Table of Contents

Apache Kafka Introduction 2

Import configuration parameter 17

Producer API 20

Apache kafka Producer workflow 24

Callback and Acknowledgment 24

Custom Partition 26

Custom Serializer 33

Producer Config 39

Consumer Group 45

Consumer API 53

Offset Management 57

Rebalance Listener 69

Exactly one processing 77

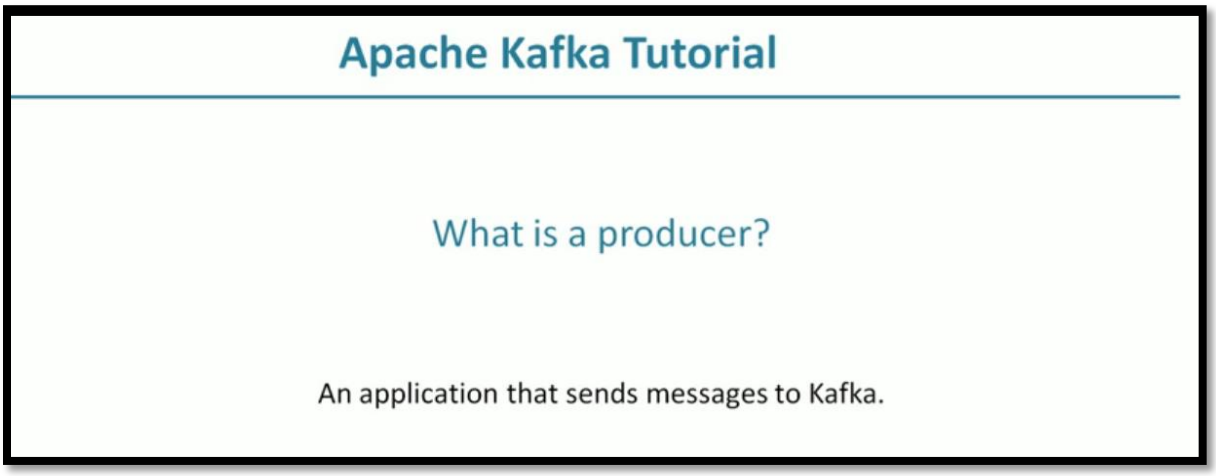
Schema Evolution 86

Confluent Platform 97

- Confluent Schema Registry 98
- Installation Step on Centos7 99

Apache Kafka Introduction

https://www.youtube.com/watch?v=udnX21__SuU&list=PLkz1SCf5iB4enAR00Z46JwY9GGkaS2NON&index=3



Apache Kafka Tutorial

What is a message?

Small to medium sized piece of data.

Apache Kafka Tutorial

What is a Message?

Apache Kafka Tutorial

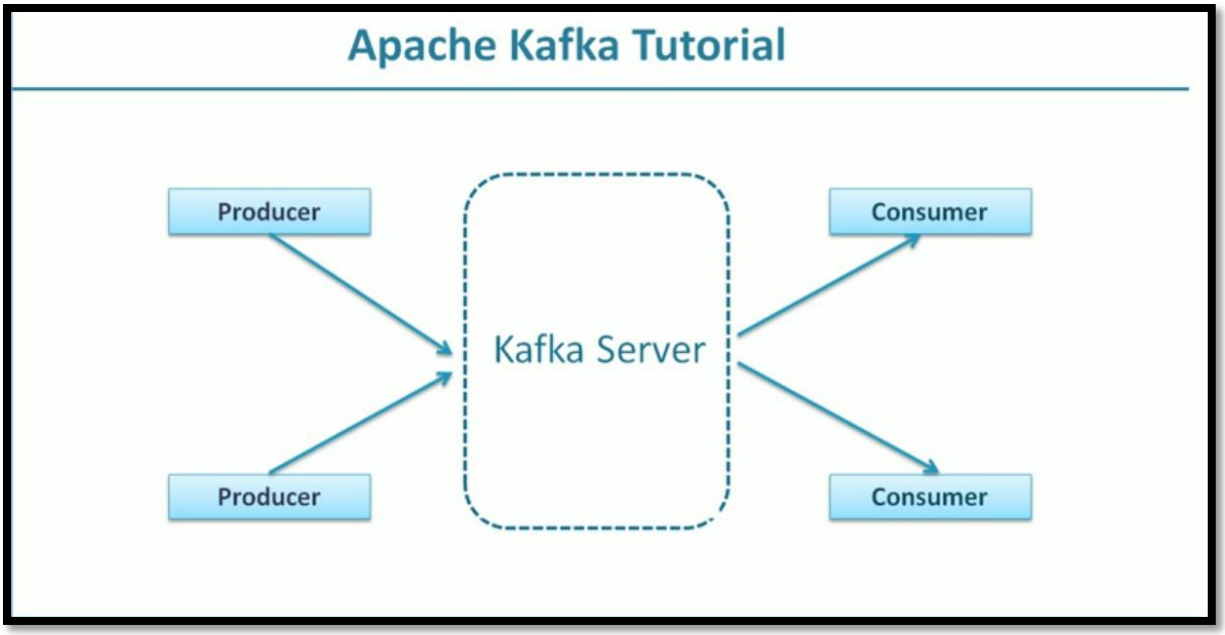
What is a Message?

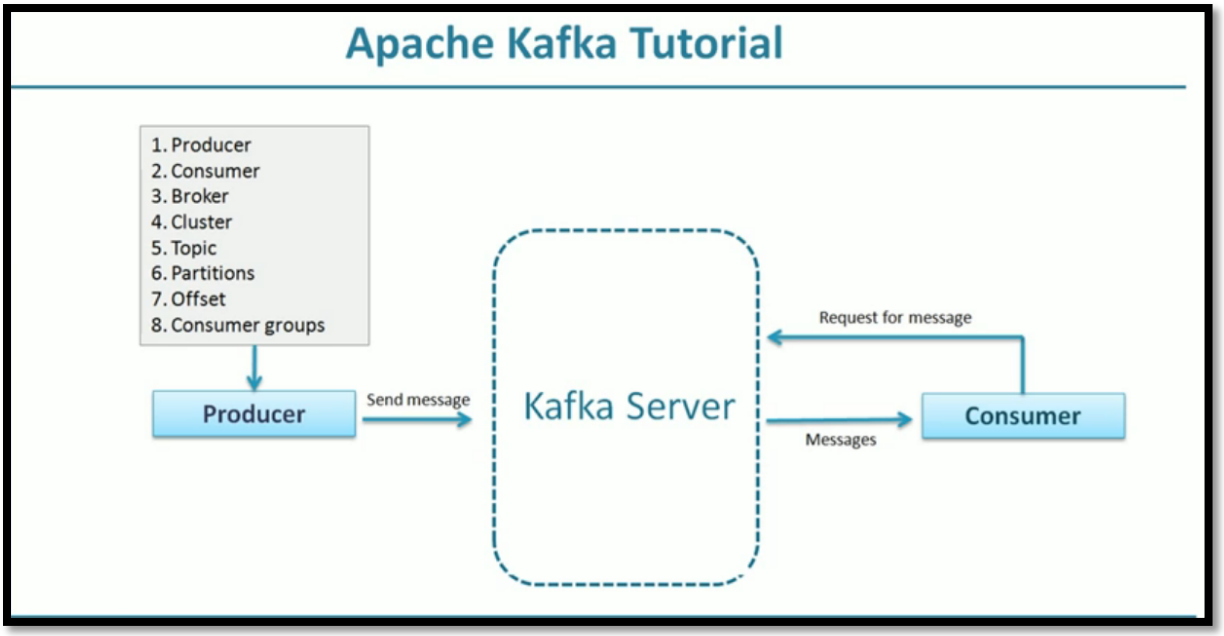
ID	Value	Description
1	Producer	An application that sends data to Kafka
2	Consumer	An application that receives data from Kafka
3	Broker	Kafka Server
4	Cluster	Group of computers
5	Topic	A name for a Kafka stream
6	Partitions	Part of a topic
7	Offset	Unique id for a message within a partition
8	Consumer groups	A group of consumers acting as a single logical unit.

Apache Kafka Tutorial

What is a consumer?

An application that reads data from Kafka


Apache Kafka Tutorial
Producer
Producer
Kafka Server
Consumer
Consumer


Apache Kafka Tutorial
1. Producer
2. Consumer
3. Broker
4. Cluster
5. Topic
6. Partitions
7. Offset
8. Consumer groups
Producer
Send message
Kafka Server
Request for message
Messages
Consumer

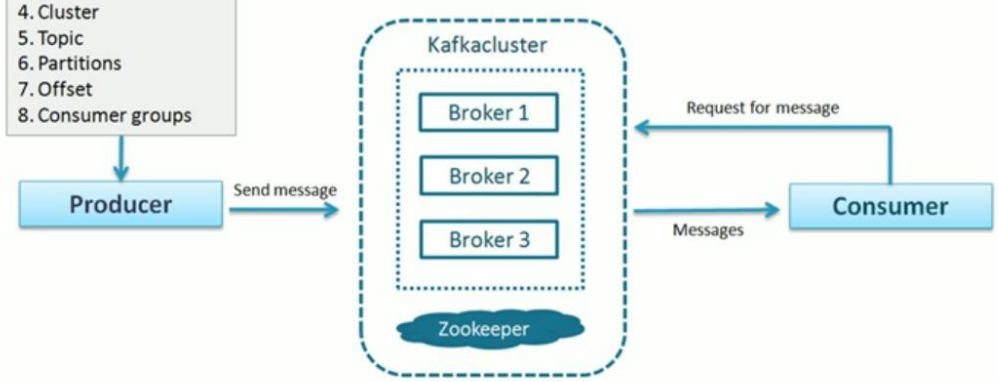
Apache Kafka Tutorial

What is a cluster?

A group of computers sharing workload for a common purpose

Apache Kafka Tutorial

1. Producer
2. Consumer
3. Broker
4. Cluster
5. Topic
6. Partitions
7. Offset
8. Consumer groups

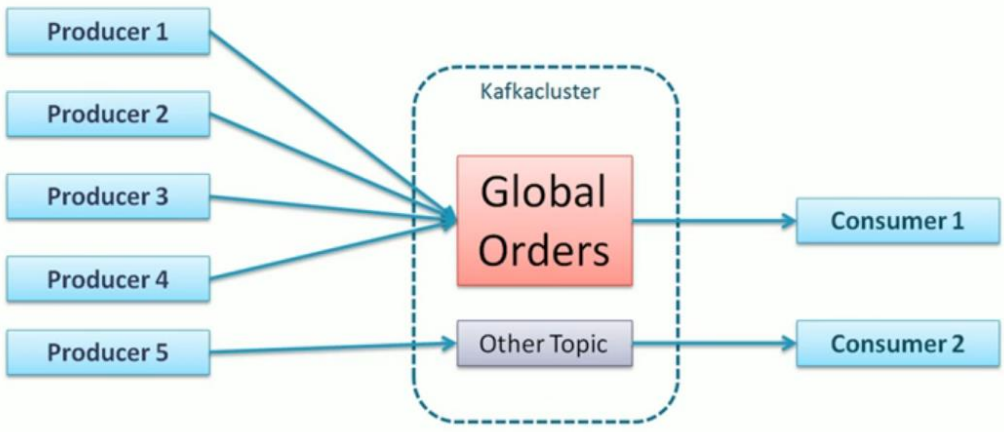


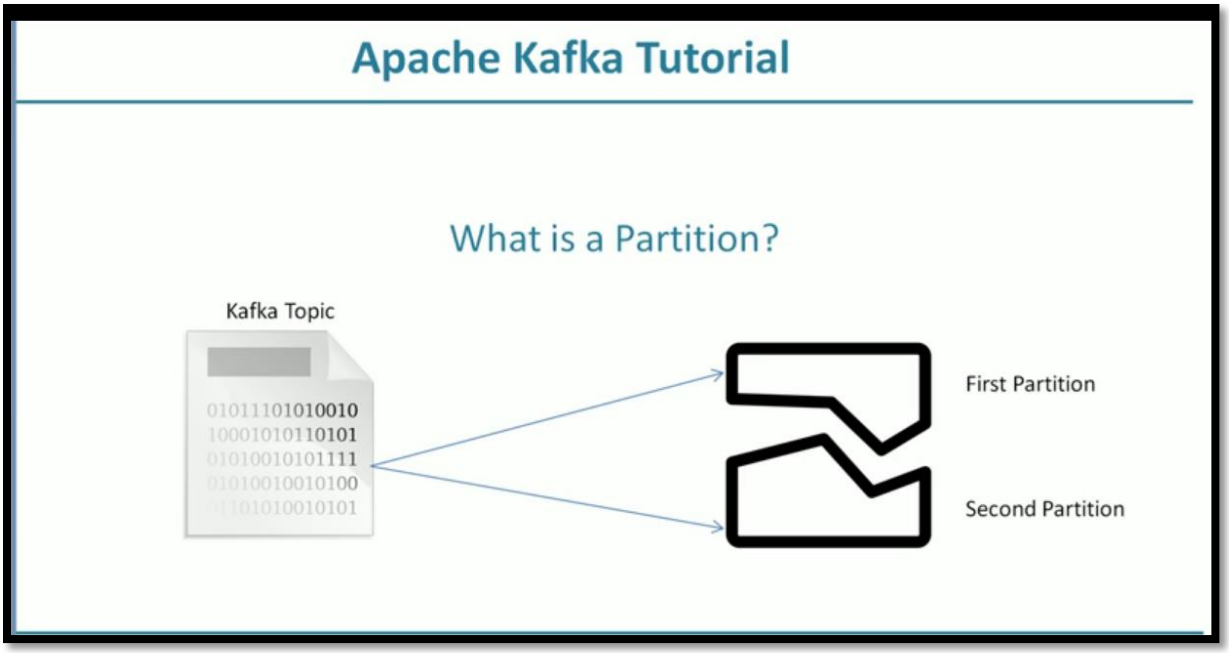
Apache Kafka Tutorial

What is a Topic?

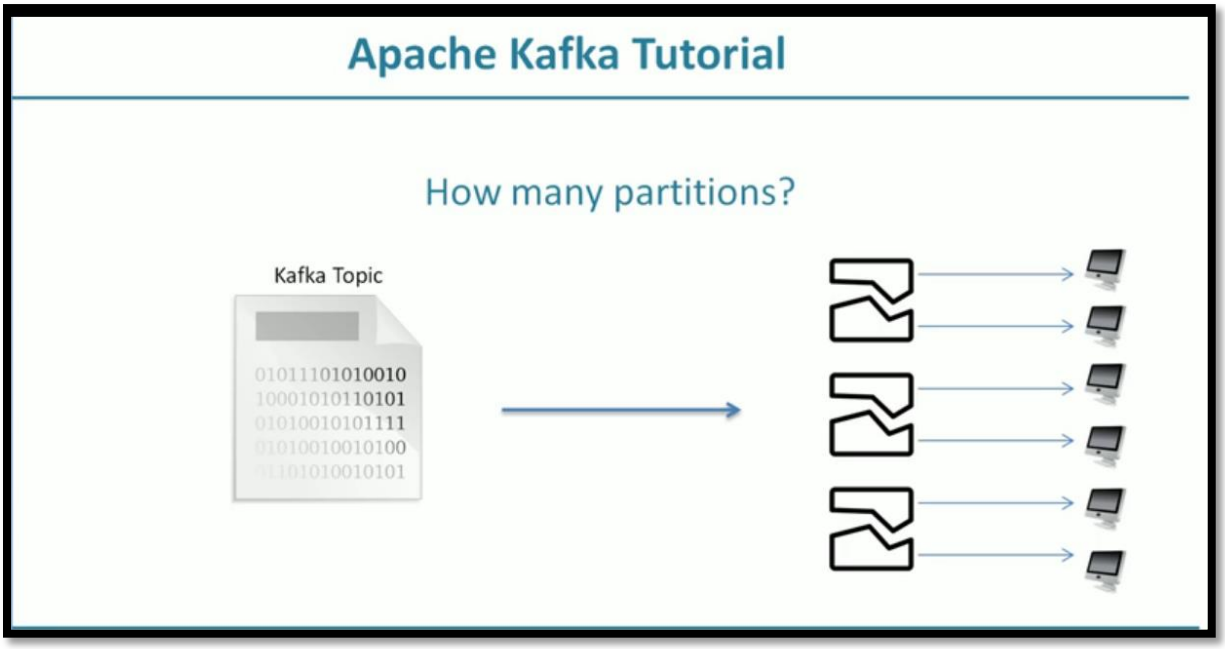
A topic is a unique name for Kafka stream

Apache Kafka Tutorial





If data size is too big one computer can't store it so kafka split the data according to partition given and store it in different machines

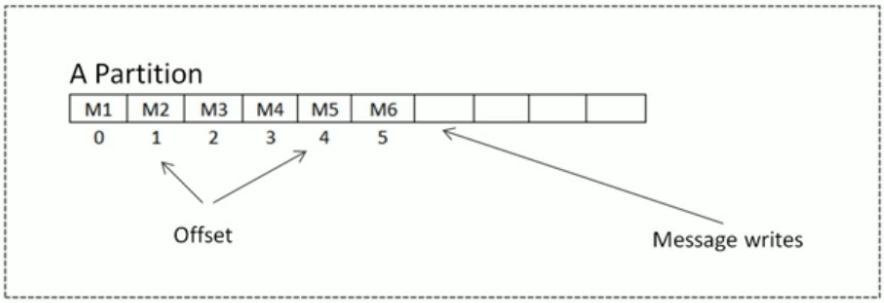


We need to decide it while creating topic

Apache Kafka Tutorial

What is offset?

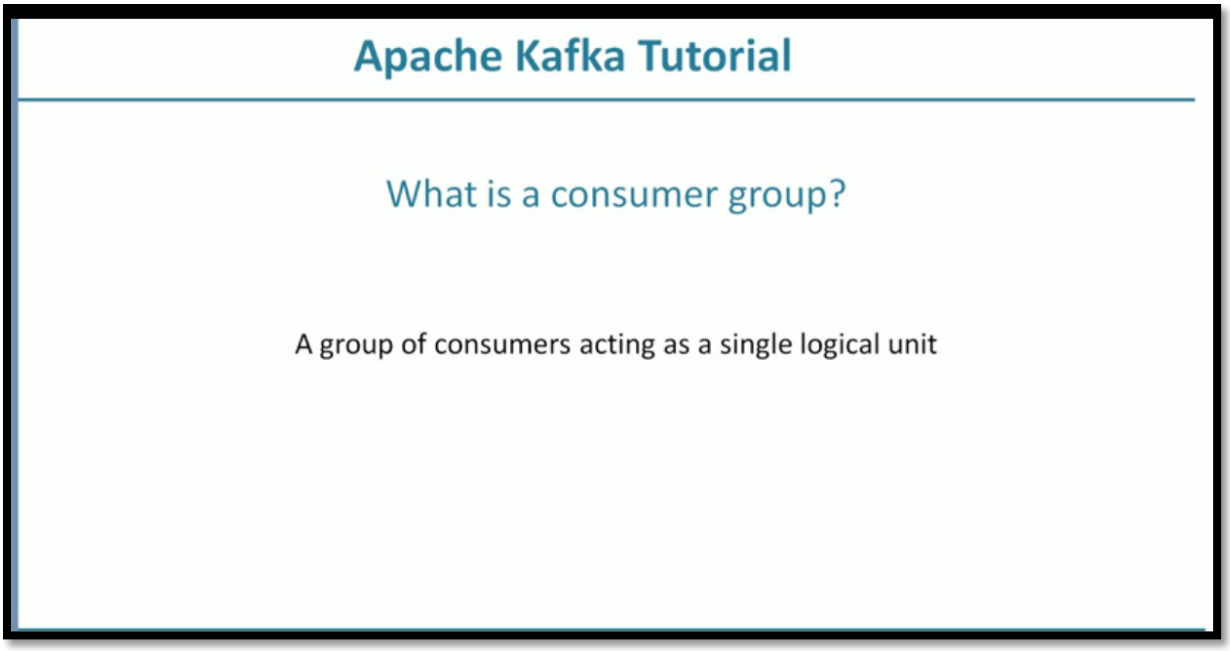
A sequence id given to messages as they arrive in a partition.

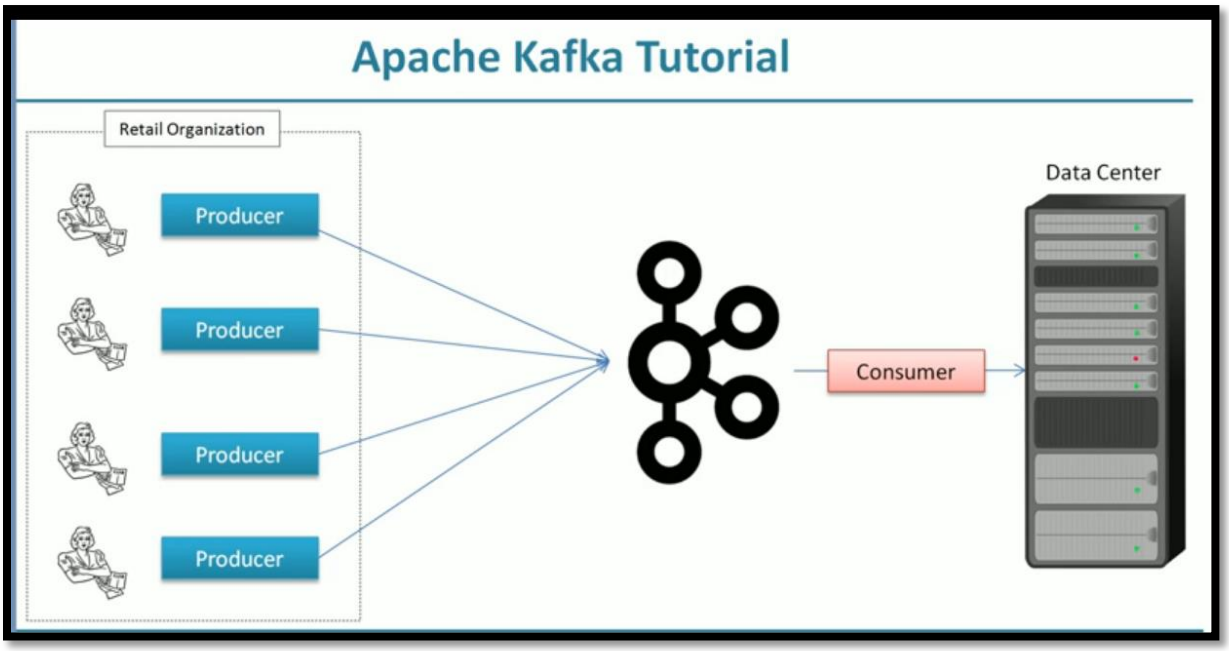


Apache Kafka Tutorial

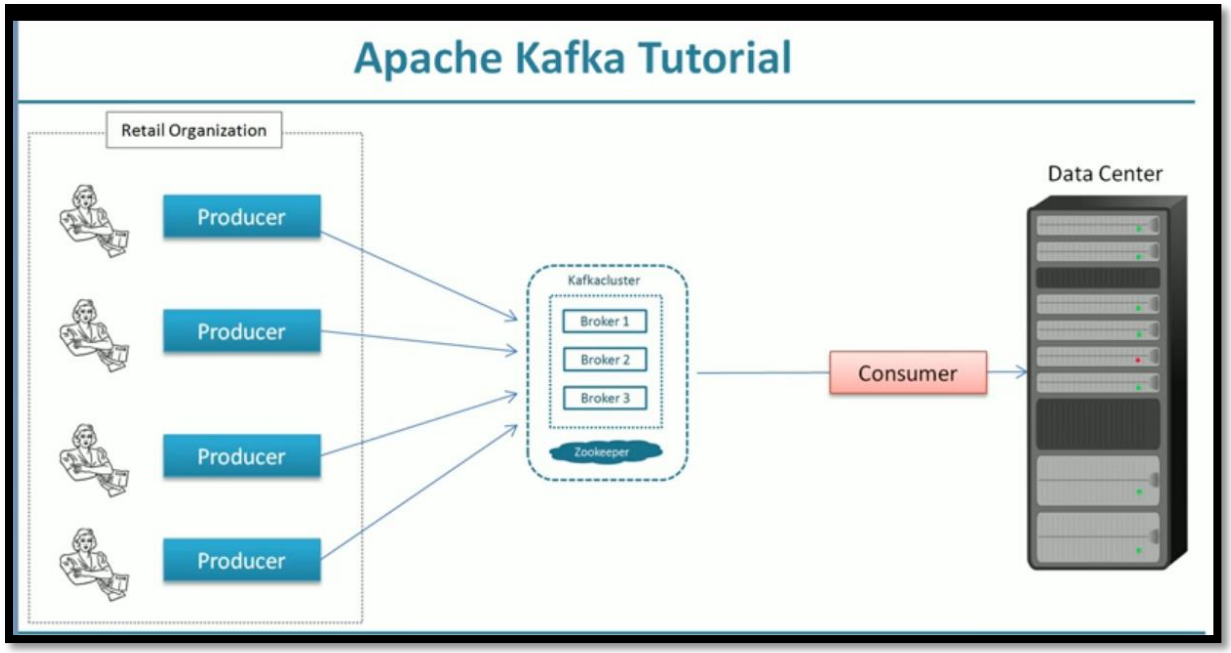
Global unique identifier of a message?

Topic Name → Partition Number → Offset



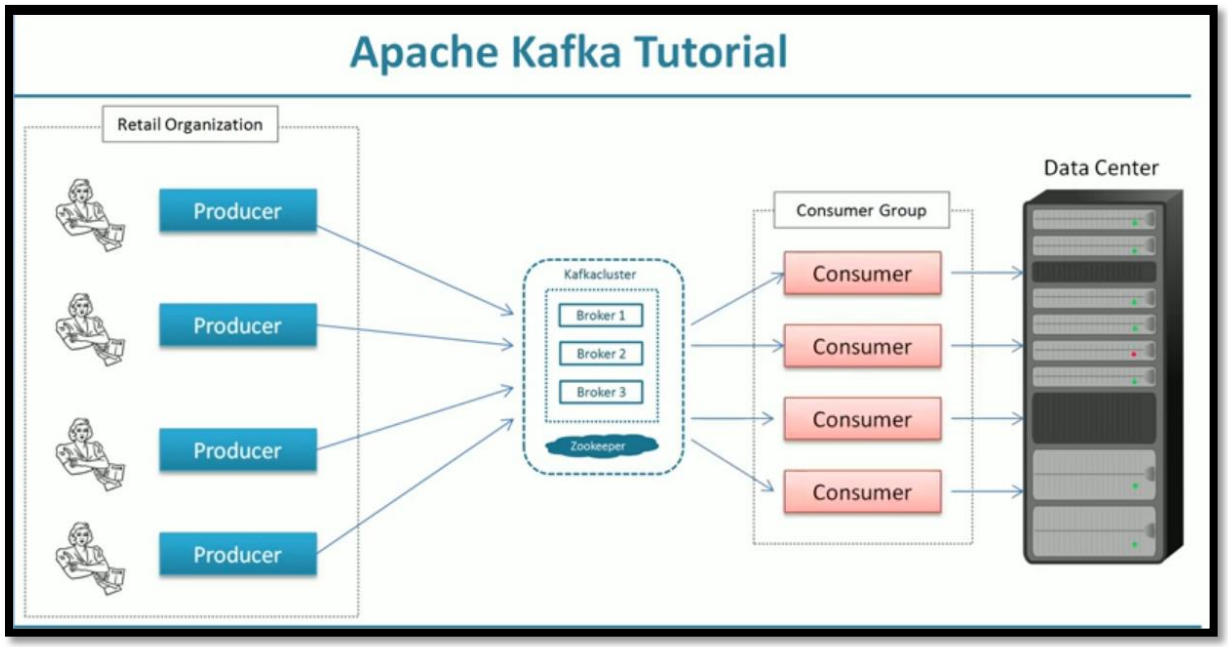


Hundreds of producers pushing data into single topic how we will handle the data volume and velocity



Kafka cluster and partitioned the topic

Now source side we have many producers but at destination side just one consumer to receive data and process data

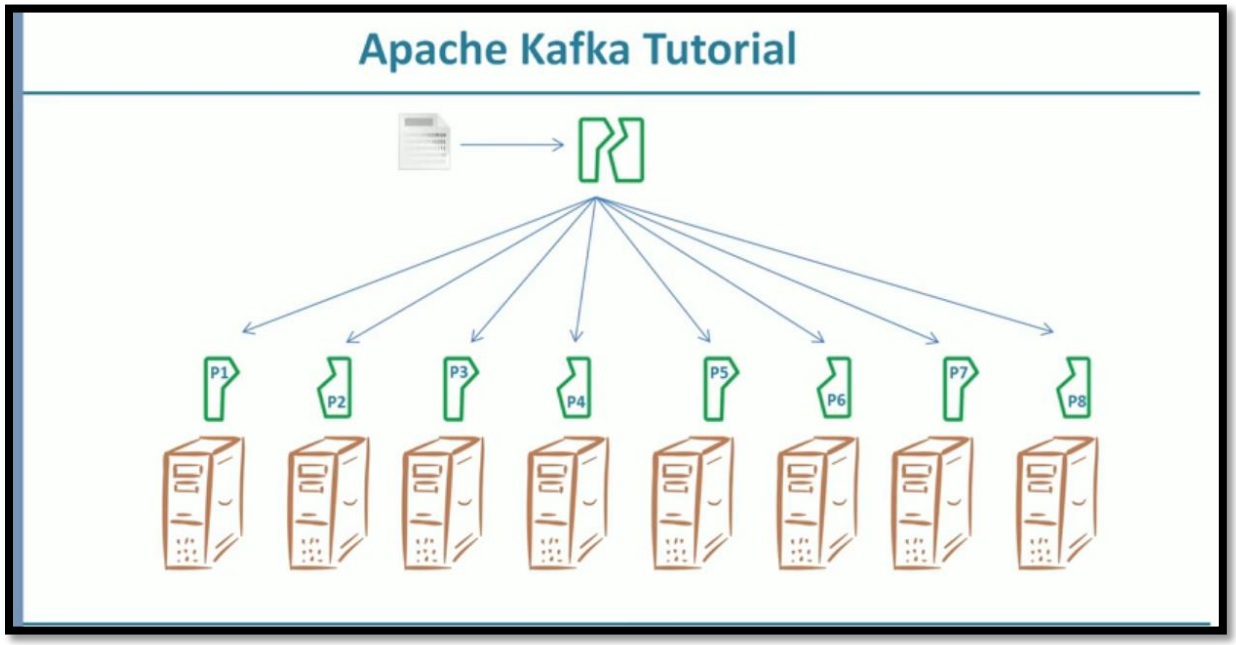


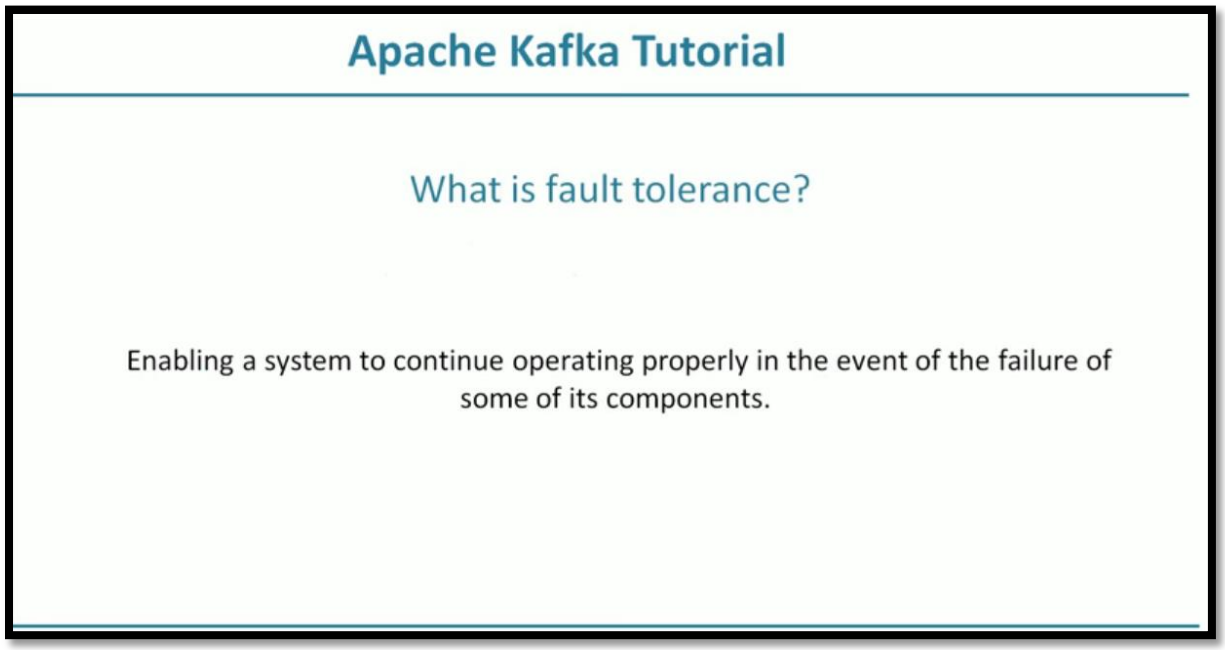
Multiple consumer divide works among them, each consumer can work on some partition

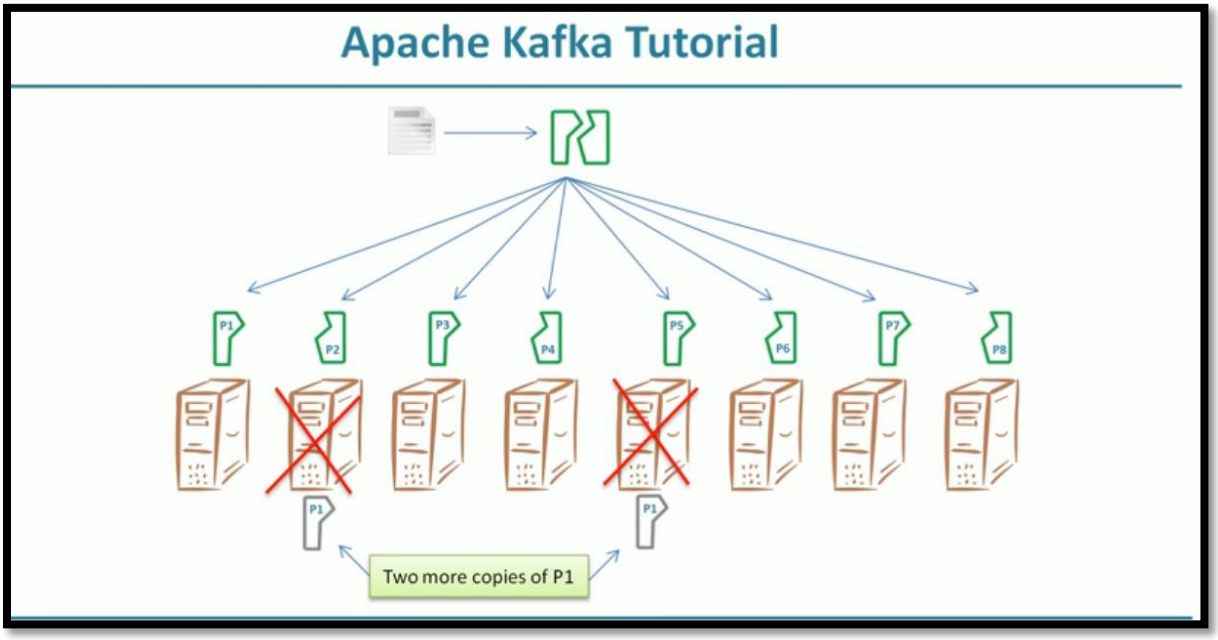
Suppose we have 12 partition and 2 kafka consumer under Consumer group , they will both read 6 partition each means they will read different set of partitions=different set of messages

Max no of consumer in a group = total no of partition you have on the topic

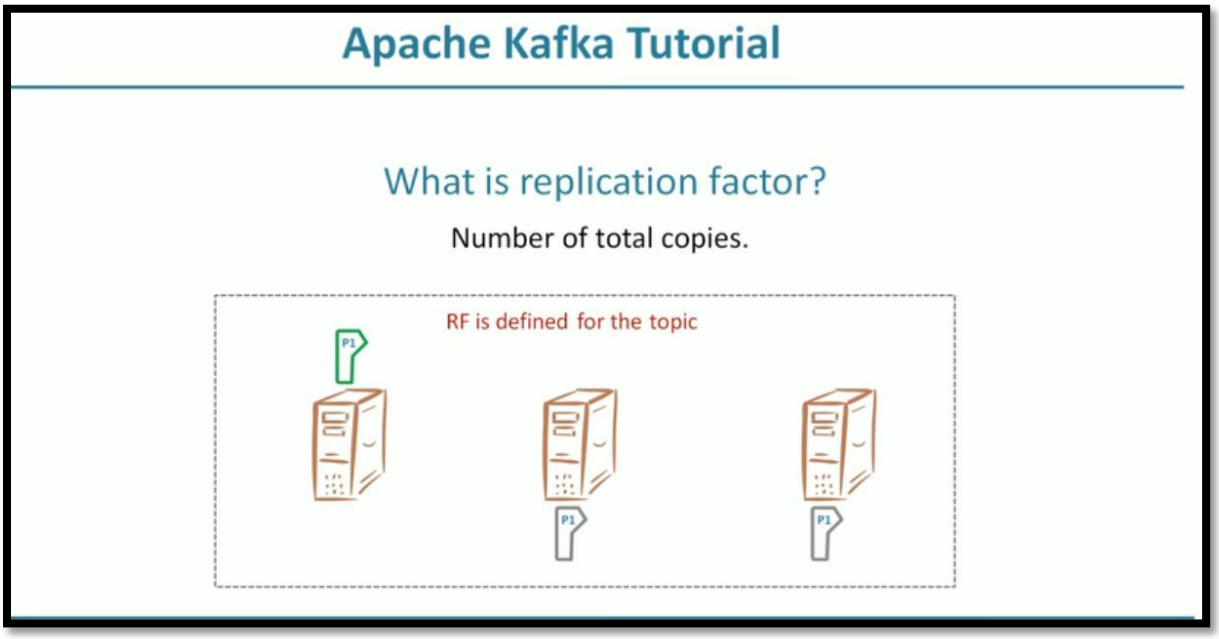
Kafka doesn't allow two consumers to read from the same partition simultaneously to avoid double reading of record



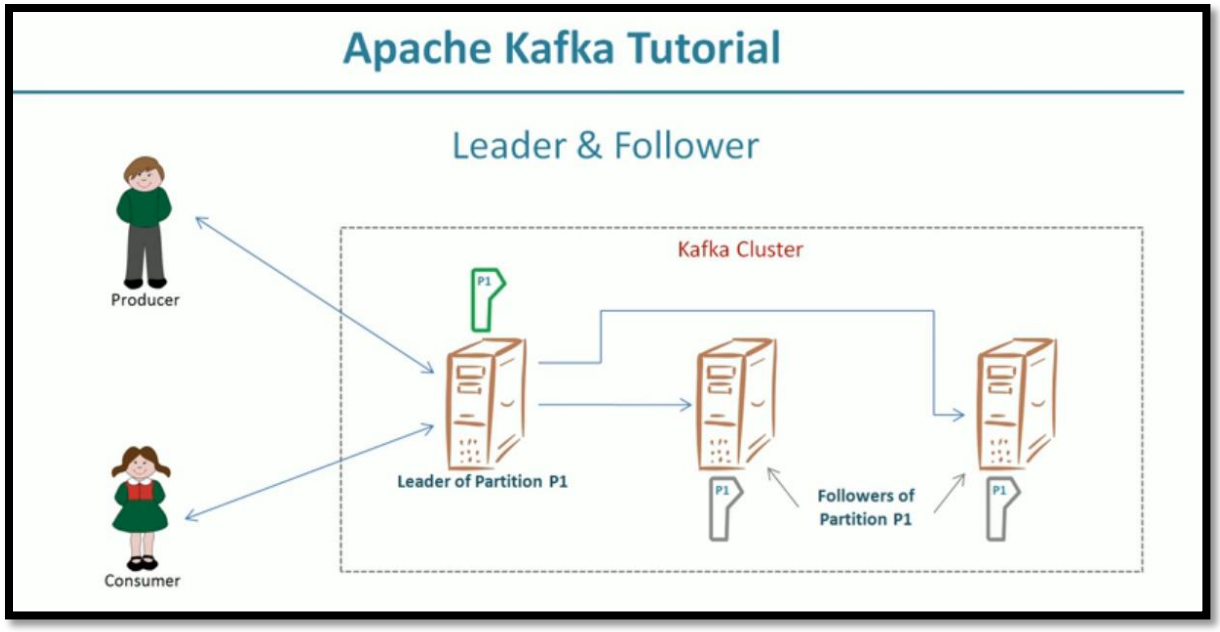




Make multiple copies of data and store them in different system



Replication factor is define for the topics not for partitions



Kafka implement a leader and follower model so for every partition one broker is elected as a leader and leader take care of all client interaction

It is leader's responsibility to receive the message, store it in local disk and send back an acknowledgement to the producer. Similarly when a consumer is willing to read data it sends a request to the leader, it is responsibility to send requested data back to the consumer.

