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 5 REPORT

DISTRIBUTED SYSTEMS

Instructor: **PROF. Nguyễn Mạnh Thìn**

Tạ Gia Khang - 2152642
Nguyễn Quang Thiện - 2152994

HO CHI MINH CITY, DECEMBER 2024

Contents

1 Homework

1.1 Configuration

Using Docker and generating a docker-compose.yml file to create 3 brokers for **Kafka Fog**, **Spark Master**, **2 Spark Workers**, **a Hadoop Namenode**, and **3 Datanodes**.

```yaml
version: '2'
services:
# kafka-fog
  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:
      - lab5_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'
```

```yaml
      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:
      - lab5_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:
      - lab5_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:
      - lab5_ds_kafka_network

  spark-master:
    image: bitnami/spark:latest
    container_name: spark-master
    environment:
      - SPARK_MODE=master
      - SPARK_RPC_AUTHENTICATION_ENABLED=no
      - SPARK_RPC_ENCRYPTION_ENABLED=no
      - SPARK_LOCAL_STORAGE_ENCRYPTION_ENABLED=no
      - SPARK_SSL_ENABLED=no
    ports:
      - "7077:7077" # Spark Master port
    networks:
      -  lab5_ds_kafka_network
    volumes:
      - ./pyspark:/opt/spark/python

  spark-worker1:
    image: bitnami/spark:latest
    container_name: spark-worker1
    environment:
      - SPARK_MODE=worker
      - SPARK_MASTER_URL=spark://spark-master:7077
    depends_on:
      - spark-master
    networks:
      -  lab5_ds_kafka_network

  spark-worker2:
    image: bitnami/spark:latest
    container_name: spark-worker2
    environment:
      - SPARK_MODE=worker
      - SPARK_MASTER_URL=spark://spark-master:7077
```

```
    depends_on:
      - spark-master
    networks:
      -  lab5_ds_kafka_network

  namenode:
    image: bde2020/hadoop-namenode:latest
    container_name: namenode
    environment:
      - CLUSTER_NAME=hadoop_cluster
      - CORE_CONF_fs_defaultFS=hdfs://namenode:9000
    ports:
      - "9000:9000" # HDFS namenode RPC port
      - "9870:9870" # HDFS namenode Web UI
    networks:
      -  lab5_ds_kafka_network
    volumes:
      - ./namenode:/hadoop/dfs/name

  datanode:
    image: bde2020/hadoop-datanode:latest
    container_name: datanode
    environment:
      - CORE_CONF_fs_defaultFS=hdfs://namenode:9000
    depends_on:
      - namenode
    networks:
      -  lab5_ds_kafka_network
    volumes:
      - ./datanode:/hadoop/dfs/data

  datanode2:
    image: bde2020/hadoop-datanode:latest
    container_name: datanode2
    environment:
      - CORE_CONF_fs_defaultFS=hdfs://namenode:9000
    depends_on:
      - namenode
    networks:
      -  lab5_ds_kafka_network
    volumes:
      - ./datanode2:/hadoop/dfs/data

  datanode3:
    image: bde2020/hadoop-datanode:latest
    container_name: datanode3
    environment:
```

```yaml
        - CORE_CONF_fs_defaultFS=hdfs://namenode:9000
      depends_on:
        - namenode
      networks:
        -  lab5_ds_kafka_network
      volumes:
        - ./datanode3:/hadoop/dfs/data

networks:
  lab5_ds_kafka_network:
    external: true

volumes:
  # Defines volumes for persistent storage of Namenode and Datanode data
  namenode:
  datanode:
  datanode2:
  datanode3:
```

After that, create the network **lab5_ds_kafka_network** by the cmd below in Docker:

```
docker network create lab5_ds_kafka_network
```

1.2 Previous LAB's Output

```
Check Output:
  docker exec -it namenode bash -c 'hdfs dfs -ls /hdfs/environment_data'
  docker exec -it namenode bash -c 'hdfs dfs -cat /hdfs/environment_data
    /*.json'
```

