**Apache Kafka**

**Introduction**

**What is Kafka ?**

Apache Kafka is a distributed publish-subscribe messaging system and a robust queue that can handle a high volume of data and enables you to pass messages from one end-point to another. Kafka is suitable for both offline and online message consumption. Kafka messages are persisted on the disk and replicated within the cluster to prevent data loss.

**Main goals of Kafka ?**

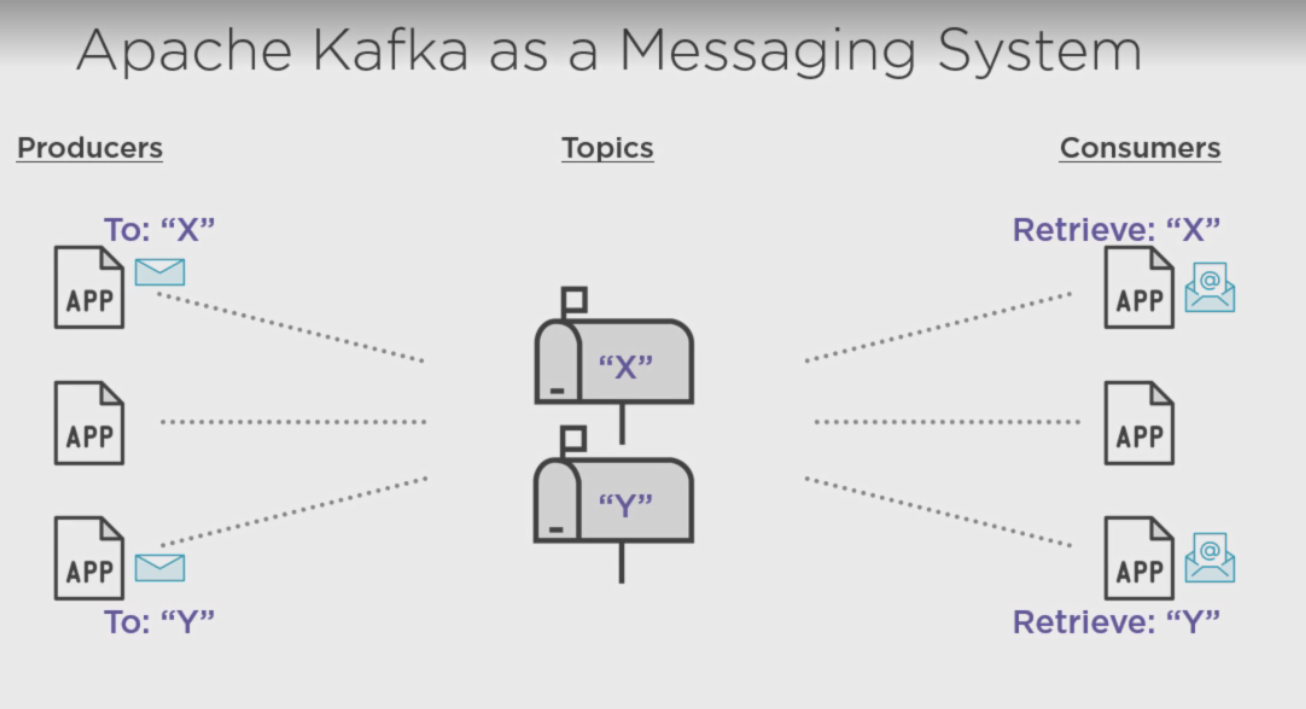
* High throughput
* Horizontal scalability
* Reliable and durable
* Loosely coupled producer and Consumers
* Flexible publisher subscriber semantics

**Why not existing pub sub messaging system ?**

Existing pub-sub architecture depends on a concept called message queue, it has some problem

* If there a huge data is getting transferred from publisher and subscriber is not that fast as publisher the system can be broken.
* We do not store the messages in messaging system. Suppose publisher publishes a message subscriber receives the message from the message queue and process it, and due to application fault of subscriber it is unable to process the message. If after the fault resolved, if subscriber tries to process the message it is not there in the queue.

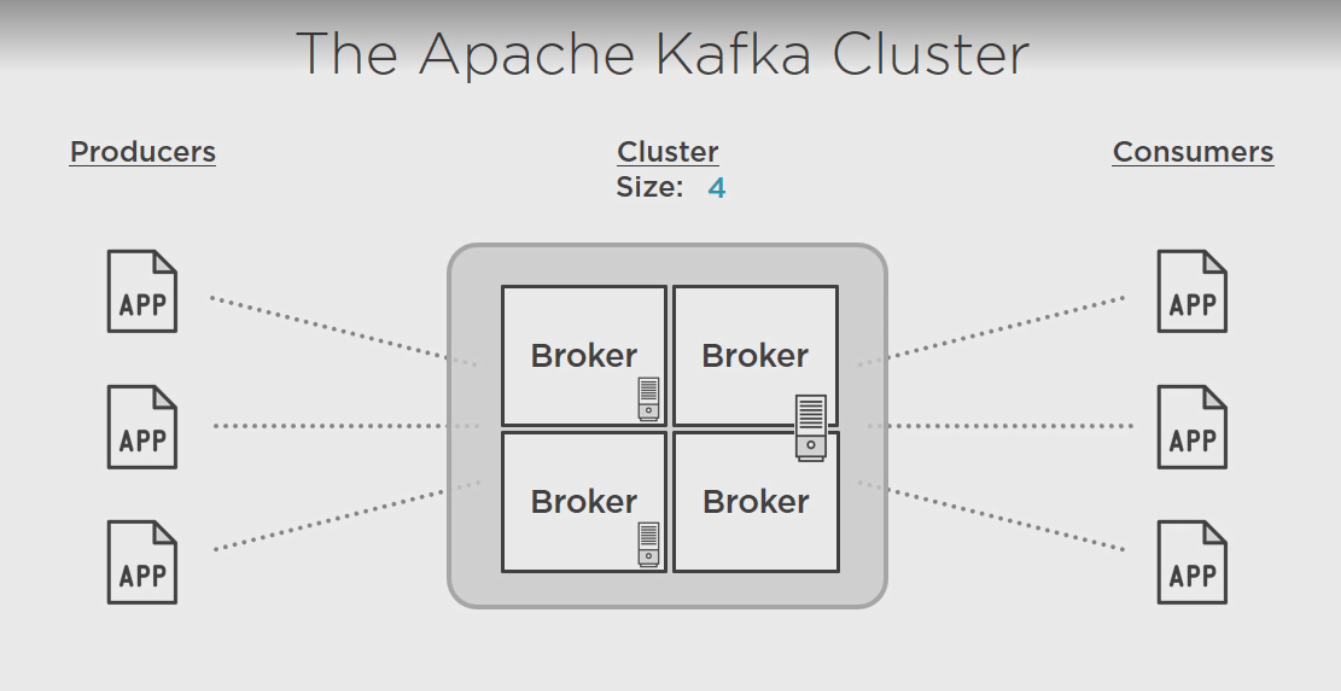
**Kafka as pub sub model**



Publishers are called producer and subscriber are called consumer in Kafka messaging system. producers send messages in a particular address and Consumer retrieve the messages from particular addresses shown in the picture.

