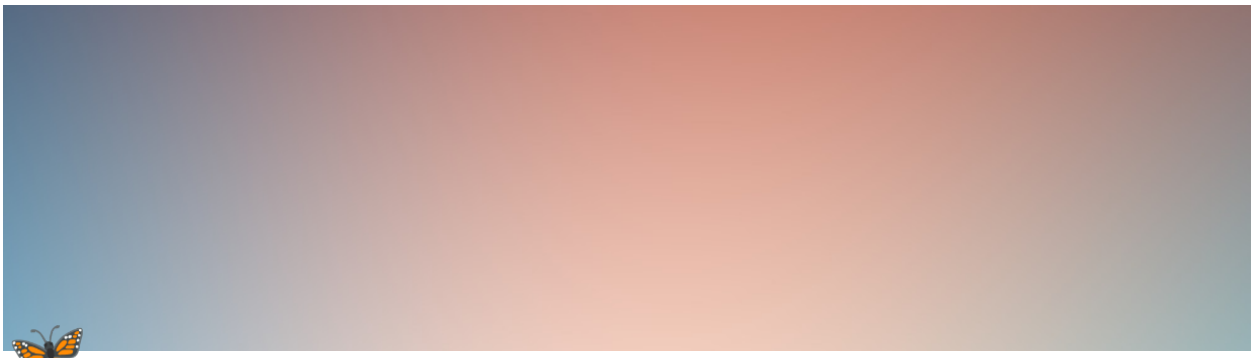


101 - Kafka

Status	Completed
Assign	
Property	

Kafka

Apache Kafka is a Horizontally Scalable , fault tolerant, Distributed streaming platform

Available Kafka installation

> We most likely to be used confluent Kafka in Production

1. Open Source – Apache Kafka
2. Commercial Distribution – confluent.io
3. Managed Service – confluent, amazon, aiven.io

Install Confluent Cloud

Install Confluent Kafka in AWS Machine

Install kafka

How To Install Apache Kafka (Via Confluent) On Ubuntu 20.04 LTS

This recipe shows how to install Apache Kafka on Ubuntu 20.04 LTS. The installation will allow for the publishing and subscribing of Avro messages, ingesting data from a database to a Kafka topic, and exporting messages from

https://www.eddycyu.dev/blog/how-to-install-apache-kafka-on-ubuntu-20.04-lts

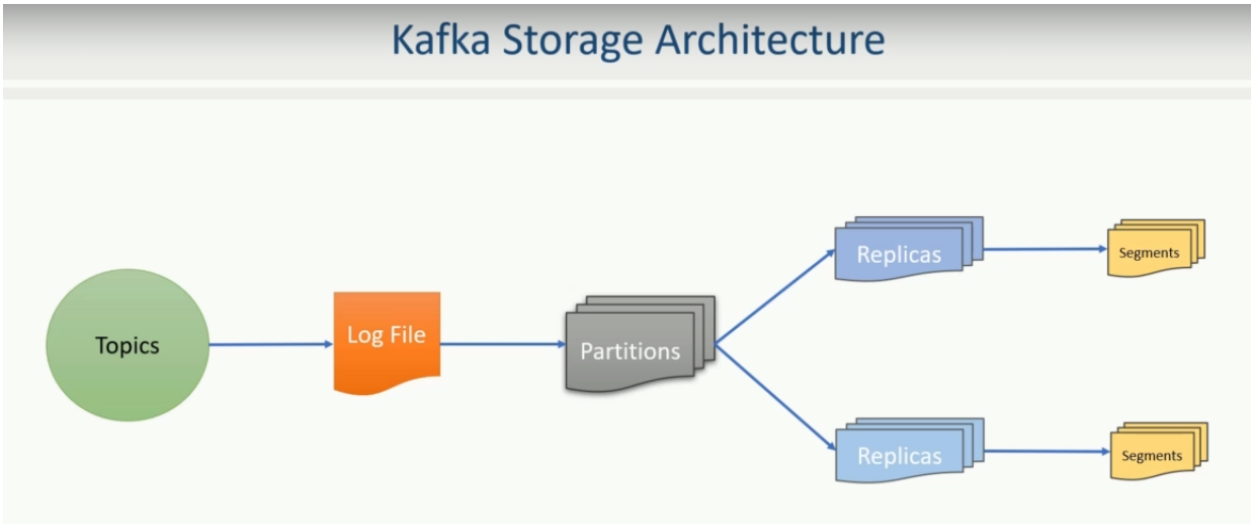
Apache Kafka is a Message Broker

Kafka Broker is a middle man between producers and consumers .

Below are broker responsibilities

1. **Receive messages from the producers and acknowledge the successful receipt**
2. **Store the message in a log file to safeguard it from potential loss.**
3. **Deliver the messages to the consumers when they request it.**

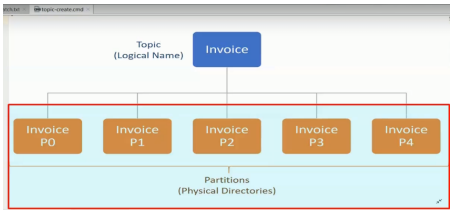
3. Kafka Storage Architecture



01. kafka Topic and partitions

> Topic is a logic name to group your messages . we need topics to store the Messages

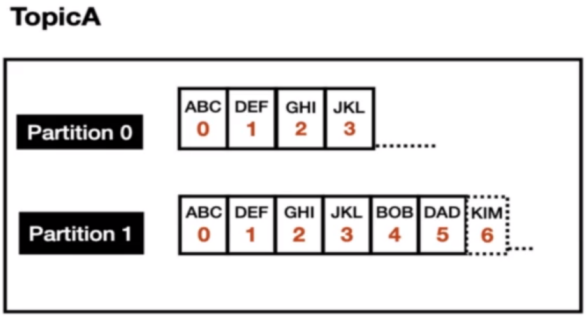
When a Broker Starts it Creates Files For storing some metadata info




```
kafka-topics --create --zookeeper localhost:2181 --topic invoice --partition 5 --rep

if we specify the partition as 5 while creating topics , we get 5 folders in each ka
```

A Single topic may store millions of messages, hence its not practical to store all the messages in a single machine we can split the Files into smaller parts. For _Apache Kafka each partition is nothing but a physical Directory. Kafka will create Separate Folder for each topic partition.

02. Topic Replication



- Each Partition is an ordered , immutable sequence of records
- Each record is assigned a sequential number called ***offset***
- Each partition is independent of each other

```
kafka-topics --list --bootstrap-server localhost:9092
```

Replication Factor

💡 **How many copies we want to maintain for each partition ———> Number of Replicas = partitions * Replication**

```
kafka-topics --create --zookeeper localhost:2181 --topic invoice --partition 5 --replication-

In this Case kafka will generate 15 Directory . These 15 directories are part of the same Top
```

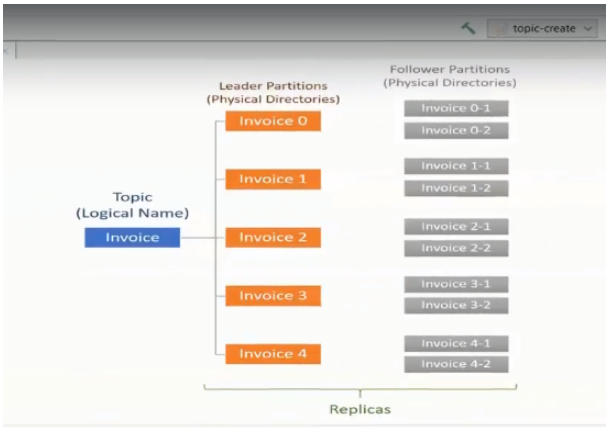
03. Partition Leader

We can classify topic partition replicas into 2 categories

1. Leader partitions
2. Follower partitions

While Creating the topic we specified the number of partitions as 5 and Kafka will create 5 Directories in the broker which act as the leader partitions . Leaders are created First.

Based on the replicas , remaining directories will be created. These are follower partitions . Followers are duplicates of leader



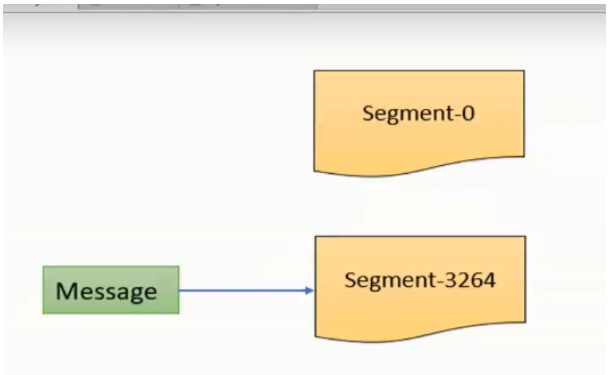
04. Log Segments

Log files are stored in the same partition Directory

Instead of creating a single Big log File , kafka will create multiple Small Log Files These Files are known as segments

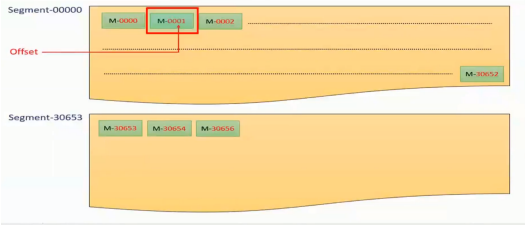
Logic to split the Segment Files

1. Kafka Broker Starts writing logs to the Partition. When the partition receives its first Message , it sores the message in the First Segment.
2. Messages will be written into the First segment until it reaches the maximum segment limit is reached.
3. Once the limit is reached old segment file will be closed by the broker and a new segment file will be created
4. Default limit of the segment File will be ONE GB of data or one week of data
5. We can configure the maximum segment File limit as well



05. Message Offsets

- Each Message in the partition is uniquely identified by the 64 bit integer called offset
- This Offset value will be continues across the segments to keep the offset unique within the partitions .
- Lets say the last message offset is m-30652 and we reach the segment File limit reached so kafka will bring up the new Segment File and Messages starts to Flow inside the New File. The offset for the first message in the new segment continues from the earlier segment and hence it will be m-30653.
- Segment file is also suffixed with the offset value . This value is the first segment offset value.



To identify a message we need below 3 identities

Topic Name

Partition Number

Offset Number

06. Kafka Message Index

kafka allows consumers to Start Fetching messages for Given Offset Number. If the consumer demands for messages beginning at offset hundred , broker must be able to locate the message. To help the broker kafka stores the index of offsets in the .index File. These index files are also segmented for easy management and stored in the same Partition directory along with the log File.

In Some Uses cases we need to fetch the messages based on the Time , These are like we want to consume all the messages which were generated after a particular time stamp value.Kafka maintains timestamps for each message builds a time index to quickly seek the First message that arrived after the given timestamp. These Timestamp index files are also segmented for easy management and stored in the same Partition directory along with the log File

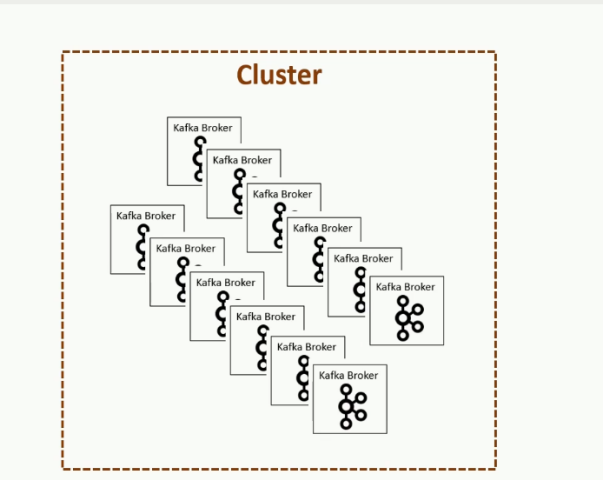
4. Kafka Cluster Architecture

Kafka Cluster is a group of brokers that work together to share the workload . we may have 100 of brokers in our Cluster.

When it comes to Cluster we need to answer Following Questions

1. Cluster Membership
2. Administrative Tasks

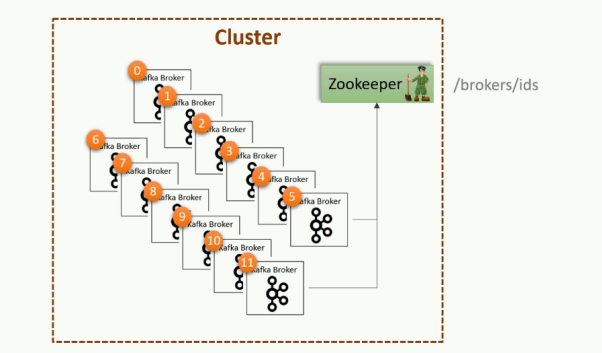
In a typical cluster we have a master node to maintain the list of active cluster members. Master always knows the state of other members.



Kafka Doesn't Follow Master Slave Architecture. Kafka Uses Zookeeper to maintain the list of active brokers

Every Broker has a unique broker id as we specify during the installation. We also specify the zookeeper connection details in the broker configuration file When the active broker starts it will create a Connection with the Zookeeper and creates a ephemeral node using the broker_id to represent the active broker session. This Ephemeral Node will be intact as long as the broker is active .

When the broker dies , zookeeper will remove the ephemeral node . So list of active brokers in the cluster is maintained as the list of ephemeral node under the /brokers/ids path in the zookeeper.



Perform the routine Admin Tasks like

- monitoring the list of active brokers in the zookeeper
- Reassigning the work when an active broker leaves the cluster

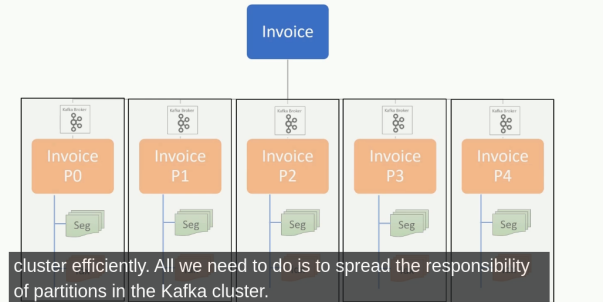
Above activities are maintained by a controller. One of the kafka broker is elected as the Controller to pick some extra responsibilities . If we have a single node cluster it will act as a broker as well as the controller.

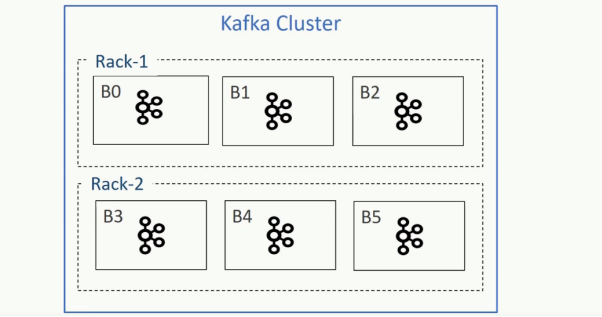
> Only one controller in the cluster at any point in Time

When the Controller notices that a broker leaves the cluster it knows that it is time to reassign the work to the other brokers.

First Broker node in the kafka cluster will be elected as controller . it will create an ephemeral node entry in the zookeeper /controller . Other broker will also try to create or become the controller but we will get an error message "NODE ALREADY EXISTS" . When the Controller dies the respective ephemeral node in the zookeeper disappears. Now every other broker will try to register them self as controller in zookeeper, but only one succeed . rest will get the error message Once again. This Process make sure that we will have only one controller at all time in our cluster.

kafka as Fault Tolerant





Lets a Kafka cluster of 6 brokers and they are placed in 2 racks each having 3 nodes .Lets create a Topic with 10 Partition and 3 replication factor , so we will have 30 replicas to be allocated with 6 Brokers.

While allocating the Partitions kafka make sure we achieved 2 goals

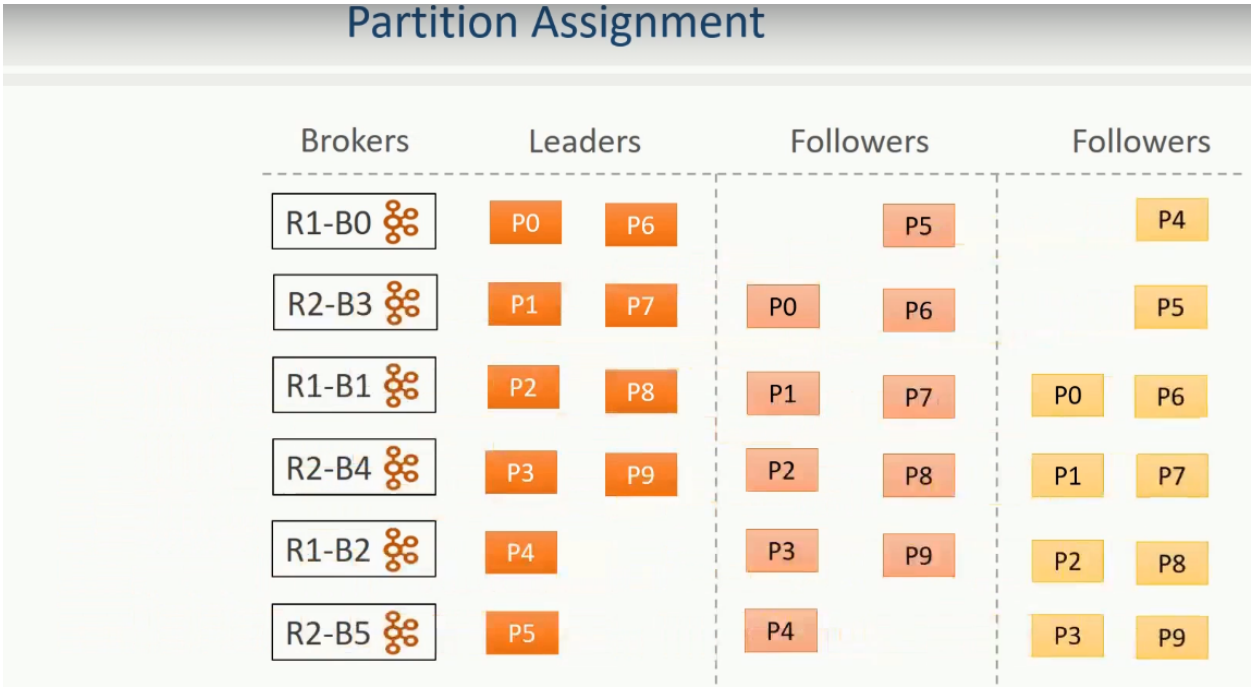
1) Event Distribution ⇒ Partitions are distributed evenly as much as possible to achieve work load balance

2) Fault tolerance =⇒ Duplicate copies must be placed in different machines to achieve fault tolerance

Partition Assignment

Kafka Creates the

1. Ordered list of Brokers
2. Leader and Followers Assignment



Even a Rack is Down we have a one broker holding the message. Even 2 brokers are down we have one more broker to serve the message

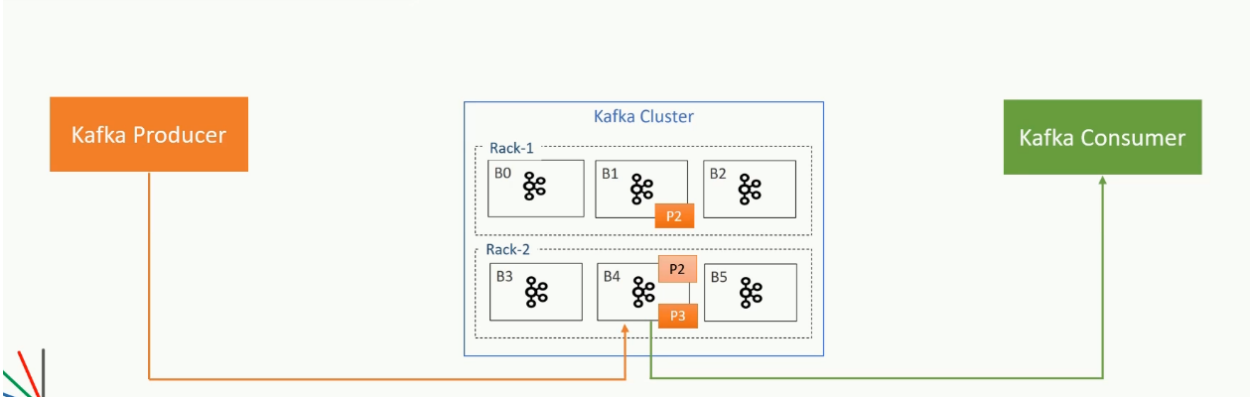
Broker Manages two Types of partitions

1. Leader partitions
2. Follower Partitions

In Kafka Broker , leader is responsible for all the request from the producers and consumers .

When Producers wants to send the Message to kafka Topics . it connects to the Cluster and query for the metadata . All brokers can answer a metadata request so producer can connect to any broker to get the metadata response. Metadata contains list of all the leader partitions and their respective hosts and port information. Now the Producer has list of all leaders . Now Producer that decides on which partitions does it want to sent the data and accordingly send the message to the Broker. Producer sends the message directly to the leader. On receiving the message leader writes the message in leader local partitions and send acknowledge signal back to the producer.

When Consumer wants to consume messages it always read it from the Leader of the partition.



Followers job is only copy the messages from leader and stay up todate with the messages.

