Course Project

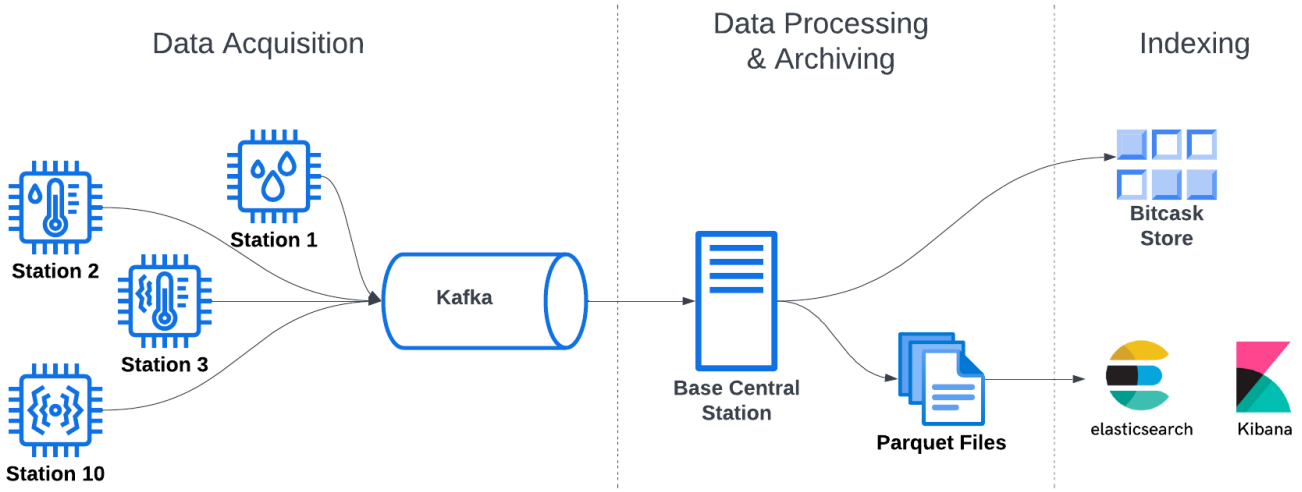
Weather Stations Monitoring

Team Members

Aya Khames Khairy	18010442
Pancee Wahid Mohamed	18010467
Mohamed Yasser Mohamed	18011648

Project Summary

The weather stations monitoring system shown in the figure was implemented in this project step by step.



A. Write Weather Station Mock

A weather station mock was implemented to output a status message every 1 second. The status message follows the provided schema:

```
{
   "station_id": 1, // Long
   "s_no": 1, // Long auto-incremental with each message per service
   "battery_status": "low", // String of (low, medium, high)
   "status_timestamp": 1681521224, // Long Unix timestamp
   "weather": {
       "humidity": 35, // Integer percentage
       "temperature": 100, // Integer in fahrenheit
       "wind_speed": 13, // Integer km/h
   }
}
```

The data in the status message is randomly changed. The timestamp is the timestamp at which the message is sent by the station. The battery status has 3 possible values: low, medium and high. They are randomly generated but following a specific distribution where from the messages sent by the weather station, 30% have the value “low”, 40% have the value “medium”, and 30% have the value “high”. Also, random 10% of the messages are dropped.

As will be shown later, 10 mock services of the weather station are run. All of them use the same class WeatherStation but an argument is passed on running which is the station id.

B. Setup Weather Station to Connect to Kafka

Java API is used to connect the weather station mock to the Kafka server. In the weather mock, after the message is generated as described before, it's published on a Kafka topic called "weather-messages".

The following command is used to see the messages published on this topic:

./bin/windows/kafka-console-consumer.bat --bootstrap-server 127.0.0.1:9092 --topic weather-messages --from-beginning

This is a sample of what is published on the topic:

Source Code:

There are two variants of the weather station on our repo. The first is WeatherStation which runs a single weather station. The second is WeatherStations which has two additional classes:

- StationsRunner: for running 10 weather station mocks as separate processes and saving the processes ids to be terminated when desired.
- StationsTerminator: for stopping the processes of weather station mocks.

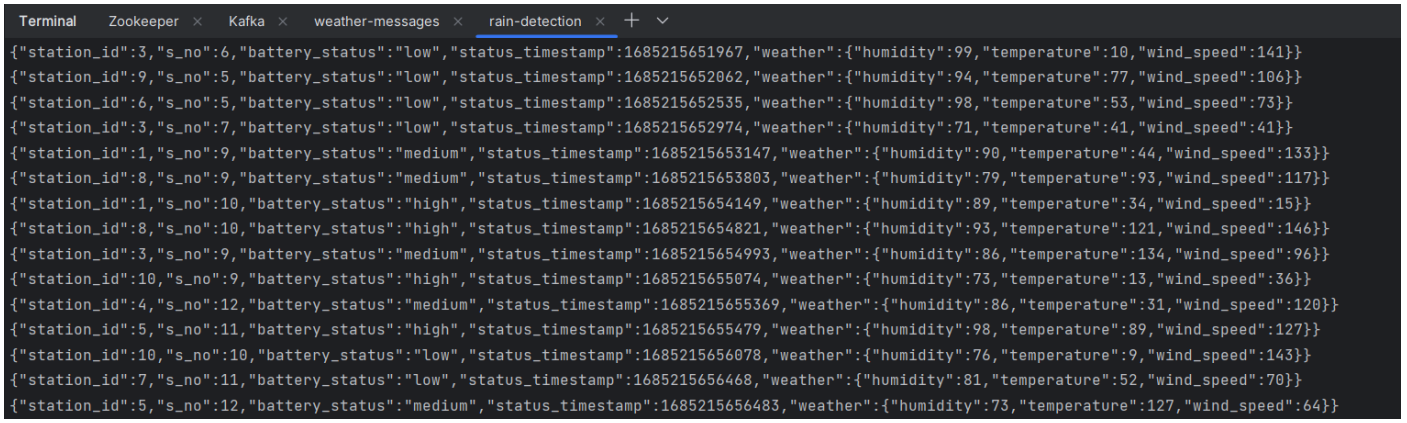
The code could be found here:

- Weather-Station-Monitoring/Weather-Station
- Weather-Station-Monitoring/Weather-Stations

C. Implement Raining Triggers in Kafka Processors

Rain Detector was implemented. It acts as both producer and consumer on Kafka. It subscribes to the “weather-messages” topic and checks the value of humidity in the consumed messages. If it’s greater than 70%, Rain Detector produces (publishes) the message on another topic “rain-detection” to indicate that it will probably rain.

