# Kafka streams

Kafka streams is an alternate for apache spark, Flink or Nifi

Kafka streams is a Kafka client library for processing & analysing the data present in the broker

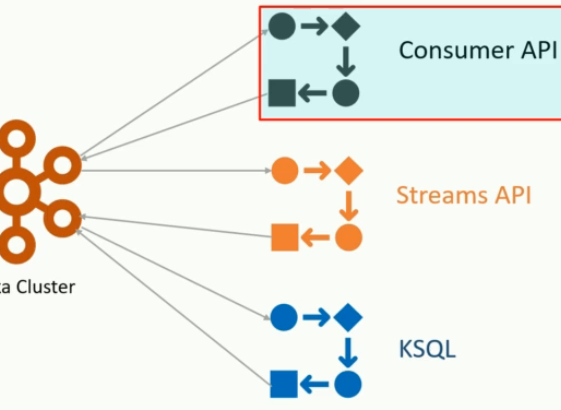
Core functionality of Kafka is available in 2 flavours Streams DSL, Processor API

Streams DSL is a high level api –recommended way it will cover most of our needs

Processor API provides

Data Lake is stationery

Stream – flow of data, or data in motion



From Kafka cluster we can read data in 3 ways as shown above

Streams library is built on top of Kafka producer and consumer api

When to use stream-

1. if there is any need of monitoring the data continuously like traffic and patient then use streams

Here patient data needs to monitor and send alerts to doctors

1. if there is any need to processing continuous flow of data like ETL then also use streams
2. Generally a typical streams processing appln will read data from 1 topic & transform & write data to another topic

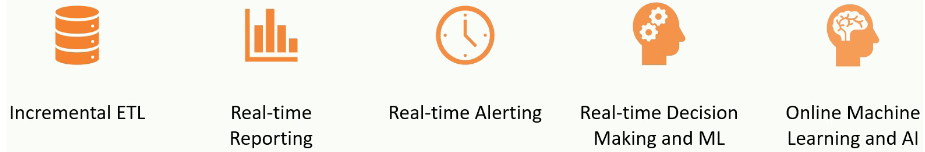
When we are dealing with data which is in motion then it is called stream processing

Definitions: - event is an outcome of an action performed by an actor

Real time use cases:-

1. Flipkart orders - Ordering event , each order in kafka when u place an order and reject an order everything is an event and placed in kafka
2. Stock market trades – real time rates

Stock market updates – means they will immediately send the updates to topic and all users will immediately get the data when all subscribers are subscribed to that topic



1. Advertisement click events- monitors when an user clicked on an add or not , if the user clicked it may go as an event and will be placed in kafka
2. All POS machine UPI transactions – when any person transacted on a pos machine it may put a message to kafka
3. Ordering food in our mobile app
4. Booking a cab- once we book it may send a message to kafka
5. In Facebook reading a post and creating a post which is a continuous flow data

If 10,000 followers are there for that user ( like sundar pichai ) means he will put that post into some topic and whoever follow him means indirectly those consumers are subscribed to that topic then immediately every will get notifications

1. ­Traffic monitoring – traffic updates may come from satellite, here streams are used to monitor this continuously flow of traffic , if high we can send them the alerts
2. Patient health monitoring- sensors will continuously send data here kafka streams will continuously monitor and do some transformations if patient readings are in danger then they will send the alerts
3. Finding fraud transactions- monitor the transaction on bank account and identify if it is fraud or not in real time and stop the transaction if it is fraud
4. Personalized customer experience- flipkart and myntra
5. Real time bidding
6. ETL scenarios- ETL- Extract, transform, load in ETL requirements after extracting it will be like streams and do some transformations and insert data into db

Data coming into my feed, If someone liked by post, commented and tagged me and who commented and liked our post and someone tagged you or if someone tagged ur friend all those events may flow into ur database partition

Misc use cases

1. Infrastructure management –monitor cpu health ram, rom
2. Campaign management- whether recent offers in myntra is driving traffic or not

## Design considerations for data integration solutions

2 applications should be integrated by following points

1. Time sensitivity 2) Decoupling 3)Data format evolution 4) Reliability 5) scalability 6)D
2. Decoupling

Says we should minimize the dependency between source and destination systems within milliseconds we should consume it should be very fast

