# DDIA


# Project: Weather Stations Monitoring.


| Name | ID |
|---|---|
| Karim Fathy | 20011116 |
| Mohamed Amr | 20011675 |
| Omar Mahmoud | 20011027 |
| Omar Tarek | 20010998 |

# Project main points:

# Bitcask

## Overview

This Java-based Bitcask server is a lightweight, concurrent TCP server designed to handle read (r), write (w), and append (a) operations from clients. It supports multi-threaded request processing and asynchronous log tracking for durability and performance optimization.

## Key Components

### 1. Server Initialization

- The server reads configuration from a system.properties file, extracting:
    - server.port: TCP port to listen on.
    - server.logs: Path to the log file(s).
    - server.hints: Path to hint files (for indexing/optimization).
    - server.threads: Number of threads used to process requests.

### 2. Client Handling (handleClient method)

- Each client connection is handled in a separate thread.
- Uses a fixed thread pool (ExecutorService) to manage and execute Worker tasks.
- Accepts commands prefixed with:
    - w: Write
    - r: Read
    - a: Read All

- Supports the --no-reply flag to execute commands without awaiting a response.
- Valid requests are submitted to a Worker class for processing.
- Results (for requests expecting replies) are collected and sent back to the client.

### 3. Worker Execution

- Each Worker instance represents a task for handling a single client command.
- Asynchronous execution allows high throughput and non-blocking processing.

### 4. Log Tracking

- LogPathTracker is initialized to monitor the log directory.
- It polls at a fixed interval (every 5 seconds) and likely handles log file rotation or cleanup.
- Helps in managing storage and maintaining performance.

**5. Server Loop**

- The main thread continuously listens for incoming TCP connections.
- Upon accepting a new connection, it delegates client handling to a new thread using handleClient.

### Concurrency & Fault Tolerance

- Utilizes Java's ExecutorService for concurrent request execution.
- Separate threads for each client ensure isolation and responsiveness.
- Graceful shutdown of executors and sockets prevents resource leaks.
- Exceptions are logged, and processing continues robustly.

---

## Client Functionality (BitcaskClient)

- **Core Role**: Provides CLI-based interaction with the Bitcask server.

- **Commands Supported**:
    - --view-all: Retrieves all stored data as a CSV file.
    - --view --key=SOME_KEY: Retrieves a value for a specific key.
    - --perf --clients=N: Launches N concurrent client threads to perform --view-all operations for performance benchmarking.

- **Implementation Highlights**:

    - Uses Socket programming to connect to the server on localhost:5000 (modifiable).
    - Exports --view-all results to timestamped CSV files at a specified file path.
    - --perf tests simulate concurrent clients to evaluate server throughput.

---

# KafkaToBitcask (Normalizer)

## Overview

This component acts as a **bridge between Kafka and the Bitcask server**, consuming streaming data from a Kafka topic (weather-station), transforming it as needed, and sending each key-value record to the Bitcask server over TCP.

## Component Role

- **Input**: Kafka topic weather-station (stream of key-value string pairs).
- **Output**: TCP-based w (write) command to the Bitcask server.
- **Purpose**: Normalize incoming Kafka messages and persist them in a Bitcask-inspired data store.

## Configuration

- **Kafka Streams App ID**: bitcask-producer
- **Kafka Bootstrap Servers**: localhost:9092
- **SerDes**: String SerDes used for both key and value.

## Stream Processing Logic

```
KStream<String, String> stream = builder.stream("input-topic");
stream.foreach((key, value) -> {
   Long id = Long.parseLong(key);
   sendToBitcask(id, value, "localhost", 5000);
});
```

- **Key Parsing**: Converts key from String to Long.
- **Value Normalization**: Replaces commas in values with semicolons to avoid CSV conflicts.
- **Command Sent**: w <key> <value>--no-reply

---

# WeatherStatusIndexer (Elasticsearch)

## Overview

This Python class reads weather data in .parquet format from a directory of station folders, flattens nested fields, tags dropped messages, and indexes the results into an **Elasticsearch** index called "weather_statuses".

## Component Role

- **Input**: Parquet files under root_data_dir/stationId_x/ folders.
- **Output**: Documents indexed in Elasticsearch.
- **Goal**: Store real-time or batch IoT weather data for analysis and querying.

## Main Functionalities

### 1. flatten_and_tag(df, station_id)

- **Flattens** the nested "weather" field.
- **Adds**:
    - station_id
    - status_timestamp as datetime
    - dropped flag: True if sequence number (s_no) skips.
- **Sorts** by s_no to detect dropped messages.

### 2. index_exists_or_create()

- Deletes the index if it already exists.
- Creates a new index with mappings:
    - station_id, s_no: long
    - battery_status: keyword
    - status_timestamp: date
    - humidity, temperature, wind_speed: integer
    - dropped: boolean

### 3. process_all_stations()

- Iterates through all subfolders.
- Extracts station ID from folder name like stationId_3.

- Reads all .parquet files.
- Applies flattening and tagging.
- Aggregates all documents.

**4. bulk_upload(docs)**

- Uses helpers.bulk() from elasticsearch-py to send all docs.

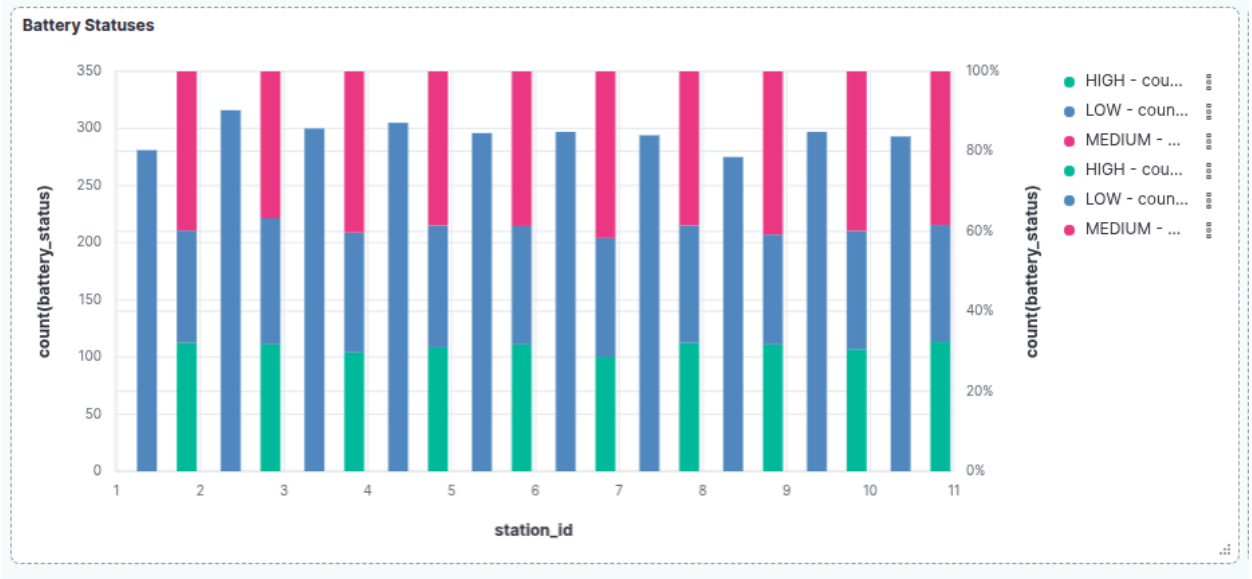
**5. run()**

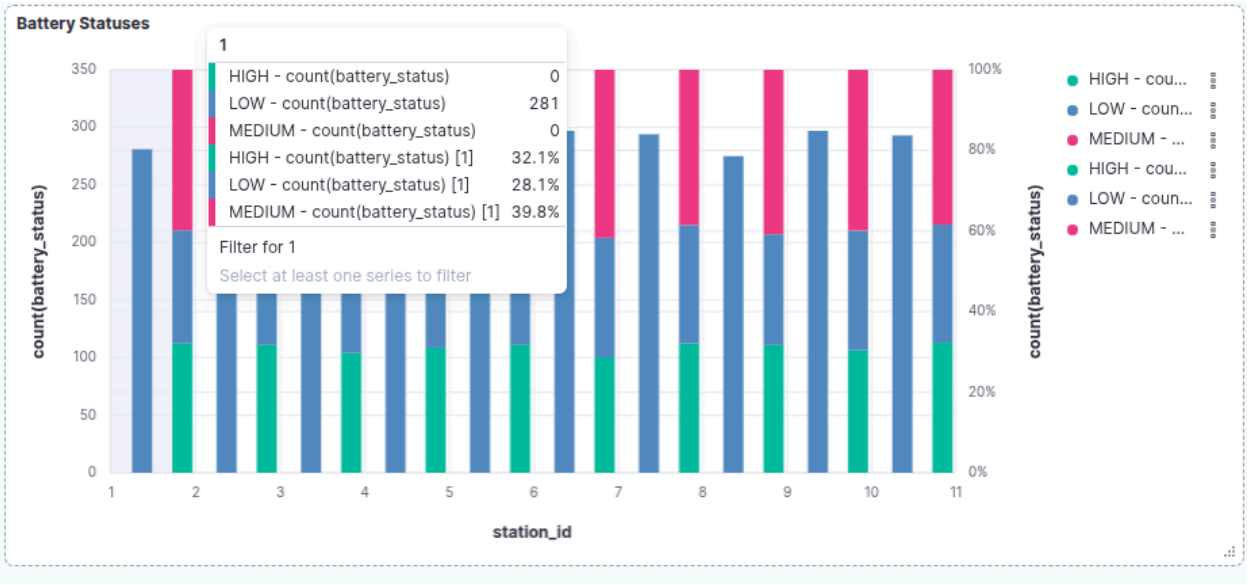
- Calls the methods in sequence: index creation → processing → upload.

