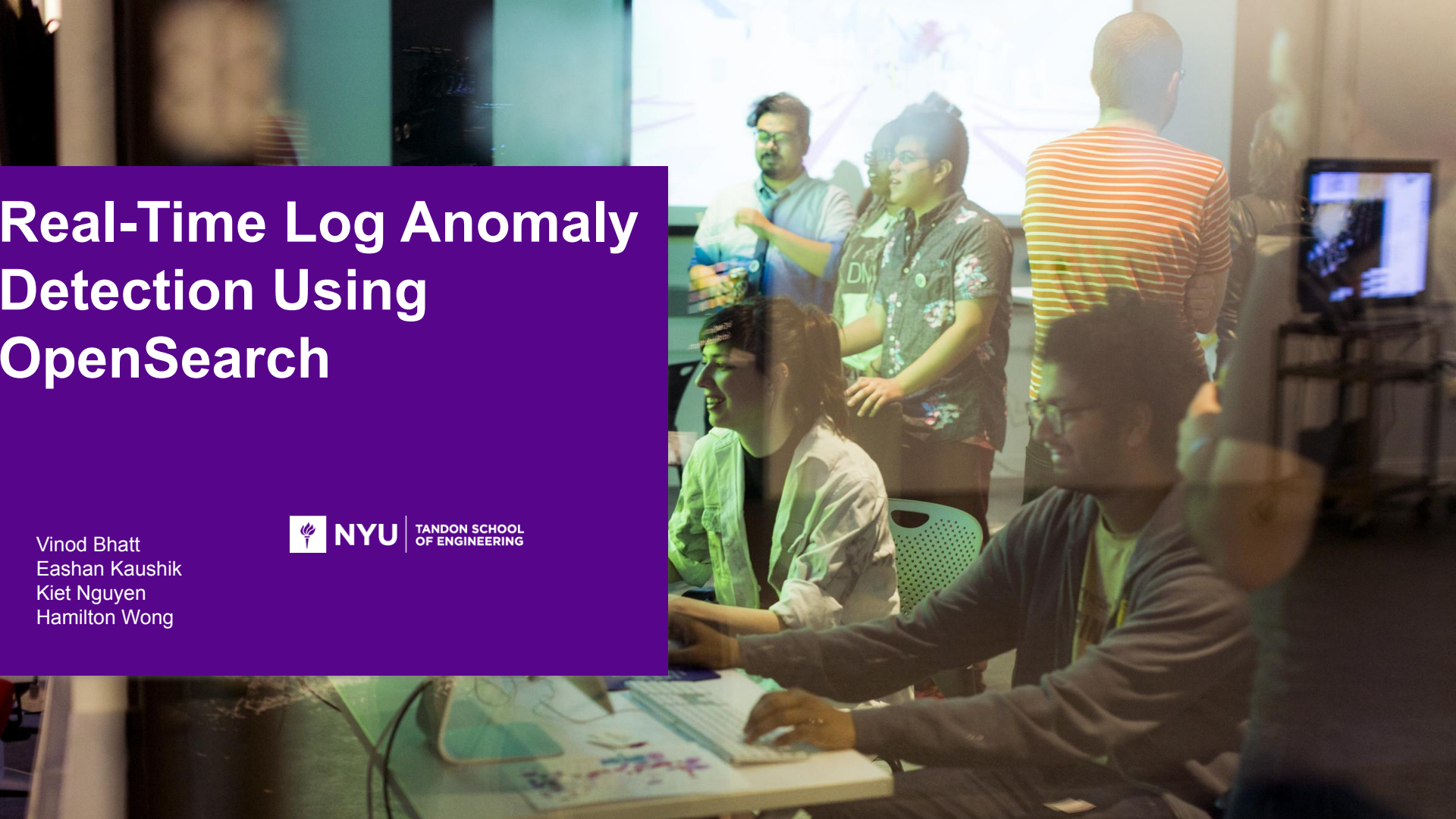
Real-Time Log Anomaly Detection Using OpenSearch