1. We cant constraint the data format , source system may not send messages to target system in desired formats we should have flexibility to convert the messages

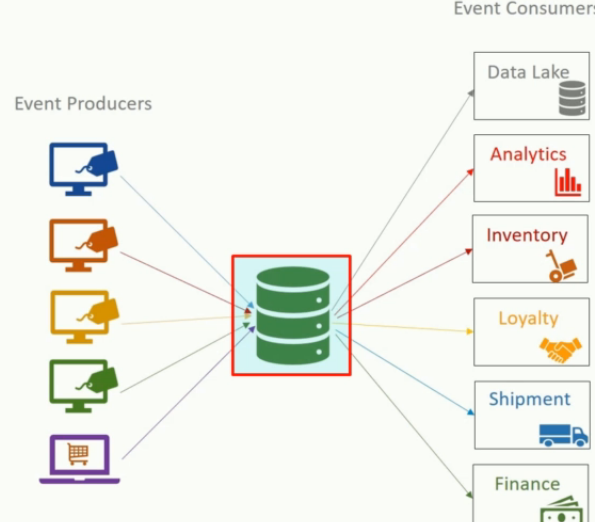
Src may send json and target may need only xml

1. Reliability- even if the consumer went down offline for some times messages should be stored until customer comes online – kafka does that, all messages will be stored in broker partition’s file segment means messages are written to files so those are reliable
2. Scalability – we want horizontal scaling this data integration solution must support at least millions of messages per second

Ways of communication

1. Shared database
2. File transfer – here will transfer file by file
3. Messaging
4. RPC remote procedure call – Its is by rest calls – worst for scalable application

### **1) Shared database- its waste**

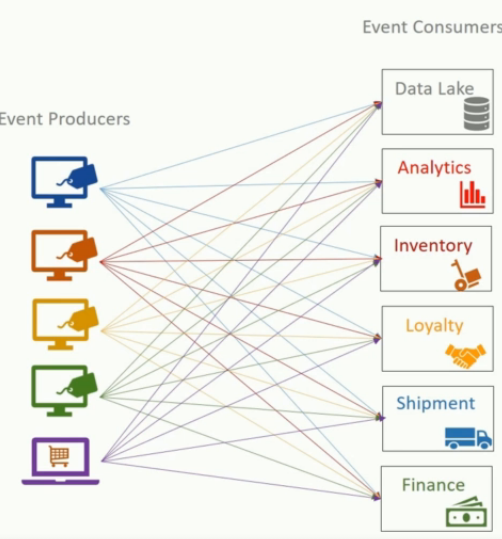
1) This is a good approach for storing now and query later not good to deal with the data in motion and not good for realtime and to deal in milliseconds

2) reg decoupling – all consumers may not support RDBMS and 3) not good for data format evolution because producer may send json and consumer may need xml

5) and this is not horizontally scalable not but the problem is   
**RMI- RPC**

Fails on decoupling (data exchange formats) and scalability , for million messages should we do million rest calls

### **Why we need Kafka Messaging than REST service calls**



**1) consumer offline :-** remote method invocation this is like REST service call, we can’t send millions of data over REST, main issue – what if the consumer is offline so is chain broken

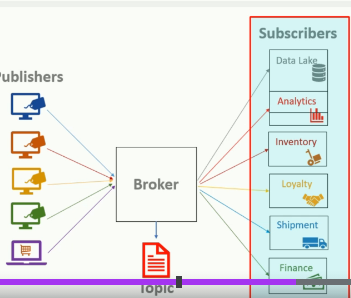
, in kafka all messages will be stored in topic-folder-files whenever he is online he can consume all the messages from the beginning

1. **Resend same message to all consumers –** un necessary REST calls if we want to send millions of same messages to all 5-6 consumers its terrible to make 6 REST calls to 6 services if one is down how can the producer send the same data again