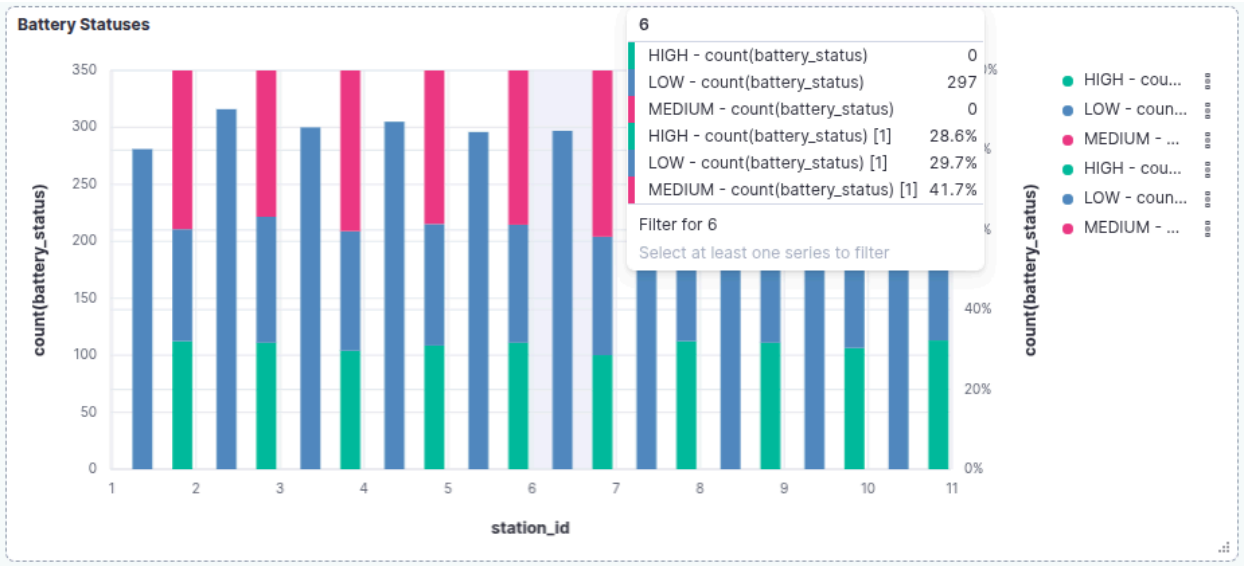
---

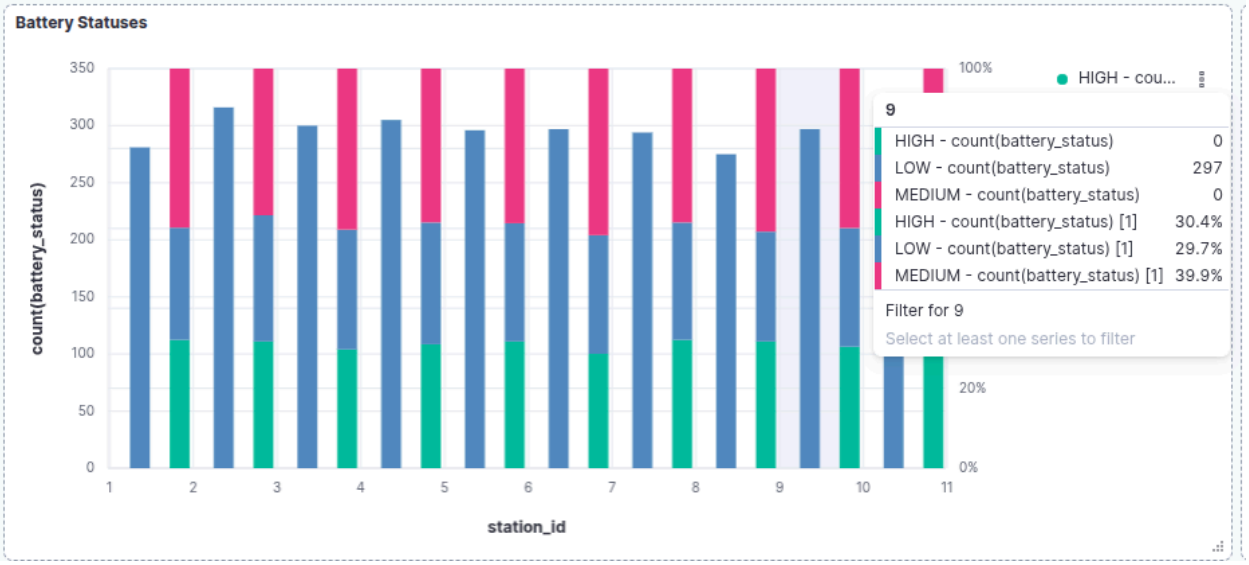
# Kibana visualization

## Battery Statuses

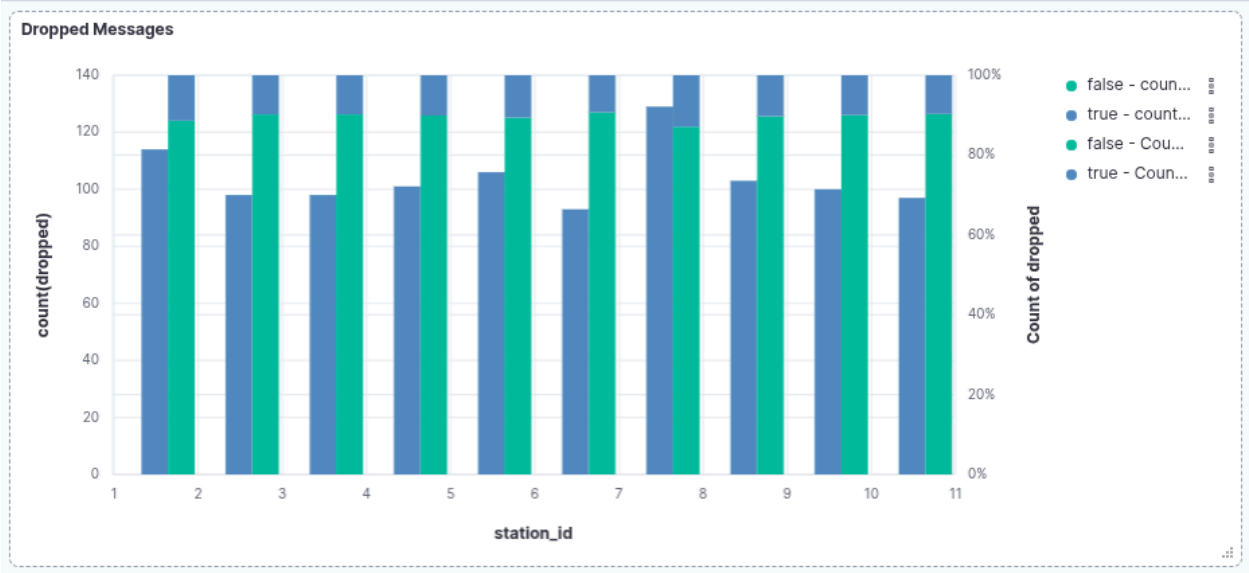


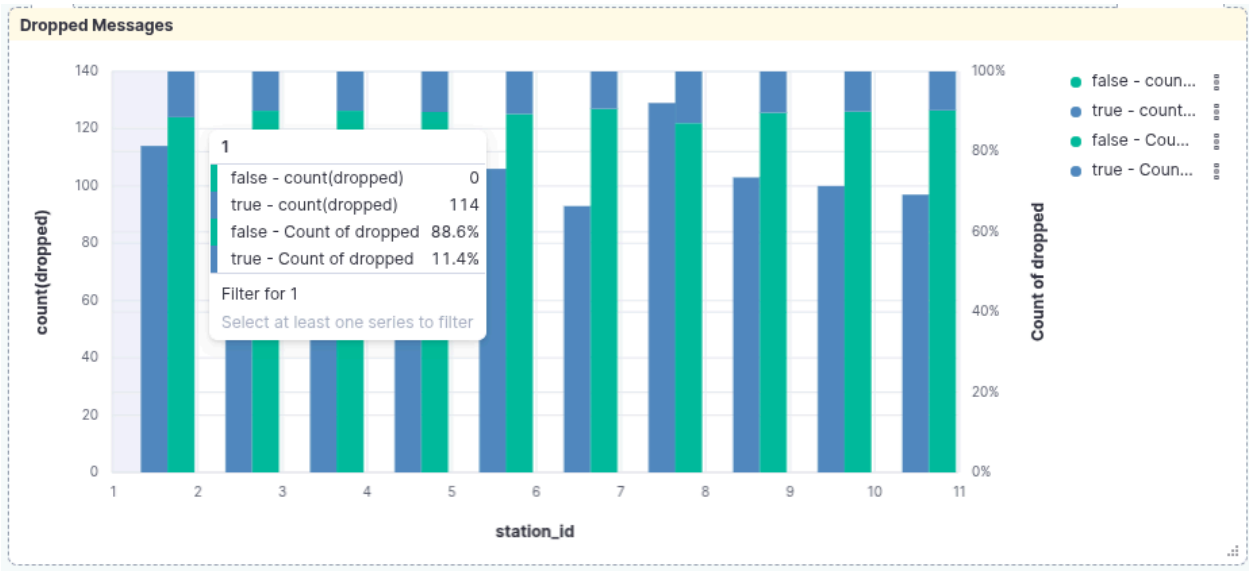

Battery Statuses
1
HIGH - count(battery_status) 0
LOW - count(battery_status) 281
MEDIUM - count(battery_status) 0
HIGH - count(battery_status) [1] 32.1%
LOW - count(battery_status) [1] 28.1%
MEDIUM - count(battery_status) [1] 39.8%
Filter for 1
Select at least one series to filter
count(battery_status)
350
300
250
200
150
100
50
0
1
2
3
4
5
6
7
8
9
10
11
station_id
100%
80%
60%
40%
20%
0%
count(battery_status)
HIGH - cou...
LOW - coun...
MEDIUM - ...
HIGH - cou...
LOW - coun...
MEDIUM - ...

