

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

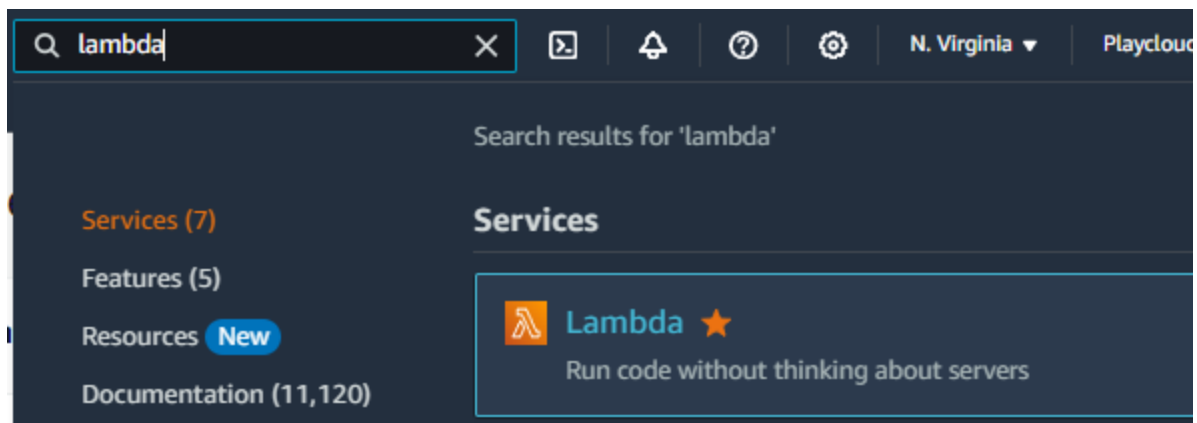
In this lab, you will:

- 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-Sandbox

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.

createLogsLambda

Use only letters, numbers, hyphens, or underscores with no spaces.

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

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

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.

PlayCloud-Sandbox

[View the PlayCloud-Sandbox role](#) on the IAM console.

- 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.

Successfully updated the function **createLogsLambda**.

Diagram

Template

 createLogsLambda

 Layers (0)

+ Add trigger

+ Add destination


Description

-

Last modified

8 minutes ago

Function ARN

 arn:aws:lambda:us-east-1:767398024881:function:createLogsLambda

Function URL

[Info](#)

-

Code

Test

Monitor

Configuration

Aliases

Versions

Code source [Info](#)

Upload from ▼

File Edit Find View Go Tools Window

Test ▼

Deploy

Go to Anything (Ctrl-P)

Environment

createLogsLambda

lambda_function.py

lambda_function ×

Environment Var ×

Execution results ×

+

```
1 import boto3
2 import time
3 import random
4
5 logs_client = boto3.client('logs')
6
7 def lambda_handler(event, context):
8     log_group_name = 'TestLogGroup'
9     log_stream_name = 'TestLogStream'
10
11     # Create log group if it doesn't exist
12     try:
13         logs_client.create_log_group(logGroupName=log_group_name)
14     except logs_client.exceptions.ResourceAlreadyExistsException:
```

5. Now, click on the Test button.

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

Configure test event



A test event is a JSON object that mocks the structure of requests emitted by AWS services to invoke a Lambda function. Use it to see the function's invocation result.

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

Test

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

hello-world

Event JSON

Format JSON

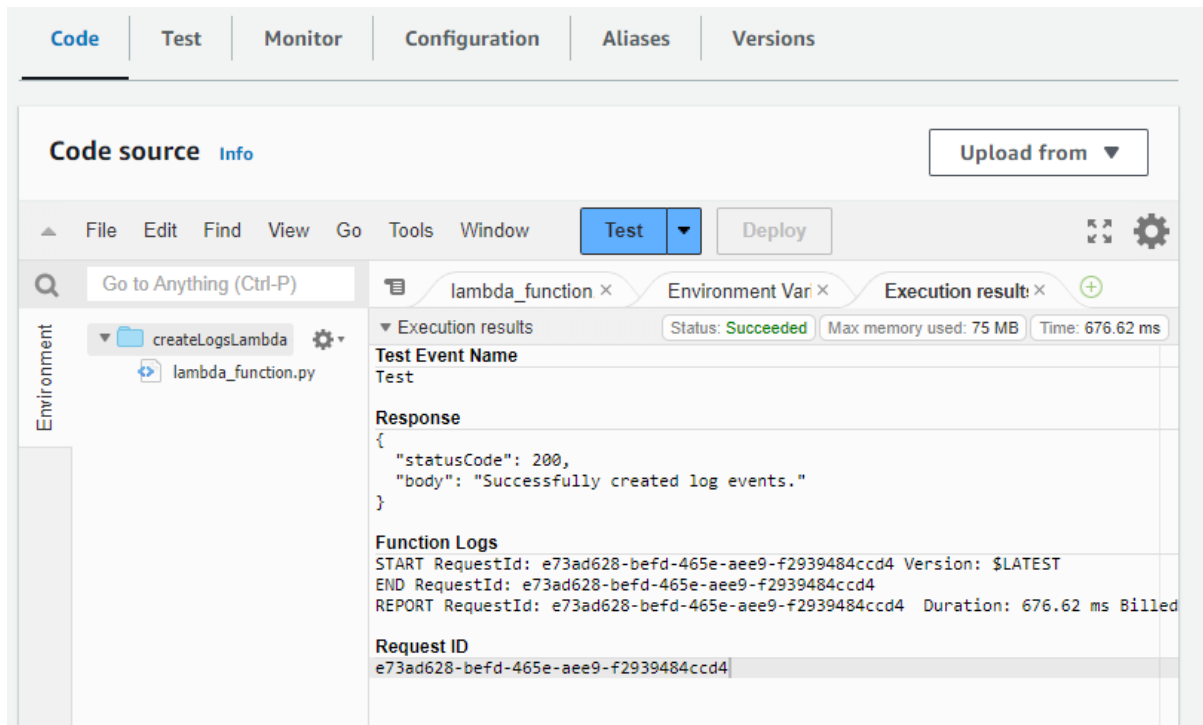
```
1 {  
2   "key1": "value1",  
3   "key2": "value2",  
4   "key3": "value3"  
5 }
```