Follower maintains the copies of partition, these followers copy the data from the leader they don't talk to producer or consumer

Start three brokers in single machine just for demo purpose

Apache Kafka Tutorial

First Broker
bin/kafka-server-start.sh config/server.properties

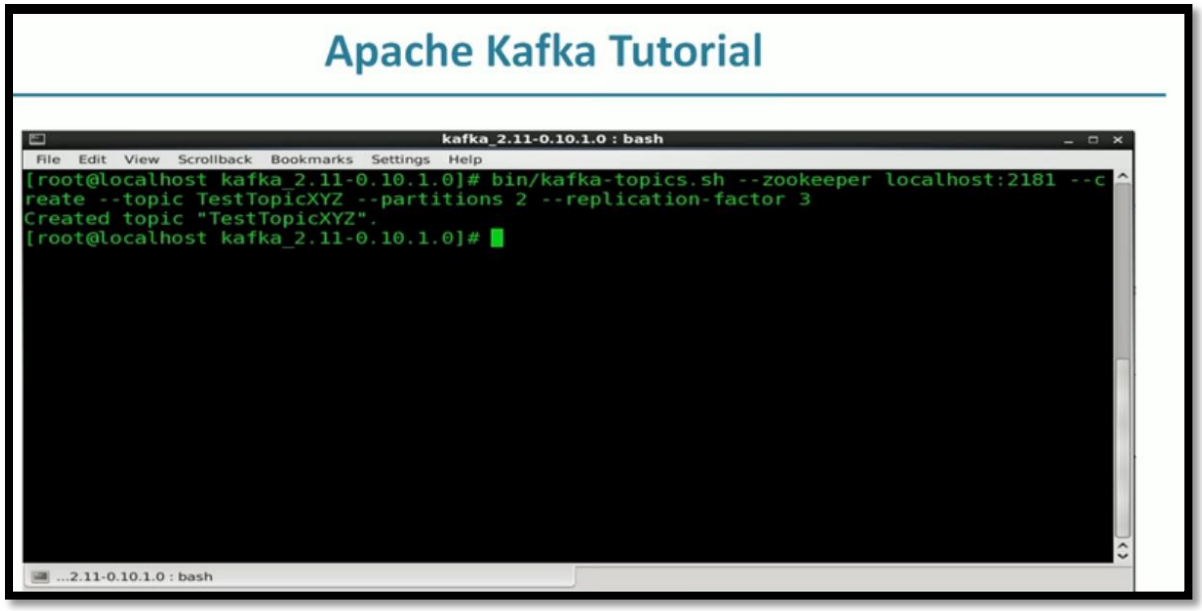
Second Broker
bin/kafka-server-start.sh config/server-1.properties

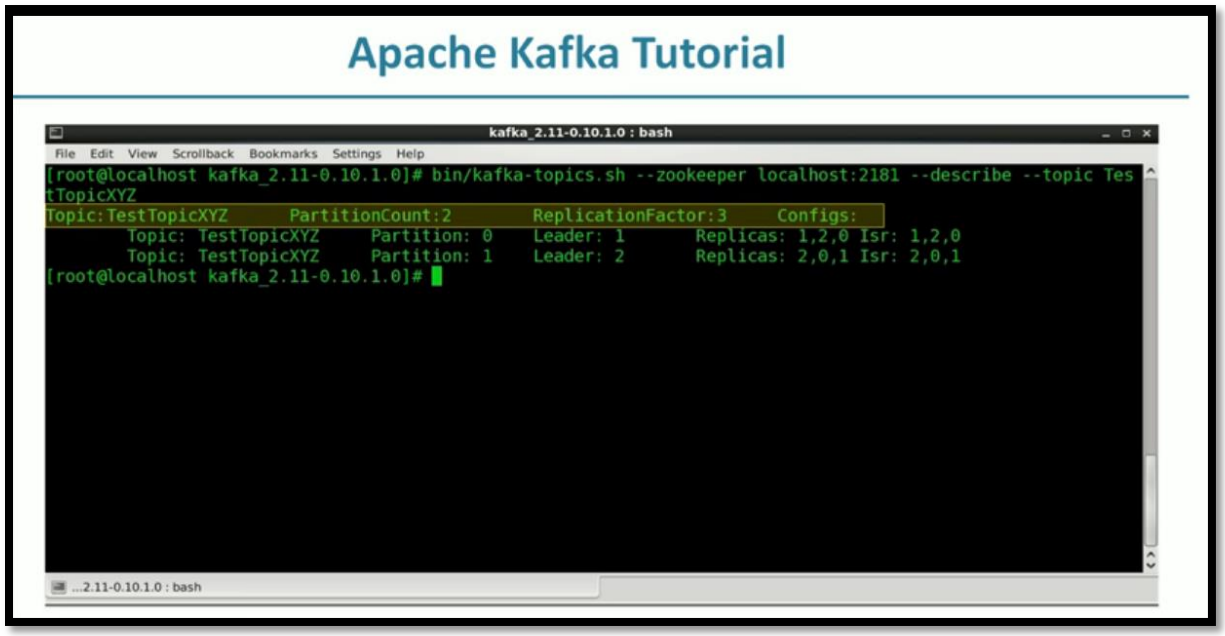
Third Broker
bin/kafka-server-start.sh config/server-2.properties

Apache Kafka Tutorial

1. broker.id
2. Port
3. log.dirs

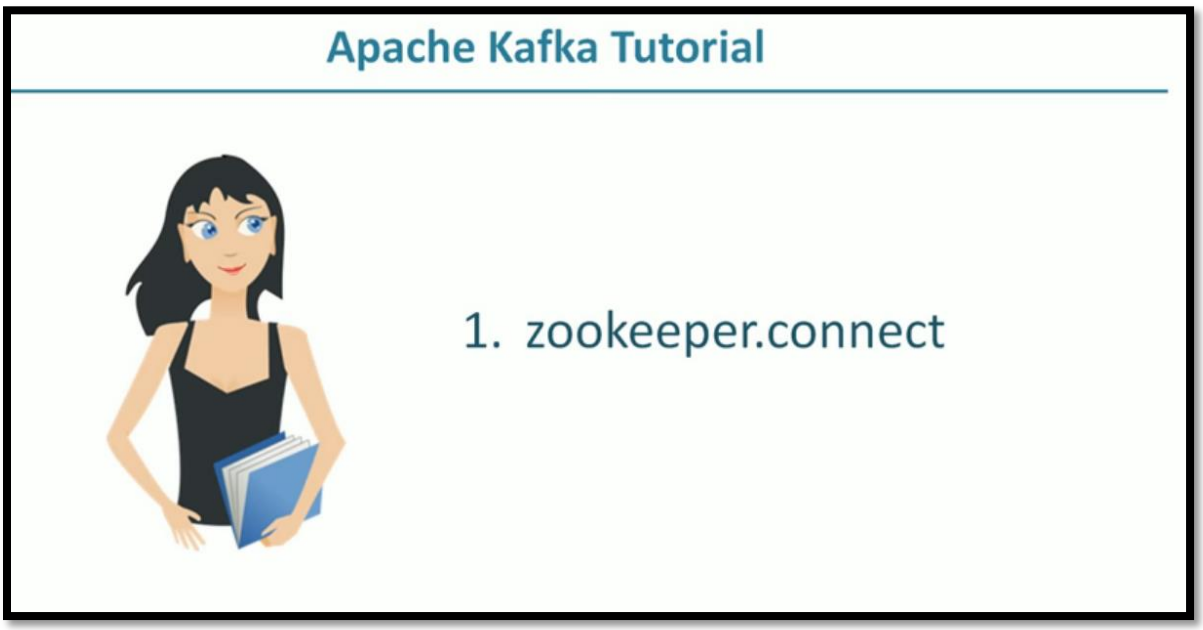
Change these three properties in each server.properties file



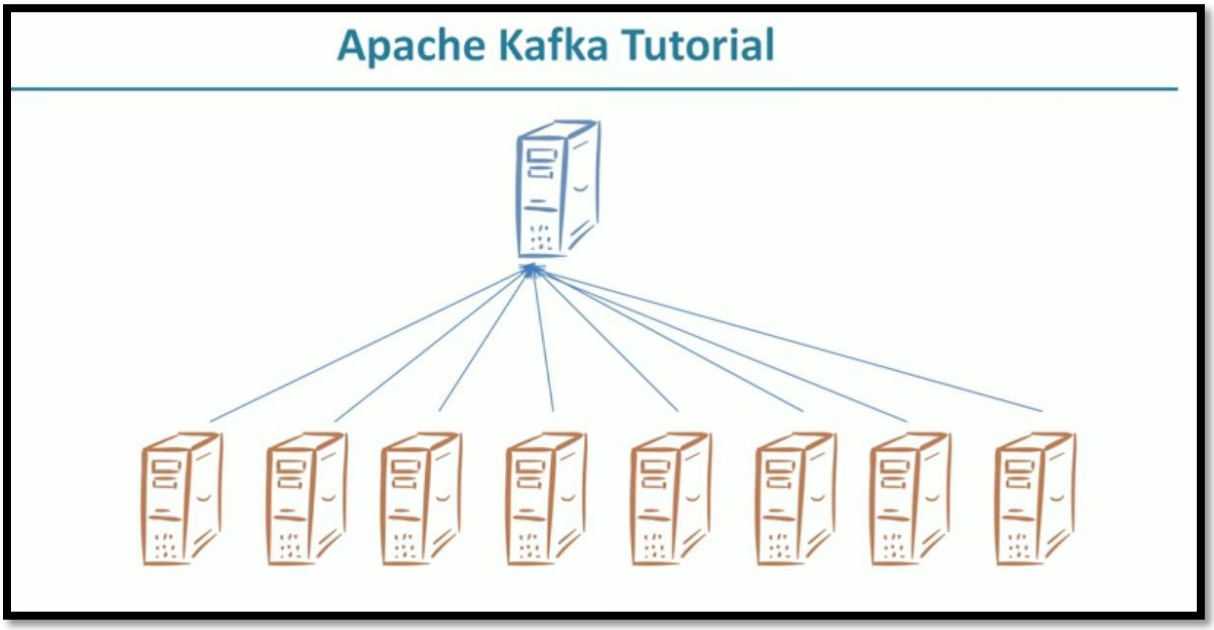


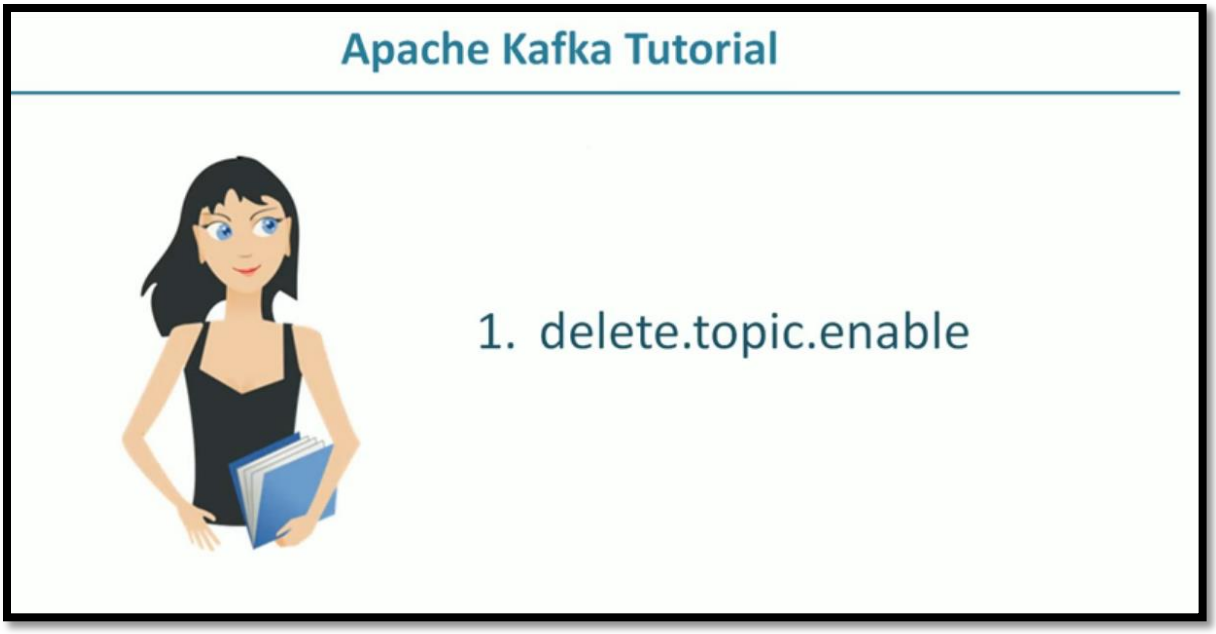
- For first partition broker1 is the leader and for second broker 2 is the leader
- Replicas shows the three copies of partitioned maintained by different brokers
- ISR is list of in synch replicas

Import configuration parameter

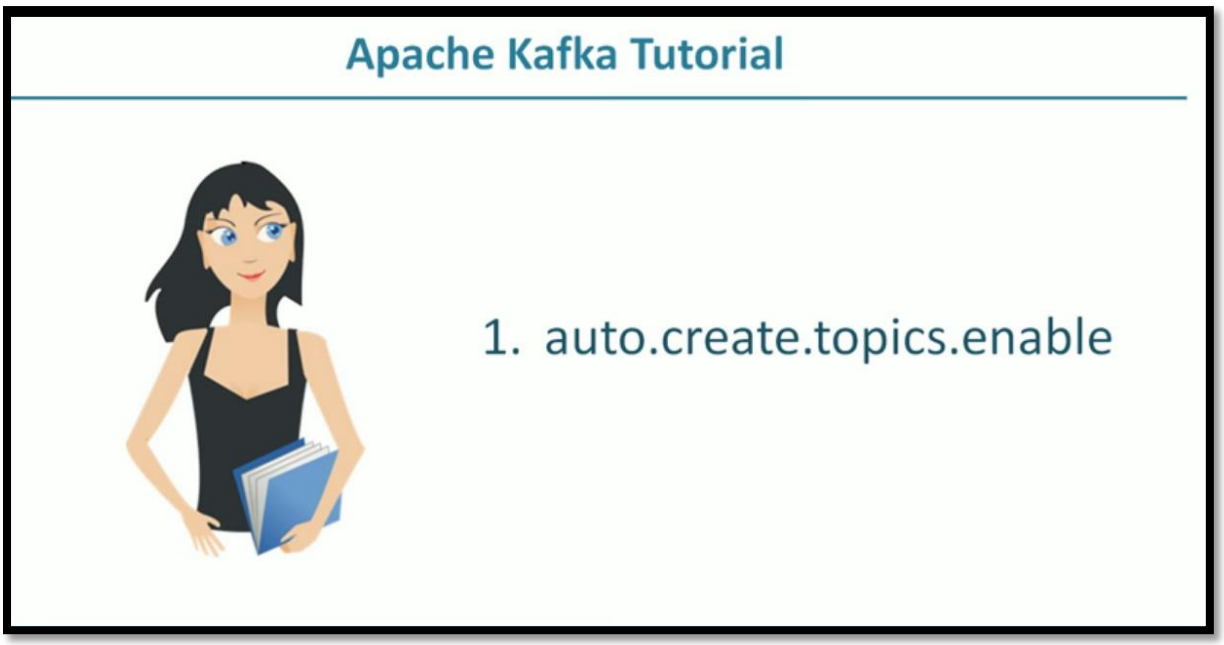


Zookeeper address that manages kafka cluster



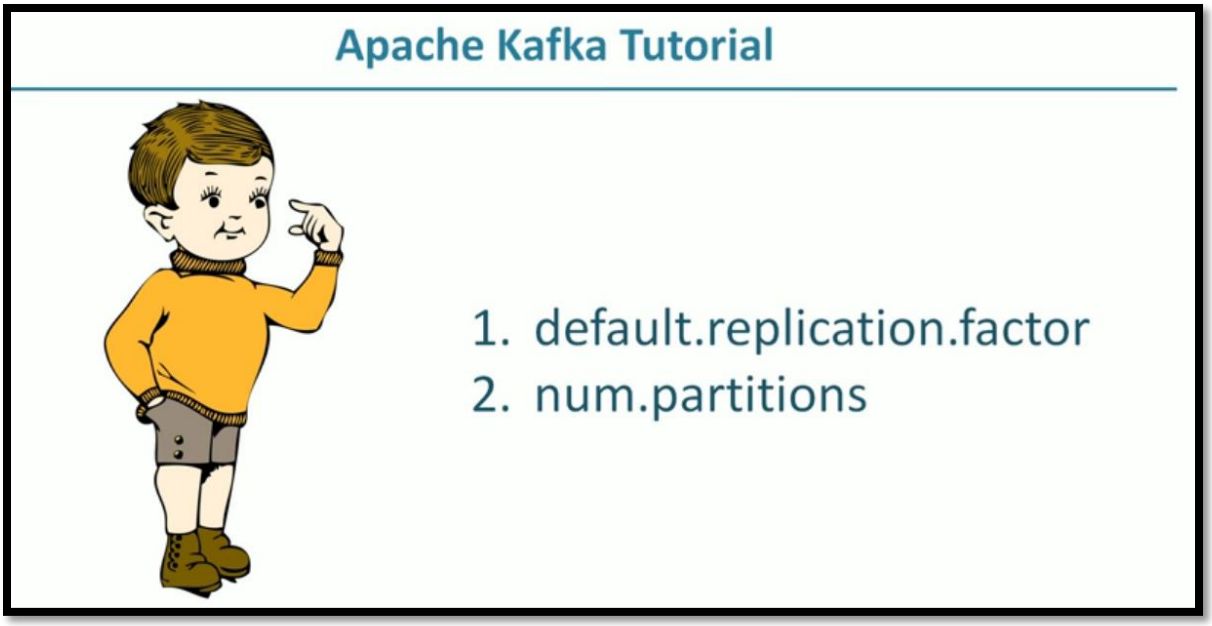


You cannot remove a topic by default value of delete.topic.enable is false , but we can change this to true in development and testing environment to delete the topic by command line tool

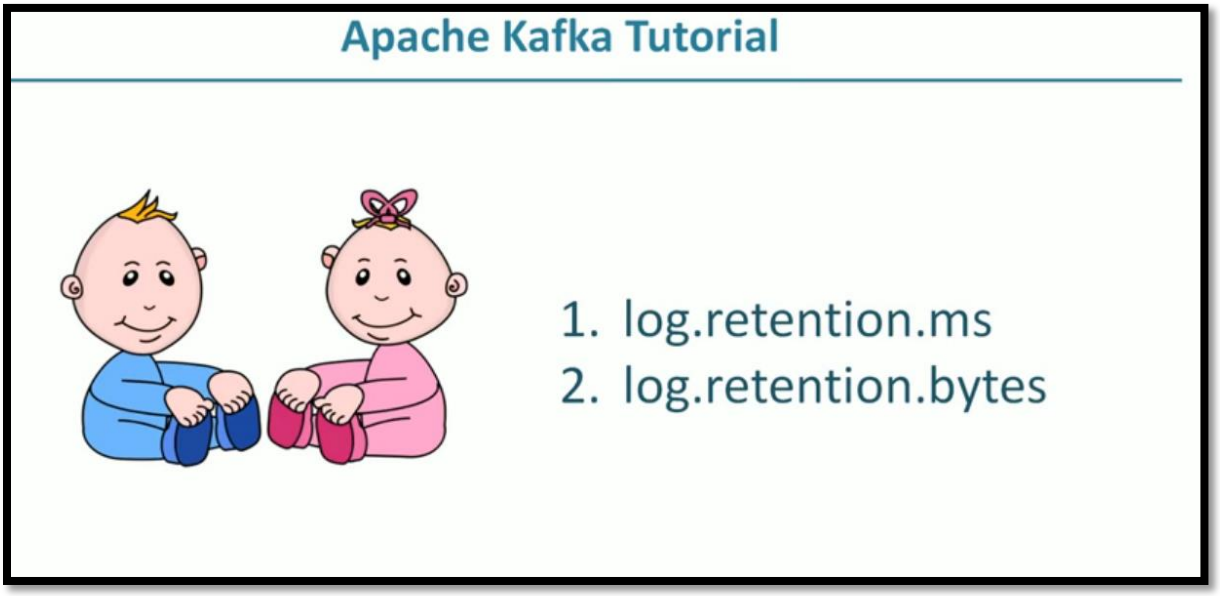


If this parameter is set to true kafka create topics automatically if producers sending message on non-existence topic

Set this parameter to false then kafka will stop creating topic automatically

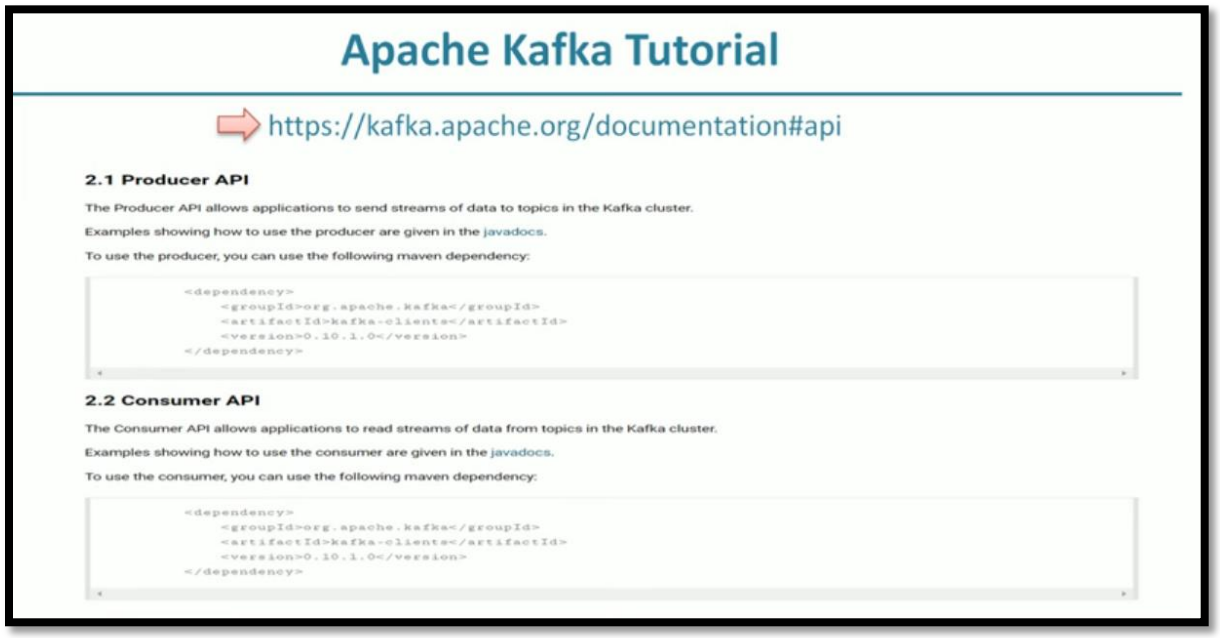


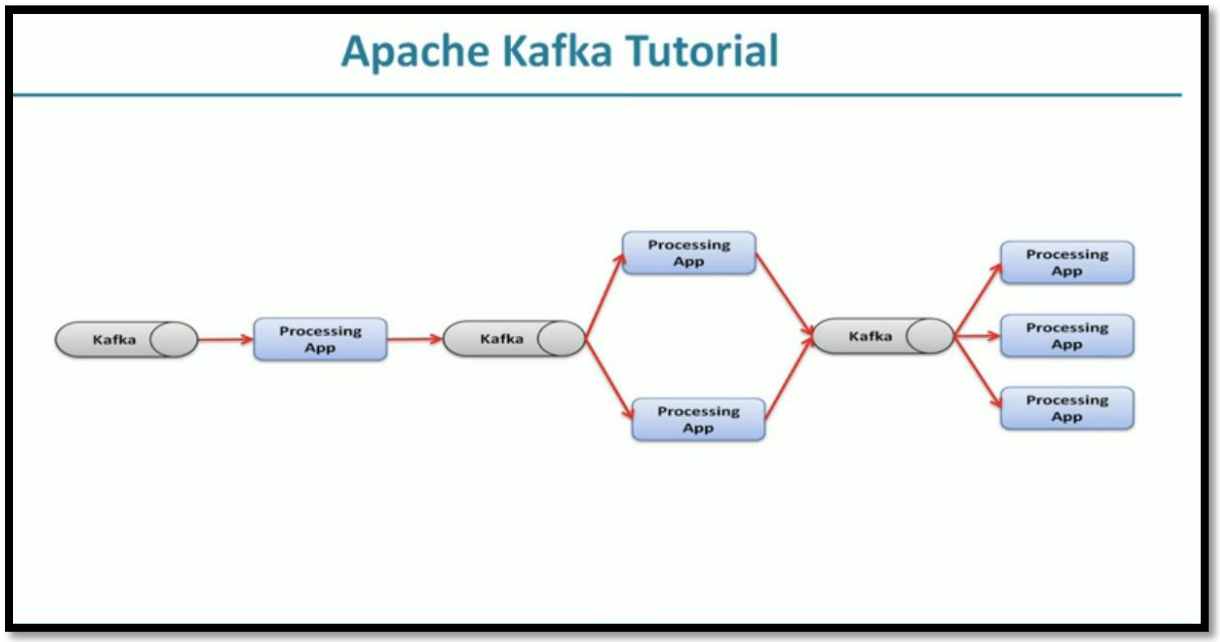
Default value for both of them is one ,if kafka creates topics automatically replication factor and partition are set to 1

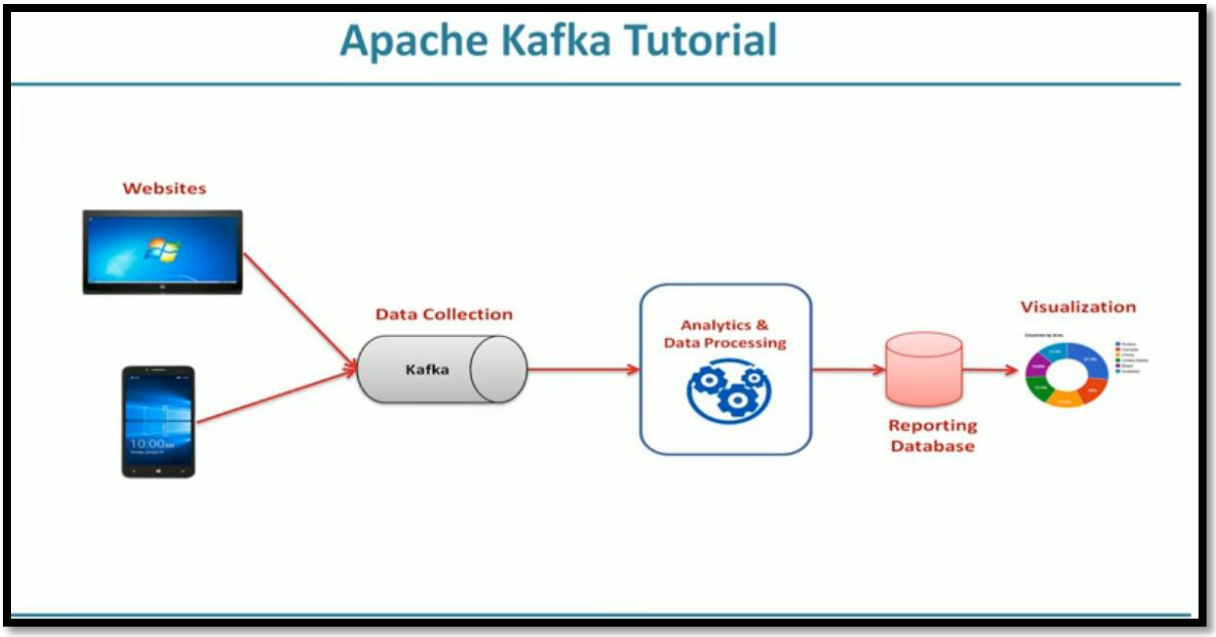


Kafka retain the data for certain amount of time and size, we can configure these two properties to configure them, default retention period is 7 days (in latest version we have log.retention.hours) log.retention.bytes applies to partition level (not a topic size) so if size of the partition reach up to this parameter kafka delete that partition

Producer API





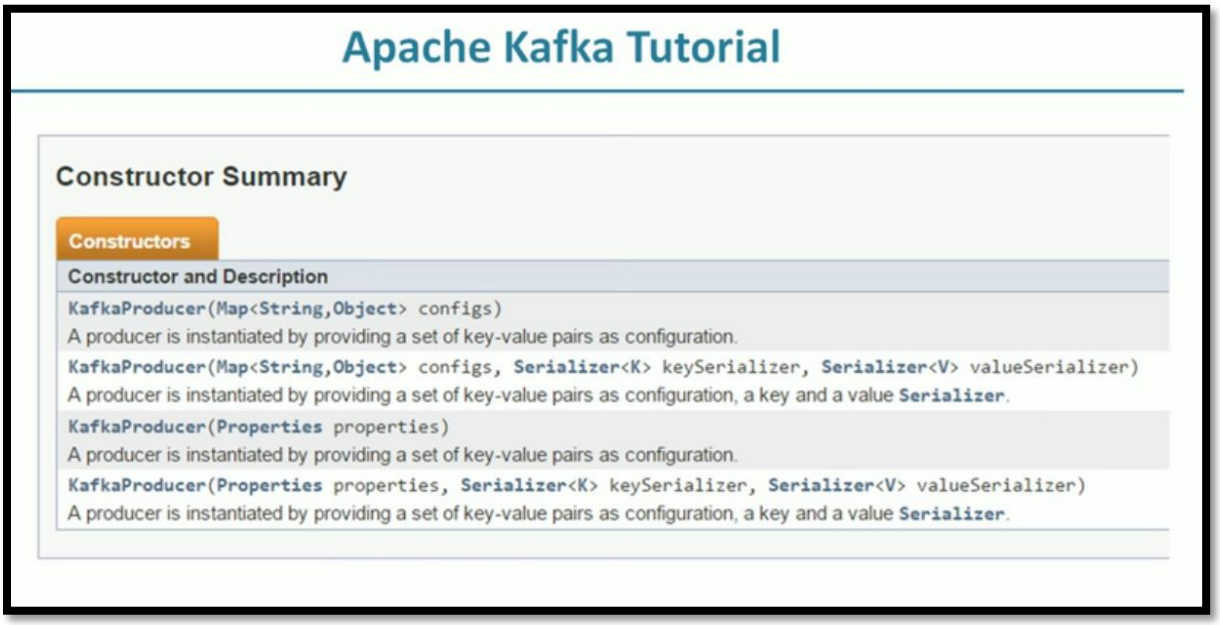


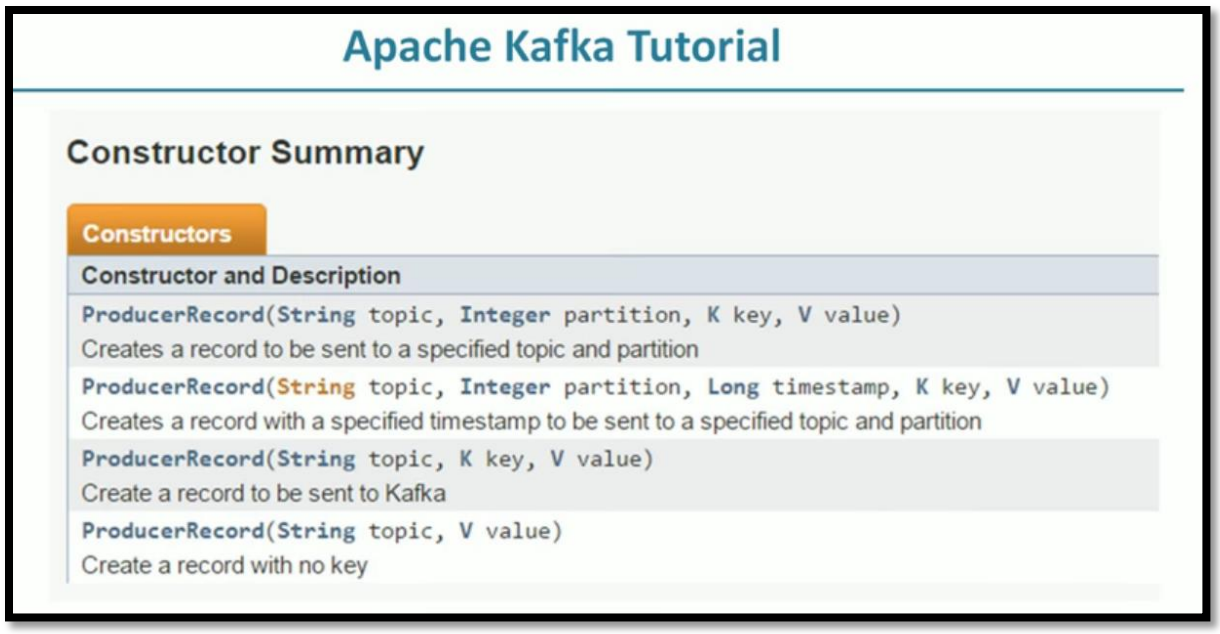
Get the code from below URL

https://github.com/LearningJournal/ApacheKafkaTutorials/tree/master/ProducerExamples

```
import java.util.*;
import org.apache.kafka.clients.producer.*;
public class SimpleProducer {
```

```
  public static void main(String[] args) throws Exception{

        String topicName = "SimpleProducerTopic";
        // generally we send key and value however key is not mandatory
        String key = "Key1";
        String value = "Value-1";

        Properties props = new Properties();
        //list of kafka brokers
        props.put("bootstrap.servers", "localhost:9092,localhost:9093");
        // type of key
        props.put("key.serializer","org.apache.kafka.common.serialization.StringSerializer");
        // type of value
        props.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer");

        // type of key.value
        Producer<String, String> producer = new KafkaProducer <>(props);
        //this object is our message
        ProducerRecord<String, String> record = new ProducerRecord<>(topicName,key,value);
        producer.send(record);
        producer.close();

        System.out.println("SimpleProducer Completed.");
  }
}
```



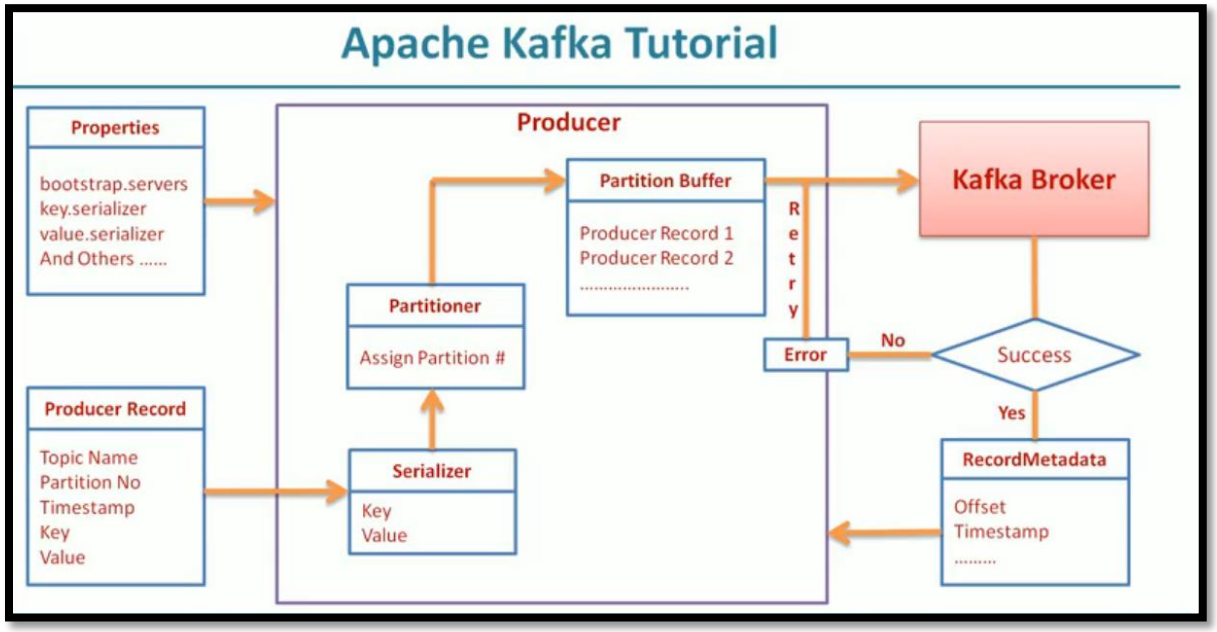


If you send multiple messages with same message key then they all will land to same partition but as key is an optional field in message so if we don't send keys with message , default partition eventually distributed messages across the partition.

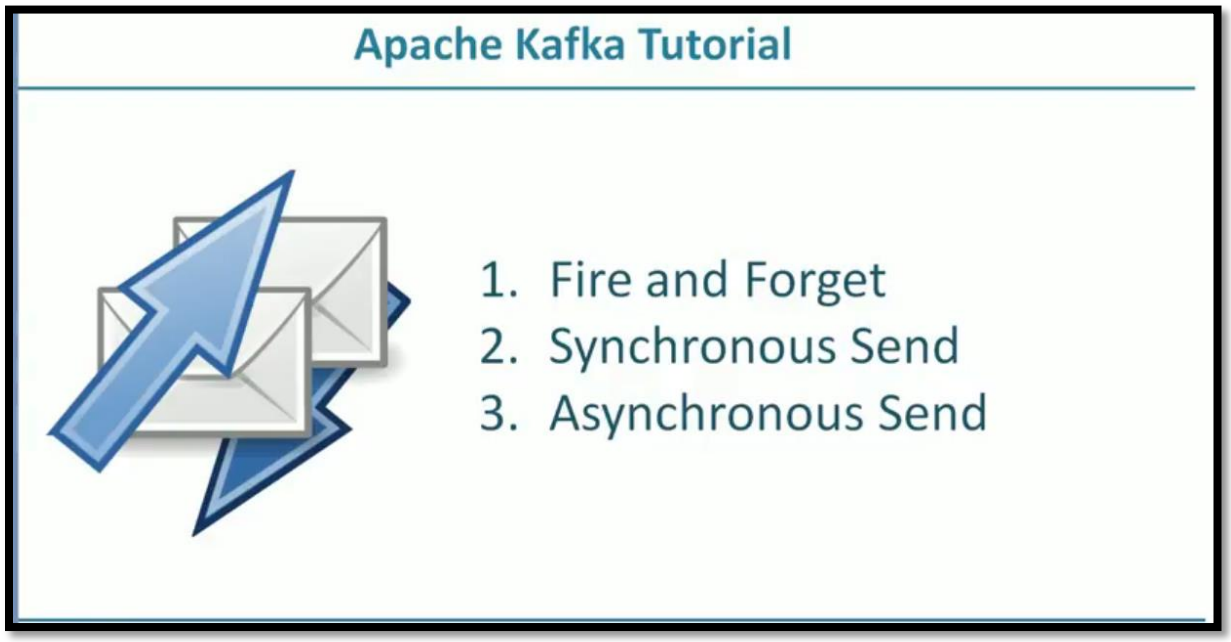
If we set the partition number in producerRecord, it disable default pardoner

We can set timestamp when we are sending the message if we don't set the timestamp would be the time when broker receive the message

Apache kafka Producer workflow



Callback and Acknowledgment



Three approaches to send message to kafka broker

1. Fire and forget – send message to broker and don't care if it is successful received by broker or not, in this approach you may miss some messages. Like in sentimental analysis we don't care about if we miss some tweet

2. Synchronous send – send message and wait for response, either get metadata record or exception, here we only care for exception as we may want to log the exception. If messages are critical and can't afford to miss anything, it will limit your throughput as we are waiting for response.

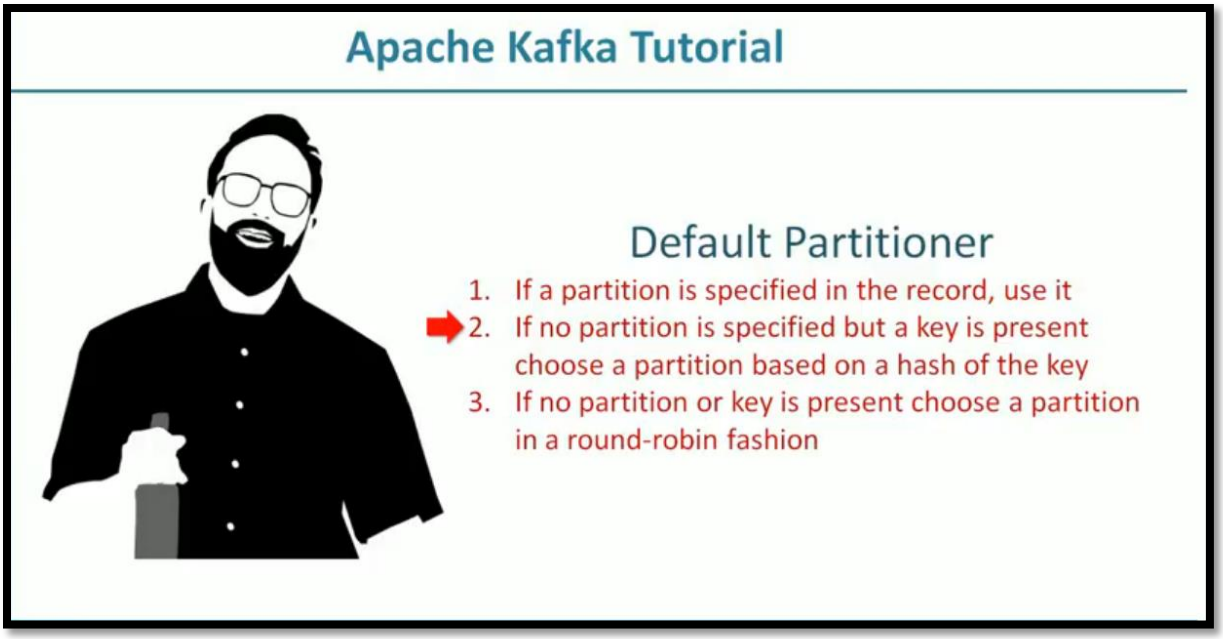
```
import java.util.*;
import org.apache.kafka.clients.producer.*;
public class SynchronousProducer {

  public static void main(String[] args) throws Exception{

    String topicName = "SynchronousProducerTopic";
       String key = "Key1";
       String value = "Value-1";

    Properties props = new Properties();
    props.put("bootstrap.servers", "localhost:9092,localhost:9093");
    props.put("key.serializer","org.apache.kafka.common.serialization.StringSerializer");
    props.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer");

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

       ProducerRecord<String, String> record = new ProducerRecord<>(topicName,key,value);

    try{
       RecordMetadata metadata = producer.send(record).get();
       System.out.println("Message is sent to Partition no " + metadata.partition() + " and offset " +
metadata.offset());
       System.out.println("SynchronousProducer Completed with success.");
    }catch (Exception e) {
       e.printStackTrace();
       System.out.println("SynchronousProducer failed with an exception");
    }finally{
       producer.close();
    }
  }
}
```

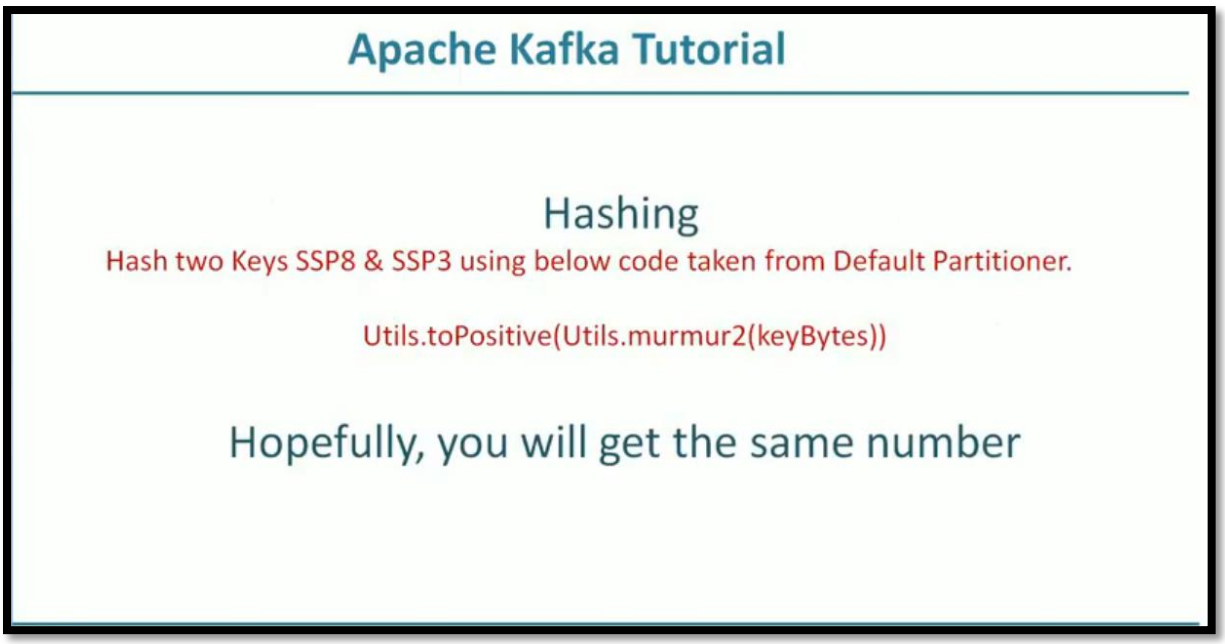
3. Asynchronous send – in this approach we send message and provide a callback function to receive an acknowledgement, we don't wait for success or failure , the producer will callback our function with recordmetada and an exception object.

```
import java.util.*;
import org.apache.kafka.clients.producer.*;

public class AsynchronousProducer {

  public static void main(String[] args) throws Exception{
    String topicName = "AsynchronousProducerTopic";
       String key = "Key1";
       String value = "Value-1";

    Properties props = new Properties();
    props.put("bootstrap.servers", "localhost:9092,localhost:9093");
    props.put("key.serializer","org.apache.kafka.common.serialization.StringSerializer");
    props.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer");

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

    ProducerRecord<String, String> record = new ProducerRecord<>(topicName,key,value);

    producer.send(record, new MyProducerCallback());
    System.out.println("AsynchronousProducer call completed");
    producer.close();

  }

}
class MyProducerCallback implements Callback{

     @Override
     public  void onCompletion(RecordMetadata recordMetadata, Exception e) {
      if (e != null)
         System.out.println("AsynchronousProducer failed with an exception");
      else
          System.out.println("AsynchronousProducer call Success:");
     }
  }
```

max.in.flight.requests.per.connection – tell how many message you can send to broker without receiving acknowledgement, the default value is 5

Custom Partition

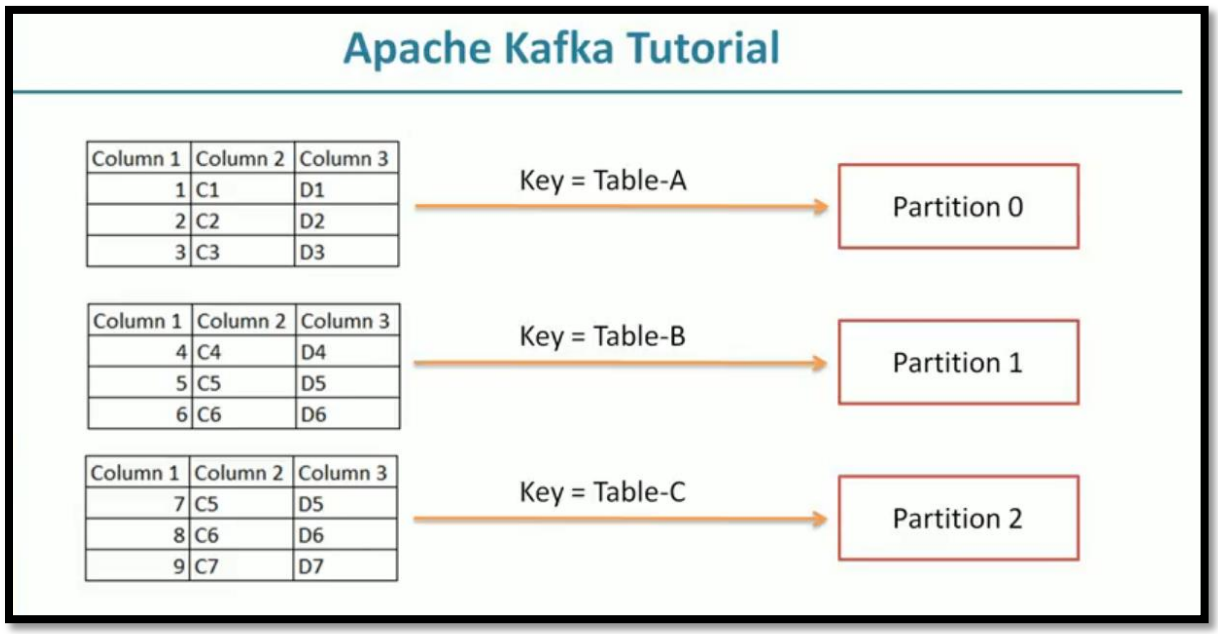


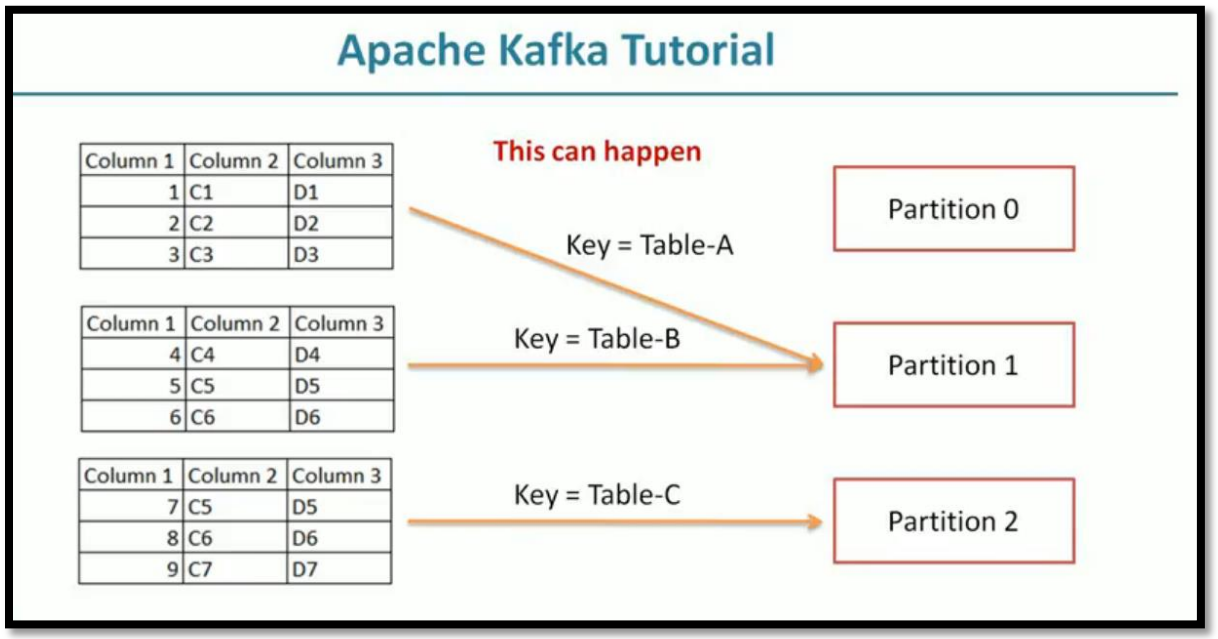


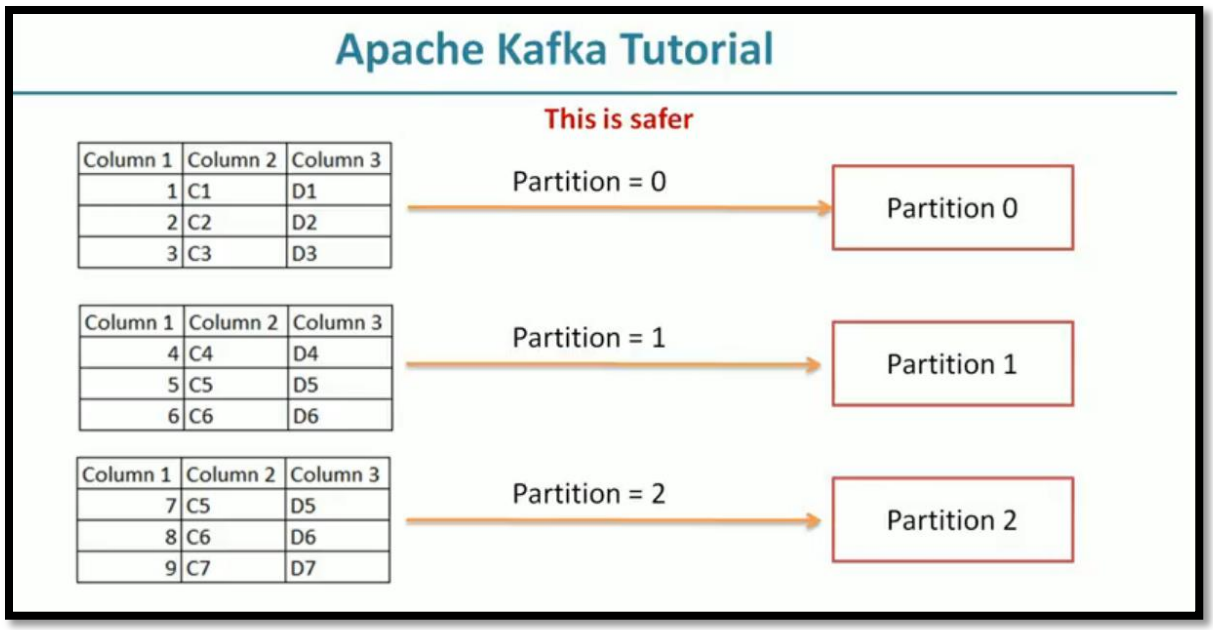
Same key will go to the same partition but this also may happen that two distinguish keys can go to same partition

If we want to create partitions in such a way that data from Table-A go to Partion-0 , Table-B to Partition-1 and so on.

