Distributed Multi-Cloud Application Observability for Advanced Logging Using KEK Stack and Docker

Kunal Gurnani
22142363
Mr. Sudarshan Deshmukh
MSc Cloud Computing

January 28, 2025

Configuration Manual

Introduction

This configuration manual provides a comprehensive guide to replicate the log analytics system project. It details all the practical steps for setup, configuration, and execution across all phases of the work.

Prerequisites

Hardware Requirements

- **EC2 instance:** At least 4 vCPUs and 8 GB RAM.
- **Local system or virtual machine:** Docker and Docker Compose installed.

Software Requirements

- Apache Kafka (v3.x)
- Elasticsearch (v8.x)
- Kibana (v8.x)
- Python 3.9 or higher
- AWS CLI

AWS Services

- AWS EC2 instance
- CloudWatch Logs

Phase 1: Log Generation

Steps

1. Create a Log Generation Script

Write a Python script to generate log entries with various levels (`INFO`, `WARNING`, `ERROR`). Save the script as `generate_logs.py`:

```
import random
import time
import logging

# Create a logger
logger = logging.getLogger('FakeLogger')
logger.setLevel(logging.INFO)

# Create file handler to write logs to a file
file_handler = logging.FileHandler('/var/log/fake_app.log')  # Define the log file lo
file_handler.setLevel(logging.INFO)

# Define a log format
formatter = logging.Formatter('%(asctime)s - %(levelname)s - %(message)s')
file_handler.setFormatter(formatter)

# Add the file handler to the logger
logger.addHandler(file_handler)

# Generate fake log messages
log_levels = [logging.INFO, logging.WARNING, logging.ERROR]

messages = [
    "User logged in",
    "User logged out",
    "File not found",
    "Connection timeout",
    "Invalid user input",
    "Database error",
    "Service unavailable",
]

def generate_log():
    while True:
        # Pick a random log level and message
        level = random.choice(log_levels)
        message = random.choice(messages)
        logger.log(level, message)

        # Sleep for a random interval between 1 and 5 seconds
```

```
        time.sleep(random.randint(1, 5))

if __name__ == "__main__":
    generate_log()
```

2. Run the Script

Run the script in the background using the following command:

```
nohup python3 generate_logs.py &
```

3. Verify Log File Creation

Check the logs being generated in real-time using:

```
tail -f /var/log/fake_app.log
```

4. Stream Logs to Kafka

Write a script to stream the logs to Kafka. Save it as `log_to_kafka.py`:

```
from kafka import KafkaProducer
import time

# Kafka producer configuration
producer = KafkaProducer(bootstrap_servers='65.1.69.224:9092')  # Update to Kafka ser

# Log file to read
log_file_path = "/var/log/fake_app.log"  # Update this path if necessary

def stream_logs_to_kafka():
    with open(log_file_path, 'r') as log_file:
        # Move to the end of the file
        log_file.seek(0, 2)
        while True:
            line = log_file.readline()
            if not line:
                time.sleep(1)  # Wait and try again if no new line is found
                continue
            # Send log line to Kafka topic
            producer.send('test-topic', value=line.encode('utf-8'))
            print(f"Sent log: {line.strip()}")

if __name__ == "__main__":
    try:
        print("Starting log streaming...")
        stream_logs_to_kafka()
    except KeyboardInterrupt:
        print("Stopping log streaming...")
        producer.close()
```

Phase 2: Kafka Setup

Steps

1. Install Kafka and Zookeeper on EC2

Download and extract the required Kafka version on your EC2 instance using the following commands:

```
wget https://downloads.apache.org/kafka/3.x/kafka_2.13-3.x.tgz
tar -xzf kafka_2.13-3.x.tgz
cd kafka_2.13-3.x
```

2. Start Zookeeper and Kafka

Start Zookeeper and Kafka services using the following commands:

```
bin/zookeeper-server-start.sh config/zookeeper.properties &
bin/kafka-server-start.sh config/server.properties &
```

3. Create a Kafka Topic

Create a Kafka topic named `log-topic` with the following command:

```
bin/kafka-topics.sh --create --topic log-topic --bootstrap-server localhost:9092
```