**Ex:**- a) sending invoice message to all 5 consumers, if kafka topics are not used then we have to do same rest call to all 5 consumers so 4 times waste REST svc call

b) That to there is no guarantee that all 5 consumers will be online

Hence if same message is needed by multiple consumers then use kafka

   
Here all publishers will send message to the broker, consumer will poll for the messages

Kafka Streams API vs Kafka Client API

If u have kafka transformation logics then u might need kafka streams API, else for ordinary consumption and storing u can happily use kafka client API

Features of Kafka streams

We can reuse the Kafka streams, these are not like java streams, as java streams once we used we cannot re use the same stream

In java, we will create a stream from the collection , similarly here we will create a stream from a topic, wherever u go we will create a stream on some data source

Streams vertical scaling

Vertical scaling means, scaling on same machine/same hardware, in single machine create multiple threads ex:- increasing RAM and ROM on same machine

Props.put(Streamsconfig.NUM\_STREAM\_THREADS\_CONFIG,3); //vertical scaling

Understand 3 threads means 3 should do diff task (if all 3 threads do same task of reading from same partition it’s called duplicate processing)hence all 3 threads will do separate tasks of reading data from separate partitions

In general topology means “how things are interconnected”

Topology means end-end stream operations right from consuming a stream, on stream we are performing some operations like creating a stream, filtering, writing result data to another topic right this entire thing is called topology

StreamsBuilder sb=new StreamsBuilder();*//This is to create a stream on a topic*KStream<Integer, String> stream = sb.stream(ApplicationConstants.*topicName*);  
System.*out*.println("stream is created on a topic "+ApplicationConstants.*topicName*);  
  
stream.foreach((k,v)->{  
 System.*out*.printf("\n received kafka message as key--> %s, value --> %s",k,v);  
});  
  
**Topology topology=sb.build(); //Here stream topology means the stream just pointed to a topic & it consumes and prints the data**Props.put(Streamsconfig.NUM\_STREAM\_THREADS\_CONFIG,3);//as u see the above topology when we configured 3 threads

Each thread will perform 1 task (each task is a copy of entire topology object)so each thread will perform separate task of reading data from separate partition

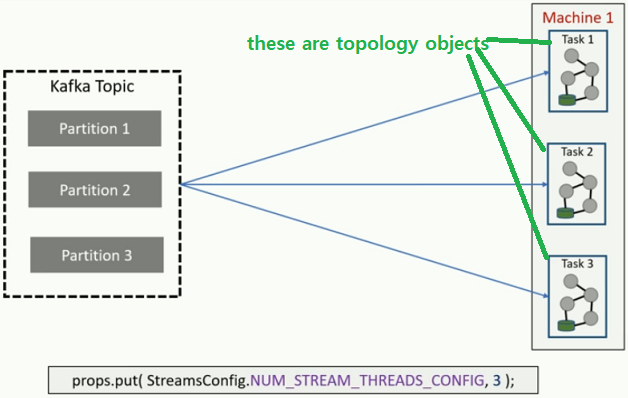
Kafka streams will divide and share the partition in case of multi-threaded scenario

KafkaStreams ks=new KafkaStreams(topology,p);  
ks.start();  
System.*out*.println("stream started");

Each thread will perform separate task (like each thread will read data from separate partition when whole topology is reading data from a topic which have many partitions)

Task means: each task will create a copy of topology objects

Number of tasks = no of partitions in input topic



the framework will see how many partitions are then it will create those many logical task

no of logical task=no of partitions

if u have only 1 thread all tasks will be performed by single thread, means 1 thread itself will read data from all partitions

