Advance DevOps Case Study

Case Study Topic: 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:

Overview

This case study focuses on a real-world implementation of monitoring, filtering, and storing AWS CloudWatch logs using an AWS Lambda function, integrated with Amazon S3 for persistent log storage. The primary goal of this project was to capture error logs generated from various AWS resources, filter them based on specific criteria, and store them for further analysis in a centralized location—an Amazon S3 bucket.

Key Features and Applications:

Key Features:

1. **Automated Log Monitoring**: Real-time capture of CloudWatch logs without manual intervention.
2. **Error Log Filtering**: Focuses on logs containing errors, filtering out noise.
3. **S3 Storage for Error Logs**: Persistently stores filtered error logs in S3 for easy access.
4. **Scalable Serverless Solution**: Uses AWS Lambda for dynamic scaling with no infrastructure management.
5. **Real-time Error Detection**: Detects and processes errors as they occur.

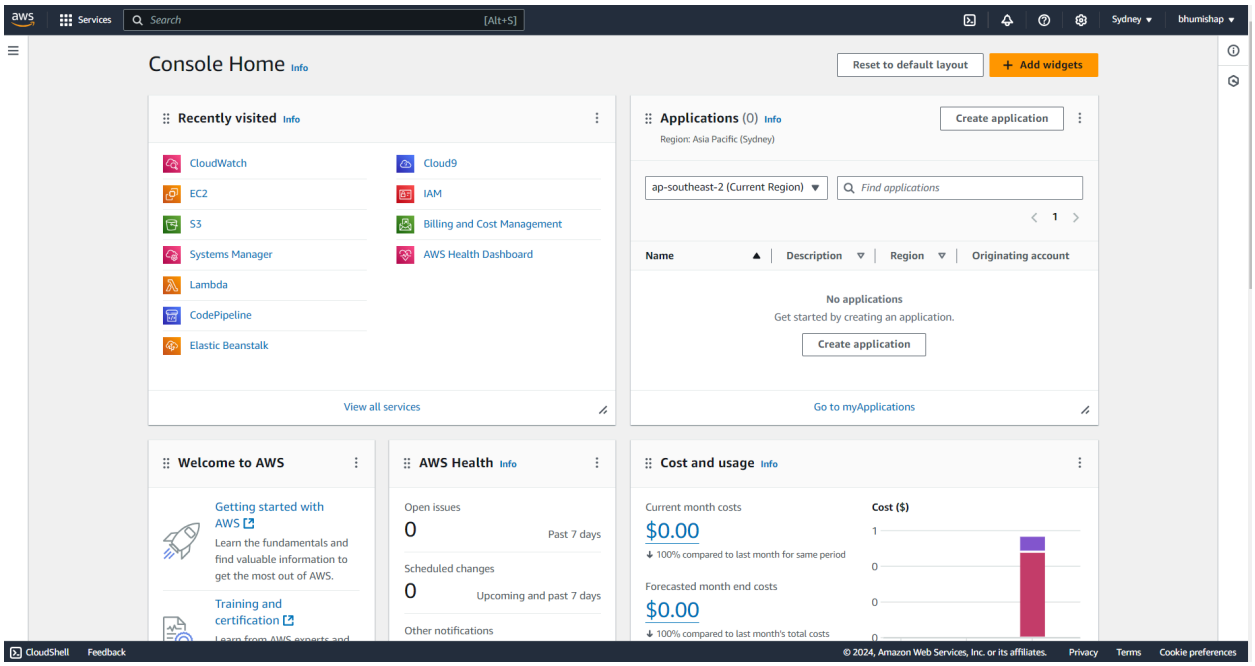
6. **Cost-effective**: Minimal infrastructure costs with serverless and S3 storage.
7. **JSON Structured Logs**: Stores logs in structured format for easy analysis.

Applications:

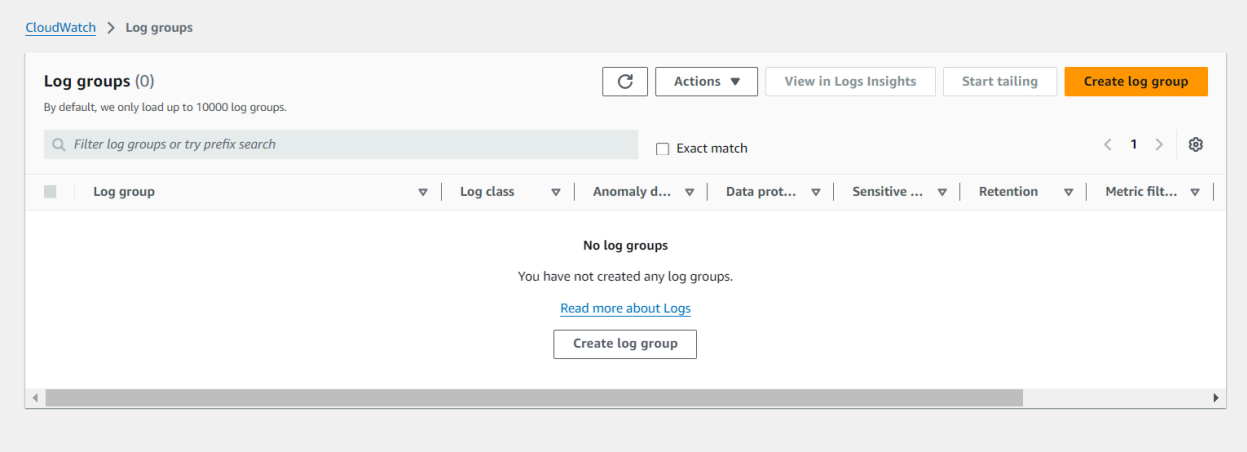
1. **Application Monitoring**: Automatically captures and filters critical error logs in real-time, improving performance tracking.
2. **System Diagnostics**: Provides centralized, structured log storage for quick issue identification.
3. **Cloud Infrastructure Management**: Scales effortlessly with cloud services, minimizing maintenance efforts and costs.
4. **Compliance Auditing**: Stores logs securely for compliance checks and audits over time.

Step-by-Step Explanation

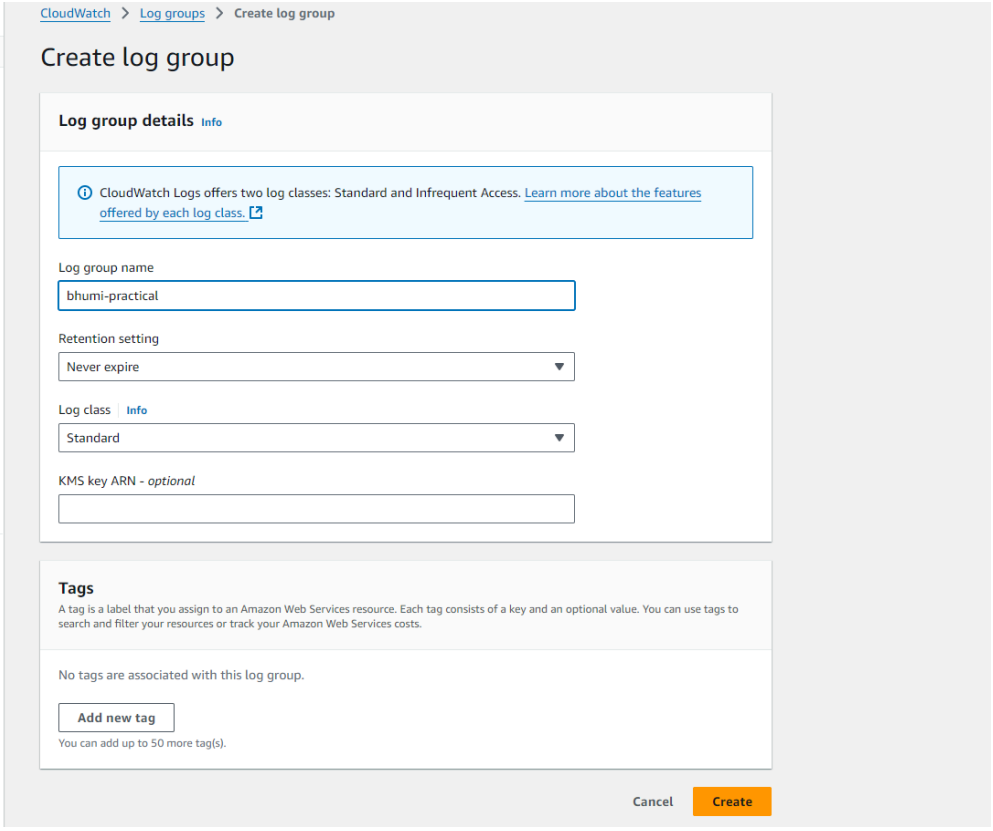
1. Log in to your AWS account and on the AWS Management Console navigate to **CloudWatch**.



