Shravani Anil Patil **D15A-38**

ADVANCED DEVOPS
CASE STUDY

I. Introduction:

In this case study on Real-Time Log Processing, the focus is on utilizing AWS services, specifically Lambda, CloudWatch, and S3, to address a log management challenge. The main objective was to establish an AWS Lambda function that triggers whenever a new log entry is added to a designated CloudWatch Log Group. This Lambda function, written in Python, filters log events based on a specified keyword, such as 'ERROR', and subsequently stores these filtered logs in an S3 bucket for further analysis and storage. This setup ensures efficient log management while providing real-time alerting and storage solutions, leveraging the seamless integration of AWS services.

Concepts Used: AWS Lambda, CloudWatch, S3.

AWS Lambda

AWS Lambda is a serverless computing service that allows you to run code without provisioning or managing servers. It automatically scales your applications by running code in response to triggers such as changes in data, updates to databases, or HTTP requests. With Lambda, you can focus on writing your application logic without worrying about the underlying infrastructure. It’s highly cost-effective since you only pay for the compute time you consume. Lambda functions can be written in various programming languages, including Python, Java, and Node.js.

CloudWatch Log Group

CloudWatch Logs is part of Amazon CloudWatch, which provides monitoring and observability for AWS resources and applications. A Log Group in CloudWatch is a collection of log streams that share the same settings, such as retention, monitoring, and access control. Log streams are sequences of log events that share the same source, for example, log entries from a specific application or service. By using log groups, you can organize and manage your logs more effectively, set retention policies, and configure alarms to notify you of specific events.

Amazon S3

Amazon S3 (Simple Storage Service) is a scalable object storage service designed for a wide range of use cases, including data storage, backup and restore, archiving, and big data analytics. S3 provides a secure and highly available environment to store any amount of data from anywhere. You organize your data in buckets, which can hold an unlimited number of objects. S3 supports features like versioning, lifecycle policies, and cross-region replication to ensure data durability and availability. It integrates seamlessly with other AWS services, making it a cornerstone for many cloud-native applications

Key Feature and Application: Real-Time Log Processing

Unique Feature:

The ability to process logs in real-time, filtering for specific events (e.g., ERROR messages) as they are generated, and storing the relevant log entries in an S3 bucket.

Practical Use:

This feature enables quick detection and response to critical system events, such as errors or security breaches. By filtering and storing error logs in real-time, DevOps teams can efficiently track, manage, and address issues before they escalate, without manually sifting through large volumes of logs. This automated workflow streamlines troubleshooting and auditing processes, improving system reliability and uptime.

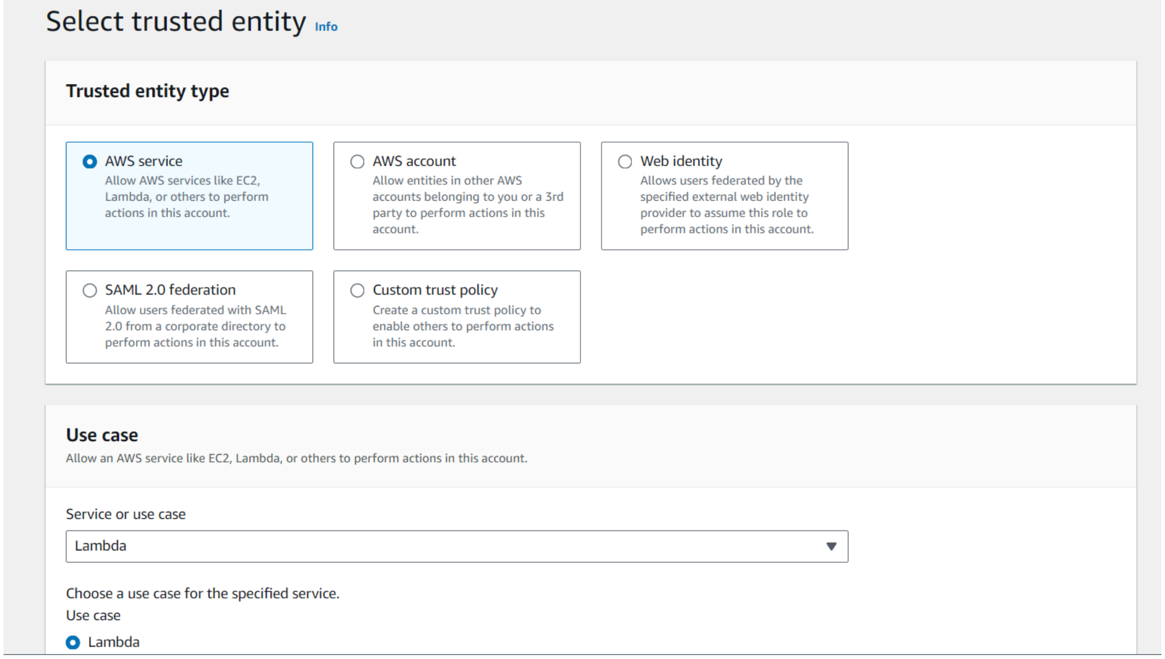
II. EXPLANATION

Step 1:Create a new IAM role

The first step in Real time log processing is to create an IAM role with necessary permissions

This user is essential for securely managing access to AWS services required for the project. By creating a dedicated IAM user, we can assign specific roles and permissions, ensuring that the Lambda function, CloudWatch Logs, and S3 bucket have the appropriate access and we don't face any permission related issues later on.

Add use case as Lambda
Add these policies to the permissions
AmazonS3FullAccess
AWSLambdaBasicExecutionRole
CloudWatchLogsFullAccess



Select trusted entity

Trusted entity type

● AWS service
Allow AWS services like EC2, Lambda, or others to perform actions in this account.

○ AWS account
Allow entities in other AWS accounts belonging to you or a 3rd party to perform actions in this account.

○ Web identity
Allows users federated by the specified external web identity provider to assume this role to perform actions in this account.

○ SAML 2.0 federation
Allow users federated with SAML 2.0 from a corporate directory to perform actions in this account.

○ Custom trust policy
Create a custom trust policy to enable others to perform actions in this account.

Use case

Allow an AWS service like EC2, Lambda, or others to perform actions in this account.

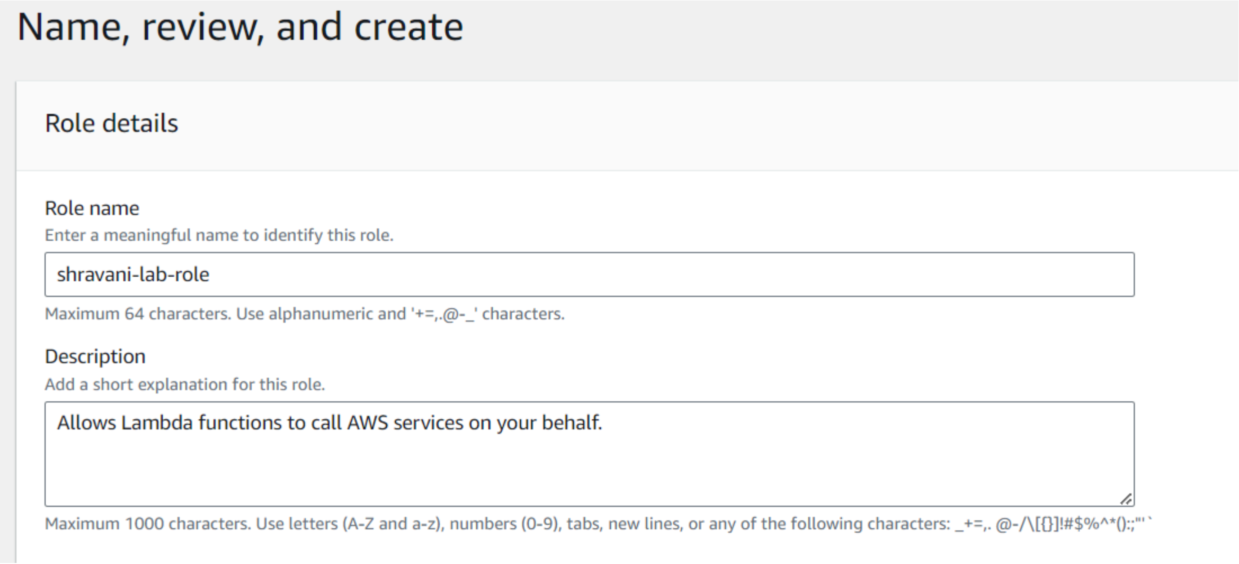
Service or use case

Lambda ▼

Choose a use case for the specified service.

