

## 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.

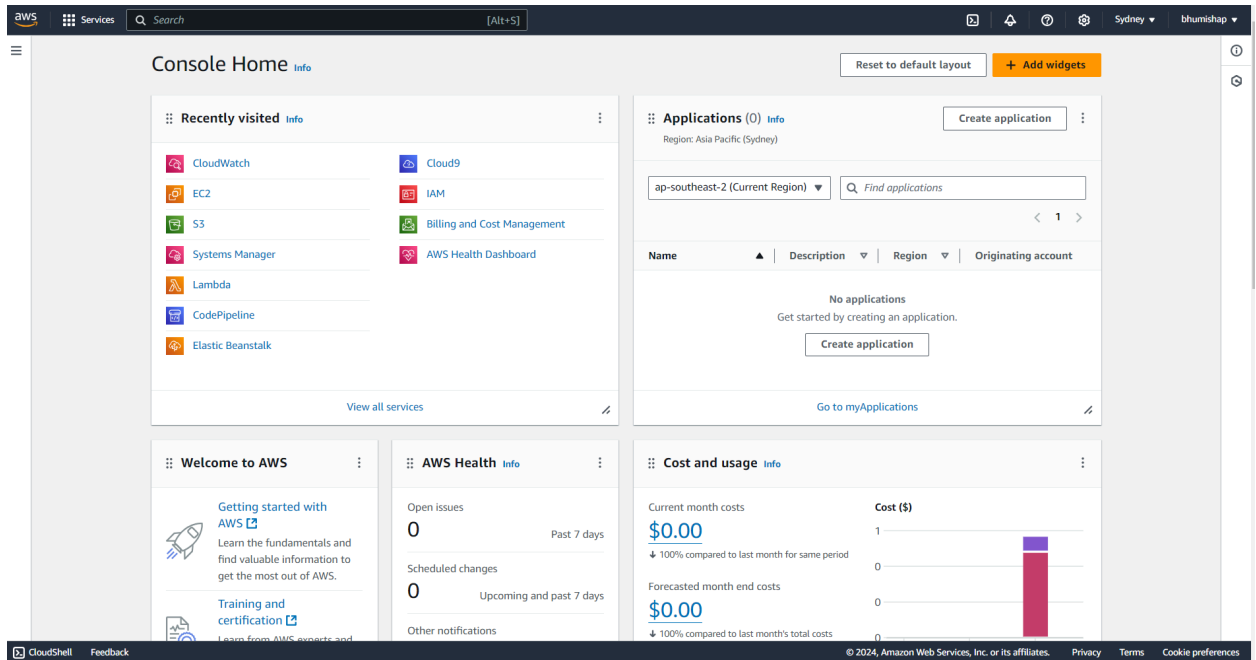
6. **Cost-effective:** Minimal infrastructure costs with serverless and S3 storage.
7. **JSON Structured Logs:** Stores logs in structured format for easy analysis.

**Applications:**

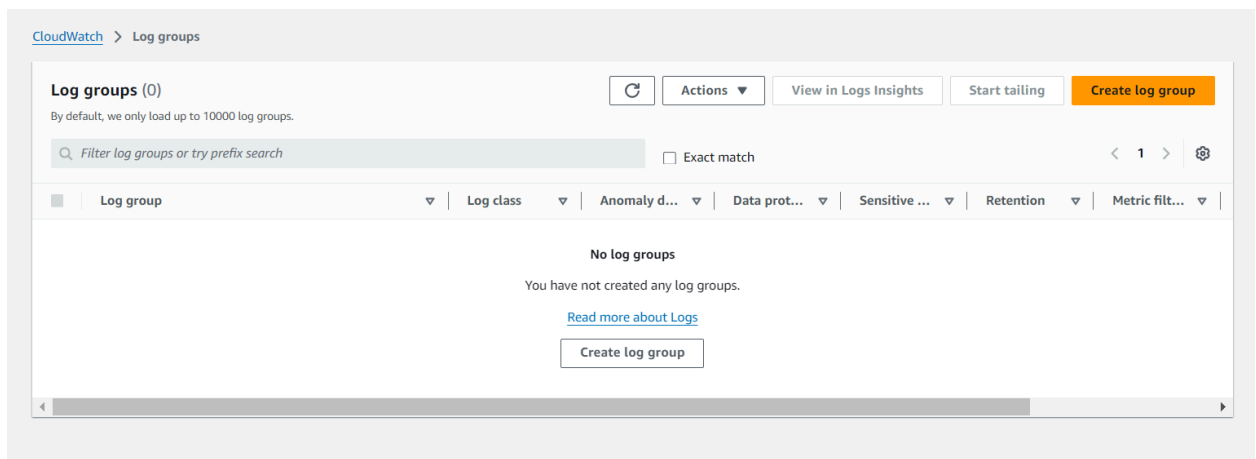
1. **Application Monitoring:** Automatically captures and filters critical error logs in real-time, improving performance tracking.
2. **System Diagnostics:** Provides centralized, structured log storage for quick issue identification.
3. **Cloud Infrastructure Management:** Scales effortlessly with 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**.

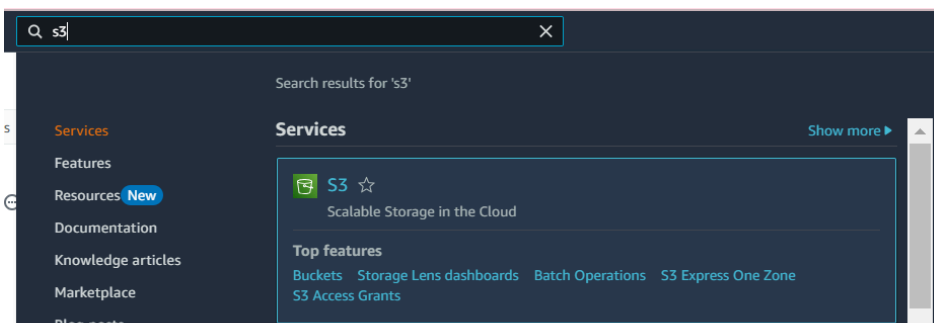
The screenshot shows the 'Create log group' page in the AWS CloudWatch console. The breadcrumb navigation is 'CloudWatch > Log groups > Create log group'. The page title is 'Create log group'. Below the title is a 'Log group details' section with an 'info' icon. A blue informational box states: 'CloudWatch Logs offers two log classes: Standard and Infrequent Access. [Learn more about the features offered by each log class.](#)'. The form fields are: 'Log group name' with the value 'bhumi-practical'; 'Retention setting' with a dropdown menu set to 'Never expire'; 'Log class' with a dropdown menu set to 'Standard'; and 'KMS key ARN - optional' with an empty text box. Below the form is a 'Tags' section with a description: 'A tag is a label that you assign to an Amazon Web Services resource. Each tag consists of a key and an optional value. You can use tags to search and filter your resources or track your Amazon Web Services costs.' It shows 'No tags are associated with this log group.' and an 'Add new tag' button. At the bottom right are 'Cancel' and 'Create' buttons.