The address is called tpoics. The messages categorized by tpic are stored in broker, which is basically a physical system.

**What is Kafka cluster ?**

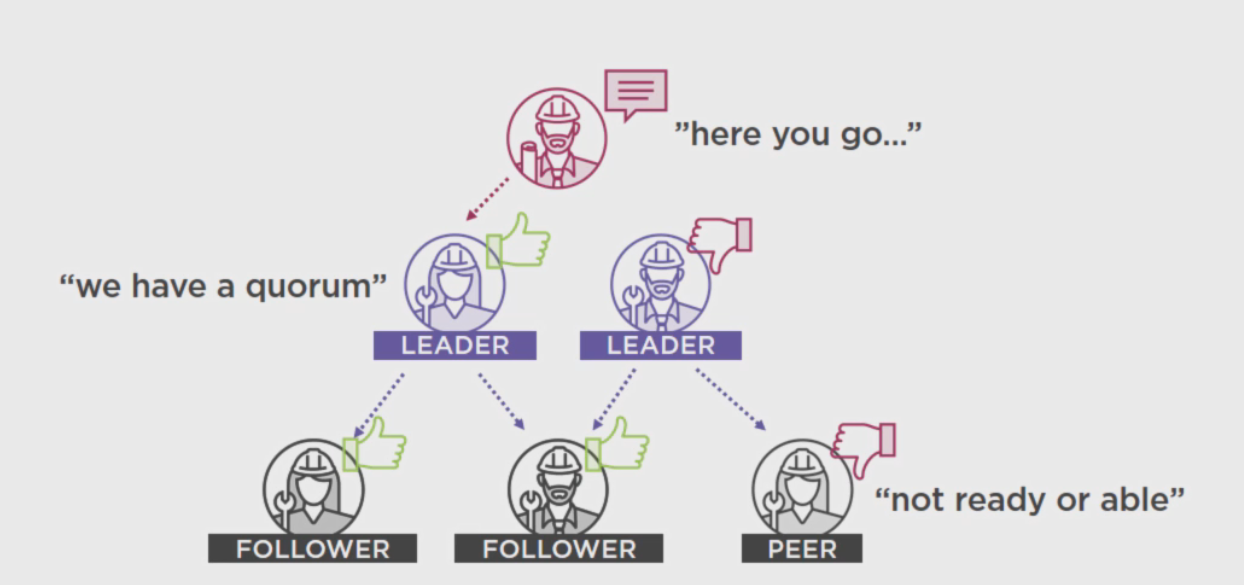


A broker is an application installed in a machine. If a 2 brokers are installed in

different machine cluster is 2, if in a same machine then also cluster is two. In the figure above 2 brokers are in different machine, another two brokers are in same machine, total cluster = 4, machine = 3. The group of broker is called kafka cluster.

**What is distributed system ?**

Distributed system is a collection of resources that are introduced to achieve a specific goal or function. It consists of multiple workers or nodes. Here is the distributed system looks like



One controller which assign some leaders, and leaders have some followers. Controller’s task is to manage the rest of the worker nodes in a distributed system.

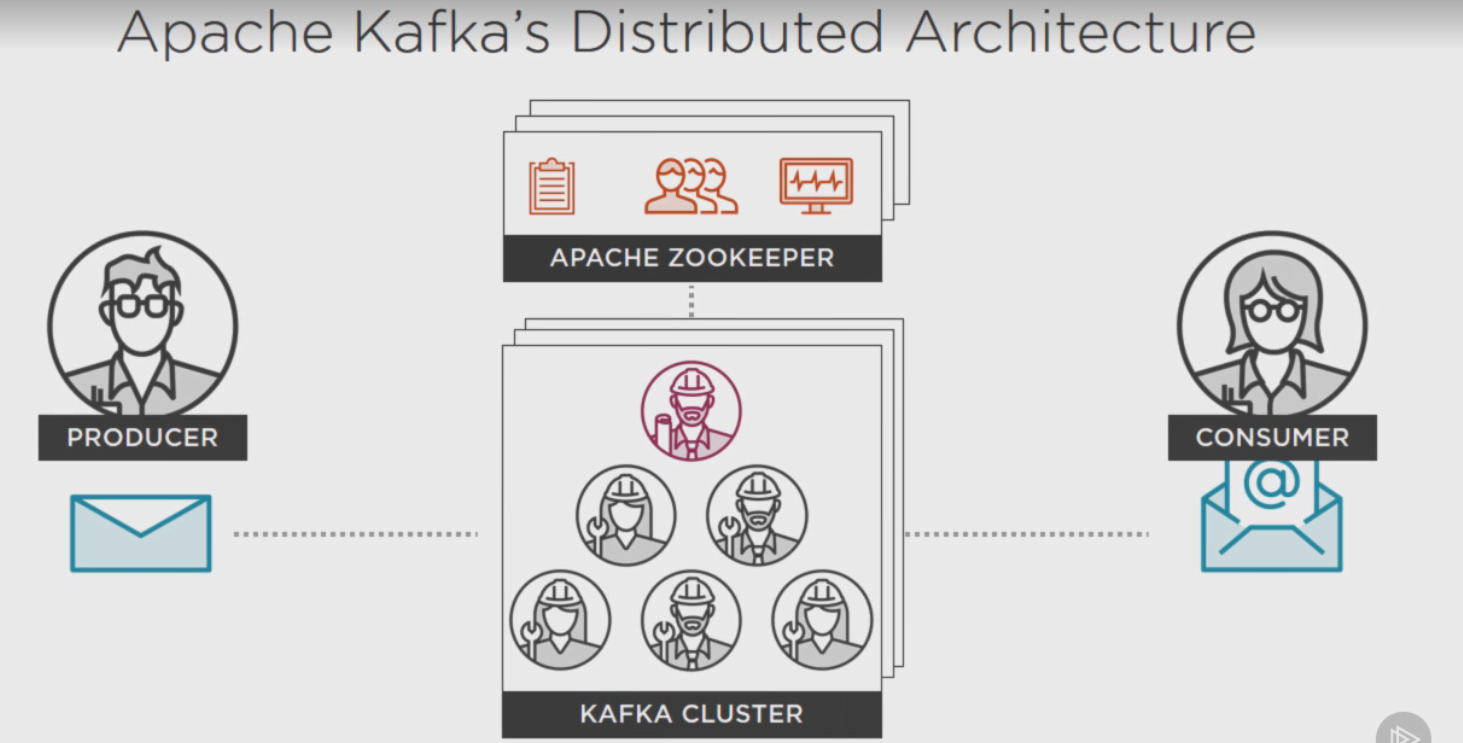
**Need of zookeeper**

There are several task required to manage the entire distributed system, incase of kafka it is called kafka cluster.