Use case

● Lambda



Name, review, and create

Role details

Role name

Enter a meaningful name to identify this role.

shravani-lab-role

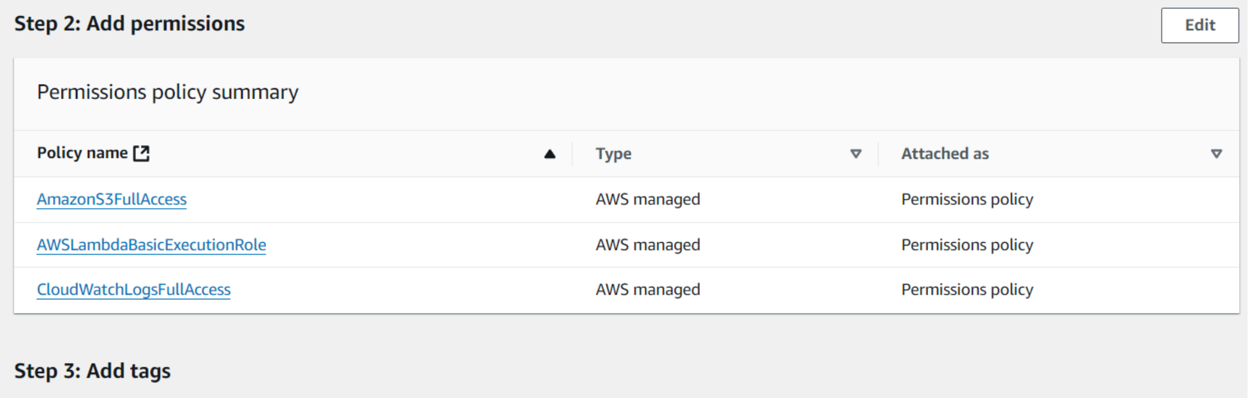
Maximum 64 characters. Use alphanumeric and '+=,.@-_' characters.

Description

Add a short explanation for this role.

Allows Lambda functions to call AWS services on your behalf.

Maximum 1000 characters. Use letters (A-Z and a-z), numbers (0-9), tabs, new lines, or any of the following characters: _+=,. @-/\[{}]!#$%^*():;"' `

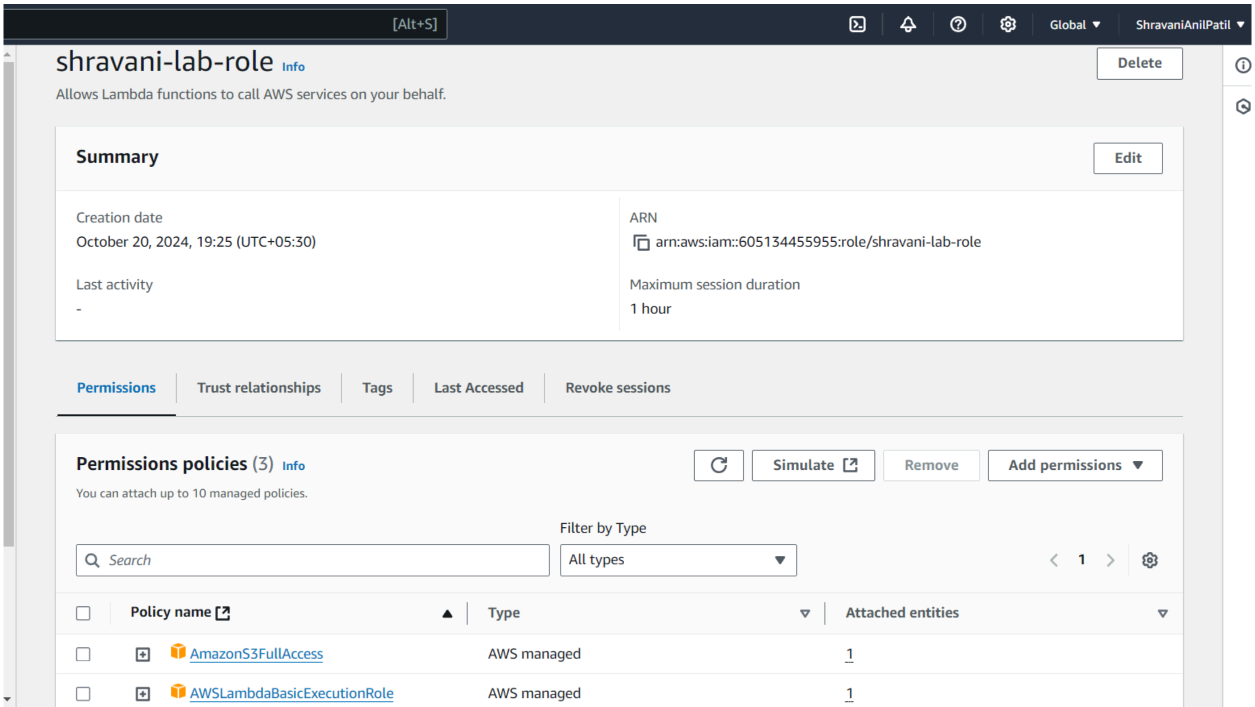


Step 2: Add permissions

Permissions policy summary

Policy name	Type	Attached as
AmazonS3FullAccess	AWS managed	Permissions policy
AWSLambdaBasicExecutionRole	AWS managed	Permissions policy
CloudWatchLogsFullAccess	AWS managed	Permissions policy

Step 3: Add tags



shravani-lab-role

Allows Lambda functions to call AWS services on your behalf.

