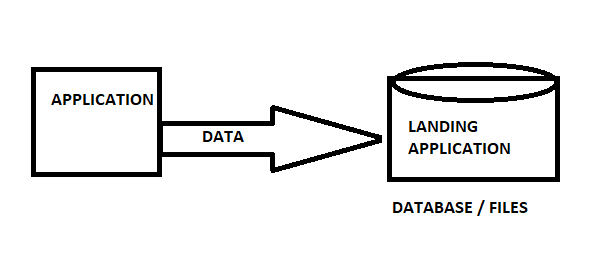
1. **DATA PROCESSING:**

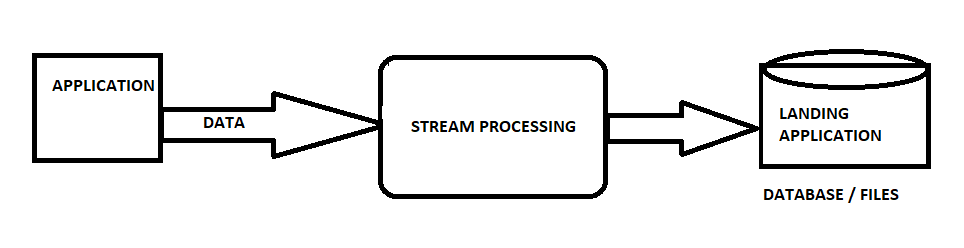
* Data processing is nothing but analysing a data set or fetching or whatever operation we do on top a data from a DB or a file system.

1. **BATCH DATA PROCESSING:**

* It’s a way of processing the data as a batch (group), probably in a regular sequence of time or whenever there is a need
* The process execution ends once after the processing of the data, so it must be kick started for next run ****

1. **STREAM DATA PROCESSING:**

* It’s a way of handling real time/live data, in other words whenever data flows in then processing happens on the data
* The process keeps running all the time until its broken, whether there is a single data or no data or enormous amount of data flows in, this will always process that.
* In other words, before data reaching the landing application, the data gets processed and used for some means

****

1. **KAFKA HISTORY:**

* Originated by **LinkedIn**
* **Open** sourced in the year of 2011
* **Confluent** was formed by people worked for Kafka in LinkedIn to work more specific on Kafka features.

1. **APACHE KAFKA:**

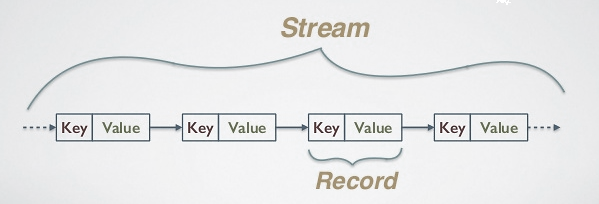
* Open sourced **and fault tolerant** **message queuing** and **stream processing** platform
* runs on **cluster** of one or more servers as distributed service.

1. **THREE MAJOR FUNCTIONS OF KAFKA:**

* **Publish-Subscribe** to stream of records, subscription can always be at topic level
* **Stores** streams of records
* **Processes** streams of records as they occur

1. **STREAM OF RECORDS:**

* It’s nothing but **continuous flow of records** from a source application
* **No** need to **send request for every new record** flow but we just receive them in tool
* **Never stops** until broken sue to circumstances

****

1. **RECORDS:**

* It’s a **key-value** pair
* Each record consists a **key, value and timestamp**

1. **DIFFERENT TYPES OF KAFKA CLUSTERS:**
2. Single Node Single Broker Cluster
3. Single Node Multiple Broker Cluster
4. Multiple Node Multiple Broker clusters
5. **DISTRIBUTED SERVICE KAFKA ARCHITECTURE:**

* Cluster of instances can be in **single node or multiple nodes** as per requirement

[](https://www.google.com/url?sa=i&rct=j&q=&esrc=s&source=images&cd=&cad=rja&uact=8&ved=2ahUKEwiF9P6ykJDhAhX97HMBHXbaCXQQjRx6BAgBEAU&url=https%3A%2F%2Fwww.tutorialspoint.com%2Fapache_kafka%2Fapache_kafka_cluster_architecture.htm&psig=AOvVaw39vHVKBDvnN-qlIjecnb-q&ust=1553150948770952)

