Guided Lab: Processing Amazon CloudWatch Logs with AWS Lambda

Description

Amazon CloudWatch Logs is a powerful service that enables real-time monitoring and logging of your applications and infrastructure in AWS. By capturing log data from various AWS resources, such as EC2 instances, Lambda functions, and API Gateway, CloudWatch Logs provides insights into system performance and operational issues. However, simply collecting logs is not enough; processing these logs to extract meaningful information is crucial. This lab will guide you through creating a Lambda function that processes logs from CloudWatch Logs, focusing on filtering and handling specific log entries, such as HTTP 500 error requests.

The main goal of this lab is to demonstrate how to automate log processing using AWS Lambda. Processing CloudWatch Logs can help identify and act on critical issues in your application, such as repeated 500 error responses, which indicate server-side problems that need immediate attention.

In this lab, you'll create sample access logs, simulate their entry into CloudWatch Logs, and then process them using a Lambda function. Specifically, you will filter the logs to extract only those with an HTTP 500 status code, which indicates internal server errors.

Prerequisites

This lab assumes you have a basic understanding of AWS Lambda, CloudWatch Logs, and Python programming.

If you find any gaps in your knowledge, consider taking the following lab:

- Creating an AWS Lambda function

Objectives

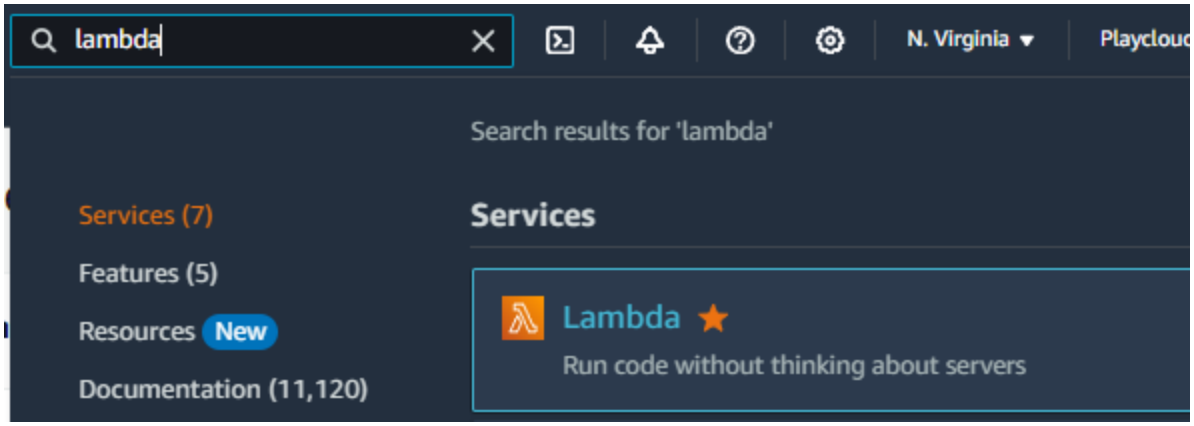
In this lab, you will:

- Learn how to create a Lambda function that generates simulated access logs and sends them to CloudWatch Logs.
- Understand how to process logs in CloudWatch Logs using a Lambda function.
- Filter and identify HTTP 500 error logs for further analysis.
- Set up a CloudWatch Logs subscription filter to trigger the log processing Lambda function automatically.
- Test and validate the end-to-end log generation and processing workflow.

Lab Steps

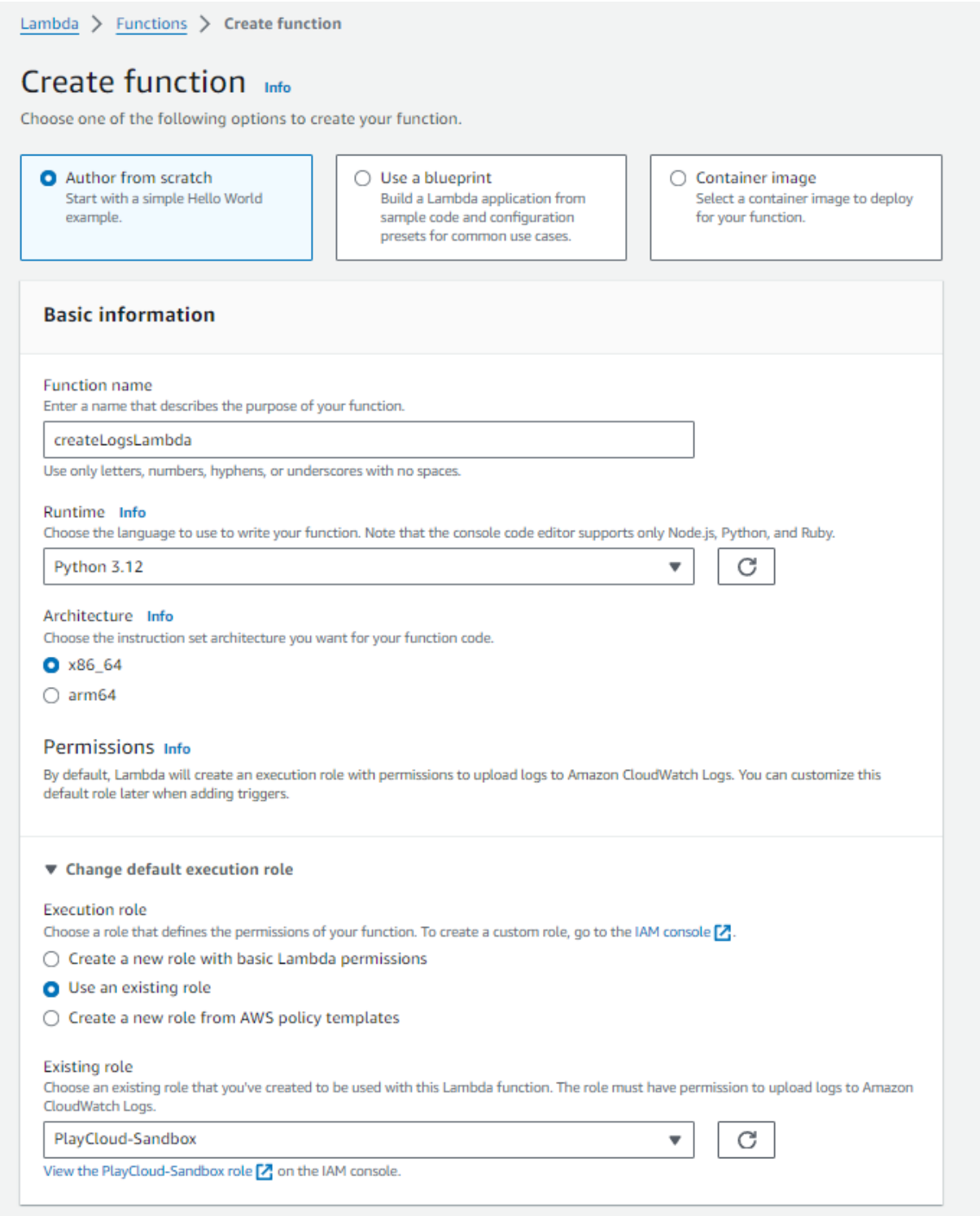
Creating Sample CloudWatch Logs Data

1. Navigate to the AWS Lambda Console



2. Create a new Lambda function using the following configurations:

- Choose **Author from scratch.**
- Function name: createLogsLambda
- Select Python 3.12 as the runtime
- .**Execution role:**
    - Select Use an Existing Role: PlayCloud-Sanbox



- Click **Create function**

3. Paste the following code to the code editor:

***Note**: We are using the **old console editor** for this lab. You can switch to the **new or old editor** as you desire; the process remains the same, but the interface may look slightly different.*

```
import boto3

import time
```

```
import random

logs_client = boto3.client('logs')

def lambda_handler(event, context):
    log_group_name = 'TestLogGroup'
    log_stream_name = 'TestLogStream'

    # Create log group if it doesn't exist
    try:
        logs_client.create_log_group(logGroupName=log_group_name)
    except logs_client.exceptions.ResourceAlreadyExistsException:
        pass  # Log group already exists

    # Create log stream if it doesn't exist
    try:
        logs_client.create_log_stream(logGroupName=log_group_name,
logStreamName=log_stream_name)
    except logs_client.exceptions.ResourceAlreadyExistsException:
        pass  # Log stream already exists

    # Sample data for log entries
    http_methods = ['GET', 'POST', 'PUT', 'DELETE']
    request_urls = [
        '/home',
        '/api/user',
        '/login',
        '/products',
        '/checkout',
        '/cart',
        '/search?q=aws',
```