Summary

Creation date
October 20, 2024, 19:25 (UTC+05:30)

ARN
arn:aws:iam::605134455955:role/shravani-lab-role

Last activity
-

Maximum session duration
1 hour

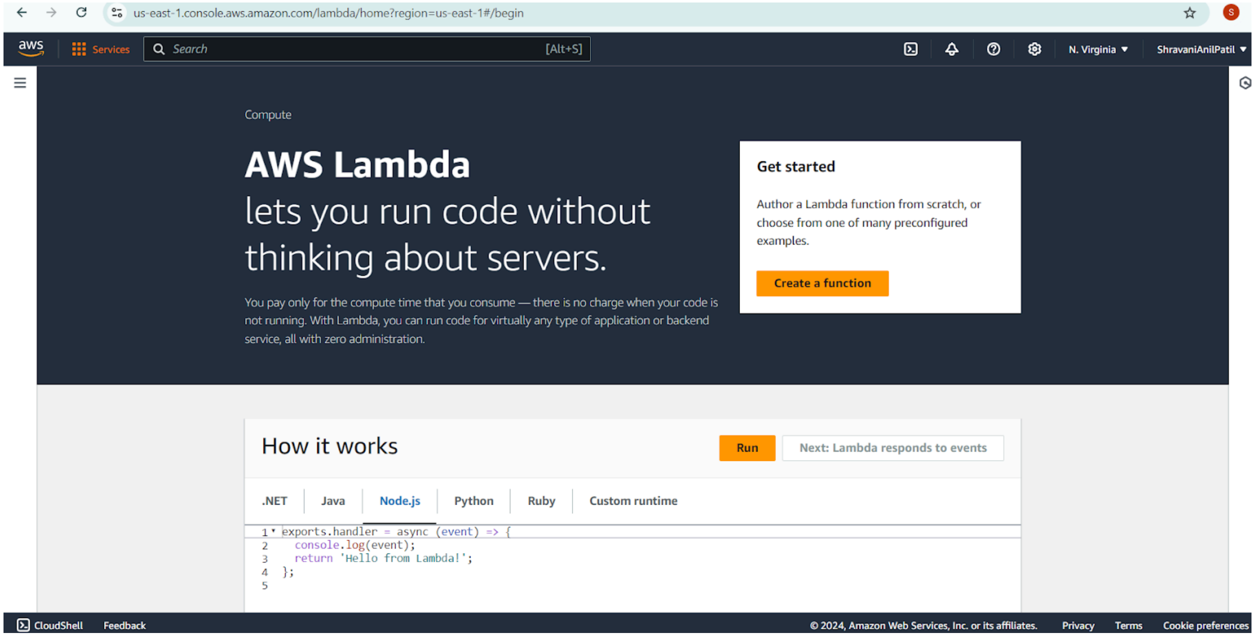
Permissions policies (3)

You can attach up to 10 managed policies.

Filter by Type: All types

□	Policy name	Type	Attached entities
□	AmazonS3FullAccess	AWS managed	1
□	AWSLambdaBasicExecutionRole	AWS managed	1

Step 2:Create AWS lambda function



us-east-1.console.aws.amazon.com/lambda/home?region=us-east-1#/begin
aws
Services
Search
[Alt+S]
N. Virginia ▾
ShravaniAnilPatil ▾

Compute

AWS Lambda
lets you run code without thinking about servers.

You pay only for the compute time that you consume — there is no charge when your code is not running. With Lambda, you can run code for virtually any type of application or backend service, all with zero administration.

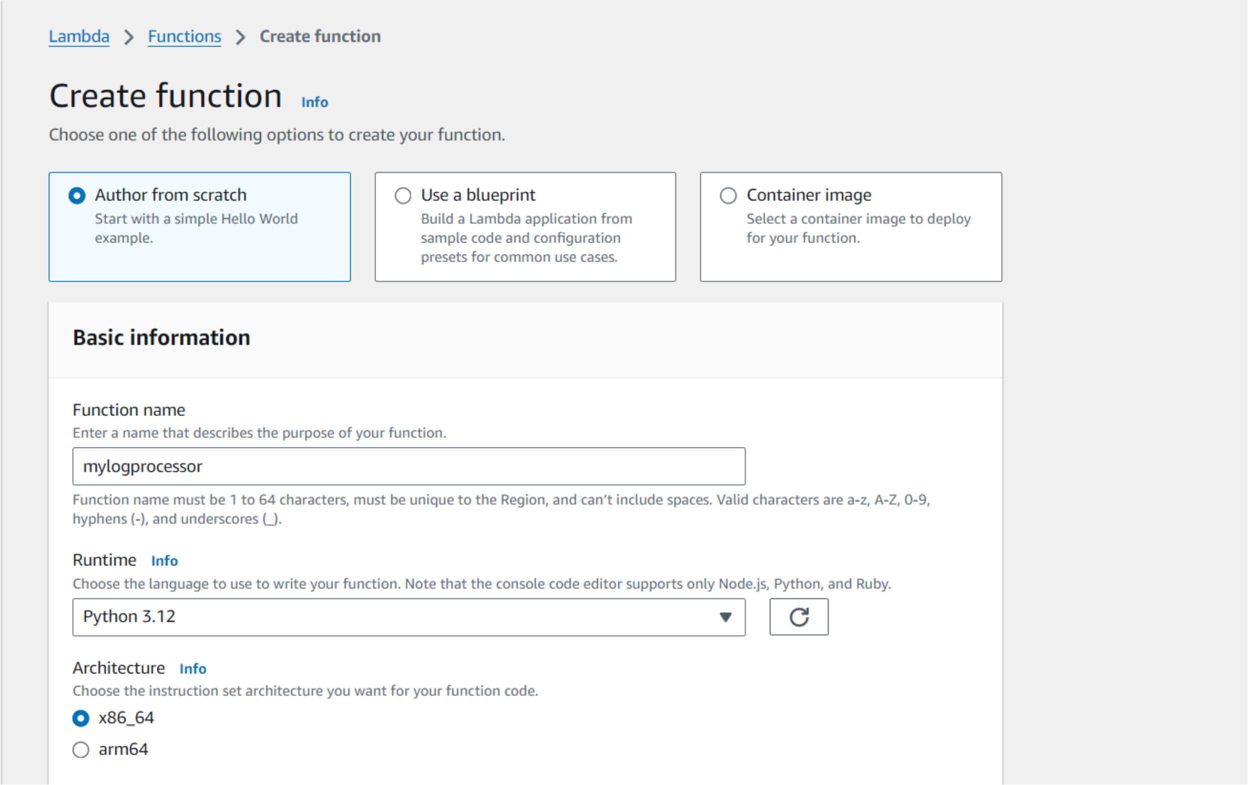
Get started

Author a Lambda function from scratch, or choose from one of many preconfigured examples.

How it works

```
exports.handler = async (event) => {
  console.log(event);
  return 'Hello from Lambda!';
};
```

Give your function a name and choose python as your runtime language



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.