4. Publish Logs to Kafka

Install the `kafka-python` library for Python to send logs to Kafka:

```
pip install kafka-python
```

Modify the log generator script to send logs to Kafka. Use the following code:

```
from kafka import KafkaProducer

# Kafka producer configuration
producer = KafkaProducer(bootstrap_servers='localhost:9092')

# Example log entry
log_entry = "Sample log entry"

# Send the log to the Kafka topic
producer.send('log-topic', log_entry.encode('utf-8'))
```

5. Kafka to Elasticsearch Integration Script

To consume logs from Kafka and ingest them into Elasticsearch, create a Python script named `kafka_to_elasticsearch.py`:

```
from kafka import KafkaConsumer
from elasticsearch import Elasticsearch
import json
import datetime

# Kafka and Elasticsearch configurations
KAFKA_BROKER = '65.1.69.224:9092'  # Update to your Kafka broker IP
TOPIC_NAME = 'test-topic'
ELASTICSEARCH_HOST = 'http://65.1.69.224:9200'
ELASTIC_USER = 'elastic'
ELASTIC_PASSWORD = 'elastic'
INDEX_NAME = 'test-index'

# Connect to Elasticsearch
es = Elasticsearch(
    [ELASTICSEARCH_HOST],
    http_auth=(ELASTIC_USER, ELASTIC_PASSWORD)
)

# Connect to Kafka
consumer = KafkaConsumer(
    TOPIC_NAME,
    bootstrap_servers=KAFKA_BROKER,
    value_deserializer=lambda m: json.loads(m.decode('utf-8')),
    auto_offset_reset='earliest',  # Start reading from the beginning
    enable_auto_commit=True
)

# Consume and ingest logs into Elasticsearch
print("Starting to consume messages from Kafka and ingest into Elasticsearch...")
for message in consumer:
    log_data = message.value
    log_entry = {
        "message": log_data.get("message", ""),
        "timestamp": log_data.get("timestamp", datetime.datetime.utcnow().isoformat()
    }

    # Index the log entry into Elasticsearch
    response = es.index(index=INDEX_NAME, body=log_entry)
    print(f"Indexed log entry: {response['_id']}")
```

6. Run the Script

Execute the script to consume logs from Kafka and ingest them into Elasticsearch:

```
python3 kafka_to_elasticsearch.py
```

Phase 3: Elasticsearch and Kibana Setup

Steps

1. Install Elasticsearch and Kibana

To install Elasticsearch and Kibana, pull the required Docker images using the following commands:

```
docker pull elasticsearch:8.x
docker pull kibana:8.x
```

2. Run Elasticsearch and Kibana

Create a `docker-compose.yml` file to configure and run Elasticsearch and Kibana. Use the following content:

```
version: '3'
services:
  elasticsearch:
    image: elasticsearch:8.x
    environment:
      - discovery.type=single-node
    ports:
      - "9200:9200"

  kibana:
    image: kibana:8.x
    ports:
      - "5601:5601"
```

Start the services with Docker Compose:

```
docker-compose up -d
```

3. Access Kibana

Open Kibana in a web browser by navigating to:

```
http://<ec2-ip>:5601
```

Follow the on-screen instructions to configure Kibana.

4. Verify Elasticsearch Setup

Use the following command to verify that Elasticsearch is working correctly:

```
curl -u elastic:elastic -X GET "http://65.1.69.224:9200/test-index/_search?pretty"
```

The output should display indexed logs from the test data. A sample response is as follows:


```
{
  "took" : 755,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 195,
      "relation" : "eq"
    },
    "max_score" : 1.0,
    "hits" : [
      {
        "_index" : "test-index",
        "_type" : "_doc",
        "_id" : "5OrbJpQBJ5Jjun4wVIWA",
        "_score" : 1.0,
        "_source" : {
          "message" : "2025-01-02 05:55:49,806 - WARNING - Connection timeout\n",
          "timestamp" : "2025-01-02T11:51:35.803506"
        }
      },
      {
        "_index" : "test-index",
        "_type" : "_doc",
        "_id" : "5erbJpQBJ5Jjun4wVIWR",
        "_score" : 1.0,
        "_source" : {
          "message" : "2025-01-02 05:55:52,674 - ERROR - Service unavailable\n",
          "timestamp" : "2025-01-02T11:51:35.821593"
        }
      },
      ...
    ]
  }
}
```