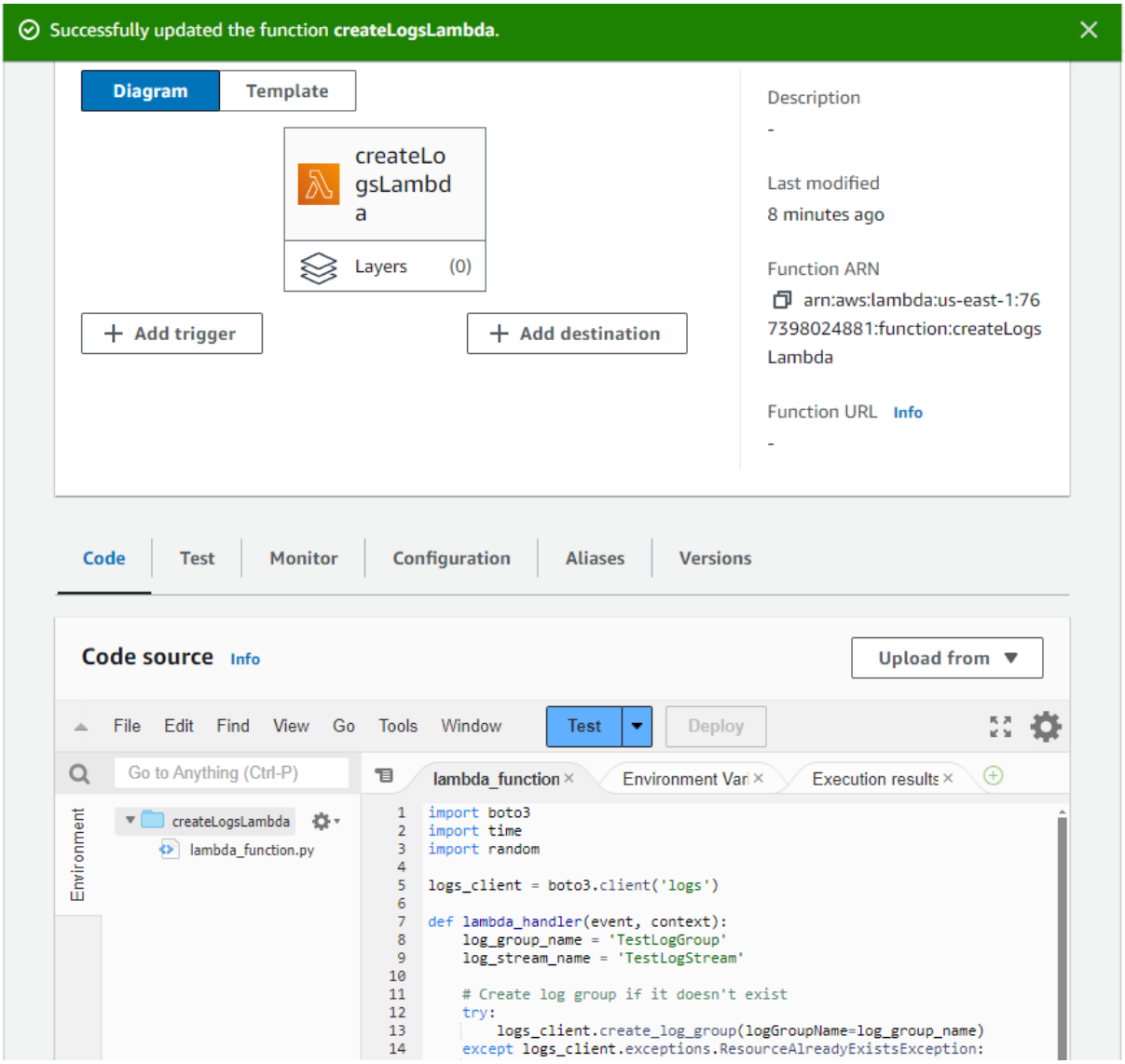
```
        '/api/order/123',
        '/api/product/567'
    ]
    user_agents = [
        'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.124 Safari/537.36',
        'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/14.0.3 Safari/605.1.15',
        'Mozilla/5.0 (iPhone; CPU iPhone OS 14_0 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/14.0 Mobile/15A5341f Safari/604.1',
        'Mozilla/5.0 (Linux; Android 10; SM-G973F) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0.4183.81 Mobile Safari/537.36'
    ]
    status_codes = [200, 201, 400, 401, 403, 404, 500]

    # Create log events
    log_events = []
    for i in range(20):
        log_event = {
            'timestamp': int(time.time() * 1000),
            'message': (
                f"{random.choice(http_methods)} "
                f"{random.choice(request_urls)} "
                f"{random.choice(status_codes)} "
                f"{random.choice(user_agents)}"
            )
        }
        log_events.append(log_event)

    # Put log events
    logs_client.put_log_events(
        logGroupName=log_group_name,
```

```
    logStreamName=log_stream_name,
    logEvents=log_events
)

return {
    'statusCode': 200,
    'body': 'Successfully created log events.'
}
```

This Lambda function creates a simulated log group and log stream in CloudWatch Logs. It then generates HTTP request logs with various HTTP methods, request paths, status codes, and user agents. The function first checks if the log group and log stream exist; if not, it creates them. After ensuring the log group and stream are set up, the function randomly generates log events and sends them to the log stream within the log group.

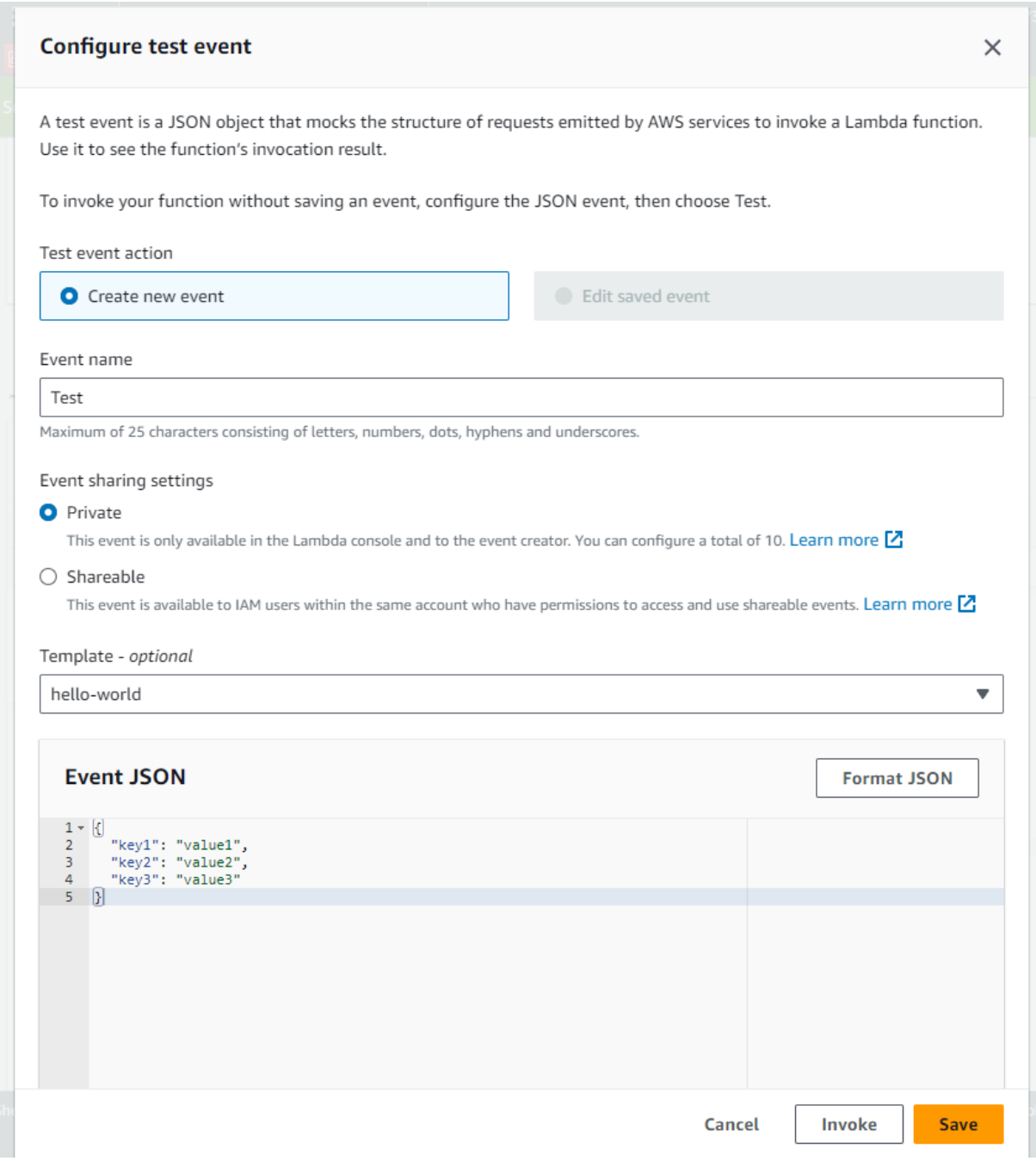
---

4. Click on **Deploy** to save changes.

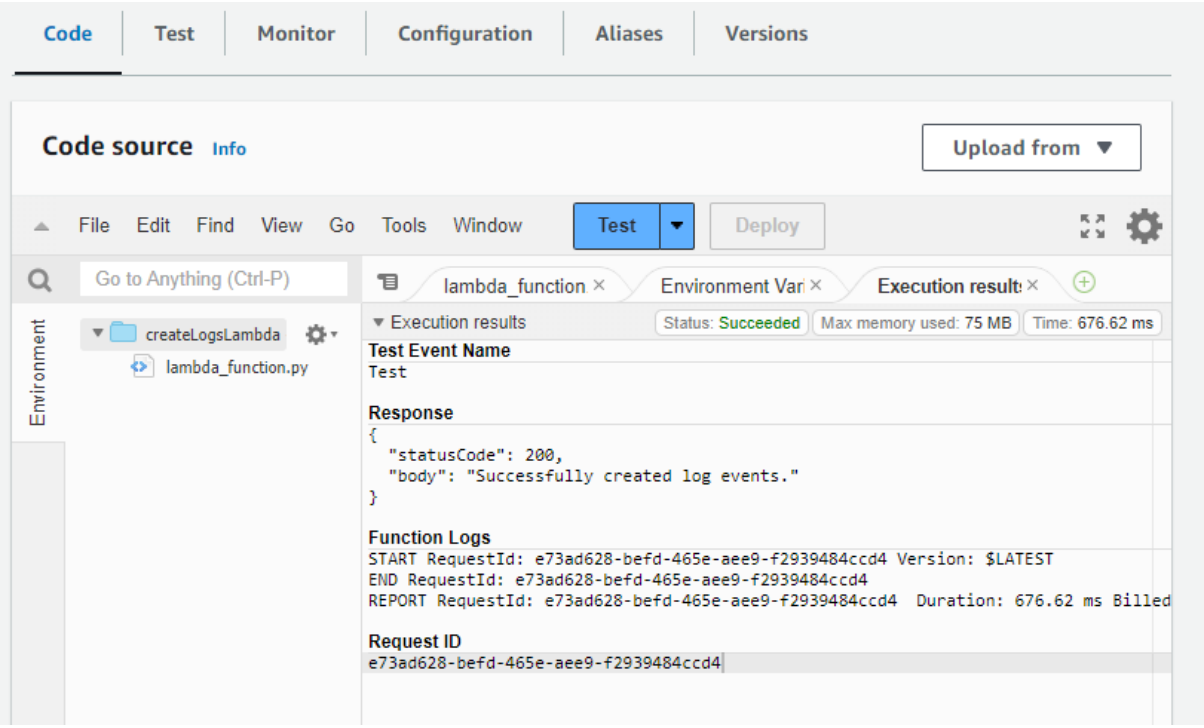


5. Now, click on the Test button.

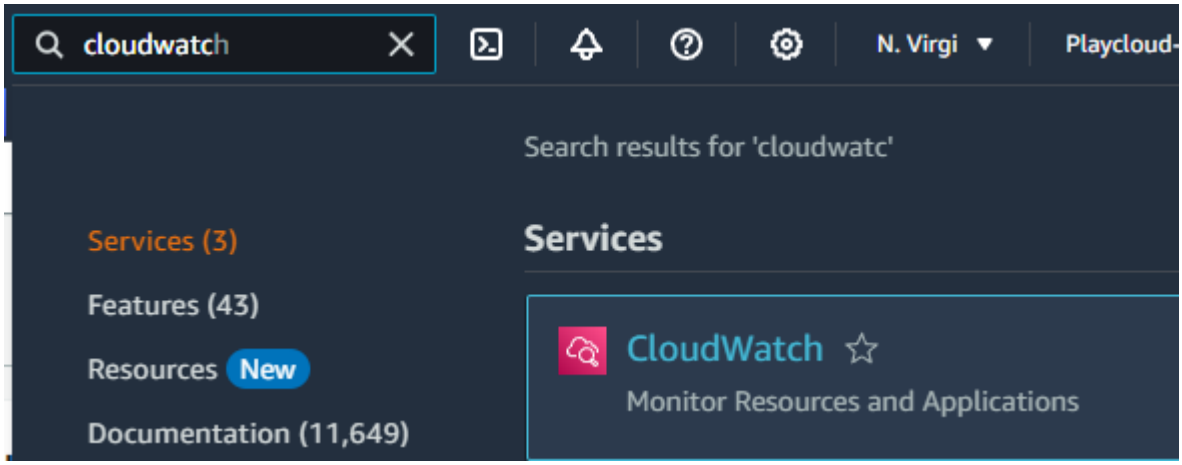
- A Configure test event dialog will appear, add Test for the **Event name,** and **Save** it.