Cancel

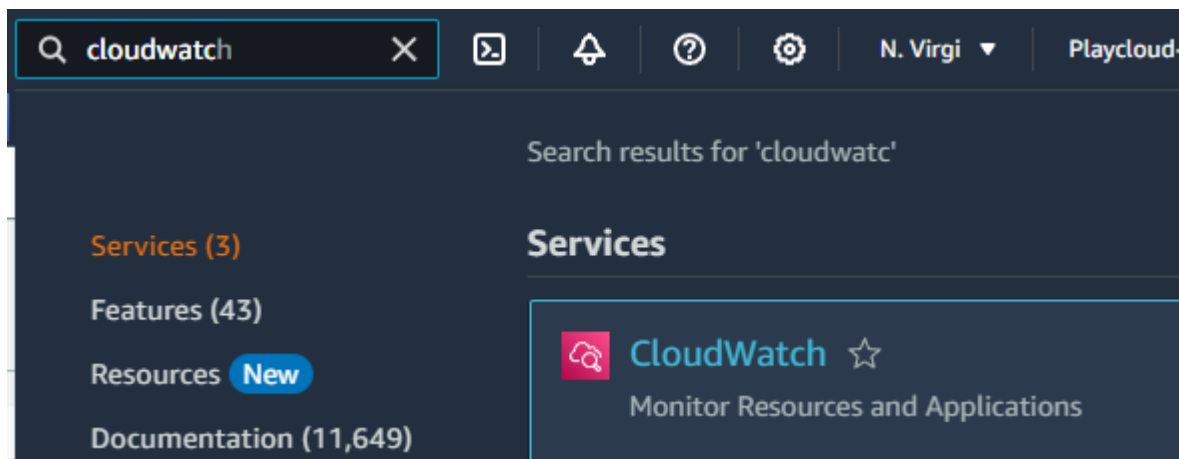
Invoke

Save

- 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.

CloudWatch > Log groups > TestLogGroup > TestLogStream

Log events ↻ Actions ▼ Start tailing Create metric filter

You can use the filter bar below to search for and match terms, phrases, or values in your log events. [Learn more about filter patterns](#) [↗](#)

1m 1h Clear UTC timezone ▼ Display ▼ ⚙

▶	Timestamp	Message
		No older events at this moment. Retry
▶	2024-08-21T05:40:00.676Z	PUT /search?q=aws 401 Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWe...
▶	2024-08-21T05:40:00.676Z	POST /api/order/123 200 Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit...
▶	2024-08-21T05:40:00.676Z	DELETE /api/product/567 403 Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) A...
▶	2024-08-21T05:40:00.676Z	DELETE /api/user 200 Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/53...
▶	2024-08-21T05:40:00.676Z	GET /api/user 400 Mozilla/5.0 (Linux; Android 10; SM-G973F) AppleWebKit/537.3...
▶	2024-08-21T05:40:00.676Z	DELETE /home 201 Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36...
▶	2024-08-21T05:40:00.676Z	GET /search?q=aws 201 Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWe...
▶	2024-08-21T05:40:00.676Z	DELETE /checkout 200 Mozilla/5.0 (Linux; Android 10; SM-G973F) AppleWebKit/53...
▶	2024-08-21T05:40:00.676Z	GET /login 201 Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/60...
▶	2024-08-21T05:40:00.676Z	POST /products 403 Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537...
▶	2024-08-21T05:40:00.676Z	POST /api/product/567 500 Mozilla/5.0 (iPhone; CPU iPhone OS 14_0 like Mac OS...
▶	2024-08-21T05:40:00.676Z	POST /home 201 Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (...)
▶	2024-08-21T05:40:00.676Z	DELETE /api/product/567 500 Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) A...
▶	2024-08-21T05:40:00.676Z	DELETE /products 403 Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/53...