Vinod Bhatt
Easha Kaushik
Kiet Nguyen
Hamilton Wong

NYU TANDON SCHOOL OF ENGINEERING

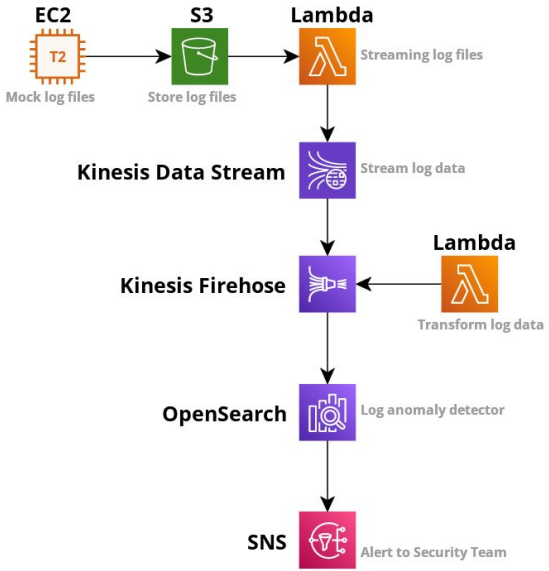


PART 01

Setup

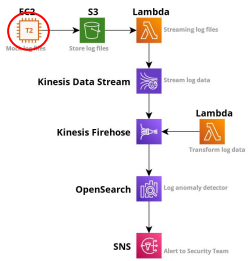
Introduction and Architecture

- 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]
```

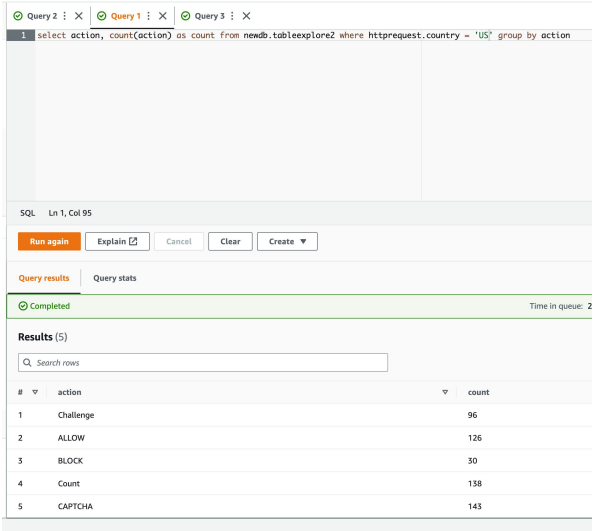
Sample Athena Queries

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



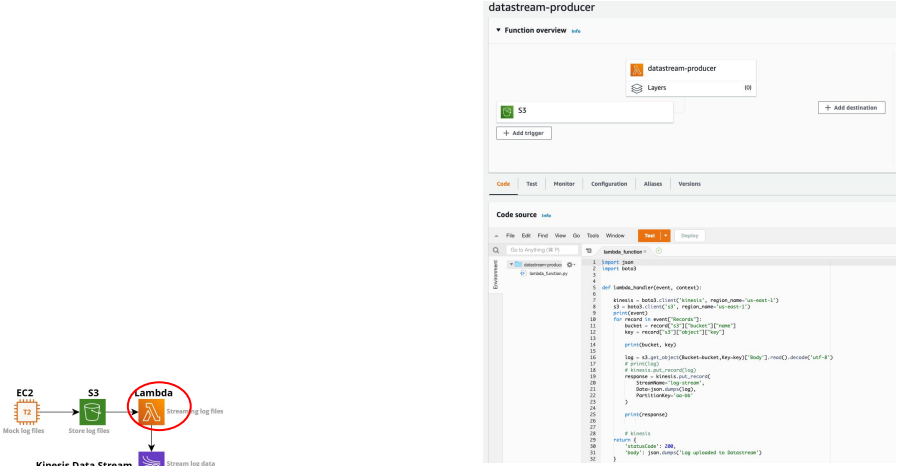
Note* there is a sql query within OpenSearch but this can be used earlier in the process

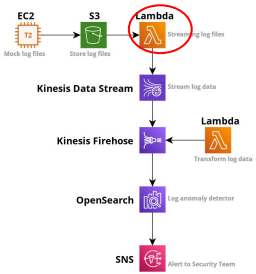
Sample Data in Table Format (Athena Query)