This is a sample of what’s published on “rain-detection” topic:



Source Code:

The code for the Rain Detector could be found on our repo:

- [Weather-Station-Monitoring/Rain-Detector]

D. Implement Central Station

Central Station is organized as follows:

- BitCask: this class handles writing to BitCask LSM directory and all related functions.
- Archive: this class handles writing to Parquet files and all related functions.
- Central Station class: this class creates instances of BitCask and Archive and subscribes to the “weather-messages” Kafka topic. On consuming each message, it calls the append function of the both classes BitCask and Archive to handle adding the consumed message to both BitCask LSM directory and Parquet files.

There are 2 directories for storing the created logs:

- bitcask
- parquet-files

BitCask Riak to store updated view of weather status

A key-value store of the stations’ statuses was implemented. The implementation is of BitCask Riak LSM as described in the BitCask paper. However, a hint file is generated and kept for each generated log file to speed up merging.
In order not to disrupt active readers, all the files are merged first creating the merged log and hint files by only reading from the old log and hint files then the old files are deleted.

This is a video showing the described behavior, it can be found in the repo. For speeding up the testing time we set the maximum size of the log file to be 100KB and the maximum number of logs to keep before merging to be 5.

This is a sample BitCask LSM directory on writing the last file before merging:

Name	Size	Date modified	Type
hint_log_1685218210178.bin	1 KB	27-May-23 11:13 PM	BIN File
hint_log_1685218429728.bin	1 KB	27-May-23 11:14 PM	BIN File
hint_log_1685218492120.bin	1 KB	27-May-23 11:15 PM	BIN File
hint_log_1685218555764.bin	1 KB	27-May-23 11:16 PM	BIN File
log_1685218210178.bin	2 KB	27-May-23 11:13 PM	BIN File
log_1685218429728.bin	98 KB	27-May-23 11:14 PM	BIN File
log_1685218492120.bin	98 KB	27-May-23 11:15 PM	BIN File
log_1685218555764.bin	98 KB	27-May-23 11:16 PM	BIN File
log_1685218617803.bin	18 KB	27-May-23 11:17 PM	BIN File

This is the format by which data is written to both segment (log) and hint files:

Segment Record log_fileID.bin fileID = timestamp of creating the segment

Station_id	timestamp	value_size	message
Long = 8 bytes	Long = 8 bytes	Short = 2 bytes	value_size bytes

Hint File Record hint_fileID.bin

	Recent Location			
Station_id	timestamp	file_id	value_size	value_offset
Long = 8 bytes	Long = 8 bytes	Long = 8 bytes	Short = 2 bytes	Long = 8 bytes

Source Code:

The source code can be found in the project Github repository along with sample BitCask LSM directory and the video mentioned earlier.

- Weather-Station-Monitoring/Central-Station
- Weather-Station-Monitoring/Central-Station/bitcask
- Weather-Station-Monitoring/Central-Station/bitcask-directory-sample-video

Archiving of all Weather Statuses in Parquet Files

Parquet files are written to after receiving 10k records (i.e., in batches of 10k records). All files are stored in the parquet_files directory. They are organized and partitioned in this folder as follows:

- For each station, there is a directory with all Parquet files storing this station's messages.
- In each station directory, Parquet files are partitioned by time. Records are written to the Parquet file corresponding to its timestamp.
- I.e, if station 1 sent a message and the status_timestamp of this message after conversion to date-time format is 2023-05-20 13:45, this record will be stored in parquet_files/s1/s1__2023-05-20__13-45__p0.parquet
  The Parquet file name is divided as follows:
  - sn: refers to station n
  - yyyy-mm-dd: refers to the message date extracted from the timestamp
  - hh-mm: refers to the message time extracted from the timestamp
  - pn: refers to part n ⇒ the Parquet file is created when writing a batch but there may be records that belong to the same file in the next batch. Appending to parquet files in Java causes some problems, so another part of the file is created.
- Parquet files of each station are partitioned by time so that the records received within each 5 minutes are in a parquet file. I.e., time sequence in filenames is: 00, 05, 10, 15, .., 55.

This is a sample parquet file “parquet-files\s3\s3__2023-05-21__10-25__p0.parquet”:

	station_id	s_no	battery_status	status_timestamp	humidity	temperature	wind_speed
1	3	894	low	1684657500135	73	94	148
2	3	895	low	1684657501143	21	6	122
3	3	897	low	1684657502156	64	133	144
4	3	898	medium	1684657503168	50	6	38
5	3	899	medium	1684657504178	98	74	81
6	3	900	high	1684657505185	80	30	59
7	3	901	high	1684657506198	1	39	124
8	3	902	medium	1684657507200	10	7	73
9	3	903	low	1684657508205	86	141	39
10	3	904	medium	1684657509209	14	6	9