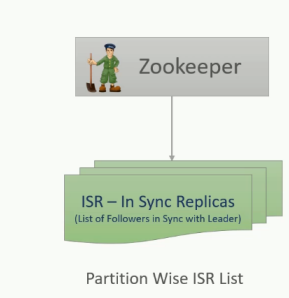
Follower Partition is responsible for copy the messages from the leader to stay update with all the messages. followers will stay sync with the leader .Follower will ask for messages to the leader, this will keep on going in the infinite loop. When the leader dies it may became the leader also it will have the latest message

ISR LIST

Some times followers are failed to in sync with Leader

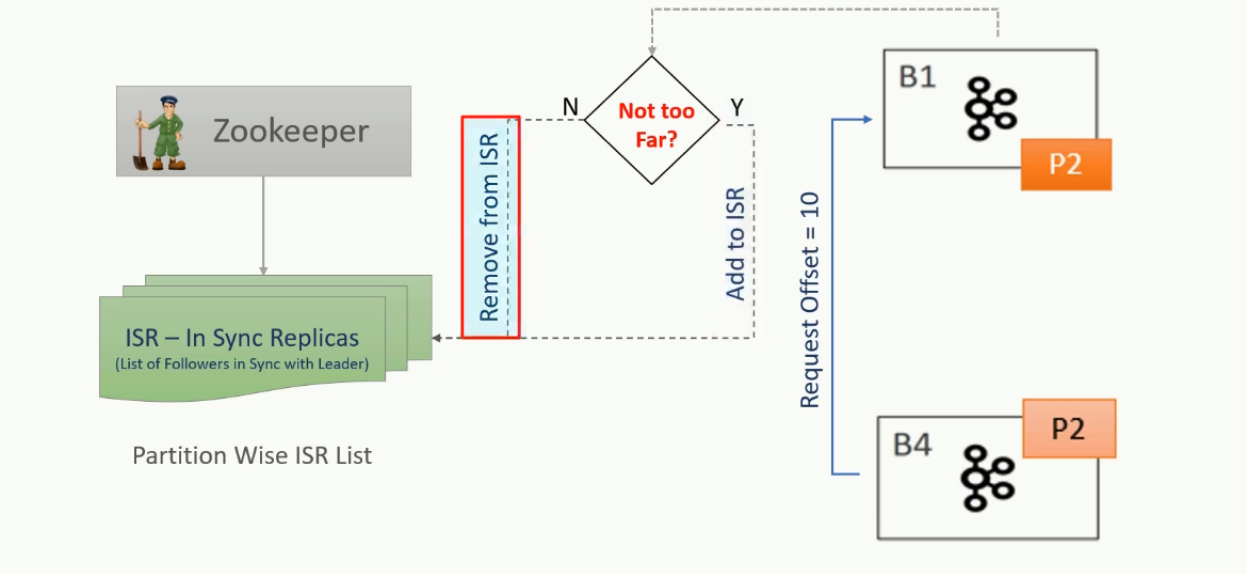
1. Network Congestion ⇒ Due to slow network follower can fall behind
2. Follower Broker Crash/Restart ⇒ since the broker dies all the replicas are fall behind

Leader has one more responsibilities to maintain the ISR (IN Sync Replicas) List . It will be persisted in the zookeeper. All the Followers in the ISR list are in sync with master , they are the excellent candidate to be elected as leader.



Follower Will connect to the leader and ask to send message from the offset zero master send all the messages . Follower will store them and again ask for messages from offset 10 . So leader will safely assume follower persist the data. Based on the offset leader will tell how far follower node is behind the leader . If the Replica is too far , follower will be removed from the ISR list. if not it will be available in the ISR List .

NOT TOO FAR value is 10 seconds by default



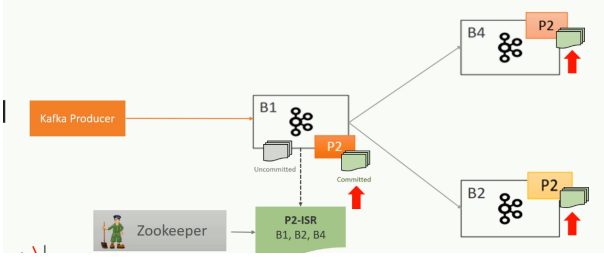
Committed Vs Un-Committed Message

We can configure leader not to consider a message committed until the message is copied to all the followers in the ISR list . In this case leader will have committed and Uncommitted messages .

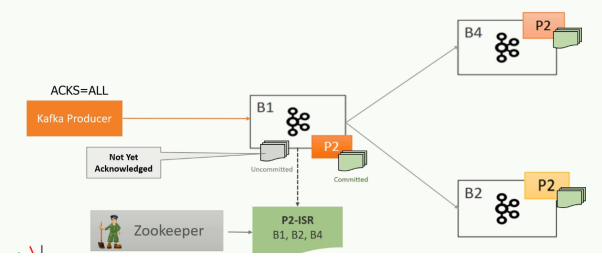
If we loose the Leader Now

1. Committed message will be gathered from the replicas
2. Un-Committed message will be lost , we will ask producer to resend the Message, since Producer doesn't receive the ACK=ALL Acknowledge from the Leader

For Committed message



For uncommitted Message



Minimum In-sync Replica

If we want to make sure the data is written to at least two replicas we need to include a property called

```
min.insync.replica = 2
```

Producer will throw an error "Not Enough Replica" when we don't have 2 followers in our ISR List

Consumer Offset

- Consumer have three options to read
  - from-beginning
  - latest
  - specific offset

For Any Consumer we need to Provide the Group ID

Lets Say Consumer-1 is consuming the Message from the Broker and it reads message upto offset3. Consumer-1 went offline for any reason and meanwhile producer sends 3 more records. Now how does the consumer knows where to read the Offset

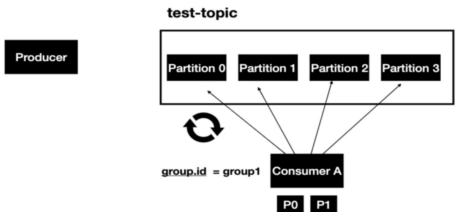
Consumer polls the Message from the Broker and once the records are read , consumer commits he offset value to the internal topic called __**consumer_offsets topic with the groupId.** Now the consumer-1 refers the __consumer_offset topic to understand the offset details and start reading the messages.

Consumer Groups

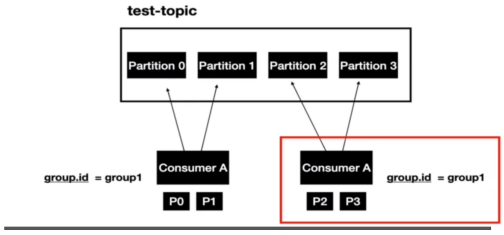
```
kafka-consumer-groups --bootstrap-server localhost:9092 --list
```

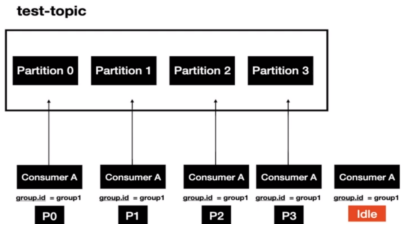
Kafka Producer are producing messages in the faster rate than the consumers processing rate and it introduced lags in the process

- Consumer Groups are used for scalable message consumption
- Each different application will have a unique consumer group
- Who manages the consumer group?
  - Kafka Broker manages the consumer-groups
  - Kafka Broker acts as a Group Co-ordinator

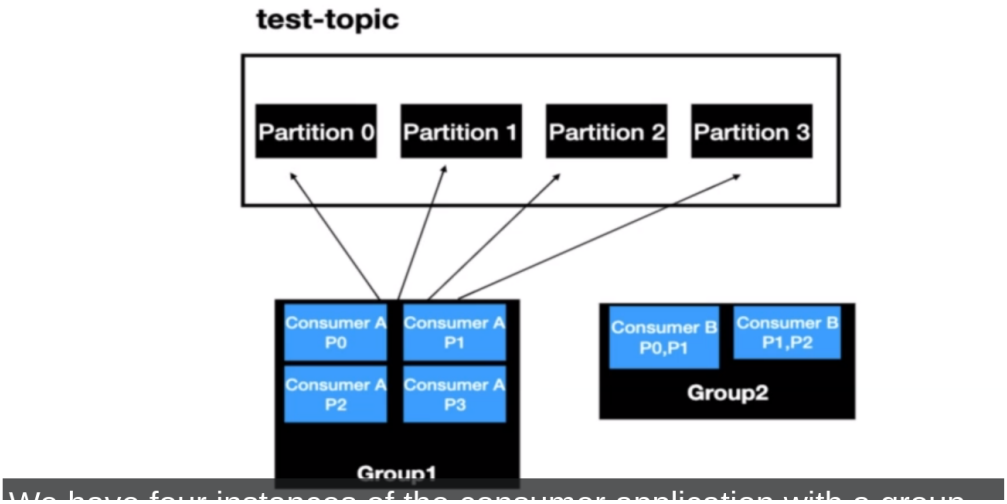


Now we add one more Consumer **we are using the same consumer groupID.** Partitions are split between these 2 instance of these consumers.





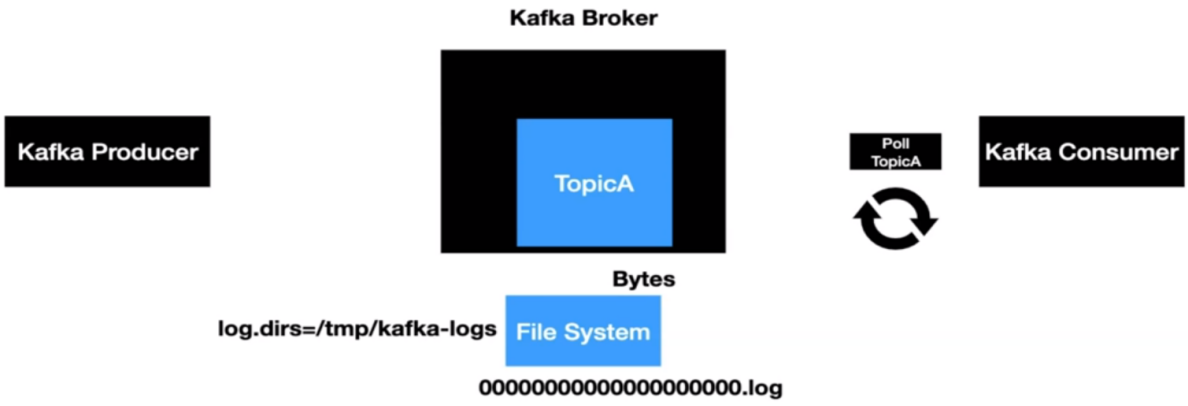
Another use case is SAme topic can be used by different teams using unique groupID



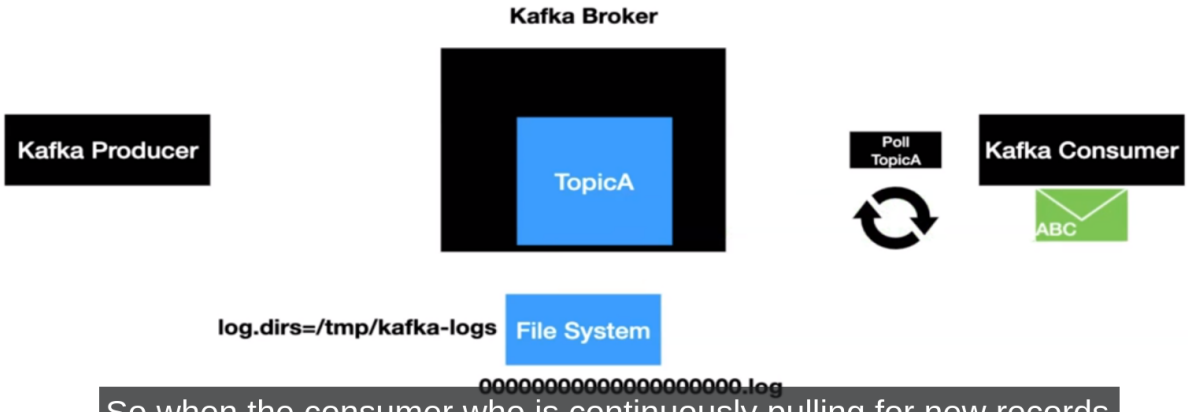
Commit logs

When Producer sends the Message it first reaches the topic.

Record then written to the File system in the Machine . This is the place where brokers are installed and records are written as bytes . The location in the File system where the file is stored can be configured via log,dirs property . REcord is stored in a .log File . Each Partition has its own .log File. It is also called as partition commit logs.



After this Process , Consumer continuously polling for the record can see only records which were committed to the File System



Retention Policy

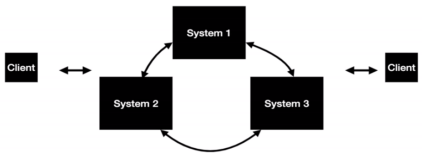
How long the message is going to be in the kafka brokers

This can be determined by the log.retention.hours property in the server.properties File

Retention Period is 7 days

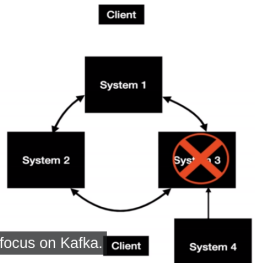
Distributed Streaming System

- Distributed systems are a collection of systems working together to deliver a value



Characteristics of Distributed System

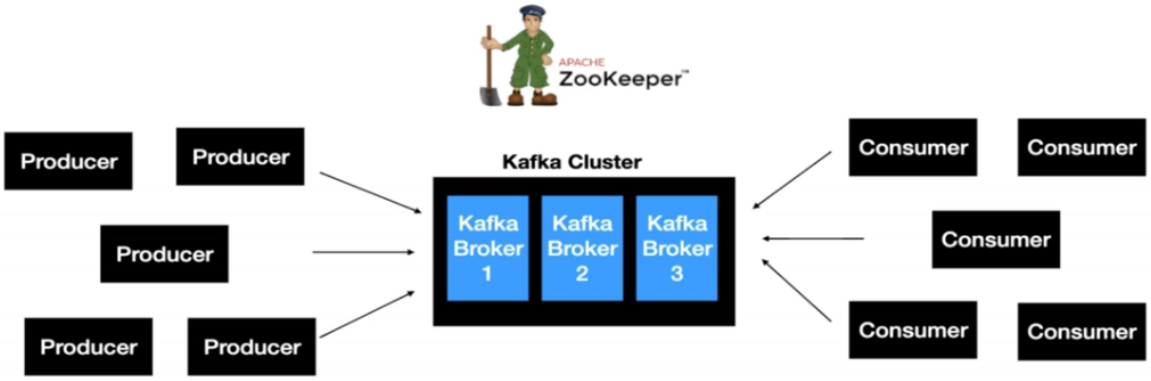
- Availability and Fault Tolerance
- Reliable Work Distribution
- Easily Scalable
- Handling Concurrency is fairly easy



distributed systems and focus on Kafka.

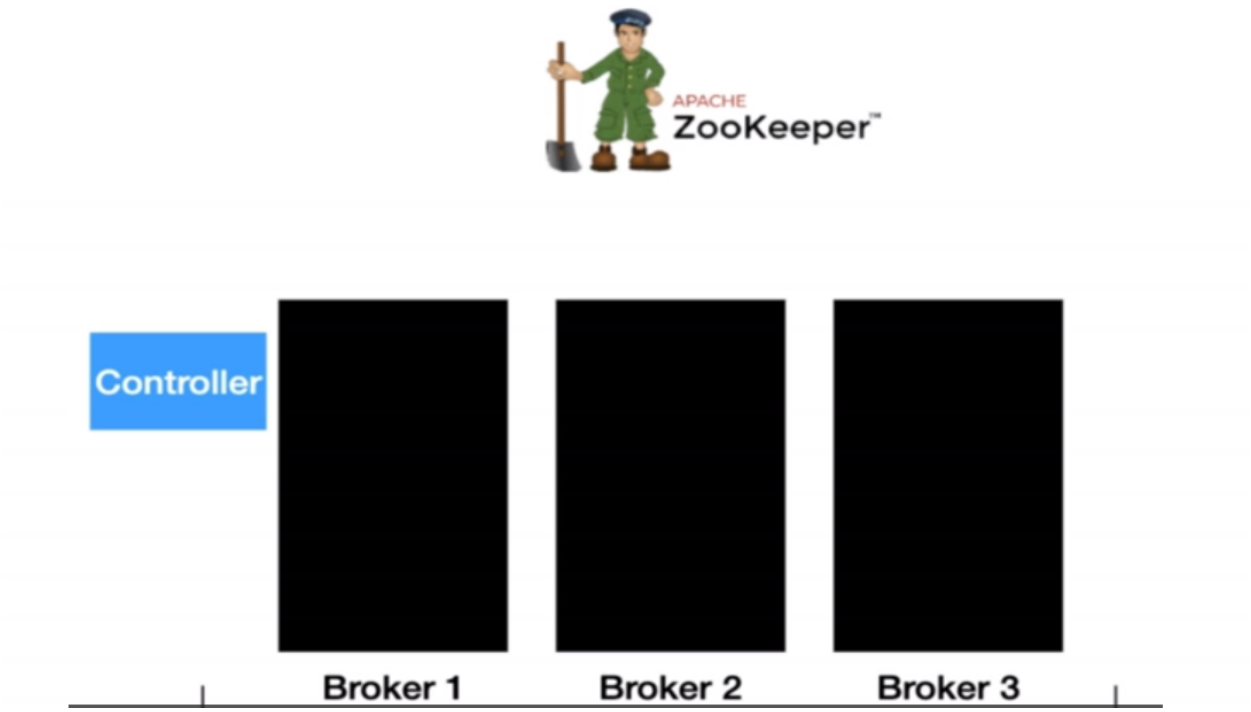
To avoid Single point of Failure we have kafka clusters which has multiple broker nodes

Kafka as a Distributed System

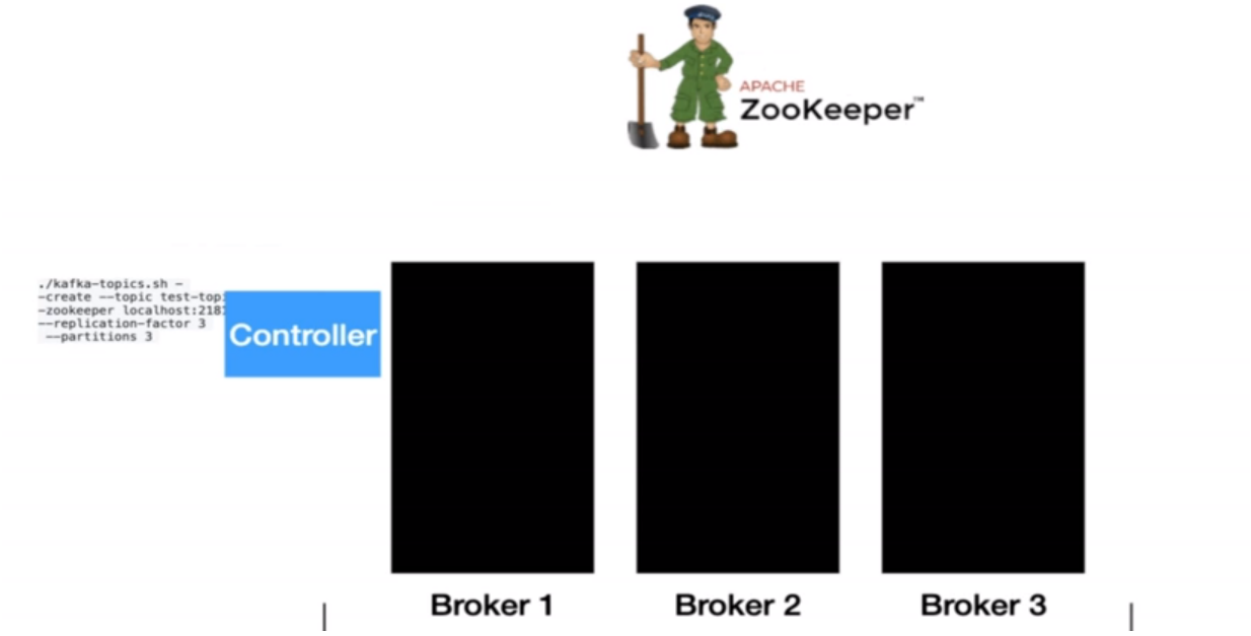


- Client requests are distributed between brokers

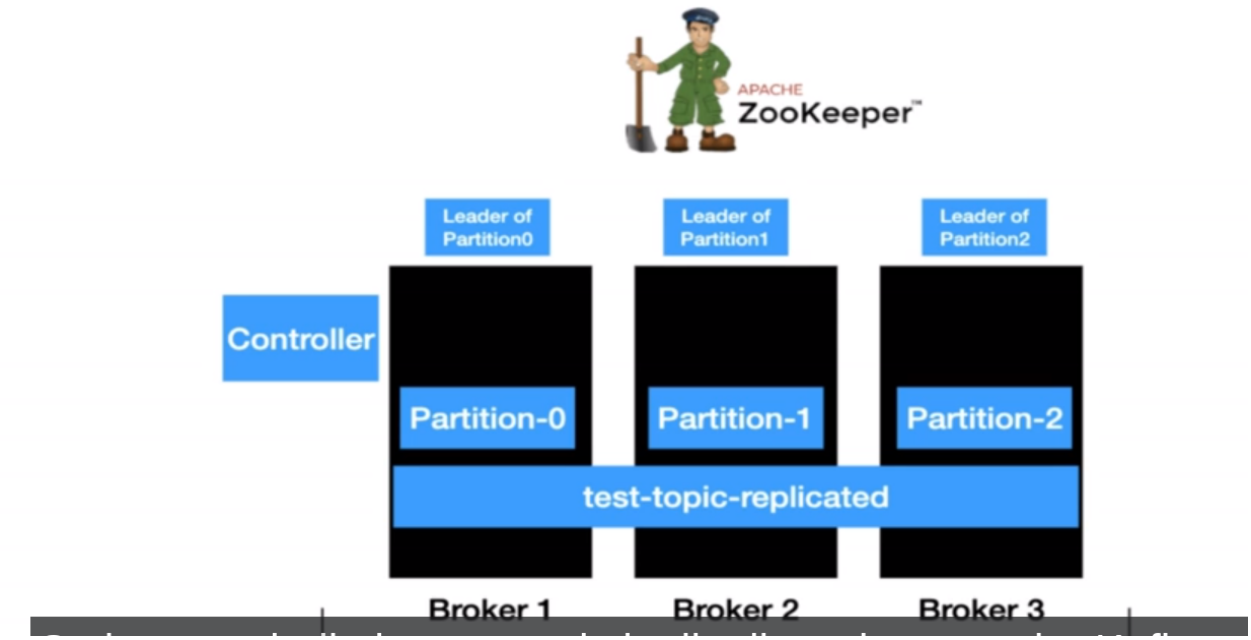
First Broker to join the Cluster will act as a Controller



When we issue the command to create the topic zookeeper will route the command to the Controller



