**Apache Kafka**

Apache Kafka is a software platform which is based on a distributed streaming process. It is a publish-subscribe messaging system which let exchanging of data between applications, servers, and processors as well.

**Streaming process**

A streaming process is the processing of data in parallelly connected systems. This process allows different applications to limit the parallel execution of the data, where one record executes without waiting for the output of the previous record. Therefore, a distributed streaming platform enables the user to simplify the task of the streaming process and parallel execution.

**Kafka core API’s**

* Producer:
* Consumer:
* Streams: Consumes an input stream from one or more topics and produce an output stream to one or more output topics.
* Connector: This API executes the reusable producer and consumer APIs with the existing data systems or applications.

**Brokers**

A broker is a container that holds several topics with their multiple partitions. The brokers in the cluster are identified by an integer id only. Kafka brokers are also known as Bootstrap brokers because connection with any one broker means connection with the entire cluster. Although a broker does not contain whole data, but each broker in the cluster knows about all other brokers, partitions as well as topics.

**Topics**

Topic is a name used to store and publish a particular stream of data.

**Partitions**

A topic is split into several parts which are known as the partitions of the topic. Each message gets stored into partitions with an incremental id known as its Offset value.

Note: The data once written to a partition can never be changed. It is immutable. The offset value always remains in an incremental state, it never goes back to an empty space. Also, the data is kept in a partition for a limited time only.

**Replication**

Replication to secure data loss even when a broker fails down. In the presence of a leader, none of the followers is allowed to serve the client's request. These replicas are known as ISR(in-sync-replica). So, Apache Kafka offers multiple ISR(in-sync-replica) for the data.

**Producers**

A producer is the one which publishes or writes data to the topics within different partitions.

**Message Keys:** The key enables the producer with two choices, i.e., either to send data to each partition (automatically) or send data to a specific partition only.

**Acknowledgment:** It means the producer can get a confirmation of its data writes by receiving the following acknowledgments.

* acks=0: This means that the producer sends the data to the broker but does not wait for the acknowledgement
* acks=1: This means that the producer will wait for the leader's acknowledgement.
* acks=all: Here, the acknowledgment is done by both the leader and its followers.

**Consumer**

A consumer can easily read data from multiple brokers at the same time.

**Consumer Groups**

Each consumer present in a group reads data directly from the exclusive partitions. In case, the number of consumers are more than the number of partitions, some of the consumers will be in an inactive state. Somehow, if we lose any active consumer within the group then the inactive one can takeover and will come in an active state to read the data.roup automatically use a **'GroupCoordinator'** and one **'ConsumerCoordinator'**, which assigns a consumer to a partition.

**Delivery semantics**

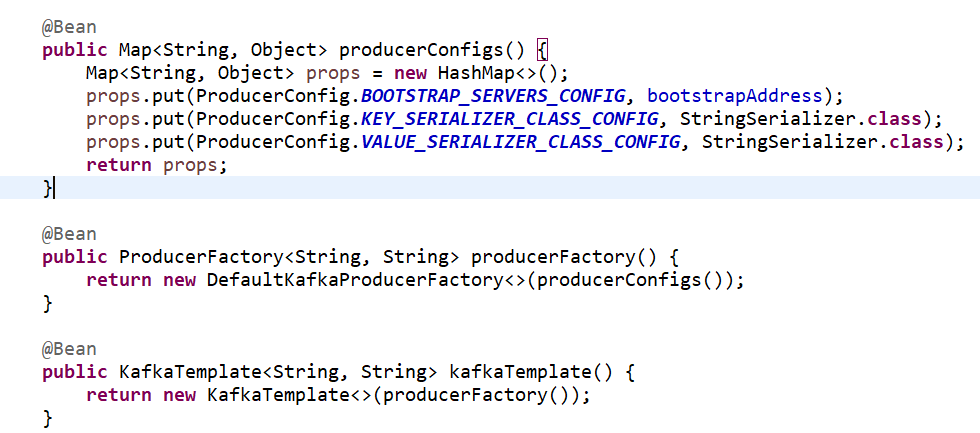
* **At most once:** Here, the offsets are committed as soon as the consumer receives the message.
* **At least once:** Here, the offsets are committed after the message has been processed.
* **Exactly once:** Here, the offsets can be achieved for Kafka to Kafka workflow only using the Kafka Streams API.

