VIET NAM NATIONAL UNIVERSITY HO CHI MINH CITY
HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

SEMESTER 241 - CO3071

LAB 1 REPORT

DISTRIBUTED SYSTEMS

Instructor: **PROF. Nguyễn Mạnh Thìn**

Tạ Gia Khang	- 2152642
Nguyễn Quang Thiện	- 2152994

HO CHI MINH CITY, NOVEMBER 2024

Contents

1 Homework

1.1 Configuration

Using Docker and generating a docker-compose.yml file to create 3 brokers with 3 different ports. The reason why using this technique is the exercise is quite similar to our assignment project: Developing A Smart Energy Consumption Prediction Pipeline, which also publishes data to Kafka and subscribes to data from Kafka.

```yaml
version: '2'
services:
  kafka1:
    image: apache/kafka:3.7.0
    hostname: kafka1
    container_name: kafka1
    ports:
      - 29092:9092
    environment:
      KAFKA_NODE_ID: 1
      KAFKA_PROCESS_ROLES: 'broker,controller'
      KAFKA_LISTENER_SECURITY_PROTOCOL_MAP:
      'CONTROLLER:PLAINTEXT,PLAINTEXT:PLAINTEXT,PLAINTEXT_HOST:PLAINTEXT'
      KAFKA_CONTROLLER_QUORUM_VOTERS: '1@kafka1:9093,2@kafka2:9093,3@kafka3:9093'
      KAFKA_LISTENERS:
      'PLAINTEXT://:29092,CONTROLLER://:9093,PLAINTEXT_HOST://:9092'
      KAFKA_INTER_BROKER_LISTENER_NAME: 'PLAINTEXT'
      KAFKA_ADVERTISED_LISTENERS:
      PLAINTEXT://kafka1:29092,PLAINTEXT_HOST://localhost:29092
      KAFKA_CONTROLLER_LISTENER_NAMES: 'CONTROLLER'
      CLUSTER_ID: '2UKPPoqNTPezeassrAogQw'
      KAFKA_OFFSETS_TOPIC_NUM_PARTITIONS: 1
      KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
      KAFKA_GROUP_INITIAL_REBALANCE_DELAY_MS: 0
      KAFKA_TRANSACTION_STATE_LOG_MIN_ISR: 1
      KAFKA_TRANSACTION_STATE_LOG_REPLICATION_FACTOR: 1
      KAFKA_LOG_DIRS: '/tmp/kraft-combined-logs'
    networks:
      - lab1_ds_kafka_network

  kafka2:
    image: apache/kafka:3.7.0
    hostname: kafka2
    container_name: kafka2
    ports:
      - 39092:9092
    environment:
      KAFKA_NODE_ID: 2
      KAFKA_PROCESS_ROLES: 'broker,controller'
      KAFKA_LISTENER_SECURITY_PROTOCOL_MAP:
      'CONTROLLER:PLAINTEXT,PLAINTEXT:PLAINTEXT,PLAINTEXT_HOST:PLAINTEXT'
      KAFKA_CONTROLLER_QUORUM_VOTERS: '1@kafka1:9093,2@kafka2:9093,3@kafka3:9093'
      KAFKA_LISTENERS:
      'PLAINTEXT://:39092,CONTROLLER://:9093,PLAINTEXT_HOST://:9092'
      KAFKA_INTER_BROKER_LISTENER_NAME: 'PLAINTEXT'
      KAFKA_ADVERTISED_LISTENERS:
      PLAINTEXT://kafka2:39092,PLAINTEXT_HOST://localhost:39092
      KAFKA_CONTROLLER_LISTENER_NAMES: 'CONTROLLER'
      CLUSTER_ID: '2UKPPoqNTPezeassrAogQw'
      KAFKA_OFFSETS_TOPIC_NUM_PARTITIONS: 1
      KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
      KAFKA_GROUP_INITIAL_REBALANCE_DELAY_MS: 0
      KAFKA_TRANSACTION_STATE_LOG_MIN_ISR: 1
      KAFKA_TRANSACTION_STATE_LOG_REPLICATION_FACTOR: 1
      KAFKA_LOG_DIRS: '/tmp/kraft-combined-logs'
    networks:
      - lab1_ds_kafka_network

  kafka3:
    image: apache/kafka:3.7.0
    hostname: kafka3
    container_name: kafka3
    ports:
      - 49092:9092
    environment:
      KAFKA_NODE_ID: 3
      KAFKA_PROCESS_ROLES: 'broker,controller'
      KAFKA_LISTENER_SECURITY_PROTOCOL_MAP:
      'CONTROLLER:PLAINTEXT,PLAINTEXT:PLAINTEXT,PLAINTEXT_HOST:PLAINTEXT'
      KAFKA_CONTROLLER_QUORUM_VOTERS: '1@kafka1:9093,2@kafka2:9093,3@kafka3:9093'
      KAFKA_LISTENERS:
      'PLAINTEXT://:49092,CONTROLLER://:9093,PLAINTEXT_HOST://:9092'
      KAFKA_INTER_BROKER_LISTENER_NAME: 'PLAINTEXT'
      KAFKA_ADVERTISED_LISTENERS:
      PLAINTEXT://kafka3:49092,PLAINTEXT_HOST://localhost:49092
      KAFKA_CONTROLLER_LISTENER_NAMES: 'CONTROLLER'
      CLUSTER_ID: '2UKPPoqNTPezeassrAogQw'
      KAFKA_OFFSETS_TOPIC_NUM_PARTITIONS: 1
      KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
      KAFKA_GROUP_INITIAL_REBALANCE_DELAY_MS: 0
      KAFKA_TRANSACTION_STATE_LOG_MIN_ISR: 1
      KAFKA_TRANSACTION_STATE_LOG_REPLICATION_FACTOR: 1
      KAFKA_LOG_DIRS: '/tmp/kraft-combined-logs'
    networks:
      - lab1_ds_kafka_network

  kafka-ui:
    container_name: kafka-ui
    image: provectuslabs/kafka-ui:latest
    ports:
      - "8080:8080"
    environment:
      DYNAMIC_CONFIG_ENABLED: 'true'
    depends_on:
      - kafka1
      - kafka2
      - kafka3
    networks:
      - lab1_ds_kafka_network

networks:
  lab1_ds_kafka_network:
    external: true
```

This Docker Compose configuration sets up a Kafka cluster with three brokers (**kafka1**, **kafka2**, **kafka3**) and a UI service (**kafka-ui**) to manage and monitor the Kafka setup. Below is an explanation of each component.

Kafka Brokers: Each Kafka broker has the following configurations.

- **Image and Hostname:** Uses **apache/kafka:3.7.0** as the image. Each broker is assigned a unique hostname and container name for identification.
- **Ports:** Exposes internal Kafka port **9092** to different external ports **29092**, **39092**, and **49092** on the host machine.
- **Environment Variables:**

  `KAFKA_NODE_ID`: Unique node ID for each broker (1, 2, 3).

  `KAFKA_PROCESS_ROLES`: Specifies the roles as both `broker` and `controller`, suitable for KRaft mode.

  `KAFKA_LISTENER_SECURITY_PROTOCOL_MAP`: Defines protocols for listeners (`CONTROLLER`, `PLAINTEXT`, `PLAINTEXT_HOST`).

