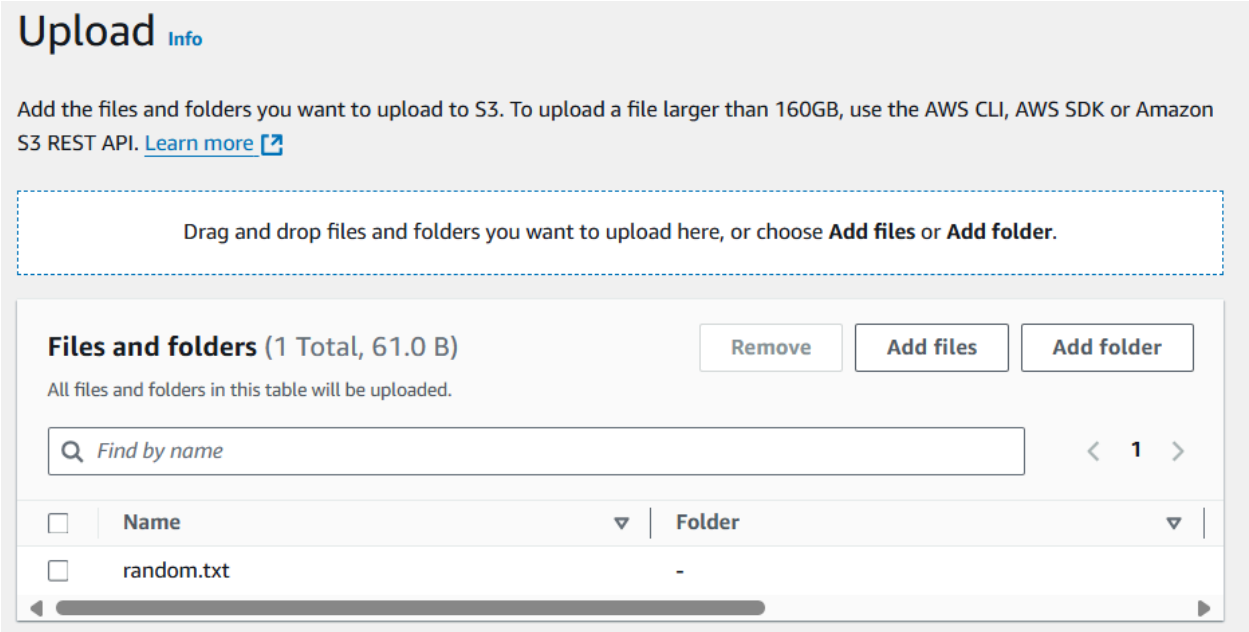
```
PS C:\Users\Avan\Desktop> cat random.txt
Advance Devops Case Study 12 performed by Avan Shetty D15C_52
PS C:\Users\Avan\Desktop>
```

b. **Upload a text file** to your S3 bucket:

➢ Go to **S3** - > bucket - > upload



➢ Uploading a .txt file with some content (eg: random.txt)

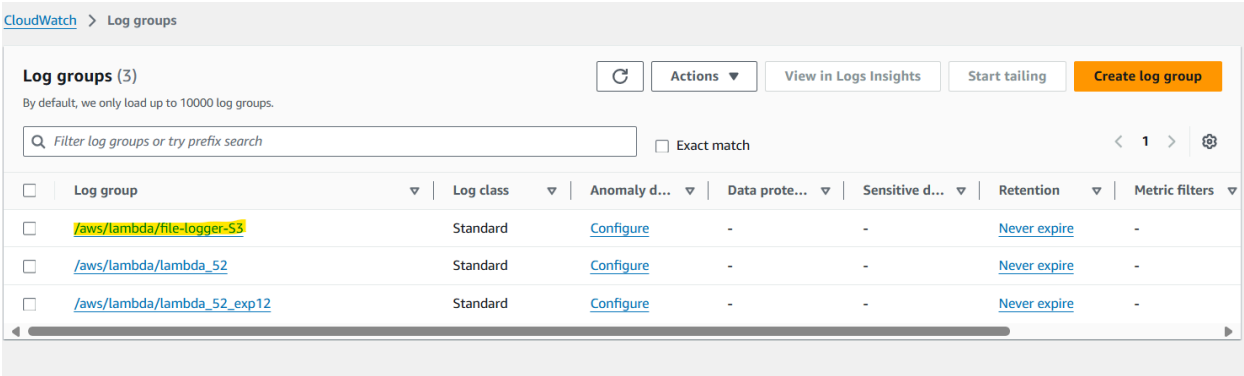


The Lambda function will automatically run when the file is uploaded.