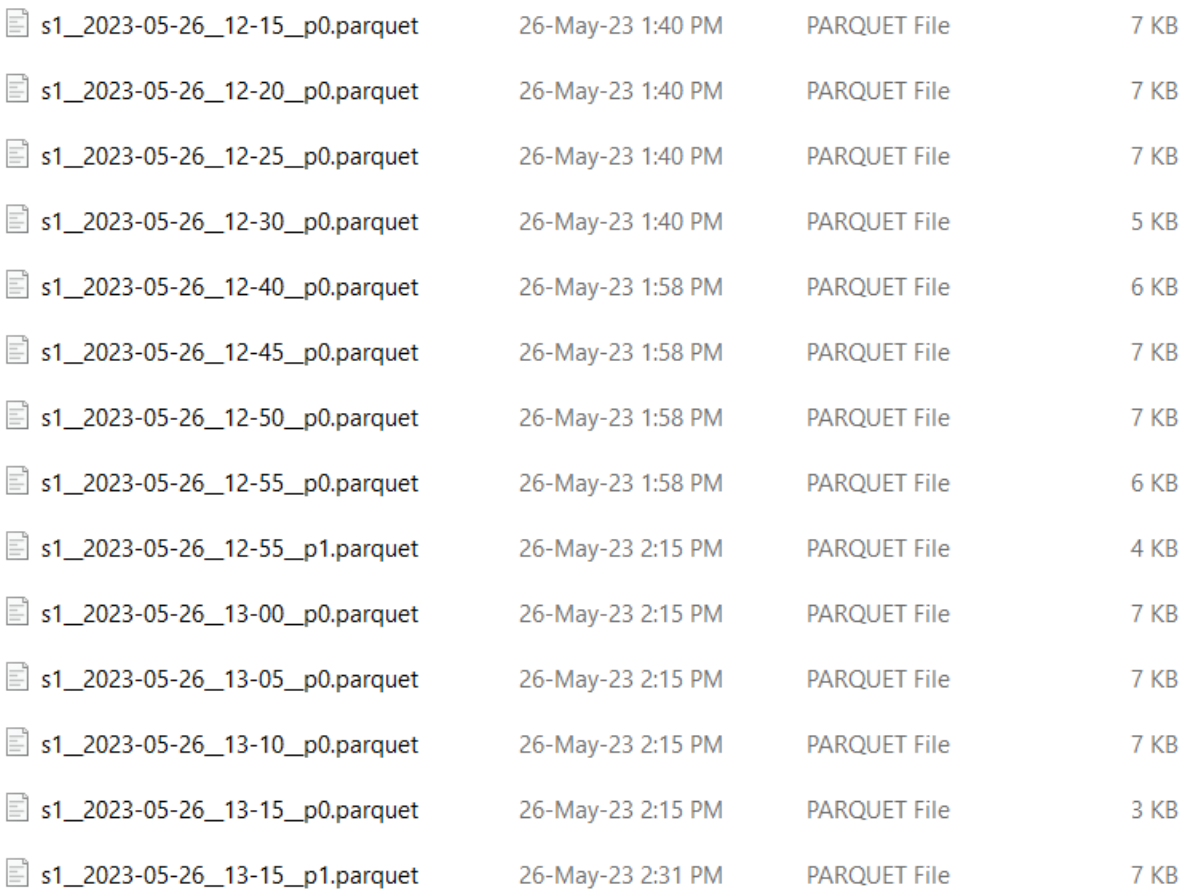
...

	station_id	s_no	battery_status	status_timestamp	humidity	temperature	wind_speed
184	3	1098	high	1684657684858	34	125	110
185	3	1099	high	1684657685864	26	22	73
186	3	1100	low	1684657686869	83	9	135
187	3	1102	low	1684657687885	36	0	130
188	3	1103	medium	1684657688891	30	124	135
189	3	1104	high	1684657689894	48	48	59
190	3	1105	medium	1684657690906	65	59	52
191	3	1106	low	1684657691913	6	28	110
192	3	1107	high	1684657692920	90	128	60
193	3	1108	medium	1684657693923	72	42	128
194	3	1109	high	1684657694934	21	4	138
195	3	1110	medium	1684657695942	100	97	56
196	3	1111	high	1684657696949	83	39	131

And this is the start of “parquet-files\s3\s3__2023-05-21__10-25__p1.parquet”:

	station_id	s_no	battery_status	status_timestamp	humidity	temperature	wind_speed
1	3	1112	low	1684657697963	49	129	113
2	3	1113	medium	1684657698965	83	16	100
3	3	1114	medium	1684657699966	73	42	93
4	3	1115	low	1684657700986	7	56	101
5	3	1116	medium	1684657701996	14	22	9
6	3	1118	low	1684657703002	36	34	89
7	3	1119	high	1684657704011	70	9	97
8	3	1120	medium	1684657705015	2	34	127
9	3	1121	medium	1684657706030	10	24	78
10	3	1122	low	1684657707031	37	71	5

This is a sample from s1 folder:



Source Code:

The source code can be found in the project Github repository along with some sample Parquet files.

- Weather-Station-Monitoring/Central-Station
- Weather-Station-Monitoring/Central-Station/parquet-files

E. Setup Historical Weather Statuses Analysis

Steps

1. Install Docker for desktop then pull nshou/elasticsearch-kibana image and run it.
    a. **docker pull nshou/elasticsearch-kibana**
    b. **docker run -d -p 9200:9200 -p 5601:5601--name elasticsearch-kibana nshou/elasticsearch-kibana.**
2. Install some needed packages and dependencies which could be found here.
3. Upload the data to the container to be later indexed to elasticsearch using a python script.
4. The python script listens on the parquet files folder to get any new file, read it into pandas data frame, connect to elasticsearch and then index this file.
5. Upload the data files of each weather station using a suitable index pattern.

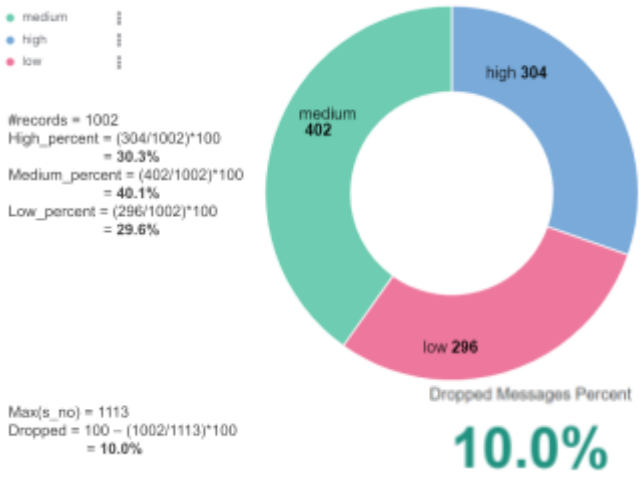
```
{
    "mappings": {
        "properties": {
            "station_id": { "type": "long" },
            "s_no":  { "type": "long" },
            "battery_status": { "type": "keyword" },
            "status_timestamp":  { "type": "long" },
            "weather_humidity": { "type": "integer" },
            "weather_temperature": { "type": "integer" },
            "weather_wind_speed": { "type": "integer" }
        }
    }
}
```

6. Used the **count()** metric to visualize the percent of different battery statuses (“low”, “medium”, “high”) and the donut-shaped chart sliced by the values of **“battery_status”**.
7. Used a special formula to estimate the dropped messages using the **“s_no”** field and the metrics **max() & count().**

```
1 - (count()/max(s_no))
```