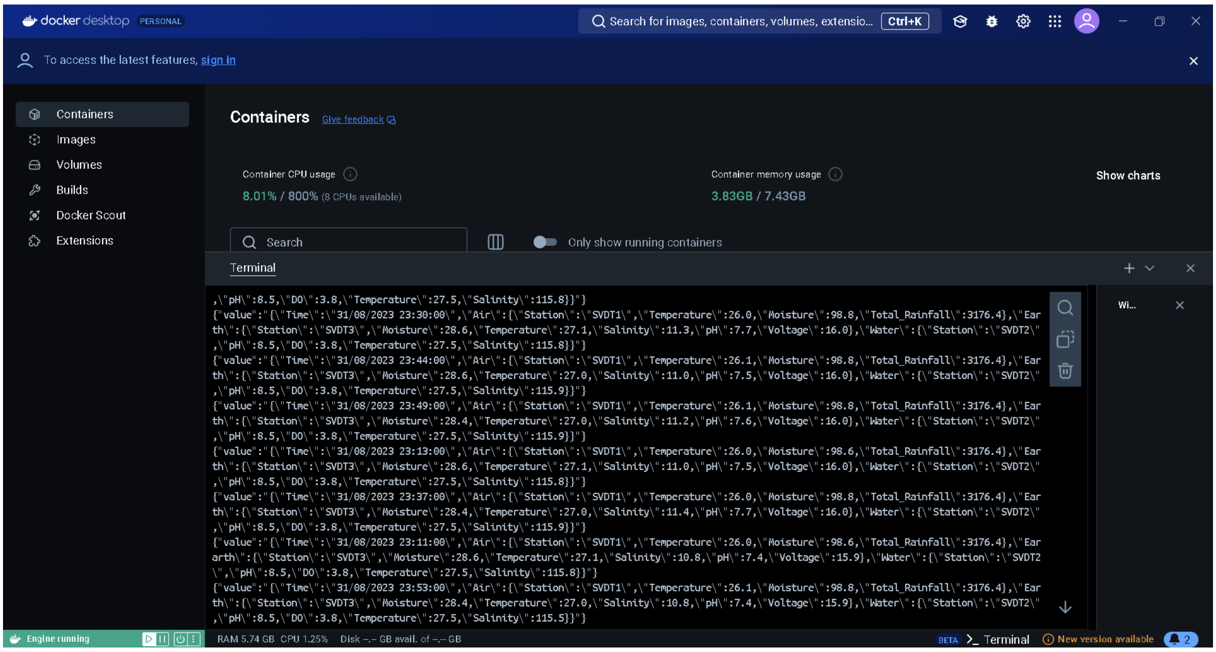

Figure 1: LAB4's Output

1.3 Loading Data

This section focuses on initializing the SparkSession and loading the environmental data stored in HDFS for distributed processing. The primary goal is to prepare the data for analysis of mango growth conditions.

- `SparkSession` serves as the entry point for PySpark applications, enabling distributed processing of large datasets. The session is named *"Distributed Computing for Mango Growth Evaluation"* for easy identification.
- The input data, stored as JSON files which contains information about air temperature, air moisture, and soil pH in HDFS at `hdfs://namenode:9000/hdfs/environment_data/*.json`.
- A schema is defined using `StructType` to parse the nested JSON structure accurately. It includes:
  - *Air*: Contains fields for temperature and moisture.
  - *Earth*: Contains the field for soil pH.
- The loaded data is parsed using `from_json`, and generating structured columns `Air_Temperature`, `Air_Moisture`, and `Earth_pH`, which are really vital for the subsequent analysis.

