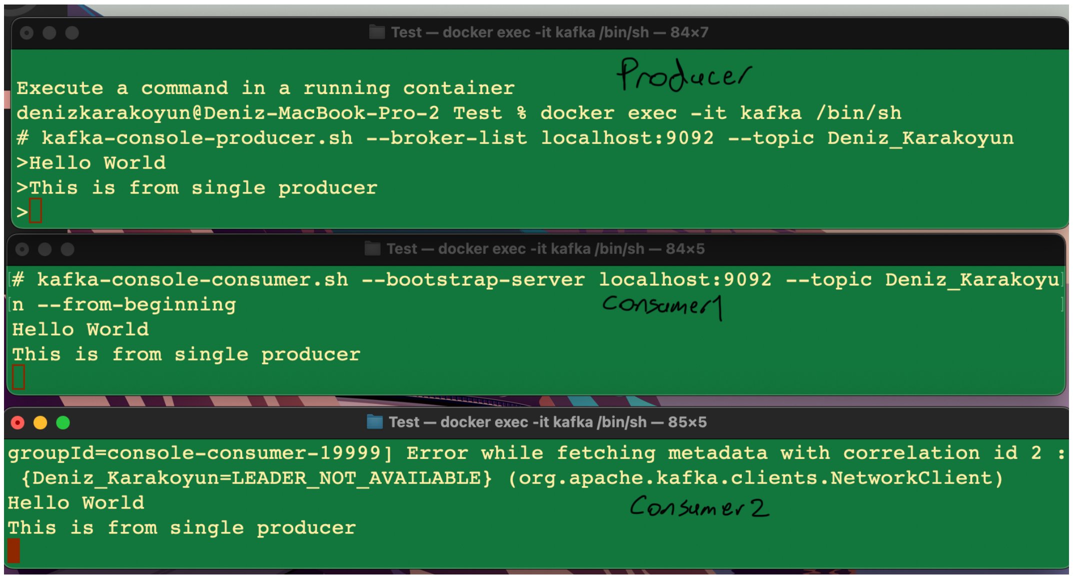
# WEEK 2

August 23, 2024

## Student Information

**Full Name:** Deniz Karakoyun
**Id Number:** 2580678

## 1) Multiple Consumer Single Producer


Test — docker exec -it kafka /bin/sh — 84×7
Producer
Test — docker exec -it kafka /bin/sh — 84×5
Consumer1
Test — docker exec -it kafka /bin/sh — 85×5
Consumer2

```
Execute a command in a running container
denizkarakoyun@Deniz-MacBook-Pro-2 Test % docker exec -it kafka /bin/sh
# kafka-console-producer.sh --broker-list localhost:9092 --topic Deniz_Karakoyun
>Hello World
>This is from single producer
>
```

```
# kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic Deniz_Karakoyu
n --from-beginning
Hello World
This is from single producer
```

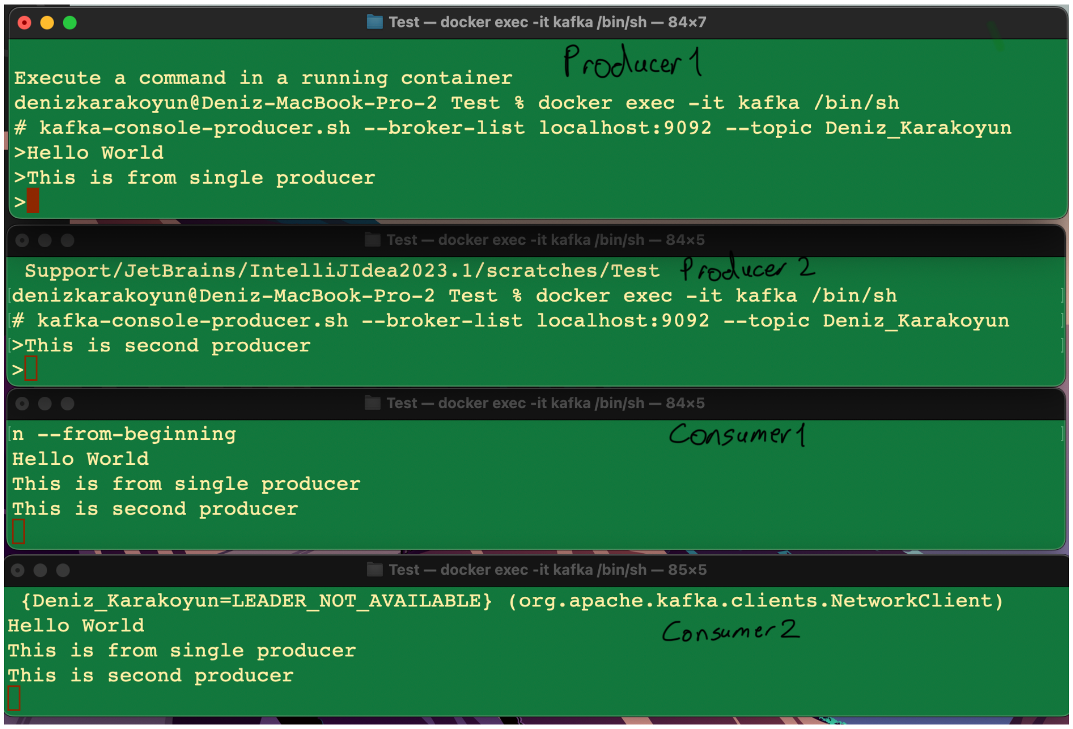
```
groupId=console-consumer-19999] Error while fetching metadata with correlation id 2 :
 {Deniz_Karakoyun=LEADER_NOT_AVAILABLE} (org.apache.kafka.clients.NetworkClient)
Hello World
This is from single producer
```

Figure 1: Multiple Consumer Single Producer Architecture

**Explanation:**
In this architecture, a single Kafka producer sends messages to a Kafka topic, which can be consumed by multiple consumers. This setup is useful for distributing the processing load among multiple consumers. Each consumer can read from the same topic independently, ensuring that

the processing of messages is parallelized. This approach is often used to scale out the processing capabilities of an application.

