Aryan Dangat D15A

# 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.

## Introduction:

- In today's cloud-centric world, organizations are increasingly leveraging serverless architectures to streamline their operations and enhance efficiency. This case study explores the implementation of a serverless logging system using AWS Lambda and Amazon S3. The objective is to automate the logging process by creating a Lambda function that is triggered whenever a text file is uploaded to an S3 bucket.
- By utilizing AWS Lambda, a fully managed serverless computing service, developers can run code in response to events without the need for provisioning or managing servers. This architecture allows for seamless scaling and reduced operational costs, making it an ideal solution for handling sporadic workloads. The integration with Amazon S3 enables users to store and retrieve data effortlessly, while AWS Cloud9 provides a powerful environment for developing and deploying Lambda functions.
- The core problem addressed in this study is to set up a Lambda function that automatically logs the contents of uploaded text files, thus enhancing data handling capabilities. The tasks involved include creating a Lambda function using Python, configuring the S3 bucket as the event trigger, and verifying the logging functionality by uploading a sample text file. This implementation not only improves efficiency but

also demonstrates the practical benefits of serverless computing in modern application development.

## Key Features

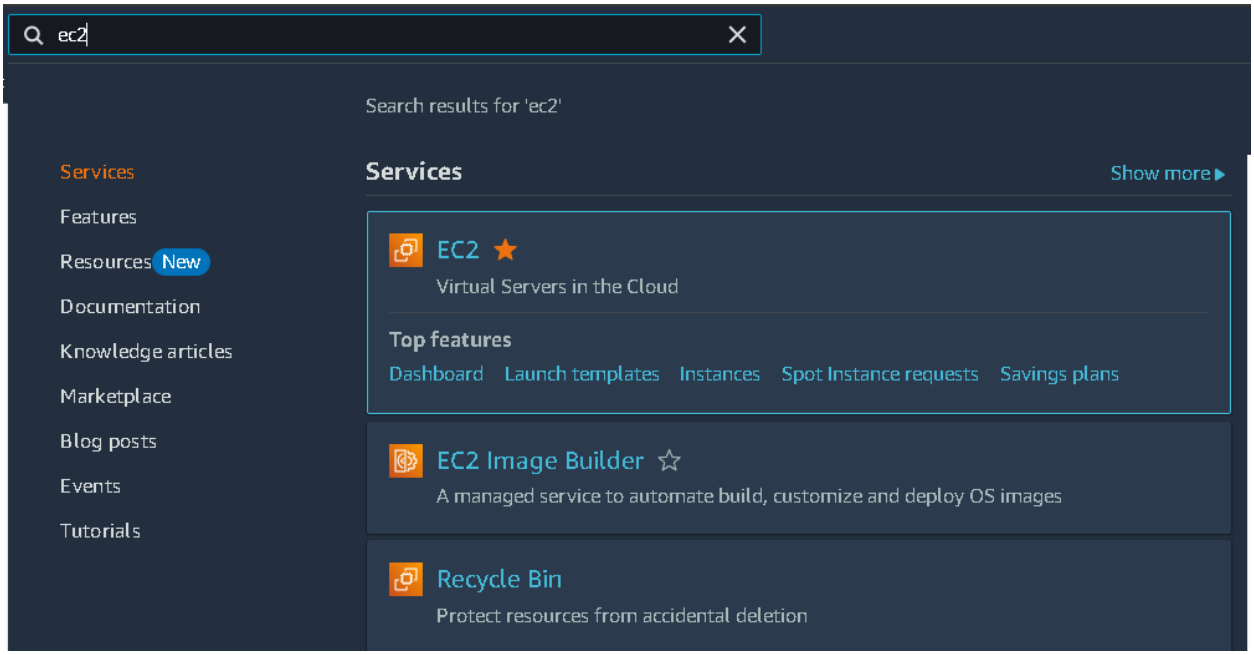
- **Event-Driven Architecture**: The system leverages an event-driven approach where the AWS Lambda function is automatically triggered by events, such as the uploading of a text file to an S3 bucket. This eliminates the need for manual intervention, ensuring a seamless workflow.
- **Serverless Computing**: By using AWS Lambda, the solution operates without the need to provision or manage server infrastructure. This allows developers to focus on writing code rather than worrying about server management, leading to increased productivity and reduced operational costs.
- **Scalability**: The serverless architecture automatically scales based on the number of file uploads, handling variable workloads efficiently. This means that the system can accommodate bursts of activity without requiring pre-planned scaling strategies.
- **Real-Time Logging**: As soon as a file is uploaded to the S3 bucket, the Lambda function reads its contents and logs the information in real-time. This immediate logging capability enhances the responsiveness of applications that depend on timely data processing.
- **Integration with AWS Services**: The solution can be easily integrated with other AWS services, such as Amazon CloudWatch for monitoring and alerting, Amazon SNS for notifications, and Amazon DynamoDB for data storage. This allows for the creation of a robust data processing pipeline.
- **Cost Efficiency**: The pay-as-you-go pricing model of AWS Lambda ensures that users only pay for the compute time used during the execution of the logging function. This model is particularly advantageous for applications with sporadic workloads, as it minimizes idle resource costs.
- **Enhanced Monitoring and Troubleshooting**: With integration into Amazon CloudWatch, users can monitor the performance of the Lambda function and track logs generated during execution. This makes it easier to troubleshoot issues and maintain operational visibility.

Aryan Dangat D15A

**STEPS:**

1. Launch an EC2 Instance

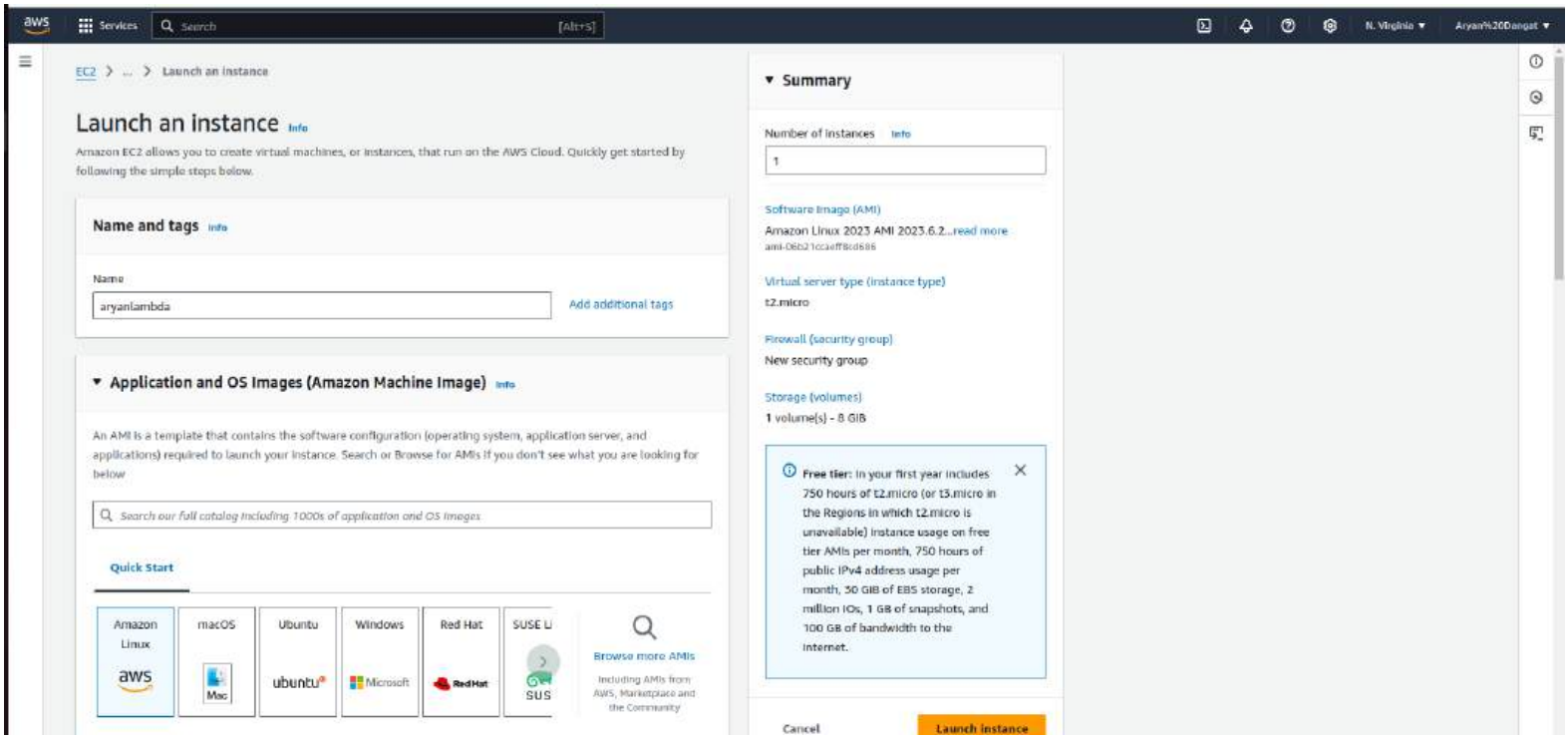
1.1 Login to AWS Console and go to EC2 service.