The parsed and structured dataset enables distributed computation while maintaining scalability and fault tolerance, leveraging the advantages of HDFS and Spark for efficient data handling.

```python
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, from_json, when, lit, avg, max, min
from pyspark.sql.types import StructType, StructField, DoubleType, StringType

# Initialize SparkSession
spark = SparkSession.builder \
    .appName("Distributed Computing for Mango Growth Evaluation") \
    .getOrCreate()

# HDFS Path
input_path = "hdfs://namenode:9000/hdfs/environment_data/*.json"
output_path = "hdfs://namenode:9000/hdfs/mango_data_distributed.txt"

# Define Schema for Nested JSON
schema = StructType([
    StructField("Air", StructType([
        StructField("Temperature", DoubleType()),
        StructField("Moisture", DoubleType())
    ])),
    StructField("Earth", StructType([
        StructField("pH", DoubleType())
    ]))
])

# Define Ideal Conditions
ideal_conditions = {
```

```
    "Air_Temperature": (23.0, 28.0),
    "Air_Moisture": (60.0, 80.0),
    "Earth_pH": (5.5, 7.0)
}

def process_data():
    # Load JSON Data and collect to driver
    df = spark.read.json(input_path)

    # Parse the "value" field
    parsed_df = df.withColumn("parsed_value", from_json(col("value"), schema))

    # Extract the parsed fields
    data_df = parsed_df.select(
        col("parsed_value.Air.Temperature").alias("Air_Temperature"),
        col("parsed_value.Air.Moisture").alias("Air_Moisture"),
        col("parsed_value.Earth.pH").alias("Earth_pH")
    )
```

1.4 Bring data to compute (Centralized Computing)

This section explains the centralized computation approach, where the data is first collected from the distributed HDFS system and processed locally.

- **Data Collection:** All records from the distributed dataset are gathered to the driver node using the `collect()` method. This transforms the distributed data into a local collection, enabling computation to occur in a centralized environment.
- **Identifying Out-of-Ideal Conditions:** The script evaluates whether environmental parameters (`Air_Temperature`, `Air_Moisture`, `Earth_pH`) fall outside their ideal ranges:
  - `Air_Temperature`: Between 23°C and 28°C.
  - `Air_Moisture`: Between 60% and 80%.
  - `Earth_pH`: Between 5.5 and 7.0.

  Each parameter is flagged with a `1` if it is out of range and `0` otherwise.
- **Real-Time Monitoring:** Using `show()` function outputs the DataFrame to monitor conditions in real-time, providing insights into the current state of the environment.
- **Time Analysis:** The total number of rows (representing minutes) where one or more parameters are out of range is counted. This is converted into hours and minutes for ease of understanding.
- **Statistical Analysis:** Using `agg()` function, the maximum and minimum values for each parameter are computed to assess the range and extremes of environmental conditions, offering valuable insights into the dataset's variability.

This method highlights the trade-off of centralized computing: while simple to implement, it may not scale efficiently for large datasets due to data transfer overheads.

```python
# Collect all data to driver node
collected_data = data_df.collect()

# Convert collected data to a new local DataFrame
local_df = spark.createDataFrame(collected_data)

# Calculate Conditions Out of Ideal
out_of_ideal_df = local_df.withColumn(
    "Air_Temperature_Out",
    when((col("Air_Temperature") <
    lit(ideal_conditions["Air_Temperature"][0])) |
         (col("Air_Temperature") >
         lit(ideal_conditions["Air_Temperature"][1])), 1).otherwise(0)
).withColumn(
    "Air_Moisture_Out",
    when((col("Air_Moisture") < lit(ideal_conditions["Air_Moisture"][0])) |
         (col("Air_Moisture") > lit(ideal_conditions["Air_Moisture"][1])),
         1).otherwise(0)
).withColumn(
    "Earth_pH_Out",
```

```
        when((col("Earth_pH") < lit(ideal_conditions["Earth_pH"][0])) |
             (col("Earth_pH") > lit(ideal_conditions["Earth_pH"][1])),
             1).otherwise(0)
    )

    # Calculate time out of ideal conditions locally
    out_of_ideal_count = out_of_ideal_df.filter(
        (col("Air_Temperature_Out") + col("Air_Moisture_Out") +
        col("Earth_pH_Out")) >= 1
    ).count()

    # Convert from minutes to hours, minutes
    hours = out_of_ideal_count // 60
    minutes = out_of_ideal_count % 60

    # Calculate statistics locally
    stats = local_df.agg(
        max("Air_Temperature").alias("Max_Air_Temperature"),
        min("Air_Temperature").alias("Min_Air_Temperature"),
        max("Air_Moisture").alias("Max_Air_Moisture"),
        min("Air_Moisture").alias("Min_Air_Moisture"),
        max("Earth_pH").alias("Max_Earth_pH"),
        min("Earth_pH").alias("Min_Earth_pH")
    ).collect()[0]
```

1.5 Bring compute to data (Distributed Computing)

This section demonstrates the distributed computing approach, where the computation is brought to the data stored in HDFS, reducing data transfer overhead and leveraging distributed resources.

