Vivekanand Education Society's Institute of Technology, Chembur, Mumbai,
Department of Technology,
Year: 2024-2025(ODD SEM)

Advance DevOps Practical Examination

Name: Roma Uday Shirodkar | Div: D15B
Roll No: 58 | Date of exam: 24/10/2024

Case Study 18:

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.

Aim: To set up an AWS Lambda function that triggers on new log entries in a CloudWatch Log Group, filters logs for specific keywords (e.g., 'ERROR'), and stores the filtered logs in an S3 bucket.

Theory:

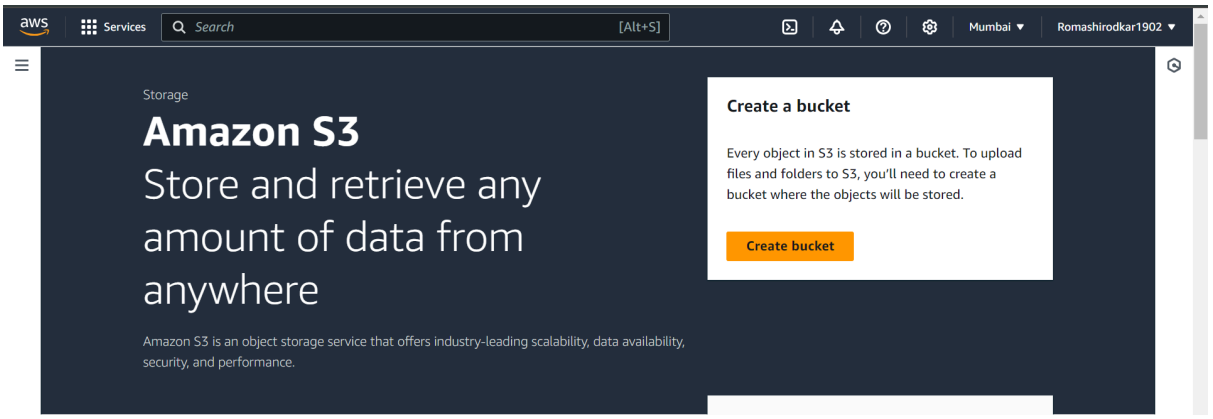
- **Amazon S3** (Simple Storage Service) is a scalable object storage service that allows users to store and retrieve data over the internet. S3 is designed for durability, availability, and performance, making it ideal for a variety of use cases, such as backup and restore, data archiving, big data analytics, and web hosting. *In this experiment, S3 will be used to store filtered log entries generated by the Lambda function.*
- **AWS Lambda** is a serverless compute service that enables users to run code without provisioning or managing servers. It automatically scales to handle incoming requests and only charges for the compute time used. Users can trigger Lambda functions in response to various AWS events, such as changes in data or system state.*In this experiment, a Lambda function will be created to automatically execute when new log entries are added to a CloudWatch Log Group. The function will filter the logs for specific keywords and store the relevant entries in the S3 bucket.*

- **Amazon CloudWatch** is a monitoring and observability service that provides real-time insights into AWS resources and applications. It collects and tracks metrics, logs, and events, allowing users to monitor system performance, troubleshoot issues, and automate responses to changes in resource states. *CloudWatch will be used to create a Log Group that collects log entries from specified sources. The Lambda function will be triggered whenever new log entries are added to this Log Group.*

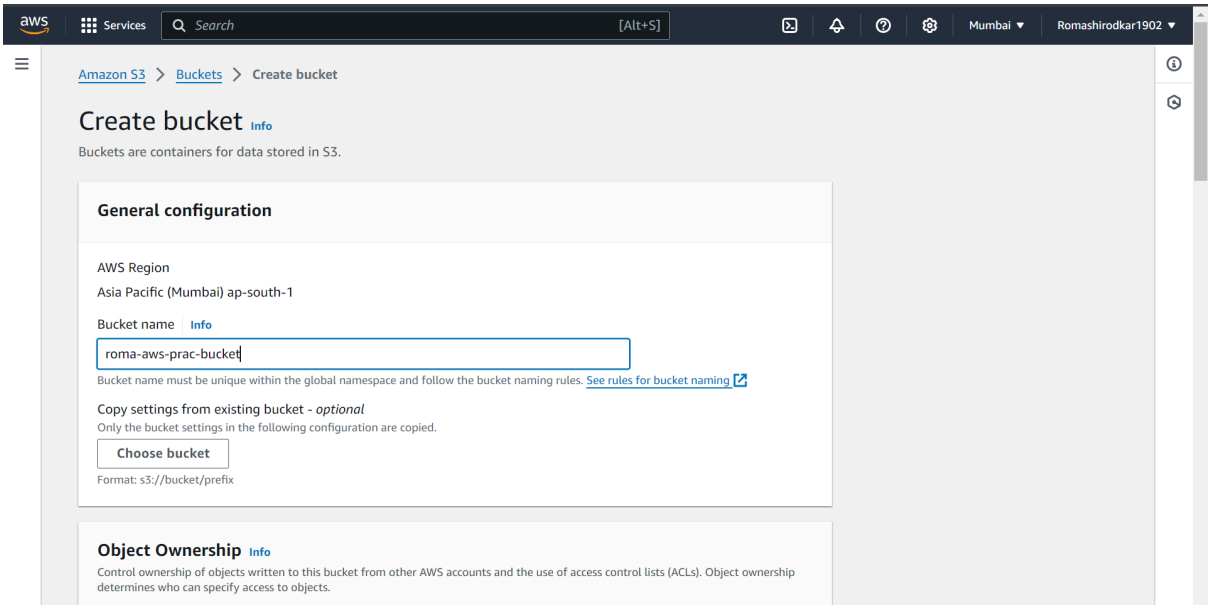
Step-by-Step Implementation:

Step 1: Create an S3 bucket.

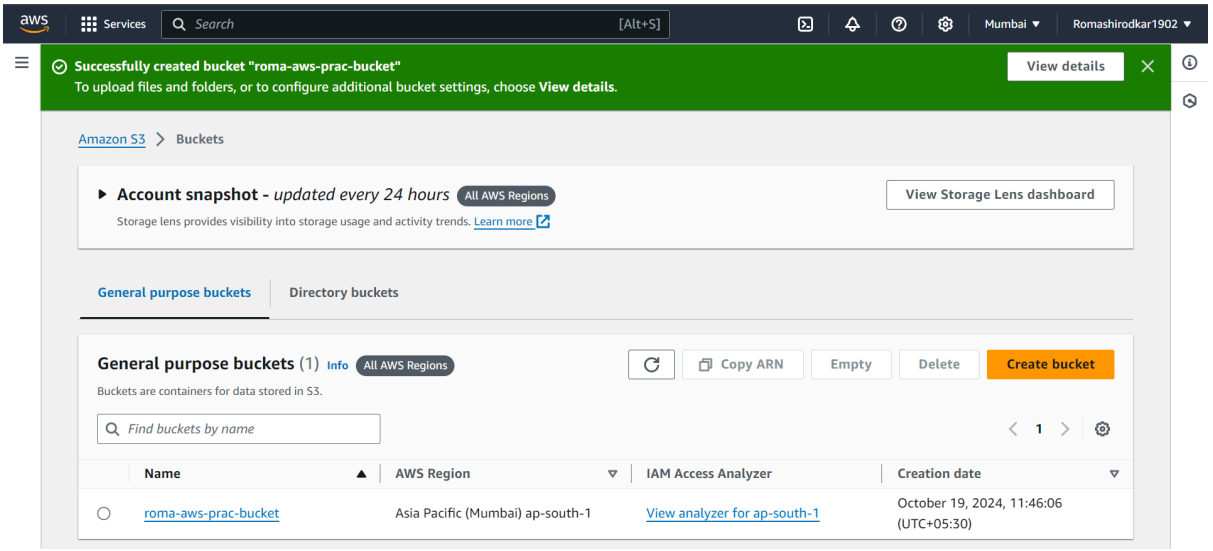
- Navigate to the **S3** service in the AWS Management Console, you will see the following page and then click on “Create bucket”.



- Name the bucket. Here I have named it as **roma-aws-prac-bucket.** Keep other options to default and click “create bucket”.

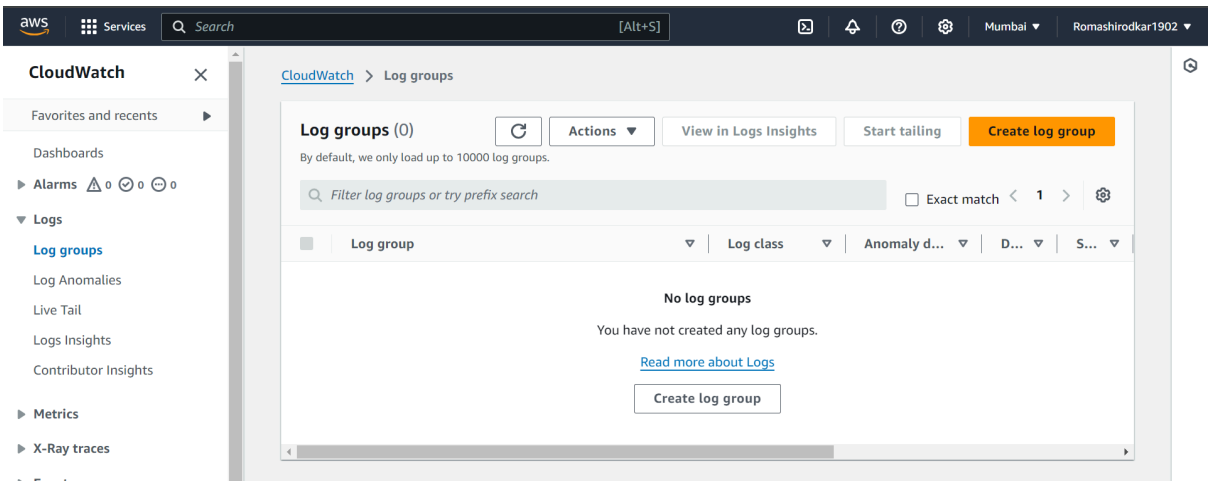


- As we can see here bucket “roma-aws-prac-bucket” has been created.

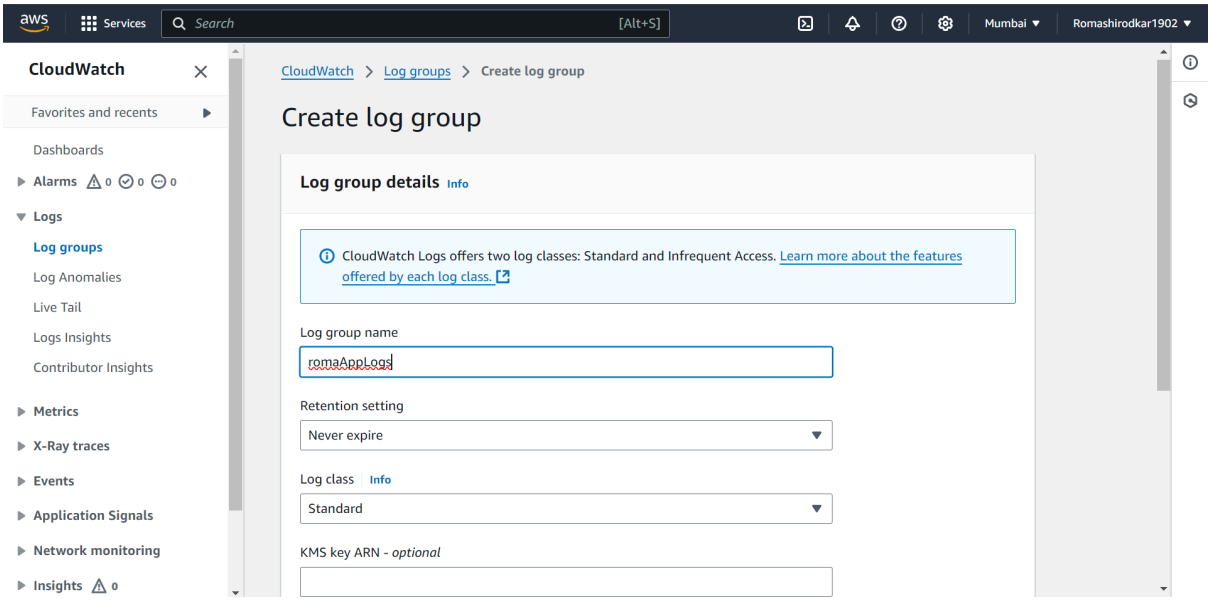


Step 2: Create a CloudWatch Log Group

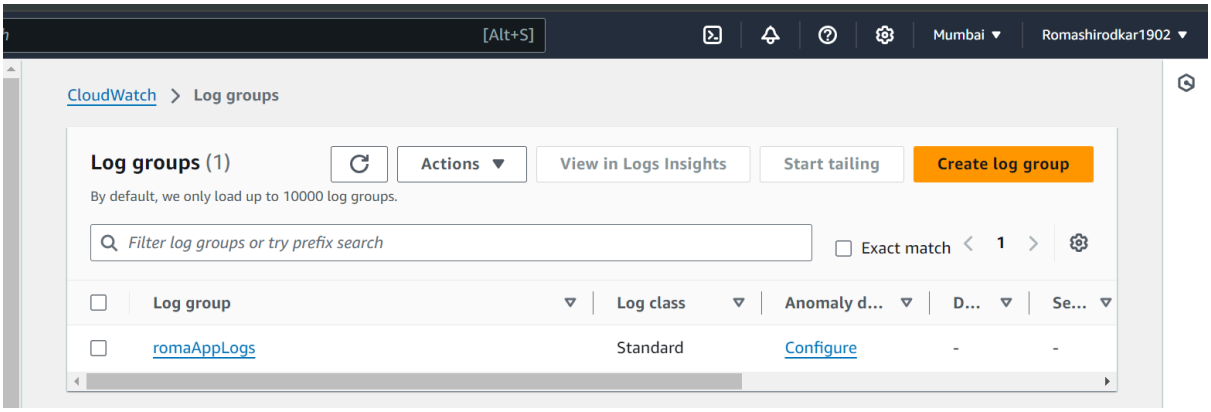
- In the AWS Console, go to CloudWatch.
- On the left panel, click on **Logs → Log groups**.
- Click **Create log group**.



