ADVANCE DEVOPS PRACTICAL EXAM

Name: - Tejas Dhondibhau Gunjal

Class: - D15A

Roll: No: - 18

Case Study: - Real-Time Log Processing

Concepts Used: AWS Lambda, CloudWatch, S3.

Problem Statement: "Set up a Lambda function that triggers whenever a new log entry is added to a CloudWatch Log Group. The Lambda function should filter specific log events and store them in an S3 bucket."

Tasks:

Create a CloudWatch Log Group and set up a Lambda function that triggers on new log entries.

Write a Python Lambda function to filter logs based on a keyword (e.g.,'ERROR').

Store the filtered logs in an S3 bucket.

Test by generating logs and checking the S3 bucket for the filtered entries.

Introduction: -

This case study explores real-time log processing using AWS services, focusing on how modern cloud infrastructure can automate monitoring, filtering, and storage of log data. The core idea is to design a scalable, serverless architecture that can respond instantly to changes in log data, ensuring that relevant information is captured without human intervention. In the DevOps world, where log generation happens continuously, this system ensures that critical logs (e.g., errors) are isolated, analysed, and stored for auditing and troubleshooting purposes.

The architecture involves two AWS Lambda functions working together:

The first Lambda function simulates an application by generating logs in CloudWatch. The second Lambda function is triggered automatically by new logs in CloudWatch, filtering the relevant entries (logs containing the keyword 'ERROR') and storing them in an S3 bucket for future analysis.

This setup showcases the power of event-driven architectures, allowing organizations to build serverless workflows that efficiently monitor, filter, and store relevant logs without manual intervention.

Key Features and Applications: -

This real-world log processing system offers several key features and practical applications, demonstrating how modern cloud infrastructure can simplify log management.

Key Features:

1. Serverless Architecture

   No need to manage servers manually—AWS Lambda handles everything. The system scales automatically based on the volume of log events, ensuring smooth performance even during sudden spikes in traffic. This reduces operational overhead and allows developers to focus on building features rather than worrying about infrastructure.

2. Real-Time Log Processing

   Logs are processed the moment they are generated. As soon as a new entry appears in the CloudWatch Log Group, the second Lambda function triggers instantly to filter relevant logs. This ensures that important events, such as errors, are caught and stored without delays, making it easy to respond to issues as they happen.

3. Efficient Filtering and Storage

   Not all logs are equally important. The Lambda function filters logs based on specific keywords (like ‘ERROR’) and stores only the relevant ones in Amazon S3. This helps reduce clutter and ensures that only actionable data is archived. It also saves storage costs by avoiding unnecessary log retention.

4. Seamless Integration with AWS Services

   The setup makes the most of AWS’s native services: CloudWatch for logging, Lambda for processing, and S3 for storage. These services are well-integrated, offering a smooth and reliable workflow with minimal configuration.

5. Scalable and Fault-Tolerant

   Whether the system generates a few logs or millions, AWS services can handle it without performance degradation. Additionally, storing filtered logs in S3 ensures durability and availability, providing access to historical data when needed.

Real World Applications: -

1. **Monitoring Production Systems**

   This setup is ideal for e-commerce websites or banking applications, where real-time tracking of errors and performance issues is essential. If an error occurs (such as a payment failure), it gets flagged and stored immediately for further investigation.

2. **IoT Device Monitoring**

   In IoT systems that generate a large volume of data, only critical logs (e.g., hardware malfunctions) need to be flagged. The Lambda-based log filter ensures that only essential logs are kept, helping system administrators focus on what matters.

3. **Compliance and Auditing**

   For industries like healthcare and finance, keeping a record of critical events is essential for compliance. Storing filtered logs in S3 provides a secure, long-term archive that can be accessed during audits or investigations.

4. **DevOps and Continuous Monitoring**

   This solution supports DevOps practices by ensuring continuous log monitoring and error tracking. Developers and operations teams can act on issues quickly, improving system reliability and reducing downtime.

Third-Year Project Integration

- **Project Name**: *Suvidha* – A service aggregation platform inspired by UrbanClap, aimed at connecting users with local service providers.
- **Relevance of Log Processing**: Real-time logging would help monitor crucial events such as:
  - Service booking failures
  - Payment errors or transaction issues
  - System downtime or performance bottlenecks
- **Impact on User Experience**: Automated log filtering and storage would ensure quick identification and resolution of issues, leading to smoother operations and better user satisfaction.
- **Benefit of Event-Driven Architecture**: Just like in the case study, *Suvidha* could use event-driven logging to ensure real-time alerts and proactive system monitoring for continuous reliability.