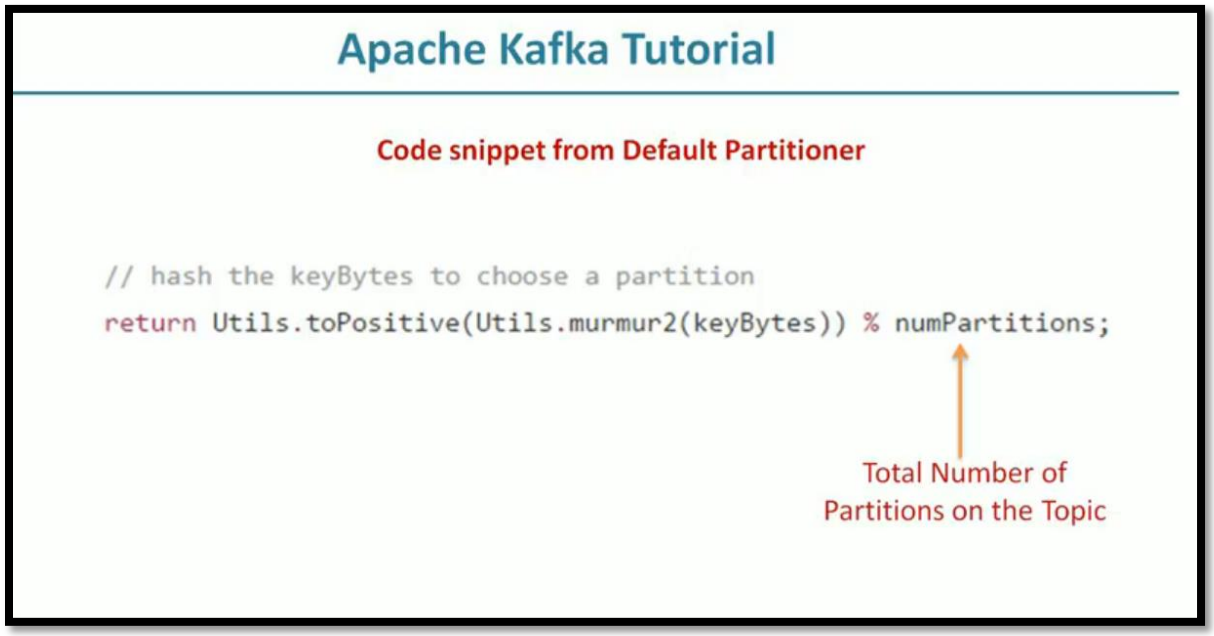
Here we are sending table name as a key







So better to send hardcoded partition number or we can create custom partition.

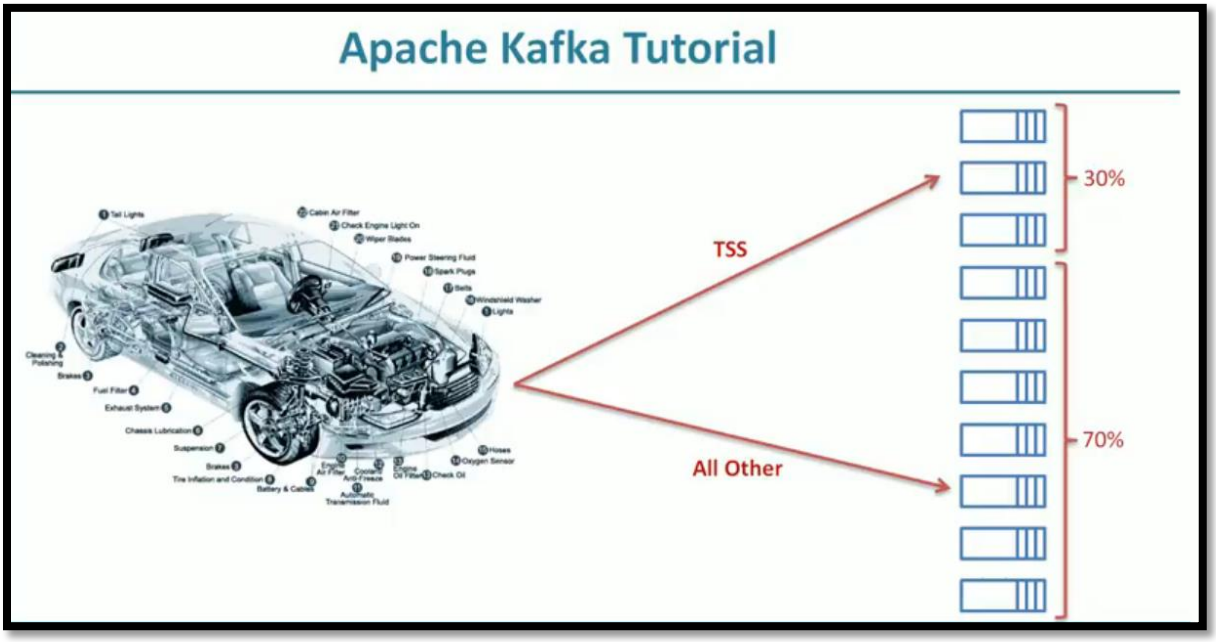


Default partition return partition based on key modulo number of partition

If you are increasing the number of partition later default partition will start returning some different numbers.

So key is not a good use for making partition

Use case for custom partition



We are receiving data from different sensors and we planned 10 partitions to a single topic but we want to 3 partitions dedicated for sensor named TSS and 7 partitions for rest of the sensors.

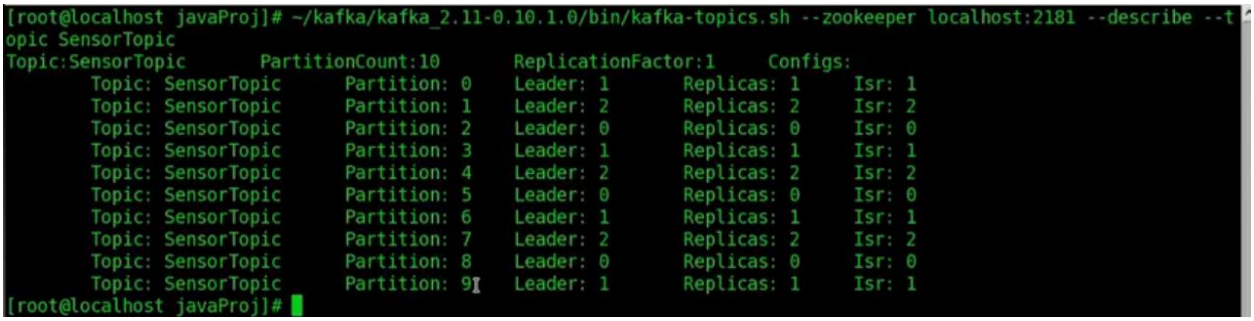
```
import java.util.*;
import org.apache.kafka.clients.producer.*;
public class SensorProducer {

  public static void main(String[] args) throws Exception{

    String topicName = "SensorTopic";

    Properties props = new Properties();
    props.put("bootstrap.servers", "localhost:9092,localhost:9093");
    props.put("key.serializer","org.apache.kafka.common.serialization.StringSerializer");
    props.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer");
    props.put("partitioner.class", "SensorPartitioner");
    props.put("speed.sensor.name", "TSS"); // it is custom property not a kafka property

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

      for (int i=0 ; i<10 ; i++)
      producer.send(new ProducerRecord<>(topicName,"SSP"+i,"500"+i));

      for (int i=0 ; i<10 ; i++)
      producer.send(new ProducerRecord<>(topicName,"TSS","500"+i));
```

```
    producer.close();

        System.out.println("SimpleProducer Completed.");
  }
}
```

```
import java.util.*;
import org.apache.kafka.clients.producer.*;
import org.apache.kafka.common.*;
import org.apache.kafka.common.utils.*;
import org.apache.kafka.common.record.*;

public class SensorPartitioner implements Partitioner {

    private String speedSensorName;

  // initialization method
    public void configure(Map<String, ?> configs) {
        speedSensorName = configs.get("speed.sensor.name").toString();

    }

    public int partition(String topic, Object key, byte[] keyBytes, Object value, byte[] valueBytes, Cluster
cluster) {

        List<PartitionInfo> partitions = cluster.partitionsForTopic(topic);
        int numPartitions = partitions.size();
// calculate 30% of total partition for TSS sensor which is 3 in our case as total partitions are 10
        int sp = (int)Math.abs(numPartitions*0.3);
        int p=0;

         if ( (keyBytes == null) || (!(key instanceof String)) )
            throw new InvalidRecordException("All messages must have sensor name as key");

         if ( ((String)key).equals(speedSensorName) )
//here we are using value for hashing because in that case key is constant (TSS) if we give the key it will
always return single partition so all the TSS data goes to single partition
            p = Utils.toPositive(Utils.murmur2(valueBytes)) % sp;
         else
            p = Utils.toPositive(Utils.murmur2(keyBytes)) % (numPartitions-sp) + sp ;

            System.out.println("Key = " + (String)key + " Partition = " + p );
            return p;
  }
 // cleanup method
    public void close() {}
```

```
}
```

We created a topic with ten partitions

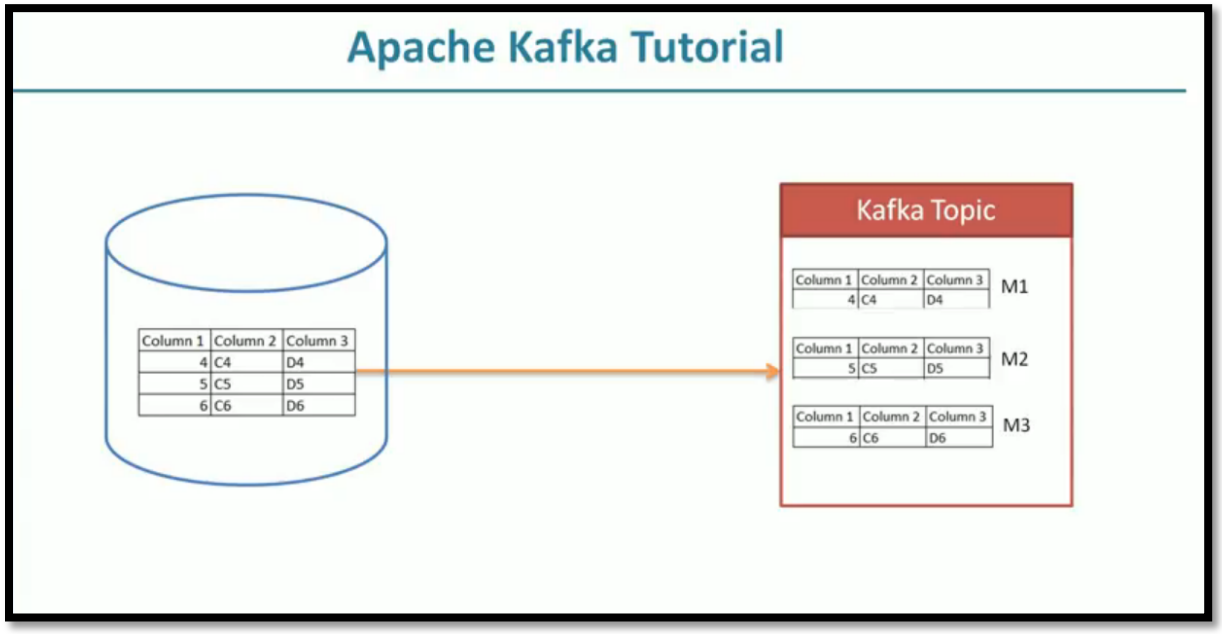


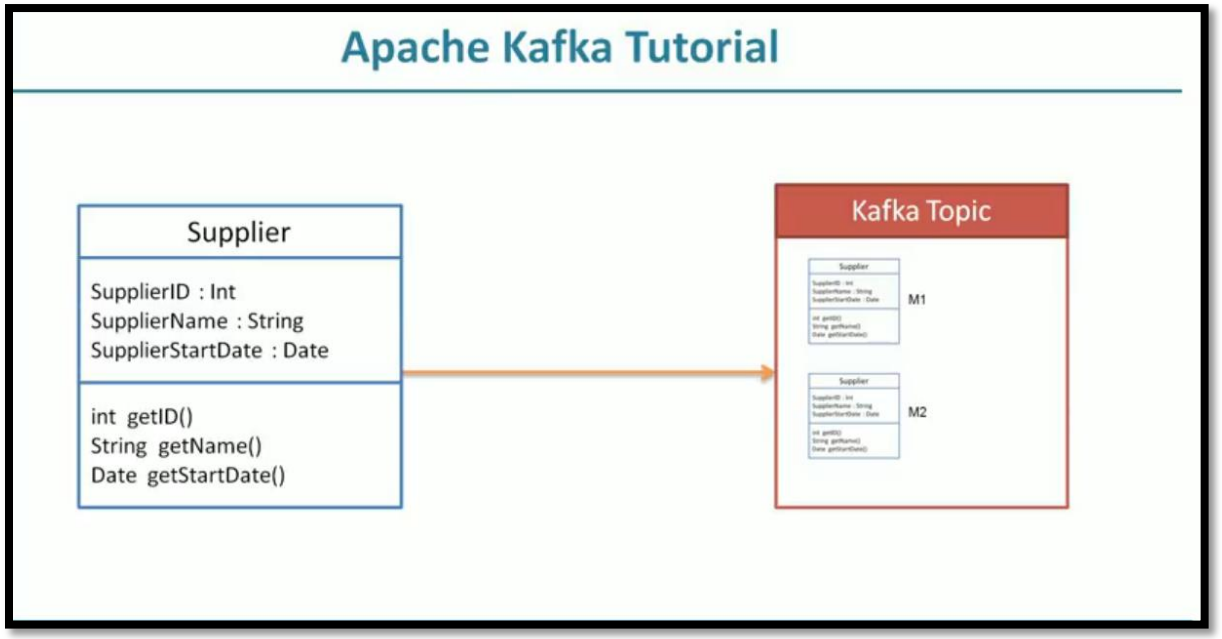
```
[root@localhost javaProj]# ~/kafka/kafka_2.11-0.10.1.0/bin/kafka-topics.sh --zookeeper localhost:2181 --describe --t
opic SensorTopic
Topic:SensorTopic       PartitionCount:10       ReplicationFactor:1     Configs:
        Topic: SensorTopic      Partition: 0    Leader: 1       Replicas: 1     Isr: 1
        Topic: SensorTopic      Partition: 1    Leader: 2       Replicas: 2     Isr: 2
        Topic: SensorTopic      Partition: 2    Leader: 0       Replicas: 0     Isr: 0
        Topic: SensorTopic      Partition: 3    Leader: 1       Replicas: 1     Isr: 1
        Topic: SensorTopic      Partition: 4    Leader: 2       Replicas: 2     Isr: 2
        Topic: SensorTopic      Partition: 5    Leader: 0       Replicas: 0     Isr: 0
        Topic: SensorTopic      Partition: 6    Leader: 1       Replicas: 1     Isr: 1
        Topic: SensorTopic      Partition: 7    Leader: 2       Replicas: 2     Isr: 2
        Topic: SensorTopic      Partition: 8    Leader: 0       Replicas: 0     Isr: 0
        Topic: SensorTopic      Partition: 9    Leader: 1       Replicas: 1     Isr: 1
[root@localhost javaProj]#
```

```
 [1] AsynchronousProducer
 [2] SensorProducer
 [3] SimpleProducer
 [4] SynchronousProducer
^JEnter number: 2

[info] Running SensorProducer
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
Key = SSP0 Partition = 4
Key = SSP1 Partition = 9
Key = SSP2 Partition = 5
Key = SSP3 Partition = 6
Key = SSP4 Partition = 3
Key = SSP5 Partition = 8
Key = SSP6 Partition = 3
Key = SSP7 Partition = 9
Key = SSP8 Partition = 6
Key = SSP9 Partition = 7
Key = TSS Partition = 1
Key = TSS Partition = 0
Key = TSS Partition = 0
Key = TSS Partition = 2
Key = TSS Partition = 0
Key = TSS Partition = 1
Key = TSS Partition = 0
Key = TSS Partition = 0
Key = TSS Partition = 1
Key = TSS Partition = 1
```

TSS is distributed in partition 0, 1 and 2 and rests are distributed 3 to 9

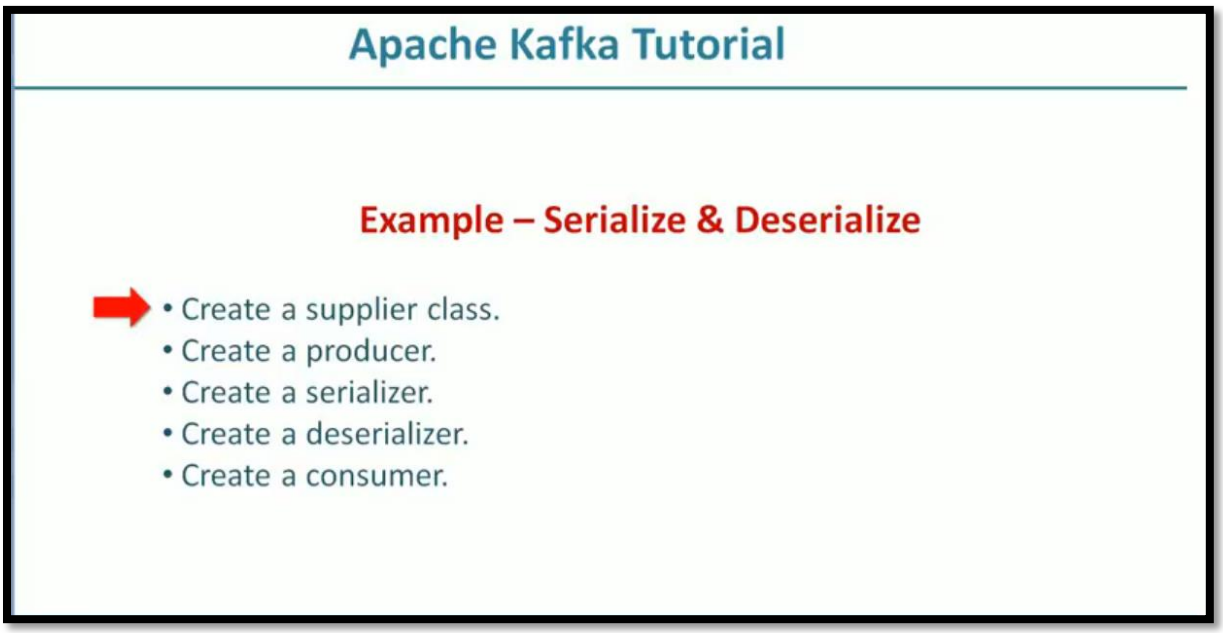
Custom Serializer





Sometime we need to send whole object as a message to kafka broker instead of plain text

For that we can use custom serializer , there are some generic serilazer available like Avro and protocol buffer, most of the time we will use generic serializer like Avro




```
import java.util.Date;
public class Supplier{
	private int supplierId;
	private String supplierName;
	private Date supplierStartDate;

	public Supplier(int id, String name, Date dt){
		this.supplierId = id;
		this.supplierName = name;
		this.supplierStartDate = dt;
	}

	public int getID(){
		return supplierId;
	}

	public String getName(){
		return supplierName;
	}

	public Date getStartDate(){
		return supplierStartDate;
	}
}
```

```
import org.apache.kafka.common.serialization.Serializer;
import org.apache.kafka.common.errors.SerializationException;
import java.io.UnsupportedEncodingException;
import java.util.Map;
import java.nio.ByteBuffer;

public class SupplierSerializer implements Serializer<Supplier> {
   private String encoding = "UTF8";

   @Override
   public void configure(Map<String, ?> configs, boolean isKey) {
           // nothing to configure
   }

   @Override
   public byte[] serialize(String topic, Supplier data) {

           int sizeOfName;
           int sizeOfDate;
           byte[] serializedName;
           byte[] serializedDate;

      try {
         if (data == null)
            return null;
                   serializedName = data.getName().getBytes(encoding);
                      sizeOfName = serializedName.length;
                      serializedDate = data.getStartDate().toString().getBytes(encoding);
                      sizeOfDate = serializedDate.length;

                      ByteBuffer buf = ByteBuffer.allocate(4+4+sizeOfName+4+sizeOfDate);
                      buf.putInt(data.getID());
                      buf.putInt(sizeOfName);
                      buf.put(serializedName);
                      buf.putInt(sizeOfDate);
                      buf.put(serializedDate);


           return buf.array();

      } catch (Exception e) {
         throw new SerializationException("Error when serializing Supplier to byte[]");
      }
   }

   @Override
   public void close() {
```

```
    // nothing to do
  }
}
```

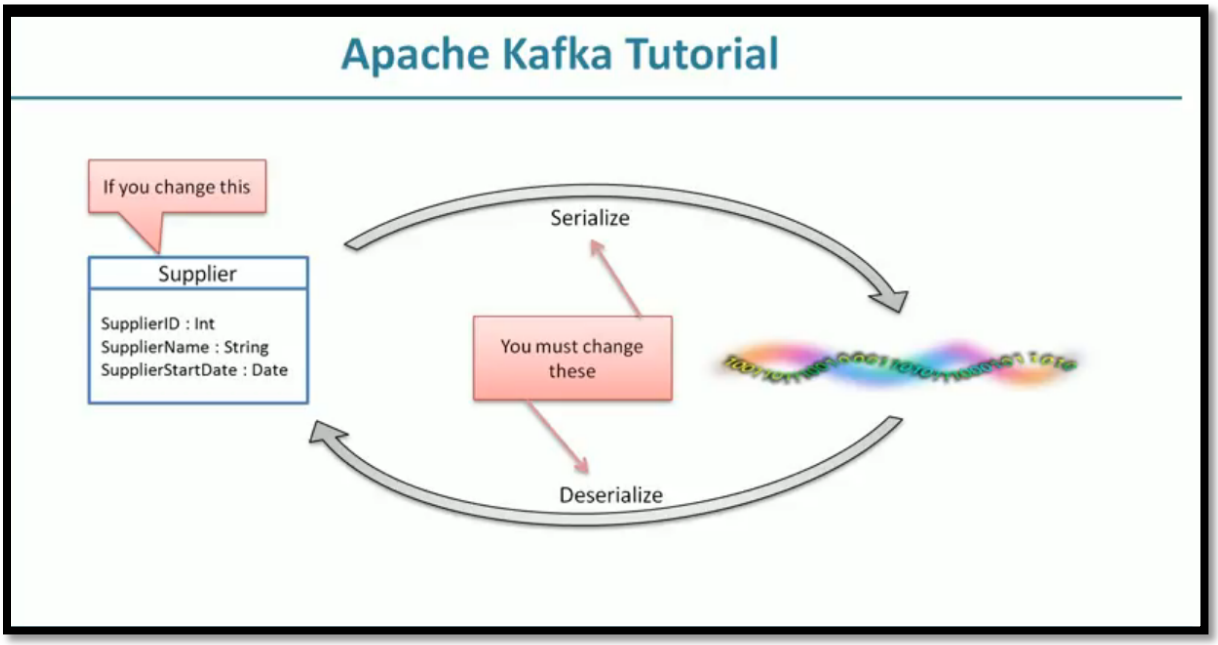
```
import java.nio.ByteBuffer;
import java.util.Date;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import org.apache.kafka.common.errors.SerializationException;
import org.apache.kafka.common.serialization.Deserializer;
import java.io.UnsupportedEncodingException;
import java.util.Map;

public class SupplierDeserializer implements Deserializer<Supplier> {
  private String encoding = "UTF8";

  @Override
  public void configure(Map<String, ?> configs, boolean isKey) {
        //Nothing to configure
    }

  @Override
  public Supplier deserialize(String topic, byte[] data) {

    try {
      if (data == null){
        System.out.println("Null recieved at deserialize");
              return null;
              }
      ByteBuffer buf = ByteBuffer.wrap(data);
      int id = buf.getInt();

      int sizeOfName = buf.getInt();
      byte[] nameBytes = new byte[sizeOfName];
      buf.get(nameBytes);
      String deserializedName = new String(nameBytes, encoding);

      int sizeOfDate = buf.getInt();
      byte[] dateBytes = new byte[sizeOfDate];
      buf.get(dateBytes);
      String dateString = new String(dateBytes,encoding);

      DateFormat df = new SimpleDateFormat("EEE MMM dd HH:mm:ss Z yyyy");

      return new Supplier(id,deserializedName,df.parse(dateString));
```

```
        } catch (Exception e) {
            throw new SerializationException("Error when deserializing byte[] to Supplier");
        }
    }

    @Override
    public void close() {
        // nothing to do
    }
}
```

```
import java.util.*;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import org.apache.kafka.clients.producer.*;
public class SupplierProducer {

   public static void main(String[] args) throws Exception{

     String topicName = "SupplierTopic";

     Properties props = new Properties();
     props.put("bootstrap.servers", "localhost:9092,localhost:9093");
     props.put("key.serializer","org.apache.kafka.common.serialization.StringSerializer");
     props.put("value.serializer", "SupplierSerializer");

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

        DateFormat df = new SimpleDateFormat("yyyy-MM-dd");
        Supplier sp1 = new Supplier(101,"Xyz Pvt Ltd.",df.parse("2016-04-01"));
        Supplier sp2 = new Supplier(102,"Abc Pvt Ltd.",df.parse("2012-01-01"));

       producer.send(new ProducerRecord<String,Supplier>(topicName,"SUP",sp1)).get();
       producer.send(new ProducerRecord<String,Supplier>(topicName,"SUP",sp2)).get();

             System.out.println("SupplierProducer Completed.");
       producer.close();

   }
}
```

```
import java.util.*;
import org.apache.kafka.clients.consumer.KafkaConsumer;
import org.apache.kafka.clients.consumer.ConsumerRecords;
import org.apache.kafka.clients.consumer.ConsumerRecord;

public class SupplierConsumer{
```

```
    public static void main(String[] args) throws Exception{

        String topicName = "SupplierTopic";
        String groupName = "SupplierTopicGroup";

        Properties props = new Properties();
        props.put("bootstrap.servers", "localhost:9092,localhost:9093");
        props.put("group.id", groupName);
        props.put("key.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");
        props.put("value.deserializer", "SupplierDeserializer");


        KafkaConsumer<String, Supplier> consumer = new KafkaConsumer<>(props);
        consumer.subscribe(Arrays.asList(topicName));

        while (true){
            ConsumerRecords<String, Supplier> records = consumer.poll(100);
            for (ConsumerRecord<String, Supplier> record : records){
                System.out.println("Supplier id= " + String.valueOf(record.value().getID()) + " Supplier
Name = " + record.value().getName() + " Supplier Start Date = " +
record.value().getStartDate().toString());
            }
        }

    }
}
```

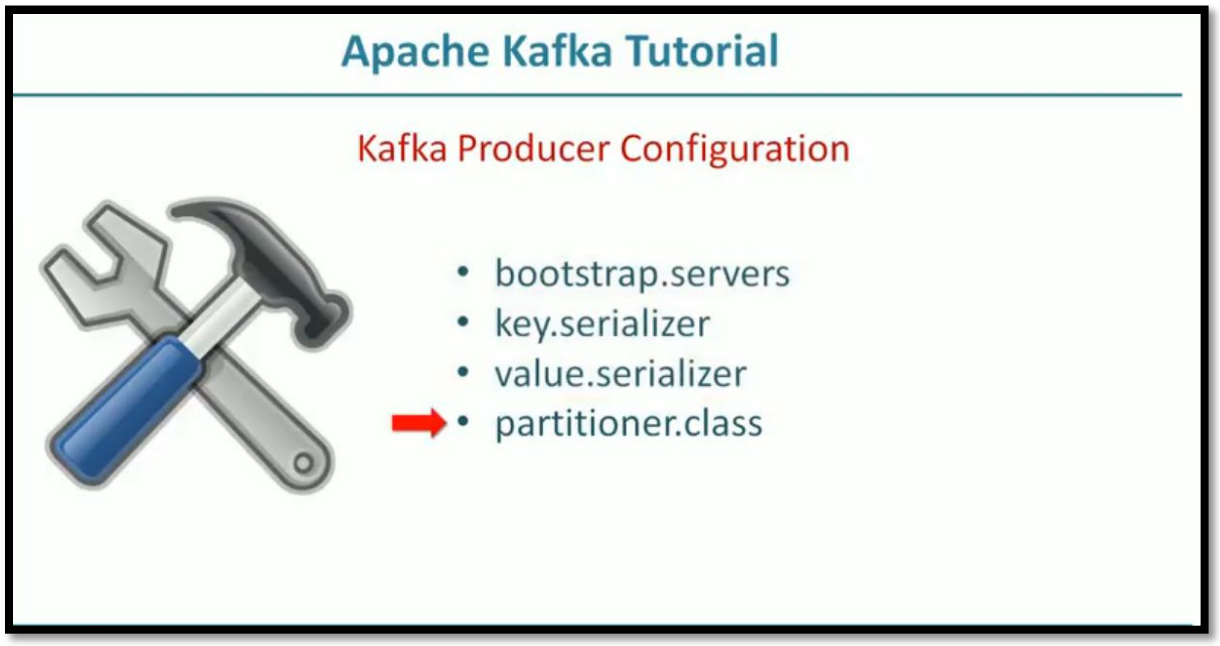


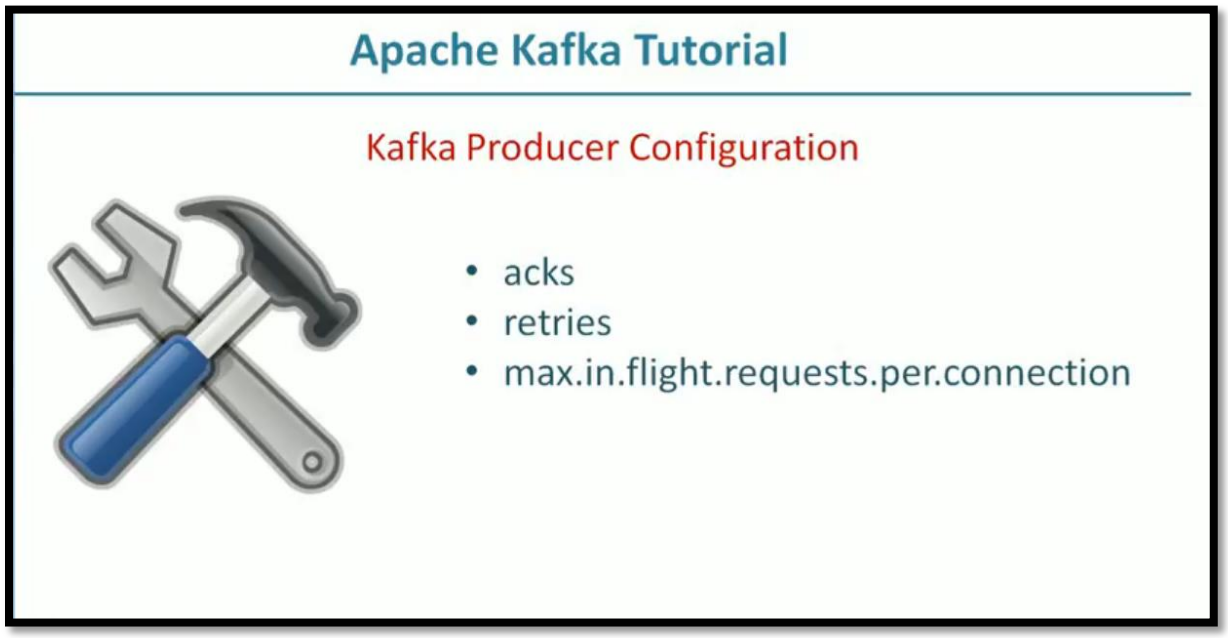
If we change the supplier by adding some new fields we need to change our serializer and deserializer class but problem of this approach is we can't read previous serialized object , so generic serialzier like Avro is good option for this case.

Producer Config

Follow this link to get full list of configuration parameters for producer

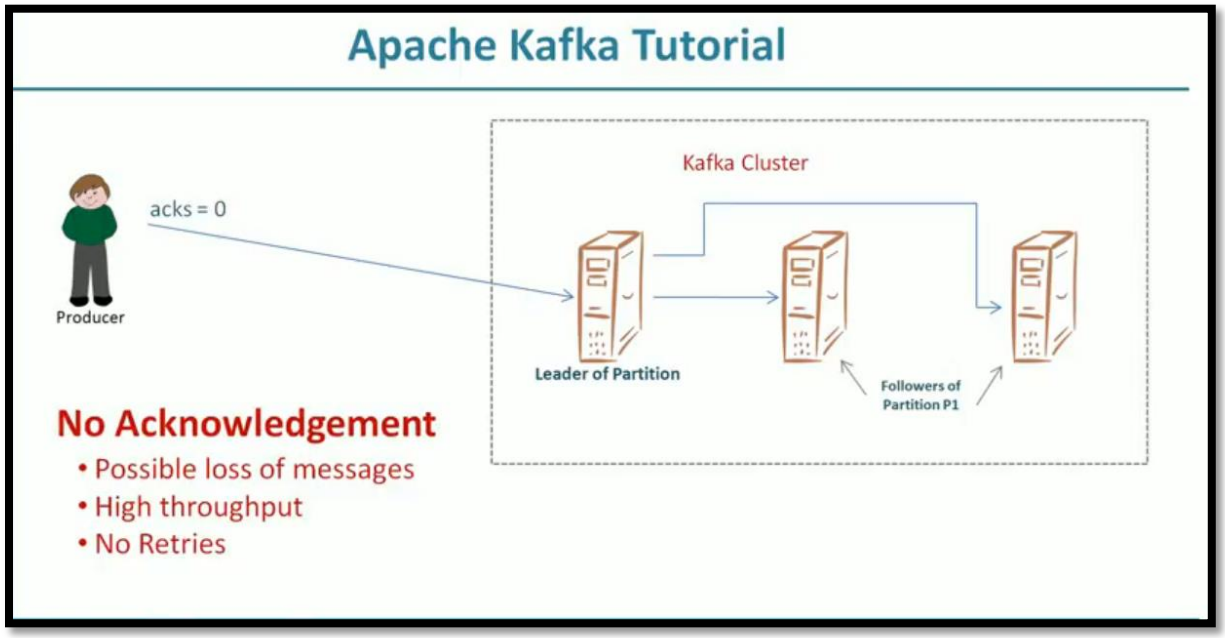
https://kafka.apache.org/documentation/#producerconfigs

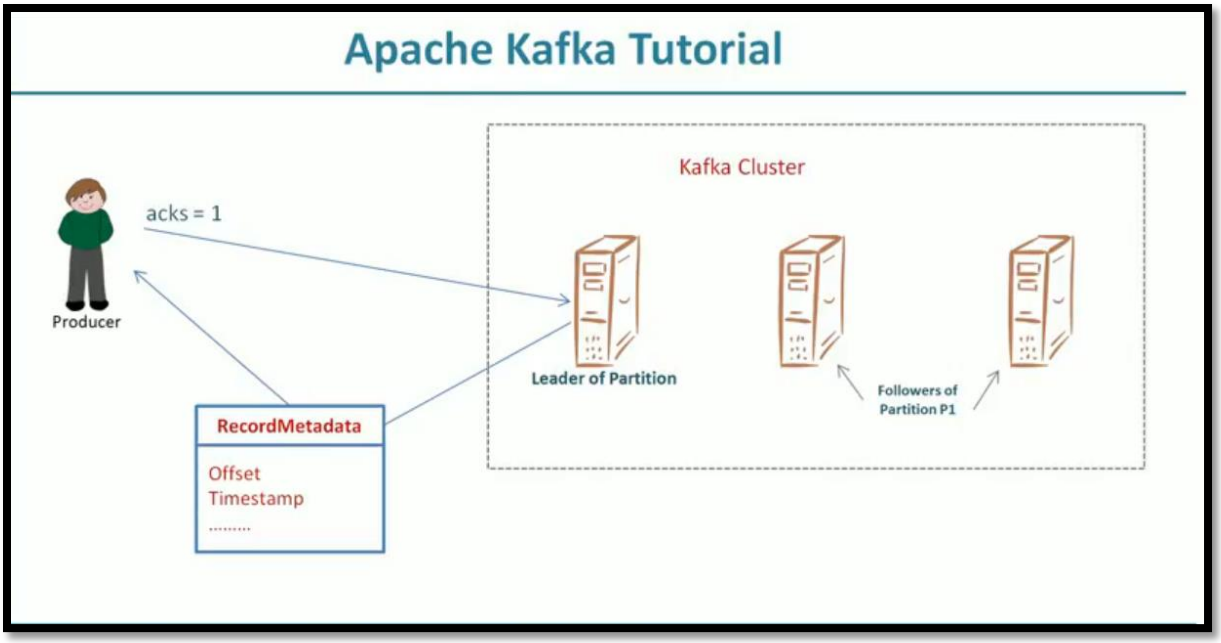




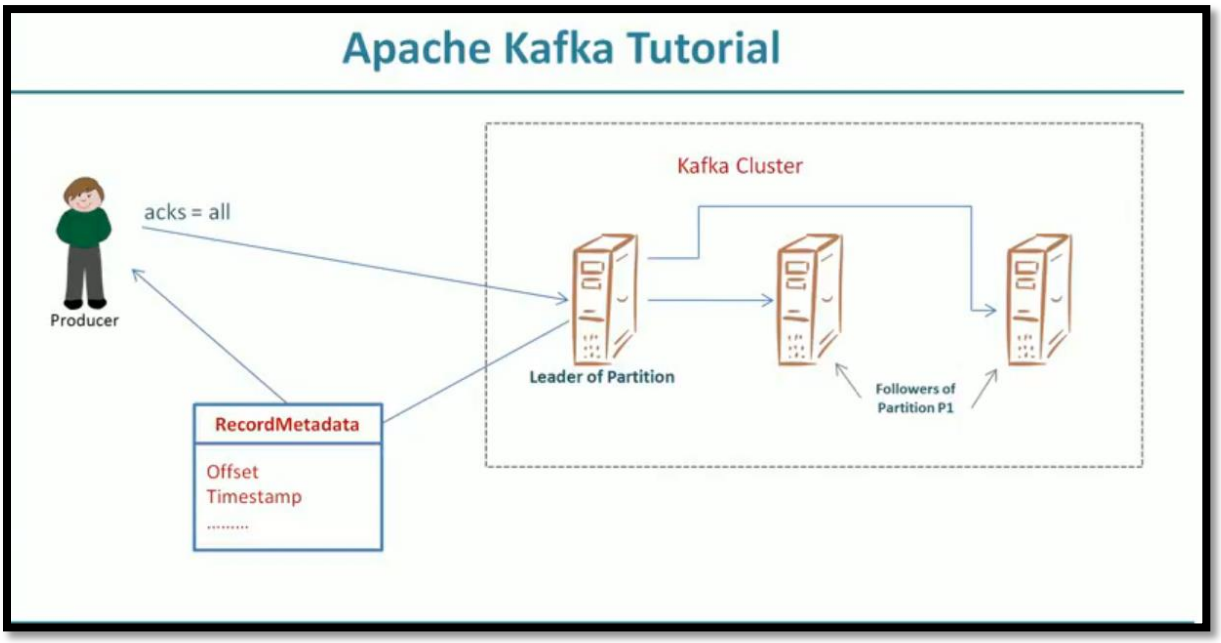
#acks – The number of acknowledgments the producer requires the leader to have received before considering a request complete. This controls the durability of records that are sent. The following settings are allowed:

- **acks=0** If set to zero then the producer will not wait for any acknowledgment from the server at all. The record will be immediately added to the socket buffer and considered sent. No guarantee can be made that the server has received the record in this case, and the retries configuration will not take effect (as the client won't generally know of any failures). The offset given back for each record will always be set to -1.
- **acks=1** This will mean the leader will write the record to its local log but will respond without awaiting full acknowledgement from all followers. In this case should the leader fail immediately after acknowledging the record but before the followers have replicated it then the record will be lost.
- **acks=all** This means the leader will wait for the full set of in-sync replicas to acknowledge the record. This guarantees that the record will not be lost as long as at least one in-sync replica remains alive. This is the strongest available guarantee. This is equivalent to the acks=-1 setting.





Leader sends acknowledgment before creating replicas , so there is chance to lose the data if leader crashes after acknowledgement



Leader acknowledge only after it receive acknowledge from all of the replicas

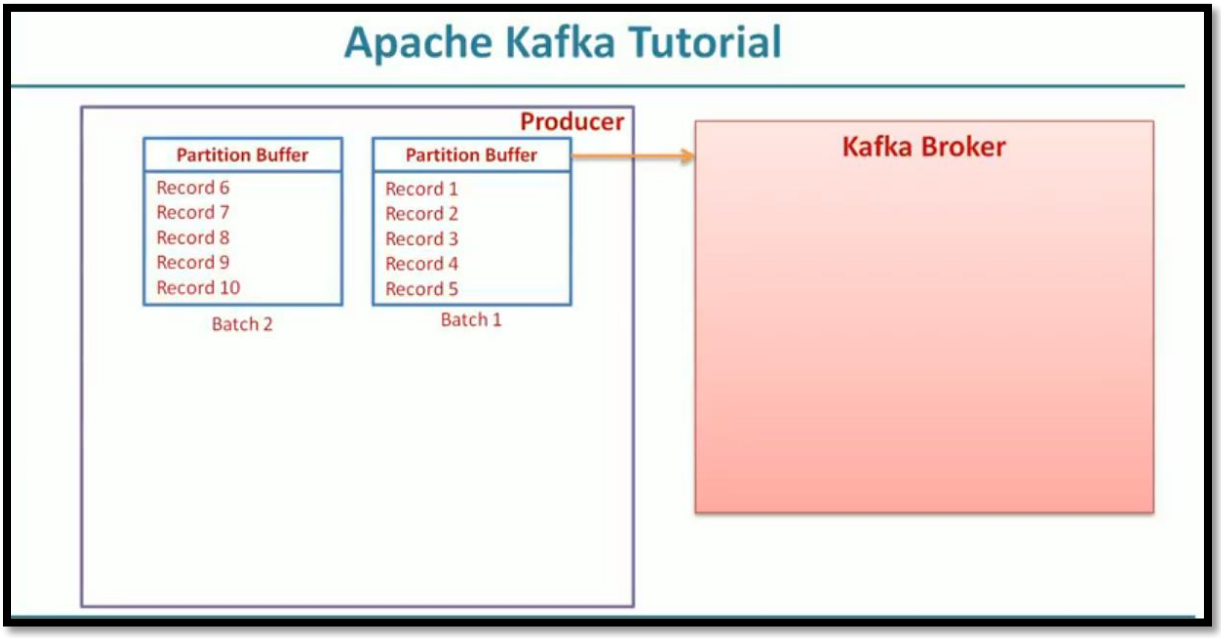
High reliable but high latency

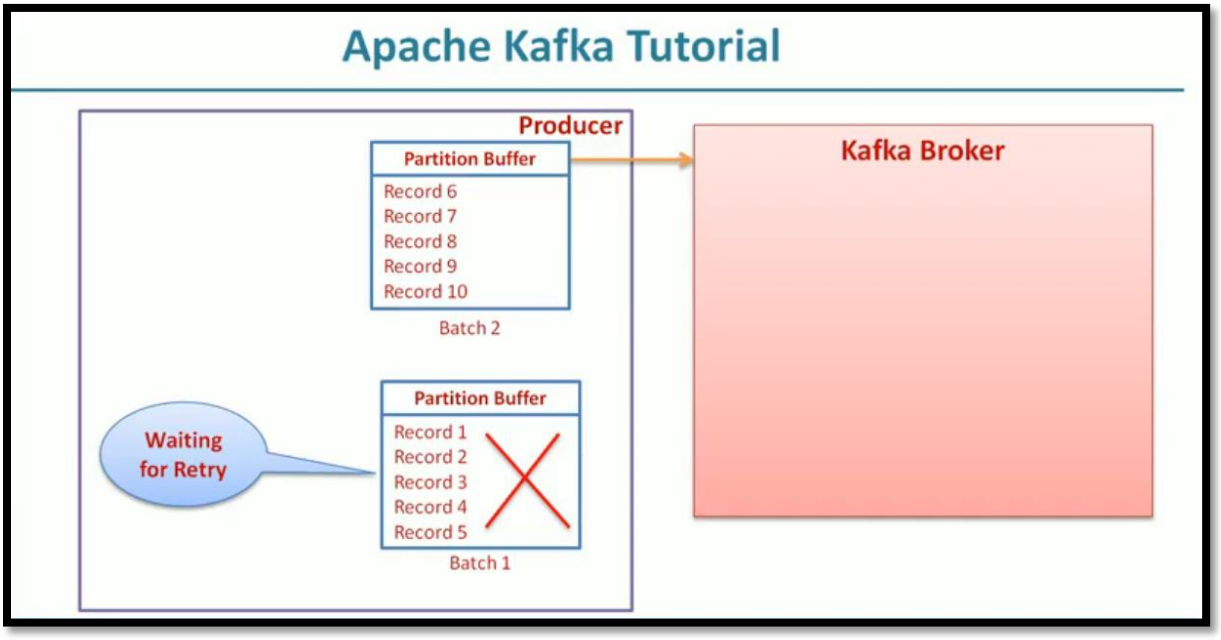
#retries - Setting a value greater than zero will cause the client to resend any record whose send fails with a potentially transient error. Note that this retry is no different than if the client resent the record

upon receiving the error. Allowing retries without setting **max.in.flight.requests.per.connection** to 1 will potentially change the ordering of records because if two batches are sent to a single partition, and the first fails and is retried but the second succeeds, then the records in the second batch may appear first.

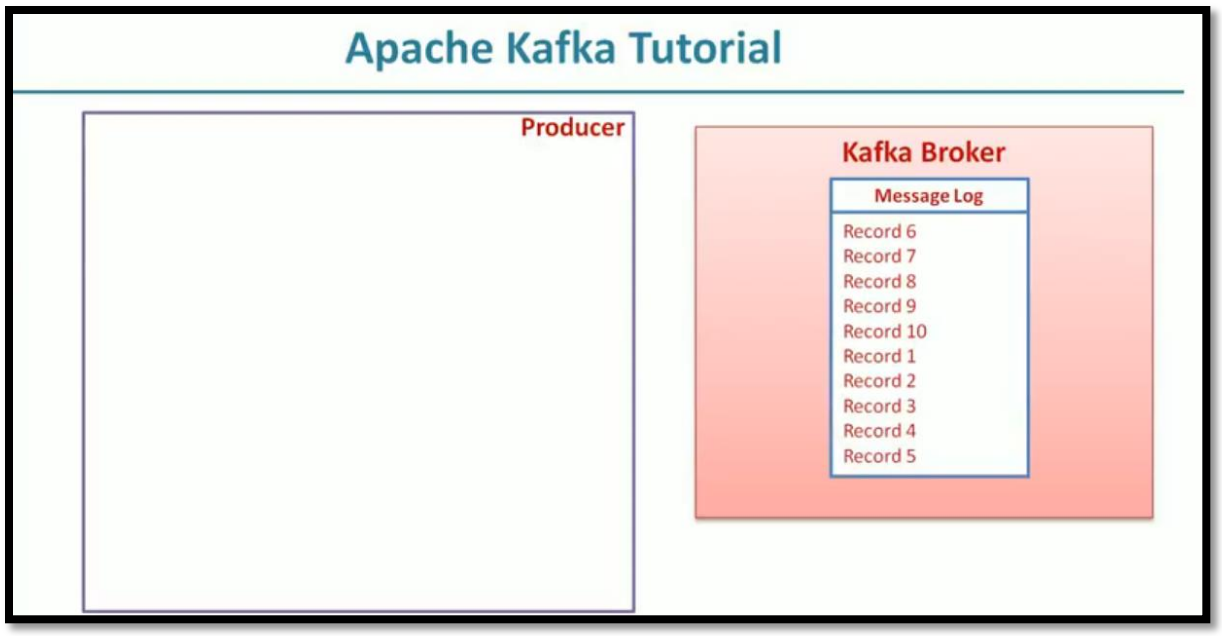
#max.in.flight.requests.per.connection - The maximum number of unacknowledged requests the client will send on a single connection before blocking. Note that if this setting is set to be greater than 1 and there are failed sends, there is a risk of message re-ordering due to retries (i.e., if retries are enabled).