* Worker node membership and naming
* Configure management.
* leader election
* health status

All these tasks are executed by apache zookeeper.

Zookeeper is a centralized service for maintaining metadata about a cluster of distributed nodes.



each of the kafka clusters and zookeepers can be scalable to handle large number of messages.

**Kafka installation directory structure**

Kafka home

libs

config

bin

libs contains all the dependent jars to run kafka, as it contains the zookeeper jar , we don’t need to install zookeeper separately, it comes with kafka installation package. config floder has some properties file

server.propeties is responsible for kafka server

zookeepr.proprties is responsible for zookeepr server

bin/widows contains the batch files required to run kafka and zookeeper.

Here is the sequence of running

* First run the zookeeper from comnd prompt

bin\windows\zookeeper-server-start.bat config\zookeeper.properties

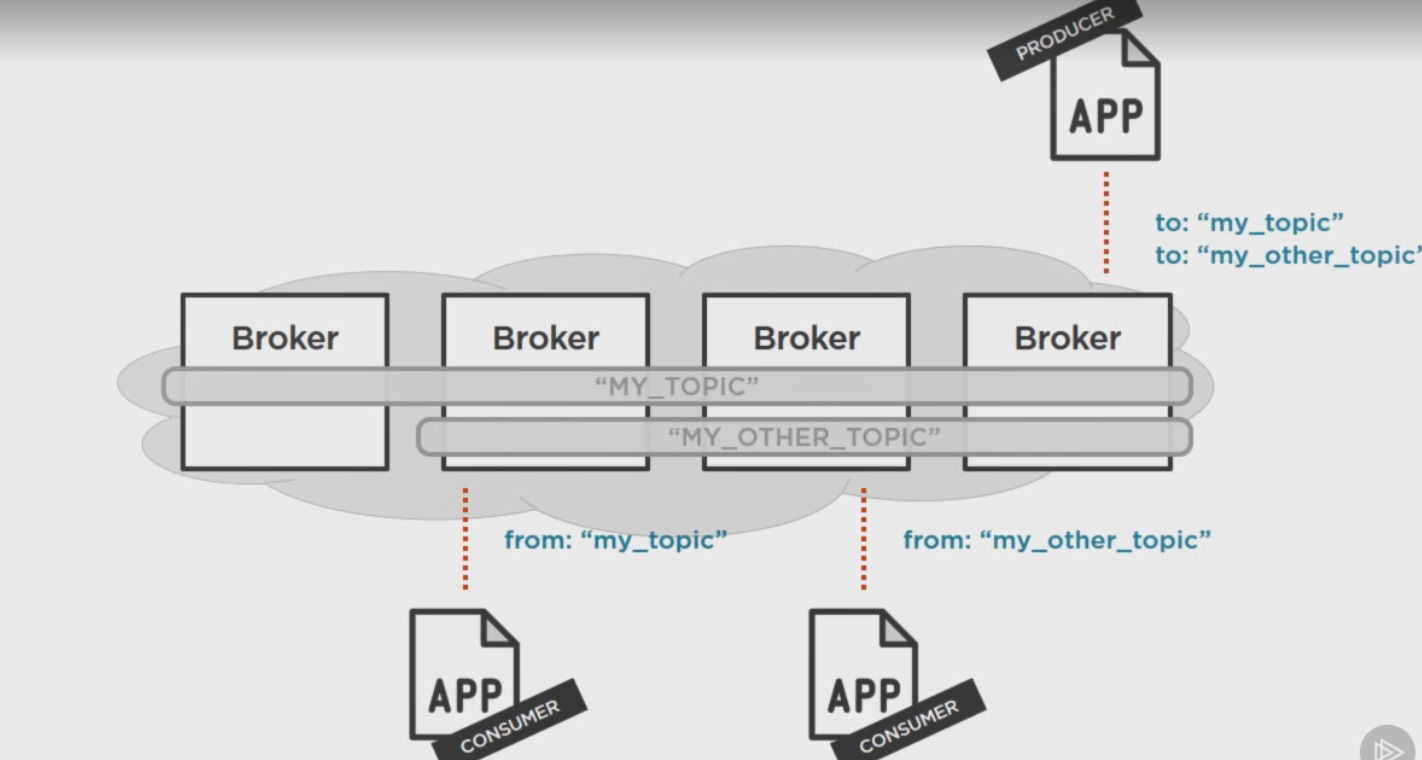
* second run the kafka from command prompt

bin\windows\kafka-server-start.bat config\server.properties

**Kafka topic and broker & producer**

**Brief overview of topic**

From above discussion we know topic is an address where producers publish the message and consumers consume the message. In kafka the topics are the treated as logical entity which is physically represented as log. here is the high level picture of topic. A topic can be distributed over multiple broker.