- **Identifying Out-of-Ideal Conditions:** Similar to the centralized approach, new columns are added to evaluate whether `Air_Temperature`, `Air_Moisture`, and `Earth_pH` fall outside their respective ideal ranges. This computation occurs in parallel across the distributed nodes, enhancing efficiency.
  - `Air_Temperature`: Ideal range is between 23°C and 28°C.
  - `Air_Moisture`: Ideal range is between 60% and 80%.
  - `Earth_pH`: Ideal range is between 5.5 and 7.0.
- **Real-Time Monitoring:** `show()` function is also utilized to display the DataFrame, allowing real-time observation of the conditions and offering insights into the current environmental state.
- **Time Analysis:** The script calculates the total time (in minutes) where one or more parameters are outside the ideal range. This is further converted to hours and minutes for better interpretability.
- **Statistical Summary:** The maximum and minimum values for each parameter are computed using Spark's `agg()` function. These calculations are distributed across the cluster, reducing processing time while handling large-scale data.

This approach effectively utilizes distributed computing capabilities to analyze environmental data at scale, minimizing the cost of moving large datasets.

```python
# Calculate Conditions Out of Ideal
out_of_ideal_df = data_df.withColumn(
    "Air_Temperature_Out",
    when((col("Air_Temperature") < lit(ideal_conditions["Air_Temperature"][0])) |
         (col("Air_Temperature") > lit(ideal_conditions["Air_Temperature"][1])),
         1).otherwise(0)
).withColumn(
    "Air_Moisture_Out",
    when((col("Air_Moisture") < lit(ideal_conditions["Air_Moisture"][0])) |
         (col("Air_Moisture") > lit(ideal_conditions["Air_Moisture"][1])),
         1).otherwise(0)
).withColumn(
    "Earth_pH_Out",
    when((col("Earth_pH") < lit(ideal_conditions["Earth_pH"][0])) |
         (col("Earth_pH") > lit(ideal_conditions["Earth_pH"][1])), 1).otherwise(0)
)

# Show the DataFrame
out_of_ideal_df.show()

# Calculate Average Time Out of Ideal Conditions
avg_out_of_ideal = out_of_ideal_df.filter((col("Air_Temperature_Out") +
col("Air_Moisture_Out") + col("Earth_pH_Out")) >= 1).count()

```

```
# Convert from minutes to hours, minutes, seconds
avg_out_of_ideal_hours = avg_out_of_ideal // 60
avg_out_of_ideal_minutes = avg_out_of_ideal % 60

# Calculate Max and Min for Each Field
stats_df = data_df.agg(
    max("Air_Temperature").alias("Max_Air_Temperature"),
    min("Air_Temperature").alias("Min_Air_Temperature"),
    max("Air_Moisture").alias("Max_Air_Moisture"),
    min("Air_Moisture").alias("Min_Air_Moisture"),
    max("Earth_pH").alias("Max_Earth_pH"),
    min("Earth_pH").alias("Min_Earth_pH")
).collect()[0]
```

1.6 Store Output

The final step involves saving the processed results back to HDFS in a structured format for future use and further analysis.

- **Formatting Results:** The computed statistics and time metrics are formatted into a readable string that summarizes the analysis.
- **Storing Results:** The function `saveAsTextFile()` writes the formatted results as a single text file to the specified HDFS directory. This ensures the outputs are accessible and reusable for downstream processes.
- **Finalization:** SparkSession is terminated to free up resources, signifying the completion of the computation process.

