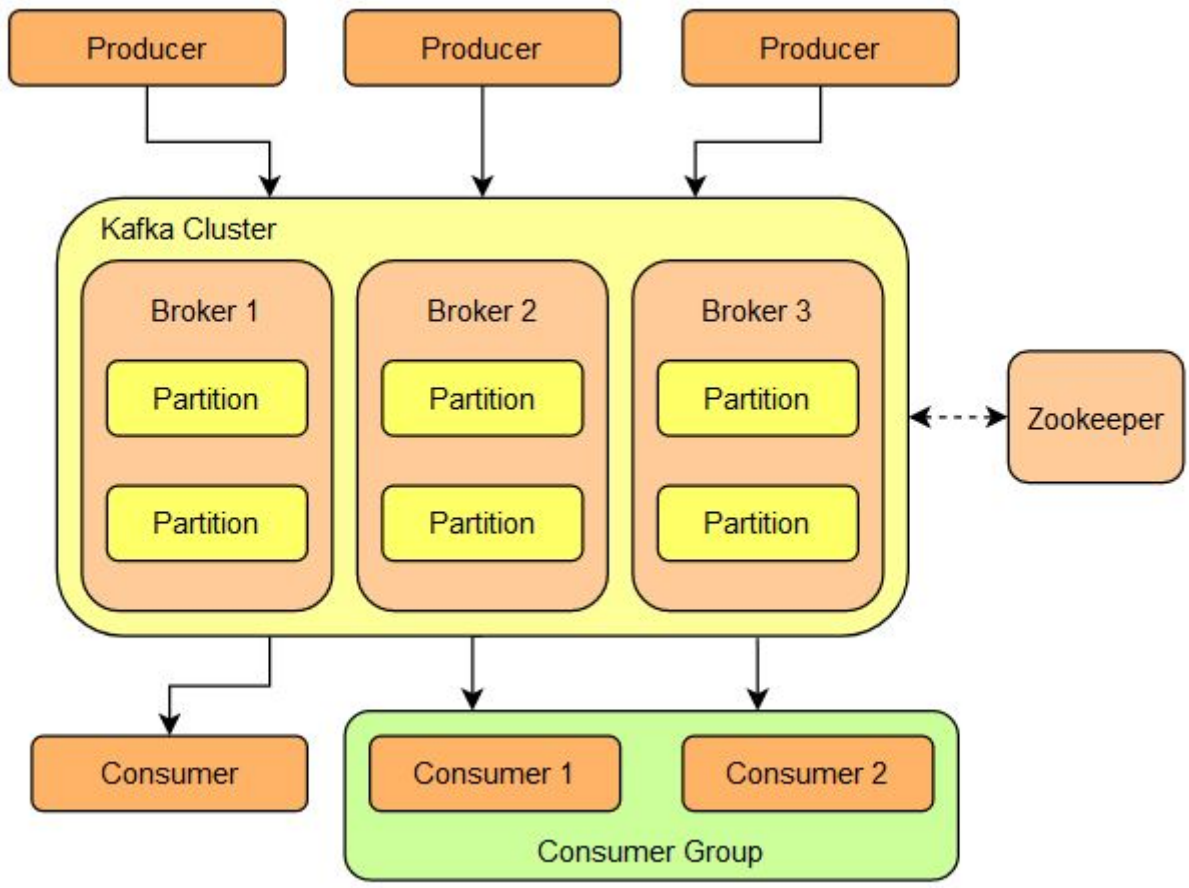
Apache Kafka: The Essential Guide

Apache Kafka is the most popular distributed streaming platform in the bigdata landscape. Initially built at LinkedIn for website telemetry, it is now used by the big names in the industry.

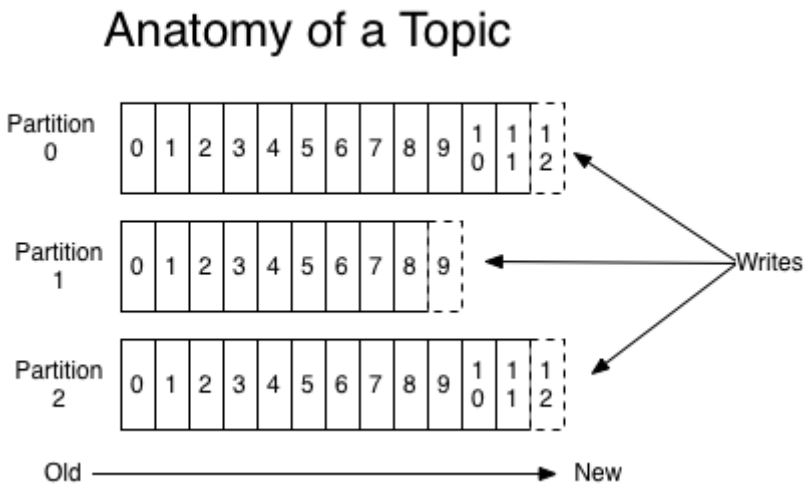
Kafka, based on publish-subscribe model, is fast and offers very high throughput, fault tolerance and horizontal scalability. It follows a distributed architecture and thus, runs as a cluster.