Creating the Log Processing Lambda Function

1. Navigate to Lambda Function Console again.

Search results for 'lambda'


Services (7)

Features (5)

Resources **New**

Documentation (11,120)

Services

 **Lambda** ★
Run code without thinking about servers

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

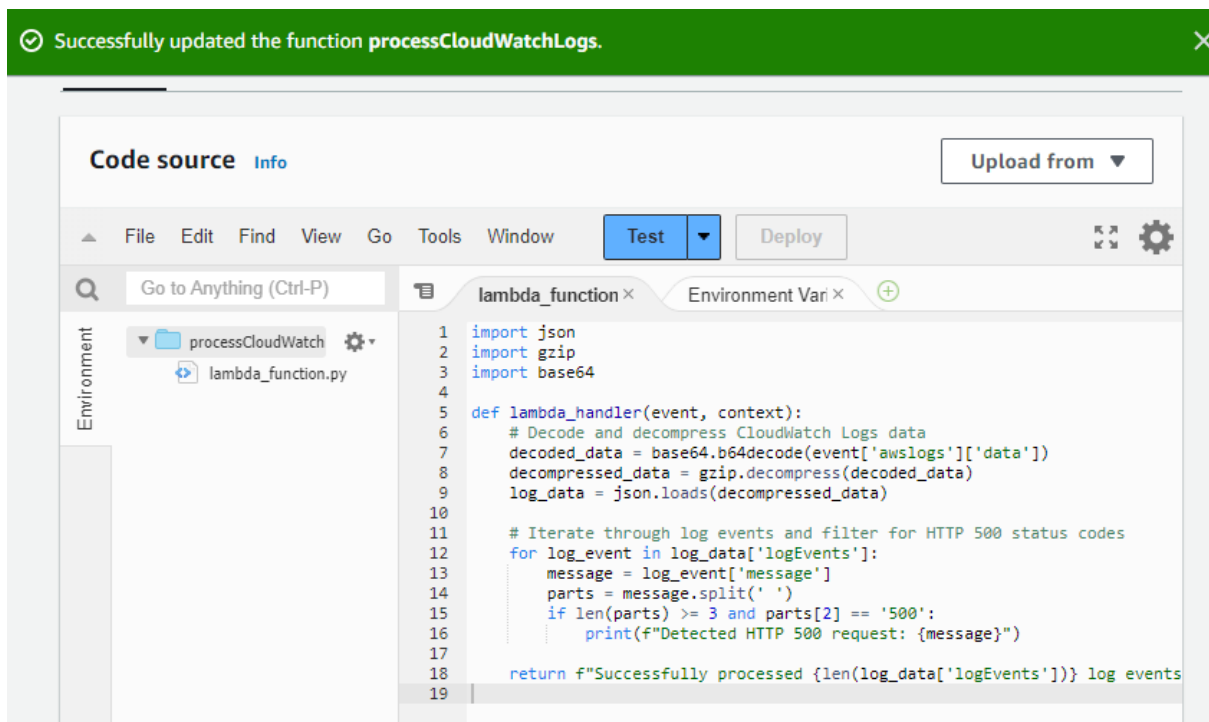
def lambda_handler(event, context):
    # Decode and decompress CloudWatch Logs data
    decoded_data = base64.b64decode(event['awslogs']['data'])
    decompressed_data = gzip.decompress(decoded_data)
    log_data = json.loads(decompressed_data)

    # 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.

Configure test event

A test event is a JSON object that mocks the structure of requests emitted by AWS services to invoke a Lambda function. Use it to see the function's invocation result.

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

CW_TEST

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

Event sharing settings

☒ Private

☐ Shareable

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

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

Template - optional

cloudwatch-logs

Event JSON

Format JSON

1 {

2 "awslogs": {

3 "data": "H4sIAAAAAAAAAAHwPwQqCQBCGX0Xm7EFtK+smZBEUgXoLCdHhFtKV3akI8d0bLYmibvPPN3wz00CJxmQnT041whwWQRIctmEcB6s

4 }

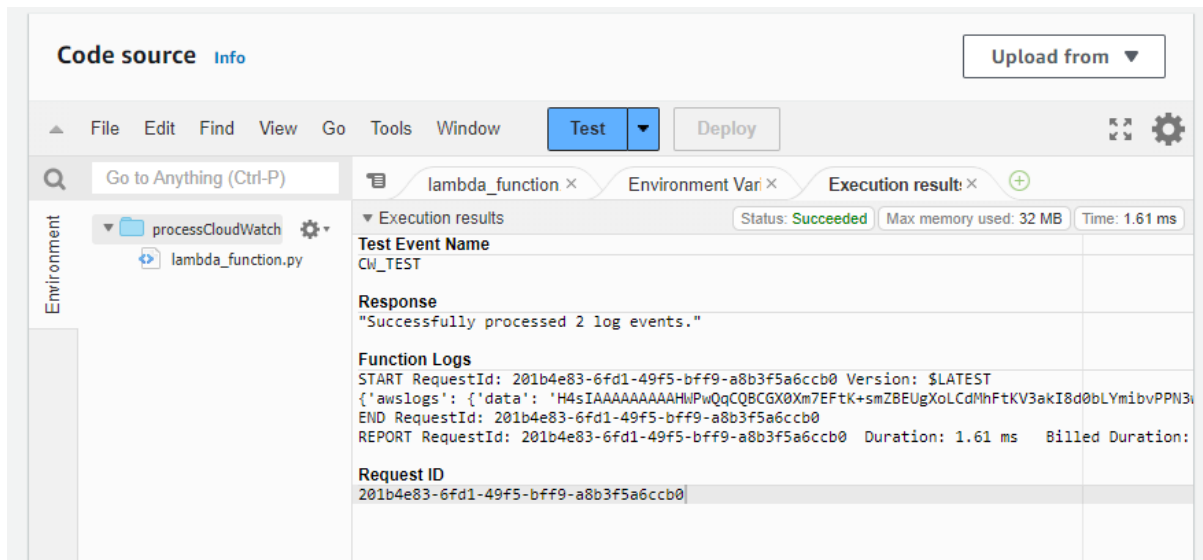
5 }

Cancel

Invoke

Save

- Click on **Test** again.