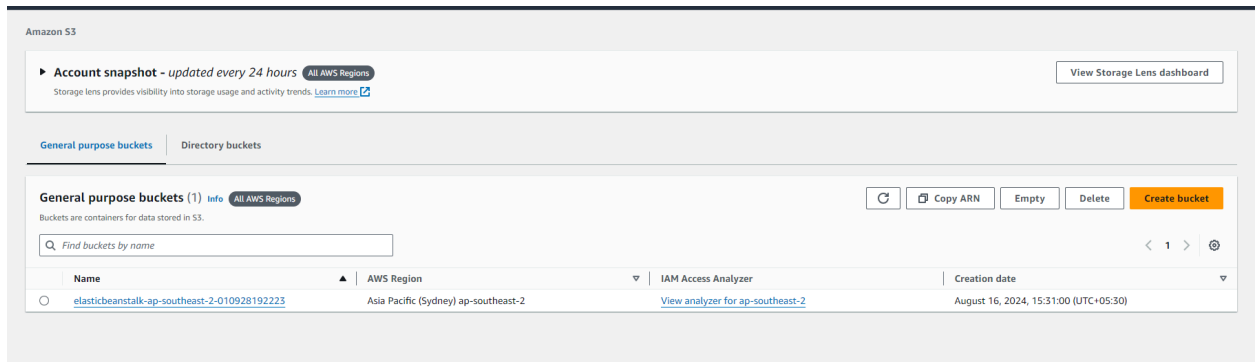
We can see that our log group has been created.

The screenshot shows the 'Log groups' page in the AWS CloudWatch console. The breadcrumb navigation is 'CloudWatch > Log groups'. The page title is 'Log groups (3)'. Below the title is a search bar with the placeholder 'Filter log groups or try prefix search' and an 'Exact match' checkbox. There are buttons for 'Actions', 'View in Logs Insights', 'Start tailing', and 'Create log group'. A table lists the log groups:

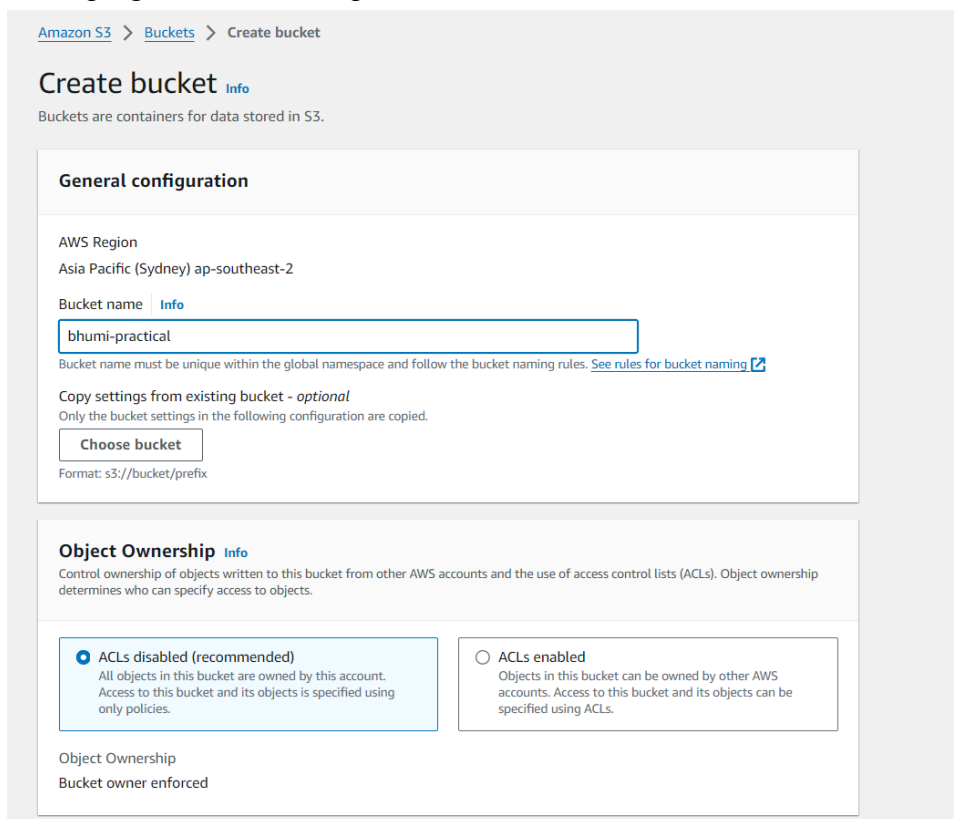
<input type="checkbox"/>	Log group	Log class	Anomaly d...	Data protection	Sensitive data c...	Retention	Metric filters	Contributo
<input type="checkbox"/>	<a href="#">/aws/lambda/practice-function</a>	Standard	<a href="#">Configure</a>	-	-	<a href="#">Never expire</a>	-	-
<input type="checkbox"/>	<a href="#">practical</a>	Standard	<a href="#">Configure</a>	-	-	<a href="#">Never expire</a>	-	-
<input type="checkbox"/>	<a href="#">bhumi-practical</a>	Standard	<a href="#">Configure</a>	-	-	<a href="#">Never expire</a>	-	-

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**.



**Block Public Access settings for this bucket**

Public access is granted to buckets and objects through access control lists (ACLs), bucket policies, access point policies, or all. In order to ensure that public access to this bucket and its objects is blocked, turn on Block all public access. These settings apply only to this bucket and its access points. AWS recommends that you turn on Block all public access, but before applying any of these settings, ensure that your applications will work correctly without public access. If you require some level of public access to this bucket or objects within, you can customize the individual settings below to suit your specific storage use cases. [Learn more](#)

☐ **Block all public access**

Turning this setting on is the same as turning on all four settings below. Each of the following settings are independent of one another.

