

Real-Time Log Anomaly Detection Using OpenSearch

Vinod Bhatt
Eashan Kaushik
Kiet Nguyen
Hamilton Wong



NYU

TANDON SCHOOL
OF ENGINEERING



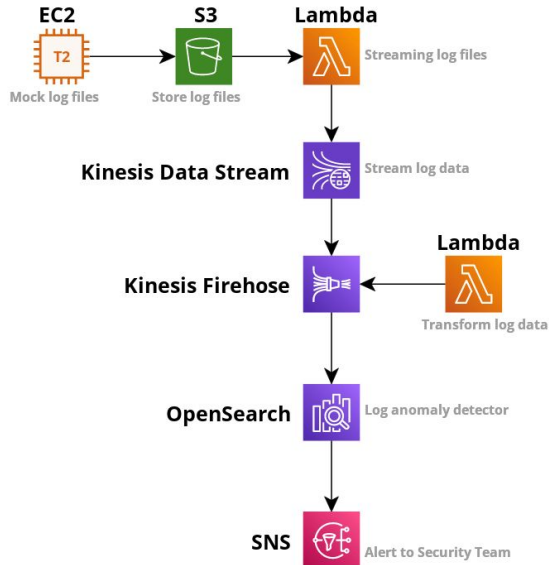
PART 01

Setup



Introduction and Architecture

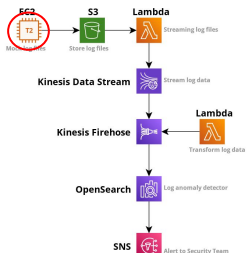
- Demoing an implementation of anomaly detection on WAF logs using OpenSearch and displayed in a dashboard
- These are fields such as Countries or Httpstatus: Good (1xx,2xx,3xx,4xx) vs Bad (5xx)
- Data is simulated with Python in EC2
- High Level Architecture is below:





Introduction

- Simulated Web Application Firewall log data using python on an EC2 instance (separate toggle in code for anomalies)
- Idea: run normal for a bit... then throw in some anomalies
- Store data in S3 bucket (though we have “live” stream of data)
- Note: sample code below are weights not probabilities



```

def anomaly_params(self):

    if self.anomaly_in == "action":
        print(f"Anomaly Generated in {self.anomaly_in}")
        self.action = random.choices(
            population=["ALLOW", "CAPTCHA", "Challenge", "Count", "BLOCK"],
            weights=[0.1, 0.1, 0.1, 0.1, 2],
            k=1
        )[0]

    if self.anomaly_in == "httpSourceName":
        print(f"Anomaly Generated in {self.anomaly_in}")
        self.httpSourceName = random.choices(
            population=["ALB", "APIGW", "APPSYNC", "CF", "-"],
            weights=[0.1, 0.1, 0.1, 0.1, 2],
            k=1
        )[0]
        self.httpSourceId = self.httpSourceName.lower()

    if self.anomaly_in == "httpStatus":
        print(f"Anomaly Generated in {self.anomaly_in}")
        self.httpStatus = random.choices(
            population=["200", "202", "307", "308", "403", "404", "502", "504"],
            weights=[0.05, 0.05, 0.05, 0.05, 2, 2, 2, 2],
            k=1
        )[0]

    if self.anomaly_in == "country":
        print(f"Anomaly Generated in {self.anomaly_in}")
        self.httpRequest_country = random.choices(
            population=["US", "UK", "IN", "XX", "YY", "ZZ", "WW"],
            weights=[0.1, 0.05, 0.05, 2, 2, 2, 2],
            k=1
        )[0]

    if self.anomaly_in == "httpMethod":
        print(f"Anomaly Generated in {self.anomaly_in}")
        self.httpRequest_httpMethod = random.choices(
            population=["GET", "HEAD", "POST", "DELETE"],
            weights=[0.15, 0.15, 0.15, 2],
            k=1
        )[0]
  