Procedure: -

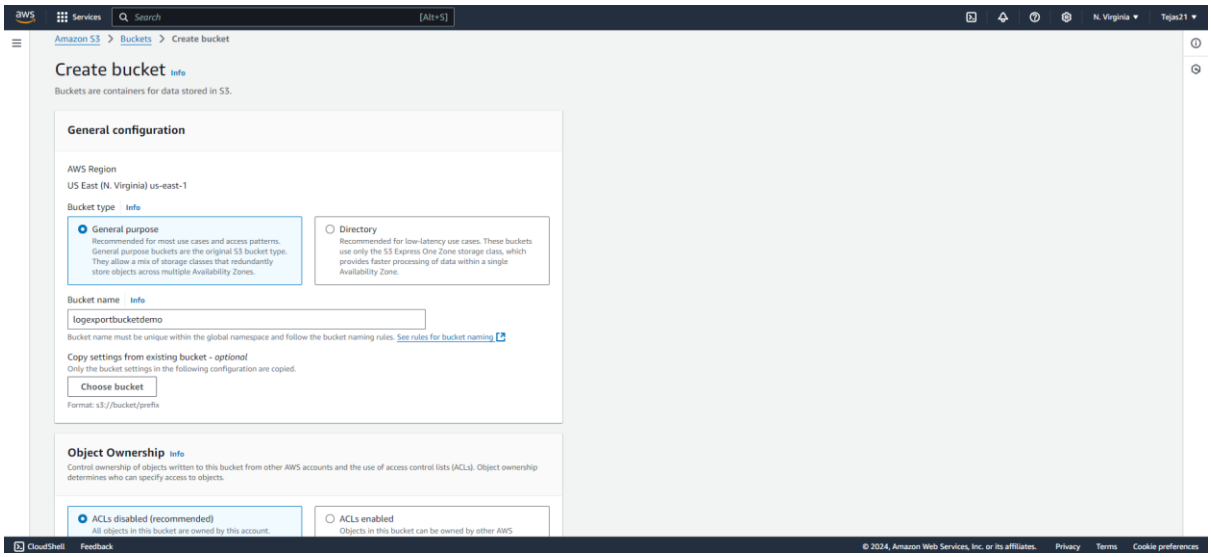
Architecture Overview

The workflow includes the following components:

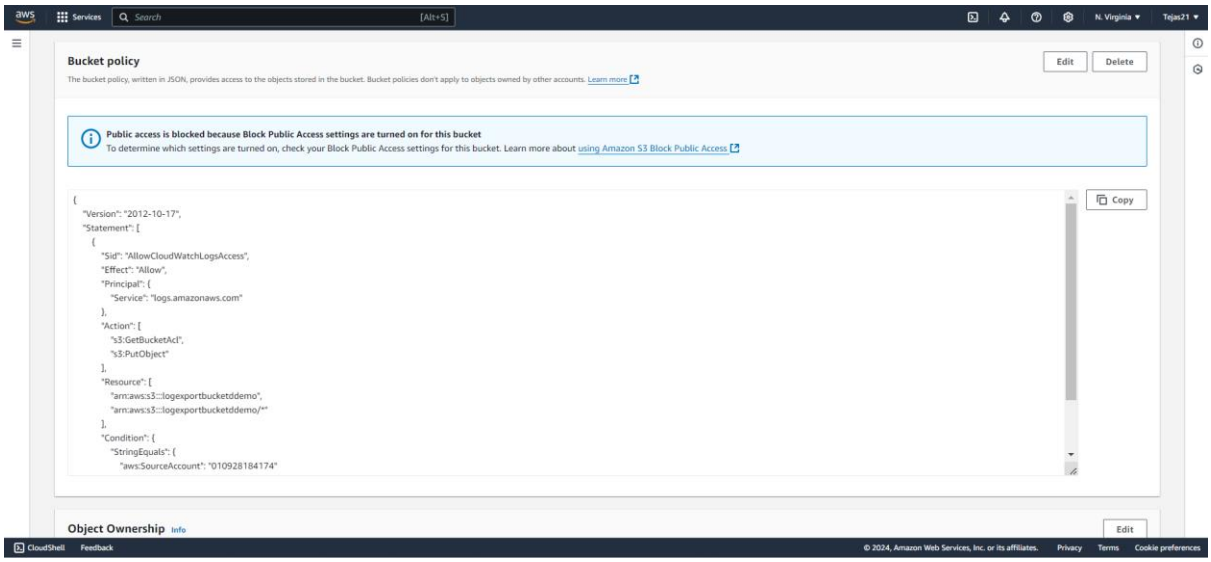
1. **Lambda Function 1:** A Python-based application generating logs.
2. **CloudWatch Log Group:** Captures logs from the first Lambda function.
3. **Lambda Function 2:** Filters logs based on keywords and stores relevant logs in an S3 bucket.
4. **S3 Bucket:** Stores filtered logs for future reference or analysis

Step 1: Create an S3 Bucket

Open the S3 Console in AWS.Click on Create Bucket, and provide a name.