- Provide a name for the log group. Here I have named my log group as **romaAppLogs.** Keep everything else as default and click “Create”.

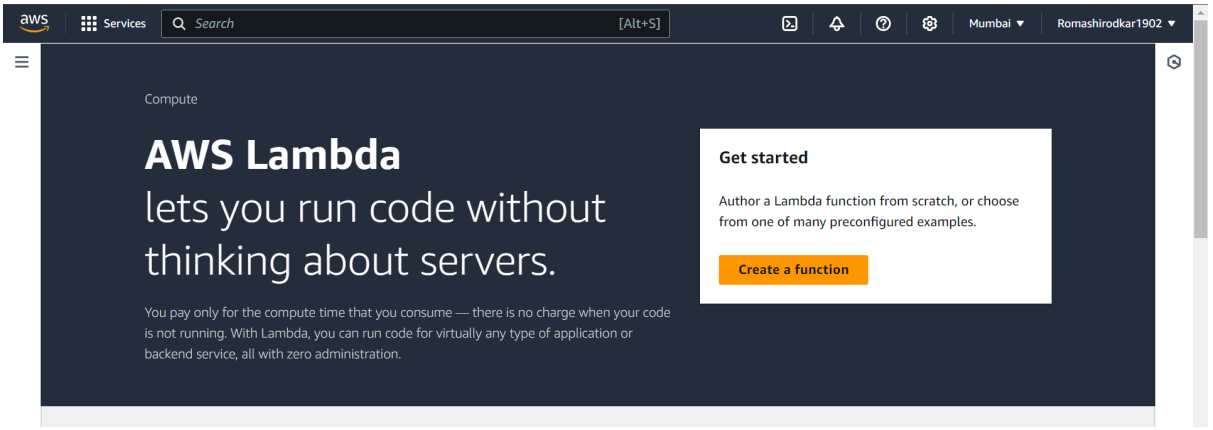


- As we can see below, a log group named romaAppLogs has been created.
- All the logs which I will generate in the further steps will be stored in this log group.



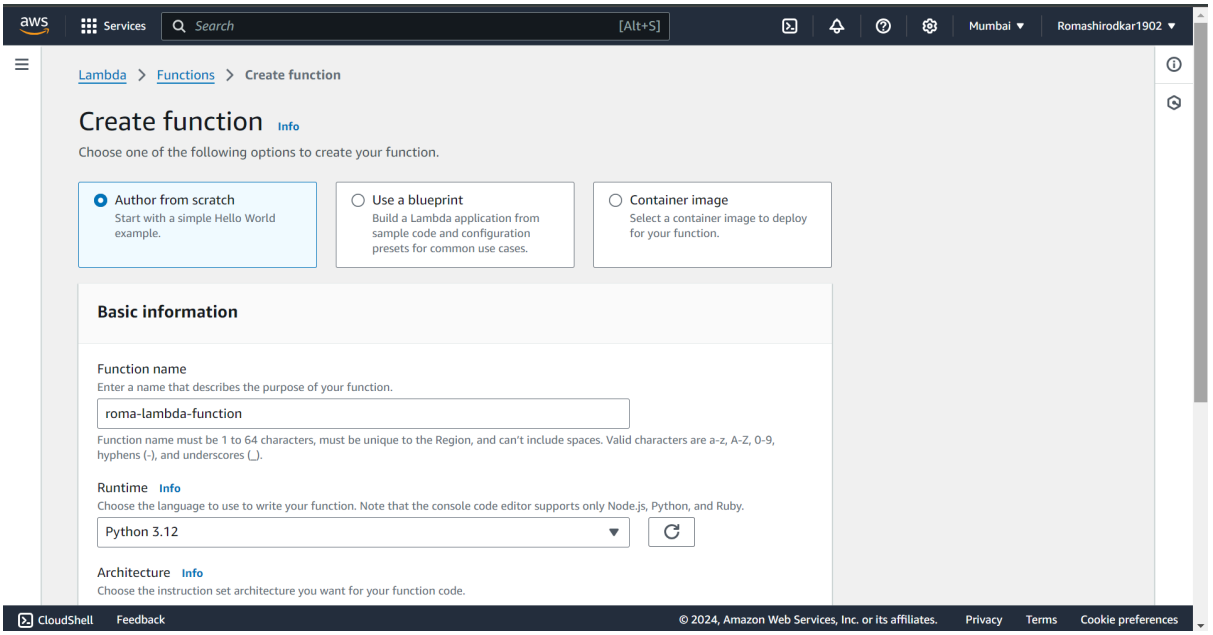
Step 3: Create a Lambda Function

- This is the lambda function which will be triggered every time new logs are generated in “romaAppLogs” log group.
- Go to **AWS Lambda**.
- Click on “Create a function”.

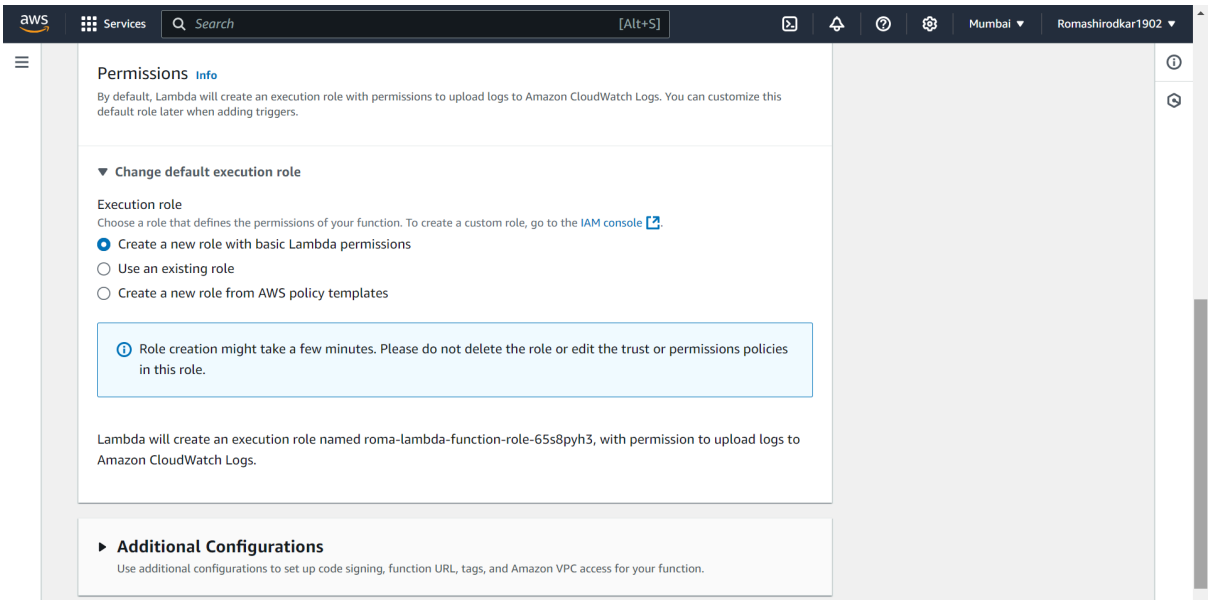


You will see the following page.

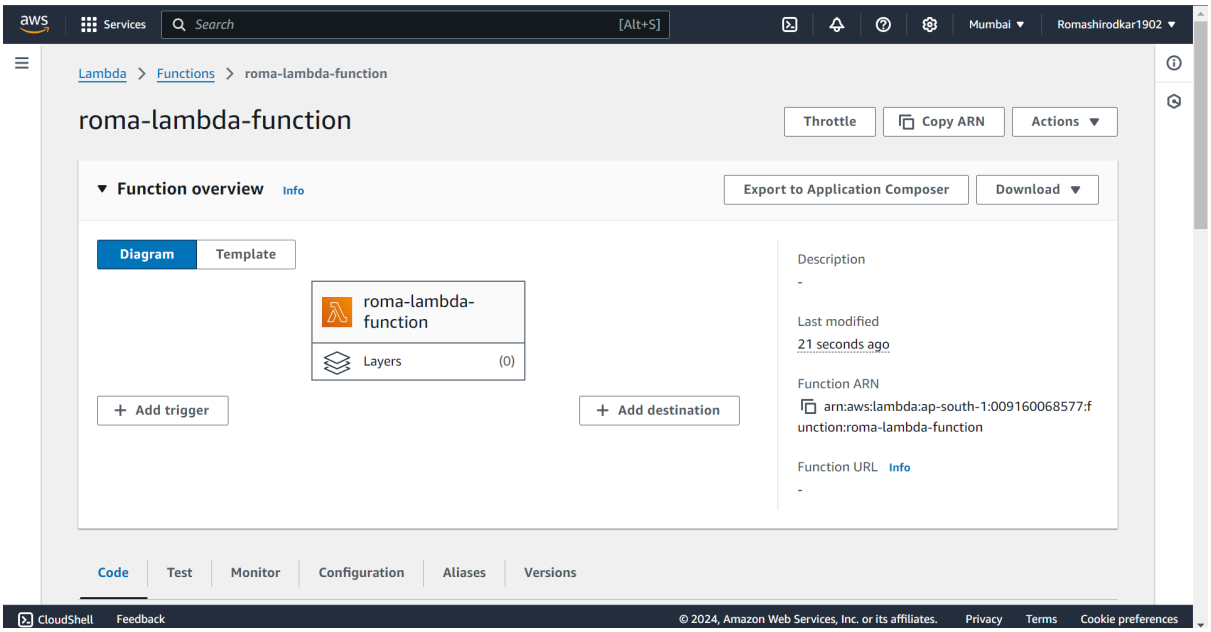
- Select “Author from scratch”.
- Provide a name to the function. Here I have named my function as **roma-lambda-function.**
- Select “Python 3.12” as Runtime.



- Scroll down. You will see the permissions tab open it and make sure “Create a new role with basic Lambda permissions” is selected. This will automatically create a role with permissions to upload logs to Amazon CloudWatch Logs.



This is the Lambda function which I created.



Step 4: Write Python Code for Lambda Function

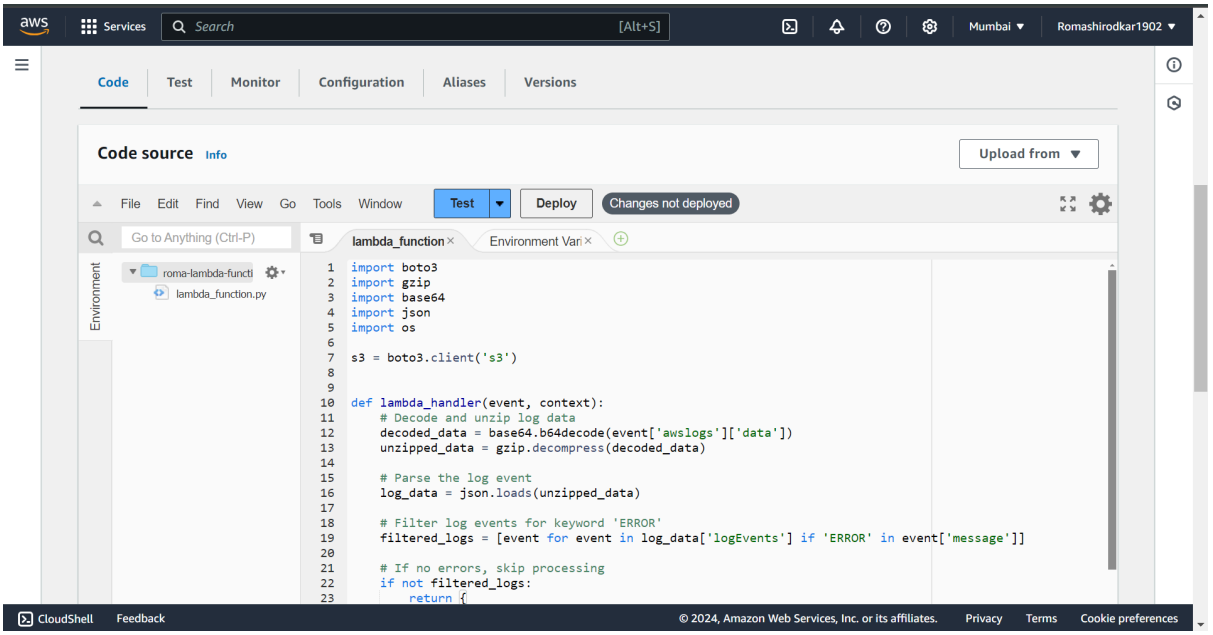
- Add this code in the code tab of the lambda function.
- This AWS Lambda function processes log data from a CloudWatch Log Group. When invoked, it decodes and decompresses the log data, then filters the log events for entries containing the keyword "ERROR."
- If any such entries are found, they are converted to a JSON string and uploaded to a specified S3 bucket. The function logs its operations, including received events and processing outcomes, for debugging purposes.
- If no error logs are found, it returns a message indicating that there were no errors.