5. Install Kibana Manually (Optional)

If Docker is not used, Kibana can also be installed manually using the following commands:

```
curl -O https://artifacts.elastic.co/downloads/kibana/kibana-7.17.0-x86_64.rpm
sudo rpm -ivh kibana-7.17.0-x86_64.rpm
```

Enable and start the Kibana service:

```
sudo nano /etc/kibana/kibana.yml  # Edit configuration file
sudo systemctl start kibana
sudo systemctl enable kibana
```

The Kibana service will be accessible at:

```
http://<ec2-ip>:5601
```

Phase 4: Kafka-to-Elasticsearch Integration

Steps

1. Create a Python Script to Consume Kafka Logs

To enable the integration of logs from Kafka into Elasticsearch, create the following Python script and save it as `kafka_to_elasticsearch.py`:

```
from kafka import KafkaConsumer
from elasticsearch import Elasticsearch
import json
import datetime

# Kafka and Elasticsearch configurations
KAFKA_BROKER = '65.1.69.224:9092'  # Update to your Kafka broker IP
TOPIC_NAME = 'test-topic'
ELASTICSEARCH_HOST = 'http://65.1.69.224:9200'
ELASTIC_USER = 'elastic'
ELASTIC_PASSWORD = 'elastic'
INDEX_NAME = 'test-index'

# Connect to Elasticsearch
es = Elasticsearch(
    [ELASTICSEARCH_HOST],
    http_auth=(ELASTIC_USER, ELASTIC_PASSWORD)
)

# Connect to Kafka
consumer = KafkaConsumer(
    TOPIC_NAME,
```

```
    bootstrap_servers=KAFKA_BROKER,
    value_deserializer=lambda m: json.loads(m.decode('utf-8')),
    auto_offset_reset='earliest',  # Start reading from the beginning
    enable_auto_commit=True
)

# Consume and ingest logs into Elasticsearch
print("Starting to consume messages from Kafka and ingest into Elasticsearch...")
for message in consumer:
    log_data = message.value
    log_entry = {
        "message": log_data.get("message", ""),
        "timestamp": log_data.get("timestamp", datetime.datetime.utcnow().isoformat()
    }

    # Index the log entry into Elasticsearch
    response = es.index(index=INDEX_NAME, body=log_entry)
    print(f"Indexed log entry: {response['_id']}")
```

2. Run the Script

Execute the script using the following command:

```
python3 kafka_to_elasticsearch.py
```

Expected Outcome

Once the script is executed:

- The Kafka consumer will begin reading logs from the `test-topic`.
- Log messages will be transformed into structured data containing a `message` and `timestamp`.
- These entries will be indexed in Elasticsearch under the `test-index`.

Sample Output

As the script runs, indexed log entries will appear in the terminal, similar to the following:

```
Starting to consume messages from Kafka and ingest into Elasticsearch...
Indexed log entry: aBc123xYz
Indexed log entry: dEf456wUv
Indexed log entry: gHi789tQr
```

Phase 5: Kibana Visualization

Steps

1. Create an Index Pattern

To start visualizing the data ingested into Elasticsearch using Kibana, follow these steps:

1. Open Kibana by navigating to `http://<ec2-ip>:5601`.
2. Go to **Stack Management** ¿ **Index Patterns**.
3. Click on **Create Index Pattern**.
4. Enter `test-index` in the index pattern field.
5. Configure the timestamp field to match your data (e.g., `@timestamp`).
6. Save the configuration.