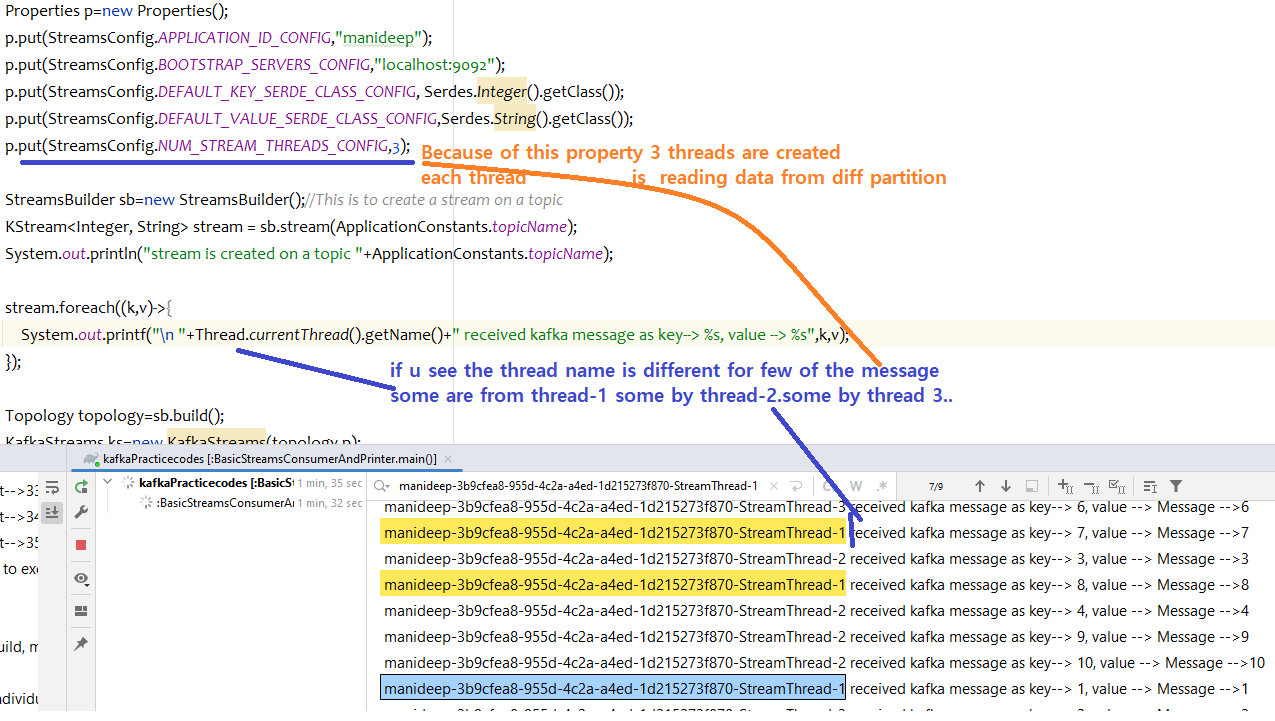
if u have 3 threads then each thread will perform 1 task like each thread will read the data from 1 partition

To configure vertical scaling we have to set a property

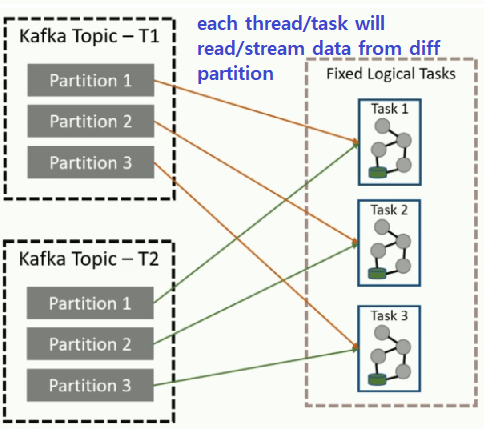
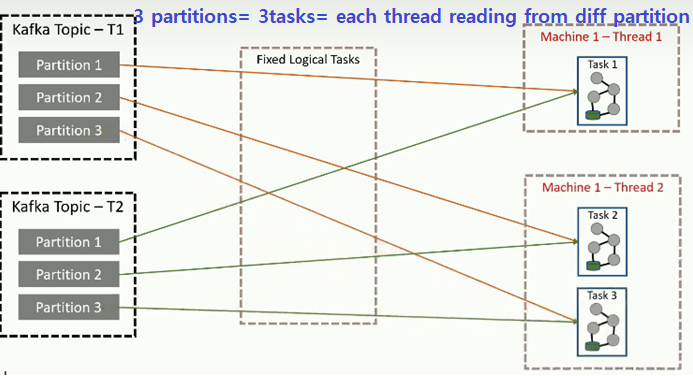
Props.put(Streamsconfig.NUM\_STREAM\_THREADS\_CONFIG,3); // Here my question is how do I know which thread is consuming data from which partition?