Assume you are delivering two batches to same partition in asynchronous mode

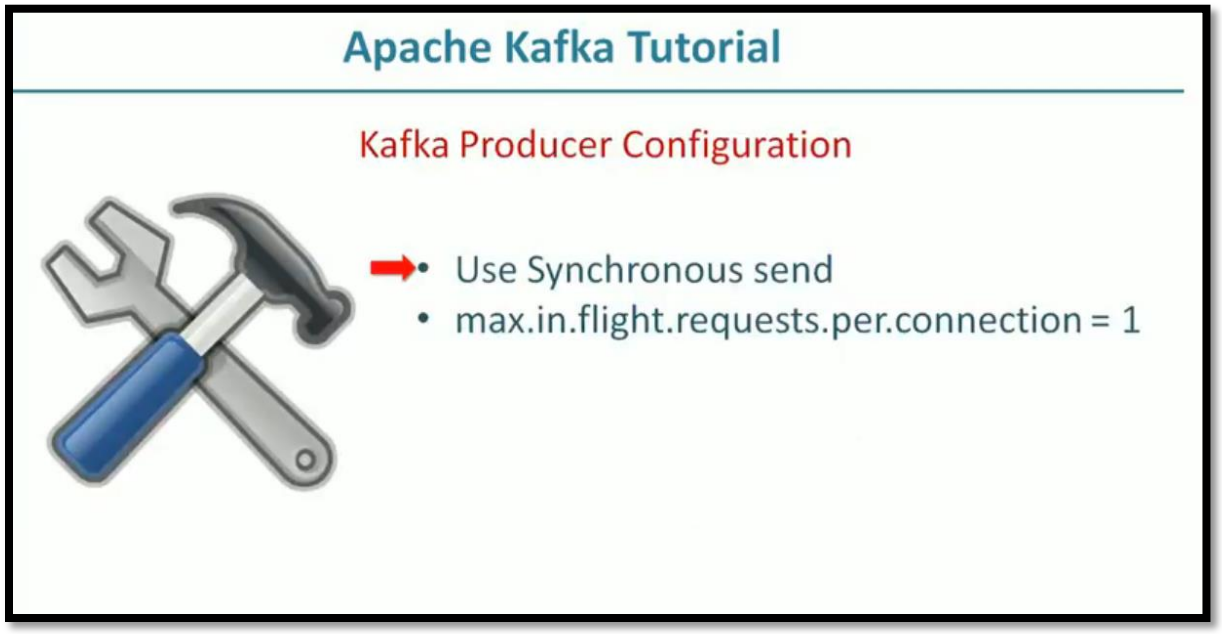




Lets suppose if batch1 fails and batch2 succeed and after retries the batch1 again and if it is successful you loose your order

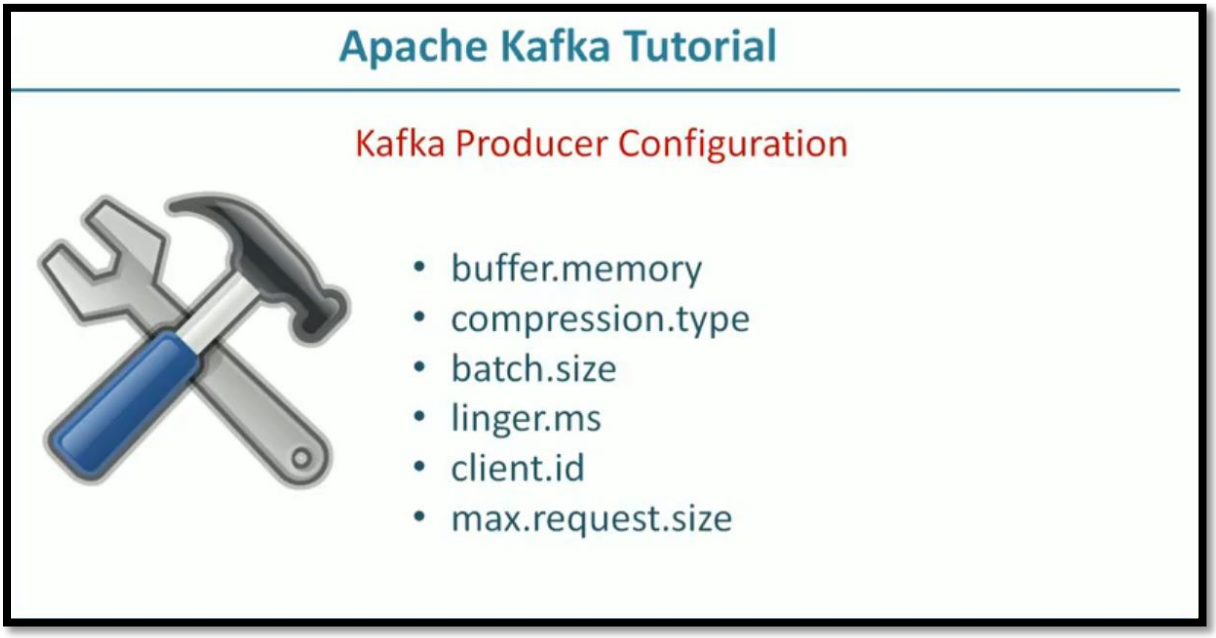


If order is critical you can use following two option

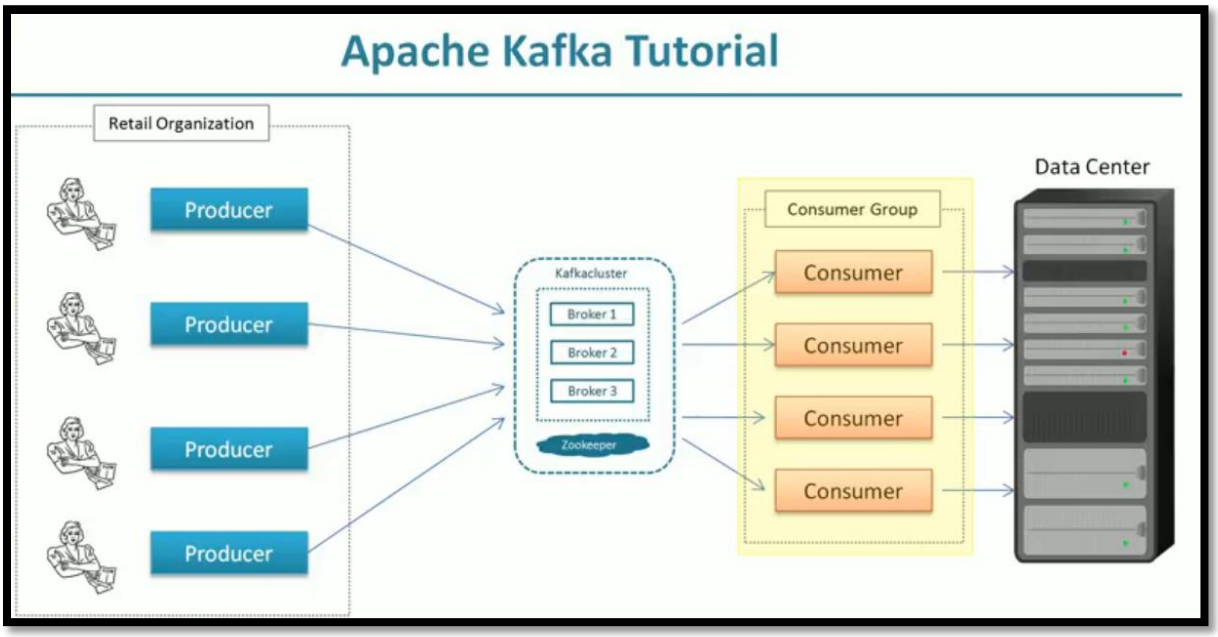


For financial transaction order is critical in that case we can use them

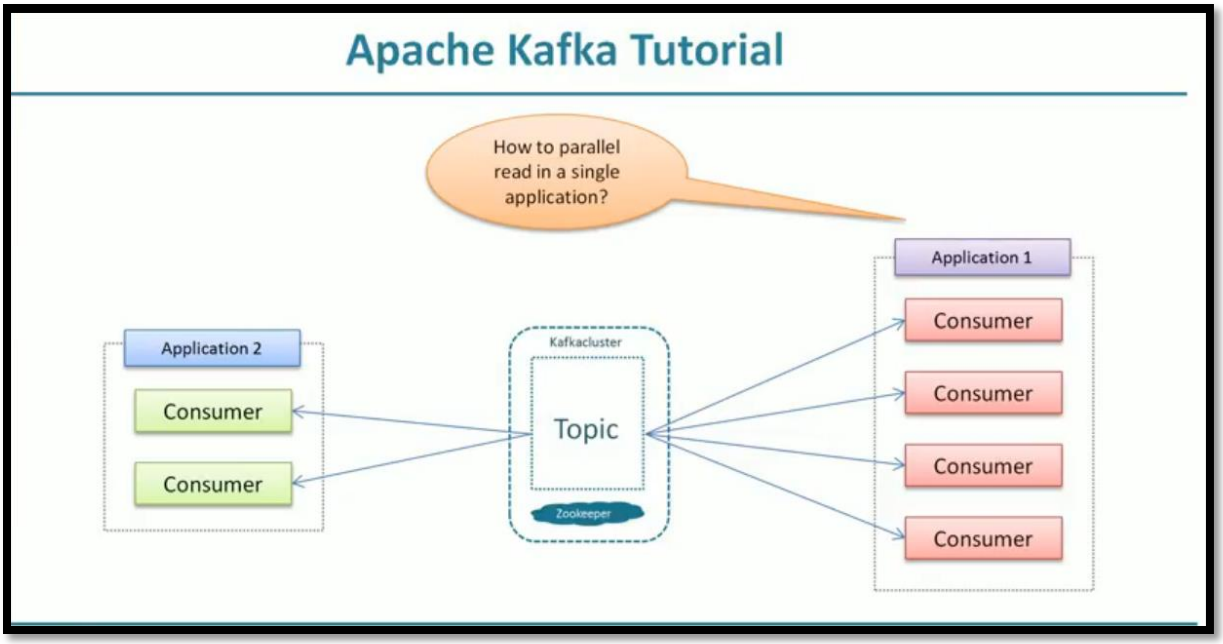
Some more important properties



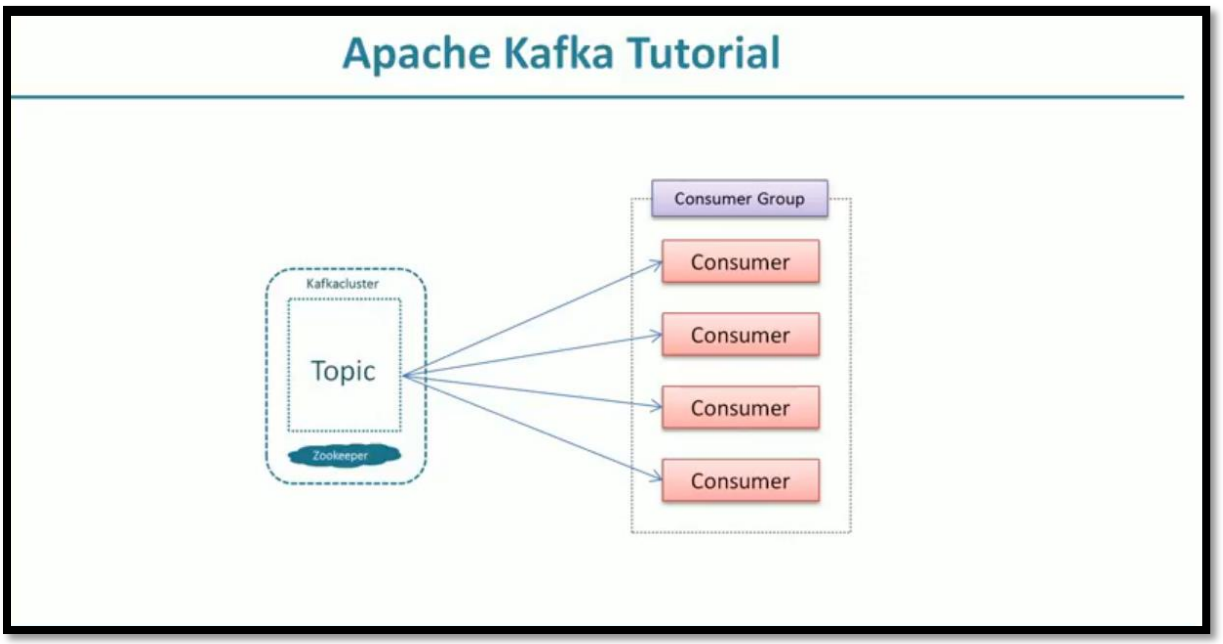
Consumer Group



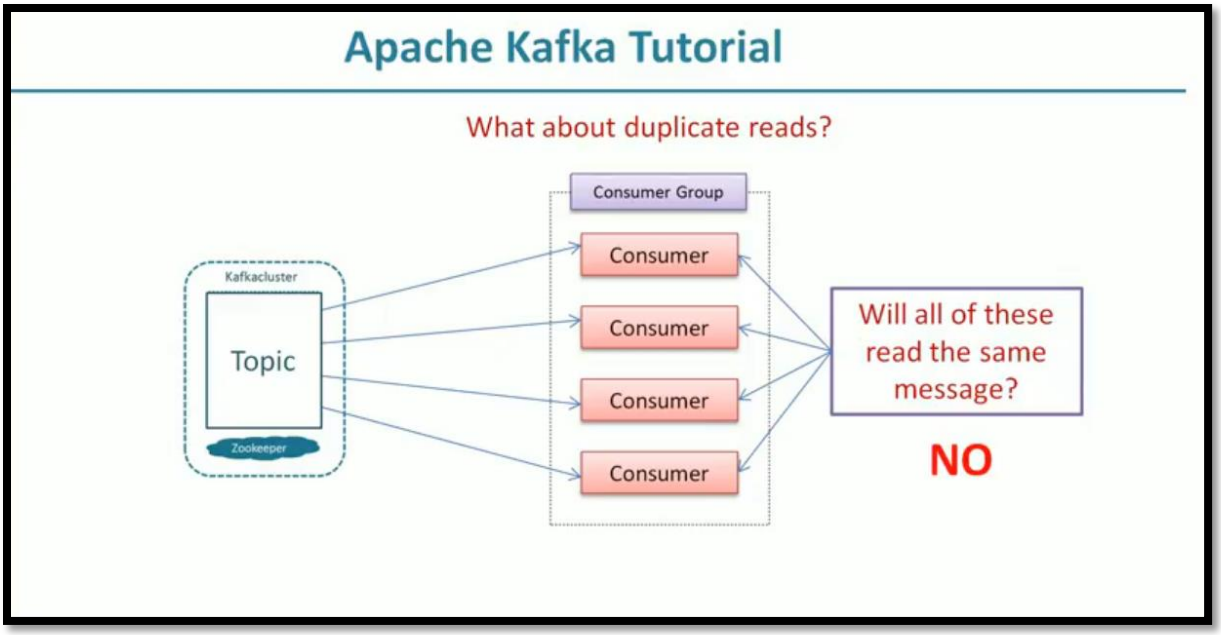
How to we read messages in parallel

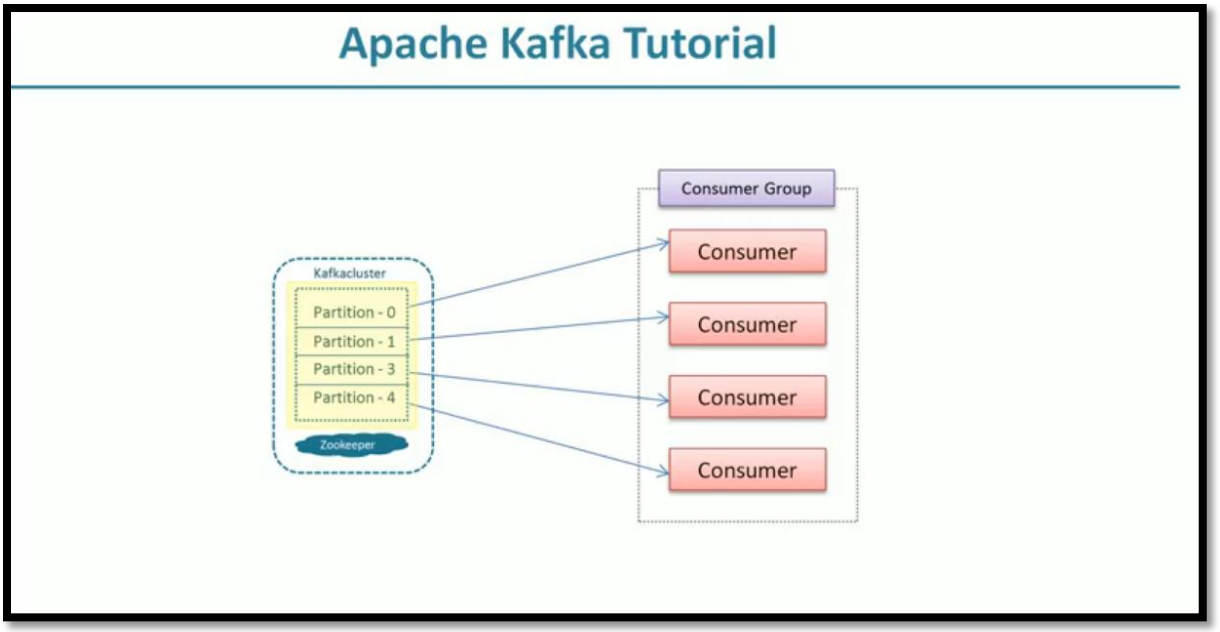


We can read by creating consumer group



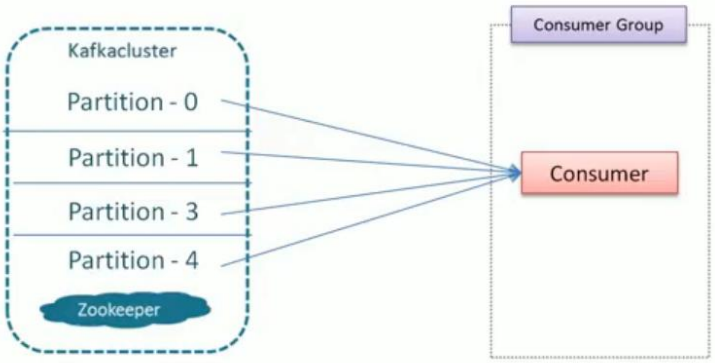
What about duplicate read?



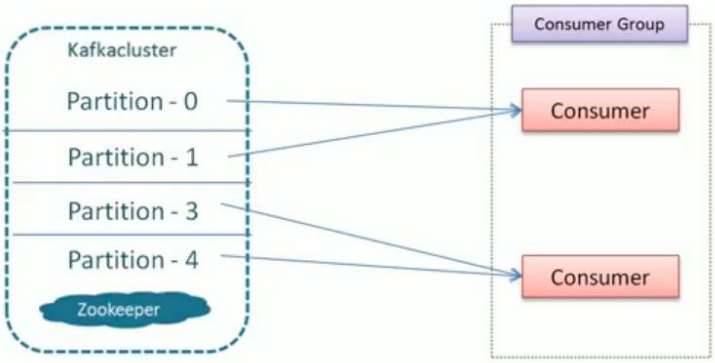


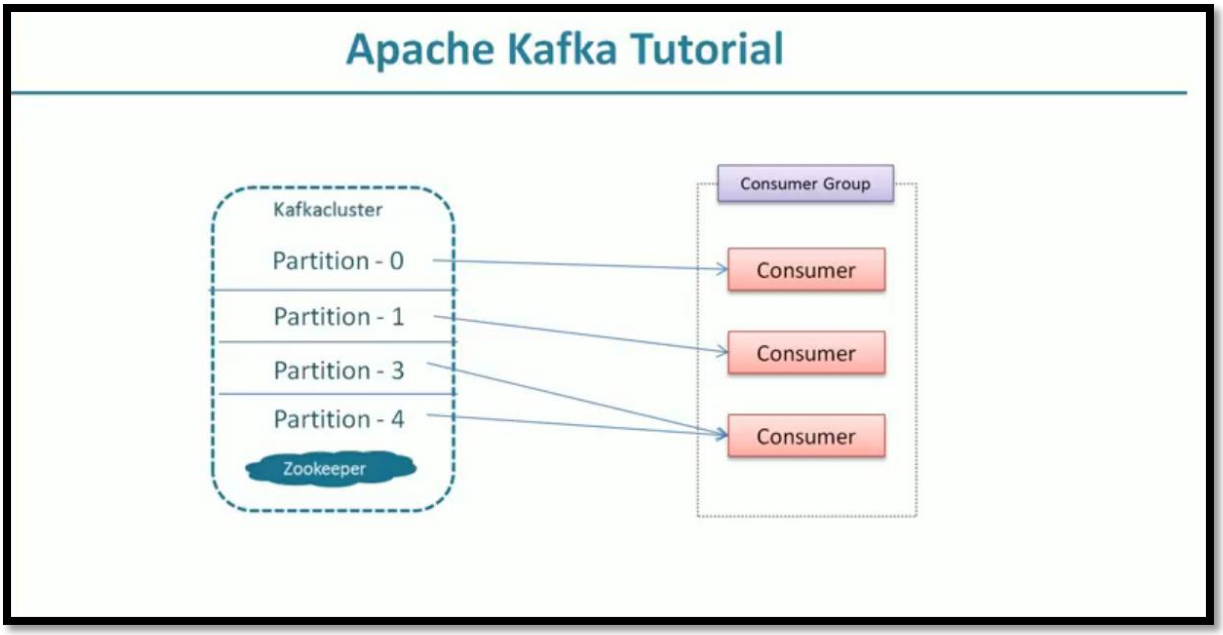
Only one consumer owns a partition at any point in time

Apache Kafka Tutorial

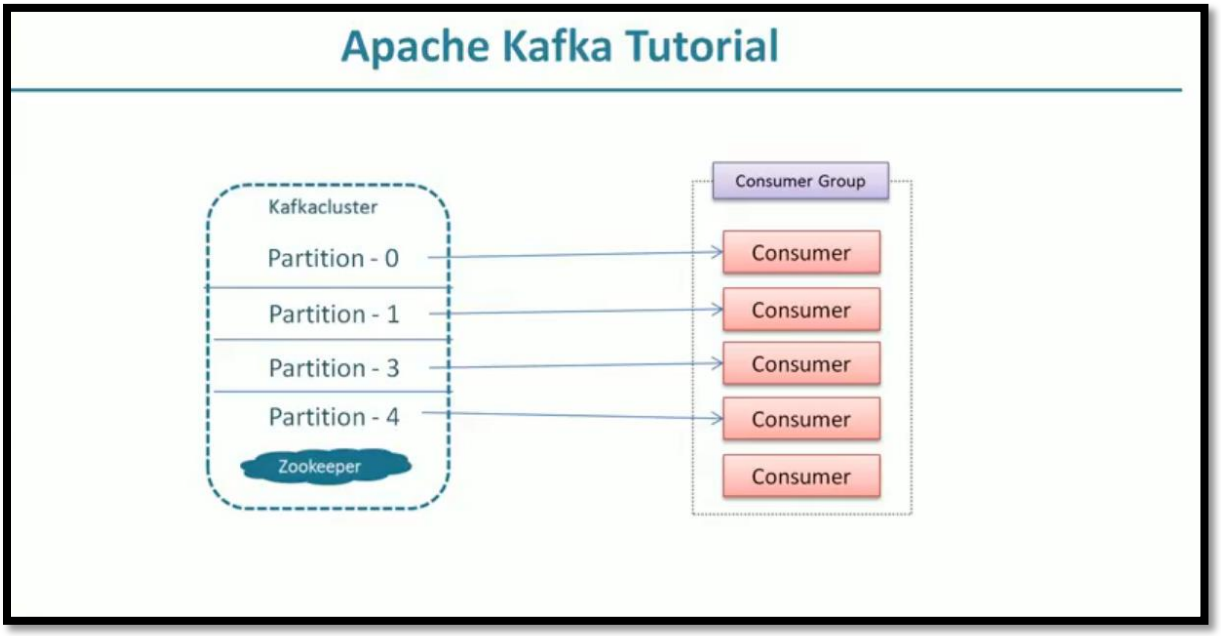


Apache Kafka Tutorial



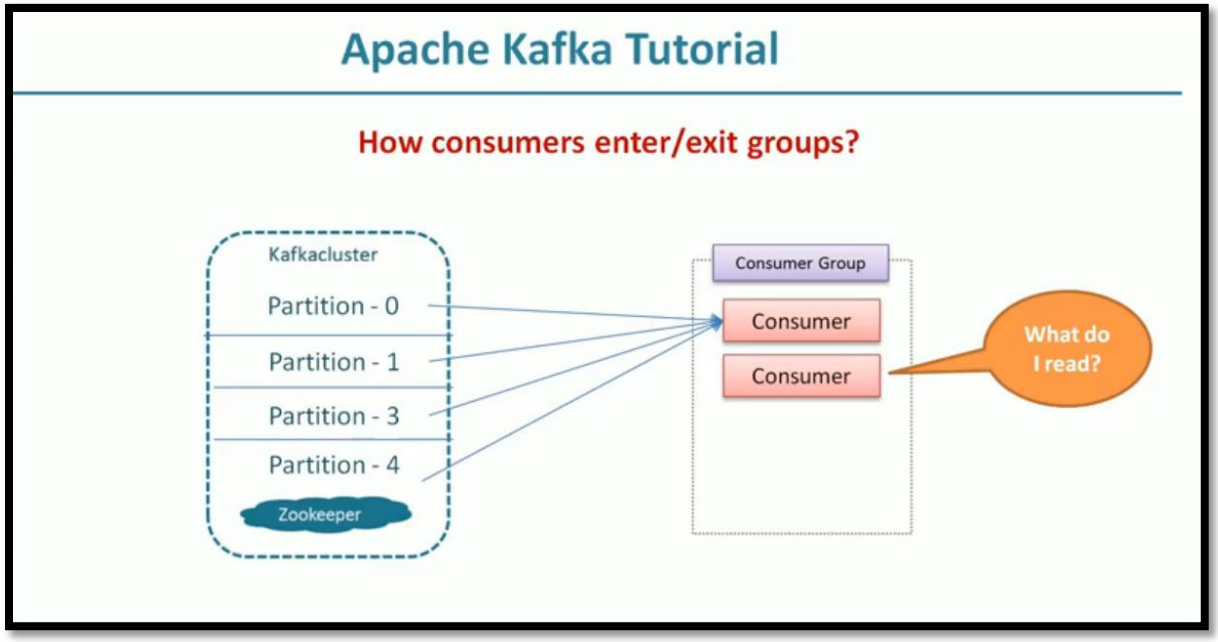


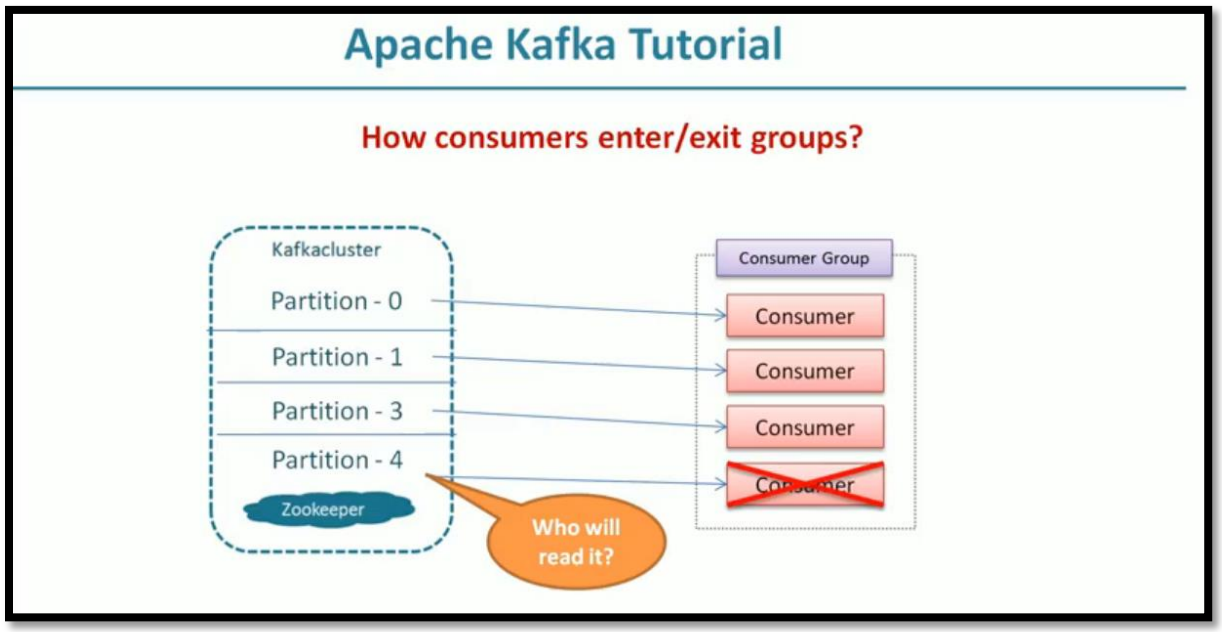
Consumer do not share a partition

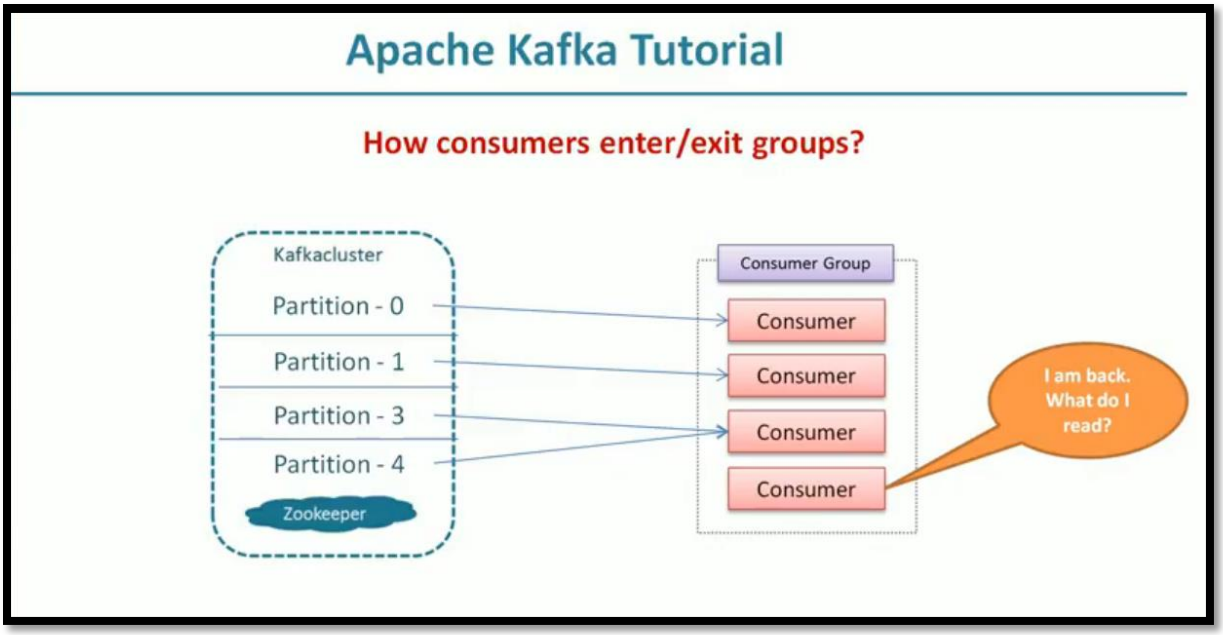


If consumers are greater than partition rest consumer will remain idle

Some problems in assignments/reassignments consumers in a group

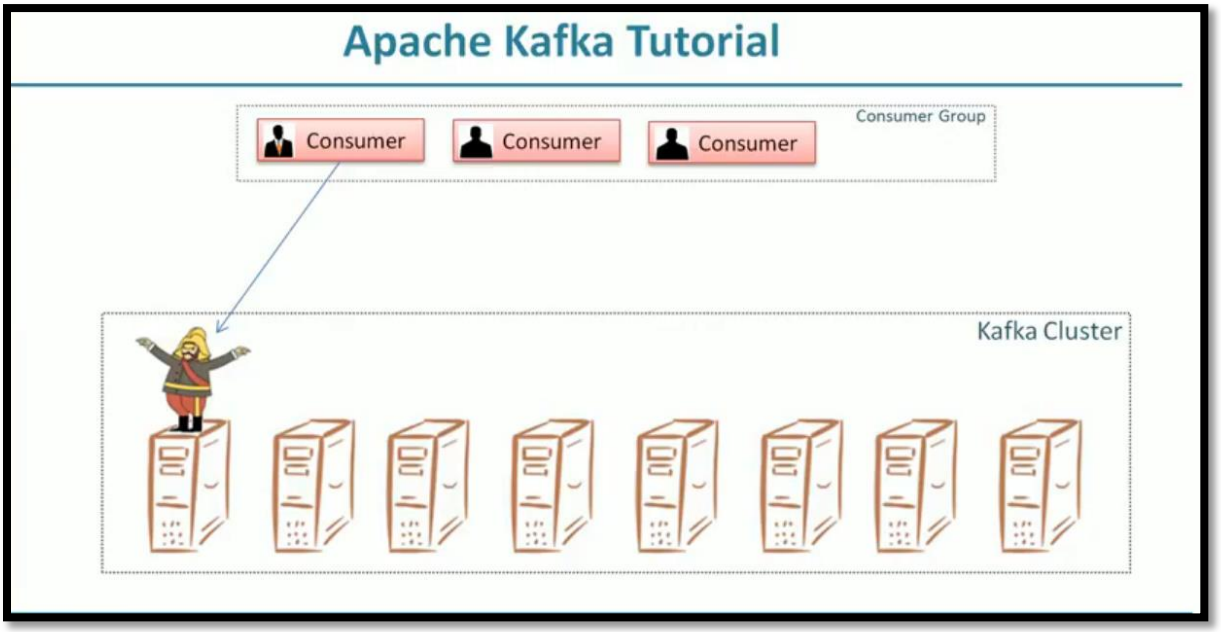






Group coordinator maintain all of these scenario

One of the kafka broker elected as a group coordinator when consumer wants to join a consumer group it sends a request to group coordinator , first consumer join group becomes leader and all other consumer joining later in the group becomes members of the group



Apache Kafka Tutorial

Coordinator

- Manage a list of group members
- Initiates a rebalance activity (Block the read for all members)
- Communicate about new assignment to consumers

Leader

- Executes the rebalance activity
- Sends new partition assignment to coordinator

Apache Kafka Tutorial

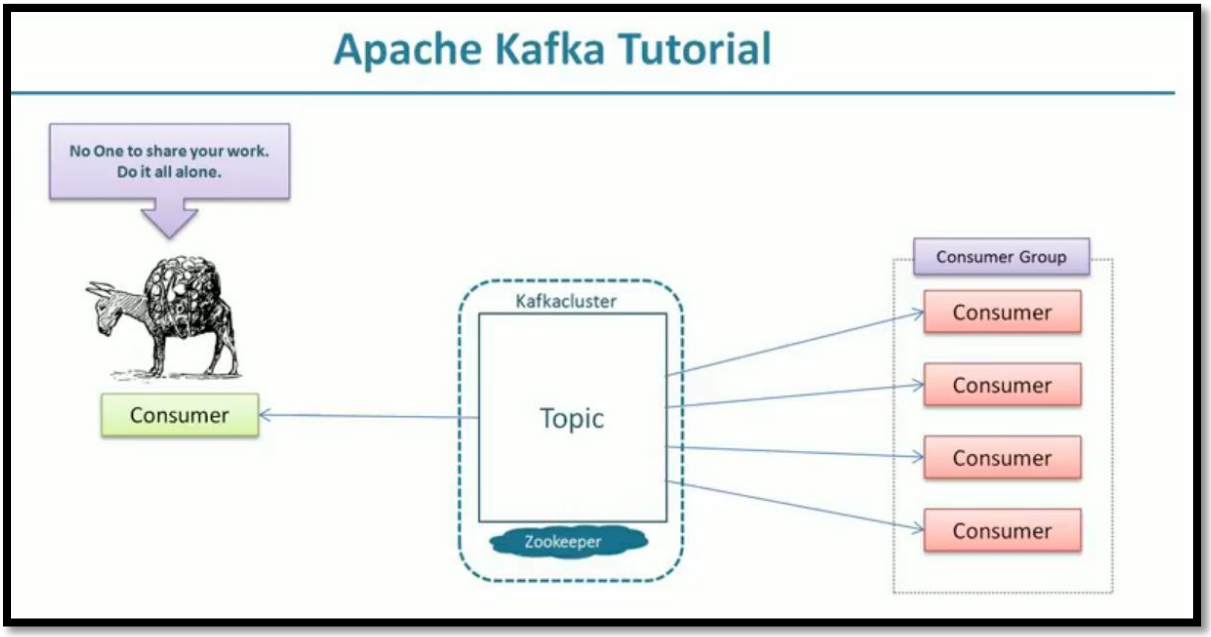
Summary

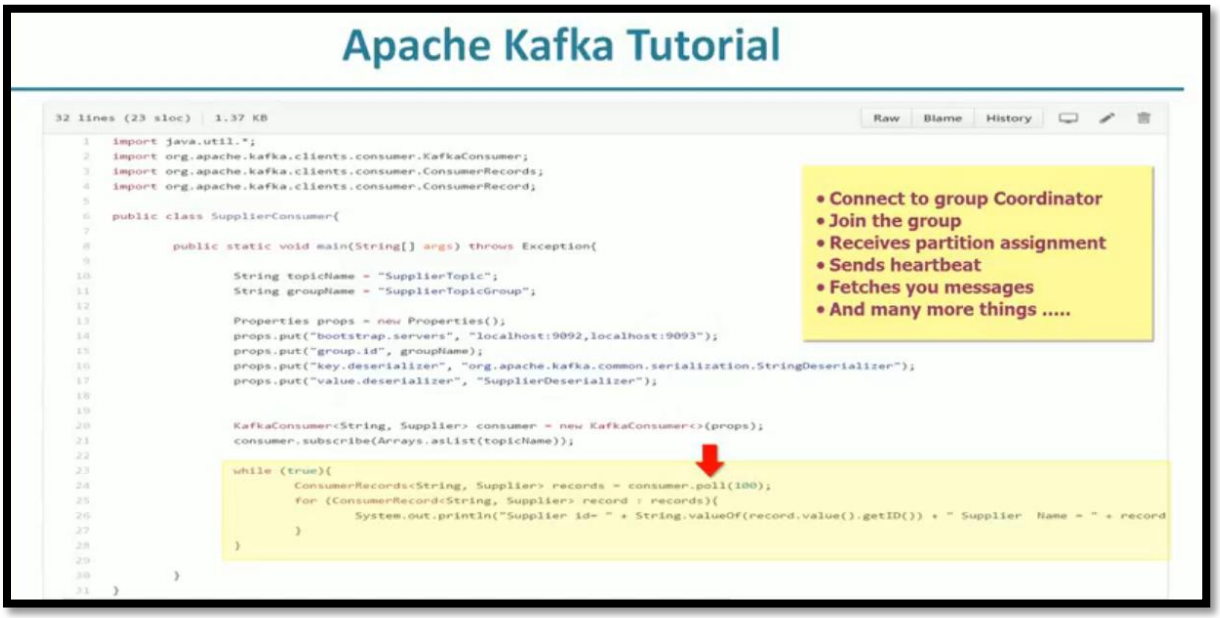
- **Consumers Groups –** used to read and process data in parallel.
- **Partitions –** are never shared among members of same group at the same time.
- **Group Coordinator –** maintains a list of active consumers.
- **Rebalance –** initiated when the list of consumers is modified.
- **Group leader –** executes a rebalance activity.

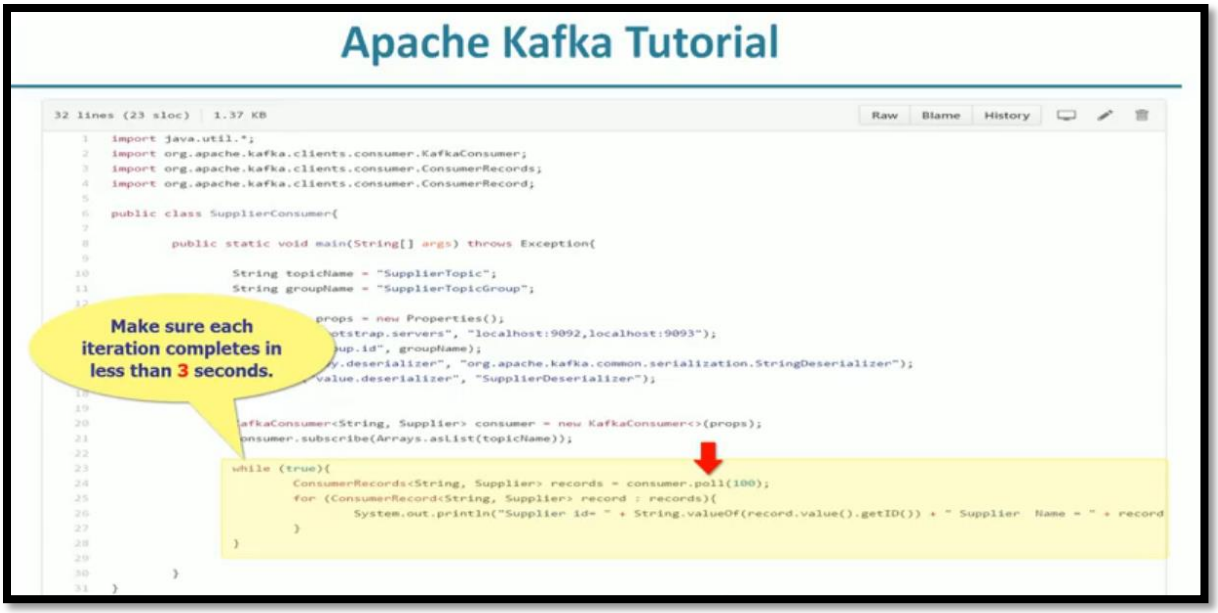
Consumer API

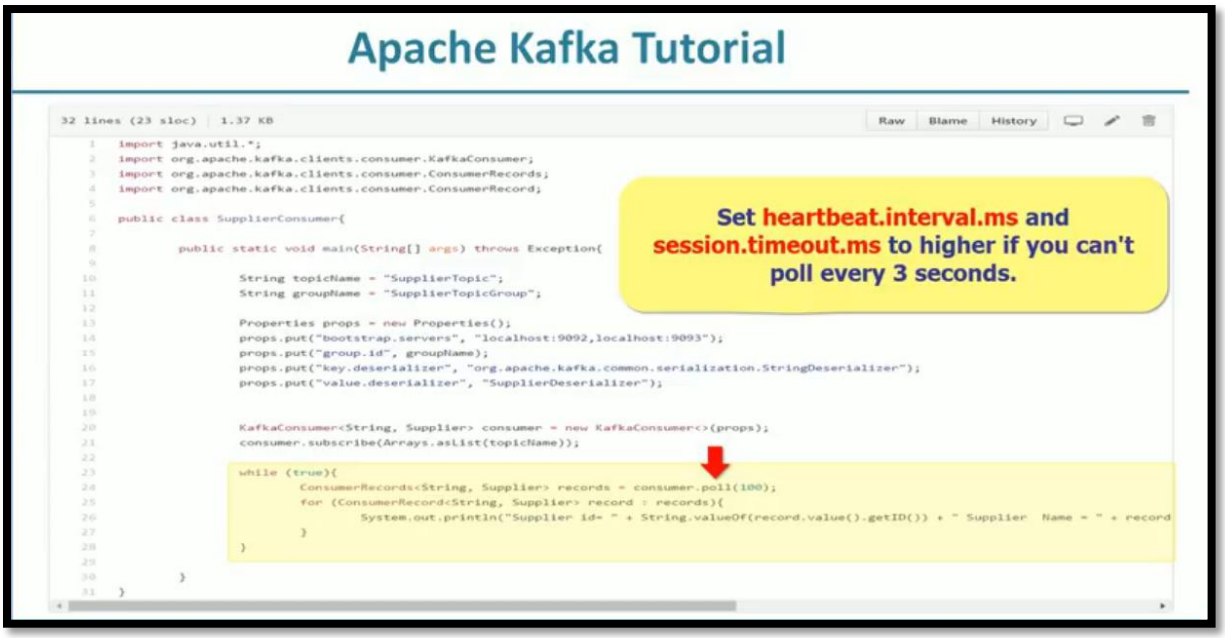
```
import java.util.*;
import org.apache.kafka.clients.consumer.KafkaConsumer;
import org.apache.kafka.clients.consumer.ConsumerRecords;
import org.apache.kafka.clients.consumer.ConsumerRecord;

public class SupplierConsumer{

	public static void main(String[] args) throws Exception{

		String topicName = "SupplierTopic";
		String groupName = "SupplierTopicGroup";

		Properties props = new Properties();
		props.put("bootstrap.servers", "localhost:9092,localhost:9093");
// consumer group name as the value of this property , we need not to create consumer group , we just need to specify the name of the group, if we don't specify this property, this consumer alone will process all the message for this topic
		props.put("group.id", groupName);
		props.put("key.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");
		props.put("value.deserializer", "SupplierDeserializer");


		KafkaConsumer<String, Supplier> consumer = new KafkaConsumer<>(props);
// consumer can subscribe to multiple topics
		consumer.subscribe(Arrays.asList(topicName));

		while (true){
		   // 100 is timeout
			ConsumerRecords<String, Supplier> records = consumer.poll(100);
			for (ConsumerRecord<String, Supplier> record : records){
				System.out.println("Supplier id= " + String.valueOf(record.value().getID()) + " Supplier Name = " + record.value().getName() + " Supplier Start Date = " + record.value().getStartDate().toString());
			}
		}

	}
}
```

If consumer is not part of consumer group









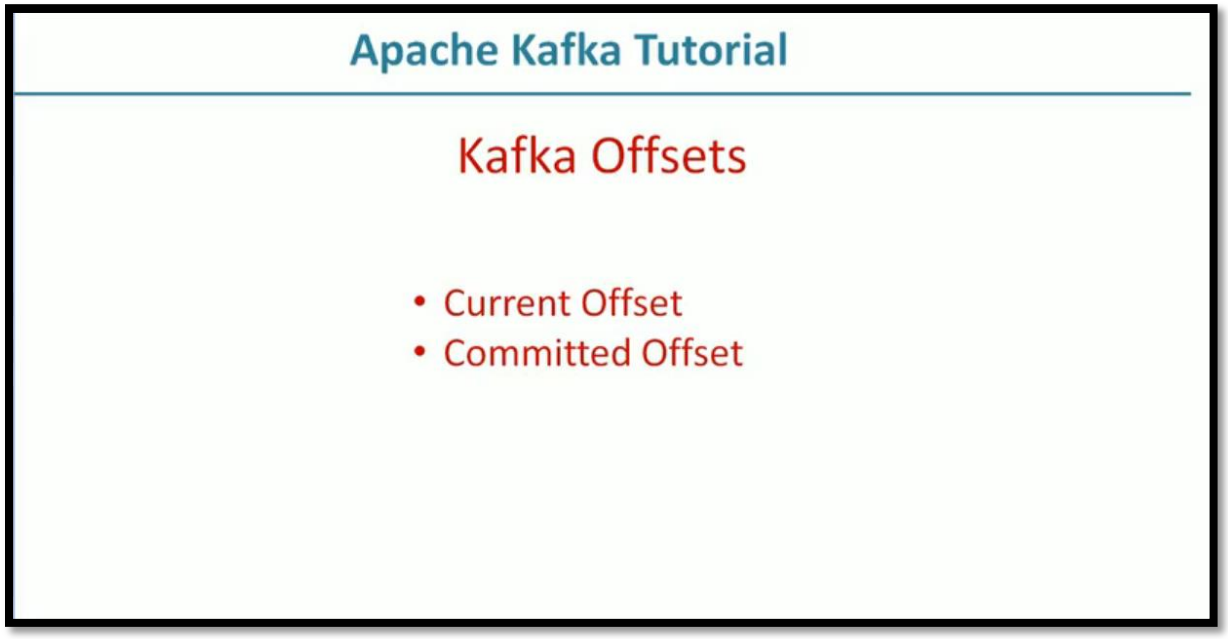
We can also use scheduler to start the consumer instead of infinite while loop