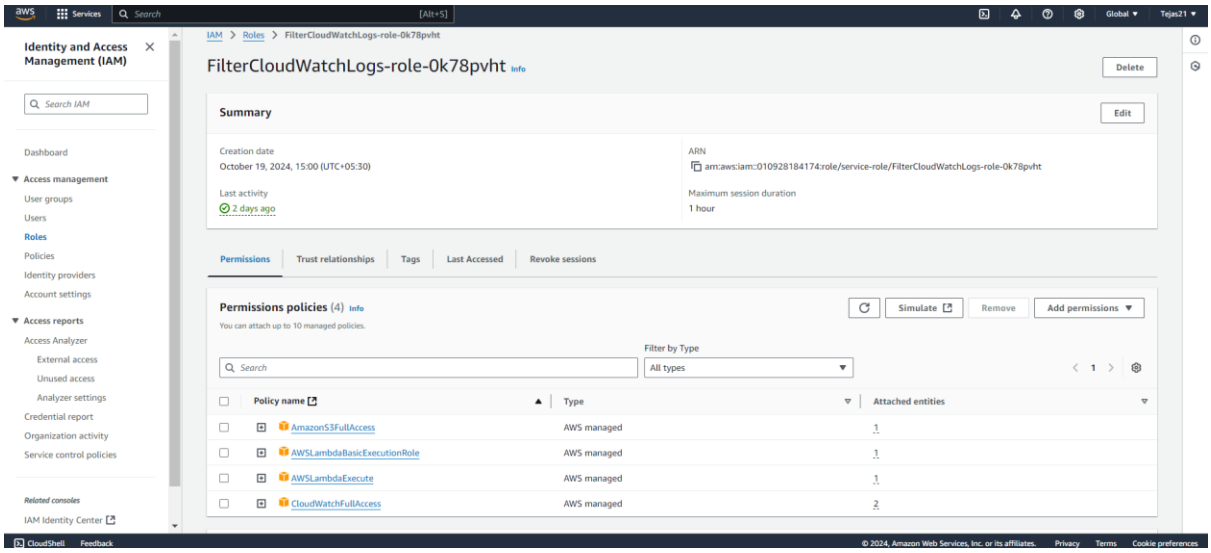
Configure the bucket permissions to allow the second Lambda function to write objects. Ensure public access is blocked unless necessary.



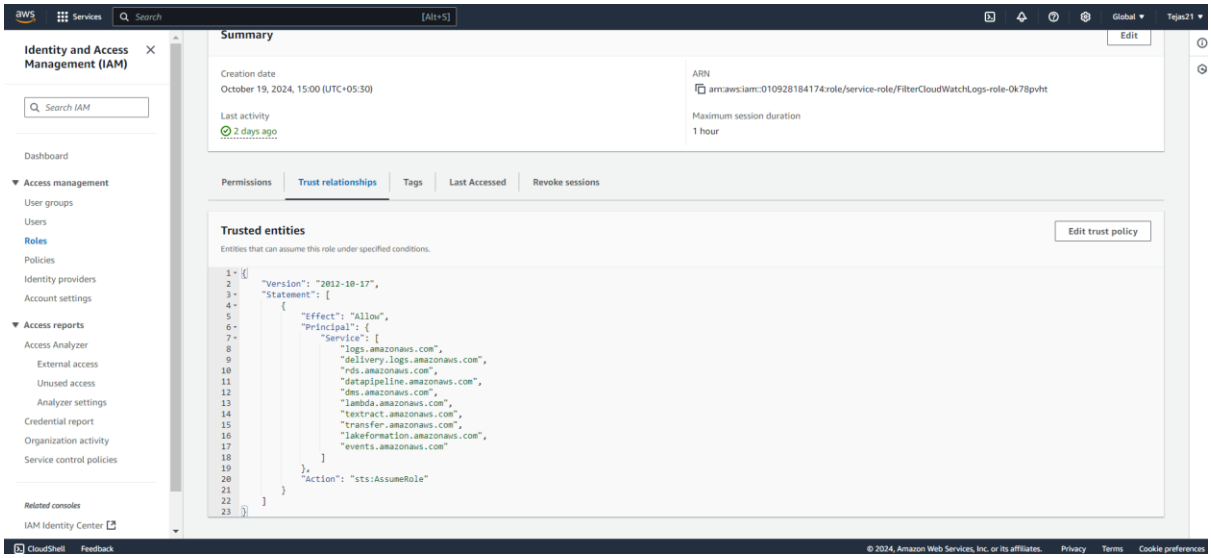
Step 2: - Configure IAM Role with Required Permissions

Attach the following policies:

- AWSLambdaBasicExecutionRole (for Lambda to execute successfully).
- CloudWatchReadOnlyAccess (for Lambda to read CloudWatch logs).
- AmazonS3FullAccess (to allow the second Lambda to write logs to the S3 bucket).

