Week 7 Assignment

Guruprasad Velikadu Krishnamoorthy

College of Science and Technology, Bellevue University

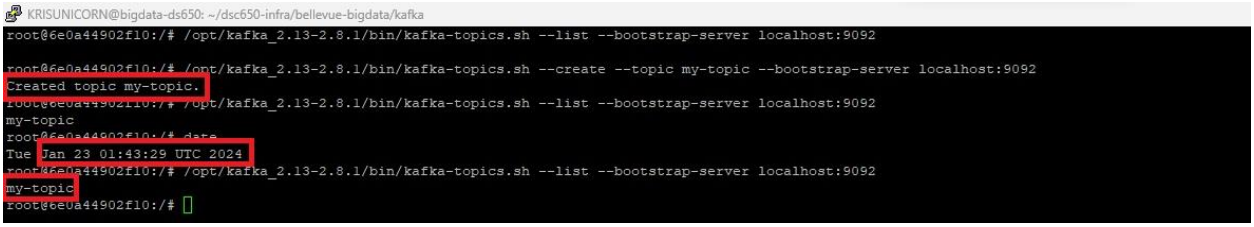
DSC650-T301: Big Data

Professor. Nasheb Ismaily

January 25, 2024

1. Topic Creation and Verification in Kafka :

Command used: /opt/kafka_2.13-2.8.1/bin/kafka-topics.sh --create --topic my-topic --bootstrap-server localhost:9092



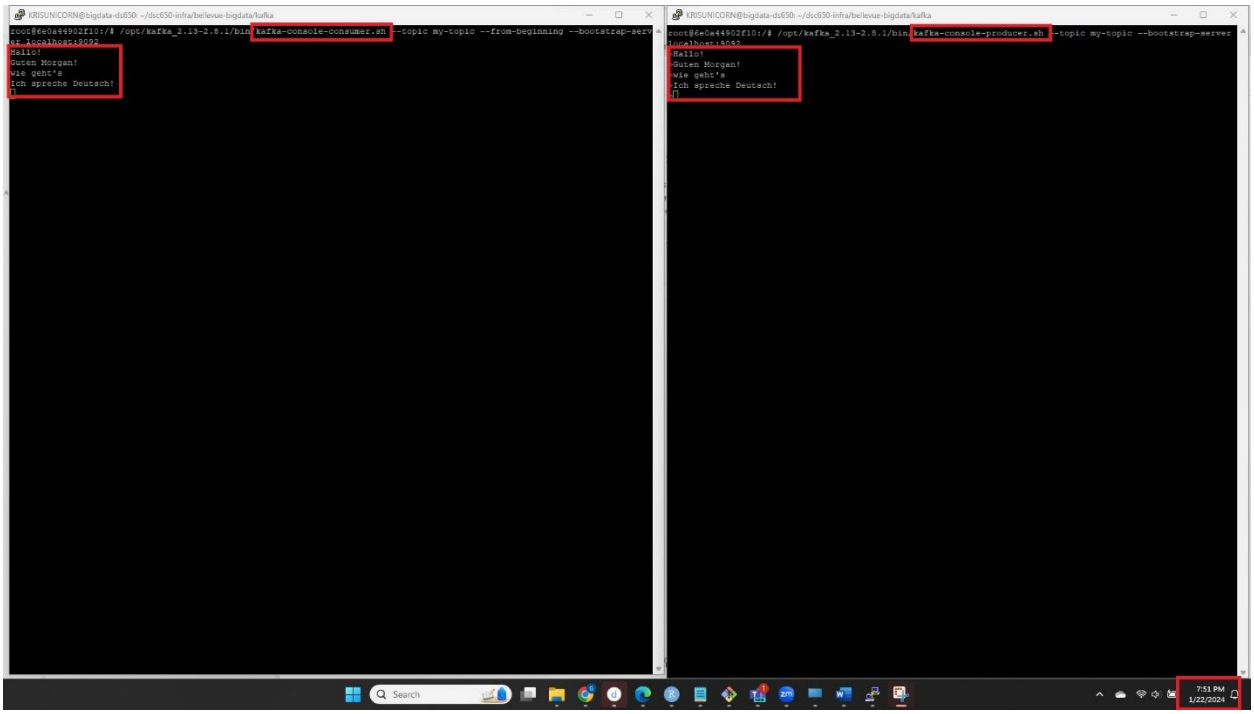
Significance: The command is to create a topic named my-topic in kafka. The argument – bootstrap-server is required and denotes the Kafka server to connect to.

2. Producing and Consuming Messages in Kafka

Commands used:

/opt/kafka_2.13-2.8.1/bin/kafka-console-producer.sh --topic my-topic --bootstrap-server localhost:9092

/opt/kafka_2.13-2.8.1/bin/kafka-console-consumer.sh --topic my-topic --from-beginning --bootstrap-server localhost:9092



The screenshot shows the Producer and consumer displays the same message.

Significance: The producer script helps to read data from standard input and publish it to Kafka. The consumer script helps to read from Kafka and publish it to the standard output. The *--from-beginning* indicates that if an offset is not mentioned for the consumer, start reading from the earliest message.

