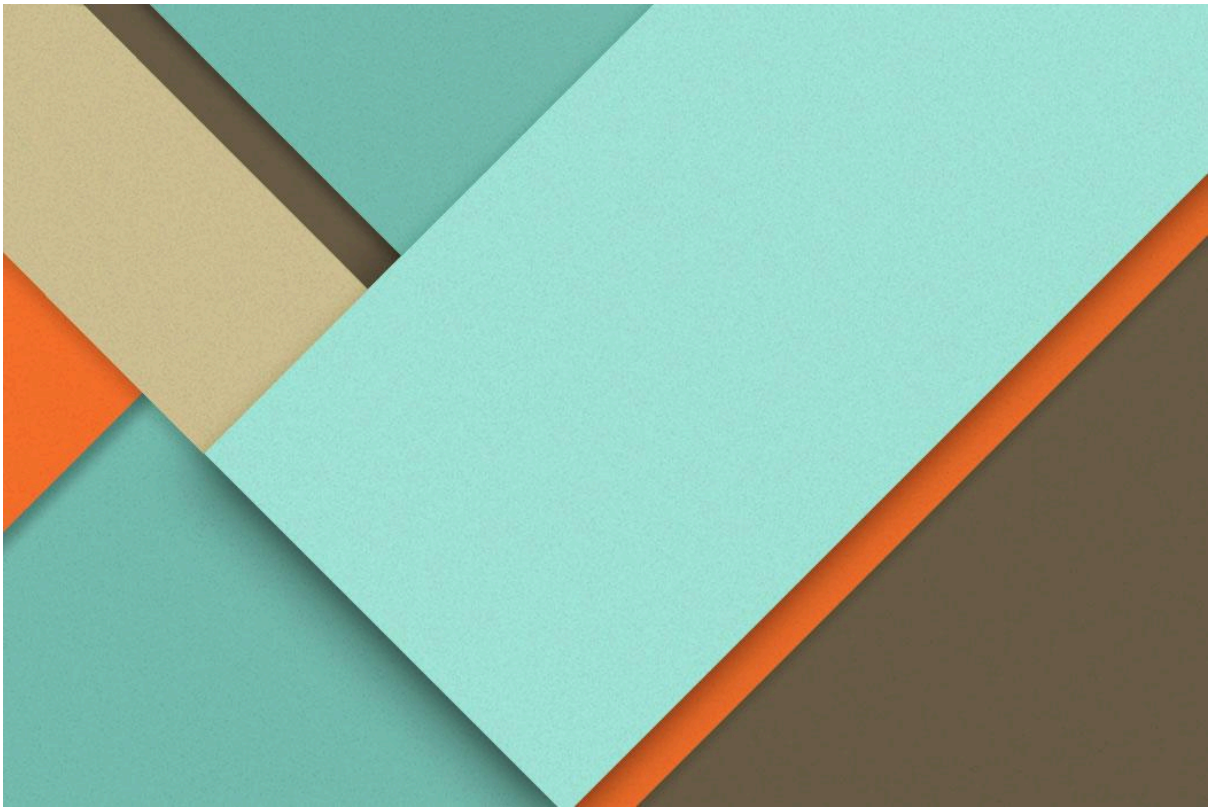


Project: Weather Stations Monitoring

Designing Data-Intensive Applications

—

Name	ID
Toka Ashraf Abo Alwafa	19015539
Rowan Nasser Edress	19015686
Nada Mohamed Ibrahim	19016782

Problem Statement

You are required to simulate a distributed weather monitoring system with multiple components, including weather stations, data processing, storage, and analysis. This system will utilize Kafka for messaging, BitCask Riak for key-value storage, and Parquet files for archiving. Additionally, the system must be deployed using Docker and Kubernetes.