- ☐ **Block public access to buckets and objects granted through new access control lists (ACLs)**  
S3 will block public access permissions applied to newly added buckets or objects, and prevent the creation of new public access ACLs for existing buckets and objects. This setting doesn't change any existing permissions that allow public access to S3 resources using ACLs.
- ☐ **Block public access to buckets and objects granted through any access control lists (ACLs)**  
S3 will ignore all ACLs that grant public access to buckets and objects.
- ☐ **Block public access to buckets and objects granted through new public bucket or access point policies**  
S3 will block new bucket and access point policies that grant public access to buckets and objects. This setting doesn't change any existing policies that allow public access to S3 resources.
- ☐ **Block public and cross-account access to buckets and objects through any public bucket or access point policies**  
S3 will ignore public and cross-account access for buckets or access points with policies that grant public access to buckets and objects.



**Turning off block all public access might result in this bucket and the objects within becoming public**  
AWS recommends that you turn on block all public access, unless public access is required for specific and verified use cases such as static website hosting.

- ☒ I acknowledge that the current settings might result in this bucket and the objects within becoming public.

**Bucket Versioning**

Versioning is a means of keeping multiple variants of an object in the same bucket. You can use versioning to preserve, retrieve, and restore every version of every object stored in your Amazon S3 bucket. With versioning, you can easily recover from both unintended user actions and application failures. [Learn more](#)

Bucket Versioning

- ☒ Disable  
☐ Enable

**Tags - optional (0)**

You can use bucket tags to track storage costs and organize buckets. [Learn more](#)

No tags associated with this bucket.

Add tag

**Default encryption** [Info](#)

Server-side encryption is automatically applied to new objects stored in this bucket.

Encryption type [Info](#)

- ☒ Server-side encryption with Amazon S3 managed keys (SSE-S3)  
☐ Server-side encryption with AWS Key Management Service keys (SSE-KMS)  
☐ Dual-layer server-side encryption with AWS Key Management Service keys (DSSE-KMS)  
Secure your objects with two separate layers of encryption. For details on pricing, see [DSSE-KMS pricing](#) on the [Storage](#) tab of the [Amazon S3 pricing page](#).

**Bucket Key**

Using an S3 Bucket Key for SSE-KMS reduces encryption costs by lowering calls to AWS KMS. S3 Bucket Keys aren't supported for DSSE-KMS. [Learn more](#)

- ☐ Disable  
☒ Enable

We can see that our bucket has been created.

**General purpose buckets (3)** [Info](#) [All AWS Regions](#)

Buckets are containers for data stored in S3.

Find buckets by name

Name	AWS Region	IAM Access Analyzer	Creation date
<a href="#">bhumi-practical</a>	Asia Pacific (Sydney) ap-southeast-2	<a href="#">View analyzer for ap-southeast-2</a>	October 21, 2024, 09:15:21 (UTC+05:30)
<a href="#">elasticbeanstalk-ap-southeast-2-010928192223</a>	Asia Pacific (Sydney) ap-southeast-2	<a href="#">View analyzer for ap-southeast-2</a>	August 16, 2024, 15:31:00 (UTC+05:30)
<a href="#">practical-bucket3</a>	Asia Pacific (Sydney) ap-southeast-2	<a href="#">View analyzer for ap-southeast-2</a>	October 20, 2024, 15:33:13 (UTC+05:30)

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

[Lambda](#) > Functions

**Functions (1)** Last fetched 10 seconds ago [Refresh](#) [Actions](#) [Create function](#)

Filter by tags and attributes or search by keyword

<input type="checkbox"/>	Function name	Description	Package type	Runtime	Last modified
<input type="checkbox"/>	<a href="#">practice-function</a>	-	Zip	Python 3.12	17 hours ago

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

**Create function** [Info](#)

Choose one of the following options to create your function.

☒ **Author from scratch**  
Start with a simple Hello World example.

☐ **Use a blueprint**  
Build a Lambda application from sample code and configuration presets for common use cases.

☐ **Container image**  
Select a container image to deploy for your function.

---

**Basic information**

**Function name**  
Enter a name that describes the purpose of your function.  
  
 Function name must be 1 to 64 characters, must be unique to the Region, and can't include spaces. Valid characters are a-z, A-Z, 0-9, hyphens (-), and underscores (\_).

**Runtime** [Info](#)  
Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.  
 [Refresh](#)

**Architecture** [Info](#)  
Choose the instruction set architecture you want for your function code.  
☒ x86\_64  
☐ arm64

**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.

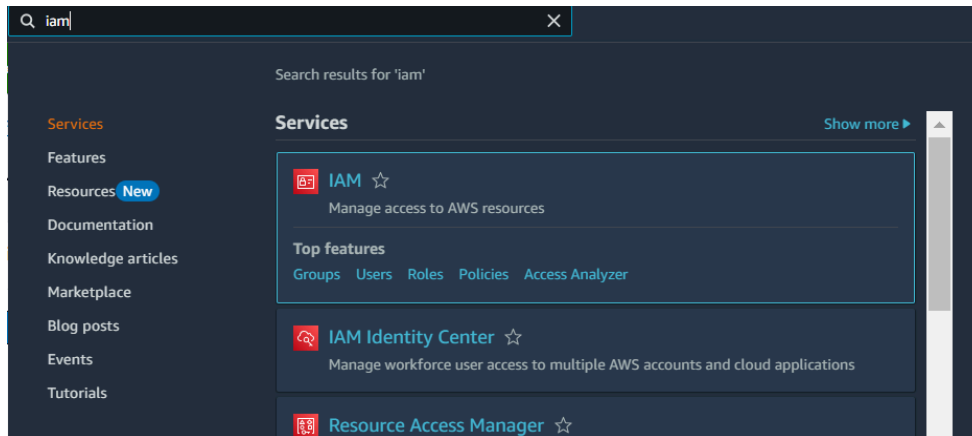
[Change default execution role](#)