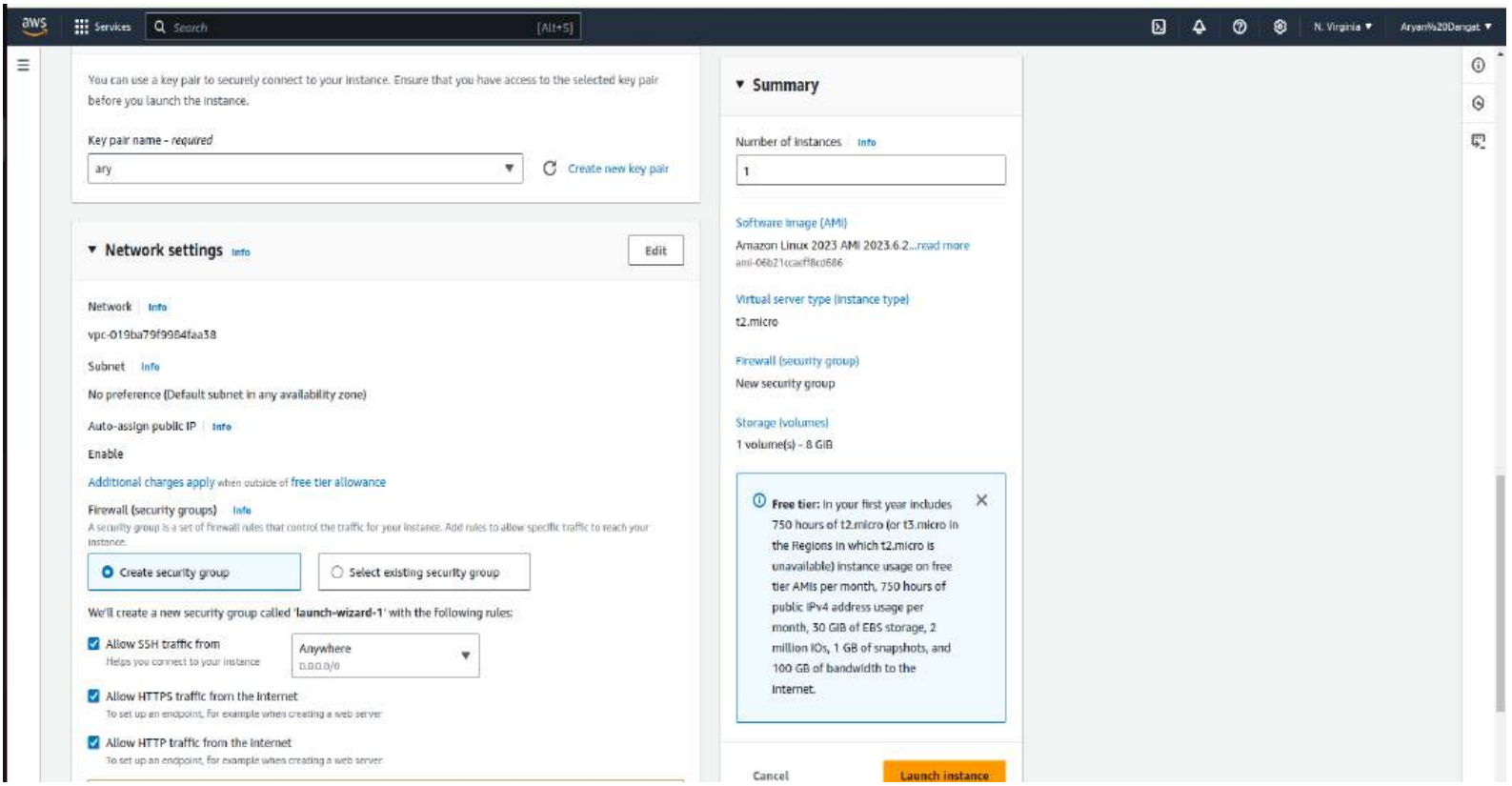



1.2 Click on "Launch Instance".

- 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).