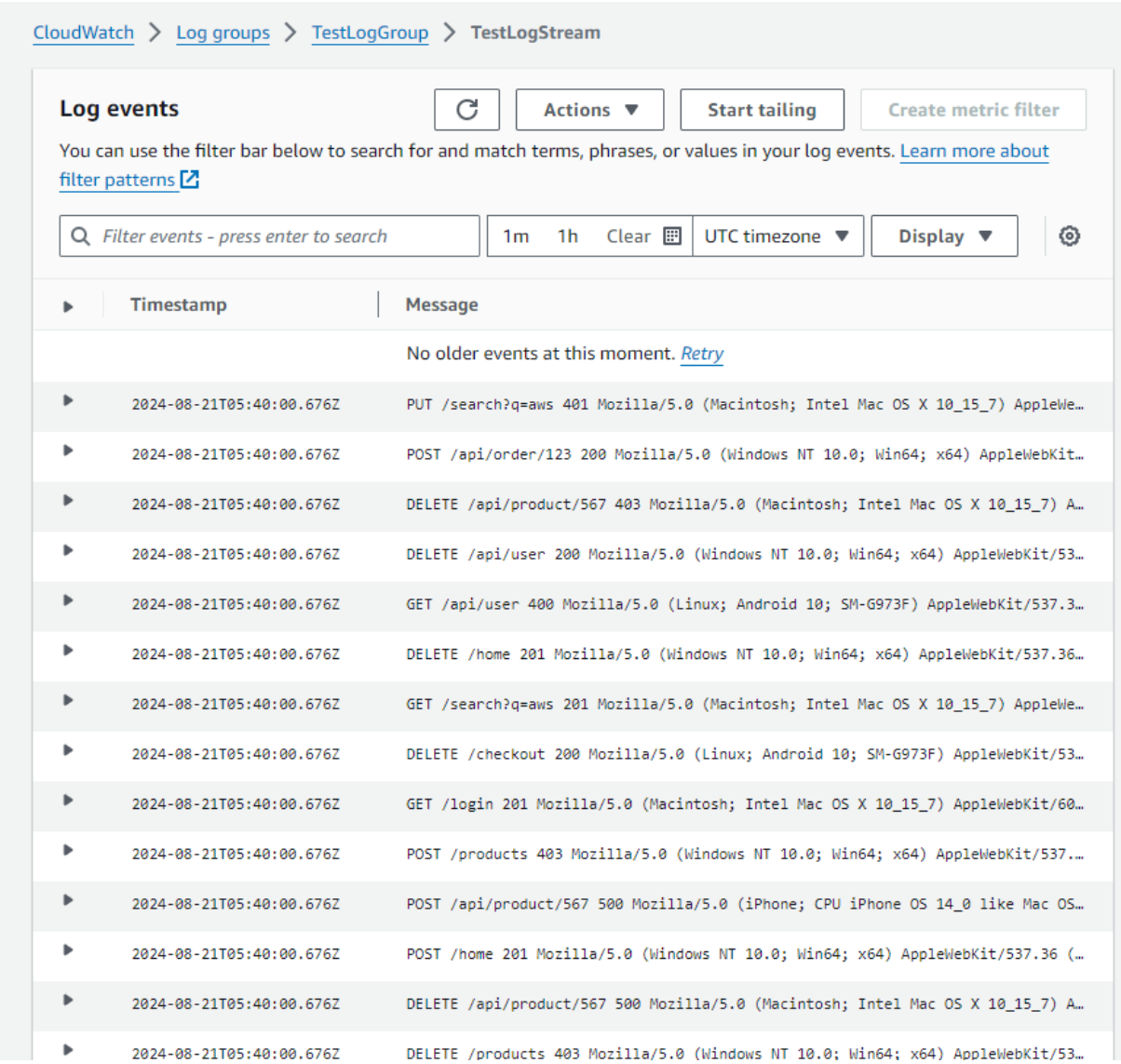
- Click on **Test** again.



6. After running the Lambda function, search for "**CloudWatch**" in the search bar, right-click, and select "**Open link in new tab**." This way, you can verify the logs while keeping the Lambda console open for the next steps.

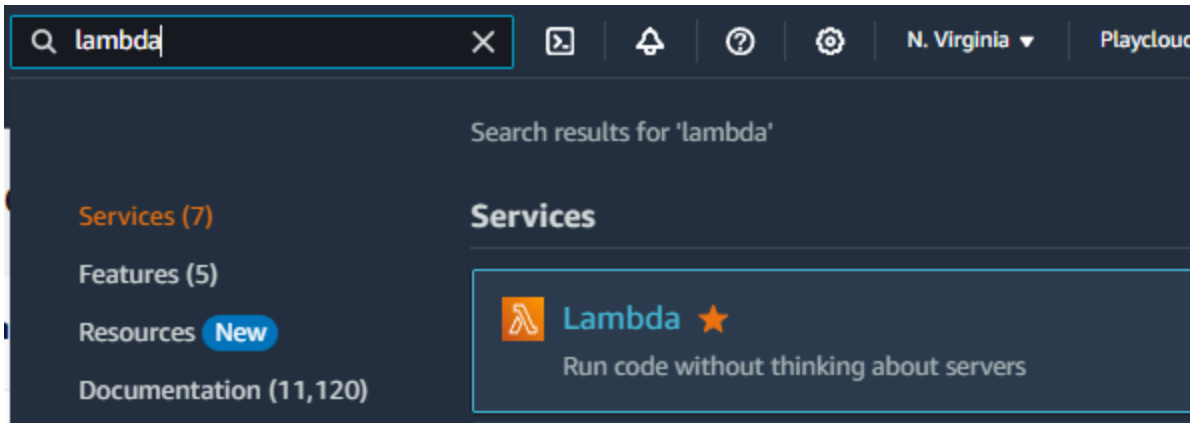


7. Navigate to the Log Groups section, and locate the TestLogGroup. Open TestLogGroup, and inside it, you should find TestLogStream. Open TestLogStream to view the generated log entries.



Creating the Log Processing Lambda Function

1. Navigate to Lambda Function Console again.



2. Create a new Lambda function using the provided configuration and code below:

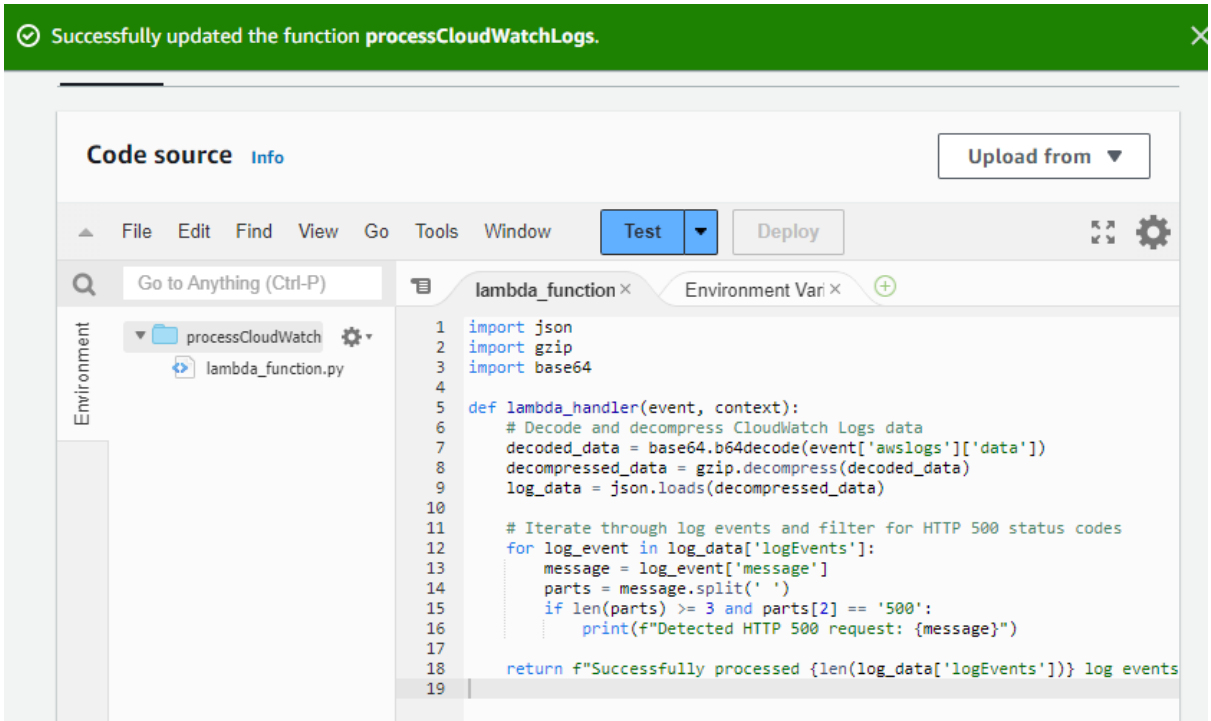
- Choose **Author from scratch.**
- Function name: **processCloudWatchLogs**
- Select Python 3.12 as the runtime.
- **Execution role:** Select Use an Existing Role: **PlayCloud-Sandbox**
- Click **Create function**
- Paste the following Code:

```
import json
import gzip
import base64

def lambda_handler(event, context):
    # Decode and decompress CloudWatch Logs data
    decoded_data = base64.b64decode(event['awslogs']['data'])
    decompressed_data = gzip.decompress(decoded_data)
    log_data = json.loads(decompressed_data)

    # Iterate through log events and filter for HTTP 500 status codes
    for log_event in log_data['logEvents']:
        message = log_event['message']
        parts = message.split(' ')
        if len(parts) >= 3 and parts[2] == '500':
            print(f"Detected HTTP 500 request: {message}")

    return f"Successfully processed {len(log_data['logEvents'])} log events."
```

This function processes the logs from CloudWatch Logs by decoding and decompressing the log data, then iterating through each log entry to check for HTTP 500 status codes. The function logs it to the console if a 500 status code is found.