CODE:

```
import boto3
import gzip
import base64
import json
import os

s3 = boto3.client('s3')
bucket_name = 'roma-aws-prac-bucket'  # Use your actual S3 bucket name

def lambda_handler(event, context):
    print(f"Received event: {event}")

    if 'awslogs' in event:
        try:
            # Decode and unzip log data
            decoded_data = base64.b64decode(event['awslogs']['data'])
            unzipped_data = gzip.decompress(decoded_data)

            # Parse the log event
            log_data = json.loads(unzipped_data)

            # Filter log events for keyword 'ERROR'
            filtered_logs = [event for event in log_data['logEvents'] if 'ERROR' in event['message']]

            # If no errors, skip processing
            if not filtered_logs:
                print("No ERROR logs found.")
                return {
                    'statusCode': 200,
                    'body': json.dumps('No errors found')
                }

            # Convert filtered logs to string
            filtered_logs_str = json.dumps(filtered_logs)

            # Define the S3 bucket and object key
            object_key = f"filtered_logs_{context.aws_request_id}.json"
```

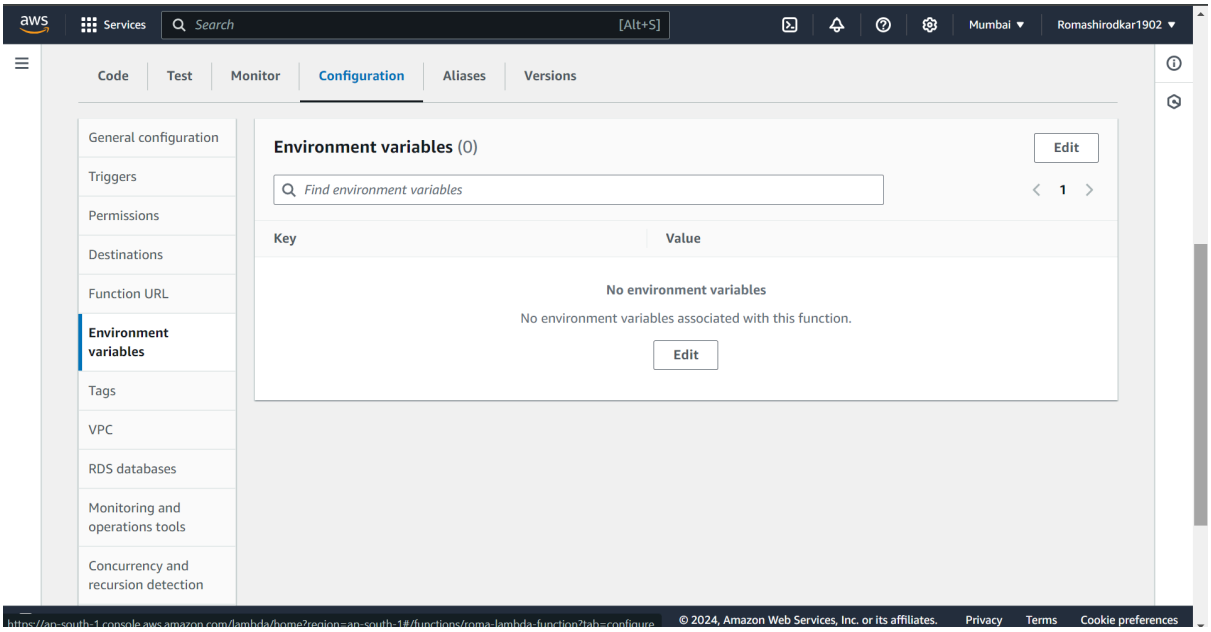
```
        # Upload filtered logs to S3
        s3.put_object(Bucket=bucket_name, Key=object_key, Body=filtered_logs_str)
        print(f"Filtered logs uploaded to S3: {object_key}")

        return {
            'statusCode': 200,
            'body': json.dumps('Filtered logs uploaded to S3')
        }

    except Exception as e:
        print(f"Error processing log data: {e}")
        return {
            'statusCode': 500,
            'body': json.dumps('Error processing log data')
        }
else:
    print("No awslogs found in event.")
    return {
        'statusCode': 400,
        'body': json.dumps('Invalid event structure')
    }
```

I have added the above code in the lambda_function.py file

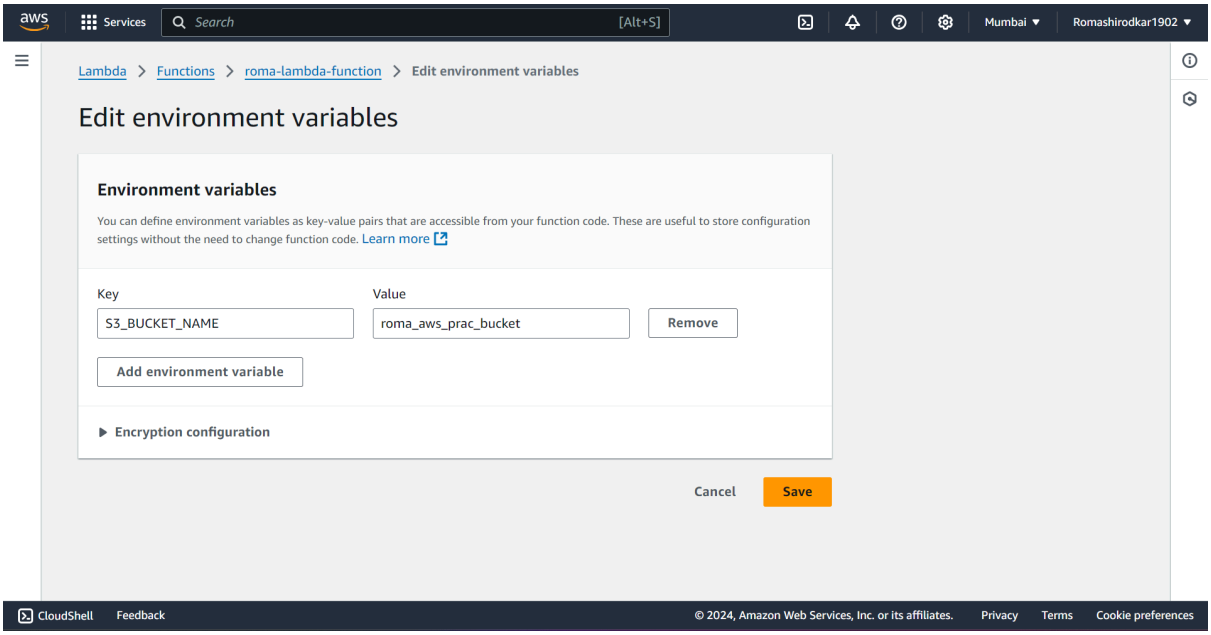


- The code provided contains one environment variable, S3_BUCKET_NAME, which is intended to hold the name of the S3 bucket where the filtered logs will be stored.

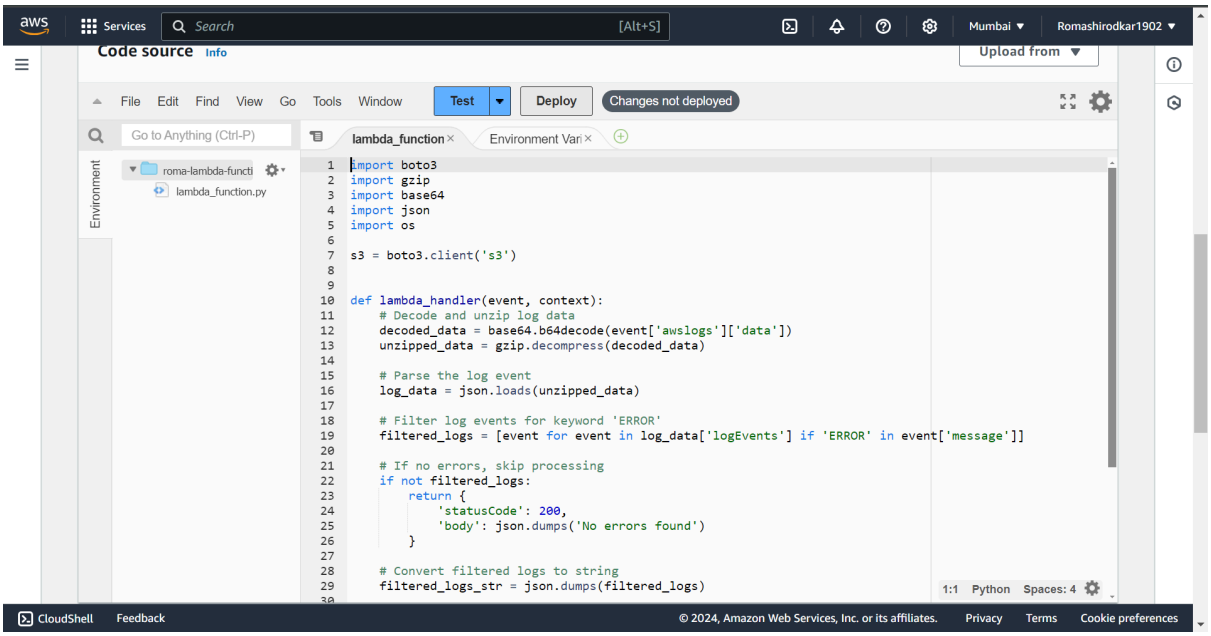
- Go to **Configuration → Environment Variables**
- Click on Edit



- In key, type “S3_BUCKET_NAME”
- In value, enter the s3 bucket name. I have added my s3 bucket’s name i.e. **Roma_aws_prac_bucket** and then save it.

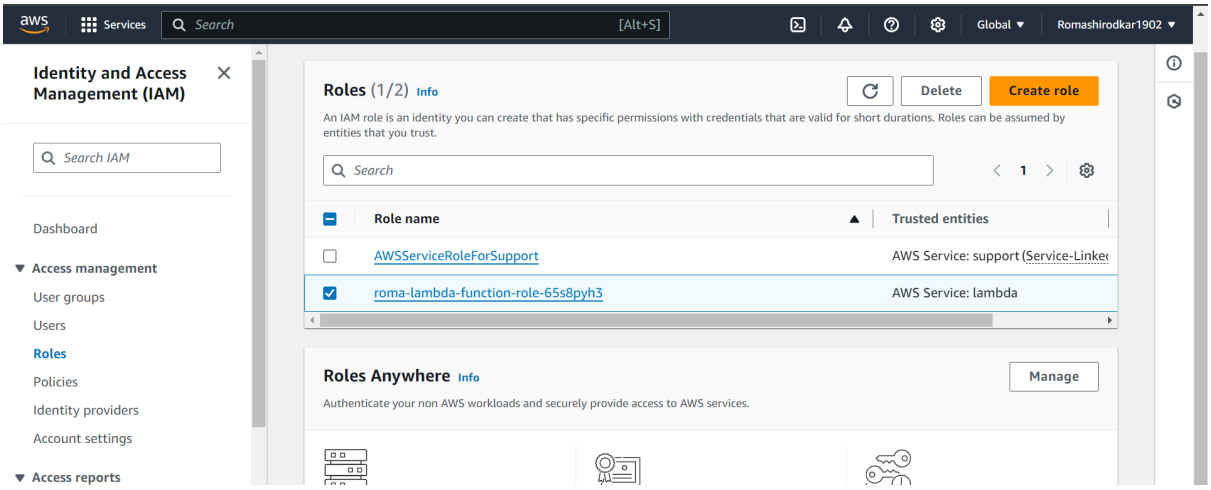


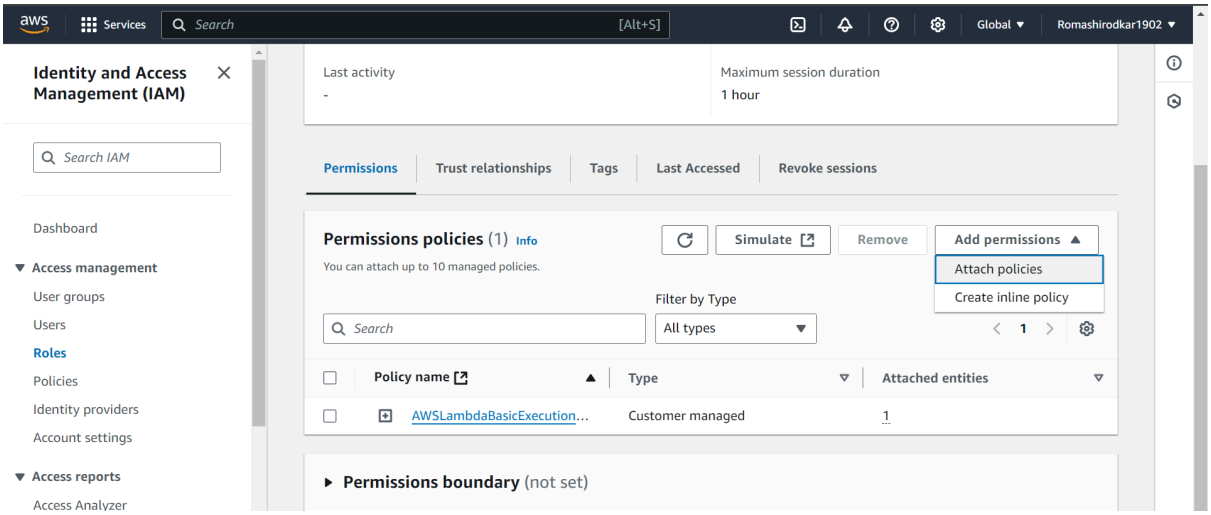
- Save your function (ctrl+S) and click Deploy.

