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

**Apache Kafka® is *a distributed streaming platform*. What exactly does that mean?**

* AK is distributer
* Publisher and subscriber messaging system
* Can handle high volume of data
* It has high reliability
* Can be scaled easily
* It is fault tolerance because data is persisted in disk before sending to consumer prevent data lose
* High throughput
* It can perform 2million right per sec

Uses case:

* Log aggregator
* Streaming real time data
* Even streaming in microservice
* Data pipeline

A streaming platform has three key capabilities:

* Publish and subscribe to streams of records, similar to a message queue or enterprise messaging system.
* Store streams of records in a fault-tolerant durable way.
* Process streams of records as they occur.

Kafka is generally used for two broad classes of applications:

* Building real-time streaming data pipelines that reliably get data between systems or applications
* Building real-time streaming applications that transform or react to the streams of data

To understand how Kafka does these things, let's dive in and explore Kafka's capabilities from the bottom up.

First a few concepts:

* Kafka is run as a cluster on one or more servers that can span multiple datacenters.
* The Kafka cluster stores streams of *records* in categories called *topics*.
* Each record consists of a key, a value, and a timestamp.



Apache kafka is messaging services, Apache Kafak is distributor publisher subscriber messaging system which can handle high volume of data, High reliability can be scaled easily.

Its fault tolerant because message persistent in disk before sending to consumer.

Usesage:

Log aggregator

Producer: An Application that sends messages to kafka.

Messages/data stream/record: is simple array of bytes to kafka.

Consumer: Its application that reads data from kafka.

Broker: is kafka server.

Kafka cluster: Kafka Cluster is cluster n number of kafka services running in it. This are indivisible server, This indivisible server is called broker.

If there are more than one broker is called kafka cluster, We can add new broker in kafka cluster easily.

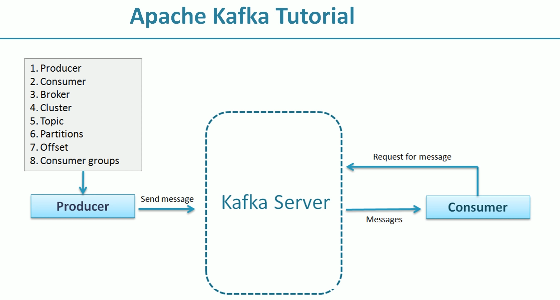
Zookeeper: is used for services registry, That registry all this kafka service, broker and it play part of routing the request from productor and consumer to brokers, all the data actually stored in zookeeper is replicated in broker. Hence data not lost when kafka is down. Zero down time.

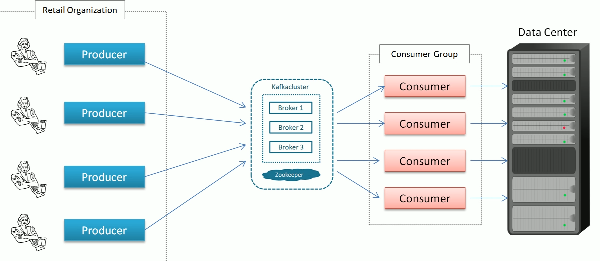
Topic: Topics are message channels and unique name for the kafka stream.

partition: Data stream can be partitioned in to n number of memory partitions, this partitions are linked topics, number of partitions mush be specified during creation of topic.

Offset: is sequence number given to record as they arrive to partition. Offset are immutable, once assigned cant be changed. Records is ordered.

Consumer group: Group of consumer acting of single logical unit. gives opportunity group all the consumer together for better handling of throughput coming from producers.





