Kafka-Lab3 - Understanding Multiple Kafka Brokers

Understanding Replication and Multiple Brokers in Kafka

Replication in Kafka:

Replication in Kafka is the process of duplicating topic partitions across multiple brokers to ensure fault tolerance and high availability. Each partition has a leader replica and multiple follower replicas distributed across different brokers. The leader handles all read and write requests, while followers synchronize the data. If the leader fails, one of the followers is promoted to leader, ensuring data availability without disruption.

Replication enhances Kafka's durability and prevents data loss in case of broker failures. With replication, even if a broker crashes, data remains accessible from other brokers. A typical replication factor of 3 ensures that there are always multiple copies of data available, making Kafka resilient for critical real-time applications like financial transactions, event streaming, and large-scale data processing.

Multiple Brokers in Kafka:

A Kafka cluster consists of multiple brokers working together to handle large volumes of data efficiently. Each broker stores partitions of different topics, distributing the workload evenly to ensure scalability and high throughput. When a producer sends data, it is directed to the correct partition and stored in one of the brokers, while consumers retrieve data from these brokers based on their configurations.

Having multiple brokers in a Kafka cluster allows horizontal scaling, where more brokers can be added as data traffic increases. This architecture ensures that Kafka can handle real-time streaming for high-demand applications like social media feeds, log processing, and IoT sensor data analysis. By spreading partitions across multiple brokers, Kafka prevents bottlenecks and enhances performance.

Real-World Applications of Replication and Multiple Brokers

Kafka's replication and multi-broker architecture make it suitable for various real-world scenarios:

- Financial Systems: Ensures transactions are not lost even in case of failures.
- E-Commerce and Retail: Enables real-time inventory updates and recommendation engines.
- IoT and Sensor Data Processing: Handles continuous streams of data from thousands of devices.
- Log Aggregation: Collects logs from distributed systems and processes them efficiently.
- Social Media Platforms: Streams live updates, notifications, and user activities.

Setting Up Multiple Kafka Brokers

Step – 1: Configure Brokers

To create multiple brokers, go to the config folder of your kafka directory. There you will see 'server.properties' file. Copy it and paste it twice with the rename. Depend on the number of brokers we want we need to have that many server.properties file. You can name them 'server-1.properties', 'server-2.properties', and so on.

Once copied, open all those files in VScode, and look for the following:

```
broker.id=1                    // Here you will be declaring your broker ID for that individual broker
```

listeners=PLAINTEXT://:9092 // Here you are providing the port number to that individual

log.dirs=/tmp/kafka-logs-1 // Create a folder outside the kafka folder named ‘kafka_logs’ and store logs for broker one in ‘server_logs_1’

zookeeper.connect=localhost:2181 //This is a zookeeper’s port, which manages all the brokers

When you create multiple brokers, you need to change broker.id, listeners, and log.dirs.

Step – 2: Start Brokers:
Navigate to the kafka directory and enter the commands given below in sequential order:

1. bin/zookeeper-server-start.sh config/zookeeper.properties
2. bin/kafka-server-start.sh config/server-1.properties
3. bin/kafka-server-start.sh config/server-2.properties
4. bin/kafka-server-start.sh config/server-3.properties

We need to start the zookeeper first because it connects all the broker to each other.

Step 3: Configure Zookeeper.properties
Open zookeeper.properties – It is in the same config folder where we had server.properties
There will be something like this ‘maxClientCnxns=1’
By default, zookeeper can manage 2 brokers, considered heavy load on zookeeper based on the default settings.
To change it in its properties, make ‘maxClientCnxns=10’ Because we will be using multiple brokers.

Creating and Viewing Kafka Topics

Step 1: Create a Topic with Replication
bin/kafka-topics.sh --create --topic zk-three-broker-topic --bootstrap-server localhost:9092 --partitions 3 --replication-factor 2

Step 2: View Topic Description
bin/kafka-topics.sh --describe --topic zk-three-broker-topic --bootstrap-server localhost:9092

Running the Code

Step 1: Start the Producer
python producer_mb.py
Select a broker and partition.
Enter messages.

Step 2: Start the Consumer
python consumer_mb.py
Select a broker and partition.
See messages along with broker and partition details.