How do I confirm 3 threads are reading from 3 diff partition? Simple just keep a logger in stream method to print thread name

I created a program to send data to 3 diff partitions, now because of this multithreaded streams consumer, each thread will read data from separate partition



Once u build the topology, logical task is equal to number to partitions, each logical task Is performed by separate thread, in the above if u have 3 available threads all 3 tasks will be started by 3 separate threads, if 3 threads are not available then if only 2 are available, 1st thread will handle 1 task of reading data from 1 partition and second thread will only handle 2 task to read data from 2nd partition as per below image



Since only 2 threads are available they shared the tasks among themelves

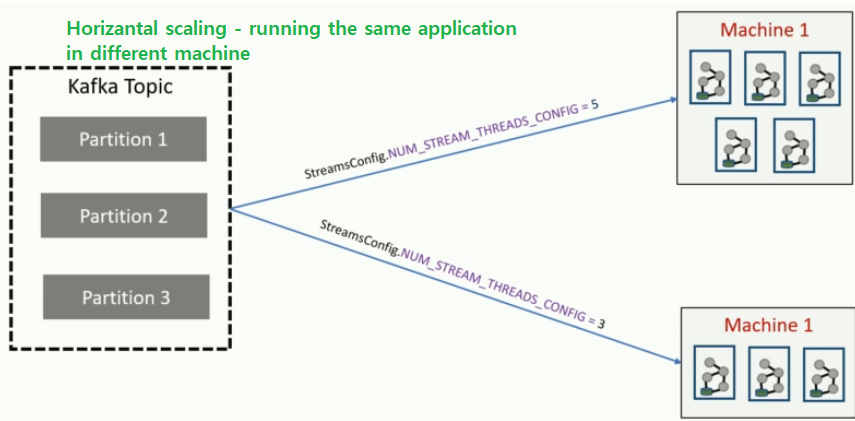
Streams horizontal scaling

It is like launching same app in multiple computers

You can scale application horizontally by starting your streams application in different computers

Same like we deploy our code in multiple servers in production / like deploying your application

In servers like oxmore/shoreview ,its like we create our streams application and generally we deploy our app in prod in 3-4 servers



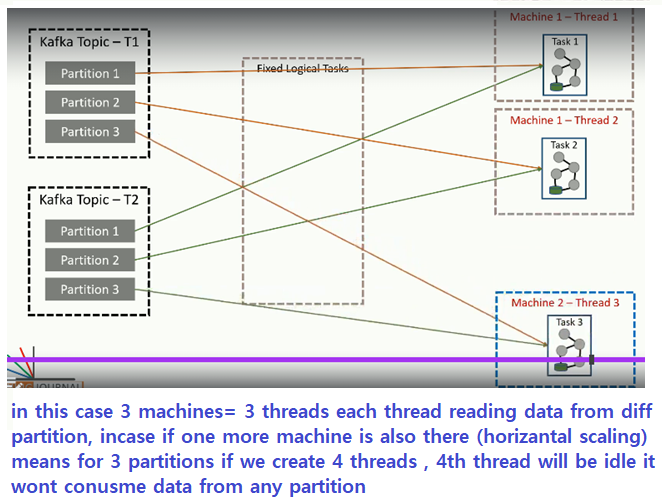
Horizontal and vertical scaling

#### 4 Threads but 3 partitions –Idle thread

If we have 4 threads to read from 3 partitions then each thread will read data from 1 partition , then 4th thread will not read from any partition and it will be idle

Simple first do vertical scaling by configuring 5 threads in your application and later to implement horizontal scaling just deploy the same application in multiple servers lets say we have deployed in 4 servers

Therefore in each server 5 threads and total 4 servers =5\*4 =20 threads are there in all vms