Installing Kafka

<https://dzone.com/articles/running-apache-kafka-on-windows-os>

1. ZooKeeper: start the Zookeeper server first.

Zkserver.cmd

**Running Kafka Server**

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

### Creating topics

kafka-topics.bat --create --zookeeper localhost:2181 --replication-factor 3 --partitions 3 --topic kafka\_consumer

kafka-topics.bat --create --zookeeper localhost:2181 --replication-factor 3 --partitions 3 --topic kafka\_test

### Creating a Producer and Consumer to Test Server

kafka-console-producer.bat --broker-list localhost:9092 --topic kafka\_consumer

kafka-console-producer.bat --broker-list localhost:9092 --topic kafka\_test

kafka-console-producer.bat --broker-list localhost:9092 --topic kafka\_example

kafka-console-consumer.bat --zookeeper localhost:2181 --topic kafka\_consumer

kafka-console-consumer.bat --zookeeper localhost:2181 --topic kafka\_test

kafka-console-consumer.bat --zookeeper localhost:2181 --topic kafka\_example

## Some Other Useful Commands

1. List Topics: kafka-topics.bat --list --zookeeper localhost:2181
2. Describe Topic: kafka-topics.bat --describe --zookeeper localhost:2181 --topic [Topic Name]
3. Read messages from beginning: kafka-console-consumer.bat --zookeeper localhost:2181 --topic [Topic Name] --from-beginning
4. Delete Topic: kafka-run-class.bat kafka.admin.TopicCommand --delete --topic [topic\_to\_delete] --zookeeper localhost:2181

<https://www.oreilly.com/library/view/kafka-the-definitive/9781491936153/ch04.html>

# Fault Tolerance

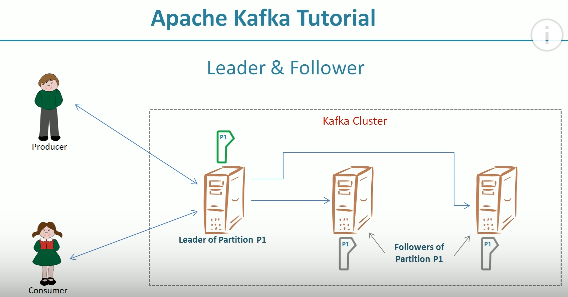
Enabling a system to continues operating in the event of failure of some of component.

While creating topic we specify replication-factor which will create multiple copy of partition.

Kafka has leader and follower node

Leader: it will interact to producer/consumer and partition is responsible for read and write in given partition, each partition has one node acting as leader

follower : is node which follows leader node instruction, If leader fails follower will become leader and pull data from partition.

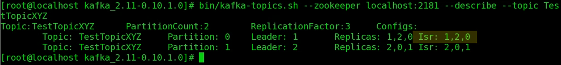


To start multiple broker on the same system we need to change following in configuration file.

Broker.id = 0 (default)

Listeners= PLAINTEXT://:9092

Log.dir= /temp/logs



# Broker Configurations

Server.properties

# Switch to enable topic deletion or not, default value is false

#delete.topic.enable=true

# The id of the broker. This must be set to a unique integer for each broker.

broker.id=0

The address the socket server listens on. It will get the value returned from

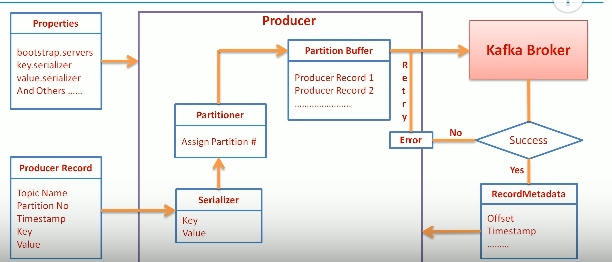
listeners=PLAINTEXT://:9092

# A comma seperated list of directories under which to store log files

log.dirs=/tmp/kafka-logs

# Producer API





# Callback & Acknowledgment

Types of sending record to kafka Broker.

* 1. Send and forget.
  2. Synchronous Send- send record and wait for the recordmetadata success/error response.



Send will return java Future object.

