**ADVANCE DEVOPS PRACTICAL EXAM**

**Name: -** Abhay Gupta

**Class: -** D15B

# 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.

1. 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.

1. 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.

1. 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.

1. 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: -**

# 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.

# 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.

# 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.

# 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:

o Service booking failures o Payment errors or transaction issues o 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

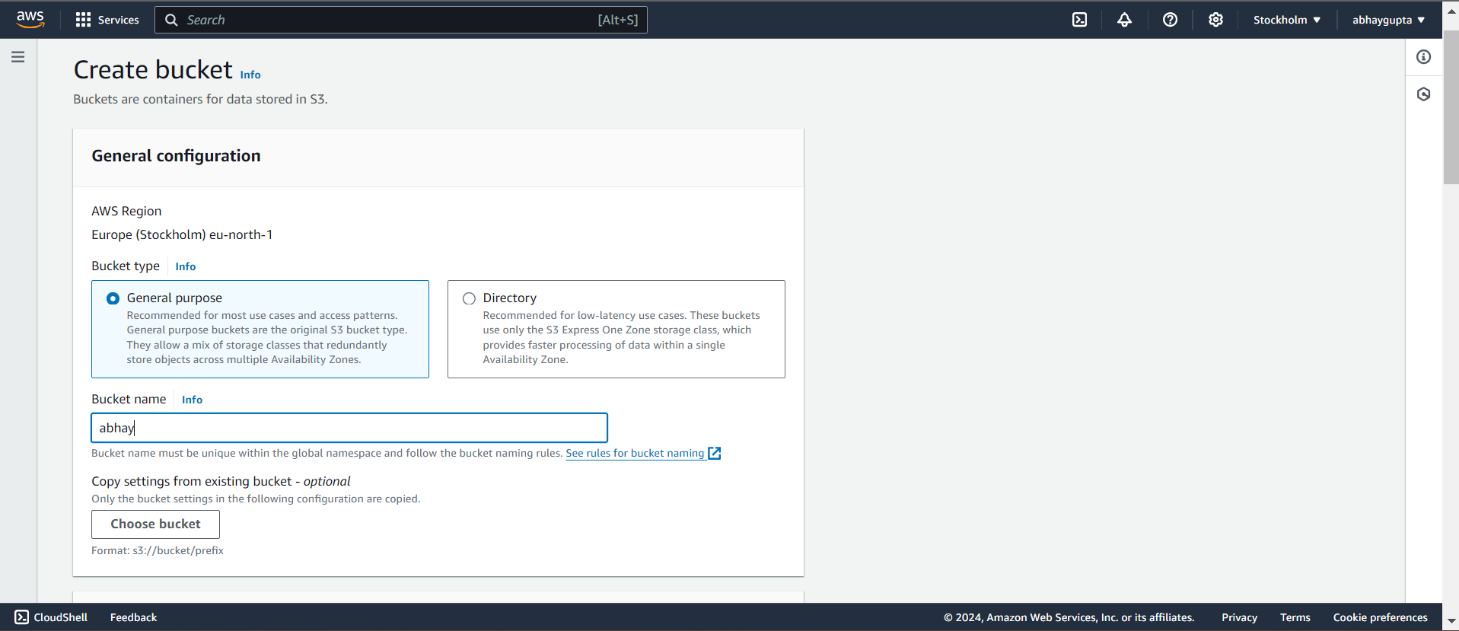
The workflow includes the following components:

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**

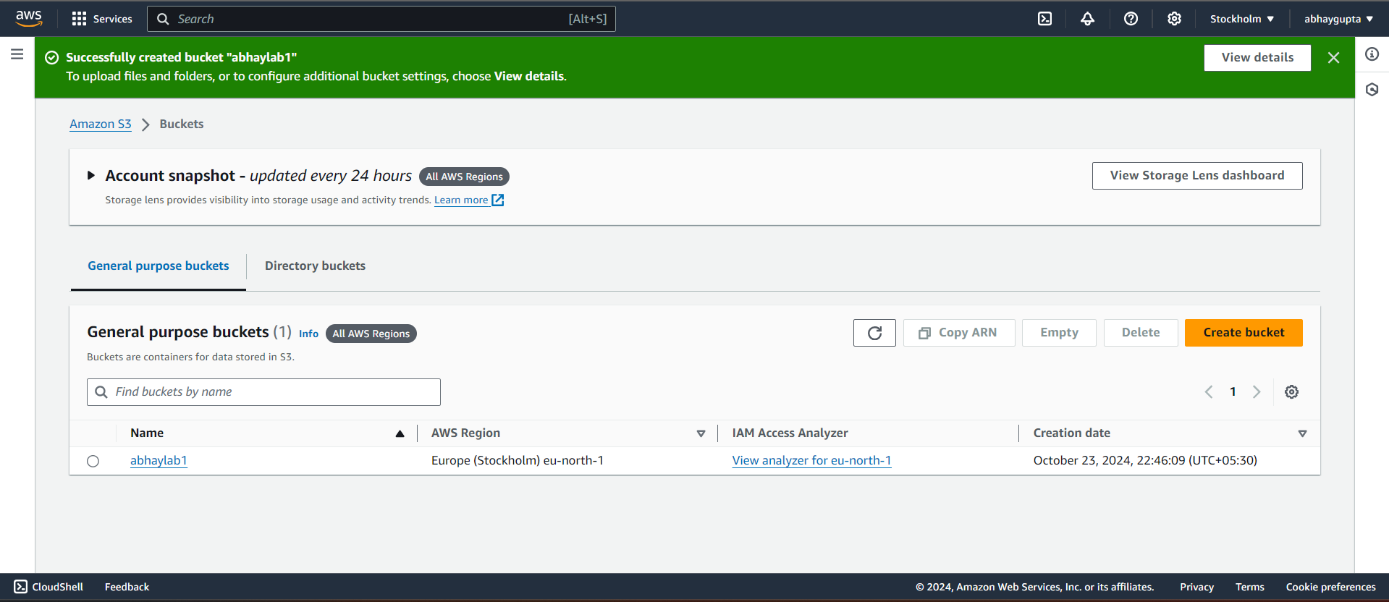
This is where the filtered logs will be stored.

1. **Go to AWS Management Console**: Open the console
2. **Navigate to S3:**



1. **Create a new bucket**:

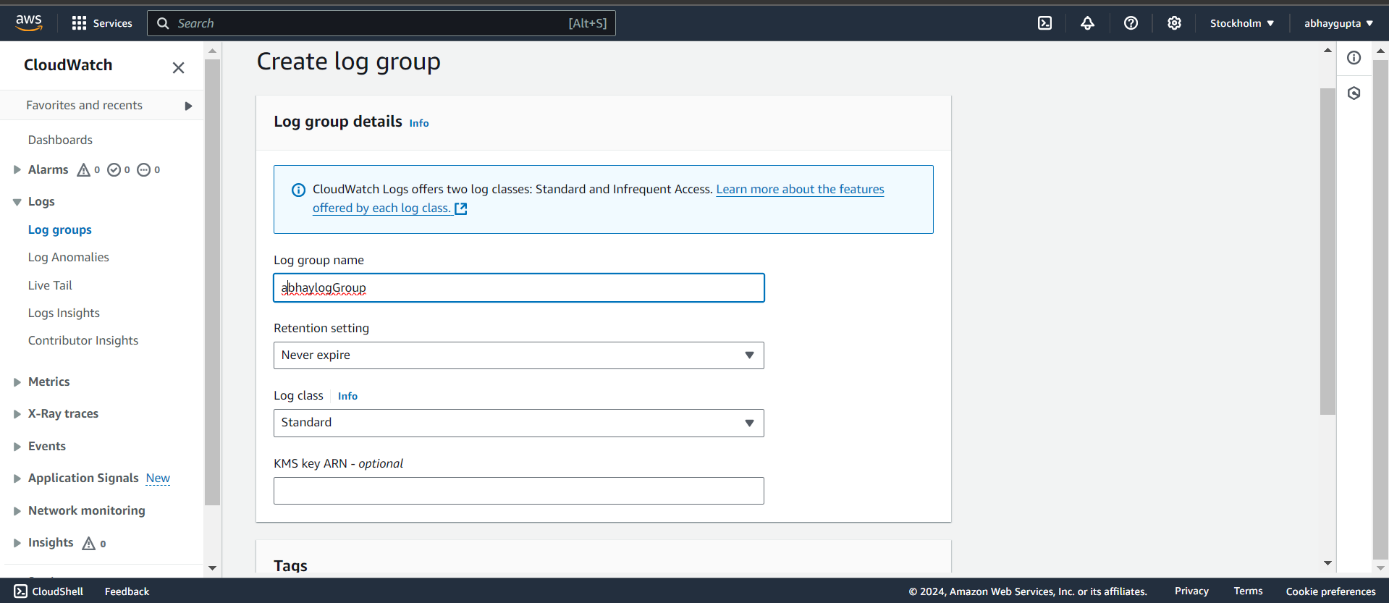
* Click **Create Bucket**.
* Enter a **Bucket Name** (e.g., my-error-logs-bucket)