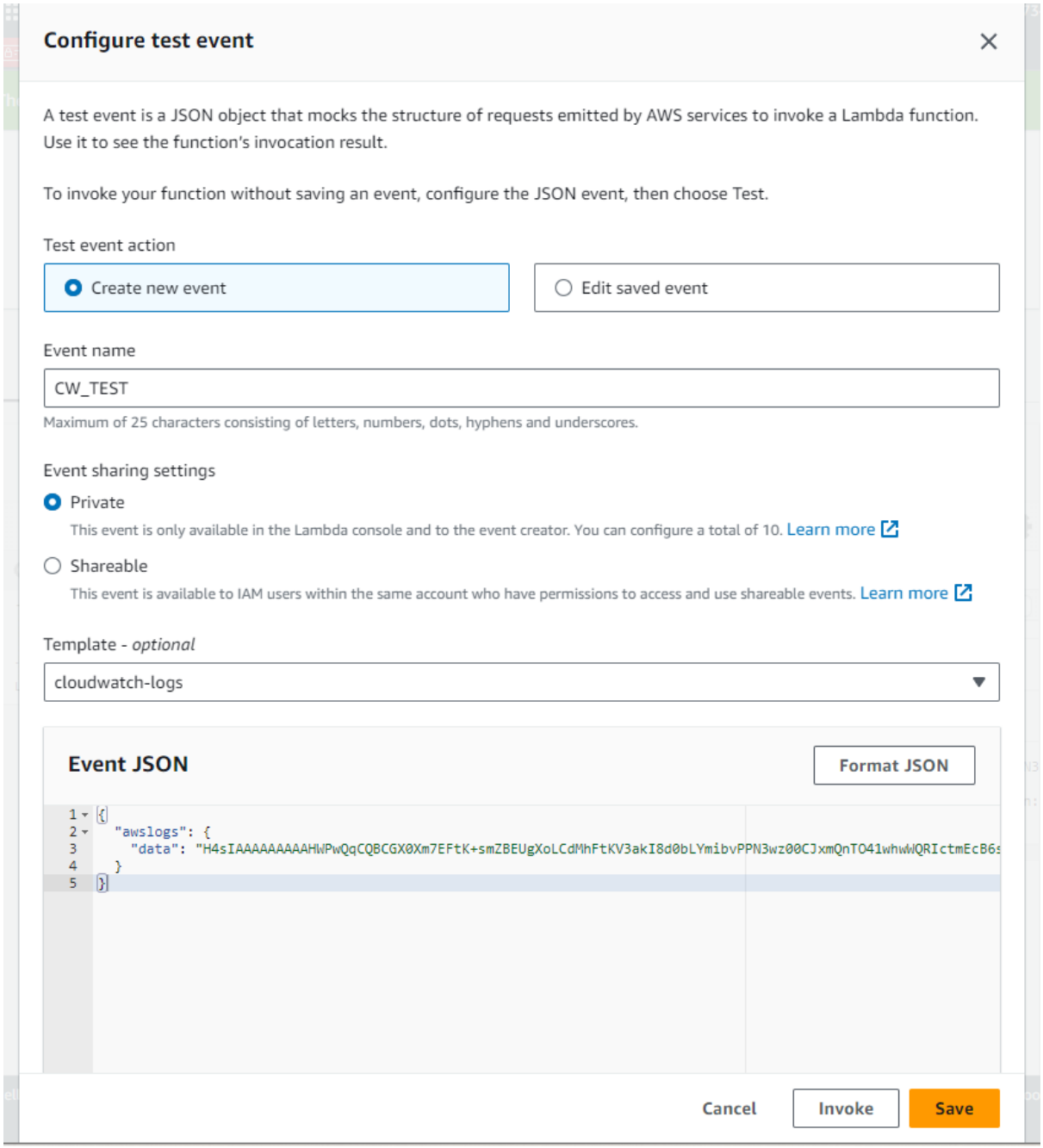
---

3. Click on **Deploy**

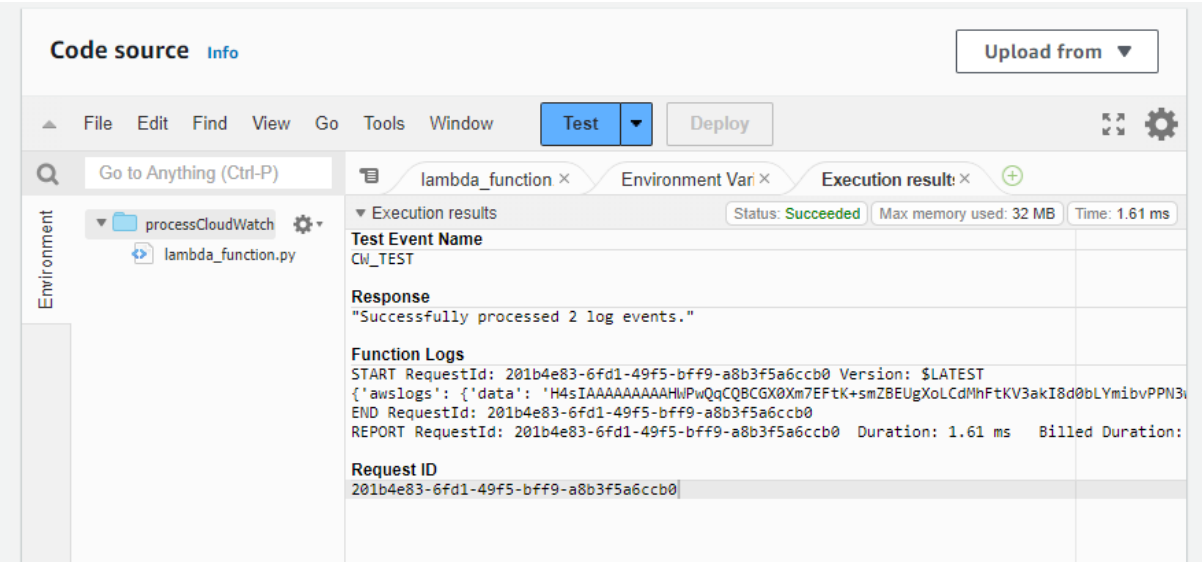


4. Now, click on the Test button.

- Similar to the previous creation of lambda, a Configure test event dialog will appear. Add CW_TEST for the **Event name**, select the cloudwatch logs in the template, and **Save** it.



- Click on **Test** again.



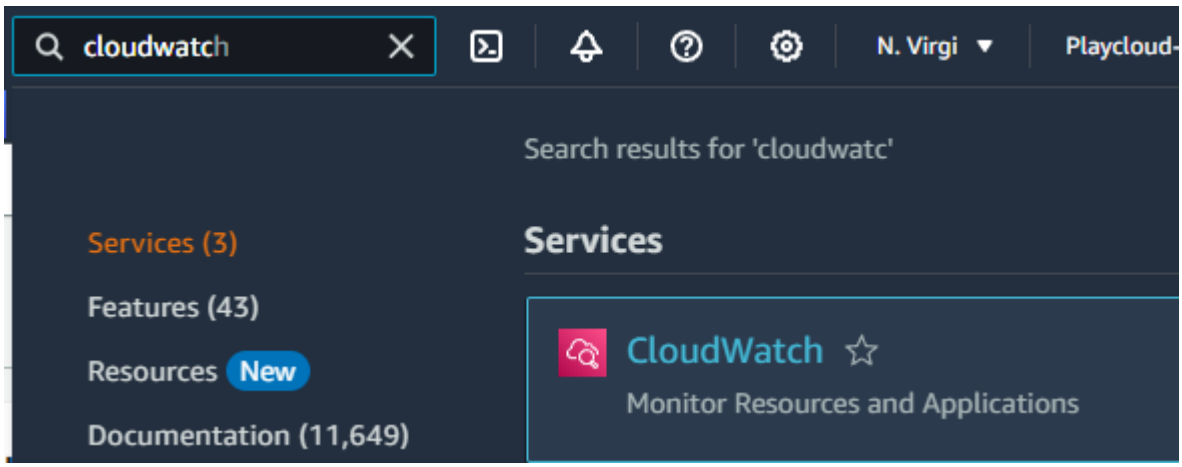
This shows the Lambda function's code is working just fine. It will also create a CloudWatch log group for this Lambda function.

Creating the CloudWatch Logs Subscription Filter

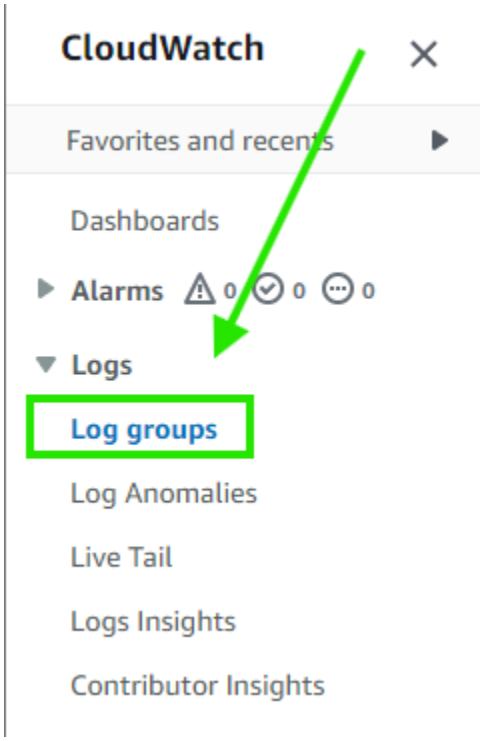
To automatically trigger the log processing Lambda function when new logs are added, you must create a CloudWatch Logs subscription filter.

---

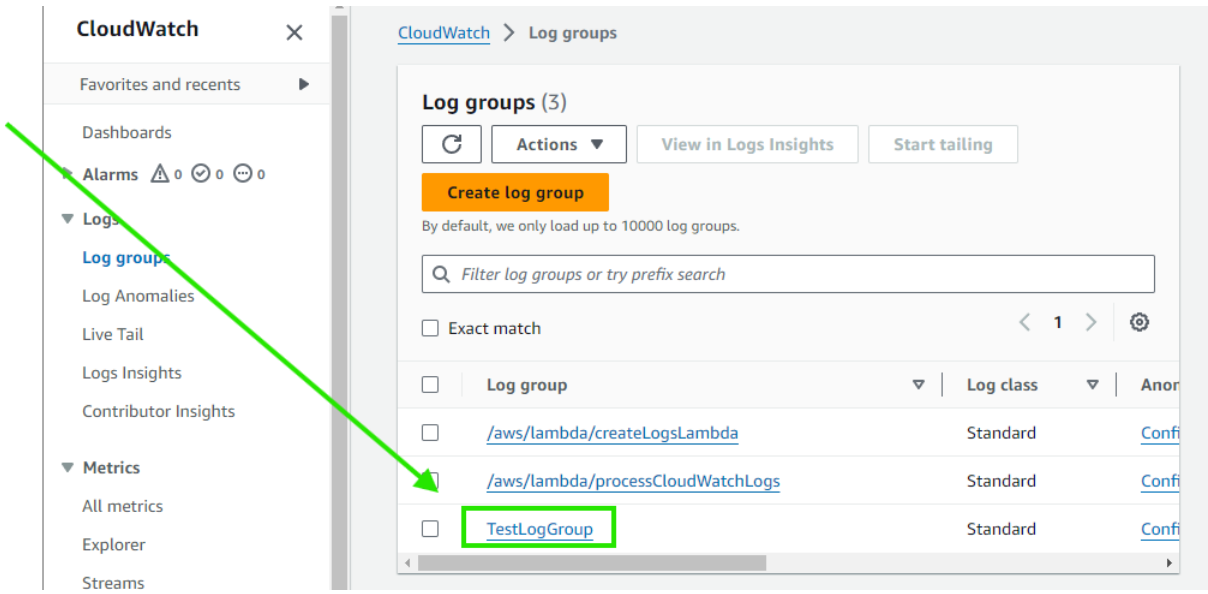
1. Navigate back to the CloudWatch Logs console.