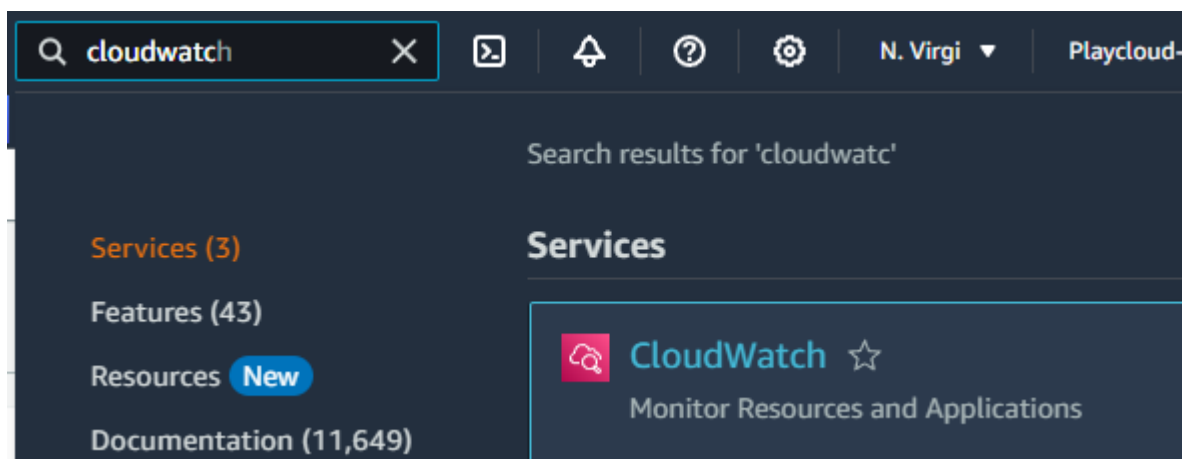


This shows the Lambda function's code is working just fine. It will also create a CloudWatch log group for this Lambda function.

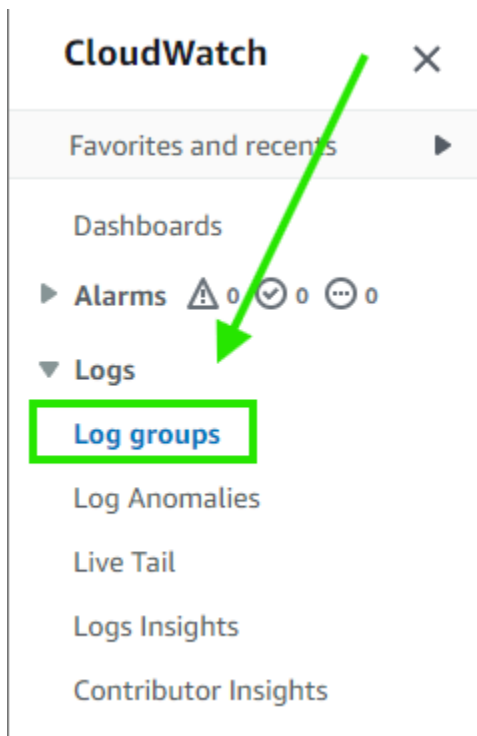
Creating the CloudWatch Logs Subscription Filter

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

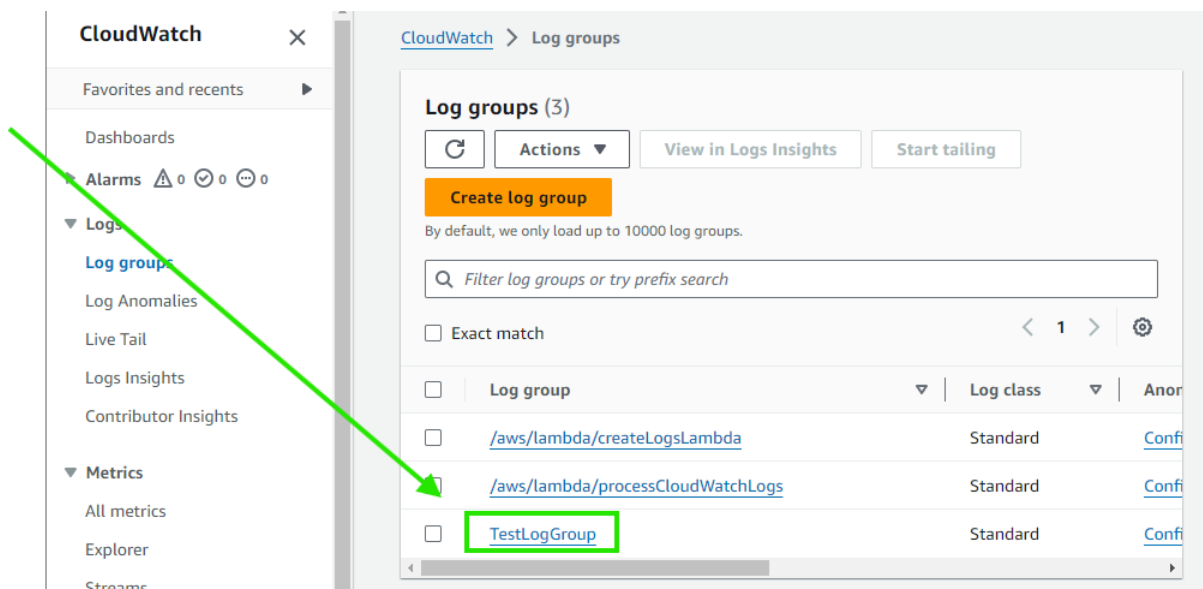
1. Navigate back to the CloudWatch Logs console.