1. **BROKERS:**

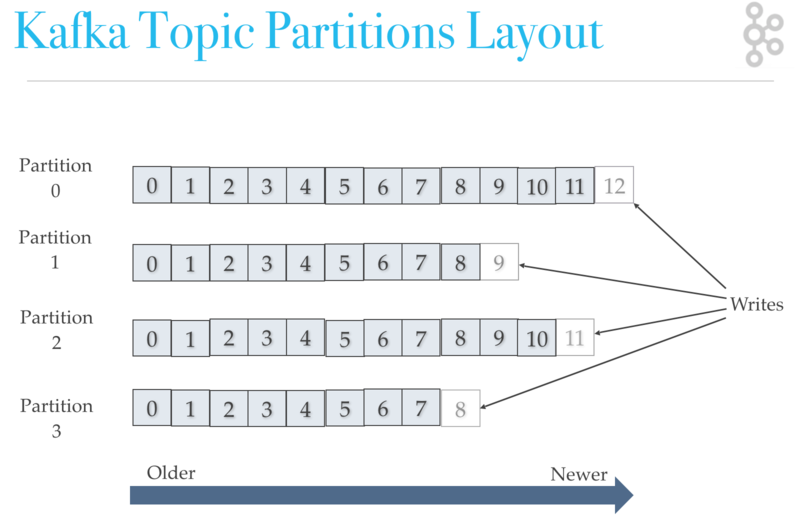
* **Every node** in the cluster that plays role of Kafka mediator between producer and consumer
* In other words, nothing but a node in cluster that runs Kafka
* Every broker will have a unique identification ID called broker-id

1. **ROLE OF ZOOKEEPER:**

* Helps each node to coordinate with the other nodes of the cluster
* Producers and consumers are notified by the zookeeper service about the **presence of new broker** in Kafka system
* Producers and consumers are notified by the zookeeper service **about the failure of a broker** in Kafka system

1. **TOPICS:**

* **Unique name** for a data stream stored in brokers
* Topics can simply be termed as a “**Category**” to store a record of data that can be defined by the user
* Always multi subscriber, means zero, one or many subscribers can subscribe to data on the topic



1. **TWO TYPES OF TOPICS:**

* Regular
* Compacted

1. **REGULAR TOPICS:**

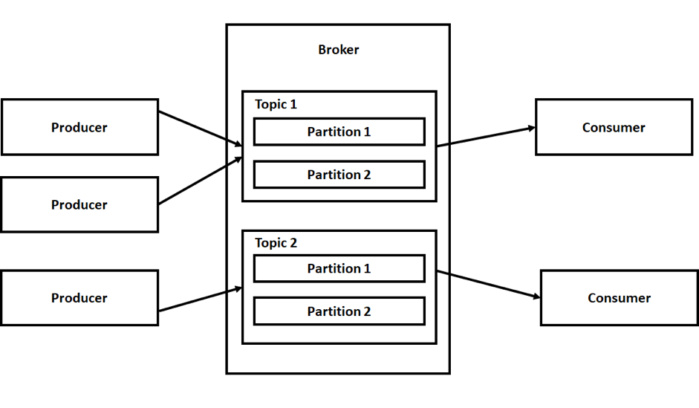
* Default retention period for topic is 7 days
* Space bound and Retention time plays major role
* If the time exceeds the retention period or space bound, then the Kafka will take care of deleting them to free storage space

1. **COMPACTED TOPICS:**

* Helps to keep the record for long
* Considers every new record with same key as updates and always keeps the latest message for the key.