Controller will distribute the partitions across the cluster

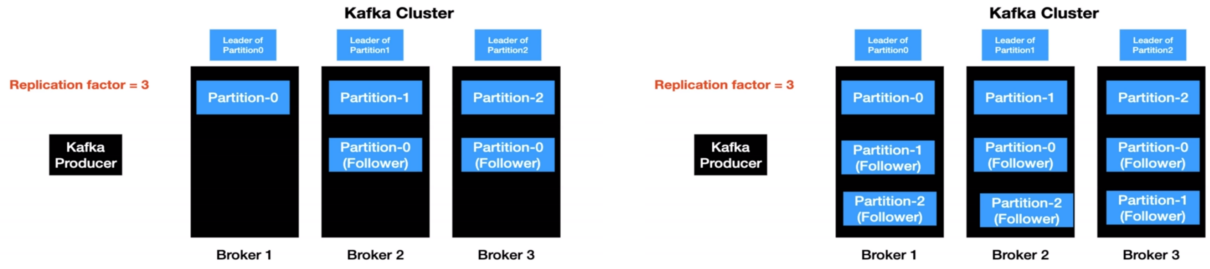


Data Loss

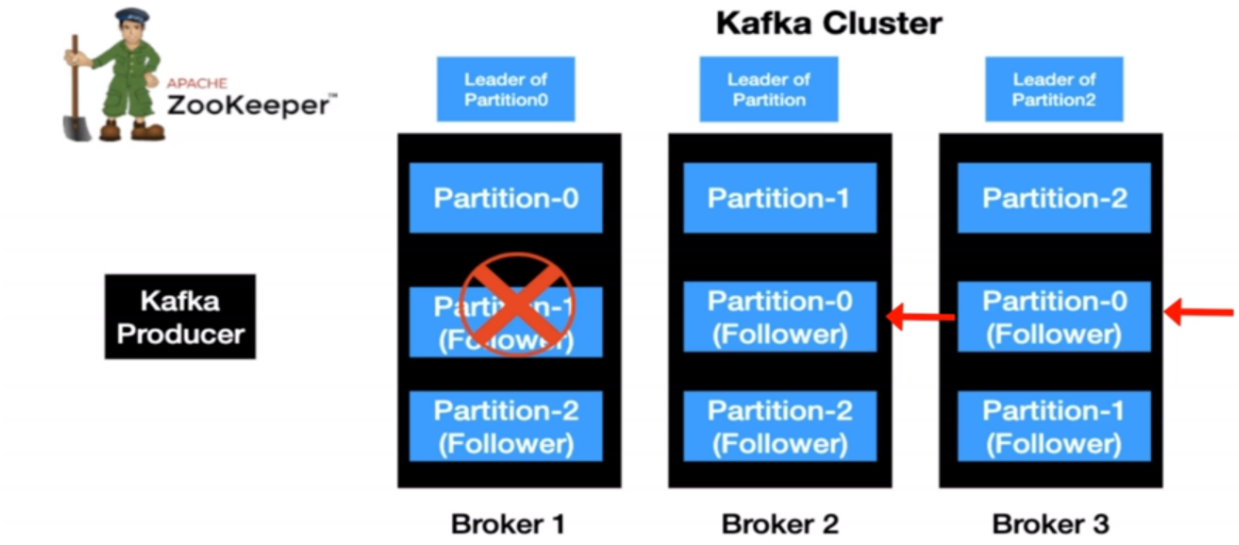
Kafka handles data loss using the replication.

```
--replication-factor 3  ---> 3 partions will be created and stored in different brokers
```

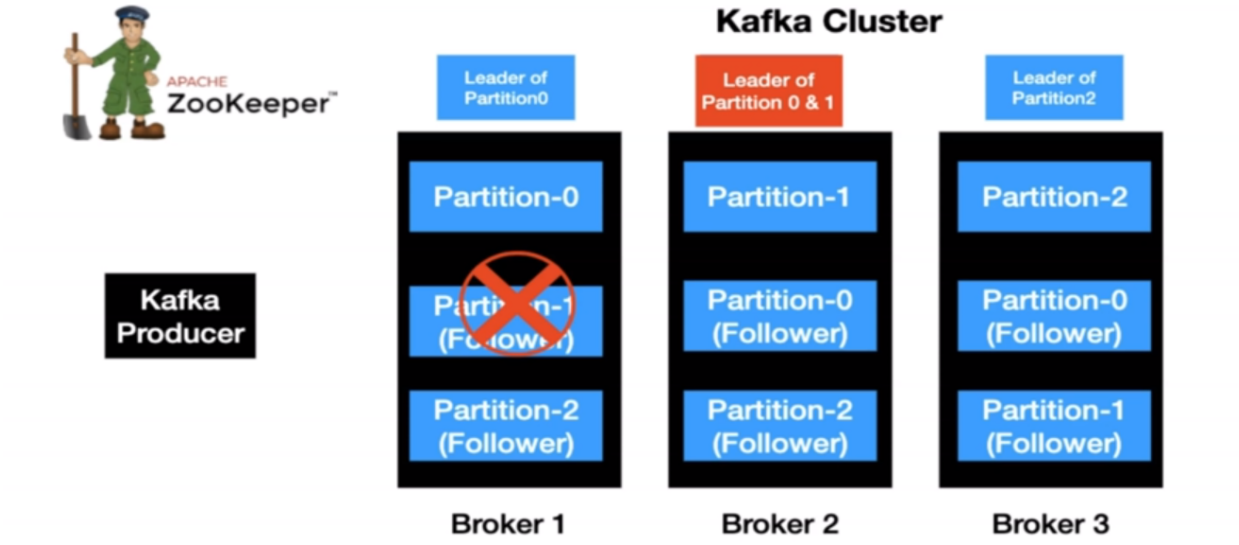
For replication 3 , Partitions are assigned as below . Each partition will have a leader as well as the follower like below



Now when a Broker is Failed , we can still access the data from other brokers



Zookeeper will be informed about the failure and it assigns a new leader to the controller . Now broker2 will be the leader for partition and partition



In-Sync Replica(ISR)

- Represents the number of replica in sync with each other in the cluster
  - Includes both **leader** and **follower** replica
- Recommended value is always greater than 1
- Ideal value is **ISR == Replication Factor**

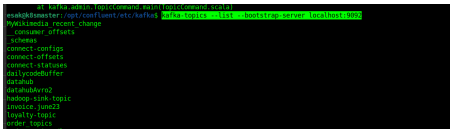
Configure ISR

```
in.insync.replicas = 2

if the topic replication is less than the min.insync.replica then we get error
```

Kafka Operations Using CLI

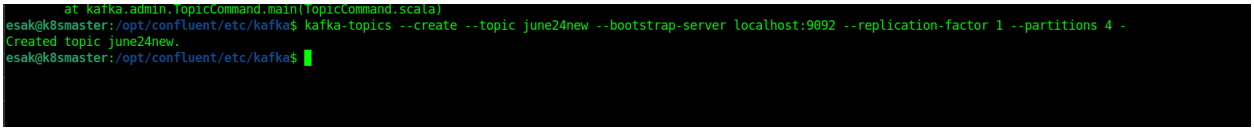
1. List all Topics

```
kafka-topics --list --bootstrap-server localhost:9092
```



2. Create a Topic

```
kafka-topics --create --topic june24new --bootstrap-server localhost:9092--replication-factpr 1 --partitions 4 --bootstrap-server localhost
```



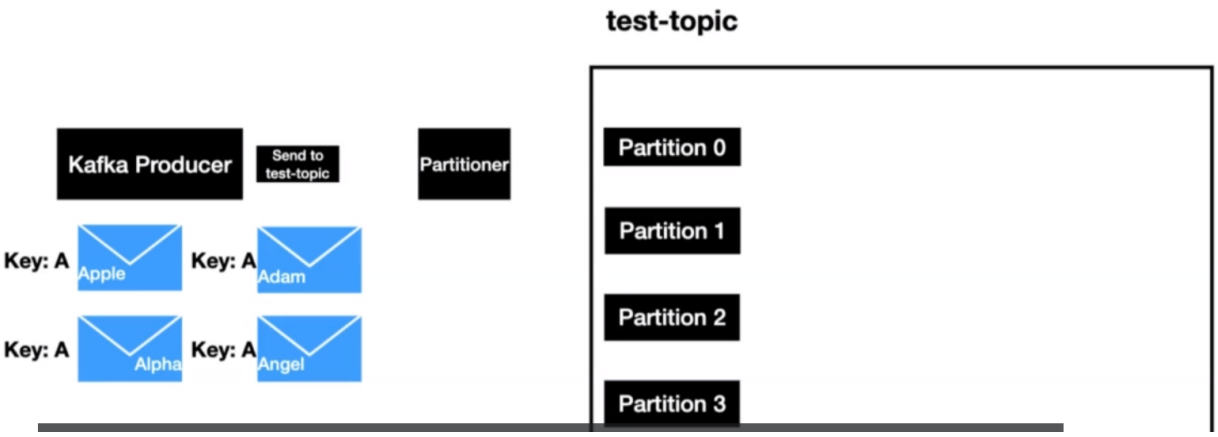
3. Produce Message via console

```
kafka-console-producer --broker-list localhost:9092 --topic june24new
```

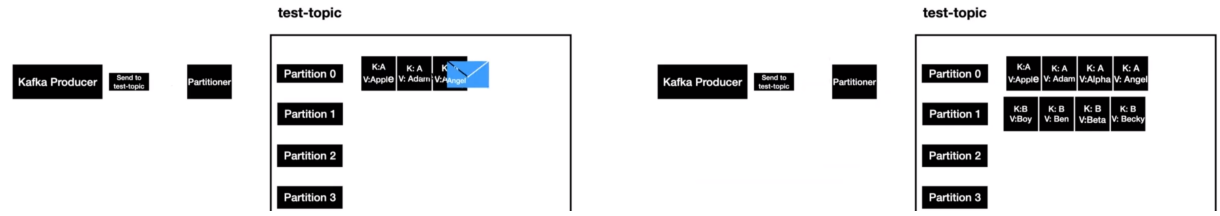
```
esak@k8smaster: /opt/confluent/etc/kafka                esak@k8smaster: /opt/confluent/etc/kafka
esak@k8smaster:/opt/confluent/etc/kafka$ kafka-console-producer --broker-list localhost:9092 --topic june24new
>esakki
>is
>a
>hero
>
```

While Producing the messages kafka will go through lot of layers one such layer is called the partition Layer and if the message has a key it goes to partition and If the message doesnt have any keys a round robin method will be used and messages will be transferred to the partition

Consumers poll the messages from all partitions at the same time which may leads to disruptions in the order of the message.

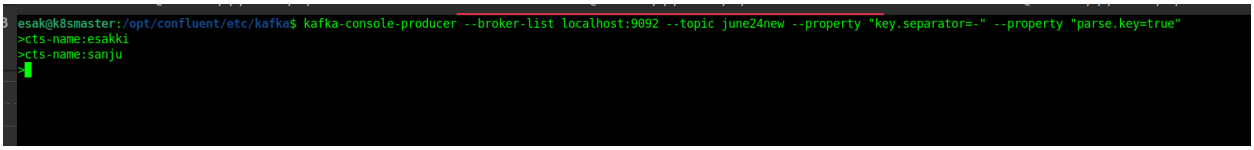


Keys are applied to a hashing algorithm and assigned to a partition .



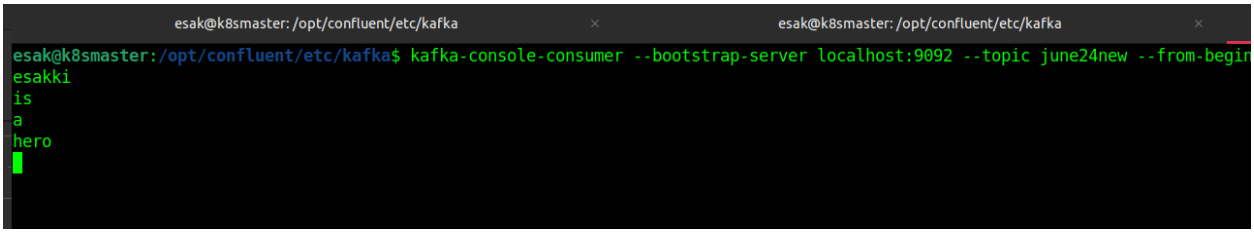
💡 SAME KEY ALWAYS RESOLVES TO SAME PARTITIONS

```
kafka-console-producer --broker-list localhost:9092 --topic june24new --property "key.separator=-" --property "parse.key=true"

Read More About Property from the Kafka DOCS
--property "key.separator=-"
--property "parse.key=true"
```



4.Consume the Message

```
kafka-console-consumer --bootstrap-server localhost:9092 --topic june24new --from-beginning
```



Consumer with the Key

```
kafka-console-consumer --bootstrap-server localhost:9092 --topic june24new --from-beginning
--property "key.seperator=-" --property "print.key=true"
```

Consumer messages with Group Id

```
kafka-console-consumer --bootstrap-server localhost:9092 --topic june24new --from-beginning -
-property "key.seperator=-" --property "print.key=true" --group esakkidevgroup
```

5. Kafka Producer

Create a simple Kafka Producer code to send 1 million String messages to a Kafka Topic

Lets Start the Kafka in Local machine

Create a new Topic

```
esak@esak-PC:~/kafka/bin$ kafka-topics.sh --create  --topic datahub --partitions 3 --bootstrap-server localhost:9092
Created topic datahub.
```

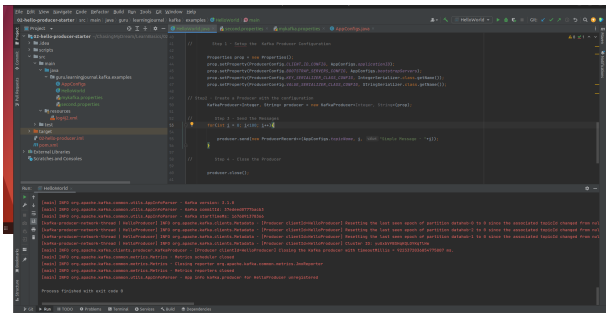
Create a New Producer

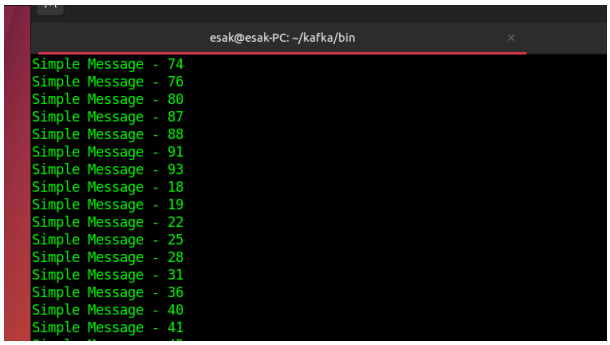
```
package guru.learningjournal.kafka.examples;

import org.apache.kafka.clients.producer.KafkaProducer;
import org.apache.kafka.clients.producer.Producer;
```

```
import org.apache.kafka.clients.producer.ProducerConfig;
import org.apache.kafka.clients.producer.ProducerRecord;
import org.apache.kafka.common.serialization.IntegerSerializer;
import org.apache.kafka.common.serialization.StringDeserializer;
import org.apache.kafka.common.serialization.StringSerializer;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.Properties;
import java.util.concurrent.ExecutionException;

public class HelloWorld {
    private static final Logger log = LoggerFactory.getLogger(HelloWorld.class);



    public static Properties loadConfig(final String configFile) throws IOException {
        if (!Files.exists(Paths.get(configFile))) {
            throw new IOException(configFile + " not found.");
        }
        final Properties cfg = new Properties();
        try (InputStream inputStream = new FileInputStream(configFile)) {
            cfg.load(inputStream);
        }
        return cfg;
    }



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

//        Step 1 - Setup the  Kafka Producer Configuration

        Properties prop = new Properties();
        prop.setProperty(ProducerConfig.CLIENT_ID_CONFIG, AppConfigs.applicationID);
        prop.setProperty(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, AppConfigs.bootstrapServers);
        prop.setProperty(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG, IntegerSerializer.class.getName());
        prop.setProperty(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG, StringSerializer.class.getName());

// Step2 - Create a Producer with the configuration
        KafkaProducer<Integer, String> producer = new KafkaProducer<Integer, String>(prop);

//         Step 3 - Send the Messages
        for(int i = 0; i<100; i++){

            producer.send(new ProducerRecord<>(AppConfigs.topicName, i, "Simple Message - "+i));
        }

//         Step 4 - Close the Producer

        producer.close();




    }
}
```

Run the Application

Checking the Console Consumer





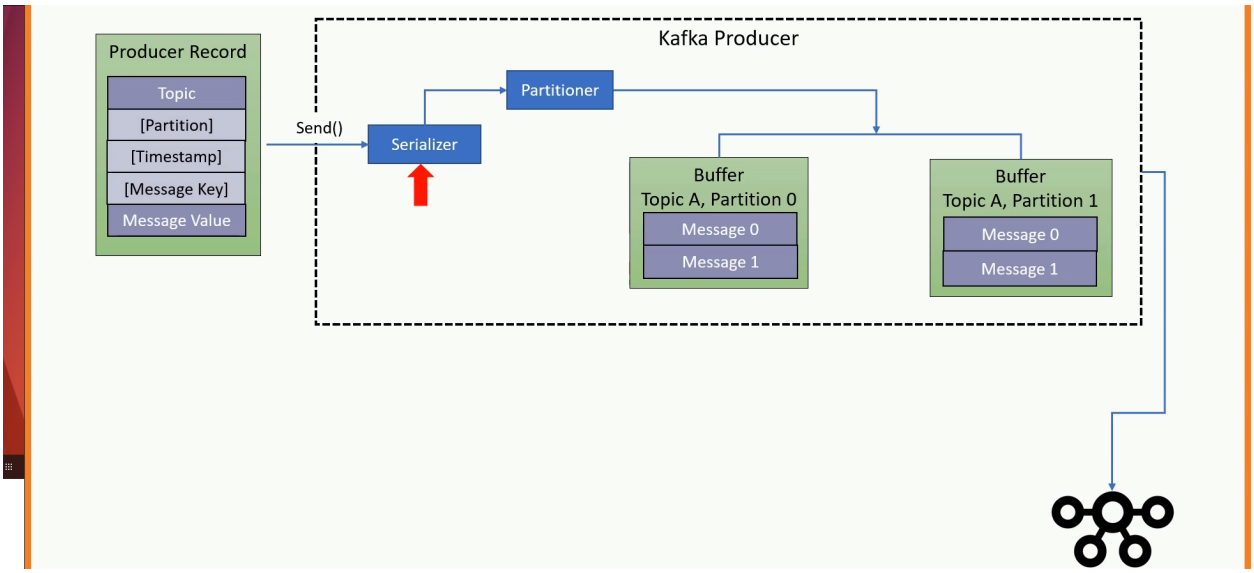
Producer Records

We must pack the message in the Producer Record Object . In that we have topic and Message to be sent is available.

Producer Serializer

Every Record Goes through Serialization, partitioning and then it is kept in the buffer.

We need to serialise the data before sending to the network. We have to Explicitly tell how to serialize the data using the Key and Value Serializer class.



Producer Partitioner

Every Producer record has a topic Name as the destination address of the data. Kafka Topics are partitioned and hence producer should decide on which partition the message should be sent.

1. we can optionally specify the Number in Partition
2. we can configure a partition class , it will determine at the runtime

```
props.put(ProducerConfig.PARTITIONER_CLASS_CONFIG, MyPartitioner.class.getName())
```

Kafka Producer comes with the default Partitioner which is the most commonly used partitioner. DEfault partitioner takes 2 approaches to determine the destimation of topic partition

- Hash Key Partitioning
  - Based on the message Key .It uses hashing to convert the value into a numeric value and it decides the partition
  - Hashing Ensures all the message with the same key goes to the same partition
  - If the partition is based on the Keys , then we should create topic with enough Partition and never increase it in the later stage.

- Round Robin partitioning
  - When the message key is null , default partitioner uses the round robin partitioning method to achieve an equal distributions among the partitions.

Message Timestamp

Message Timestamp is Optional . For real time streaming Application timestamp is most critical value.

Even if don't Specify also . all kafka messages are timestamped . Kafka uses below Time stamping Concepts

1. Create Time
   a. Create Time is the time when the message was produced
2. Log Append Time
   a. Log Append Time is the time when the message was received at the broker side.
   b. Broker will override the Timestamp with the Current Timestamp before appending it to the log

```
message.timestamp.type=0  ---> Create Time
message.timestamp.type=1  ---> Append Time
```

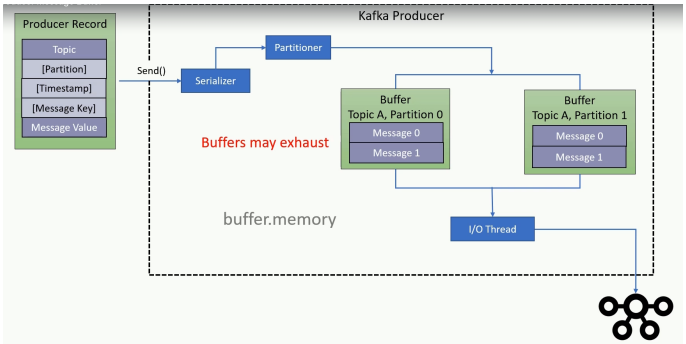
Message Buffer

Once the message is serialized and assigned a target partition number, message goes to sit in the buffer waiting to be transmitted.

Producer also runs the I/O Thread that is responsible for turning these records into a requests and transferring to the Cluster.

Buffering arrangements make the send method asynchronous. end method will add the message to the buffer and return with out blocking . These records are transferred to the broker by some background thread.

If we send the Message More Faster and Buffer is Filled or I/O thread is taking lot of time to push records to the broker then send method will be blocked and producer will end up in Timeout Exception. Default Producer memory is 32MB.

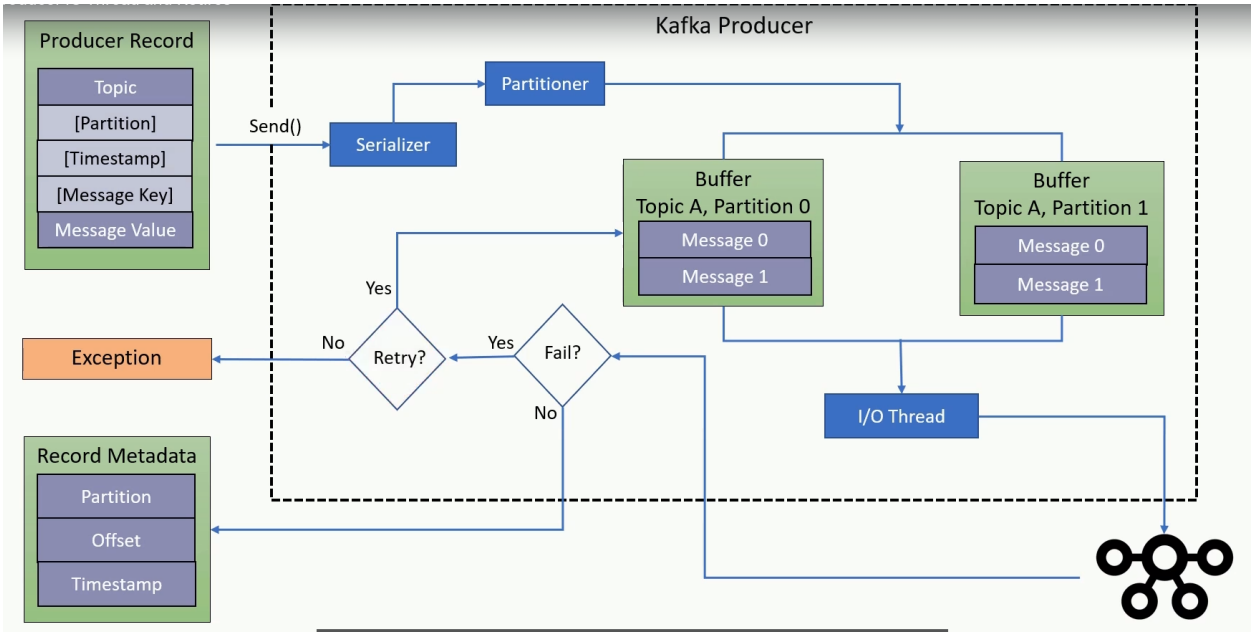


```
we can increase the Producer memory
buffer.memory=2G
```

Producer IO Threads

IO Thread will transfer the serialized message from the topic partition buffer to the broker . When the broker receives the message, it send back the acknowledgement.