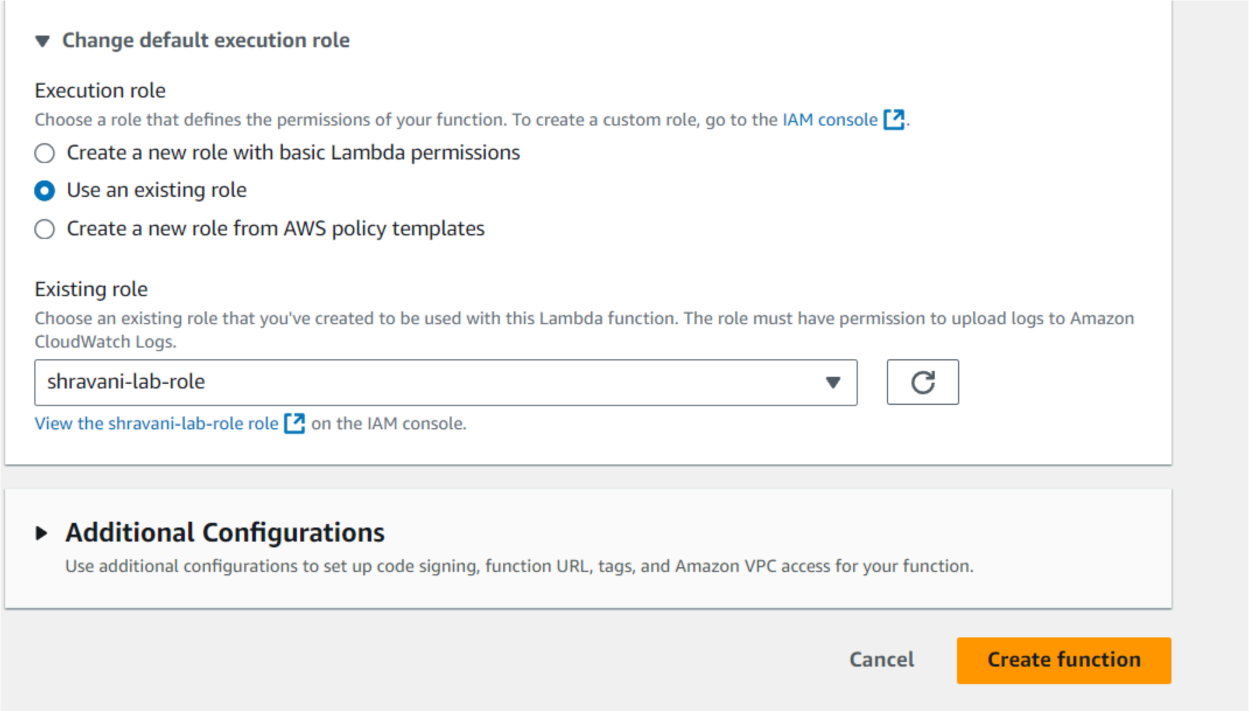
Basic information

Function name
Enter a name that describes the purpose of your function.
mylogprocessor
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.
Python 3.12 ▾

Architecture Info
Choose the instruction set architecture you want for your function code.
● x86_64
○ arm64

Add role created in Step 1



▼ Change default execution role

Execution role

Choose a role that defines the permissions of your function. To create a custom role, go to the IAM console.

○ Create a new role with basic Lambda permissions

● Use an existing role

○ Create a new role from AWS policy templates

Existing role

Choose an existing role that you've created to be used with this Lambda function. The role must have permission to upload logs to Amazon CloudWatch Logs.

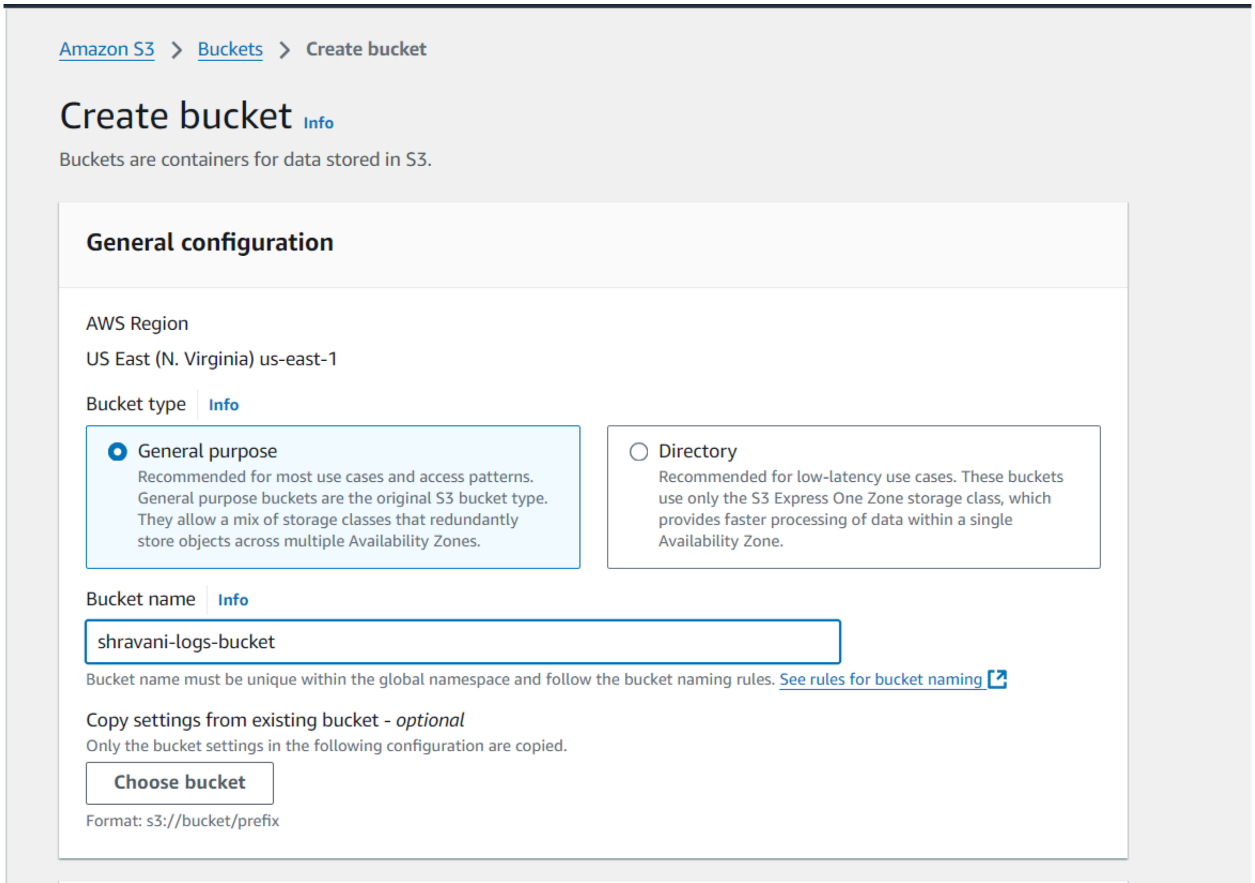
shravani-lab-role ▼

View the shravani-lab-role role on the IAM console.

▶ Additional Configurations

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

Step 3:Create S3 bucket



Amazon S3 > Buckets > Create bucket

Create bucket Info

Buckets are containers for data stored in S3.

General configuration

AWS Region

US East (N. Virginia) us-east-1

Bucket type Info

● General purpose
Recommended for most use cases and access patterns. General purpose buckets are the original S3 bucket type. They allow a mix of storage classes that redundantly store objects across multiple Availability Zones.

○ Directory
Recommended for low-latency use cases. These buckets use only the S3 Express One Zone storage class, which provides faster processing of data within a single Availability Zone.

Bucket name Info

shravani-logs-bucket

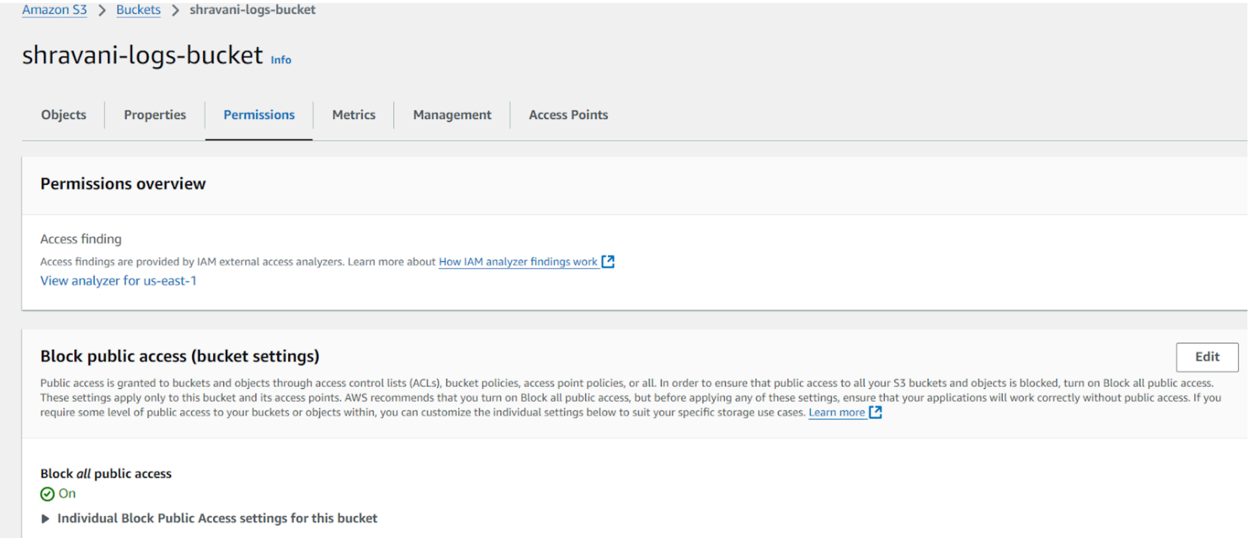
Bucket name must be unique within the global namespace and follow the bucket naming rules. See rules for bucket naming

Copy settings from existing bucket - *optional*

Only the bucket settings in the following configuration are copied.