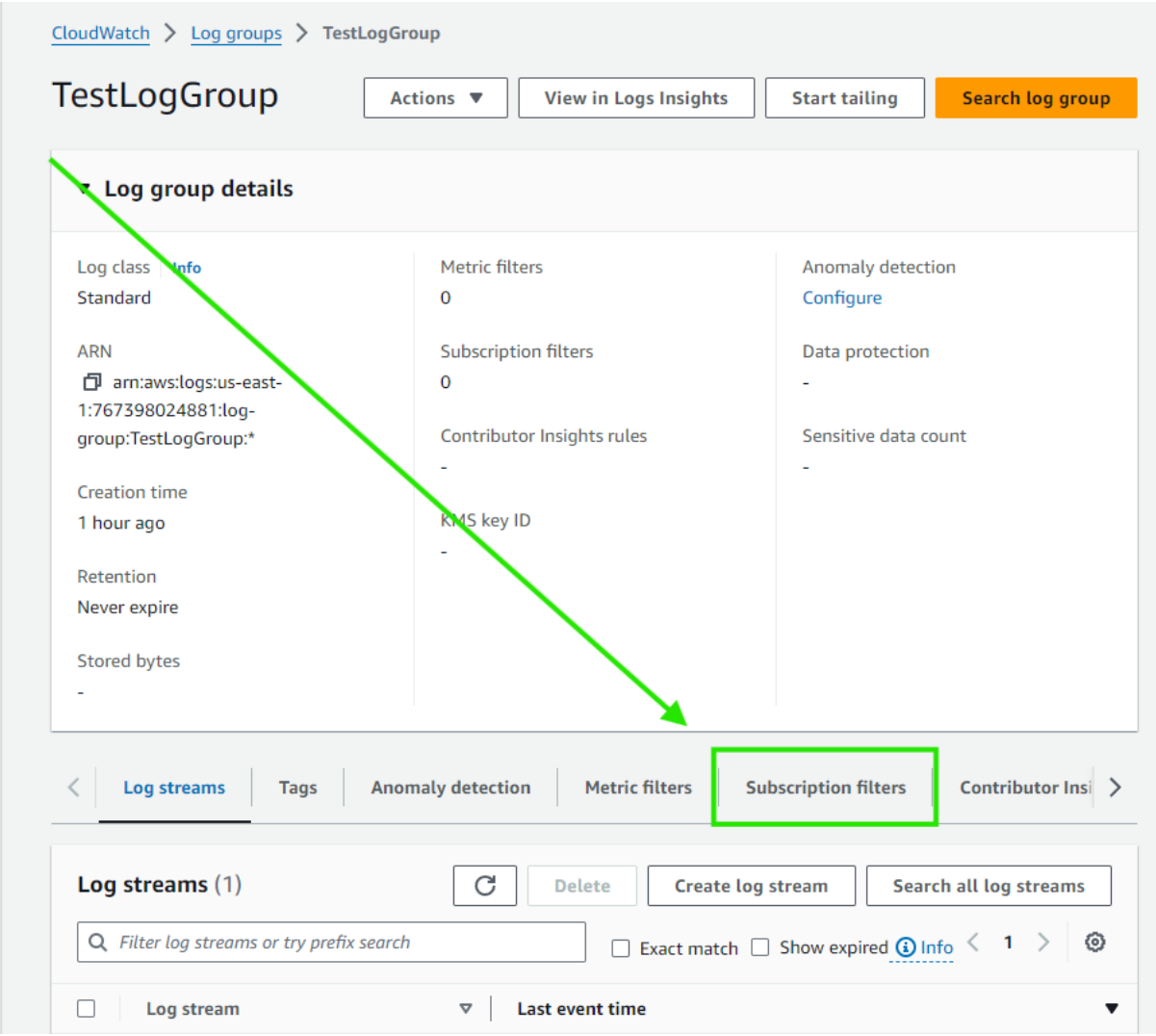
2. Click on **Log groups**



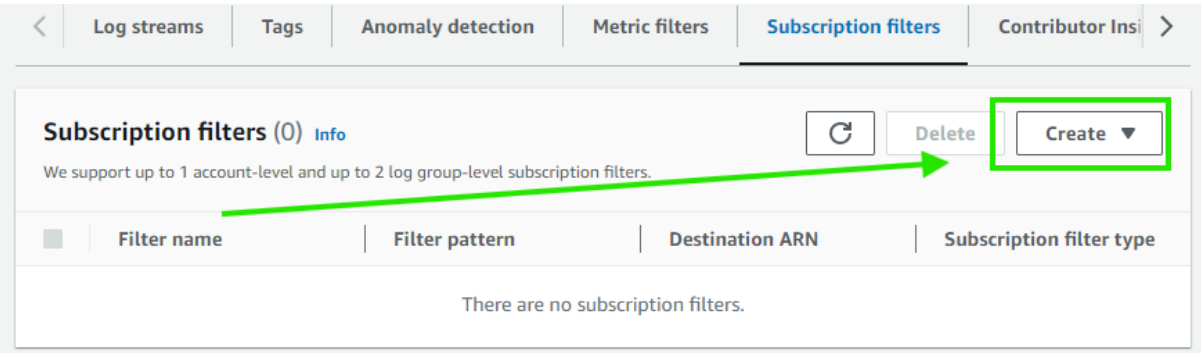
3. Click on TestLogGroup.



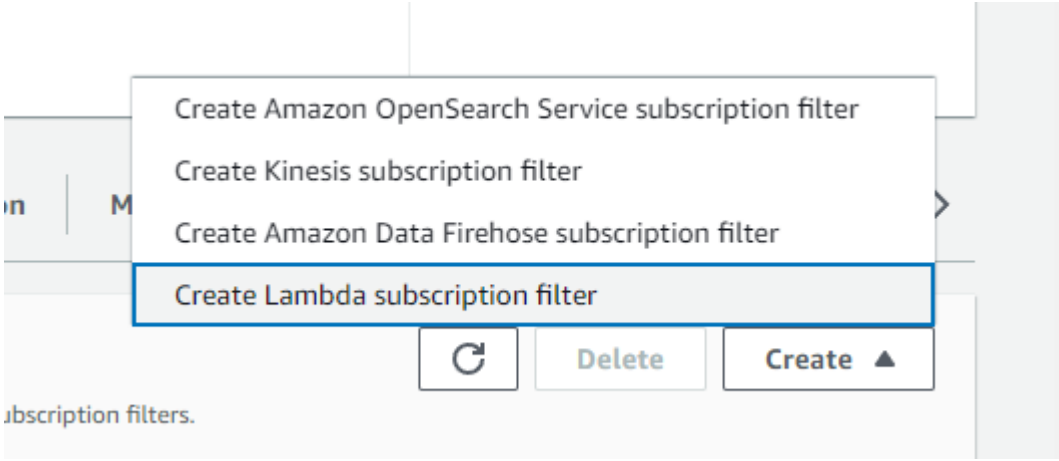
4. Navigate to the **Subscription Filter tab.**