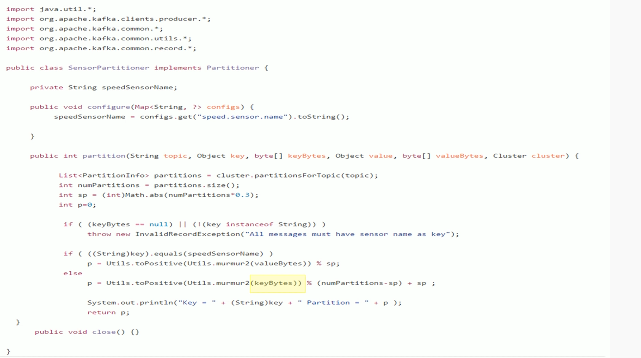
* 1. Asynchronous Send(better throughput): Send the record and not wait for response.

Config used to control number record that can be sent without waiting for response.

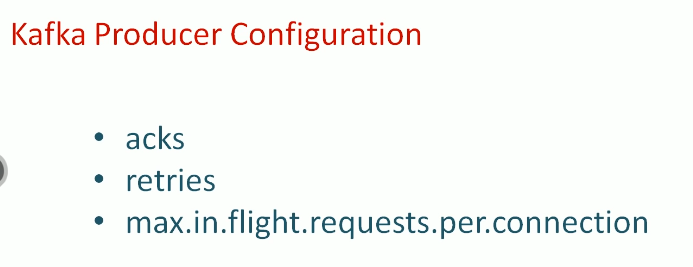
Max.in.flight.request.per.connection



# Custom Partitioner



# Custom Serializer

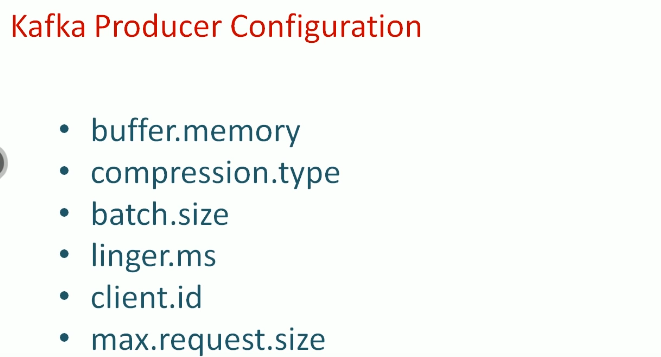


acks=0 – No Acknowledgement, Possible loose of message, High throughput, No Retries.

acks=1 – Less throughput, wait for acknowledgement, changes of data loose when leader crashes.

acks=all(best practices) – No changes of data loose, acknowledgement is sent after replication in followers.

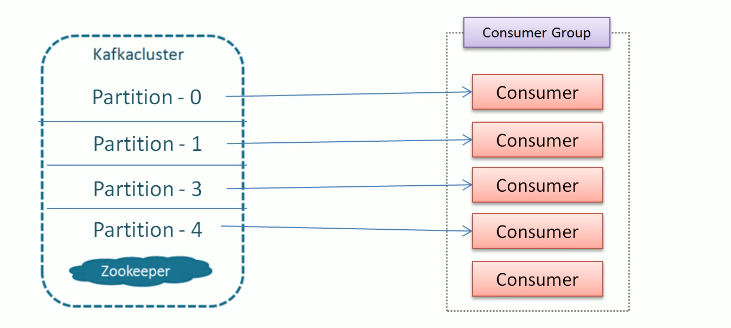
Max.in.flight.requests.per.connection – is used when we are sending record asynchronized, by setting for large value we can increase throughput and memory must be high.