```

```

def normal_params(self):

    self.action = random.choices(
        population=["ALLOW", "CAPTCHA", "Challenge", "Count", "BLOCK"],
        weights=[0.25, 0.25, 0.25, 0.25, 0.001],
        k=1
    )[0]

    self.httpSourceName = random.choices(
        population=["ALB", "APIGW", "APPSYNC", "CF", "-"],
        weights=[0.25, 0.25, 0.25, 0.25, 0.001],
        k=1
    )[0]
    self.httpSourceId = self.httpSourceName.lower()
    self.httpStatus = random.choices(
        population=["200", "202", "307", "308", "403", "404", "502", "504"],
        weights=[0.2, 0.2, 0.2, 0.2, 0.001, 0.001, 0.001, 0.001],
        k=1
    )[0]
    self.httpRequest_country = random.choices(
        population=["US", "UK", "IN", "XX", "YY", "ZZ", "WW"],
        weights=[0.33, 0.33, 0.33, 0.001, 0.001, 0.001, 0.001],
        k=1
    )[0]
    self.httpRequest_httpMethod = random.choices(
        population=["GET", "HEAD", "POST", "DELETE"],
        weights=[0.33, 0.33, 0.33, 0.001],
        k=1
    )[0]
  
```



Sample Athena Queries

This is mostly here as a side note to investigate data manually with SQL*. The distribution of results in aggregate should validate our script

For exploration, we could use Athena to process our bucket with Logs and create a standard database table. Below is the original JSON log

```
[ec2-user@ip-172-31-93-128 ~]$ cat log.json
{
  "timestamp": 1576280412771,
  "formatVersion": 1,
  "action": "",
  "httpSourceName": "",
  "httpSourceId": "",
  "httpStatus": "",
  "httpRequest": {
    "clientIp": "1.1.1.1",
    "country": "",
    "headers": [],
    "uri": "",
    "args": "",
    "httpVersion": "HTTP/1.1",
    "httpMethod": "",
    "requestId": "null"
  }
}
```

Using Athena, we can run standard SQL queries to explore the data

The screenshot shows the AWS Athena console interface. At the top, there are tabs for Query 1, Query 2, and Query 3. The active query is Query 1, which contains the following SQL statement:

```
select action, count(action) as count from newdb.tableexplore2 where httpRequest.country = 'US' group by action
```

Below the query editor, there are buttons for 'Run again', 'Explain', 'Cancel', 'Clear', and 'Create'. The 'Run again' button is highlighted. Below the buttons, there are tabs for 'Query results' and 'Query stats'. The 'Query results' tab is selected, showing a green status bar indicating 'Completed' and 'Time in queue: 2'. Below the status bar, there is a search bar for 'Search rows'. The results are displayed in a table with 5 rows and 2 columns: 'action' and 'count'.

#	action	count
1	Challenge	96
2	ALLOW	126
3	BLOCK	30
4	Count	138
5	CAPTCHA	143

Note* there is a sql query within OpenSearch but this can be used earlier in the process

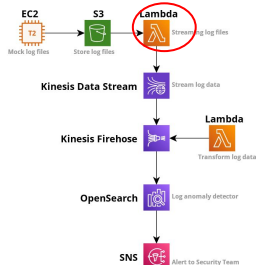


Sample Data in Table Format (Athena Query)

#	timestamp	formatversion	action	httpsourcename	httpsourceid	httpstatus	httprequest
1	1669682102.123015	1	Count	APPSYNC	appsync	202	{clientip=1.1.1.1, country=US, headers=[], uri=, args=, httpversion=HTTP/1.1, httpmethod=POST, requestid=null}
2	1669683881.757911	1	CAPTCHA	APIGW	apigw	202	{clientip=1.1.1.1, country=US, headers=[], uri=, args=, httpversion=HTTP/1.1, httpmethod=POST, requestid=null}
3	1669683731.925704	1	Challenge	ALB	alb	202	{clientip=1.1.1.1, country=UK, headers=[], uri=, args=, httpversion=HTTP/1.1, httpmethod=POST, requestid=null}
4	1669680672.38562	1	Count	APPSYNC	appsync	307	{clientip=1.1.1.1, country=UK, headers=[], uri=, args=, httpversion=HTTP/1.1, httpmethod=GET, requestid=null}
5	1669676941.352223	1	Challenge	CF	cf	200	{clientip=1.1.1.1, country=UK, headers=[], uri=, args=, httpversion=HTTP/1.1, httpmethod=GET, requestid=null}
6	1669678752.41599	1	Count	APPSYNC	appsync	308	{clientip=1.1.1.1, country=UK, headers=[], uri=, args=, httpversion=HTTP/1.1, httpmethod=GET, requestid=null}
7	1669674882.195312	1	CAPTCHA	ALB	alb	202	{clientip=1.1.1.1, country=IN, headers=[], uri=, args=, httpversion=HTTP/1.1, httpmethod=HEAD, requestid=null}
8	1669675241.970901	1	Count	CF	cf	308	{clientip=1.1.1.1, country=UK, headers=[], uri=, args=, httpversion=HTTP/1.1, httpmethod=POST, requestid=null}
9	1669684411.878921	1	CAPTCHA	APPSYNC	appsync	202	{clientip=1.1.1.1, country=UK, headers=[], uri=, args=, httpversion=HTTP/1.1, httpmethod=HEAD, requestid=null}
10	1669680372.319761	1	Count	ALB	alb	307	{clientip=1.1.1.1, country=UK, headers=[], uri=, args=, httpversion=HTTP/1.1, httpmethod=GET, requestid=null}