Requirements

A) Weather Station Mock

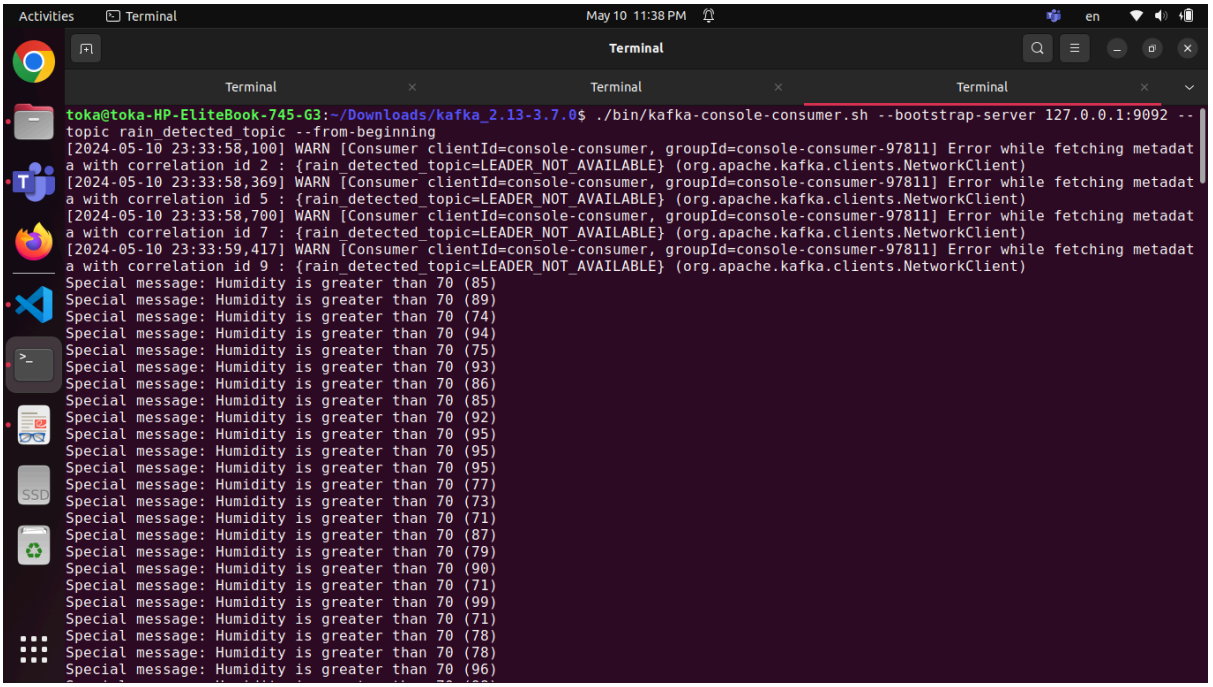
- The weather station ID is retrieved, and a WeatherStation task is scheduled to run every second using a scheduled executor service.
- The WeatherStation class, which implements Runnable, runs periodically and generates a weather status message each time it executes.
- Within each execution:
    - A Unix timestamp is generated.
    - Random values for humidity, temperature, and wind speed are created.
    - The battery status is randomly updated.
    - Occasionally, the message is dropped to simulate message loss.
    - If not dropped, a JSON string is constructed with the weather data.

B) Kafka Integration:

Weather Station Connection to use Kafka's produce API to send status messages to certain topic (weather_topic) in the Kafka server.

C) Rain Detection using Kafka Processors:

- Use Kafka Processors to detect if humidity > 70% by consuming from the topic that contains all weather statuses and extract the humidity to see if it more than 70 print special message in another topic (rain_detected_topic).



D) Central Station Implementation:

1. BitCask Riak for Status Storage:

- Maintain an updated key-value store of each station's latest status using BitCask Riak LSM. and implementing hint files to help in rehash for recovery also schedule the compaction process to help to reclaim space by removing old versions of keys and consolidating data, ensuring that the storage remains efficient and does not degrade over time.
- Bitcask can offer the following benefits in our project:

  1. The stations can generate a lot of data in real-time. Bitcask's high write performance ensures that all this data can be logged efficiently without slowing down the system.

  2. Quick Data Access as when querying the status of a particular station, the in-memory index allows for very fast data retrieval, making the system responsive.

  3. In case of system crashes, the hint files help in quickly recovering the state, ensuring that the system can get back online with minimal data loss.

  4. Scalability as the number of stations increases, the system can handle the increased load without a significant drop in performance, thanks to the scalable nature of Bitcask.

5. Efficient Storage Management: as regular compaction ensures that the storage remains efficient and does not bloat with outdated data, keeping the system performance optimal.

2. Historical Data Archiving:

- Archive all weather statuses that are stored in kafka into Parquet files.
- Partition Parquet files by time and station ID by making a folder with the date and this contains the ten stations each one contains parquet files partitioned by the time.
- Write records in batches to avoid i/o blocking by making the operation of writing not start until the size of the batch is reached.

E) Historical Weather Status Analysis:

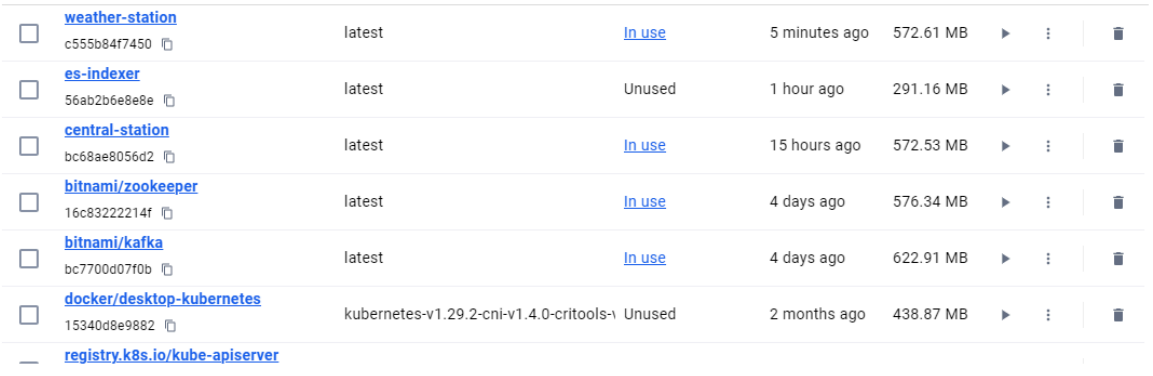
1. ElasticSearch Integration:

- Direct all weather statuses to ElasticSearch for indexing.
- Make Query to get the percentage of the dropped messages by dividing the total messages that arrived to elastic search by the total messages without dropping that calculated using the distinct values of s_no and multiply them by the number of stations.

2. Kibana is used to visualize and confirm the percentage of the battery status and dropped messages.

F) Deployment using Kubernetes:

1. Docker Configuration:



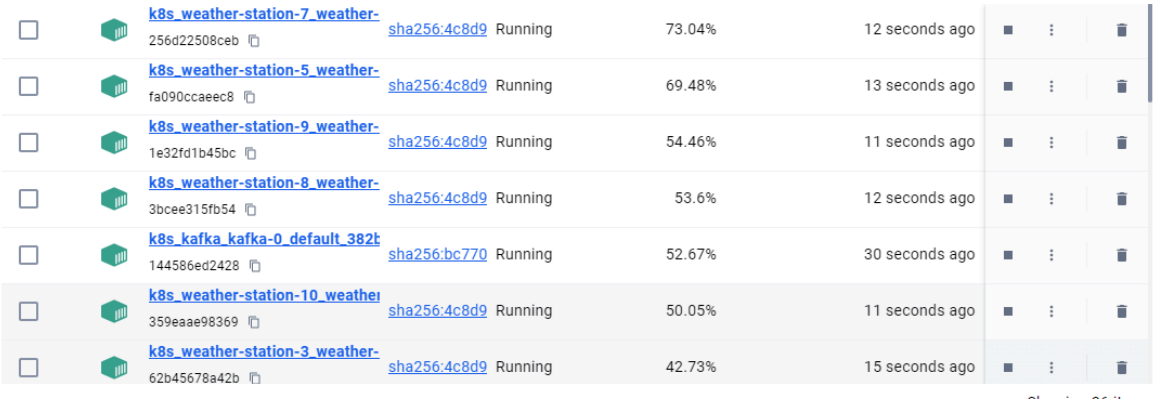
- Write Dockerfile for the central server.
  The Dockerfile sets up an OpenJDK environment, configures it to use a specific security policy, and then runs a Java application within that environment.
- Write Dockerfile for the weather stations.
  The Dockerfile sets up an OpenJDK environment, copies the application JAR

file to the appropriate directory, sets the working directory, and runs the Java application.