5. Click on **Create**

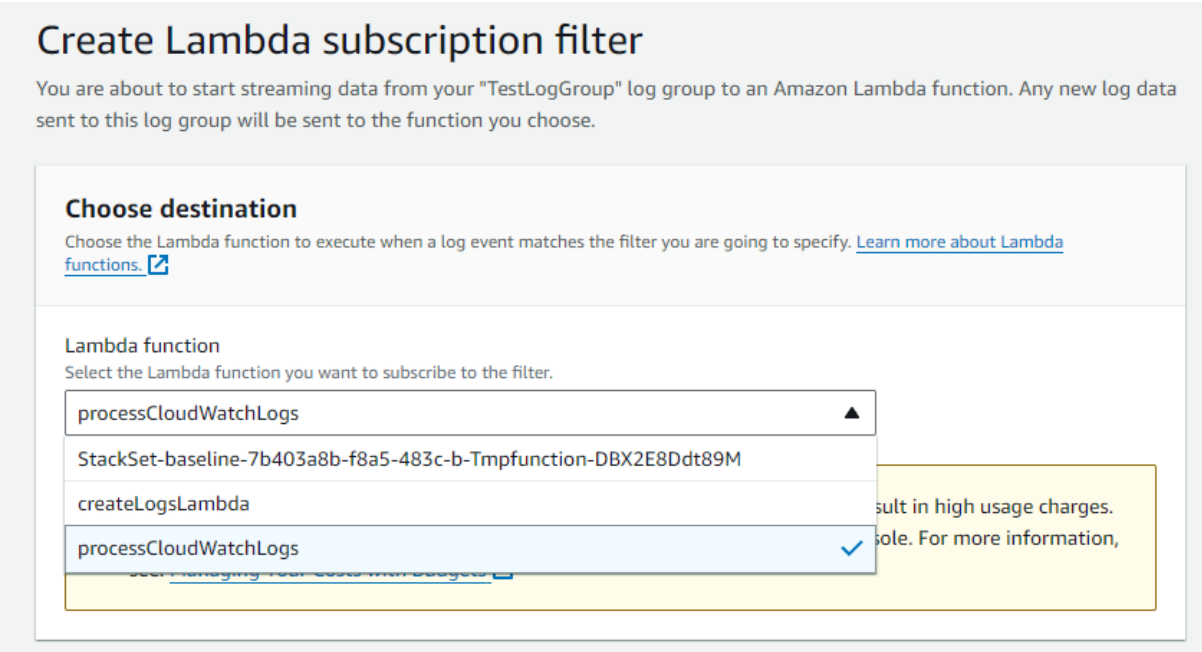


6. Select **Create Lambda subscription filter.**

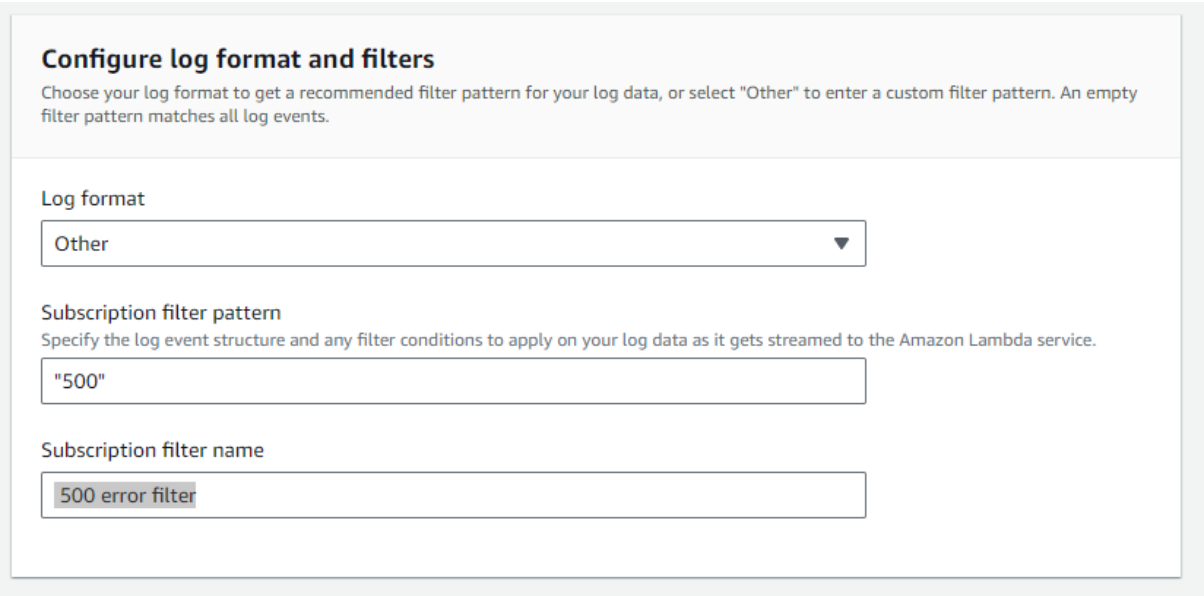


7. In the Create Lambda subscription filter page, follow the settings below:

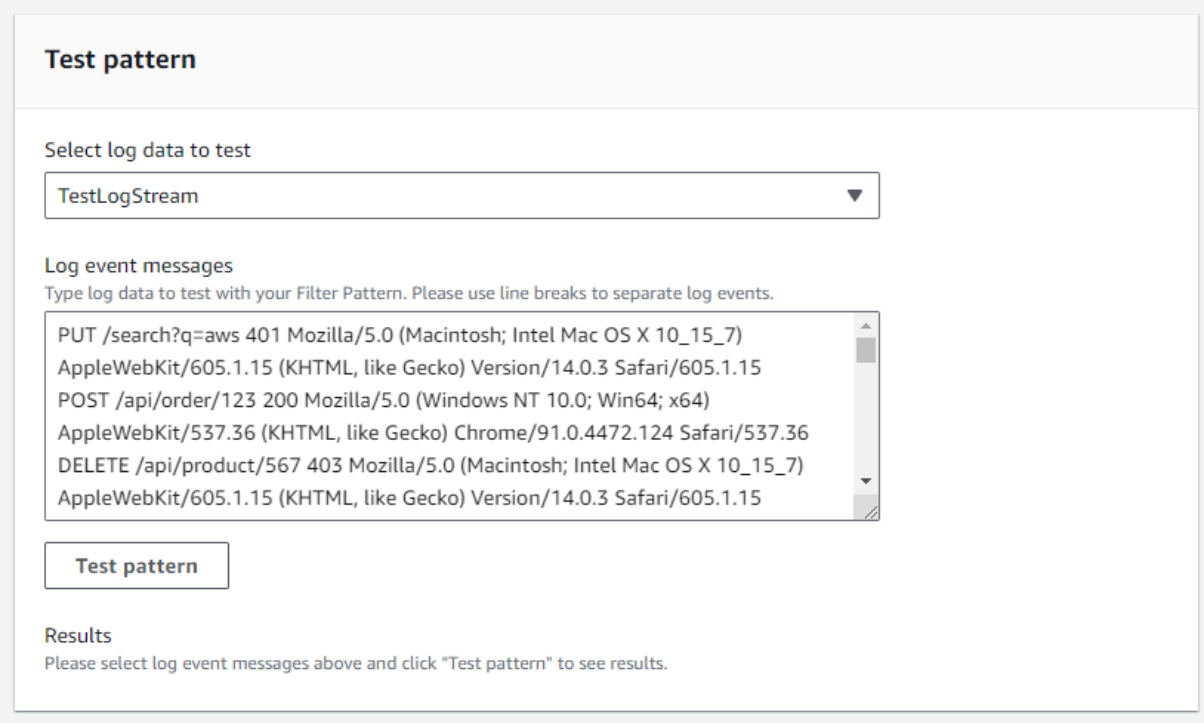
- Choose destination:
    - Lambda function: Select processCloudWatchLogs



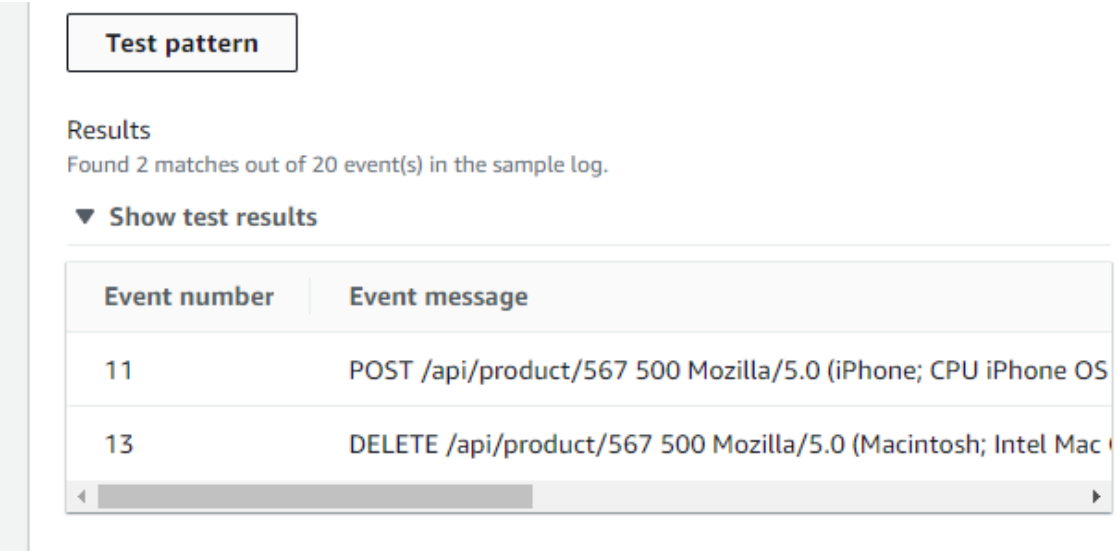
- **Configure log format and filters**:
    - Log format: Other
    - Subscription filter pattern: “500”
    - Subscription filter name: **500 error filter**



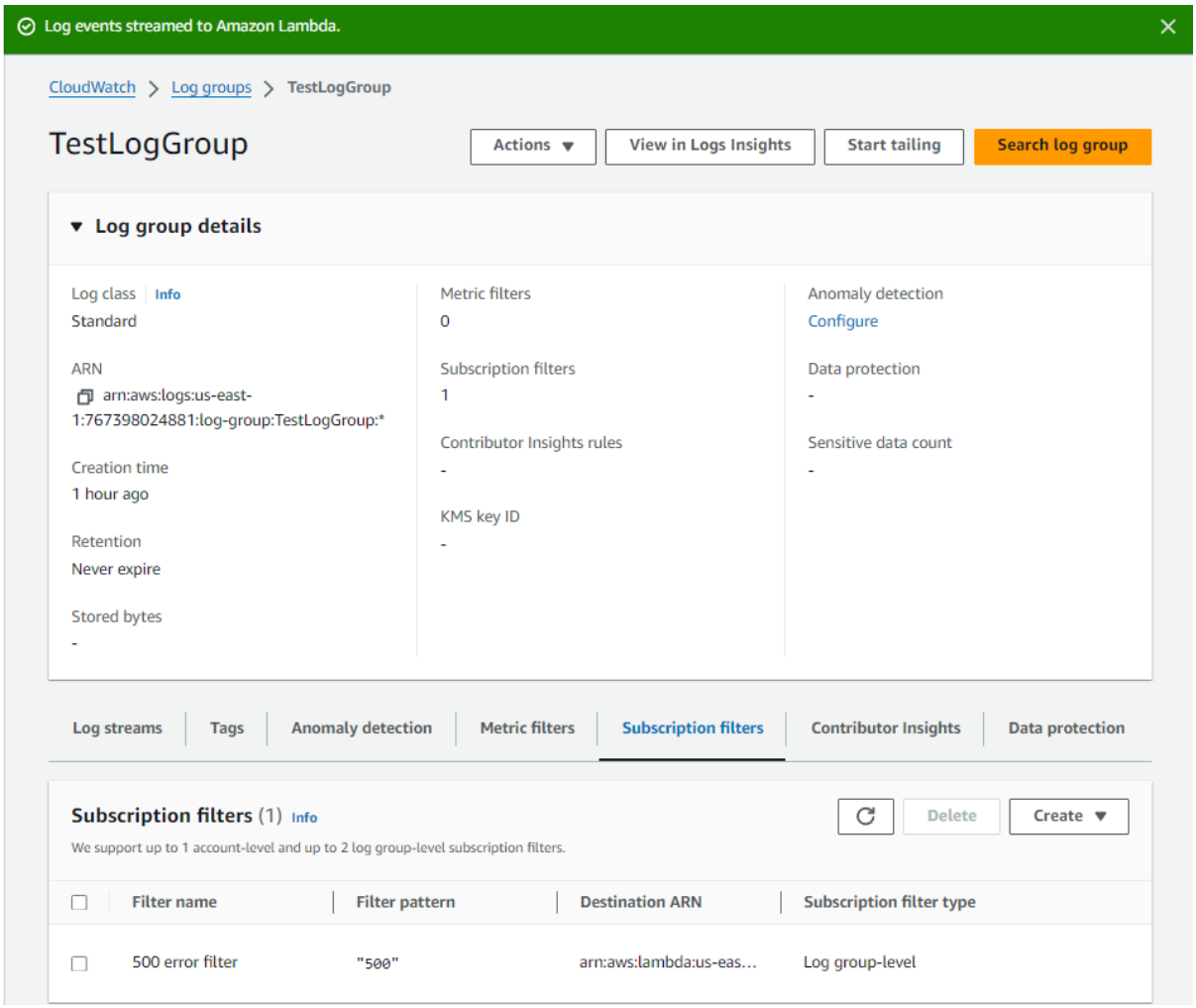
- **Test pattern**
    - Select log data to test: TestLogStream.