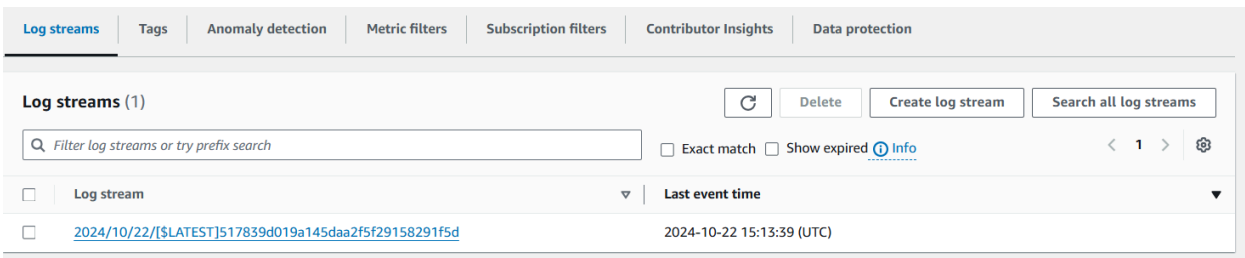
11. Check Logs in **CloudWatch**

a. In the AWS Console, go to **CloudWatch** > **Logs**.

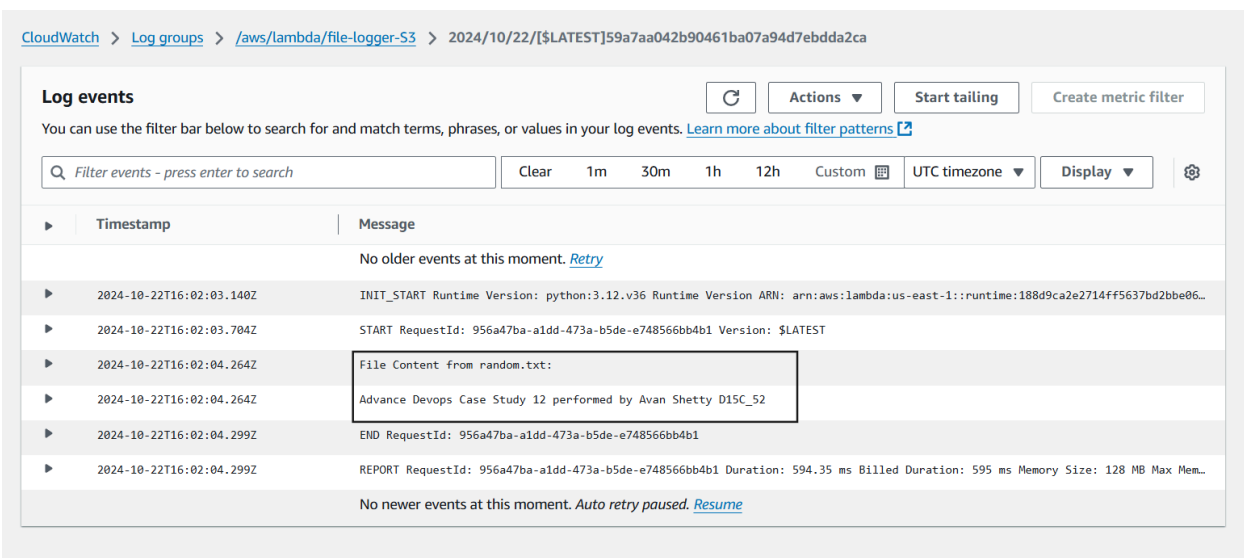
b. Under **Log Groups**, find the log group for your Lambda function (**/aws/lambda/file-logger-S3**).



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



d. From this you are able to monitor and log the output of your Lambda function and view the contents of the text file



In this way the logs for the file from S3 bucket are viewed using the EC2 and the Lambda function

Conclusion:
This case study shows how using AWS Lambda and S3 can simplify logging and processing data automatically when files are uploaded. By setting up a Lambda function to run whenever a new file is added to an S3 bucket, I can easily keep track of different files and monitor the logs of that uploaded file. During this process, I ran into an error in the Lambda handler because the input event wasn't set up correctly, which taught me how important it is to structure the data properly. Plus, this setup can handle different types of files, making it really versatile for various uses.This in all improved my understanding of serverless architectures and how they can streamline data management tasks.