## 2) Multiple Consumer Multiple Producer


Test — docker exec -it kafka /bin/sh — 84×7
Producer 1
Test — docker exec -it kafka /bin/sh — 84×5
Producer 2
Test — docker exec -it kafka /bin/sh — 84×5
Consumer 1
Test — docker exec -it kafka /bin/sh — 85×5
Consumer 2

```
Execute a command in a running container
denizkarakoyun@Deniz-MacBook-Pro-2 Test % docker exec -it kafka /bin/sh
# kafka-console-producer.sh --broker-list localhost:9092 --topic Deniz_Karakoyun
>Hello World
>This is from single producer
>
```

```
 Support/JetBrains/IntelliJIdea2023.1/scratches/Test
denizkarakoyun@Deniz-MacBook-Pro-2 Test % docker exec -it kafka /bin/sh
# kafka-console-producer.sh --broker-list localhost:9092 --topic Deniz_Karakoyun
>This is second producer
>
```

```
n --from-beginning
Hello World
This is from single producer
This is second producer
```

```
 {Deniz_Karakoyun=LEADER_NOT_AVAILABLE} (org.apache.kafka.clients.NetworkClient)
Hello World
This is from single producer
This is second producer
```

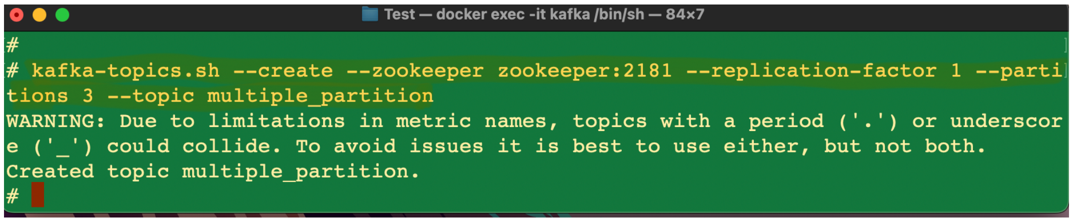
Figure 2: Multiple Consumer Multiple Producer Architecture

**Explanation:**
This more complex architecture involves multiple producers sending messages to Kafka topics, which are then consumed by multiple consumers. This setup offers high flexibility and fault tolerance. Multiple producers can independently send data to the same or different topics, and multiple consumers can read from these topics. This model is particularly useful in scenarios requiring high availability and scalability, as both producers and consumers can operate independently and handle failures gracefully.

## 3) Multiple Partitions

Kafka topics are divided into partitions to allow parallel processing and increase performance. Each partition is an ordered log of messages. Understanding how partitions work is crucial for optimizing Kafka performance and ensuring data is processed efficiently.

## Creating a Topic with Multiple Partitions

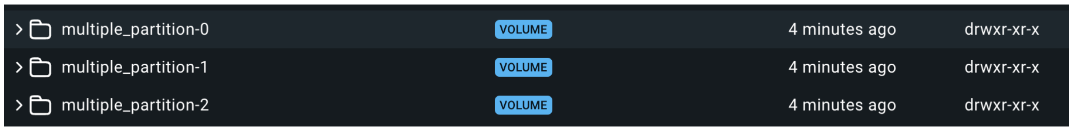


```
Test — docker exec -it kafka /bin/sh — 84×7
#
# kafka-topics.sh --create --zookeeper zookeeper:2181 --replication-factor 1 --parti
tions 3 --topic multiple_partition
WARNING: Due to limitations in metric names, topics with a period ('.') or underscor
e ('_') could collide. To avoid issues it is best to use either, but not both.
Created topic multiple_partition.
#
```

Figure 3: Creating a Topic with Multiple Partitions

**Explanation:**
Creating a topic with multiple partitions allows Kafka to distribute the load of incoming messages across multiple brokers. This setup enhances performance by enabling parallel processing. Each partition can be stored on a different broker, and producers write messages to partitions based on a configured partitioning strategy. This approach also improves fault tolerance, as the failure of one partition does not affect the others.

## Viewing Partition Logs

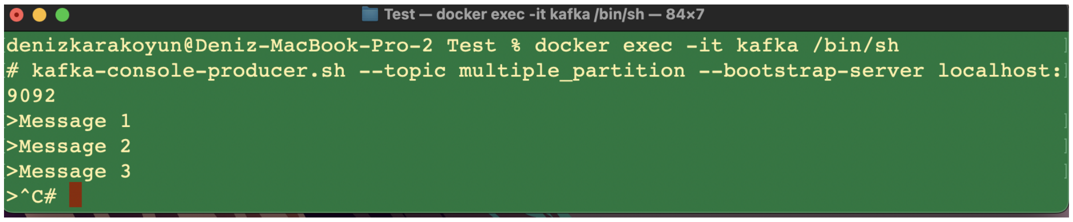


```
> multiple_partition-0    VOLUME    4 minutes ago    drwxr-xr-x
> multiple_partition-1    VOLUME    4 minutes ago    drwxr-xr-x
> multiple_partition-2    VOLUME    4 minutes ago    drwxr-xr-x
```

Figure 4: Logs of Multiple Partitions

**Explanation:**
The logs of multiple partitions display the contents and status of each partition. Kafka maintains separate logs for each partition, and these logs are managed independently. Understanding these logs helps in monitoring the data flow and diagnosing issues related to message distribution and partitioning.

## Writing Messages to a Topic with Multiple Partitions

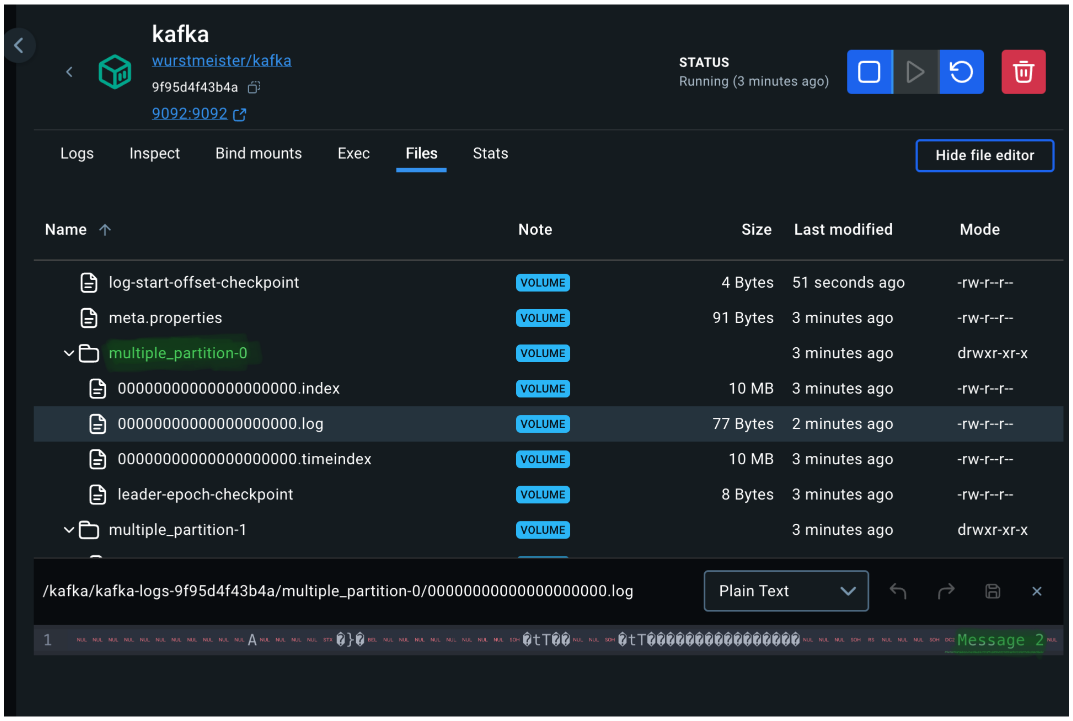


```
Test — docker exec -it kafka /bin/sh — 84×7
denizkarakoyun@Deniz-MacBook-Pro-2 Test % docker exec -it kafka /bin/sh
# kafka-console-producer.sh --topic multiple_partition --bootstrap-server localhost:
9092
>Message 1
>Message 2
>Message 3
>^C#
```