# Consumer Groups

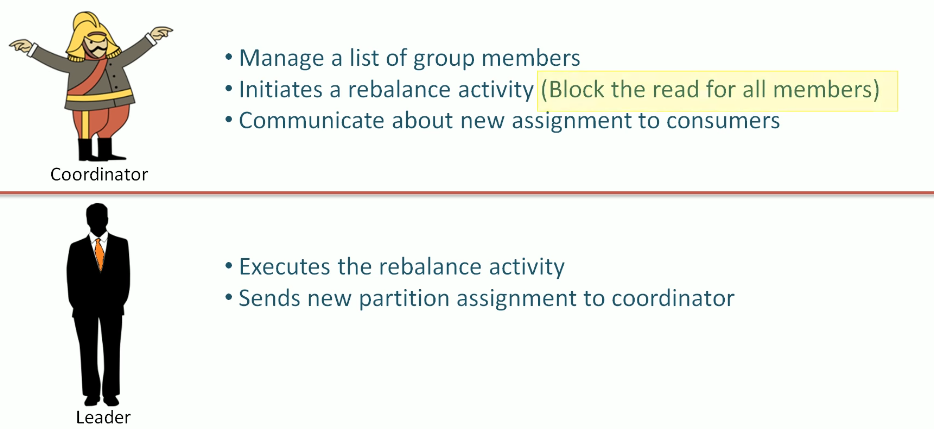
There is no problem if we add new producer at Producer side. If we add consumer then we need go lot of thing at consumer side.

Consumer will not share the partition, only one consumer owns the partition at any point of time .



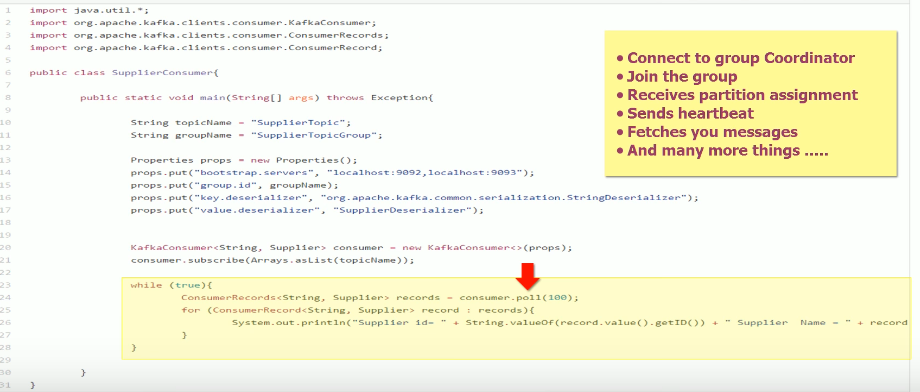
How the consumer managed from consumer group entry/exit?

Group Coordinator will manage. One of the kafka broker is elected as group coordinator