Lambda

- Here we use a Lambda to have S3 as a “trigger”
- This moves the logs to Kinesis Datastream (Producer)



datastream-producer

Function overview

Layers

SS

+ Add trigger

+ Add destination

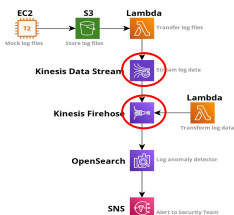
Code

```

1  import json
2  import boto3
3
4
5  def lambda_handler(event, context):
6
7      kinesis = boto3.client('kinesis', region_name='us-east-1')
8      s3 = boto3.client('s3', region_name='us-east-1')
9
10     for record in event['Records']:
11         message = record['body']
12         key = record['key']
13
14         print(record['key'])
15
16         log = s3.get_object(Bucket=key, Key=key)['Body'].read().decode('utf-8')
17         # print(log)
18         # kinesis.put_record(LogGroupName='log-stream',
19                             # data=log, partition_key=key)
20         # kinesis.put_record(LogGroupName='log-stream',
21                             # data=log, partition_key=key)
22     }
23     print(response)
24
25     # kinesis
26     response = kinesis.put_record(
27         LogGroupName='log-stream',
28         data=log,
29         partition_key=key
30     )
31     print(response)
32 
```

Kinesis Data Stream

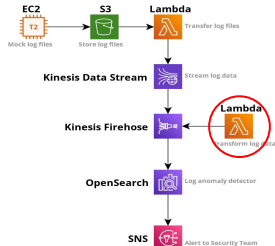
- This step mainly gets the data into a form that can be received and loaded to OpenSearch
- The Lambda in the architecture is here to transform the logs (next slide)





Lambda - Kinesis Firehose

- Data needs to be transformed by the delivery stream to Open Search
- “Normal” data is transformed as “good” and “Abnormal” is “bad” in a new field with a 1 or 0 indicator
- Mainly going from many to one. For example, you can have “US”, “UK”, “IN” as known and “aa”, “bb” as unknown.



```
1 import boto3
2 import json
3
4 def lambda_handler(event, context):
5     output = []
6     print(event)
7
8     # Loop through records in incoming event
9     for record in event['records']:
10         # Extract message
11         message = json.loads(json.loads(boto3.boto3.Session(profile_name='firehose').get_client('kinesis').get_record('firehose', {'recordId': record['id']})))
12         print('timestamp: ', message['timestamp'])
13         print('action: ', message['action'])
14         print('httpSourceName: ', message['httpSourceName'])
15         print('httpSourceId: ', message['httpSourceId'])
16         print('httpStatus: ', message['httpStatus'])
17         print('country: ', message['httpRequest']['country'])
18         print('httpMethod: ', message['httpRequest']['httpMethod'])
19
20         timestamp = message['timestamp']
21         action = message['action']
22         httpSourceName = message['httpSourceName']
23         httpSourceId = message['httpSourceId']
24         httpStatus = message['httpStatus']
25         country = message['httpRequest']['country']
26         httpMethod = message['httpRequest']['httpMethod']
27
28         # Filter action
29         if action in ['GET', 'POST', 'PUT', 'DELETE', 'PATCH', 'OPTIONS', 'HEAD', 'TRACE']:
30             # Filter httpSourceName
31             if httpSourceName in ['www.example.com', 'www.example.org', 'www.example.net']:
32                 # Filter httpStatus
```



Sample transformed data from OpenSearch query

logs (100)