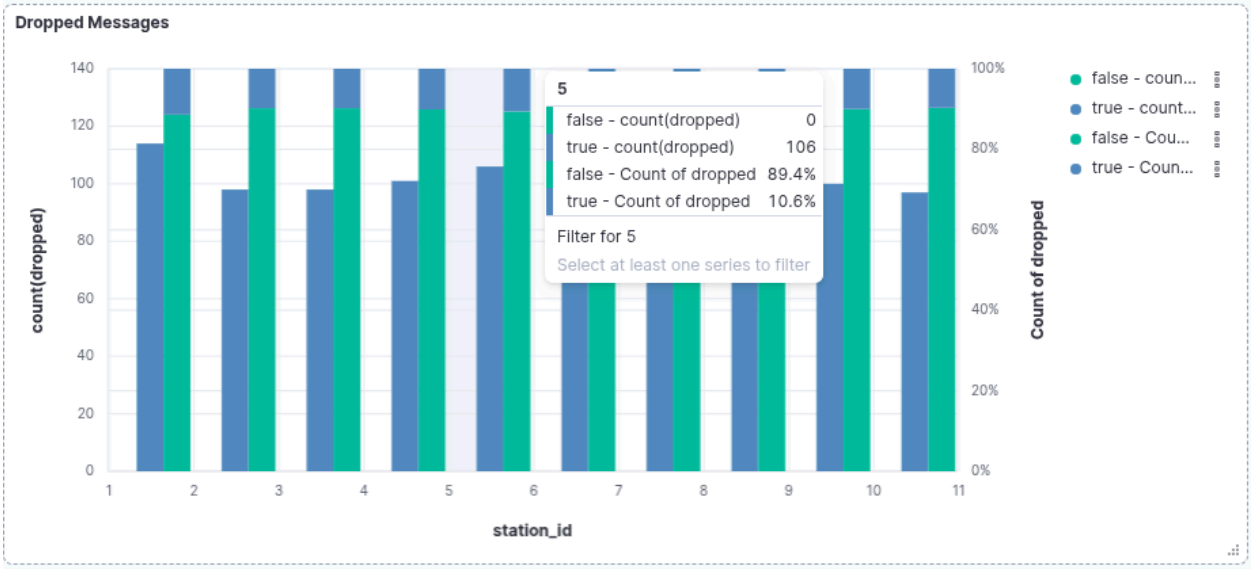


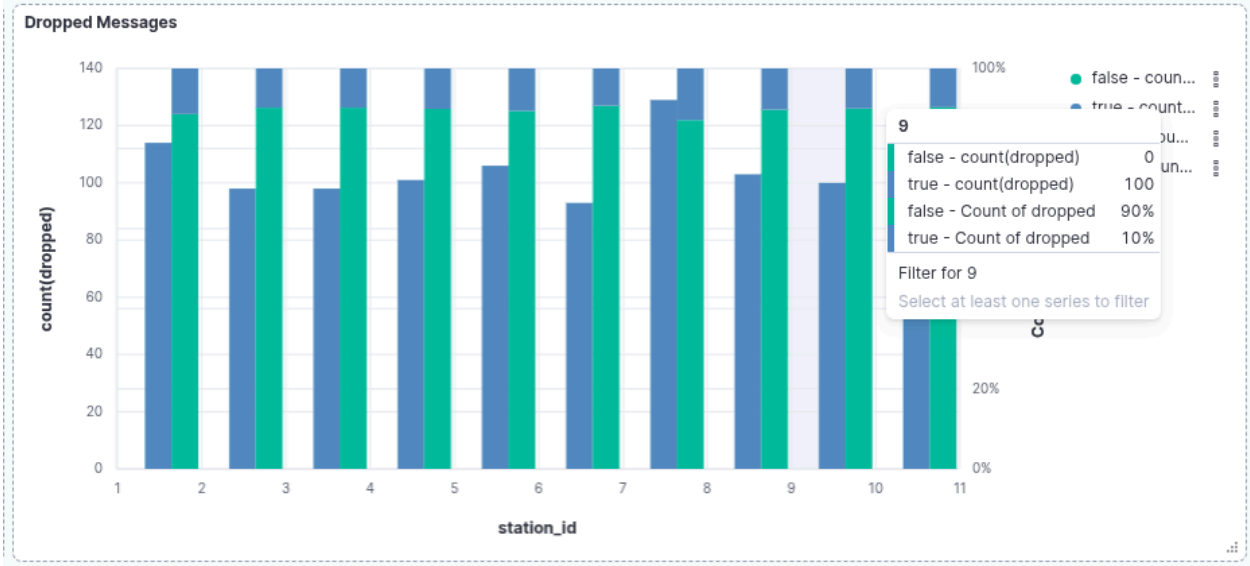




# Dropped Messages


Dropped Messages
count(dropped)
Count of dropped
station_id
false - coun...
true - count...
false - Cou...
true - Coun...


Dropped Messages
count(dropped)
Count of dropped
station_id
1
false - count(dropped) 0
true - count(dropped) 114
false - Count of dropped 88.6%
true - Count of dropped 11.4%
Filter for 1
Select at least one series to filter
false - coun...
true - count...
false - Cou...
true - Coun...



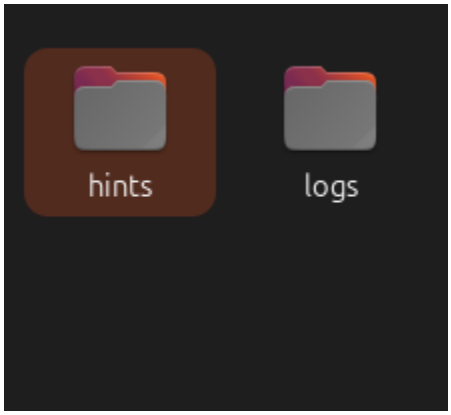



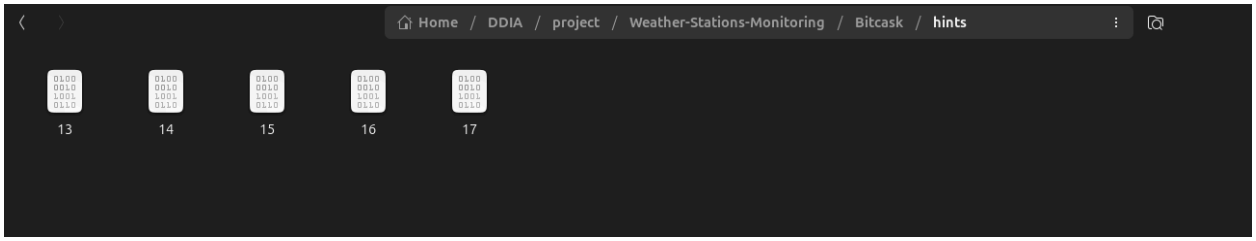

---

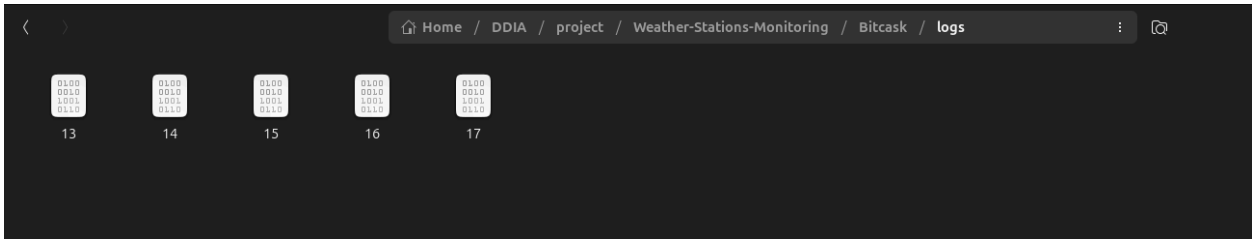
# BitCask Riak LSM directory

## Structure

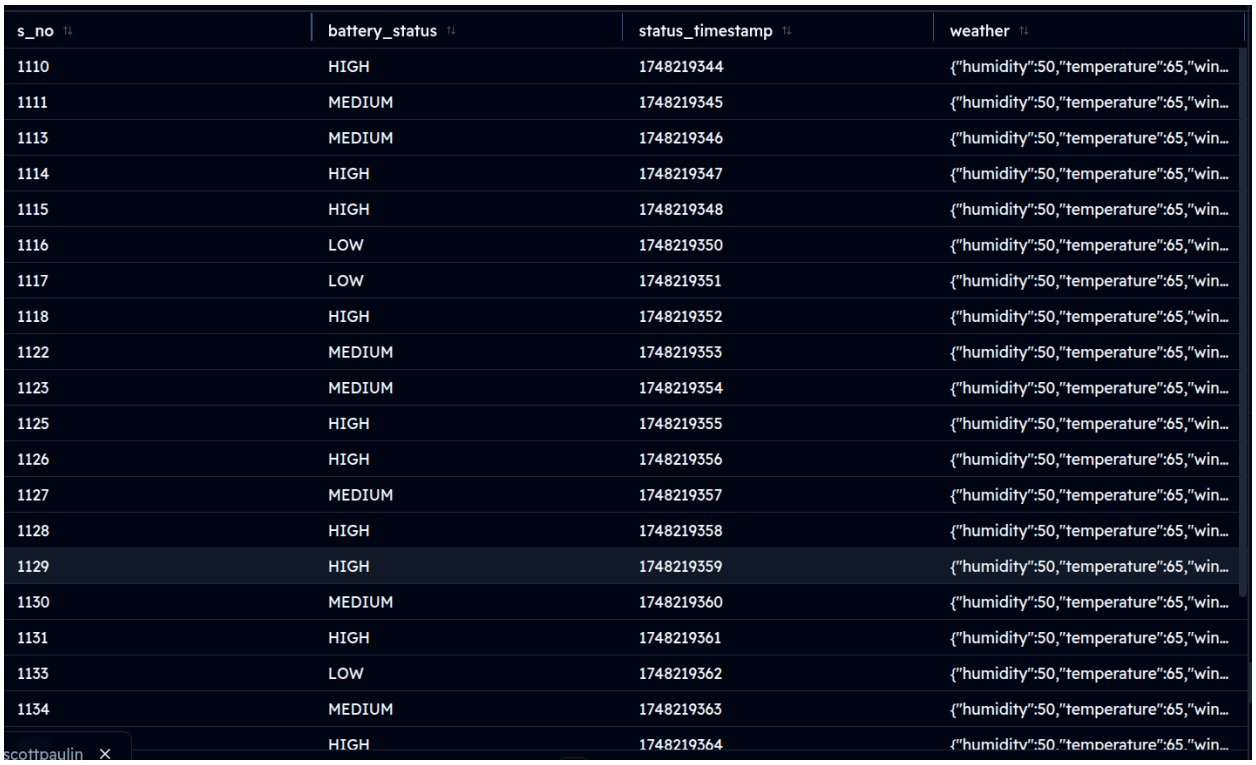
|
|--> logs —> 0, 1, 2, 3, 4
|--> hints —> 0, 1, 2, 3, 4


hints
logs


Home / DDIA / project / Weather-Stations-Monitoring / Bitcask / hints
13
14
15
16
17