# CreatingConsumer



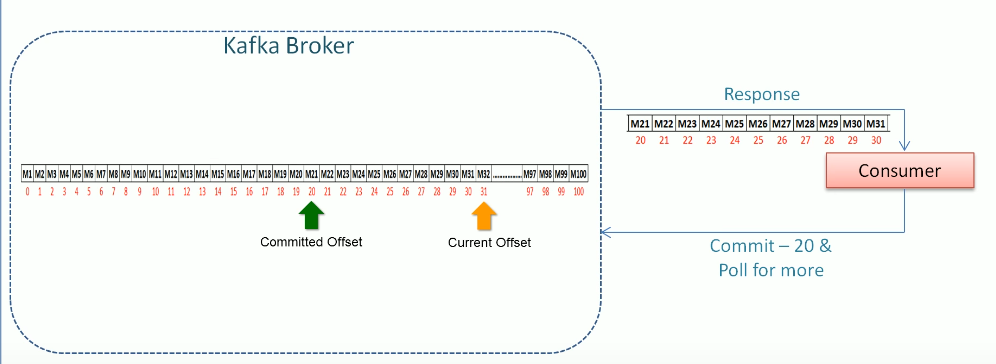
# Offset Management

Kafka maintain two offset

* Current offset
* Committed offset

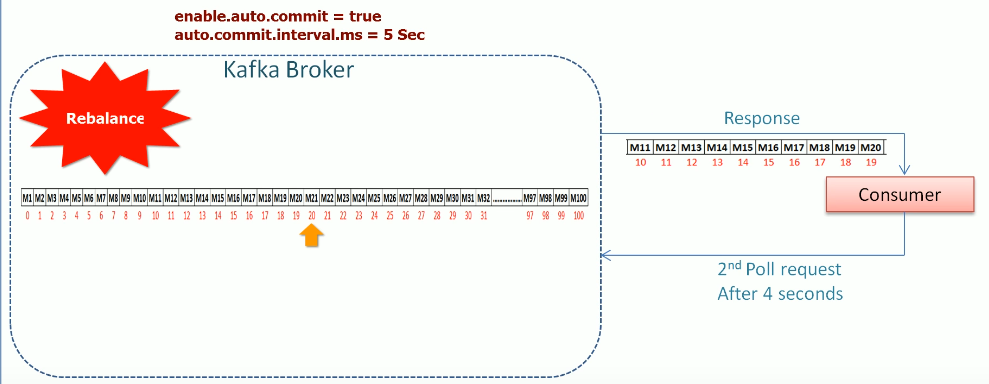
Current offset is integer value which point to record need to be read.

Committed offset is the position that is already processed by consumer.



How to commit?

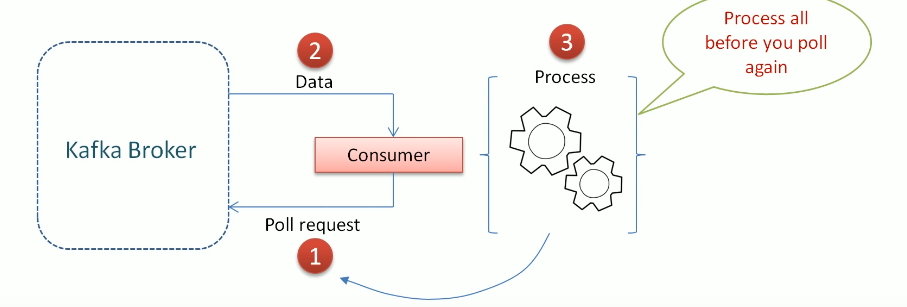
* Auto commit
* Manual commit



Manual Commit has two ways

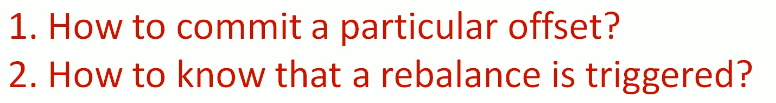
* Commit Sync – it is blocking method and retire if there are recoverable errors
* Commit Async – it will send request continue and it will not retire.

# Rebalance Listener

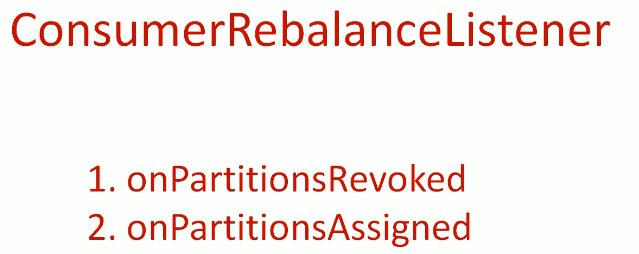


Causes following problem.

* Delay in next poll
* If we don’t poll record for longtime group coordinator will assume consumer is death and rebalance is called.