  `KAFKA_CONTROLLER_QUORUM_VOTERS`: Lists nodes in the controller quorum in the format `<node ID>@<hostname>:<port>`.

  `KAFKA_LISTENERS`: Defines listeners including `PLAINTEXT`, `CONTROLLER`, and `PLAINTEXT_HOST`.

  `KAFKA_INTER_BROKER_LISTENER_NAME`: Specifies the inter-broker communication listener (`PLAINTEXT`).

  `KAFKA_ADVERTISED_LISTENERS`: Advertised addresses for external connections.

  `CLUSTER_ID`: Shared cluster ID for all brokers.

  `KAFKA_OFFSETS_TOPIC_NUM_PARTITIONS`, `KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR`: Configures the offset topic.

  `KAFKA_GROUP_INITIAL_REBALANCE_DELAY_MS`: Delay in ms for consumer group rebalancing.

  `KAFKA_TRANSACTION_STATE_LOG_MIN_ISR`, `KAFKA_TRANSACTION_STATE_LOG_REPLICATION_FACTOR`: Configures transaction state log for fault tolerance.

  `KAFKA_LOG_DIRS`: Specifies log data storage location.

Kafka UI: The Kafka UI (**kafka-ui**) provides a web interface for Kafka management.

- **Image and Ports:** Uses **provectuslabs/kafka-ui** image and exposes port **8080**.
- **Environment Variables:**

  `DYNAMIC_CONFIG_ENABLED`: Enables dynamic configuration updates.
- **Dependencies:** Depends on Kafka brokers to ensure they are running before **kafka-ui** starts.

Network Configuration: The network **lab1_ds_kafka_network** is an external Docker network that enables communication among all services.

1.2 Implementation

1.2.1 Objectives

- **Kafka Topic Setup:** Create dedicated Kafka topics for each data type with partitions representing different sensor stations.
- **Concurrent Data Streaming:** Implement a producer to send data concurrently to Kafka topics using custom serialization, partitioning, and error handling.
- **Partitioned Data Consumption:** Create multiple consumers assigned to partitions to process and log data independently.

1.2.2 Producer

```
KAFKA_BOOTSTRAP_SERVERS = ['localhost:29092', 'localhost:39092', 'localhost:49092']
producer = Producer({'bootstrap.servers': ','.join(KAFKA_BOOTSTRAP_SERVERS)})
partition = 2  # Set partition count for topic
```

The producer connects to local Kafka brokers. Here, two partitions are defined per topic to ensure data is distributed according to sensor station ID.

```
parser = argparse.ArgumentParser()
parser.add_argument("--thread", type=int, default=3, help='Number of threads for concurrent data streaming')
parser.add_argument("--delay", type=int, default=10, help='Delay time (seconds) between messages')
args = parser.parse_args()
```

Arguments allow dynamic control over the number of threads (`--thread`) and message delay interval (`--delay`), enhancing flexibility in testing and real-time data flow control.

```
csv_files = {'air': "AIR2308.csv", 'earth': "EARTH2308.csv", 'water': "WATER2308.csv"}
KAFKA_TOPICS = {'air': 'AIR', 'earth': 'EARTH', 'water': 'WATER'}
```

Mappings link each data source to its respective file and Kafka topic, ensuring clarity and structure for data handling within each partitioned Kafka topic.

The `generate_data` function reads CSV data, serializes each row, and sends it to Kafka topics using the specified partition.

```python
def generate_data(source, num):
    csv_file = csv_files[source]
    KAFKA_TOPIC = KAFKA_TOPICS[source]
    df = pd.read_csv(csv_file)

    for index in range(len(df)):
        try:
            partition_id = index % partition  # Ensures data distribution across
            partitions
            row = df.iloc[index]
            msg = {col: (value if isinstance(value, (int, float)) else str(value))
            for col in df.columns}
            producer.produce(KAFKA_TOPIC, partition=partition_id,
            value=json.dumps(msg))
            producer.poll(1)
            print(f"Published {source.upper()} data: ", msg)
            time.sleep(_delay)
        except KeyboardInterrupt:
            break
        except Exception as e:
            print(f"Error: {e}")
            break
```

- Partitioning Logic: The modulo operation (`index % partition`) assigns each message to a specific partition.
- Serialization: Data is converted to JSON format, suitable for Kafka's message format requirements.
- Error Handling: Catch-all exception handling ensures stability in case of data or network interruptions.

The producer initiates concurrent threads for each data type (air, earth, and water) to stream data simultaneously, improving data throughput.

```python
if __name__ == '__main__':
    thread_lst = []
    sources = ['air', 'earth', 'water']

    for i in range(_thread):
        source = sources[i]
        t = threading.Thread(target=generate_data, args=(source, i))
        t.start()
        thread_lst.append(t)

    for t in thread_lst:
        t.join()
    print("Data streaming completed.")
```

1.2.3 Consumer

The use of partitioned consumers enables concurrent, independent data processing across sensor stations. Each consumer only reads from one partition, maintaining a streamlined workflow where each subset of data is processed separately. This design aligns with the objective of efficient, real-time data handling, essential in applications where latency and reliability are critical.

Each consumer is configured to connect to the Kafka topic and process data exclusively from one partition. This setup ensures that each station's data is independently managed.

```python
from confluent_kafka import Consumer, KafkaError
import json

# Configuration Parameters
KAFKA_BOOTSTRAP_SERVERS = 'localhost:29092,localhost:39092,localhost:49092'
GROUP_ID = 'sensor_data_group'
TOPIC = 'AIR'  # Modify as necessary for 'EARTH' or 'WATER' consumers

# Initialize Consumer and Assign Partition
consumer = Consumer({
    'bootstrap.servers': KAFKA_BOOTSTRAP_SERVERS,
    'group.id': GROUP_ID,
    'auto.offset.reset': 'earliest'
})

partition_id = 0  # Adjust to 1 for the second consumer
consumer.assign([{'topic': TOPIC, 'partition': partition_id}])
```

Each consumer polls for messages in its assigned partition, processes them, and logs the data for further analysis.

```python
try:
    while True:
        msg = consumer.poll(1.0)  # Poll for new messages

        if msg is None:
            continue  # No new messages in queue
        if msg.error():
            if msg.error().code() == KafkaError._PARTITION_EOF:
                print(f"End of partition {partition_id} reached.")
            else:
                print(f"Error: {msg.error()}")
            continue

        # Deserialize and log message
        message = json.loads(msg.value().decode('utf-8'))
        print(f"Data from partition {partition_id}: ", message)

except KeyboardInterrupt:
    print("Consumer interrupted")

finally:
    consumer.close()  # Ensure clean shutdown
```