Analysis Results

Weather_Station_One



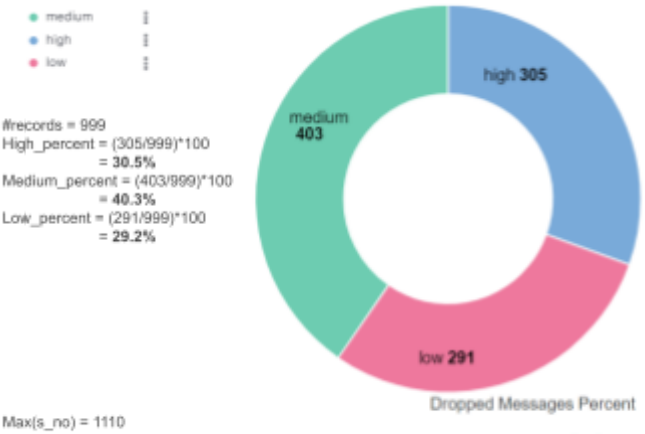
#records = 1002
High_percent = (304/1002)*100
= **30.3%**
Medium_percent = (402/1002)*100
= **40.1%**
Low_percent = (296/1002)*100
= **29.6%**

Max(s_no) = 1113
Dropped = 100 – (1002/1113)*100
= **10.0%**

Dropped Messages Percent

10.0%

Weather_Station_Two



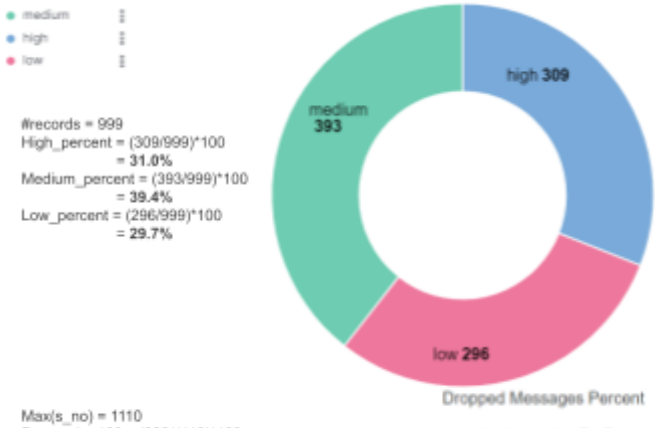
#records = 999
High_percent = (305/999)*100
= **30.5%**
Medium_percent = (403/999)*100
= **40.3%**
Low_percent = (291/999)*100
= **29.2%**

Max(s_no) = 1110
Dropped = 100 – (999/1110)*100
= **10.0%**

Dropped Messages Percent

10.0%

Weather_Station_Three



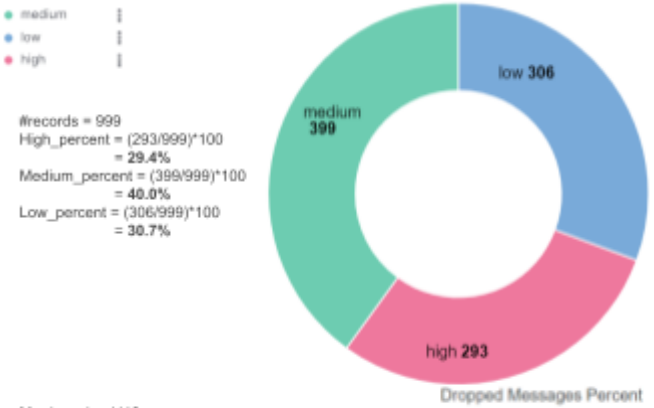
#records = 999
High_percent = (309/999)*100
= **31.0%**
Medium_percent = (393/999)*100
= **39.4%**
Low_percent = (296/999)*100
= **29.7%**

Max(s_no) = 1110
Dropped = 100 – (999/1110)*100
= **10.0%**

Dropped Messages Percent

10.0%

Weather_Station_Four



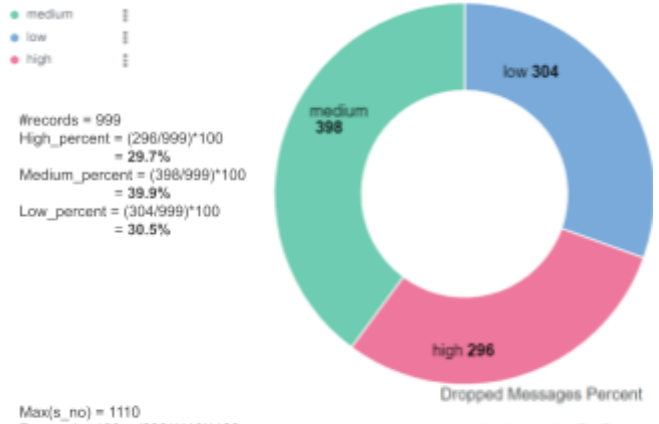
#records = 999
High_percent = (293/999)*100
= **29.4%**
Medium_percent = (399/999)*100
= **40.0%**
Low_percent = (306/999)*100
= **30.7%**

Max(s_no) = 1110
Dropped = 100 – (999/1110)*100
= **10.0%**

Dropped Messages Percent

10.0%

Weather_Station_Five



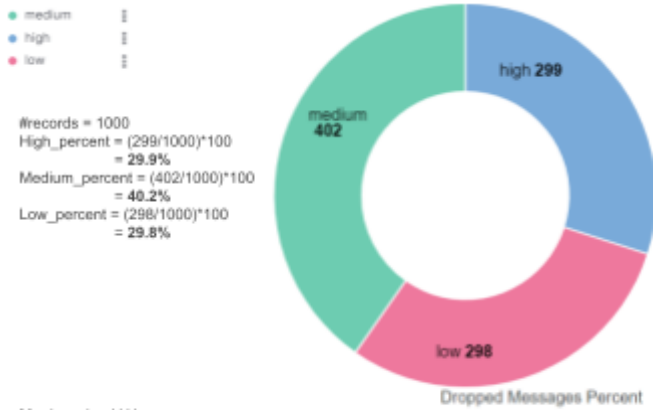
#records = 999
High_percent = (296/999)*100
= **29.7%**
Medium_percent = (398/999)*100
= **39.9%**
Low_percent = (304/999)*100
= **30.5%**