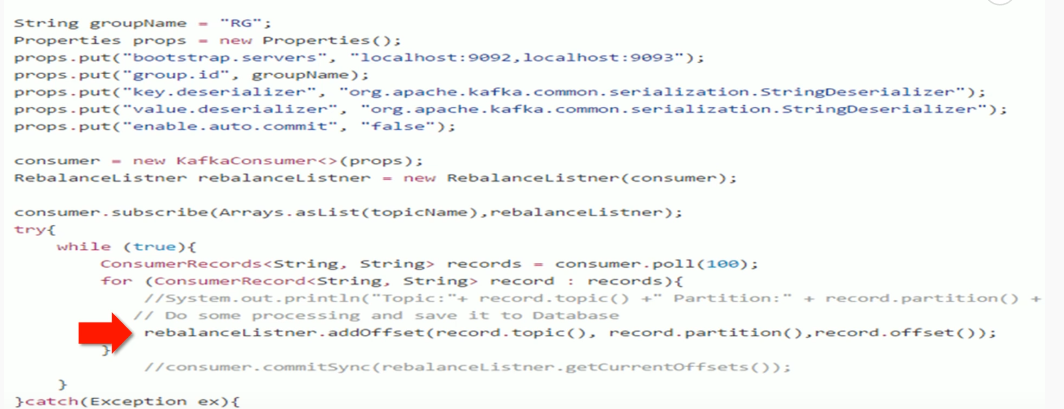


Commit the intermediate offset frequently. Maintain the offset of processed record. Commit when rebalance is triggered.

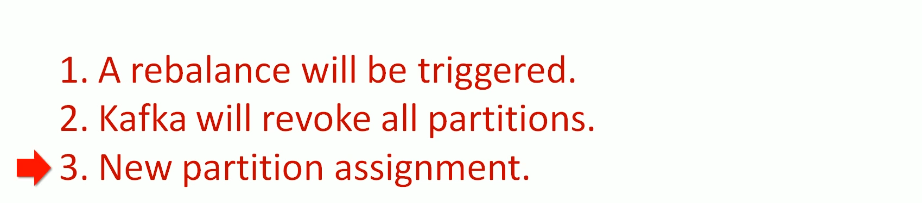


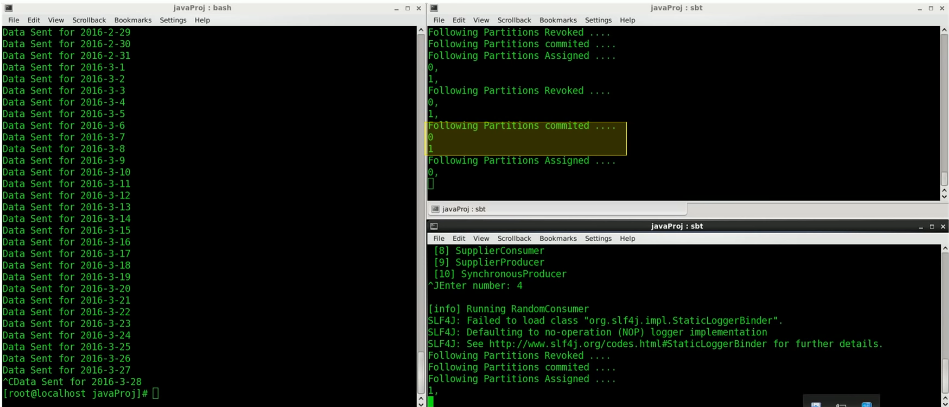


Listener is set at time of consumer subscribe and kafka will call the rebalance when taking away consumer.

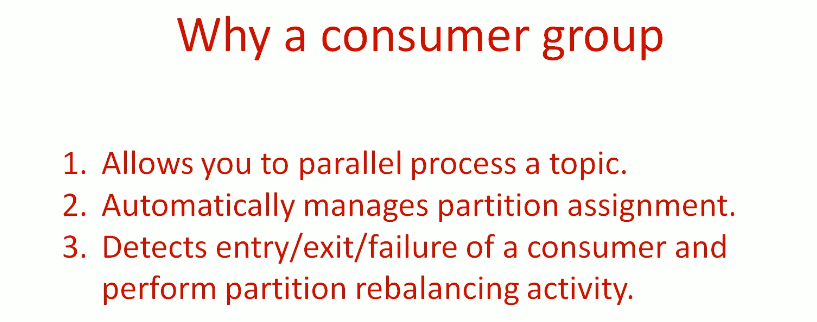


What happened if me consumer is assigned?