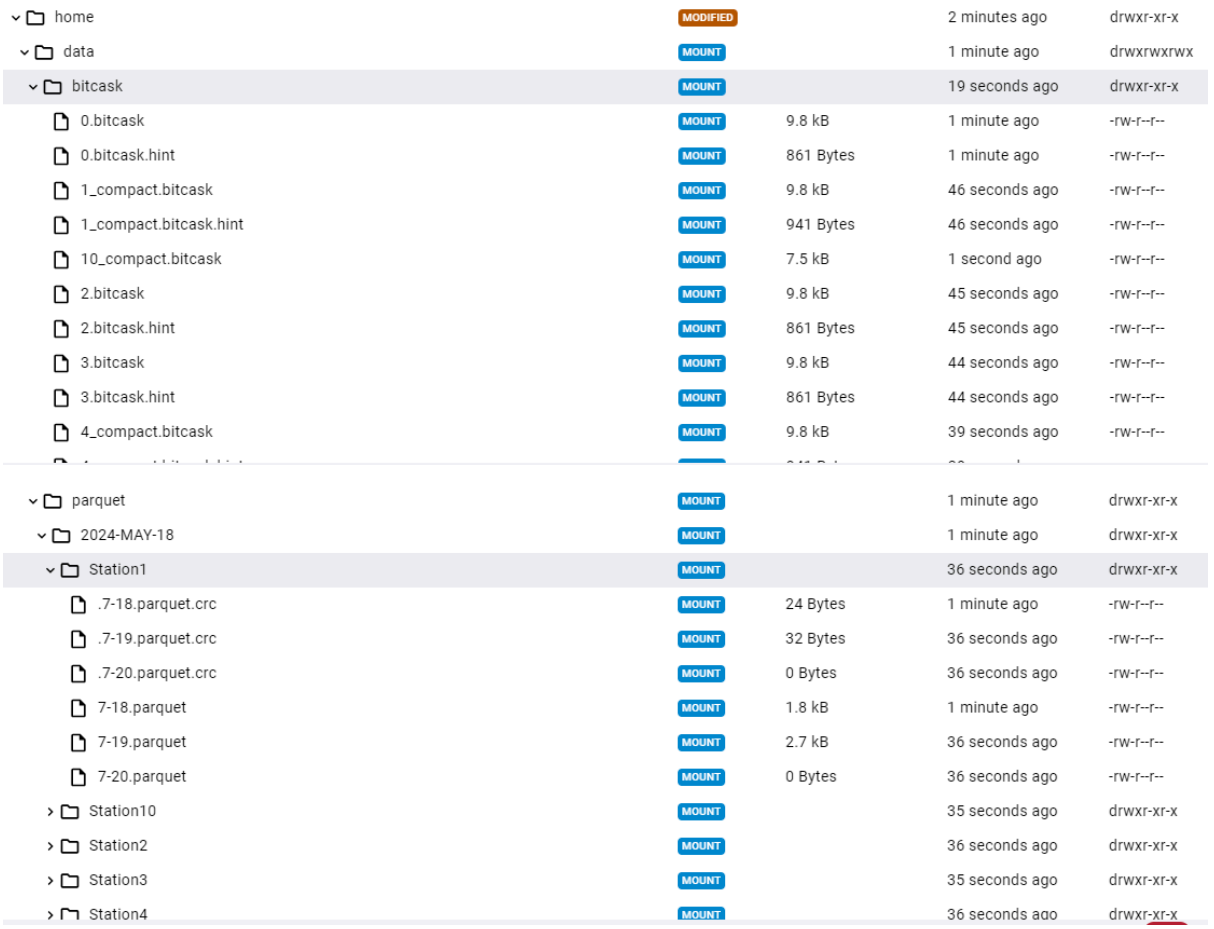
- Write Dockerfile for the Indexer
  This Dockerfile sets up a Docker image with Python installed, installs the necessary dependencies, copies the Elasticsearch script into the container, and specifies the command to run the script when the container starts.

2. Kubernetes Setup:

- Deploy 10 weather station services using their Docker image.

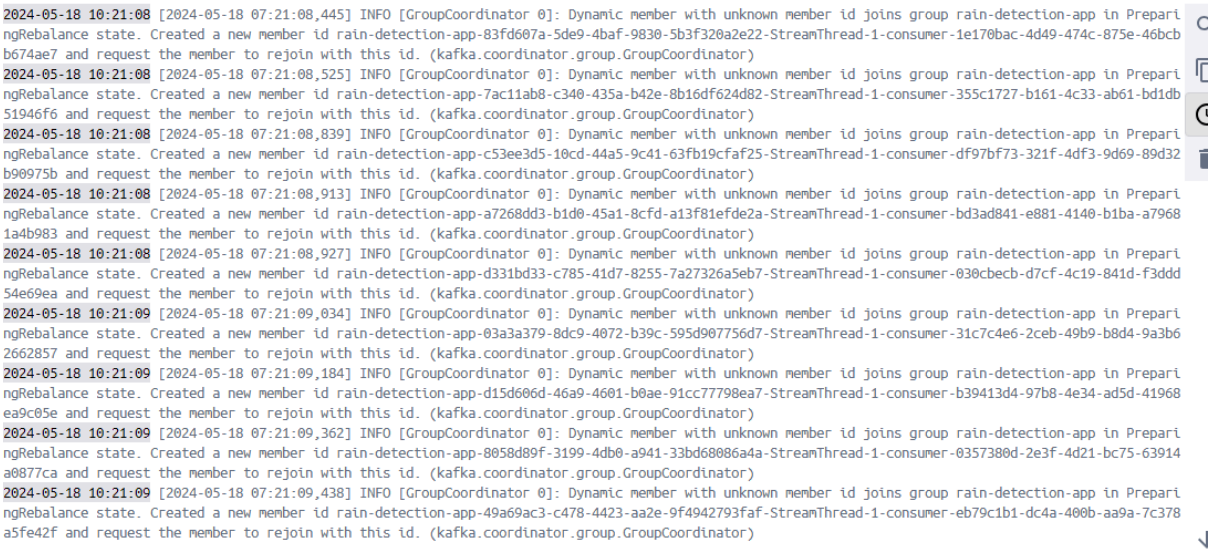


- This YAML file defines 10 Deployments and corresponding Services for 10 instances of the weather station application. Each instance has its own unique identifier and is exposed through a separate service.
- Deploy a central server service using its Docker image.



This YAML file ensures that the "central-station" component runs as a single replica, with access to Kafka for messaging and persistent storage through shared volumes. The service exposes port 8080 to facilitate communication with other components within the Kubernetes cluster.