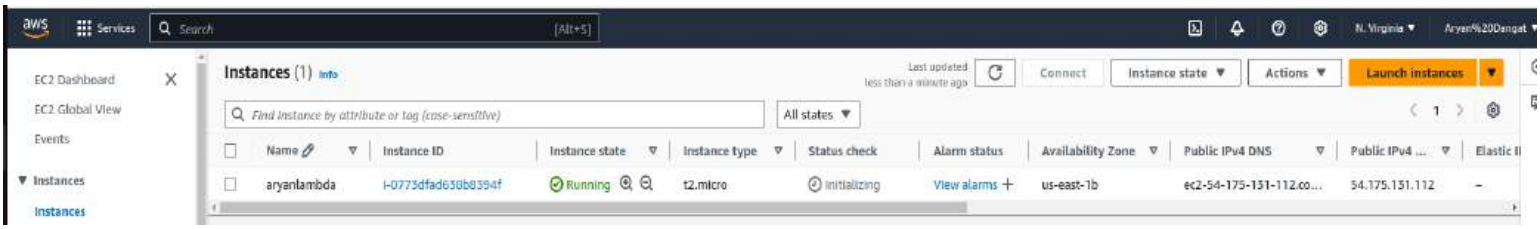
Aryan Dangat D15A



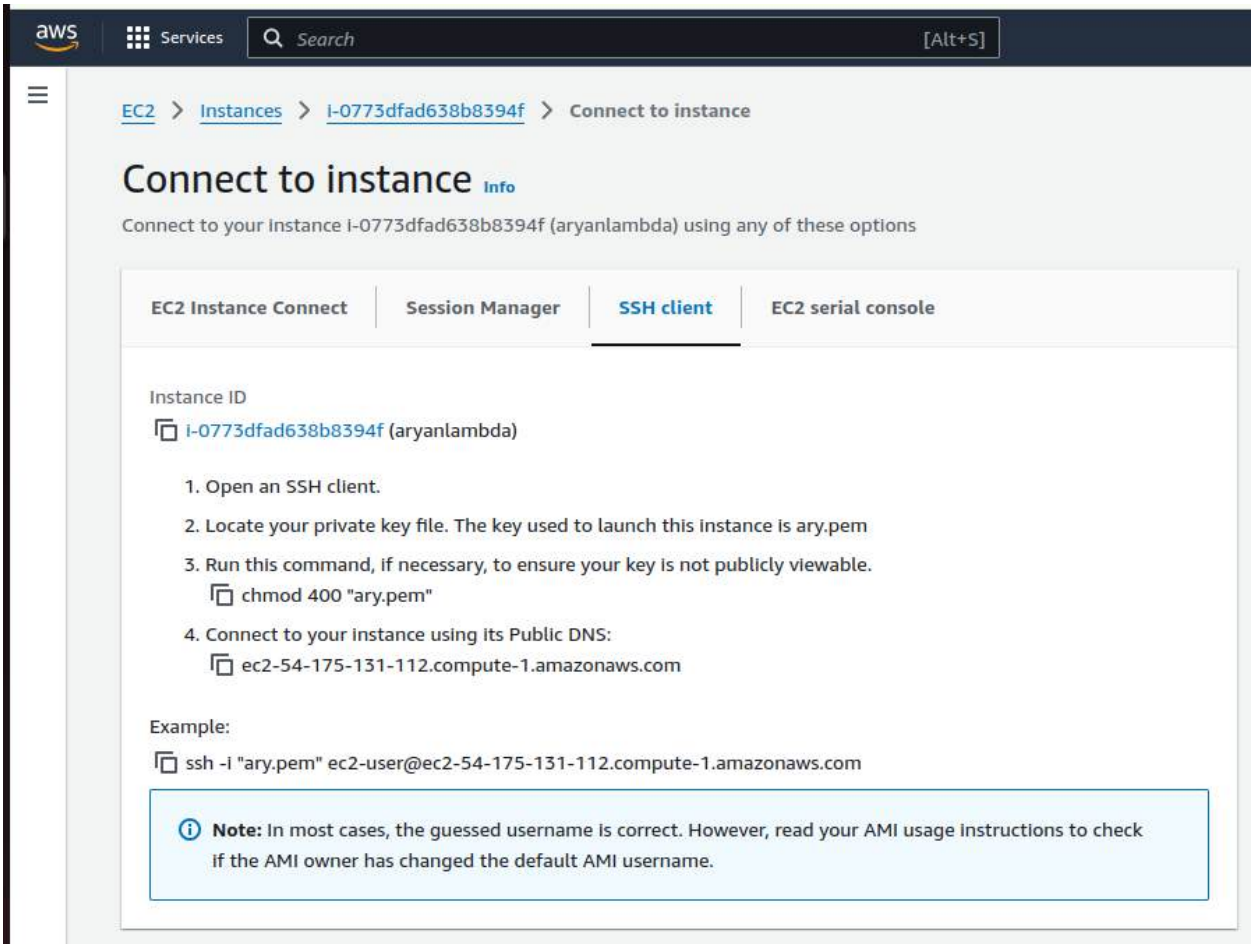
Aryan Dangat D15A

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

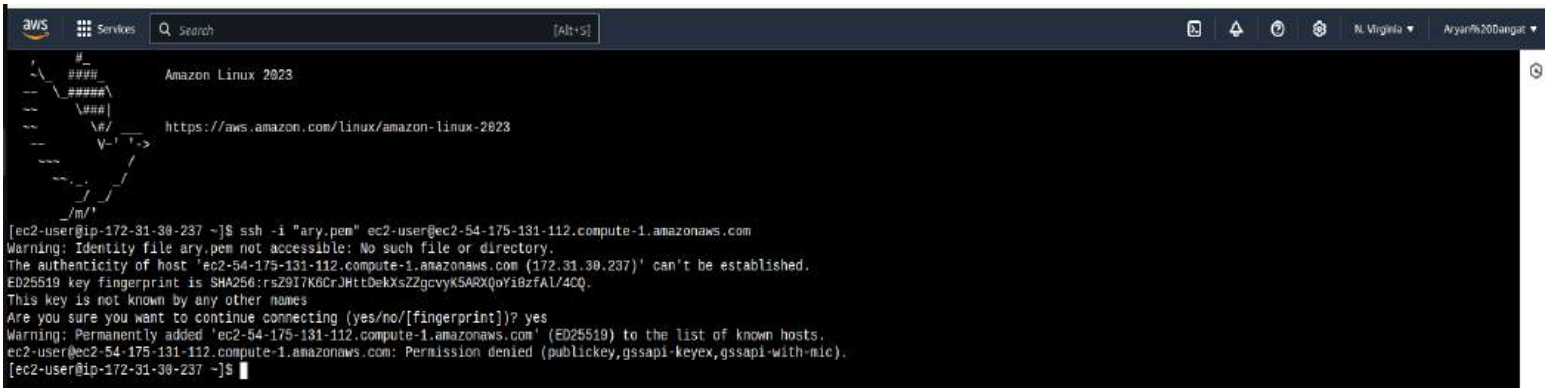



1.4 Connect to the EC2 instance via SSH:

ssh -i <your-key.pem> ec2-user@<your-ec2-public-dns>



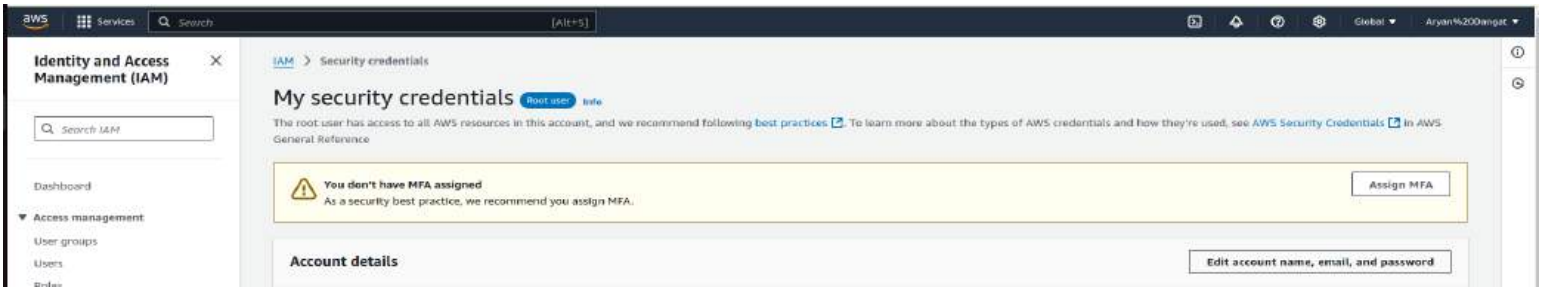
Aryan Dangat D15A




2. Create Access keys for Root user

2.1 **Access the Root User Security Credentials**:

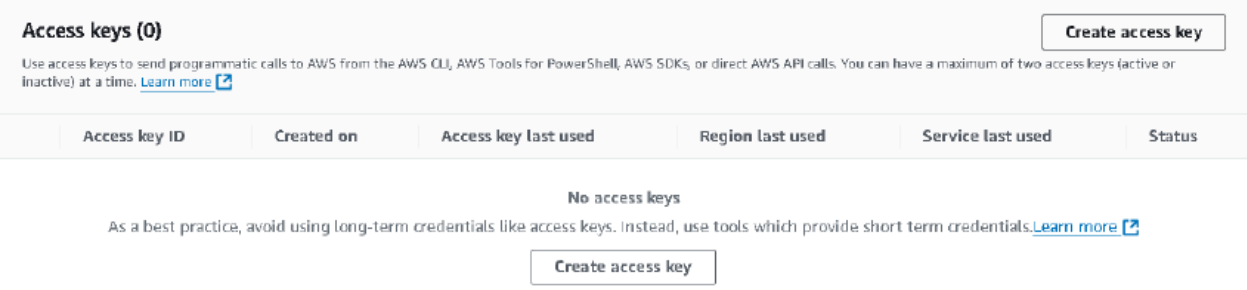
- In the top-right corner of AWS Management Console, click on your account name or email address, and then click **Security Credentials** from the dropdown menu.

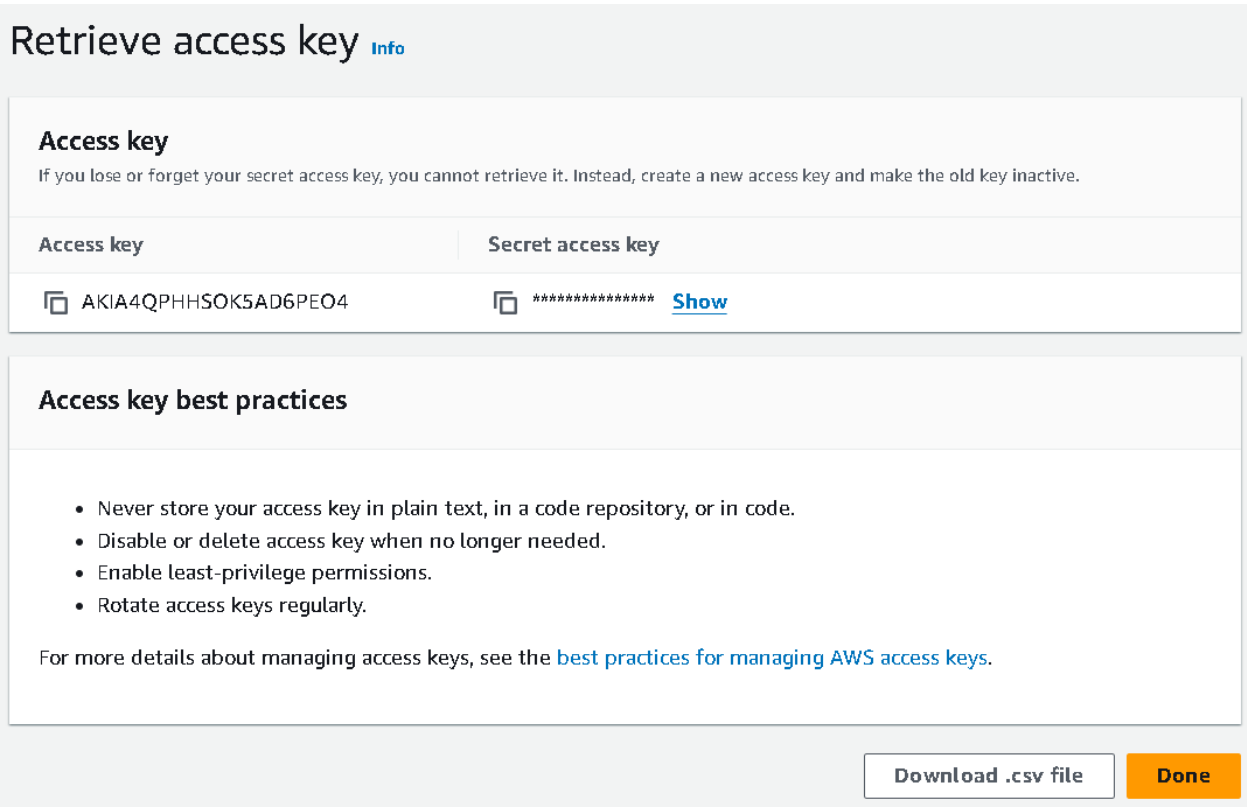



2.2 **Manage Root Access Keys**:

- 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.

Aryan Dangat D15A




3. Install AWS CLI and Configure EC2

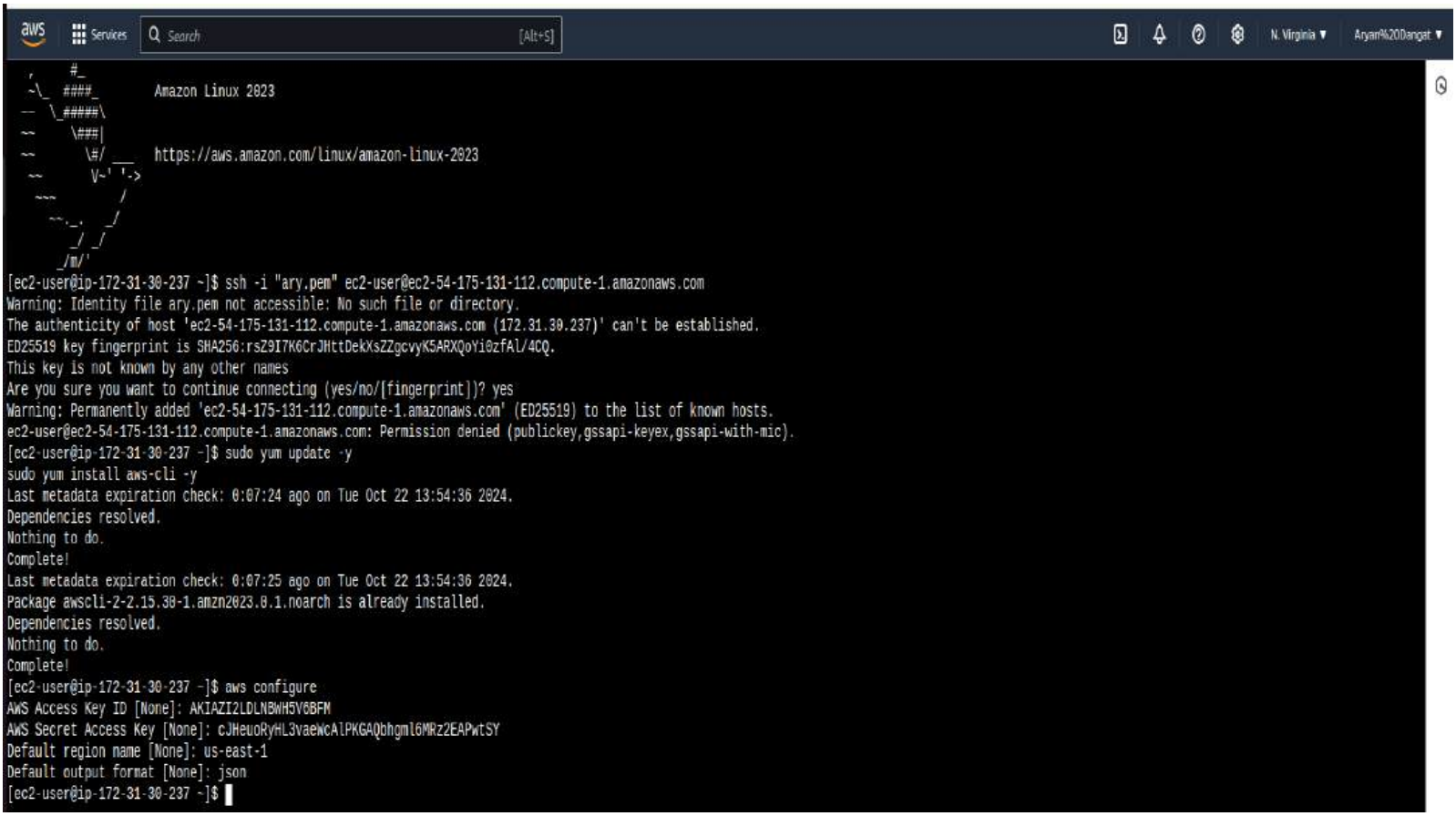
3.1 Update packages and install AWS CLI:

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`



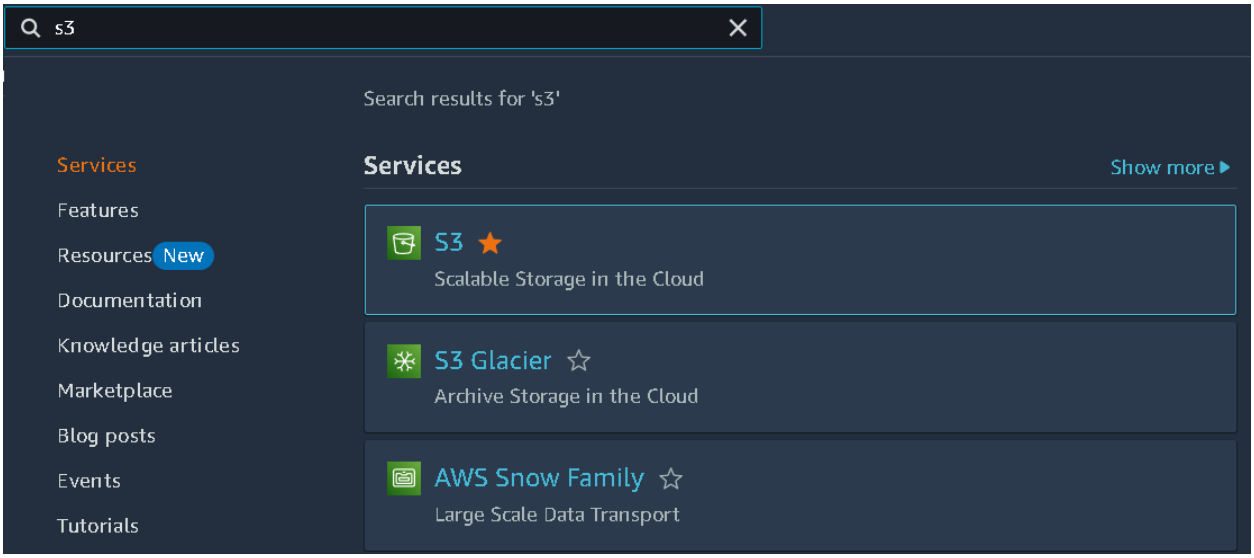

3.3 **Install Python and pip** (since Lambda uses Python):

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

```
[ec2-user@ip-172-31-33-47 ~]$ sudo yum install python3 -y
sudo yum install python3-pip -y
Last metadata expiration check: 0:26:12 ago on Sun Oct 20 11:13:19 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: 0:26:12 ago on Sun Oct 20 11:13:19 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          1.5 MB/s |  92 kB     00:00
(2/2): python3-pip-21.3.1-2.amzn2023.0.8.noarch.rpm            15 MB/s | 1.8 MB     00:00
--------------------------------------------------------------------------------
Total                                                          11 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

Complete!
```

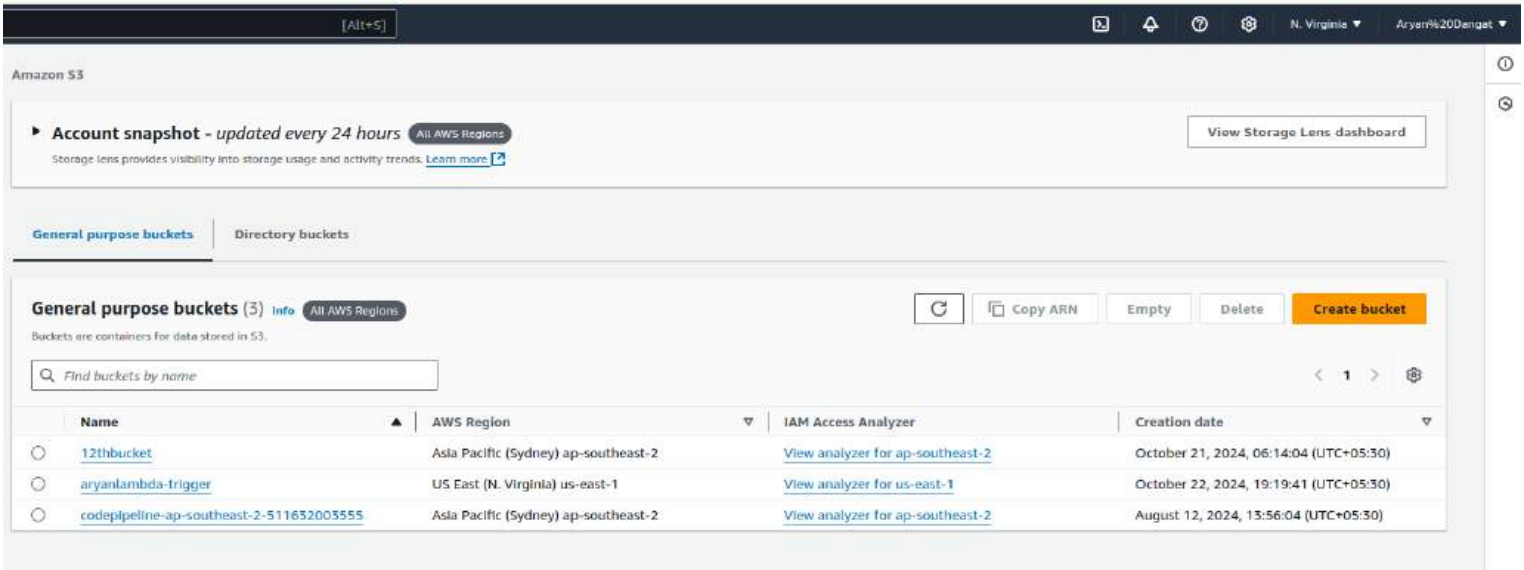
4. Create and S3 Bucket

4.1 In the AWS Management Console, go to **S3**.



4.2 Click **Create bucket**:

- **Bucket Name**: Give a unique name (e.g., `lambda-s3-trigger-bucket`).
- **Region**: Keep the same as your AWS Configuration (e.g., `us-east-1`).
- Keep other settings default.