This step ensures that all insights derived from the distributed computing process are preserved in a distributed storage system, enabling easy sharing and accessibility.

```
    # Format results
    results = (
        f"Average Time Out of Ideal: {hours} hours {minutes} minutes\n"
        f"Max Air Temperature: {stats['Max_Air_Temperature']}°C\n"
        f"Min Air Temperature: {stats['Min_Air_Temperature']}°C\n"
        f"Max Air Moisture: {stats['Max_Air_Moisture']}%\n"
        f"Min Air Moisture: {stats['Min_Air_Moisture']}%\n"
        f"Max Earth pH: {stats['Max_Earth_pH']}\n"
        f"Min Earth pH: {stats['Min_Earth_pH']}\n"
    )

    # Write results to HDFS as a single text file
    spark.sparkContext.parallelize([results]).saveAsTextFile(output_path)

# Process the data
process_data()

# Stop SparkSession
spark.stop()
```

1.7 Demonstration

1.7.1 Centralized Computing Method

```
Push code to Spark:
  docker cp data_to_compute.py spark-master:/opt/spark/python/
    data_to_compute.py
Run code:
  docker exec -it spark-master bash
  spark-submit \
    --packages org.apache.spark:spark-sql-kafka-0-10_2.12:3.5.3 \
    --master spark://spark-master:7077 \
  /opt/spark/python/data_to_compute.py
```

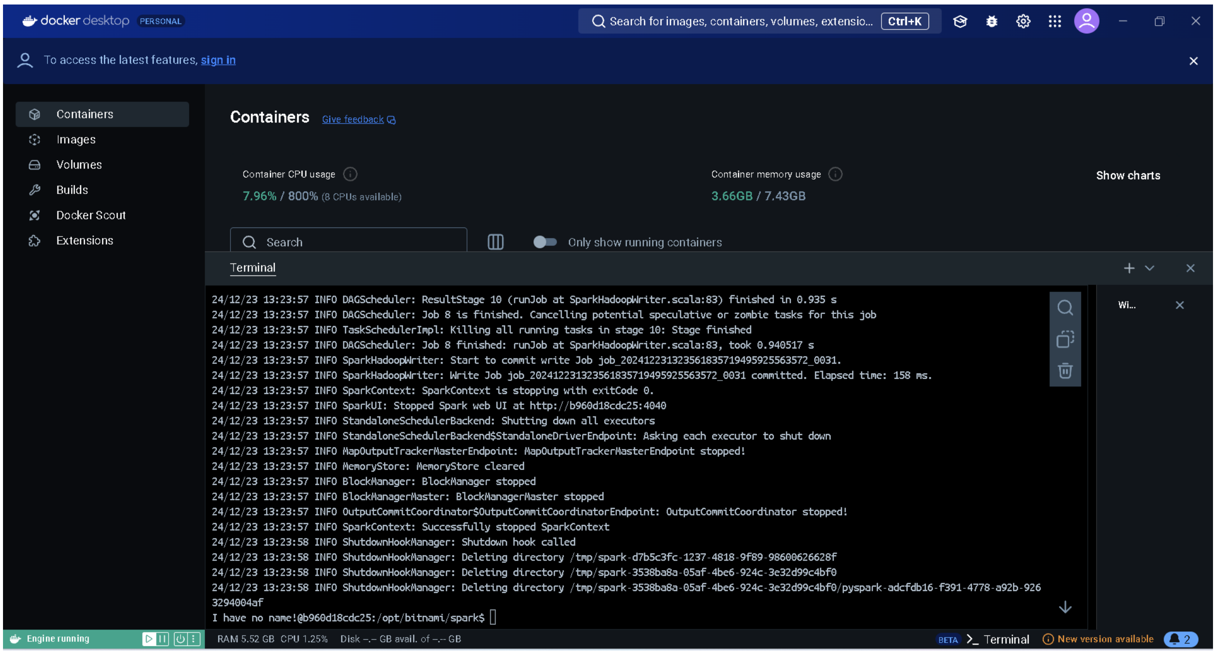

Figure 2: Finish Centralized Computing

```
Check Output:
  docker exec -it namenode bash -c 'hdfs dfs -cat /hdfs/
    mango_data_centralized.txt/part-*'
```