If the Background IO Thread doesn't receives an error or any ACK it may retry sending the message few more times before throwing back error



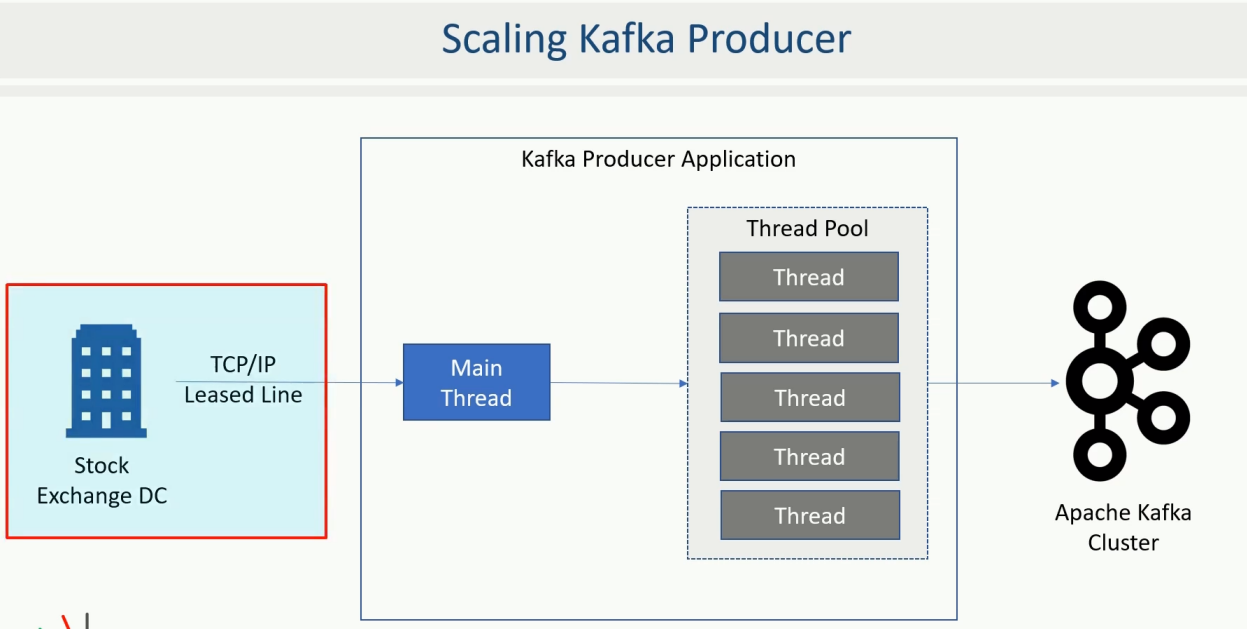
ADVANCED Producers

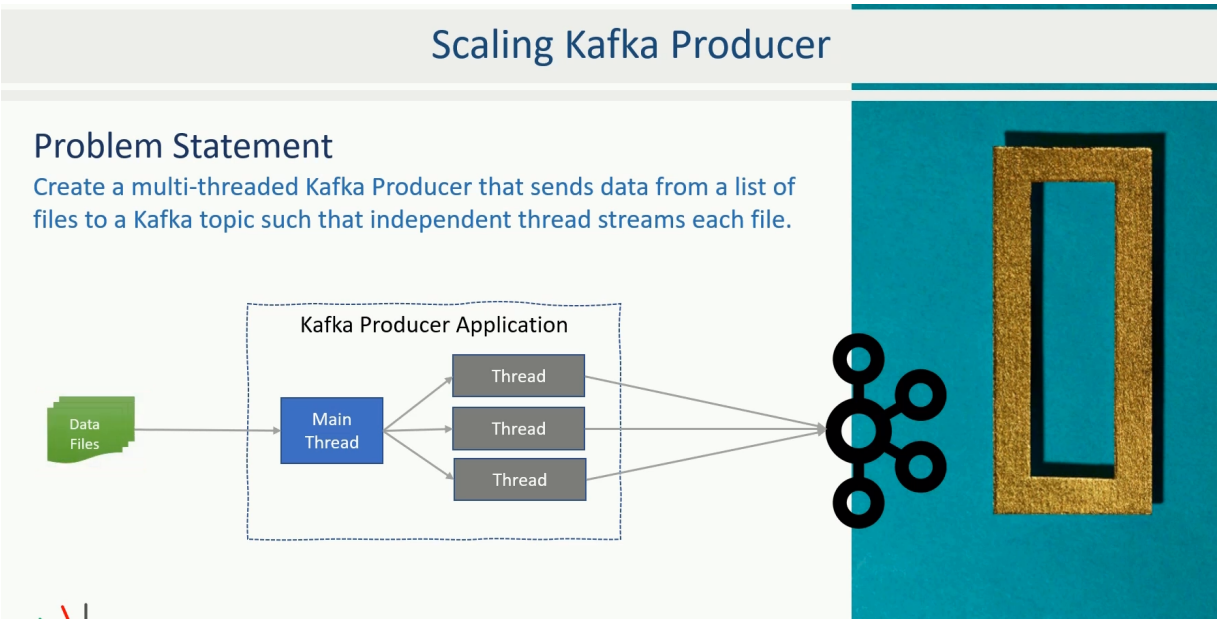
Multi Thread Producer

Main Thread listens to the sockets and get the messages as they arrive and immediately hand over to another Thread.And it starts reading the message again.

Other Thread are responsible for uncompress the packer and process it and send it to kafka broker

Kafka Producer is Thread-Safe. So your Application can share the same producer object across multiple threads and send messages in parallel.





Lets Create a Dispatcher Class

```
package guru.learningjournal.kafka.examples;

import org.apache.kafka.clients.producer.KafkaProducer;
import org.apache.kafka.clients.producer.ProducerRecord;

import java.io.File;
import java.util.Scanner;

public class Dispatcher implements Runnable {

    private String fileLocation ;
    private String topicName;

    private KafkaProducer<Integer, String> producer;
```

```
    Dispatcher(KafkaProducer<Integer, String> producer, String topicName, String fileLocation) {
        this.producer = producer;
        this.topicName = topicName;
        this.fileLocation = fileLocation;
    }

    @Override
    public void run() {

        File file = new File(fileLocation);
        int counter = 0;
        try {
            Scanner scanner = new Scanner(file);
            while(scanner.hasNext()){
                String line = scanner.nextLine();
                producer.send(new ProducerRecord<>(topicName, null, line));
                counter++;
            }
            System.out.println("Finished Sending Ttoal Record COunt of "+counter);
        } catch (Exception e ){
            System.out.println("Error occured");
        }


    }
}
```

Lets Create a Main Class

```
package guru.learningjournal.kafka.examples;

import org.apache.kafka.clients.producer.KafkaProducer;
import org.apache.kafka.clients.producer.Producer;
import org.apache.kafka.clients.producer.ProducerConfig;
import org.apache.kafka.common.serialization.IntegerSerializer;
import org.apache.kafka.common.serialization.StringSerializer;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStream;
import java.util.Properties;

public class DispatcherDemo {

    public static void main(String[] args) {

        Properties props = new Properties();

        try {
            InputStream inputstream = new FileInputStream("src/main/java/guru/learningjournal/kafka/examples/third_kafka.properties");
            props.load(inputstream);
            props.put(ProducerConfig.CLIENT_ID_CONFIG, AppConfigs.applicationID);
            props.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG, IntegerSerializer.class.getName());
            props.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG, StringSerializer.class.getName());

        }catch(FileNotFoundException fe){
            System.out.println("File Not Found");
        } catch (IOException e) {
            throw new RuntimeException(e);
        }


        KafkaProducer<Integer, String > producer = new KafkaProducer<Integer, String>(props);
        Thread[] dispatchers = new Thread[AppConfigs.eventFiles.length];

        for(int i=0; i< AppConfigs.eventFiles.length; i++){
            dispatchers[i] = new Thread( new Dispatcher(producer, AppConfigs.topicName, AppConfigs.eventFiles[i]));
            dispatchers[i].start();
        }

        try{

            for(Thread t: dispatchers ) t.join();
        }catch (Exception e ){

        } finally {
```

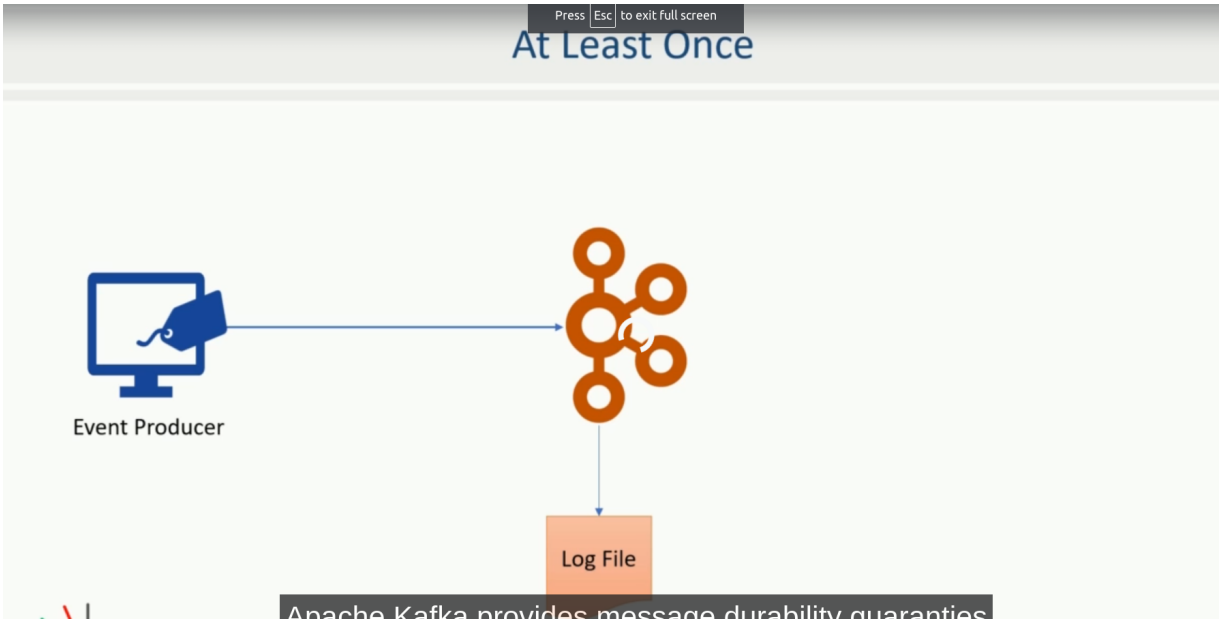
```
            producer.close();
        }



    }
}
```

At least Once

Apache Kafka Provides message durability guaranties by committing the messages at partition logs. Which means once the leader persist the message in the leader partition we cant loose the message till the leader is alive.

To Loss the data because of leader failure we have the replication. Kafka implements replication using followers.

When the data is persisted to the leader as well as the followers in the ISR List we consider the message is Fully committed.



Lets Assume that the IO sends the message to the Broker and broker stores it in the partition and sends the ack back to the IO. But the IO didnot receive the ACK assuming that the process was Failed and resend the Message to Broker Again Which results in Duplicates.

This is called atleast-once semantics.

We can achieve at-most-once semantics by setting the broker settings as

```
retries to zero
```

Exactly Once Semantics

we need to set the below property and kafka will take care of implementing exactly once.

```
enable.idempotence = true
```

Transactional Producer

It will be like ALL Or Nothing , if any Errors occured all the transaction will be rolled Back to original State

For implementing it

```
replication factor >= 3
min.insync.replicas >= 2

ProducerConfig.TRANSACTIONAL_ID_CONFIG Should be added to the Properties
```

WE can implement Transaction Producer in 3 Step Process

1. Initialization
2. BEGIN Transactions
3. Commit Or Abort transaction

7. Types And Serialization

In real Time we will be working with Complex Objects.

Step 1 - Create Java Types

1. we want to Create message Schema using schema definition Language.
2. We want IDE to build java Class from the schema definition Language
    a. we can Use Json Schemas to POJOS
    b. We can Use AVRO schema to POJOS

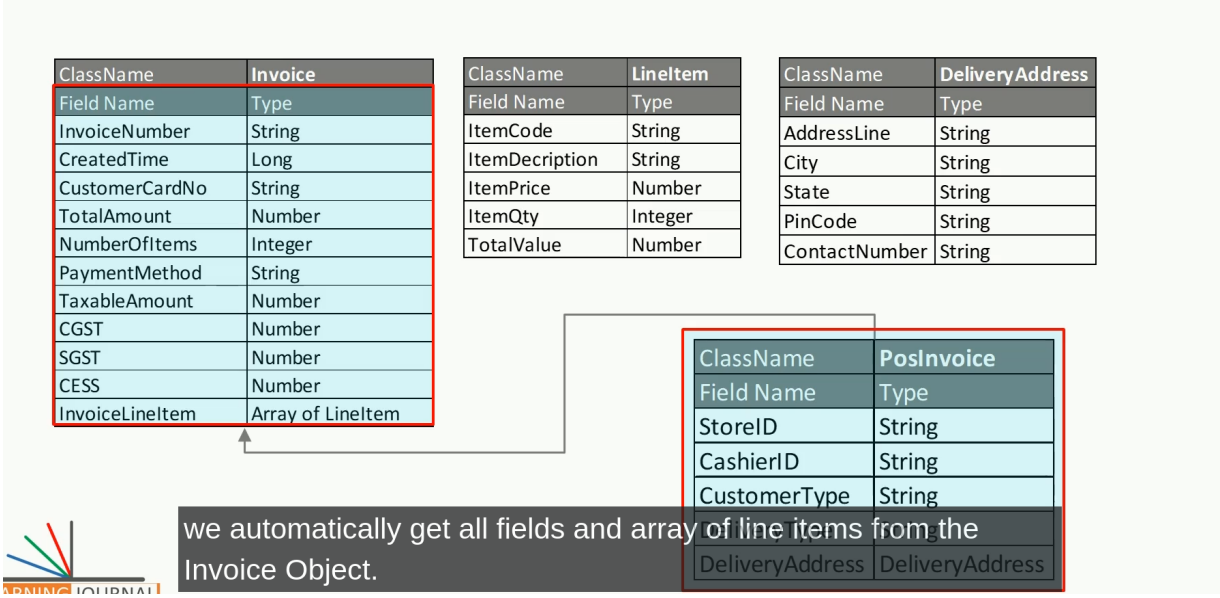
Step 2 - Serialise Java Types

we can use

1. Json Serialization
2. Avro Serialization

JSON SERIALIZATION

Lets Say we want to model an invoice event line below



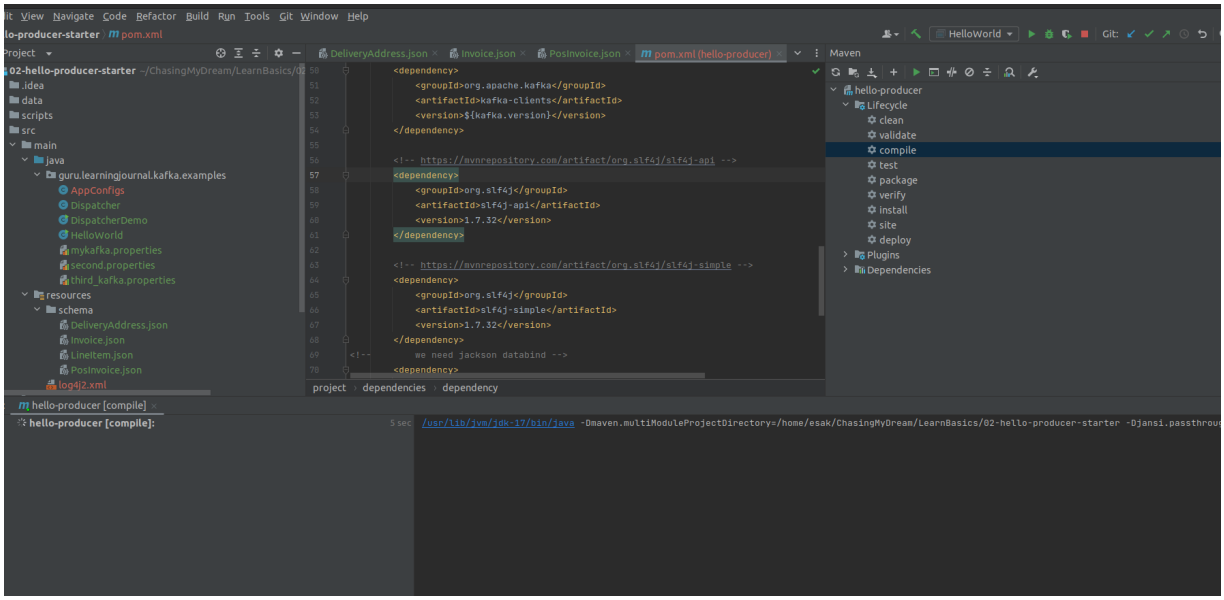
Lets Create Our First Schema

Lets Add these Maven Plugin Which convert the Json Schema into Jaa Code Automatically

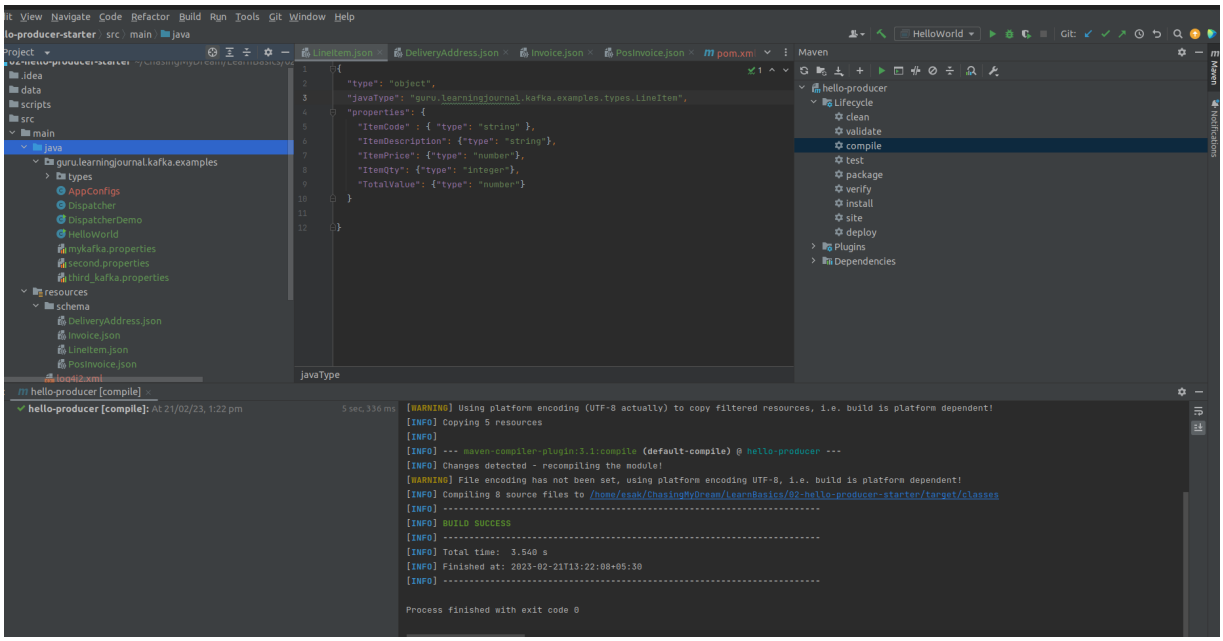
```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>guru.learningjournal.kafka.examples</groupId>
    <artifactId>hello-producer</artifactId>
    <version>2.3.0</version>

    <properties>

        <kafka.version>3.1.0</kafka.version>
        <maven.compiler.source>17</maven.compiler.source>
        <maven.compiler.target>17</maven.compiler.target>
    </properties>
<build>
    <plugins>

        <plugin>
            <groupId>org.jsonschema2pojo</groupId>
            <artifactId>jsonschema2pojo-maven-plugin</artifactId>
            <version>1.2.1</version>

            <executions>
                <execution>
                    <goals>
                        <goal>generate</goal>
                    </goals>
                    <configuration>
                        <sourceDirectory>${project.basedir}/src/main/resources/schema</sourceDirectory>
                        <outputDirectory>${project.basedir}/src/main/java/</outputDirectory>
                        <includeAdditionalProperties>false</includeAdditionalProperties>
                        <includeHashcodeAndEquals>false</includeHashcodeAndEquals>
                        <generateBuilders>true</generateBuilders>

                    </configuration>
                </execution>
            </executions>
        </plugin>

    </plugins>
```

```
</build>



    <dependencies>
        <!-- Apache Kafka Clients-->
        <dependency>
            <groupId>org.apache.kafka</groupId>
            <artifactId>kafka-clients</artifactId>
            <version>${kafka.version}</version>
        </dependency>

        <!-- https://mvnrepository.com/artifact/org.slf4j/slf4j-api -->
        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-api</artifactId>
            <version>1.7.32</version>
        </dependency>

        <!-- https://mvnrepository.com/artifact/org.slf4j/slf4j-simple -->
        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-simple</artifactId>
            <version>1.7.32</version>
        </dependency>

    </dependencies>

</project>
```