1. **PARTITION:**

* Every Topic must have at least one partition
* Every partition has one broker as a leader and other brokers as followers/replicas
* Partition here is nothing but splitting a single topic into multiple parallelized parts
* Leader partition handles all the read and writes but if leader fails then one of the replicas will become leader automatically
* Partition size can be increased after creation but can’t be decreased

[](https://www.google.com/url?sa=i&rct=j&q=&esrc=s&source=images&cd=&cad=rja&uact=8&ved=2ahUKEwibiM72j5DhAhXf7nMBHespBn4QjRx6BAgBEAU&url=https%3A%2F%2Fwww.infoworld.com%2Farticle%2F3215165%2Fhow-to-use-apache-kafka-messaging-in-net.html&psig=AOvVaw0v_dFJJ6llESCr4xww9Jco&ust=1553150774766446)

1. **PARTITION LEADER:**

* Its partition leader’s responsibility to get the data from producer, stores it and sends back the acknowledgement
* Its leader’s responsibility to send back the requested data to the consumer

1. **PARTITION FOLLOWER:**

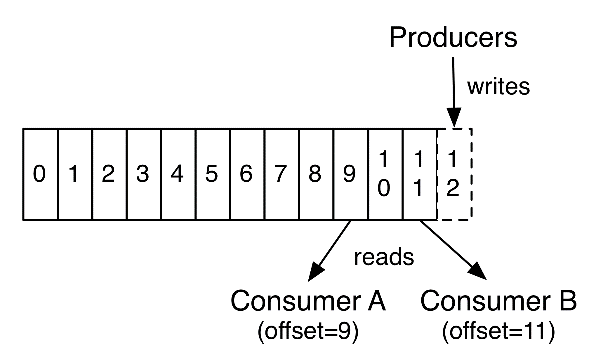
* Don’t talk to producer or consumer but with the leader
* If the leader goes down, then Kafka takes care of making any one follower as the leader

1. **PRODUCERECORDS:**

* Every produce record will have topic name, partition ID (optional), key (optional) and Value in it.
* If partition ID not specified, then records get stored in **key % number of partitions** to find partition
* If both partition ID and key are not specified, then **round robin** method followed

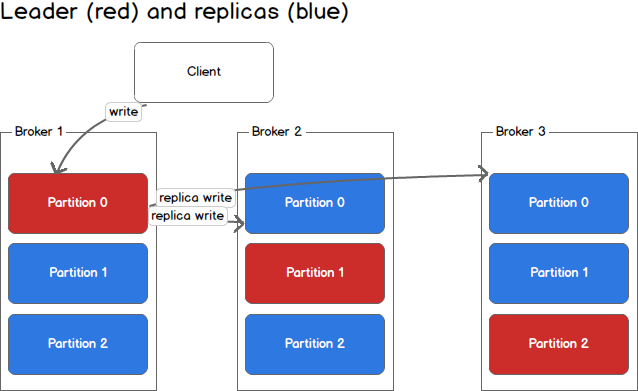
1. **OFFSET:**

* Unique identification as index for the records stored in partition like array kind
* Order will be as per the broker receives the record message
* Only metadata consumer maintains is offset of records



1. **REPLICAS:**

* Replicas remain as the backup partitions that helps to get the data’s restored on loss
* Replicas won’t be accessed by consumers directly in regular publish subscription messaging service
* Each broker acts as a leader for a partition and follower (replica store) for other partitions so load is well balanced in cluster



1. **REPLLICATION FACTOR:**

* its specified in topic level though it’s about how many leaders and followers are to be created for the partitions created under the topic
* if the replication factor is specified as 1 then only one partition is created in the node and no replicas will be created
* if the replication factor is specified as 3 then one leader partition is created in the node and 2 replicas in other nodes, in total 3
* though there is no restriction in numbers for replication-factor, 3 is reasonable but we can specify any number if we run-on cost-effective machines

1. **ISR / IN SYNC REPLICAS:**

* its nothing but the replicas in sync with the others (leader and followers), if some broker goes down in between and comes back then it may not be in sync with the leader and other followers.