Choose bucket

Format: s3://bucket/prefix



Amazon S3 > Buckets > shravani-logs-bucket

shravani-logs-bucket Info

Objects | Properties | Permissions | Metrics | Management | Access Points

Permissions overview

Access finding

Access findings are provided by IAM external access analyzers. Learn more about How IAM analyzer findings work

View analyzer for us-east-1

Block public access (bucket settings) Edit

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 all your S3 buckets and 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 your buckets or objects within, you can customize the individual settings below to suit your specific storage use cases. Learn more

Block *all* public access

✓ On

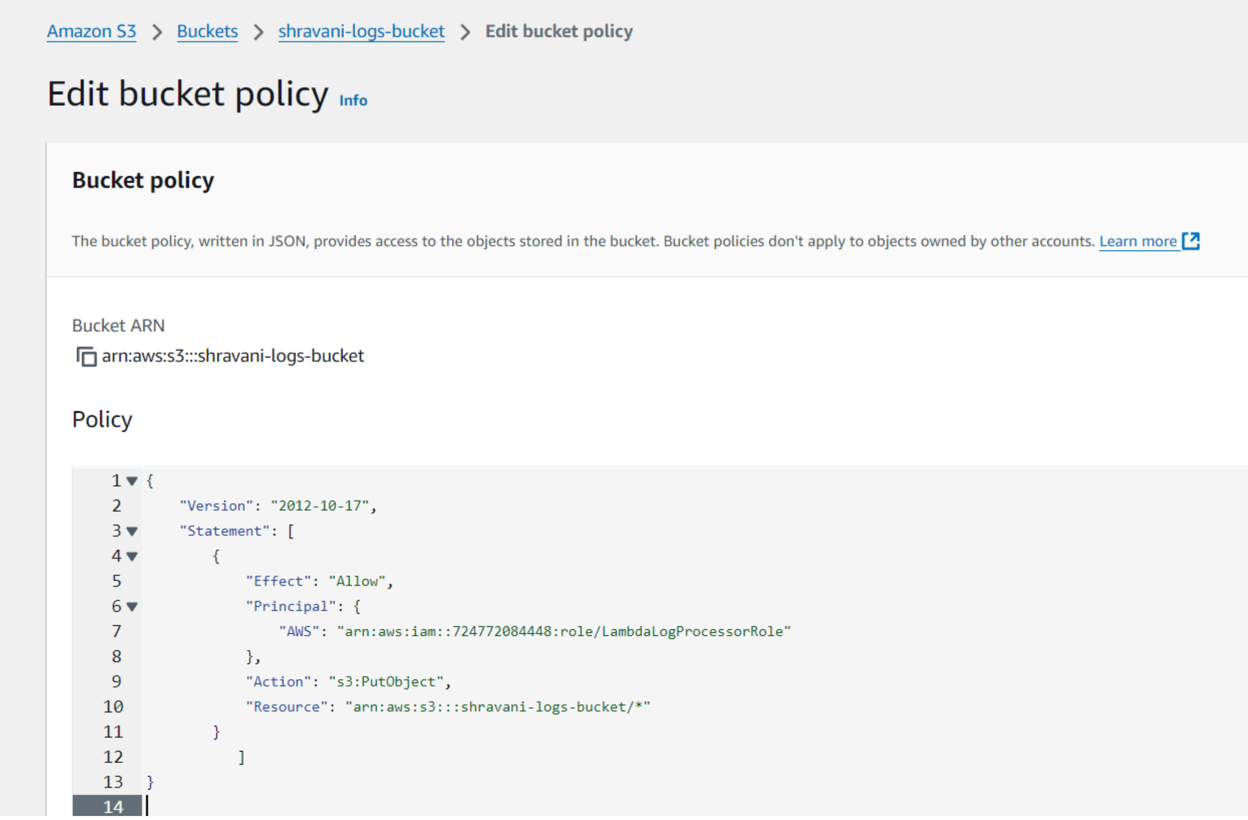
▶ Individual Block Public Access settings for this bucket

Edit the bucket policy

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::724772084448:role/LambdaLogProcessorRole"
      },
      "Action": "s3:PutObject",
      "Resource": "arn:aws:s3:::shravani-logs-bucket/*"
    }
  ]
}
```

Here, 724772084448 is my account id for AWS and LambdaLogProcessorRole is the IAM role created in step 1.

"Resource": "arn:aws:s3:::shravani-logs-bucket/ : this specifies the name of my S3 bucket



Amazon S3 > Buckets > shravani-logs-bucket > Edit bucket policy

Edit bucket policy Info

Bucket policy

The bucket policy, written in JSON, provides access to the objects stored in the bucket. Bucket policies don't apply to objects owned by other accounts. Learn more

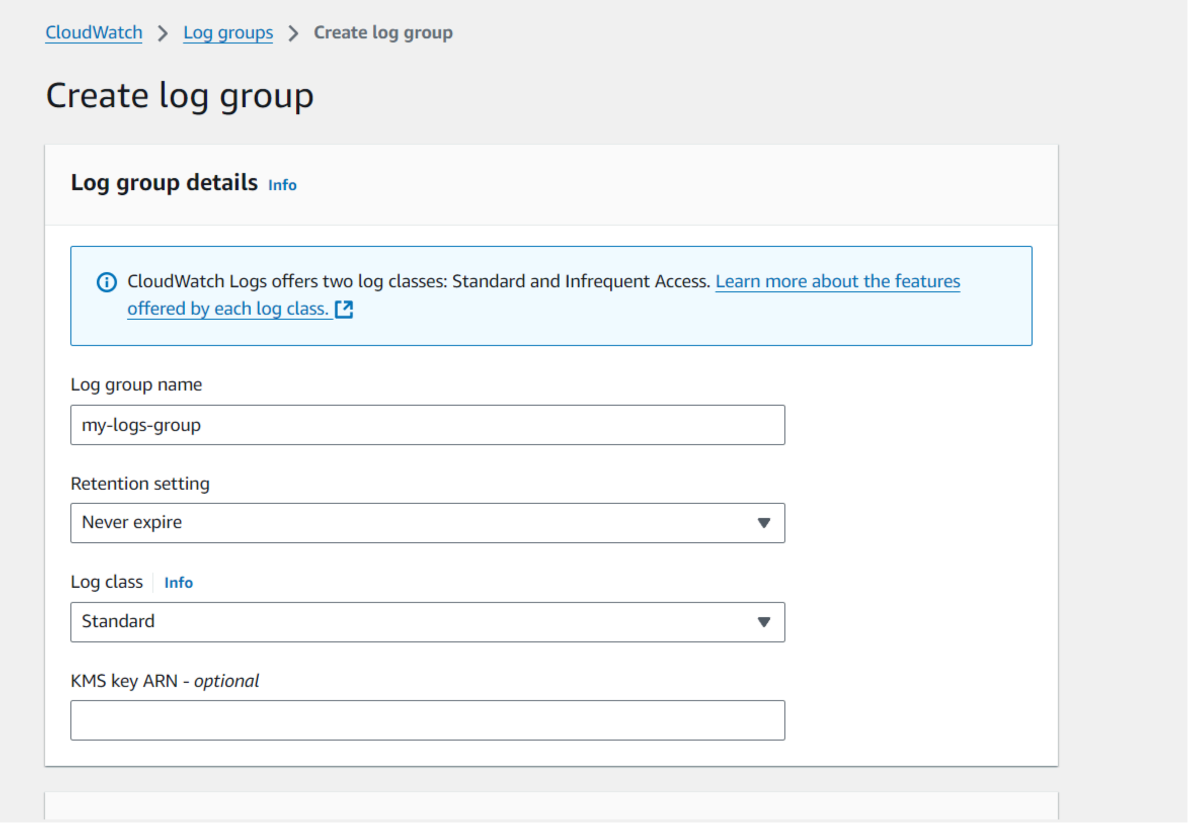
Bucket ARN

arn:aws:s3:::shravani-logs-bucket

Policy

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::724772084448:role/LambdaLogProcessorRole"
            },
            "Action": "s3:PutObject",
            "Resource": "arn:aws:s3:::shravani-logs-bucket/*"
        }
          ]
}
```