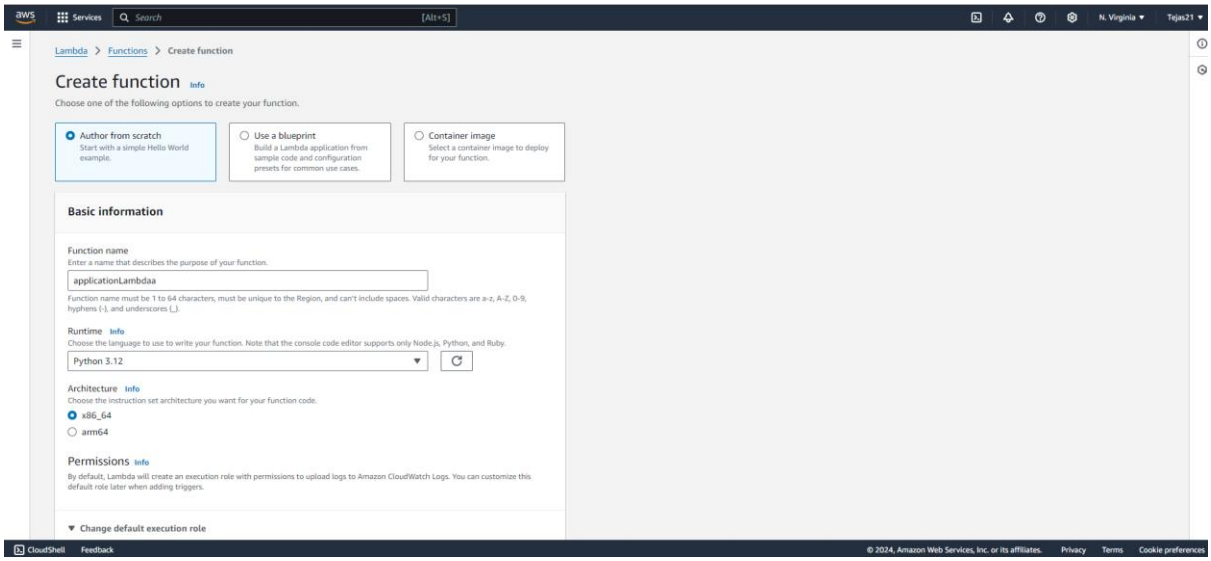


Choose the Lambda service as the trusted entity.

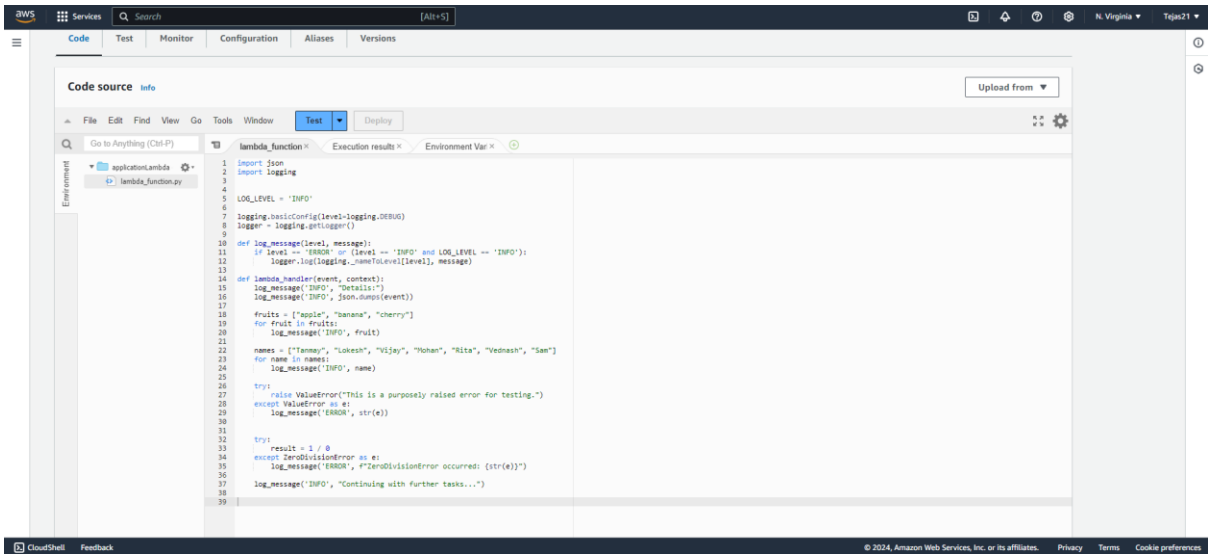


Step 3: Lambda Function 1 – Generate Logs in CloudWatch

Open the Lambda Console and click Create Function. Select Author from scratch and name the function.