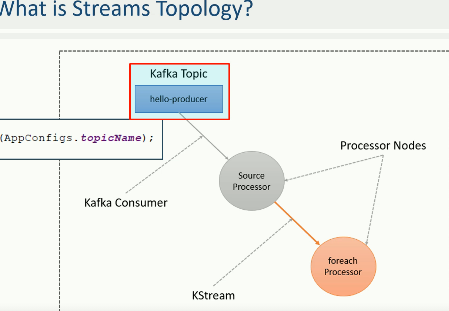


Here we did horizontal scaling and vertical scaling , each machine we have separate threads , example 1st machine have 2 threads and we did horizontal scaling we deployed the same application in second machine also which have 1 thread so they have share the logical tasks among themselves

Ideally there should be 6 threads, each machine should have 3 threads , but in 1st machine as we don’t have enough 3 as we have only 2 it used 2 threads and in 2nd machine also we should have 3 threads but only1 is available so used that

Kafka Terminologies

Source processor vs sink processor

Source processor (it will process from source) will internally uses a kafka consumer to read data from topic

This Source processor (kafka consumer) can consume data from multiple topics

Sink processor will internally uses a kafka producer to write to a topic

These processors can read & write the data to the topic  
For Source processor input is topic and this source processor will internally create a consumer and create a stream on that topic

`

Methods of kstream

Filter(),Map(),flatMap() ,mapValues(),flatMapValues()– these are intermediate operations and these methods will create another streams

forEach(), to() – these methods are terminal operations,

to() method is a sink processor , it will take the stream and write the data to that topic

#### Remaining misc classes

Produced Object- used on KStream.to(String topicName,Produced p) this is a sink processor which will write the data to the topic, this object is used to hold

Consumed object

Sample program

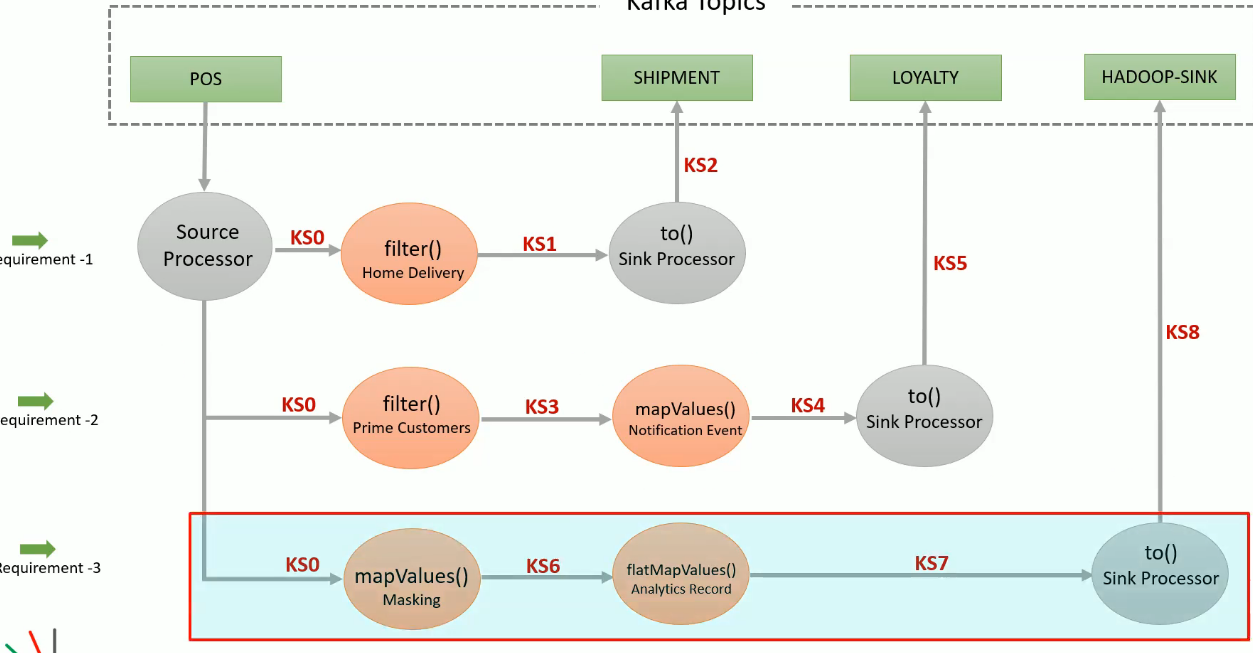
Requirement

1) Select Invoices where DeliveryType = "HOME-DELIVERY" and push them to the shipment service queue.