Ideally we should not put properties inside code we should maintain a property file and load that file in code

```
import java.util.*;
import java.io.*;
import org.apache.kafka.clients.consumer.KafkaConsumer;
import org.apache.kafka.clients.consumer.ConsumerRecords;
import org.apache.kafka.clients.consumer.ConsumerRecord;
```

```
public class NewSupplierConsumer{

   public static void main(String[] args) throws Exception{

      String topicName = "SupplierTopic";
      String groupName = "SupplierTopicGroup";
      Properties props = new Properties();
      //props.put("bootstrap.servers", "localhost:9092,localhost:9093");
      //props.put("group.id", groupName);
      //props.put("key.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");
      //props.put("value.deserializer", "SupplierDeserializer");

      InputStream input = null;
      KafkaConsumer<String, Supplier> consumer = null;

      try {
         input = new FileInputStream("SupplierConsumer.properties");
         props.load(input);
         consumer = new KafkaConsumer<>(props);
         consumer.subscribe(Arrays.asList(topicName));

         while (true){
            ConsumerRecords<String, Supplier> records = consumer.poll(100);
            for (ConsumerRecord<String, Supplier> record : records){
               System.out.println("Supplier id= " + String.valueOf(record.value().getID()) + " Supplier  Name
= " + record.value().getName() + " Supplier Start Date = " + record.value().getStartDate().toString());
            }
         }
      }catch(Exception ex){
         ex.printStackTrace();
      }finally{
         input.close();
         consumer.close();
      }
   }
}
```

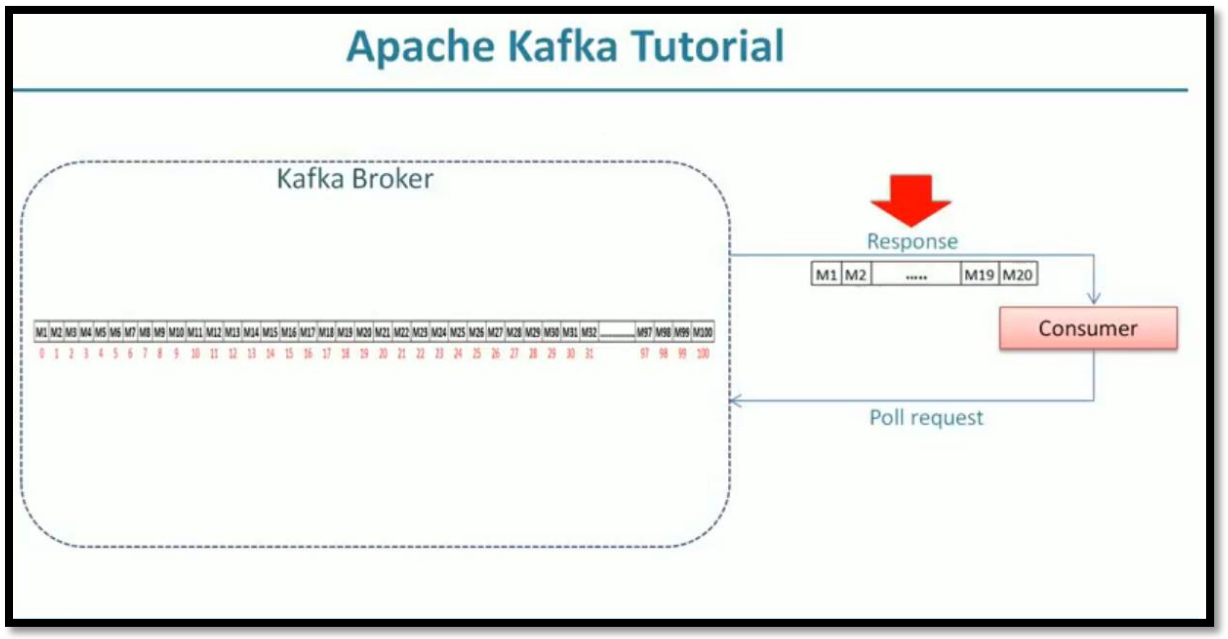
Offset Management

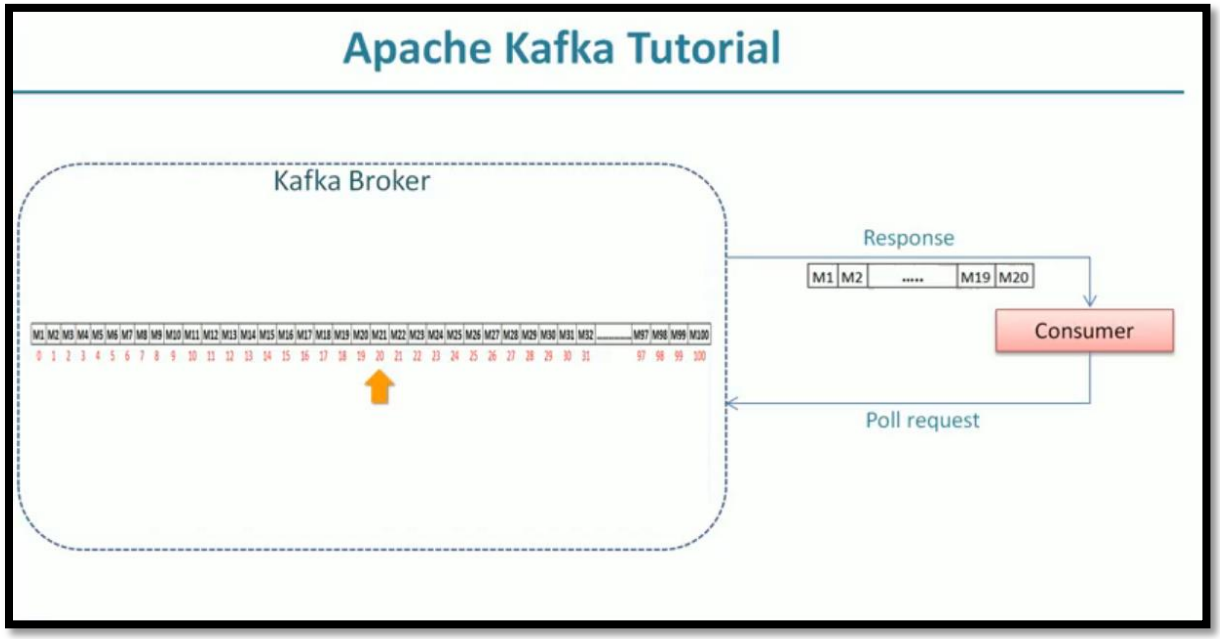


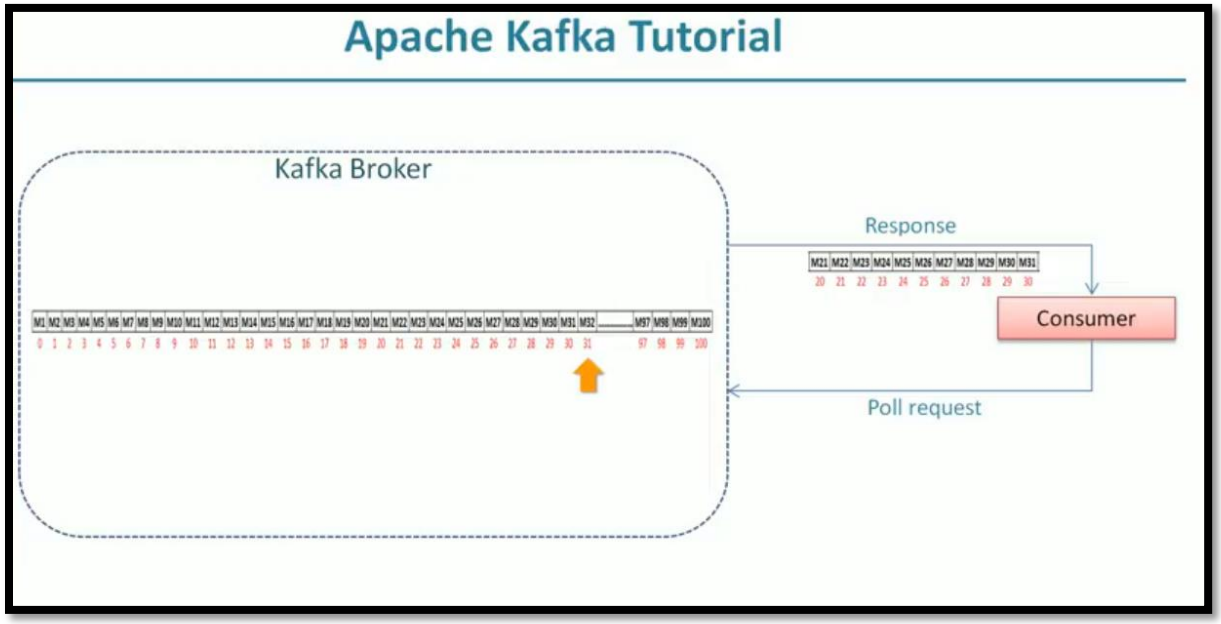
Current offset - is used to avoid resending same record again to the same consumer

Committed Offset – is used to avoid resending the same record to new consumer in the event of partition rebalance.

Kafka manages current and committed offset for each consumer




Apache Kafka Tutorial
Kafka Broker
Response
M1 M2 ..... M19 M20
Consumer
Poll request

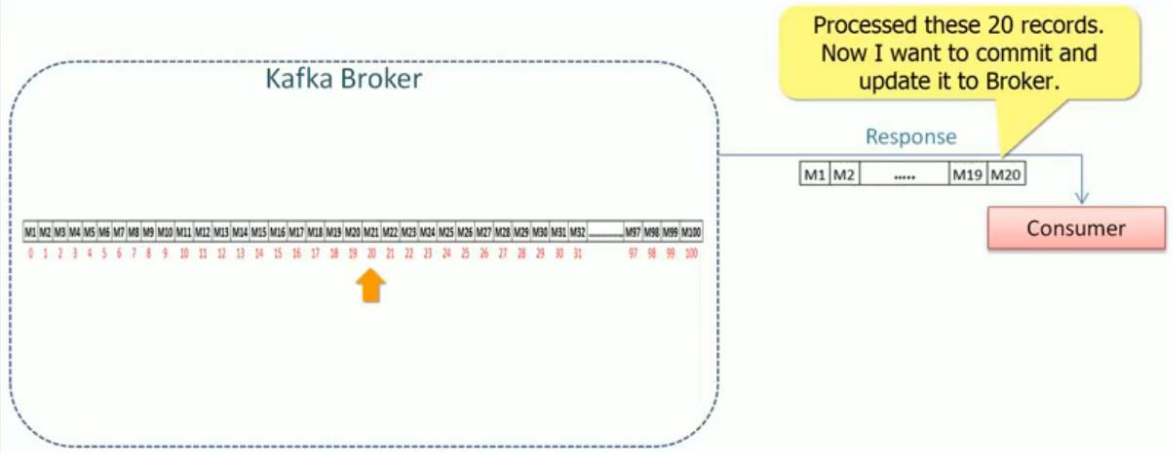

Apache Kafka Tutorial
Kafka Broker
Response
M21 M22 M23 M24 M25 M26 M27 M28 M29 M30 M31
Consumer
Poll request

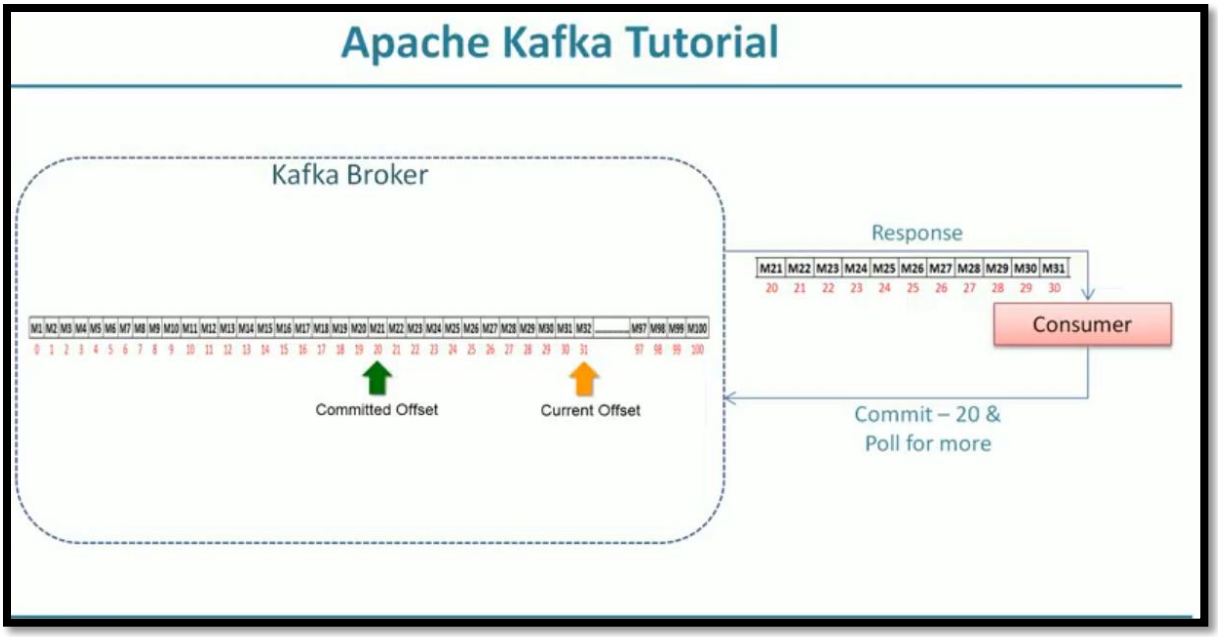
Apache Kafka Tutorial

Committed Offset

The offset position that is already processed by a consumer.

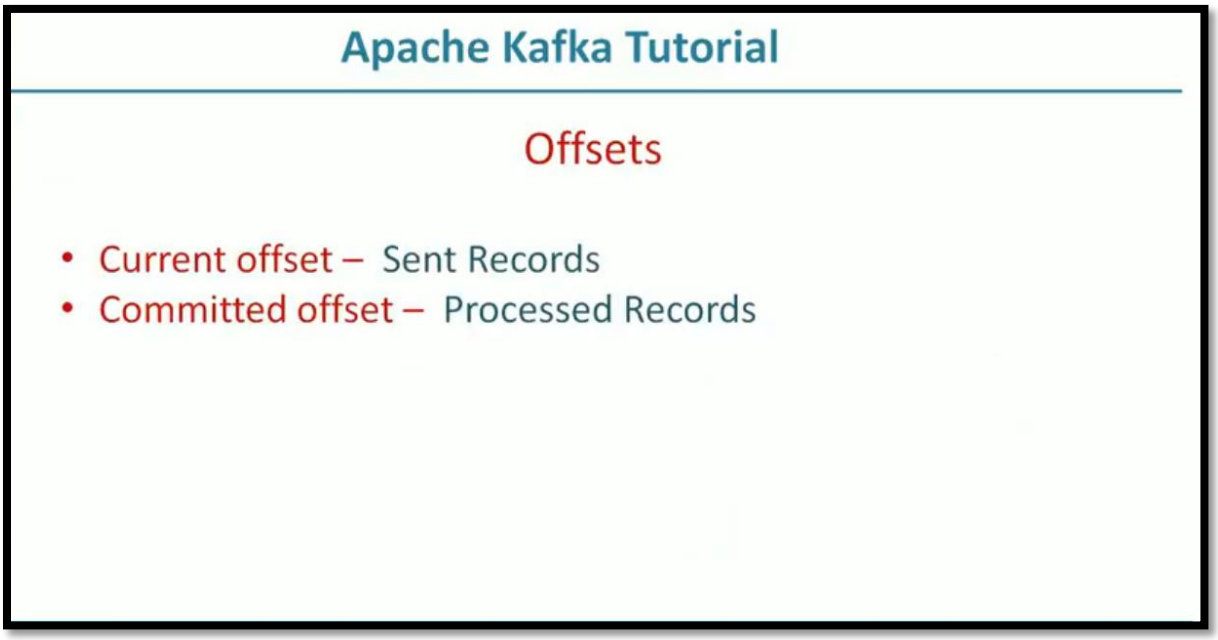
Apache Kafka Tutorial





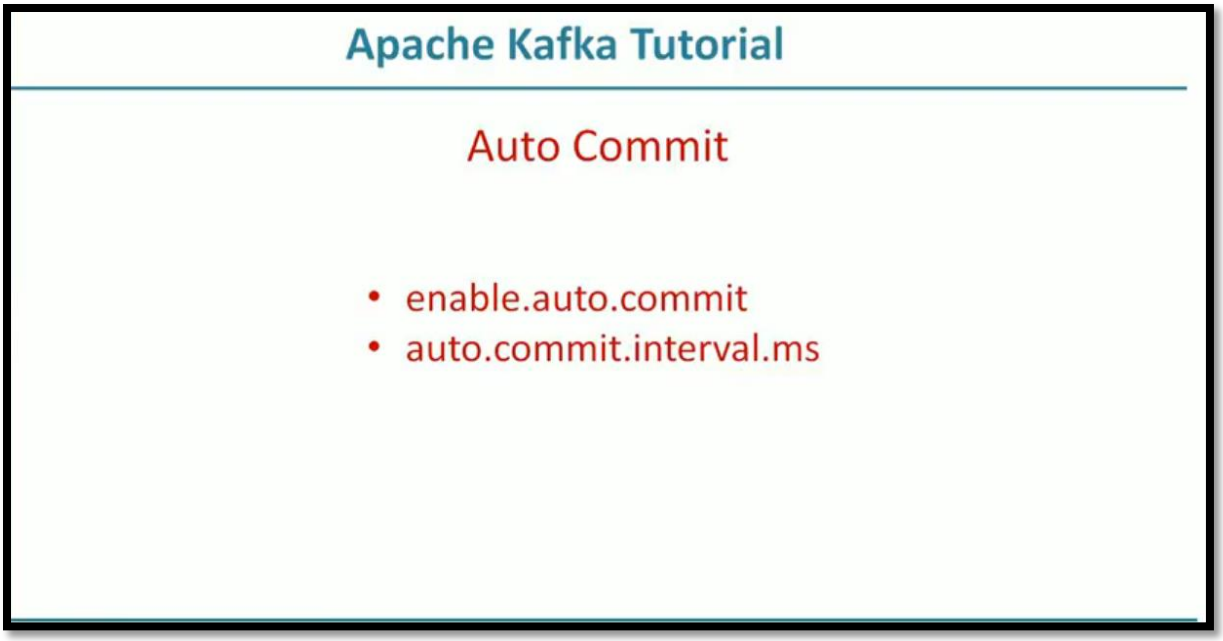
Committed offset is the last record that consumer has successfully processed

In the event of rebalancing when a new consumer assign the same partition, it should now where to start reading means what records are already processed by the previous owner



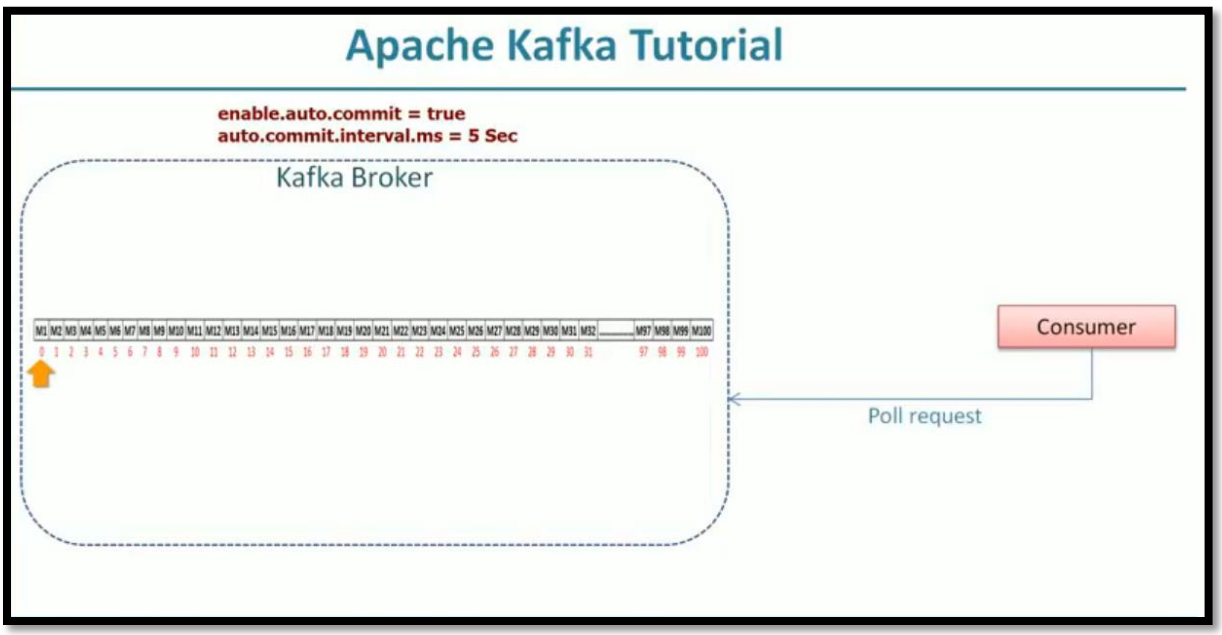
How to commit an offset?

- Auto Commit
- Manual Commit

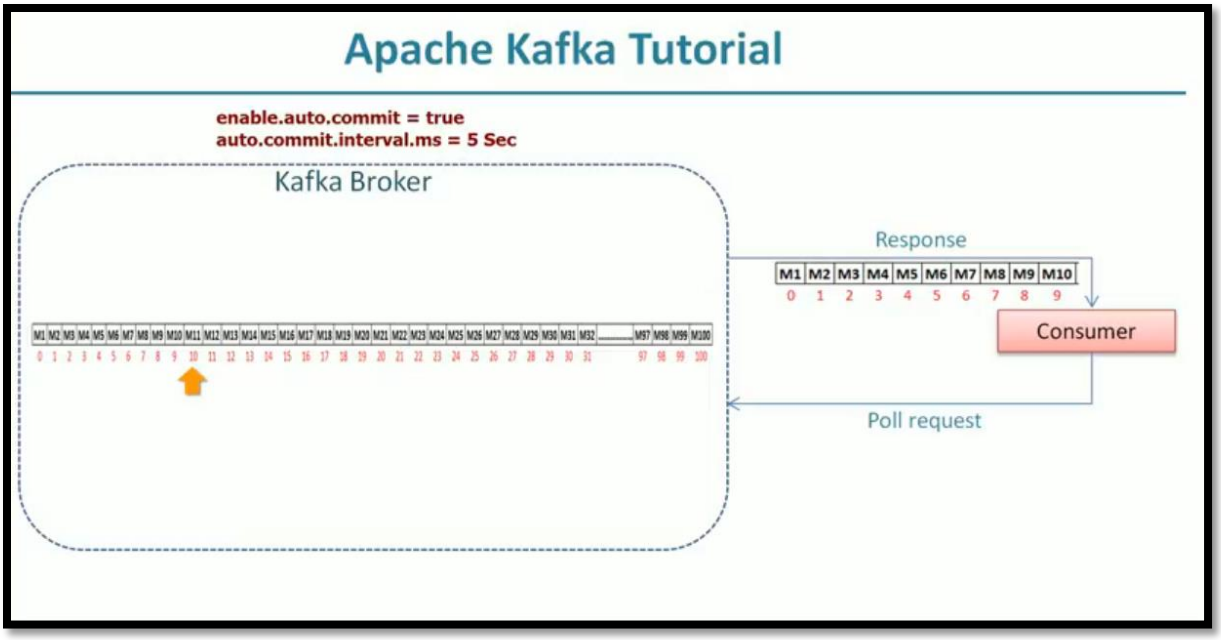


enable.auto.commit is by default set to true

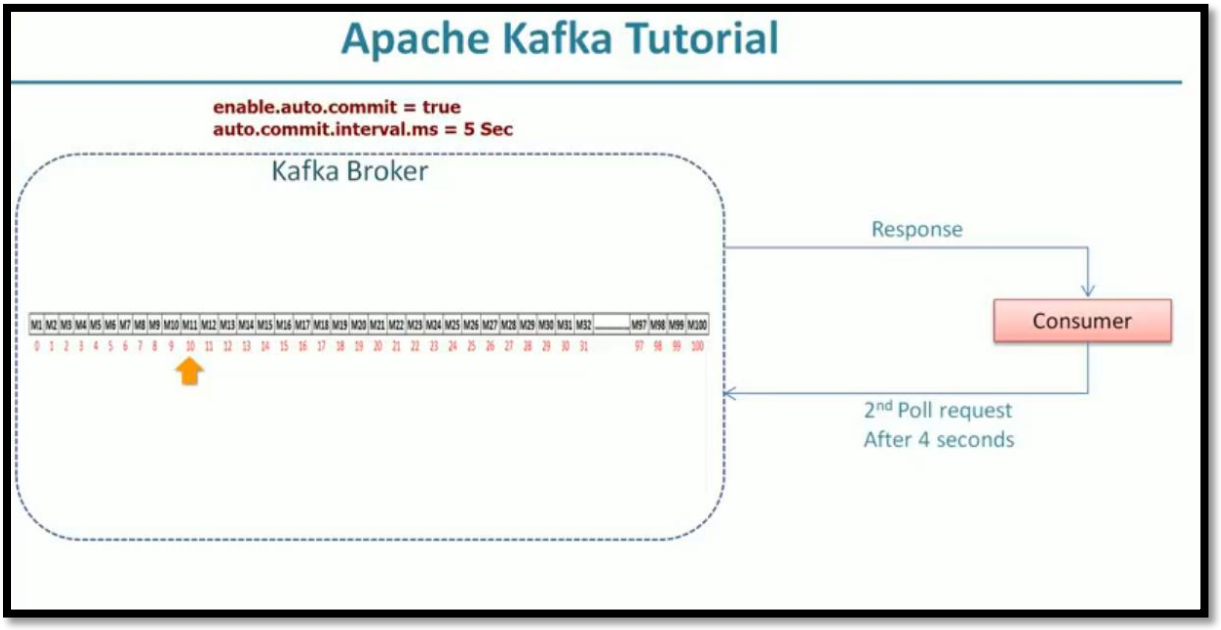
auto.commit.interval.ms defines interval of auto commit default value is 5 sec



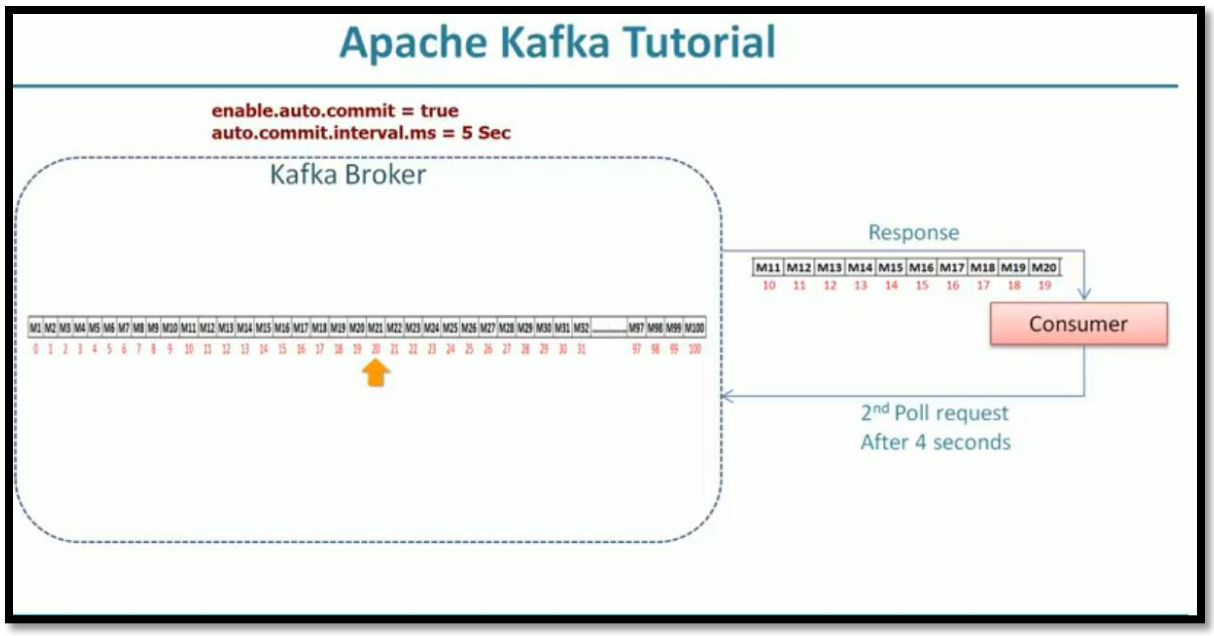
You receive 10 messages and consumer increases the offset to 10



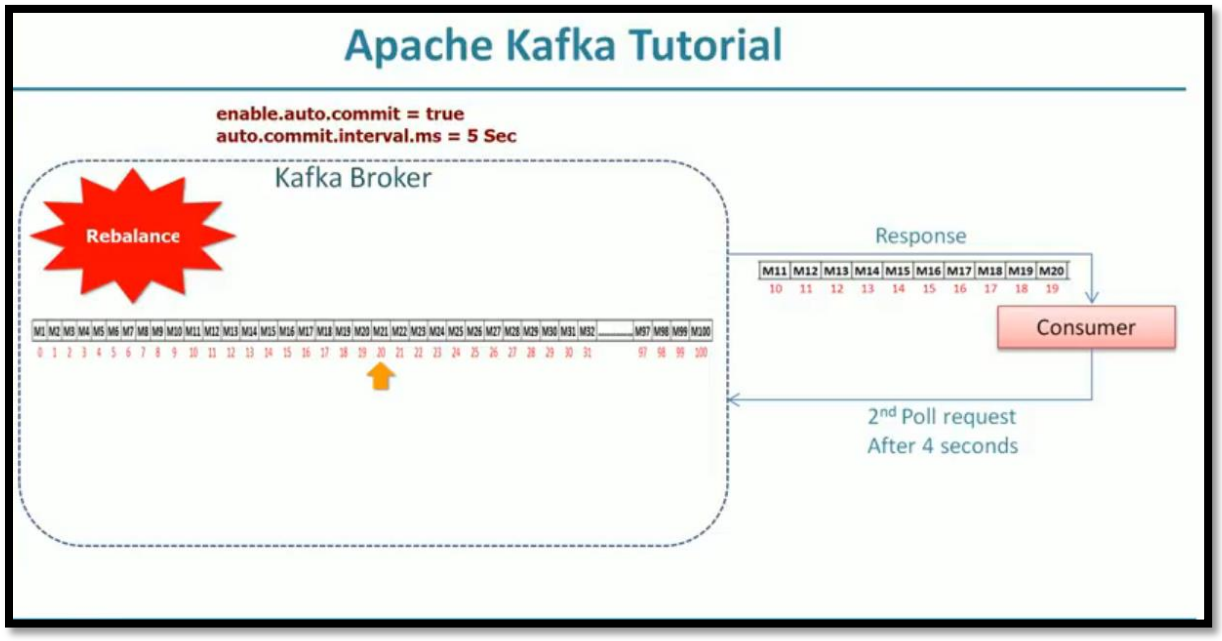
You take 4 seconds to process the message and make a new call



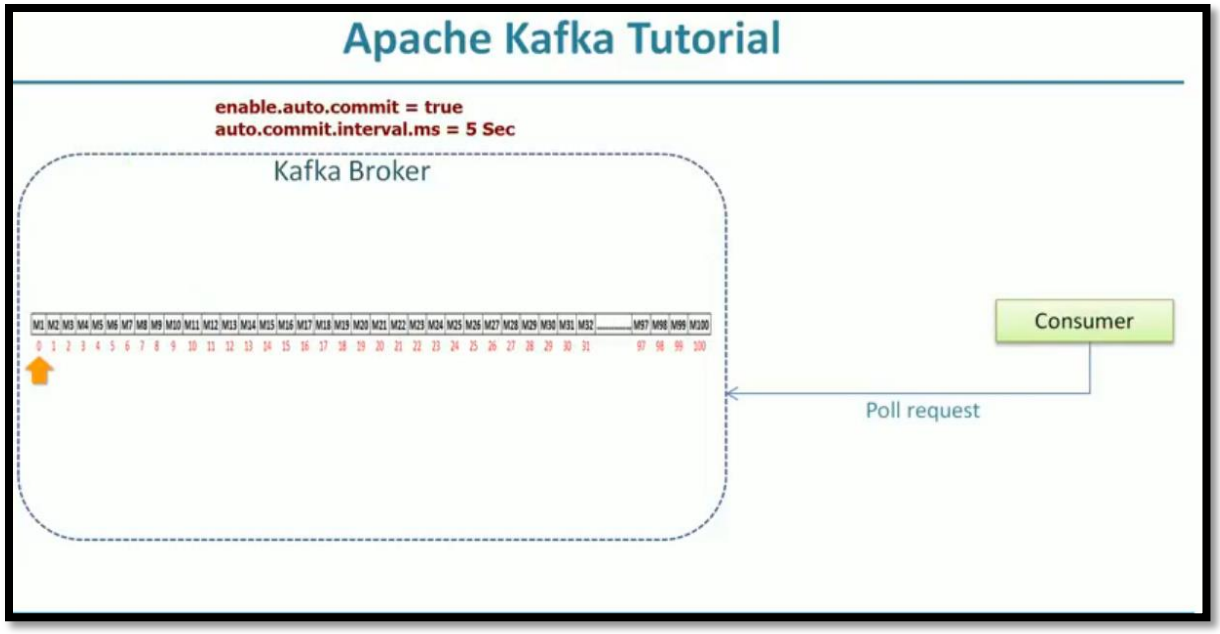
Since you haven't passed five seconds consumer will not commit the offset , you receive another set of records



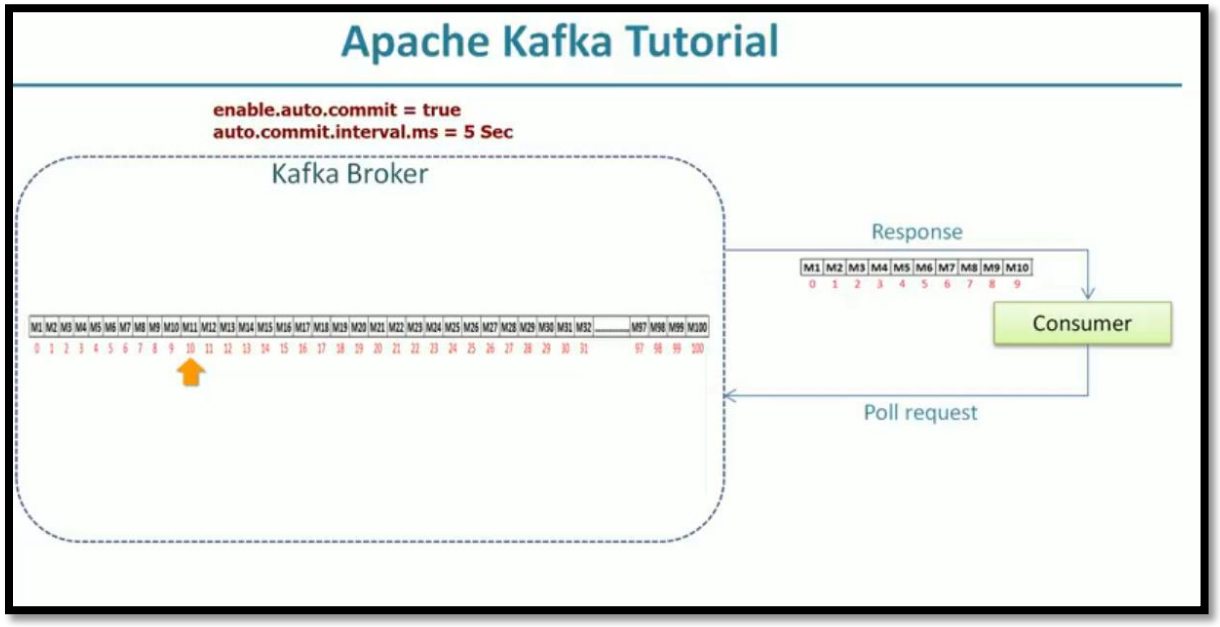
For some reason rebalance occurs at this movement



In this scenario first 10 records are already processed but nothing is committed yet



Now partition goes to a different consumer because of rebalance event since we don't have the committed offset the new owner of this partition start processing from the beginning and process first 10 records once again



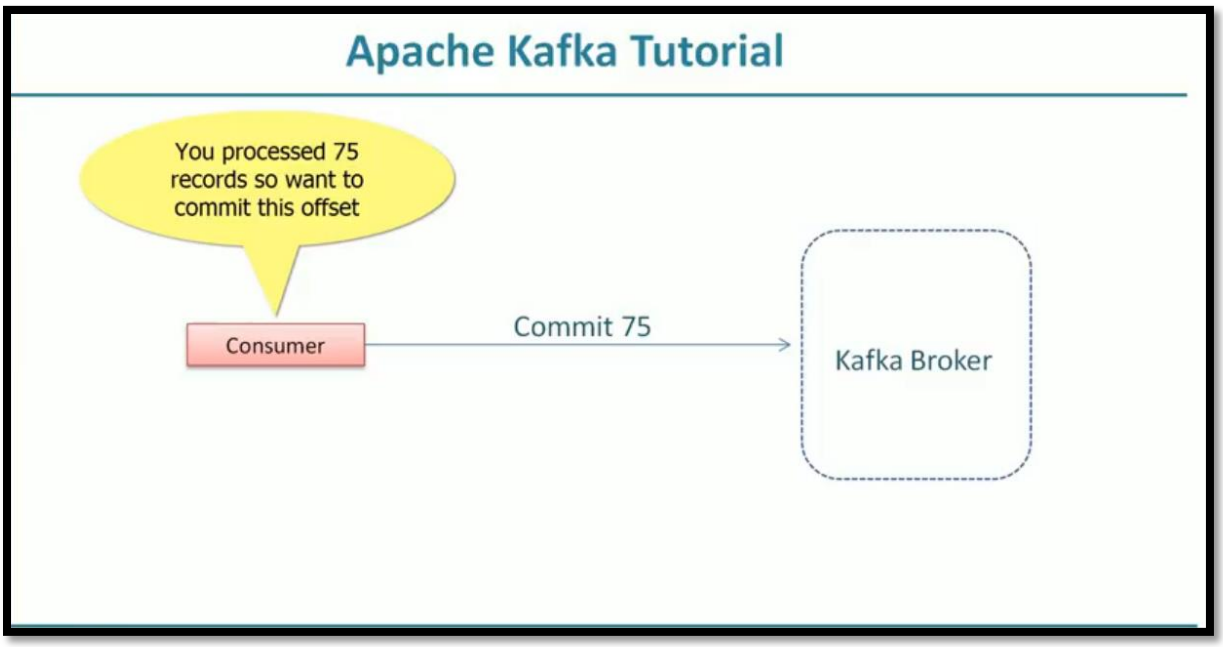
The solution to this particular problem is the manual commit set auto commit to false and manual commit after processing the records

Manual commit has two approaches

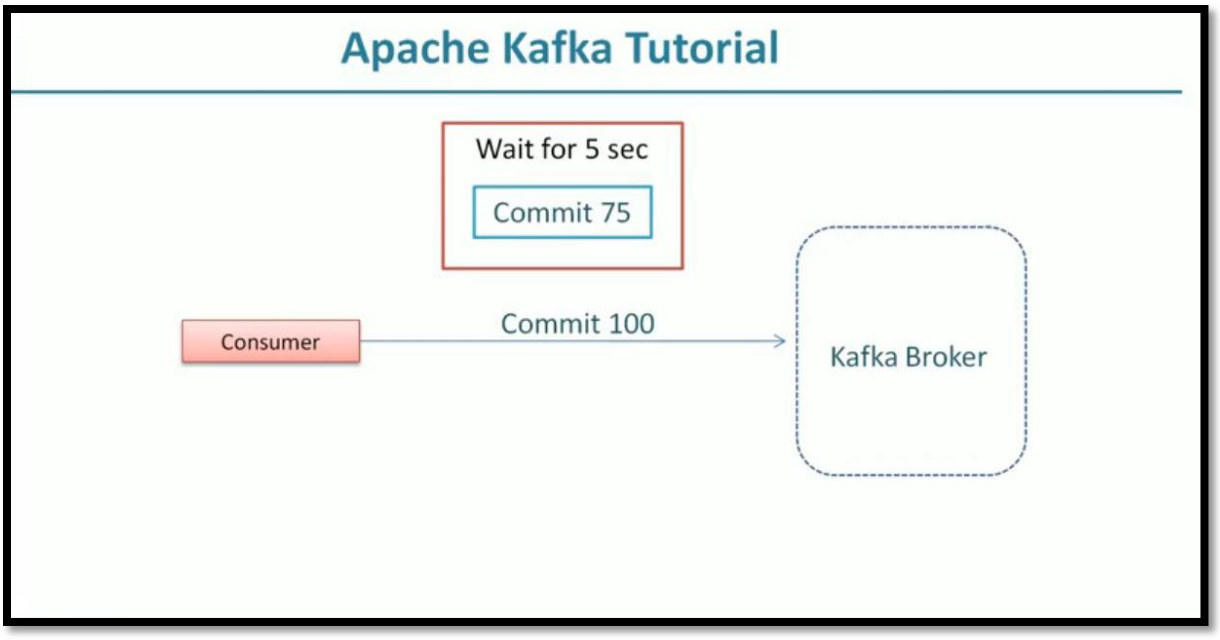
- Commit Sync – blocking method it will block your call for completing a commit operation it will also retries if there are any recoverable errors like rebalancing
- Commit Async – unblocking method but commit Async will not retries

Why Aycn Commit won’t retry ?