4.3 Create the bucket.

5. Create the Lambda Function code.

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

nano lambda_function.py

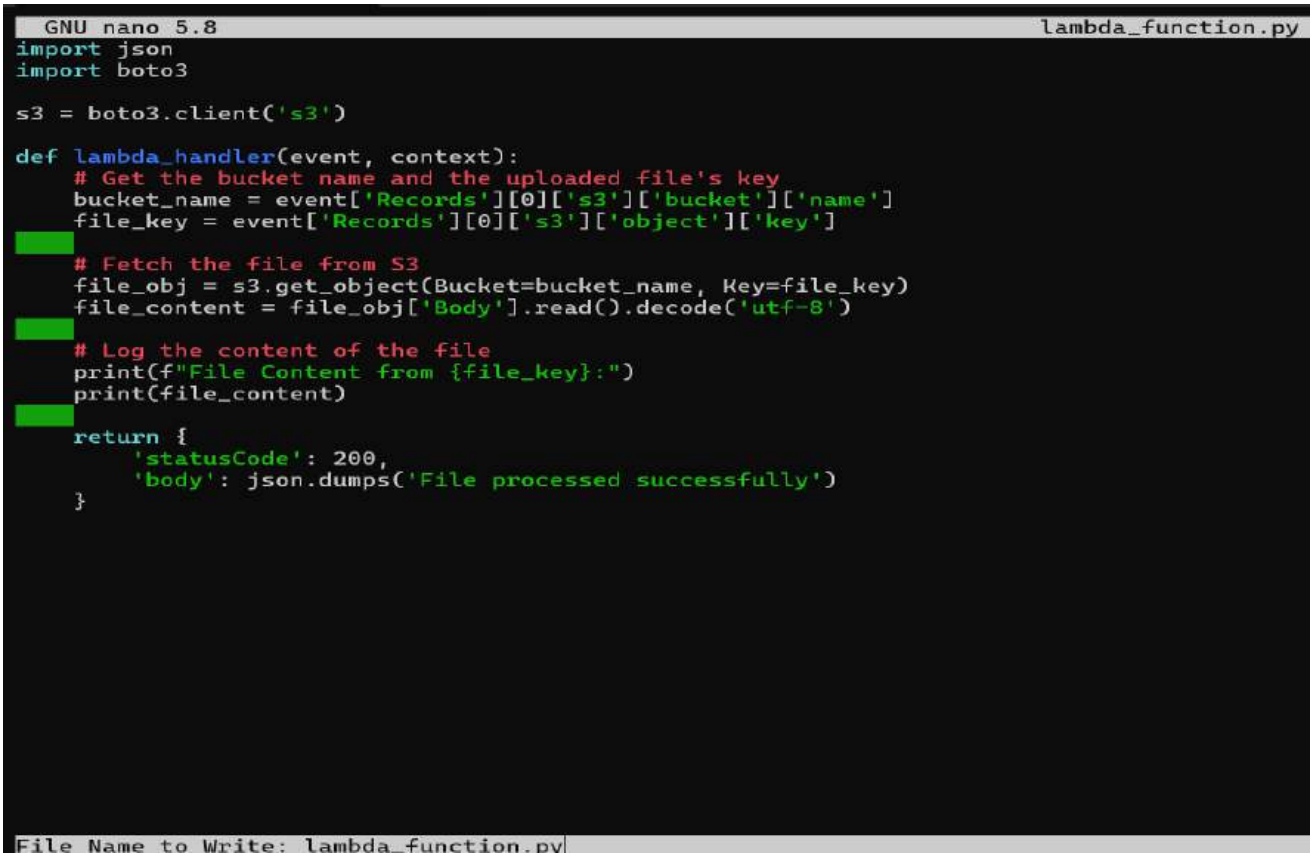
```
[ec2-user@ip-172-31-33-47 ~]$ nano lambda_function.py
```

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`.



6. Deploy the Lambda function from EC2

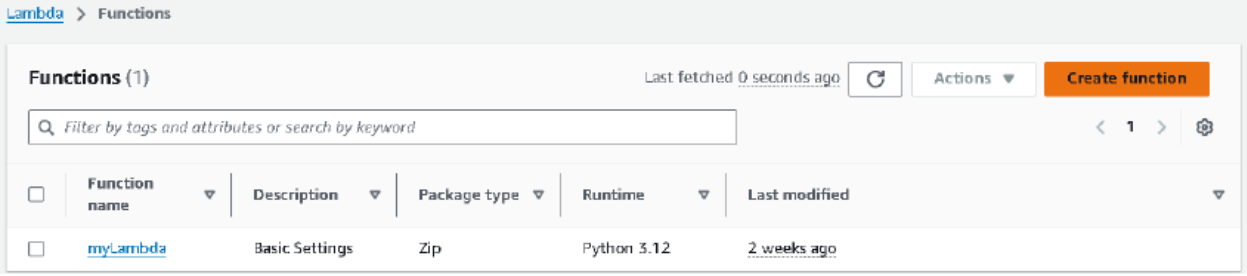
6.1 Package the Lambda function:

zip function.zip lambda_function.py

```
[ec2-user@ip-172-31-33-47 ~]$ zip function.zip lambda_function.py
  adding: lambda_function.py (deflated 47%)
[ec2-user@ip-172-31-33-47 ~]$
```

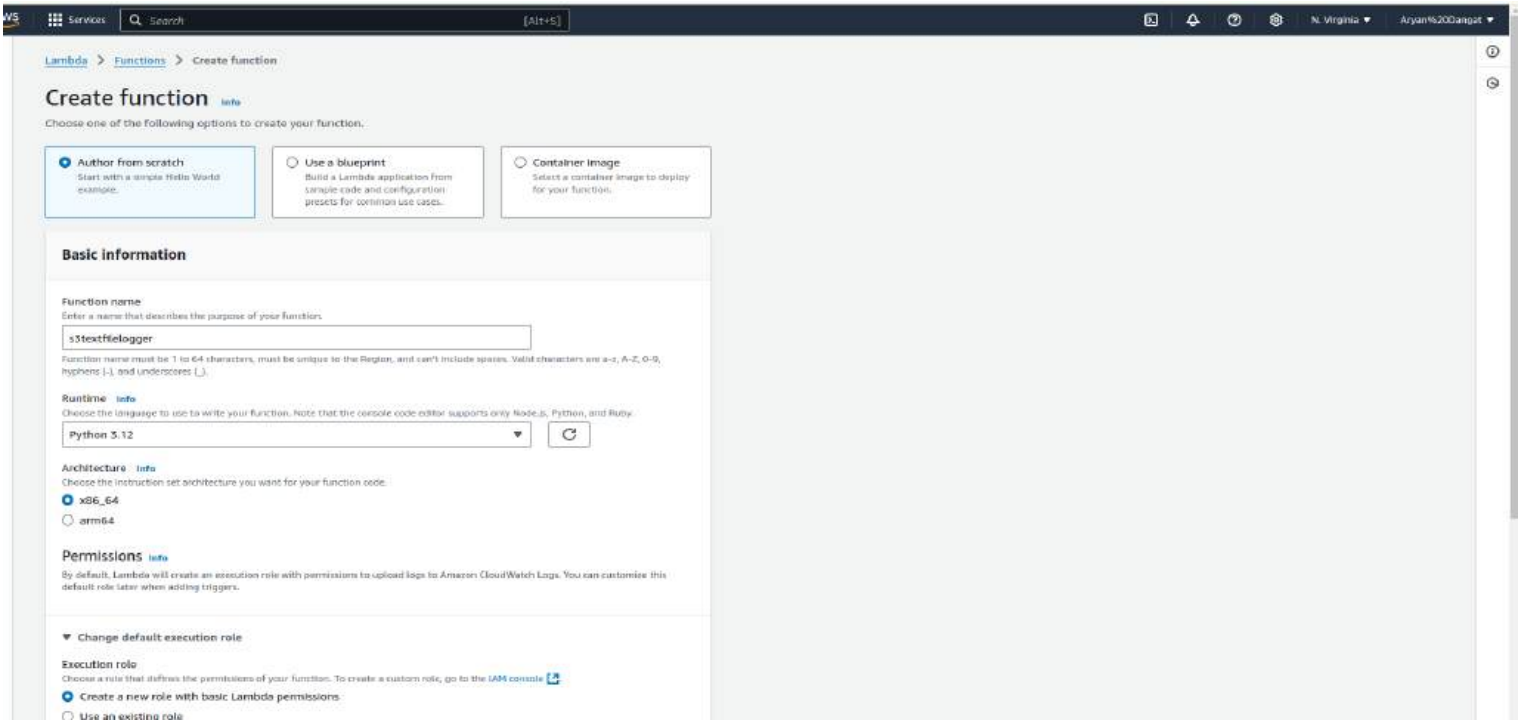
6.2 **Create a Lambda function in AWS Console**:

- Go to **Lambda** > **Create Function**.




- Choose **Author from Scratch**:
  - **Function Name**: `S3TextFileLogger`
  - **Runtime**: Python 3.12
  - **Execution Role**: Select "Create a new role with basic Lambda permissions."

Aryan Dangat D15A




## Permissions Info

By default, Lambda will create an execution role with permissions to upload logs to Amazon CloudWatch Logs. You can customize this default role later when adding triggers.

▼ **Change default execution role**

**Execution role**

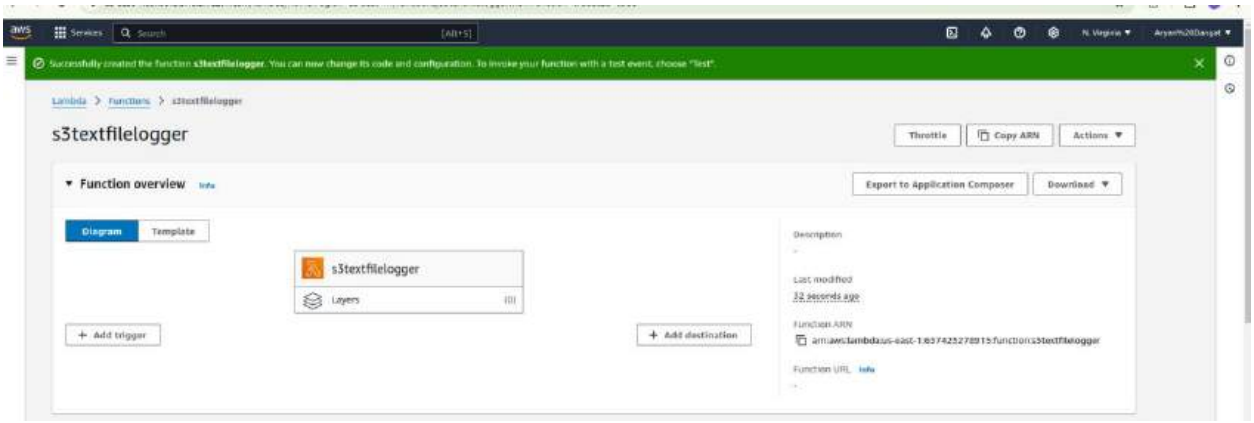
Choose a role that defines the permissions of your function. To create a custom role, go to the IAM console.

- (●) Create a new role with basic Lambda permissions
- ( ) Use an existing role
- ( ) Create a new role from AWS policy templates

ⓘ Role creation might take a few minutes. Please do not delete the role or edit the trust or permissions policies in this role.

Lambda will create an execution role named S3TextFileLogger-role-l6qnx3qp, with permission to upload logs to Amazon CloudWatch Logs.