Home / DDIA / project / Weather-Stations-Monitoring / Bitcask / logs
13
14
15
16
17

# Example parquet file:



| s_no | battery_status | status_timestamp | weather |
|---|---|---|---|
| 1110 | HIGH | 1748219344 | {"humidity":50,"temperature":65,"win... |
| 1111 | MEDIUM | 1748219345 | {"humidity":50,"temperature":65,"win... |
| 1113 | MEDIUM | 1748219346 | {"humidity":50,"temperature":65,"win... |
| 1114 | HIGH | 1748219347 | {"humidity":50,"temperature":65,"win... |
| 1115 | HIGH | 1748219348 | {"humidity":50,"temperature":65,"win... |
| 1116 | LOW | 1748219350 | {"humidity":50,"temperature":65,"win... |
| 1117 | LOW | 1748219351 | {"humidity":50,"temperature":65,"win... |
| 1118 | HIGH | 1748219352 | {"humidity":50,"temperature":65,"win... |
| 1122 | MEDIUM | 1748219353 | {"humidity":50,"temperature":65,"win... |
| 1123 | MEDIUM | 1748219354 | {"humidity":50,"temperature":65,"win... |
| 1125 | HIGH | 1748219355 | {"humidity":50,"temperature":65,"win... |
| 1126 | HIGH | 1748219356 | {"humidity":50,"temperature":65,"win... |
| 1127 | MEDIUM | 1748219357 | {"humidity":50,"temperature":65,"win... |
| 1128 | HIGH | 1748219358 | {"humidity":50,"temperature":65,"win... |
| 1129 | HIGH | 1748219359 | {"humidity":50,"temperature":65,"win... |
| 1130 | MEDIUM | 1748219360 | {"humidity":50,"temperature":65,"win... |
| 1131 | HIGH | 1748219361 | {"humidity":50,"temperature":65,"win... |
| 1133 | LOW | 1748219362 | {"humidity":50,"temperature":65,"win... |
| 1134 | MEDIUM | 1748219363 | {"humidity":50,"temperature":65,"win... |
| | HIGH | 1748219364 | {"humidity":50,"temperature":65,"win... |

scottpaulin

Used online viewer to view parquet file in columns

# Source Code:

## Weather station:

```
public class Main {
   private static final Random rand = new Random();

   public static void main(String[] args) throws InterruptedException {

       long stationId = 1;
       String stID = System.getenv("STATION_ID");
       // Check if station ID is in the format "weather-station-X"
       if (stID != null && stID.matches("weather-station-\\d+")) {
           try {
               String idStr = stID.substring(stID.lastIndexOf('-') + 1);
               stationId = Long.parseLong(idStr)+1;
               System.out.println("Found weather-station format ID: " + stationId);
           } catch (NumberFormatException e) {
               System.err.println("Failed to parse station number from: " + stID);
           }
       }
       System.out.println("Station ID: " + stationId);

       long statusMsgCounter = 0;

       String bootstrapServers = System.getenv("KAFKA_BOOTSTRAP_SERVERS");
       if (bootstrapServers == null || bootstrapServers.isEmpty()) {
           bootstrapServers = "localhost:9092"; // Default to Kubernetes service name
       }

       Properties props = new Properties();
       props.setProperty(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, bootstrapServers);
       props.setProperty(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG, StringSerializer.class.getName());
       props.setProperty(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG,
StringSerializer.class.getName());
       System.out.println("Connecting to Kafka at: " + bootstrapServers);

       ObjectMapper objectMapper = new ObjectMapper();
       try(Producer<String,String> producer = new KafkaProducer<>(props)){
           System.out.println("Weather station " + stationId + " started. Sending data to Kafka...");

           while(true){
               if (dropMsg()){
                   statusMsgCounter++;
```

```
                    System.out.println("Message dropped");
                    continue;
                }
                WeatherStationMsg msg = createWeatherStationMsg(stationId, statusMsgCounter++);
                try {
                    String jsonMsg = objectMapper.writeValueAsString(msg);
                    ProducerRecord<String,String> record = new ProducerRecord<>("weather-station",
String.valueOf(stationId), jsonMsg);
                    producer.send(record, (metadata, exception) -> {
                        if (exception != null) {
                            System.out.println("Error sending message: " + exception.getMessage());
                        } else {
                            System.out.println("Message sent to topic " + metadata.topic() + " partition
" + metadata.partition() + " offset " + metadata.offset());
                        }
                    });
                } catch (JsonProcessingException e) {
                    System.out.println("message to json error");
                    throw new RuntimeException(e);
                }
                Thread.sleep(1000);
            }
        }
    }
    private static boolean dropMsg(){
        return rand.nextInt(100) < 10; // 10% chance to drop the message
    }
    private static WeatherStationMsg createWeatherStationMsg(long stationId, long statusMsgCounter) {

        int temperature = rand.nextInt(40,100) ;
        int humidity = rand.nextInt(101);
        int windSpeed = rand.nextInt(26);

        WeatherData weatherData = new WeatherData(temperature, humidity, windSpeed);
        int batteryLevel = rand.nextInt(101);

        BatteryStatus batteryStatus ;
        if (batteryLevel>=70){
            batteryStatus = BatteryStatus.HIGH;
        } else if (batteryLevel>=30){
            batteryStatus = BatteryStatus.MEDIUM;
        } else {
            batteryStatus = BatteryStatus.LOW;
        }
        return new WeatherStationMsg(stationId, statusMsgCounter, batteryStatus, weatherData);
```

```
    }
}
```

Model classes for message creation and serialization to json:

```
package org.example;

import java.time.Instant;

public class WeatherStationMsg {
    private long station_id;
    private long s_no;//auto increment
    private BatteryStatus battery_status;
    private long status_timestamp;
    private WeatherData weather;

    public WeatherStationMsg(long stationId,long s_no,BatteryStatus batteryStatus, WeatherData
weatherData) {
        this.station_id = stationId;
        this.s_no = s_no;
        this.battery_status = batteryStatus;
        this.status_timestamp = Instant.now().getEpochSecond();
        this.weather = weatherData;
    }

    public long getStation_id() {
        return station_id;
    }

    public void setStation_id(long station_id) {
        this.station_id = station_id;
    }

    public long getS_no() {
        return s_no;
    }

    public void setS_no(long s_no) {
        this.s_no = s_no;
    }

    public BatteryStatus getBattery_status() {
        return battery_status;
```

```
    }

    public void setBattery_status(BatteryStatus battery_status) {
        this.battery_status = battery_status;
    }

    public long getStatus_timestamp() {
        return status_timestamp;
    }

    public void setStatus_timestamp(long status_timestamp) {
        this.status_timestamp = status_timestamp;
    }

    public WeatherData getWeather() {
        return weather;
    }

    public void setWeather(WeatherData weather) {
        this.weather = weather;
    }
}
```

```
package org.example;

public class WeatherData {
    private final int humidity;
    private final int temperature;
    private final int wind_speed;

    public WeatherData(int temperature, int humidity, int windSpeed) {
        this.temperature = temperature;
        this.humidity = humidity;
        this.wind_speed = windSpeed;
    }

    public int getHumidity() {
        return humidity;
    }

    public int getTemperature() {
        return temperature;
    }
```

```
    public int getWind_speed() {
        return wind_speed;
    }
}
```

```
package org.example;

public enum BatteryStatus {
    HIGH,
    MEDIUM,
    LOW
}
```

Dockerfile to create weather station image:

```
FROM eclipse-temurin:21-jdk-alpine

WORKDIR /app

# Copy the pre-built JAR file from your local filesystem
COPY target/weather_station-1.0-SNAPSHOT-jar-with-dependencies.jar app.jar

# Default environment variables
ENV KAFKA_BOOTSTRAP_SERVERS=kafka:9092
ENV STATION_ID=1

CMD ["java", "-jar", "app.jar"]
```

Central station:

1) Kafka message to bitcask

```
public class KafkaToBitcask {

   public static void main(String[] args) {

       String KAFKA_BOOTSTRAP_SERVERS = System.getenv("KAFKA_BOOTSTRAP_SERVERS");
       if (KAFKA_BOOTSTRAP_SERVERS == null) {
           System.err.println("Environment variable KAFKA_BOOTSTRAP_SERVERS is not set.");
           System.exit(1);
       }
       String KAFKA_TOPIC = System.getenv("KAFKA_TOPIC");
       if (KAFKA_TOPIC == null) {
           System.err.println("Environment variable KAFKA_TOPIC is not set.");
           System.exit(1);
       }

       String BITCASK_SERVER_HOST = System.getenv("BITCASK_SERVER_HOST");
       if (BITCASK_SERVER_HOST == null) {
           System.err.println("Environment variable BITCASK_SERVER_HOST is not set.");
           System.exit(1);
       }

       String BITCASK_SERVER_PORT_str = System.getenv("BITCASK_SERVER_PORT");
       if (BITCASK_SERVER_PORT_str == null) {
           System.err.println("Environment variable BITCASK_SERVER_PORT is not set.");
           System.exit(1);
       }
       int BITCASK_SERVER_PORT = Integer.parseInt(BITCASK_SERVER_PORT_str);
       // Check if the environment variables are set
       if (KAFKA_BOOTSTRAP_SERVERS.isEmpty() || BITCASK_SERVER_HOST.isEmpty() ||
BITCASK_SERVER_PORT_str.isEmpty()) {
           System.err.println("Environment variables are not set.");
           System.exit(1);
       }
       // Kafka Streams Configuration
       Properties props = new Properties();
       props.put(StreamsConfig.APPLICATION_ID_CONFIG, "message-normailzer");
       props.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG, KAFKA_BOOTSTRAP_SERVERS);
       props.put(StreamsConfig.DEFAULT_KEY_SERDE_CLASS_CONFIG, Serdes.String().getClass());
       props.put(StreamsConfig.DEFAULT_VALUE_SERDE_CLASS_CONFIG, Serdes.String().getClass());
```

```
        // Define Stream Processing
        StreamsBuilder builder = new StreamsBuilder();
        KStream<String, String> stream = builder.stream(KAFKA_TOPIC);

        stream.foreach((key, value) -> {
            try {
                Long id = Long.parseLong(key);

                // Send to Bitcask server via TCP
                sendToBitcask(id, value, BITCASK_SERVER_HOST, BITCASK_SERVER_PORT);  // change host/port
as needed
            } catch (Exception e) {
                e.printStackTrace();
            }
        });

        KafkaStreams streams = new KafkaStreams(builder.build(), props);
        streams.start();

        Runtime.getRuntime().addShutdownHook(new Thread(streams::close));
    }

    public static void sendToBitcask(Long key, String value, String host, int port) throws IOException {
        try (Socket socket = new Socket(host, port);
             PrintWriter writer = new PrintWriter(
                     new OutputStreamWriter(socket.getOutputStream(), StandardCharsets.UTF_8), true)){

            value = value.replace(',',';');
            String request = "w" + " " + key + " " + value + "--no-reply";
            writer.println(request);

        }  catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

## 2) Rain-detector:

```
public class Main {
   public static void main(String[] args) {

       String KAFKA_BOOTSTRAP_SERVERS = System.getenv("KAFKA_BOOTSTRAP_SERVERS");
       if (KAFKA_BOOTSTRAP_SERVERS == null) {
           KAFKA_BOOTSTRAP_SERVERS = "localhost:9092";
       }


       Properties props = new Properties();
       props.setProperty(StreamsConfig.APPLICATION_ID_CONFIG,"rain-detector");
       props.setProperty(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG,KAFKA_BOOTSTRAP_SERVERS);

props.setProperty(StreamsConfig.DEFAULT_KEY_SERDE_CLASS_CONFIG,Serdes.String().getClass().getName());

props.setProperty(StreamsConfig.DEFAULT_VALUE_SERDE_CLASS_CONFIG,Serdes.String().getClass().getName());


       ObjectMapper objMapper = new ObjectMapper();

       StreamsBuilder builder = new StreamsBuilder();

       KStream<String, String> inputStream = builder.stream("weather-station");

       KStream<String,String> rainStream = inputStream.filter((key, value) -> {
           try {
               JsonNode jsonNode = objMapper.readTree(value);
               int humidity = jsonNode.get("weather").get("humidity").asInt();
               return humidity >= 70;
           } catch (Exception e) {
               e.printStackTrace();
               return false;
           }
       }).mapValues((value) ->{
           try{

               JsonNode jsNode = objMapper.readTree(value);
               int stationId = jsNode.get("station_id").asInt();
               int humidity = jsNode.get("weather").get("humidity").asInt();
               String s = "Rain detected at station ID: " + stationId + " at humidity level: " +
humidity;
               System.out.println(s);
               return s;
```

```
            }
            catch(Exception e){
                e.printStackTrace();
                return "Error processing message";
            }
        });

        rainStream.to("rain-alerts");

        KafkaStreams streams = new KafkaStreams(builder.build(), props);
        streams.start();
        Runtime.getRuntime().addShutdownHook(new Thread(streams::close));
    }
}
```

## 3) Parquet-maker:

```
public class KafkaToParquetWriter {

   private final String bootstrapServers;
   private final String topic;
   private final String groupId;
   private final String outputDir;
   private final long maxFileSizeBytes;
   private final int batchSize;
   private final Schema schema;
   private long estimatedRecordSize;
   private final AtomicBoolean running = new AtomicBoolean(true);

   // Keep track of active writers and their current sizes
   private final Map<String, WriterInfo> activeWriters = new ConcurrentHashMap<>();
   private final Map<String, Long> lastActivityTimestamp = new ConcurrentHashMap<>();
   private final Map<String, List<GenericRecord>> recordBatches = new ConcurrentHashMap<>();
   private final long inactivityThresholdMs;

   // Track file sequence numbers for each partition
   private final Map<String, Integer> partitionFileCounters = new ConcurrentHashMap<>();

   public KafkaToParquetWriter(String bootstrapServers, String topic, String groupId,
                               String outputDir, Schema schema,long maxFileSizeBytesMB, int batchSize) {
       this.bootstrapServers = bootstrapServers;
       this.topic = topic;
       this.groupId = groupId;
       this.outputDir = outputDir;
       this.schema = schema;
       this.maxFileSizeBytes = maxFileSizeBytesMB * 1024 * 1024; // Not used with batch-per-file
approach
       this.batchSize = batchSize; // Each batch gets its own file
       this.estimatedRecordSize = -1; // Initialize estimated record size
       this.inactivityThresholdMs = 300000; // 5 minutes inactivity threshold

       // Create output directory if it doesn't exist
       File directory = new File(outputDir);
       if (!directory.exists()) {
           directory.mkdirs();
       }
   }

   public void start() {
       // Create Kafka consumer
```

```
        Properties props = new Properties();
        props.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, bootstrapServers);
        props.put(ConsumerConfig.GROUP_ID_CONFIG, groupId);
        props.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG, StringDeserializer.class.getName());
        props.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG, StringDeserializer.class.getName());
        props.put(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG, "earliest");

        // Start a timer to periodically check for inactive partitions
        Timer inactivityTimer = new Timer("InactivityChecker", true);
        inactivityTimer.scheduleAtFixedRate(new TimerTask() {
            @Override
            public void run() {
                checkInactivePartitions();
            }
        }, 300000, 300000); // Check every 5 min

        try (KafkaConsumer<String, String> consumer = new KafkaConsumer<>(props)) {
            consumer.subscribe(Collections.singletonList(topic));

            // Main polling loop
            // Main polling loop
            while (running.get()) {
                ConsumerRecords<String, String> records = consumer.poll(Duration.ofMillis(100));
                boolean newMsgs = false;

                for (ConsumerRecord<String, String> record : records) {
                    try {
                        newMsgs = true;
                        // Parse the JSON message
                        JSONObject json = new JSONObject(record.value());

                        // Extract timestamp and station ID
                        String stationIdStr = json.optString("station_id");

                        // Handle timestamp - could be either Unix timestamp (long) or ISO formatted
string
                        LocalDateTime dateTime;
                        if (json.has("status_timestamp")) {
                            Object timestampObj = json.get("status_timestamp");
                            if (timestampObj instanceof Long || timestampObj instanceof Integer) {
                                // Handle numeric timestamp (Unix timestamp in seconds)
                                long timestamp = json.getLong("status_timestamp");
                                dateTime = LocalDateTime.ofEpochSecond(timestamp, 0,
java.time.ZoneOffset.UTC);
                            } else {
```

```
                            // Handle string timestamp in ISO format
                            String timestamp = json.getString("status_timestamp");
                            try {
                                dateTime = LocalDateTime.parse(timestamp,
DateTimeFormatter.ISO_DATE_TIME);
                            } catch (Exception e) {
                                // Fallback - use current time if parsing fails
                                System.err.println("Could not parse timestamp: " + timestamp + ",
using current time");
                                dateTime = LocalDateTime.now();
                            }
                        }
                    } else {
                        // No timestamp in the message, use current time
                        System.err.println("Message missing timestamp, using current time");
                        dateTime = LocalDateTime.now();
                    }

                    if (stationIdStr.isEmpty()) {
                        System.err.println("Message missing station_id field: " + record.value());
                        continue;
                    }

                    String datePath = dateTime.format(DateTimeFormatter.ofPattern("yyyy/MM/dd"));

                    // Swap the partitioning order: stationId/datePath instead of datePath/stationId
                    String partitionKey = stationIdStr + "/" + datePath.replace("/", "_");

                    // Update last activity timestamp for this partition
                    lastActivityTimestamp.put(partitionKey, System.currentTimeMillis());

                    // Add to appropriate batch
                    recordBatches.computeIfAbsent(partitionKey, k -> new ArrayList<>())
                        .add(convertJsonToAvro(json));

                    // Process batch if it reaches the batch size
                    List<GenericRecord> batch = recordBatches.get(partitionKey);
                    if (batch.size() >= batchSize) {
                        writeToParquet(partitionKey, batch);
                        recordBatches.put(partitionKey, new ArrayList<>());
                    }
                } catch (Exception e) {
                    System.err.println("Error processing record: " + e.getMessage());
                    e.printStackTrace();
                }
```

```
            }
            if (newMsgs){
                consumer.commitSync();
            }
        }

        // Clean up timer when shutting down
        inactivityTimer.cancel();

        // Close all active writers when shutting down
        closeAllWriters();
    } catch (Exception e) {
        System.err.println("Error in Kafka consumer: " + e.getMessage());
        e.printStackTrace();
    }
}

private GenericRecord convertJsonToAvro(JSONObject json) {
    GenericRecord avroRecord = new GenericData.Record(schema);

    // Map JSON fields to Avro schema fields - handle nested structures
    for (Schema.Field field : schema.getFields()) {
        String fieldName = field.name();
        if (json.has(fieldName)) {
            if (fieldName.equals("weather") && !json.isNull("weather")) {
                // Handle nested weather object
                JSONObject weatherJson = json.getJSONObject("weather");
                GenericRecord weatherRecord = new GenericData.Record(field.schema());

