Real-time Log Analytics and Anomaly Detection using ELK Stack and Python

Vamsi Nirogi

Project Overview

This project implements a real-time ETL (Extract, Transform, Load) pipeline designed to collect, process, and analyse SSH system logs to detect anomalous login activities. The pipeline leverages Elasticsearch, Logstash, and Kibana (ELK stack) for efficient log management and visualization, coupled with a Python-based anomaly detection module powered by Isolation Forest.

Objectives

- Set up a robust real-time data pipeline using Elasticsearch, Logstash, and Kibana.
- Perform log parsing and indexing to enable quick querying and visualization.
- Apply machine learning (Isolation Forest) for automatic anomaly detection on login patterns.
- Visualize anomalies clearly, allowing for immediate identification and investigation.

Technologies Used

- Elasticsearch (7.17): For storing, indexing, and querying structured log data.
- Logstash: For real-time log ingestion, parsing, and transformation.
- Kibana: For intuitive and interactive data visualization.
- Python: For advanced analysis and anomaly detection using scikit-learn.
- Isolation Forest: A robust anomaly detection algorithm suitable for log data.

Methodology

Data Collection and Processing (ETL)

- Logstash collects and parses raw SSH logs from system files.
- First created a folder “elk-etl-ai” on desktop using mkdir command in cmd prompt
- Then created 3 different files “docker-compose.yml”, “sample.log”, “logstash.conf”
- Logs are structured into fields (timestamp, IP, username, status) and sent to Elasticsearch in real-time.
- Elasticsearch indexes structured logs for efficient searching and analysis.

Docker-compose:

- To make the ETL environment I used docker as the interface.
- Then with the help of Docker CLI tool “docker-compose” installed all the ELK tools on the system.

```
version: '3.8'
services:
  elasticsearch:
    image: docker.elastic.co/elasticsearch/elasticsearch:8.13.2
    environment:
      - discovery.type=single-node
      - xpack.security.enabled=false
    ports:
      - "9200:9200"
    networks:
      - elk
  logstash:
    image: docker.elastic.co/logstash/logstash:8.13.2
    volumes:
      - ./logstash.conf:/usr/share/logstash/pipeline/logstash.conf
      - ./sample.log:/usr/share/logstash/sample.log
```

```
    depends_on:
      - elasticsearch
    networks:
      - elk
  kibana:
    image: docker.elastic.co/kibana/kibana:8.13.2
    ports:
      - "5601:5601"
    environment:
      - ELASTICSEARCH_HOSTS=http://elasticsearch:9200
    depends_on:
      - elasticsearch
    networks:
      - elk
networks:
  elk:
    driver: bridge
```