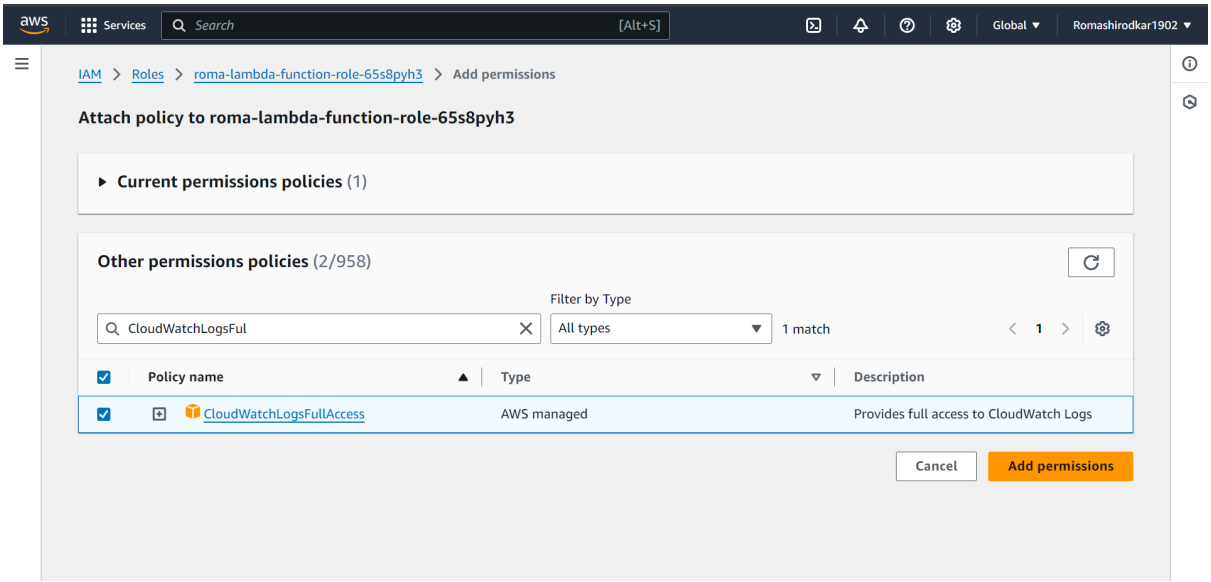


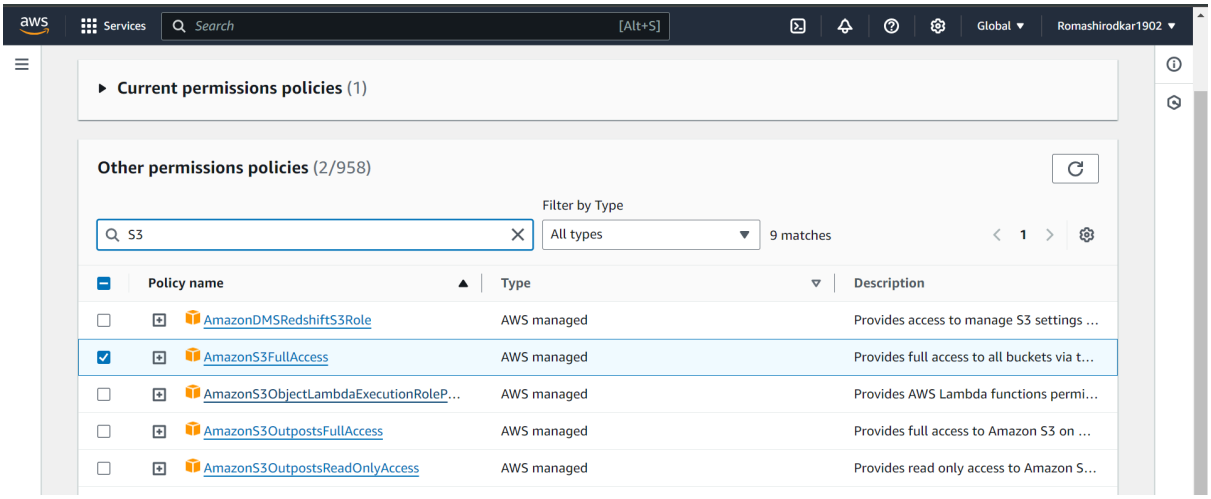
Step 5: Set Permissions for Lambda
Go to IAM in the AWS Console.
Find the role created for your Lambda function.
Attach the following managed policies:

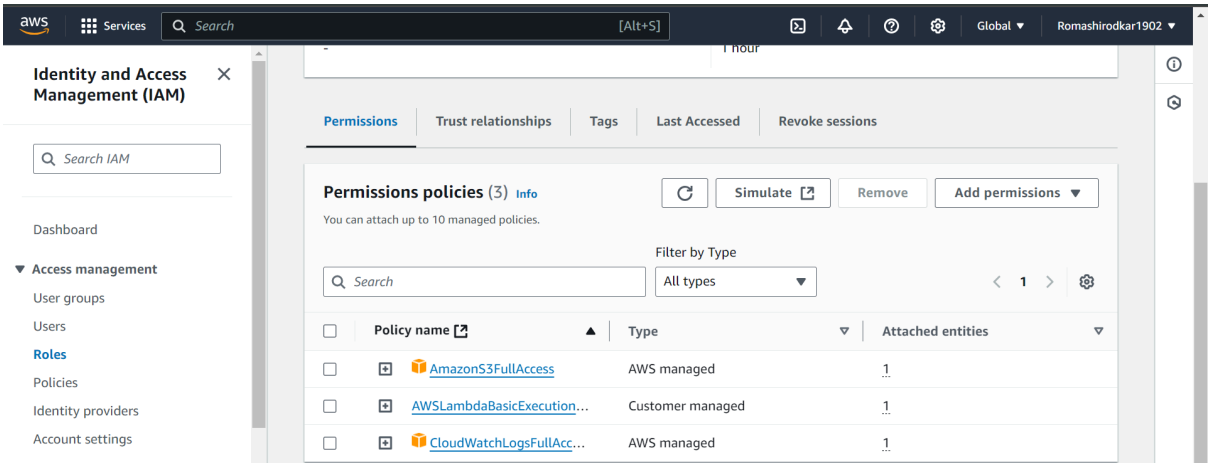
- AWSLambdaBasicExecutionRole
- CloudWatchLogsFullAccess
- AmazonS3FullAccess (or a custom S3 policy with write access to the specific bucket).




aws
Services
Search
[Alt+S]
Global
Romashirodkar1902
Identity and Access Management (IAM)
Search IAM
Dashboard
Access management
User groups
Users
Roles
Policies
Identity providers
Account settings
Access reports
Access Analyzer
Last activity
-
Maximum session duration
1 hour
Permissions
Trust relationships
Tags
Last Accessed
Revoke sessions
Permissions policies (1) Info
You can attach up to 10 managed policies.
Simulate
Remove
Add permissions
Attach policies
Create inline policy
Filter by Type
Search
All types
Policy name
Type
Attached entities
AWSLambdaBasicExecution...
Customer managed
1
Permissions boundary (not set)

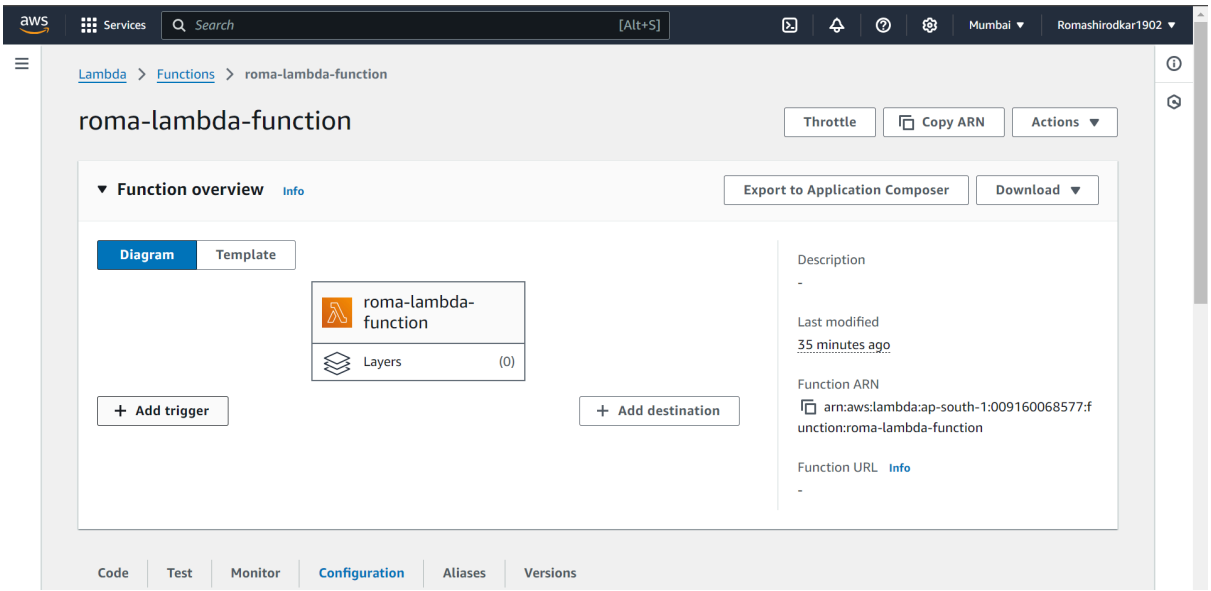

aws
Services
Search
[Alt+S]
Global
Romashirodkar1902
IAM > Roles > roma-lambda-function-role-65s8pyh3 > Add permissions
Attach policy to roma-lambda-function-role-65s8pyh3
Current permissions policies (1)
Other permissions policies (2/958)
Filter by Type
CloudWatchLogsFul
All types
1 match
Policy name
Type
Description
CloudWatchLogsFullAccess
AWS managed
Provides full access to CloudWatch Logs
Cancel
Add permissions


aws
Services
Search
[Alt+S]
Global
Romashirodkar1902
Current permissions policies (1)
Other permissions policies (2/958)
Filter by Type
S3
All types
9 matches
Policy name
Type
Description
AmazonDMSRedshiftS3Role
AWS managed
Provides access to manage S3 settings ...
AmazonS3FullAccess
AWS managed
Provides full access to all buckets via t...
AmazonS3ObjectLambdaExecutionRoleP...
AWS managed
Provides AWS Lambda functions permi...
AmazonS3OutpostsFullAccess
AWS managed
Provides full access to Amazon S3 on ...
AmazonS3OutpostsReadOnlyAccess
AWS managed
Provides read only access to Amazon S...