- Deploy Kafka and Zookeeper as two services.



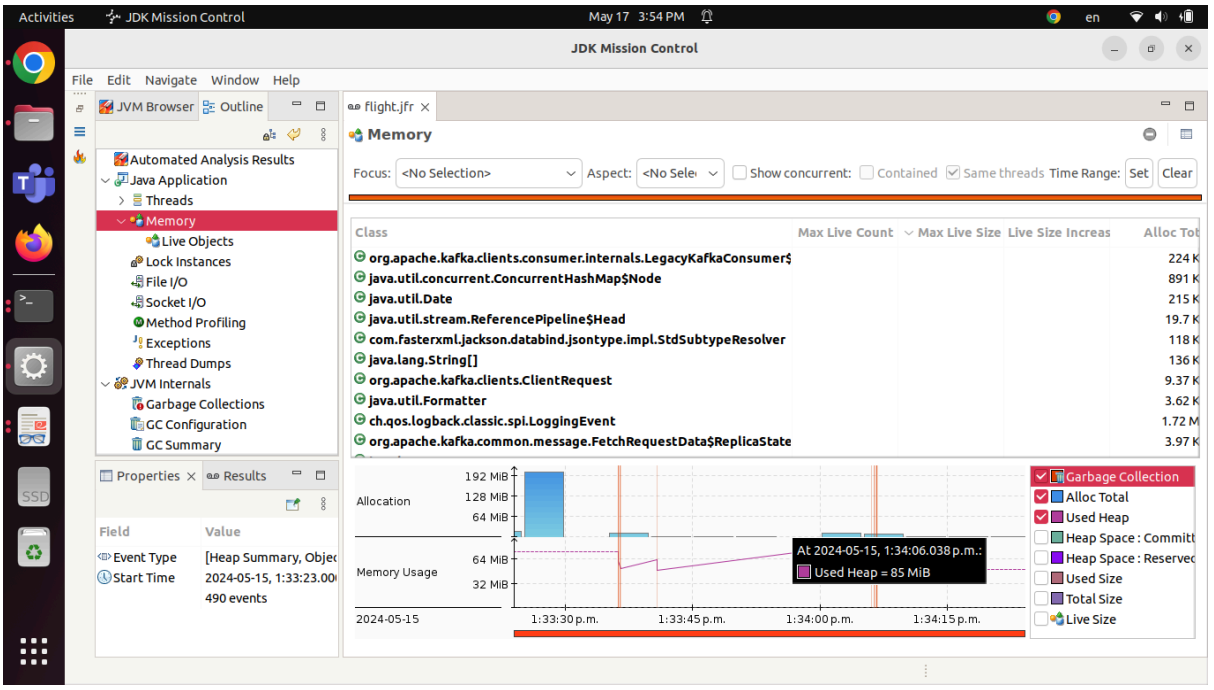
This YAML file sets up a Zookeeper service and StatefulSet, a Kafka service

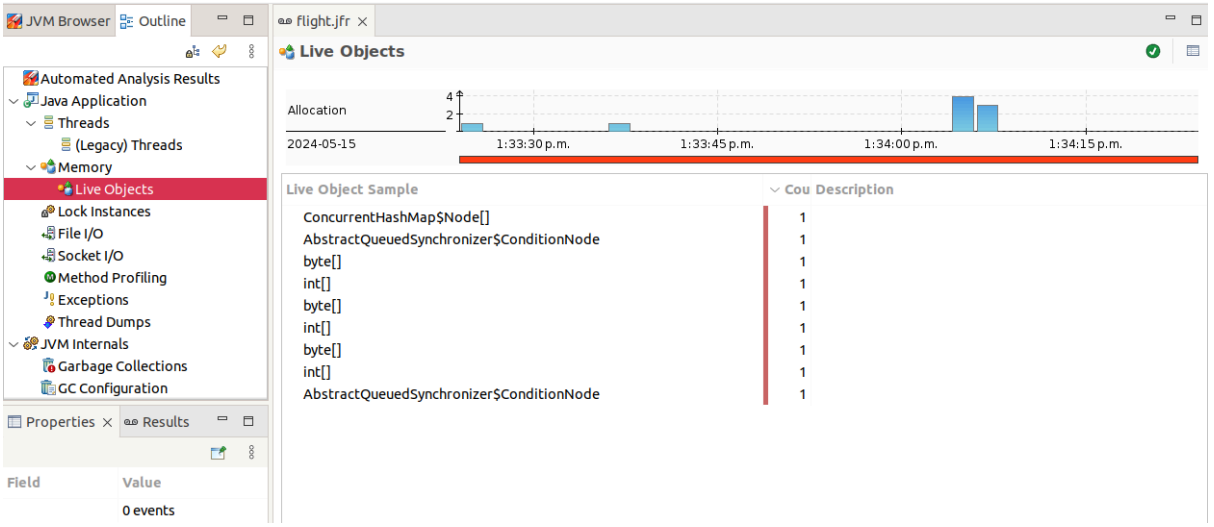
and StatefulSet, along with a shared volume for data storage, and a ConfigMap for Kafka topics.