---

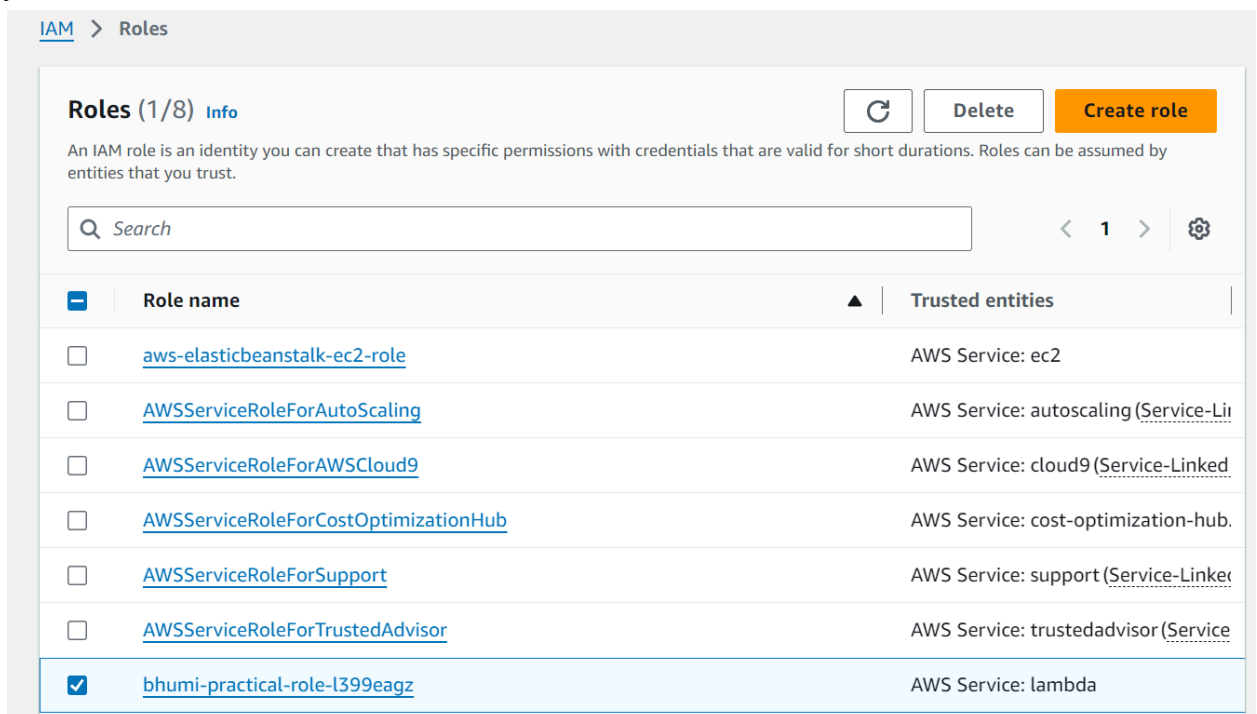
**Additional Configurations**  
Use additional configurations to set up code signing, function URL, tags, and Amazon VPC access for your function.

[Cancel](#) [Create function](#)

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

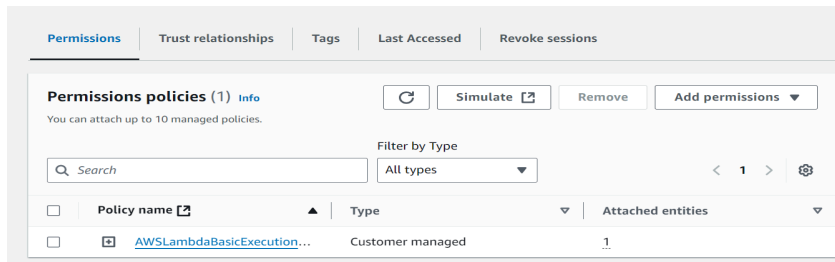


Here, you can see the role associated with your lambda function. It has the same name as your lambda function with role. Click on the name.

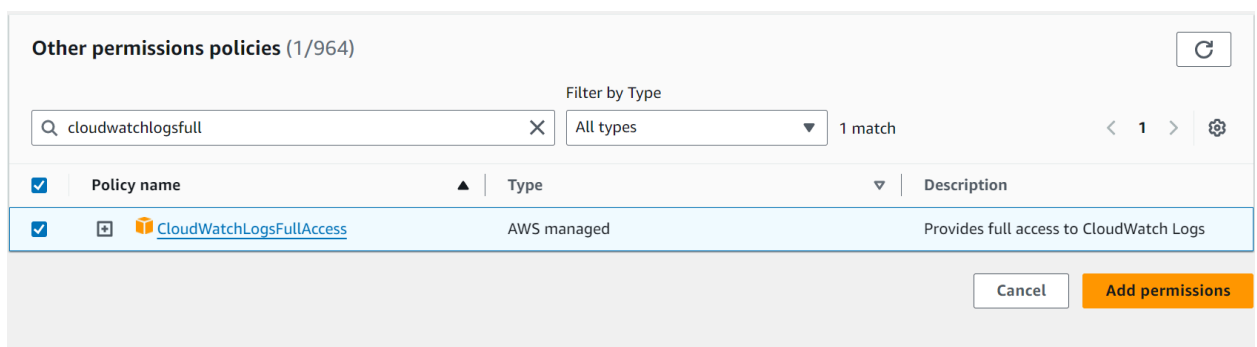
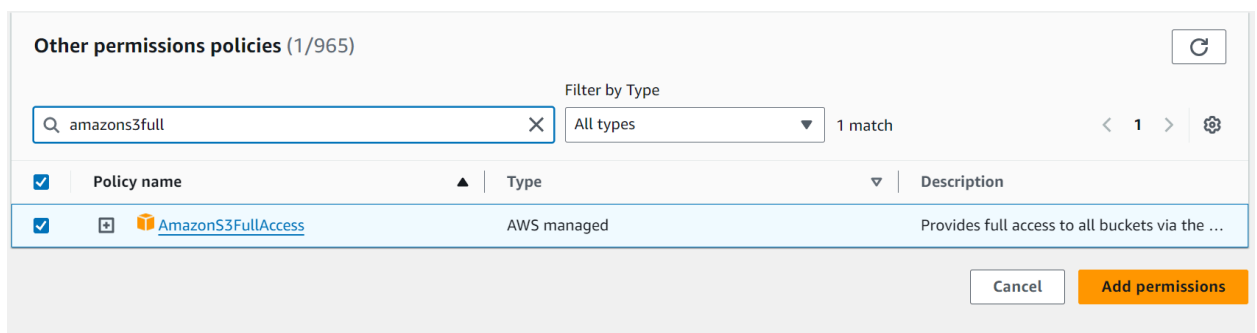
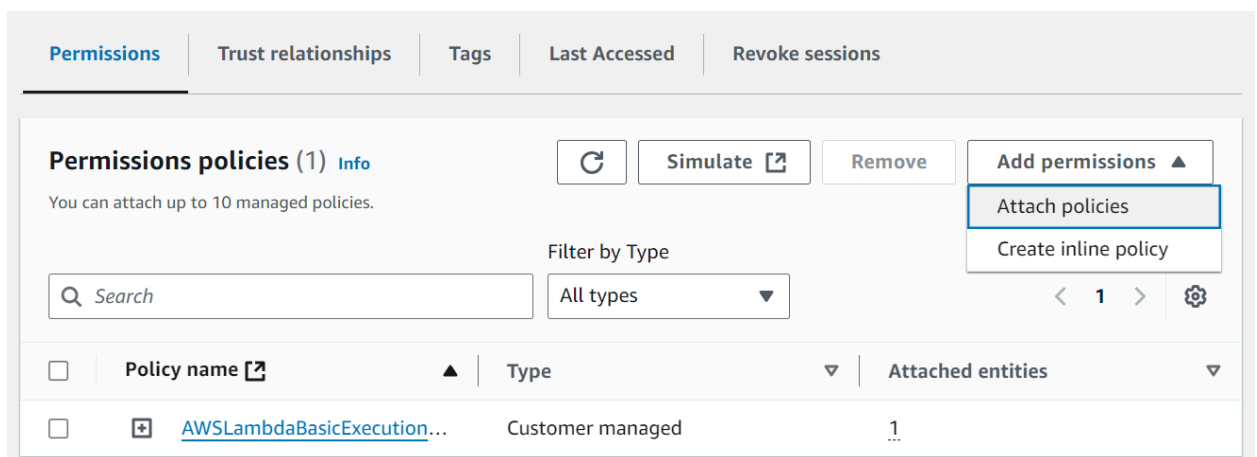