                // Map weather fields
                for (Schema.Field weatherField : field.schema().getFields()) {
                    String weatherFieldName = weatherField.name();
                    if (weatherJson.has(weatherFieldName)) {
                        weatherRecord.put(weatherFieldName, weatherJson.get(weatherFieldName));
                    }
                }

                avroRecord.put(fieldName, weatherRecord);
            } else {
                // Handle regular fields
                avroRecord.put(fieldName, json.get(fieldName));
            }
        }
    }
```

```
        return avroRecord;
    }

    private synchronized void writeToParquet(String partitionKey, List<GenericRecord> records) throws
IOException {
        if (records.isEmpty()) {
            return;
        }

        // Get and increment the file counter for this partition
        int fileId = partitionFileCounters.compute(partitionKey, (key, count) -> count == null ? 1 :
count + 1);

        // Create a new file name for this batch
        String fileName = "part-" + String.format("%06d", fileId) + ".parquet";
        String filePath = outputDir + "/" + partitionKey + "/" + fileName;

        // Ensure directory exists
        File directory = new File(new File(filePath).getParent());
        if (!directory.exists()) {
            directory.mkdirs();
        }

        // Create a new writer for this batch
        ParquetWriter<GenericRecord> writer = AvroParquetWriter
            .<GenericRecord>builder(new Path(filePath))
            .withSchema(schema)
            .withCompressionCodec(CompressionCodecName.UNCOMPRESSED)
            .build();

        // Write all records in the batch
        long batchSize = 0;
        for (GenericRecord record : records) {
            writer.write(record);
            if (estimatedRecordSize == -1) {
                estimatedRecordSize = estimateRecordSize(record);
            }
            batchSize += estimatedRecordSize;
        }

        // Close the writer immediately after writing the batch
        writer.close();

        System.out.println("Wrote batch of " + records.size() + " records to " + filePath +
            ", estimated size: " + (batchSize / (1024 * 1024)) + " MB");
```

```
    }

    private long estimateRecordSize(GenericRecord record) {
        // Estimate the size of the record by summing the sizes of its fields
        long size = 0;
        for (Schema.Field field : record.getSchema().getFields()) {
            Object value = record.get(field.name());
            if (value != null) {
                if (value instanceof CharSequence) {
                    size += ((CharSequence) value).length() * 2; // Approximate size of a string
                } else if (value instanceof Integer) {
                    size += Integer.BYTES;
                } else if (value instanceof Long) {
                    size += Long.BYTES;
                } else if (value instanceof Float) {
                    size += Float.BYTES;
                } else if (value instanceof Double) {
                    size += Double.BYTES;
                } else if (value instanceof GenericRecord) {
                    size += estimateRecordSize((GenericRecord) value); // Recursively estimate nested
records
                }
            }
        }
        return size;
    }

    private synchronized void closeAllWriters() {
        // Write any remaining batches when shutting down
        for (Map.Entry<String, List<GenericRecord>> entry : recordBatches.entrySet()) {
            if (!entry.getValue().isEmpty()) {
                try {
                    writeToParquet(entry.getKey(), entry.getValue());
                } catch (IOException e) {
                    System.err.println("Error writing final batch for " + entry.getKey() + ": " +
e.getMessage());
                }
            }
        }
        recordBatches.clear();
    }

    public void shutdown() {
        running.set(false);
    }
```

```
    /**
     * Checks for partitions that haven't received data for a while
     * and flushes any pending records to disk
     */
    private void checkInactivePartitions() {
        long currentTime = System.currentTimeMillis();
        Set<String> partitionsToCheck = new HashSet<>(lastActivityTimestamp.keySet());

        for (String partitionKey : partitionsToCheck) {
            Long lastActivity = lastActivityTimestamp.get(partitionKey);
            if (lastActivity != null && (currentTime - lastActivity) > inactivityThresholdMs) {
                try {
                    System.out.println("Partition " + partitionKey + " inactive for " +
                        (currentTime - lastActivity) / 1000 + " seconds. Flushing remaining records.");

                    // Get buffered records for this partition
                    List<GenericRecord> records = recordBatches.get(partitionKey);
                    if (records != null && !records.isEmpty()) {
                        writeToParquet(partitionKey, records);
                        recordBatches.put(partitionKey, new ArrayList<>());
                    }

                    // Remove from activity tracking
                    lastActivityTimestamp.remove(partitionKey);
                } catch (IOException e) {
                    System.err.println("Error flushing inactive partition " + partitionKey + ": " +
e.getMessage());
                }
            }
        }
    }

    // Helper class to track writer and file size
    private static class WriterInfo {
        private final ParquetWriter<GenericRecord> writer;
        private long currentSize;
        private final String filePath;

        public WriterInfo(ParquetWriter<GenericRecord> writer, long initialSize, String filePath) {
            this.writer = writer;
            this.currentSize = initialSize;
            this.filePath = filePath;
        }
```

```
    public ParquetWriter<GenericRecord> getWriter() {
        return writer;
    }

    public long getCurrentSize() {
        return currentSize;
    }

    public void incrementSize(long bytes) {
        currentSize += bytes;
    }

    public String getFilePath() {
        return filePath;
    }
}

// Example main method to demonstrate usage
public static void main(String[] args) {
    // Updated schema definition to match the new message format
    String schemaJson = "{\"type\":\"record\",\"name\":\"WeatherStationRecord\",\"fields\":["
        + "{\"name\":\"s_no\",\"type\":\"long\"},"
        + "{\"name\":\"battery_status\",\"type\":\"string\"},"
        + "{\"name\":\"status_timestamp\",\"type\":\"long\"},"
        + "{\"name\":\"weather\",\"type\":{\"type\":\"record\",\"name\":\"WeatherData\",\"fields\":["
        + "  {\"name\":\"humidity\",\"type\":\"int\"},"
        + "  {\"name\":\"temperature\",\"type\":\"int\"},"
        + "  {\"name\":\"wind_speed\",\"type\":\"int\"}"
        + "]}}]}";

    Schema schema = new Schema.Parser().parse(schemaJson);

    String bootStrapServer = System.getenv("BOOTSTRAP_SERVER");
    if (bootStrapServer == null) {
        System.err.println("BOOTSTRAP_SERVER environment variable is not set.");
        bootStrapServer = "localhost:9092"; // Default value
    }
    String topic = System.getenv("TOPIC");
    if (topic == null) {
        System.err.println("TOPIC environment variable is not set.");
        topic = "weather-station"; // Default value
    }
    String outputDir = System.getenv("OUTPUT_DIR");
    if (outputDir == null) {
        System.err.println("OUTPUT_DIR environment variable is not set.");
```

```
            outputDir = "/out/weather-data"; // Default value
        }
        String maxFileSizeMB = System.getenv("MAX_FILE_SIZE_MB");
        long maxFileSizeBytesMB = 1024; // Default to 1GB
        if (maxFileSizeMB != null) {
            try {
                maxFileSizeBytesMB = Long.parseLong(maxFileSizeMB);
            } catch (NumberFormatException e) {
                System.err.println("Invalid MAX_FILE_SIZE_MB value, using default 1GB.");
            }
        }
        String batchSizeStr = System.getenv("BATCH_SIZE");
        int batchSize = 1000; // Default to 10K records per batch
        if (batchSizeStr != null) {
            try {
                batchSize = Integer.parseInt(batchSizeStr);
            } catch (NumberFormatException e) {
                System.err.println("Invalid BATCH_SIZE value, using default 10K.");
            }
        }


        KafkaToParquetWriter writer = new KafkaToParquetWriter(
            bootStrapServer,
            topic,
            "parquet-writer-group",
            outputDir,
            schema,
            maxFileSizeBytesMB,
            batchSize
        );

        // Add shutdown hook
        Runtime.getRuntime().addShutdownHook(new Thread(writer::shutdown));

        // Start processing
        writer.start();
    }
}
```

Docker file for each module:

1) Kafka-to-bitcask:

```
# Dockerfile for Normalizer
FROM eclipse-temurin:21-jdk

WORKDIR /app

COPY target/Normalizer-1.0-SNAPSHOT.jar /app/normalizer.jar

RUN mkdir -p /app/jfr-recordings

ENV KAFKA_BOOTSTRAP_SERVERS=kafka:9092
ENV KAFKA_TOPIC=weather-station
ENV BITCASK_SERVER_HOST=bitcask
ENV BITCASK_SERVER_PORT=5000


CMD ["java", "-XX:+FlightRecorder",
"-XX:StartFlightRecording=duration=1m,filename=/app/jfr-recordings/recording.jfr,settings=profile", "-jar", "normalizer.jar"]
```

2) Rain-detector:

```
FROM eclipse-temurin:21-jdk

WORKDIR /app

# Copy the pre-built JAR file from your local machine to the container
COPY target/raining_detector-1.0-SNAPSHOT-jar-with-dependencies.jar .