2. Click on **Log groups**



3. Click on TestLogGroup.



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

CloudWatch > Log groups > TestLogGroup

TestLogGroup

Actions ▾ View in Logs Insights Start tailing Search log group

Log group details

Log class Info Standard	Metric filters 0	Anomaly detection Configure
ARN <code>arn:aws:logs:us-east-1:767398024881:log-group:TestLogGroup:*</code>	Subscription filters 0	Data protection -
Creation time 1 hour ago	Contributor Insights rules -	Sensitive data count -
Retention Never expire	KMS key ID -	
Stored bytes -		

< Log streams Tags Anomaly detection Metric filters **Subscription filters** Contributor Insights >

Log streams (1) [Delete](#) [Create log stream](#) [Search all log streams](#)

☐ Exact match ☐ Show expired [Info](#) < 1 >

<input type="checkbox"/>	Log stream	▼	Last event time	▼
--------------------------	------------	---	-----------------	---

5. Click on **Create**

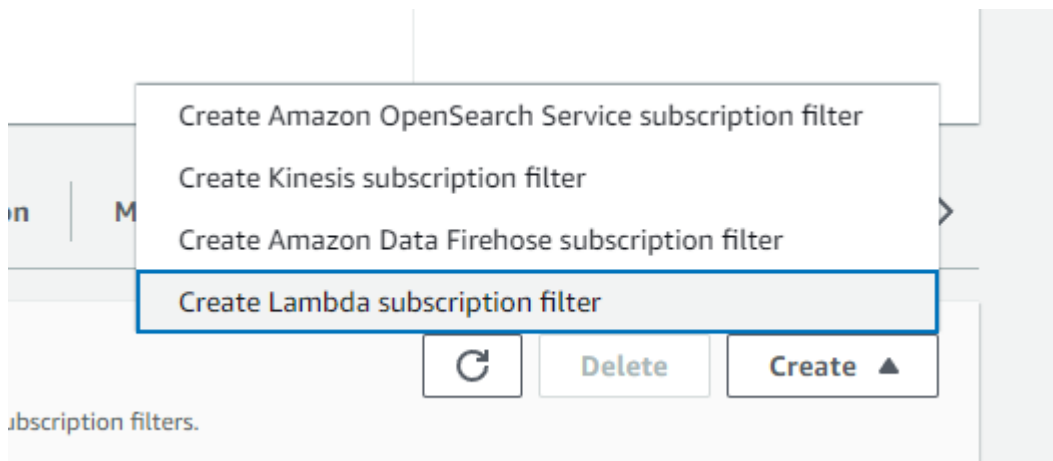
< Log streams Tags Anomaly detection Metric filters **Subscription filters** Contributor Insights >

Subscription filters (0) [Info](#) [Delete](#) [Create ▾](#)

We support up to 1 account-level and up to 2 log group-level subscription filters.

<input type="checkbox"/>	Filter name	Filter pattern	Destination ARN	Subscription filter type
There are no subscription filters.				

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



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

- Choose destination:
 - Lambda function: Select processCloudWatchLogs

Create Lambda subscription filter

You are about to start streaming data from your "TestLogGroup" log group to an Amazon Lambda function. Any new log data sent to this log group will be sent to the function you choose.

Choose destination

Choose the Lambda function to execute when a log event matches the filter you are going to specify. [Learn more about Lambda functions.](#)

Lambda function

Select the Lambda function you want to subscribe to the filter.

processCloudWatchLogs

StackSet-baseline-7b403a8b-f8a5-483c-b-Tmpfunction-DBX2E8Ddt89M

createLogsLambda

processCloudWatchLogs

result in high usage charges. For more information,

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

Configure log format and filters

Choose your log format to get a recommended filter pattern for your log data, or select "Other" to enter a custom filter pattern. An empty filter pattern matches all log events.

Log format

Other

Subscription filter pattern

Specify the log event structure and any filter conditions to apply on your log data as it gets streamed to the Amazon Lambda service.

"500"

Subscription filter name

500 error filter

- **Test pattern**

- Select log data to test: TestLogStream.

Test pattern

Select log data to test

TestLogStream

Log event messages

Type log data to test with your Filter Pattern. Please use line breaks to separate log events.

```
PUT /search?q=aws 401 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
POST /api/order/123 200 Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.124 Safari/537.36
DELETE /api/product/567 403 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
```

Test pattern

Results

Please select log event messages above and click "Test pattern" to see results.