1. **PRODUCER:**

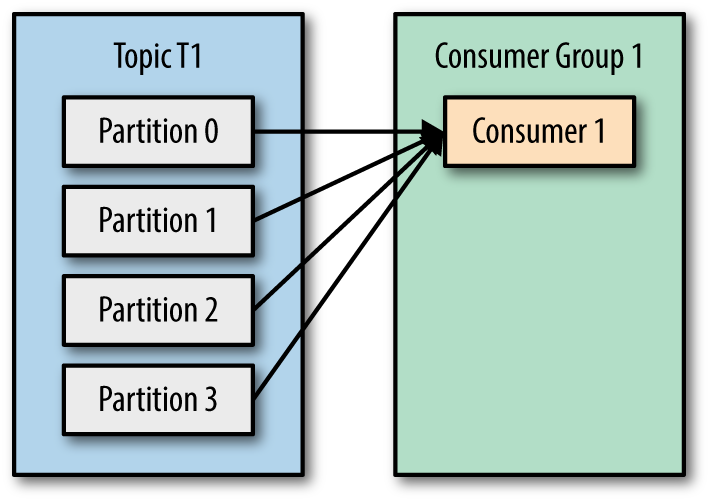
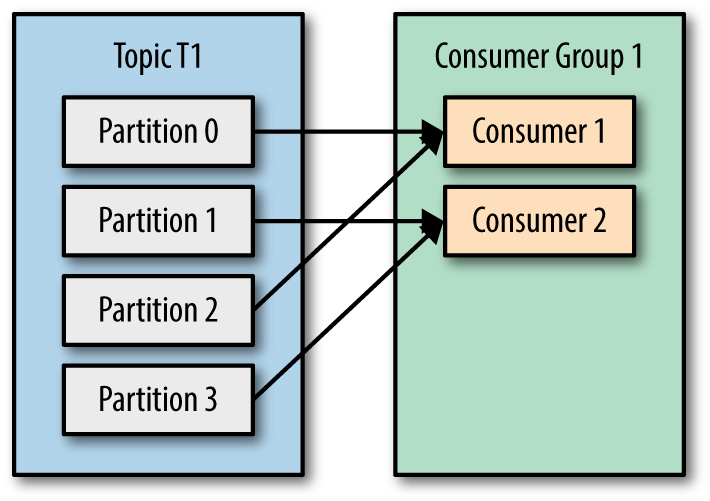
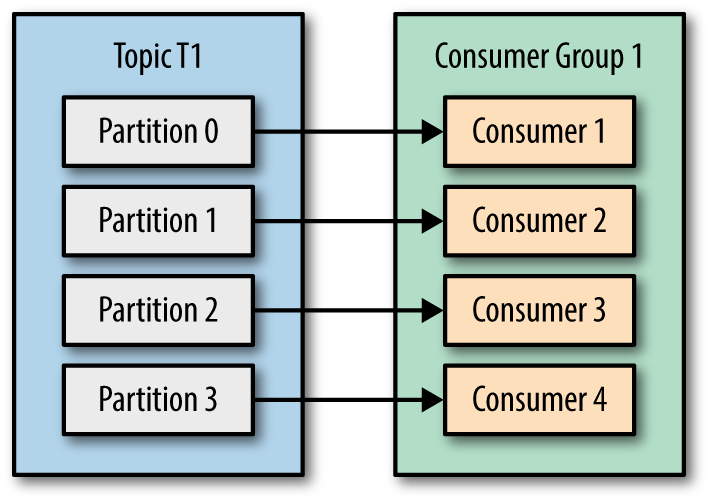
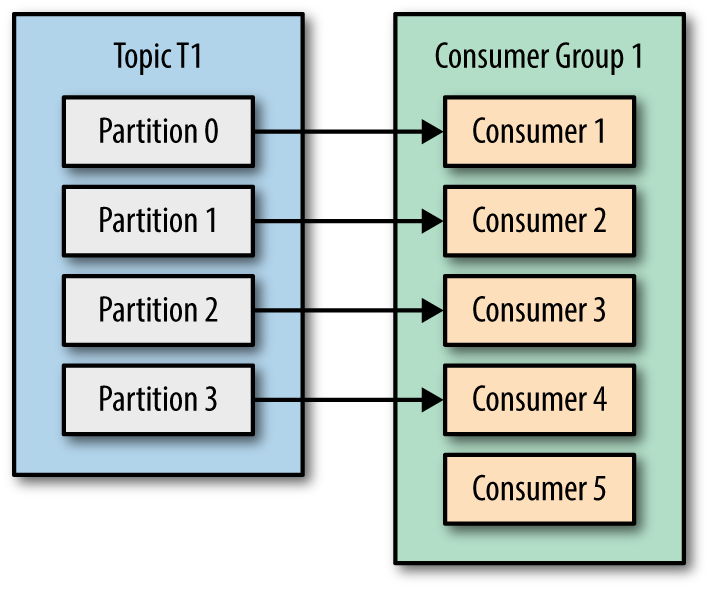