Kafka is written in scala and hence offers seamless integration with multiple frameworks like Apache Spark, Flink, Beam etc.



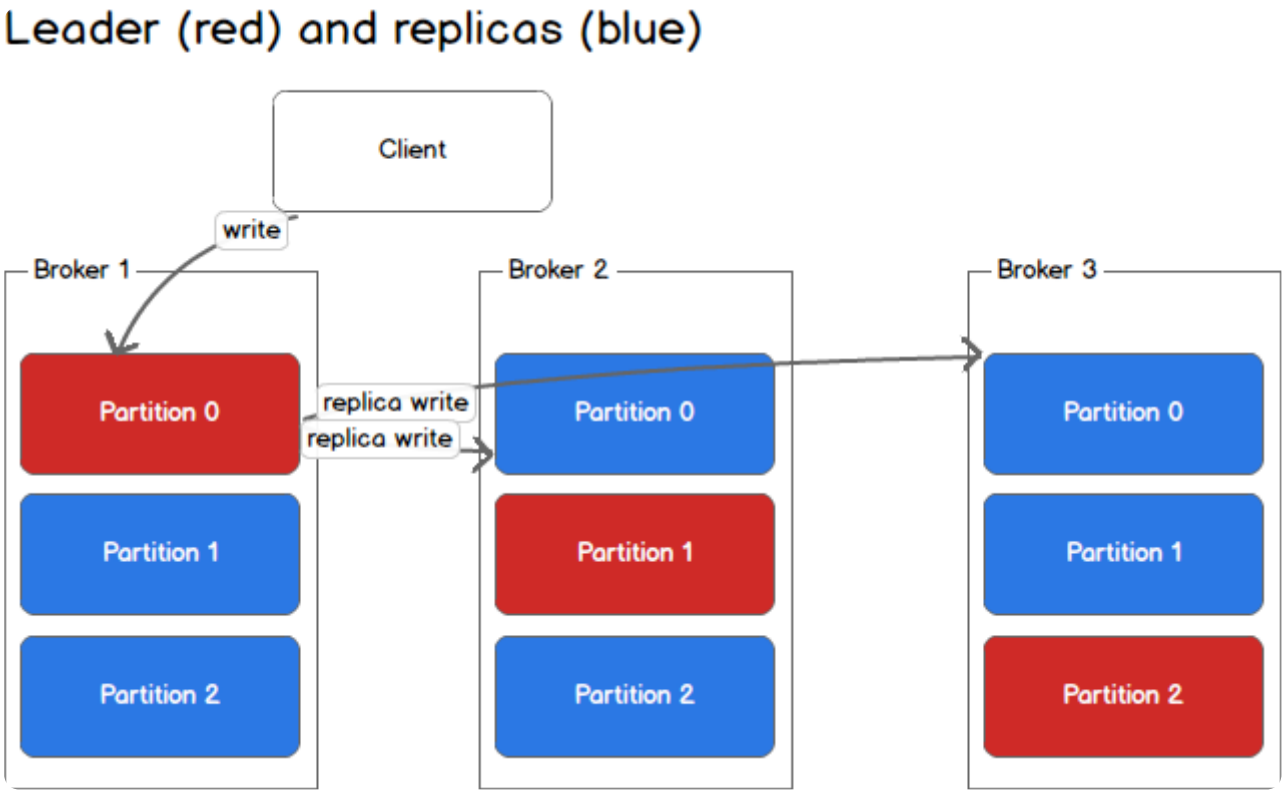
Concepts

- **Topics**
  - A topic contains a stream of messages belonging to a category. It is the feed to which messages are published.
  - Topics are composed of one or more partitions.
- **Partitions**
  - A partition is an immutable ordered sequence of messages.
  - Partitions are stored as segments in the disk.
  - Partitions are always distributed equally among available Brokers.
  - Partitions cannot be created dynamically, the number has to be set during the creation of topic.
  - Partitions can be added to a topic later, but the topic would have to go through a reassignment process.
  - The more the number of partitions, the higher the parallelism, thus higher the throughput.
- **Offsets**
  - Every message in the partition has a unique identifier attached to it called the Offset
  - Offsets are always sequential within a partition, not across partitions.