- Configure shared storage for Parquet files and BitCask Riak LSM.
  This YAML file creates a PV named "shared-volume" with a capacity of 10 gigabytes, accessible by multiple nodes with read-write access. It also creates a corresponding PVC with the same name and access mode, allowing pods to dynamically claim and use the storage provided by the PV.

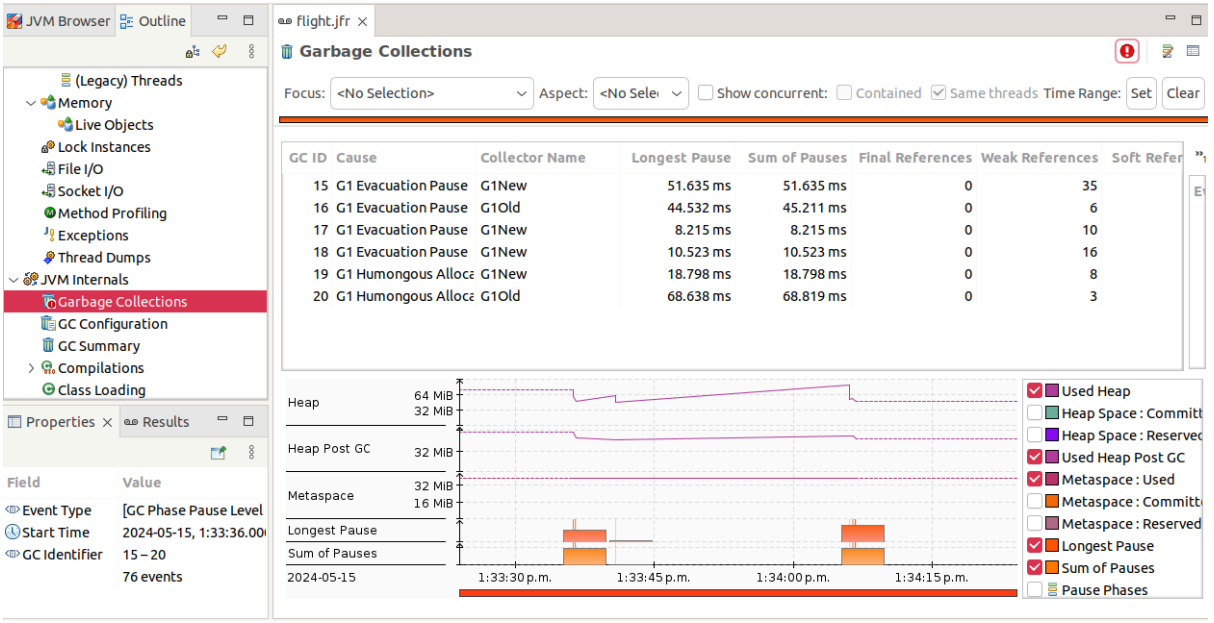
G) Profile Central Station using JFR

1. Top 10 Classes with highest total memory

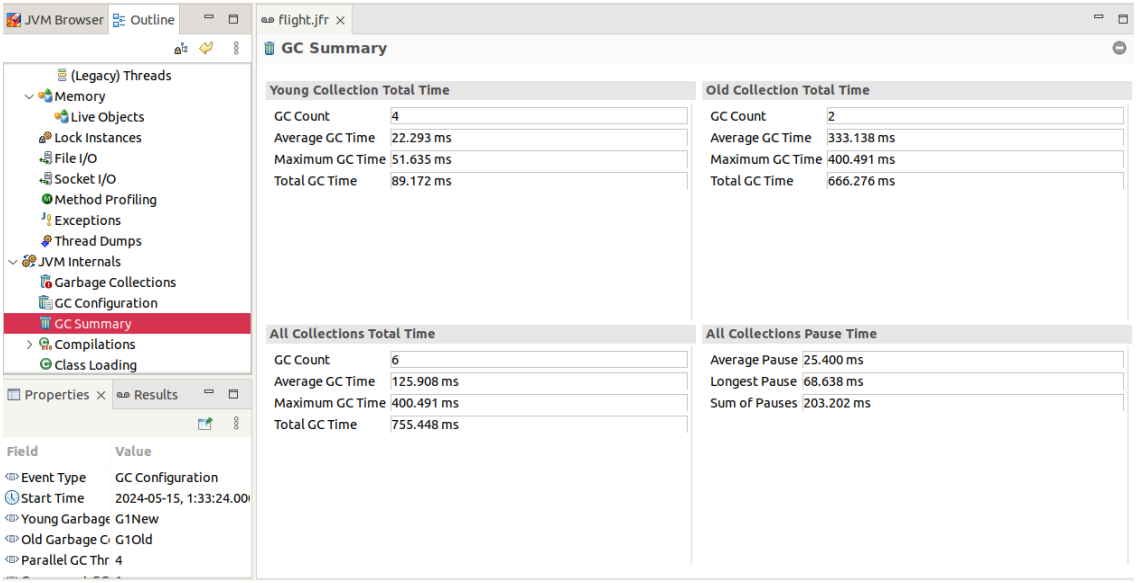