#	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

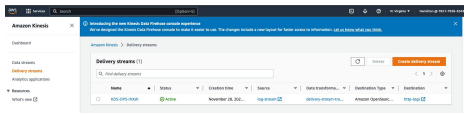
- **Here we use a Lambda to have S3 as a “trigger”**
- **This moves the logs to Kinesis Datastream (Producer)**

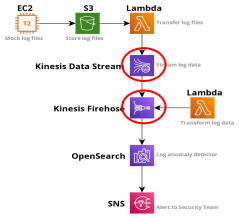




Kinesis Data Stream

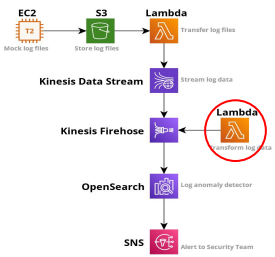
- **This step mainly gets the data into a form that can be received and loaded to OpenSearch**
- **The Lambda in the architecture is here to transform the logs (next slide)**





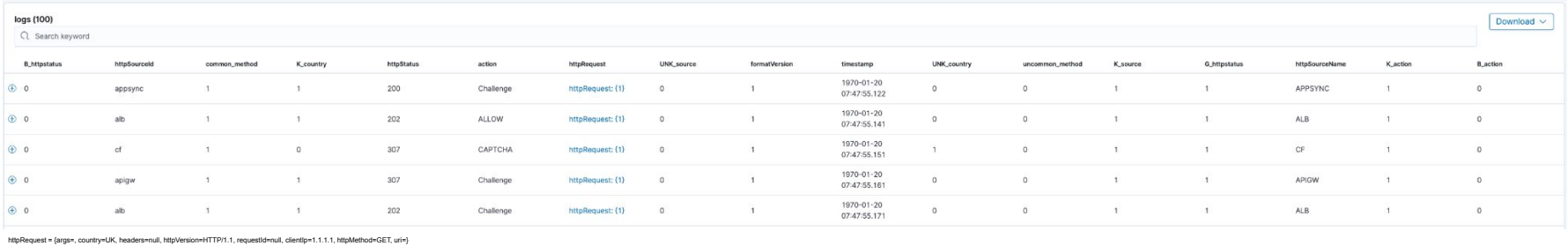
Lambda - Kinesis Firehose

- **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.**