1.2.4 Observation

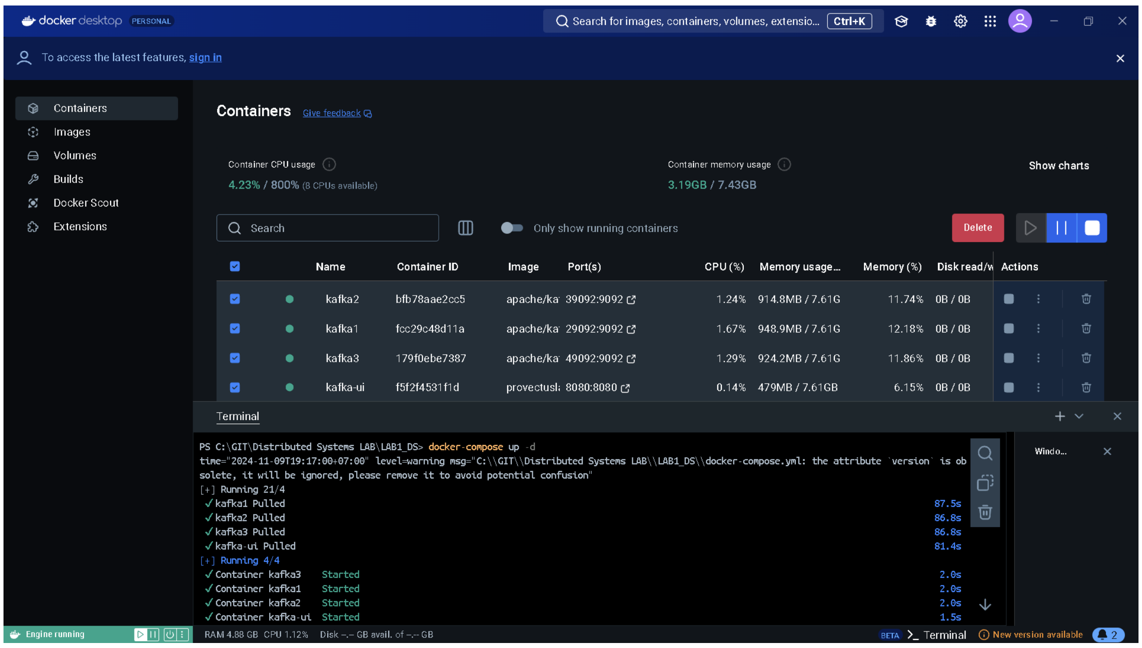
Producer Observations

- **Concurrent Data Transmission:** Multi-threaded production allowed simultaneous publishing to all three topics (air, earth, water), ensuring timely data flow.
- **Partitioning:** Messages were effectively distributed across partitions by station, facilitating balanced load and independent processing.
- **Error Handling:** Exceptions were logged without disrupting the overall flow, and re-sending capabilities ensured data integrity during temporary disconnections.
- **Performance and Scalability:** The producer maintained high throughput and is scalable, though optimizations may be needed as data sources expand.

Consumer Observations

- **Partition-Specific Processing:** Each consumer was assigned to a partition, which allowed isolated and fault-tolerant data handling by station.
- **Low Latency:** Consumers processed messages promptly, crucial for real-time data monitoring.
- **Fault Tolerance:** Error handling and reconnection protocols ensured resilience, allowing consumers to resume from the last processed message.
- **Scalability:** Additional consumers could be added for new partitions, supporting future data load increases.

1.3 Results



Container CPU usage
4.23% / 800% (8 CPUs available)

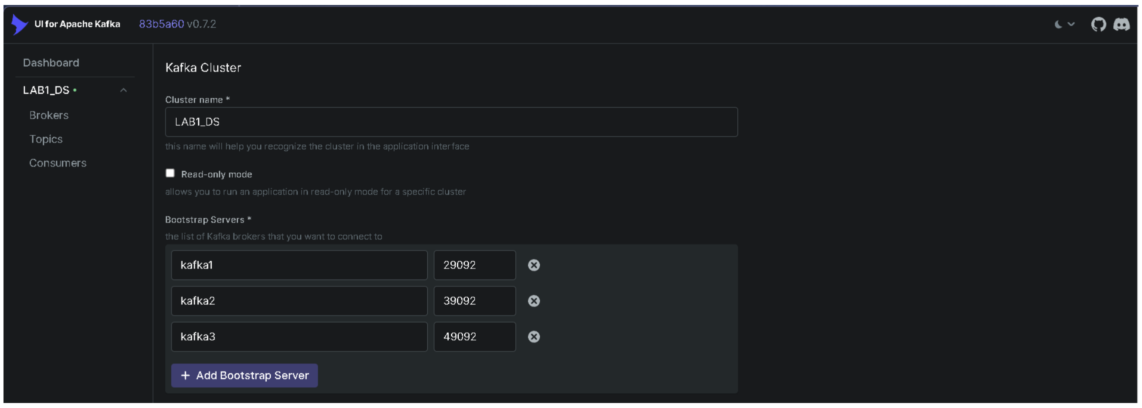
Container memory usage
3.19GB / 7.43GB

Name	Container ID	Image	Port(s)	CPU (%)	Memory usage...	Memory (%)	Disk read/w	Actions
kafka2	bfb78aae2cc5	apache/ka	39092:9092	1.24%	914.8MB / 7.61G	11.74%	0B / 0B	
kafka1	fcc29c48d11a	apache/ka	29092:9092	1.67%	948.9MB / 7.61G	12.18%	0B / 0B	
kafka3	179f0ebe7387	apache/ka	49092:9092	1.29%	924.2MB / 7.61G	11.86%	0B / 0B	
kafka-ui	f5f2f4531f1d	provectusl:	8080:8080	0.14%	479MB / 7.61GB	6.15%	0B / 0B	

```
PS C:\GIT\Distributed Systems LAB\LAB1_DS> docker-compose up -d
time="2024-11-09T19:17:00+07:00" level=warning msg="C:\\GIT\\Distributed Systems LAB\\LAB1_DS\\docker-compose.yml: the attribute `version` is ob
solete, it will be ignored, please remove it to avoid potential confusion"
[+] Running 21/4
 ✓kafka1 Pulled                                                                                                 87.5s
 ✓kafka2 Pulled                                                                                                 86.8s
 ✓kafka3 Pulled                                                                                                 86.8s
 ✓kafka-ui Pulled                                                                                               81.4s
[+] Running 4/4
 ✓Container kafka3    Started                                                                                    2.0s
 ✓Container kafka1    Started                                                                                    2.0s
 ✓Container kafka2    Started                                                                                    2.0s
 ✓Container kafka-ui  Started                                                                                    1.5s
```

RAM 4.88 GB CPU 1.12% Disk –.– GB avail. of –.-- GB

Figure 1: Docker Compose



Cluster name *

LAB1_DS

this name will help you recognize the cluster in the application interface

Read-only mode

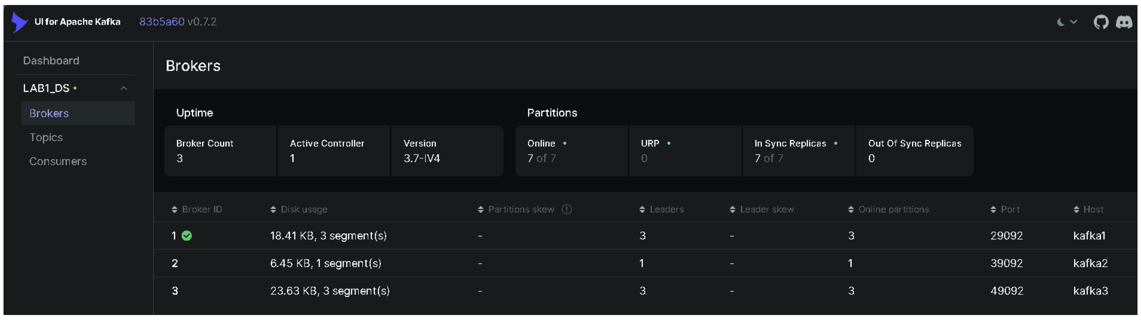
allows you to run an application in read-only mode for a specific cluster