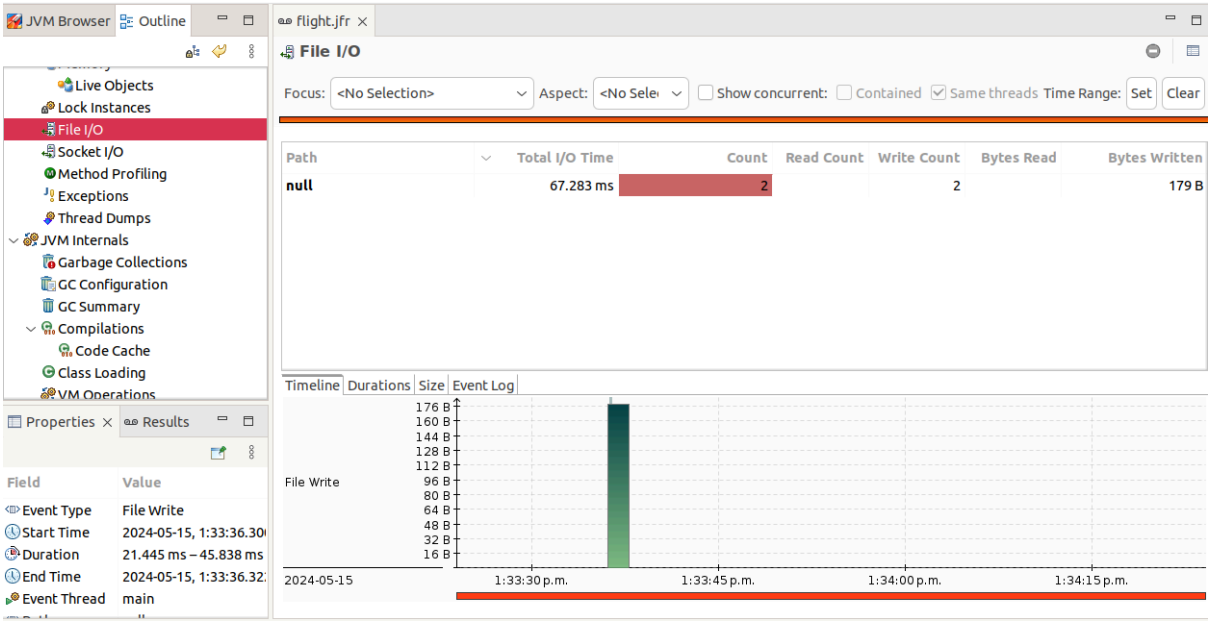
2. GC pauses count



3. GC maximum pause duration



4. List of I/O operations



Results :

Docker & Kubernetes files:

- K8s yaml file: Link
- Weather Station Dockerfile

```
# Use an OpenJDK base image
FROM openjdk:23-slim

# Copy necessary files
COPY Weather_Stations_Monitoring.jar /app/Weather_Station.jar

# Set the working directory
WORKDIR /app

# Command to run the application
CMD ["java", "-jar", "Weather_Station.jar"]
```