```
import base64
import json

def lambda_handler(event,context):
    output = []
    print(event)

    # Loop through records in incoming event
    for record in event["records"]:
        # extract message
        message = json.loads(json.loads(base64.b64decode(record["data"])))
        print("timestamp: ", message["timestamp"])
        print("action: ", message["action"])
        print("HttpSourceName: ", message["HttpSourceName"])
        print("HttpSourceId: ", message["HttpSourceId"])
        print("HttpStatus: ", message["HttpStatus"])
        print("country: ", message["HttpRequest"]["country"])
        print("HttpMethod: ", message["HttpRequest"]["HttpMethod"])

        timestamp = message["timestamp"]
        action = message["action"]
        HttpSourceName = message["HttpSourceName"]
        HttpSourceId = message["HttpSourceId"]
        HttpStatus = message["HttpStatus"]
        country = message["HttpRequest"]["country"]
        HttpMethod = message["HttpRequest"]["HttpMethod"]

        K_action, B_action = filter_action(action) # ALLOW, CAPTCHA, Challenge, Count, BLOCK
        K_source, UNK_source = filter_HttpSourceName(HttpSourceName) # ALB, APIGW, APPSYNC, CF, UNK
        G_httpstatus, B_httpstatus = filter_HttpStatus(HttpStatus) # two, three, four, five
        K_country, UNK_country = filter_country(country) # US, UK, IN, UNK_country