ENV KAFKA_BOOTSTRAP_SERVERS=kafka:9092

# Create directory for JFR recordings
RUN mkdir -p /app/jfr-recordings

# Run the application with a one-minute JFR recording
CMD ["java", "-XX:+FlightRecorder",
"-XX:StartFlightRecording=duration=1m,filename=/app/jfr-recordings/recording.jfr,settings=profile","-jar","raining_detector-1.0-SNAPSHOT-jar-with-dependencies.jar"]
```

3) Parquet-maker:

```
FROM eclipse-temurin:21-jdk

WORKDIR /app

COPY target/parquet-maker-1.0-SNAPSHOT-jar-with-dependencies.jar
/app/parquet-maker.jar

ENV BOOTSTRAP_SERVER=kafka:9092
ENV TOPIC=weather-station
ENV OUTPUT_DIR=/app/output
ENV MAX_FILE_SIZE_MB=1024
ENV BATCH_SIZE=100

# Create directory for JFR recordings
RUN mkdir -p /app/jfr-recordings

# Enable Java Flight Recorder with continuous recording
# Start with JFR recording to file with 1-hour chunks, rotating up to 5 files
CMD ["java", \
    "-XX:+FlightRecorder", \

"-XX:StartFlightRecording=duration=1m,filename=/app/jfr-recordings/recording.jf
r,settings=profile", \
    "-jar", "parquet-maker.jar"]
```

---

Kubernetes yaml config files:

1) Bitcask deployment and service:

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: bitcask
spec:
 replicas: 1
 selector:
   matchLabels:
     app: bitcask
 template:
   metadata:
     labels:
       app: bitcask
   spec:
     initContainers:
     - name: wait-for-pvc
       image: busybox:latest
       command: ['sh', '-c', 'until ls -la /data && touch /data/test-file && rm /data/test-file; do echo
"Waiting for PVC to be mounted..."; sleep 2; done; mkdir -p /data/logs /data/hints;echo "PVC is mounted
and writable!"']
       volumeMounts:
       - name: bitcask-data
         mountPath: /data
     containers:
     - name: bitcask
       image: bitcask:latest
       imagePullPolicy: Never
       ports:
       - containerPort: 5000
       volumeMounts:
       - name: bitcask-data
         mountPath: /data
     volumes:
     - name: bitcask-data
       persistentVolumeClaim:
         claimName: bitcask-pvc
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
```

```
 name: bitcask-pvc
spec:
 accessModes:
   - ReadWriteOnce
 resources:
   requests:
     storage: 1Gi
 storageClassName: standard
---
apiVersion: v1
kind: Service
metadata:
 name: bitcask
 labels:
   app: bitcask
spec:
 type: NodePort
 selector:
   app: bitcask
 ports:
 - protocol: TCP
   port: 5000
   targetPort: 5000
   name: bitcask-port
---
```

2) Kafka deployment and service:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
 name: kafka
spec:
 replicas: 1
 selector:
   matchLabels:
     app: kafka
 template:
   metadata:
     labels:
       app: kafka
   spec:
     containers:
     - name: kafka
       image: bitnami/kafka:latest
       resources:
         requests:
           memory: "1Gi"
           cpu: "500m"
         limits:
           memory: "2Gi"
           cpu: "1000m"
       env:
       - name: KAFKA_CFG_AUTO_CREATE_TOPICS_ENABLE
         value: "true"
       - name: KAFKA_CFG_LISTENER_SECURITY_PROTOCOL_MAP
         value: "INTERNAL:PLAINTEXT,EXTERNAL:PLAINTEXT,CONTROLLER:PLAINTEXT"
       - name: KAFKA_CFG_CONTROLLER_LISTENER_NAMES
         value: "CONTROLLER"
       - name: KAFKA_KRAFT_CLUSTER_ID
         value: "kimokono"
       - name: ALLOW_PLAINTEXT_LISTENER
         value: "yes"
       - name: KAFKA_CFG_LISTENERS
         value: "INTERNAL://:9092,EXTERNAL://:29092,CONTROLLER://:9093"
       - name: KAFKA_CFG_CONTROLLER_QUORUM_VOTERS
         value: "0@kafka:9093"
       - name: KAFKA_CFG_ADVERTISED_LISTENERS
         value: "INTERNAL://kafka:9092,EXTERNAL://localhost:29092"
       - name: KAFKA_ENABLE_KRAFT
         value: "yes"
```

```
        - name: KAFKA_CFG_PROCESS_ROLES
          value: "broker,controller"
        - name: KAFKA_CFG_INTER_BROKER_LISTENER_NAME
          value: "INTERNAL"
        - name: KAFKA_CFG_NODE_ID
          value: "0"
        ports:
        - containerPort: 9092
        - containerPort: 9093
        - containerPort: 29092
        volumeMounts:
        - name: kafka-data
          mountPath: /bitnami/kafka
      volumes:
      - name: kafka-data
        persistentVolumeClaim:
          claimName: kafka-pvc
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
 name: kafka-pvc
spec:
 accessModes:
   - ReadWriteOnce
 resources:
   requests:
     storage: 10Gi
 storageClassName: standard
---
apiVersion: v1
kind: Service
metadata:
 name: kafka
spec:
 type: NodePort
 ports:
   - port: 9092
     targetPort: 9092
     name:  internal
   - port: 9093
     targetPort: 9093
     name: controller
   - port: 29092
     targetPort: 29092
```

```
    name: external
selector:
  app: kafka
```

3) bitcask message normailzer deployment:

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: bitcask-normalizer
spec:
 selector:
   matchLabels:
     app: bitcask-normalizer
 template:
   metadata:
     labels:
       app: bitcask-normalizer
   spec:
     containers:
     - name: bitcask-normalizer
       image: bitcask-normalizer:latest
       imagePullPolicy: Never
       env:
       - name: KAFKA_BOOTSTRAP_SERVERS
         value: "kafka:9092"
       - name: KAFKA_TOPIC
         value: "weather-station"
       - name: BITCASK_SERVER_HOST
         value: "bitcask"
       - name: BITCASK_SERVER_PORT
         value: "5000"
```

4) parquet-maker deployment:

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: parquet-maker
spec:
 selector:
   matchLabels:
     app: parquet-maker
 template:
   metadata:
     labels:
       app: parquet-maker
   spec:
     initContainers:
     - name: init-parquet-maker
       image: busybox
       command: ['sh', '-c', 'until ls -la /app/output; do echo "Waiting for persistent storage to be
ready..."; sleep 2; done;']
       volumeMounts:
       - name: parquet-data
         mountPath: /app/output
     containers:
     - name: parquet-maker
       image: parquet-maker:latest
       imagePullPolicy: Never
       env:
       - name: BOOTSTRAP_SERVER
         value: "kafka:9092"
       - name: TOPIC
         value: "weather-station"
       - name: OUTPUT_DIR
         value: "/app/output"
       - name: MAX_FILE_SIZE_MB
         value: "1"
       - name: BATCH_SIZE
         value: "1000"
       volumeMounts:
       - name: parquet-data
         mountPath: /app/output
      volumes:
     - name: parquet-data
       persistentVolumeClaim:
         claimName: parquet-data-pvc
```

```
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
 name: parquet-data-pvc
spec:
 accessModes:
 - ReadWriteOnce
 resources:
   requests:
     storage: 10Gi
 storageClassName: standard
---
```

## 5) Rain detector deployment:

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: rain-detector
spec:
 replicas: 1
 selector:
   matchLabels:
     app: rain-detector
 template:
   metadata:
     labels:
       app: rain-detector
   spec:
     containers:
     - name: rain-detector
       image: rain-detector:latest
       imagePullPolicy: Never
       env:
       - name: KAFKA_BOOTSTRAP_SERVERS
         value: "kafka:9092"
```

## 6) Weather-station statful-set:

Used statefulSet mainely for the naming of the pods only as weather stations don’t need to be stateful or keep any data

```yaml
apiVersion: apps/v1
kind: StatefulSet
metadata:
 name: weather-station
spec:
 serviceName: "weather-station"
 replicas: 10
 selector:
   matchLabels:
     app: weather-station
 template:
   metadata:
     labels:
       app: weather-station
   spec:
     containers:
     - name: weather-station
       image: weather-station:latest
       imagePullPolicy: Never
       env:
       - name: KAFKA_BOOTSTRAP_SERVERS
         value: "kafka:9092"
       - name: STATION_ID
         valueFrom:
           fieldRef:
             fieldPath: metadata.name
---
apiVersion: v1
kind: Service
metadata:
 name: weather-station
 labels:
   app: weather-station
spec:
 clusterIP: None
 selector:
   app: weather-station
```

7) elasticsearch and kibana deployment:

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: elasticsearch
spec:
 replicas: 1
 selector:
   matchLabels:
     app: elasticsearch
 template:
   metadata:
     labels:
       app: elasticsearch
   spec:
     initContainers:
     - name: init-sysctl
       image: busybox:latest
       command:
       - sysctl
       - -w
       - vm.max_map_count=262144
       securityContext:
         privileged: true
     containers:
     - name: elasticsearch
       image: nshou/elasticsearch-kibana
       resources:
         requests:
           memory: "1Gi"
           cpu: "500m"
         limits:
           memory: "2Gi"
           cpu: "1000m"
       ports:
       - containerPort: 9200
       - containerPort: 5601
       env:
       - name: SSL_MODE
         value: "false"
       - name: ES_JAVA_OPTS
         value: "-Xms512m -Xmx512m"
       - name: discovery.type
         value: "single-node"
```

```yaml
---
apiVersion: v1
kind: Service
metadata:
 name: elasticsearch
spec:
 type: NodePort
 ports:
 - port: 9200
   targetPort: 9200
   name:  elasticsearch
 - port: 5601
   targetPort: 5601
   name: kibana
 selector:
   app: elasticsearch
```

## 8) parquet-to-es cronJob:

Used to load newly added file to the persistent-volume of the parquet maker pod into elastic search after transforming data for the desired business needs.