Figure 5: Writing Messages to a Topic with Multiple Partitions

**Explanation:**
When messages are written to a topic with multiple partitions, Kafka distributes these messages based on the configured partitioning strategy. Typically, messages are assigned to partitions using a key-based or round-robin approach. This distribution ensures that the workload is evenly spread and allows consumers to process messages from different partitions in parallel.

## Partition Content Analysis

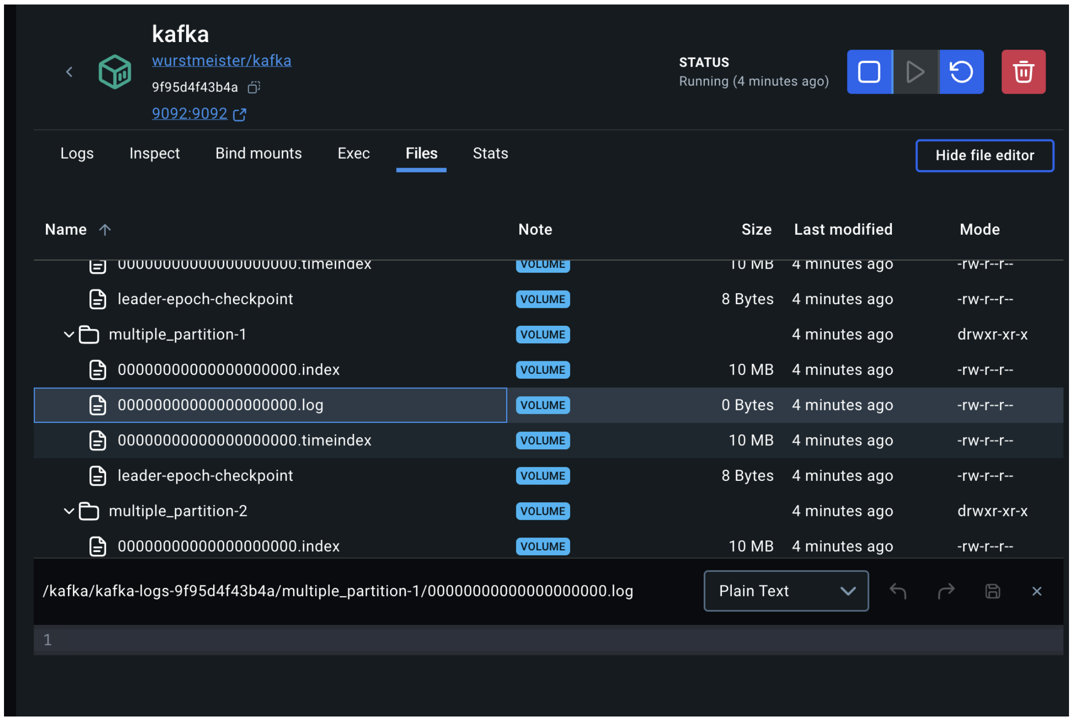


| Name | Note | Size | Last modified | Mode |
|---|---|---|---|---|
| log-start-offset-checkpoint | VOLUME | 4 Bytes | 51 seconds ago | -rw-r--r-- |
| meta.properties | VOLUME | 91 Bytes | 3 minutes ago | -rw-r--r-- |
| multiple_partition-0 | VOLUME | | 3 minutes ago | drwxr-xr-x |
| 00000000000000000000.index | VOLUME | 10 MB | 3 minutes ago | -rw-r--r-- |
| 00000000000000000000.log | VOLUME | 77 Bytes | 2 minutes ago | -rw-r--r-- |
| 00000000000000000000.timeindex | VOLUME | 10 MB | 3 minutes ago | -rw-r--r-- |
| leader-epoch-checkpoint | VOLUME | 8 Bytes | 3 minutes ago | -rw-r--r-- |
| multiple_partition-1 | VOLUME | | 3 minutes ago | drwxr-xr-x |

```
/kafka/kafka-logs-9f95d4f43b4a/multiple_partition-0/00000000000000000000.log
NUL NUL NUL NUL NUL NUL NUL NUL NUL NUL NUL A NUL NUL NUL NUL STX �}� BEL NUL NUL NUL NUL NUL NUL NUL NUL SOH �tT�� NUL NUL SOH �tT����������������� NUL NUL NUL SOH RS NUL NUL NUL SOH DC2 Message 2 NUL
```

Figure 6: Partition 0 and Its Content

**Explanation:**
Partition 0 contains the message "Message 2". Each partition holds an ordered log of messages. Understanding the content of each partition helps in analyzing the message distribution and ensures that data is processed as expected.