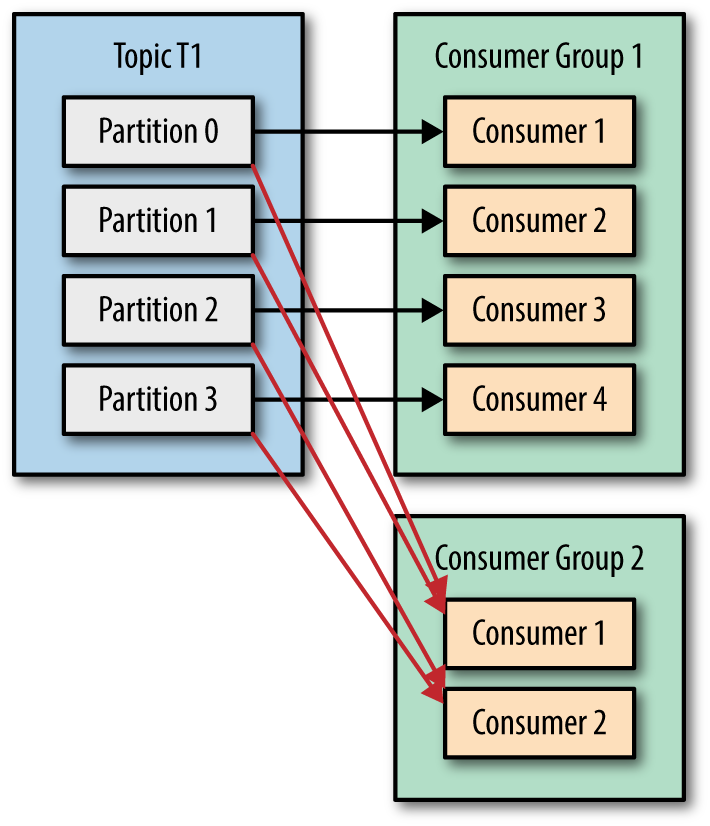
* Instances of an application that publishes data to the topics of their choice and responsible for choosing which record to store in which topic and which partition.