        common_method, uncommon_method = filter_HttpMethod(HttpMethod) # common_method, uncommon_method
```

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	G_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)

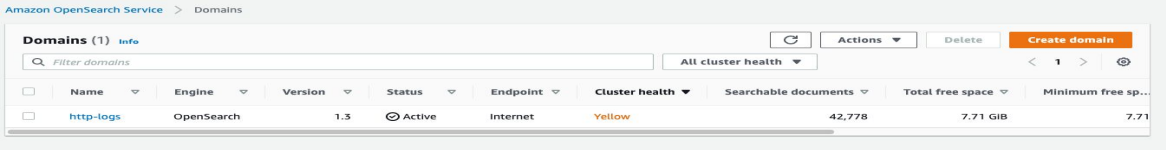
Search keyword

Download

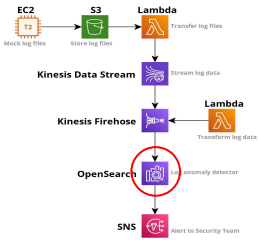
httpRequest.country	K_country_count	UNK_country_count
IN	16713	0
UK	16979	0
US	17017	0
WW	0	68
XX	0	54
YY	0	65
ZZ	0	77

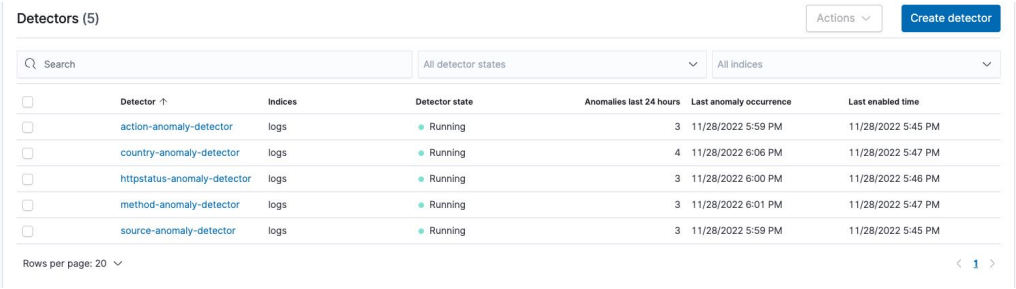
OpenSearch / Dashboards

- **Create Domain:**



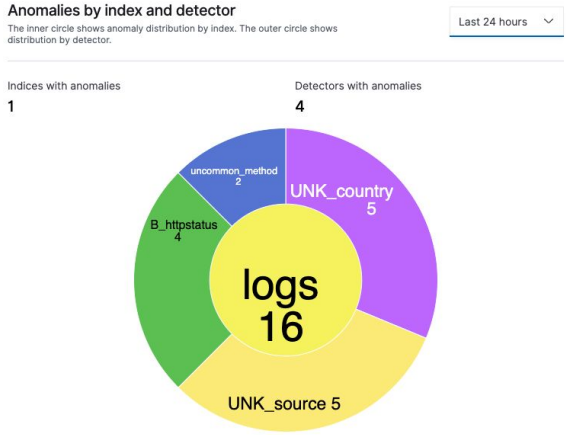
- **Data is fed to Opensearch's Dashboard a visual dashboard creator**
- **Create Anomaly Detector**

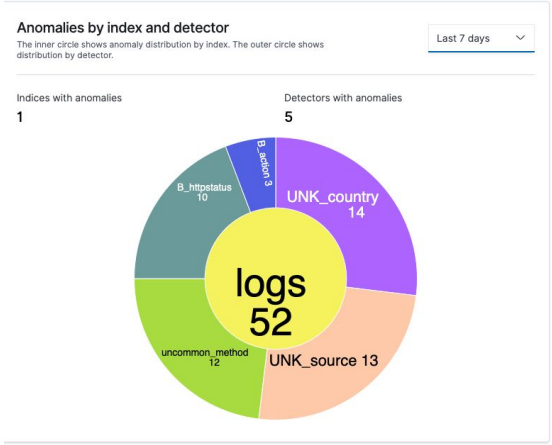


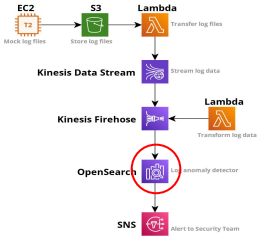


Dashboard

- **Sample Dashboard of Detectors in action**

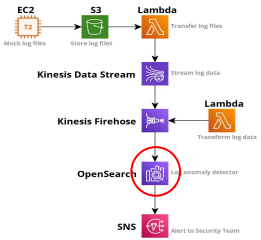


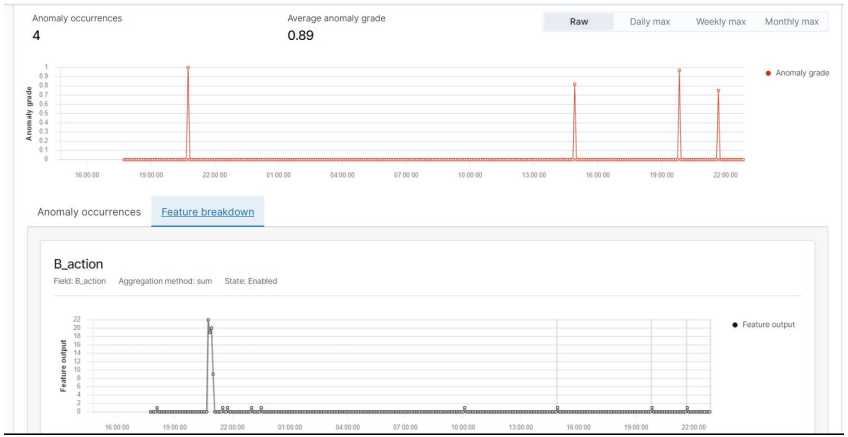




Dashboard

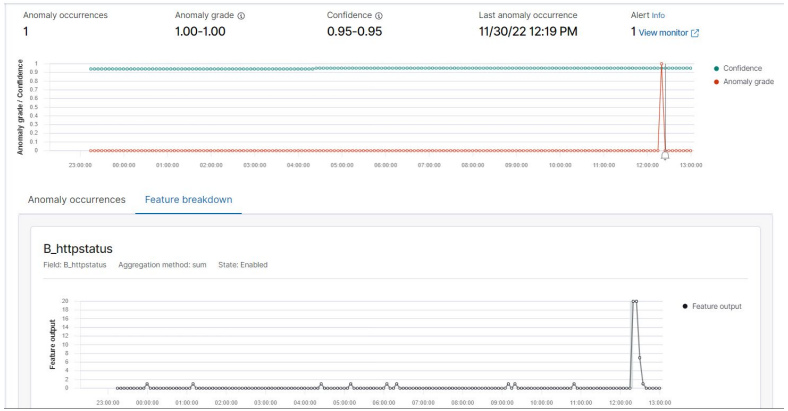
- **Historical Analysis for Block action anomaly**



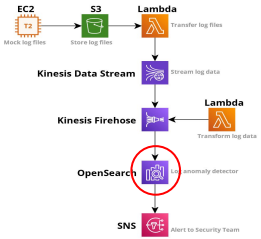


Dashboard

Live HttpStatus anomaly trends

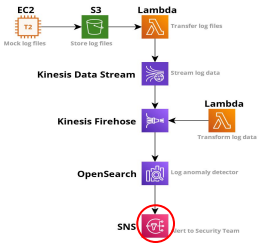


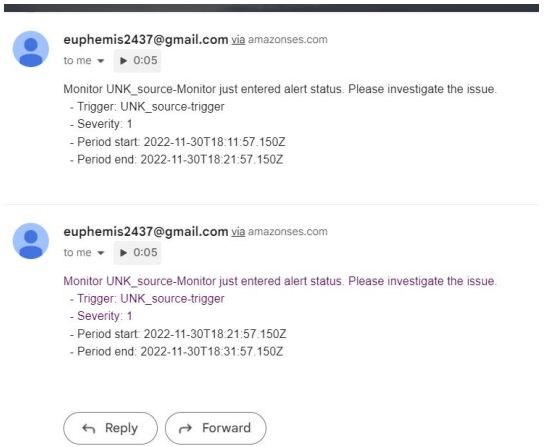
Live unknown country anomaly trends



SNS - Alert!

- **SNS alerts send on anomalies**





PART 02

Takeaways

Takeaways

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

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

End