Search keyword

Download

B_httpstatus	httpSourceId	common_method	K_country	HttpStatus	action	httpRequest	UNK_source	formatVersion	timestamp	UNK_country	uncommon_method	K_source	O_httpstatus	httpSourceName	K_action	B_action
0	appsync	1	1	200	Challenge	httpRequest: (1)	0	1	1970-01-20 07:47:55.122	0	0	1	1	APPSYNC	1	0
0	alb	1	1	202	ALLOW	httpRequest: (1)	0	1	1970-01-20 07:47:55.141	0	0	1	1	ALB	1	0
0	cf	1	0	307	CAPTCHA	httpRequest: (1)	0	1	1970-01-20 07:47:55.151	1	0	1	1	CF	1	0
0	apigw	1	1	307	Challenge	httpRequest: (1)	0	1	1970-01-20 07:47:55.161	0	0	1	1	APIGW	1	0
0	alb	1	1	202	Challenge	httpRequest: (1)	0	1	1970-01-20 07:47:55.171	0	0	1	1	ALB	1	0

HttpRequest = {args=, country=UK, headers=null, httpVersion=HTTP/1.1, requestId=null, clientIp=1.1.1.1, httpMethod=GET, uri=}

Country Example

logs (7)

Search keyword

Download

httpRequest.country	K_country_count	UNK_country_count
IN	16713	0
UK	16979	0
US	17017	0
WW	0	68
XX	0	54
YY	0	65
ZZ	0	77

OpenSearch / Dashboards

•Create Domain:

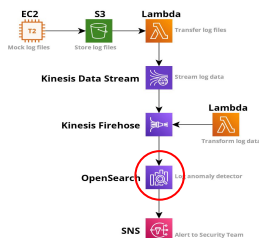
Amazon OpenSearch Service > Domains

Domains (1) [Info](#)

<input type="checkbox"/>	Name	Engine	Version	Status	Endpoint	Cluster health	Searchable documents	Total free space	Minimum free sp...
<input type="checkbox"/>	http-logs	OpenSearch	1.3	Active	Internet	Yellow	42,778	7.71 GiB	7.71

•Data is fed to Opensearch's Dashboard a visual dashboard creator

•Create Anomaly Detector



Detectors (5)

<input type="checkbox"/>	Detector ↑	Indices	Detector state	Anomalies last 24 hours	Last anomaly occurrence	Last enabled time
<input type="checkbox"/>	action-anomaly-detector	logs	Running	3	11/28/2022 5:59 PM	11/28/2022 5:45 PM
<input type="checkbox"/>	country-anomaly-detector	logs	Running	4	11/28/2022 6:06 PM	11/28/2022 5:47 PM
<input type="checkbox"/>	httpstatus-anomaly-detector	logs	Running	3	11/28/2022 6:00 PM	11/28/2022 5:46 PM
<input type="checkbox"/>	method-anomaly-detector	logs	Running	3	11/28/2022 6:01 PM	11/28/2022 5:47 PM
<input type="checkbox"/>	source-anomaly-detector	logs	Running	3	11/28/2022 5:59 PM	11/28/2022 5:45 PM

Rows per page: 20

Dashboard

•Sample Dashboard of Detectors in action

Anomalies by index and detector

The inner circle shows anomaly distribution by index. The outer circle shows distribution by detector.

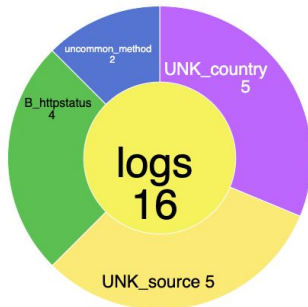
Last 24 hours

Indices with anomalies

1

Detectors with anomalies

4



Anomalies by index and detector

The inner circle shows anomaly distribution by index. The outer circle shows distribution by detector.