-
    - Click on **Test pattern** to test and check the Results. You should see the logs with HTTP 500 error requests.



- Click on **Start streaming.**



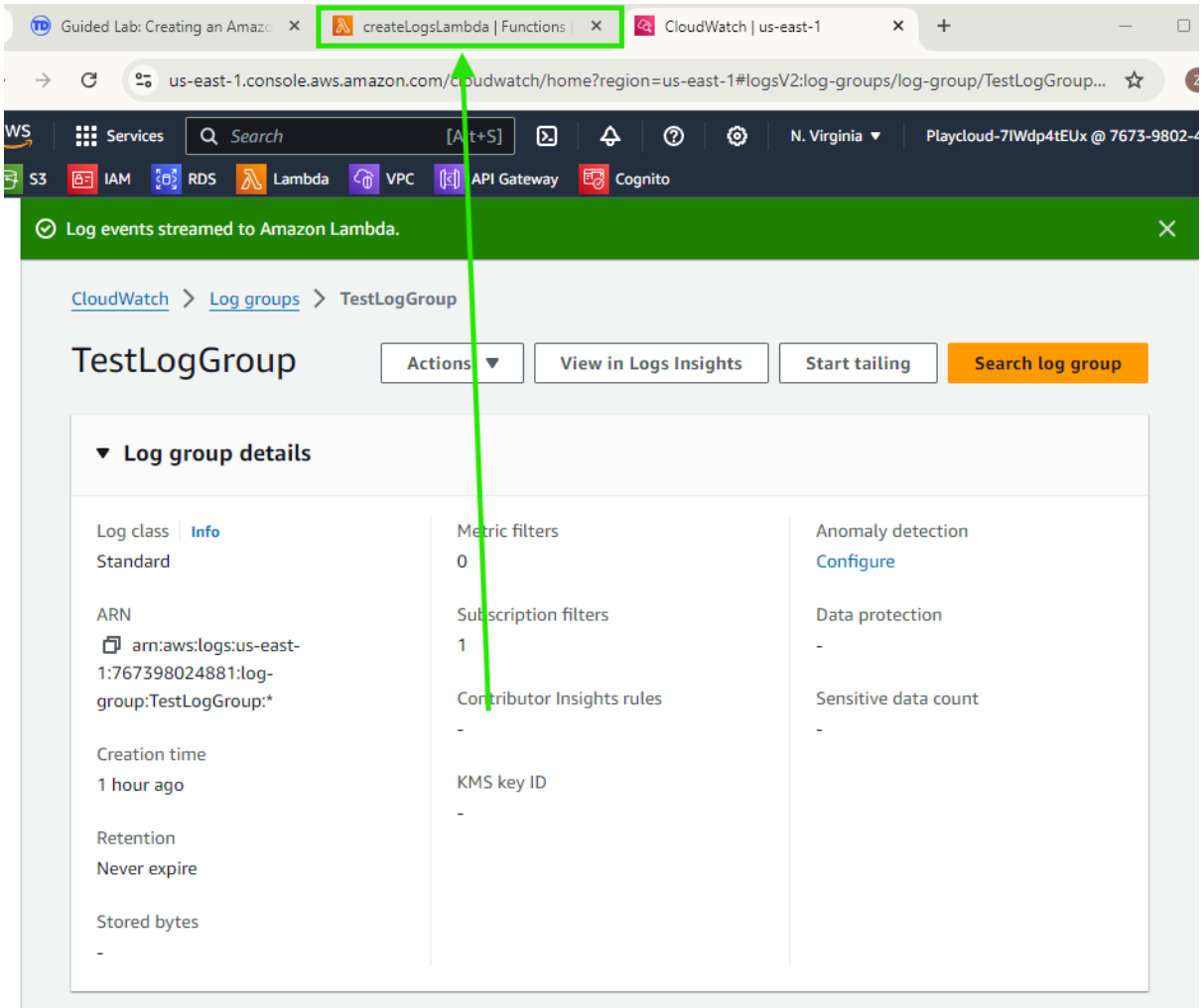
Testing the Log Processing

Finally, you'll test the entire setup by triggering the log creation Lambda function. This will add logs to the log group, triggering the log-processing Lambda function.

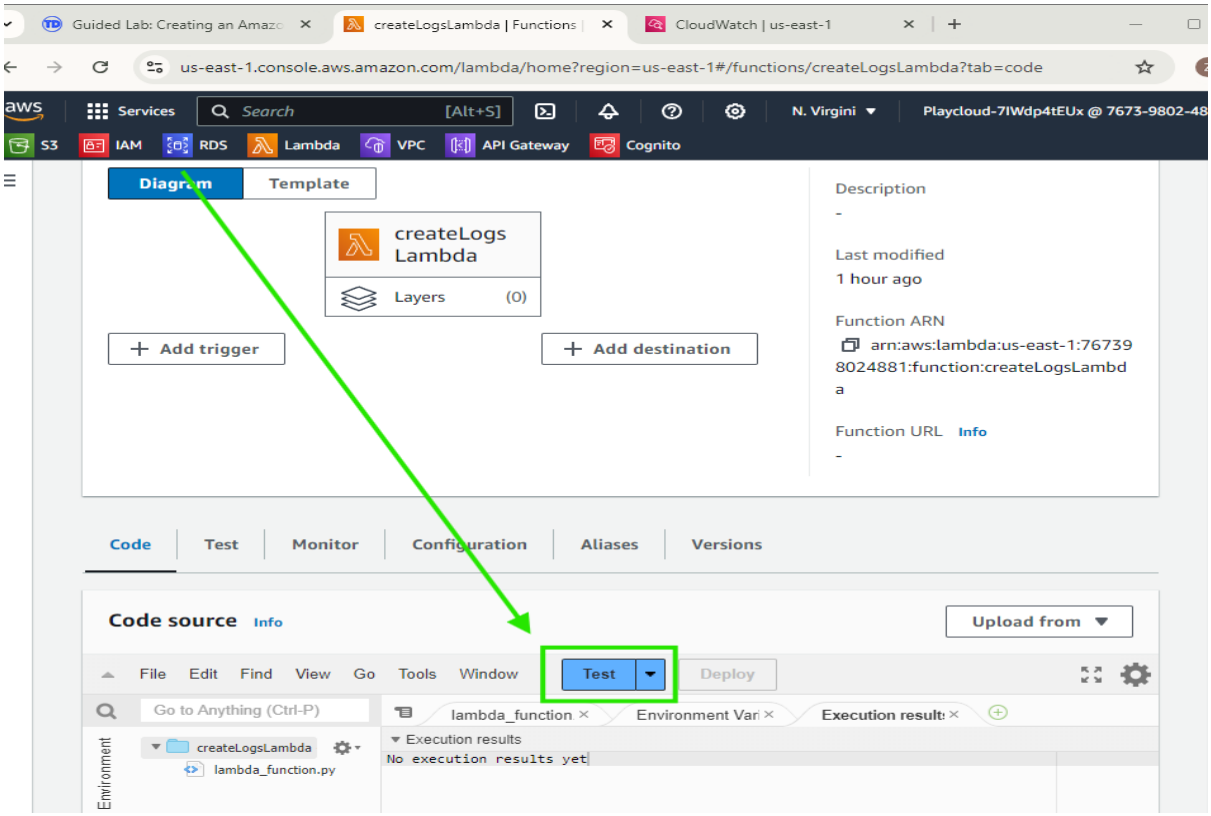
---

1. Return to the browser tab where the **createLogsLambda Function** is open.

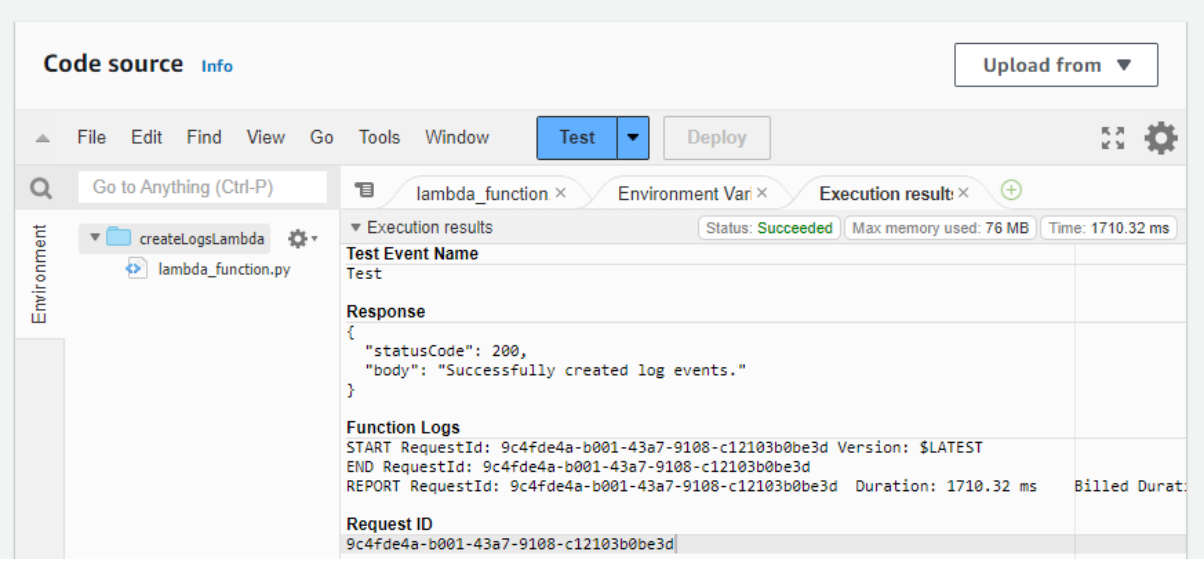
*Remember that in the **Creating Sample CloudWatch Logs Data** section, step 6, we left this tab open.*