Now Go to Maven Life Cycle and execute the compile option

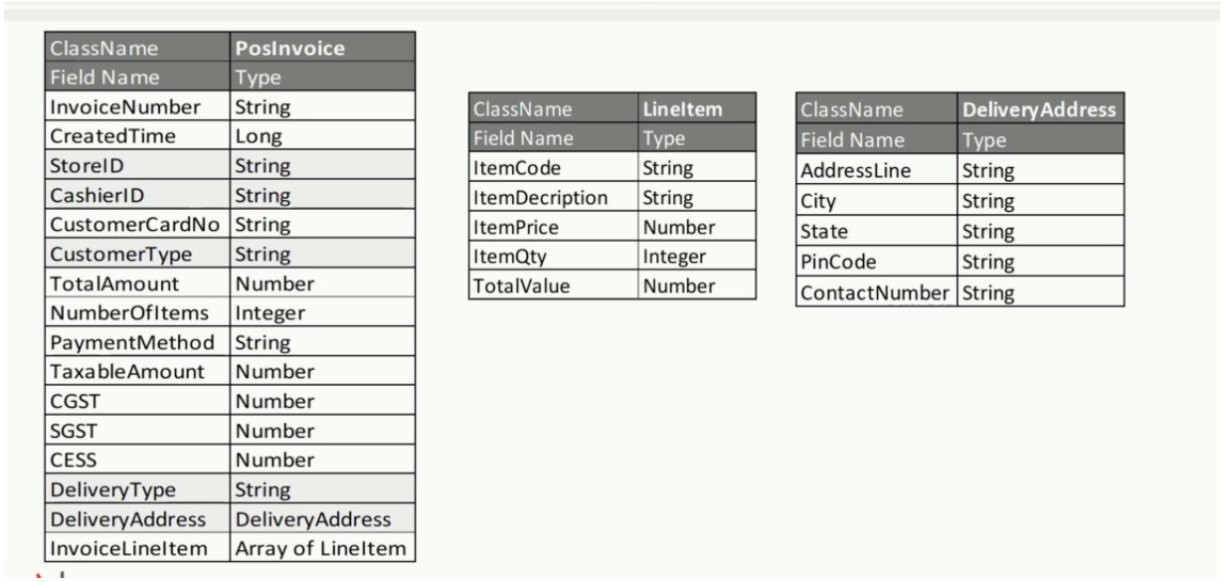


Once the Build is sucessful , we get the Java Classes



Lets Learn About Avro Serialization

Lets Create the same set of schema , But Avro doestnt Support Inheritance so we combine the POS with the INvoice Schema



ClassName	PosInvoice
Field Name	Type
InvoiceNumber	String
CreatedTime	Long
StoreID	String
CashierID	String
CustomerCardNo	String
CustomerType	String
TotalAmount	Number
NumberOfItems	Integer
PaymentMethod	String
TaxableAmount	Number
CGST	Number
SGST	Number
CESS	Number
DeliveryType	String
DeliveryAddress	DeliveryAddress
InvoiceLineItem	Array of LineItem

ClassName	LineItem
Field Name	Type
ItemCode	String
ItemDecription	String
ItemPrice	Number
ItemQty	Integer
TotalValue	Number

ClassName	DeliveryAddress
Field Name	Type
AddressLine	String
City	String
State	String
PinCode	String
ContactNumber	String

Lets Create the LineItem

```
{
  "namespace": "guru.learningjournal.kafka.avroexamples.types",
  "type": "record",
  "name": "LineItem",
  "fields": [
    {"name": "ItemCode","type": ["null","string"]},
    {"name": "ItemDescription","type": ["null","string"]},
    {"name": "ItemPrice","type": ["null","double"]},
    {"name": "ItemQty","type": ["null","int"]},
```

```
    {"name": "TotalValue","type": ["null","double"]}
  ]
}
```

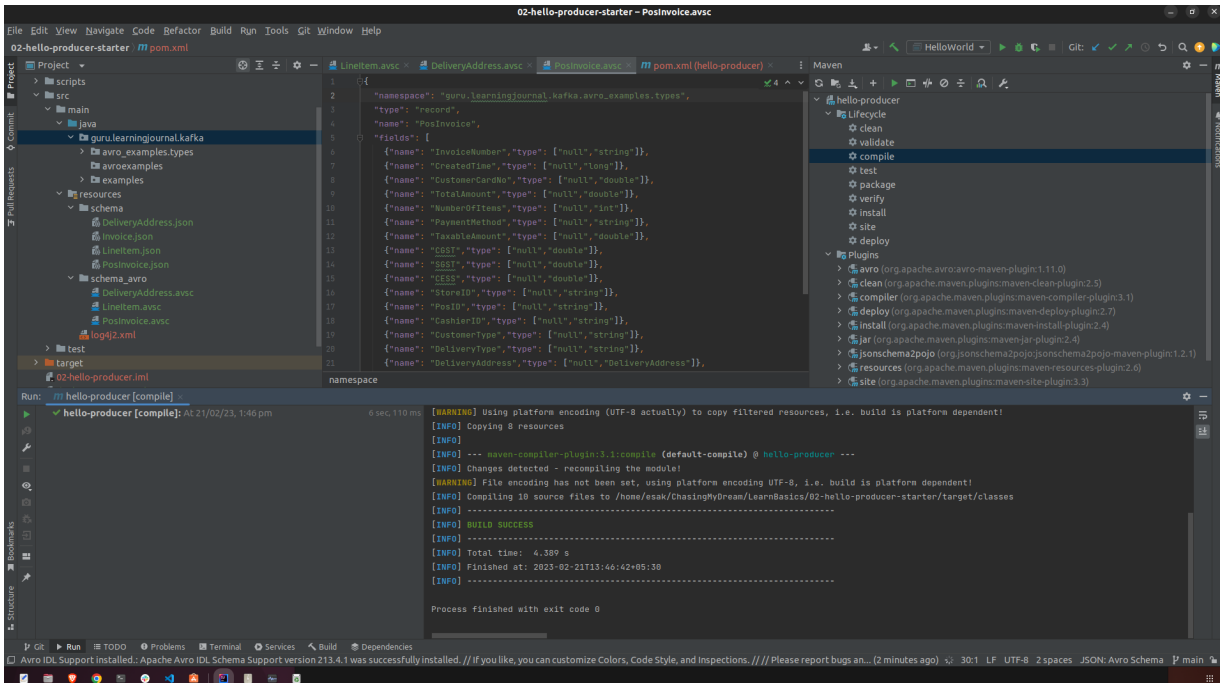
Lets Create the Delivery Address

```
{
  "namespace": "guru.learningjournal.kafka.avroexamples.types",
  "type": "record",
  "name": "DeliveryAddress",
  "fields": [
    {"name": "AddressLine","type": ["null","string"]},
    {"name": "City","type": ["null","string"]},
    {"name": "State","type": ["null","string"]},
    {"name": "PinCode","type": ["null","string"]},
    {"name": "ContactNumber","type": ["null","string"]}
  ]
}
```

Lets Create POS INvoice Schema

```
{
  "namespace": "guru.learningjournal.kafka.avroexamples.types",
  "type": "record",
  "name": "PosInvoice",
  "fields": [
    {"name": "InvoiceNumber","type": ["null","string"]},
    {"name": "CreatedTime","type": ["null","long"]},
    {"name": "CustomerCardNo","type": ["null","double"]},
    {"name": "TotalAmount","type": ["null","double"]},
    {"name": "NumberOfItems","type": ["null","int"]},
    {"name": "PaymentMethod","type": ["null","string"]},
    {"name": "TaxableAmount","type": ["null","double"]},
    {"name": "CGST","type": ["null","double"]},
    {"name": "SGST","type": ["null","double"]},
    {"name": "CESS","type": ["null","double"]},
    {"name": "StoreID","type": ["null","string"]},
    {"name": "PosID","type": ["null","string"]},
    {"name": "CashierID","type": ["null","string"]},
    {"name": "CustomerType","type": ["null","string"]},
    {"name": "DeliveryType","type": ["null","string"]},
    {"name": "DeliveryAddress","type": ["null","DeliveryAddress"]},
    {"name": "InvoiceLineItems","type": {"type": "array", "items": "LineItem"}}
  ]
}
```

Lets Compile the Project



Now all of our Classes are serializable.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>guru.learningjournal.kafka.examples</groupId>
    <artifactId>hello-producer</artifactId>
    <version>2.3.0</version>

    <properties>

        <kafka.version>3.1.0</kafka.version>
        <maven.compiler.source>17</maven.compiler.source>
        <maven.compiler.target>17</maven.compiler.target>
    </properties>
<build>
    <plugins>


        <plugin>
            <groupId>org.jsonschema2pojo</groupId>
            <artifactId>jsonschema2pojo-maven-plugin</artifactId>
            <version>1.2.1</version>

            <executions>
                <execution>
                    <goals>
                        <goal>generate</goal>
                    </goals>
                    <configuration>
                        <sourceDirectory>${project.basedir}/src/main/resources/schema</sourceDirectory>
                        <outputDirectory>${project.basedir}/src/main/java/</outputDirectory>
                        <includeAdditionalProperties>false</includeAdditionalProperties>
                        <includeHashcodeAndEquals>false</includeHashcodeAndEquals>
                        <generateBuilders>true</generateBuilders>

                    </configuration>
                </execution>
            </executions>
        </plugin>

        <plugin>
            <groupId>org.apache.avro</groupId>
```

```
            <artifactId>avro-maven-plugin</artifactId>
            <version>1.11.0</version>
            <executions>
                <execution>
                    <phase>generate-sources</phase>
                    <goals>
                        <goal>schema</goal>
                    </goals>
                    <configuration>
                        <sourceDirectory>${project.basedir}/src/main/resources/schema</sourceDirectory>
                        <outputDirectory>${project.basedir}/src/main/java/</outputDirectory>
                        <imports>
                            <import>${project.basedir}/src/main/resources/schema_avro/LineItem.avsc</import>
                            <import>${project.basedir}/src/main/resources/schema_avro/DeliveryAddress.avsc</import>
                        </imports>
                    </configuration>
                </execution>
            </executions>
        </plugin>

    </plugins>
</build>



    <dependencies>
        <!-- Apache Kafka Clients-->
        <dependency>
            <groupId>org.apache.kafka</groupId>
            <artifactId>kafka-clients</artifactId>
            <version>${kafka.version}</version>
        </dependency>

        <!-- https://mvnrepository.com/artifact/org.slf4j/slf4j-api -->
        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-api</artifactId>
            <version>1.7.32</version>
        </dependency>

        <!-- https://mvnrepository.com/artifact/org.slf4j/slf4j-simple -->
        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-simple</artifactId>
            <version>1.7.32</version>
        </dependency>
<!--        we need jackson databind -->
        <dependency>
            <groupId>commons-lang</groupId>
            <artifactId>commons-lang</artifactId>
            <version>2.6</version>
        </dependency>
        <dependency>
            <groupId>com.fasterxml.jackson.core</groupId>
            <artifactId>jackson-annotations</artifactId>
            <version>2.14.2</version>
        </dependency>
<!--        Adding avro depedency -->

        <dependency>
            <groupId>org.apache.avro</groupId>
            <artifactId>avro</artifactId>
            <version>1.11.0</version>
        </dependency>


    </dependencies>

</project>
```

8. Produce Records with schemas

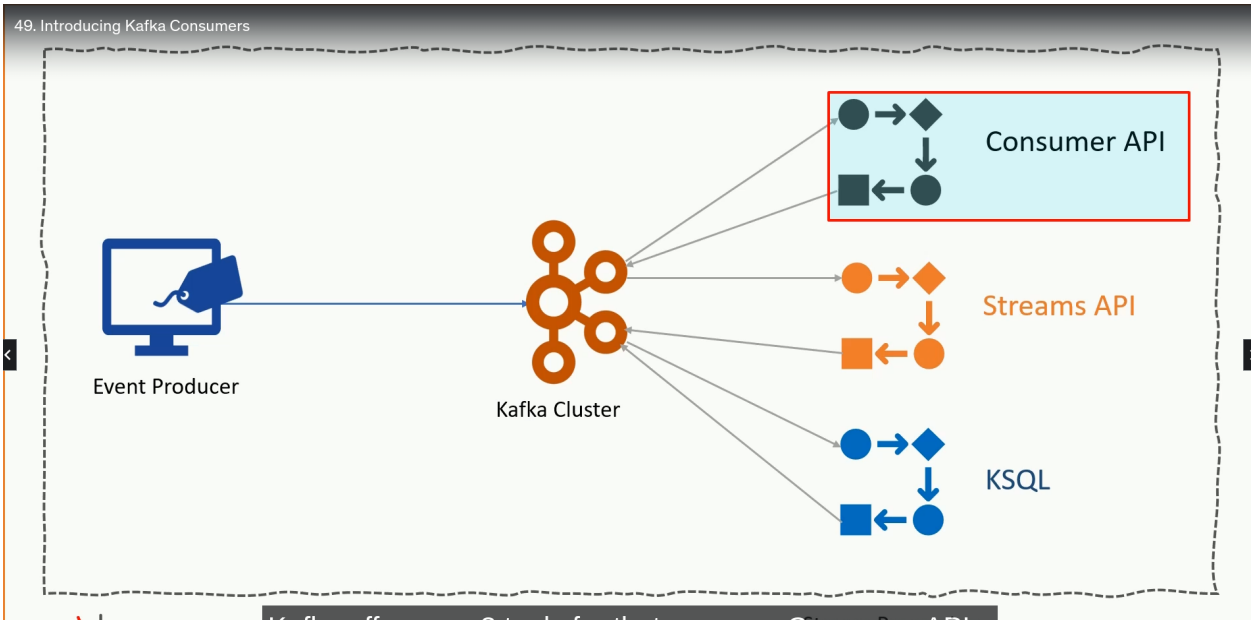
We want to produce records and send it to Kafka Broker

Create Simple Data Producer with Avro Schemas

Create Simple Data with Multi Thread producer

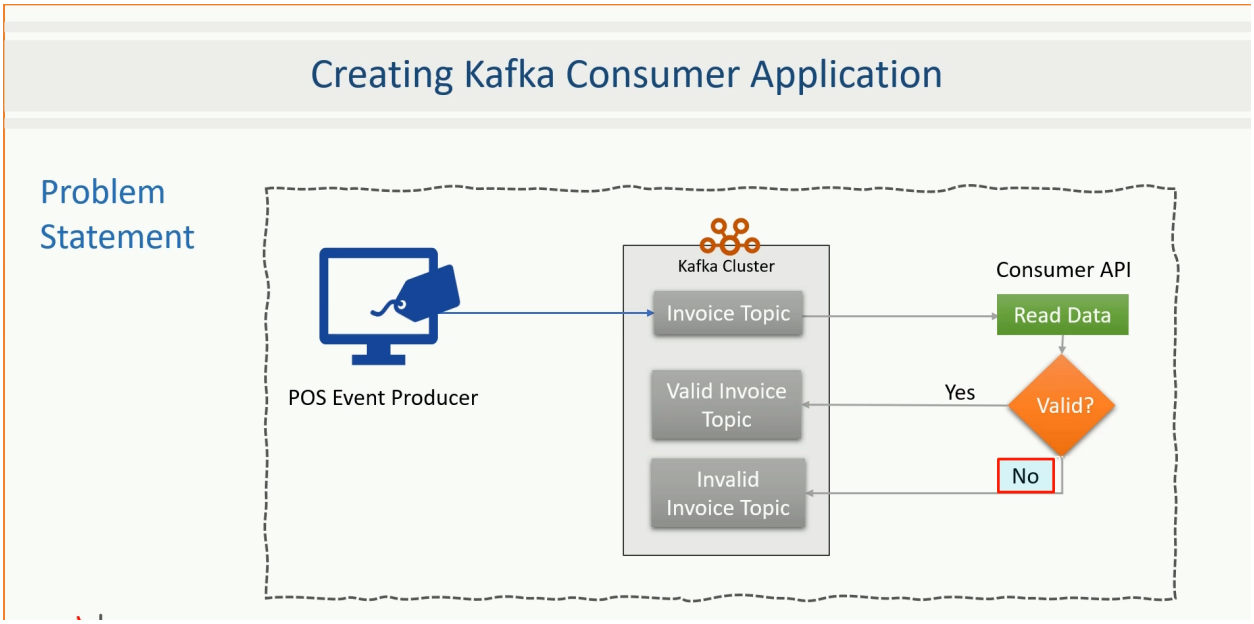
Consumers

Once your streams starts flowing into kafka you are ready to tap into these streams and plug in our stream processing applications.



Lets Build our Consumers

We have the generated a producer which generates series of invoices and send it to kafka broker. we want to read all invoice in real time and apply some business rules and if its passed send them to the valid kafka topic .



Lets Build the valid business usecases

```
Lets say if Delivery Type is Home Delivery and delivery address contact number is NUll , then we should treat it as INvalid Records
```

Step 1 Create the Properties

```
Properties consumerprops = new Properties();
        consumerprops.put(ConsumerConfig.CLIENT_ID_CONFIG, AppConfig.applciationId);
        consumerprops.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, AppConfig.bootstrapServers);
        consumerprops.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG, StringDeserializer.class);
        consumerprops.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG, StringDeserializer.class);
        consumerprops.put(ConsumerConfig.GROUP_ID_CONFIG, AppConfig.groupid);
        consumerprops.put(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG, "earliest");
```

Step 2 Instance Of Kafka Consumer Class

```
KafkaConsumer<String, String> consumer = new KafkaConsumer<>(consumerprops);
```

Step 3 Subscribe to the Topic

```
consumer.subscribe(Arrays.asList(AppConfig.topicName));
```

Step 4 Read the topic Message

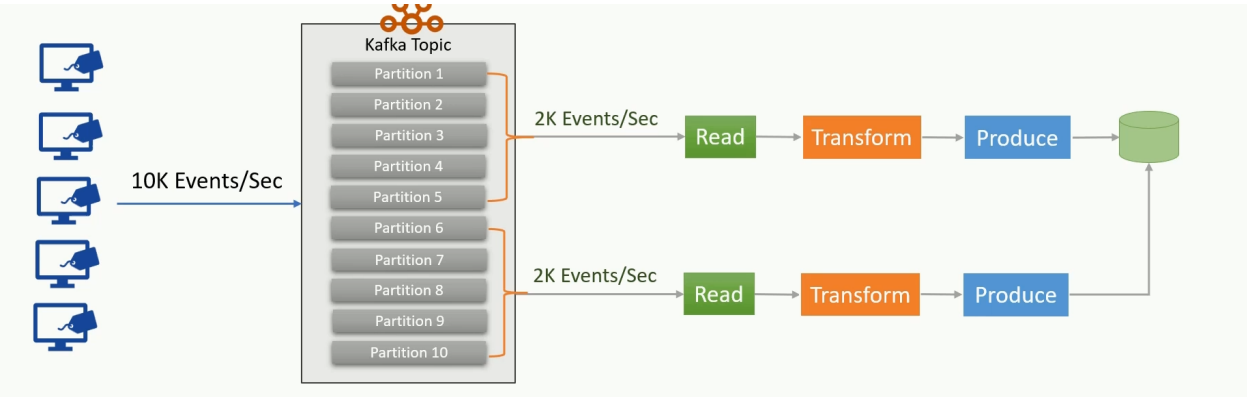
```
while(true) {
            ConsumerRecords<String, String> records = consumer.poll(Duration.ofMillis(100));
            for(ConsumerRecord<String, String> rec : records) {
                System.out.println(rec.key() + "-------"+rec.value());
            }
        }
```

OverAll Code

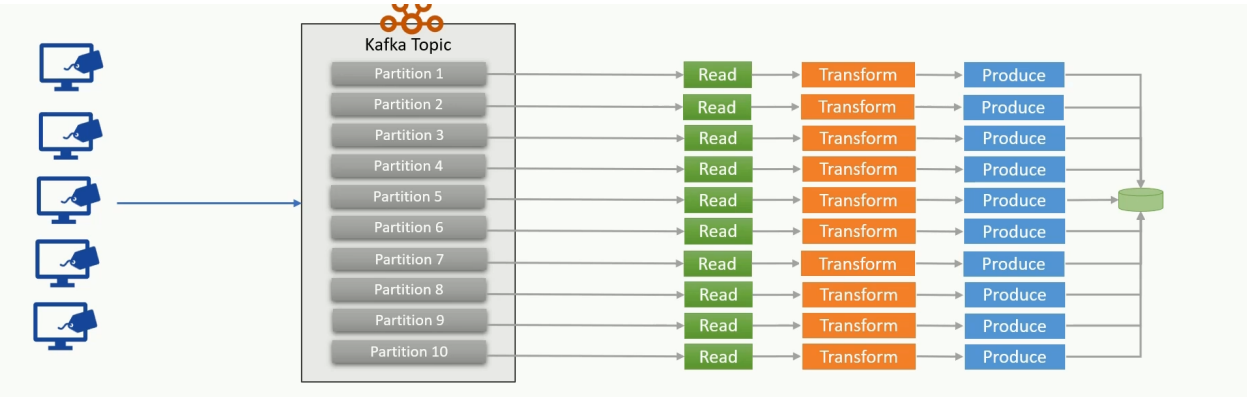
```
package org.example;

import org.apache.kafka.clients.consumer.*;
import org.apache.kafka.common.serialization.StringDeserializer;


import java.time.Duration;
import java.util.Arrays;
import java.util.Properties;

public class SimpleConsumer {

    public static void main(String[] args) {

        Properties consumerprops = new Properties();
        consumerprops.put(ConsumerConfig.CLIENT_ID_CONFIG, AppConfig.applciationId);
        consumerprops.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, AppConfig.bootstrapServers);
        consumerprops.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG, StringDeserializer.class);
        consumerprops.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG, StringDeserializer.class);
        consumerprops.put(ConsumerConfig.GROUP_ID_CONFIG, AppConfig.groupid);
        consumerprops.put(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG, "earliest");

//
        KafkaConsumer<String, String> consumer = new KafkaConsumer<>(consumerprops);

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

        while(true) {
            ConsumerRecords<String, String> records = consumer.poll(Duration.ofMillis(100));
            for(ConsumerRecord<String, String> rec : records) {
                System.out.println(rec.key() + "-------"+rec.value());
            }
        }