- Click **Create Function**.




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

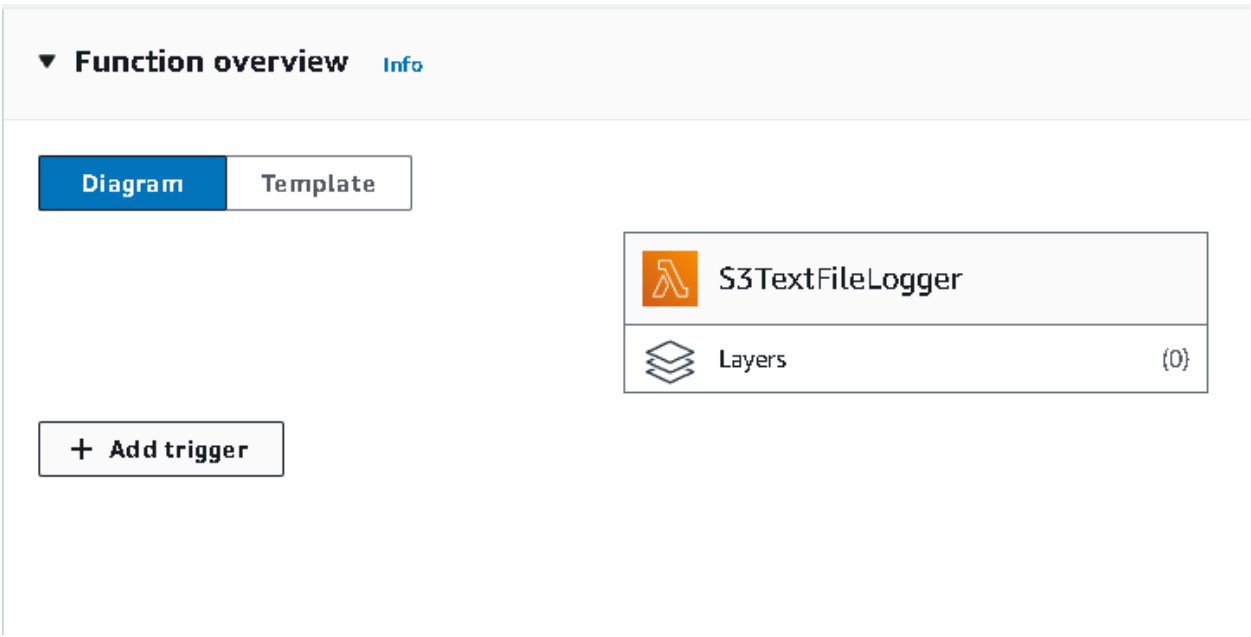
aws lambda update-function-code --function-name S3TextFileLogger --zip-file fileb://function.zip

```
[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+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"
    }
```

7. Configure S3 as the Trigger

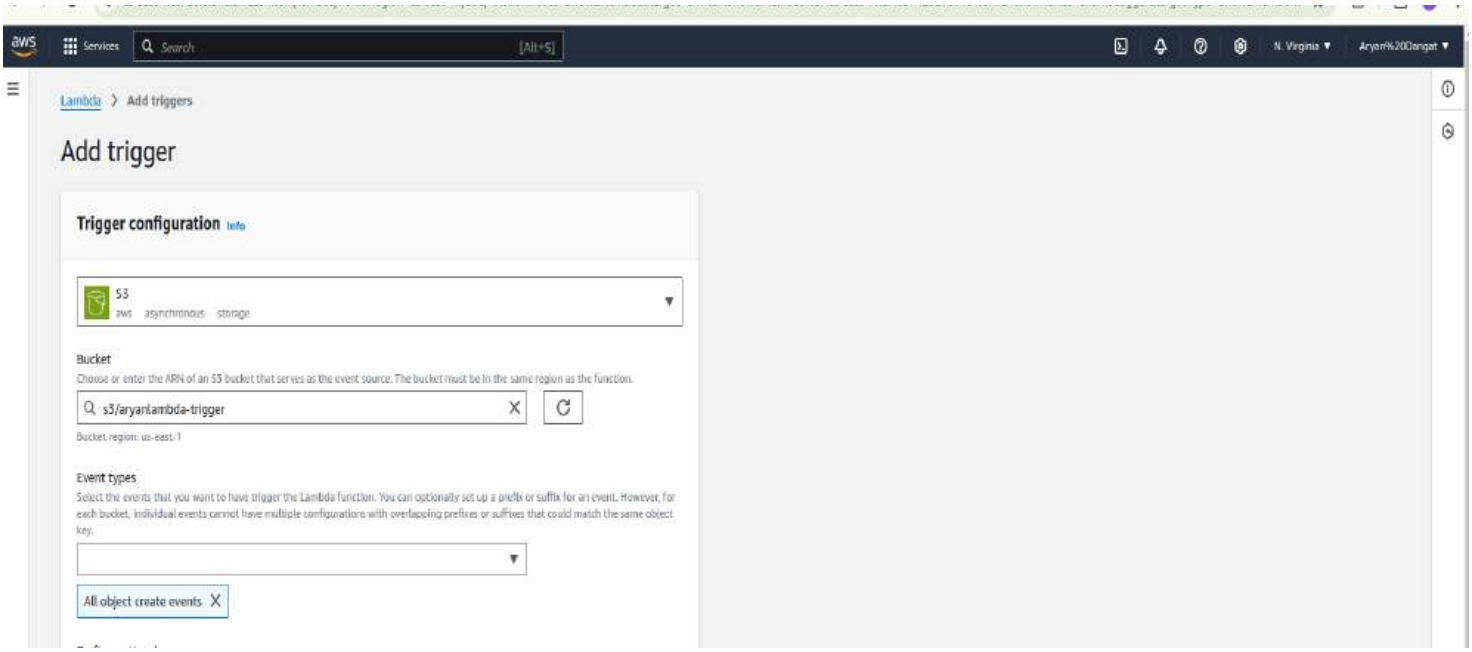
7.1 **In Lambda console**, go to the **Function Overview** section and click **Add Trigger**.