Step 4: Create CloudWatch groups



CloudWatch > Log groups > Create log group

Create log group

Log group details Info

CloudWatch Logs offers two log classes: Standard and Infrequent Access. Learn more about the features offered by each log class.

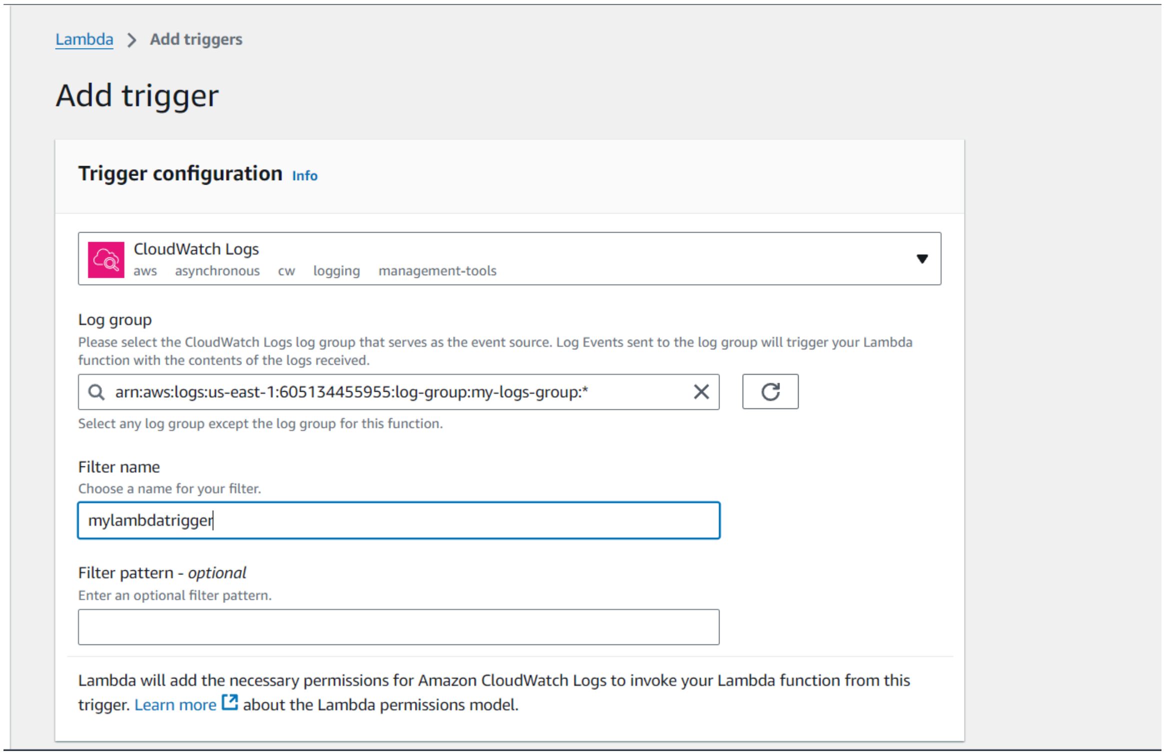
Log group name: my-logs-group

Retention setting: Never expire

Log class Info: Standard

KMS key ARN - *optional*:

Step 5:Add the cloudwatch group trigger in Lambda function



Lambda > Add triggers

Add trigger

Trigger configuration Info

CloudWatch Logs
aws asynchronous cw logging management-tools

Log group
Please select the CloudWatch Logs log group that serves as the event source. Log Events sent to the log group will trigger your Lambda function with the contents of the logs received.

arn:aws:logs:us-east-1:605134455955:log-group:my-logs-group:*

Select any log group except the log group for this function.

Filter name
Choose a name for your filter.

mylambdatrigger

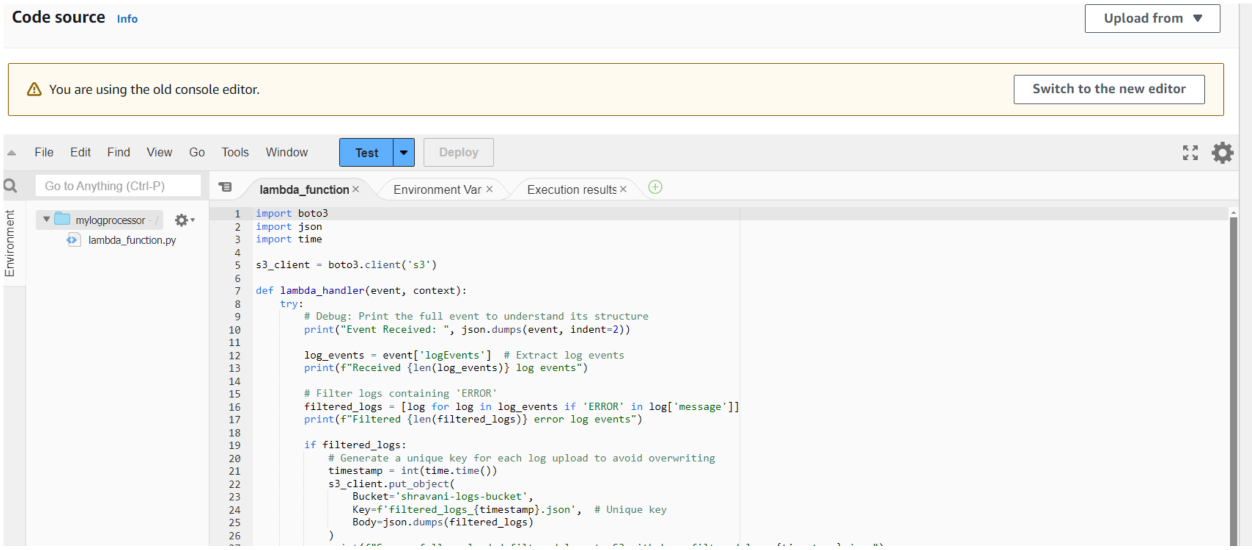
Filter pattern - *optional*
Enter an optional filter pattern.