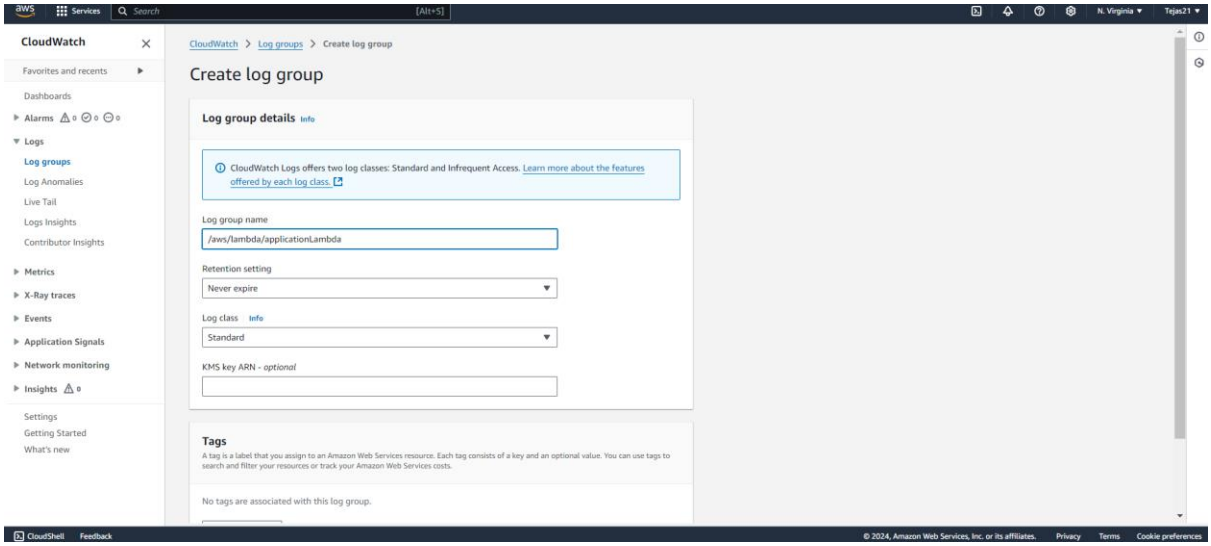
Deploy the Lambda function. Test the function from the Lambda console to ensure logs are generated in CloudWatch.



Step 4: CloudWatch Log Group Setup

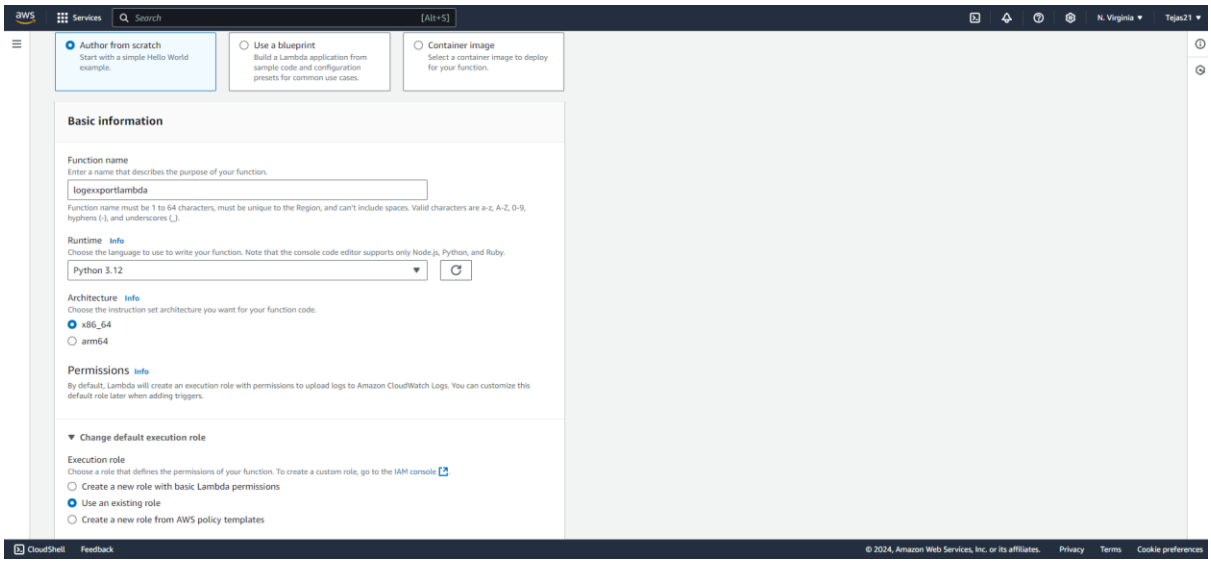
Go to CloudWatch Console > Log Groups.