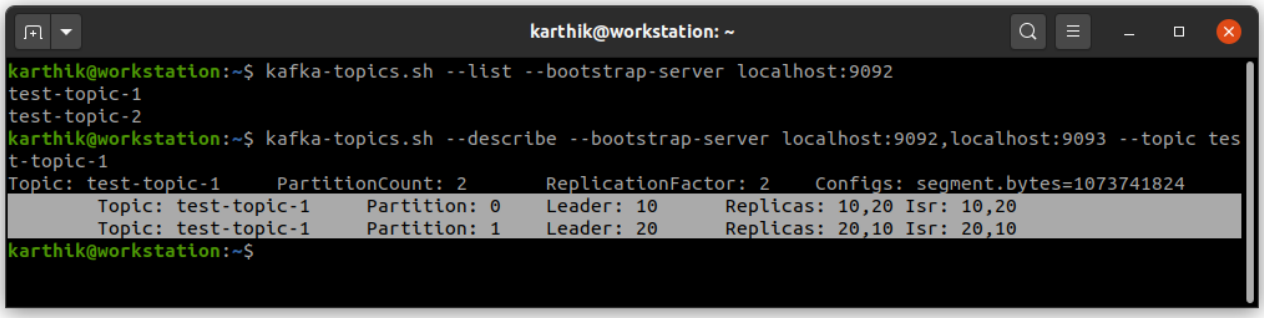


- **Replicas**
  - Replicas are copies of partitions.
  - Replicas are maintained for high availability and fault tolerance
  - No two replicas of same partition will ever reside in the same broker

- **Leader**
  - Out of the replicated partitions, exactly one of the replica is elected as a Leader
  - Kafka clients can only read from and write to leader replica
  - Other replicas fetch messages from the leader



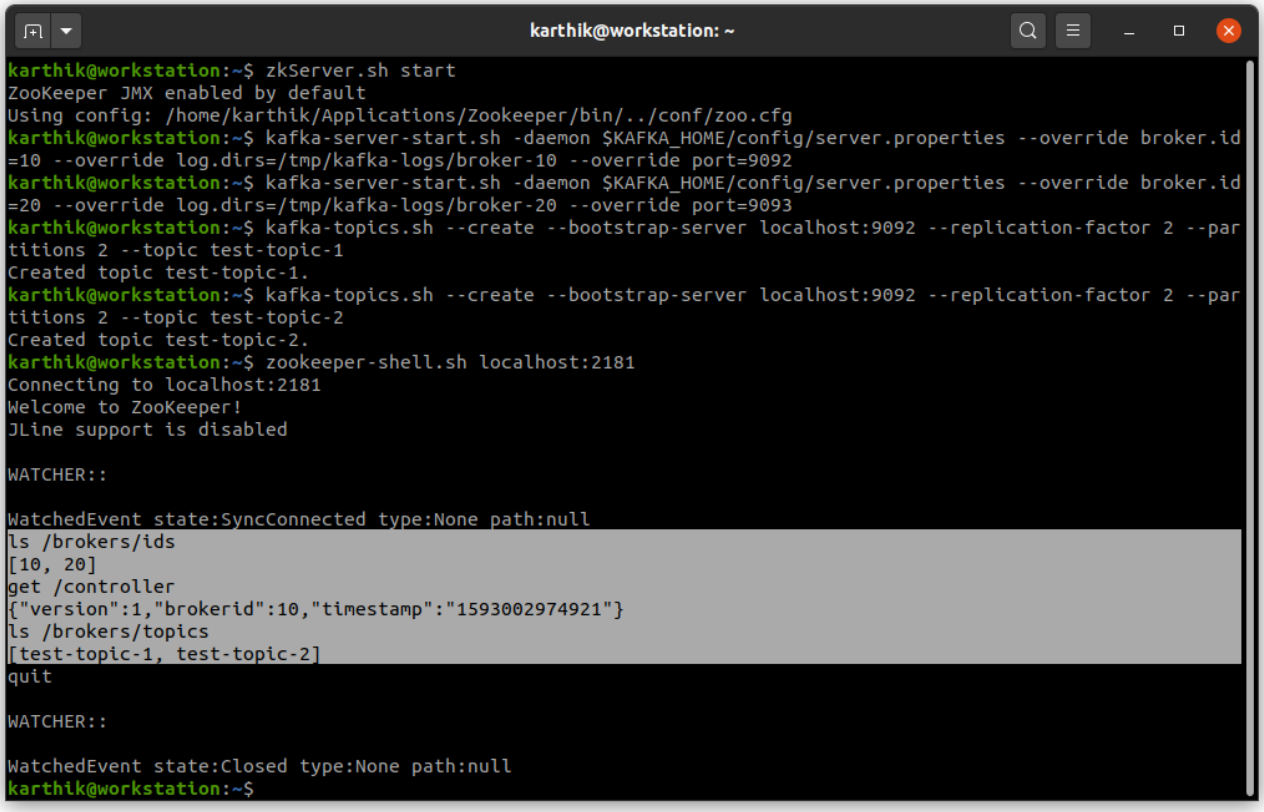
- **In-Sync Replica [ISR]**
  - Replicas which have same latest offset as the leader
  - When leader goes down, one of the ISR is chosen as the leader