Bootstrap Servers *

the list of Kafka brokers that you want to connect to

kafka1	29092
kafka2	39092
kafka3	49092

Figure 2: Configure Kafka Cluster



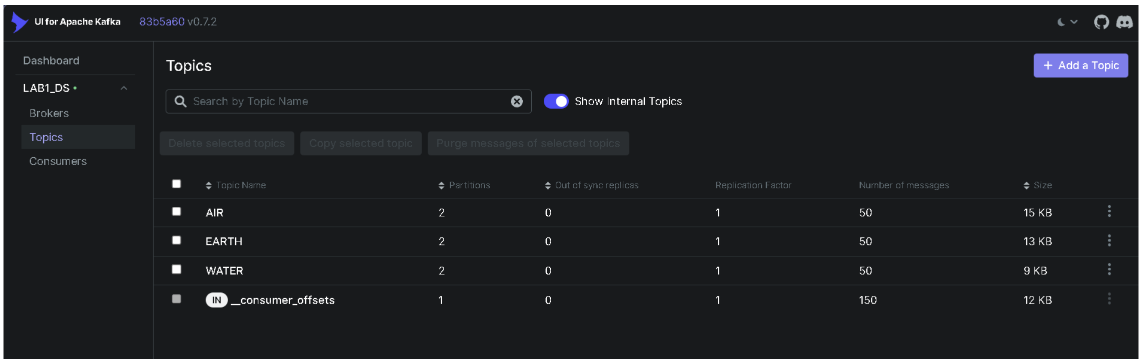
Broker Count	Active Controller	Version	Online	URP	In Sync Replicas	Out Of Sync Replicas
3	1	3.7-IV4	7 of 7	0	7 of 7	0

Broker ID	Disk usage	Partitions skew	Leaders	Leader skew	Online partitions	Port	Host
1 ✓	18.41 KB, 3 segment(s)	-	3	-	3	29092	kafka1
2	6.45 KB, 1 segment(s)	-	1	-	1	39092	kafka2
3	23.63 KB, 3 segment(s)	-	3	-	3	49092	kafka3

Figure 3: Brokers connected after configuring Docker compose and Kafka cluster

In a Kafka cluster with multiple brokers, a producer can send data to any broker. The receiving broker will determine the leader of the target partition and route the data there, ensuring it's logged and replicated according to the topic's replication factor. This design allows any broker to act as a gateway, distributing load and maintaining flexible data routing, which contributes to Kafka's fault tolerance and scalability.

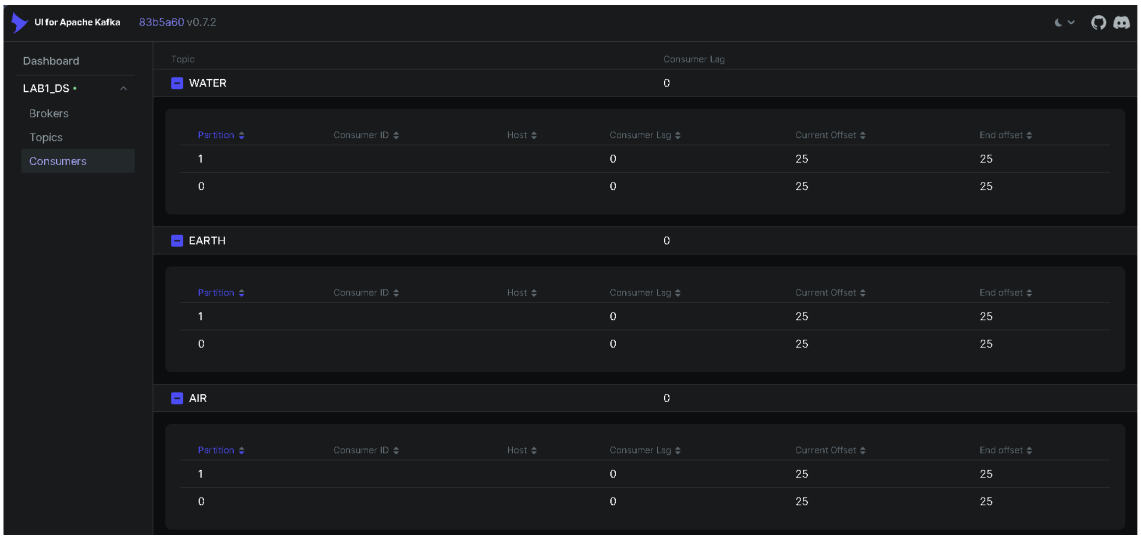
That's why the online partition of broker 2 in the image has only 1, not the leader for a particular topic or partition but can still accept data from producers.



Topic Name	Partitions	Out of sync replicas	Replication Factor	Number of messages	Size
AIR	2	0	1	50	15 KB
EARTH	2	0	1	50	13 KB
WATER	2	0	1	50	9 KB
IN __consumer_offsets	1	0	1	150	12 KB

Figure 4: Topics with 2 partitions

Generating the number of partitions, and replication factors of the Producer topics. (the docker-compose file just defines the number of partitions and replication factors of the Consumer)



Topic	Consumer Lag
WATER	0

Partition	Consumer ID	Host	Consumer Lag	Current Offset	End offset
1			0	25	25
0			0	25	25

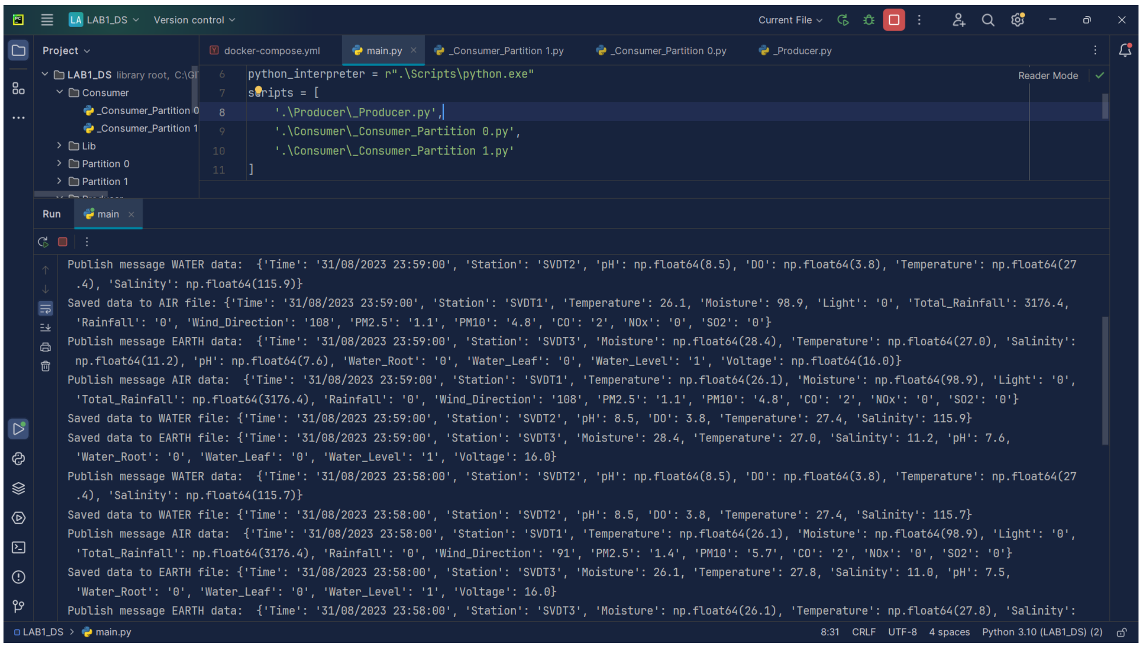
Topic	Consumer Lag
EARTH	0

Partition	Consumer ID	Host	Consumer Lag	Current Offset	End offset
1			0	25	25
0			0	25	25