**Step 2: Create a CloudWatch Log Group**

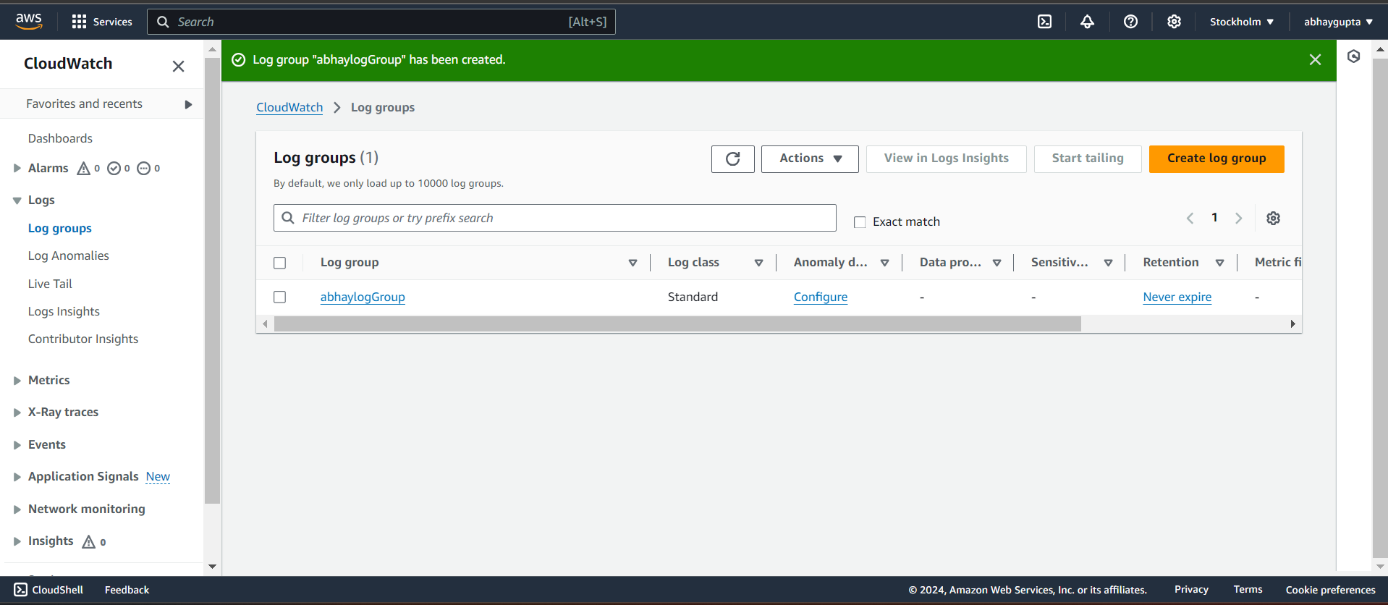
CloudWatch Logs will store the logs generated by your application, and Lambda will monitor this log group for new entries.

1. **Go to Logs:**



1. **Create a Log Group:**

* Click Create log group.
* Enter a name for the log group (e.g., MyLogGroup), then click Create.

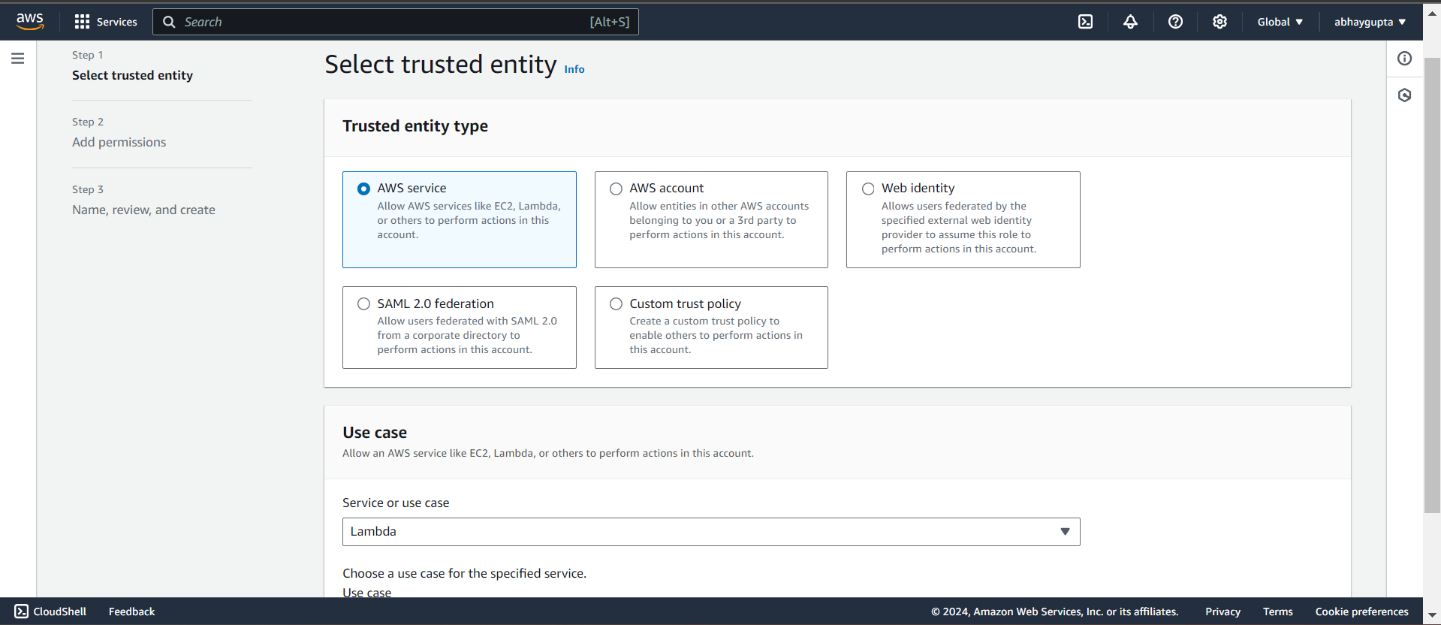


**Step 3: Create an IAM Role for Lambda**

This IAM role allows Lambda to access CloudWatch Logs and store logs in S3.