3. Kafka Performance Tests:

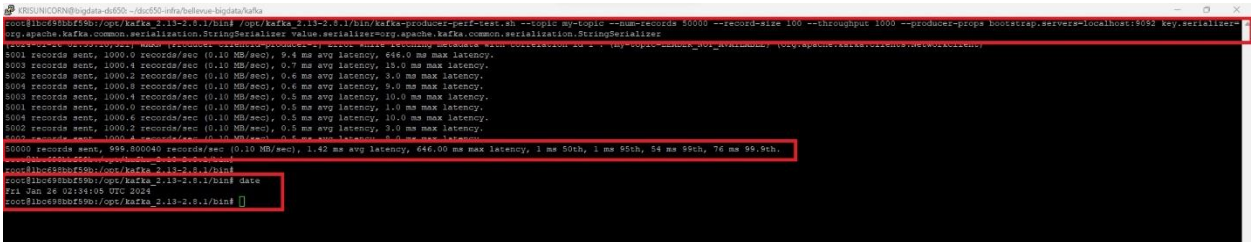
3.1. Producer Performance test:

```
/opt/kafka_2.13-2.8.1/bin/kafka-producer-perf-test.sh --topic my-topic
--num-records 50000 --record-size 100 --throughput 1000 --producer-prop
s bootstrap.servers=localhost:9092 key.serializer=org.apache.kafka.comm
on.serialization.StringSerializer value.serializer=org.apache.kafka.com
mon.serialization.StringSerializer
```

Significance of the command:

As the name suggests it is used to verify the producer performance. The argument *--num-records* indicates the number of messages to produce. *--record size* indicates the message size in bytes. *–throughput* indicates the maximum messages/sec throttling limit. *--producer-props* indicates the producer-related configuration such as bootstrap servers, client ID

Screenshot of the producer output:



50000 records sent, 999.800040 records/sec (0.10 MB/sec), 1.42 ms avg latency, 646.00 ms max latency, 1 ms 50th, 1 ms 95th, 54 ms 99th, 76 ms 99.9th.

Meaning of the Producer Output:

50k records were sent of which approximately 1000 records were produced per second on average. The maximum latency was 646 milliseconds and average latency was 1.42 milliseconds.

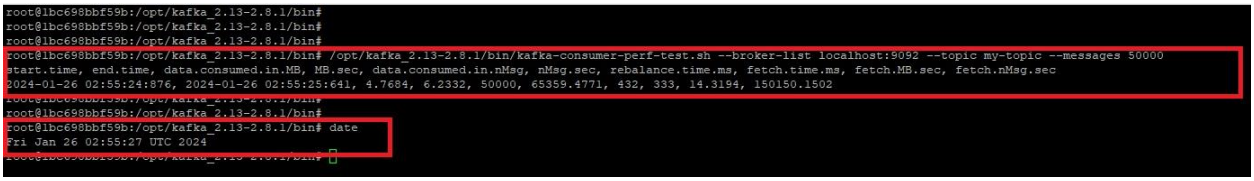
The 50th or 95th percentile of 1 millisecond denotes that 50% or 95% of the messages took less than 1 millisecond between the time they were produced and when they were written in the broker filesystem.

About 1% of messages (99th percentile) had a delay of 54 milliseconds between the time they were produced and published in the broker filesystem. About 0.1% of the messages had a delay of 76 milliseconds (99.9th percentile)

3.2. Consumer Performance Test:

```
/opt/kafka_2.13-2.8.1/bin/kafka-consumer-perf-test.sh --broker-list loc
alhost:9092 --topic my-topic --messages 50000
```

Screenshot of the output:



start.time, end.time, data.consumed.in.MB, MB.sec, data.consumed.in.nMsg, nMsg.sec, rebalance.time.ms, fetch.time.ms, fetch.MB.sec, fetch.nMsg.sec

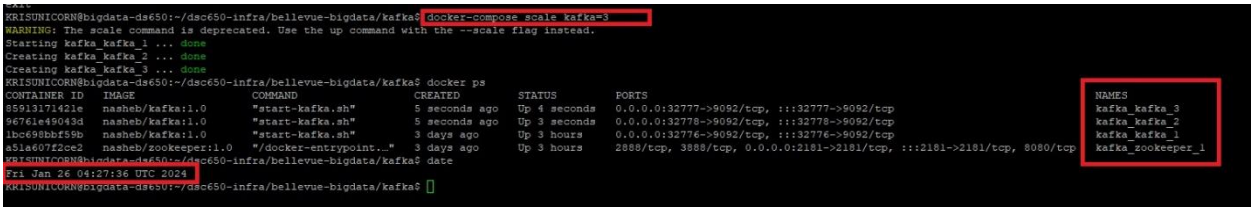
2024-01-26 02:55:24:876, 2024-01-26 02:55:25:641, 4.7684, 6.2332, 50000, 65359.4771, 432, 333, 14.3194, 150150.1502

Meaning of the Consumer Output:

Each block of the output has been color-coded for easy understanding. The start time and end time indicate when the consumer started reading and finished reading the messages. The output also indicates the total amount of data consumed in MB per second. In total ***50000 messages*** of size ***4.7MB*** were consumed. The ***throughput*** is also included in the output which is ***6.23 MB/sec or 65359 messages/ sec***. The rebalancing time is ***432 milliseconds*** which is the maximum time for the worker to join the group in the event of a rebalancing. The fetch time indicates the time to fetch the data which was ***333 milliseconds***. Other fetch metrics such as the number of messages fetched per second and in MB are also included.