Topic	Consumer Lag
AIR	0

Partition	Consumer ID	Host	Consumer Lag	Current Offset	End offset
1			0	25	25
0			0	25	25

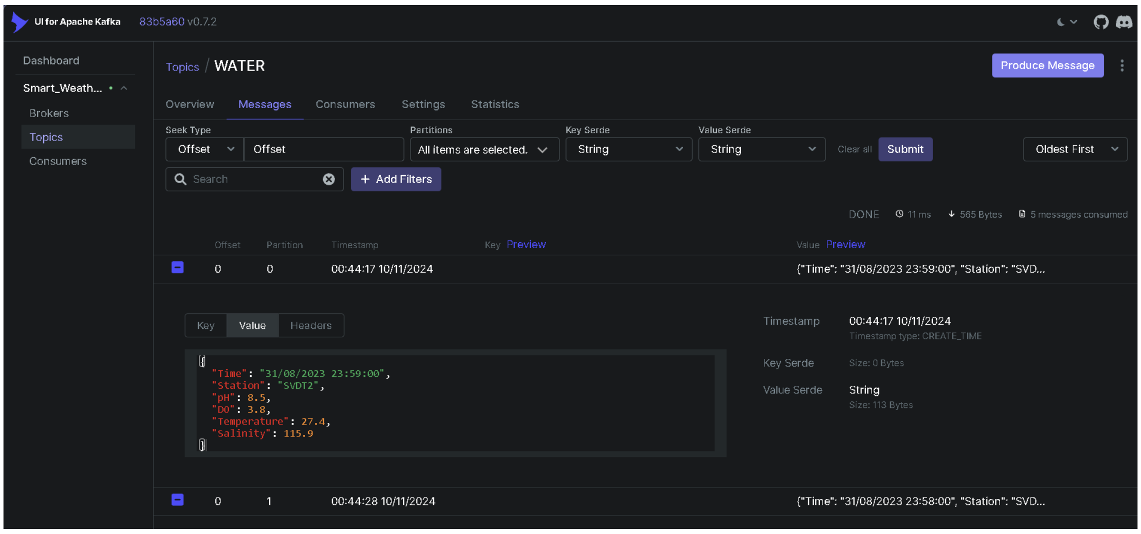
Figure 5: Consumer Data Topics



```
python_interpreter = r".\Scripts\python.exe"
scripts = [
    '.\Producer\_Producer.py',
    '.\Consumer\_Consumer_Partition 0.py',
    '.\Consumer\_Consumer_Partition 1.py'
]
```

```
Publish message WATER data:  {'Time': '31/08/2023 23:59:00', 'Station': 'SVDT2', 'pH': np.float64(8.5), 'DO': np.float64(3.8), 'Temperature': np.float64(27
 .4), 'Salinity': np.float64(115.9)}
Saved data to AIR file: {'Time': '31/08/2023 23:59:00', 'Station': 'SVDT1', 'Temperature': 26.1, 'Moisture': 98.9, 'Light': '0', 'Total_Rainfall': 3176.4,
 'Rainfall': '0', 'Wind_Direction': '108', 'PM2.5': '1.1', 'PM10': '4.8', 'CO': '2', 'NOx': '0', 'SO2': '0'}
Publish message EARTH data:  {'Time': '31/08/2023 23:59:00', 'Station': 'SVDT3', 'Moisture': np.float64(28.4), 'Temperature': np.float64(27.0), 'Salinity':
 np.float64(11.2), 'pH': np.float64(7.6), 'Water_Root': '0', 'Water_Leaf': '0', 'Water_Level': '1', 'Voltage': np.float64(16.0)}
Publish message AIR data:  {'Time': '31/08/2023 23:59:00', 'Station': 'SVDT1', 'Temperature': np.float64(26.1), 'Moisture': np.float64(98.9), 'Light': '0',
 'Total_Rainfall': np.float64(3176.4), 'Rainfall': '0', 'Wind_Direction': '108', 'PM2.5': '1.1', 'PM10': '4.8', 'CO': '2', 'NOx': '0', 'SO2': '0'}
Saved data to WATER file: {'Time': '31/08/2023 23:59:00', 'Station': 'SVDT2', 'pH': 8.5, 'DO': 3.8, 'Temperature': 27.4, 'Salinity': 115.9}
Saved data to EARTH file: {'Time': '31/08/2023 23:59:00', 'Station': 'SVDT3', 'Moisture': 28.4, 'Temperature': 27.0, 'Salinity': 11.2, 'pH': 7.6,
 'Water_Root': '0', 'Water_Leaf': '0', 'Water_Level': '1', 'Voltage': 16.0}
Publish message WATER data:  {'Time': '31/08/2023 23:58:00', 'Station': 'SVDT2', 'pH': np.float64(8.5), 'DO': np.float64(3.8), 'Temperature': np.float64(27
 .4), 'Salinity': np.float64(115.7)}
Saved data to WATER file: {'Time': '31/08/2023 23:58:00', 'Station': 'SVDT2', 'pH': 8.5, 'DO': 3.8, 'Temperature': 27.4, 'Salinity': 115.7}
Publish message AIR data:  {'Time': '31/08/2023 23:58:00', 'Station': 'SVDT1', 'Temperature': np.float64(26.1), 'Moisture': np.float64(98.9), 'Light': '0',
 'Total_Rainfall': np.float64(3176.4), 'Rainfall': '0', 'Wind_Direction': '91', 'PM2.5': '1.4', 'PM10': '5.7', 'CO': '2', 'NOx': '0', 'SO2': '0'}
Saved data to EARTH file: {'Time': '31/08/2023 23:58:00', 'Station': 'SVDT3', 'Moisture': 26.1, 'Temperature': 27.8, 'Salinity': 11.0, 'pH': 7.5,
 'Water_Root': '0', 'Water_Leaf': '0', 'Water_Level': '1', 'Voltage': 16.0}
Publish message EARTH data:  {'Time': '31/08/2023 23:58:00', 'Station': 'SVDT3', 'Moisture': np.float64(26.1), 'Temperature': np.float64(27.8), 'Salinity':
```

Figure 6: CPub/Sub Data Messages



Seek Type: Offset | Offset
Partitions: All items are selected.
Key Serde: String
Value Serde: String
Oldest First