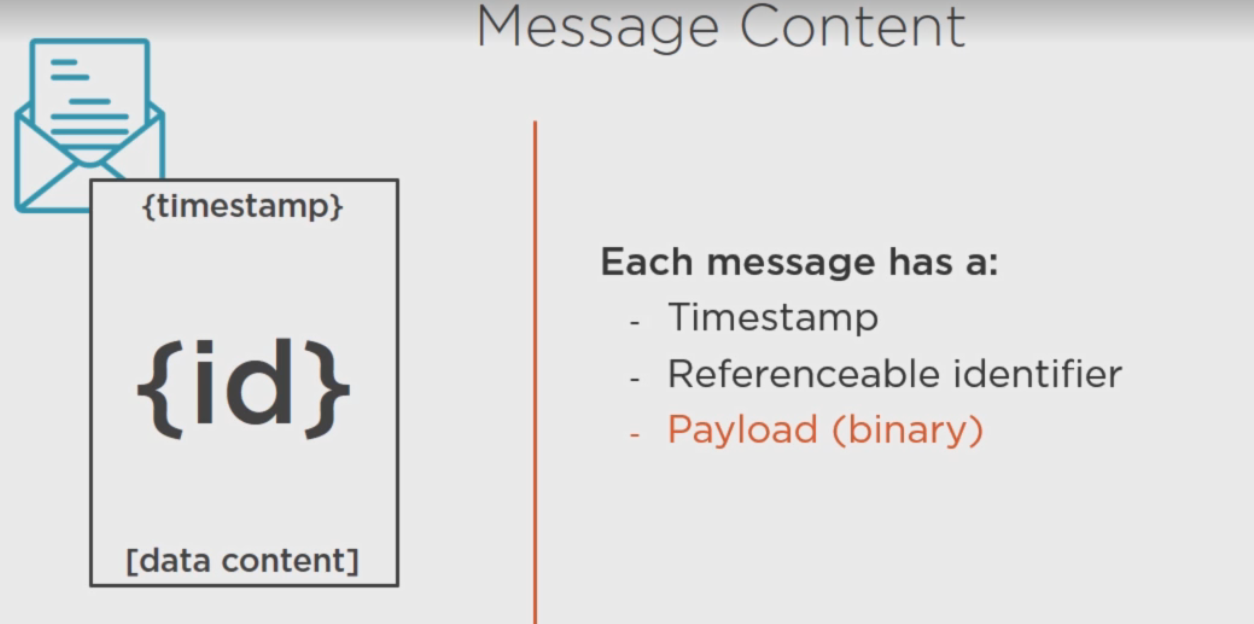


**Event sourcing**

An architectural style or approach to maintaining an application’s state by capturing all changes as sequence of time ordered., immutable events.

Kafka follows this same architecture the store the messages in a time order manner in a topic.

**Message content**



**What is message offset ?**

It is a placeholder which holds the last read position . It is maintained by the kafka consumer.

A consumer can read the messages from a topic from the beginning or from a specific position if it already read few messages and trying to resume. Here green colored app is reading from beginning and blue is reading from3rd position . The offset will be increment itself as it continues reading





**Message retention policy**

* kafka retains all published messages regardless of consumption
* Retention period is configurable. default it is 168 hours or 7 days. after that they were removed to give room for new messages.
* Retention period is defined on per topic basis.

**Create a topic and produce consume a simple message**

1. Start zookeeper and kafka server, zookeeper starts at 2181 and kafka at 9092
2. go to bin/windows and enter

kafka-topics.bat --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic myTopic

there will be directory created inside kafka-logs folder called myTopic-0, there two files created one is \*.log and another is \*.index

1. from the same directory start a producer

kafka-topics.bat --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic myTopic

1. Open another command prompt and start consumer

kafka-console-consumer.bat --zookeeper localhost:2181 --topic myTopic --from-beginning

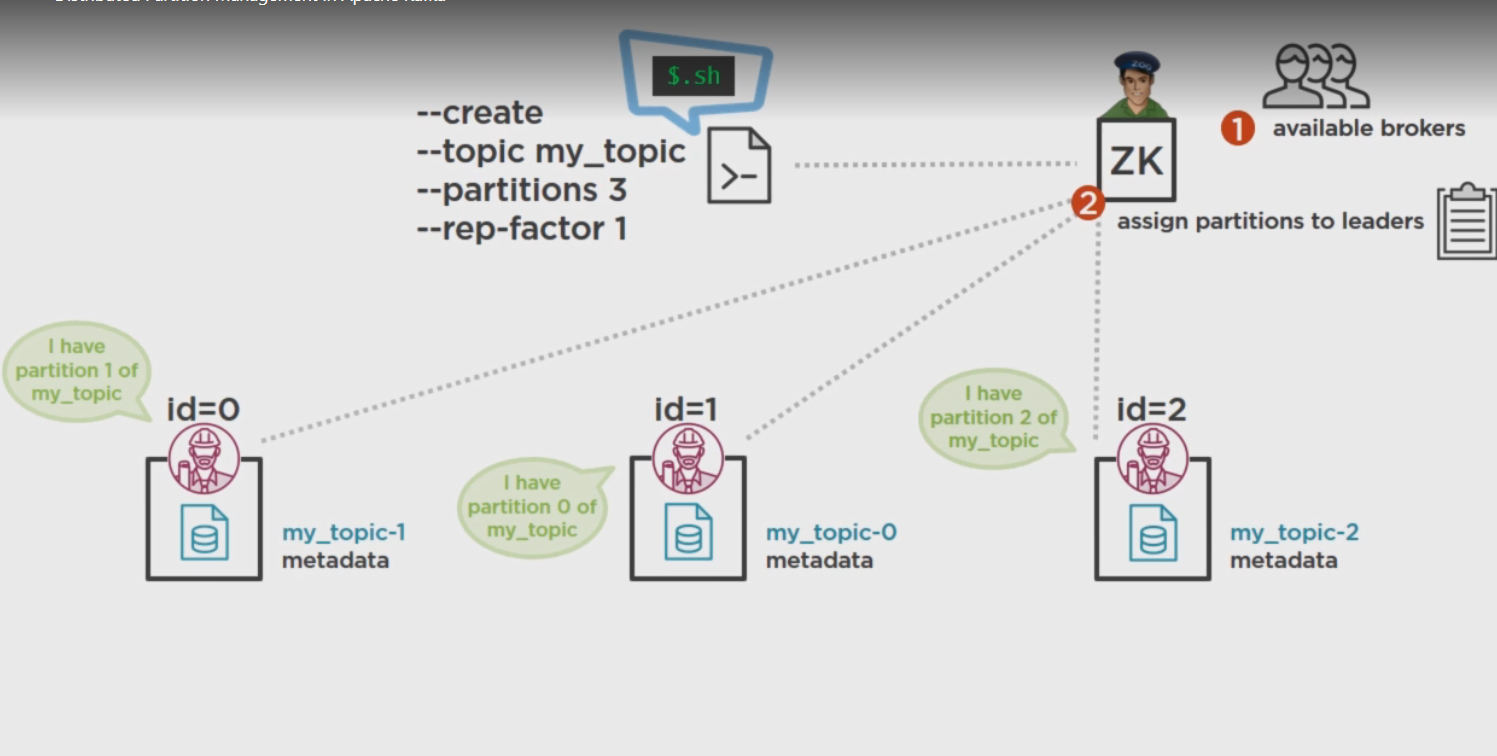
1. write messages in producer window one by one, and we can see that in consumer window. We can start consumer at any point of time, as we mentioned –from-beginning that means it will read the messages from the topic from first timestamp
2. .log and .index file now have some entry