**Start ZooKeeper**

1. zookeeper-server-start.bat config\zookeeper.properties

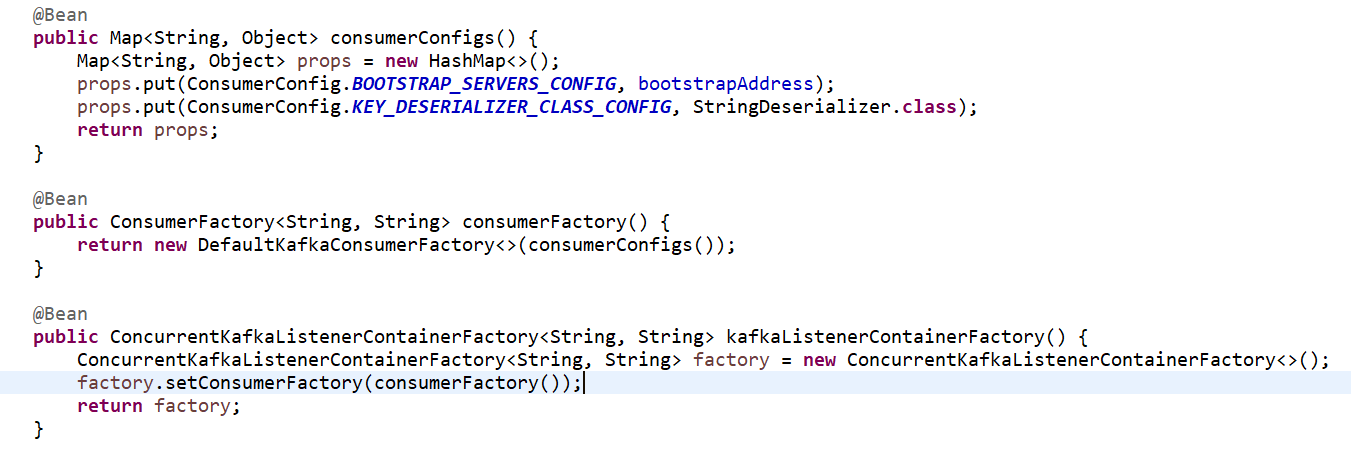
afka-topics.bat -zookeeper localhost:2181 -topic <topic\_name> --create --partitions <value> --replication <value>

**Spring boot with Kafka**



* **ProducerFactory** is responsible for creating Kafka Producer instances.
* **KafkaTemplate** helps us to send messages to their respective topic.
* **BOOTSTRAP\_SERVERS\_CONFIG** - Host and port on which Kafka is running.
* **KEY\_SERIALIZER\_CLASS\_CONFIG** - Serializer class to be used for the key.
* **VALUE\_SERIALIZER\_CLASS\_CONFIG** - Serializer class to be used for the value.

**Consumer configuration**



* **ConcurrentKafkaListenerContainerFactory-**The KafkaListenerContainer receives all the messages from all topics or partitions on a single thread.
* **@KafkaListener** to use this annotation we should add the **@EnableKafka** annotation on one of our **@Configuration** classes.
* We can also use the @KafkaListener annotation at class level. If we do so, we need to specify @KafkaHandler at the method level.
* @KafkaHandler(isDefault = true)
* It will try to match that type against the method signatures to find out which method to call.

**@KafkaListener attributes**

* topics
* groupId
* topicPartitions



**Kafka commands**

* zookeeper-server-start.bat C:\kafka\_2.13-2.6.0\config\zookeeper.properties
* kafka-server-start.bat C:\kafka\_2.13-2.6.0\config\server.properties
* kafka-topics.bat --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic Kafka\_Example
* kafka-console-consumer.bat --bootstrap-server localhost:9092 --topic Kafka\_Example --from-beginning
* kafka-console-producer.bat --broker-list localhost:9092 --topic Kafka\_Example\_json