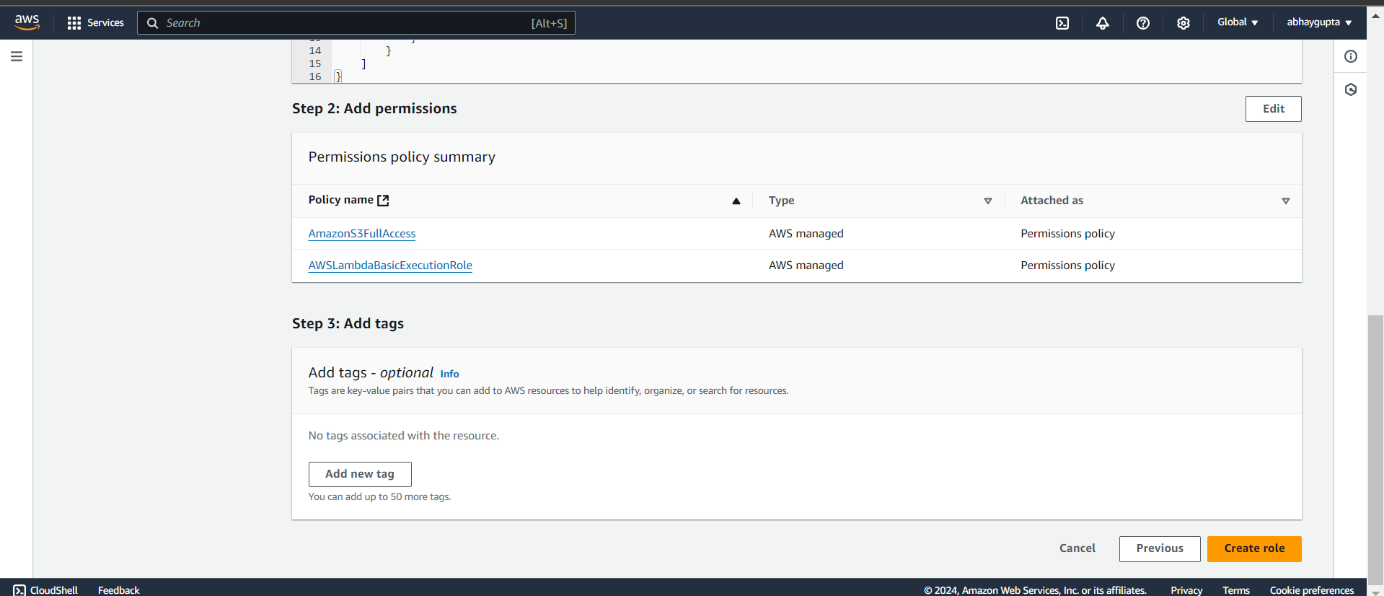
1. **Navigate to IAM**
2. **Create Role**
3. **Select Trusted Entity**:

* For **Trusted entity type**, choose **AWS Service**.
* Select **Lambda** as the service that will use this role.

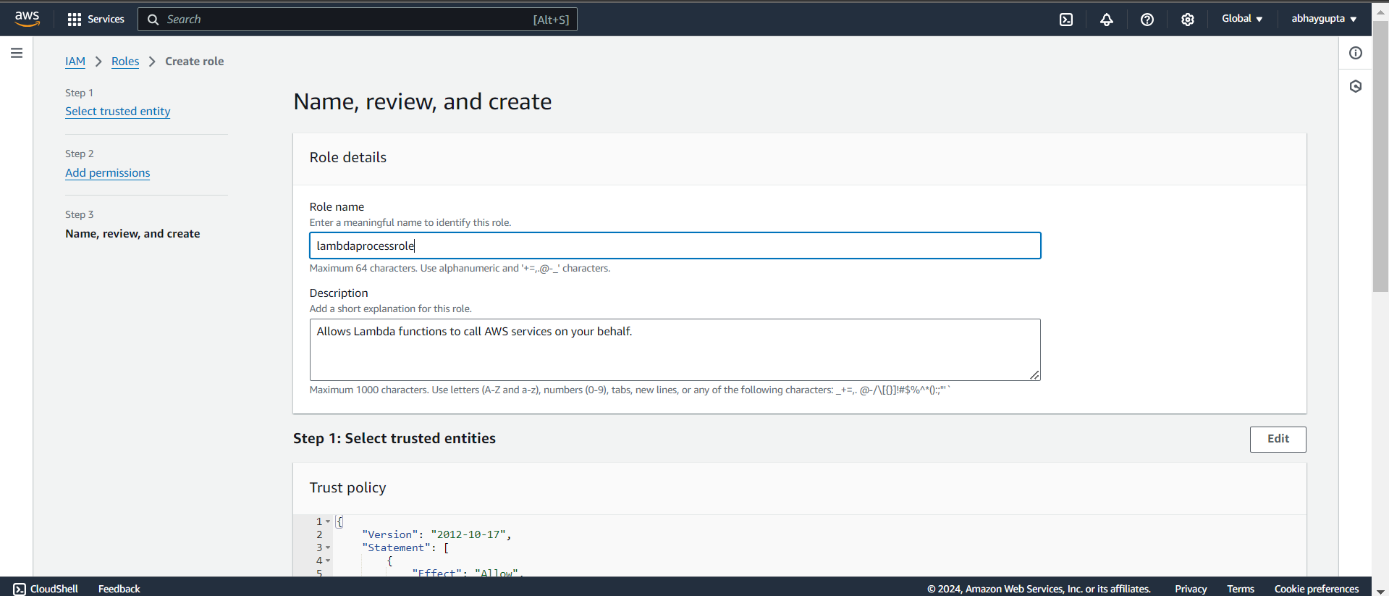


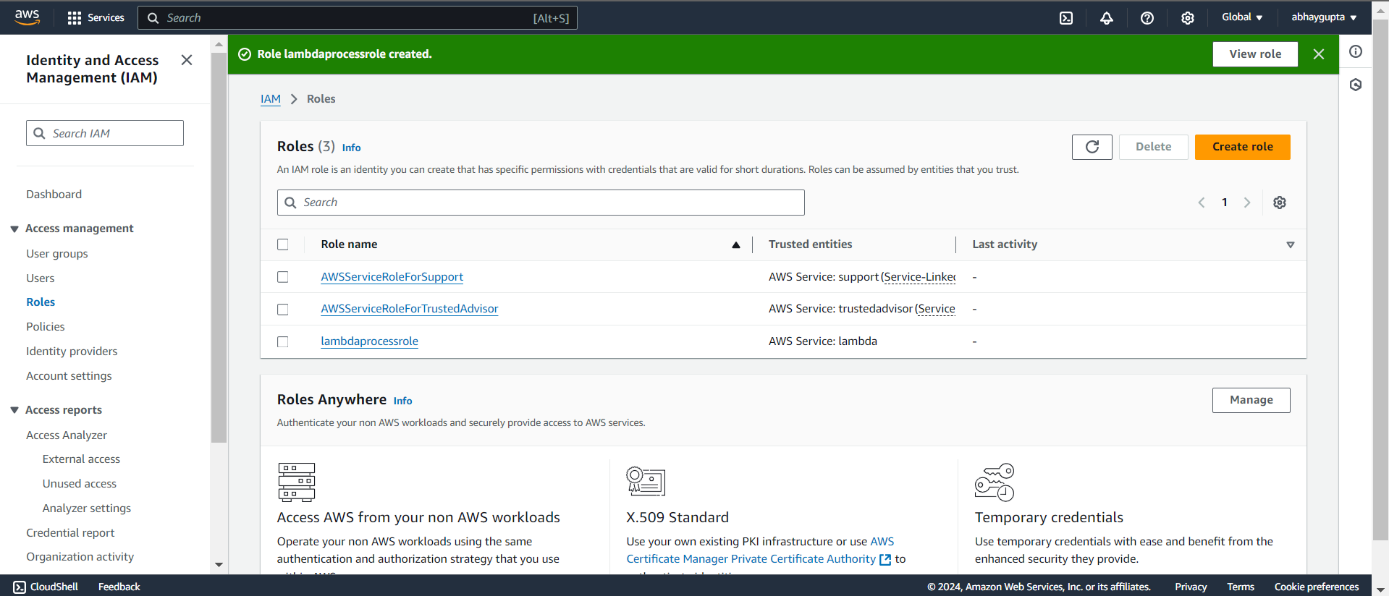
1. **Attach Policies:**

* Click **Next** and search for the following policies:
  + **AWSLambdaBasicExecutionRole** (provides access to CloudWatch Logs for Lambda).
  + **AmazonS3FullAccess** (gives full access to S3).
* Attach these policies by checking the boxes.