**Kafka Partition basic**

At the time of creating topic we use --partitions 1 --topic myTopic

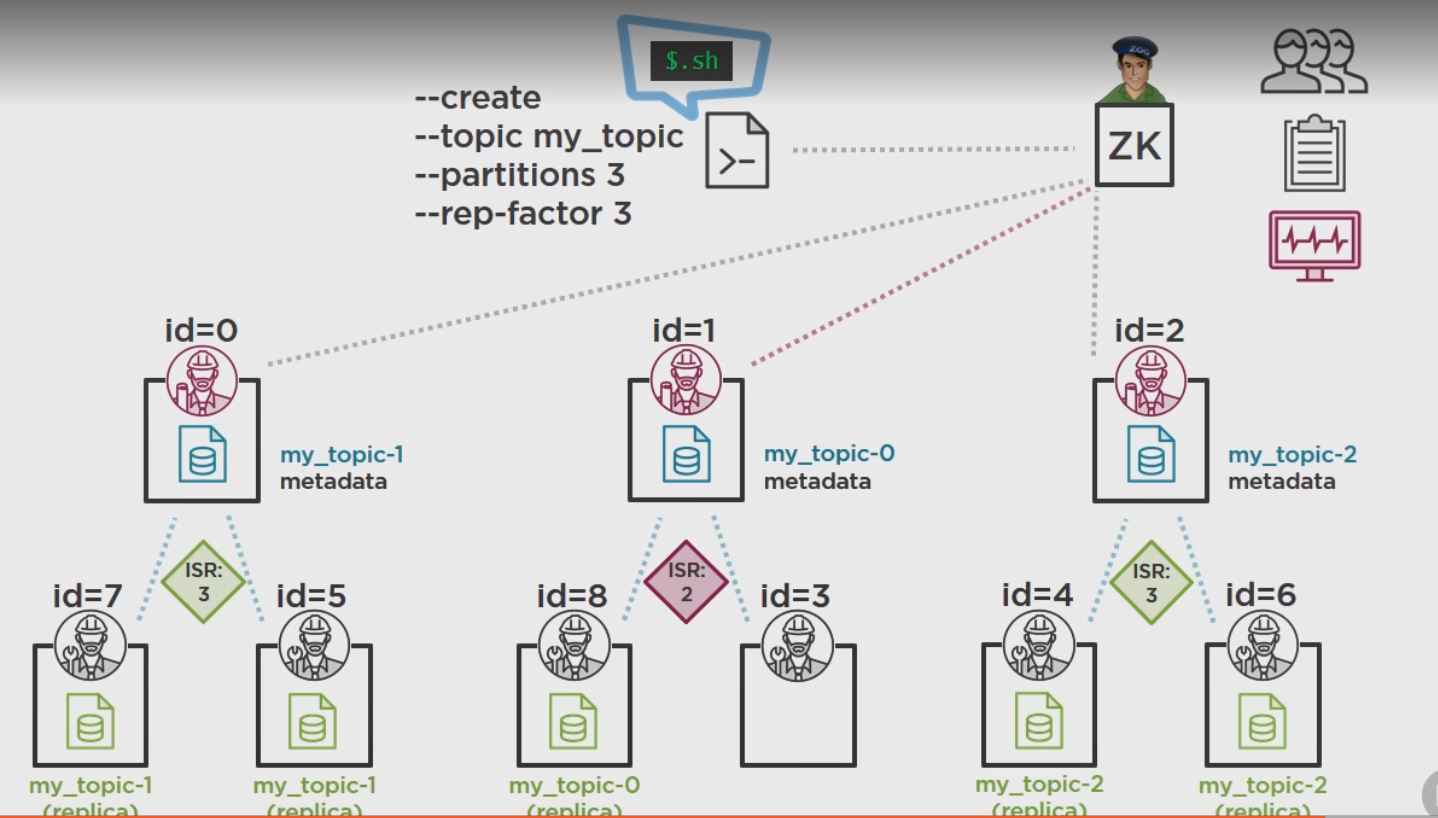
instead of that we can call --partitions 3, it will create 3 partitions of a single topic myTopic, suppose there are be 3 broker nodes(it can be of any number does not depened on number of partition) which contain the messages. One single partition cannot be split again by number of brokers, they can be replicated. For each partition we can have separate directory. as example

my\_topic-0, my\_topic-1 my\_topic-3



**Replication factor**

Like partition we can replicate each partition depending on replication factor, suppose we have replication factor 3 , then each broker becomes the leader and replicate their content with 2 brokers. Here ISR is a In Sync Replica which counts the number of replicated broker available for each partition. the ISR value should be at least 1 for each partition , if any node faces any issue.



**Kafka cluster vs partition vs replication factor**

Messages produced by Producer stored in kafka broker , managed by zoo keeper and consumed by consumer or consumer group. we can create 1 or more broker in kafka server, that is called number of broker in a kafka cluster. Each topic is divided by one or more partition, which is the smallest possible unit of a topic. each partition can be stored in one broker. They can be replicated but not further divided. As example there can be 2 brokers, 3 topics and 10 partitions.

broker 1 contains topic 0 partition 0 to 3

broker 2 contains topic 1 partition 0 to 3

broker1 can again contains remaining topic2 (10-(3+3)) = 4 partitions

So broker1 can contain 6 partitions of 2 topics.

broker2 can contain 4 partitions and 2 topics

Now to ensure fault tolerance kafka provides replication factor. If in this scenario replication factor is 3 then each broker will form a team of 3 brokers. the new brokers are followers which will copy the topics from the leader broker. So in this example broker 1 and broker 2 each has 2 followers, each follower contains same number of partitions as they are just a replica of broker 1 and broker2 respectively.