Max(s_no) = 1110
Dropped = 100 – (999/1110)*100
= **10.0%**

Dropped Messages Percent

10.0%

Weather_Station_Six



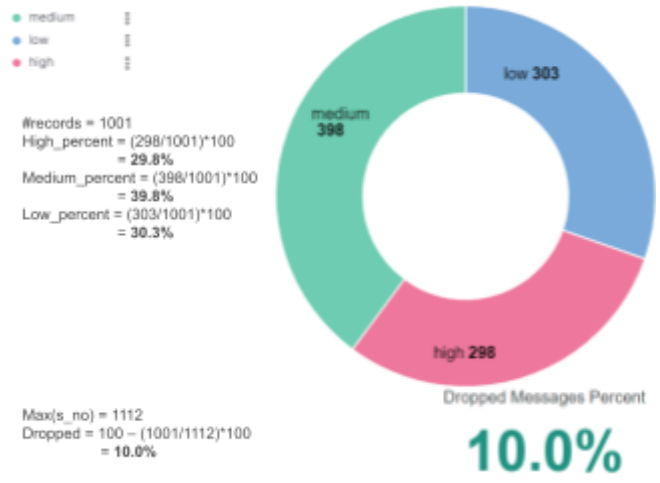
#records = 1000
High_percent = (299/1000)*100
= **29.9%**
Medium_percent = (402/1000)*100
= **40.2%**
Low_percent = (298/1000)*100
= **29.8%**

Max(s_no) = 1111
Dropped = 100 – (1000/1111)*100
= **10.0%**

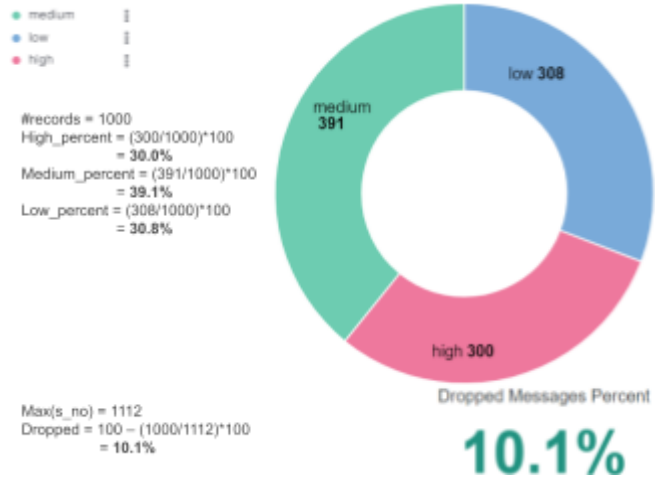
Dropped Messages Percent

10.0%

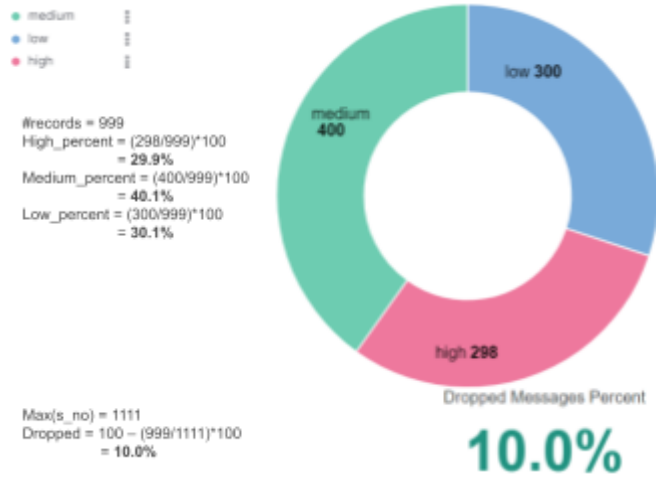
Weather_Station_Seven



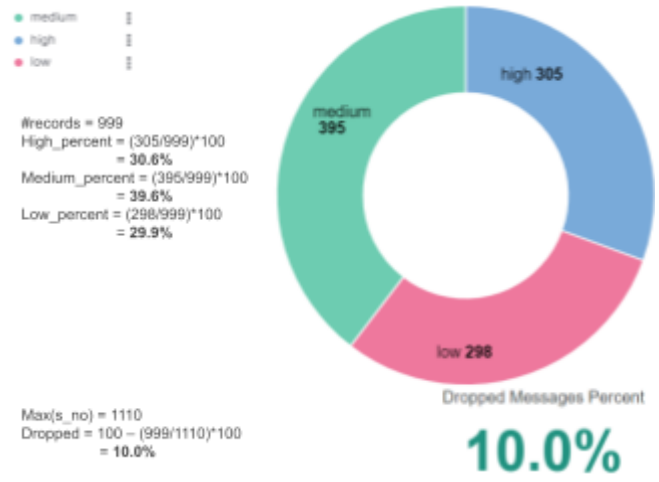
Weather_Station_Nine



Weather_Station_Eight



Weather_Station_Ten



Source Code:

The python script can be found in the project Github repository along with a notes text file to tell what to install in the container.

- [Weather-Station-Monitoring//Weather-Statuses-Analysis/code/es_write.py]
- [Weather-Station-Monitoring//Weather-Statuses-Analysis/code/NOTES.txt]

F. Deploy Using Kubernetes

Deploy using Docker

Docker file of the central station:

```
FROM openjdk:19

WORKDIR /usr/src/bitcask

COPY out ./out

VOLUME /bitcask

VOLUME /parquet-files

ENTRYPOINT sleep 60 && java -jar out/artifacts/Central_Station_jar/Central-Station.jar
```

Docker file of weather station:

```
FROM openjdk:19

WORKDIR /usr/src/myapp

COPY out ./out

ENTRYPOINT sleep 90 && java -jar out/artifacts/Weather_Station_jar/Weather-Station.jar 10
```

This is the same for the 10 weather stations with one change which is the station id. This is of station 10.

Docker file of rain detector:

```
FROM openjdk:19

WORKDIR /usr/src/myapp

COPY out ./out

ENTRYPOINT sleep 60 && java -jar out/artifacts/Rain_Detector_jar/Rain-Detector.jar
```

docker-compose.yaml

The docker-compose.yaml file can be found in the repo. Services are defined as shown in the sample screenshot.