```
apiVersion: batch/v1
kind: CronJob
metadata:
 name: parquet-to-elastic
spec:

 schedule: "*/30 * * * *" # Runs every half an hour
 jobTemplate:
   spec:
     template:
       spec:
         containers:
         - name: parquet-to-elastic
           image: parquet-to-es:latest
           imagePullPolicy: Never
           env:
           - name: PARQUET_DIR
             value: "/app/weather-data"
           - name: PROCESSED_FILES
             value: "/app/out/processed_files.txt"
           - name: ELASTIC_HOST
             value: "elasticsearch" # Elasticsearch service name
           - name: ELASTIC_PORT
             value: "9200"
           - name: ELASTIC_INDEX
             value: "weather_stations_metrics"
           volumeMounts:
           - name: parquet-data
             mountPath: "/app/weather-data"
           - name: processed-files
             mountPath: "/app/out"
         restartPolicy: OnFailure
         volumes:
         - name: parquet-data
           persistentVolumeClaim:
             claimName: parquet-data-pvc
         - name: processed-files
           persistentVolumeClaim:
             claimName: processed-files-pvc
---
apiVersion: v1
```

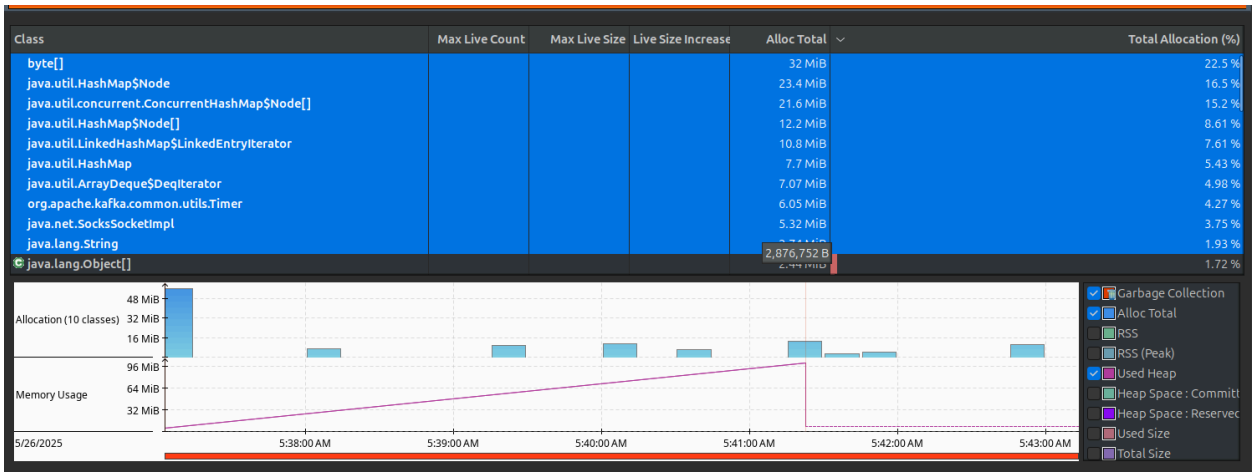
```yaml
kind: PersistentVolumeClaim
metadata:
 name: processed-files-pvc
spec:
 accessModes:
 - ReadWriteOnce
 resources:
   requests:
     storage: 1Gi # Adjust size as needed
 storageClassName: standard
---
```

# Profiling center station using JFR(java flight recorder)

The **Central station** consists of 3 main but separate modules ( message to bitcask normalizer + parquet-maker + rain-detector module) each module does one job and has its own consumer group in kafka ( they are connected to weather stations in pub-sub pattern )
Run was for nearly 3 minutes to ensure IO writes.

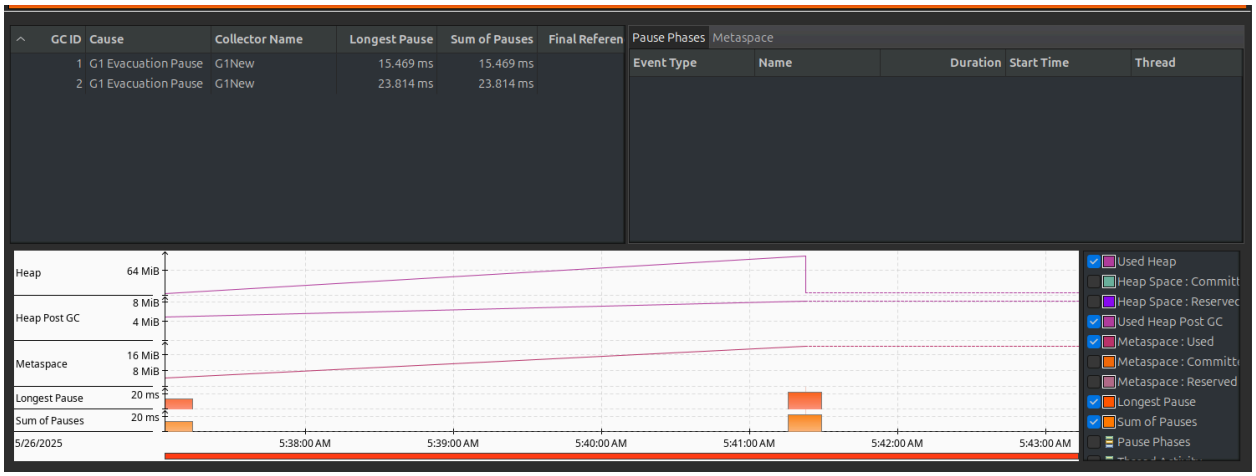
## Main points :

- Top 10 Classes with highest total memory
    1. byte[]
    2. java.util.HashMap$Node
    3. java.util.concurrent.ConcurrentHashMap$Node[]
    4. java.util.HashMap$Node[]
    5. java.util.LinkedHashMap$LinkedEntryIterator
    6. java.util.HashMap
    7. java.util.ArrayDeque$DeqIterator
    8. org.apache.kafka.common.utils.Timer
    9. java.net.SocksSocketImpl
    10. java.lang.String



● GC pauses count

**2 pause counts**

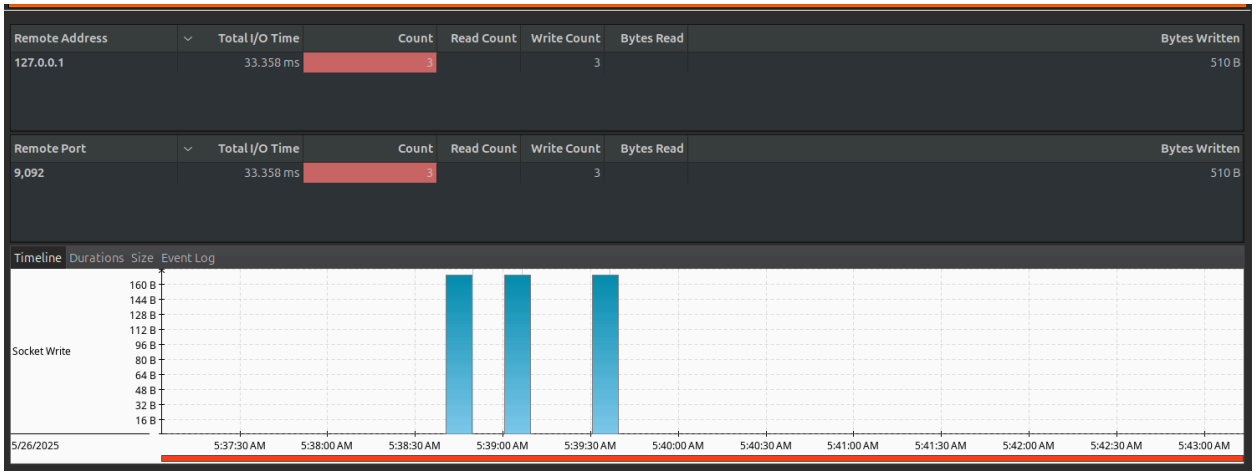



● GC maximum pause duration

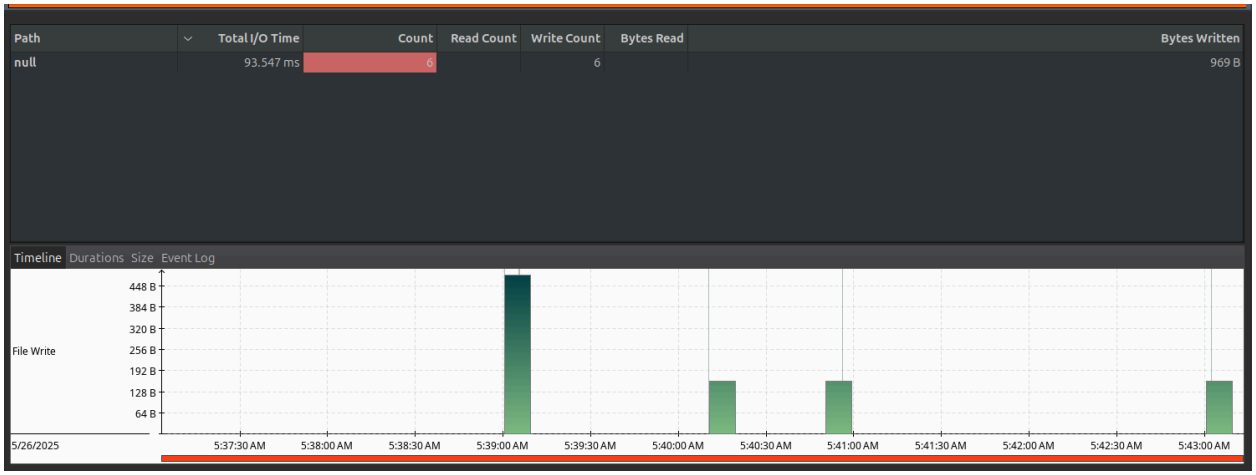
23.814 ms

● List of I/O operations

3 socket I/O operations each one takes the socket time of 10 ms sending messages



6 file I/O operations:



# Bonus:

## Integration patterns used in this project:

1) Channel adapter:
   A message normalizer module in the central station provides the capability for the bitcask server to poll message from the kafka topic and save the latest value in its file system
2) Invalid message channel:
   For modules of two module of the central station if a message has a missing format or a wrong json was passed inside the weather-station topic the module invalidates that message and passes it to another topic for invalid messages
3) Content enricher:
   Using a separate module that runs as a cronJob on k8s we modify the parquet files produced by parquet-maker to add some points for analytics (show the percentage of dropped messages)
4) Message Normalizer:
   If we can consider a message normalizer as the basic form of message translator then the message-to-bitcask module can be considered a message normalizer,since it can be extended to allow for other weather stations to send messages in different (should be supported) schema in the same chanel and it will be kept in bitcask
5) Content filter:
   The rain-detector module sends a more concise message through its channel where more modules can connect to it and use the filtered message
   Example rain-detector message:

   | Rain detected at station ID: 1 at humidity level: 76 |
   |---|

6) Polling consumer:
   The kafka consumer api acts as a polling consumer where it keeps receiving messages and answers to poll() call from receiver
7) Durable subscriber:
   We can argue that the the elastic-search pod may act as a subscriber to message summary that are kept in the form of .parquet files and more message will be produced even if the “subscriber” is not up