Lambda will add the necessary permissions for Amazon CloudWatch Logs to invoke your Lambda function from this trigger. Learn more about the Lambda permissions model.

Step 6: In the lambda function add the code:

```
import boto3
import json
import time

s3_client = boto3.client('s3')

def lambda_handler(event, context):
    try:
        # Debug: Print the full event to understand its structure
        print("Event Received: ", json.dumps(event, indent=2))

        log_events = event['logEvents']  # Extract log events
        print(f"Received {len(log_events)} log events")

        # Filter logs containing 'ERROR'
        filtered_logs = [log for log in log_events if 'ERROR' in log['message']]
        print(f"Filtered {len(filtered_logs)} error log events")

        if filtered_logs:
            # Generate a unique key for each log upload to avoid overwriting
            timestamp = int(time.time())
            s3_client.put_object(
                Bucket='shravani-logs-bucket',
                Key=f'filtered_logs_{timestamp}.json',  # Unique key
                Body=json.dumps(filtered_logs)
            )
            print(f"Successfully uploaded filtered logs to S3 with key:
filtered_logs_{timestamp}.json")

        return {
            'statusCode': 200,
            'body': json.dumps('Logs processed successfully!')
        }
    except KeyError as e:
        print(f"KeyError: {e}")
        raise e
    except Exception as e:
        print(f"Exception: {e}")
        raise e
```

Note: In the above code, add the name of S3 bucket created earlier.

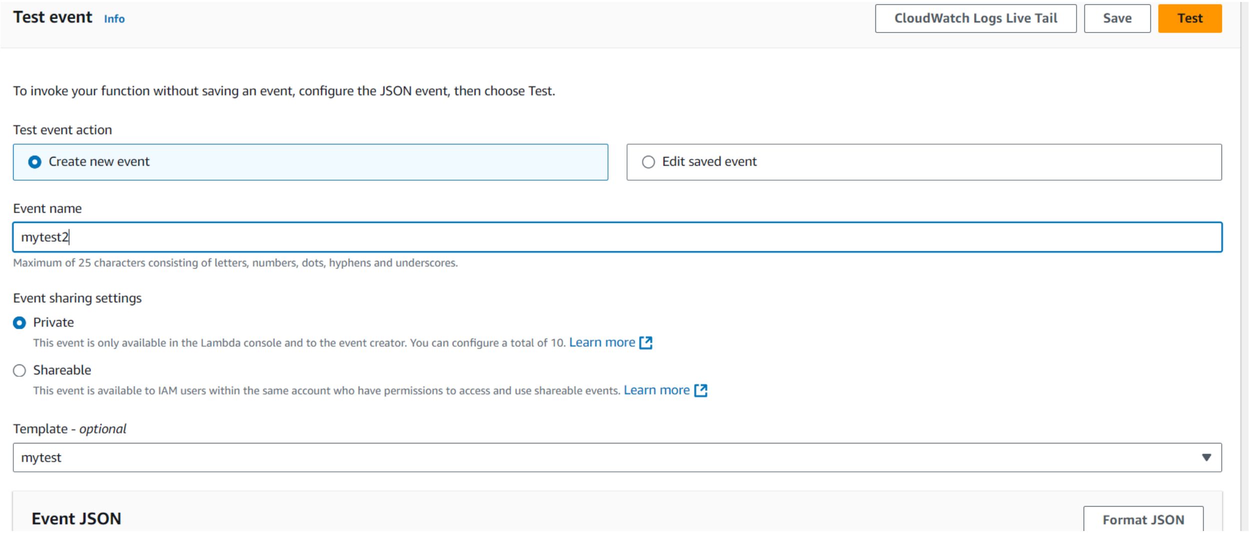


⚠ You are using the old console editor.

```
import boto3
import json
import time

s3_client = boto3.client('s3')

def lambda_handler(event, context):
    try:
        # Debug: Print the full event to understand its structure
        print("Event Received: ", json.dumps(event, indent=2))

        log_events = event['logEvents']  # Extract log events
        print(f"Received {len(log_events)} log events")

        # Filter logs containing 'ERROR'
        filtered_logs = [log for log in log_events if 'ERROR' in log['message']]
        print(f"Filtered {len(filtered_logs)} error log events")

        if filtered_logs:
            # Generate a unique key for each log upload to avoid overwriting
            timestamp = int(time.time())
            s3_client.put_object(
                Bucket='shravani-logs-bucket',
                Key=f'filtered_logs_{timestamp}.json',  # Unique key
                Body=json.dumps(filtered_logs)
            )
```

Deploy to save the code

Step 7: Create a new test event to test the setup.



To invoke your function without saving an event, configure the JSON event, then choose Test.

Test event action

● Create new event ○ Edit saved event

Event name

mytest2

Maximum of 25 characters consisting of letters, numbers, dots, hyphens and underscores.

Event sharing settings

● Private

This event is only available in the Lambda console and to the event creator. You can configure a total of 10. Learn more

○ Shareable

This event is available to IAM users within the same account who have permissions to access and use shareable events. Learn more

Template - *optional*

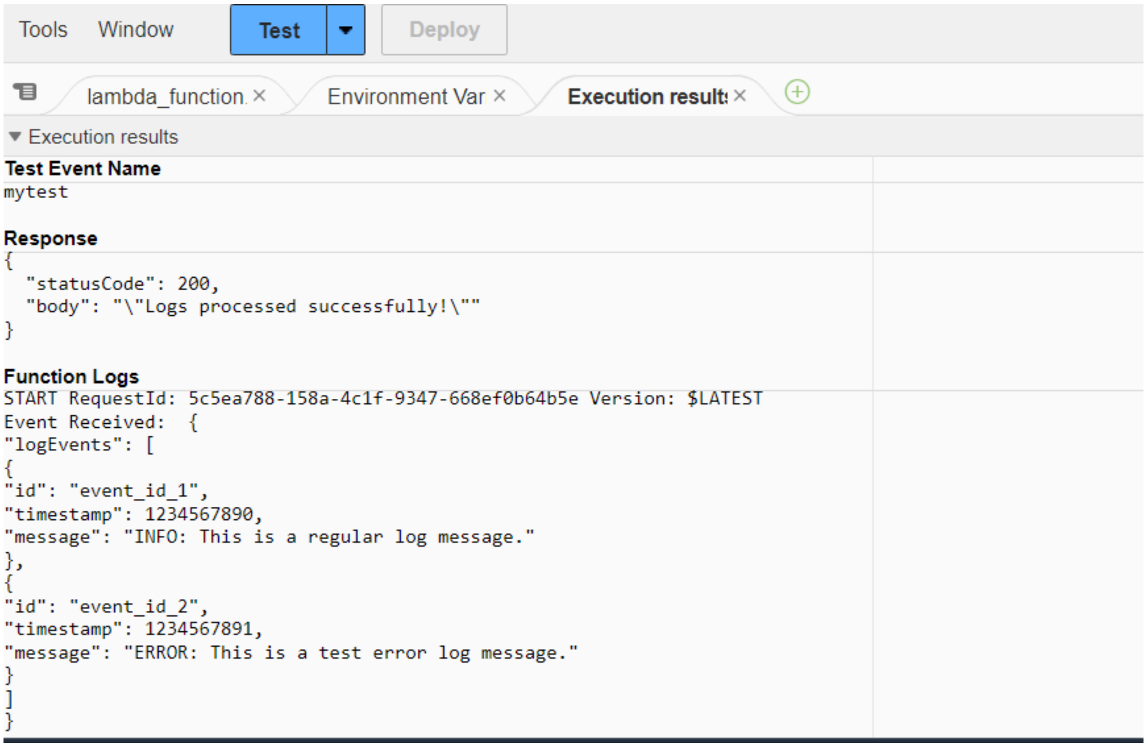
mytest ▼

Event JSON

Add this to the json part

```
{
 "logEvents": [
   {
     "id": "event_id_1",
     "timestamp": 1234567890,
     "message": "INFO: This is a regular log message."
   },
   {
     "id": "event_id_2",
     "timestamp": 1234567891,
     "message": "ERROR: This is a test error log message."
   }
 ]
}
```