# Exactly once processing



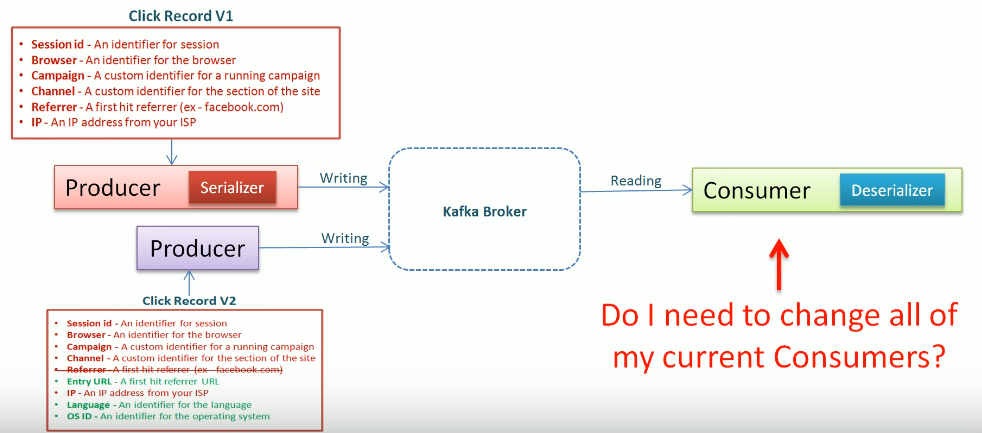
Taking the control of partition assignment instead of kafka do it.

* Partition assignment
* Offset committee

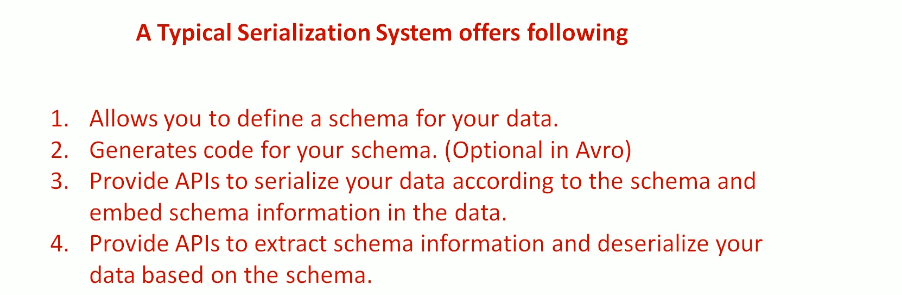
Solve duplicate read of record.

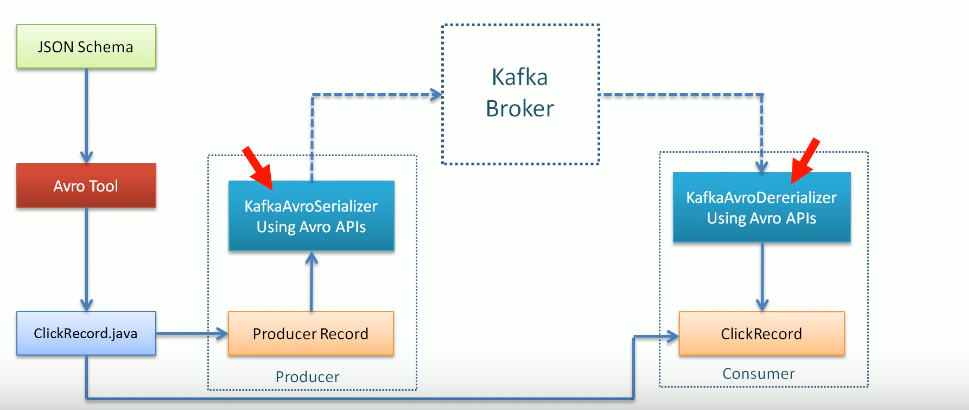
Manually handle offset by storing in DB and use that offset for commit.

# Schema Evolution Part 1



Solution is AVRO tool for serialization and deserialization.





# Scaling Kafka Producer