Aryan Dangat D15A



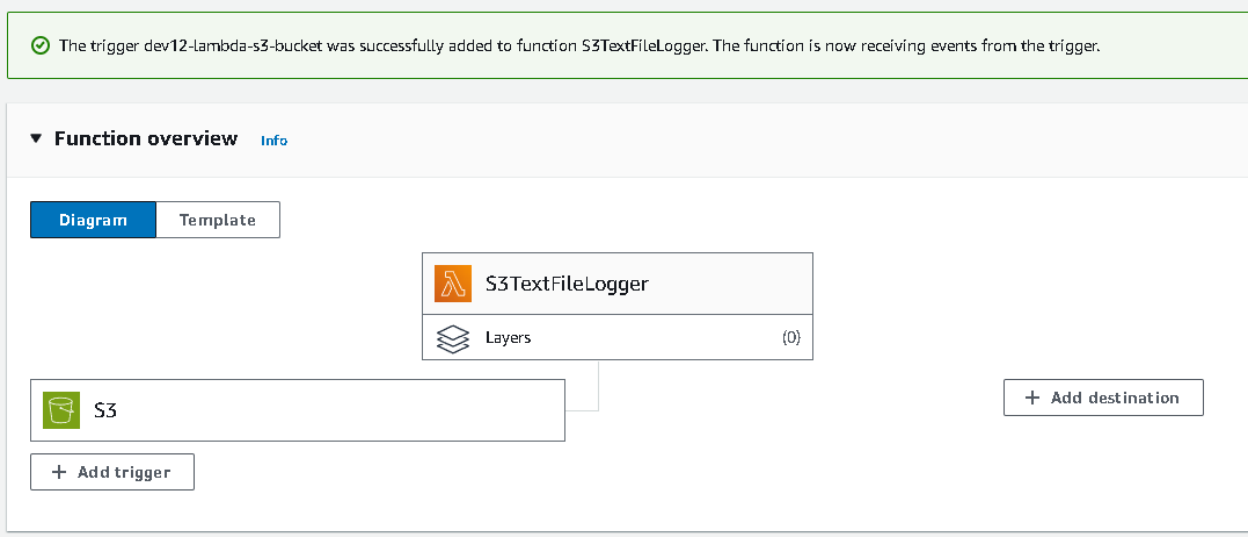

7.2 Choose **S3** as the trigger:

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




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

Aryan Dangat D15A

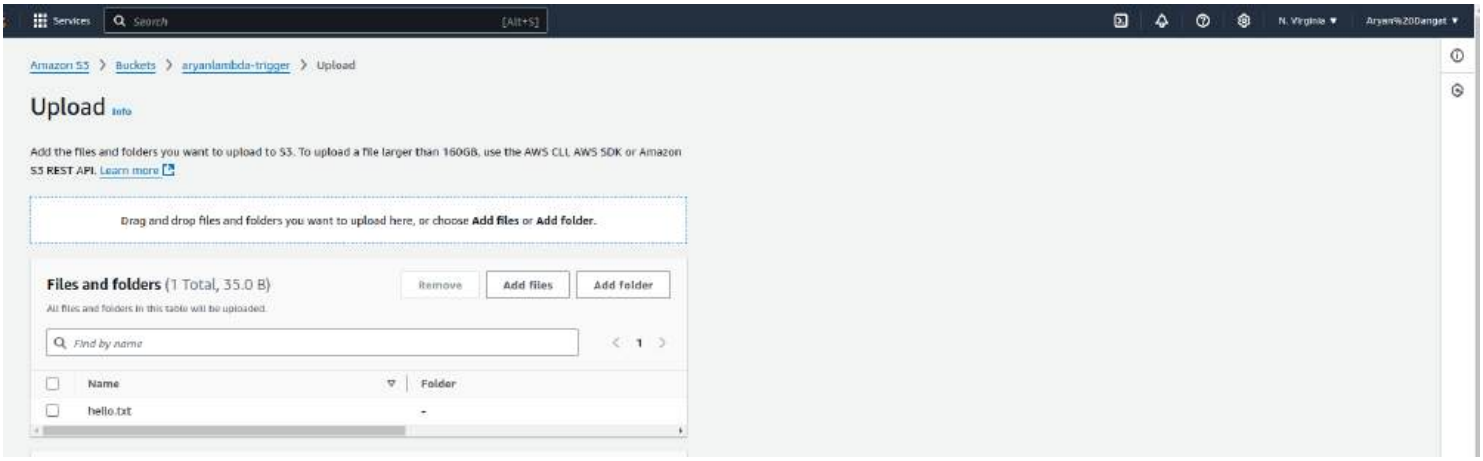



8. Upload a File and Test

8.1 Create a text file in your local host with some content.

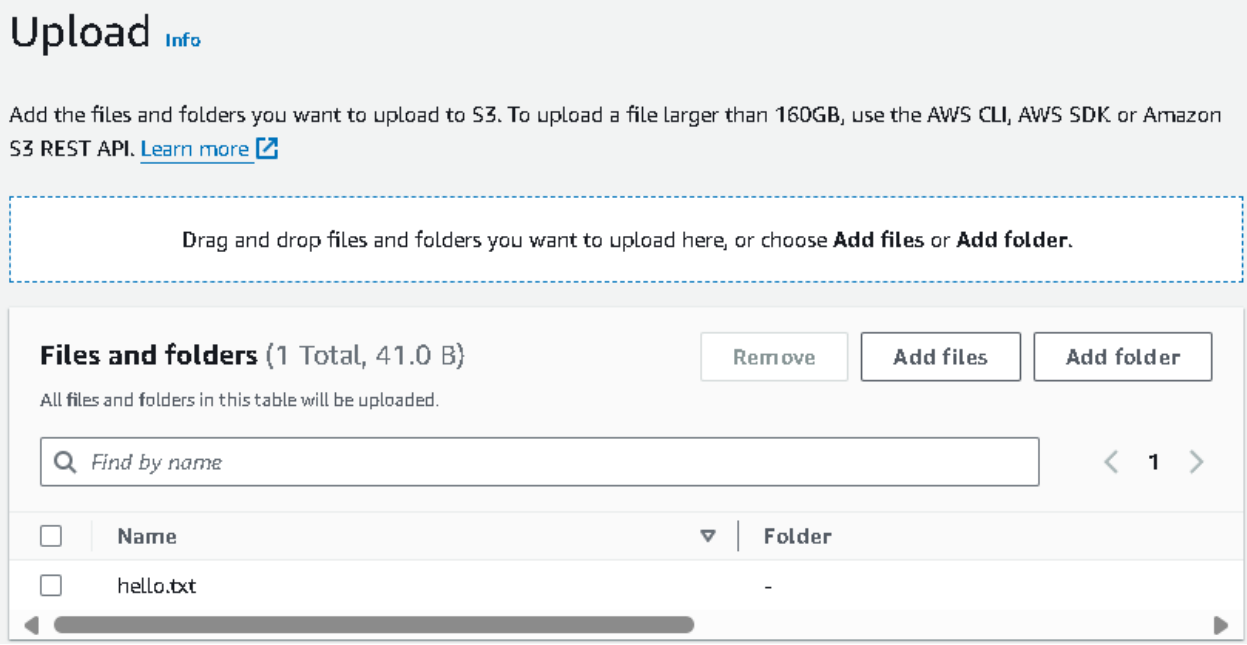

8.2 **Upload a text file** to your S3 bucket:

Go to **S3** > your bucket > **Upl**




**oad**.

- Upload a `.txt` file with some content (e.g., `hello.txt`)

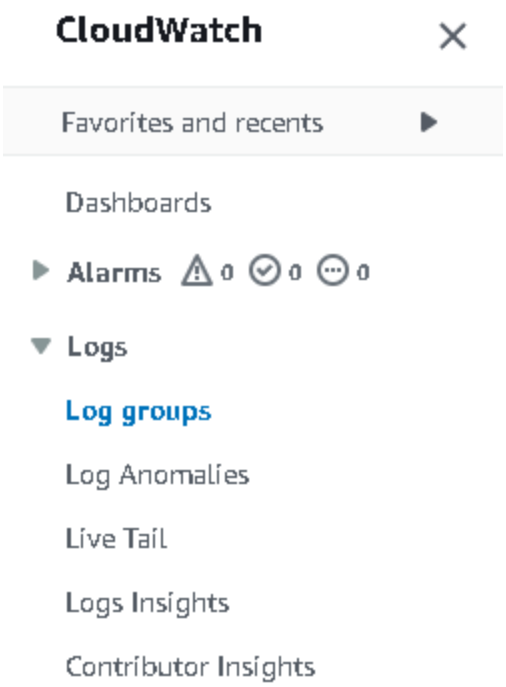



The Lambda function will automatically run when the file is uploaded.