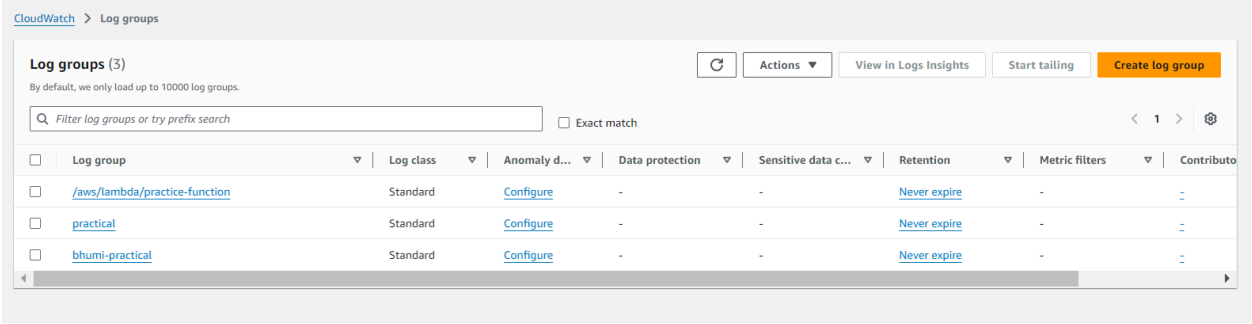
2. In the sidebar, select **Logs** and then click on **Create log group**. Give your log group a name, for example, ‘bhumi-practical’



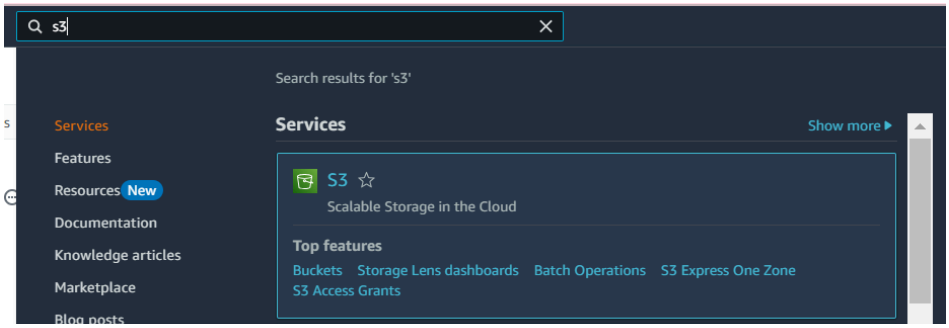
Keep the rest of the settings default and **Create**.

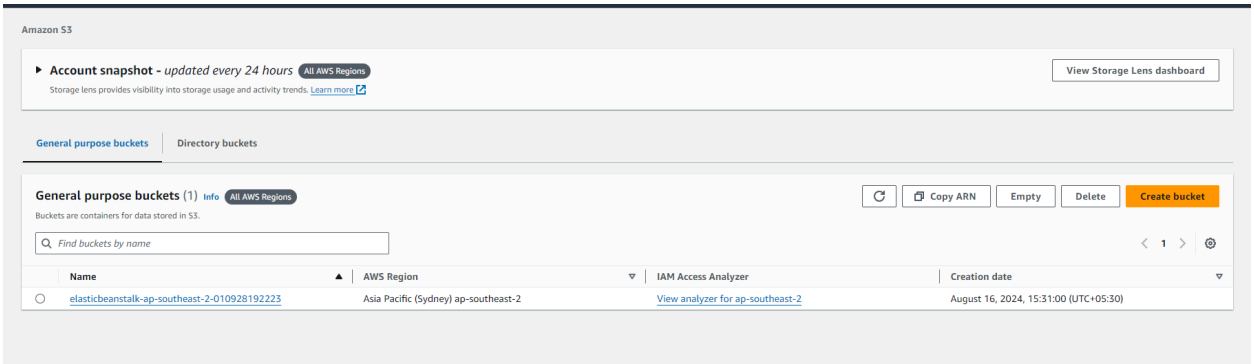


We can see that our log group has been created.

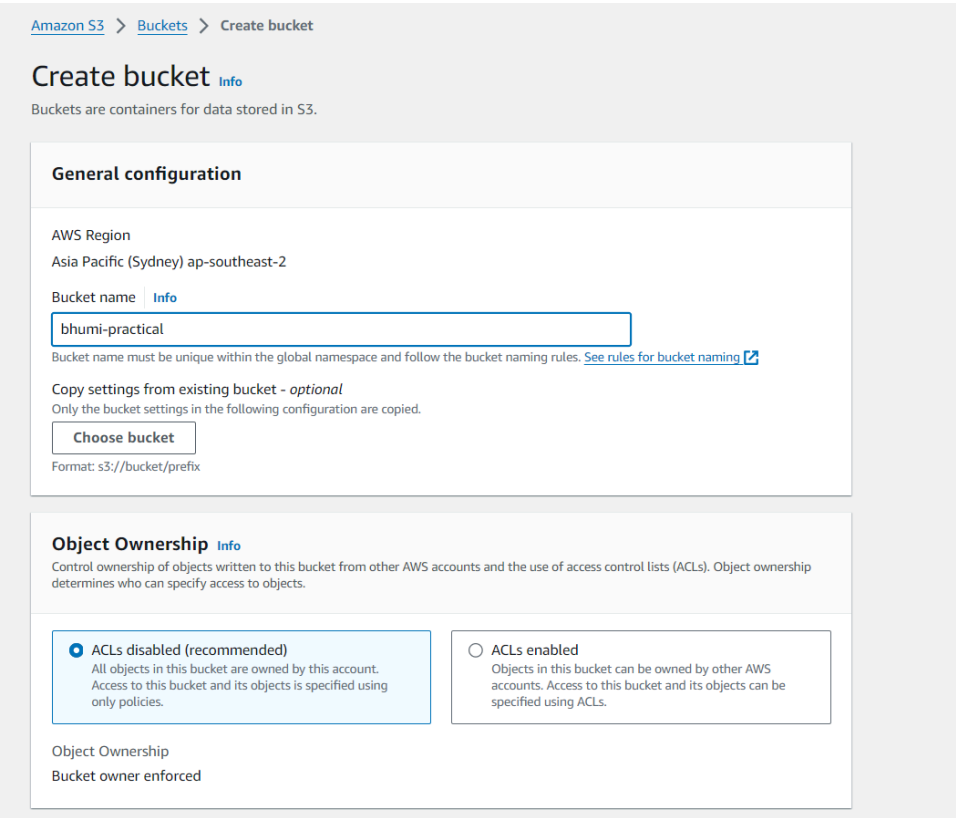


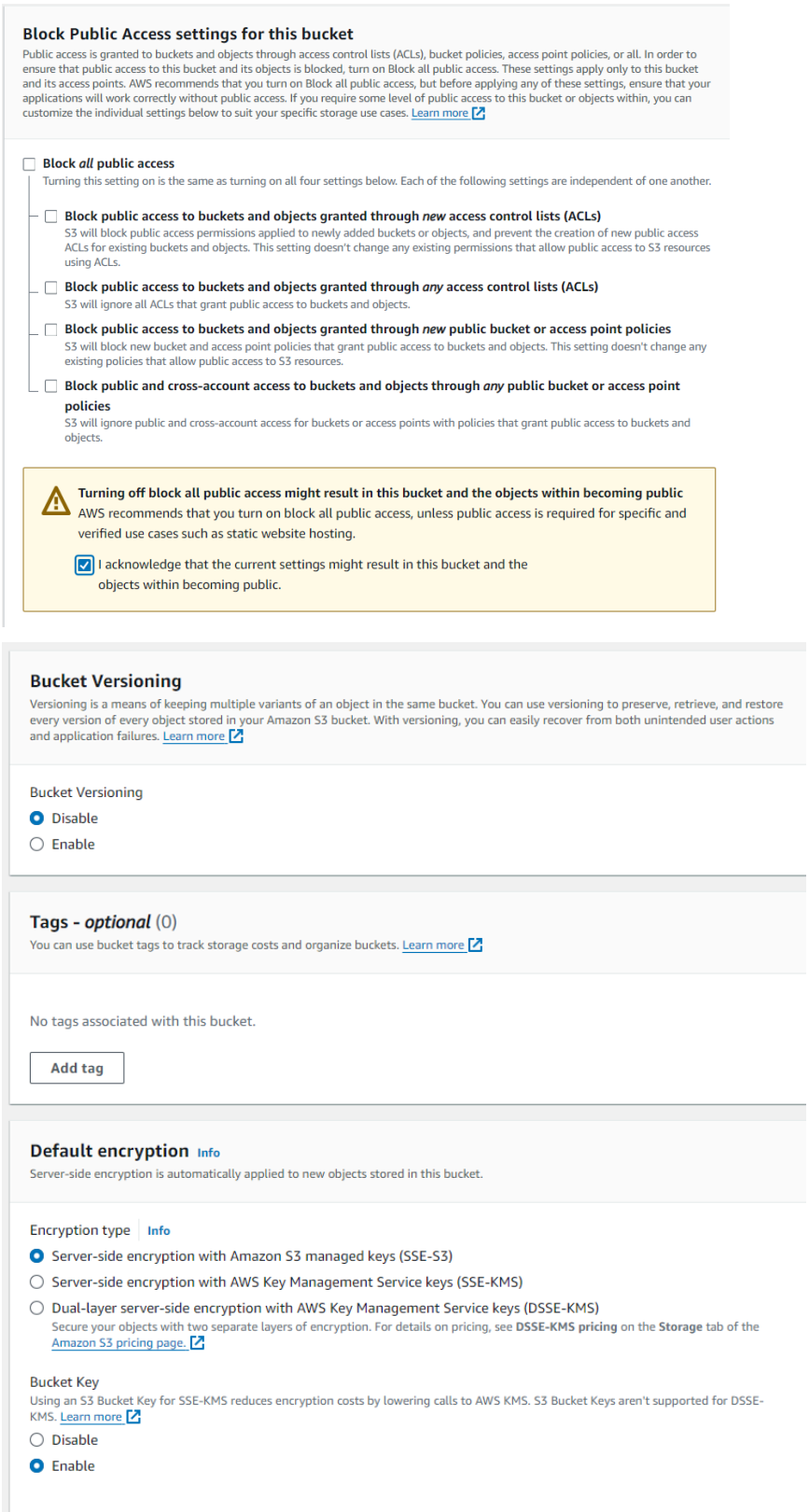
3. Now search for S3 and navigate to the **Create bucket** option and click on it.