2. Select Invoices where CustomerType = "PRIME" and create a notification event for the Loyalty Management Service. The format for the new notification event is given here.

3. Select all Invoices, mask the personal information, and create records for Trend Analytics. When the records are ready, persist them to Hadoop storage for batch analytics. The format for the new Hadoop record is also given.

DAG (Direct acyclic graph)



Sample codes

String topicName="tcso";  
  
Properties p=new Properties();  
p.put(StreamsConfig.APPLICATION\_ID\_CONFIG,"manideep");

// Every stream appln should have appln id, If we have multiple instances of same streams app, then all apps should have same appln id

p.put(StreamsConfig.*BOOTSTRAP\_SERVERS\_CONFIG*,"localhost:9092");  
p.put(StreamsConfig.*DEFAULT\_KEY\_SERDE\_CLASS\_CONFIG*, Serdes.*Integer*().getClass());  
p.put(StreamsConfig.*DEFAULT\_VALUE\_SERDE\_CLASS\_CONFIG*,Serdes.*String*().getClass());

p.put(StreamsConfig.*NUM\_STREAM\_THREADS\_CONFIG*,3);

StreamsBuilder sb=new StreamsBuilder();*//This is to create a stream on a topic*KStream<Integer, String> stream = sb.stream(ApplicationConstants.*topicName*);  
System.*out*.println("stream is created on a topic "+ApplicationConstants.*topicName*);  
  
stream.foreach((k,v)->{  
 System.*out*.printf("\n received kafka message as key--> %s, value --> %s",k,v);  
});  
  
Topology topology=sb.build();  
KafkaStreams ks=new KafkaStreams(topology,p);  
ks.start();  
System.*out*.println("stream started");

#### Avro serializer

I couldnot practice as if u want to send data in avro we need schema registry and in my local due to windows machine I cant up the schema registry

Note:- if u want to send the data in avro format we should use avro serializer,

The <groupId>io.confluent</groupId>

<artifactId>kafka-avro-serializer</artifactId>

The Kafka avro serializer jar is not present in maven repo it is present only in confluent repo hence to pull the jars

We should add confluent repository alone

<repositories>

<repository>

<id>confluent</id>

<url>https://packages.confluent.io/maven/</url>

</repository>

</repositories>

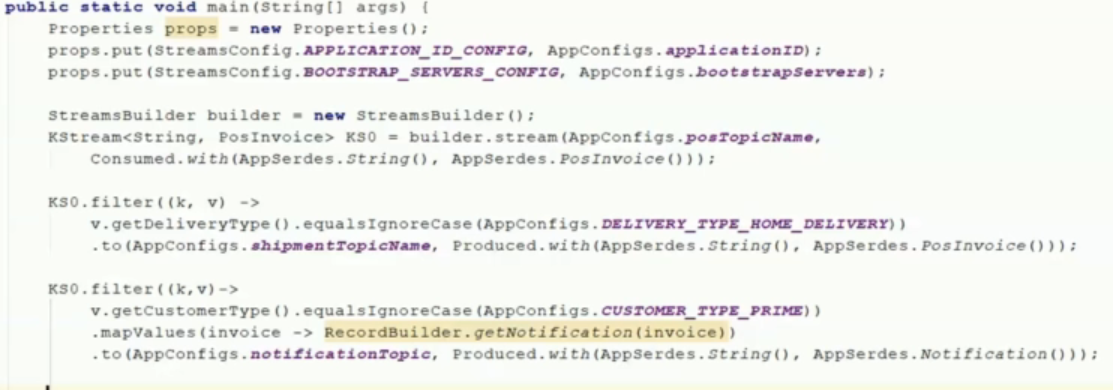
Some errors and solutions

* sometimes streams application won’t consume data

Then delete the logs from Kafka folder and after deleting “tmp” folder ,

Application will consume only the data which was sent after application starts

Ex:2 Consuming from one topic & filtering & writing data to another topic



Here to() is used to send data to topic , produced with is to mention serialization and deseriailzation