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

Test pattern

Results

Found 2 matches out of 20 event(s) in the sample log.

▼ Show test results

Event number	Event message
11	POST /api/product/567 500 Mozilla/5.0 (iPhone; CPU iPhone OS
13	DELETE /api/product/567 500 Mozilla/5.0 (Macintosh; Intel Mac

- Click on **Start streaming**.

Log events streamed to Amazon Lambda.

[CloudWatch](#) > [Log groups](#) > TestLogGroup

TestLogGroup

Actions ▼

View in Logs Insights

Start tailing

Search log group

▼ Log group details

Log class [Info](#)

Standard

ARN
 `arn:aws:logs:us-east-1:767398024881:log-group:TestLogGroup:*`

Creation time
1 hour ago

Retention
Never expire

Stored bytes
-

Metric filters
0

Subscription filters
1

Contributor Insights rules
-

KMS key ID
-

Anomaly detection
[Configure](#)

Data protection
-

Sensitive data count
-

Log streams

Tags

Anomaly detection

Metric filters

Subscription filters

Contributor Insights

Data protection

Subscription filters (1) [Info](#)

Delete

Create ▼

We support up to 1 account-level and up to 2 log group-level subscription filters.

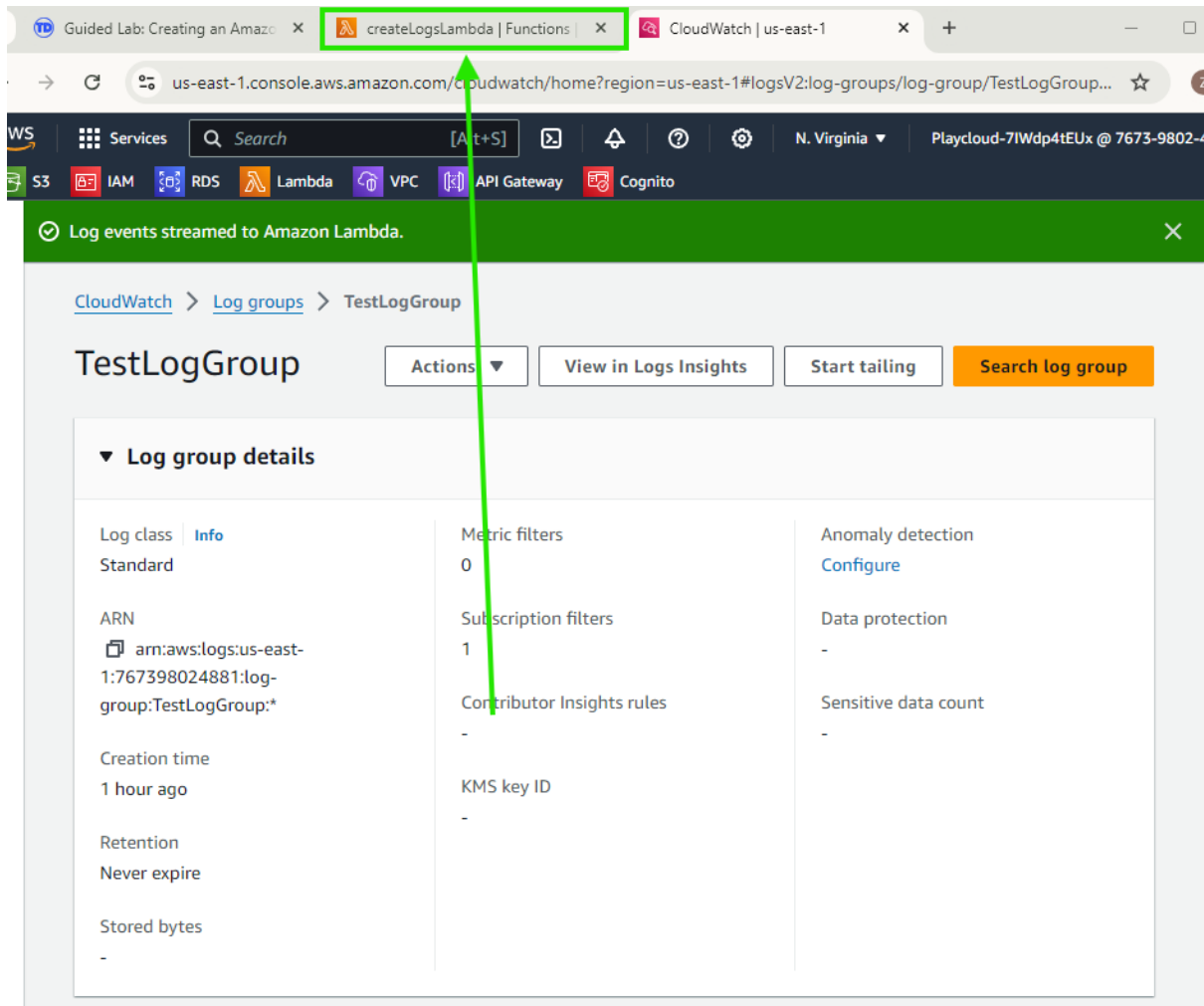
<input type="checkbox"/>	Filter name	Filter pattern	Destination ARN	Subscription filter type
<input type="checkbox"/>	500 error filter	"500"	arn:aws:lambda:us-eas...	Log group-level