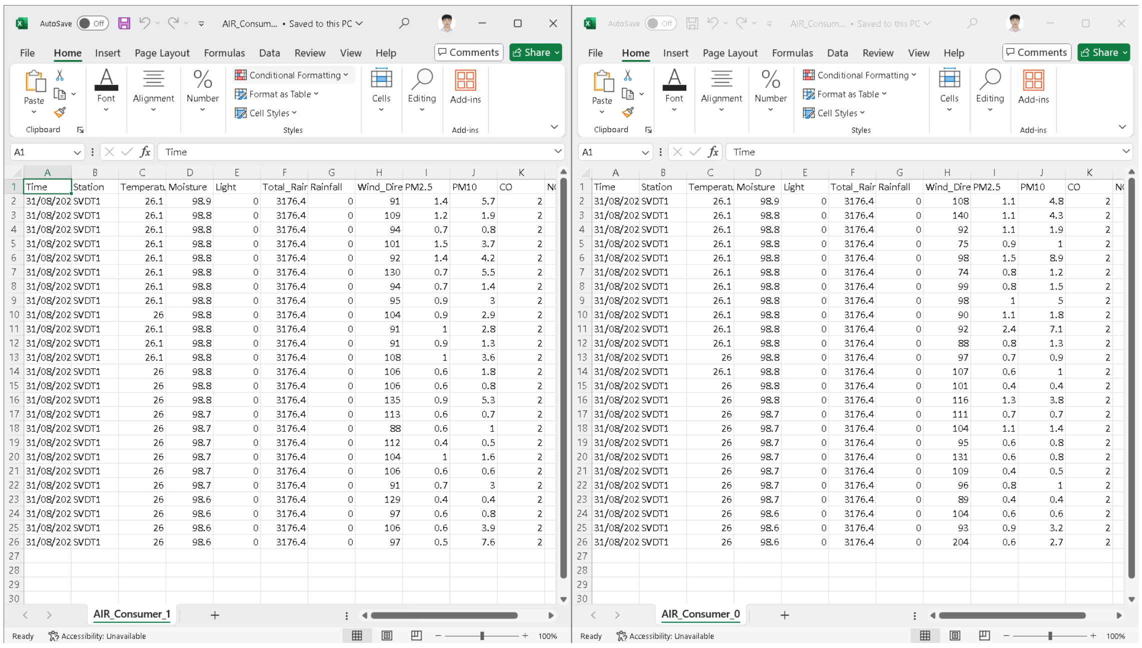
DONE 11 ms 565 Bytes 5 messages consumed

Offset	Partition	Timestamp	Key Preview	Value Preview
0	0	00:44:17 10/11/2024		{"Time": "31/08/2023 23:59:00", "Station": "SVD...
0	1	00:44:28 10/11/2024		{"Time": "31/08/2023 23:58:00", "Station": "SVD...

```
{
  "Time": "31/08/2023 23:59:00",
  "Station": "SVDT2",
  "pH": 8.5,
  "DO": 3.8,
  "Temperature": 27.4,
  "Salinity": 115.9
}
```

Timestamp: 00:44:17 10/11/2024
Timestamp type: CREATE_TIME
Key Serde: Size: 0 Bytes
Value Serde: String
Size: 113 Bytes

Figure 7: Data Kafka receive



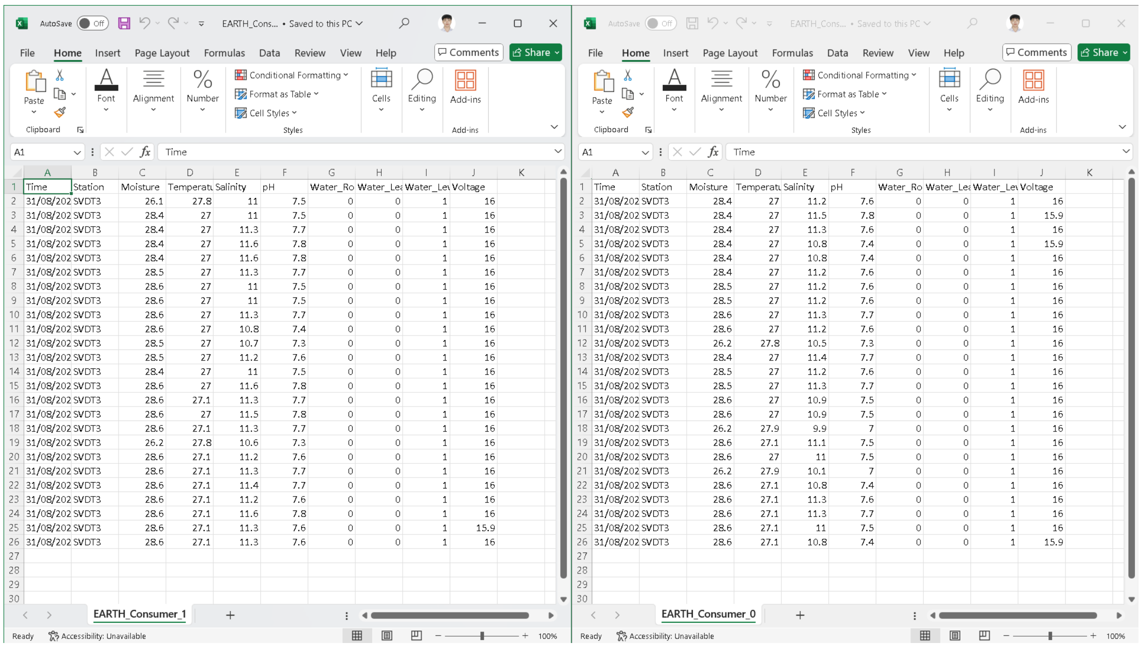
AIR_Consumer_1

	Time	Station	Temperatu	Moisture	Light	Total_Rair	Rainfall	Wind_Dire	PM2.5	PM10	CO
2	31/08/202	SVDT1	26.1	98.9	0	3176.4	0	91	1.4	5.7	2
3	31/08/202	SVDT1	26.1	98.8	0	3176.4	0	109	1.2	1.9	2
4	31/08/202	SVDT1	26.1	98.8	0	3176.4	0	94	0.7	0.8	2
5	31/08/202	SVDT1	26.1	98.8	0	3176.4	0	101	1.5	3.7	2
6	31/08/202	SVDT1	26.1	98.8	0	3176.4	0	92	1.4	4.2	2
7	31/08/202	SVDT1	26.1	98.8	0	3176.4	0	130	0.7	5.5	2
8	31/08/202	SVDT1	26.1	98.8	0	3176.4	0	94	0.7	1.4	2
9	31/08/202	SVDT1	26.1	98.8	0	3176.4	0	95	0.9	3	2
10	31/08/202	SVDT1	26	98.8	0	3176.4	0	104	0.9	2.9	2
11	31/08/202	SVDT1	26.1	98.8	0	3176.4	0	91	1	2.8	2
12	31/08/202	SVDT1	26.1	98.8	0	3176.4	0	91	0.9	1.3	2
13	31/08/202	SVDT1	26.1	98.8	0	3176.4	0	108	1	3.6	2
14	31/08/202	SVDT1	26	98.8	0	3176.4	0	106	0.6	1.8	2
15	31/08/202	SVDT1	26	98.8	0	3176.4	0	106	0.6	0.8	2
16	31/08/202	SVDT1	26	98.8	0	3176.4	0	135	0.9	5.3	2
17	31/08/202	SVDT1	26	98.7	0	3176.4	0	113	0.6	0.7	2
18	31/08/202	SVDT1	26	98.7	0	3176.4	0	88	0.6	1	2
19	31/08/202	SVDT1	26	98.7	0	3176.4	0	112	0.4	0.5	2
20	31/08/202	SVDT1	26	98.7	0	3176.4	0	104	1	1.6	2
21	31/08/202	SVDT1	26	98.7	0	3176.4	0	106	0.6	0.6	2
22	31/08/202	SVDT1	26	98.7	0	3176.4	0	91	0.7	3	2
23	31/08/202	SVDT1	26	98.6	0	3176.4	0	129	0.4	0.4	2
24	31/08/202	SVDT1	26	98.6	0	3176.4	0	97	0.6	0.8	2
25	31/08/202	SVDT1	26	98.6	0	3176.4	0	106	0.6	3.9	2
26	31/08/202	SVDT1	26	98.6	0	3176.4	0	97	0.5	7.6	2