Next, scroll down to find the permissions section like shown in the screenshot.





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).



Policy was successfully attached to role.

Permissions Trust relationships Tags Last Accessed Revoke sessions

**Permissions policies (3)** [Info](#) [Refresh](#) [Simulate](#) [Remove](#) [Add permissions](#)

You can attach up to 10 managed policies.

Filter by Type

All types < 1 > [Settings](#)

<input type="checkbox"/>	Policy name <a href="#">↗</a>	Type	Attached entities
<input type="checkbox"/>	<a href="#">AmazonS3FullAccess</a>	AWS managed	2
<input type="checkbox"/>	<a href="#">AWSLambdaBasicExecution...</a>	Customer managed	1
<input type="checkbox"/>	<a href="#">CloudWatchLogsFullAcc...</a>	AWS managed	2

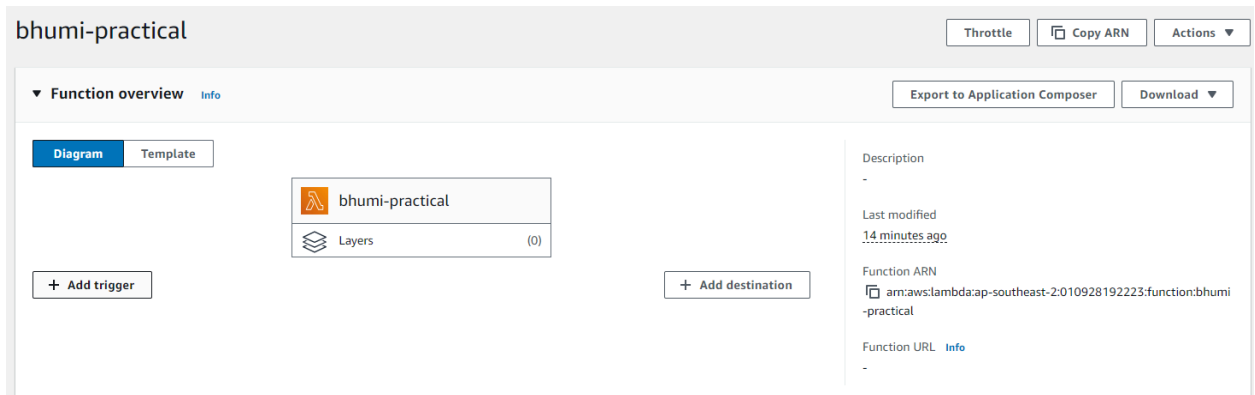
7. Navigate back to the lambda function, in the code section we add the following code. This Lambda function will filter logs containing the keyword "ERROR" and store them in an S3 bucket.

```

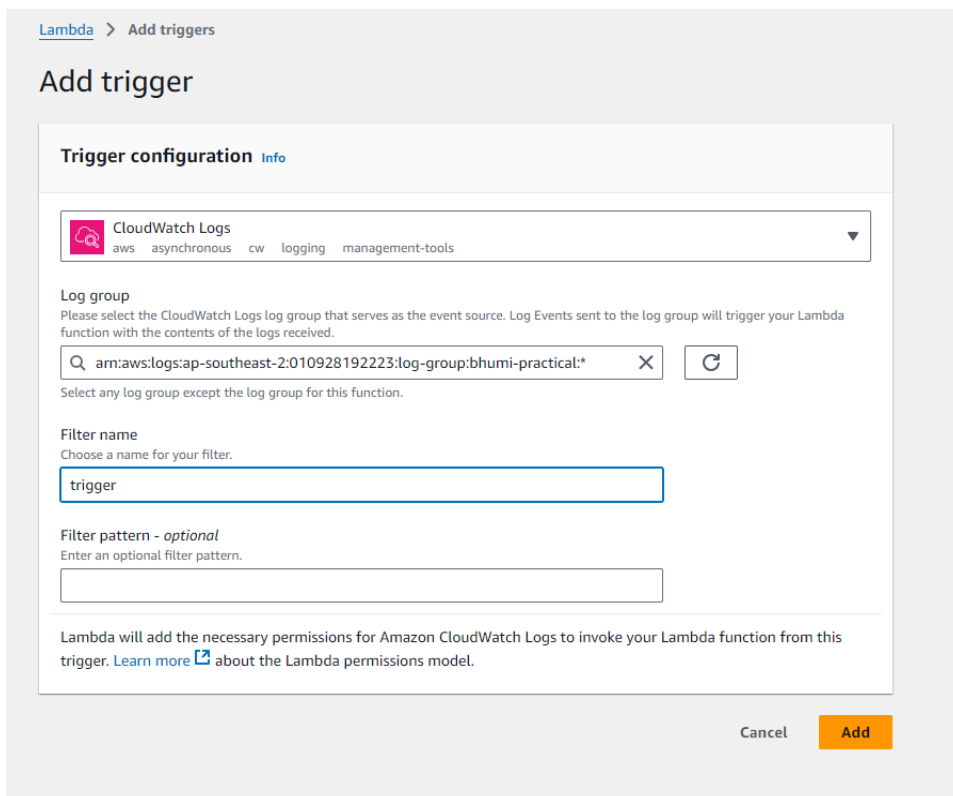
File Edit Find View Go Tools Window Test Deploy Changes not deployed
Go to Anything (Ctrl-P)
Environment
  bhumi-practical
    lambda_function.py
lambda_function
Environment Var
1 import gzip
2 import json
3 import base64
4 import boto3
5
6 s3 = boto3.client('s3')
7 bucket_name = 'bhumi-practical'
8 file_key = 'filtered-logs/error_logs.json'
9
10 def lambda_handler(event, context):
11     # Cloudwatch Logs data is base64 and gzip compressed
12     cw_data = event['awslogs']['data']
13
14     # Decode the Cloudwatch Logs data
15     compressed_payload = base64.b64decode(cw_data)
16     uncompressed_payload = gzip.decompress(compressed_payload)
17
18     # Convert the uncompressed payload to a string and then to a JSON object
19     log_data = json.loads(uncompressed_payload)
20
21     # Filter logs that contain the word "ERROR"
22     error_logs = [event for event in log_data['logEvents'] if 'ERROR' in event['message']]
23
24     print("New Error Logs: ", error_logs)
25
26     if error_logs:
27         try:
28             # Try to get the existing error_logs.json file from S3
29             response = s3.get_object(Bucket=bucket_name, Key=file_key)
30             existing_logs = json.loads(response['Body'].read().decode('utf-8'))
31             print("Existing Logs from S3: ", existing_logs)
32         except s3.exceptions.NoSuchKey:
33             # If the file doesn't exist, initialize an empty list
34             existing_logs = []
35             print("No existing logs found, creating new log file.")
36
37     # Append the new error logs to the existing logs
38     existing_logs.extend(error_logs)
39     print("Combined Logs (Existing + New): ", existing_logs)
40
41     # Convert the updated logs to string format
42     error_logs_str = json.dumps(existing_logs, indent=4)
43
44     # Store the updated logs in the S3 bucket
45     s3.put_object(
46         Bucket=bucket_name,
47         Key=file_key,
48         Body=error_logs_str
49     )
50
51     return {
52         'statusCode': 200,
53         'body': json.dumps('Logs processed successfully')
54     }