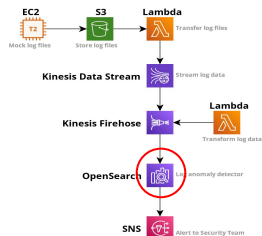
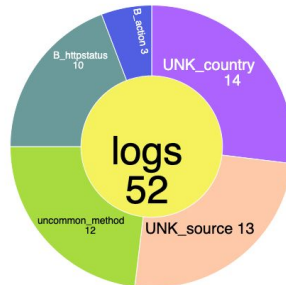
Last 7 days

Indices with anomalies

1

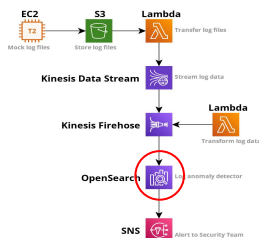
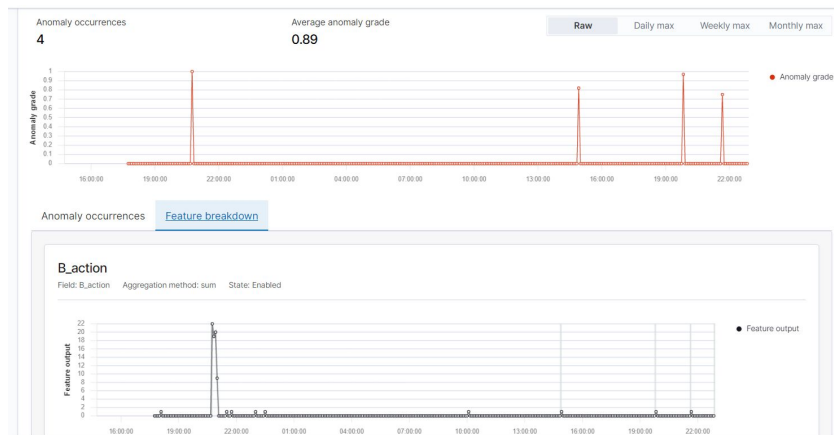
Detectors with anomalies

5



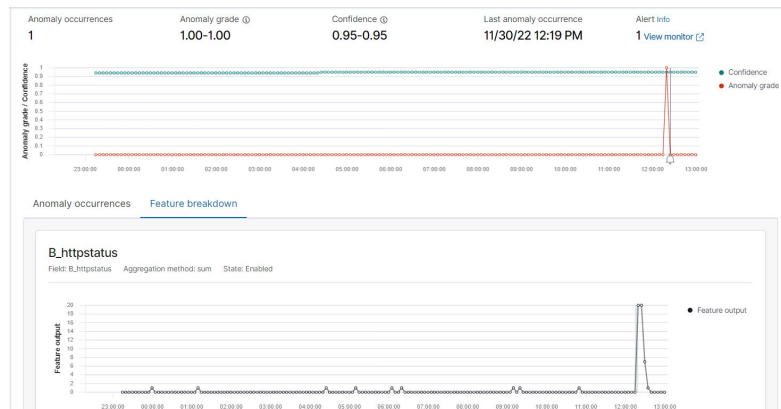
Dashboard

- Historical Analysis for Block action anomaly

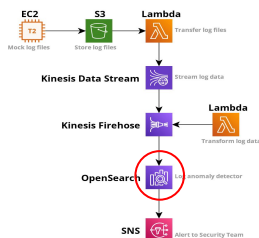


Dashboard

Live HttpStatus anomaly trends

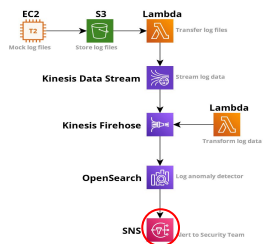



Live unknown country anomaly trends



SNS - Alert!

- SNS alerts send on anomalies





euphemis2437@gmail.com via amazonses.com

to me ▾ ▶ 0:05

Monitor UNK_source-Monitor just entered alert status. Please investigate the issue.

- Trigger: UNK_source-trigger
- Severity: 1
- Period start: 2022-11-30T18:11:57.150Z
- Period end: 2022-11-30T18:21:57.150Z


euphemis2437@gmail.com via amazonses.com

to me ▾ ▶ 0:05

Monitor UNK_source-Monitor just entered alert status. Please investigate the issue.

- Trigger: UNK_source-trigger
- Severity: 1
- Period start: 2022-11-30T18:21:57.150Z
- Period end: 2022-11-30T18:31:57.150Z

↩ Reply

➡ Forward



NYU

TANDON SCHOOL
OF ENGINEERING

PART 02

Takeaways



Takeaways

Typical Use Case for such implementation: detecting anomalies, alerting security team and presenting dashboard to executive showing KPI on:

- WAF (Web based Application firewall)
- System logs
- Cloudwatch logs
- Splunk logs
- Cloudtrail logs

End