    }



}
```

Consumer Group And Scalability

WE can Scale consumer process by dividing the work among multiple consumers , we allow various consumers to read the same topic. We can split the data among consumers by assigning them one or more partitions. IN this each consumer is assigned to their Set of partitions and they read only from the assigned partitions



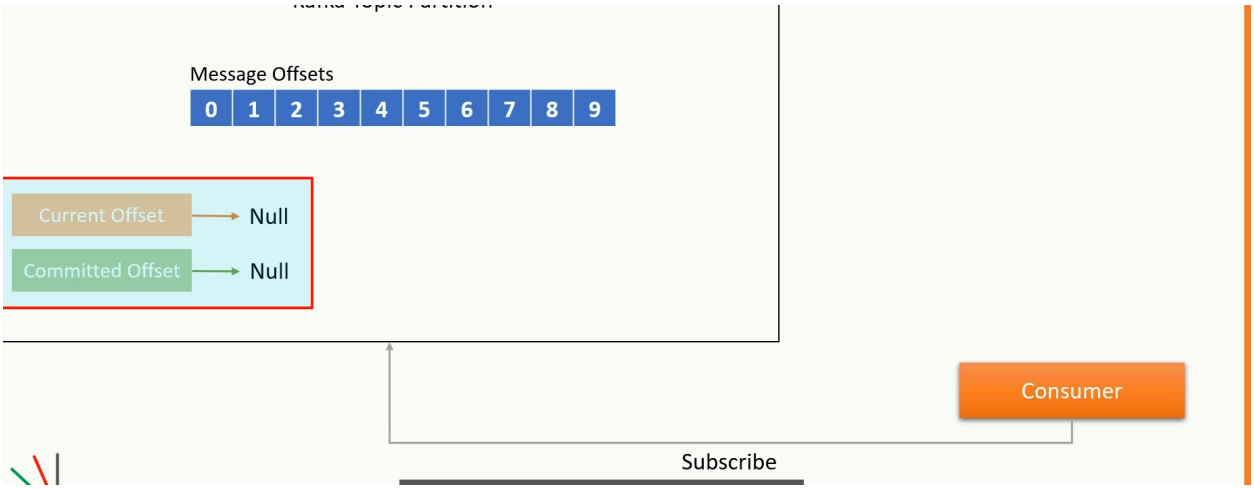
Max number of consumers are equal to the maximum number of partitions



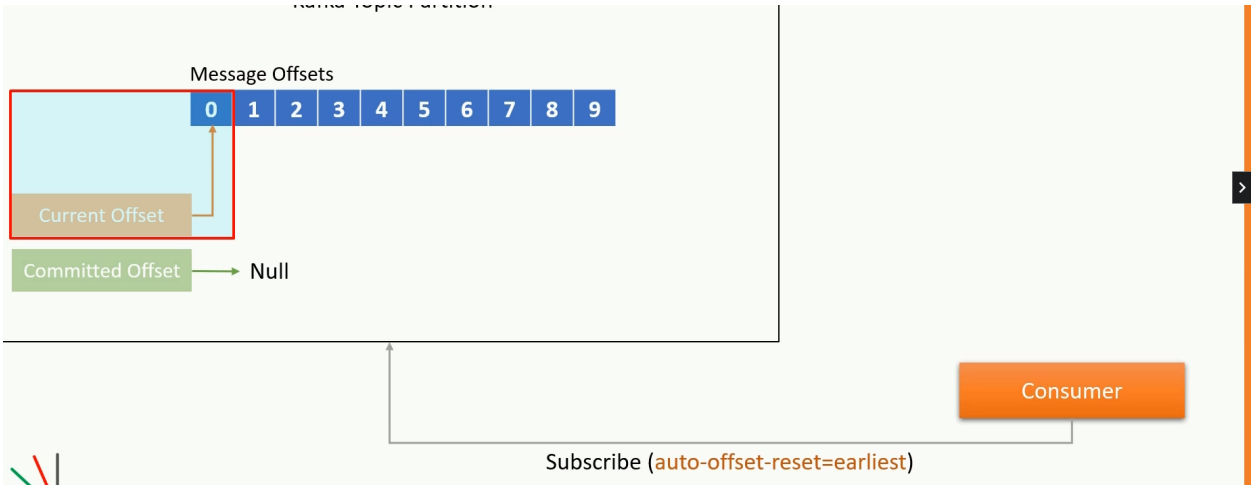
Current Offsets Vs Committed Offsets

IN a Consumer Group , consumer was reading data from the broker and after a while it got crashed. So kafka Automatically re balance the partition to another consumer in the group. New Should shouldn't reprocess the events that are already processed by old failed consumer.

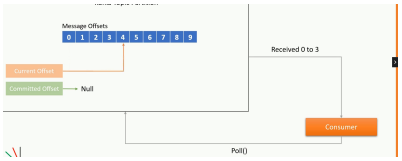
Offset is used to identify the message in the partitions.Kafka maintains 2 more offsets for each partitions



Current Offset ==⇒ It starts giving all the messages from the beginning

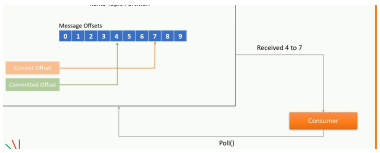


Once we start to Poll the Messages , lets Say poll 3 messages

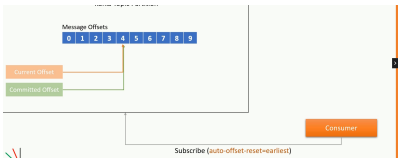


Current Offset is live with the Consumer session , once the Consumer Dies it move back to the earliest offset position

Every Time poll consumer will automatically committed the earlier current offset and send some more records. This is known as auto commit.



When the consumer restarts , current offest is assigned to the committed offset and stars reading the data.



If you are interested in Loading the Data to Datalake then Go For Spark Streaming

If you are interested in Microservices Then Go for Kafka Streams

Lets Produce Data with Avro Serialization

Section 2 - Kafka With SpringBoot

2.01 Creating the Producer

Lets Build a Multi Module maven Project

2.02 Create a Simple String Kafka Producer

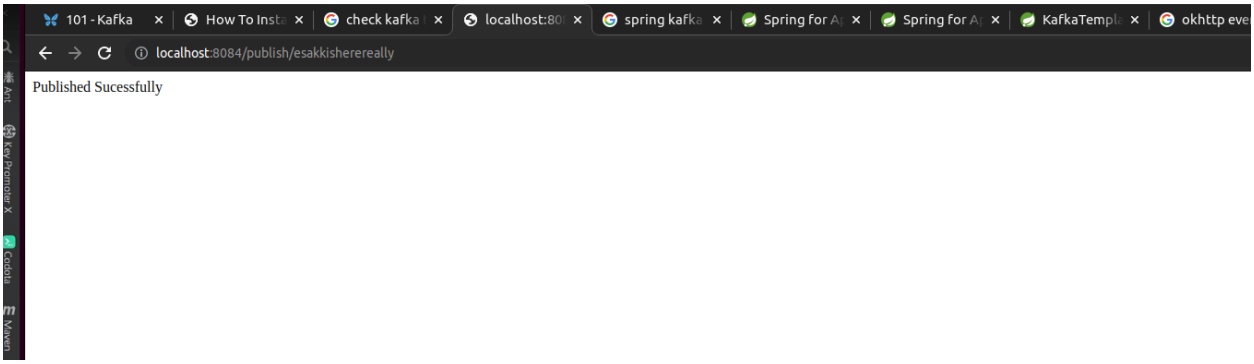
Main Class

```
@SpringBootApplication
public class ProcuerAppMain {

    public static void main(String[] args) {
        SpringApplication.run(ProcuerAppMain.class);
    }
}
```

Controller Class

```
package com.cogesak.dailyKafka;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.kafka.core.KafkaTemplate;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class Controller {

    @Autowired
    KafkaTemplate<String, String> kafkatemplate;

    public static final String TOPIC = "dailycodeBuffer";

    @GetMapping("/publish/{message}")
    public String publishMessage(@PathVariable("message") String message )
    {

        kafkatemplate.send(TOPIC, message);
        return "Published Sucessfully";
    }
}
```

Now when the URL is request we see a Message in the Kafka Topic



2.04 Sending the Data in Json Format

Producer Config

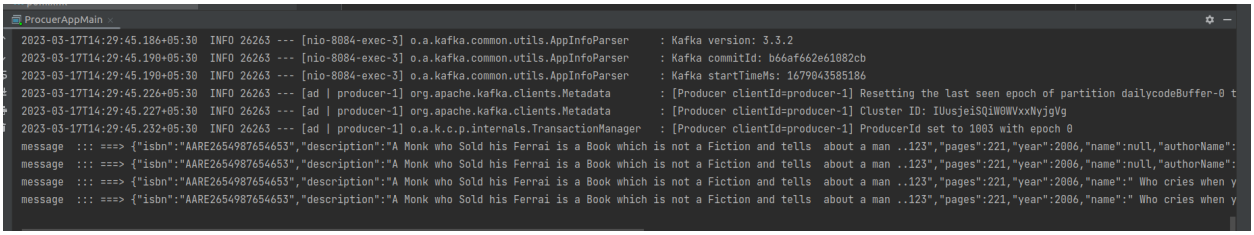
```
package com.cogesak.dailyKafka;

import org.apache.kafka.clients.producer.ProducerConfig;
import org.apache.kafka.common.serialization.StringSerializer;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.kafka.core.DefaultKafkaProducerFactory;
import org.springframework.kafka.core.KafkaTemplate;
import org.springframework.kafka.core.ProducerFactory;
import org.springframework.kafka.support.serializer.JsonSerializer;

import java.util.HashMap;
import java.util.Map;


@Configuration
public class kafkaConfig {
    @Bean
    public ProducerFactory<String, Book> producerFactory () {

        Map<String, Object> config = new HashMap<>();
        config.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:9092");
        config.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG, StringSerializer.class);
        config.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG, JsonSerializer.class);

        return new DefaultKafkaProducerFactory<>(config);
    }


    @Bean
    public KafkaTemplate kafkatemplate () {
        return new KafkaTemplate<>(producerFactory());
    }
}
```

Controller class

```
@PostMapping("/book")
    public String publishBook(@RequestBody Book book) {
        bookkafkaTemplate.send(TOPIC, book);
        return "Published Sucessfully";
    }
```

```
test
esak@esak-PC:~$ kafka-console-consumer.sh --topic dailycodeBuffer --from-beginning --bootstrap-server localhost:9092
esakkishere
esakkishere
esakkishere
esakkishere
esakkishere
esakkishereeally
{"isbn":"AARE2654987654653","description":"A Monk who Sold his Ferrai is a Book which is not a Fiction and tells  about a man ..","page
s":221,"year":2006,"name":null,"authorName":null}
{"isbn":"AARE2654987654653","description":"A Monk who Sold his Ferrai is a Book which is not a Fiction and tells  about a man ..12","pa
ges":221,"year":2006,"name":null,"authorName":null}
{"isbn":"AARE2654987654653","description":"A Monk who Sold his Ferrai is a Book which is not a Fiction and tells  about a man ..123","p
ages":221,"year":2006,"name":null,"authorName":null}
```

2.05 Lets Write the Consumer



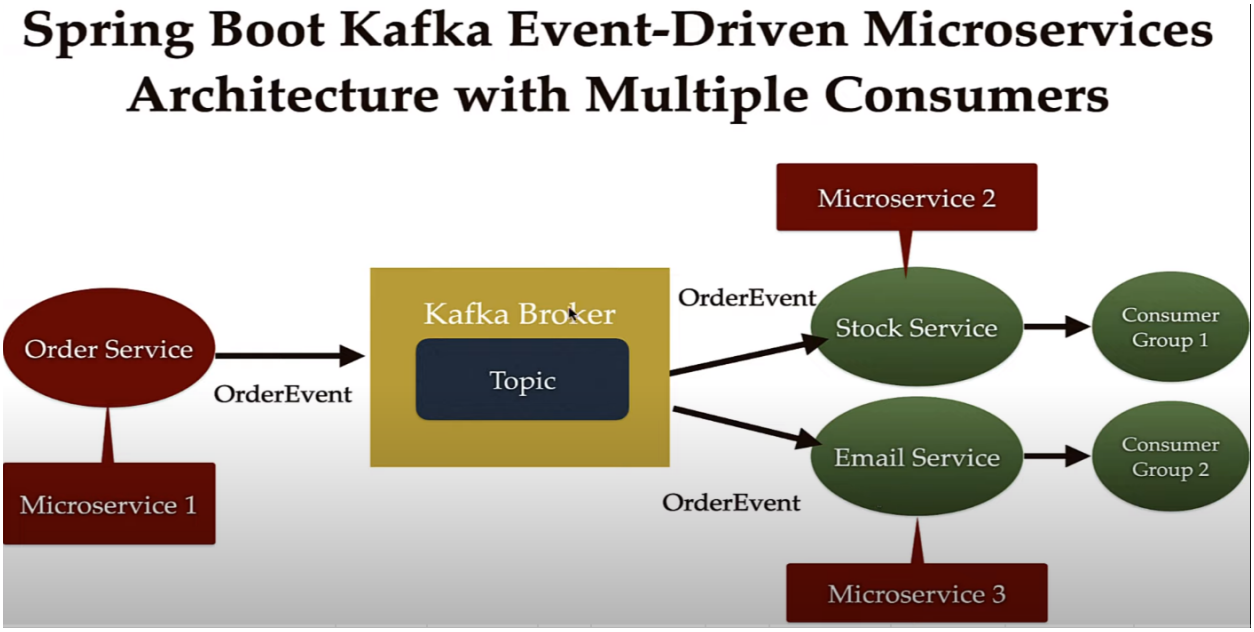

```
@Bean
    public ConsumerFactory<String, String> consumerfactory() {
        Map<String, Object> consumerconfig = new HashMap<>();
        consumerconfig.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:9092");
        consumerconfig.put(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG, "beginning");
        consumerconfig.put(ConsumerConfig.GROUP_ID_CONFIG, "cogesak_group_111");
        consumerconfig.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG, StringDeserializer.class);
        consumerconfig.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG, StringDeserializer.class);

        return new DefaultKafkaConsumerFactory<>(consumerconfig);


    }

    @Bean
    public ConcurrentKafkaListenerContainerFactory containerFactory() {
        ConcurrentKafkaListenerContainerFactory<String, String> factory = new ConcurrentKafkaListenerContainerFactory<>();
        factory.setConsumerFactory(consumerfactory());
        return factory;
    }
```

Create the Kafka Consumer Class

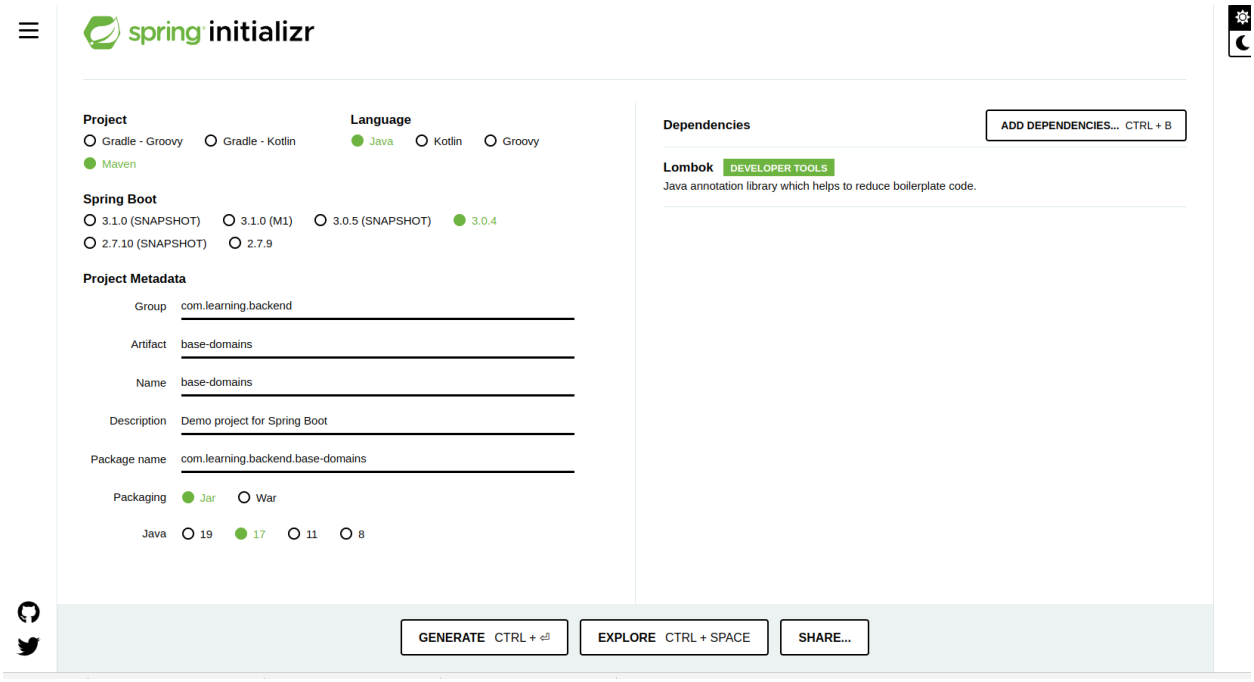
```
package com.cogesak.dailyKafka;

import org.springframework.kafka.annotation.EnableKafka;
import org.springframework.stereotype.Component;
import org.springframework.kafka.annotation.KafkaListener;

@EnableKafka
@Component
public class kafkaConsumer {

    @KafkaListener(topics = "dailycodeBuffer", groupId = "cogesak_group_111")
    public void consume(String message) {

        System.out.println("message  ::: ===> "+message);
    }
}
```

Section 5 - Simple Project

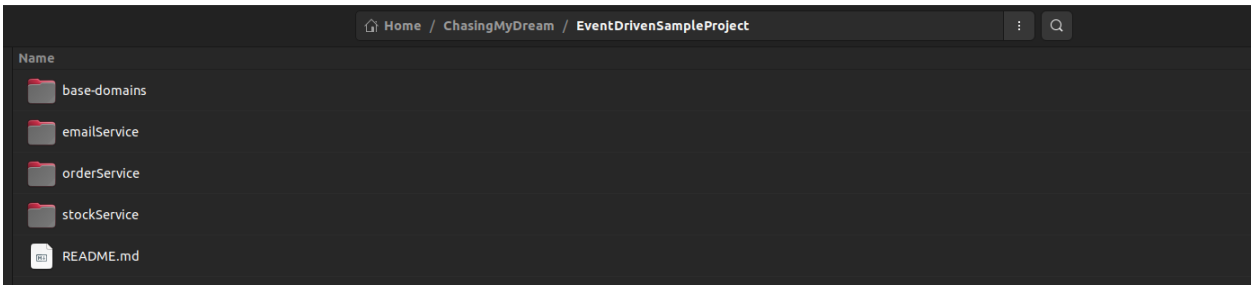
Create a simple Project where we use kafka as a message Broker



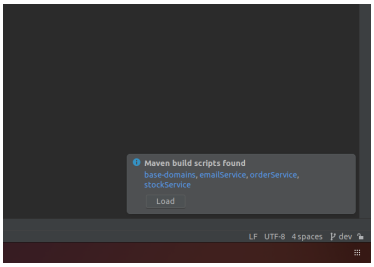
Create OrderService , StockService, EmailService, baseDomains in Start.spring.io websites

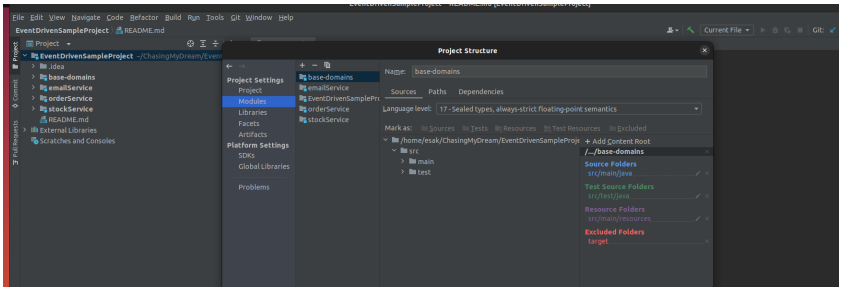


Create a Base Project and Move all the Extracted Folders to the base root Folder



Open the Project in Intelij and verify the These project are imported as Modules





Step00 - we are going to run these spring boot project in Different Ports

```
server.port=6060
server.port=6061
server.port=6062
server.port=6063
```

Creating DTO Classes

DTO calsses are used to transfer the data between services

Lets Create a OrderEvent dto Class which is used to Transfer across services using kafka

```
package com.learning.backend.basedomains.DTO;

import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@AllArgsConstructor
@NoArgsConstructor
@Builder
public class OrderEvent {

    private String message;
    private String status;
```

```
    private Orders orders;
}
```

Lets create Orders Class

```
package com.learning.backend.basedomains.DTO;

import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@AllArgsConstructor
@NoArgsConstructor
@Builder
public class Orders {

    private String orderId;
    private String orderName;
    private int qty;
    private Double price;

}
```

Step 01 - Building the Order Service Producer

Add the properties as below

```
server.port=6060
spring.kafka.producer.bootstrap-servers=localhost:9092
spring.kafka.producer.key-serializer=org.apache.kafka.common.serialization.StringSerializer
spring.kafka.producer.value-serializer=org.springframework.kafka.support.serializer.JsonSerializer
spring.kafka.topic.name=order_topics
```

Create a New Topic

```
package com.learning.backend.orderService.config;

import org.apache.kafka.clients.admin.NewTopic;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.kafka.config.TopicBuilder;

@Configuration
public class KafkaConfig {
    @Value("${spring.kafka.topic.name}")
    private String topicName;

    @Bean
    public NewTopic createTopic() {
        return TopicBuilder.name(topicName).build();
    }
}
```

Add the OrderEvent Class to the pom

```
<!--    Add ORder Event Class as dependency -->
    <dependency>
      <groupId>com.learning.backend</groupId>
      <artifactId>base-domains</artifactId>
      <version>0.0.1-SNAPSHOT</version>
    </dependency>
```

Publish the Event to Kafka

Create the Controller

```
package com.learning.backend.orderService.controller;

import com.learning.backend.basedomains.DTO.OrderEvent;
import com.learning.backend.basedomains.DTO.Orders;
import com.learning.backend.orderService.kafka.OrderProducer;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import java.util.UUID;

@RestController
@RequestMapping("/api/v1")
public class OrderController {

    @Autowired
    public OrderProducer orderprod;

    @PostMapping("/order")
    public String createOrder(@RequestBody Orders orders) {

        orders.setOrderId(UUID.randomUUID().toString());

        OrderEvent orderevent = new OrderEvent();
        orderevent.setStatus("PENDING");
        orderevent.setMessage("Order status is in Pending Status..");
        orderevent.setOrders(orders);
        System.out.println(orders);
        System.out.println(orderevent);
        orderprod.sendMessages(orderevent);

        return "Order Event Has been Sent Sucessfully";

    }
}
```

Create the Kafka Config

```
package com.learning.backend.orderService.config;

import com.learning.backend.basedomains.DTO.OrderEvent;
import org.apache.kafka.clients.admin.NewTopic;
import org.apache.kafka.clients.producer.ProducerConfig;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.kafka.config.TopicBuilder;
import org.springframework.kafka.core.DefaultKafkaProducerFactory;
import org.springframework.kafka.core.KafkaTemplate;
import org.springframework.kafka.core.ProducerFactory;

import java.util.HashMap;
import java.util.Map;

@Configuration
public class KafkaConfig {
    @Value("${spring.kafka.topic.name}")
    private String topicName;

    @Value("${spring.kafka.producer.bootstrap-servers}")
    private String bootstrapservers;

    @Value("${spring.kafka.producer.key-serializer}")
    private String keyserializertype;

    @Value("${spring.kafka.producer.value-serializer}")
    private String valueserializertype;
```