AIR_Consumer_0

	Time	Station	Temperatu	Moisture	Light	Total_Rair	Rainfall	Wind_Dire	PM2.5	PM10	CO
2	31/08/202	SVDT1	26.1	98.9	0	3176.4	0	108	1.1	4.8	2
3	31/08/202	SVDT1	26.1	98.8	0	3176.4	0	140	1.1	4.3	2
4	31/08/202	SVDT1	26.1	98.8	0	3176.4	0	92	1.1	1.9	2
5	31/08/202	SVDT1	26.1	98.8	0	3176.4	0	75	0.9	1	2
6	31/08/202	SVDT1	26.1	98.8	0	3176.4	0	98	1.5	8.9	2
7	31/08/202	SVDT1	26.1	98.8	0	3176.4	0	74	0.8	1.2	2
8	31/08/202	SVDT1	26.1	98.8	0	3176.4	0	99	0.8	1.5	2
9	31/08/202	SVDT1	26.1	98.8	0	3176.4	0	98	1	5	2
10	31/08/202	SVDT1	26.1	98.8	0	3176.4	0	90	1.1	1.8	2
11	31/08/202	SVDT1	26.1	98.8	0	3176.4	0	92	2.4	7.1	2
12	31/08/202	SVDT1	26.1	98.8	0	3176.4	0	88	0.8	1.3	2
13	31/08/202	SVDT1	26	98.8	0	3176.4	0	97	0.7	0.9	2
14	31/08/202	SVDT1	26.1	98.8	0	3176.4	0	107	0.6	1	2
15	31/08/202	SVDT1	26	98.8	0	3176.4	0	101	0.4	0.4	2
16	31/08/202	SVDT1	26	98.8	0	3176.4	0	116	1.3	3.8	2
17	31/08/202	SVDT1	26	98.7	0	3176.4	0	111	0.7	0.7	2
18	31/08/202	SVDT1	26	98.7	0	3176.4	0	104	1.1	1.4	2
19	31/08/202	SVDT1	26	98.7	0	3176.4	0	95	0.6	0.8	2
20	31/08/202	SVDT1	26	98.7	0	3176.4	0	131	0.6	0.8	2
21	31/08/202	SVDT1	26	98.7	0	3176.4	0	109	0.4	0.5	2
22	31/08/202	SVDT1	26	98.7	0	3176.4	0	96	0.8	1	2
23	31/08/202	SVDT1	26	98.7	0	3176.4	0	89	0.4	0.4	2
24	31/08/202	SVDT1	26	98.6	0	3176.4	0	104	0.6	0.6	2
25	31/08/202	SVDT1	26	98.6	0	3176.4	0	93	0.9	3.2	2
26	31/08/202	SVDT1	26	98.6	0	3176.4	0	204	0.6	2.7	2

Figure 8: Output 1



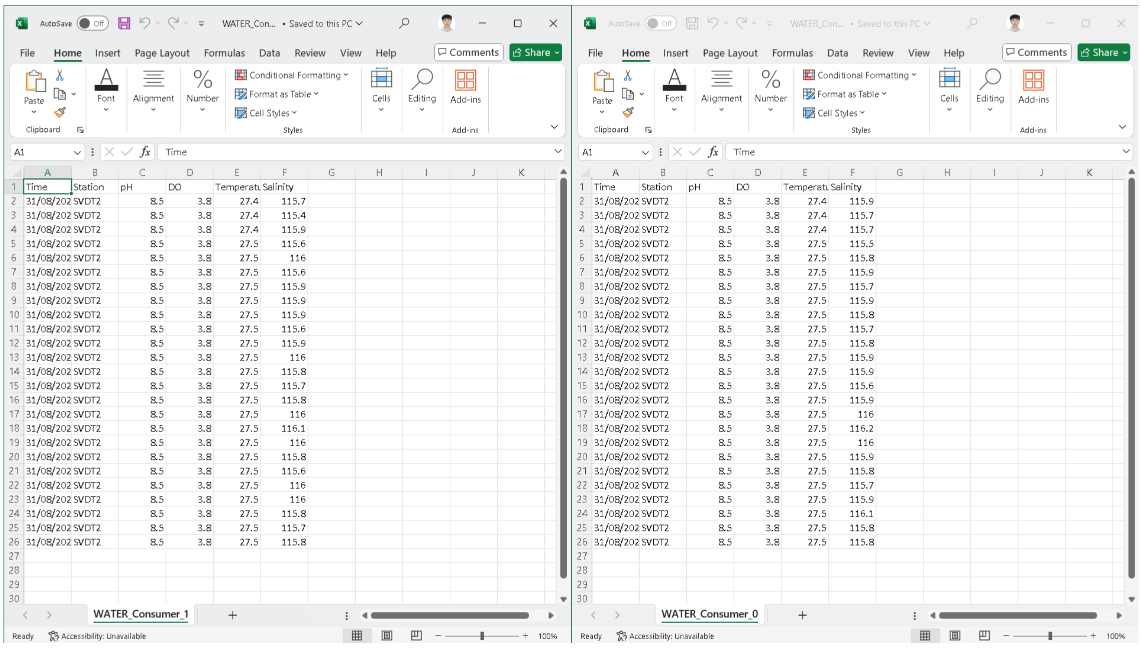
EARTH_Consumer_1

1	Time	Station	Moisture	Temperatu	Salinity	pH	Water_Ro	Water_Lea	Water_Lev	Voltage
2	31/08/202	SVDT3	26.1	27.8	11	7.5	0	0	1	16
3	31/08/202	SVDT3	28.4	27	11	7.5	0	0	1	16
4	31/08/202	SVDT3	28.4	27	11.3	7.7	0	0	1	16
5	31/08/202	SVDT3	28.4	27	11.6	7.8	0	0	1	16
6	31/08/202	SVDT3	28.4	27	11.6	7.8	0	0	1	16
7	31/08/202	SVDT3	28.5	27	11.3	7.7	0	0	1	16
8	31/08/202	SVDT3	28.6	27	11	7.5	0	0	1	16
9	31/08/202	SVDT3	28.6	27	11	7.5	0	0	1	16
10	31/08/202	SVDT3	28.6	27	11.3	7.7	0	0	1	16
11	31/08/202	SVDT3	28.6	27	10.8	7.4	0	0	1	16
12	31/08/202	SVDT3	28.5	27	10.7	7.3	0	0	1	16
13	31/08/202	SVDT3	28.5	27	11.2	7.6	0	0	1	16
14	31/08/202	SVDT3	28.4	27	11	7.5	0	0	1	16
15	31/08/202	SVDT3	28.6	27	11.6	7.8	0	0	1	16
16	31/08/202	SVDT3	28.6	27.1	11.3	7.7	0	0	1	16
17	31/08/202	SVDT3	28.6	27	11.5	7.8	0	0	1	16
18	31/08/202	SVDT3	28.6	27.1	11.3	7.7	0	0	1	16
19	31/08/202	SVDT3	26.2	27.8	10.6	7.3	0	0	1	16
20	31/08/202	SVDT3	28.6	27.1	11.2	7.6	0	0	1	16
21	31/08/202	SVDT3	28.6	27.1	11.3	7.7	0	0	1	16
22	31/08/202	SVDT3	28.6	27.1	11.4	7.7	0	0	1	16
23	31/08/202	SVDT3	28.6	27.1	11.2	7.6	0	0	1	16
24	31/08/202	SVDT3	28.6	27.1	11.6	7.8	0	0	1	16
25	31/08/202	SVDT3	28.6	27.1	11.3	7.6	0	0	1	15.9
26	31/08/202	SVDT3	28.6	27.1	11.3	7.6	0	0	1	16