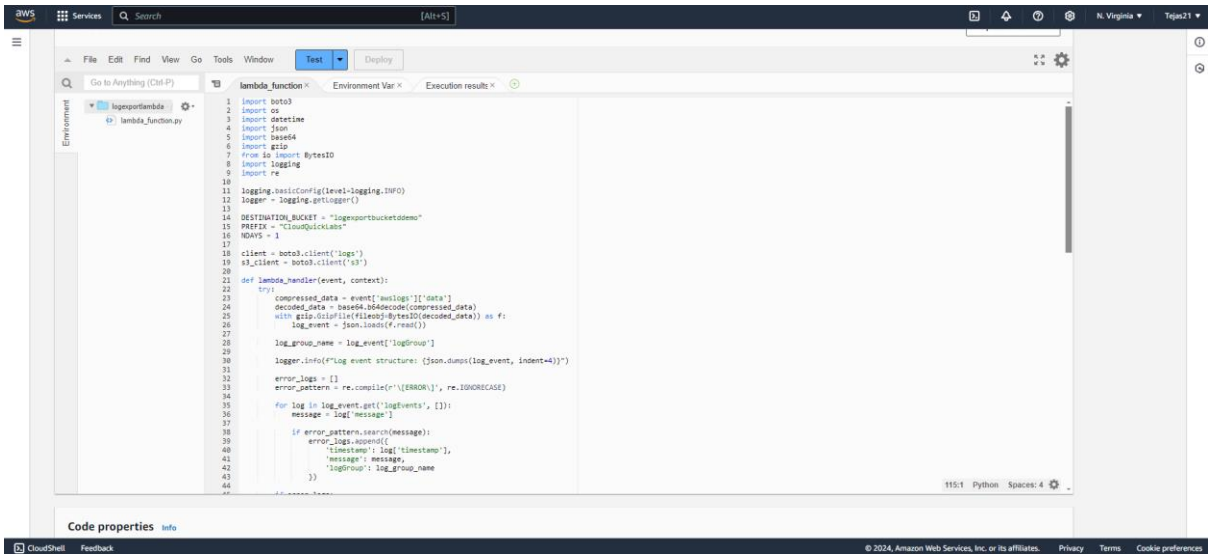
Locate the log group generated by Lambda Function 1 (it should be automatically created , although we can also create manually).



Step 5: Lambda Function 2 – Filter Logs and Store in S3

Open the Lambda Console and create another function (logexportlambda).





Step 6: Add Trigger using Subscription Filter

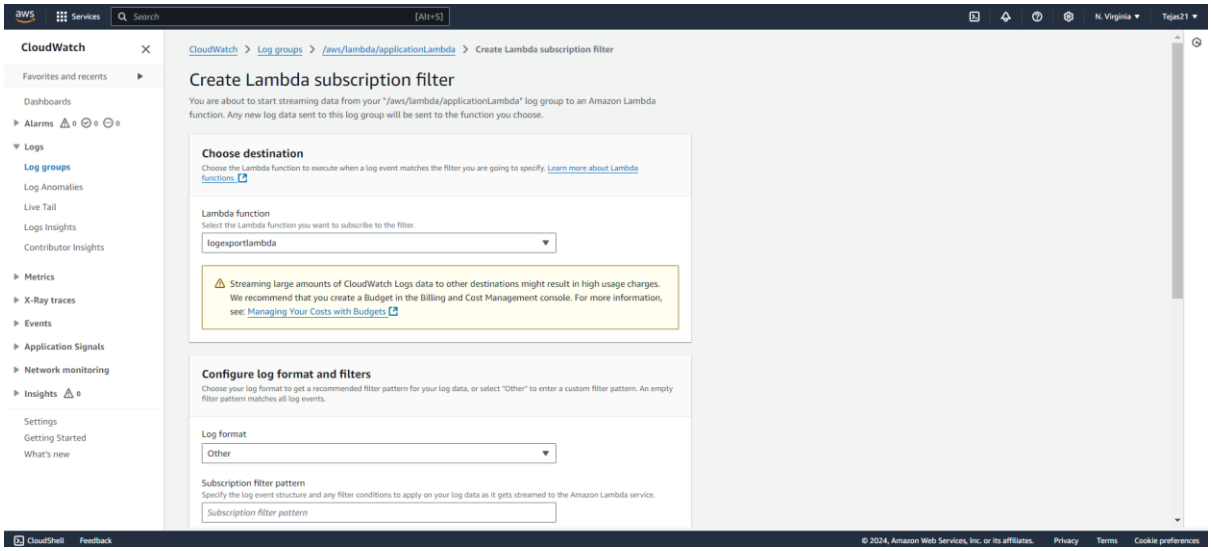
Open CloudWatch Console → Go to Log Groups.