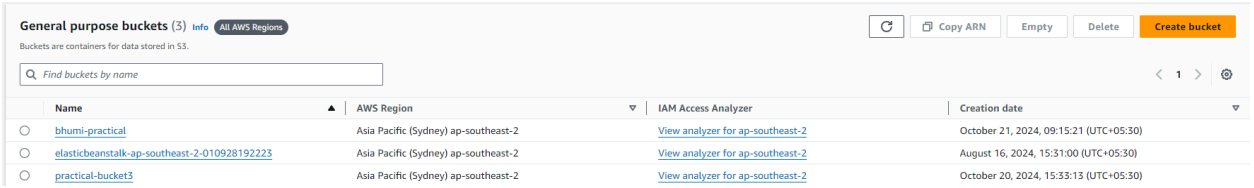


Give a name to your bucket. Then **untick the block all public access**, this will ensure that the bucket is public and has proper permissions for our Lambda function to write to it. Keeping the rest of the options default, we click on **Create**.

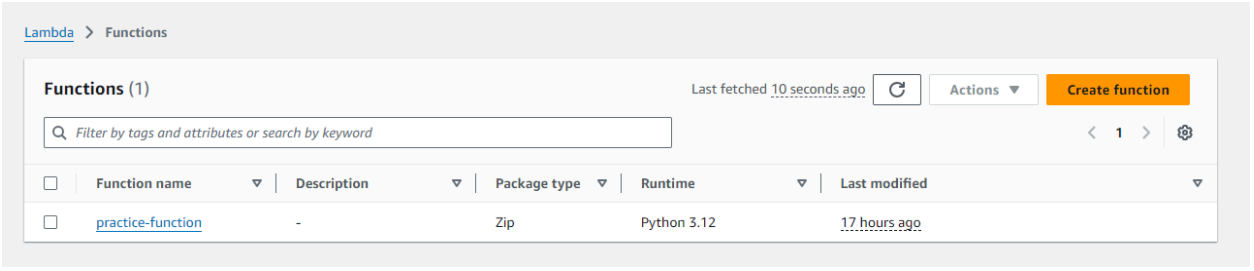




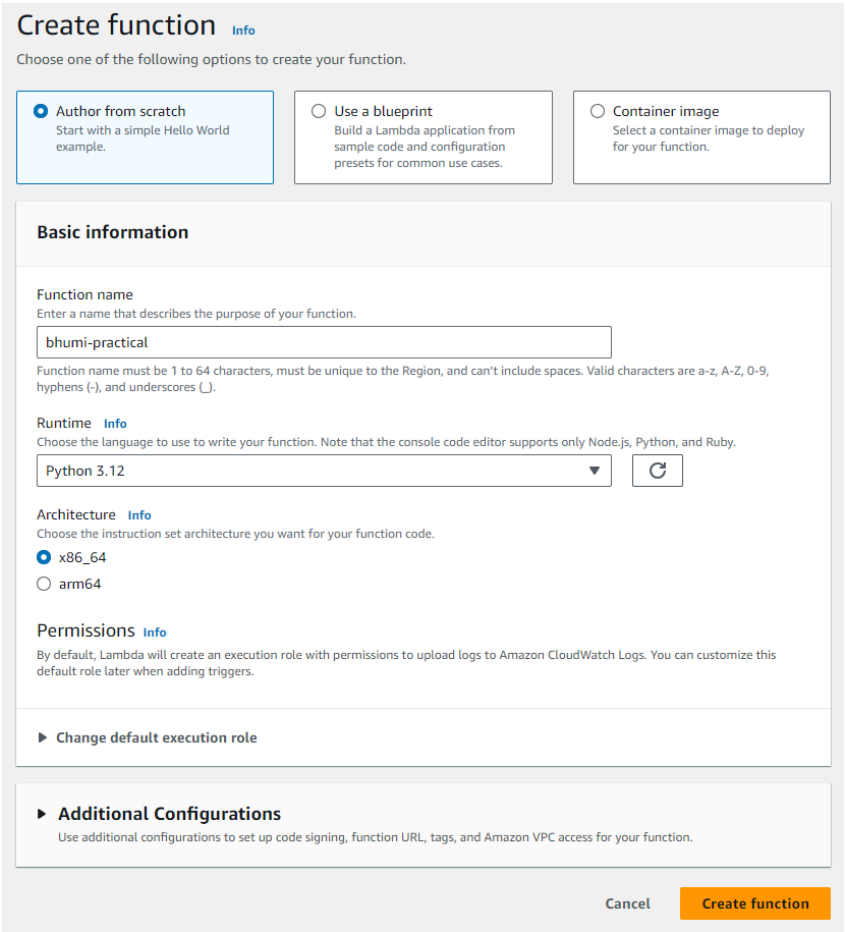
We can see that our bucket has been created.



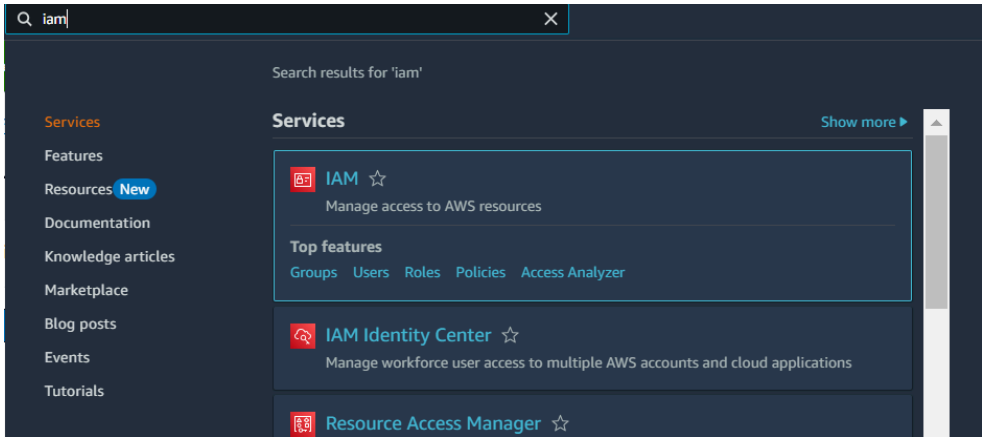
4. Next, we need to create a lambda function. We click on the **Create function**.



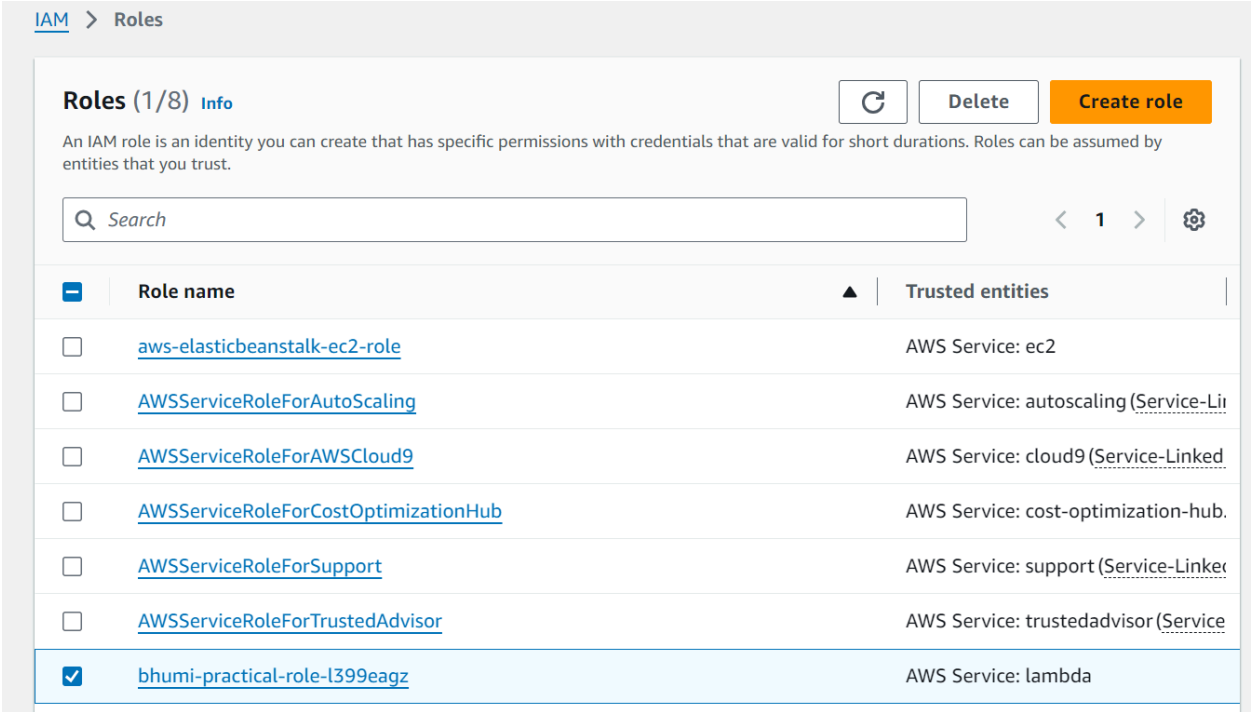
We select the option '**Author from Scratch**'. Click **Create function**, choose **Author from scratch**, and give it a name like bhumi-practical. Set **Runtime** to **Python 3.12**. We keep the rest of the settings default and create the function



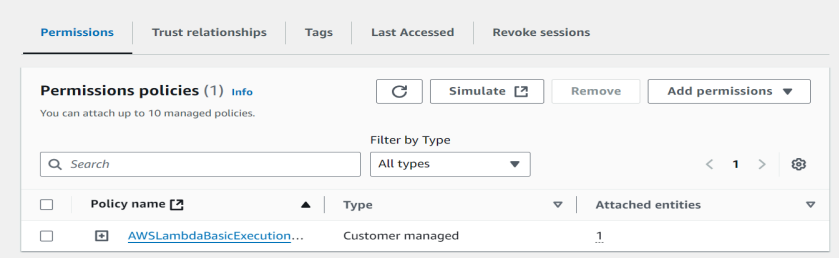
5. Now search for **IAM** in the console and then go to **Roles** from the options on the left sidebar.