-This is the yaml code used to install all the ELK on the system

---

Used SSH logs (Sample log details)

```
Jun 19 11:00:00 server sshd[1000]: Failed password for root from 192.168.1.10 port 22 ssh2
Jun 19 11:02:00 server sshd[1001]: Accepted password for user from 192.168.1.11 port 22 ssh2
Jun 19 11:05:00 server sshd[1002]: Failed password for admin from 192.168.1.12 port 22 ssh2
Jun 19 11:10:00 server sshd[1003]: Failed password for user from 192.168.1.13 port 22 ssh2
Jun 19 11:15:00 server sshd[1004]: Accepted password for user from 192.168.1.14 port 22 ssh2
Jun 19 11:20:00 server sshd[1005]: Failed password for guest from 192.168.1.15 port 22 ssh2
Jun 19 11:25:00 server sshd[1006]: Accepted password for root from 192.168.1.16 port 22 ssh2
Jun 19 11:30:00 server sshd[1007]: Failed password for user from 192.168.1.17 port 22 ssh2
Jun 19 11:35:00 server sshd[1008]: Failed password for admin from 192.168.1.18 port 22 ssh2
Jun 19 11:40:00 server sshd[1009]: Accepted password for guest from 192.168.1.19 port 22 ssh2
Jun 19 11:45:00 server sshd[1010]: Failed password for root from 192.168.1.20 port 22 ssh2
Jun 19 11:50:00 server sshd[1011]: Failed password for admin from 192.168.1.21 port 22 ssh2
Jun 19 11:55:00 server sshd[1012]: Accepted password for user from 192.168.1.22 port 22 ssh2
Jun 19 12:00:00 server sshd[1013]: Failed password for user from 192.168.1.23 port 22 ssh2
Jun 19 12:05:00 server sshd[1014]: Accepted password for root from 192.168.1.24 port 22 ssh2
Jun 19 12:10:00 server sshd[1015]: Failed password for guest from 192.168.1.25 port 22 ssh2
Jun 19 12:15:00 server sshd[1016]: Accepted password for admin from 192.168.1.26 port 22 ssh2
Jun 19 12:20:00 server sshd[1017]: Failed password for user from 192.168.1.27 port 22 ssh2
Jun 19 12:25:00 server sshd[1018]: Failed password for root from 192.168.1.28 port 22 ssh2
Jun 19 12:30:00 server sshd[1019]: Accepted password for guest from 192.168.1.29 port 22 ssh2
Jun 19 12:35:00 server sshd[1020]: Failed password for user from 192.168.1.30 port 22 ssh2
Jun 19 12:40:00 server sshd[1021]: Failed password for admin from 192.168.1.31 port 22 ssh2
Jun 19 12:45:00 server sshd[1022]: Accepted password for user from 192.168.1.32 port 22 ssh2
Jun 19 12:50:00 server sshd[1023]: Failed password for root from 192.168.1.33 port 22 ssh2
Jun 19 12:55:00 server sshd[1024]: Accepted password for admin from 192.168.1.34 port 22 ssh2
Jun 19 13:00:00 server sshd[1025]: Failed password for guest from 192.168.1.35 port 22 ssh2
Jun 19 13:05:00 server sshd[1026]: Failed password for user from 192.168.1.36 port 22 ssh2
Jun 19 13:10:00 server sshd[1027]: Accepted password for root from 192.168.1.37 port 22 ssh2
Jun 19 13:15:00 server sshd[1028]: Failed password for admin from 192.168.1.38 port 22 ssh2
Jun 19 13:20:00 server sshd[1029]: Accepted password for guest from 192.168.1.39 port 22 ssh2
```

Logstash.conf:

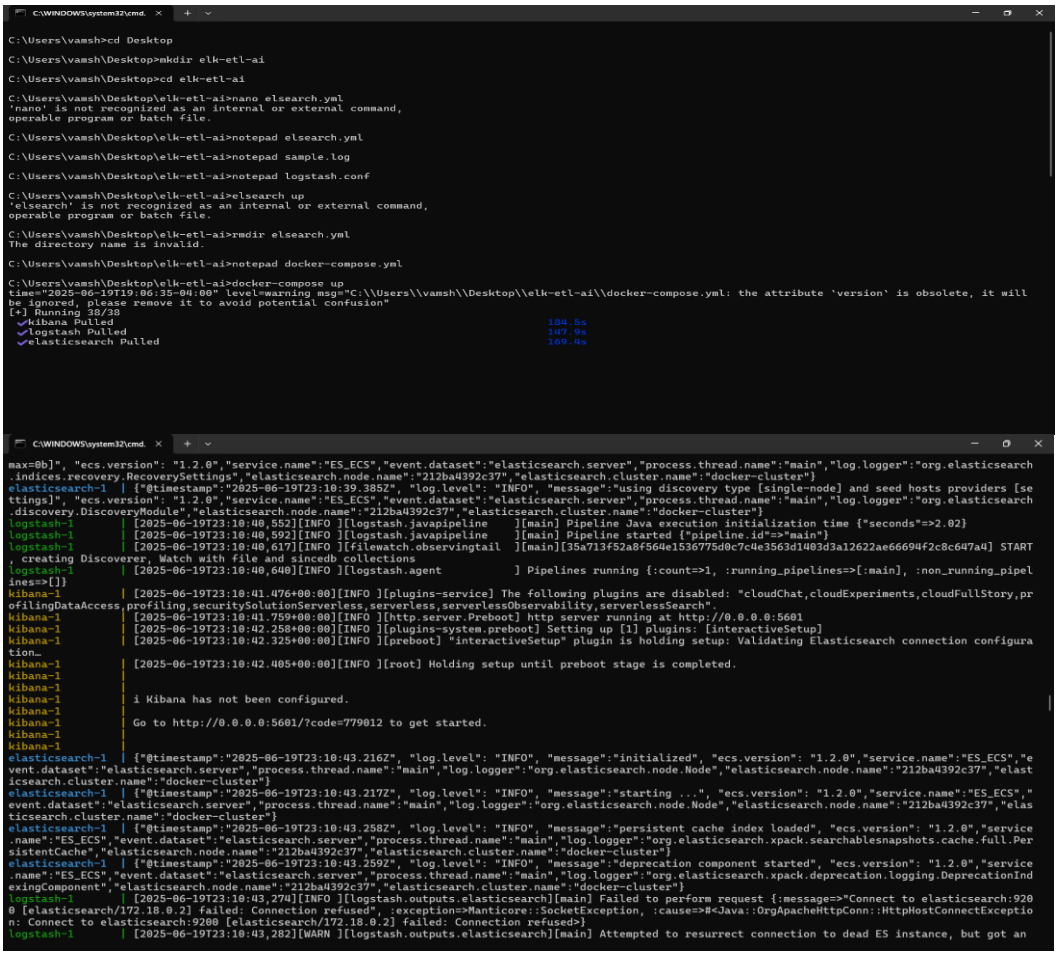
- This file tells Logstash how to read and parse the logs

```
input {
  file {
    path => "/usr/share/logstash/sample.log"
    start_position => "beginning"
```

```
    sincedb_path => "/dev/null"
  }
}
filter {
  grok {
    match => {
      "message"          =>          "%{SYSLOGTIMESTAMP:timestamp}          %{HOSTNAME:host}
%{WORD:process}\[%{NUMBER:pid}\]: %{GREEDYDATA:log_message}"
    }
  }
  date {
    match => ["timestamp", "MMM dd HH:mm:ss"]
    target => "@timestamp"
  }
}
output {
  elasticsearch {
    hosts => ["http://elasticsearch:9200"]
    index => "logs-sample"
  }
  stdout { codec => rubydebug }
}
```

Starting ELK Stack:

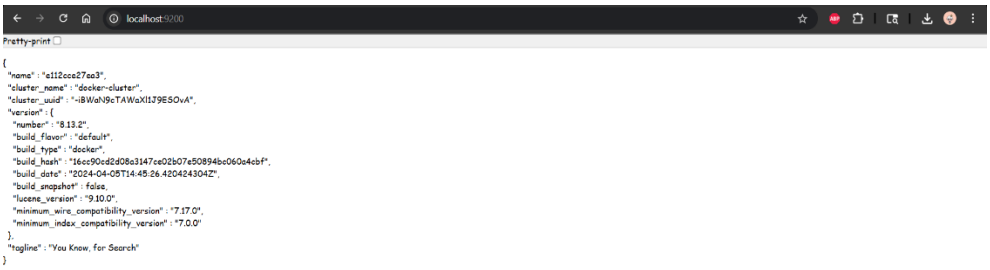
- In terminal, inside the “elk-etl-ai” folder, gave this command to run the ELK stack – “docker-compose up”
- Resulting it to create all the stack.



Checking if Elasticsearch working or not:

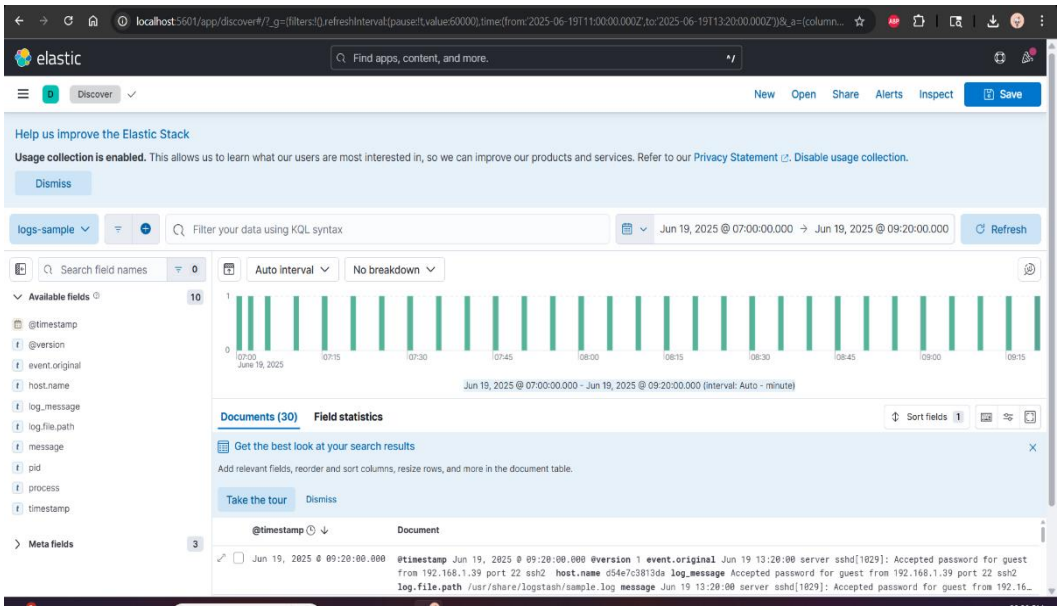
- “Docker-compose up” command will start Elasticsearch at port : 9200, so after opening browser via

http://localhost:9200 , if we get result as shown below then our elasticsearch is working.



Data Visualization

- **Kibana** provides an interactive dashboard that visualizes login events over time, categorizing events by success or failure.
- Custom dashboards created in Kibana enable rapid monitoring and drill-down investigation of anomalies.
- Used port http://localhost:5601 (acc. To docker-compose up cmd)



Anomaly Detection with Python

- Python scripts query Elasticsearch, extract relevant log data, and perform feature engineering.
- The **Isolation Forest** model identifies outliers or anomalies by assessing login frequency, time patterns, and success/failure rates.
- Visualization of anomalies is achieved via matplotlib, clearly distinguishing anomalous events from normal patterns.

Python code used to find the anomality in our ELK stack

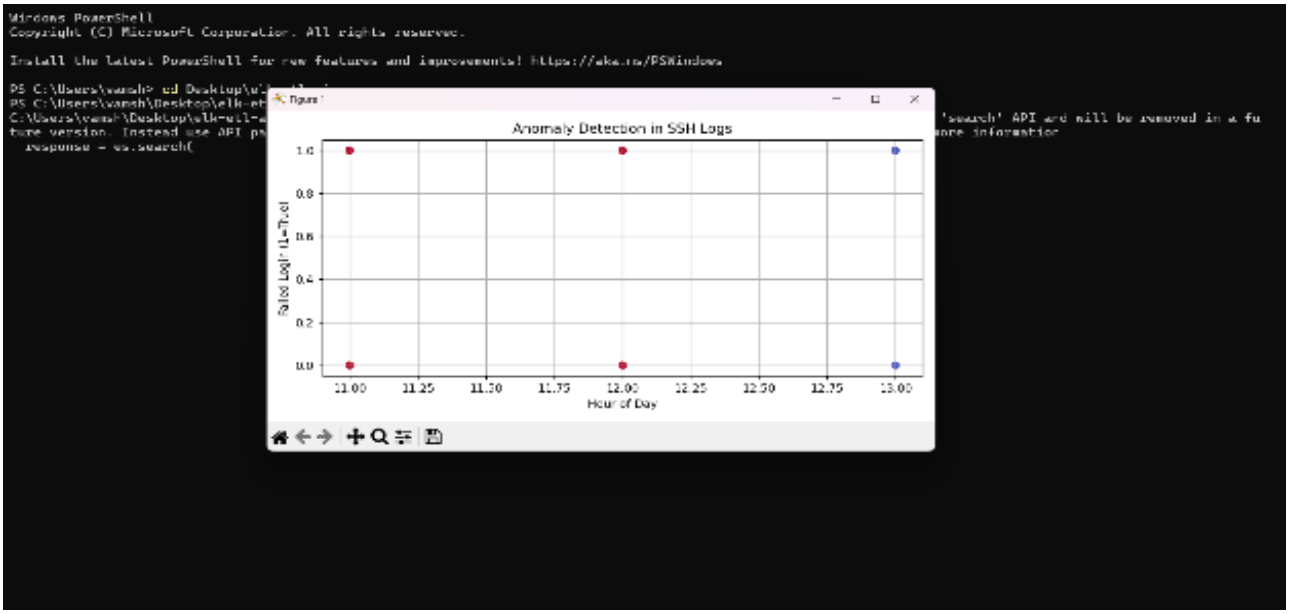
```
from elasticsearch import Elasticsearch
import pandas as pd
from sklearn.ensemble import IsolationForest
import matplotlib.pyplot as plt
```

```
es = Elasticsearch("http://localhost:9200")# Connecting to Elasticsearch


#logs
response = es.search(
    index="logs-sample",
    body={
        "size": 1000,
        "query": {
            "match_all": {}
        }
    }
)


data = [hit["_source"] for hit in response["hits"]["hits"]]# Converting to DataFrame
df = pd.DataFrame(data)


df["hour"] = pd.to_datetime(df["@timestamp"]).dt.hour
df["fail_flag"] = df["log_message"].str.contains("Failed", case=False).astype(int)

# Anomaly detection
model = IsolationForest(contamination=0.2, random_state=42)
df["anomaly"] = model.fit_predict(df[["hour", "fail_flag"]])

# Visualization
plt.figure(figsize=(8, 4))
plt.scatter(df["hour"], df["fail_flag"], c=df["anomaly"], cmap="coolwarm", marker="o")
plt.title("Anomaly Detection in SSH Logs")
plt.xlabel("Hour of Day")
plt.ylabel("Failed Login (1=True)")
plt.grid(True)
plt.tight_layout()
plt.show()
```