```
    @Bean
    public NewTopic createTopic() {
        return TopicBuilder.name(topicName).build();
    }

    @Bean
    public  ProducerFactory<String, OrderEvent> Orderproducerconfig () {

        Map<String, Object> conf = new HashMap<>();
        conf.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, bootstrapservers);
        conf.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG, keyserializertype);
        conf.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG, valueserializertype);

        return new DefaultKafkaProducerFactory<>(conf);

    }

    @Bean
    public KafkaTemplate kafaktemplatemethod () {
        return new KafkaTemplate(Orderproducerconfig());
    }
}
```

Send the Message

```
package com.learning.backend.orderService.kafka;

import com.learning.backend.basedomains.DTO.OrderEvent;
import org.apache.kafka.clients.admin.NewTopic;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.kafka.core.KafkaTemplate;
import org.springframework.kafka.core.ProducerFactory;
import org.springframework.stereotype.Service;

import java.util.HashMap;
import java.util.Map;

@Service
public class OrderProducer {

    public static final Logger LOGGER = LoggerFactory.getLogger(OrderProducer.class);
    @Autowired
    private NewTopic orderstopic;

    @Value("${spring.kafka.topic.name}")
    private String topicName;

    @Autowired
    private KafkaTemplate<String, OrderEvent> kafkatemplate;



    public void sendMessages(OrderEvent orderevent){

        LOGGER.info(String.format("Order Event is ==> %s", orderevent.toString()));

        kafkatemplate.send( topicName, orderevent);
    }
}
```

```
esak@esak-PC:~/ChasingMyDream/EventDrivenSampleProject$ kafka-console-consumer.sh --topic order_topics --from-beginning --bootstrap-server localhost:9092
{"message":"Order status is in Pending Status..","status":"PENDING","orders":{"orderId":"2e09aef2-e86b-4825-a30c-15f8180cb04e","orderName":null,"qty":10,"price":1599.99}}
{"message":"Order status is in Pending Status..","status":"PENDING","orders":{"orderId":"a4bd99a2-7b5f-4975-a2c7-4877b52ac415","orderName":null,"qty":3,"price":1199.99}}
{"message":"Order status is in Pending Status..","status":"PENDING","orders":{"orderId":"dfca1109-c0a2-4013-9492-7727a2fd028a","orderName":null,"qty":3,"price":1199.99}}
{"message":"Order status is in Pending Status..","status":"PENDING","orders":{"orderId":"5ccefedf-2bfd-481d-b420-1ad5037ff1e4","orderName":null,"qty":3,"price":1199.99}}
{"message":"Order status is in Pending Status..","status":"PENDING","orders":{"orderId":"f34f6624-0c7b-4eaf-b7b6-c616cb3bb420","orderName":null,"qty":3,"price":1199.99}}
{"message":"Order status is in Pending Status..","status":"PENDING","orders":{"orderId":"baa6543b-363d-49c5-8b49-33fc5ab63702","orderName":"Mac Book Pro - M1 ","qty":3,"price":1199.99}}
{"message":"Order status is in Pending Status..","status":"PENDING","orders":{"orderId":"7383b37a-d52b-4047-a983-7c7e3de5688b","orderName":"Mac Book Pro - M1 ","qty":3,"price":1199.99}}
{"message":"Order status is in Pending Status..","status":"PENDING","orders":{"orderId":"7a4f60d3-dce9-4db3-970f-00d243210780","orderName":"Mac Book Pro - M1 ","qty":3,"price":1199.99}}
{"message":"Order status is in Pending Status..","status":"PENDING","orders":{"orderId":"dcb14548-7bf7-46e5-894e-cc3fe4a292b3","orderName":"Mac Book Pro - M1 ","qty":3,"price":1199.99}}
{"message":"Order status is in Pending Status..","status":"PENDING","orders":{"orderId":"1190befe-1403-45ed-8a64-4ac24a456bd4","orderName":"DELL Inspiraon 15  ","qty":3,"price":1199.99}}
```

Sending Message with Ack

```
package com.learning.backend.orderService.kafka;

import com.learning.backend.basedomains.DTO.OrderEvent;
import org.apache.kafka.clients.admin.NewTopic;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.kafka.core.KafkaTemplate;
import org.springframework.kafka.core.ProducerFactory;
import org.springframework.kafka.support.SendResult;
import org.springframework.stereotype.Service;
import org.springframework.util.concurrent.ListenableFuture;

import java.util.HashMap;
import java.util.Map;
import java.util.concurrent.CompletableFuture;



@Service
public class OrderProducer {

    public static final Logger LOGGER = LoggerFactory.getLogger(OrderProducer.class);
    @Autowired
    private NewTopic orderstopic;

    @Value("${spring.kafka.topic.name}")
    private String topicName;

    @Autowired
    private KafkaTemplate<String, OrderEvent> kafkatemplate;



    public void sendMessages(OrderEvent orderevent){

        LOGGER.info(String.format("Order Event is ==> %s", orderevent));

        CompletableFuture<SendResult<String,OrderEvent>>  myfuture = kafkatemplate.send(topicName, orderevent);
        myfuture.whenComplete( (dresult, ex) -> {
            if (ex != null) {
                LOGGER.error("Execution failed", ex);
            } else {
                LOGGER.info("we get a result !!");
                LOGGER.info("Execution completed: {}", dresult);
                LOGGER.info("Execution completed: producerREcord {}", dresult.getProducerRecord());
                LOGGER.info("Execution completed: producerMetadata {}", dresult.getRecordMetadata());
                LOGGER.info("Execution completed: producerString{}", dresult.toString());

            }
        });


    }
}
```

Working with StockService

When to use ConcurrentKafkaListenerContainerFactory?

I am new to kafka and i went through the documentation but I couldn't understand anything. Can someone please explain when to use the ConcurrentKafkaListenerContainerFactory class? I have used the

https://stackoverflow.com/questions/55023240/when-to-use-concurrentkafkalistenercontainerfactory

More Reference

Using Kafka with Spring Boot

How to use Spring Kafka to send messages to and receive messages from Kafka.

https://reflectoring.io/spring-boot-kafka/

Consumer Code Config

```
package com.learning.backend.emailService.kafka;

import com.learning.backend.basedomains.DTO.OrderEvent;
import org.slf4j.LoggerFactory;
import org.slf4j.Logger;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.kafka.annotation.EnableKafka;
import org.springframework.kafka.annotation.KafkaListener;
import org.springframework.stereotype.Component;

@Component
@EnableKafka
public class OrderConsumer {

    public static final Logger LOGGER = LoggerFactory.getLogger(OrderConsumer.class);

    @KafkaListener(topics =  "${spring.kafka.topic.name}" , groupId = "${spring.kafka.consumer.group_id}")
    public void consumer(OrderEvent orderevent) {

        LOGGER.info(String.format("Order event received in Email --%s", orderevent));
        System.out.println(orderevent);

//        Send Email Based on the Order Event Details


    }
}
```

Consumer Code

```
package com.learning.backend.emailService.config;

import org.apache.kafka.clients.consumer.ConsumerConfig;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.kafka.config.ConcurrentKafkaListenerContainerFactory;
import org.springframework.kafka.core.ConsumerFactory;
import org.springframework.kafka.core.DefaultKafkaConsumerFactory;

import java.util.HashMap;
import java.util.Map;

@Configuration
public class KafkaConfig {

    @Value("${spring.kafka.consumer.key-deserializer}")
    private String keydeserializertype;

    @Value("${spring.kafka.consumer.value-deserializer}")
    private String valuedeserializertype;


    @Bean
    public ConsumerFactory<String, String> consumerfactory() {
        Map<String, Object> consumerconfig = new HashMap<>();
        consumerconfig.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:9092");
        consumerconfig.put(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG, "beginning");
```

```
        consumerconfig.put(ConsumerConfig.GROUP_ID_CONFIG, "cogesak_group_111");
        consumerconfig.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG, keydeserializertype);
        consumerconfig.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG, valuedeserializertype);

        return new DefaultKafkaConsumerFactory<>(consumerconfig);


    }

    @Bean
    public ConcurrentKafkaListenerContainerFactory containerFactory() {
        ConcurrentKafkaListenerContainerFactory<String, String> factory = new ConcurrentKafkaListenerContainerFactory<>();
        factory.setConsumerFactory(consumerfactory());
        return factory;
    }

}
```

Email Service

```
server.port=6061
spring.kafka.consumer.bootstrap-servers=localhost:9092
spring.kafka.consumer.group_id=email_group
spring.kafka.consumer.auto-offset-reset=earliest
spring.kafka.consumer.key-deserializer=org.apache.kafka.common.serialization.StringDeserializer
spring.kafka.consumer.value-deserializer=org.springframework.kafka.support.serializer.JsonDeserializer
spring.kafka.consumer.properties.spring.json.trusted.packages=*
spring.kafka.topic.name=order_topics
spring.mail.host=smtp.gmail.com
spring.mail.port=587
spring.mail.username=esakdev9999@gmail.com
spring.mail.password=yxwhuyhodsykpjmt
spring.mail.properties.mail.smtp.auth=true
spring.mail.properties.mail.smtp.starttls.enable=true
```

Email Sender Code

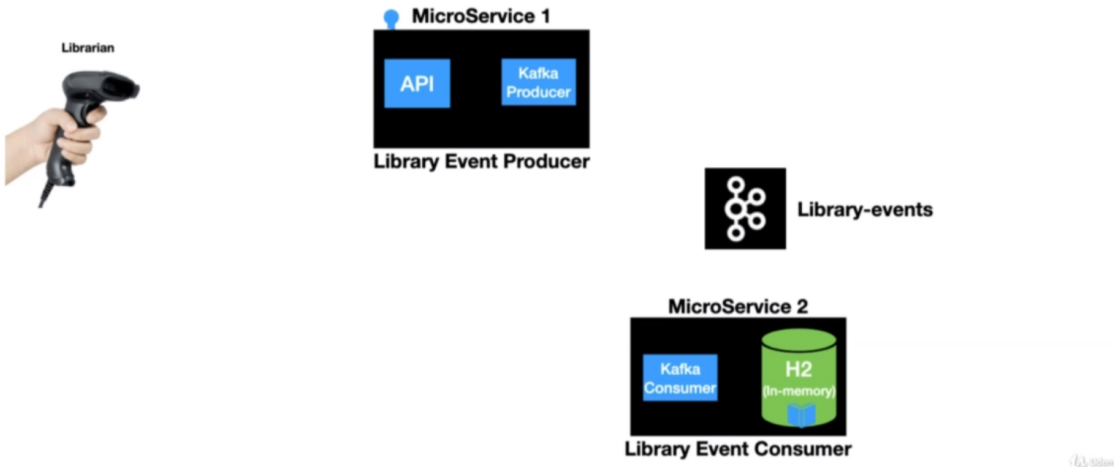
```
package com.learning.backend.emailService.service;

import jakarta.mail.MessagingException;
import jakarta.mail.internet.MimeMessage;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.mail.MailSender;
import org.springframework.mail.javamail.JavaMailSender;
import org.springframework.mail.javamail.MimeMessageHelper;
import org.springframework.stereotype.Service;

@Service
public class EmailSender {

    @Autowired
    private JavaMailSender mailsender;
    public void sendEmail(String toEmail, String body, String subject, String Attachment) throws MessagingException {
        MimeMessage message = mailsender.createMimeMessage();
        MimeMessageHelper helper = new MimeMessageHelper(message, true);

        helper.setFrom("spring001mailer#gmail.com");
        helper.setTo(toEmail);
        helper.setText(body);
        helper.setSubject(subject);

        String htmlContent = "<h1>This is a test Spring Boot email</h1>" +
                "<p>It can contain <strong>HTML</strong> content.</p>";



        message.setContent(htmlContent, "text/html; charset=utf-8");


//        To attach a File
```

```
//        FileSystemResource fileSystem = new FileSystemResource( new File(Attachment));
//        mimemessagehelper.addAttachment(fileSystem.getFilename(), fileSystem);



        mailsender.send(message);


    }
}
```

Calling the Email Sender Function

```
package com.learning.backend.emailService.kafka;

import com.learning.backend.basedomains.DTO.OrderEvent;
import com.learning.backend.emailService.service.EmailSender;
import jakarta.mail.MessagingException;
import org.slf4j.LoggerFactory;
import org.slf4j.Logger;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.kafka.annotation.EnableKafka;
import org.springframework.kafka.annotation.KafkaListener;
import org.springframework.stereotype.Component;

@Component
@EnableKafka
public class OrderConsumer {

    @Autowired
    private EmailSender emailsender;
    public static final Logger LOGGER = LoggerFactory.getLogger(OrderConsumer.class);

    @KafkaListener(topics =  "${spring.kafka.topic.name}" , groupId = "${spring.kafka.consumer.group_id}")
    public void consumer(OrderEvent orderevent) throws MessagingException {

        LOGGER.info(String.format("Order event received in Email --%s", orderevent));
        System.out.println(orderevent);

//        Send Email Based on the Order Event Details


        emailsender.sendEmail("esakkisankart@gmail.com", String.format(" We have received below Order %s", orderevent),
String.format("Order received %s",orderevent.getOrders().getOrderId()) , "");

        LOGGER.info("Email Send Sucessfully ");
    }
}
```

Building a Library Inventory Application



01. Setup the Library Event Producer

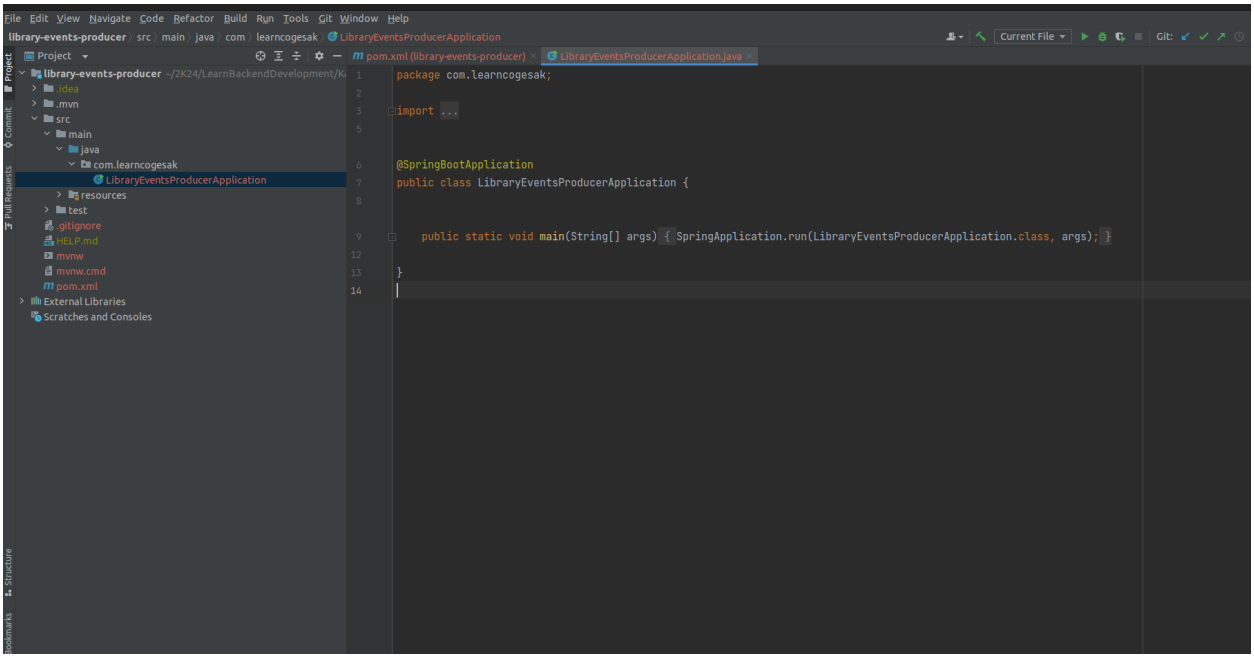
1.1 Required Maven dependency

Go To https://start.spring.io/ and generate the Below File

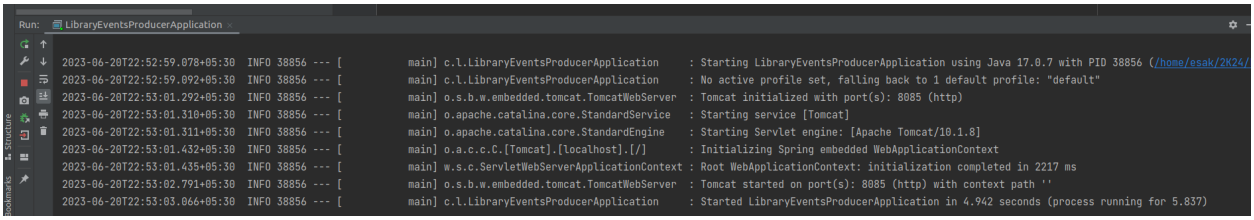
```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>3.1.0</version>
    <relativePath/> <!-- lookup parent from repository -->
  </parent>
  <groupId>com.learncogesak</groupId>
  <artifactId>library-events-producer</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>library-events-producer</name>
  <description>Demo project for Spring Boot</description>
  <properties>
    <java.version>17</java.version>
  </properties>
  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-validation</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.kafka</groupId>
      <artifactId>spring-kafka</artifactId>
    </dependency>

    <dependency>
      <groupId>org.projectlombok</groupId>
      <artifactId>lombok</artifactId>
```

```
      <optional>true</optional>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-test</artifactId>
      <scope>test</scope>
    </dependency>
    <dependency>
      <groupId>org.springframework.kafka</groupId>
      <artifactId>spring-kafka-test</artifactId>
      <scope>test</scope>
    </dependency>
  </dependencies>

  <build>
    <plugins>
      <plugin>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-maven-plugin</artifactId>
        <configuration>
          <excludes>
            <exclude>
              <groupId>org.projectlombok</groupId>
              <artifactId>lombok</artifactId>
            </exclude>
          </excludes>
        </configuration>
      </plugin>
    </plugins>
  </build>

</project>
```

open the Project in Intelij and dependencies will be downloaded automatically



Start the Application



2. Build the Library Event Domain

library create event

```
{
  "libraryEventId": null,
  "libraryEventType": "NEW",
  "book": {
    "bookId": 123,
    "bookName": "Kafka Using Spring Boot",
    "bookAuthor": "Dilip"
  }
}
```

Library update Event

```
{
  "libraryEventId": 123,
  "libraryEventType": "UPDATE",
  "book": {
    "bookId": 123,
    "bookName": "Kafka Using Spring Boot",
    "bookAuthor": "Dilip"
  }
}
```

2.1 Create DTO Class

Create the Library Event Record

```
package com.learncogesak.Domain;



public record LibraryEvent(
        Integer libraryEventId,
        LibraryEventType libraryEventType,
        Book book
) {

}
```

Here we have create 2 Enum Classes

```
package com.learncogesak.Domain;

public enum LibraryEventType {
    NEW,
    UPDATE
}
```

WE have created Book Object

```
package com.learncogesak.Domain;

public record Book(
        Integer bookId,
        String bookName,
        String bookAuthorName
) {
}
```

This is the representation of data flow into our system

3. POST the data to the Producer code

lets create the Controller

we are sending the Messages using the controller Endpoint

```
package com.learncogesak.controller;

import com.learncogesak.Domain.LibraryEvent;
import lombok.extern.slf4j.Slf4j;
import org.apache.coyote.Response;
import org.springframework.http.HttpStatus;
import org.springframework.http.HttpStatusCode;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RestController;

@Slf4j
@RestController
public class LibraryEventsController {

    @PostMapping("/v1/libraryevent")
    public ResponseEntity<LibraryEvent> postLibraryEvent(@RequestBody LibraryEvent libraryevent) {

        log.info("INFO :: we are in the event controller ");
        log.info("INFO :: library Events : {}", libraryevent);

         // invoke Kafka Producer

        return ResponseEntity.status(HttpStatus.CREATED).body(libraryevent);
    }
}


we create the controller using the @RestController Annotation



we are returing the response to the Client via ResponseEntity
```

To Test the Controller we are gonna curl the endpoint

```
curl -i \
-d '{"libraryEventId":null,"libraryEventType": "NEW","book":{"bookId":456,"bookName":"Kafka Using Spring Boot","bookAuthor":"Dilip"}}' \
-H "Content-Type: application/json" \
-X POST http://localhost:8085/v1/libraryevent
```

4 . Kafka Template to Produce Messages

Kafka Template.send() Method will produce the records to the Kafka Topics . Before delivering it to the Kafka Topics messages will undergo some more Layers.

1. serializer - Any records that’s sent to kafka needs to be serialized to bytes . we need to serialize key and values .
2. Partition-er —> Which partition message is going to be placed , mostly we use default partitioner

3. Record Accumulator —> Records Accumulator buffers the records and the records are sent to the kafka topic once the buffer is full. Which helps the number of requests to the cluster from the application which in turn improves the kafka performance.Each RecorddBatch has a size.
    a. Records will be Send to kafka on the 2 different reasons
        i. If the Record Batch is Full
        ii. if the Record batch is available for more than the linger.ms

Mandatory Values :

bootstrap-servers ::
key-serializer ::
value-serializer ::

Lets Create a application.yml File

```
server:
  port: 8085
spring:
  profiles:
    active: local
---
spring:
  config:
    activate:
      on-profile: local
  kafka:
    producer:
      bootstrap-servers: localhost:9092
      key-serializer: org.apache.kafka.common.serialization.IntegerSerializer
      value-serializer: org.apache.kafka.common.serialization.StringSerializer
```

5. Kafka Admin

we can create kafka topics programatically using the spring-kafka

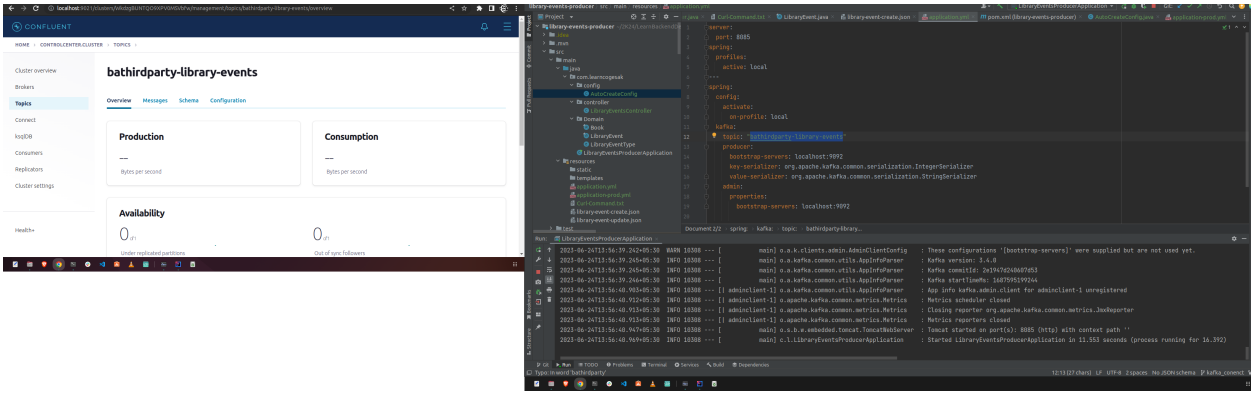
1. we need to create a bean of Type KafkaAdmin in SpringConfiguration
2. we need to create a NewTopic in SpringConfiguration

Create a new class called AutoCreateConfig