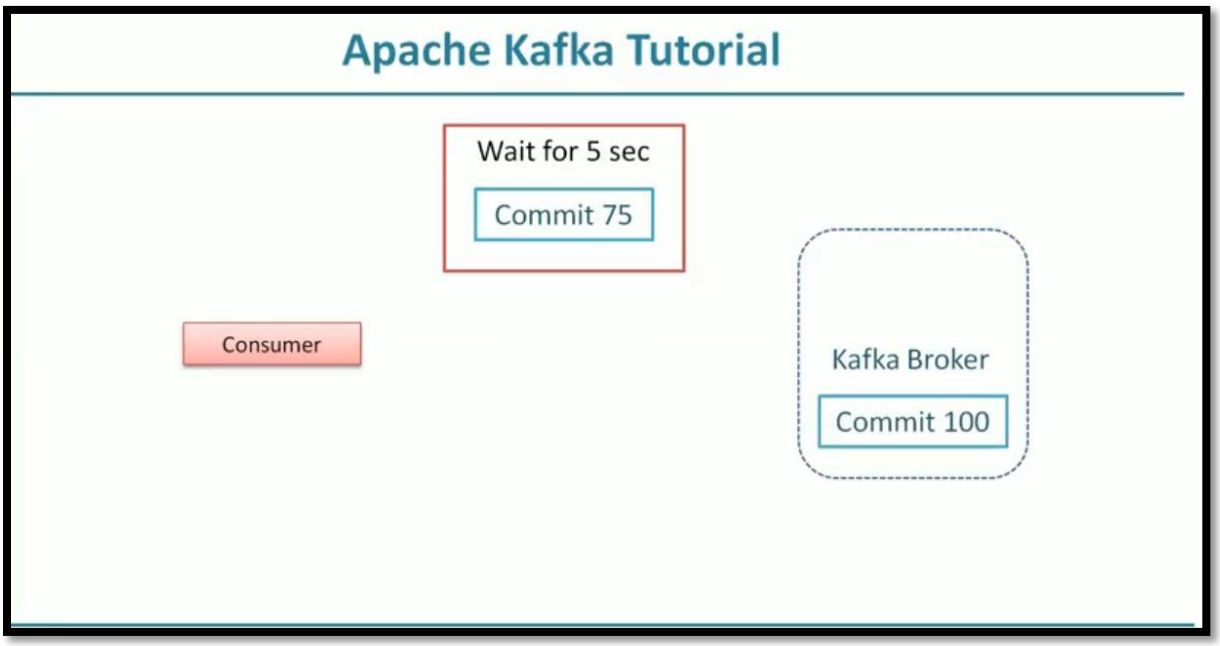
Assume you are trying to commit offset as 75 , it failed for some recoverable reason and you want to retry it after few seconds



Since this was an asynchronous call so without knowing that your previous commit is waiting you initiated another commit , this time it is to commit offset 100



Your commit 100 is successful while commit 75 is waiting for retry

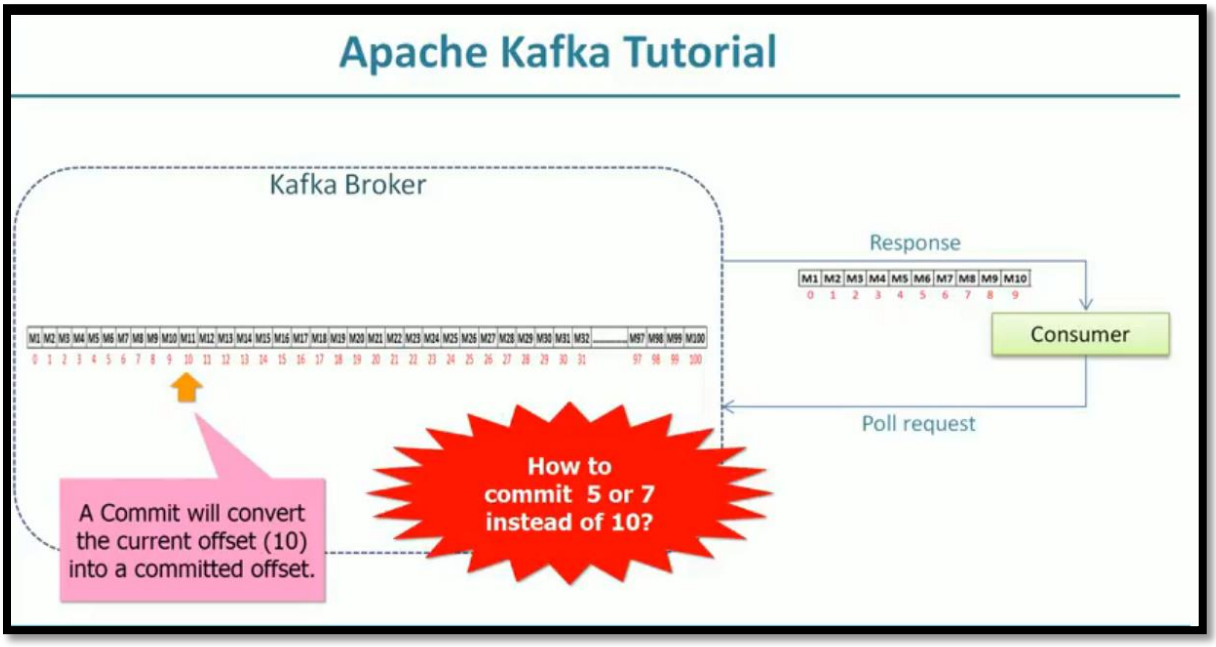


In this case you don't want to commit 75 after commit 100 that may cause issues so they designed asynchronous commit to not to retry

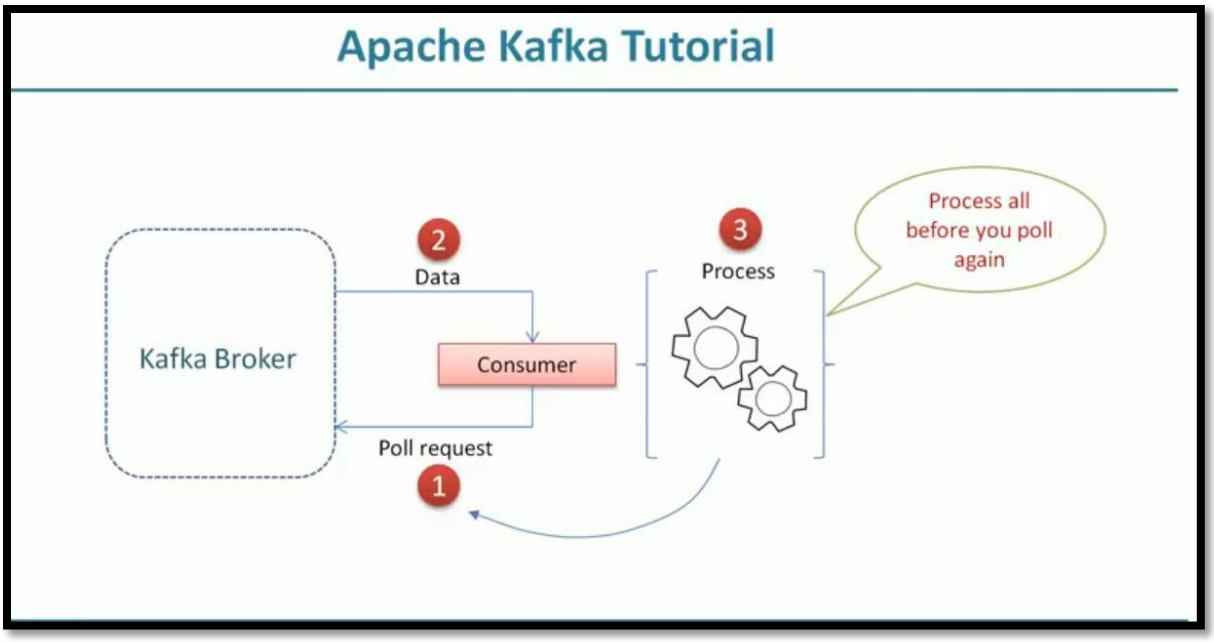
Code

```
import java.util.*;
import java.io.*;
import org.apache.kafka.clients.consumer.KafkaConsumer;
import org.apache.kafka.clients.consumer.ConsumerRecords;
import org.apache.kafka.clients.consumer.ConsumerRecord;

public class ManualConsumer{

   public static void main(String[] args) throws Exception{

      String topicName = "SupplierTopic";
      String groupName = "SupplierTopicGroup";

      Properties props = new Properties();
      props.put("bootstrap.servers", "localhost:9092,localhost:9093");
      props.put("group.id", groupName);
      props.put("key.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");
      props.put("value.deserializer", "SupplierDeserializer");
      props.put("enable.auto.commit", "false");

      KafkaConsumer<String, Supplier> consumer = null;

      try {
         consumer = new KafkaConsumer<>(props);
         consumer.subscribe(Arrays.asList(topicName));

         while (true){
            ConsumerRecords<String, Supplier> records = consumer.poll(100);
            for (ConsumerRecord<String, Supplier> record : records){
               System.out.println("Supplier id= " + String.valueOf(record.value().getID()) + " Supplier  Name
= " + record.value().getName() + " Supplier Start Date = " + record.value().getStartDate().toString());
            }
            consumer.commitAsync();
         }
      }catch(Exception ex){
         ex.printStackTrace();
      }finally{
         consumer.commitSync();
         consumer.close();
      }
   }
}
```

Rebalance Listener



Suppose we are getting lots of data using the poll method and it is going to take some reasonable time to complete the processing of all of the records



If you are taking lots of time to process the records you have two types of risks

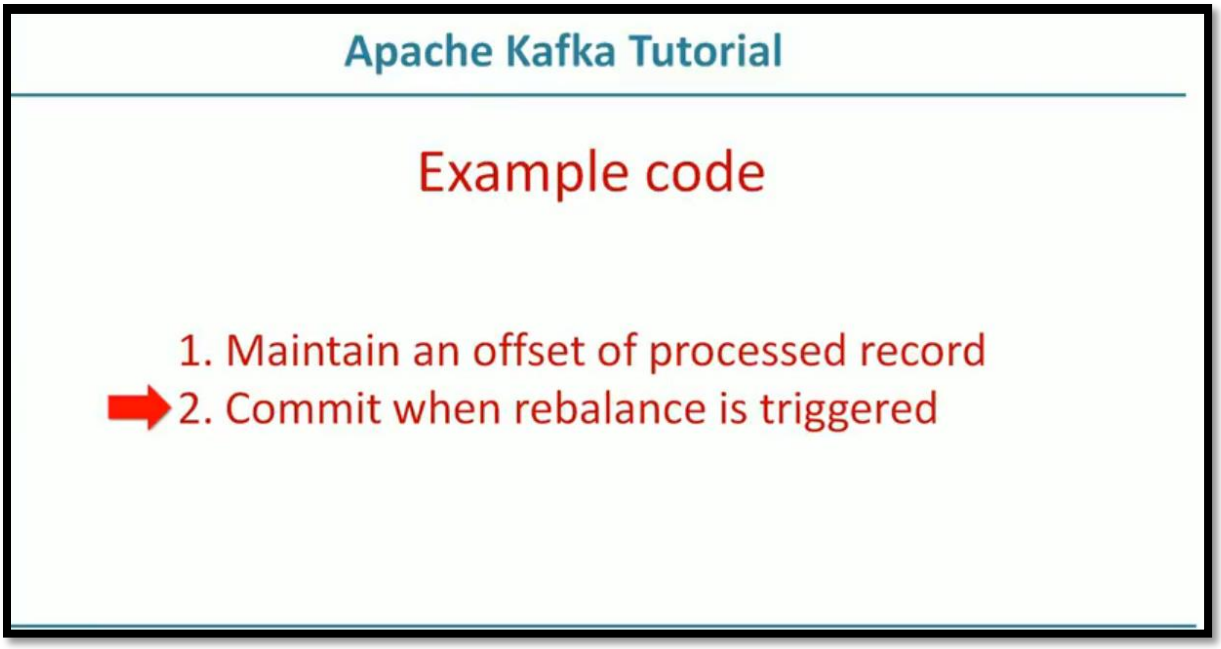
- Delay in next poll – if you don't poll for a long group coordinator might think that you are dead and trigger group rebalancing activity
- Rebalance is triggered - Rebalance activity for some other reason while you were processing the extensive records

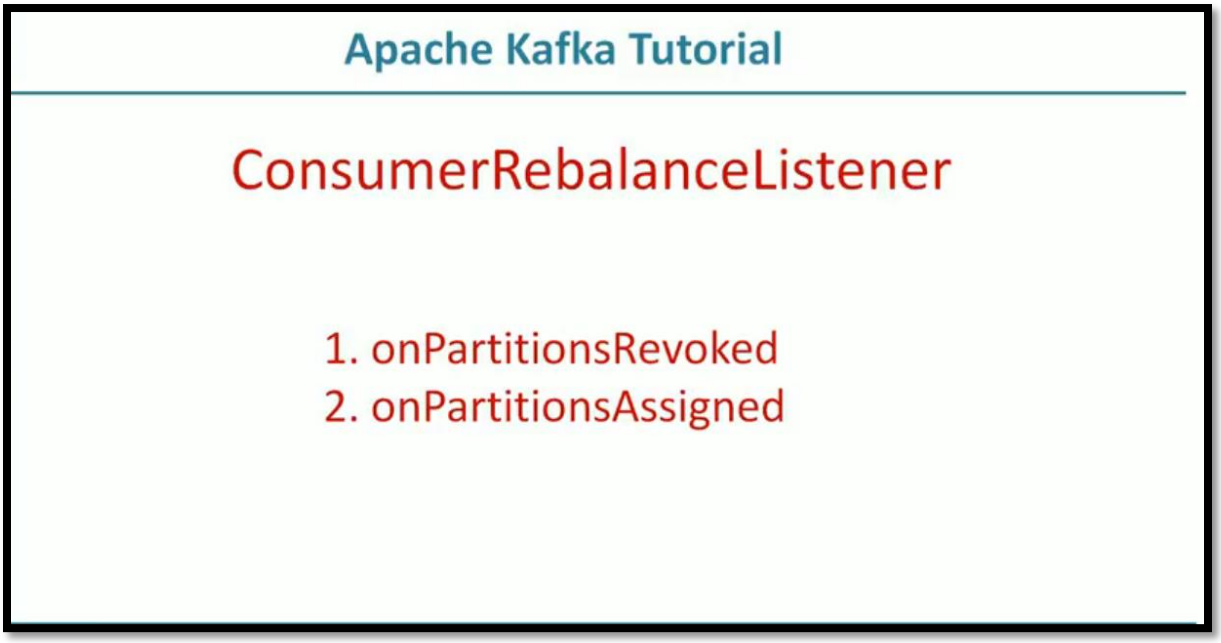
In both the cases your current partition will be taken away from you and assign to some other consumer in such case you will want to commit whatever you had already processed before the ownership of partition is taken from you and the new owner of the partition is supposed to start consuming it from the last committed offset

Apache Kafka Tutorial

Apache Kafka Tutorial

1. How to commit a particular offset?
2. How to know that a rebalance is triggered?





We can commit on onPartitionsRevoked

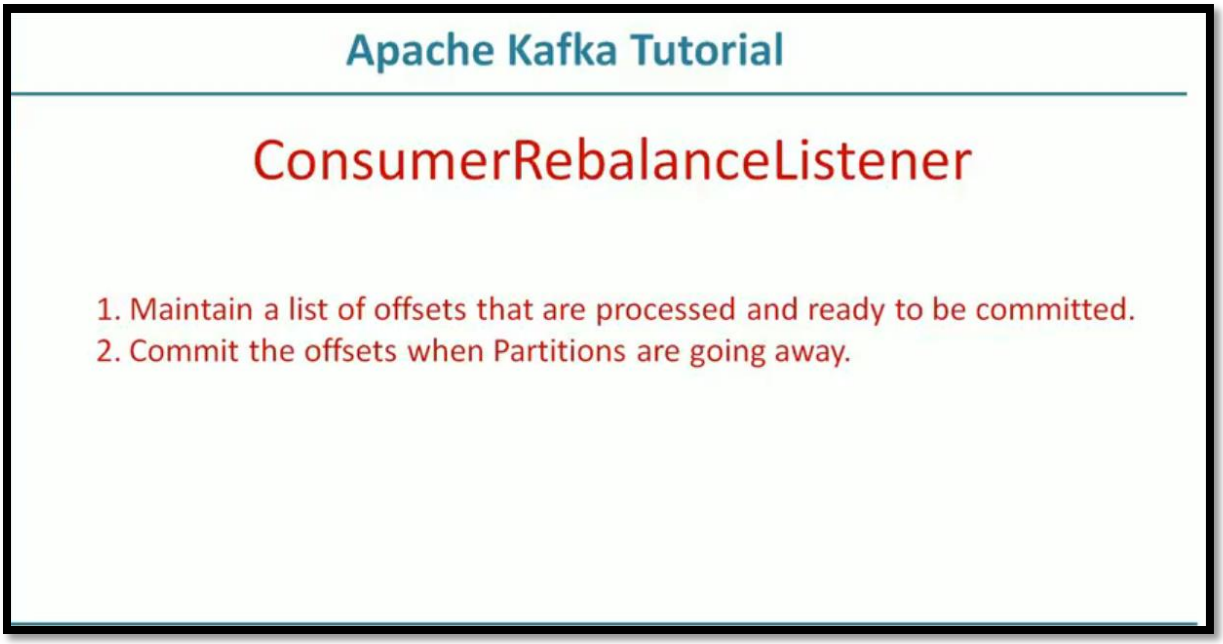
```
import java.util.*;
import org.apache.kafka.clients.producer.*;
public class RandomProducer {

  public static void main(String[] args) throws InterruptedException{

    String topicName = "RandomProducerTopic";
```

```
    String msg;

    Properties props = new Properties();
    props.put("bootstrap.servers", "localhost:9092,localhost:9093");
    props.put("key.serializer","org.apache.kafka.common.serialization.StringSerializer");
    props.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer");

    Producer<String, String> producer = new KafkaProducer <>(props);
    Random rg = new Random();
    Calendar dt = Calendar.getInstance();
    dt.set(2016,1,1);
    try{
       while(true){
          for (int i=0;i<100;i++){
            msg = dt.get(Calendar.YEAR)+"-"+dt.get(Calendar.MONTH)+"-"+dt.get(Calendar.DATE) + "," +
rg.nextInt(1000);
            producer.send(new ProducerRecord<String, String>(topicName,0,"Key",msg)).get();
            msg = dt.get(Calendar.YEAR)+"-"+dt.get(Calendar.MONTH)+"-"+dt.get(Calendar.DATE) + "," +
rg.nextInt(1000);
            producer.send(new ProducerRecord<String, String>(topicName,1,"Key",msg)).get();
          }
          dt.add(Calendar.DATE,1);
          System.out.println("Data Sent for " + dt.get(Calendar.YEAR) + "-" + dt.get(Calendar.MONTH) + "-
" + dt.get(Calendar.DATE) );
       }
    }
    catch(Exception ex){
       System.out.println("Intrupted");
    }
    finally{
       producer.close();
     }

  }
}
```

Here we are sending the same data to two partitions

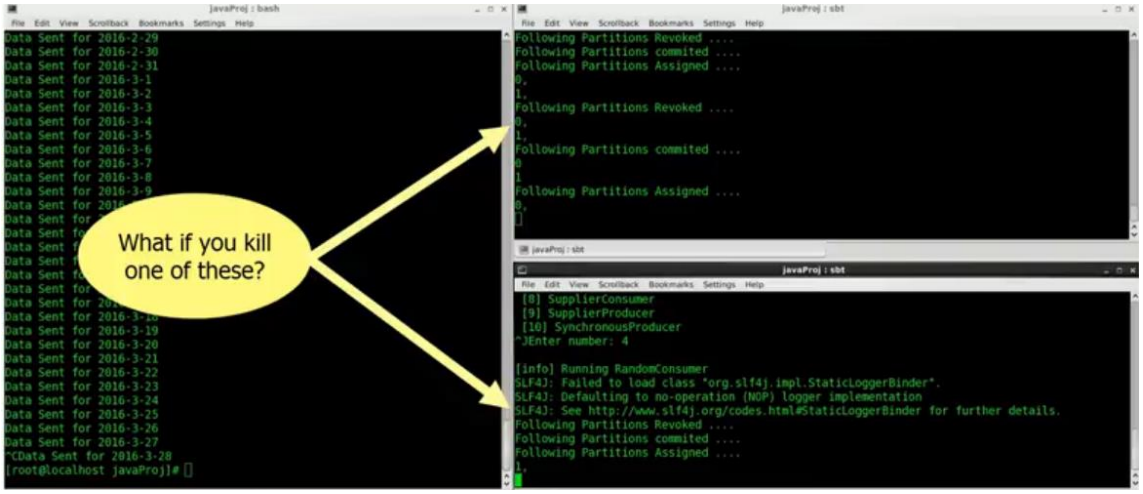
```
import java.util.*;
import org.apache.kafka.clients.consumer.*;
import org.apache.kafka.common.*;

public class RandomConsumer{


   public static void main(String[] args) throws Exception{
```

```
        String topicName = "RandomProducerTopic";
        KafkaConsumer<String, String> consumer = null;

        String groupName = "RG";
        Properties props = new Properties();
        props.put("bootstrap.servers", "localhost:9092,localhost:9093");
        props.put("group.id", groupName);
        props.put("key.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");
        props.put("value.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");
        props.put("enable.auto.commit", "false");

        consumer = new KafkaConsumer<>(props);
        RebalanceListner rebalanceListner = new RebalanceListner(consumer);

        consumer.subscribe(Arrays.asList(topicName),rebalanceListner);
        try{
           while (true){
              ConsumerRecords<String, String> records = consumer.poll(100);
              for (ConsumerRecord<String, String> record : records){
                 //System.out.println("Topic:"+ record.topic() +" Partition:" + record.partition() + " Offset:"
+ record.offset() + " Value:"+ record.value());
                // Do some processing and save it to Database
                 rebalanceListner.addOffset(record.topic(), record.partition(),record.offset());
              }
                 //consumer.commitSync(rebalanceListner.getCurrentOffsets());
           }
        }catch(Exception ex){
           System.out.println("Exception.");
           ex.printStackTrace();
        }
        finally{
              consumer.close();
        }
   }

}
```

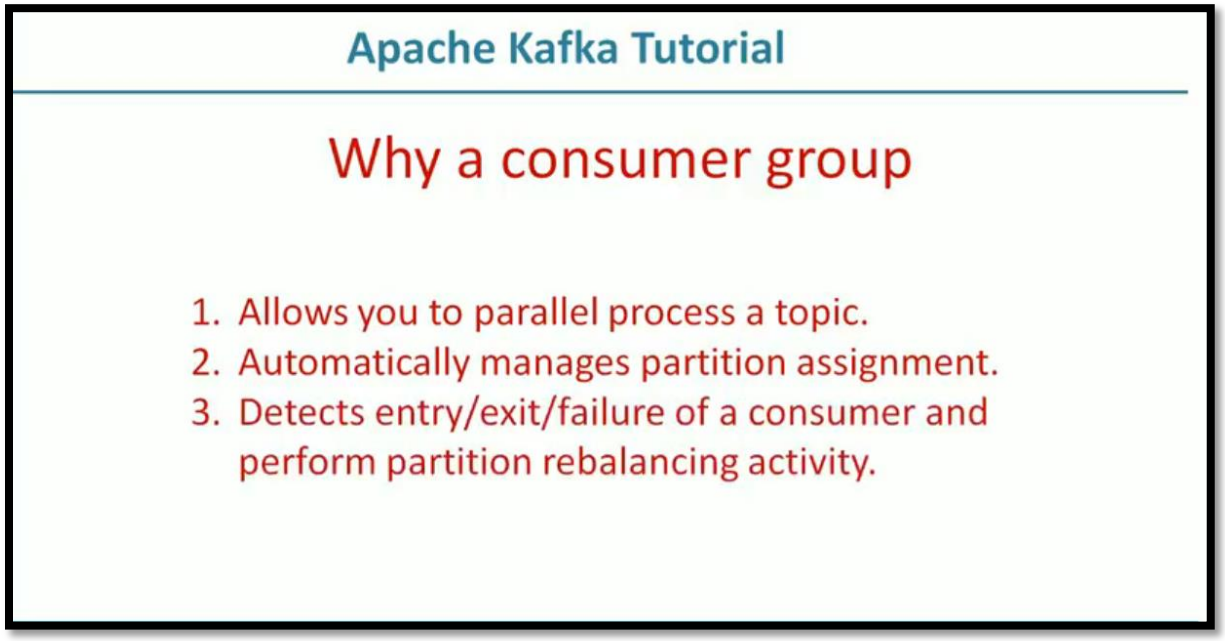


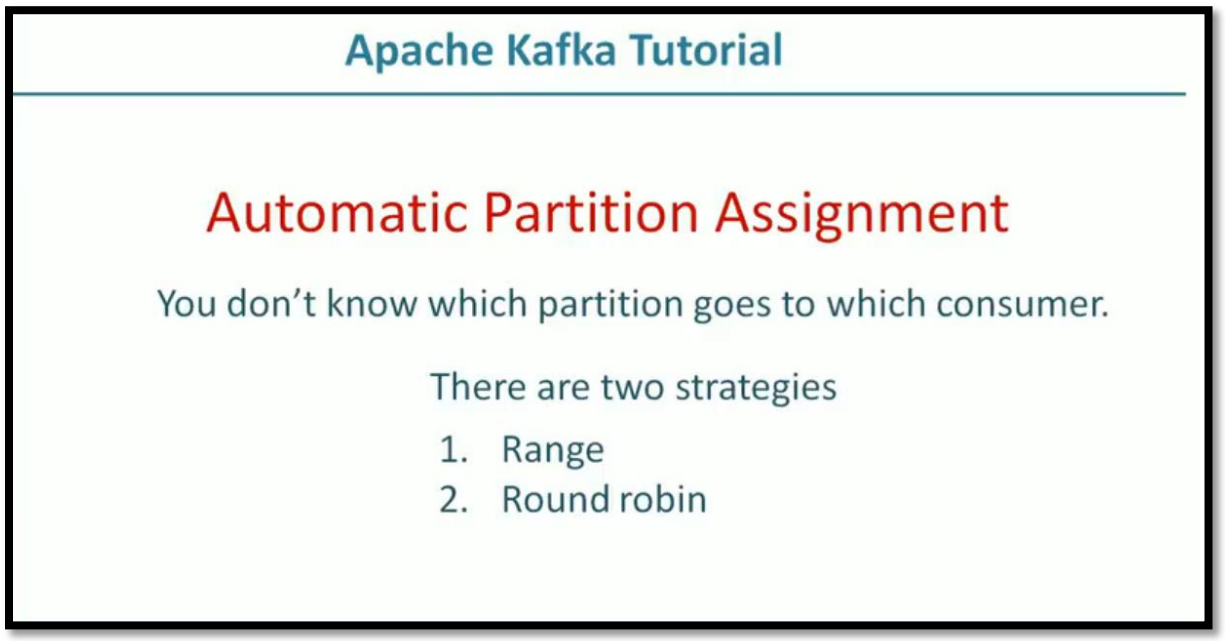

```
import java.util.*;
import org.apache.kafka.clients.consumer.*;
import org.apache.kafka.common.*;

public class RebalanceListner implements ConsumerRebalanceListener {
   private KafkaConsumer consumer;
   private Map<TopicPartition, OffsetAndMetadata> currentOffsets = new HashMap();

   public RebalanceListner(KafkaConsumer con){
      this.consumer=con;
   }

   public void addOffset(String topic, int partition, long offset){
      currentOffsets.put(new TopicPartition(topic, partition),new OffsetAndMetadata(offset,"Commit"));
   }

   public Map<TopicPartition, OffsetAndMetadata> getCurrentOffsets(){
      return currentOffsets;
   }

   public void onPartitionsAssigned(Collection<TopicPartition> partitions) {
      System.out.println("Following Partitions Assigned ....");
      for(TopicPartition partition: partitions)
         System.out.println(partition.partition()+",");
   }

   public void onPartitionsRevoked(Collection<TopicPartition> partitions) {
      System.out.println("Following Partitions Revoked ....");
```

```
        for(TopicPartition partition: partitions)
            System.out.println(partition.partition()+",");


        System.out.println("Following Partitions commited ...." );
        for(TopicPartition tp: currentOffsets.keySet())
            System.out.println(tp.partition());

        consumer.commitSync(currentOffsets);
        currentOffsets.clear();
    }
}
```

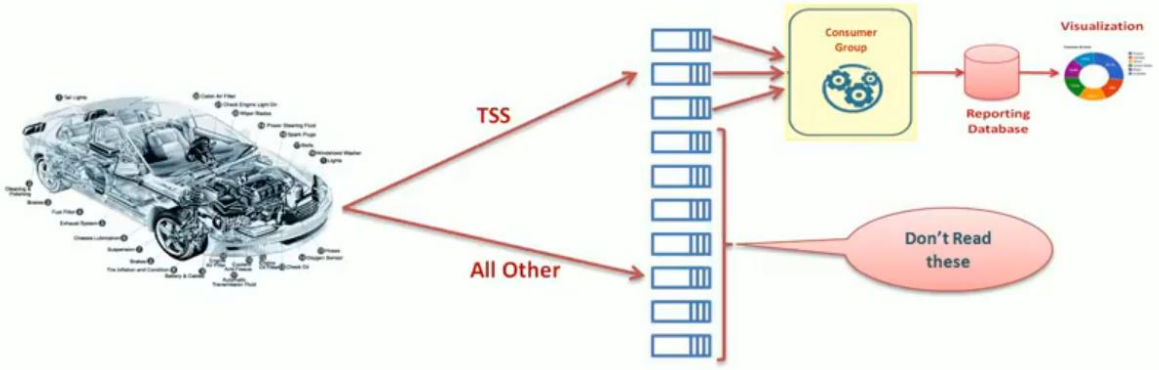


Exactly one processing

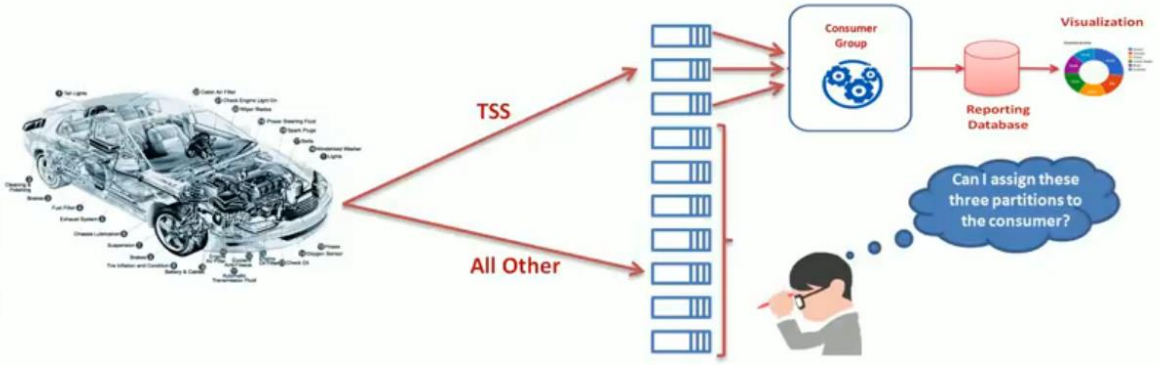




Apache Kafka Tutorial

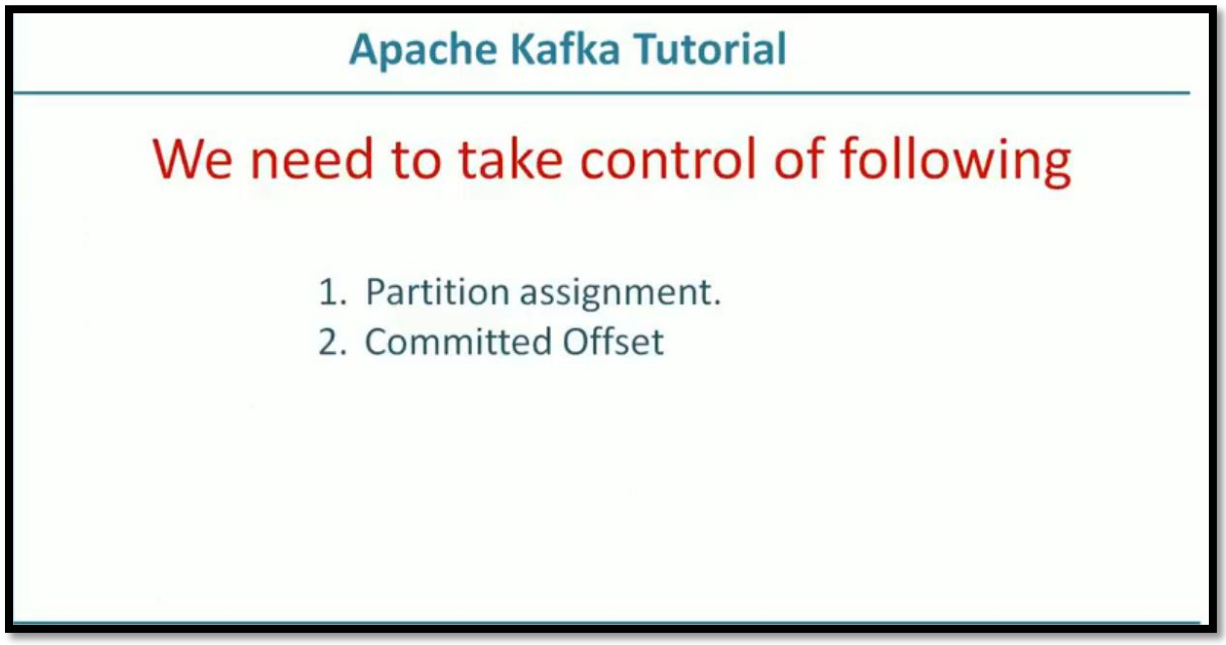


Apache Kafka Tutorial



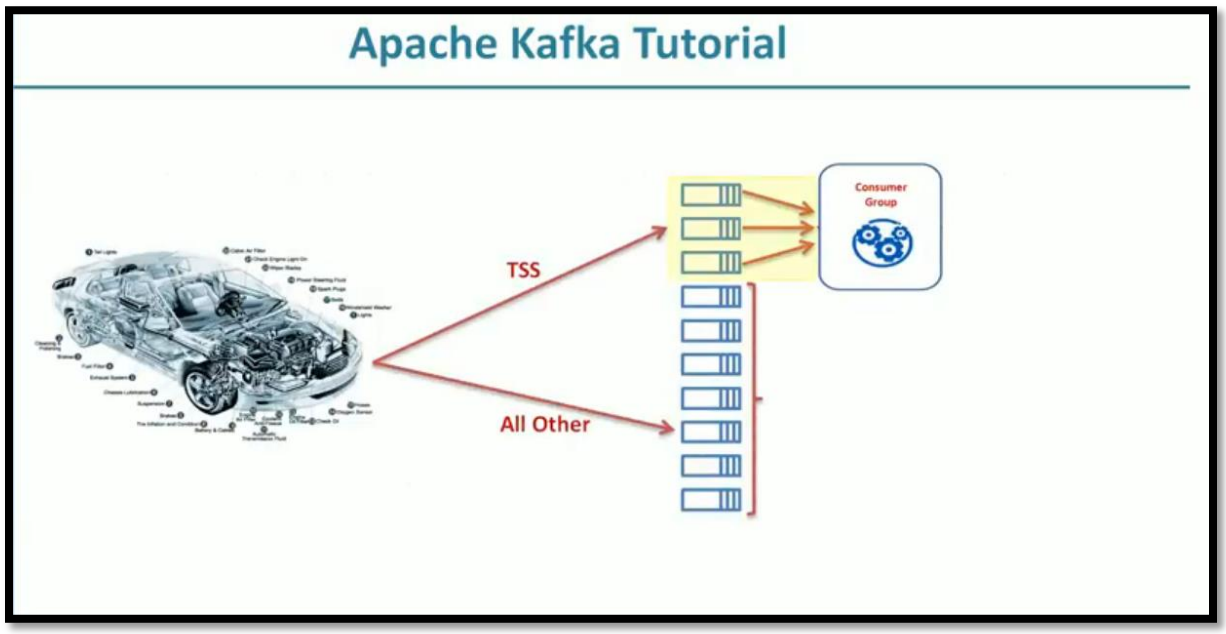
```
ing groupName = "RG";
perties props = new Properties();
s.put("bootstrap.servers", "localhost:9092,localhost:9093");
s.put("group.id", groupName);
s.put("key.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");
s.put("value.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");
s.put("enable.auto.commit", "false");

sumer = new KafkaConsumer<>(props);
alanceListner rebalanceListner = new RebalanceListner(consumer);

sumer.subscribe(Arrays.asList(topicName),rebalanceListner);
{
 while (true){
     ConsumerRecords<String, String> records = consumer.poll(100);
     for (ConsumerRecord<String, String> record : records){
         //System.out.println("Topic:"+ record.topic() +" Partition:" + record.partition()
         // Do some processing and save it to Database
         rebalanceListner.addOffset(record.topic(), record.partition(),record.offset());
     }
         //consumer.commitSync(rebalanceListner.getCurrentOffsets());
 }
tch(Exception ex){
 System.out.println("Exception.");
 ex.printStackTrace();

ally{
     consumer.close();
```

Problem – since we are saving the record to database and then adding offset to listener, these two steps are not atomic, it is possible after saving it to database consumer crashes before committing the offset , so we need to create these two steps as single atomic transaction

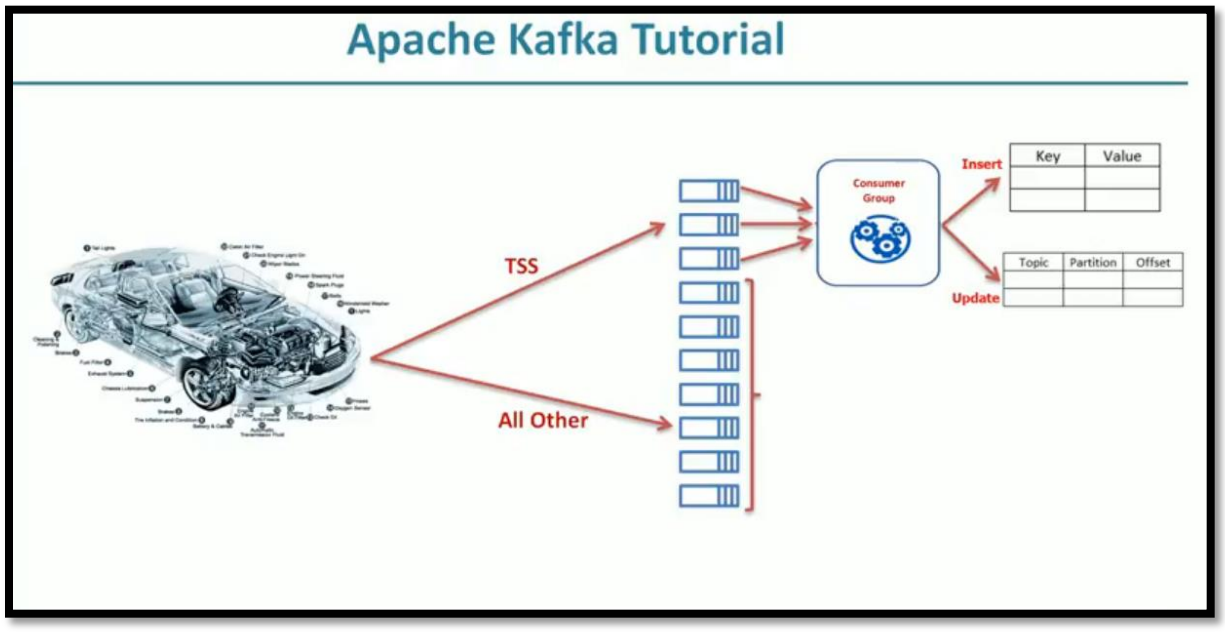


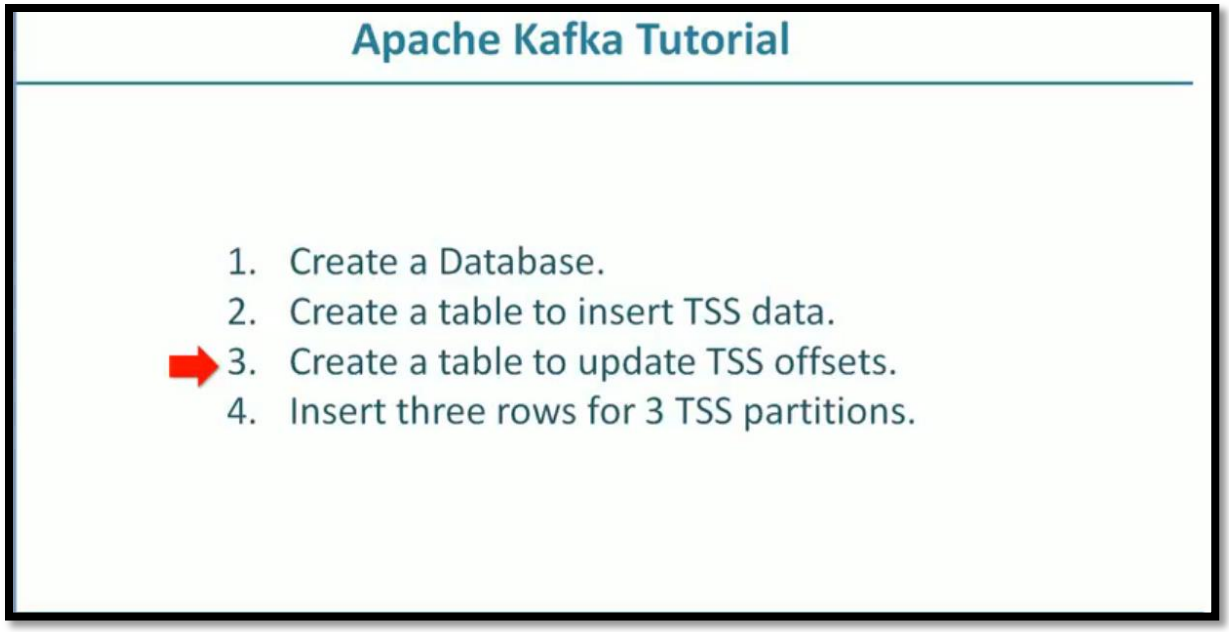
- We don't want kafka to assign partition to different consumer, we want to take control and assign partition

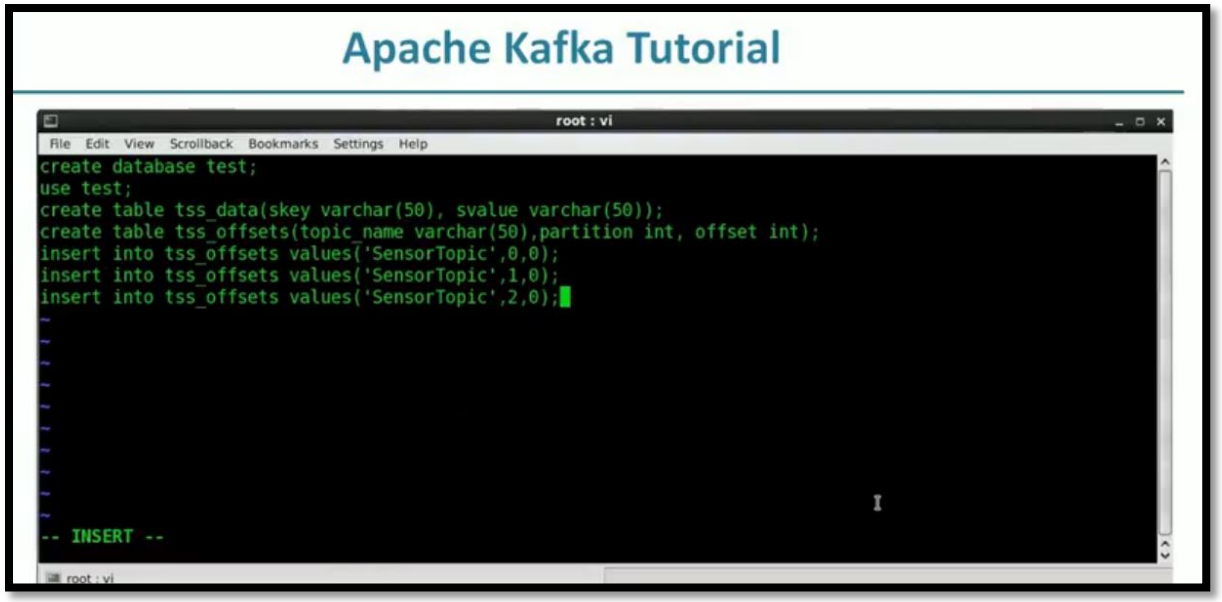
- We don’t want kafka to store the offset



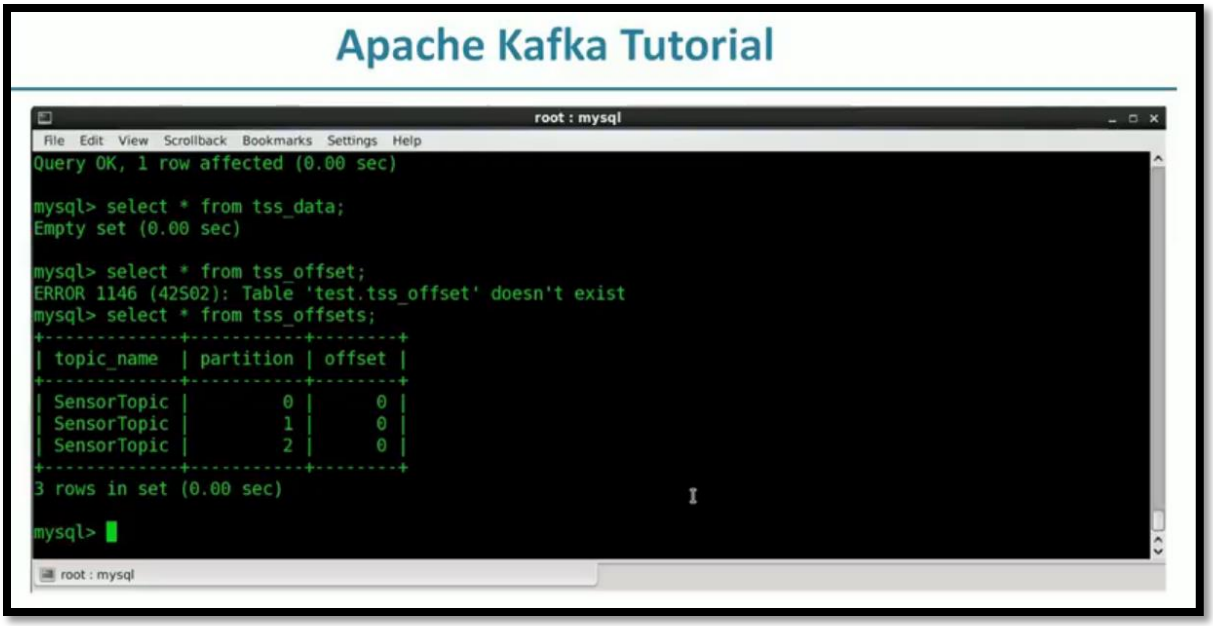
We need to create a consumer that will assign these 3 partitions to itself then it will start reading messages from these 3 partitions and insert each message into a database table, it will not commit the offset back to kafka instead it will update the current offset into another database table , the insert and update statement will be part of the single transaction so either both will complete or both will fail







Consumer will insert the data into tss_data



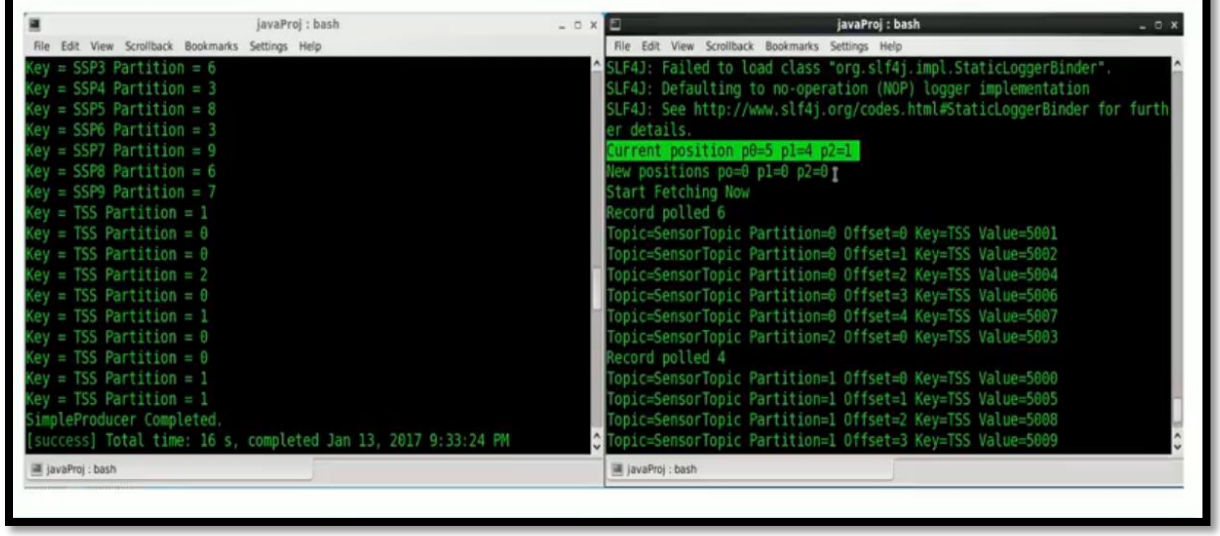
Initial offset set to 0 , consumer will start reading from the beginning and keep updating this table

```
import java.util.*;
import org.apache.kafka.clients.consumer.*;
import org.apache.kafka.common.*;
import java.sql.*;

public class SensorConsumer{


   public static void main(String[] args) throws Exception{

        String topicName = "SensorTopic";
        KafkaConsumer<String, String> consumer = null;
        int rCount;

        Properties props = new Properties();
        props.put("bootstrap.servers", "localhost:9092,localhost:9093");
        props.put("key.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");
        props.put("value.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");
        props.put("enable.auto.commit", "false");

        consumer = new KafkaConsumer<>(props);
        TopicPartition p0 = new TopicPartition(topicName, 0);
        TopicPartition p1 = new TopicPartition(topicName, 1);
        TopicPartition p2 = new TopicPartition(topicName, 2);

        consumer.assign(Arrays.asList(p0,p1,p2));
        System.out.println("Current position p0=" + consumer.position(p0)
```