1. **CONSUMER:**

* Only metadata saved in the consumer is offset position of partition, in other words position of that consumer in logs
* Consumer controls the offset position for streaming so it can advance linearly, or it can consume records in any order it wants
* Means consumer can come into Kafka and leave without any impact on cluster or other consumer
* Kafka doesn’t allow more than two consumers to read data from same partition
* Aware of the number of partitions in the topic interested.
* Labels themselves in the Consumer group name,

1. **CONSUMER GROUP NAME:**

* Single application wants to read data using multiple consumer instances (Can be in same machine or in cluster of nodes) then combined group can be created for consumers of that application
* Consumers can join a group by using the same group id
* Each partition is consumed by exactly one consumer in a group at once, but single consumer can read data from multiple partition.

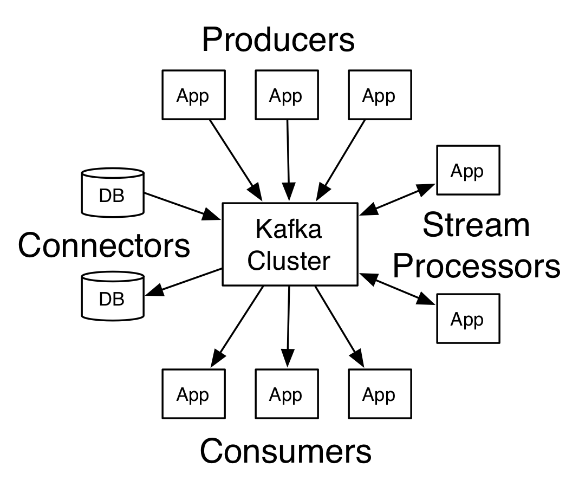
    

1. **GROUP COORDINATOR IN KAFKA:**

* Group coordinator resides in a cluster node, that coordinates the consumers from a consumer group, like managing the joining, relieving and reassigning of flow from partition to consumers. (Manages the consumer group members)
* First consumer who reaches the group coordinator becomes the leader
* Rebalance is something when there is a need of reassigning the flow of data from partition to another consumer due to a consumer instance failed.
* And no consumer instance can read that partition until rebalancing is done.

1. **MAJOR APIs in KAFKA:**

* Producer API
* Consumer API
* Streams API
* Connector API
* AdminClient API



1. **MAVEN DEPENDENCIES FOR MAJOR APIS**

* Can use below dependency to use Producer API, Consumer API, AdminClient API

<**dependency**>

     <**groupId**>org.apache.kafka</**groupId**>

     <**artifactId**>kafka-clients</**artifactId**>

     <**version**>2.1.0</**version**>

</**dependency**>

* Can use below dependency to use Streams API

**<dependency>**

**<groupId>**org.apache.kafka**</groupId>**

**<artifactId>**kafka-streams**</artifactId>**

**<version>**2.1.0**</version>**

**</dependency>**

1. **PRODUCER API:**

* Allows an application to publish a stream of records into one or more topics

1. **CONSUMER API:**

* Allows an application to subscribe to one or more topics and process the stream of records produced to them.

1. **STREAMS API:**

* Allows an application to act as a stream processor, consuming an input stream from one or more topics and producing an input stream to one or more topics
* Effectively transforming input streams to output streams

1. **CONNECTOR API:**

* Allows building and running reusable producers and consumers to connect topics to existing application
* Example, a connector to relational database means it capture every change to the tables

1. **STEPS TO START WITH KAFKA:**
2. Start Zookeeper
3. Start brokers
4. Create topic and partitions
5. Start Producer
6. Consume from Consumer
7. **KAFKA DOWNLOAD PATH:**
8. Download the Apache Kafka latest version from below URL,

<https://kafka.apache.org/downloads>

1. Un tar the downloaded Kafka file
2. **IMPORTANT BROKER RELATED CONFIGURATION:**

* **Broker.id**

1. Unique identifier for a broker
2. Default value – 0 for first broker

* **Port**

1. It’s a port used by the broker to communicate with APIs
2. Default – 9092
3. No need to change it, if Kafka runs on multiple machine but must be changed for every broker instance if its run-in single machine

* **Log.dirs**

1. Where the Kafka logs get saved for the broker
2. Default- /tmp/Kafka-logs

* **Zookeeper.connect**

1. Takes the zookeeper instance details to connect with the broker
2. It is critical that every broker knows zookeeper address, so this helps
3. Default- localhost:2181

* **Delete.topic.enable**

1. It’s with true/false value
2. If we need to enable the topic can be deleted from broker, then this helps
3. Default- false

* **Auto.create.topics.enable**

1. If a producer sends a message to nonexistence topic, then topic will be created in broker, but it’s not recommended as topic should be created by topic management tool only in production
2. Default-false

* **Default.replication.factor**

1. This is when the auto topic creation is enabled, how many replication factors must be considered
2. Default – 1

* **Num.partitions**

1. This is when the auto topic creation is enabled, how many numbers of partitions are to be considered
2. Default -1

* **Log.retention.ms**

1. This is related to retention of the data’s in time for a partition and important configuration
2. Default – 7 days

* **Log.retention.bytes**

1. This is related to retention of the data’s in size for a partition and important configuration
2. **STEPS IN SIMPLE JAVA CODING ON KAFKA STREAM:**

STEP -1: Instantiate the **Properties** instance

STEP-2 : Configure the properties instance with with application\_id, bootstrap-servers, SeDes