kafka
wurstmeister/kafka
9f95d4f43b4a
9092:9092

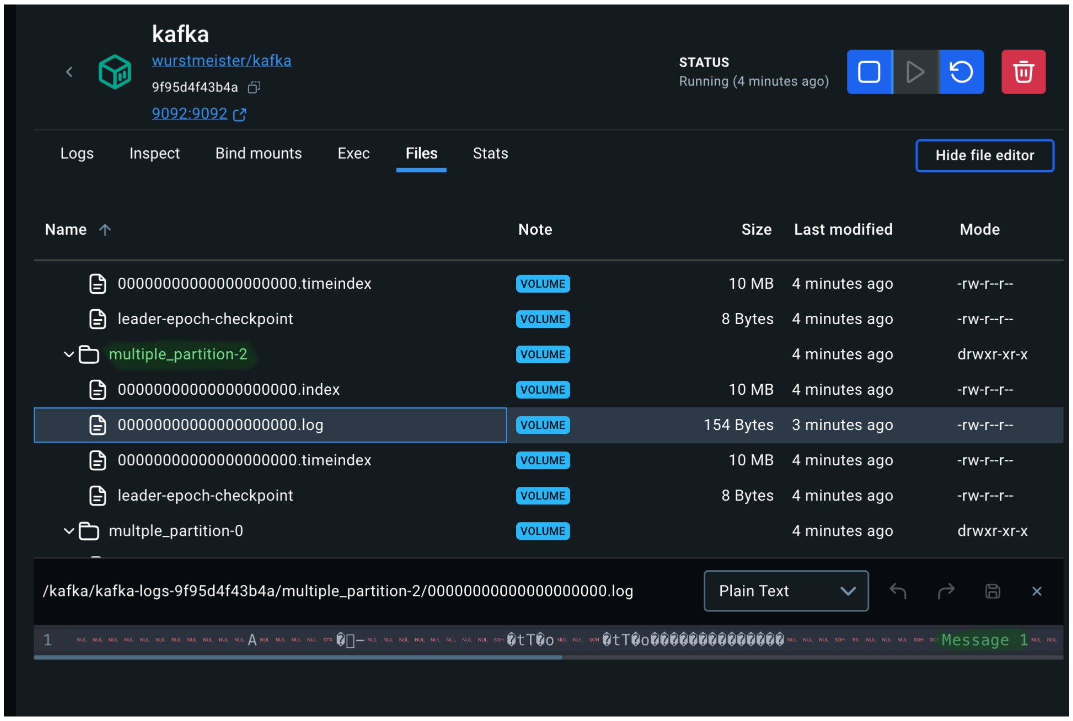
STATUS
Running (4 minutes ago)

| Name | Note | Size | Last modified | Mode |
|---|---|---|---|---|
| 00000000000000000000.timeindex | VOLUME | 10 MB | 4 minutes ago | -rw-r--r-- |
| leader-epoch-checkpoint | VOLUME | 8 Bytes | 4 minutes ago | -rw-r--r-- |
| multiple_partition-1 | VOLUME | | 4 minutes ago | drwxr-xr-x |
| 00000000000000000000.index | VOLUME | 10 MB | 4 minutes ago | -rw-r--r-- |
| 00000000000000000000.log | VOLUME | 0 Bytes | 4 minutes ago | -rw-r--r-- |
| 00000000000000000000.timeindex | VOLUME | 10 MB | 4 minutes ago | -rw-r--r-- |
| leader-epoch-checkpoint | VOLUME | 8 Bytes | 4 minutes ago | -rw-r--r-- |
| multiple_partition-2 | VOLUME | | 4 minutes ago | drwxr-xr-x |
| 00000000000000000000.index | VOLUME | 10 MB | 4 minutes ago | -rw-r--r-- |

/kafka/kafka-logs-9f95d4f43b4a/multiple_partition-1/00000000000000000000.log

Figure 7: Partition 1 and Its Content

**Explanation:**
Partition 1 currently has no messages. This could be due to a variety of reasons, such as the partition not receiving any data or messages being distributed to other partitions based on the partitioning strategy.



kafka
wurstmeister/kafka
9f95d4f43b4a
9092:9092

STATUS
Running (4 minutes ago)

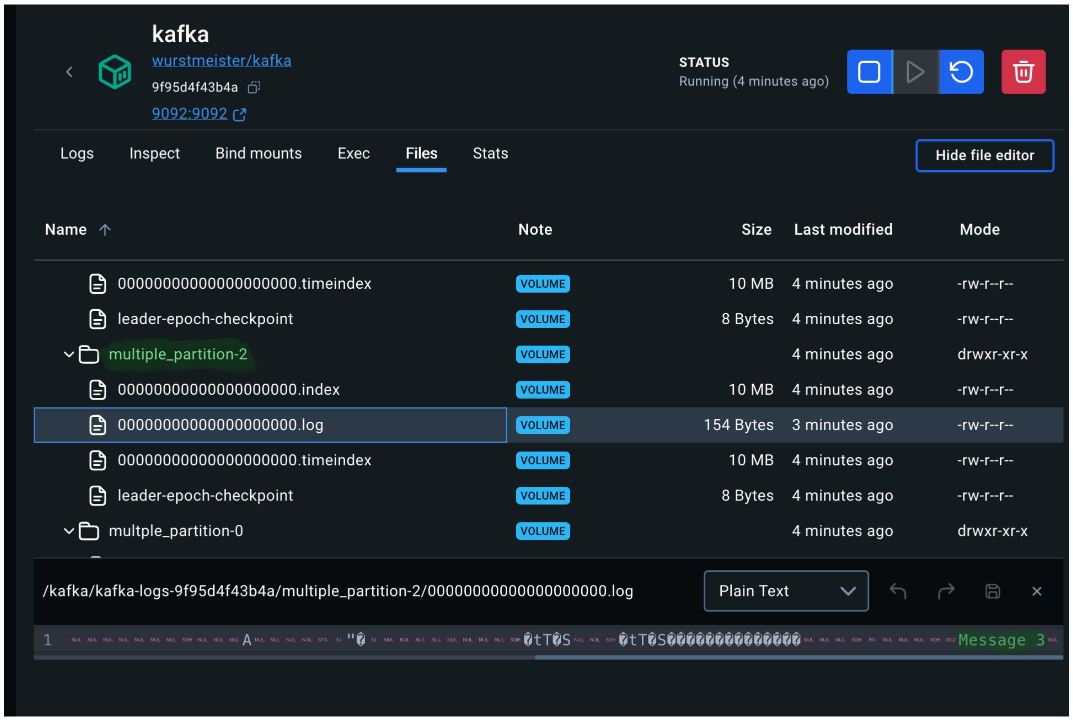
| Name | Note | Size | Last modified | Mode |
|---|---|---|---|---|
| 00000000000000000000.timeindex | VOLUME | 10 MB | 4 minutes ago | -rw-r--r-- |
| leader-epoch-checkpoint | VOLUME | 8 Bytes | 4 minutes ago | -rw-r--r-- |
| multiple_partition-2 | VOLUME |  | 4 minutes ago | drwxr-xr-x |
| 00000000000000000000.index | VOLUME | 10 MB | 4 minutes ago | -rw-r--r-- |
| 00000000000000000000.log | VOLUME | 154 Bytes | 3 minutes ago | -rw-r--r-- |
| 00000000000000000000.timeindex | VOLUME | 10 MB | 4 minutes ago | -rw-r--r-- |
| leader-epoch-checkpoint | VOLUME | 8 Bytes | 4 minutes ago | -rw-r--r-- |
| multple_partition-0 | VOLUME |  | 4 minutes ago | drwxr-xr-x |