The services are: zookeeper, kafka, weather-station-1, weather-station-2, weather-station-3, weather-station-4, weather-station-5, weather-station-6, weather-station-7, weather-station-8, weather-station-9, weather-station-10, rain-detector and central-station.

```yaml
version: '1'
services:
  zookeeper:
    image: docker.io/bitnami/zookeeper:latest
    container_name: zookeeper
    ports:
      - '2181:2181'
    environment:
      - ALLOW_ANONYMOUS_LOGIN=yes

  kafka:
    image: docker.io/bitnami/kafka:latest
    container_name: kafka
    ports:
      - '9092:9092'
    environment:
      - KAFKA_CFG_ZOOKEEPER_CONNECT=zookeeper:2181
      - KAFKA_ENABLE_KRAFT=no
      - ALLOW_PLAINTEXT_LISTENER=yes
      - KAFKA_CFG_LISTENERS=PLAINTEXT://:9092
      - KAFKA_CFG_ADVERTISED_LISTENERS=PLAINTEXT://kafka:9092
      - KAFKA_CFG_AUTO_CREATE_TOPICS_ENABLE=true
    depends_on:
      - zookeeper

```

All weather-station services are as shown:

```yaml
  weather-station-1:
    image: 'weather-station-1-image:latest'
    container_name: weather-station-1
    ports:
      - '8081:8081'
    depends_on:
      - kafka

  weather-station-2:
    image: 'weather-station-2-image:latest'
    container_name: weather-station-2
    ports:
      - '8082:8082'
    depends_on:
      - kafka

```

The rain-detector service and the central-service and the shared storage declaration:

```
106     rain-detector:
107       image: 'rain-detector-image:latest'
108       container_name: rain-detector
109       ports:
110         - '8091:8091'
111       depends_on:
112         - kafka
113
114     central-station:
115       image: 'central-station-image:latest'
116       container_name: central-station
117       ports:
118         - '8092:8092'
119       depends_on:
120         - kafka
121       volumes:
122         - weather_stations:/bitcask
123         - weather_stations:/parquet-files
124
125   volumes:
126     weather_stations:
```

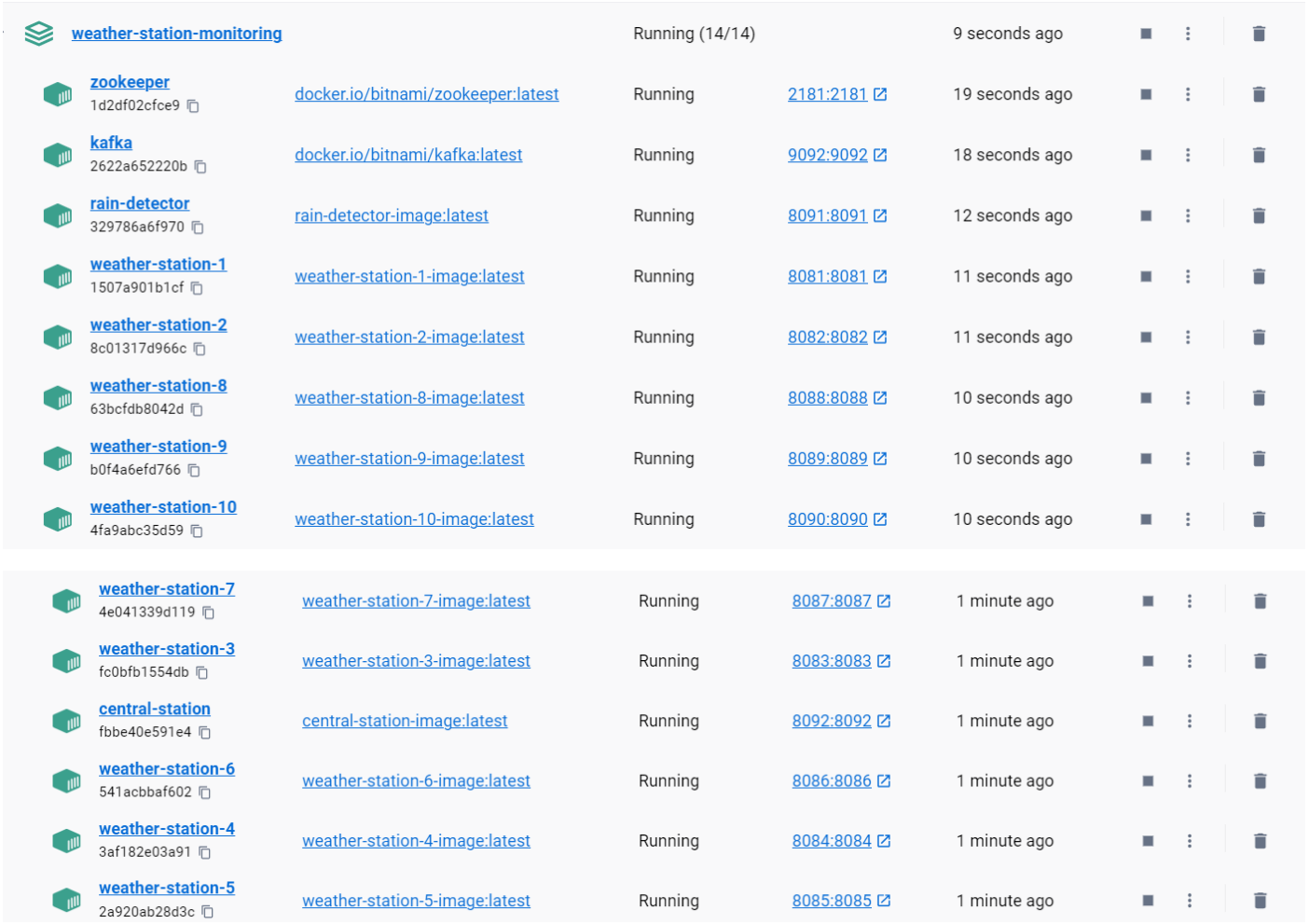
On running the command “docker-compose up”:

```
PS D:\Projects\Weather-Station-Monitoring> docker-compose up
[+] Running 15/15
 ✓ Network weather-station-monitoring_default  Created          0.2s
 ✓ Container zookeeper                        Created          0.8s
 ✓ Container kafka                            Created          0.4s
 ✓ Container rain-detector                    Created          2.5s
 ✓ Container weather-station-9                Created          3.1s
 ✓ Container weather-station-3                Created          3.1s
 ✓ Container central-station                  Created          2.8s
 ✓ Container weather-station-2                Created          2.2s
 ✓ Container weather-station-10               Created          2.5s
 ✓ Container weather-station-1                Created          2.5s
 ✓ Container weather-station-5                Created          2.2s
 ✓ Container weather-station-7                Created          3.1s
 ✓ Container weather-station-4                Created          3.1s
 ✓ Container weather-station-8                Created          2.3s
 ✓ Container weather-station-6                Created          3.1s
Attaching to central-station, kafka, rain-detector, weather-station-1, weather-station-10, weather-station-2
, weather-station-3, weather-station-4, weather-station-5, weather-station-6, weather-station-7, weather-sta
tion-8, weather-station-9, zookeeper
```

And these are the running containers after starting the services using “docker-compose.yaml”:

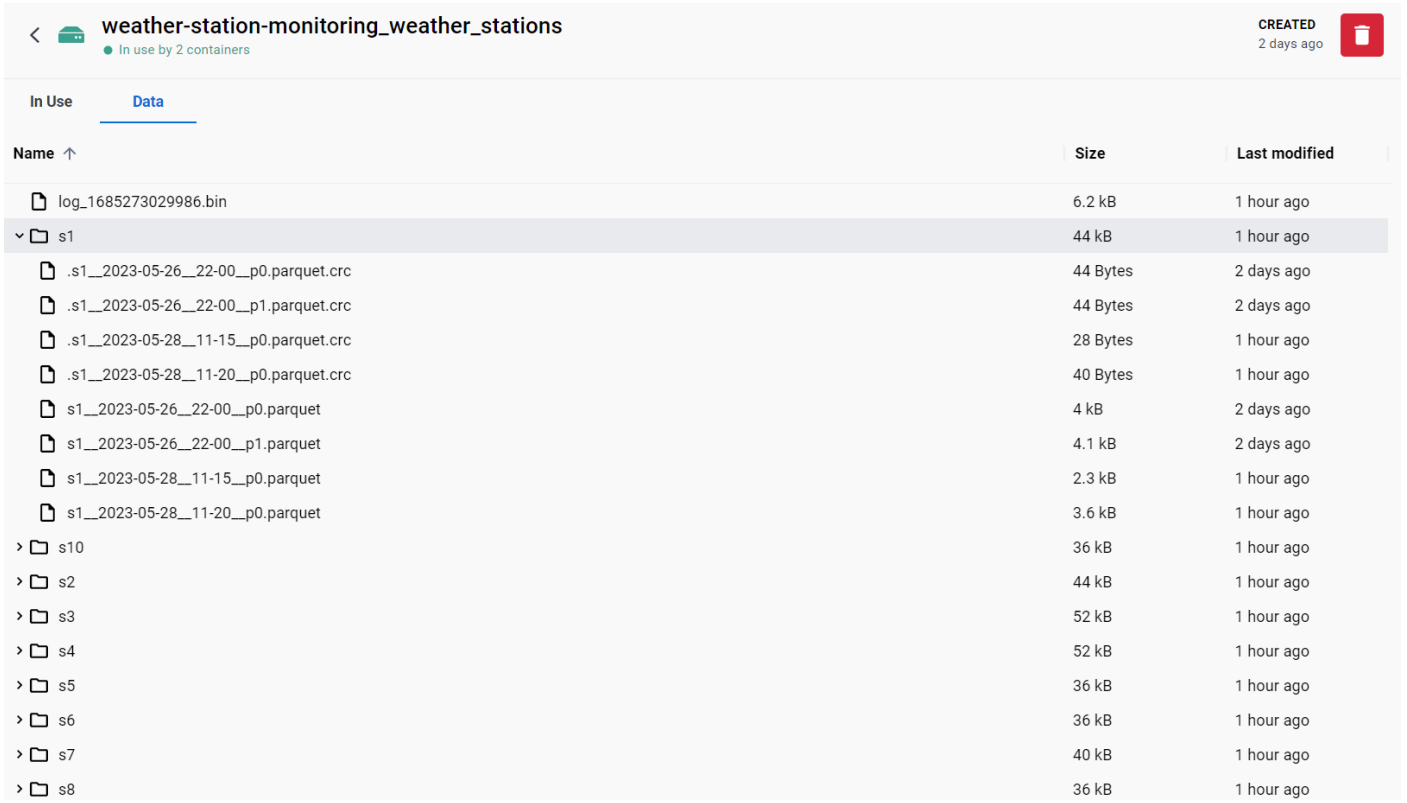
```
PS D:\Projects\Weather-Station-Monitoring> docker ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED         STATUS         PORTS
NAMES
541acbbaf602   weather-station-6-image:latest       "/bin/sh -c 'sleep 9…"   2 minutes ago   Up 2 minutes   0.0.0.0:8086->8086/tcp
weather-station-6
3af182e03a91   weather-station-4-image:latest       "/bin/sh -c 'sleep 9…"   3 minutes ago   Up 2 minutes   0.0.0.0:8084->8084/tcp
weather-station-4
1507a901b1cf   weather-station-1-image:latest       "/bin/sh -c 'sleep 9…"   3 minutes ago   Up 2 minutes   0.0.0.0:8081->8081/tcp
weather-station-1
4fa9abc35d59   weather-station-10-image:latest      "/bin/sh -c 'sleep 9…"   3 minutes ago   Up 2 minutes   0.0.0.0:8090->8090/tcp
weather-station-10
fc0bfb1554db   weather-station-3-image:latest       "/bin/sh -c 'sleep 9…"   3 minutes ago   Up 2 minutes   0.0.0.0:8083->8083/tcp
weather-station-3
4e041339d119   weather-station-7-image:latest       "/bin/sh -c 'sleep 9…"   3 minutes ago   Up 2 minutes   0.0.0.0:8087->8087/tcp
weather-station-7
8c01317d966c   weather-station-2-image:latest       "/bin/sh -c 'sleep 9…"   3 minutes ago   Up 2 minutes   0.0.0.0:8082->8082/tcp
weather-station-2
63bcfdb8042d   weather-station-8-image:latest       "/bin/sh -c 'sleep 9…"   3 minutes ago   Up 2 minutes   0.0.0.0:8088->8088/tcp
weather-station-8
2a920ab28d3c   weather-station-5-image:latest       "/bin/sh -c 'sleep 9…"   3 minutes ago   Up 2 minutes   0.0.0.0:8085->8085/tcp
weather-station-5
fbbe40e591e4   central-station-image:latest         "/bin/sh -c 'sleep 6…"   3 minutes ago   Up 2 minutes   0.0.0.0:8092->8092/tcp
central-station
b0f4a6efd766   weather-station-9-image:latest       "/bin/sh -c 'sleep 9…"   3 minutes ago   Up 2 minutes   0.0.0.0:8089->8089/tcp
weather-station-9
329786a6f970   rain-detector-image:latest           "/bin/sh -c 'sleep 6…"   3 minutes ago   Up 2 minutes   0.0.0.0:8091->8091/tcp
rain-detector
2622a652220b   bitnami/kafka:latest                 "/opt/bitnami/script…"   3 minutes ago   Up 2 minutes   0.0.0.0:9092->9092/tcp
kafka
1d2df02cfce9   bitnami/zookeeper:latest             "/opt/bitnami/script…"   3 minutes ago   Up 2 minutes   2888/tcp, 3888/tcp, 0.0.0.0:2181->2181/tcp, 8080/tcp
zookeeper
```

From docker UI:



This screenshot shows the system's behaviour. Station 8 produces a message then the central station consumes it and so on.