```

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**.

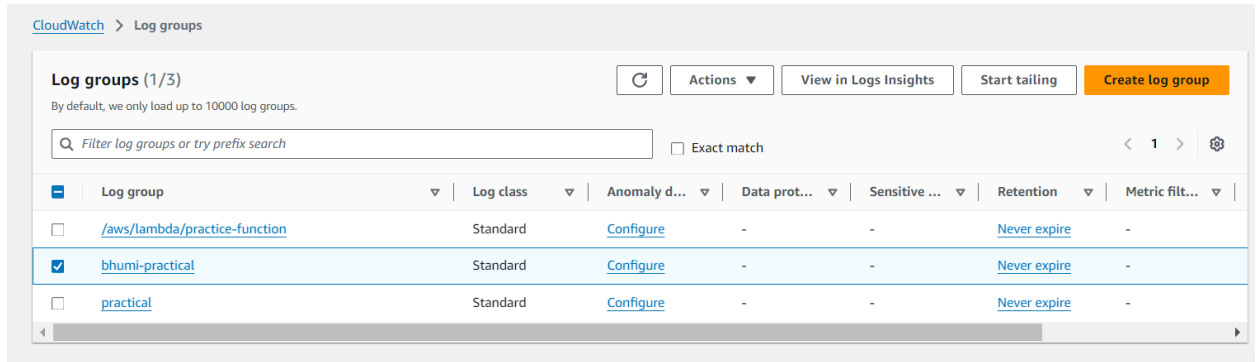
The screenshot shows the Amazon S3 console interface for a bucket named 'bhumi-practical'. The 'Permissions' tab is selected, displaying the 'Permissions overview' section. Below this, there are sections for 'Block public access (bucket settings)' and 'Bucket policy'. The 'Bucket policy' section shows a policy that is currently empty, with a 'No policy to display.' message and a 'Copy' button.

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.