1. **Name the Role**:

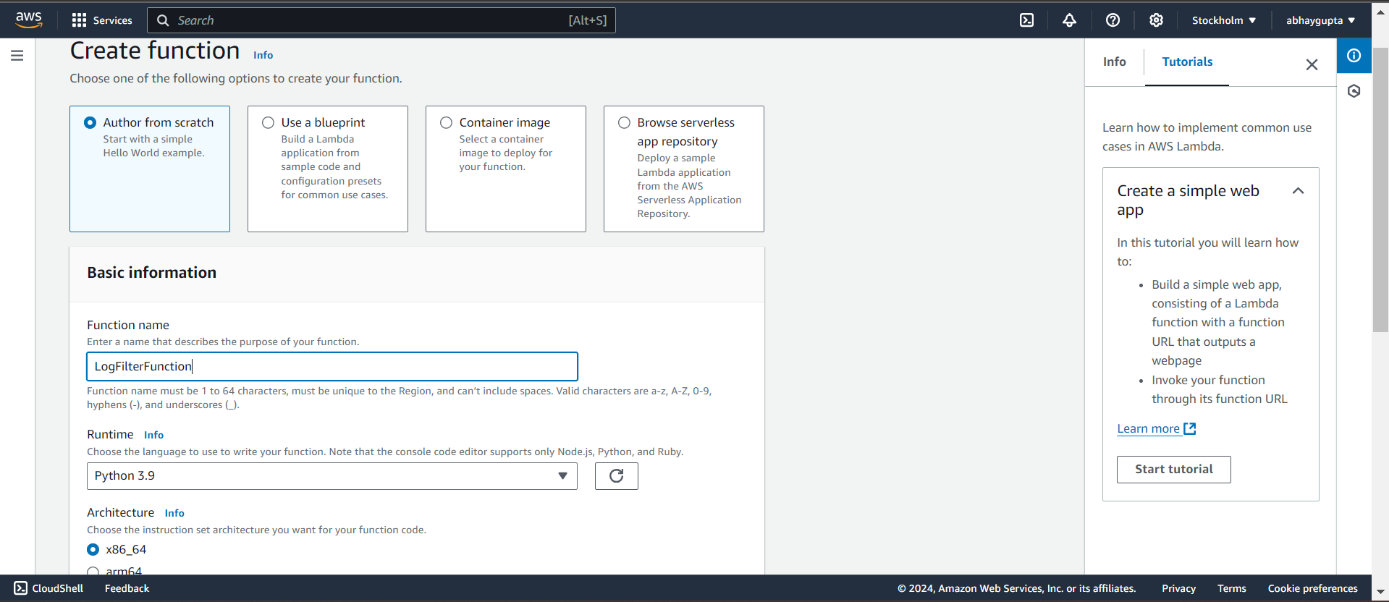




**Step 4: Create the Lambda Function**

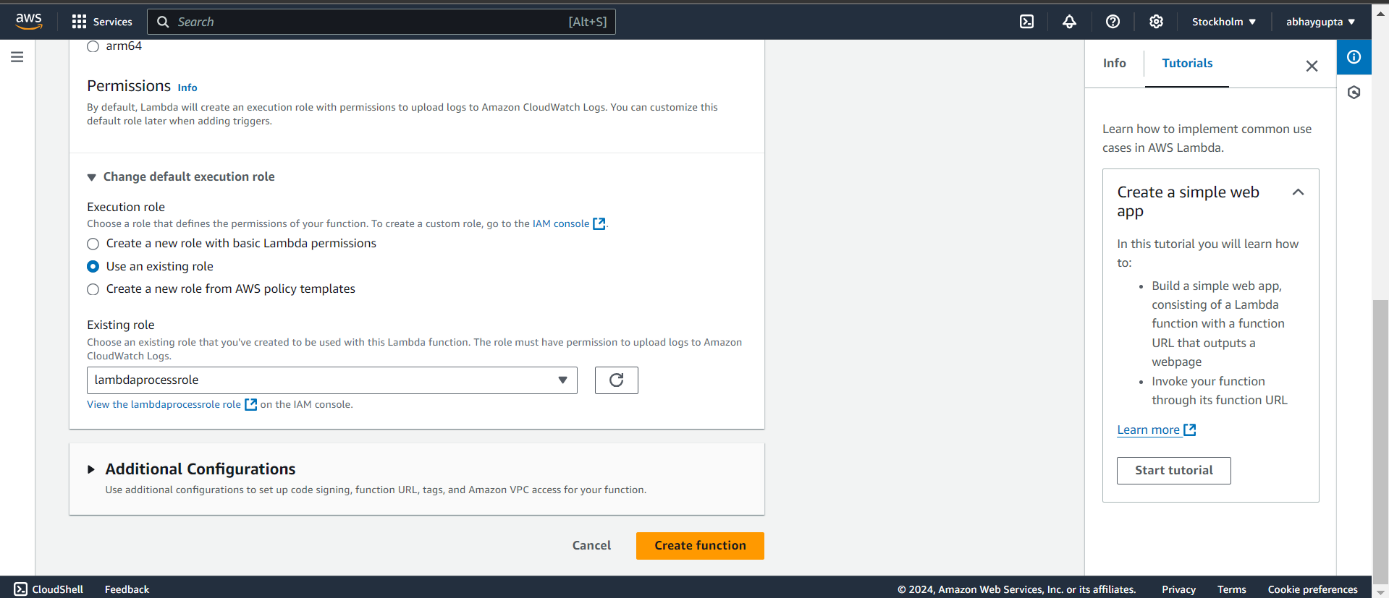
This Lambda function will be triggered when new log events are generated in CloudWatch Logs.

1. **Navigate to Lambda**:
   * In the AWS Management Console, type **Lambda** in the search bar and click on **Lambda**.
2. **Create a New Function**:
   * Click **Create function**.
   * Select **Author from scratch**.
   * **Function name**: Enter a name (e.g., LogFilterFunction).
   * **Runtime**: Choose **Python 3.x** (e.g., Python 3.9).