/kafka/kafka-logs-9f95d4f43b4a/multiple_partition-2/00000000000000000000.log

```
NUL NUL NUL NUL NUL NUL NUL NUL NUL NUL NUL A NUL NUL NUL NUL STX �▯– NUL NUL NUL NUL NUL NUL NUL NUL SOH �tT�o NUL NUL SOH �tT�o��������������� NUL NUL NUL SOH RS NUL NUL NUL SOH DC2 Message 1 NUL NUL
```

Figure 8: Partition 2 and Its Content

**Explanation:**
Partition 2 contains the messages "Message 1" and "Message 3". Each partition can hold multiple messages, and the order of messages is preserved within a partition. Analyzing the contents of each partition helps in understanding the message distribution and verifying the correctness of the data flow.



| Name | Note | Size | Last modified | Mode |
|---|---|---|---|---|
| 00000000000000000000.timeindex | VOLUME | 10 MB | 4 minutes ago | -rw-r--r-- |
| leader-epoch-checkpoint | VOLUME | 8 Bytes | 4 minutes ago | -rw-r--r-- |
| multiple_partition-2 | VOLUME | | 4 minutes ago | drwxr-xr-x |
| 00000000000000000000.index | VOLUME | 10 MB | 4 minutes ago | -rw-r--r-- |
| 00000000000000000000.log | VOLUME | 154 Bytes | 3 minutes ago | -rw-r--r-- |
| 00000000000000000000.timeindex | VOLUME | 10 MB | 4 minutes ago | -rw-r--r-- |
| leader-epoch-checkpoint | VOLUME | 8 Bytes | 4 minutes ago | -rw-r--r-- |
| multple_partition-0 | VOLUME | | 4 minutes ago | drwxr-xr-x |

```
/kafka/kafka-logs-9f95d4f43b4a/multiple_partition-2/00000000000000000000.log
A "� �tT�S �tT�S���������������� Message 3
```

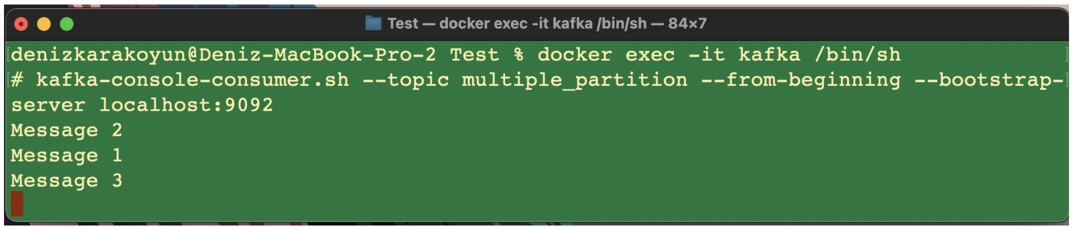
Figure 9: Detailed View of Partition 2

**Explanation:**
This detailed view of Partition 2 shows the messages "Message 1" and "Message 3" in more detail. Such detailed analysis can be useful for debugging and ensuring that messages are being distributed and stored correctly within each partition.

## 4) Multiple Partitions: Reading a Topic

In Kafka, topics are divided into multiple partitions, facilitating parallel processing and enhanced performance. Understanding how to consume messages from these partitions is crucial for effective data management and analysis.

### Consuming Messages from a Topic with Multiple Partitions



```
denizkarakoyun@Deniz-MacBook-Pro-2 Test % docker exec -it kafka /bin/sh
# kafka-console-consumer.sh --topic multiple_partition --from-beginning --bootstrap-
server localhost:9092
Message 2
Message 1
Message 3
```

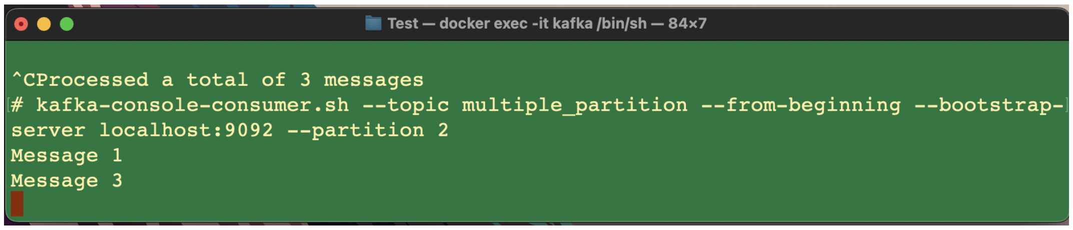
Figure 10: Consuming Messages From a Topic with Multiple Partitions

**Explanation:**
When consuming messages from a topic with multiple partitions, Kafka allows parallel reading of messages from all partitions. This parallelism improves the efficiency and scalability of data processing.

- **Consumers**: A consumer (or consumer group) subscribes to a topic and reads messages from all partitions associated with that topic.
- **Partition Assignment**: Each consumer in the group is assigned one or more partitions. Kafka ensures that each partition is only read by one consumer within a group at any given time.
- **Message Distribution**: Messages are distributed among partitions based on the producer's partitioning strategy. Consumers then read messages from their assigned partitions.

The figure above demonstrates how multiple consumers can read messages from different partitions of the same topic, enhancing parallel processing and system scalability.

## Consuming Messages from a Specified Partition



```
^CProcessed a total of 3 messages
# kafka-console-consumer.sh --topic multiple_partition --from-beginning --bootstrap-
server localhost:9092 --partition 2
Message 1
Message 3
```

Figure 11: Consuming Messages From a Specified Partition

**Explanation:**
Sometimes, it is necessary to consume messages from a specific partition rather than from all partitions of a topic. This targeted approach is useful for tasks such as debugging or analyzing specific data subsets.

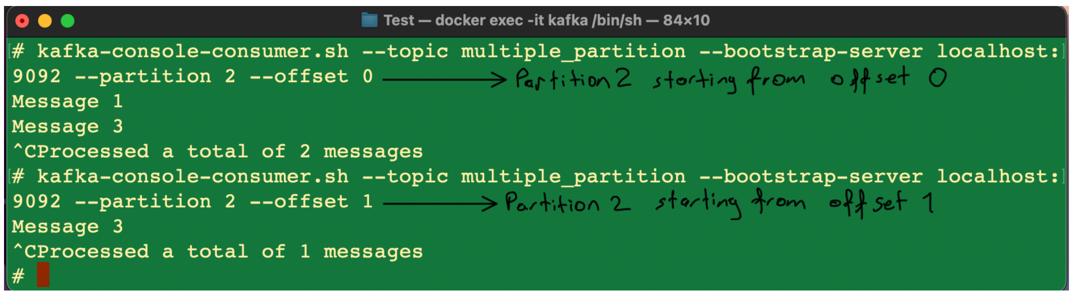
- **Partition Specification**: Configure your consumer to read from a specific partition by specifying its number in the consumer configuration.
- **Consumer Configuration**: Programmatically create a `TopicPartition` object and assign it to the consumer to interact with the desired partition.
- **Use Case**: This method is beneficial for isolating and analyzing messages from a particular partition or when troubleshooting distribution issues.