Results

- After running the above python code, we will get something like below image which is a graph.
- This graph identifies anomalous login activities, potentially indicative of security threats or suspicious behaviour.
- X - axis (hour of day) which is Extracted from each log entry's timestamp.
- It ranges from around 11.0 to 13.0, meaning you had logs between **11:00 AM to 1:00 PM**.
- Y – axis (Failed login (1 = true)) which is a binary indicator where:
  1: The log line contains "Failed password"
  0: The log line contains "Accepted password"
- **Red dot**: **Anomaly**, flagged by the IsolationForest AI model
- **Blue dot**: **Normal**

- The IsolationForest model looks for patterns in two dimensions, It marks a point as an anomaly if:
  - It stands out in terms of time (e.g., failed at an odd hour)
  - There's an unusual number of failures/successes in a narrow time window
  - It's statistically isolated compared to the cluster



In this current result graph:

- The plot shows clear **clusters of normal behavior** at the edges of your observed time window (around 11:00 AM and 1:00 PM).
- **Red points** highlight attempts at unexpected times or unusual patterns of failure/success in the given timeframe, indicating potential security concerns or suspicious activities.

Future Enhancements

- Implement real-time alerting mechanisms (email, Slack) upon detecting critical anomalies.
- Extend the solution to include additional log types (e.g., application logs, firewall logs) for comprehensive security monitoring.

Conclusion

This project effectively demonstrates a practical implementation of real-time log analytics and anomaly detection using the ELK stack and Python. The solution provides significant value in cybersecurity contexts, enhancing real-time monitoring capabilities and proactive threat detection.

References

[1] Elastic, "Tutorial: Getting started with anomaly detection," ***Elastic Docs*****, 2025. Available: https://www.elastic.co/docs/explore-analyze/machine-learning/anomaly-detection**

[2] O. Mykhaylova, A. Shtypka, and T. Fedynyshyn, "An Isolation Forest-based approach for brute force attack detection," in ***Proc. 1st Int. Workshop on Bioinformatics and Applied Information Technologies (BAIT'2024)*****, Zboriv, Ukraine, Oct. 2024, pp. 1–12.**

[3] J. Nyyssölä and M. Mäntylä, "Speed and performance of parserless and unsupervised anomaly detection methods on software logs," ***ArXiv*****, Dec. 2023. Available: https://arxiv.org/abs/2312.01934**