4. Expanding Kafka and Running Additional Performance Tests

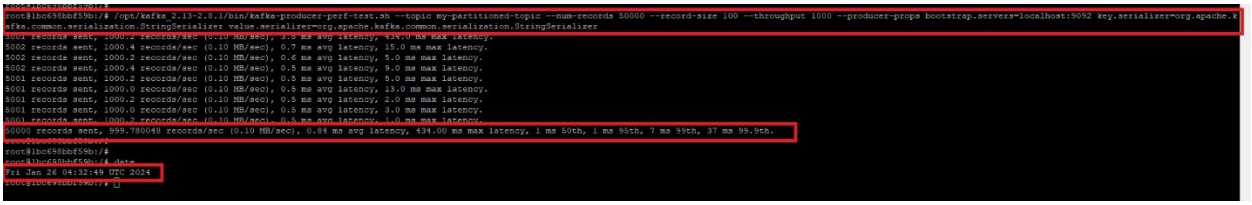
Screenshot showing the number of Kafka containers scaled up to 3



4.1. Producer Performance test with scaled instances:

```
/opt/kafka_2.13-2.8.1/bin/kafka-producer-perf-test.sh --topic my-partit
ioned-topic --num-records 50000 --record-size 100 --throughput 1000 --p
roducer-props bootstrap.servers=localhost:9092 key.serializer=org.apach
e.kafka.common.serialization.StringSerializer value.serializer=org.apac
he.kafka.common.serialization.StringSerializer
```

Screenshot of the output:



50000 records sent, 999.780048 records/sec (0.10 MB/sec), 0.84 ms avg latency, 434.00 m s max latency, 1 ms 50th, 1 ms 95th, 7 ms 99th, 37 ms 99.9th.

Comparison of producer performance between the Single Instance and scaled s etup:

Producer Parameters	Single Instance	Scaled setup
Total Records sent	50000	50000
Records/sec	999.80004	999.780048
Average Latency	1.42 ms	0.84 ms
Maximum Latency	646 ms	434 ms
50th Percentile	1 ms	1 ms
95th Percentile	1 ms	1 ms
99th Percentile	54 ms	7 ms
99.9th Percentile	76 ms	37 ms

(Table 1- Producer comparison)

With the scaled setup with 3 Kafka containers, the same number of 50000 records were produced, the topic used is my-partitioned-topic. About 1000 records/second were produced same as the individual setup. However, the biggest improvement is in the latency, where the average latency dropped to 0.84 milliseconds from 1.42ms with the individual setup. Similar improvements were also observed in the maximum latency as well. The 50th and 95th percentile records were still 1 millisecond indicating that for 95% of the records the time taken to be produced and when they were written in the broker filesystem was within 1 millisecond. However, the 99th percentile and 99.9th percentile also saw great improvements with the 3-container setup. The comparison is done in Table 1 above.

4.2. Consumer Performance test with scaled instances:

```
/opt/kafka_2.13-2.8.1/bin/kafka-consumer-perf-test.sh --broker-list loc
alhost:9092 --topic my-partitioned-topic --messages 50000
```

Screenshot of the output:

start.time, end.time, data.consumed.in.MB, MB.sec, data.consumed.in.nMsg, nMsg.sec, rebalance.time.ms, fetch.ti me.ms, fetch.MB.sec, fetch.nMsg.sec

2024-01-26 04:54:45:571, 2024-01-26 04:54:46:293, 4.7923, 6.6375, 50251, 69599.7230, 413, 309, 15.5091, 162624.5955

Comparison of consumer performance between the Single Instance and scaled setup:

Consumer Parameters	Single Instance	Scaled setup
start.time	2024-01-26 02:55:24:876	2024-01-26 04:54:45:571
end.time	2024-01-26 02:55:25:641	2024-01-26 04:54:46:293
data.consumed.in.MB	4.7684	4.7923
MB.sec	6.2332	6.6375
data.consumed.in.nMsg	50000	50251
nMsg.sec	65359.4771	69599.723
rebalance.time.ms	432	413
fetch.time.ms	333	309
fetch.MB.sec	14.3194	15.5091
fetch.nMsg.sec	150150.1502	162624.5955

(Table 2- Consumer comparison)

Table 2 compares the results of the consumer performance between individual and scaled setups. In the scaled setup, the rebalance time which is the maximum time for the worker to join the group in the event of a rebalancing had reduced from 432 milliseconds to 413 milliseconds. Also, the fetch time to fetch the records has reduced from 333 milliseconds to 309 milliseconds. Similarly, there is an improvement in the number of messages fetched per second and the number of MB fetched per second.