Number of brokers : 2

Number of topics : 3

Number of partitions : 10

After replication

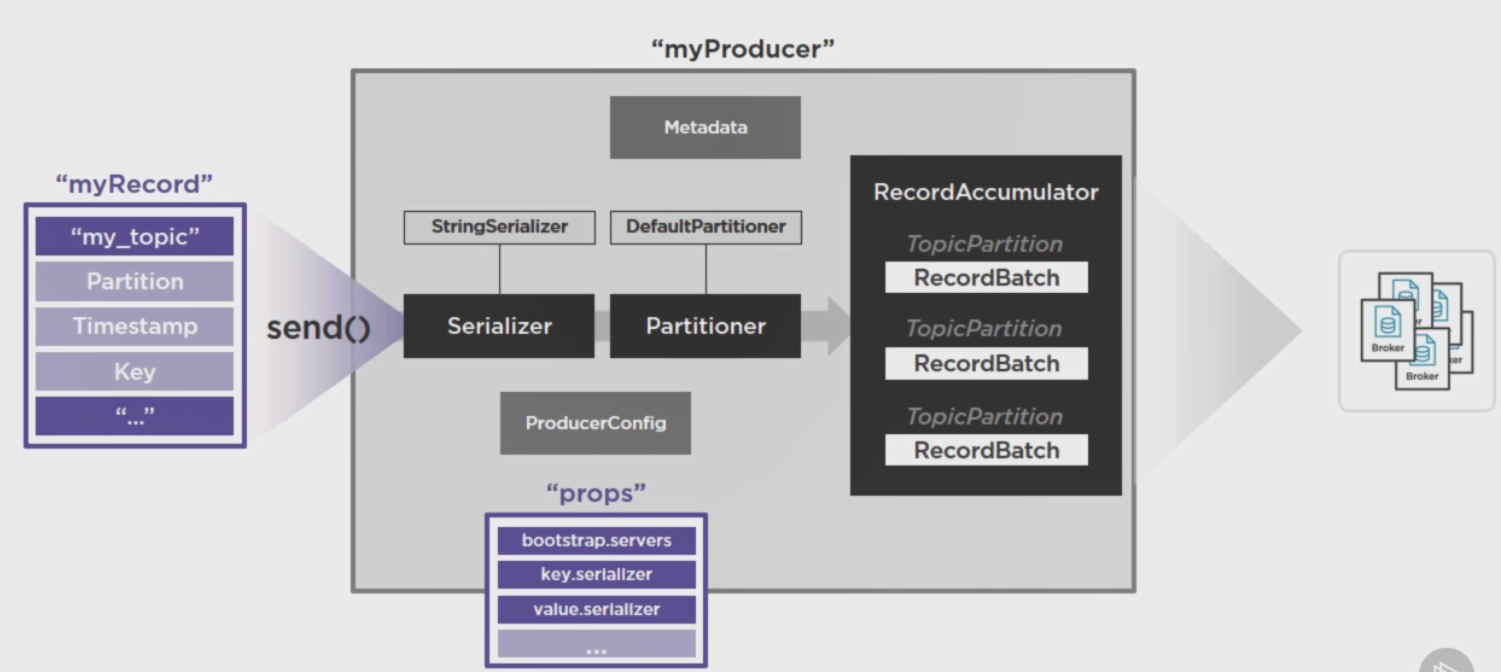
Number of broker : 3 \*2 = 6

number of topics : 3

Number of unique partitions : 10

Total partitions : 4\*3 + 6\*3 = 30

**Message producing technique :**



Kafka producer first read the metadata from the zookeeper and acquire the connection. Metadata contains all the information regarding partitions and the brokers.

My record will be sent in a pipeline, where partitioner and serializer are used.

create property config

Properties props = **new** Properties();

props.put("bootstrap.servers", "localhost:9092");

//Set acknowledgements for producer requests.

props.put("acks", "all");

//If the request fails, the producer can automatically retry,

props.put("retries", 0);

//Specify buffer size in config

props.put("batch.size", 16384);

//Reduce the no of requests less than 0

props.put("linger.ms", 1);

//The buffer.memory controls the total amount of memory available to the producer for buffering.

props.put("buffer.memory", 33554432);

props.put(ProducerConfig.***VALUE\_SERIALIZER\_CLASS\_CONFIG***,StringSerializer.**class**.getName());

props.put(ProducerConfig.***KEY\_SERIALIZER\_CLASS\_CONFIG***,StringSerializer.**class**.getName());

Producer<String, String> producer = **new** KafkaProducer <String, String>(props);

**for**(**int** i = 0; i < 10; i++)

producer.send(new ProducerRecord<String,String> (topicName,Integer.*toString*(i), Integer.*toString*(i)));

System.***out***.println("Message sent successfully");

producer.close();

**What is ProducerRecord ?**

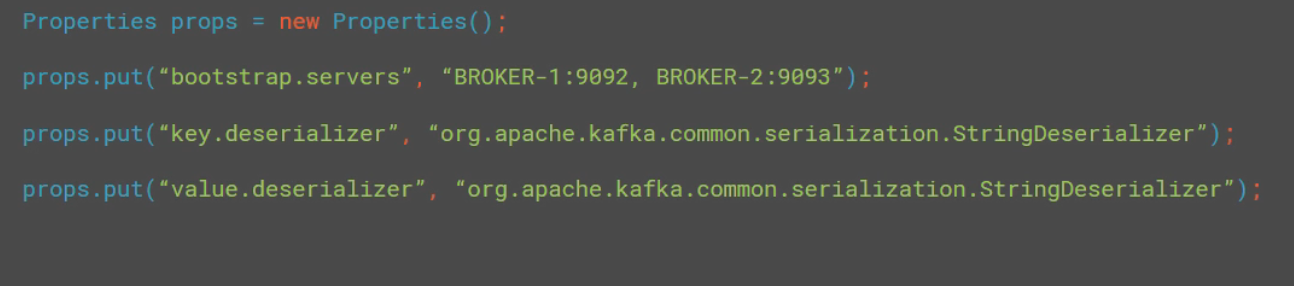
Producer sends messages to Kafka in the form of records. A record is a key-value pair. It contains the topic name and partition number to be sent. If we do not set parttion number, then it will use default partitioner. Kafka broker keeps records inside topic partitions. Records sequence is maintained at the partition level. You can define the logic on which basis partition will be determined.

**How to set bootstrap servers?**

BOOTSTRAP\_SERVERS\_CONFIG or bootstrap.servers property provides the kafka broker’s address. If we have cluster then the server addresses are comma separated like localhost:9091,localhost:9092 . often we provide the zookeepr address here, that is wrong, zookeeper is the controller of all the brokers of kafka server.

**Kafka Consumer**

Some main properties config for kafka consumer client



**Subscribe :**

single consumer can subscribe one or more topics

consumer.subscribe(topicname) or consumer.subscribe(list of topic name)

**Is subscribe method increamental ?**

No,

consumer.subscribe(Arrays.asList(“topic1”))

consumer.subscribe(Arrays.asList(“topic2”))

If we do that consumer will override the topic, means consumer will start subscribing topic2 and automatically unsubscribe topic1

**Unsubscibe :**

two way we can do that consumer.unsubscribe() it will unsubscribe from all the topics. and consumer.subscribe(empty list)