Testing the Log Processing

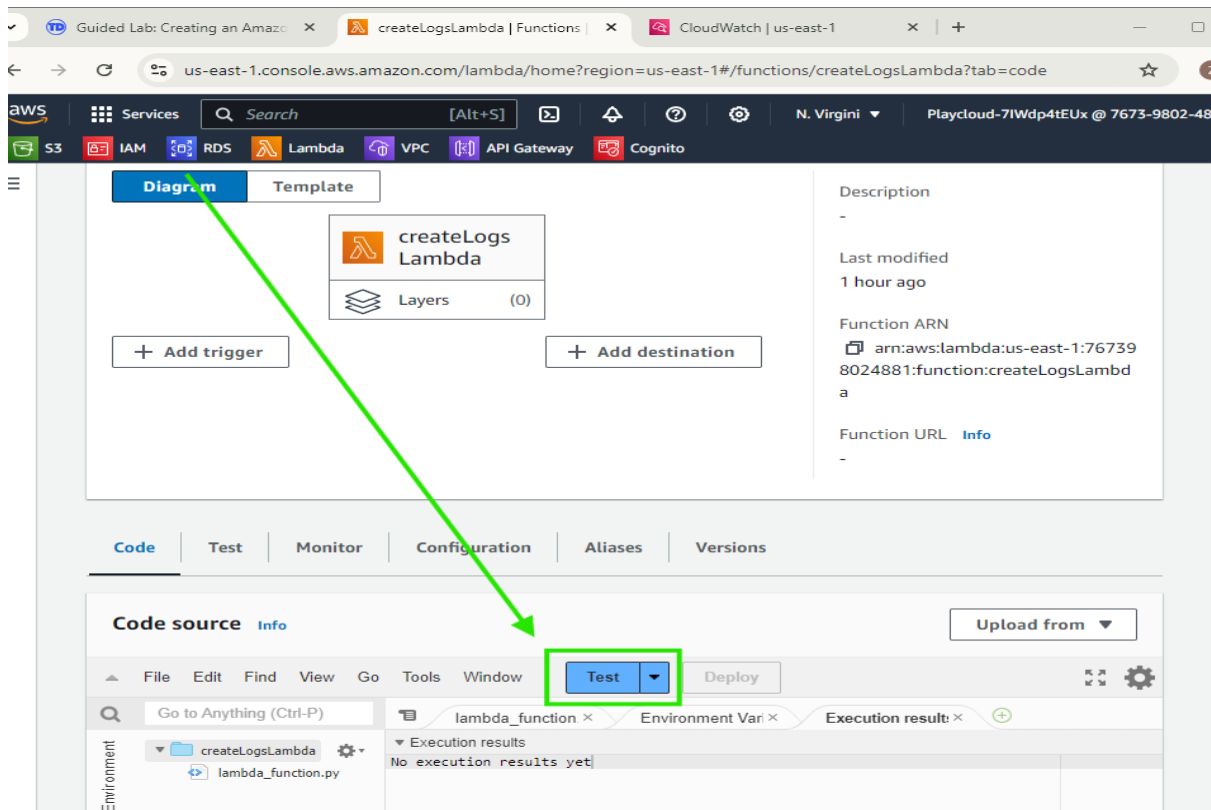
Finally, you'll test the entire setup by triggering the log creation Lambda function. This will add logs to the log group, triggering the log-processing Lambda function.

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

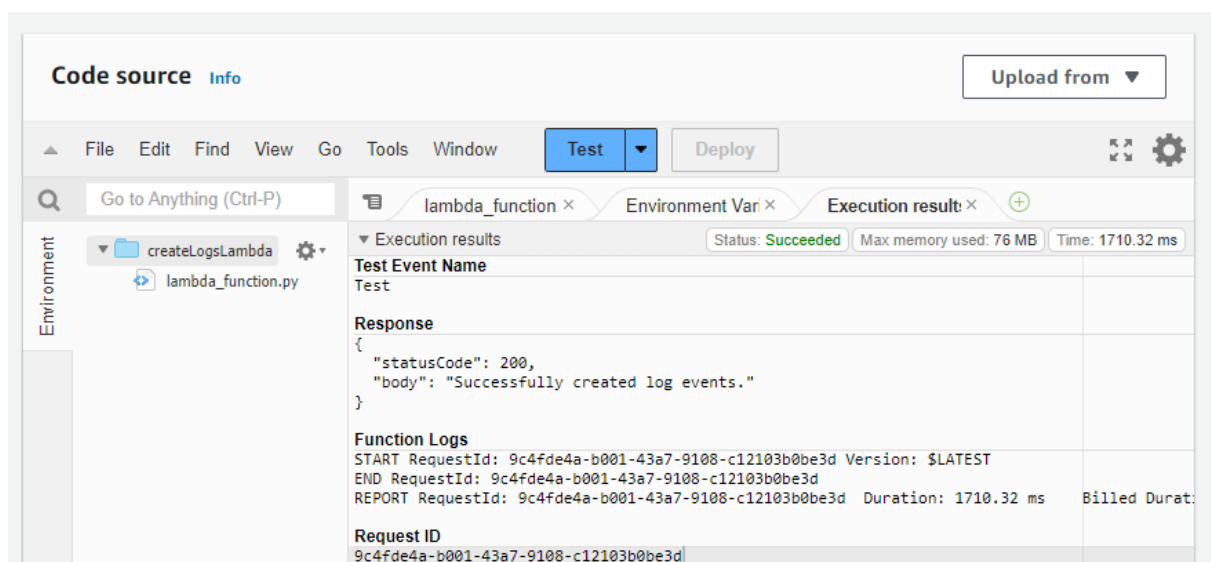
*Remember that in the **Creating Sample CloudWatch Logs Data** section, step 6, we left this tab open.*



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.