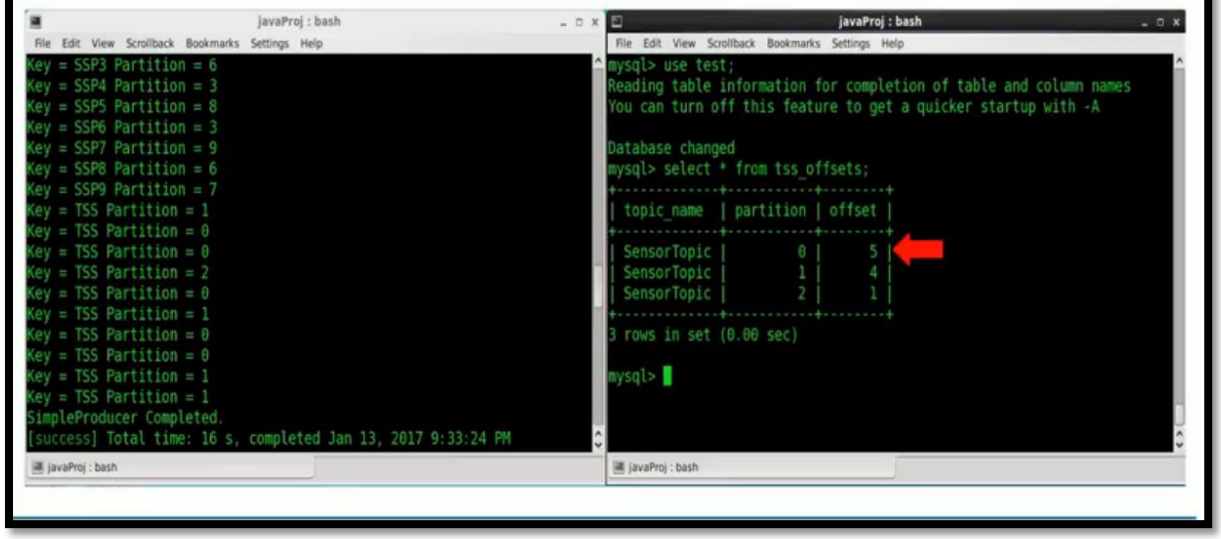
```
                    + " p1=" + consumer.position(p1)
                    + " p2=" + consumer.position(p2));

    //adjusted offset to appropriate position
        consumer.seek(p0, getOffsetFromDB(p0));
        consumer.seek(p1, getOffsetFromDB(p1));
        consumer.seek(p2, getOffsetFromDB(p2));
        System.out.println("New positions po=" + consumer.position(p0)
                    + " p1=" + consumer.position(p1)
                    + " p2=" + consumer.position(p2));

        System.out.println("Start Fetching Now");
        try{
           do{
              ConsumerRecords<String, String> records = consumer.poll(1000);
              System.out.println("Record polled " + records.count());
              rCount = records.count();
              for (ConsumerRecord<String, String> record : records){

                 saveAndCommit(consumer,record);
              }
           }while (rCount>0);
        }catch(Exception ex){
           System.out.println("Exception in main.");
        }
        finally{
              consumer.close();
        }
  }

  private static long getOffsetFromDB(TopicPartition p){
     long offset = 0;
     try{
           Class.forName("com.mysql.jdbc.Driver");
           Connection
con=DriverManager.getConnection("jdbc:mysql://localhost:3306/test","root","pandey");

           String sql = "select offset from tss_offsets where topic_name='" + p.topic() + "' and partition="
+ p.partition();
           Statement stmt=con.createStatement();
           ResultSet rs = stmt.executeQuery(sql);
           if (rs.next())
              offset = rs.getInt("offset");
           stmt.close();
           con.close();
        }catch(Exception e){
           System.out.println("Exception in getOffsetFromDB");
        }
```

```
      return offset;
   }

   private static void saveAndCommit(KafkaConsumer<String, String> c, ConsumerRecord<String, String>
r){
      System.out.println("Topic=" + r.topic() + " Partition=" + r.partition() + " Offset=" + r.offset() + " Key="
+ r.key() + " Value=" + r.value());
      try{
         Class.forName("com.mysql.jdbc.Driver");
         Connection
con=DriverManager.getConnection("jdbc:mysql://localhost:3306/test","root","pandey");
         con.setAutoCommit(false);

         String insertSQL = "insert into tss_data values(?,?)";
         PreparedStatement psInsert = con.prepareStatement(insertSQL);
         psInsert.setString(1,r.key());
         psInsert.setString(2,r.value());

         String updateSQL = "update tss_offsets set offset=? where topic_name=? and partition=?";
         PreparedStatement psUpdate = con.prepareStatement(updateSQL);
         psUpdate.setLong(1,r.offset()+1);
         psUpdate.setString(2,r.topic());
         psUpdate.setInt(3,r.partition());

         psInsert.executeUpdate();
         psUpdate.executeUpdate();
         con.commit();
         con.close();
      }catch(Exception e){
         System.out.println("Exception in saveAndCommit");
      }

   }
}
```

Apache Kafka Tutorial

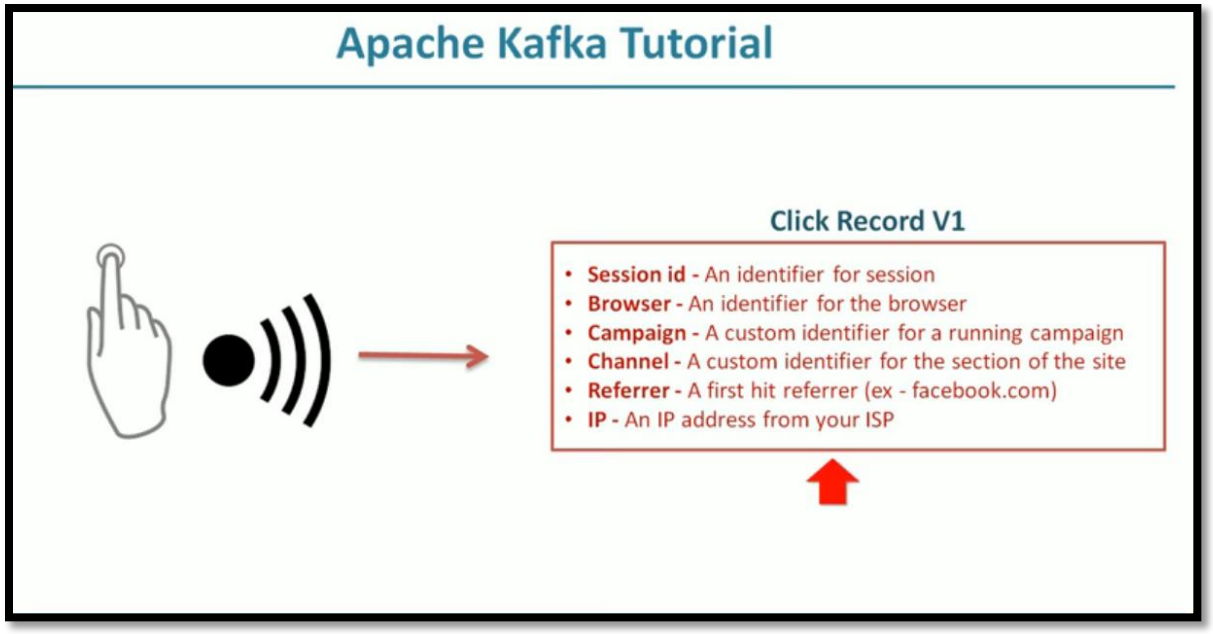


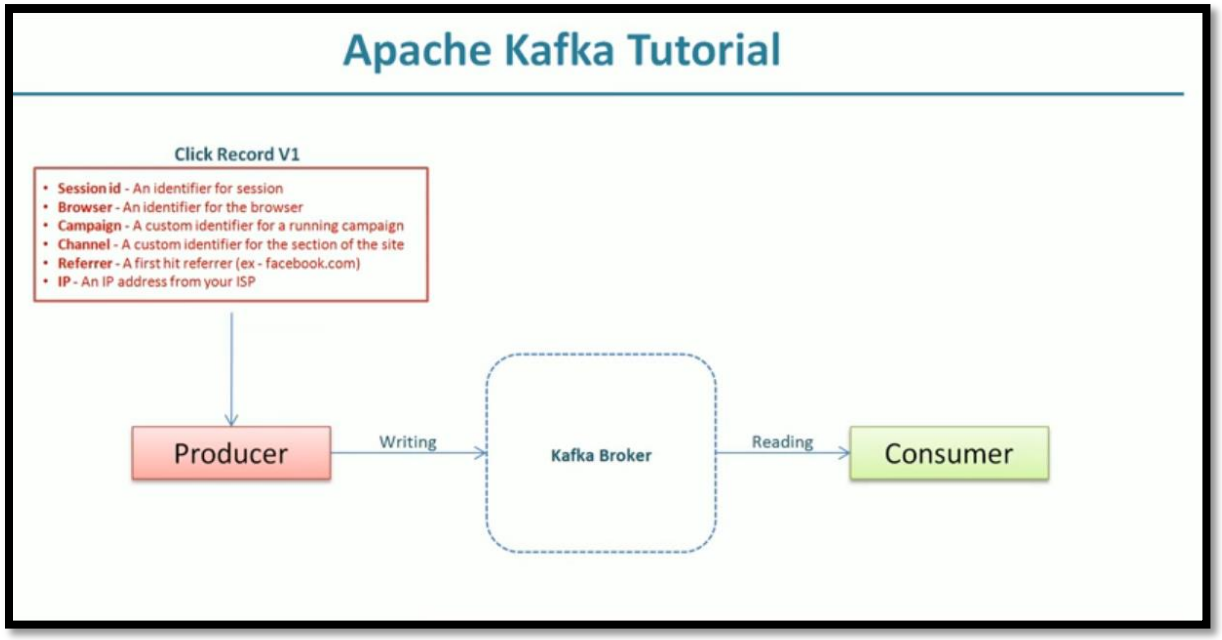
Apache Kafka Tutorial



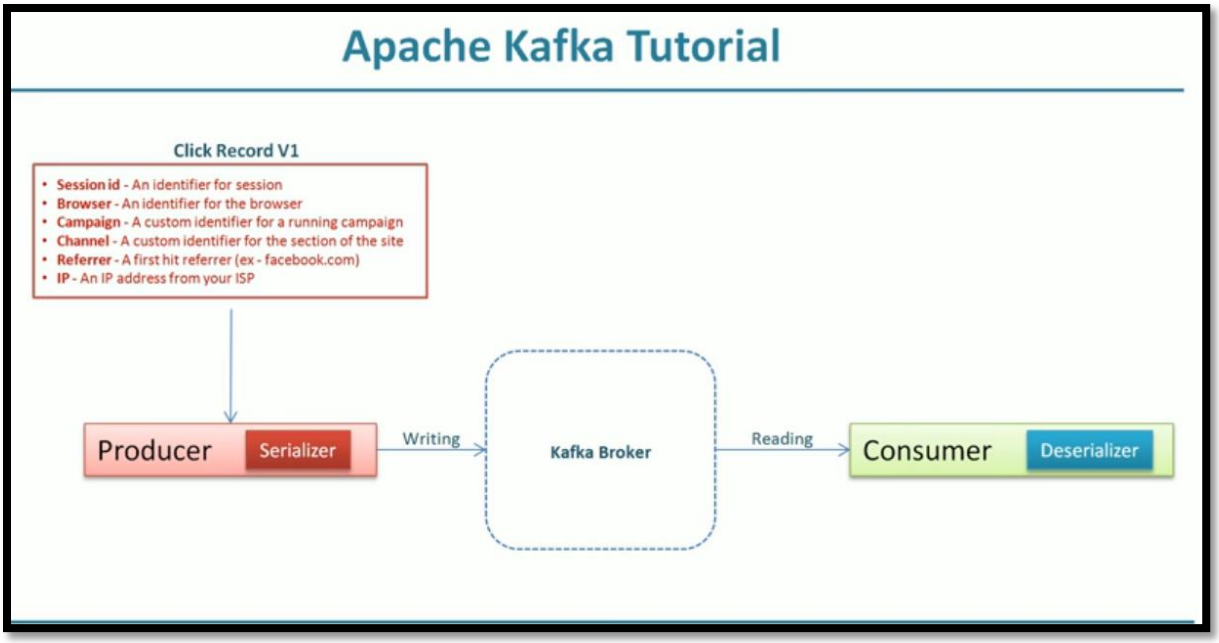
Schema Evolution

Schema evolution dealing with changes in message records over the period of time





Apache Kafka Tutorial



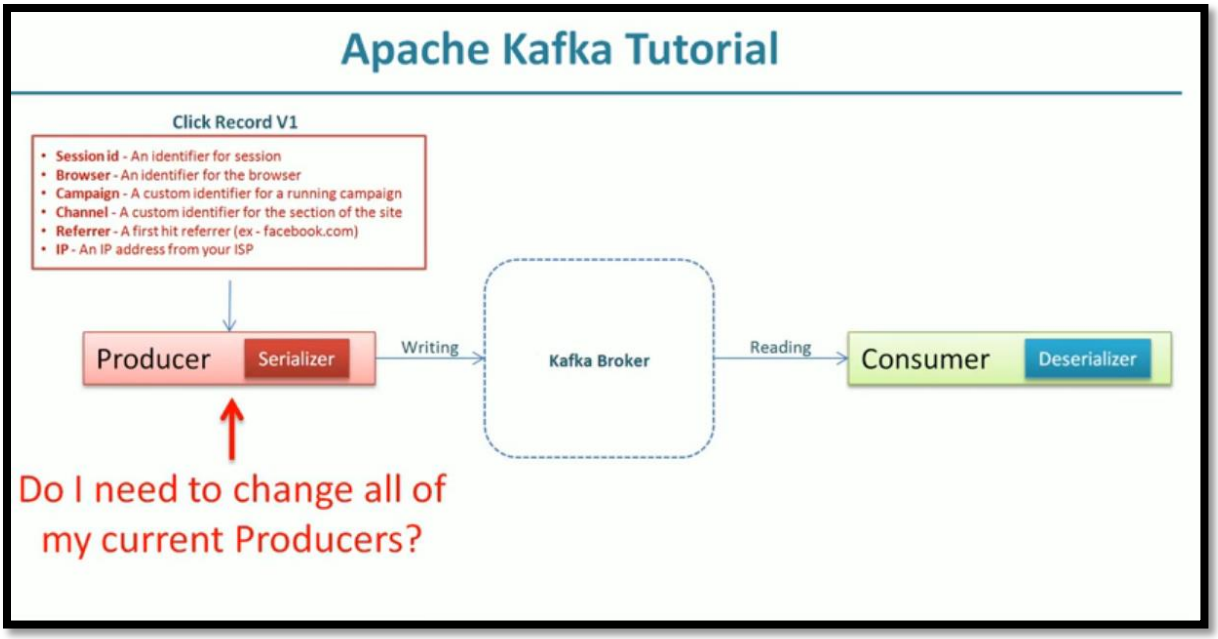
Apache Kafka Tutorial

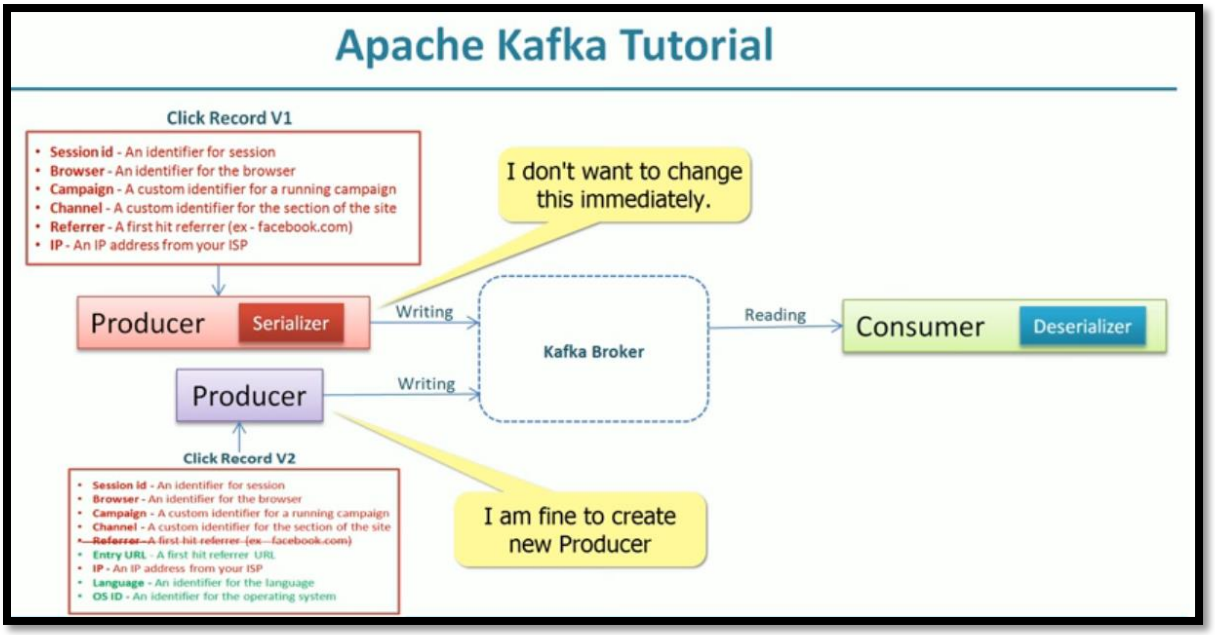
Click Record V1

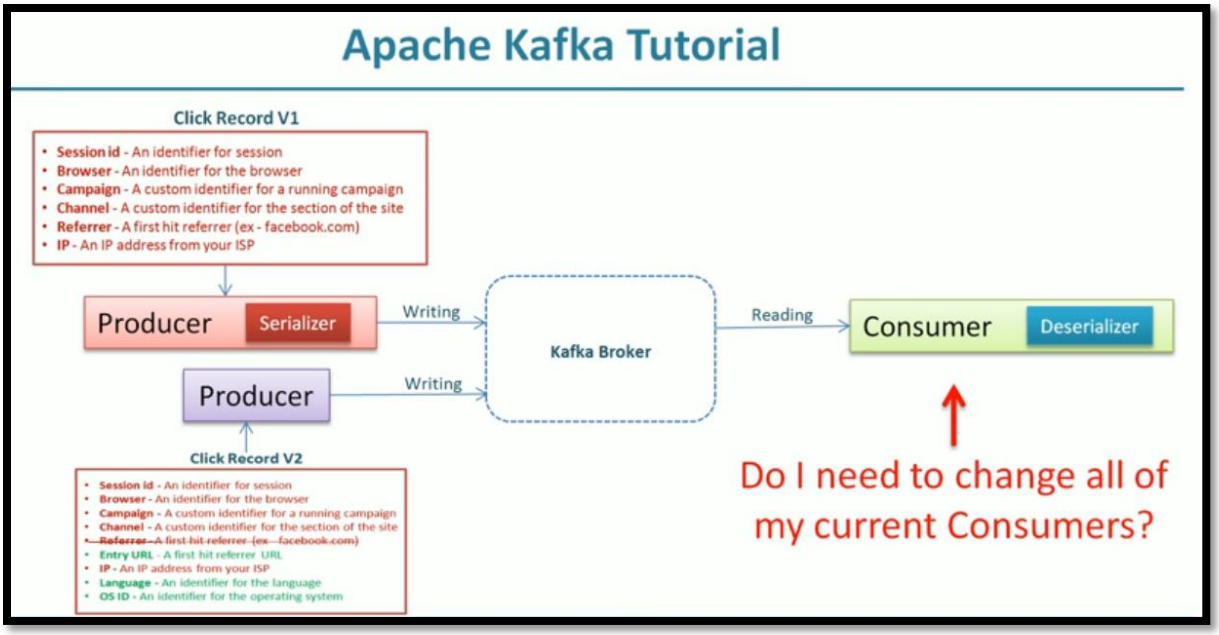
- **Session id** - An identifier for session
- **Browser** - An identifier for the browser
- **Campaign** - A custom identifier for a running campaign
- **Channel** - A custom identifier for the section of the site
- **Referrer** - A first hit referrer (ex - facebook.com)
- **IP** - An IP address from your ISP

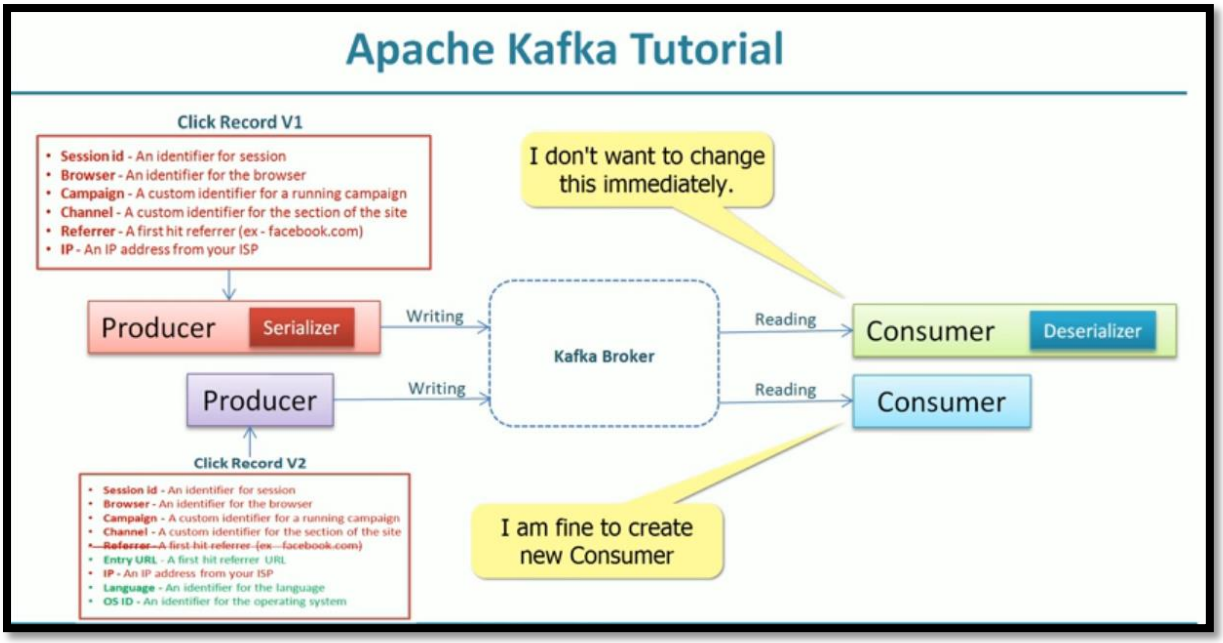
Click Record V2

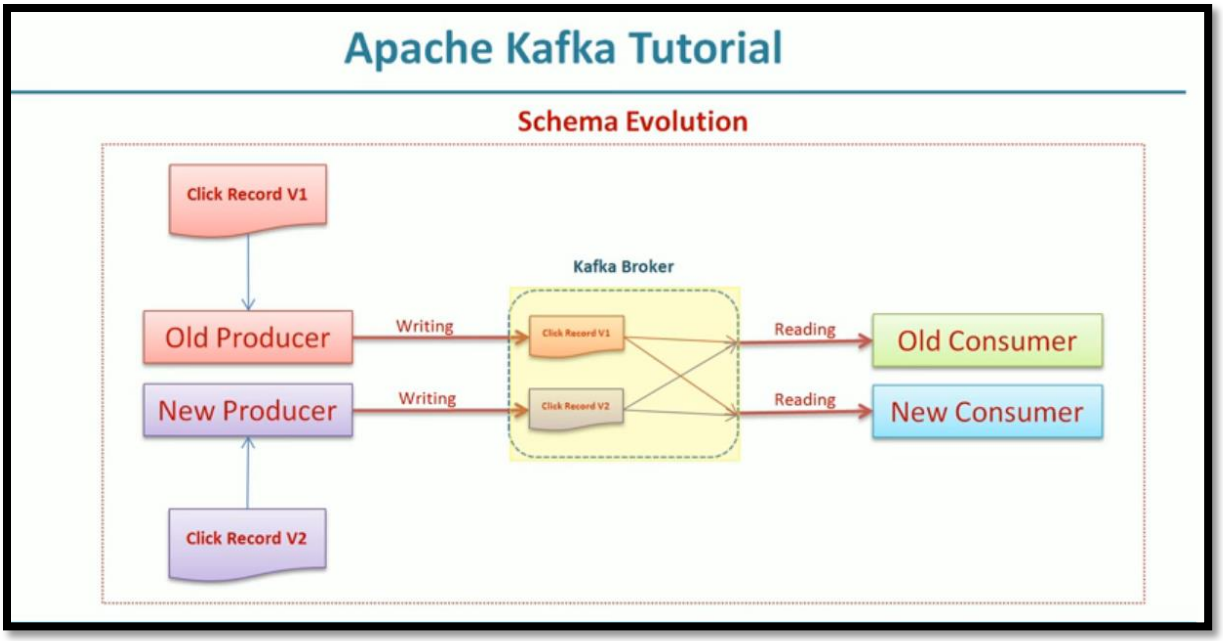
- **Session id** - An identifier for session
- **Browser** - An identifier for the browser
- **Campaign** - A custom identifier for a running campaign
- **Channel** - A custom identifier for the section of the site
- ~~**Referrer** - A first hit referrer (ex - facebook.com)~~
- **Entry URL** - A first hit referrer URL
- **IP** - An IP address from your ISP
- **Language** - An identifier for the language
- **OS ID** - An identifier for the operating system


Apache Kafka Tutorial
Click Record V1
Session id - An identifier for session
Browser - An identifier for the browser
Campaign - A custom identifier for a running campaign
Channel - A custom identifier for the section of the site
Referrer - A first hit referrer (ex - facebook.com)
IP - An IP address from your ISP
Producer
Serializer
Writing
Kafka Broker
Reading
Consumer
Deserializer
Do I need to change all of my current Producers?


Apache Kafka Tutorial
Click Record V1
Session id - An identifier for session
Browser - An identifier for the browser
Campaign - A custom identifier for a running campaign
Channel - A custom identifier for the section of the site
Referrer - A first hit referrer (ex - facebook.com)
IP - An IP address from your ISP
I don't want to change this immediately.
Producer
Serializer
Writing
Kafka Broker
Reading
Consumer
Deserializer
Producer
Writing
Click Record V2
Session id - An identifier for session
Browser - An identifier for the browser
Campaign - A custom identifier for a running campaign
Channel - A custom identifier for the section of the site
Referrer - A first hit referrer (ex - facebook.com)
Entry URL - A first hit referrer URL
IP - An IP address from your ISP
Language - An identifier for the language
OS ID - An identifier for the operating system
I am fine to create new Producer

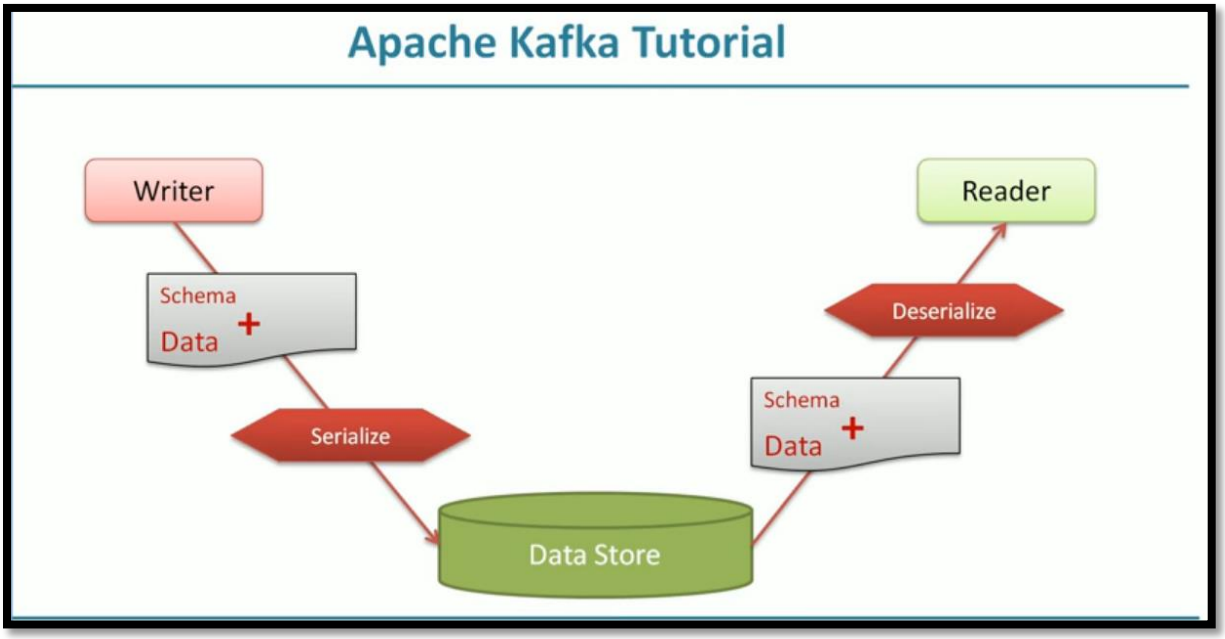

Apache Kafka Tutorial
Click Record V1
Session id - An identifier for session
Browser - An identifier for the browser
Campaign - A custom identifier for a running campaign
Channel - A custom identifier for the section of the site
Referrer - A first hit referrer (ex - facebook.com)
IP - An IP address from your ISP
Producer
Serializer
Writing
Kafka Broker
Reading
Consumer
Deserializer
Producer
Writing
Click Record V2
Session id - An identifier for session
Browser - An identifier for the browser
Campaign - A custom identifier for a running campaign
Channel - A custom identifier for the section of the site
Referrer - A first hit referrer (ex - facebook.com)
Entry URL - A first hit referrer URL
IP - An IP address from your ISP
Language - An identifier for the language
OS ID - An identifier for the operating system
Do I need to change all of my current Consumers?


Apache Kafka Tutorial
Click Record V1
Session id - An identifier for session
Browser - An identifier for the browser
Campaign - A custom identifier for a running campaign
Channel - A custom identifier for the section of the site
Referrer - A first hit referrer (ex - facebook.com)
IP - An IP address from your ISP
I don't want to change this immediately.
Producer
Serializer
Writing
Kafka Broker
Reading
Consumer
Deserializer
Producer
Writing
Reading
Consumer
Click Record V2
Session id - An identifier for session
Browser - An identifier for the browser
Campaign - A custom identifier for a running campaign
Channel - A custom identifier for the section of the site
Referrer - A first hit referrer (ex - facebook.com)
Entry URL - A first hit referrer URL
IP - An IP address from your ISP
Language - An identifier for the language
OS ID - An identifier for the operating system
I am fine to create new Consumer



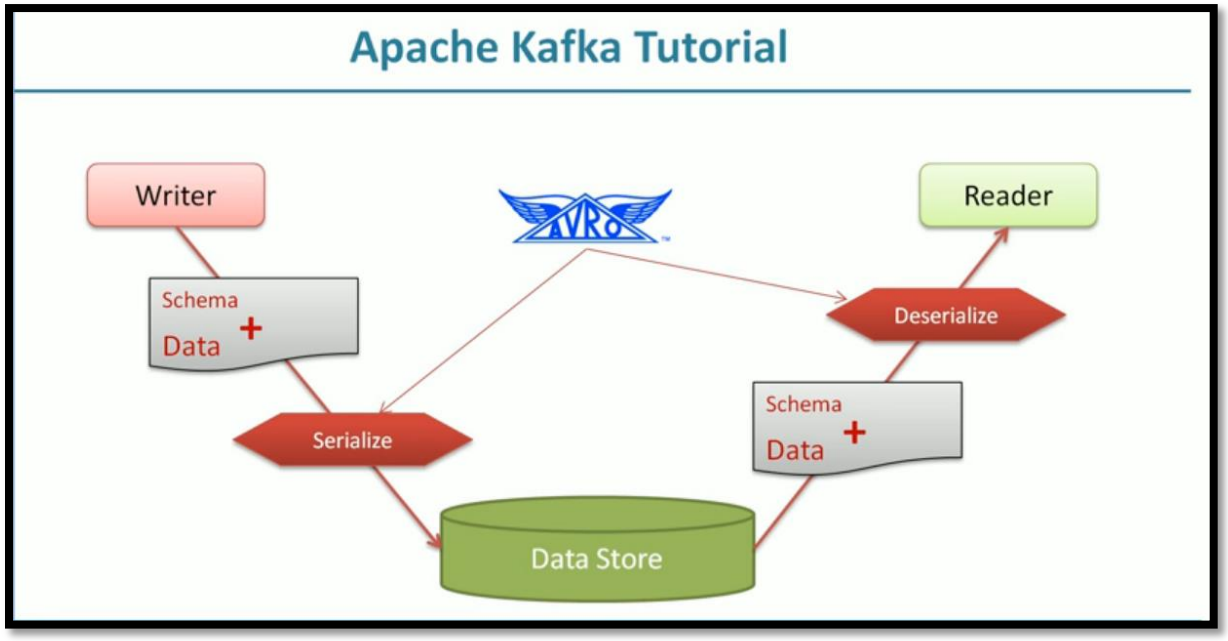
We want to support both old and new schemas simultaneously

We need to have a combination of old and new producer as well as combination of old and new consumer, kafka should be able to store both type of messages on the same topic. Consumer should be read both types of messages without any error



The Industry solution to handle schema evolution is to include schema information with the data when someone is writing data should write both schema and data and when someone wants to read the data they first read schema and then read data based on the schema

There are prebuild and reusable serialization system to help us and simplify the whole process of translation messages according to schema and embedding schema information in the message records

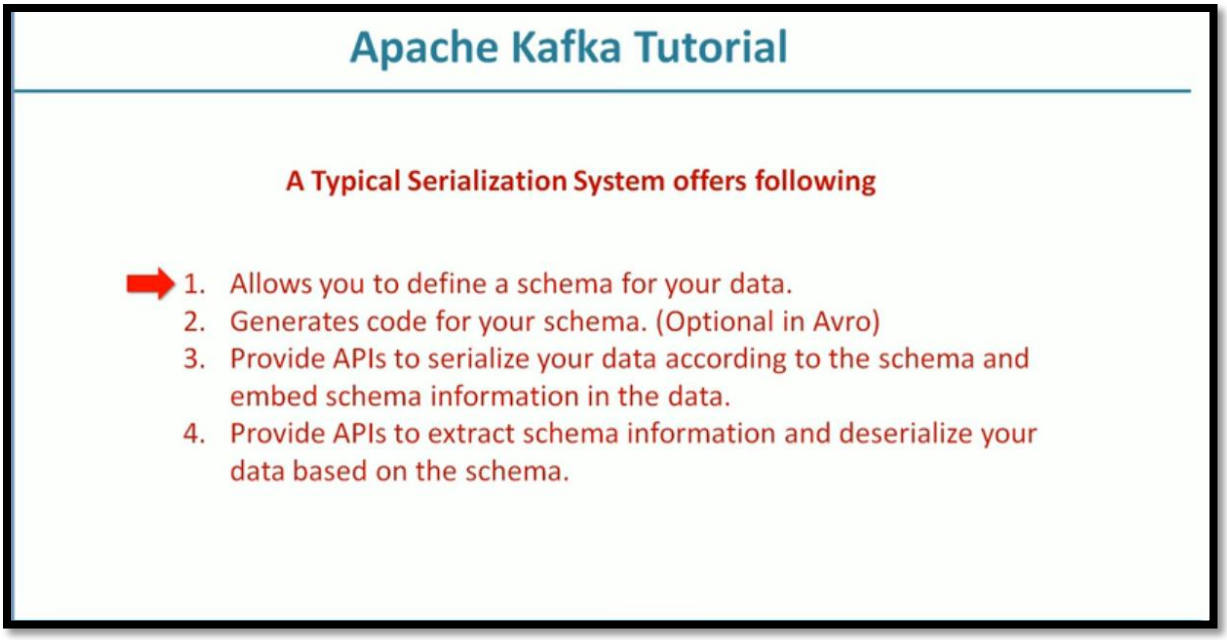


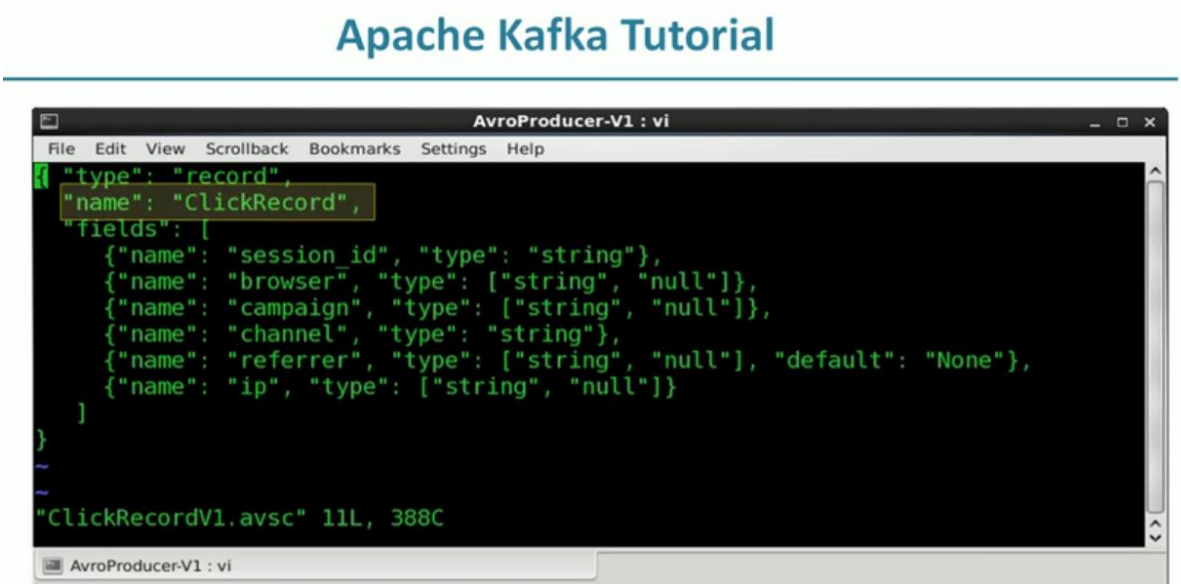
AVRO is one of them it is most popular serialization system for Hadoop and its eco system, Kafka follows the same approach and uses Avro to handle the schema evolution problem

See below URL

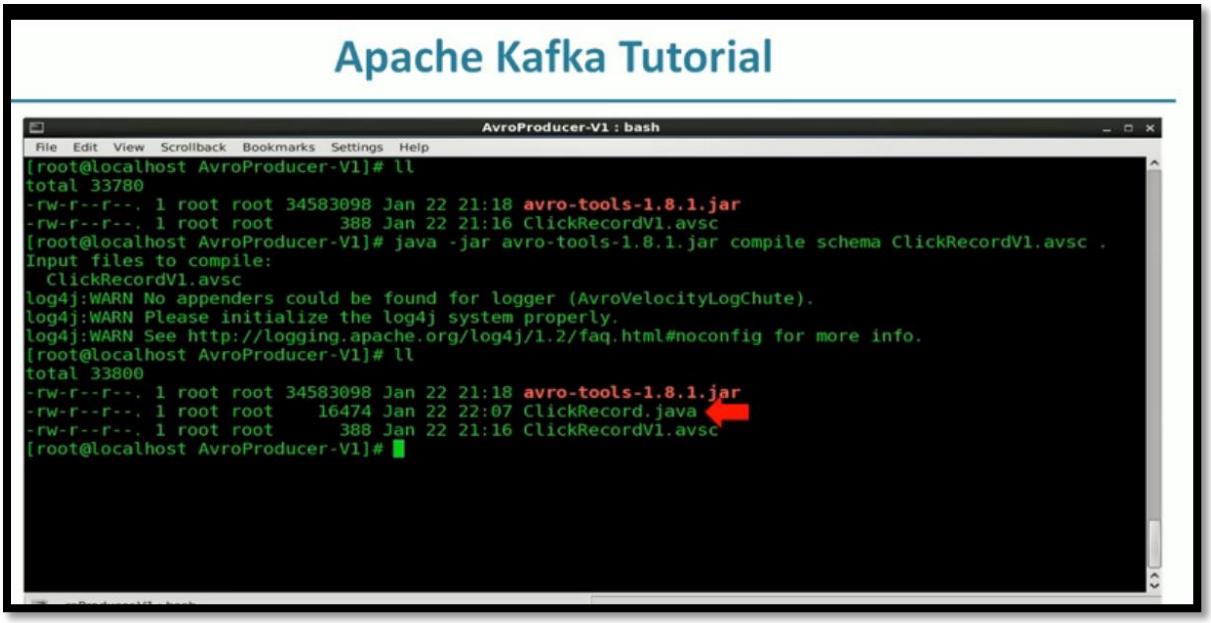
http://www-us.apache.org/dist/avro/stable/java/

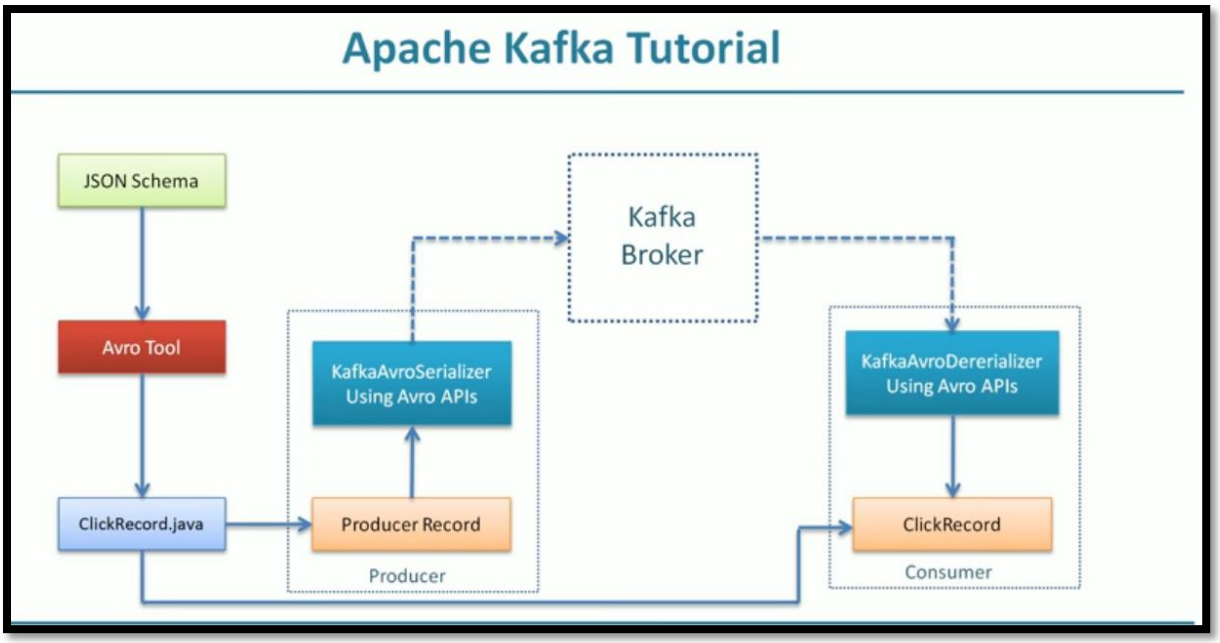
http://avro.apache.org/releases.html#Download



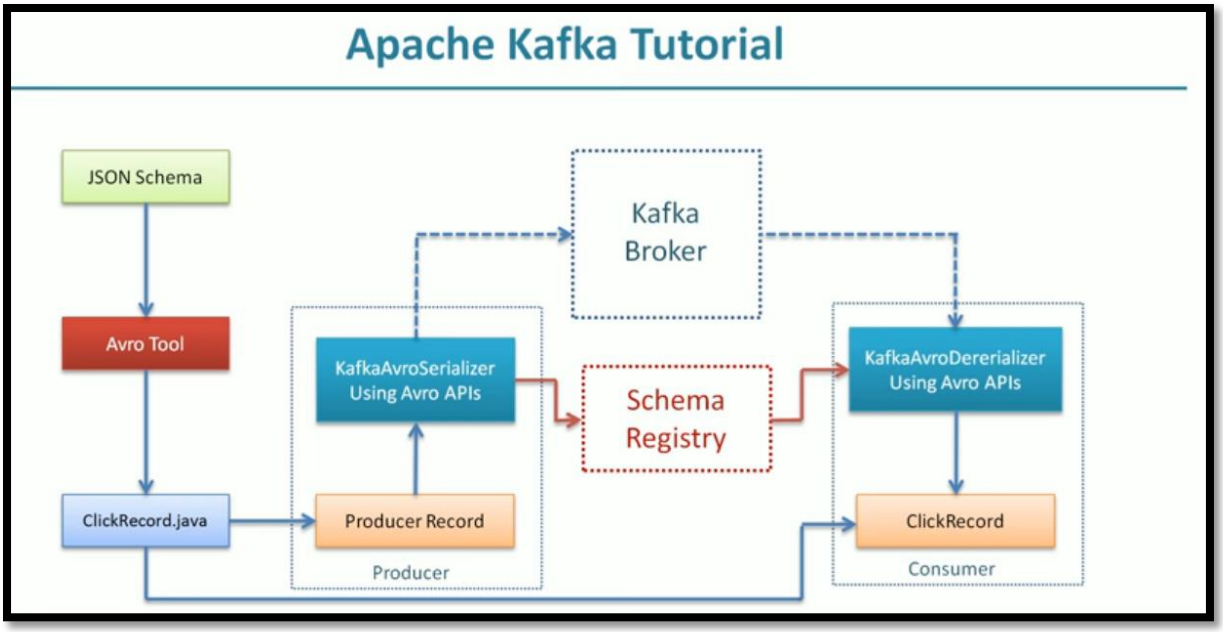


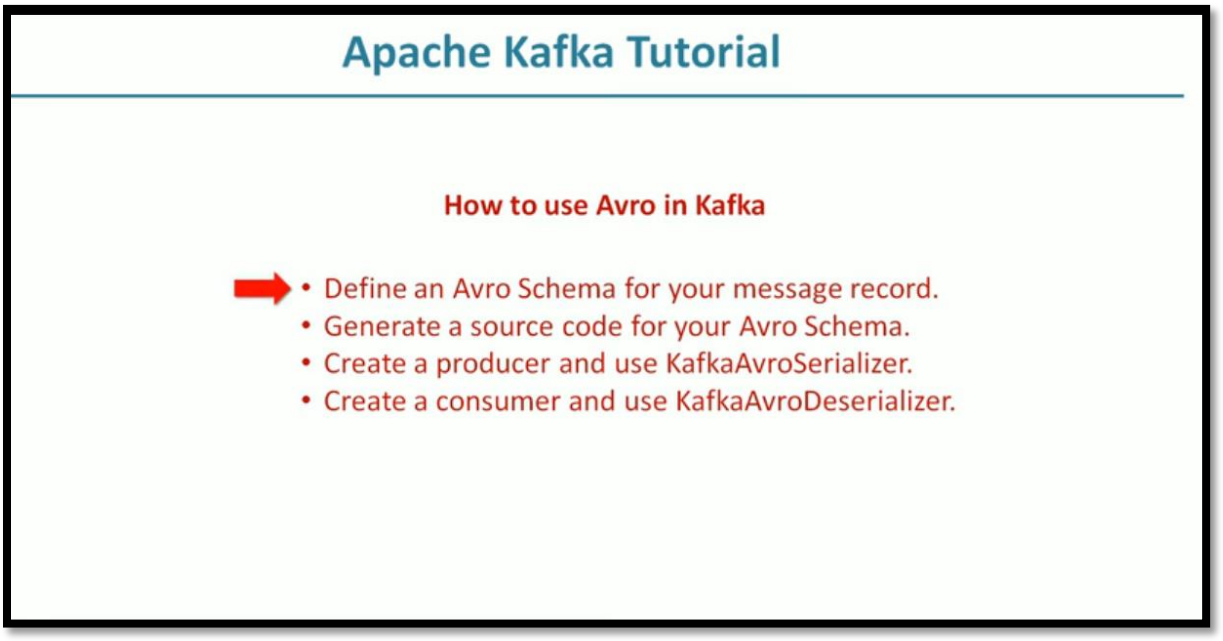
Schema definition





The deseralizer should now the schema without knowing the schema it can't deserialize raw bytes back to an object that's where the schema registry is useful the kafka avro serializer will stor the schema definition into the scheam registry and include an ID of the schema into the message record when kafka avro deserializer sees the message it take the schema id from the message and get the schem defination from the schema registry,once we have the schema details and message byte it is simple to deserialize them






```
import java.util.*;
import org.apache.kafka.clients.producer.*;
public class AvroProducer {

   public static void main(String[] args) throws Exception{

      String topicName = "AvroClicks";
      String msg;
```

```
        Properties props = new Properties();
        props.put("bootstrap.servers", "localhost:9092,localhost:9093");
        props.put("key.serializer","org.apache.kafka.common.serialization.StringSerializer");
        props.put("value.serializer", "io.confluent.kafka.serializers.KafkaAvroSerializer");
//
        props.put("schema.registry.url", "http://localhost:8081");

        Producer<String, ClickRecord> producer = new KafkaProducer <>(props);
        ClickRecord cr = new ClickRecord();
        try{
           cr.setSessionId("10001");
           cr.setChannel("HomePage");
           cr.setIp("192.168.0.1");

           producer.send(new ProducerRecord<String,
ClickRecord>(topicName,cr.getSessionId().toString(),cr)).get();

           System.out.println("Complete");
        }
        catch(Exception ex){
           ex.printStackTrace(System.out);
        }
        finally{
           producer.close();
        }

     }
}
```

ClickRecordV1.avsc

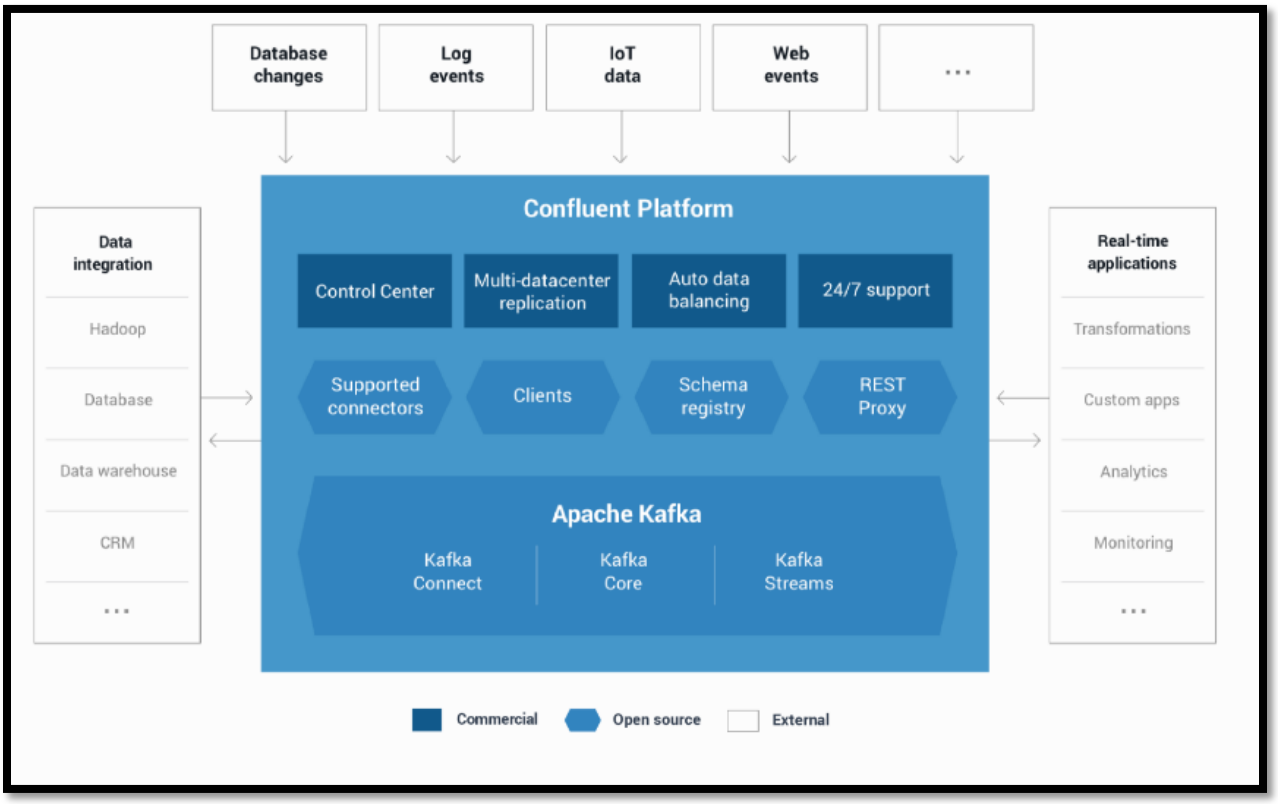
```
 { "type":
 "record",
               "name": "ClickRecord",
               "fields": [
                  {"name": "session_id", "type": "string"},
                  {"name": "browser", "type": ["string", "null"]},
                  {"name": "campaign", "type": ["string", "null"]},
                  {"name": "channel", "type": "string"},
                  {"name": "referrer", "type": ["string", "null"], "default": "None"},
                  {"name": "ip", "type": ["string", "null"]}
                ]
             }
```

```
import java.util.*;
import org.apache.kafka.clients.consumer.*;


public class AvroConsumer{

   public static void main(String[] args) throws Exception{

      String topicName = "AvroClicks";

      String groupName = "RG";
      Properties props = new Properties();
      props.put("bootstrap.servers", "localhost:9092,localhost:9093");
      props.put("group.id", groupName);
      props.put("key.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");
      props.put("value.deserializer", "io.confluent.kafka.serializers.KafkaAvroDeserializer");
      props.put("schema.registry.url", "http://localhost:8081");
//this is mandatory for avro read
      props.put("specific.avro.reader", "true");

      KafkaConsumer<String, ClickRecord> consumer = new KafkaConsumer<>(props);
      consumer.subscribe(Arrays.asList(topicName));
      try{
         while (true){
            ConsumerRecords<String, ClickRecord> records = consumer.poll(100);
            for (ConsumerRecord<String, ClickRecord> record : records){
                  System.out.println("Session id="+ record.value().getSessionId()
                              + " Channel=" + record.value().getChannel()
                              + " Referrer=" + record.value().getReferrer());
               }
            }
         }catch(Exception ex){
            ex.printStackTrace();
         }
         finally{
            consumer.close();
         }
   }

}
```

For that we need to download confluent platform

Confluent Platform

The Confluent Platform makes it easy build real-time data pipelines and streaming applications. By integrating data from multiple sources and locations into a single, central streaming platform for your company, the Confluent Platform lets you focus on how to derive business value from your data rather than worrying about the underlying mechanics such as how data is being transported or massaged between various systems. Specifically, the Confluent Platform simplifies connecting data sources to Kafka, building applications with Kafka, as well as securing, monitoring, and managing your Kafka infrastructure.