Step 8: Test the code



Test Event Name

```
mytest
```

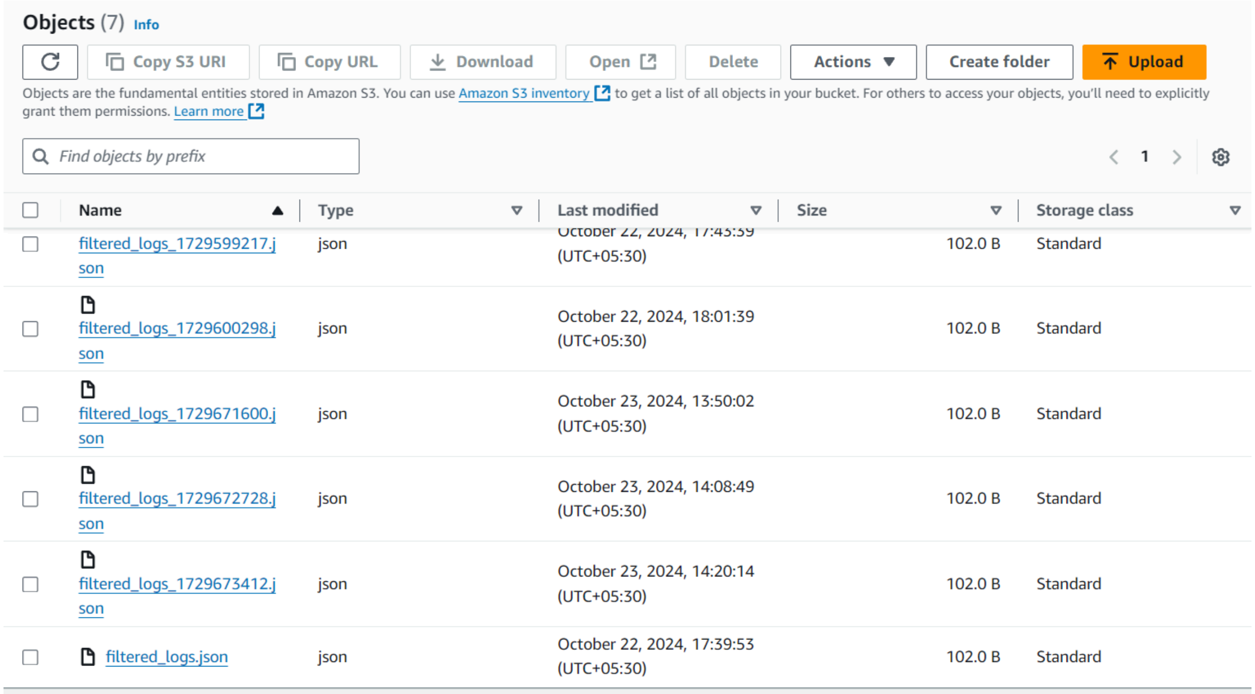
Response

```
{
  "statusCode": 200,
  "body": "\"Logs processed successfully!\""
}
```

Function Logs

```
START RequestId: 5c5ea788-158a-4c1f-9347-668ef0b64b5e Version: $LATEST
Event Received:  {
"logEvents": [
{
"id": "event_id_1",
"timestamp": 1234567890,
"message": "INFO: This is a regular log message."
},
{
"id": "event_id_2",
"timestamp": 1234567891,
"message": "ERROR: This is a test error log message."
}
]
}
```

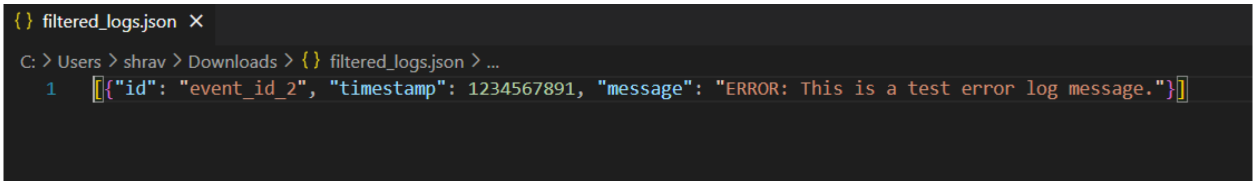
Step 9: To verify, go back to your S3 bucket. A new item called filtered_logs.json is added in the S3 bucket



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

□	Name ▲	Type	Last modified	Size	Storage class
□	filtered_logs_1729599217.json	json	October 22, 2024, 17:43:59 (UTC+05:30)	102.0 B	Standard
□	filtered_logs_1729600298.json	json	October 22, 2024, 18:01:39 (UTC+05:30)	102.0 B	Standard
□	filtered_logs_1729671600.json	json	October 23, 2024, 13:50:02 (UTC+05:30)	102.0 B	Standard
□	filtered_logs_1729672728.json	json	October 23, 2024, 14:08:49 (UTC+05:30)	102.0 B	Standard
□	filtered_logs_1729673412.json	json	October 23, 2024, 14:20:14 (UTC+05:30)	102.0 B	Standard
□	filtered_logs.json	json	October 22, 2024, 17:39:53 (UTC+05:30)	102.0 B	Standard

On opening the json file, we can see the output.


{ } filtered_logs.json ×
C: > Users > shrav > Downloads > { } filtered_logs.json > ...

```
[{"id": "event_id_2", "timestamp": 1234567891, "message": "ERROR: This is a test error log message."}]
```

Guidelines:

1. **Use a personal AWS account** if your AWS Academy account lacks sufficient privileges for the default role. This ensures you have the necessary access to all services without limitations.
2. **Principle of Least Privilege**: Assign only the minimum permissions required for IAM roles and policies. This reduces the potential security risks by limiting access to only essential AWS resources.
3. **Logging and Monitoring**: Enable comprehensive logging for your Lambda functions and monitor their performance using CloudWatch. This approach enhances your ability to troubleshoot issues quickly and ensures the system runs smoothly.

Conclusion:

In this case study, the integration of **AWS Lambda**, **CloudWatch Logs**, and **S3** for real-time log processing was successfully demonstrated. To ensure secure access to the necessary AWS services, an **IAM user** with fine-grained permissions was created, following the principle of least privilege. A **Lambda function** was configured to automatically trigger when new log entries were added to a **CloudWatch Log Group**, filtering specific log events based on predefined keywords, such as ERROR. The filtered logs were then stored in an **S3 bucket** for efficient analysis and long-term storage.

This setup significantly enhanced the system's log monitoring, alerting, and troubleshooting capabilities by automating the log management process. Following best practices, such as minimizing resource access and implementing robust error handling and retries within the Lambda function, ensured that the solution was secure, reliable, and scalable. This architecture effectively streamlines operational workflows, enabling real-time insights and improving overall system observability.