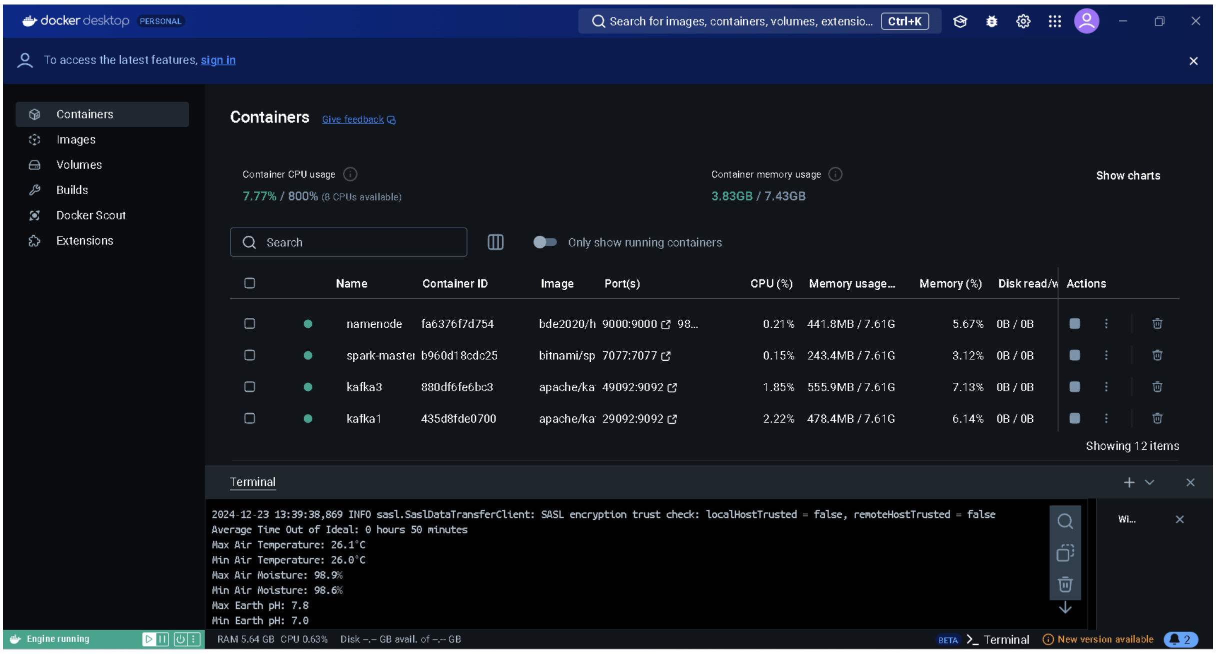

Figure 3: Output Centralized Computing

1.7.2 Distributed Computing Method

```
Push code to Spark:
  docker cp compute_to_data.py spark-master:/opt/spark/python/
    compute_to_data.py
Run code:
  docker exec -it spark-master bash
  spark-submit \
    --packages org.apache.spark:spark-sql-kafka-0-10_2.12:3.5.3 \
    --master spark://spark-master:7077 \
  /opt/spark/python/compute_to_data.py
```