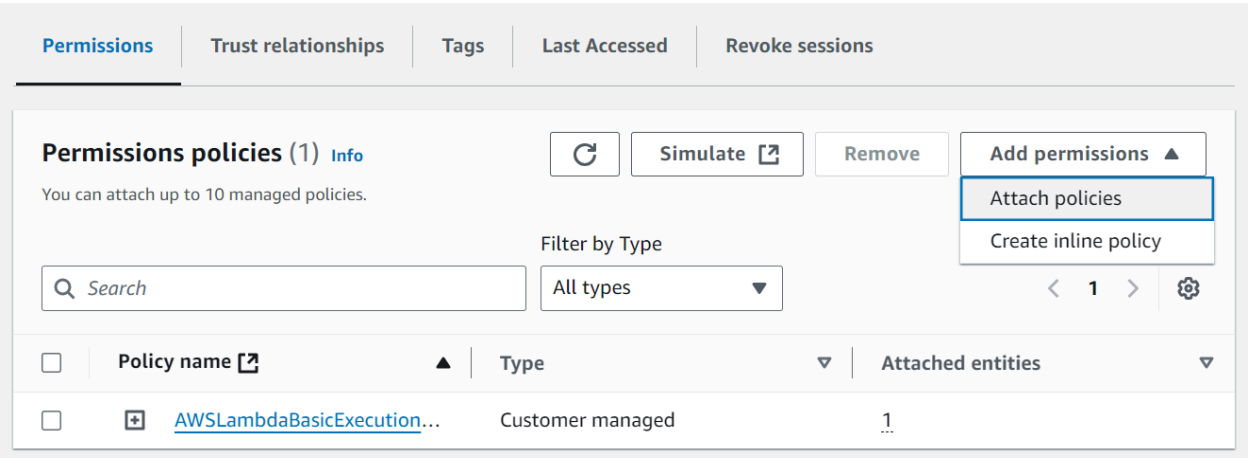
Here, you can see the role associated with your lambda function. It has the same name as your lambda function with role. Click on the name.

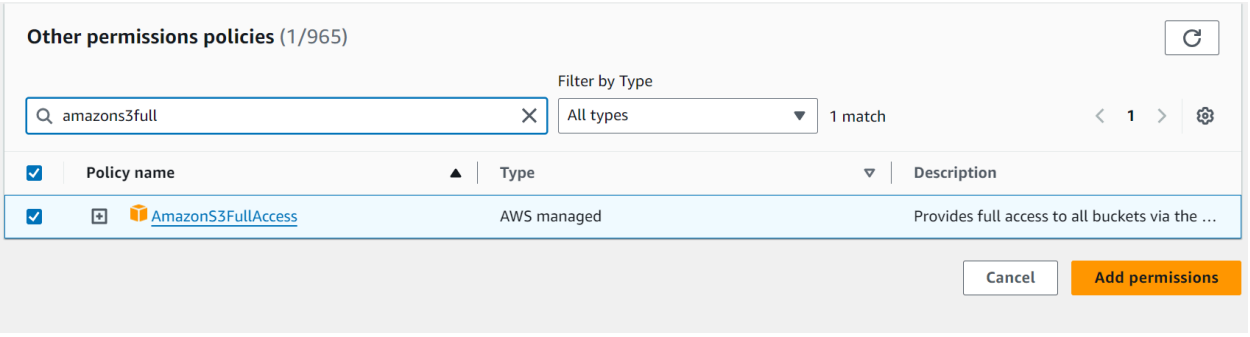


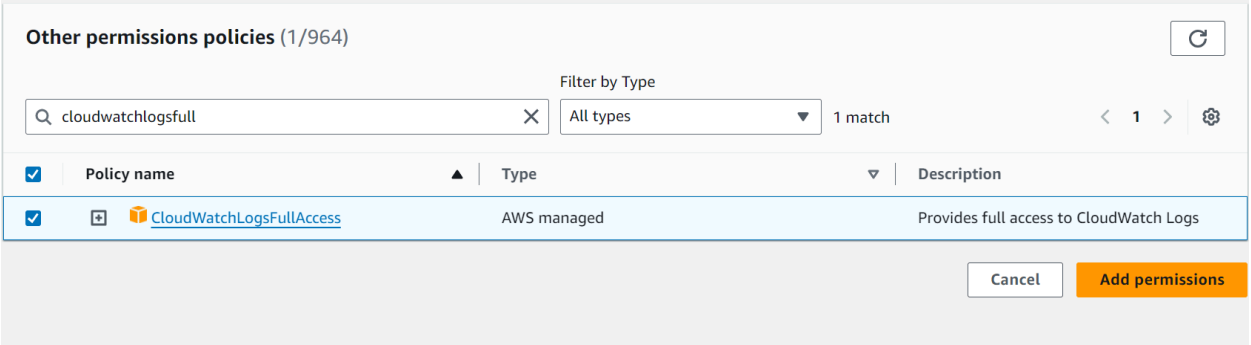
Next, scroll down to find the permissions section like shown in the screenshot.

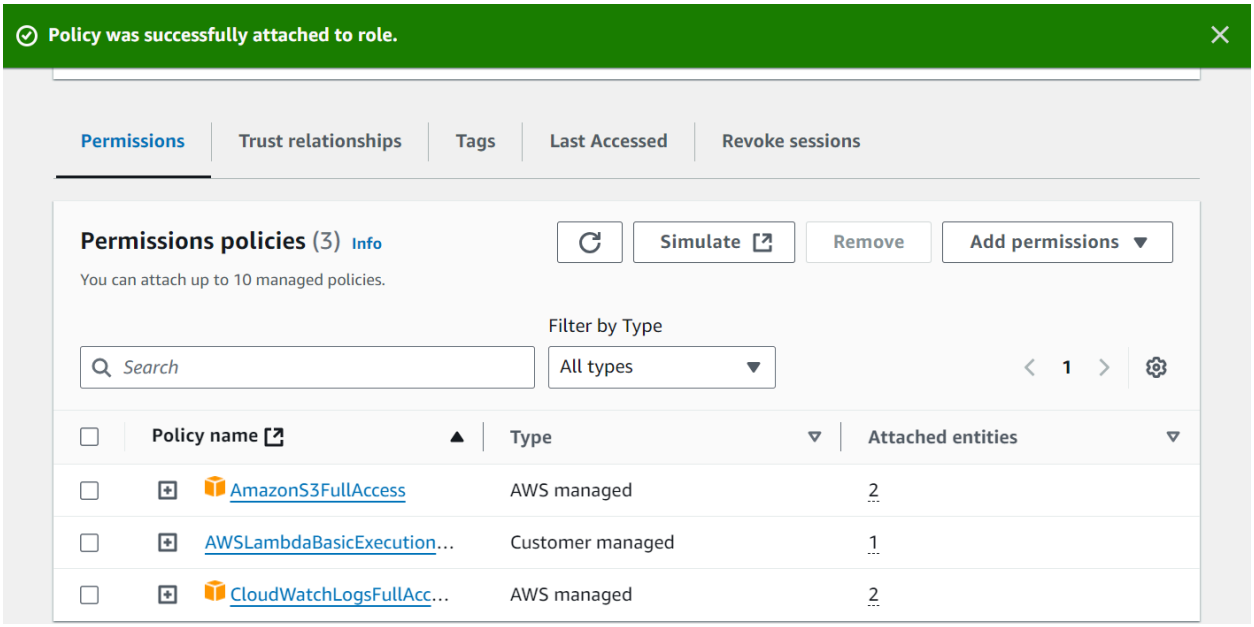


6. Click on the Add permissions button and a dropdown will appear, from that select the attach policies. You'll need to attach the following policies to the role:
   - AmazonS3FullAccess (to allow Lambda to write to S3).
   - CloudWatchLogsFullAccess (to allow Lambda to read logs from CloudWatch).