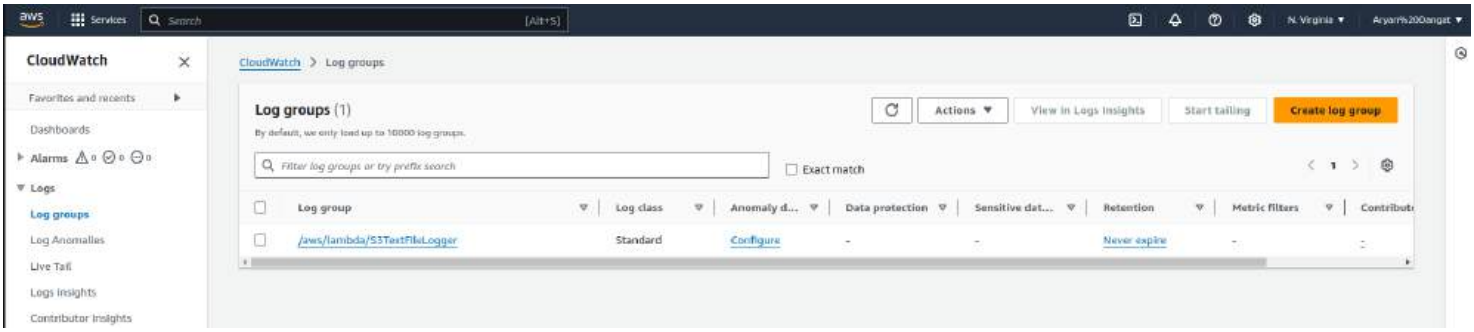
9. Check Logs in CloudWatch

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

Aryan Dangat D15A

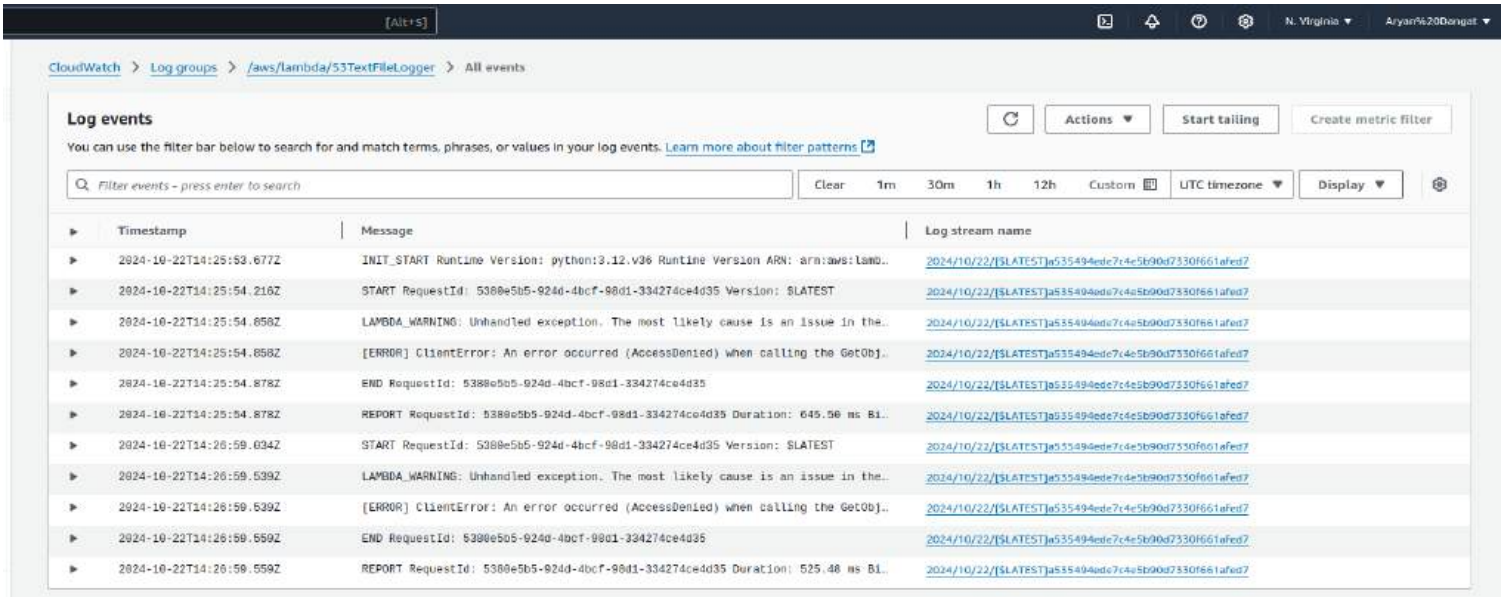



9.2 Under **Log Groups**, find the log group for your Lambda function (`/aws/lambda/S3TextFileLogger`).

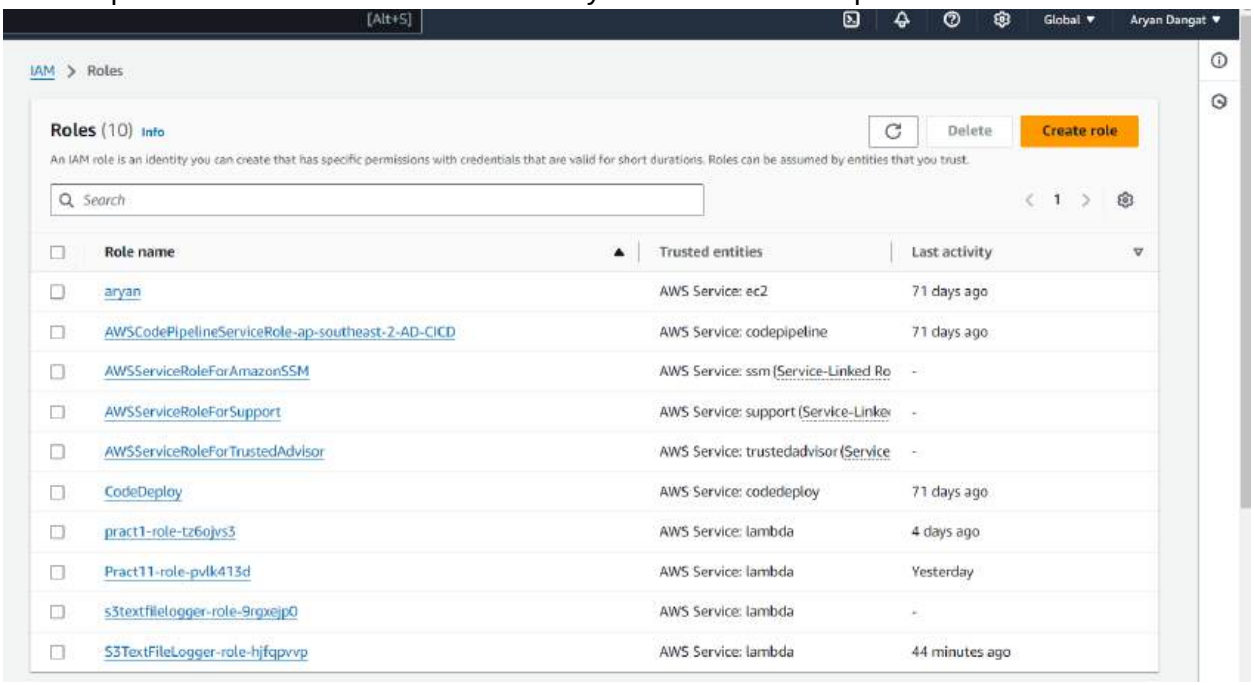



9.3 Open the latest log stream to see the file content logged by the Lambda function.

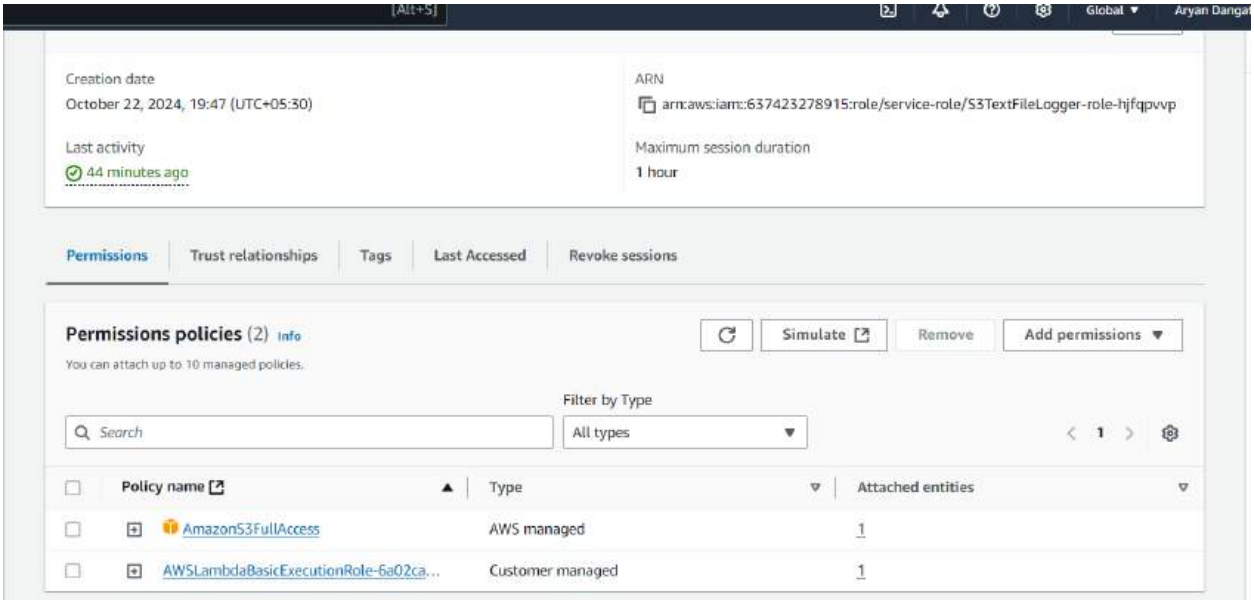
Aryan Dangat D15A

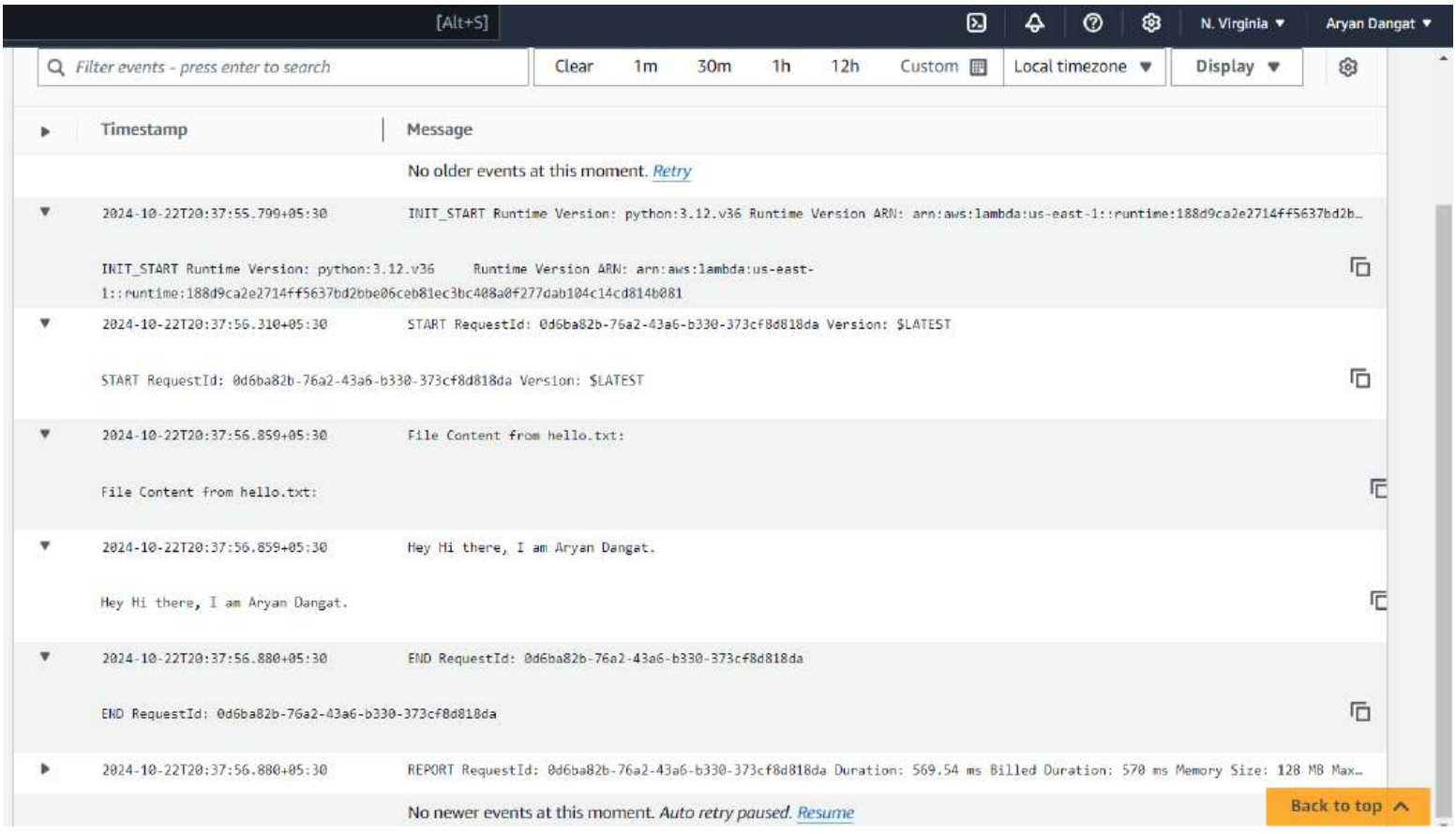



Edit the permissions of the s3 bucket to rectify the access denied problem.



Aryan Dangat D15A



Aryan Dangat D15A