The figure illustrates how a consumer can be configured to read from a specified partition, focusing on targeted message retrieval.

## Consuming Messages from a Specified Partition and Offset

**Explanation:**
In addition to specifying a partition, you may want to start consuming messages from a specific offset within that partition. This functionality is important for processing messages from a specific point or event.

- **Offset Specification**: Set the offset from which to start consuming messages. This is useful for resuming from a known point or skipping certain messages.
- **Consumer Configuration**: Use the `assign` method to set both the partition and offset directly, or use the `seek` method to move the consumer's position to the desired offset.
- **Use Case**: Helpful for message replay, failure recovery, or analyzing specific data ranges.



```
# kafka-console-consumer.sh --topic multiple_partition --bootstrap-server localhost:
9092 --partition 2 --offset 0 ———> Partition 2 starting from offset 0
Message 1
Message 3
^CProcessed a total of 2 messages
# kafka-console-consumer.sh --topic multiple_partition --bootstrap-server localhost:
9092 --partition 2 --offset 1 ———> Partition 2 starting from offset 1
Message 3
^CProcessed a total of 1 messages
#
```

Figure 12: Consuming Messages From a Specified Partition and Specified Offset

The consumer will begin reading from the specified offset within the partition, providing precise control over message consumption.

**Summary:**

- **General Process**: Consuming messages from all partitions of a topic enhances parallel processing.
- **Specified Partition**: Allows targeted message retrieval from a single partition.
- **Specified Partition and Offset**: Provides fine-grained control over message consumption starting from a specific point.

These are tthe parts of process of consuming messages from Kafka topics with multiple partitions, as well as handling specific partitions and offsets. Adjust the figures and paths as necessary to align with your specific setup and use case.

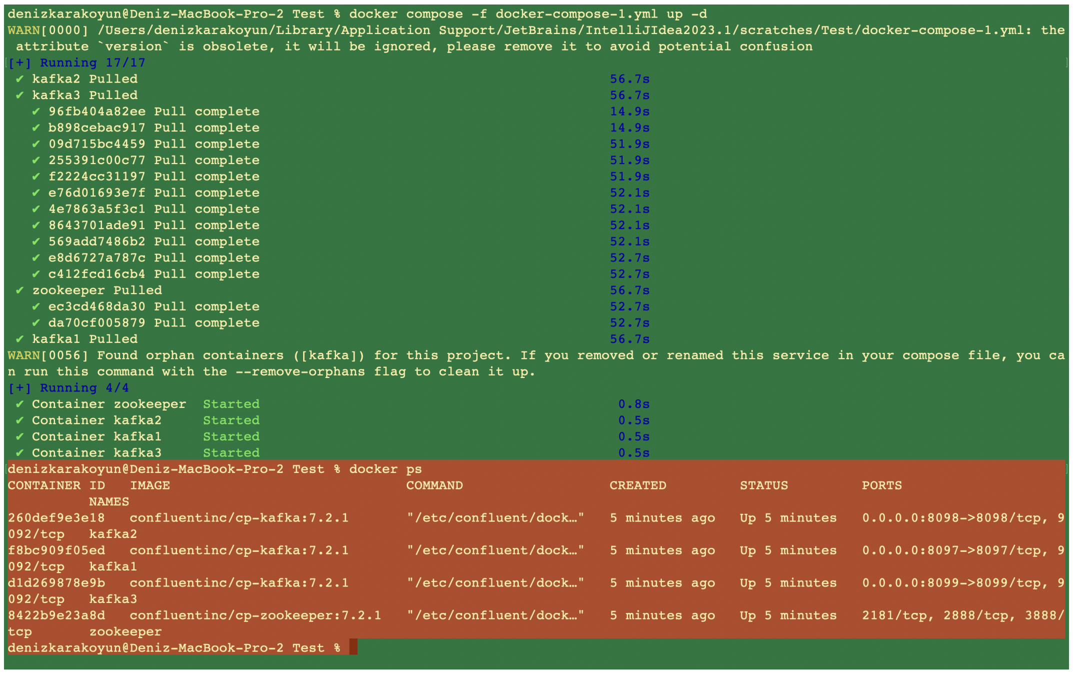
## 5) Multiple Brokers

In Apache Kafka, a **broker** is a server responsible for receiving, storing, and serving messages. Kafka's architecture is designed to be **distributed**, meaning it can run on multiple brokers, providing scalability, fault tolerance, and increased performance.