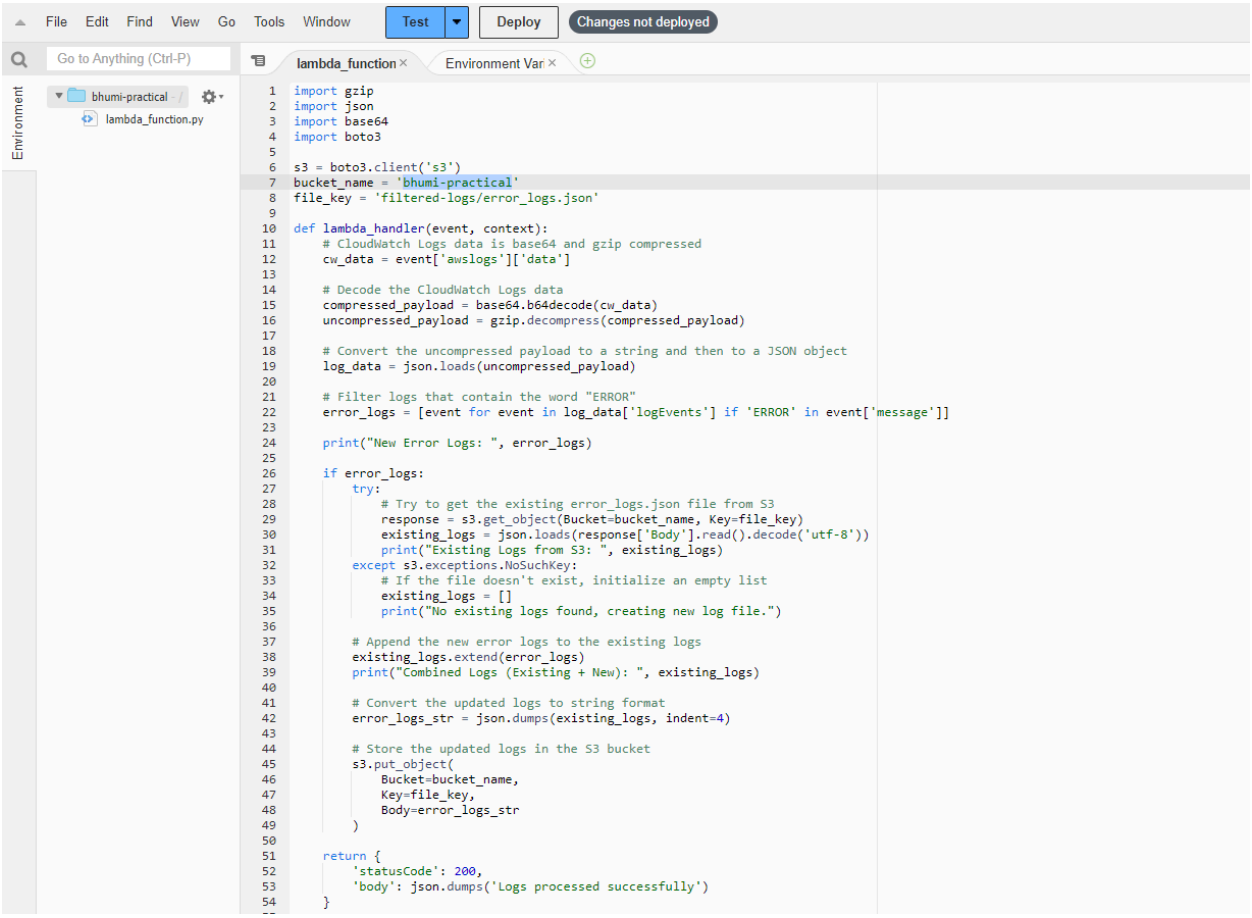




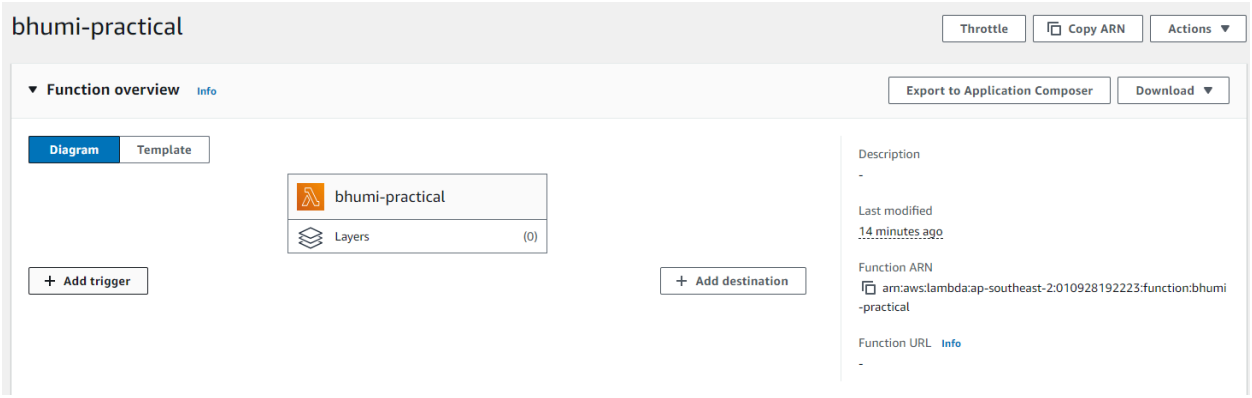




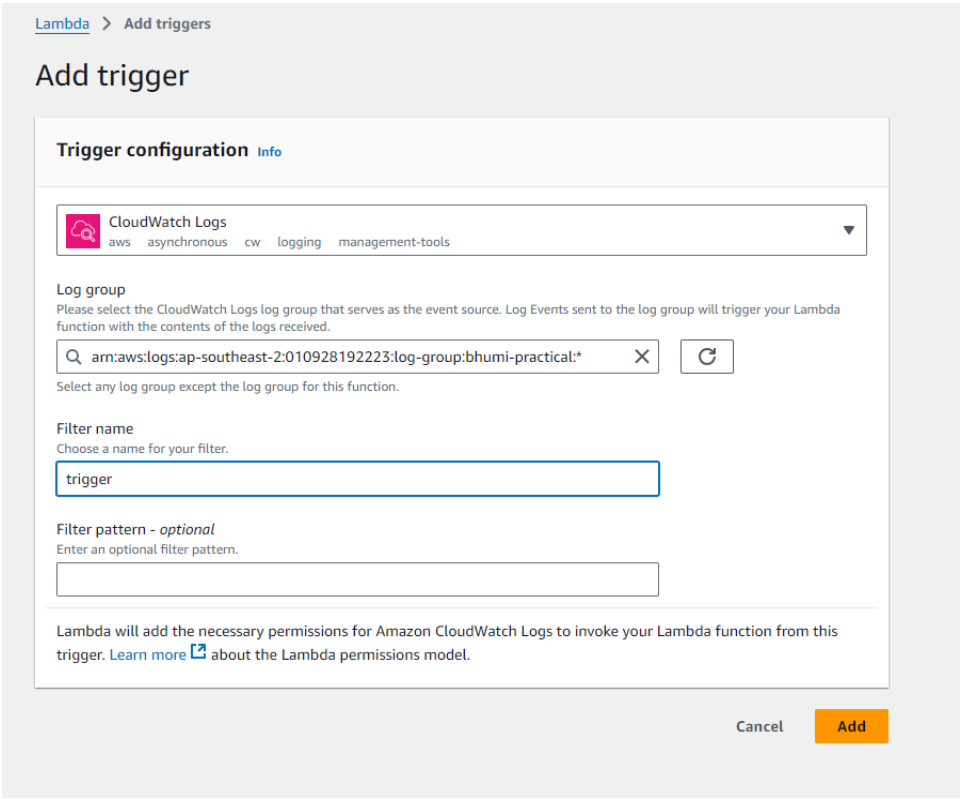
7. Navigate back to the lambda function, in the code section we add the following code. This Lambda function will filter logs containing the keyword "ERROR" and store them in an S3 bucket.



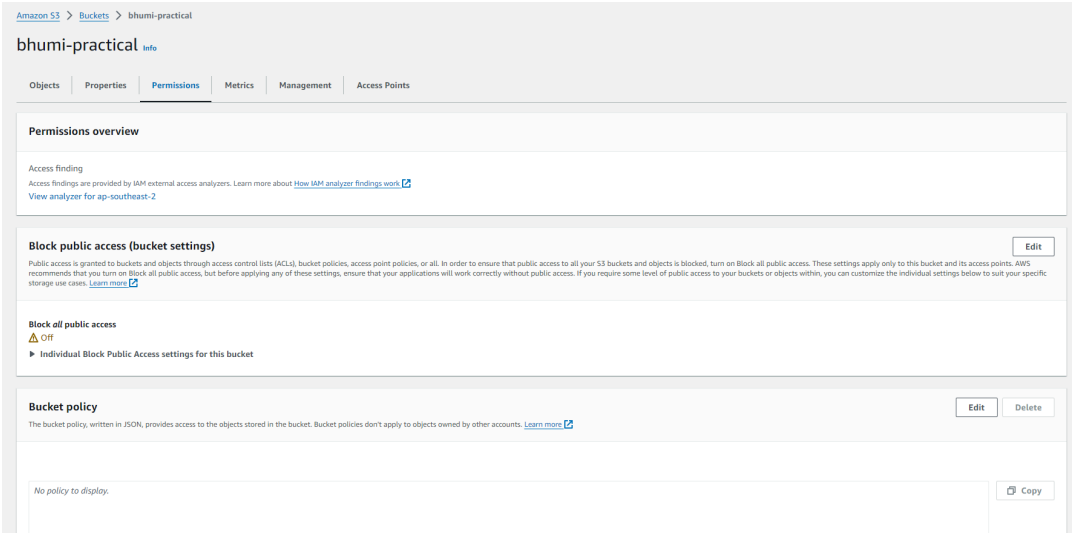
8. In the **Lambda function**, click on **Add Trigger**.