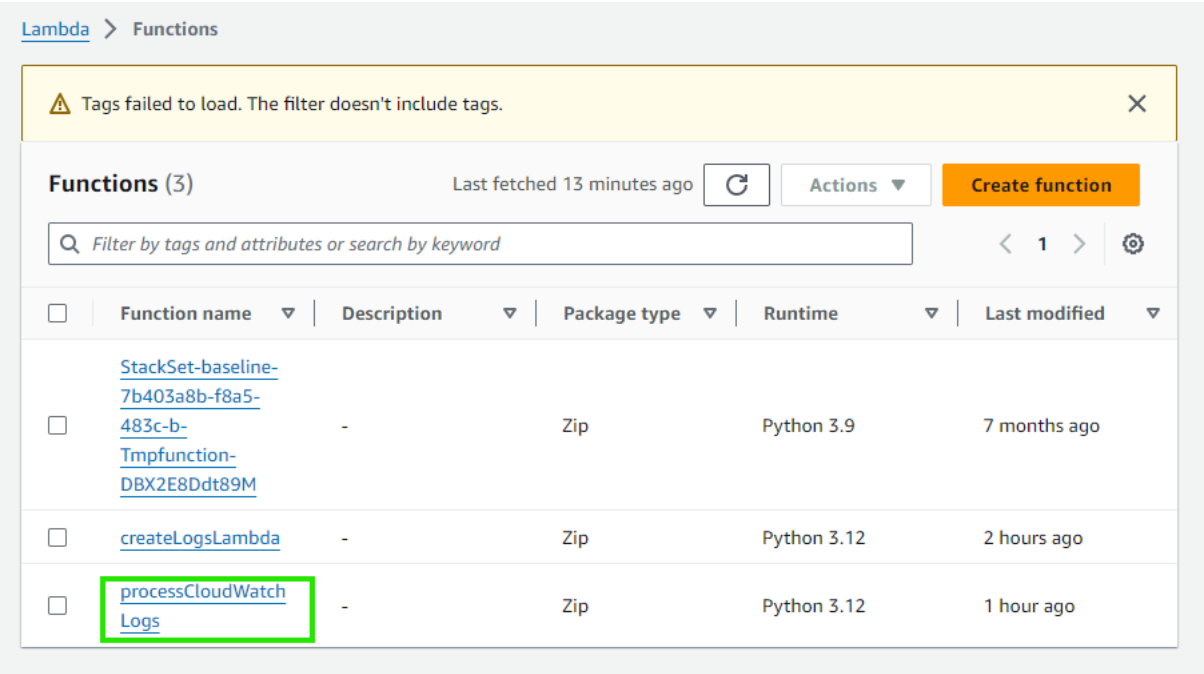
2. Trigger the Lambda by clicking on the **Test** button.



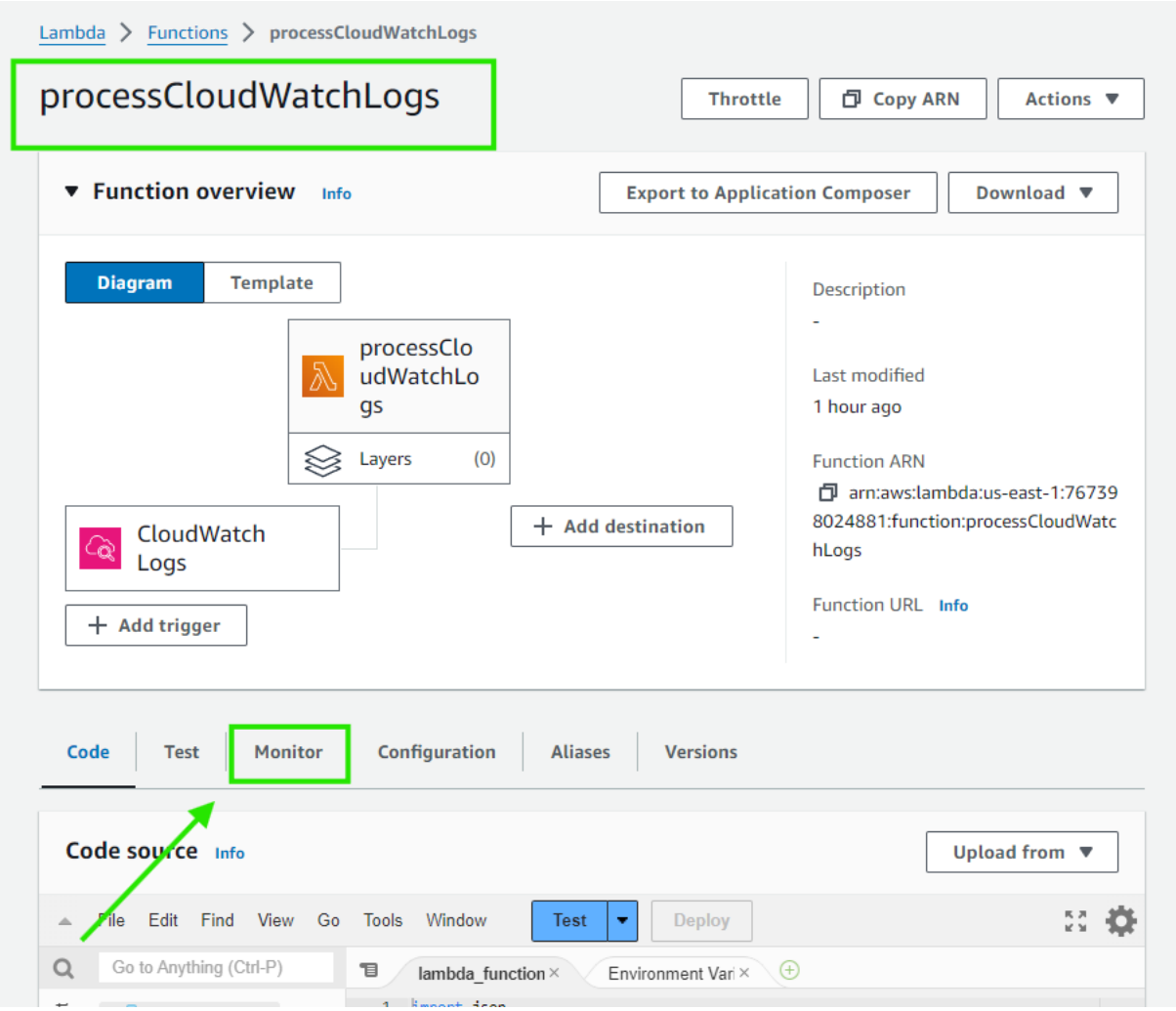
- After testing, it should successfully create log events.



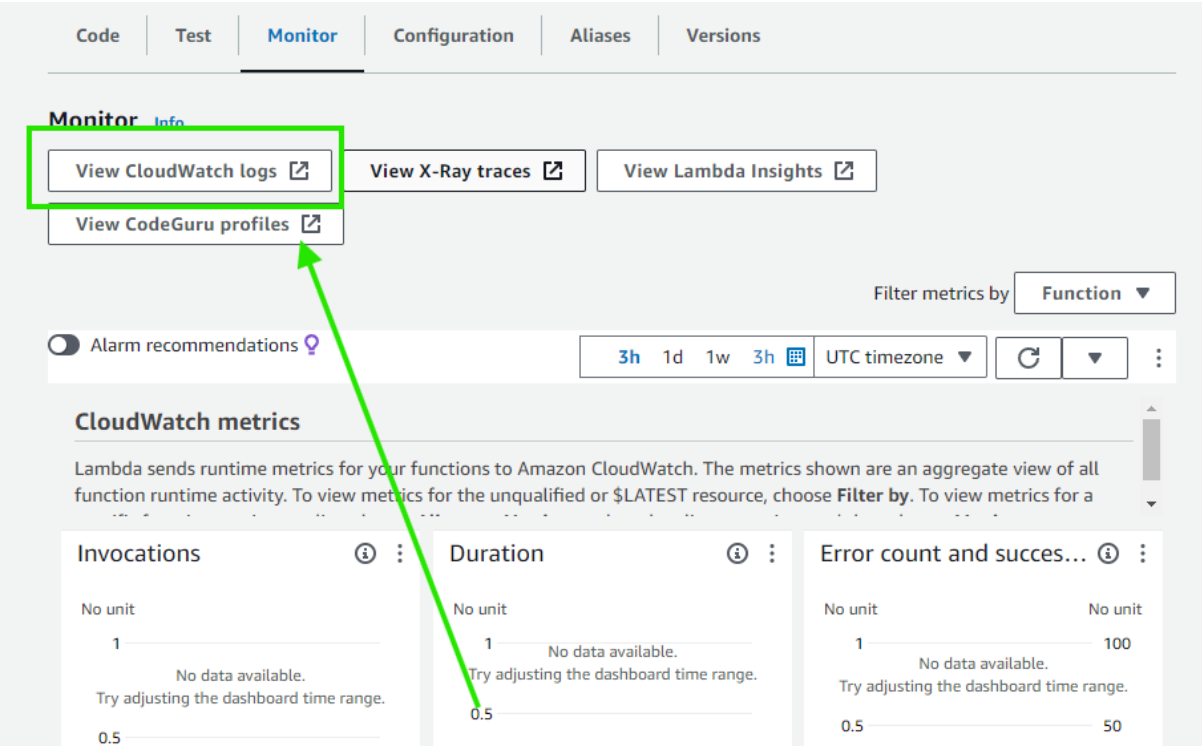
3. Return to **Lambda > Functions** Dashboard. Navigate to the processCloudWatchLogs Lambda Function.



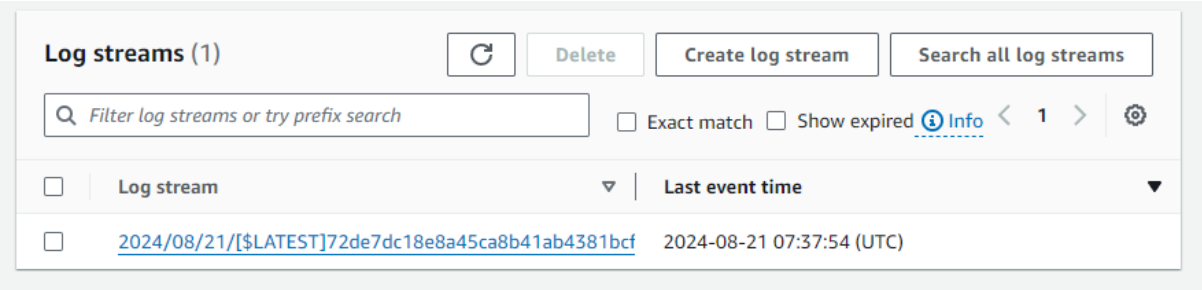
4. Then on, the **Monitor** tab



5. Click on **View CloudWatch logs.** You will be redirected to the CloudWatch logs for this lambda.



6. Scroll down and Click the latest Log streams



7. You should see similar logs just like the image below:

Notice that the Lambda function successfully processed the log events and identified the HTTP 500 requests.

That’s it! Congratulations! You have gained hands-on experience with AWS Lambda and Amazon CloudWatch Logs, setting up a complete workflow for generating, processing, and filtering log data. By simulating access logs and automatically detecting HTTP 500 error responses, you learned how Lambda functions could be utilized to monitor and react to critical events in real-time. This is an introductory demonstration of how AWS services can be integrated to create a powerful log processing pipeline.

While this lab focused on detecting specific error codes, the same principles can be applied to monitor other critical metrics or patterns in your logs. Additionally, the processed log data can be further integrated with other AWS services or external systems, such as sending notifications through Amazon SNS, triggering workflows in AWS Step Functions, or even notifying teams through platforms like Slack. Happy learning!