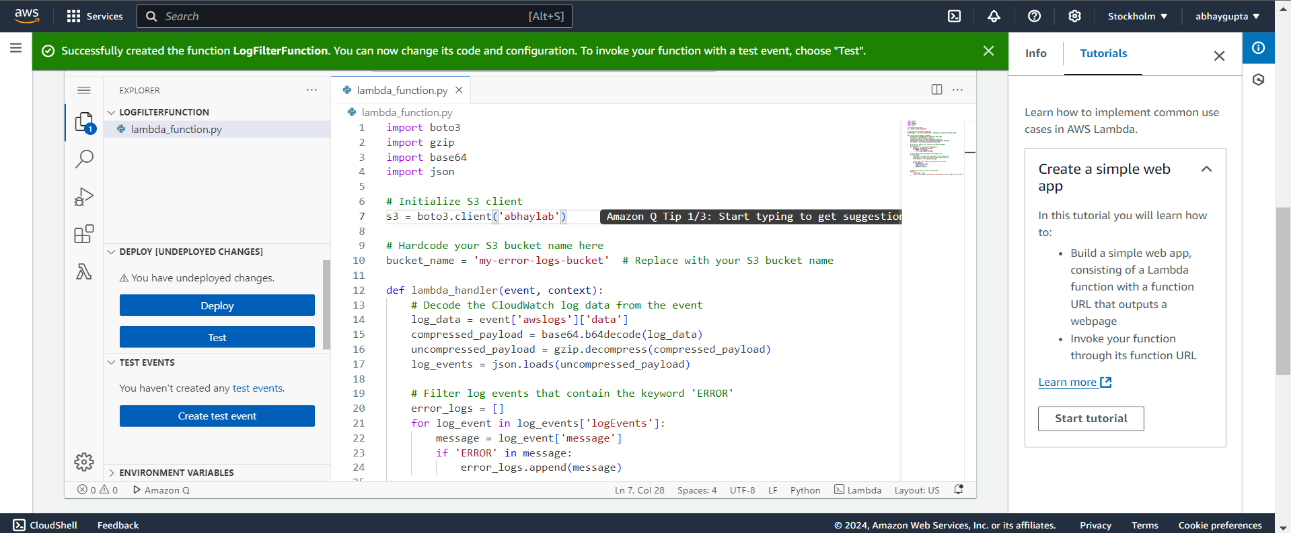


1. **Role**:

* Under Permissions, choose **Use an existing role** and select the role you created earlier (LambdaLogProcessorRole).

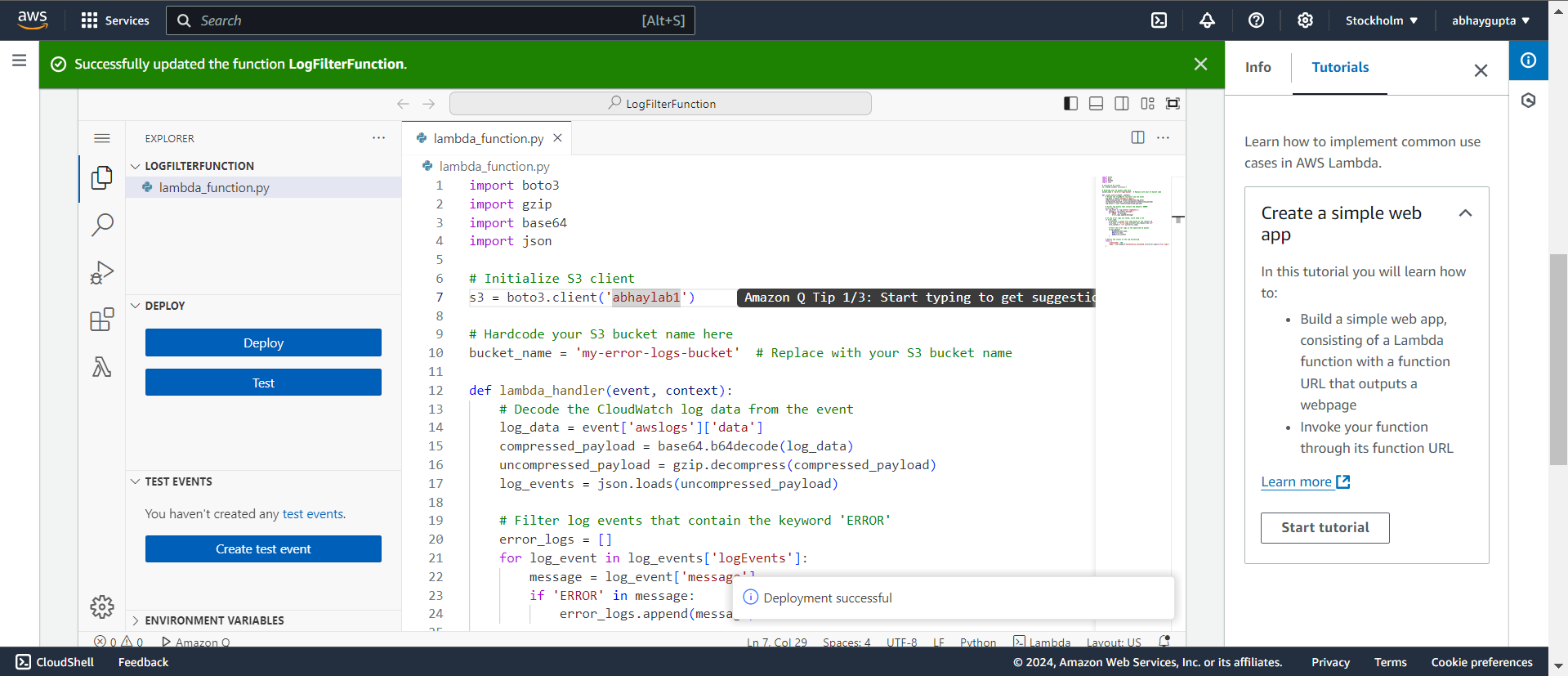


1. **Click Create Function**.



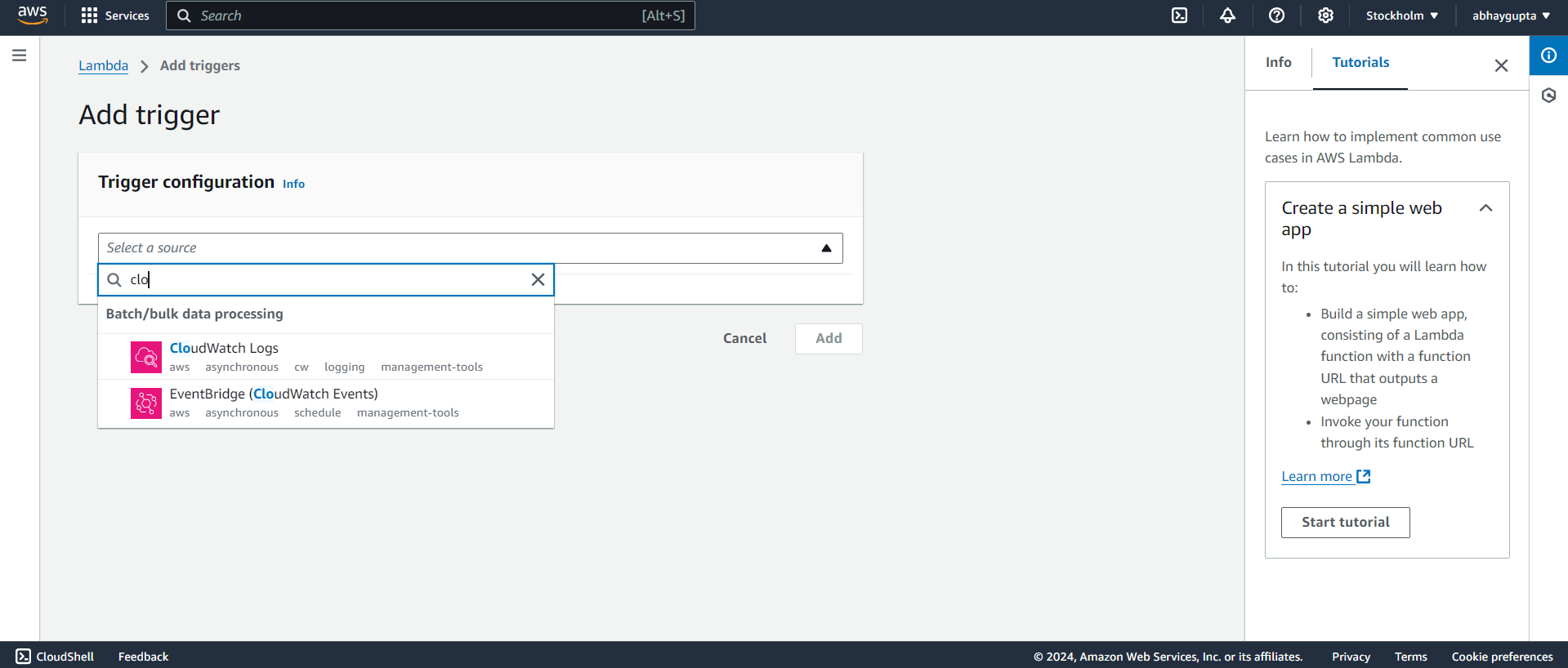
**Step 5: Write the Python Lambda Function**

1. **Scroll down to the "Function code" section** in the Lambda console.
2. **Replace the default code with the following code** (update the bucket name with your S3 bucket name):
3. **Click Deploy** to save your function.