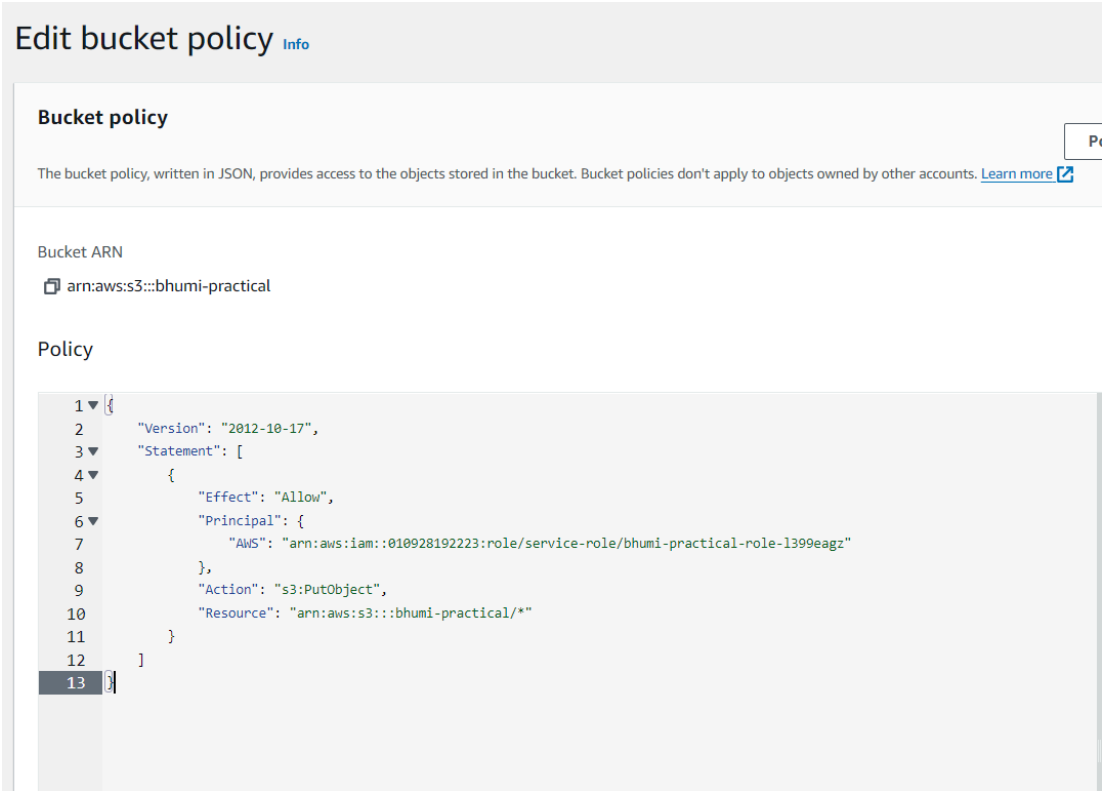
Select **CloudWatch Logs** from the dropdown. Choose the log group you created earlier (bhumi-practical). Choose a filter name and click on **Add**.



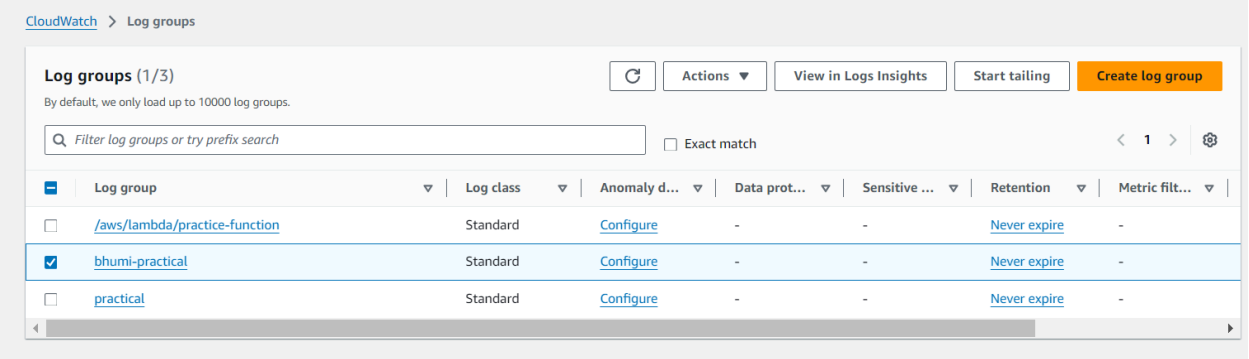
9. Go to the **S3 Console** and select the bucket you created. Navigate to the **Permissions** tab. Scroll down to **Bucket Policy** and click **Edit**.



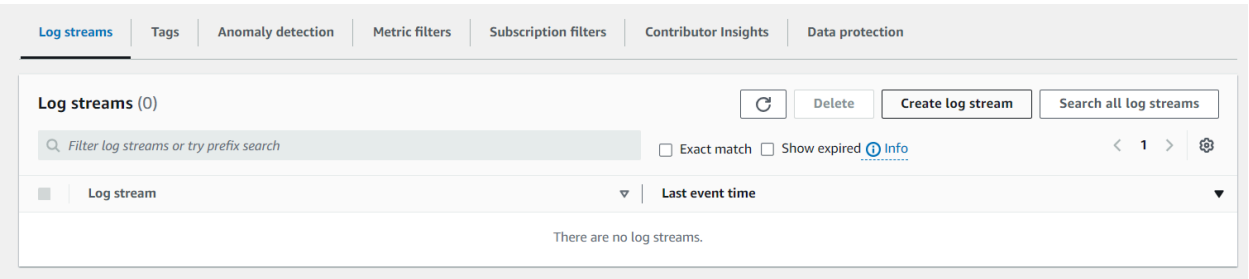
In the bucket policy, make sure you add the correct **bucket name** in the resource and the correct **role ARN** for your lambda role. This policy gives your Lambda function permission to upload objects to your bucket.

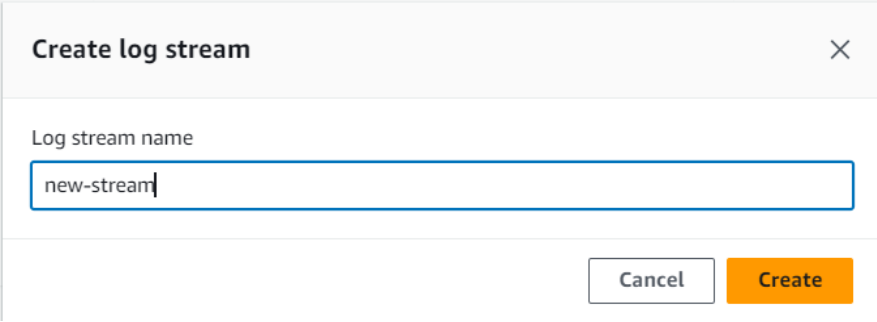


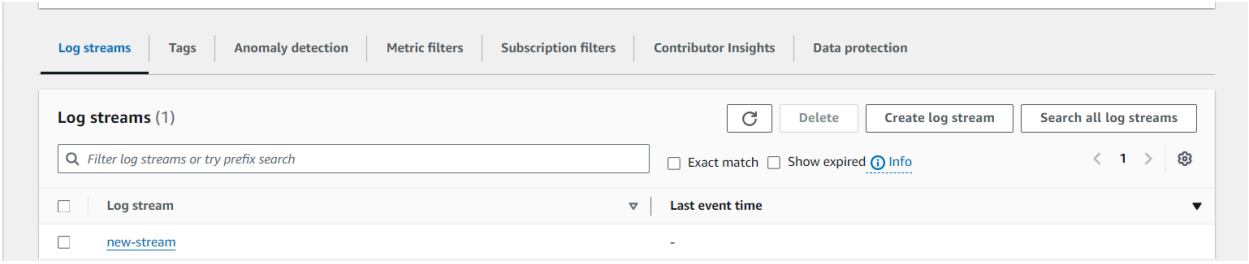
10. On the CloudWatch dashboard, on the left sidebar, click on **Logs → Log Groups**. This will display all the Log Groups associated with your AWS account. Select the one associated with your lambda function.



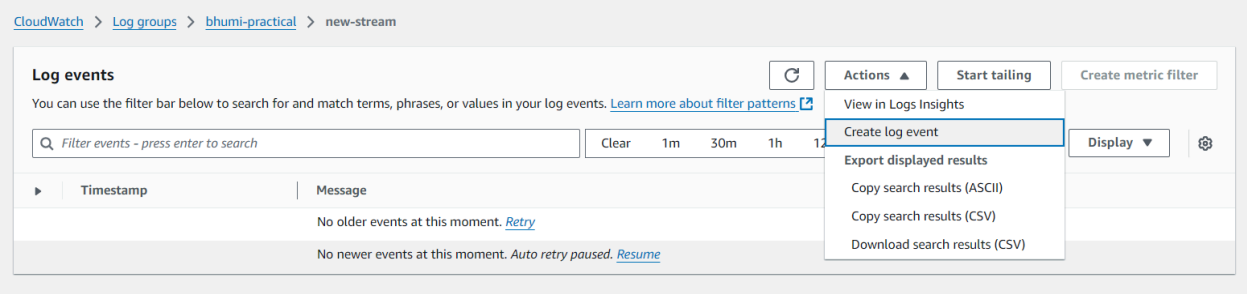
Inside the Log Group, you'll see **Log Streams**. A log stream is a sequence of log events for a specific resource that writes to CloudWatch. Click the **Create log stream** button, name the stream (e.g., new-stream), and proceed.



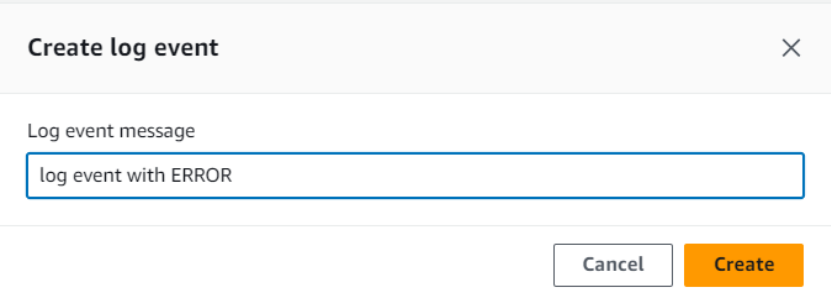




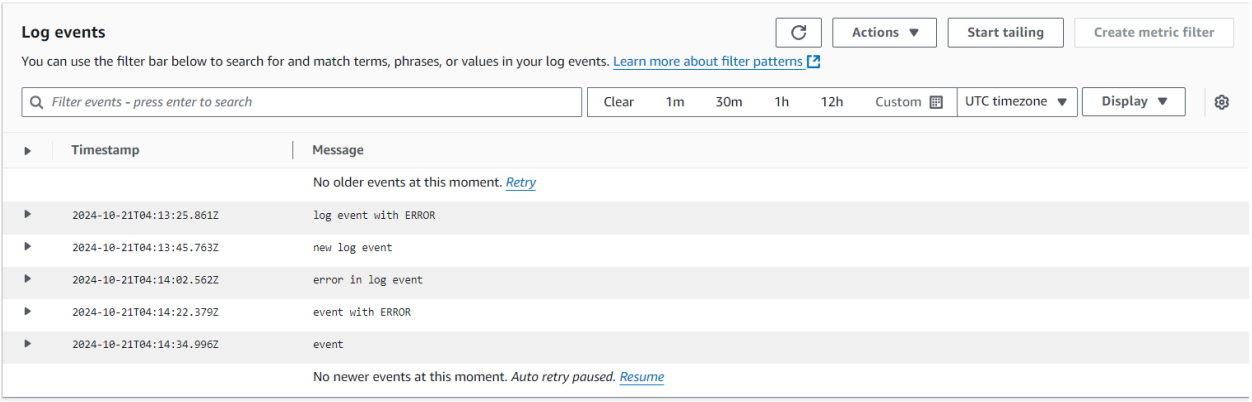
11. Click on your **Log Stream** (e.g., new-stream). You will now see an option to add log events manually. Click **Actions → Create log event**.