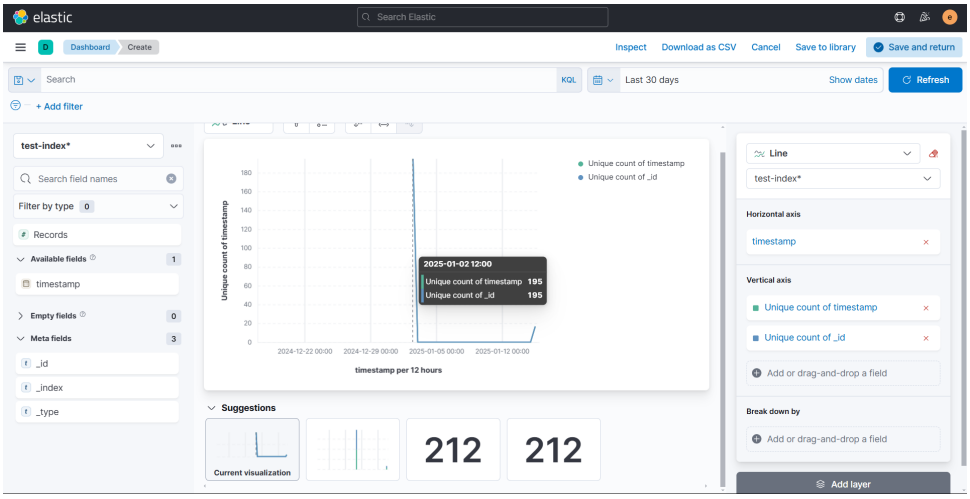

Figure 1: Visual showing requests count per dates and counts per date

2. Build Dashboards

Kibana provides tools to explore and visualize log data. Use the following features:

Discover Tab:

- Navigate to the **Discover** tab.
- Use the query bar to filter logs (e.g., `level: ERROR`).
- View logs in a tabular format or expand individual entries for more details.

Visualize Tab:

- Navigate to the **Visualize** tab.
- Click on **Create Visualization** and select a chart type (e.g., Pie Chart, Bar Graph).
- Select the `test-index` as the data source.
- Configure aggregations such as:

  - Split data by `log level` for a Pie Chart.
  - Use `time histogram` for Bar Graphs to visualize logs over time.
- Save the visualization for future use.

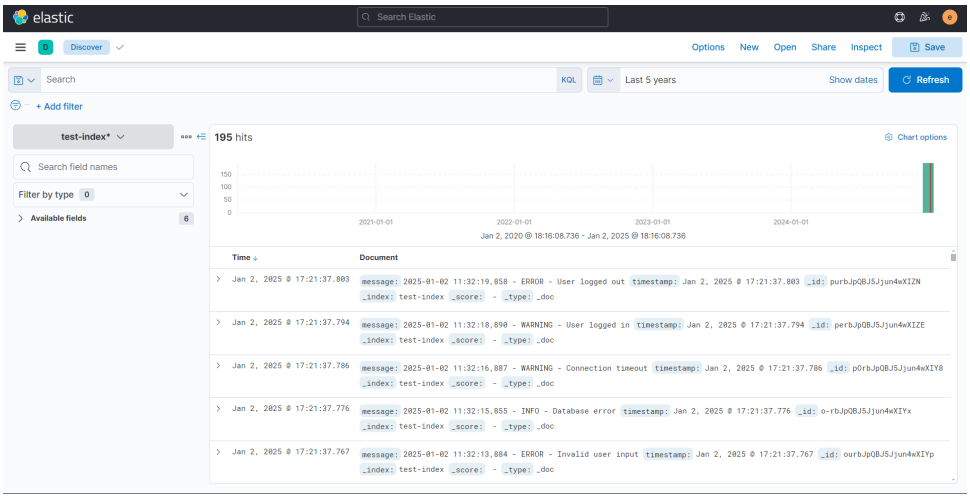

Figure 2: Logs records

Expected Outcome

After completing these steps, you will:

- Be able to explore logs in the **Discover** tab.
- Create insightful visualizations such as Pie Charts and Bar Graphs based on log levels and time-based trends.
- Use these visualizations to build dashboards for monitoring.