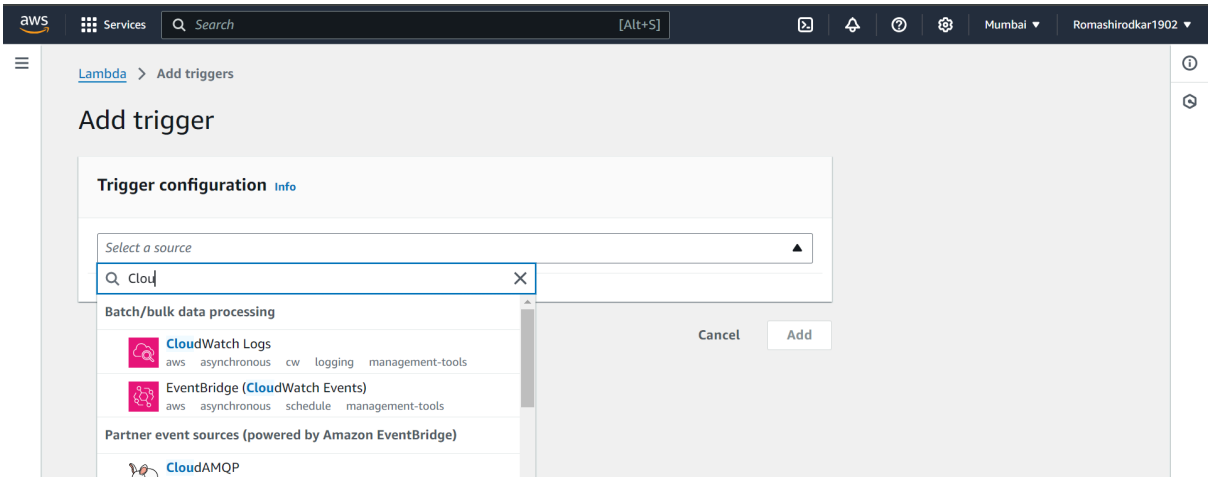


Step 6: Create a CloudWatch Log Group Trigger

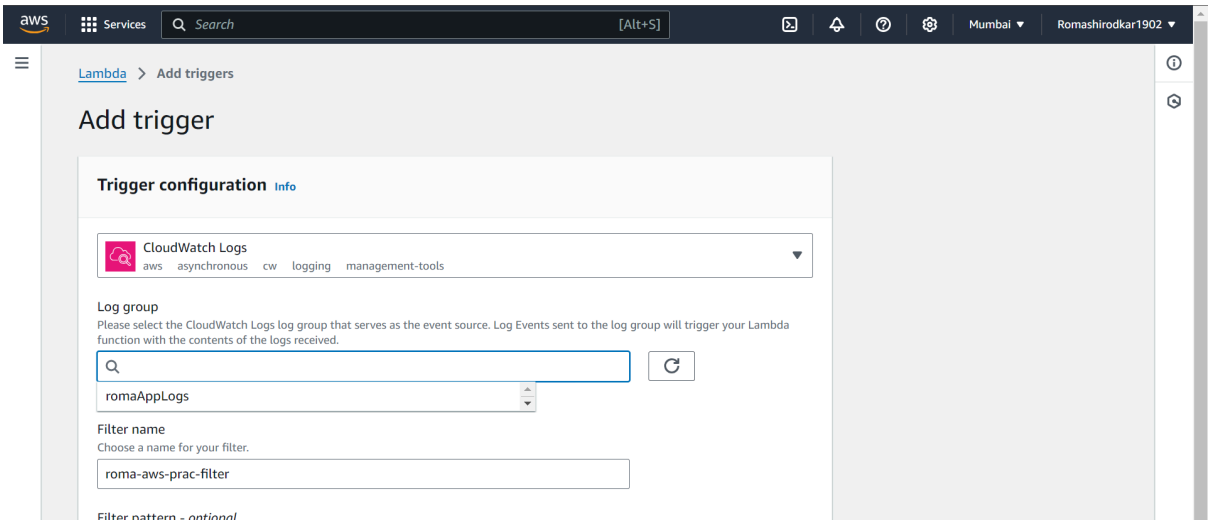
Navigate to lambda function created and click on “Add trigger”



Select **CloudWatch Logs** as the source.



Provide a filter name and also choose the log group created earlier.

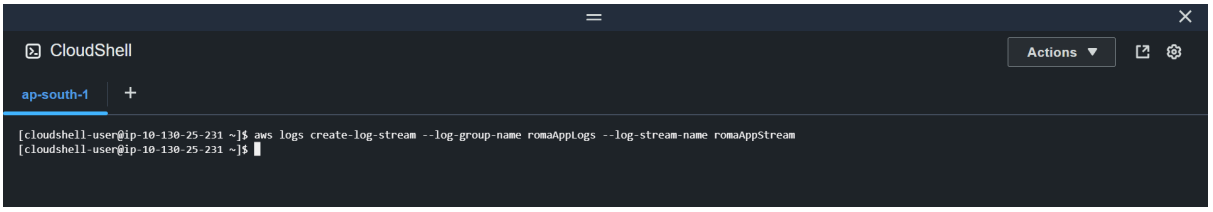


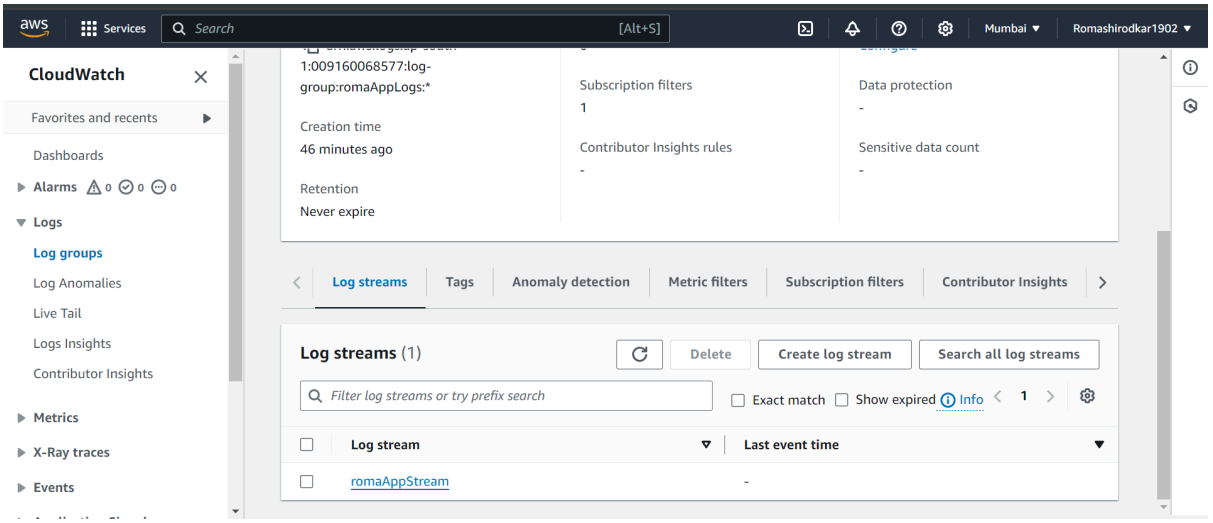
Step 7: Generate Logs for testing

Open AWS CLI and enter the following command

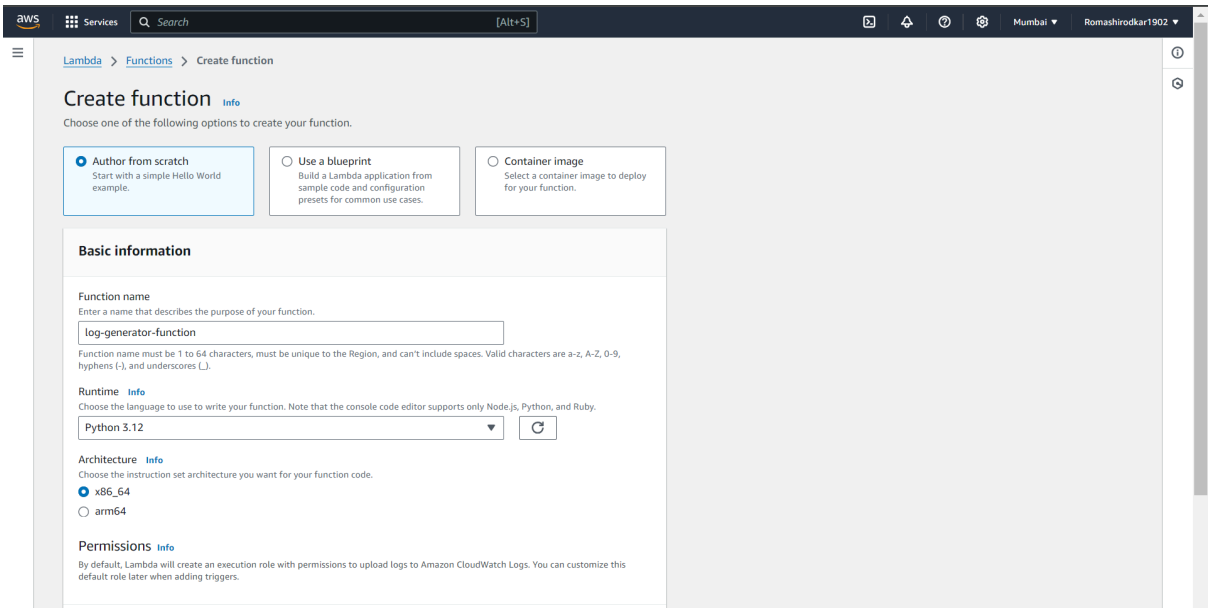
aws logs create-log-stream --log-group-name romaAppLogs --log-stream-name romaAppStream.

This will create a Log stream named romaAppStream for romaAppLogs log group





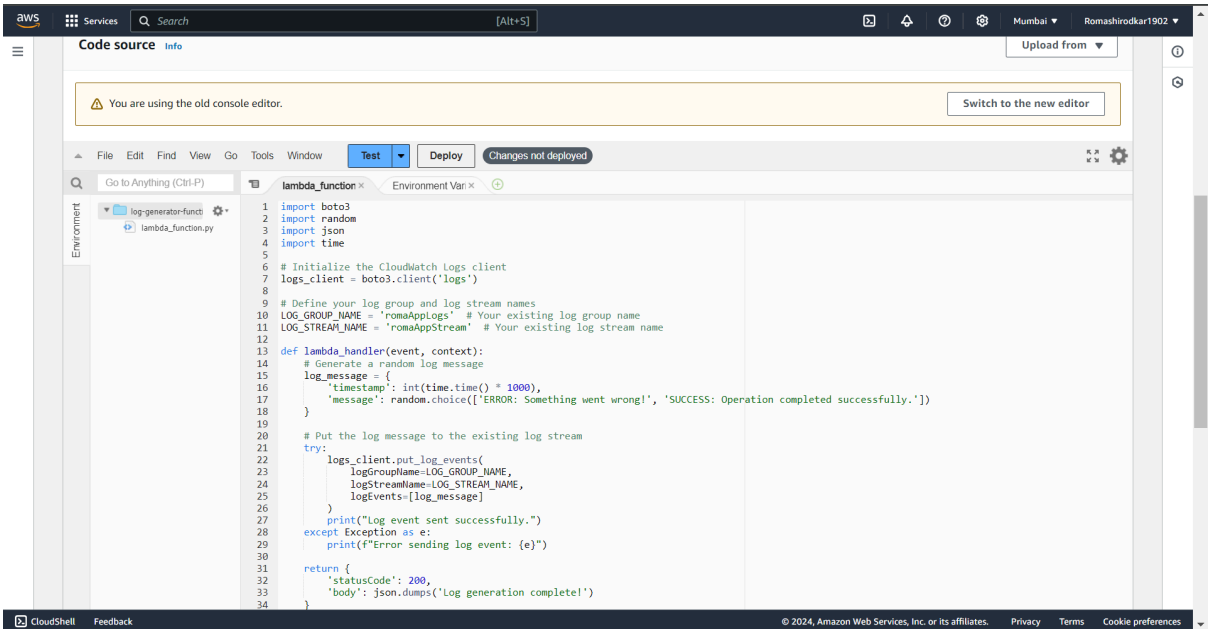
Now we will create another lambda function to generate logs