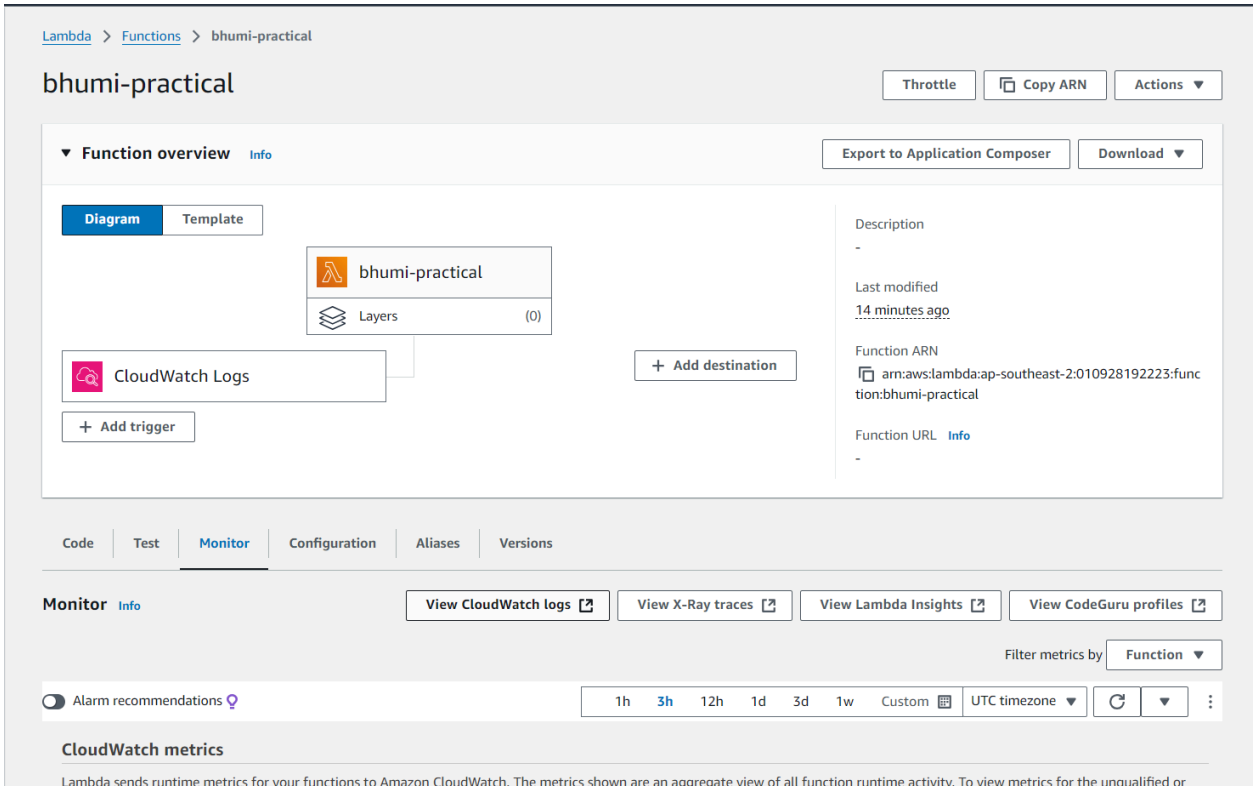
Enter the log message you want to trigger your Lambda function. Since we're filtering for the keyword ERROR, enter something like:



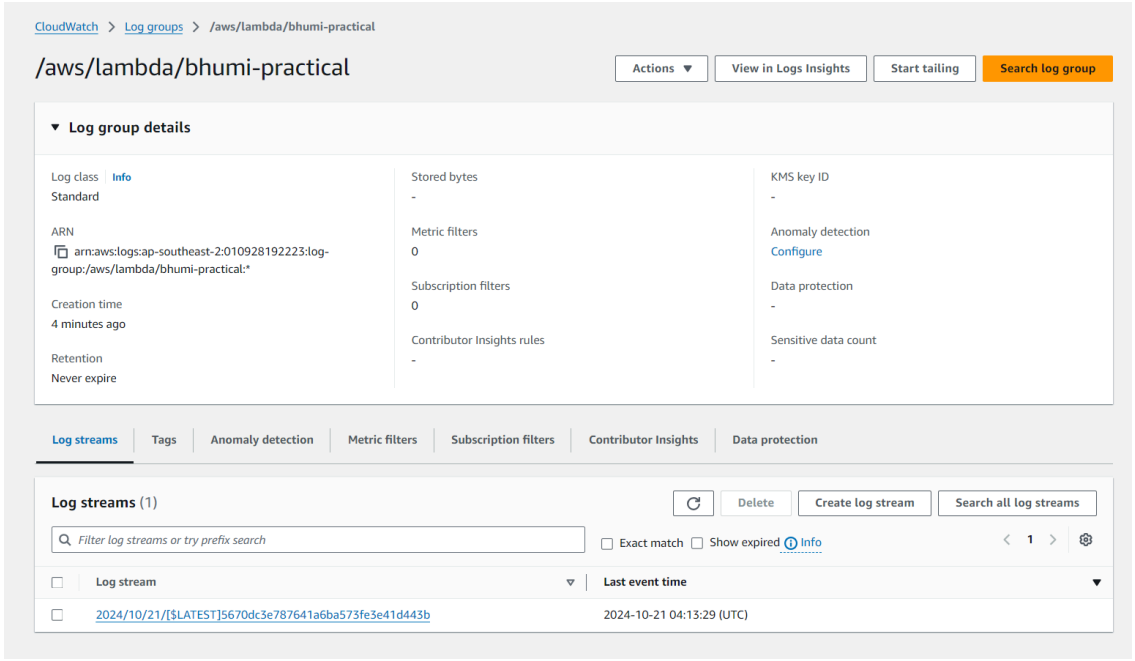
Create more log events like so with and without the keyword ERROR.



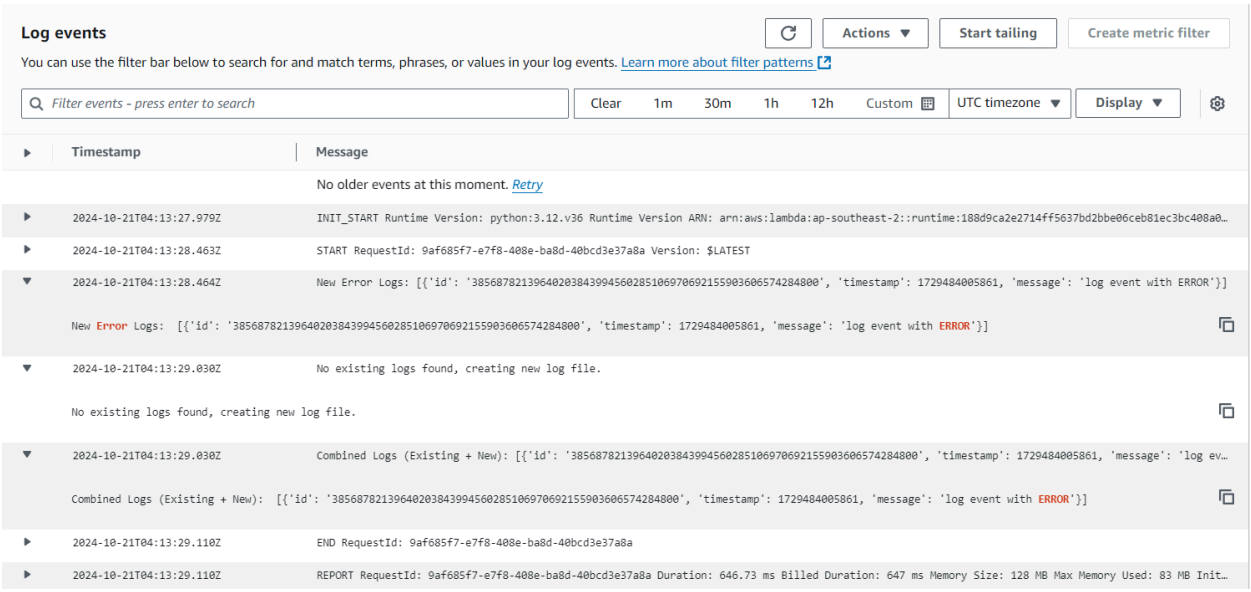
12. Find your Lambda function (the one you set up to filter logs and write to S3), and click on it. In the Lambda function dashboard, click on the **Monitoring** tab. In the Monitoring tab, click on **View logs in CloudWatch**.