Select the Log Group created by Lambda Function 1

Click on Actions → Create Subscription Filter.

Enter a Filter Name

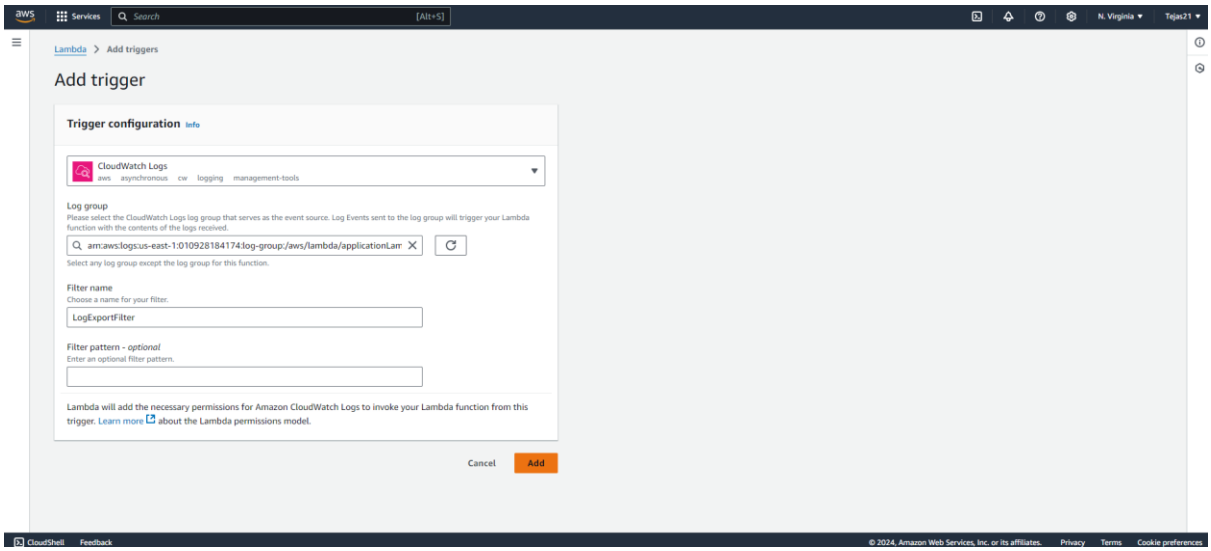
In the Destination Function field, select your second Lambda function (logexportlambda).



Now go to the second lambda function.

In the Configuration tab of the Lambda function, click Add Trigger.

In the Trigger Configuration panel, select CloudWatch Logs from the list of available services and Select the Log Group created by the first Lambda function (log-generator-lambda).

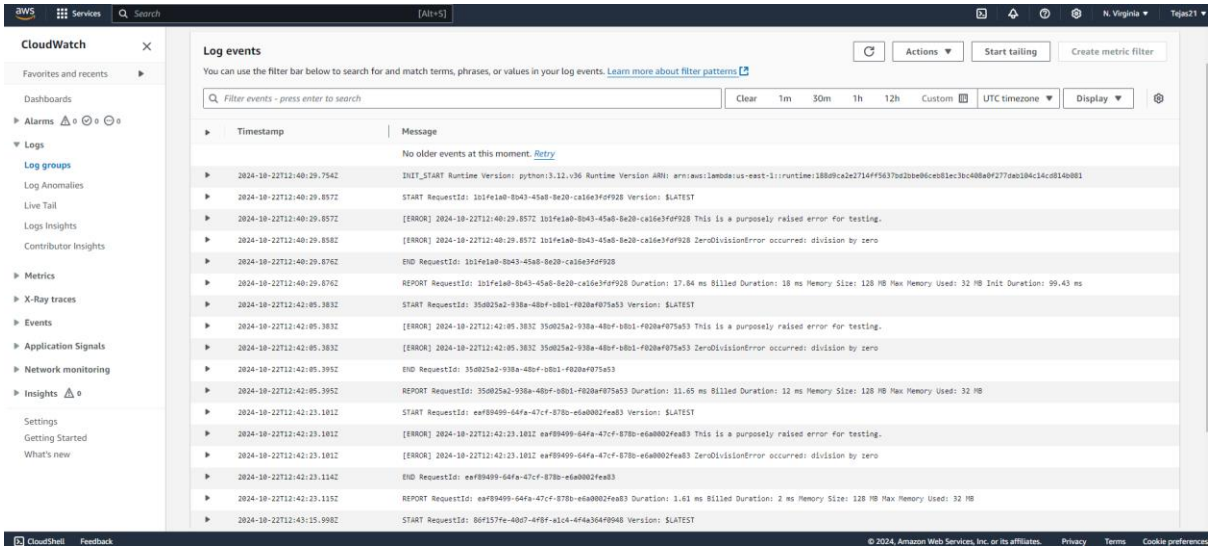


Step 7: - Testing the setup

Run the First Lambda Function:

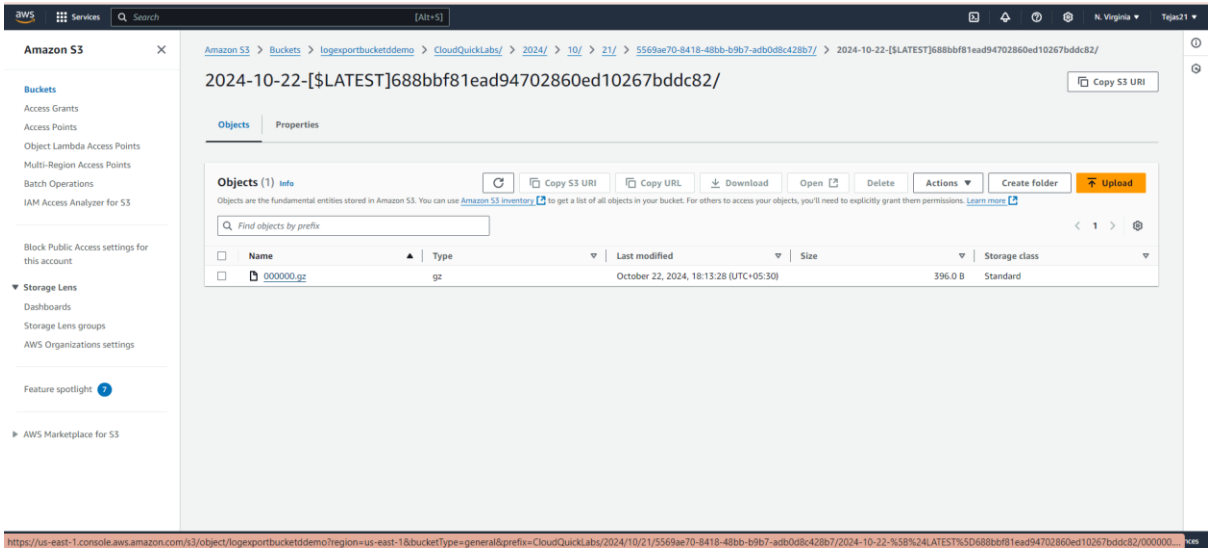
Go to Lambda Function 1 and click Test.

Ensure that logs are generated in the CloudWatch Log Group.

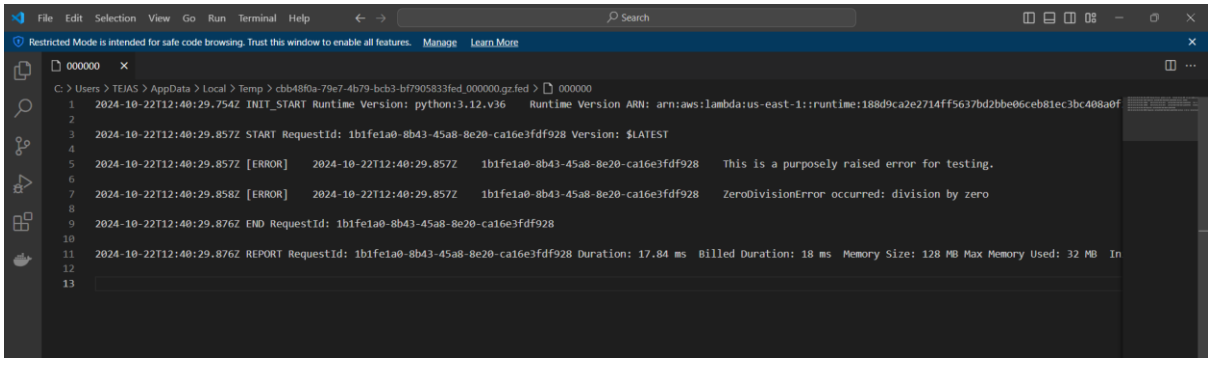


Check the S3 Bucket:

Verify that the filtered logs are saved in the bucket after the second Lambda function runs.



The log files should contain only entries with the keyword ‘ERROR’.



Conclusion: -

This case study effectively demonstrates a real-time log processing architecture using AWS services, simulating how applications generate and manage logs. By leveraging two Lambda functions, it automates log generation, filtering, and storage, ensuring that relevant errors are captured and stored efficiently. This setup not only showcases the benefits of serverless computing but also emphasizes the importance of proactive log management in enhancing application reliability and user experience.