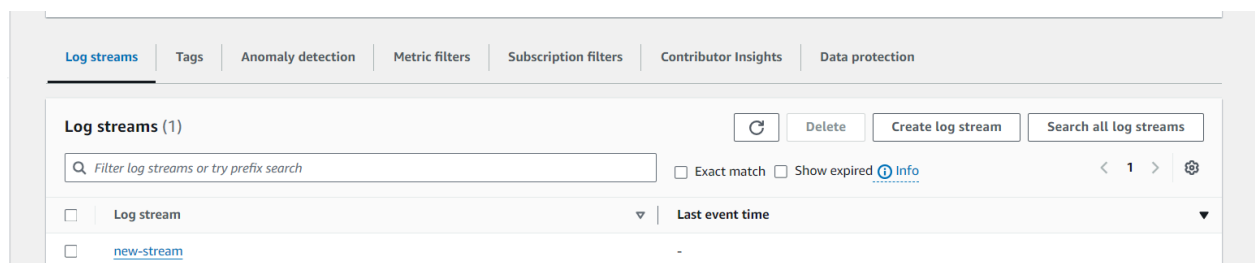
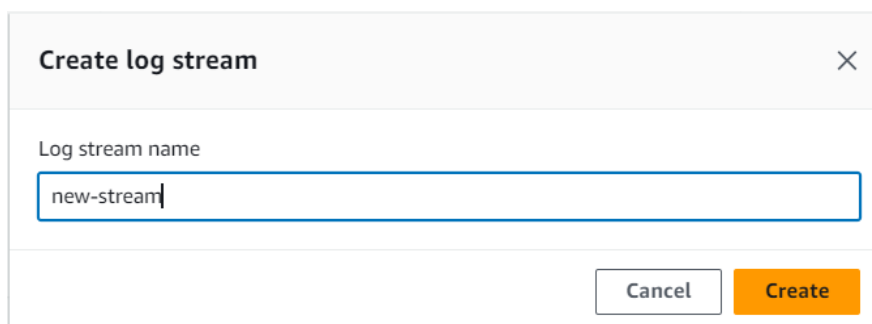
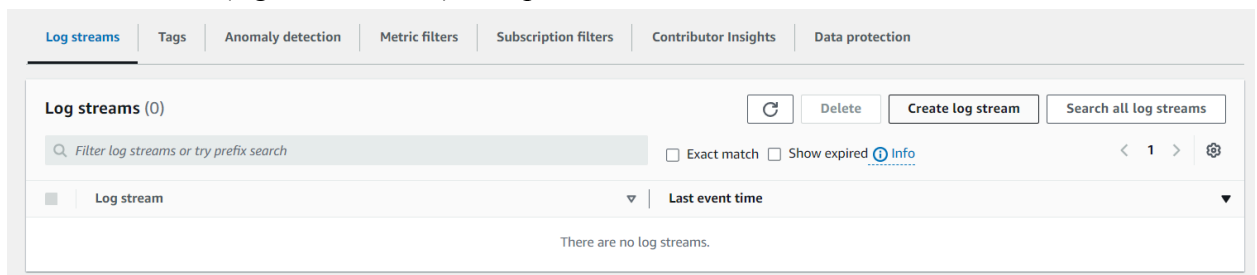
The screenshot shows the 'Edit bucket policy' page in the Amazon S3 console. The 'Bucket policy' section is visible, showing the bucket ARN as 'arn:aws:s3:::bhumi-practical'. Below this, the 'Policy' section displays a JSON policy document in a code editor. The policy grants 'PutObject' permission to the 'arn:aws:iam::010928192223:role/service-role/bhumi-practical-role-l399eagz' role.

```
1 {  
2   "Version": "2012-10-17",  
3   "Statement": [  
4     {  
5       "Effect": "Allow",  
6       "Principal": {  
7         "AWS": "arn:aws:iam::010928192223:role/service-role/bhumi-practical-role-l399eagz"  
8       },  
9       "Action": "s3:PutObject",  
10      "Resource": "arn:aws:s3:::bhumi-practical/*"  
11    }  
12  ]  
13 }
```

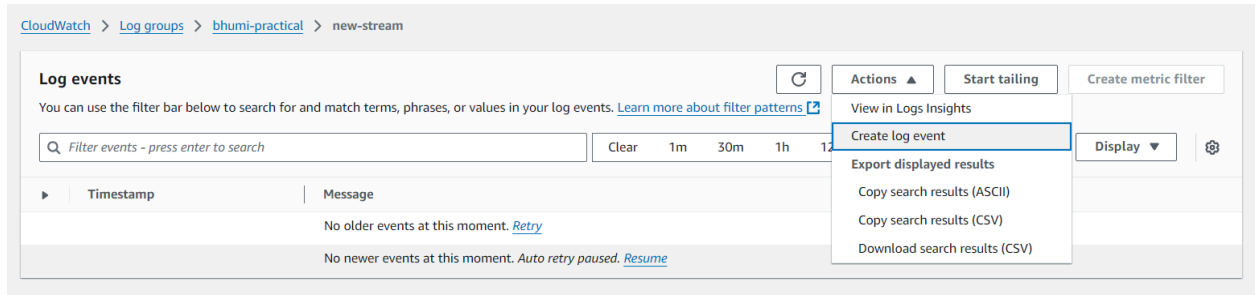
10. On the CloudWatch dashboard, on the left sidebar, click on **Logs** → **Log Groups**. This will display all the Log Groups associated with your AWS account. Select the one associated with your lambda function.



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 log event

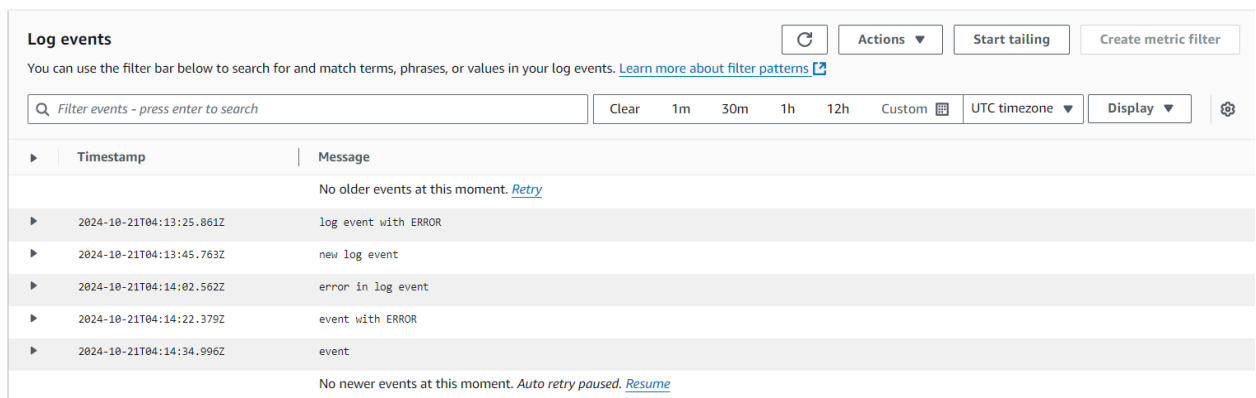
Log event message

log event with ERROR

Cancel

Create

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**.

Lambda > Functions > bhumi-practical

## bhumi-practical

Throttle Copy ARN Actions

Function overview Info

Export to Application Composer Download

Diagram Template

bhumi-practical

Layers (0)

CloudWatch Logs

+ Add trigger

+ Add destination

Description

-

Last modified

14 minutes ago

Function ARN

arn:aws:lambda:ap-southeast-2:010928192223:function:bhumi-practical

Function URL Info

-

Code Test Monitor Configuration Aliases Versions

Monitor Info

View CloudWatch logs View X-Ray traces View Lambda Insights View CodeGuru profiles

Filter metrics by Function

Alarm recommendations

1h 3h 12h 1d 3d 1w Custom UTC timezone

### CloudWatch metrics

Lambda sends runtime metrics for your functions to Amazon CloudWatch. The metrics shown are an aggregate view of all function runtime activity. To view metrics for the unqualified or

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.