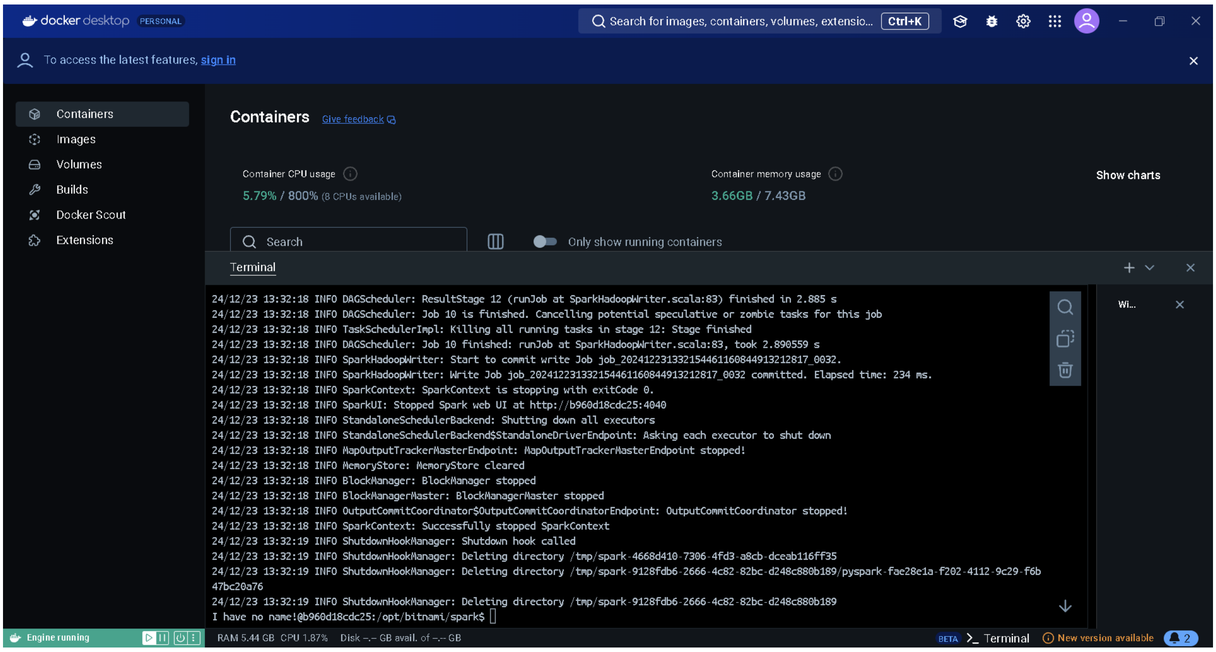

Figure 4: Finish Distributed Computing

```
Check Output:
  docker exec -it namenode bash -c 'hdfs dfs -cat /hdfs/
   mango_data_distributed.txt/part-*'
```

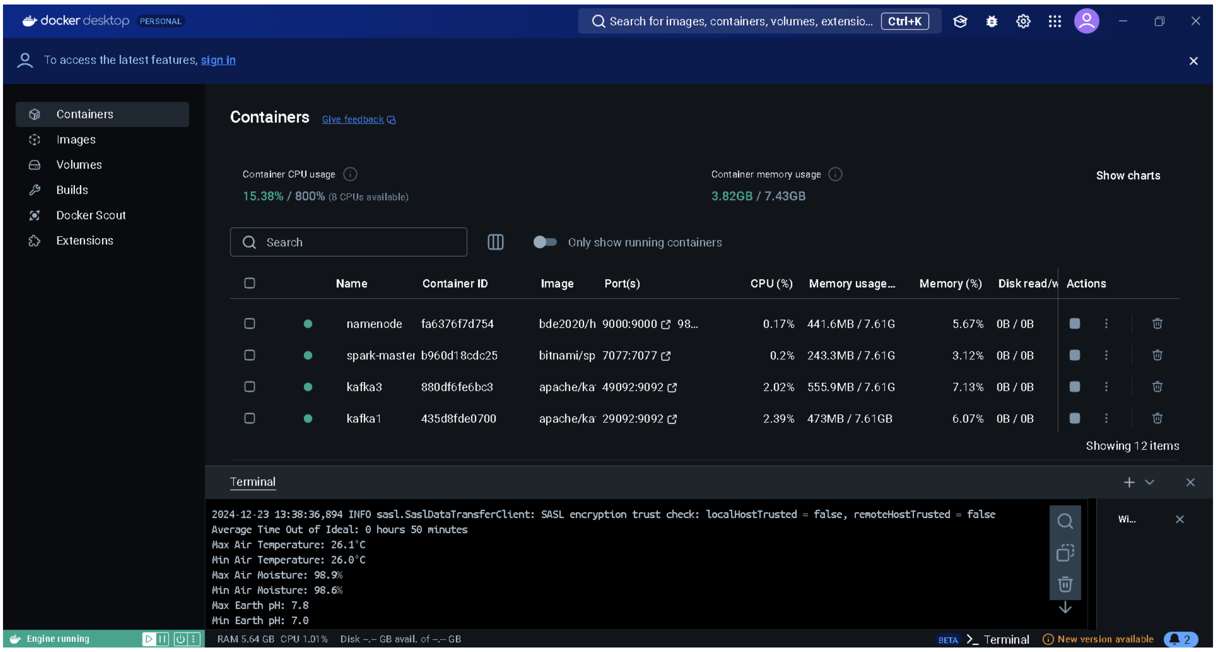

Figure 5: Output Distributed Computing

Observing the two output images generated by the two methods reveals that the results of both approaches are identical.