Lambda > Functions

⚠ Tags failed to load. The filter doesn't include tags. ✕

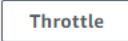
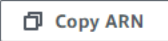

Functions (3) Last fetched 13 minutes ago  **Actions**  **Create function**

🔍 Filter by tags and attributes or search by keyword

<input type="checkbox"/>	Function name	Description	Package type	Runtime	Last modified
<input type="checkbox"/>	StackSet-baseline-7b403a8b-f8a5-483c-b-Tmpfunction-DBX2E8Ddt89M	-	Zip	Python 3.9	7 months ago
<input type="checkbox"/>	createLogsLambda	-	Zip	Python 3.12	2 hours ago
<input type="checkbox"/>	processCloudWatchLogs	-	Zip	Python 3.12	1 hour ago

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

Lambda > Functions > processCloudWatchLogs

processCloudWatchLogs   **Actions** 







▼ **Function overview** [Info](#)  **Download** 


Diagram **Template**

 processCloudWatchLogs

 Layers (0)


 CloudWatch Logs

 **Add trigger**

 **Add destination**


Description
-


Last modified
1 hour ago

Function ARN
 arn:aws:lambda:us-east-1:767398024881:function:processCloudWatchLogs

Function URL [Info](#)
-

Code **Test** **Monitor** **Configuration** **Aliases** **Versions**

Code source [Info](#) 

File Edit Find View Go Tools Window **Test**  **Deploy**

Go to Anything (Ctrl-P) lambda_function x Environment Vari x

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

The screenshot shows the AWS Lambda console's 'Monitor' tab. At the top, there are tabs for 'Code', 'Test', 'Monitor' (selected), 'Configuration', 'Aliases', and 'Versions'. Below these, there are buttons for 'View CloudWatch logs', 'View X-Ray traces', and 'View Lambda Insights'. A green box highlights the 'View CloudWatch logs' button, and a green arrow points from it to the 'Log streams' section in the next screenshot. Below the buttons, there is a 'Filter metrics by' dropdown set to 'Function'. There is also a toggle for 'Alarm recommendations' and a time range selector set to '3h'. The main section is titled 'CloudWatch metrics' and contains three charts: 'Invocations', 'Duration', and 'Error count and success...'. All charts show 'No data available' and prompt the user to 'Try adjusting the dashboard time range'.

6. Scroll down and Click the latest Log streams

The screenshot shows the 'Log streams (1)' section in the AWS Lambda console. It includes a search bar, a 'Filter log streams or try prefix search' input, and buttons for 'Exact match', 'Show expired', 'Info', and a pagination control showing '1'. Below the search bar is a table with one log stream entry:

Log stream	Last event time
2024/08/21/[\$LATEST]72de7dc18e8a45ca8b41ab4381bcf	2024-08-21 07:37:54 (UTC)

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

The screenshot shows the content of the log stream. It is a list of log events with timestamps and messages. The messages include 'START', 'Detected HTTP 500 request', 'END', and 'REPORT'.

Timestamp	Message
2024-08-21T07:44:14.235Z	START RequestId: 9550c2fa-8db6-487b-96e8-3fc582b66fd9 Version: \$LATEST
2024-08-21T07:44:14.235Z	{'awslogs': {'data': 'H4sIAAAAAAAAA/9XTW0/bMBQA4L9i+QmkzPHdTVIUsdJNkFEpYZ00oSptX...
2024-08-21T07:44:14.235Z	Detected HTTP 500 request: POST /api/user 500 Mozilla/5.0 (iPhone; CPU iPhone O...
2024-08-21T07:44:14.235Z	Detected HTTP 500 request: POST /cart 500 Mozilla/5.0 (Macintosh; Intel Mac OS ...
2024-08-21T07:44:14.235Z	Detected HTTP 500 request: GET /cart 500 Mozilla/5.0 (Linux; Android 10; SM-G97...
2024-08-21T07:44:14.235Z	Detected HTTP 500 request: PUT /checkout 500 Mozilla/5.0 (Linux; Android 10; SM...
2024-08-21T07:44:14.235Z	Detected HTTP 500 request: PUT /api/user 500 Mozilla/5.0 (Linux; Android 10; SM...
2024-08-21T07:44:14.235Z	Detected HTTP 500 request: GET /checkout 500 Mozilla/5.0 (Windows NT 10.0; Win6...
2024-08-21T07:44:14.237Z	END RequestId: 9550c2fa-8db6-487b-96e8-3fc582b66fd9
2024-08-21T07:44:14.237Z	REPORT RequestId: 9550c2fa-8db6-487b-96e8-3fc582b66fd9 Duration: 2.10 ms Billed...

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!