## Key Concepts

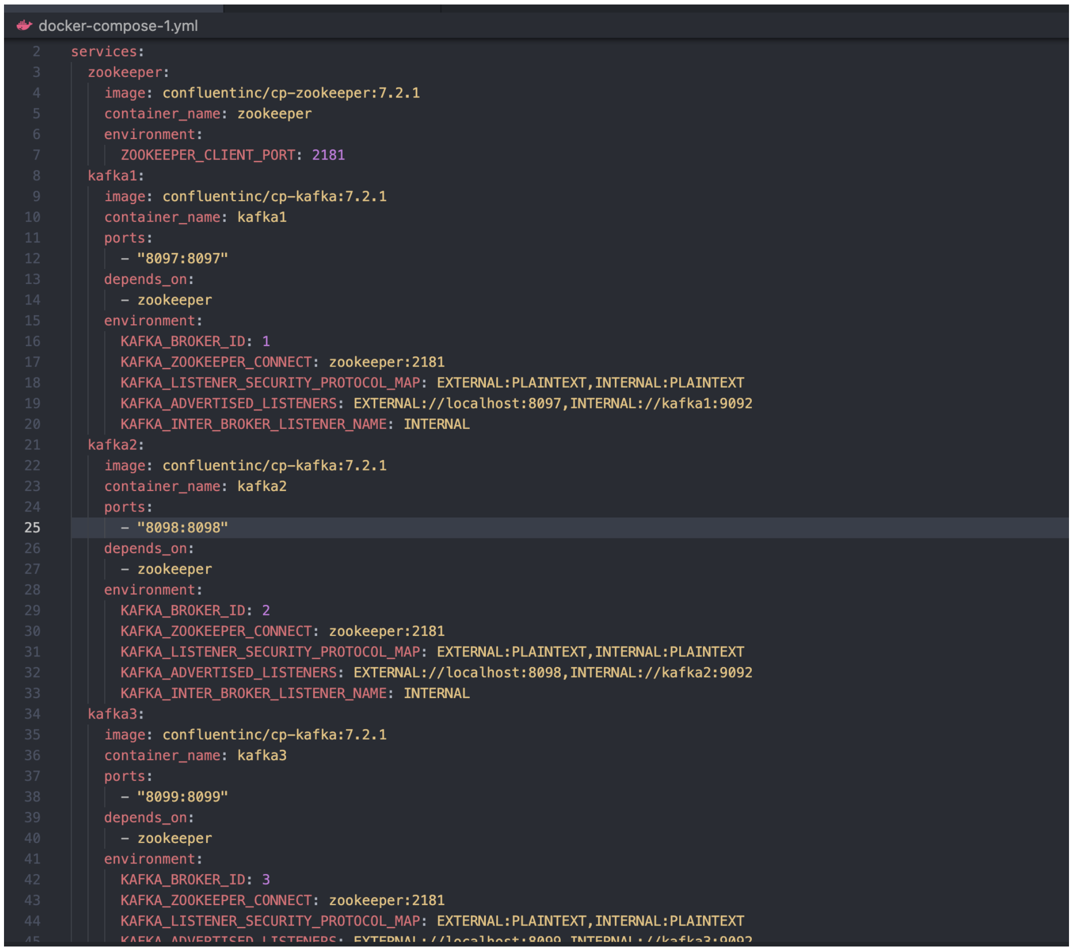
- **Broker**: A single Kafka server that handles producer requests, stores data, and serves consumers.
- **Cluster**: A group of Kafka brokers working together. In a Kafka cluster, data is distributed across multiple brokers.
- **Partitioning**: Each topic in Kafka is divided into multiple partitions, which are distributed across different brokers. This enables parallel processing and better load distribution.
- **Replication**: Kafka replicates each partition across multiple brokers, ensuring data durability and availability even in the event of broker failures.
- **Leader and Follower**: For each partition, one broker acts as the leader, and the others are followers. The leader handles all read and write requests for the partition, while followers replicate the data. If a leader broker fails, a follower is automatically elected as the new leader.
- **High Availability**: With multiple brokers, Kafka can continue operating even if one or more brokers fail, as data is replicated across the cluster.

By distributing data across multiple brokers, Kafka can achieve high throughput, fault tolerance, and scalability, making it suitable for large-scale, real-time data streaming applications.



```
denizkarakoyun@Deniz-MacBook-Pro-2 Test % docker compose -f docker-compose-1.yml up -d
WARN[0000] /Users/denizkarakoyun/Library/Application Support/JetBrains/IntelliJIdea2023.1/scratches/Test/docker-compose-1.yml: the
 attribute `version` is obsolete, it will be ignored, please remove it to avoid potential confusion
[+] Running 17/17
 ✔ kafka2 Pulled                                                                   56.7s
 ✔ kafka3 Pulled                                                                   56.7s
   ✔ 96fb404a82ee Pull complete                                                    14.9s
   ✔ b898cebac917 Pull complete                                                    14.9s
   ✔ 09d715bc4459 Pull complete                                                    51.9s
   ✔ 255391c00c77 Pull complete                                                    51.9s
   ✔ f2224cc31197 Pull complete                                                    51.9s
   ✔ e76d01693e7f Pull complete                                                    52.1s
   ✔ 4e7863a5f3c1 Pull complete                                                    52.1s
   ✔ 8643701ade91 Pull complete                                                    52.1s
   ✔ 569add7486b2 Pull complete                                                    52.1s
   ✔ e8d6727a787c Pull complete                                                    52.7s
   ✔ c412fcd16cb4 Pull complete                                                    52.7s
 ✔ zookeeper Pulled                                                                56.7s
   ✔ ec3cd468da30 Pull complete                                                    52.7s
   ✔ da70cf005879 Pull complete                                                    52.7s
 ✔ kafka1 Pulled                                                                   56.7s
WARN[0056] Found orphan containers ([kafka]) for this project. If you removed or renamed this service in your compose file, you ca
n run this command with the --remove-orphans flag to clean it up.
[+] Running 4/4
 ✔ Container zookeeper  Started                                                     0.8s
 ✔ Container kafka2     Started                                                     0.5s
 ✔ Container kafka1     Started                                                     0.5s
 ✔ Container kafka3     Started                                                     0.5s
denizkarakoyun@Deniz-MacBook-Pro-2 Test % docker ps
CONTAINER ID   IMAGE                             COMMAND                  CREATED         STATUS         PORTS
          NAMES
260def9e3e18   confluentinc/cp-kafka:7.2.1       "/etc/confluent/dock…"   5 minutes ago   Up 5 minutes   0.0.0.0:8098->8098/tcp, 9
092/tcp   kafka2
f8bc909f05ed   confluentinc/cp-kafka:7.2.1       "/etc/confluent/dock…"   5 minutes ago   Up 5 minutes   0.0.0.0:8097->8097/tcp, 9
092/tcp   kafka1
d1d269878e9b   confluentinc/cp-kafka:7.2.1       "/etc/confluent/dock…"   5 minutes ago   Up 5 minutes   0.0.0.0:8099->8099/tcp, 9
092/tcp   kafka3
8422b9e23a8d   confluentinc/cp-zookeeper:7.2.1   "/etc/confluent/dock…"   5 minutes ago   Up 5 minutes   2181/tcp, 2888/tcp, 3888/
tcp       zookeeper
denizkarakoyun@Deniz-MacBook-Pro-2 Test %
```