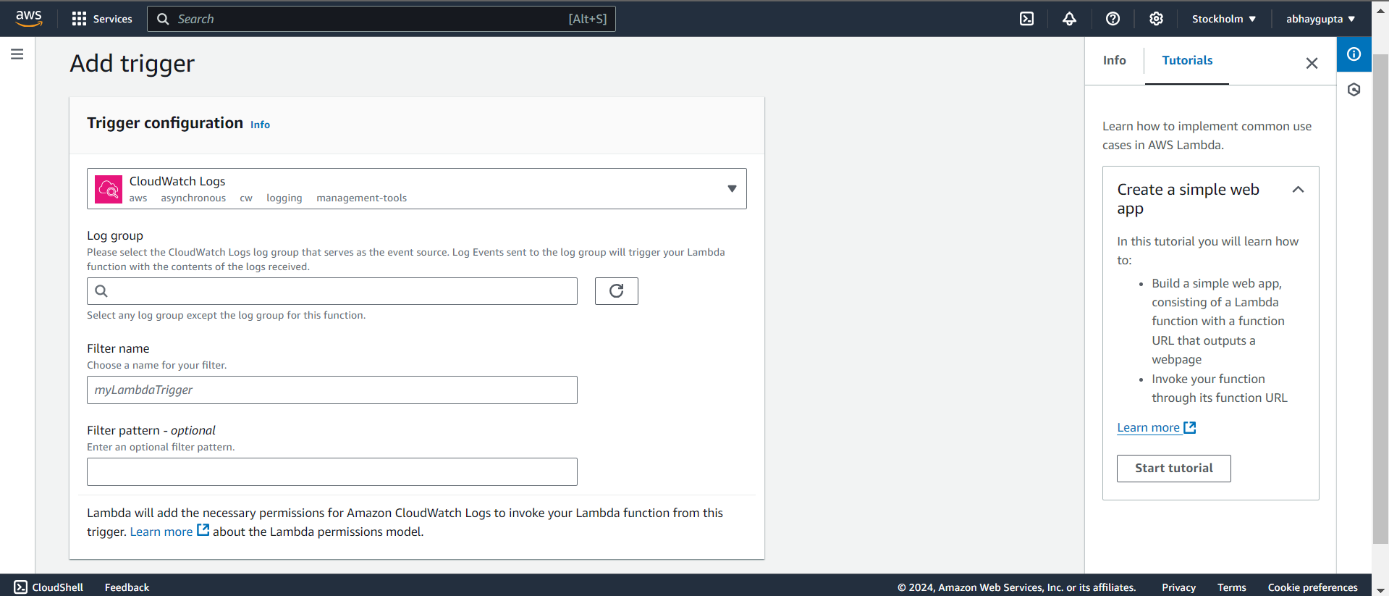


**Step 6: Set the Lambda Trigger (CloudWatch Logs)**

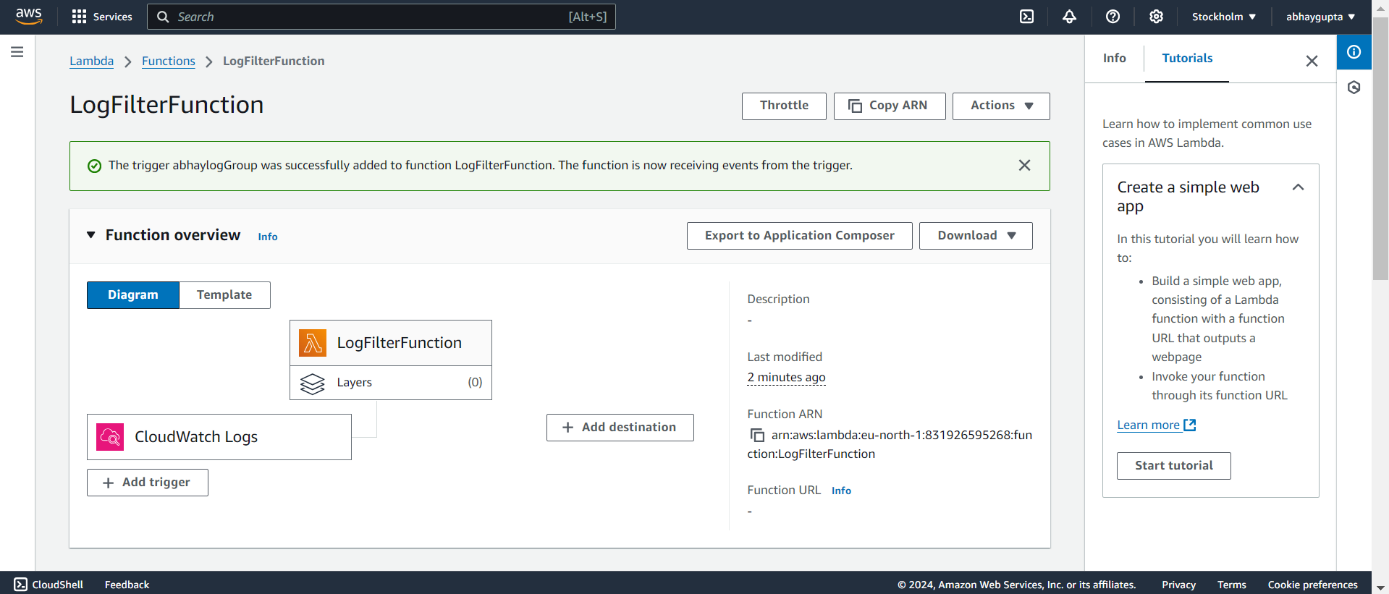
1. **Add a Trigger**:
   * Scroll up to **Function overview** and click **Add Trigger**.
2. **Select CloudWatch Logs**:
   * In the dropdown menu, select **CloudWatch Logs** as the trigger.



1. **Configure the Trigger**:
   * Choose the **Log Group** you created earlier (MyLogGroup).
   * In **Filter name**, you can provide a filter name (optional).