STEP-3 : Instantiate the **SteamsBuilder** instance

STEP-4 : Create an object instance for **KStream** and initialize using the StreamsBuilder class’ **stream** method

STEP-5 : Create an instance for **KTable** to process the data and get it in format

1. **STARTING THE ZOOKEEPER INSTANCE:**

* can use the convenience script packaged with Kafka to get a single-node Zookeeper instance

WINDOWS:

***C:\kafka-2.1.1-src\bin>zookeeper-server-start.bat ..\..\config\zookeeper.properties***

LINUX:

1. **STARTING THE KAFAKA SERVER:**

WINDOWS:

C:\kafka\_2.12-2.1.1\bin\windows>kafka-server-start.bat ..\..\config\server.properties

LINUX:

**> *bin/kafka-server-start.sh config/server.properties***

1. **CREATING A TOPIC:**

**WINDOWS:**

C:\kafka\_2.12-2.1.1\bin\windows>kafka-topics.bat --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 -topic topicname

**LINUX:**

**> *bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 –topic topic-name***

1. **LISTING THE EXISTING TOPICS:**

Windows:

C:\kafka\_2.12-2.1.1\bin\windows>kafka-topics.bat --list --zookeeper localhost:2181

Linux:

**> *bin/kafka-topics.sh --list --zookeeper localhost:2181***

1. **STARTING AND SENDING MESSAGES FROM PRODUCER CONSOLE:**

* Kafka comes with a command line client that will take input from a file or from standard input and send it out as messages to the Kafka cluster.
* By default, each line will be sent as a separate message.

WINDOWS:

C:\kafka\_2.12-2.1.1\bin\windows>kafka-console-producer.bat --broker-list localhost:9092 -topic topicname

LINUX:

**> *bin/kafka-console-producer.sh --broker-list localhost:9092 -- topic topic-name***

***> This is a message***

***> This is another message***

1. **STARTING AND CONSUMING MESSAGES IN THE CONSUMER CONSOLE:**

**WINDOWS:**

**C:\kafka\_2.12-2.1.1\bin\windows>kafka-console-consumer.bat --bootstrap-server localhost:9092 -topic topicname --from-beginning**

**LINUX:**

***> bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 –topic topic-name --from-beginning***

***> This is a message***

***> This is another message***

1. **SETTING UP THE MULTI CLUSTER BROKER USING PROPERTIES:**

***>copy config/server.properties config/server1.properties***

***>copy config/server.properties config/server2.properties***

* Then we can edit both files to changes the listeners port number, broker-id, log directory accordingly

***config/server-1.properties:***

***broker.id=1***

***listeners=PLAINTEXT://:9093***

***log.dirs=/tmp/kafka-logs-1***

***config/server-2.properties:***

***broker.id=2***

***listeners=PLAINTEXT://:9094***

***log.dirs=/tmp/kafka-logs-2***

* Broker-id must be unique, and it is permanent name for every broker

1. **STARTING UP THE MULTI CLUSTER BROKERS:**

**> bin/kafka-server-start.sh config/server1.properties**

**> bin/kafka-server-start.sh config/server2.properties**

1. **CREATING THE TOPIC WITH REPLICATION FACTOR FOR MULTI CLUSTER:**

***> bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 3 --partitions 1 --topic my-new-topic***

1. **DESCRIBING A TOPIC ON CLUSTER:**

***> bin/kafka-topics.sh --describe --zookeeper localhost:2181 --topic my-new-topic***

**OUTPUT:**

***Topic:my-new-topic   PartitionCount:1    ReplicationFactor:3 Configs:***

***Topic: my-new-topic Partition: 0    Leader: 1   Replicas: 1,2,0 Isr: 1,2,0***

1. **DELETE A TOPIC ON CLUSTER:**

WINDOWS:

Kafka-topics.bat –delete –zookeeper localhost:2181 -topic topicname

1. **CREATE THE TOPIC IN COMPACT MODE:**

WINDOWS:

C:\kafka\_2.12-2.1.1\bin\windows>kafka-topics.bat --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 -topic topicname --config cleanup.policy=compact