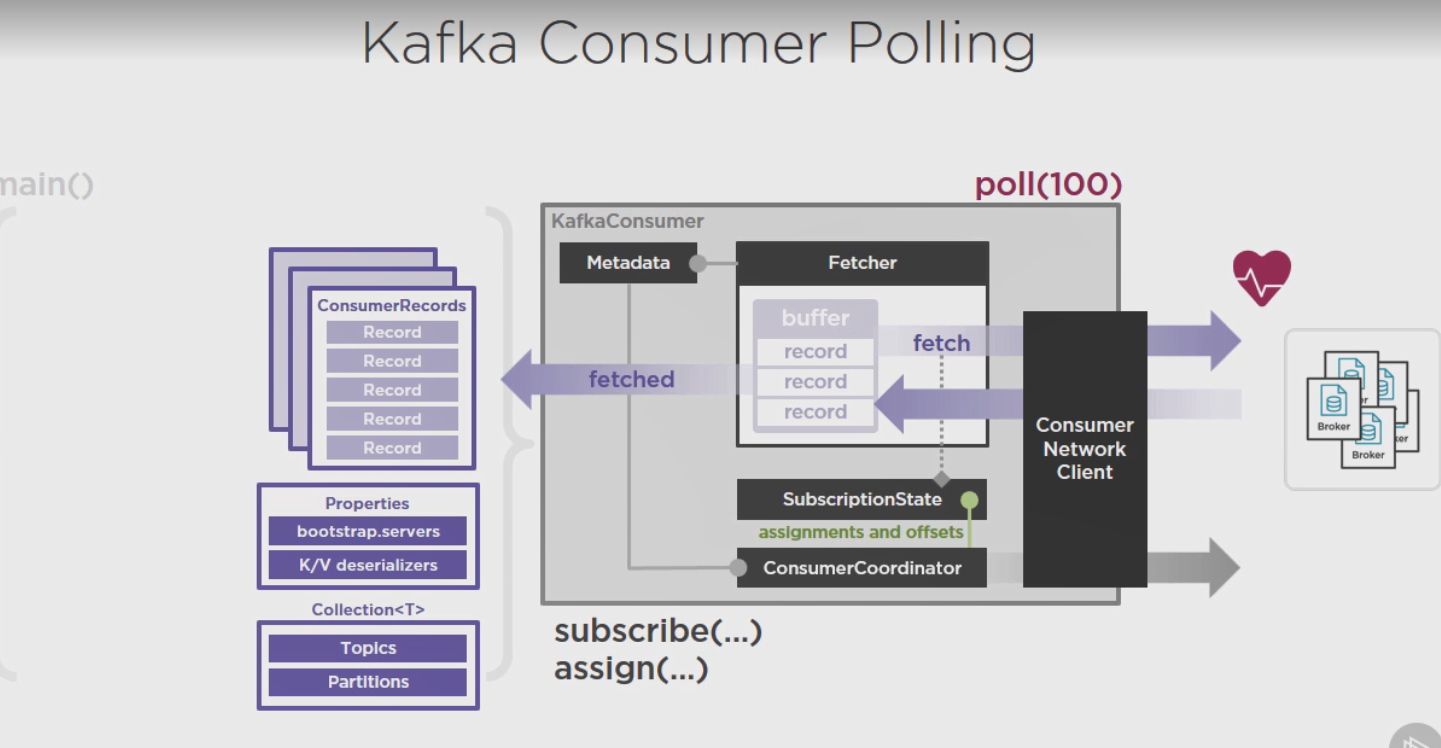
**What is assign() ?**



If consumer know the partion index and want to subscribe specific partitions of several topics then use assign method. Once the assign() method is called, consumer does not care about other partions, like if any new partitions are added later for the topic, consumer does not subscribe them automatically unlike subscribe() method

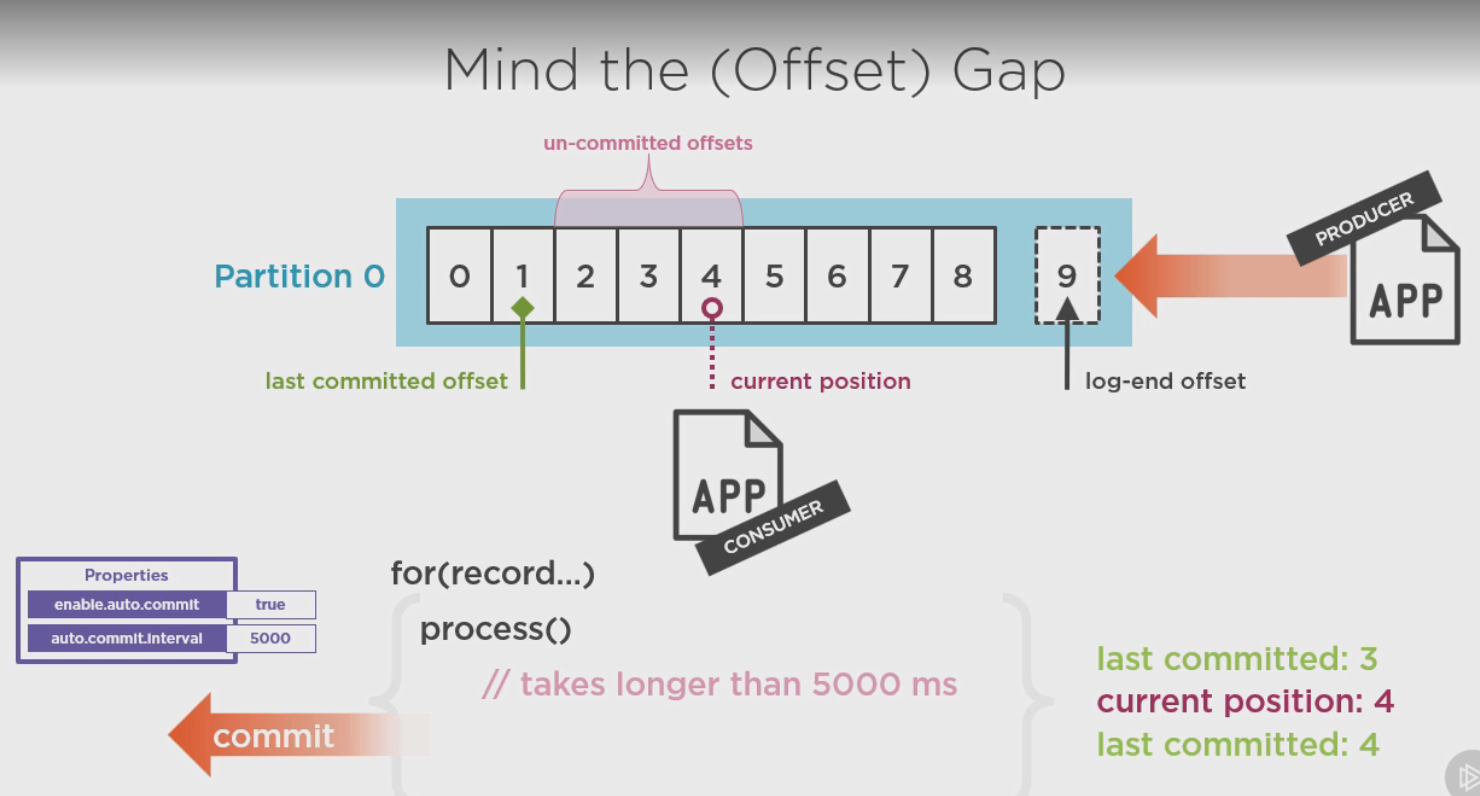
Note : two different consumer cannot subscribe one single parttion

**Simple Kafka consumer**



1. When poll( max\_size ) is called on consumer object it requests kafka bootstrap server or cluster to provide metadata.
2. Fetcher object is responsible to fetch all the data from kafka cluster with the help of Consumer network client. It continuously send the heartbeat to the Kafka cluster.
3. ConsumerCoordinator’s responsibility is to update metadata everytime when any topic is updated or deleted and update the state in SubscriptionState object.
4. ConsumerCoordinator also update the offset and commit it to kafka cluster.
5. Fetcher fetches the latest information as a list of consumer record categorized by topic and partition.