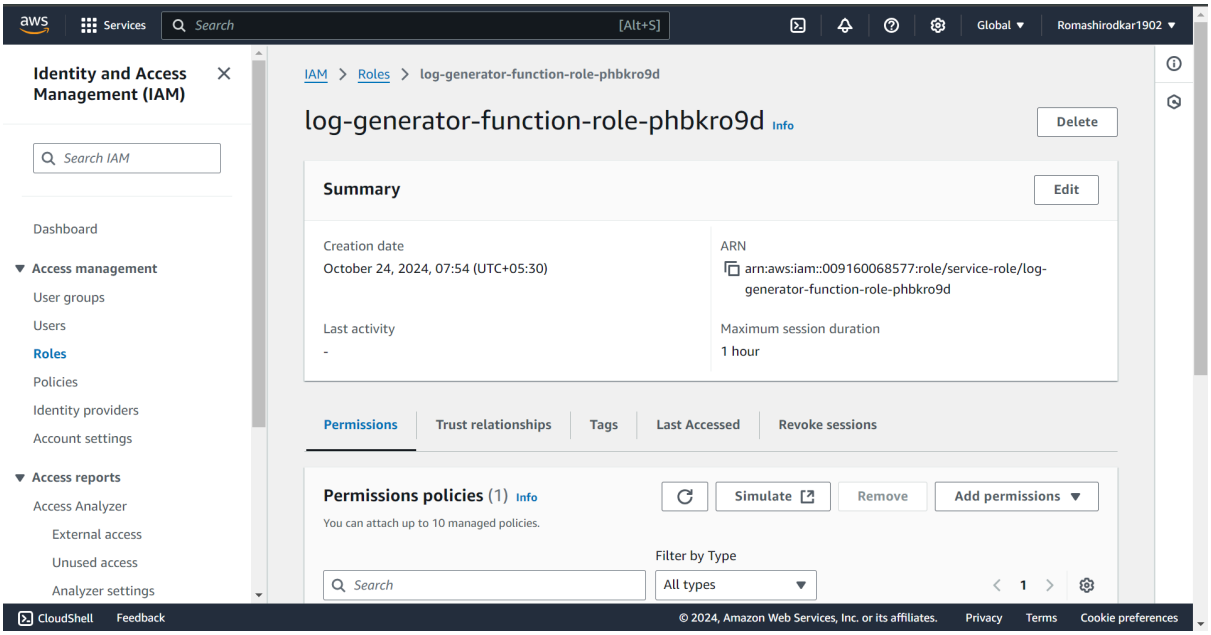
Add this code

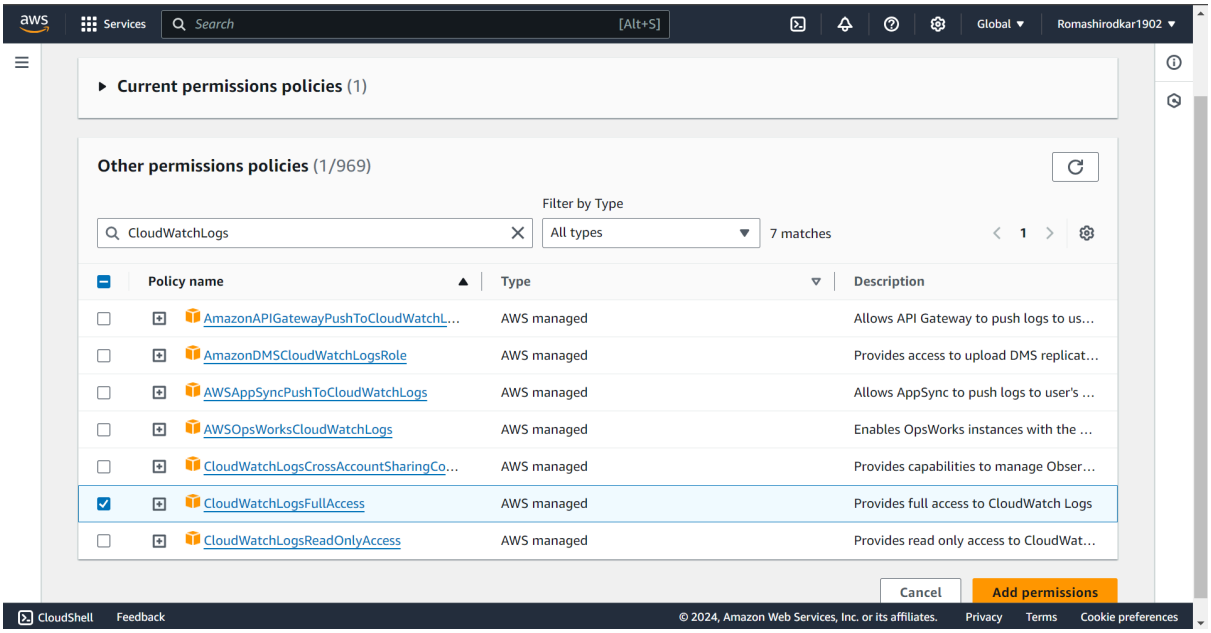
```
import boto3
import random
import json
import time

# Initialize the CloudWatch Logs client
logs_client = boto3.client('logs')

# Define your log group and log stream names
LOG_GROUP_NAME = 'romaAppLogs'  # Your existing log group name
LOG_STREAM_NAME = 'romaAppStream'  # Your existing log stream name

def lambda_handler(event, context):
    # Get the sequence token for the log stream
    try:
        response = logs_client.describe_log_streams(
            logGroupName=LOG_GROUP_NAME,
            logStreamNamePrefix=LOG_STREAM_NAME
        )
        log_stream = response['logStreams'][0]
        sequence_token = log_stream.get('uploadSequenceToken', None)
    except Exception as e:
        print(f"Error retrieving sequence token: {e}")
        return {
            'statusCode': 500,
```

```
            'body': json.dumps('Error retrieving sequence token')
        }

    # Start continuous log generation (run for 14 minutes)
    end_time = time.time() + (14 * 60)  # 14 minutes loop (as Lambda timeout is max 15 minutes)

    while time.time() < end_time:
        # Generate a random log message
        log_message = {
            'timestamp': int(time.time() * 1000),
            'message': random.choice(['ERROR: Something went wrong!', 'SUCCESS: Operation
completed successfully.'])
        }

        # Prepare the parameters for putting log events
        put_log_events_args = {
            'logGroupName': LOG_GROUP_NAME,
            'logStreamName': LOG_STREAM_NAME,
            'logEvents': [log_message]
        }

        # Include the sequence token if it exists
        if sequence_token:
            put_log_events_args['sequenceToken'] = sequence_token

        # Send the log event to the log stream
        try:
            response = logs_client.put_log_events(**put_log_events_args)
            sequence_token = response['nextSequenceToken']  # Update the sequence token for the next
call
            print("Log event sent successfully.")
        except Exception as e:
            print(f"Error sending log event: {e}")
            return {
                'statusCode': 500,
                'body': json.dumps('Error sending log event')
            }

        # Pause for a while before generating the next log (e.g., every 5 seconds)
        time.sleep(5)

    return {
        'statusCode': 200,
        'body': json.dumps('Continuous log generation complete!')
    }
```

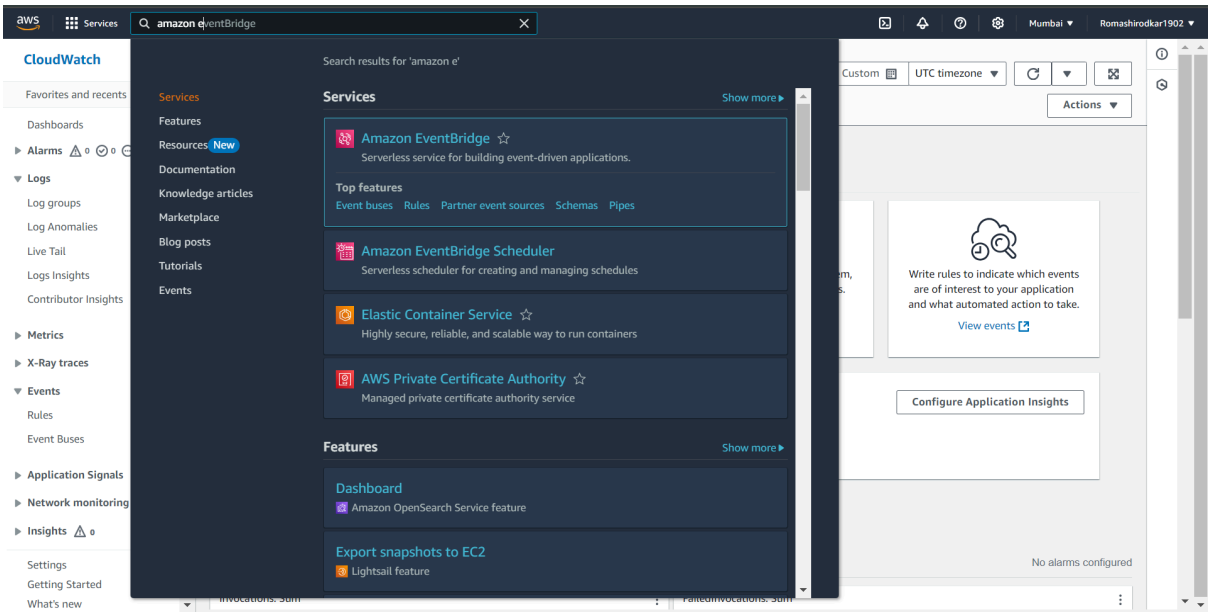


This function will need permission to put logs in CloudWatch log groups.
So Navigate to IAM console and attach policies.

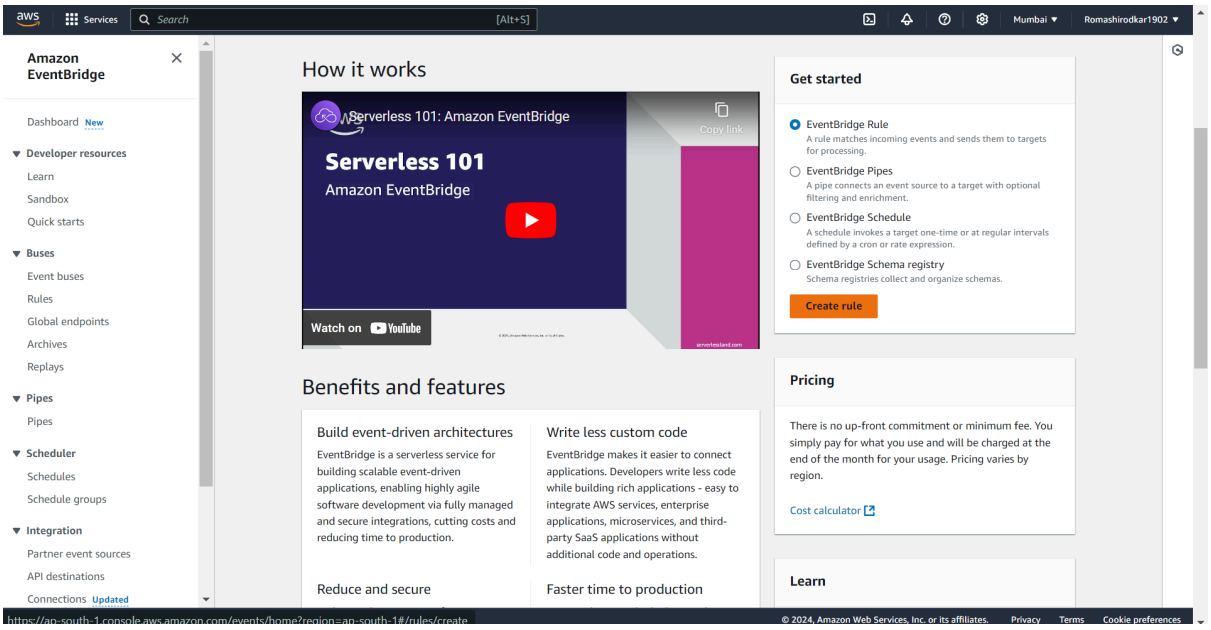




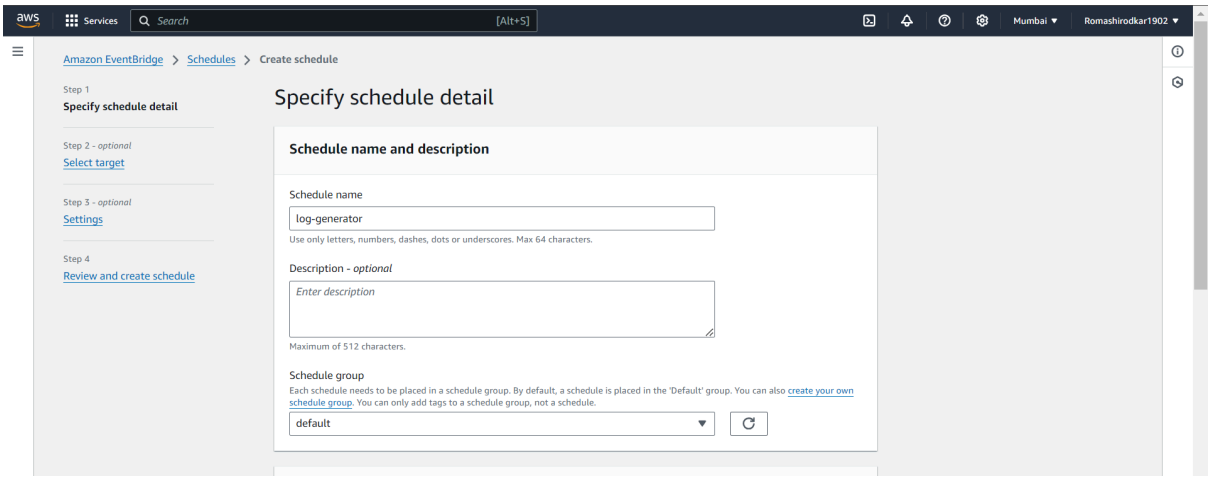
To automatically trigger this function we create a scheduler in Amazon EventBridge