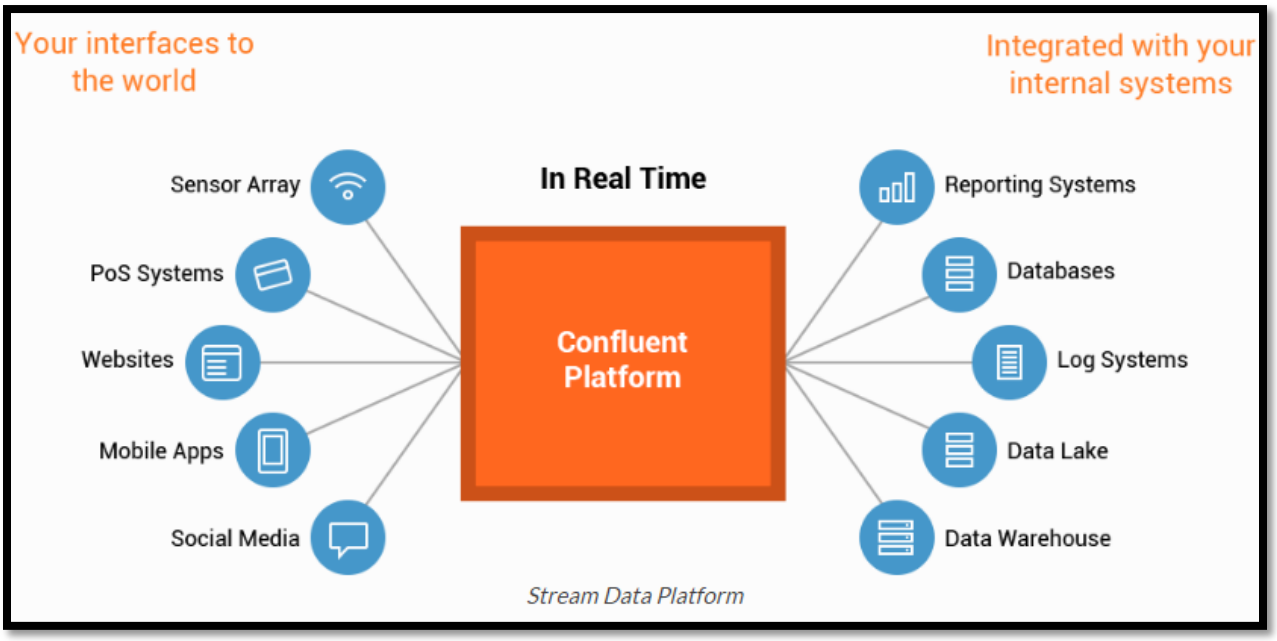


As its backbone, the Confluent Platform leverages Apache Kafka, the most popular open source distributed streaming platform. It has three key capabilities:

- It lets you publish and subscribe to streams of records
- It lets you store streams of records in a fault tolerant way
- It lets you process streams of records

In confluent platform Apache Kafka package together with additional components, all these elements together with apache Kafka make it a powerful and flexible streaming platform

The Confluent Platform is a streaming platform that enables you to organize and manage data from many different sources with one reliable, high performance system.



Confluent Schema Registry

One of the most difficult challenges with loosely coupled systems is ensuring compatibility of data and code as the system grows and evolves. With a messaging service like Kafka, services that interact with each other must agree on a common format, called a *schema*, for messages. In many systems, these formats are ad hoc, only implicitly defined by the code, and often are duplicated across each system that uses that message type.

As requirements change, it becomes necessary to evolve these formats. With only an ad-hoc definition, it is very difficult for developers to determine what the impact of their change might be.

The Confluent Schema Registry enables safe, zero downtime evolution of schemas by centralizing the management of schemas written for the Avro serialization system. It tracks all versions of schemas used for every topic in Kafka and only allows evolution of schemas according to user-defined compatibility settings. This gives developers confidence that they can safely modify schemas as necessary without worrying that doing so will break a different service they may not even be aware of.

The Schema Registry also includes plugins for Kafka clients that handle schema storage and retrieval for Kafka messages that are sent in the Avro format. This integration is seamless – if you are already using Kafka with Avro data, using the Schema Registry only requires including the serializers with your application and changing one setting.

The Confluent Schema Registry is available as open source software under the Apache License v2.0 license. Source code at https://github.com/confluentinc/schema-registry.

The Confluent Schema Registry is available as open source software under the Apache License v2.0 license. Source code at https://github.com/confluentinc/schema-registry.

Installation Step on Centos7

Please follow below link for install confluent platform

http://docs.confluent.io/3.1.2/installation.html

First install Confluent's public key, which is used to sign packages in the yum repository.

```
$ sudo rpm --import http://packages.confluent.io/rpm/3.1/archive.key
```

Add the repository to your **/etc/yum.repos.d/** directory in a file named **confluent.repo**

If you are using RHEL/Centos/Oracle 7

```
[Confluent.dist]
name=Confluent repository (dist)
baseurl=http://packages.confluent.io/rpm/3.1/7
gpgcheck=1
gpgkey=http://packages.confluent.io/rpm/3.1/archive.key
enabled=1

[Confluent]
name=Confluent repository
baseurl=http://packages.confluent.io/rpm/3.1
gpgcheck=1
gpgkey=http://packages.confluent.io/rpm/3.1/archive.key
enabled=1
```

It is recommended to clear the yum caches before proceeding:

```
$ sudo yum clean all
```

The repository is now ready for use.

You can install Confluent Open Source with:

```
$ sudo yum install confluent-platform-oss-2.11
```

The number at the end of the package name specifies the Scala version. Currently supported versions are 2.11 (recommended) and 2.10.

Here 2.11 is Scala version

```
[root@mac92 yum.repos.d]# sudo yum install confluent-platform-oss-2.11
Loaded plugins: fastestmirror, langpacks
Repository HDP-UTILS-1.1.0.20 is listed more than once in the configuration
Confluent                                                 | 2.9 kB  00:00:00
Confluent.dist                                            | 2.9 kB  00:00:00
HDP-2.3                                                   | 2.9 kB  00:00:00
HDP-2.3.4.0                                               | 2.9 kB  00:00:00
HDP-UTILS-1.1.0.20                                        | 2.9 kB  00:00:00
Updates-ambari-2.2.0.0                                    | 2.9 kB  00:00:00
base                                                      | 3.6 kB  00:00:00
cloudera-cdh5                                             |  951 B  00:00:00
cloudera-manager                                          | 2.9 kB  00:00:00
epel/x86_64/metalink                                      | 6.1 kB  00:00:00
epel                                                      | 4.3 kB  00:00:00
extras                                                    | 3.4 kB  00:00:00
mesosphere                                                | 2.9 kB  00:00:00
mesosphere-noarch                                         | 2.9 kB  00:00:00
mysql-connectors-community                                | 2.5 kB  00:00:00
```

```
mysql-tools-community                                                | 2.5 kB  00:00:00
mysql56-community                                                    | 2.5 kB  00:00:00
updates                                                              | 3.4 kB  00:00:00
(1/19): HDP-2.3.4.0/primary_db                                       |  61 kB  00:00:00
(2/19): HDP-2.3/primary_db                                           |  61 kB  00:00:00
(3/19): Updates-ambari-2.2.0.0/primary_db                            | 5.8 kB  00:00:00
(4/19): HDP-UTILS-1.1.0.20/primary_db                                |  28 kB  00:00:00
(5/19): cloudera-manager/primary_db                                  |  11 kB  00:00:00
(6/19): Confluent/primary_db                                         |  17 kB  00:00:00
(7/19): Confluent.dist/primary_db                                    | 8.5 kB  00:00:00
(8/19): base/7/x86_64/group_gz                                       | 155 kB  00:00:00
(9/19): extras/7/x86_64/primary_db                                   | 121 kB  00:00:00
(10/19): mesosphere/x86_64/primary_db                                |  35 kB  00:00:00
(11/19): mesosphere-noarch/primary_db                                | 2.5 kB  00:00:00
(12/19): mysql-connectors-community/x86_64/primary_db                |  13 kB
00:00:00
(13/19): mysql-tools-community/x86_64/primary_db                     |  32 kB
00:00:00
(14/19): mysql56-community/x86_64/primary_db                         | 159 kB  00:00:00
(15/19): base/7/x86_64/primary_db                                    | 5.6 MB  00:00:01
(16/19): epel/x86_64/group_gz                                        | 170 kB  00:00:01
(17/19): updates/7/x86_64/primary_db                                 | 2.2 MB  00:00:01
(18/19): epel/x86_64/updateinfo                                      | 732 kB  00:00:03
(19/19): epel/x86_64/primary_db                                      | 4.5 MB  00:00:10
```

```
cloudera-cdh5/primary                                                    | 43 kB  00:00:00

Determining fastest mirrors

 * base: centos.excellmedia.net

 * epel: epel.mirror.net.in

 * extras: centos.excellmedia.net

 * updates: centos.excellmedia.net

cloudera-cdh5                                                             146/146

Resolving Dependencies

--> Running transaction check

---> Package confluent-platform-oss-2.11.noarch 0:3.1.2-1 will be installed

--> Processing Dependency: confluent-kafka-connect-elasticsearch >= 3.1.2 for package: confluent-
platform-oss-2.11-3.1.2-1.noarch

--> Processing Dependency: confluent-kafka-connect-hdfs >= 3.1.2 for package: confluent-platform-oss-
2.11-3.1.2-1.noarch

--> Processing Dependency: confluent-camus >= 3.1.2 for package: confluent-platform-oss-2.11-3.1.2-
1.noarch

--> Processing Dependency: confluent-kafka-connect-jdbc >= 3.1.2 for package: confluent-platform-oss-
2.11-3.1.2-1.noarch

--> Processing Dependency: confluent-schema-registry >= 3.1.2 for package: confluent-platform-oss-
2.11-3.1.2-1.noarch

--> Processing Dependency: confluent-kafka-2.11 >= 0.10.1.1 for package: confluent-platform-oss-2.11-
3.1.2-1.noarch

--> Processing Dependency: confluent-kafka-rest >= 3.1.2 for package: confluent-platform-oss-2.11-
3.1.2-1.noarch

--> Running transaction check

---> Package confluent-camus.noarch 0:3.1.2-1 will be installed

---> Package confluent-kafka-2.11.noarch 0:0.10.1.1-1 will be installed
```

---> Package confluent-kafka-connect-elasticsearch.noarch 0:3.1.2-1 will be installed

--> Processing Dependency: confluent-common for package: confluent-kafka-connect-elasticsearch-3.1.2-1.noarch

---> Package confluent-kafka-connect-hdfs.noarch 0:3.1.2-1 will be installed

---> Package confluent-kafka-connect-jdbc.noarch 0:3.1.2-1 will be installed

---> Package confluent-kafka-rest.noarch 0:3.1.2-1 will be installed

--> Processing Dependency: confluent-rest-utils for package: confluent-kafka-rest-3.1.2-1.noarch

---> Package confluent-schema-registry.noarch 0:3.1.2-1 will be installed

--> Running transaction check

---> Package confluent-common.noarch 0:3.1.2-1 will be installed

---> Package confluent-rest-utils.noarch 0:3.1.2-1 will be installed

--> Finished Dependency Resolution

Dependencies Resolved

Package	Arch	Version	Repository	Size
Installing:				
confluent-platform-oss-2.11	noarch	3.1.2-1	Confluent	6.6 k
Installing for dependencies:				
confluent-camus	noarch	3.1.2-1	Confluent	19 M
confluent-common	noarch	3.1.2-1	Confluent	2.0 M
confluent-kafka-2.11	noarch	0.10.1.1-1	Confluent	36 M

```
 confluent-kafka-connect-elasticsearch noarch 3.1.2-1 Confluent 4.3
M

 confluent-kafka-connect-hdfs noarch 3.1.2-1 Confluent 86 M

 confluent-kafka-connect-jdbc noarch 3.1.2-1 Confluent 6.0 M

 confluent-kafka-rest noarch 3.1.2-1 Confluent 16 M

 confluent-rest-utils noarch 3.1.2-1 Confluent 7.0 M

 confluent-schema-registry noarch 3.1.2-1 Confluent 26 M


Transaction Summary

================================================================================
================================================

Install  1 Package (+9 Dependent packages)


Total download size: 203 M

Installed size: 229 M

Is this ok [y/d/N]: y

Downloading packages:

(1/10): confluent-common-3.1.2-1.noarch.rpm                    | 2.0 MB  00:00:04

(2/10): confluent-kafka-2.11-0.10.1.1-1.noarch.rpm             |  36 MB  00:00:24

(3/10): confluent-camus-3.1.2-1.noarch.rpm                     |  19 MB  00:00:33

(4/10): confluent-kafka-connect-elasticsearch-3.1.2-1.noarch.rpm | 4.3 MB
00:00:07

(5/10): confluent-kafka-connect-jdbc-3.1.2-1.noarch.rpm         | 6.0 MB
00:00:07

(6/10): confluent-kafka-rest-3.1.2-1.noarch.rpm                 |  16 MB  00:00:09
```

```
(7/10): confluent-platform-oss-2.11-3.1.2-1.noarch.rpm                    | 6.6 kB  00:00:00
(8/10): confluent-rest-utils-3.1.2-1.noarch.rpm                           | 7.0 MB  00:00:12
(9/10): confluent-schema-registry-3.1.2-1.noarch.rpm                      |  26 MB  00:00:25
(10/10): confluent-kafka-connect-hdfs-3.1.2-1.noarch.rpm                  |  86 MB
00:01:00
----------------------------------------------------------------------------------------------------------------------------------
Total                                                  2.2 MB/s | 203 MB  00:01:33
Running transaction check
Running transaction test
Transaction test succeeded
Running transaction
  Installing : confluent-common-3.1.2-1.noarch                                1/10
  Installing : confluent-rest-utils-3.1.2-1.noarch                            2/10
  Installing : confluent-schema-registry-3.1.2-1.noarch                       3/10
  Installing : confluent-kafka-rest-3.1.2-1.noarch                            4/10
  Installing : confluent-kafka-connect-jdbc-3.1.2-1.noarch                    5/10
  Installing : confluent-kafka-connect-elasticsearch-3.1.2-1.noarch           6/10
  Installing : confluent-kafka-connect-hdfs-3.1.2-1.noarch                    7/10
  Installing : confluent-camus-3.1.2-1.noarch                                 8/10
  Installing : confluent-kafka-2.11-0.10.1.1-1.noarch                         9/10
  Installing : confluent-platform-oss-2.11-3.1.2-1.noarch                     10/10
  Verifying  : confluent-kafka-2.11-0.10.1.1-1.noarch                         1/10
  Verifying  : confluent-platform-oss-2.11-3.1.2-1.noarch                     2/10
  Verifying  : confluent-schema-registry-3.1.2-1.noarch                       3/10
```

```
  Verifying  : confluent-kafka-rest-3.1.2-1.noarch                                 4/10
  Verifying  : confluent-kafka-connect-jdbc-3.1.2-1.noarch                         5/10
  Verifying  : confluent-kafka-connect-elasticsearch-3.1.2-1.noarch                6/10
  Verifying  : confluent-camus-3.1.2-1.noarch                                      7/10
  Verifying  : confluent-rest-utils-3.1.2-1.noarch                                 8/10
  Verifying  : confluent-kafka-connect-hdfs-3.1.2-1.noarch                         9/10
  Verifying  : confluent-common-3.1.2-1.noarch                                    10/10

Installed:
  confluent-platform-oss-2.11.noarch 0:3.1.2-1

Dependency Installed:
  confluent-camus.noarch 0:3.1.2-1                confluent-common.noarch 0:3.1.2-1
  confluent-kafka-2.11.noarch 0:0.10.1.1-1        confluent-kafka-connect-elasticsearch.noarch
0:3.1.2-1
  confluent-kafka-connect-hdfs.noarch 0:3.1.2-1   confluent-kafka-connect-jdbc.noarch 0:3.1.2-1
  confluent-kafka-rest.noarch 0:3.1.2-1           confluent-rest-utils.noarch 0:3.1.2-1
  confluent-schema-registry.noarch 0:3.1.2-1

Complete!
[root@mac92 yum.repos.d]#
```

Components are installed under **/etc/kafka**, **/etc/kafka-rest**, **/etc/schema-registry** and **/etc/camus**.

Start zookeeper, Kafka and schema registry

```
$ zookeeper-server-start /etc/kafka/zookeeper.properties
$kafka-server-start /etc/kafka/server.properties
$schema-registry-start /etc/schema-registry/schema-registry.properties
```

Schema registry default listen on 8081 check schema-registry.properties file

```
[root@mac92 schema-registry]# schema-registry-start /etc/schema-registry/schema-registry.properties
[2017-02-08 09:17:49,139] INFO SchemaRegistryConfig values:
	metric.reporters = []
	kafkastore.sasl.kerberos.kinit.cmd = /usr/bin/kinit
	response.mediatype.default = application/vnd.schemaregistry.v1+json
	kafkastore.ssl.trustmanager.algorithm = PKIX
	authentication.realm =
	ssl.keystore.type = JKS
	kafkastore.topic = _schemas
	metrics.jmx.prefix = kafka.schema.registry
	kafkastore.ssl.enabled.protocols = TLSv1.2,TLSv1.1,TLSv1
	kafkastore.topic.replication.factor = 3
	ssl.truststore.password =
	kafkastore.timeout.ms = 500
	host.name = mac92.cybage.com
	kafkastore.bootstrap.servers = []
	schema.registry.zk.namespace = schema_registry
	kafkastore.sasl.kerberos.ticket.renew.window.factor = 0.8
	kafkastore.sasl.kerberos.service.name =
	ssl.endpoint.identification.algorithm =
	compression.enable = false
	kafkastore.ssl.truststore.type = JKS
	avro.compatibility.level = backward
	kafkastore.ssl.protocol = TLS
	kafkastore.ssl.provider =
	kafkastore.ssl.truststore.location =
	response.mediatype.preferred = [application/vnd.schemaregistry.v1+json,
application/vnd.schemaregistry+json, application/json]
	kafkastore.ssl.keystore.type = JKS
	ssl.truststore.type = JKS
	kafkastore.ssl.truststore.password =
	access.control.allow.origin =
	ssl.truststore.location =
	ssl.keystore.password =
	port = 8081
	kafkastore.ssl.keystore.location =
	master.eligibility = true
	ssl.client.auth = false
	kafkastore.ssl.keystore.password =
```

```
	kafkastore.security.protocol = PLAINTEXT
	ssl.trustmanager.algorithm =
	authentication.method = NONE
	request.logger.name = io.confluent.rest-utils.requests
	ssl.key.password =
	kafkastore.zk.session.timeout.ms = 30000
	kafkastore.sasl.mechanism = GSSAPI
	kafkastore.sasl.kerberos.ticket.renew.jitter = 0.05
	kafkastore.ssl.key.password =
	zookeeper.set.acl = false
	authentication.roles = [*]
	metrics.num.samples = 2
	ssl.protocol = TLS
	kafkastore.ssl.keymanager.algorithm = SunX509
	kafkastore.connection.url = localhost:2181
	debug = false
	listeners = [http://0.0.0.0:8081]
	ssl.provider =
	ssl.enabled.protocols = []
	shutdown.graceful.ms = 1000
	ssl.keystore.location =
	ssl.cipher.suites = []
	kafkastore.ssl.endpoint.identification.algorithm =
	kafkastore.ssl.cipher.suites =
	access.control.allow.methods =
	kafkastore.sasl.kerberos.min.time.before.relogin = 60000
	ssl.keymanager.algorithm =
	metrics.sample.window.ms = 30000
	kafkastore.init.timeout.ms = 60000
 (io.confluent.kafka.schemaregistry.rest.SchemaRegistryConfig:169)
[2017-02-08 09:17:50,800] INFO Initializing KafkaStore with broker endpoints:
PLAINTEXT://mac92.cybage.com:9092 (io.confluent.kafka.schemaregistry.storage.KafkaStore:127)
[2017-02-08 09:17:50,815] WARN Creating the schema topic _schemas using a replication factor of 1,
which is less than the desired one of 3. If this is a production environment, it's crucial to add more
brokers and increase the replication factor of the topic.
(io.confluent.kafka.schemaregistry.storage.KafkaStore:258)
[2017-02-08 09:17:51,663] INFO Initialized last consumed offset to -1
(io.confluent.kafka.schemaregistry.storage.KafkaStoreReaderThread:122)
[2017-02-08 09:17:51,664] INFO [kafka-store-reader-thread-_schemas], Starting
(io.confluent.kafka.schemaregistry.storage.KafkaStoreReaderThread:70)
[2017-02-08 09:17:51,881] INFO Wait to catch up until the offset of the last message at 0
(io.confluent.kafka.schemaregistry.storage.KafkaStore:343)
[2017-02-08 09:17:52,113] INFO Created schema registry namespace localhost:2181/schema_registry
(io.confluent.kafka.schemaregistry.storage.KafkaSchemaRegistry:238)
[2017-02-08 09:17:52,179] INFO Successfully elected the new master:
{"host":"mac92.cybage.com","port":8081,"master_eligibility":true,"version":1}
(io.confluent.kafka.schemaregistry.zookeeper.ZookeeperMasterElector:83)
[2017-02-08 09:17:52,229] INFO Successfully elected the new master:
```

```
{"host":"mac92.cybage.com","port":8081,"master_eligibility":true,"version":1}
(io.confluent.kafka.schemaregistry.zookeeper.ZookeeperMasterElector:83)
[2017-02-08 09:17:52,339] INFO Logging initialized @4316ms (org.eclipse.jetty.util.log:186)
[2017-02-08 09:17:52,409] INFO Adding listener: http://0.0.0.0:8081 (io.confluent.rest.Application:174)
[2017-02-08 09:17:52,470] INFO jetty-9.2.12.v20150709 (org.eclipse.jetty.server.Server:327)
[2017-02-08 09:17:53,395] INFO HV000001: Hibernate Validator 5.1.2.Final
(org.hibernate.validator.internal.util.Version:27)
[2017-02-08 09:17:53,625] INFO Started o.e.j.s.ServletContextHandler@5c153b9e{/,null,AVAILABLE}
(org.eclipse.jetty.server.handler.ContextHandler:744)
[2017-02-08 09:17:53,666] INFO Started
NetworkTrafficServerConnector@784c3487{HTTP/1.1}{0.0.0.0:8081}
(org.eclipse.jetty.server.NetworkTrafficServerConnector:266)
[2017-02-08 09:17:53,668] INFO Started @5645ms (org.eclipse.jetty.server.Server:379)
[2017-02-08 09:17:53,669] INFO Server started, listening for requests...
(io.confluent.kafka.schemaregistry.rest.SchemaRegistryMain:45)
```

To test it we can use console command first

```
kafka-avro-console-producer \
         --broker-list localhost:9092 --topic test \
         --property
value.schema='{"type":"record","name":"myrecord","fields":[{"name":"f1","type":"string"}]}'
```

Once started, the process will wait for you to enter messages, one per line, and will send them immediately when you hit the Enter key. Try entering a couple of messages:

```
{"f1": "value1"}
{"f1": "value2"}
{"f1": "value3"}
```

```
[root@mac92 ~]# kafka-avro-console-producer         --broker-list localhost:9092 --topic test         --
property value.schema='{"type":"record","name":"myrecord","fields":[{"name":"f1","type":"string"}]}'
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/share/java/kafka-serde-tools/slf4j-log4j12-
1.7.6.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/share/java/schema-registry/slf4j-log4j12-
1.7.6.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
{"f1": "value1"}
{"f1": "value2"}
{"f1": "value3"}
```

Now start consumer in separate window

```
[root@mac92 ~]# kafka-avro-console-consumer --topic test \
>       --zookeeper localhost:2181 \
>       --from-beginning
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/share/java/kafka-serde-tools/slf4j-log4j12-
1.7.6.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/share/java/schema-registry/slf4j-log4j12-
1.7.6.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
Using the ConsoleConsumer with old consumer is deprecated and will be removed in a future major
release. Consider using the new consumer by passing [bootstrap-server] instead of [zookeeper].
{"f1":"value1"}
{"f1":"value1"}
{"f1":"value2"}
```

You should see all the messages you created in the previous step written to the console in the same format.

The consumer does not exit after reading all the messages so it can listen for and process new messages as they are published. Try keeping the consumer running and repeating step 5 – you will see messages delivered to the consumer immediately after you hit Enter for each message in the producer.

When you're done, shut down the consumer with Ctrl+C.

Now let's try to produce data to the same topic using an incompatible schema. We'll run the producer with nearly the same command, but change the schema to expect plain integers.

```
kafka-avro-console-producer \
        --broker-list localhost:9092 --topic test \
        --property value.schema='{"type":"int"}'
```

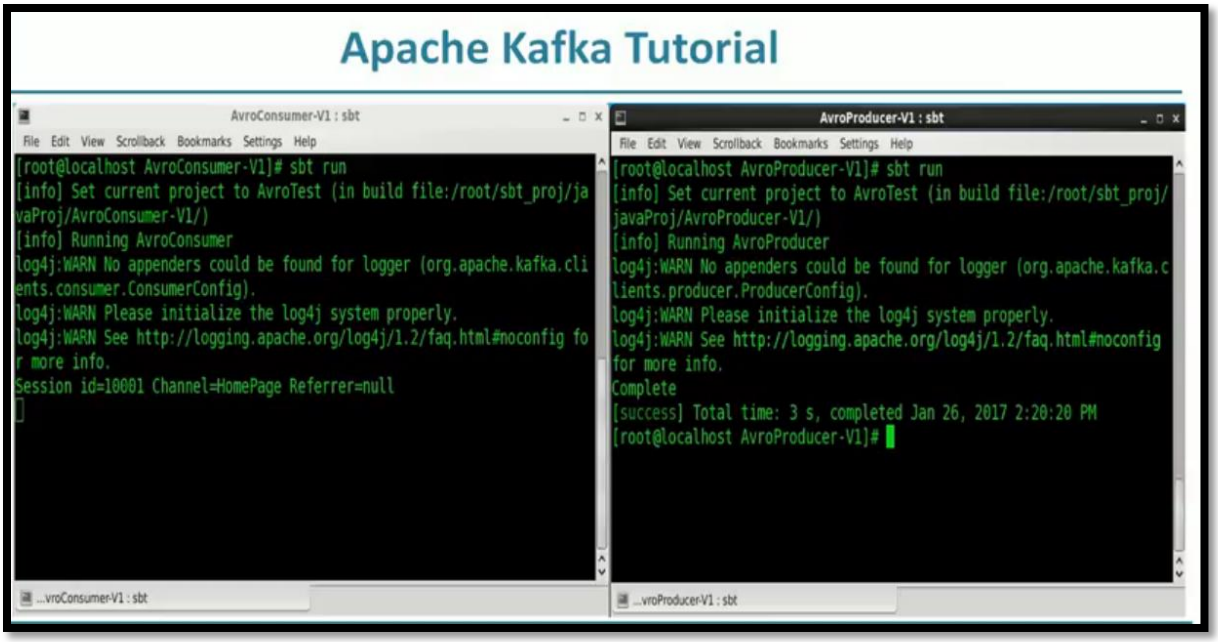
Now if you enter an integer and hit enter, you should see the following (expected) exception:

```
org.apache.kafka.common.errors.SerializationException: Error registering Avro schema: "int"
Caused by: io.confluent.kafka.schemaregistry.client.rest.exceptions.RestClientException:
Schema being registered is incompatible with the latest schema; error code: 409
       at
io.confluent.kafka.schemaregistry.client.rest.utils.RestUtils.httpRequest(RestUtils.java:146)
       at
io.confluent.kafka.schemaregistry.client.rest.utils.RestUtils.registerSchema(RestUtils.java:17
4)
       at
io.confluent.kafka.schemaregistry.client.CachedSchemaRegistryClient.registerAndGetId(CachedSch
emaRegistryClient.java:51)
       at
io.confluent.kafka.schemaregistry.client.CachedSchemaRegistryClient.register(CachedSchemaRegis
```

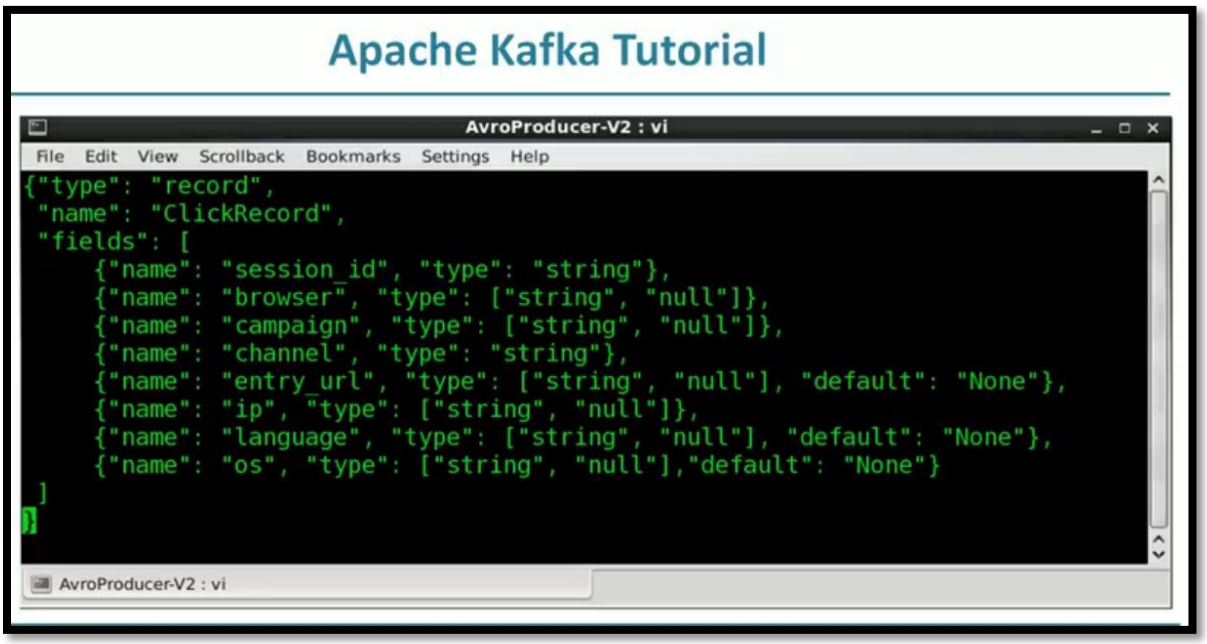
```
tryClient.java:89)
        at
io.confluent.kafka.serializers.AbstractKafkaAvroSerializer.serializeImpl(AbstractKafkaAvroSeri
alizer.java:49)
        at
io.confluent.kafka.formatter.AvroMessageReader.readMessage(AvroMessageReader.java:155)
        at kafka.tools.ConsoleProducer$.main(ConsoleProducer.scala:94)
        at kafka.tools.ConsoleProducer.main(ConsoleProducer.scala)
```

It is because we have change the registered schema with topic

```
kafka-avro-console-producer \
>       --broker-list localhost:9092 --topic test \
>       --property value.schema='{"type":"int"}'
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/share/java/kafka-serde-tools/slf4j-log4j12-
1.7.6.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/share/java/schema-registry/slf4j-log4j12-
1.7.6.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
123
org.apache.kafka.common.errors.SerializationException: Error registering Avro schema: "int"
Caused by: io.confluent.kafka.schemaregistry.client.rest.exceptions.RestClientException: Schema being
registered is incompatible with the latest schema; error code: 409
    at io.confluent.kafka.schemaregistry.client.rest.RestService.sendHttpRequest(RestService.java:170)
    at io.confluent.kafka.schemaregistry.client.rest.RestService.httpRequest(RestService.java:187)
    at io.confluent.kafka.schemaregistry.client.rest.RestService.registerSchema(RestService.java:238)
    at io.confluent.kafka.schemaregistry.client.rest.RestService.registerSchema(RestService.java:230)
    at io.confluent.kafka.schemaregistry.client.rest.RestService.registerSchema(RestService.java:225)
    at
io.confluent.kafka.schemaregistry.client.CachedSchemaRegistryClient.registerAndGetId(CachedSchema
RegistryClient.java:59)
    at
io.confluent.kafka.schemaregistry.client.CachedSchemaRegistryClient.register(CachedSchemaRegistryCli
ent.java:91)
    at
io.confluent.kafka.serializers.AbstractKafkaAvroSerializer.serializeImpl(AbstractKafkaAvroSerializer.java:
72)
    at io.confluent.kafka.formatter.AvroMessageReader.readMessage(AvroMessageReader.java:158)
    at kafka.tools.ConsoleProducer$.main(ConsoleProducer.scala:55)
    at kafka.tools.ConsoleProducer.main(ConsoleProducer.scala)
[root@mac92 ~]#
```



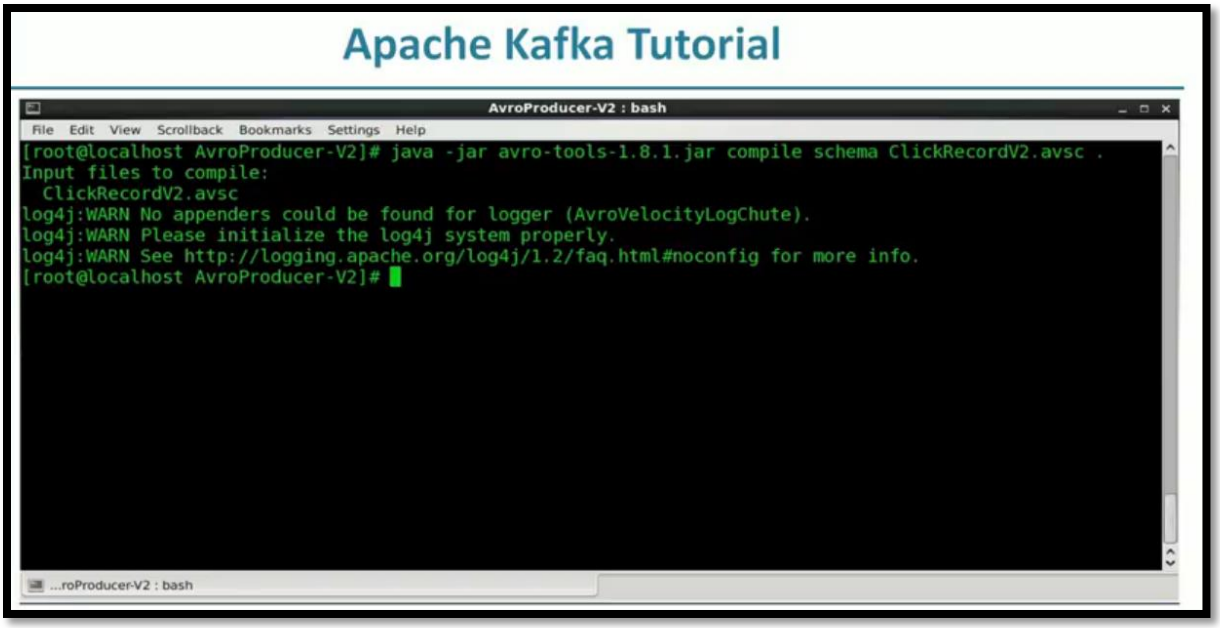
Now change the schema , name and type should be the same as old schema



You cannot change the schema randomly there are some rule for avro schema resolution check below link

https://avro.apache.org/docs/current/spec.html#Schema+Resolution

Generate class for new schema

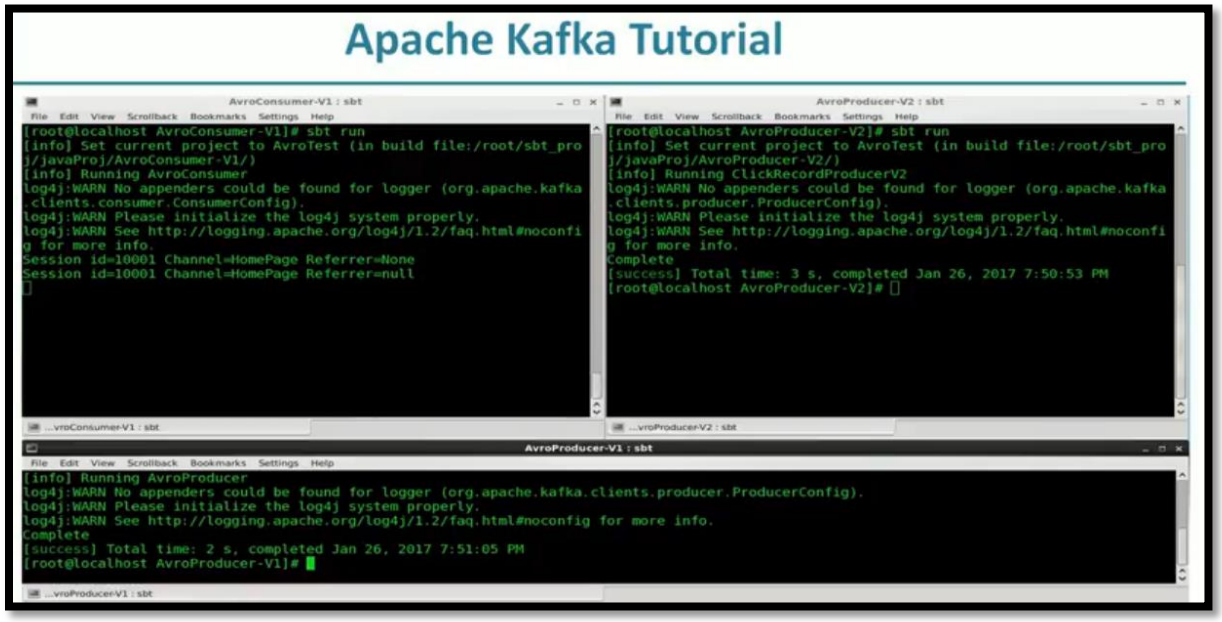


Create new producer

```
import java.util.*;
import org.apache.kafka.clients.producer.*;
public class ClickRecordProducerV2 {

   public static void main(String[] args) throws Exception{

      String topicName = "AvroClicks";
      String msg;

      Properties props = new Properties();
      props.put("bootstrap.servers", "localhost:9092,localhost:9093");
      props.put("key.serializer","org.apache.kafka.common.serialization.StringSerializer");
      props.put("value.serializer", "io.confluent.kafka.serializers.KafkaAvroSerializer");
      props.put("schema.registry.url", "http://localhost:8081");

      Producer<String, ClickRecord> producer = new KafkaProducer <>(props);
      ClickRecord cr = new ClickRecord();
      try{
         cr.setSessionId("10001");
         cr.setChannel("HomePage");
         cr.setIp("192.168.0.1");
         cr.setLanguage("Spanish");
         cr.setOs("Mac");
         cr.setEntryUrl("http://facebook.com/myadd");
```

```
        producer.send(new ProducerRecord<String,
ClickRecord>(topicName,cr.getSessionId().toString(),cr)).get();

        System.out.println("Complete");
      }
      catch(Exception ex){
        ex.printStackTrace(System.out);
      }
      finally{
        producer.close();
      }

  }
}
```

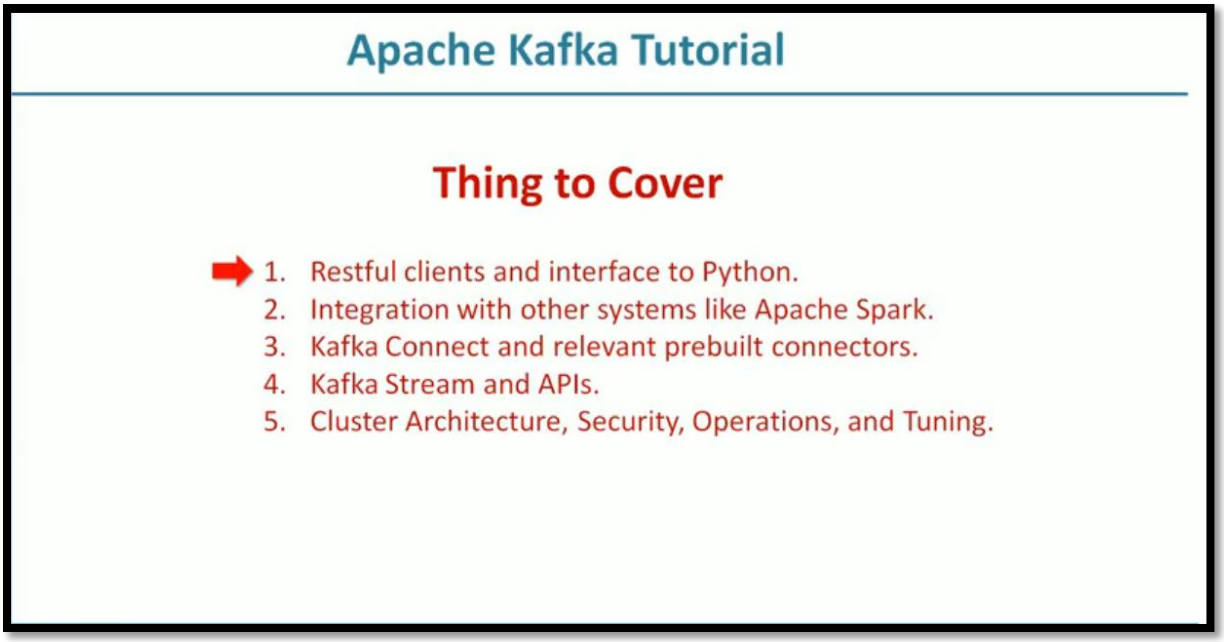
Code is similar just we use new ClickRecord.java class



Old consumer can read both schema but it will produce null values for new added field we can create new consumer to read new fields

```
import java.util.*;
import org.apache.kafka.clients.consumer.*;


public class ClickRecordConsumerV2{
```

```
   public static void main(String[] args) throws Exception{

      String topicName = "AvroClicks";

      String groupName = "RG";
      Properties props = new Properties();
      props.put("bootstrap.servers", "localhost:9092,localhost:9093");
      props.put("group.id", groupName);
      props.put("key.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");
      props.put("value.deserializer", "io.confluent.kafka.serializers.KafkaAvroDeserializer");
      props.put("schema.registry.url", "http://localhost:8081");
      props.put("specific.avro.reader", "true");

      KafkaConsumer<String, ClickRecord> consumer = new KafkaConsumer<>(props);
      consumer.subscribe(Arrays.asList(topicName));
      try{
         while (true){
            ConsumerRecords<String, ClickRecord> records = consumer.poll(100);
            for (ConsumerRecord<String, ClickRecord> record : records){
                  System.out.println("Session id="+ record.value().getSessionId()
                              + " Channel=" + record.value().getChannel()
                              + " Entry URL=" + record.value().getEntryUrl()
                              + " Language=" + record.value().getLanguage());
              }
            }
         }catch(Exception ex){
            ex.printStackTrace();
         }
         finally{
            consumer.close();
         }
   }

}
```



https://www.safaribooksonline.com/library/view/kafka-the-definitive/9781491936153/ch02.html