**Consumer Offset**



There are two kind of offsets available one is un committed and other one is committed . here in this picture we have our last committed offset at 1 and current position is 4. , so 2,3,4 are un committed offset.

There are two properties by which we can manage offsets

enable.auto.commit = true : means when the data of the offset is processed by the consumer it will be auto committed

auto.commit.interval = 5000 means if a message takes more than 5000 miliseconds or 5 second then the offset will be committed, before the processing completed.

If we want to change the offset commit logic we need to change the auto.commit.interval = false and write the logic accordingly.

**Where offsets are stored ?**

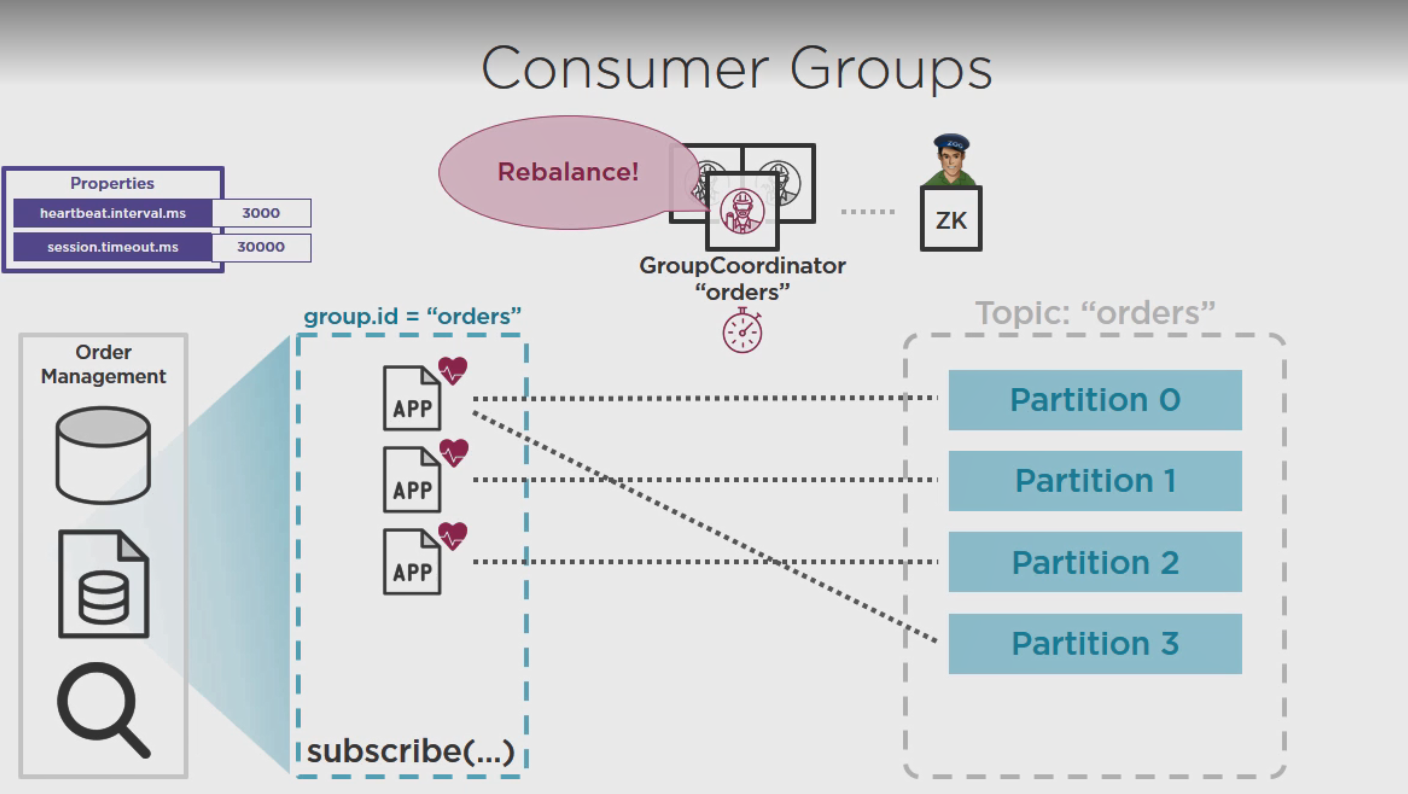
offsets are stored in topic named \_\_consumer\_offsets, default partition is 50 .

the consumercoordinator is responsible to commit these offsets to the kafka cluster

**Is poll() method of consumer is multithreaded ?**

No. This is single threaded

**What is consumer group**



To scale the consumer for large amount of message consumption we can create a cluster of consumer which is called consumer group. which is denoted by this property “group.id” , all consumer group member have same group id. Kafka will choose a group coordinator among them.

**What is consumer group rebalancing ?**

Each consumer member send a heartbeat to the assigned partitions. Group coordinator has the responsibility to monitor the heartbeats. interval of heartbeats denoted by

“heartbeat.intervals.ms” and if any consumer group member does not send a message within a particular time “session.timeout.ms” group coordinator will replace the member with another consumer. this is called rebalancing.

Not only consumer fault , if a new partition is added for the subscribed topic, then also the rebalancing can happen. the newly added partition is assigned to one of the consumer group member.

**Example**

String topicName = "test";

Properties props = **new** Properties();

props.put("bootstrap.servers", "localhost:9092");

props.put("group.id", "test");

props.put("enable.auto.commit", "true");

props.put("auto.commit.interval.ms", "1000");

props.put("session.timeout.ms", "30000");

props.put("value.deserializer",StringDeserializer.**class**.getName());

props.put("key.deserializer",StringDeserializer.**class**.getName());

KafkaConsumer<String, String> consumer = **new** KafkaConsumer<String, String>(props);

//Kafka Consumer subscribes list of topics here.

consumer.subscribe(Arrays.*asList*(topicName));

//print the topic name

System.***out***.println("Subscribed to topic " + topicName);

**int** i = 0;

**while** (**true**) {

ConsumerRecords<String, String> records = consumer.poll(100);

**if**(records.count() > 0) {

**for** (ConsumerRecord<String, String> record : records) {

// print the offset,key and value for the consumer records.

System.***out***.printf("partition = %s, offset = %d, key = %s, value = %s\n",

record.partition(),record.offset(), record.key(), record.value());

}

System.***out***.println("record count "+records.count());

**break**;

}

}