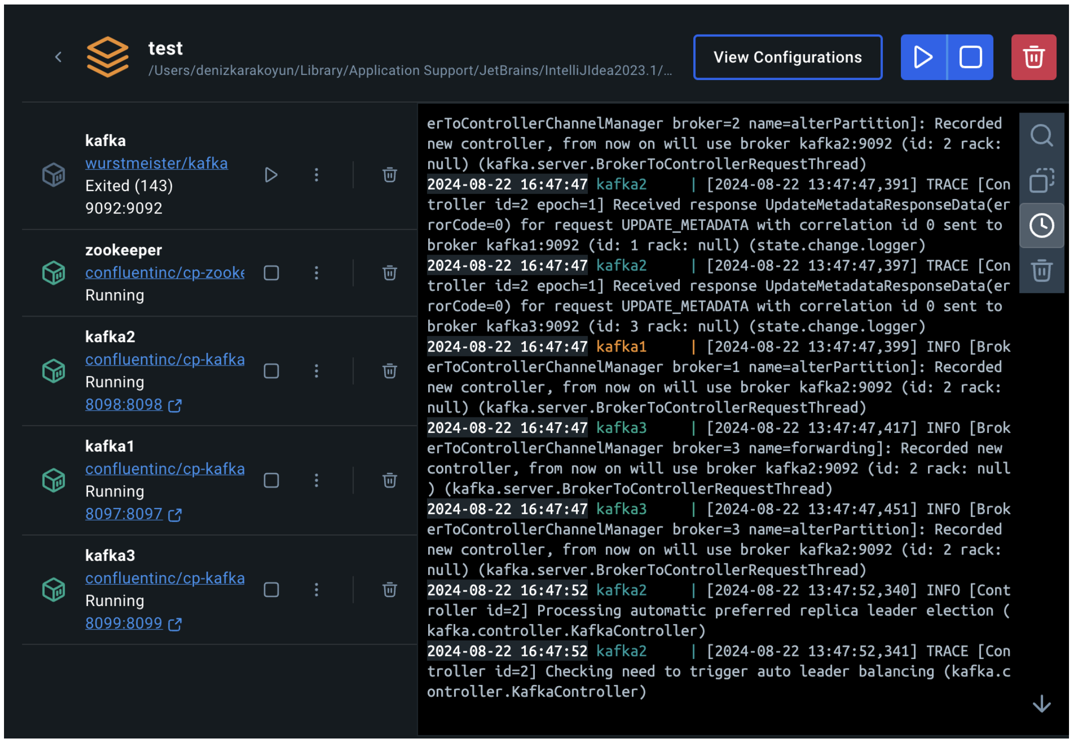
Figure 13: This figure illustrates the configuration of multiple brokers in Kafka by setting server properties. Each broker has a unique configuration, allowing them to work together in a cluster.



docker-compose-1.yml

```yaml
services:
  zookeeper:
    image: confluentinc/cp-zookeeper:7.2.1
    container_name: zookeeper
    environment:
      ZOOKEEPER_CLIENT_PORT: 2181
  kafka1:
    image: confluentinc/cp-kafka:7.2.1
    container_name: kafka1
    ports:
      - "8097:8097"
    depends_on:
      - zookeeper
    environment:
      KAFKA_BROKER_ID: 1
      KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
      KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: EXTERNAL:PLAINTEXT,INTERNAL:PLAINTEXT
      KAFKA_ADVERTISED_LISTENERS: EXTERNAL://localhost:8097,INTERNAL://kafka1:9092
      KAFKA_INTER_BROKER_LISTENER_NAME: INTERNAL
  kafka2:
    image: confluentinc/cp-kafka:7.2.1
    container_name: kafka2
    ports:
      - "8098:8098"
    depends_on:
      - zookeeper
    environment:
      KAFKA_BROKER_ID: 2
      KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
      KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: EXTERNAL:PLAINTEXT,INTERNAL:PLAINTEXT
      KAFKA_ADVERTISED_LISTENERS: EXTERNAL://localhost:8098,INTERNAL://kafka2:9092
      KAFKA_INTER_BROKER_LISTENER_NAME: INTERNAL
  kafka3:
    image: confluentinc/cp-kafka:7.2.1
    container_name: kafka3
    ports:
      - "8099:8099"
    depends_on:
      - zookeeper
    environment:
      KAFKA_BROKER_ID: 3
      KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
      KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: EXTERNAL:PLAINTEXT,INTERNAL:PLAINTEXT
      KAFKA_ADVERTISED_LISTENERS: EXTERNAL://localhost:8099,INTERNAL://kafka3:9092
```

Figure 14: This figure shows the regulation and tuning of server settings to optimize performance across multiple brokers. Proper regulation ensures efficient resource utilization and fault tolerance.



test
/Users/denizkarakoyun/Library/Application Support/JetBrains/IntelliJIdea2023.1/...

kafka
wurstmeister/kafka
Exited (143)
9092:9092

zookeeper
confluentinc/cp-zooke
Running

kafka2
confluentinc/cp-kafka
Running
8098:8098

kafka1
confluentinc/cp-kafka
Running
8097:8097

kafka3
confluentinc/cp-kafka
Running
8099:8099

```
erToControllerChannelManager broker=2 name=alterPartition]: Recorded
new controller, from now on will use broker kafka2:9092 (id: 2 rack:
null) (kafka.server.BrokerToControllerRequestThread)
2024-08-22 16:47:47 kafka2      | [2024-08-22 13:47:47,391] TRACE [Con
troller id=2 epoch=1] Received response UpdateMetadataResponseData(er
rorCode=0) for request UPDATE_METADATA with correlation id 0 sent to
broker kafka1:9092 (id: 1 rack: null) (state.change.logger)
2024-08-22 16:47:47 kafka2      | [2024-08-22 13:47:47,397] TRACE [Con
troller id=2 epoch=1] Received response UpdateMetadataResponseData(er
rorCode=0) for request UPDATE_METADATA with correlation id 0 sent to
broker kafka3:9092 (id: 3 rack: null) (state.change.logger)
2024-08-22 16:47:47 kafka1      | [2024-08-22 13:47:47,399] INFO [Brok
erToControllerChannelManager broker=1 name=alterPartition]: Recorded
new controller, from now on will use broker kafka2:9092 (id: 2 rack:
null) (kafka.server.BrokerToControllerRequestThread)
2024-08-22 16:47:47 kafka3      | [2024-08-22 13:47:47,417] INFO [Brok
erToControllerChannelManager broker=3 name=forwarding]: Recorded new
controller, from now on will use broker kafka2:9092 (id: 2 rack: null
) (kafka.server.BrokerToControllerRequestThread)
2024-08-22 16:47:47 kafka3      | [2024-08-22 13:47:47,451] INFO [Brok
erToControllerChannelManager broker=3 name=alterPartition]: Recorded
new controller, from now on will use broker kafka2:9092 (id: 2 rack:
null) (kafka.server.BrokerToControllerRequestThread)
2024-08-22 16:47:52 kafka2      | [2024-08-22 13:47:52,340] INFO [Cont
roller id=2] Processing automatic preferred replica leader election (
kafka.controller.KafkaController)
2024-08-22 16:47:52 kafka2      | [2024-08-22 13:47:52,341] TRACE [Con
troller id=2] Checking need to trigger auto leader balancing (kafka.c
ontroller.KafkaController)
```

Figure 15: This figure demonstrates the process of starting multiple brokers in a Kafka cluster. Each broker is initialized, and they communicate to form a cohesive cluster, enabling distributed data processing.