- **Segments**
  - Segments are basically set of files on disk

    - When data is written to a partition, it actually means that the data was written to a segment.
    - Partitions are logical abstractions of Segments.
    - A segment contains an index file and log file.
    - Log file stores all the messages being written to the partition
    - Index file stores the offset and the location of the message in the log file
    - Each segment is identified (filename) by the first offset that it holds
    - Segments are size bound, new segments are created after configured size threshold is reached (default is 1 GB)
    - **Active Segment**
        - The segment to which the data is currently being written to
    - Here is a link with more details on kafka storage internals.
- **Brokers**
    - Brokers are nodes that hold the partitions
    - Brokers receive messages from producers and write them to segment file of the said partition
    - Brokers serve messages and keep track of the offsets read by consumers
    - Multiple brokers constitute a kafka cluster
    - Brokers within a cluster communicate with each other
    - Every broker knows the other brokers and the partitions they hold
    - Brokers can be restored with state data from ZooKeeper
    - **Controller**
        - A broker which is also a Leader of kafka cluster (not to be confused with partition leaders)
        - Each kafka cluster can only have one controller
        - Controller elects leaders for partitions
        - When a broker goes down, the controller automatically elects a new leader for all partitions that had that particular broker as their leader
        - Leaders are elected randomly out of in-sync replicas (ISR) of a partition
    - **Controller Selection**

- When the cluster is first set up, all the brokers request ephemeral node to ZooKeeper
- ZooKeeper creates one ephemeral node and the first broker to request gets selected as Controller and rest of brokers become **Watchers**
- Controller will know when new broker is added, existing broker is removed / goes down, by watching the list of nodes in ZooKeeper path
- If the controller goes down (ephemeral node goes down as well, as it is bound to the session of the controller), ZooKeeper notifies the Watchers about this and the Watchers again request an ephemeral node. Thus the above cycle repeats



Apache Kafka Clients

- Kafka clients use websockets to create network connection to the server
- Clients initially connect to one of the brokers in the cluster, that broker gives the details about all the brokers in the cluster

- Clients then make individual network connection to every broker where the partitions of the topic resides
- **Producers**
    - Client application that send messages to a kafka topic.
    - Broker sends the acknowledgement to the producer only after message is successfully written to the disk
    - Producers can choose to send messages to the partition of their choice.
    - Producer calls as asynchronous
- **Consumers**
    - Client application that read messages from a kafka topic.
    - Consumers can subscribe to one or more topics.
    - Consumers consume message through brokers
    - Consumers send acknowledgement after reading a message
    - When acknowledgement is received by kafka, all offsets lesser than the acknowledged offset for that partition are auto marked as acknowledged
    - Consumers are pull based
    - Consumers can move back and forth in a partition by specifying the offset value from where they want to read the messages.
    - A single consumer will read data from all partitions of a topic
    - **Consumer Group**
        - Consumers that share the same group ID constitute a consumer group
        - Consumer groups are used to create parallelism and thus scalability
        - Brokers ensure each partition of a topic is being read by only a single consumer in the consumer group
        - Partitions are equally distributed among consumers in a consumer group
        - Each consumer is usually assigned partitions residing in the same broker
        - If number of partitions is equal to number of consumers in the group, then each consumer is assigned a single partition

        - If number of consumers in the group is more than the number of partitions, the extra set of consumers will not get any data
        - When a consumer is added or removed from a consumer group, a consumer rebalancing action is triggered

Other Components

- **Stream Processors / Kafka Streams**
    - Read from a kafka topic and write to a kafka topic
    - Allow some trivial transformations to be done
    - Allows to join data, fix out of order messages etc
- **Connectors / Kafka Connect**
    - Tool for scalably and reliably streaming data between Apache Kafka and other systems
    - Connectors can move large collections of data into and out of Kafka
    - Example: MirrorMaker

About the author

I am Karthik, I am a software professional and love solving data problems. I currently work as a senior engineer in a global IT firm and have also had the pleasure of taking startups from drawing board to production. In this blog, I intend to share my learnings and contribute to the community.

Feel free to reach out to me over mail at karthik[at]barrelsofdata.com or through my LinkedIn profile.

Topics

Apache Hadoop

Apache Kafka

Apache Spark

Apache Zookeeper

DevOps

Google Cloud Platform

Spring

Privacy policy | contact[at]barrelsofdata.com