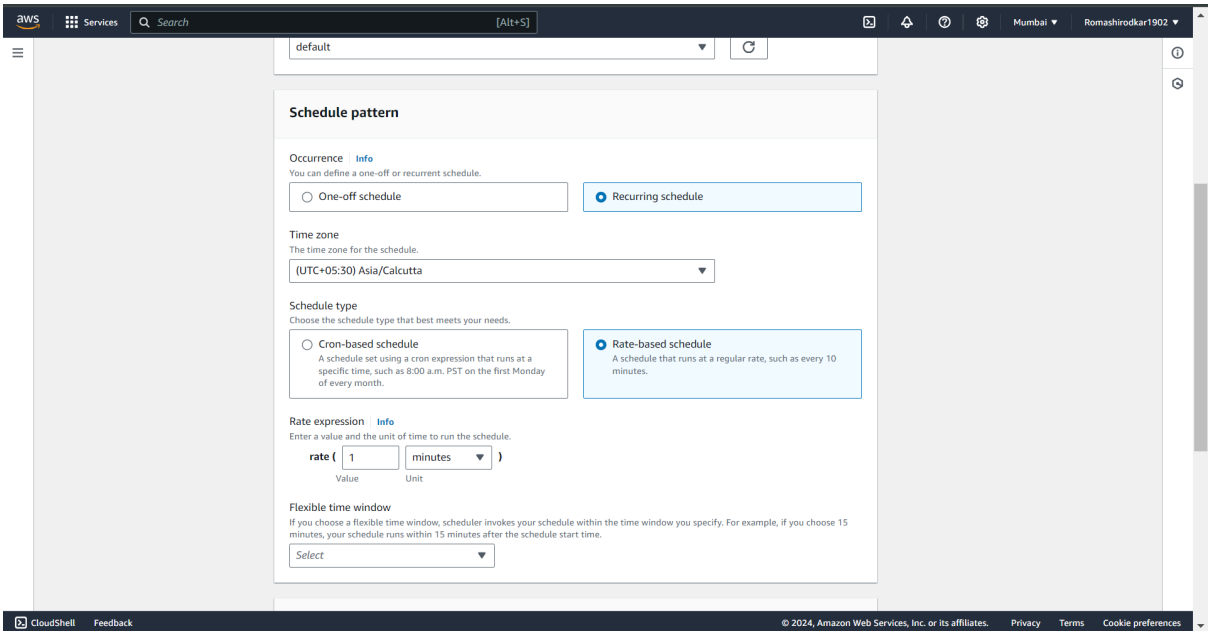
Click on Create Rule



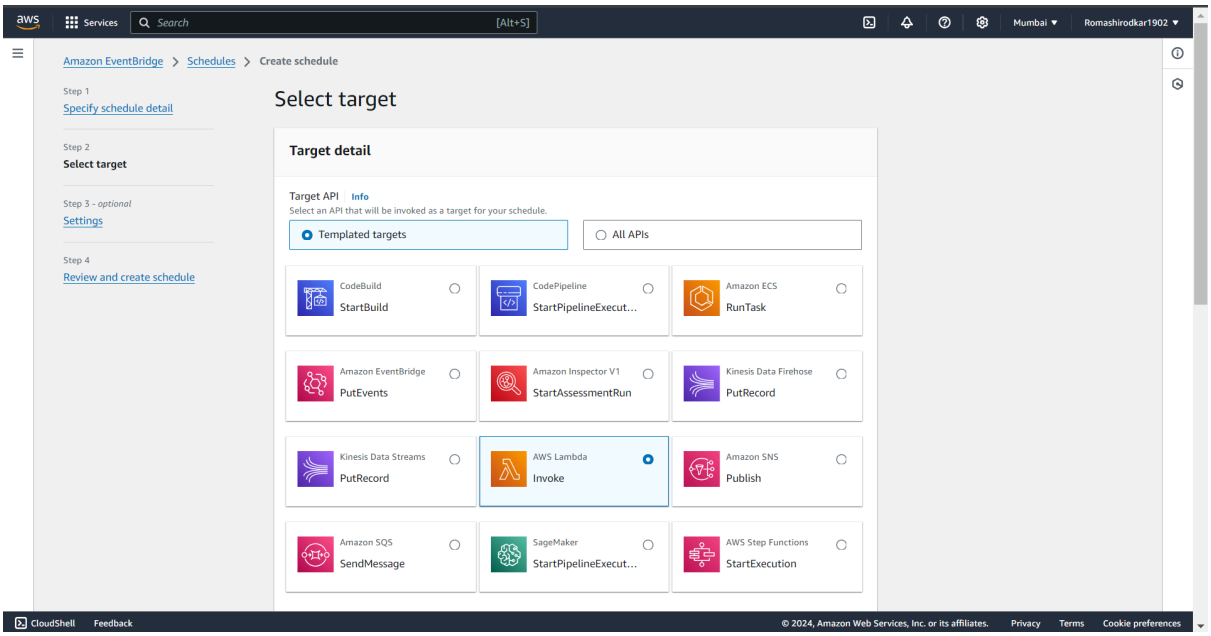
Provide details such as name for scheduler

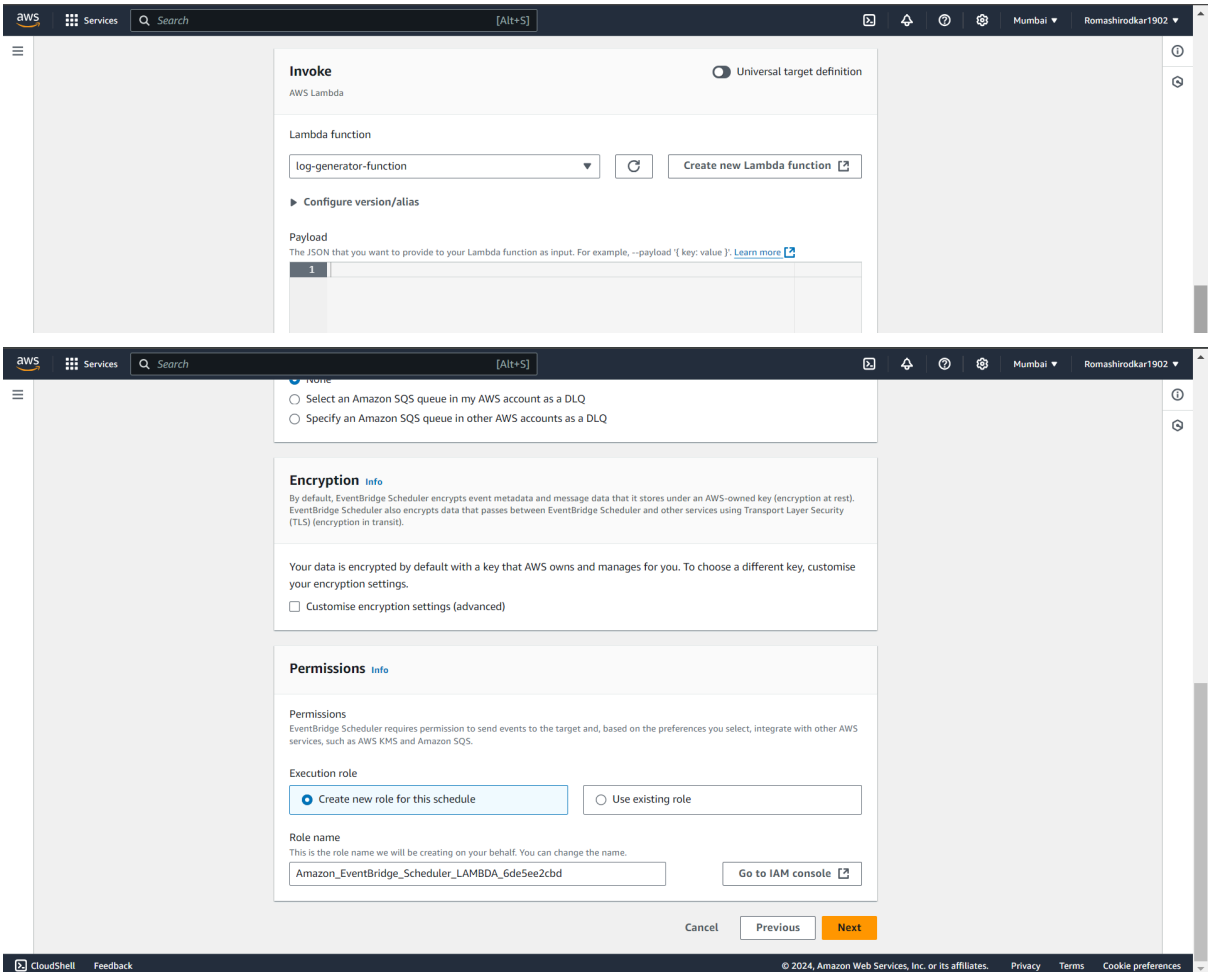


Select rate based scheduler and set (1 minutes) as rate

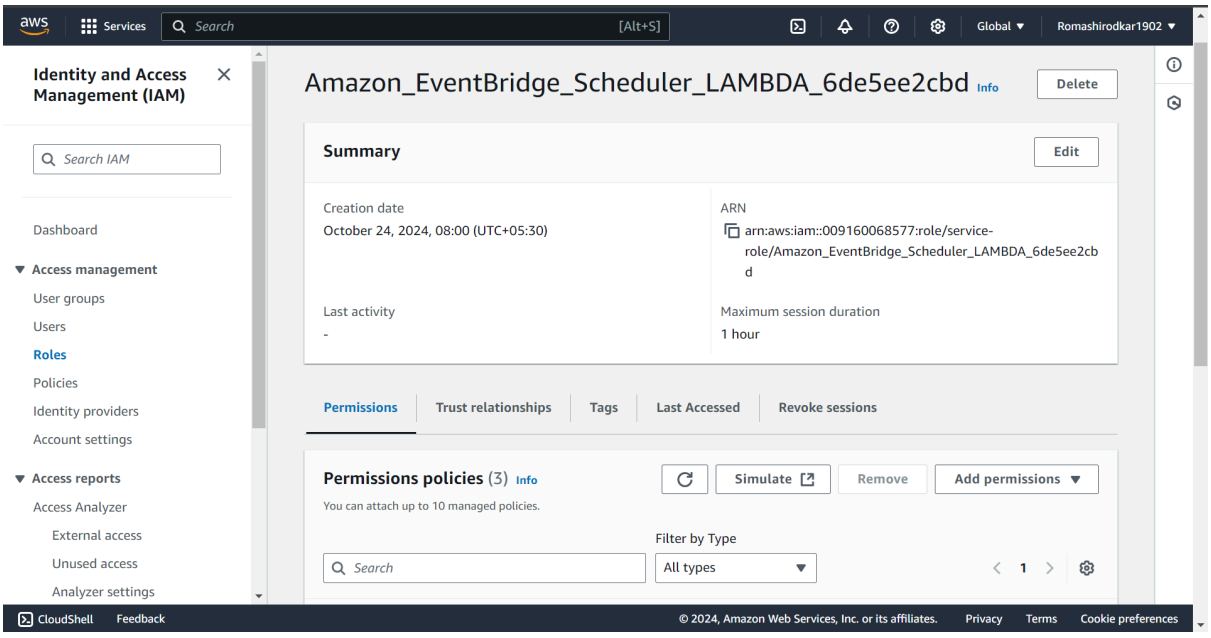


In target select Lambda and select log-generator-function.

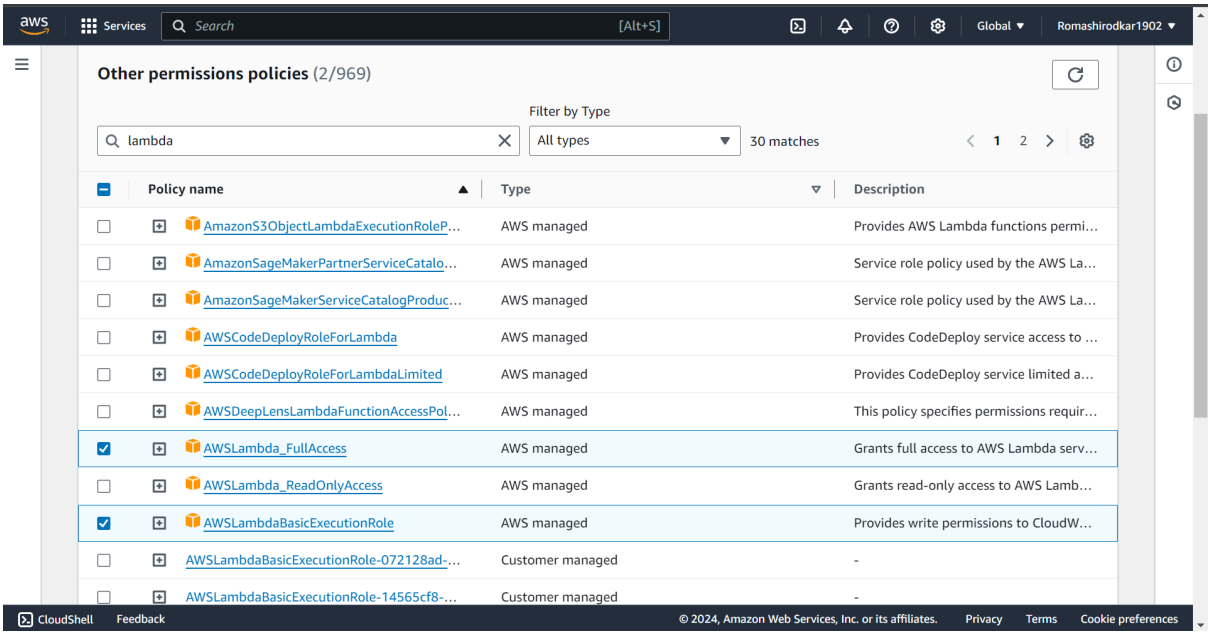




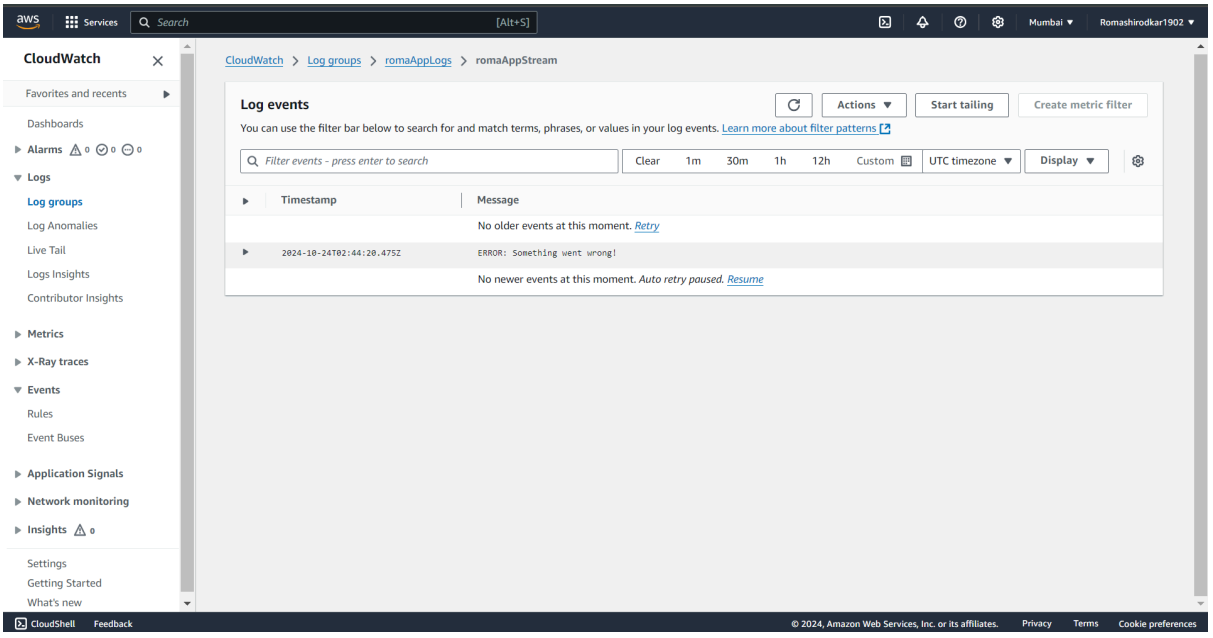
This scheduler needs permission to execute function.