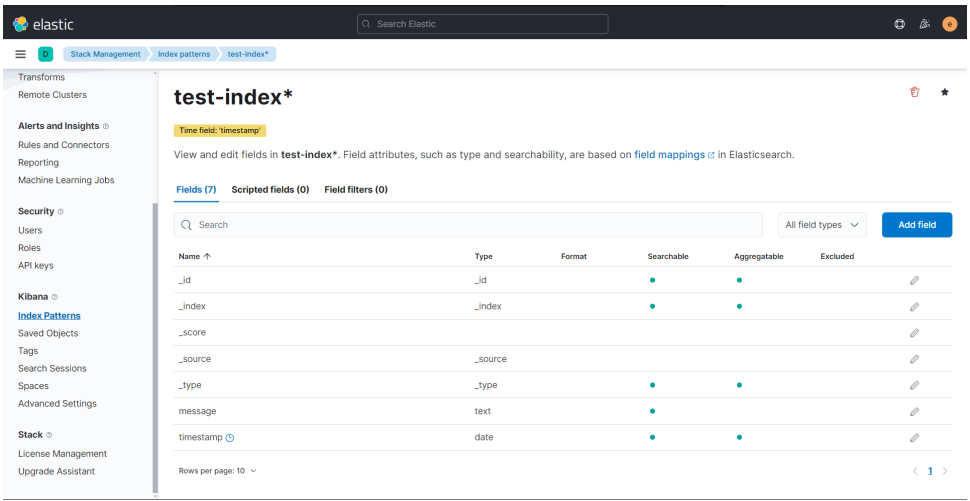

Figure 3: Logs Structures per category division

Summary

Kibana offers an intuitive interface to interact with Elasticsearch data. With the Index Pattern created, the logs can be explored, visualized, and added to dashboards for effective monitoring of log data. This phase completes the visualization component of the system.

Phase 6: Docker Compose

This phase focuses on using Docker Compose to set up the entire log analytics stack, integrating Kafka, Zookeeper, Elasticsearch, and Kibana.

1. Dockerfile for Kafka and Zookeeper

This Dockerfile sets up Kafka and Zookeeper, which are required to run Kafka.

```
# Base image
FROM wurstmeister/kafka:2.13-2.8.1

# Environment variables for Kafka and Zookeeper
ENV KAFKA_ZOOKEEPER_CONNECT=zookeeper:2181
ENV KAFKA_ADVERTISED_HOST_NAME=localhost
ENV KAFKA_BROKER_ID=1
ENV KAFKA_LOG_DIRS=/kafka/kafka-logs
ENV KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR=1

# Expose necessary ports
EXPOSE 9092
```

Listing 1: Kafka and Zookeeper Dockerfile

2. Dockerfile for Elasticsearch

This Dockerfile sets up Elasticsearch for data ingestion.

```
# Base image
FROM docker.elastic.co/elasticsearch/elasticsearch:8.10.0

# Set environment variables
ENV discovery.type=single-node

# Expose necessary port
EXPOSE 9200
```

Listing 2: Elasticsearch Dockerfile

3. Dockerfile for Kibana

This Dockerfile sets up Kibana for visualization.

```
# Base image
FROM docker.elastic.co/kibana/kibana:8.10.0

# Expose necessary port
EXPOSE 5601
```

Listing 3: Kibana Dockerfile

4. Docker Compose Configuration

The following Docker Compose file integrates all services: Kafka, Zookeeper, Elasticsearch, and Kibana into a single setup.

```
version: '3.8'
services:
  zookeeper:
    image: wurstmeister/zookeeper
    ports:
      - "2181:2181"

  kafka:
    image: wurstmeister/kafka
    ports:
      - "9092:9092"
    environment:
      KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
      KAFKA_ADVERTISED_HOST_NAME: localhost
      KAFKA_BROKER_ID: 1
      KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
    depends_on:
      - zookeeper

  elasticsearch:
    image: docker.elastic.co/elasticsearch/elasticsearch:8.10.0
    environment:
      discovery.type: single-node
    ports:
      - "9200:9200"

  kibana:
    image: docker.elastic.co/kibana/kibana:8.10.0
    ports:
      - "5601:5601"
    depends_on:
      - elasticsearch
```

Listing 4: docker-compose.yml

5. Running the Setup

To start all services using Docker Compose, follow these steps:

1. Place the `docker-compose.yml` file in your project directory.
2. Run the following command to start all services:

```
    docker-compose up
```

3. Verify that the services are running:

- Kafka: Connect to `localhost:9092`.
- Elasticsearch: Access `http://localhost:9200`.
- Kibana: Access `http://localhost:5601`.

After running the commands above, all components will be up and running, allowing log processing, storage, and visualization using Kibana dashboards.