Shared storage:



Deploy using Kubernetes

Following the provided tutorial and using Kompose, the k8s yaml files were generated using the command "kompose convert" which generates the services and deployments yaml files from the docker-compose file.

K8s yaml files (services and deployments):

- Weather-Station-Monitoring/K8s yaml files

Using kubectl, the services and deployments were created using the command "kubectl apply -f <yaml_file_name>.

Screenshot of Kubernetes services created:

```
PS D:\Projects\Weather-Station-Monitoring> kubectl get services
NAME                TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)    AGE
central-station     ClusterIP   10.97.203.138    <none>        8092/TCP   24m
kafka               ClusterIP   10.110.117.127   <none>        9092/TCP   25m
kubernetes          ClusterIP   10.96.0.1        <none>        443/TCP    26m
rain-detector       ClusterIP   10.108.128.82    <none>        8091/TCP   13m
weather-station-1   ClusterIP   10.104.79.163    <none>        8081/TCP   13m
weather-station-10  ClusterIP   10.103.236.11    <none>        8090/TCP   12m
weather-station-2   ClusterIP   10.97.45.2       <none>        8082/TCP   13m
weather-station-3   ClusterIP   10.103.141.92    <none>        8083/TCP   13m
weather-station-4   ClusterIP   10.104.175.92    <none>        8084/TCP   13m
weather-station-5   ClusterIP   10.99.1.142      <none>        8085/TCP   13m
weather-station-6   ClusterIP   10.100.108.199   <none>        8086/TCP   13m
weather-station-7   ClusterIP   10.108.231.215   <none>        8087/TCP   12m
weather-station-8   ClusterIP   10.107.111.55    <none>        8088/TCP   12m
weather-station-9   ClusterIP   10.101.234.180   <none>        8089/TCP   12m
zookeeper           ClusterIP   10.108.79.89     <none>        2181/TCP   25m
```

Screenshot of Kubernetes deployments created:

```
PS D:\Projects\Weather-Station-Monitoring> kubectl get deployments
NAME                READY   UP-TO-DATE   AVAILABLE   AGE
central-station     1/1     1            1           17m
kafka               1/1     1            1           24m
rain-detector       1/1     1            1           2m45s
weather-station-1   1/1     1            1           2m31s
weather-station-10  1/1     1            1           107s
weather-station-2   1/1     1            1           2m24s
weather-station-3   1/1     1            1           2m19s
weather-station-4   1/1     1            1           2m16s
weather-station-5   1/1     1            1           2m12s
weather-station-6   1/1     1            1           2m8s
weather-station-7   1/1     1            1           2m3s
weather-station-8   1/1     1            1           118s
weather-station-9   1/1     1            1           111s
zookeeper           1/1     1            1           25m
```

Screenshot of Kubernetes pods running:

```
PS D:\Projects\Weather-Station-Monitoring> kubectl get pods -o wide
NAME                                READY   STATUS    RESTARTS   AGE   IP            NODE
central-station-dcb96d56b-ngbps     1/1     Running   0          16m   10.244.0.28   minikube
kafka-589564b754-qkmmz              1/1     Running   0          23m   10.244.0.26   minikube
rain-detector-b797448cb-whtgq       1/1     Running   0          66s   10.244.0.31   minikube
weather-station-1-58775b6d49-7vrjv  1/1     Running   0          52s   10.244.0.32   minikube
weather-station-10-64ff474d59-pz5pp 1/1     Running   0          8s    10.244.0.41   minikube
weather-station-2-58d874cd97-zmrqd  1/1     Running   0          45s   10.244.0.33   minikube
weather-station-3-776444b669-kblfl  1/1     Running   0          40s   10.244.0.34   minikube
weather-station-4-7894d7bb8f-x2d4h  1/1     Running   0          37s   10.244.0.35   minikube
weather-station-5-db9b4bf7-sqw7k    1/1     Running   0          33s   10.244.0.36   minikube
weather-station-6-54f69d6fdf-25kww  1/1     Running   0          29s   10.244.0.37   minikube
weather-station-7-759c7fd5bf-n7wcz  1/1     Running   0          24s   10.244.0.38   minikube
weather-station-8-d67bf78c7-5h9tc   1/1     Running   0          19s   10.244.0.39   minikube
weather-station-9-8657cfc745-mtnjx  1/1     Running   0          12s   10.244.0.40   minikube
zookeeper-588d84df44-2p2x9          1/1     Running   0          23m   10.244.0.25   minikube
```

G. Profile Central Station Using JFR

Top 10 classes with highest total memory:

Class	Max Live Count	Max Live Size	Live Size Incre...	Alloc Total	Total Allocatio...
byte[]				600 MiB	96.1 %
java.util.ArrayList$Itr				11.2 MiB	1.79 %
java.util.concurrent.ConcurrentHashMap$KeyIterator				3.07 MiB	0.492 %
org.apache.kafka.clients.consumer.KafkaConsumer$$Lambda$29...				1.59 MiB	0.255 %
java.util.HashMap$KeyIterator				1.16 MiB	0.186 %
java.util.LinkedHashMap				1.16 MiB	0.186 %
java.util.stream.ReferencePipeline$Head				1.16 MiB	0.186 %
java.util.stream.ReferencePipeline$2$1				1.15 MiB	0.185 %
java.util.HashMap$KeySpliterator				638 KiB	0.0997 %
java.util.HashMap$EntryIterator				613 KiB	0.0959 %

GC pauses count and maximum pause duration:

GC Summary

Young Collection Total Time

GC Count	3
Average GC Time	1.475 ms
Maximum GC Time	1.871 ms
Total GC Time	4.426 ms

Old Collection Total Time

GC Count	3
Average GC Time	18.076 ms
Maximum GC Time	35.905 ms
Total GC Time	54.228 ms

All Collections Total Time

GC Count	6
Average GC Time	9.776 ms
Maximum GC Time	35.905 ms
Total GC Time	58.654 ms

All Collections Pause Time

Average Pause	3.813 ms
Longest Pause	18.566 ms
Sum of Pauses	26.690 ms

List of I/O operations:

Path	Total I/O Time	Count	Read Count	Write Count	Bytes Read	Bytes Written
bitcask\log_1685227896761.bin	362.684 ms	1		1		8 B