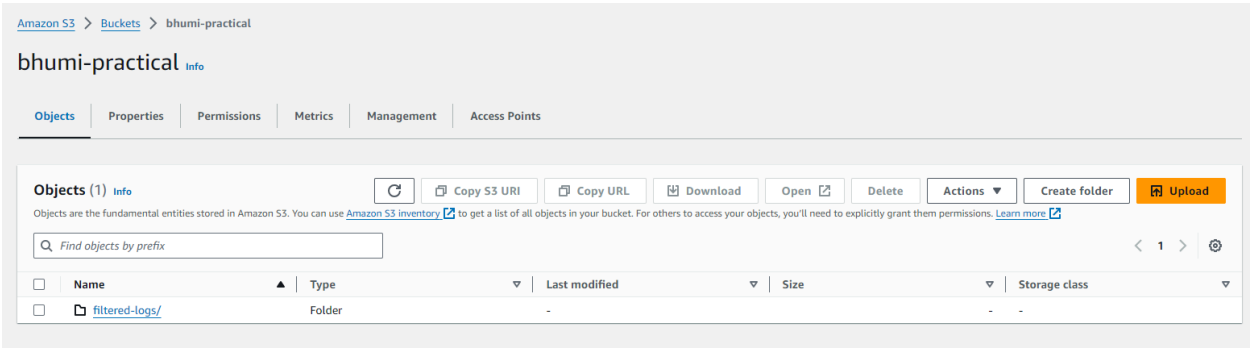
Then it will lead you to a generated log group for this function. Select the latest log stream. This will open CloudWatch and show you logs generated by your Lambda function.

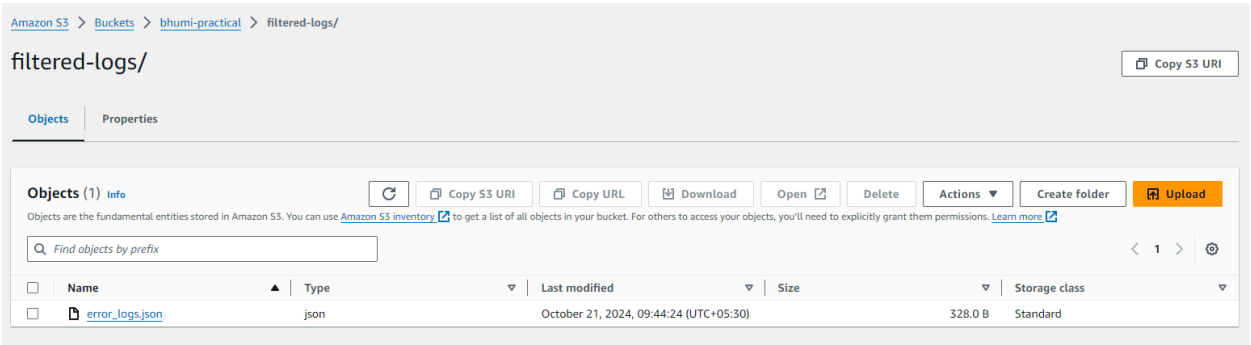


Look for recent log entries to see if the Lambda function was triggered. Check for any errors or information logs indicating that the function processed the log event and uploaded data to S3.

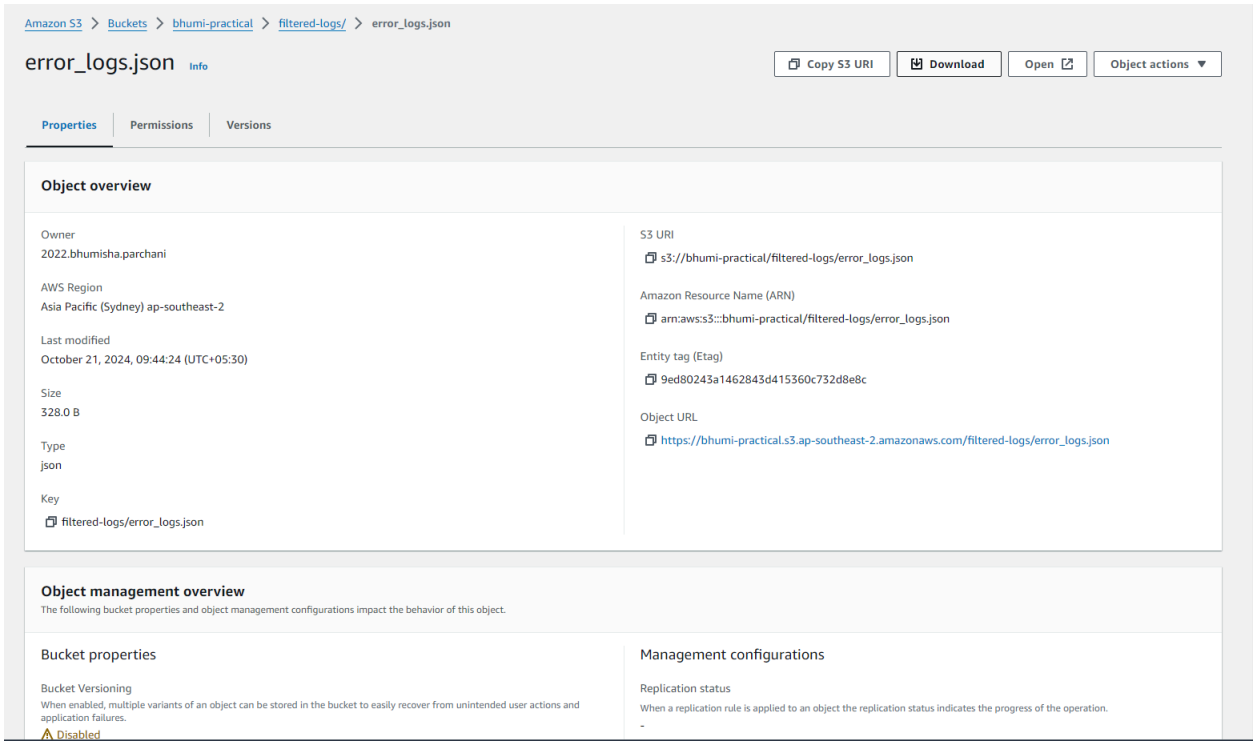


13. In the list of buckets, find and click on the bucket you configured in your Lambda function. Inside the S3 bucket, look for newly uploaded files. They should contain logs that match the filter pattern.

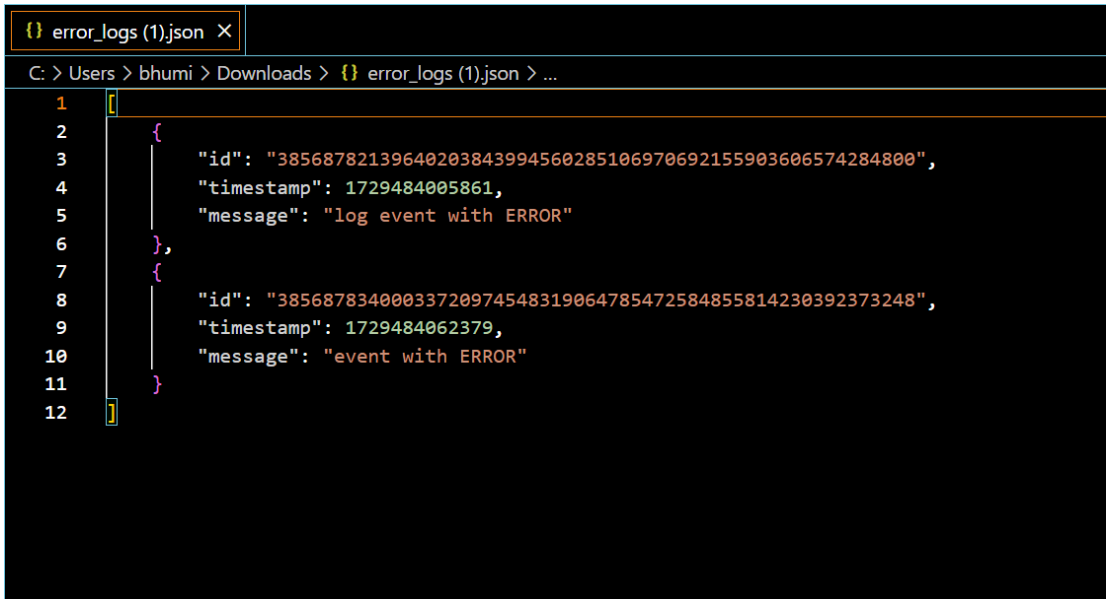




Like we configured in the code part. The logs are stored in the error_logs.json file. Download or Open the file.



In the file, we can see the records of all the logs with the keyword ERROR.



Additional Guidelines

In case the error logs are not stored or visible in the S3 bucket, here are some common ways to troubleshoot the issue:

1. **Check CloudWatch Logs**: Look for errors or warnings during the execution of the Lambda function to identify issues with log processing or permissions.
2. **Review IAM Permissions**: Ensure the Lambda function's IAM role has the correct permissions for accessing CloudWatch Logs and storing data in S3.
3. **Check the S3 Bucket Configuration**: Verify that the bucket name in the Lambda code matches the actual bucket name in your S3 account.
4. **Examine Lambda Error Logs**: Ensure that all logs are being captured and filtered correctly. Update the code logic if logs are overwritten instead of appended.
5. **Ensure Event Source Mapping**: Make sure the Lambda function is correctly set up to trigger from CloudWatch Logs or other sources.

Conclusion

This case study demonstrates how AWS services like Lambda, CloudWatch Logs, and S3 can be effectively combined to automate the error-log monitoring process. CloudWatch captures log data, which Lambda filters for errors and stores in S3 for easy retrieval. This approach enables automated log processing without the need for complex infrastructure. The case study required setting up a CloudWatch Log group, a Lambda function, an S3 bucket, and configuring appropriate permissions for the Lambda function's IAM role.

This solution highlights the simplicity and efficiency of using serverless architecture for real-time log monitoring and error tracking, making it highly scalable and cost-effective for large-scale systems.