CloudWatch > Log groups > /aws/lambda/bhumi-practical

## /aws/lambda/bhumi-practical

Actions View in Logs Insights Start tailing Search log group

Log group details

Log class Standard	Stored bytes -	KMS key ID -
ARN arn:aws:logs:ap-southeast-2:010928192223:log-group:/aws/lambda/bhumi-practical:*	Metric filters 0	Anomaly detection Configure
Creation time 4 minutes ago	Subscription filters 0	Data protection -
Retention Never expire	Contributor Insights rules -	Sensitive data count -

Log streams Tags Anomaly detection Metric filters Subscription filters Contributor Insights Data protection

Log streams (1)

Filter log streams or try prefix search

Exact match Show expired Info

Log stream	Last event time
2024/10/21/[LATEST]5670dc3e787641a6ba573fe3e41d443b	2024-10-21 04:13:29 (UTC)

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.

**Log events** [Refresh](#) [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](#)

[Clear](#) [1m](#) [30m](#) [1h](#) [12h](#) [Custom](#) [UTC timezone](#) [Display](#) [Settings](#)

Timestamp	Message
No older events at this moment. <a href="#">Retry</a>	
2024-10-21T04:13:27.979Z	INIT_START Runtime Version: python:3.12.v36 Runtime Version ARN: arn:aws:lambda:ap-southeast-2::runtime:188d9ca2e2714ff5637bd2bbe06ceb81ec3bc408a0...
2024-10-21T04:13:28.463Z	START RequestId: 9af685f7-e7f8-408e-ba8d-40bcd3e37a8a Version: \$LATEST
2024-10-21T04:13:28.464Z	New Error Logs: [{"id": "38568782139640203843994560285106970692155903606574284800", "timestamp": "1729484005861", "message": "log event with ERROR"}]
New Error Logs: [{"id": "38568782139640203843994560285106970692155903606574284800", "timestamp": "1729484005861", "message": "log event with ERROR"}]	
2024-10-21T04:13:29.030Z	No existing logs found, creating new log file.
No existing logs found, creating new log file.	
2024-10-21T04:13:29.030Z	Combined Logs (Existing + New): [{"id": "38568782139640203843994560285106970692155903606574284800", "timestamp": "1729484005861", "message": "log ev..."}]
Combined Logs (Existing + New): [{"id": "38568782139640203843994560285106970692155903606574284800", "timestamp": "1729484005861", "message": "log event with ERROR"}]	
2024-10-21T04:13:29.110Z	END RequestId: 9af685f7-e7f8-408e-ba8d-40bcd3e37a8a
2024-10-21T04:13:29.110Z	REPORT RequestId: 9af685f7-e7f8-408e-ba8d-40bcd3e37a8a Duration: 646.73 ms Billed Duration: 647 ms Memory Size: 128 MB Max Memory Used: 83 MB Init...

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.

Amazon S3 > Buckets > bhumi-practical

**bhumi-practical** [Info](#)

[Objects](#) [Properties](#) [Permissions](#) [Metrics](#) [Management](#) [Access Points](#)

**Objects (1)** [Info](#) [Refresh](#) [Copy S3 URI](#) [Copy URL](#) [Download](#) [Open](#) [Delete](#) [Actions](#) [Create folder](#) [Upload](#)

Objects are the fundamental entities stored in Amazon S3. You can use [Amazon S3 inventory](#) to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. [Learn more](#)

<input type="checkbox"/>	Name	Type	Last modified	Size	Storage class
<input type="checkbox"/>	<a href="#">filtered-logs/</a>	Folder	-	-	-

Amazon S3 > Buckets > bhumi-practical > filtered-logs/

**filtered-logs/** [Copy S3 URI](#)

[Objects](#) [Properties](#)

**Objects (1)** [Info](#) [Refresh](#) [Copy S3 URI](#) [Copy URL](#) [Download](#) [Open](#) [Delete](#) [Actions](#) [Create folder](#) [Upload](#)

Objects are the fundamental entities stored in Amazon S3. You can use [Amazon S3 inventory](#) to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. [Learn more](#)

<input type="checkbox"/>	Name	Type	Last modified	Size	Storage class
<input type="checkbox"/>	<a href="#">error_logs.json</a>	json	October 21, 2024, 09:44:24 (UTC+05:30)	328.0 B	Standard



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

The screenshot shows the Amazon S3 console interface for the object `error_logs.json`. The breadcrumb navigation is `Amazon S3 > Buckets > bhumi-practical > filtered-logs/ > error_logs.json`. The object name `error_logs.json` is highlighted, with an `Info` link. Action buttons include `Copy S3 URI`, `Download`, `Open`, and `Object actions`. The `Properties` tab is active, displaying the following details:

Object overview	
Owner 2022.bhumisha.parchani	S3 URI <code>s3://bhumi-practical/filtered-logs/error_logs.json</code>
AWS Region Asia Pacific (Sydney) ap-southeast-2	Amazon Resource Name (ARN) <code>arn:aws:s3:::bhumi-practical/filtered-logs/error_logs.json</code>
Last modified October 21, 2024, 09:44:24 (UTC+05:30)	Entity tag (Etag) <code>9ed80243a1462843d415360c732d8e8c</code>
Size 328.0 B	Object URL <code>https://bhumi-practical.s3.ap-southeast-2.amazonaws.com/filtered-logs/error_logs.json</code>
Type json	
Key <code>filtered-logs/error_logs.json</code>	

Below the overview, the `Object management overview` section states: "The following bucket properties and object management configurations impact the behavior of this object."

Bucket properties	Management configurations
Bucket Versioning When enabled, multiple variants of an object can be stored in the bucket to easily recover from unintended user actions and application failures. <b>Disabled</b>	Replication status When a replication rule is applied to an object the replication status indicates the progress of the operation. -

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

The screenshot shows a code editor with the file `error_logs (1).json` open. The file path is `C: > Users > bhumi > Downloads > error_logs (1).json > ...`. The JSON content is as follows:

```
1 [
2   {
3     "id": "38568782139640203843994560285106970692155903606574284800",
4     "timestamp": 1729484005861,
5     "message": "log event with ERROR"
6   },
7   {
8     "id": "38568783400033720974548319064785472584855814230392373248",
9     "timestamp": 1729484062379,
10    "message": "event with ERROR"
11  }
12 ]
```

## 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.