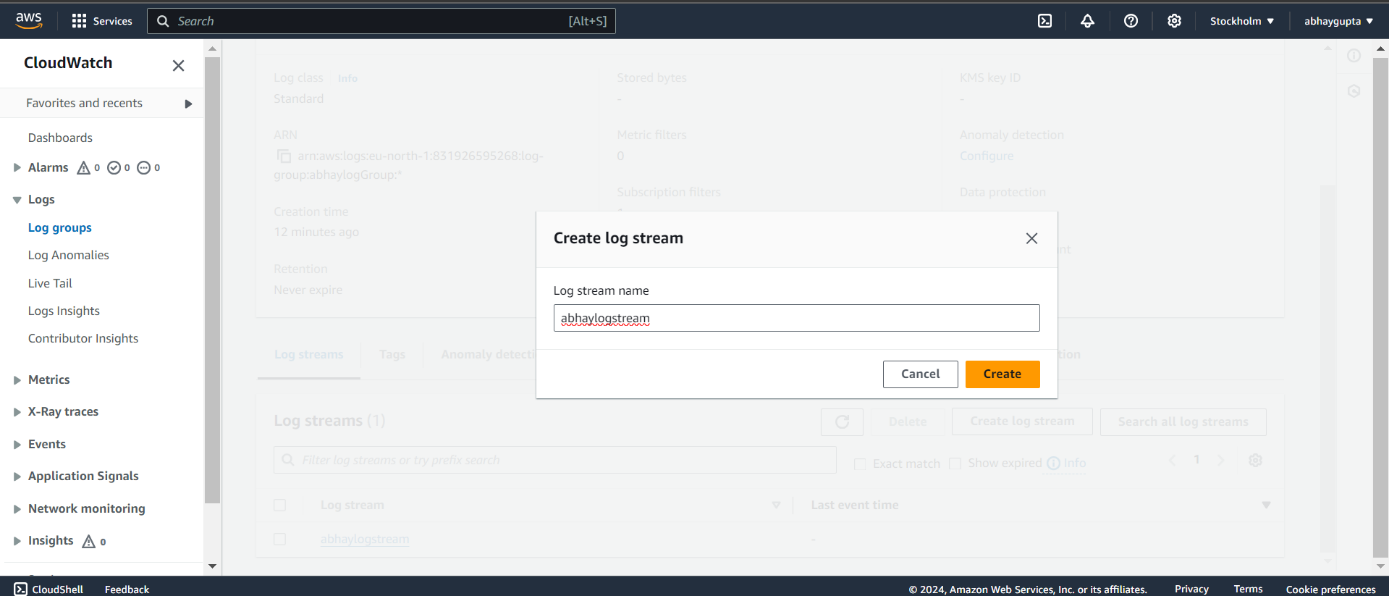
1. **Click Add** to set up the trigger.



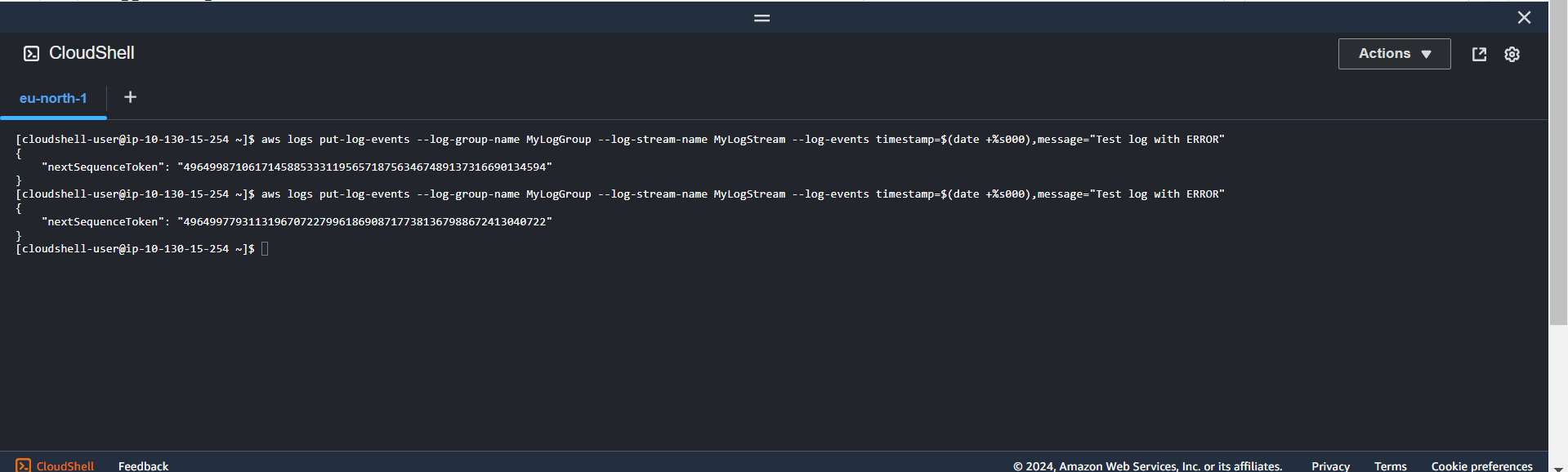
**Step 7: Test the Setup by Generating Logs**

Now, you need to generate logs with the keyword 'ERROR' to test your setup.

1. **Go to CloudWatch Logs**:
   * Navigate back to **CloudWatch** → **Logs**.
   * Click on your log group (MyLogGroup).
2. **Create a Log Stream** (if needed):
   * Click **Create log stream** and name it (e.g., MyLogStream).



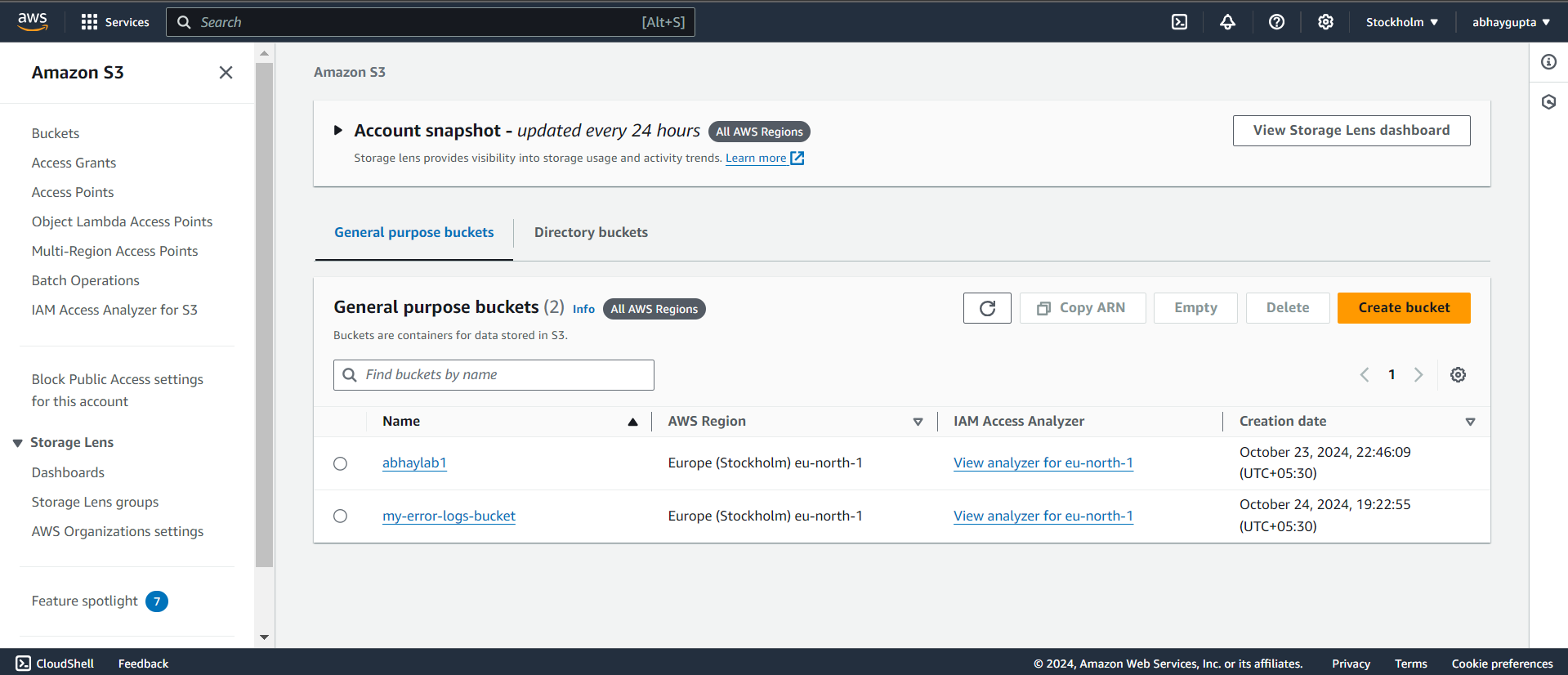
1. **Manually Add Logs** (or run an application that generates logs): You can manually generate logs using the AWS CLI or other methods.