- Central Server Dockerfile

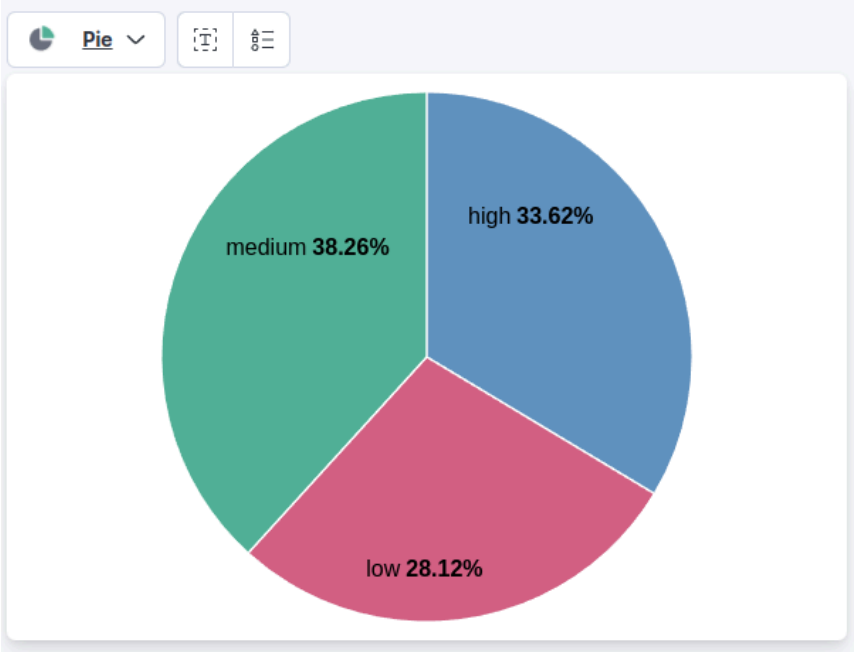
```
# Use an official OpenJDK runtime as a parent image
FROM openjdk:23-slim

# Set the working directory
WORKDIR /app

# Copy the application JAR file and any other necessary files
COPY Weather_Stations_Monitoring.jar /app/Weather_Stations_Monitoring.jar

# Copy the policy file to the container
COPY java.policy /app/java.policy

# Set the Java options to use the policy file
ENV JAVA_OPTS="-Djava.security.manager -Djava.security.policy=/app/java.policy"

# Run the application
CMD ["sh", "-c", "java $JAVA_OPTS -jar Weather_Stations_Monitoring.jar"]
```

- Indexer Dockerfile

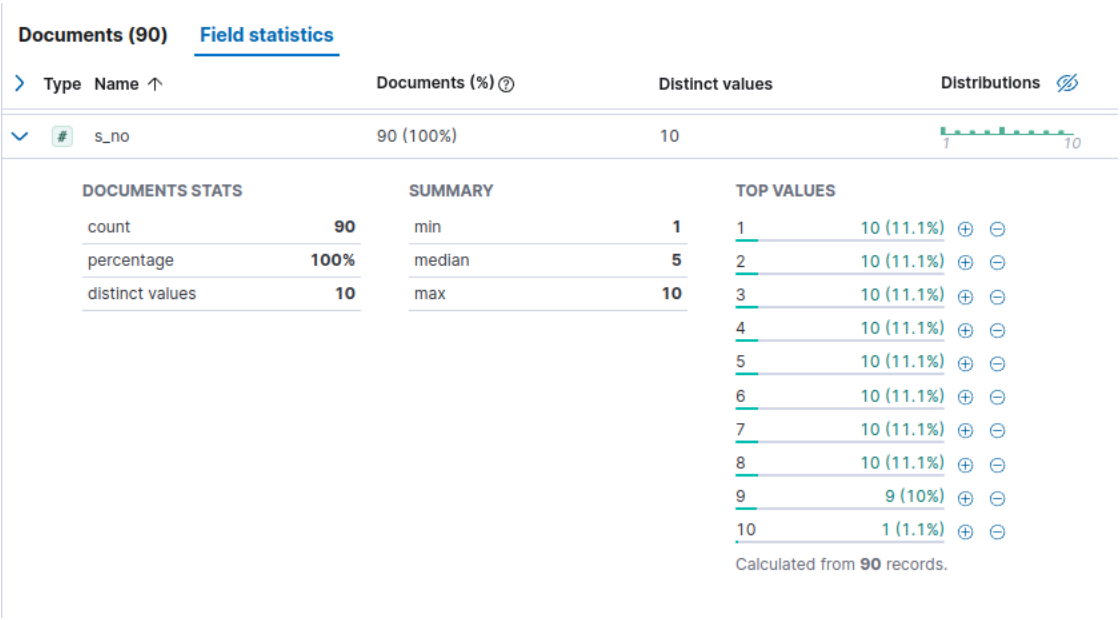
```
# Dockerfile for Elasticsearch Script
FROM python:3.8-slim

WORKDIR /app

# Install required Python packages
COPY requirements.txt requirements.txt
RUN pip install --no-cache-dir -r requirements.txt

# Copy the script into the container
COPY elastic_search.py elastic_search.py

# Set the entrypoint for the container
CMD ["python", "elastic_search.py"]
```

- **Screenshots for Kibana visualization confirming:**

  - Battery status distribution (30% low - 40% medium - 30% high)



  - 10% dropped messages



- [Sample Parquet File Link]

- [Sample BitCask Riak LSM directory]

Bonus: Open Meteo

- It is designed to fetch specific hourly weather data from an API for a given date and station, handling necessary date formatting and JSON parsing to retrieve and return the desired information of the weather that represents humidity, temperature and wind speed.
- Channel Adapter is designed to take the json object that resulted from the api to open meteo and extract the desired data to make an object of the weather information to use in the weather station mock.