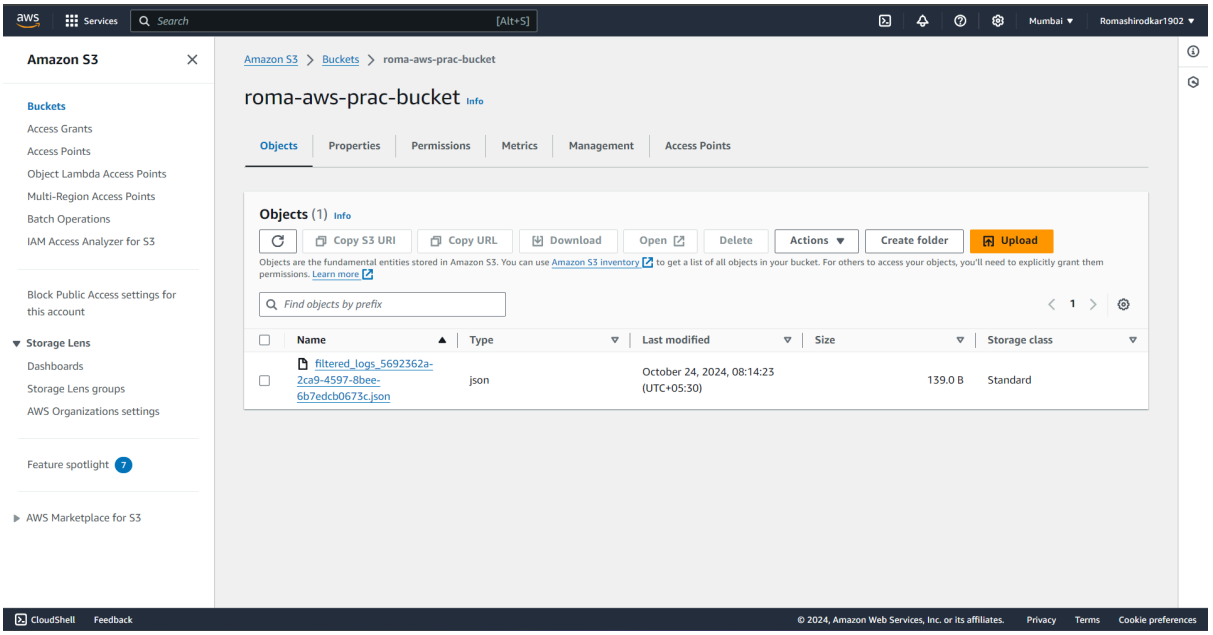


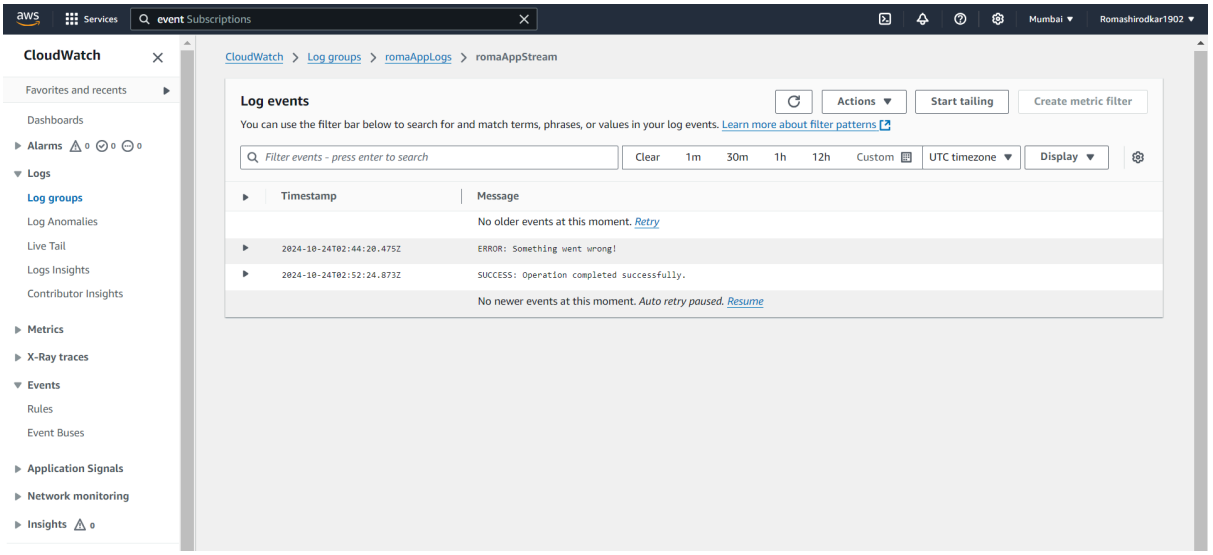
Attach the following policies

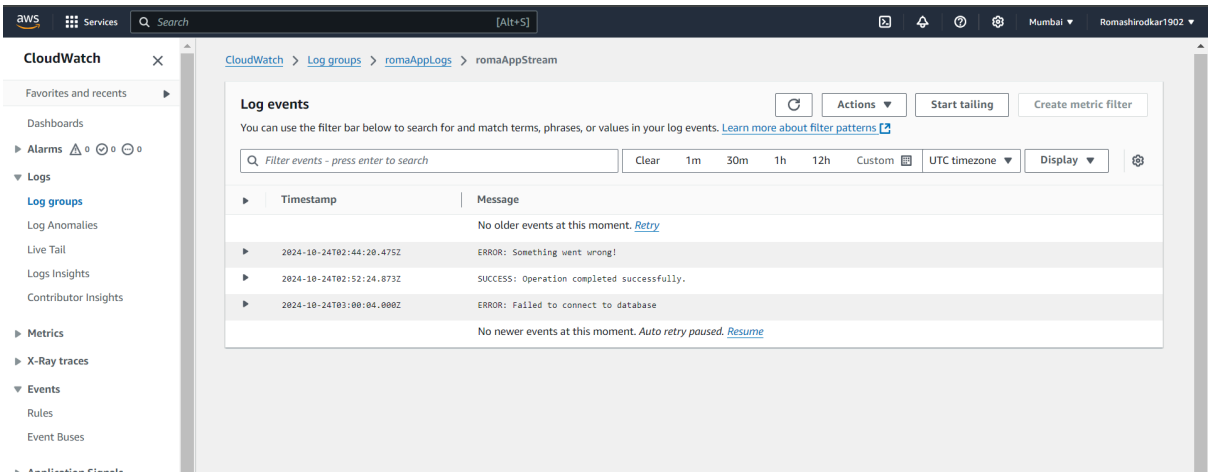


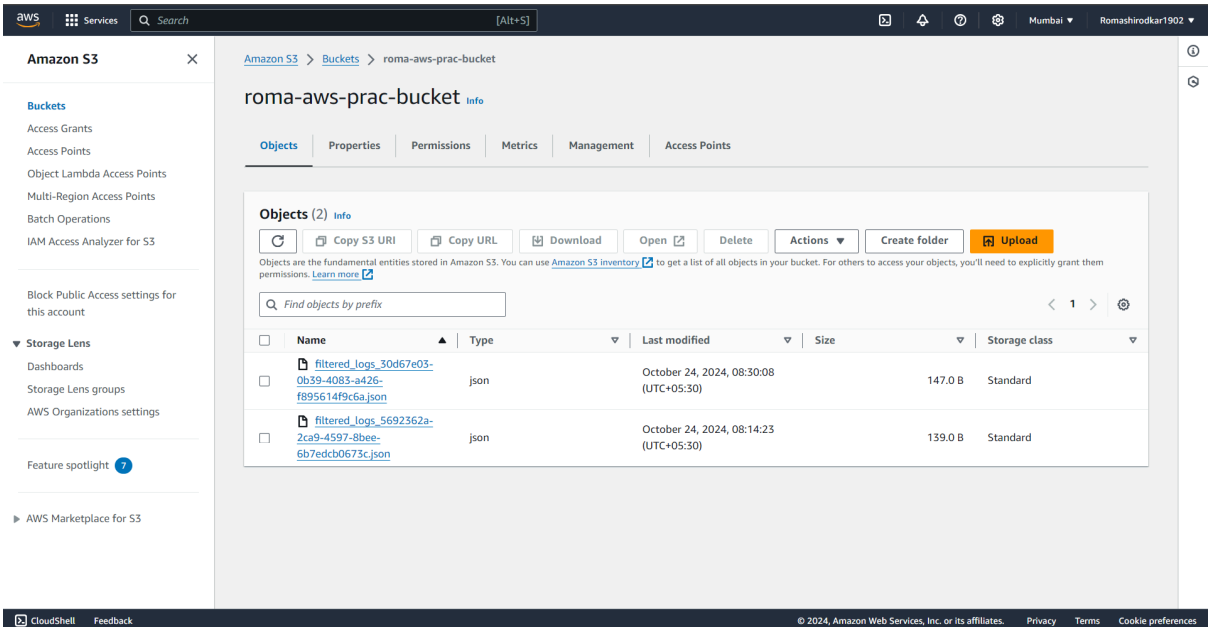
After that if we check CloudWatch Logs and S3 bucket we see the following.




Amazon S3
Buckets
Access Grants
Access Points
Object Lambda Access Points
Multi-Region Access Points
Batch Operations
IAM Access Analyzer for S3
Block Public Access settings for this account
Storage Lens
Dashboards
Storage Lens groups
AWS Organizations settings
Feature spotlight
AWS Marketplace for S3
Amazon S3 > Buckets > roma-aws-prac-bucket
roma-aws-prac-bucket Info
Objects
Properties
Permissions
Metrics
Management
Access Points
Objects (1) Info
Copy S3 URI
Copy URL
Download
Open
Delete
Actions
Create folder
Upload
Find objects by prefix
Name
Type
Last modified
Size
Storage class
filtered_logs_5692362a-2ca9-4597-8bee-6b7edcb0673c.json
json
October 24, 2024, 08:14:23 (UTC+05:30)
139.0 B
Standard
CloudShell
Feedback
© 2024, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences


CloudWatch
Favorites and recents
Dashboards
Alarms
Logs
Log groups
Log Anomalies
Live Tail
Logs Insights
Contributor Insights
Metrics
X-Ray traces
Events
Rules
Event Buses
Application Signals
Network monitoring
Insights
CloudWatch > Log groups > romaAppLogs > romaAppStream
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
Filter events - press enter to search
Clear
1m
30m
1h
12h
Custom
UTC timezone
Display
Timestamp
Message
No older events at this moment. Retry
2024-10-24T02:44:20.475Z
ERROR: Something went wrong!
2024-10-24T02:52:24.873Z
SUCCESS: Operation completed successfully.
No newer events at this moment. Auto retry paused. Resume


CloudWatch
Favorites and recents
Dashboards
Alarms
Logs
Log groups
Log Anomalies
Live Tail
Logs Insights
Contributor Insights
Metrics
X-Ray traces
Events
Rules
Event Buses
CloudWatch > Log groups > romaAppLogs > romaAppStream
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
Filter events - press enter to search
Clear
1m
30m
1h
12h
Custom
UTC timezone
Display
Timestamp
Message
No older events at this moment. Retry
2024-10-24T02:44:20.475Z
ERROR: Something went wrong!
2024-10-24T02:52:24.873Z
SUCCESS: Operation completed successfully.
2024-10-24T03:00:04.000Z
ERROR: Failed to connect to database
No newer events at this moment. Auto retry paused. Resume



As we can see in the above picture, logs are being filtered as we generate them.

Conclusion: In conclusion, this case study showcases the efficient use of AWS Lambda to automate log generation and processing in CloudWatch Logs. By generating random logs, filtering for specific error messages like "ERROR," and storing the filtered logs in Amazon S3, the Lambda function enhances real-time operational monitoring. The integration with S3 ensures scalable log storage and easy access for further analysis. This serverless solution eliminates the need for manual log management, optimizes resource usage, and minimizes operational costs. Overall, the automated system improves application reliability and facilitates faster incident resolution, enhancing performance monitoring in real time.