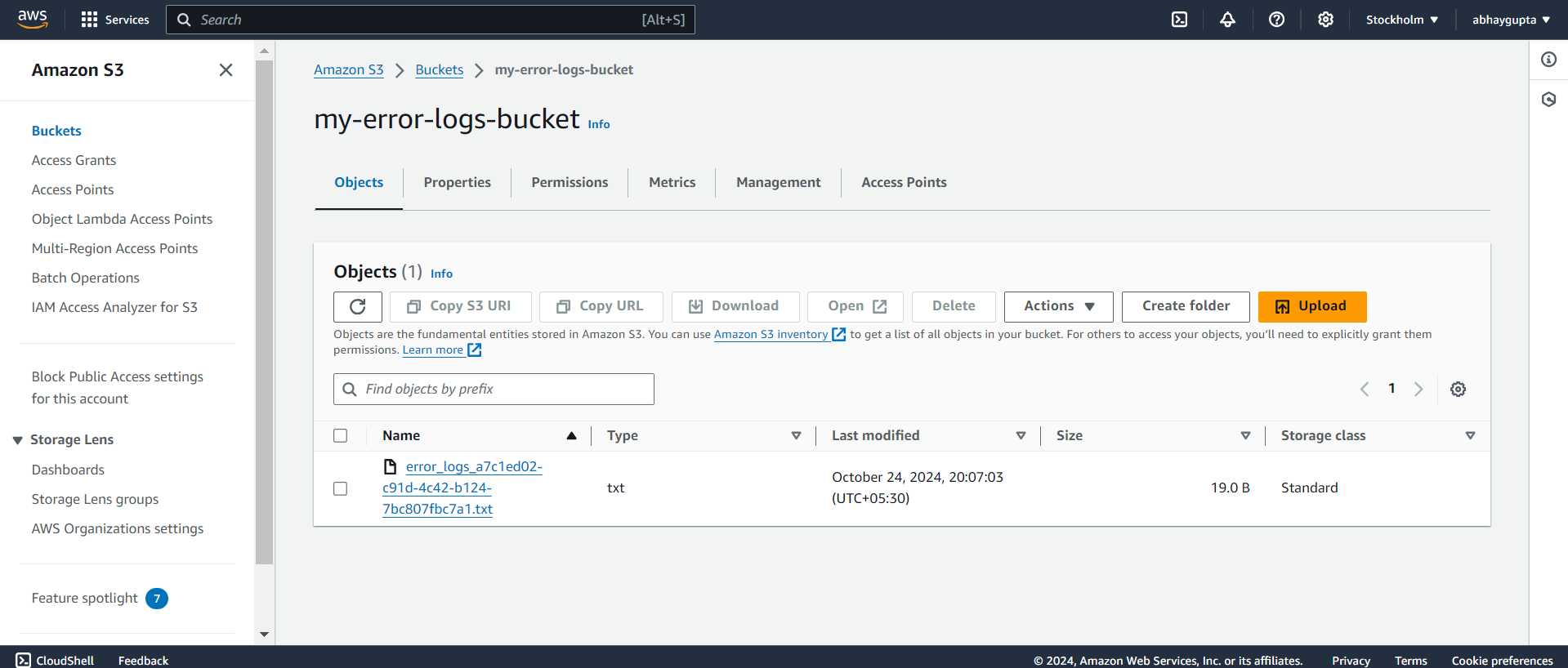
aws logs put-log-events --log-group-name MyLogGroup --log-stream-name MyLogStream --log-events timestamp=$(date +%s000),message="Test log with ERROR"

**Step 8: Verify the Logs in S3**

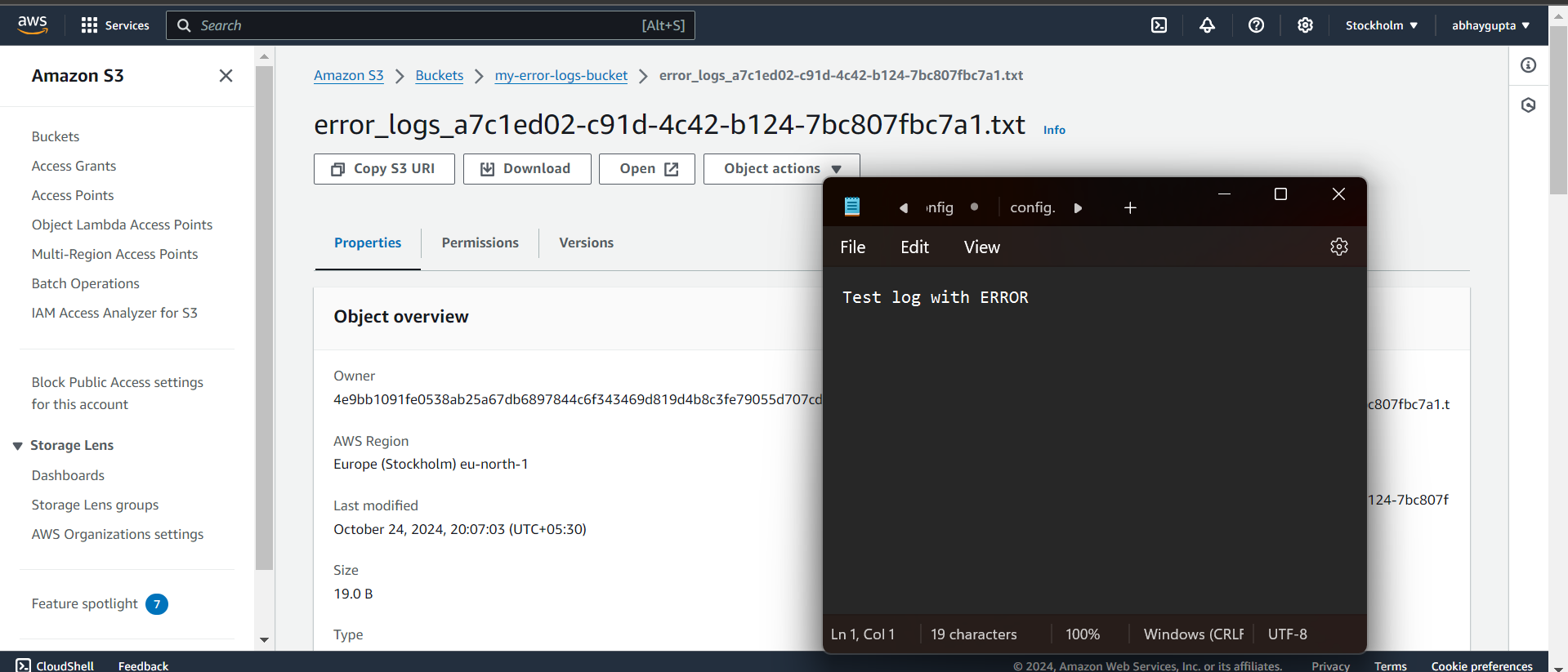
1. **Navigate to S3**:
   * Go back to **S3** in the AWS Console.



1. **Check Your S3 Bucket**:
   * Open your bucket (my-error-logs-bucket).
   * You should see a new file (e.g., error\_logs\_<request\_id>.txt).



1. **Open the File**:
   * Inside the file, you should see the logs that contain 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.