EARTH_Consumer_0

1	Time	Station	Moisture	Temperatu	Salinity	pH	Water_Ro	Water_Lea	Water_Lev	Voltage
2	31/08/202	SVDT3	28.4	27	11.2	7.6	0	0	1	16
3	31/08/202	SVDT3	28.4	27	11.5	7.8	0	0	1	15.9
4	31/08/202	SVDT3	28.4	27	11.3	7.6	0	0	1	16
5	31/08/202	SVDT3	28.4	27	10.8	7.4	0	0	1	15.9
6	31/08/202	SVDT3	28.4	27	10.8	7.4	0	0	1	16
7	31/08/202	SVDT3	28.4	27	11.2	7.6	0	0	1	16
8	31/08/202	SVDT3	28.5	27	11.2	7.6	0	0	1	16
9	31/08/202	SVDT3	28.5	27	11.2	7.6	0	0	1	16
10	31/08/202	SVDT3	28.6	27	11.3	7.7	0	0	1	16
11	31/08/202	SVDT3	28.6	27	11.2	7.6	0	0	1	16
12	31/08/202	SVDT3	26.2	27.8	10.5	7.3	0	0	1	16
13	31/08/202	SVDT3	28.4	27	11.4	7.7	0	0	1	16
14	31/08/202	SVDT3	28.5	27	11.2	7.6	0	0	1	16
15	31/08/202	SVDT3	28.5	27	11.3	7.7	0	0	1	16
16	31/08/202	SVDT3	28.6	27	10.9	7.5	0	0	1	16
17	31/08/202	SVDT3	28.6	27	10.9	7.5	0	0	1	16
18	31/08/202	SVDT3	26.2	27.9	9.9	7	0	0	1	16
19	31/08/202	SVDT3	28.6	27.1	11.1	7.5	0	0	1	16
20	31/08/202	SVDT3	28.6	27	11	7.5	0	0	1	16
21	31/08/202	SVDT3	26.2	27.9	10.1	7	0	0	1	16
22	31/08/202	SVDT3	28.6	27.1	10.8	7.4	0	0	1	16
23	31/08/202	SVDT3	28.6	27.1	11.3	7.6	0	0	1	16
24	31/08/202	SVDT3	28.6	27.1	11.3	7.7	0	0	1	16
25	31/08/202	SVDT3	28.6	27.1	11	7.5	0	0	1	16
26	31/08/202	SVDT3	28.6	27.1	10.8	7.4	0	0	1	15.9

Figure 9: Output 2



WATER_Consumer_1

1	Time	Station	pH	DO	Temperatu	Salinity
2	31/08/202	SVDT2	8.5	3.8	27.4	115.7
3	31/08/202	SVDT2	8.5	3.8	27.4	115.4
4	31/08/202	SVDT2	8.5	3.8	27.4	115.9
5	31/08/202	SVDT2	8.5	3.8	27.5	115.6
6	31/08/202	SVDT2	8.5	3.8	27.5	116
7	31/08/202	SVDT2	8.5	3.8	27.5	115.6
8	31/08/202	SVDT2	8.5	3.8	27.5	115.9
9	31/08/202	SVDT2	8.5	3.8	27.5	115.9
10	31/08/202	SVDT2	8.5	3.8	27.5	115.9
11	31/08/202	SVDT2	8.5	3.8	27.5	115.6
12	31/08/202	SVDT2	8.5	3.8	27.5	115.9
13	31/08/202	SVDT2	8.5	3.8	27.5	116
14	31/08/202	SVDT2	8.5	3.8	27.5	115.8
15	31/08/202	SVDT2	8.5	3.8	27.5	115.7
16	31/08/202	SVDT2	8.5	3.8	27.5	115.8
17	31/08/202	SVDT2	8.5	3.8	27.5	116
18	31/08/202	SVDT2	8.5	3.8	27.5	116.1
19	31/08/202	SVDT2	8.5	3.8	27.5	116
20	31/08/202	SVDT2	8.5	3.8	27.5	115.8
21	31/08/202	SVDT2	8.5	3.8	27.5	115.6
22	31/08/202	SVDT2	8.5	3.8	27.5	116
23	31/08/202	SVDT2	8.5	3.8	27.5	116
24	31/08/202	SVDT2	8.5	3.8	27.5	115.8
25	31/08/202	SVDT2	8.5	3.8	27.5	115.7
26	31/08/202	SVDT2	8.5	3.8	27.5	115.8

WATER_Consumer_0

1	Time	Station	pH	DO	Temperatu	Salinity
2	31/08/202	SVDT2	8.5	3.8	27.4	115.9
3	31/08/202	SVDT2	8.5	3.8	27.4	115.7
4	31/08/202	SVDT2	8.5	3.8	27.4	115.7
5	31/08/202	SVDT2	8.5	3.8	27.5	115.5
6	31/08/202	SVDT2	8.5	3.8	27.5	115.8
7	31/08/202	SVDT2	8.5	3.8	27.5	115.9
8	31/08/202	SVDT2	8.5	3.8	27.5	115.7
9	31/08/202	SVDT2	8.5	3.8	27.5	115.9
10	31/08/202	SVDT2	8.5	3.8	27.5	115.8
11	31/08/202	SVDT2	8.5	3.8	27.5	115.7
12	31/08/202	SVDT2	8.5	3.8	27.5	115.8
13	31/08/202	SVDT2	8.5	3.8	27.5	115.9
14	31/08/202	SVDT2	8.5	3.8	27.5	115.9
15	31/08/202	SVDT2	8.5	3.8	27.5	115.6
16	31/08/202	SVDT2	8.5	3.8	27.5	115.9
17	31/08/202	SVDT2	8.5	3.8	27.5	116
18	31/08/202	SVDT2	8.5	3.8	27.5	116.2
19	31/08/202	SVDT2	8.5	3.8	27.5	116
20	31/08/202	SVDT2	8.5	3.8	27.5	115.9
21	31/08/202	SVDT2	8.5	3.8	27.5	115.8
22	31/08/202	SVDT2	8.5	3.8	27.5	115.7
23	31/08/202	SVDT2	8.5	3.8	27.5	115.9
24	31/08/202	SVDT2	8.5	3.8	27.5	116.1
25	31/08/202	SVDT2	8.5	3.8	27.5	115.8
26	31/08/202	SVDT2	8.5	3.8	27.5	115.8

Figure 10: Output 3