```
package com.learncogesak.config;

import org.apache.kafka.clients.admin.NewTopic;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.kafka.config.TopicBuilder;
```

```
@Configuration
public class AutoCreateConfig {

    @Value("${spring.kafka.topic}")
    public String AppTopicName;

    @Bean
    public NewTopic libraryEvents() {
        return TopicBuilder.name(AppTopicName).
                partitions(1).
                replicas(1).
                build();
    }
}
```

Topic is created automatically



6.Produce the Message

KafkaTemplate is used to communicate with the kafka Cluster. This kafkatemplate will be created automatically by the spring framework itself,

```
package com.learncogesak.producer;

import org.springframework.kafka.core.KafkaTemplate;
import org.springframework.stereotype.Component;

@Component
@Sl4j
public class libraryEventsProducer {

    private KafkaTemplate<Integer, String>   kafkatemplate;

//    Lets Use Constructor injection to generate the class

    public libraryEventsProducer(KafkaTemplate<Integer, String> kafkatemplate) {
        this.kafkatemplate = kafkatemplate;
    }

}
```

we are going to produce the library events from the endpoints to the kafka cluster

```
we are going to use the kafkaTemplate to Send the Messages and

this Function is basically return the CompletableFuture Class .
```

Full Code

```
package com.learncogesak.producer;

import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.learncogesak.Domain.LibraryEvent;
import lombok.extern.slf4j.Slf4j;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.kafka.core.KafkaTemplate;
import org.springframework.kafka.support.SendResult;
import org.springframework.stereotype.Component;

import java.util.concurrent.CompletableFuture;

@Component
@Slf4j
public class libraryEventsProducer {

    @Value("${spring.kafka.topic}")
    public String topicName;

    private final KafkaTemplate<Integer, String>   kafkatemplate;

    private final ObjectMapper objectMapper;

//    Lets Use Constructor injection to generate the class
    public libraryEventsProducer(KafkaTemplate<Integer, String> kafkatempla
        this.kafkatemplate = kafkatemplate;
        this.objectMapper = objectMapper;
    }

    public CompletableFuture<SendResult<Integer, String>> sendLibraryEvents
throws JsonProcessingException {

        var key = libraryEvent.libraryEventId();
        var message = objectMapper.writeValueAsString(libraryEvent);
```

```
 This Feature has been added to Java 8 .

var results = kafkatemplate.send(topicName, key, message);

This completableFuture class returns sucess or Error as below

results.whenComplete( (sendResult, throwable) -> {
            if(throwable != null) {

                handleFailure(key, message, throwable);
            }else{

                handleSucesss(key, message, sendResult);
            }
        });


if the result is completed then results will be stored in sendREsults variable , here kafka will return the REcordMetadata INformation
if the results is failed we get a throwable object which has the required information

here we are using 2 different methods to print the Results

private void handleSucesss(Integer key, String message, SendResult<Integer, String> sendResult) {
        log.info("Message Sent Sucessfully for the key - {} and Value: {} , Partition is :: {}", key, message, sendResult.getRecordMetadata
    }

    private void handleFailure(Integer key, String message, Throwable throwable) {
        log.error("Error Sending the Message and the Exception is -- {}",  throwable.getMessage(), throwable);
    }
```

```
        var results = kafkatemplate.send(topicName, key, message);

        return results.whenComplete( (sendResult, throwable) -> {
            if(throwable != null) {

                handleFailure(key, message, throwable);
            }else{

                handleSucesss(key, message, sendResult);
            }
        });
    }

    private void handleSucesss(Integer key, String message, SendResult<Inte
        log.info("Message Sent Sucessfully for the key - {} and Value: {} ,
key, message, sendResult.getRecordMetadata());
    }

    private void handleFailure(Integer key, String message, Throw
        log.error("Error Sending the Message and the Exception is -- {}",
throwable.getMessage(), throwable);
    }
```

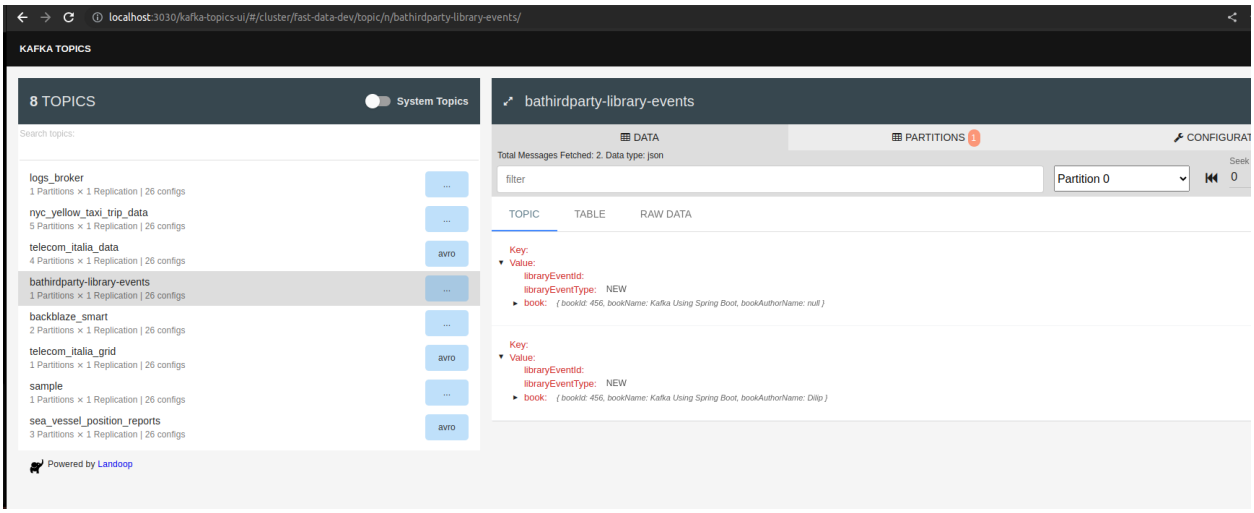
Lets Call these From our Endpoints

Lets inject the libraryEventsProducer to the Control Class

```
package com.learncogesak.controller;

import com.fasterxml.jackson.core.JsonProcessingException;
import com.learncogesak.Domain.LibraryEvent;
import com.learncogesak.producer.LibraryEventsProducer;
import lombok.extern.slf4j.Slf4j;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RestController;

@Slf4j
@RestController
public class LibraryEventsController {

    private final LibraryEventsProducer libraryEventsProducer;

    public LibraryEventsController(LibraryEventsProducer libraryEventsProducer) {
        this.libraryEventsProducer = libraryEventsProducer;
    }

    @PostMapping("/v1/libraryevent")
    public ResponseEntity<LibraryEvent> postLibraryEvent(@RequestBody LibraryEvent libraryevent) throws JsonProcessingException {

        log.info("INFO :: we are in the event controller ");
        log.info("INFO :: library Events : {}", libraryevent);

         // invoke Kafka Producer
        libraryEventsProducer.sendLibraryEvents(libraryevent)

        return ResponseEntity.status(HttpStatus.CREATED).body(libraryevent);
    }
}
```

```
POST WITH-NULL-LIBRARY-EVENT-ID
----------------------
```

```
curl -i \
-d '{"libraryEventId":null,"libraryEventType": "NEW","book":{"bookId":456,"bookName":"Kafka Using Spring Boot","bookAuthorName":"Dilip"}}'
-H "Content-Type: application/json" \
-X POST http://localhost:8085/v1/libraryevent
```



Approach 2

we are using the blocking call to send the messages to kafka

```
we are sending the messahe to kafak and its ablocking call
var results = kafkatemplate.send(topicName, key, message)..get();

we are sending the message to kafka and it waits for 2 MInutes
var results = kafkatemplate.send(topicName, key, message).get(2, TimeUnit.MINUTES);
```

Approach 3

Producer Records

```
public CompletableFuture<SendResult<Integer, String>> sendLibraryEvents_v3(LibraryEvent libraryEvent) throws JsonProcessingException {

        var key = libraryEvent.libraryEventId();
        var message = objectMapper.writeValueAsString(libraryEvent);

//         Create Producer Record
        var appproducerrecord = buildProducerRecord(key,message);


        var results = kafkatemplate.send(appproducerrecord);

        return results.whenComplete( (sendResult, throwable) -> {
            if(throwable != null) {

                handleFailure(key, message, throwable);
            }else{

                handleSucesss(key, message, sendResult);
            }
        });
    }
```

we are going to create header and we are going to add it to the ProducerRecord

```
private ProducerRecord<Integer, String> buildProducerRecord(Integer key, String message) {

        List<Header> headersData = List.of(new RecordHeader("Event-Source", "Scanner".getBytes()));
        return new ProducerRecord<>(topicName, null, key, message, headersData);
    }
```

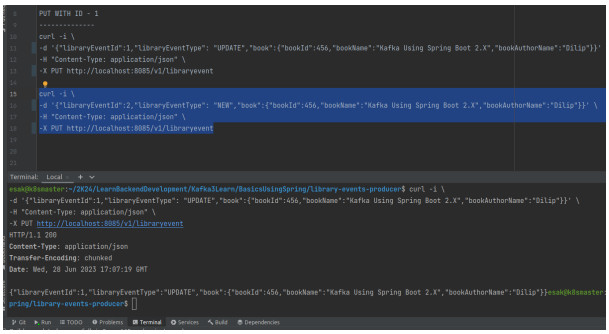
Send Messages with Key

lets create a new Update Endpoint

```
@PutMapping("/v1/libraryevent")
    public ResponseEntity<?> updateLibraryEvent(@RequestBody  LibraryEvent libraryEvent) throws JsonProcessingException {
        log.info("INFO : we are going to update the library Event ");

        if (libraryEvent.libraryEventId() == null ) {
            return ResponseEntity.status(HttpStatus.BAD_REQUEST).body("PASS the library Event ID ");
        }
        if(! libraryEvent.libraryEventType().equals(LibraryEventType.UPDATE)) {
            return ResponseEntity.status(HttpStatus.BAD_REQUEST).body("Only Update Event Type is Supported");
        }
        // Kafka Producer
        libraryEventsProducer.sendLibraryEvents(libraryEvent);

        log.info("INFO :: we have updated the event");
        return ResponseEntity.status(HttpStatus.OK).body(libraryEvent);
    }
}
```

Lets Issue the Curl Command and Verify it

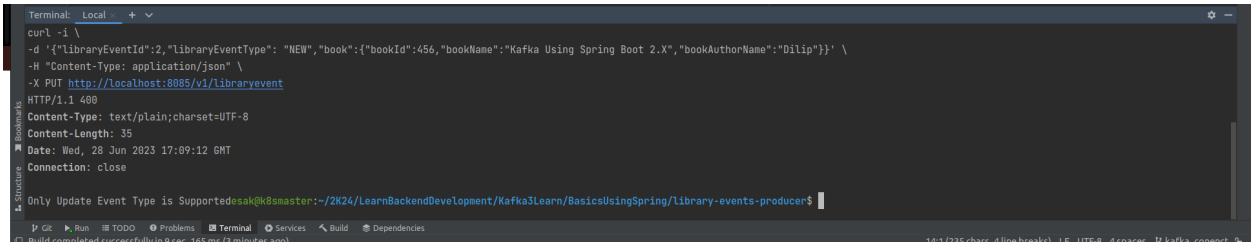
```
curl -i \
-d '{"libraryEventId":1,"libraryEventType": "UPDATE","book":{"bookId":456,"bookName":"Kafka Using Spring Boot 2.X","bookAuthorName":"Dilip"
-H "Content-Type: application/json" \
-X PUT http://localhost:8085/v1/libraryevent
```



Lets the New Curl Command

```
curl -i \
-d '{"libraryEventId":2,"libraryEventType": "NEW","book":{"bookId":456,"bookName":"Kafka Using Spring Boot 2.X","bookAuthorName":"Dilip"}}'
-H "Content-Type: application/json" \
-X PUT http://localhost:8085/v1/libraryevent
```

When we try to issue a NEW library Event we get the Exception as per our Application



Producer Configuration

acks

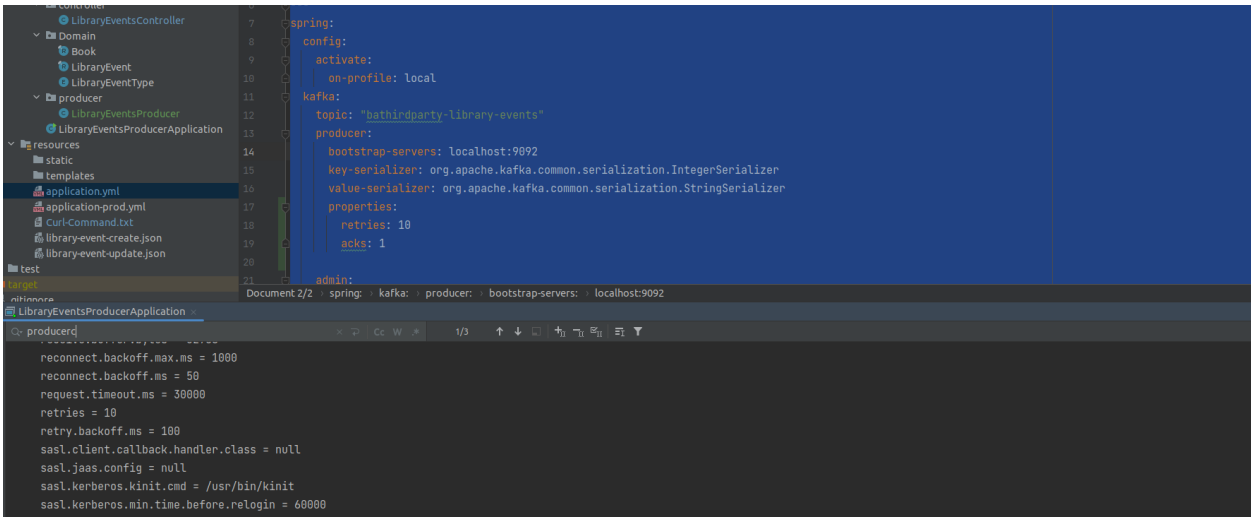
- acks
  - acks = 0, 1 and -1(all)
    - acks = 1 -> guarantees message is written to a leader
    - acks = -1(all) -> guarantees message is written to a leader and to all the replicas (Default)
    - acks=0 -> no guarantee (Not Recommended)

retries

- retries
  - Integer value = [0 - 2147483647]
  - In Spring Kafka, the default value is **-> 2147483647**
- retry.backoff.ms
  - Integer value represented in milliseconds
  - Default value is 100ms

we can override these Properties in the application File

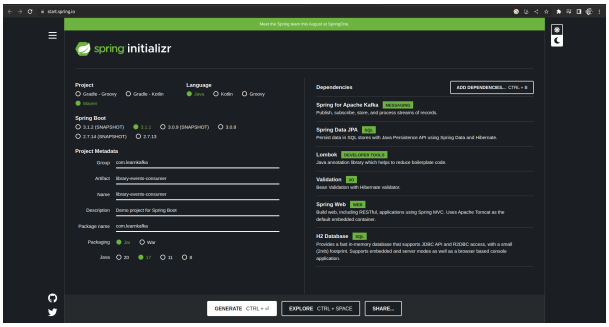
```
server:
  port: 8085
spring:
  profiles:
    active: local
---
spring:
  config:
    activate:
      on-profile: local
  kafka:
    topic: "bathirdparty-library-events"
    producer:
      bootstrap-servers: localhost:9092
      key-serializer: org.apache.kafka.common.serialization.IntegerSerializer
      value-serializer: org.apache.kafka.common.serialization.StringSerializer
      properties:
        retries: 10
        acks: 1

    admin:
      properties:
        bootstrap-servers: localhost:9092
```



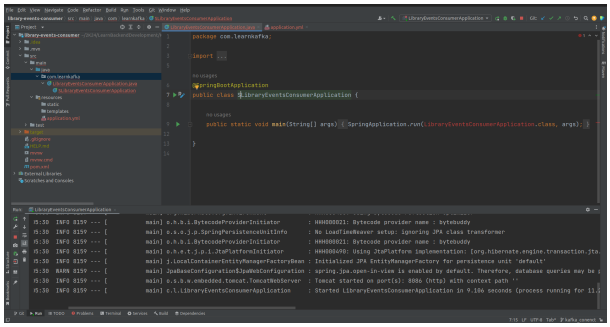
Consumers

Lets Create a Spring Project

Setup the Project



Run the Project



Ways to Consumer Messages

1. **KafkaMessageListenerContainer**
    a. This Was Implemention of MessageListenerContainer
    b. This Class takes care of polling the records and committing the offsets after the records are processed
    c. This Class is Single Threaded
2. **ConcurrentMessageListenerContainer**
    a. This represents multiple KafkaMessageListenerContainer
3. **@listener Annotations**
    a. Simplest way to build the kafka Conusmer

```
@KafkaListener(topics = {"${spring.kafka.topic}"})
public void onMessage(ConsumerRecord<Integer, String> consumerRecord) {
    log.info("OnMessage Record : {} ", consumerRecord);
}
```

b.

- Configuration Sample Code

```
@Configuration
@EnableKafka
@Slf4j
public class LibraryEventsConsumerConfig {
```

Kafka Consumer Mandatory Properties

bootstrap.servers -> is needed too

```
key-deserializer: org.apache.kafka.common.serialization.IntegerDeserializer
value-deserializer: org.apache.kafka.common.serialization.StringDeserializer
group-id: library-events-listener-group
```

Lets Create the application Yaml

```
spring:
  profiles:
    active: local
server:
  port: 8086
---
spring:
  config:
    activate:
      on-profile: local
  kafka:
    consumer:
      bootstrap-servers: localhost:9092
      key-deserializer: org.apache.kafka.common.serialization.IntegerDeserializer
      value-deserializer: org.apache.kafka.common.serialization.StringDeserializer
```

Lets Create the Config Class

```
package com.learnkafka.config;

import org.springframework.context.annotation.Configuration;
import org.springframework.kafka.annotation.EnableKafka;

@Configuration
@EnableKafka
public class LibraryEventsConsumerConfig {
}
```

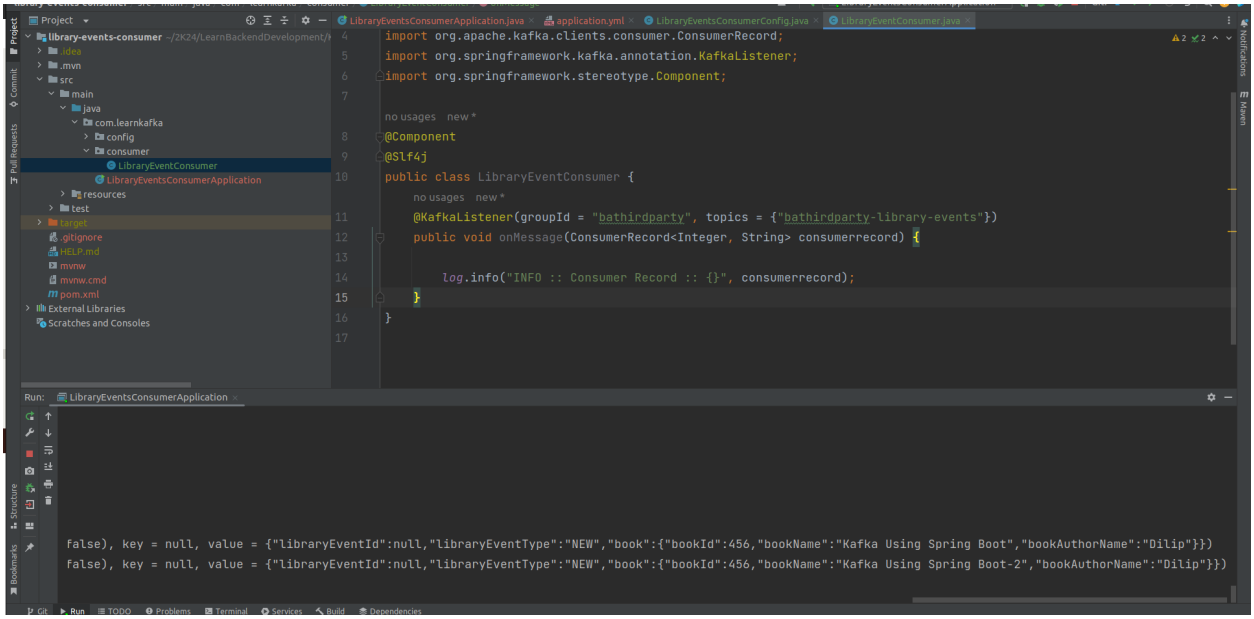
Create a Kafka Consumer

```
@Component
@Slf4j
public class LibraryEventConsumer {

    @KafkaListener()  ==================> Thiis Makes this Method as Consumer
    public void onMessage() {

    }
}

we are having a simple Consumer which retrieves the recrods and prints it
```

Kafka Listener container which is going to pull the records , its going to get multiple records but we are processing the records one by one

in the Application Logs if we see the message as "**2023-07-03T14:09:08.698+05:30 INFO 15032 --- [ntainer#0-0-C-1] o.s.k.l.KafkaMessageListenerContainer : bathirdparty: partitions assigned: [bathirdparty-library-events-0]**" That means we have connected and consumer ready to accept the messages.

Once we Produce the Message we can view the data as below



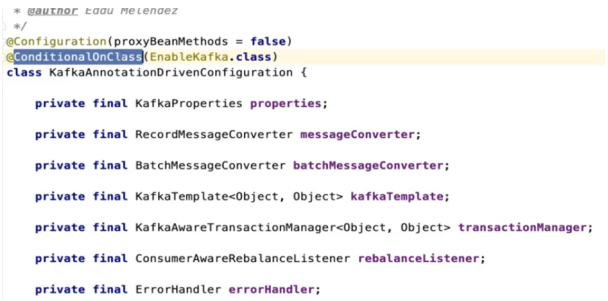
Lets update Consumer Configuration

When we start the Application , kafkaAutoConfiguration Class will be started Automatically , inside that class we have the @EnableConfiguration(KafkaProperties.class) and

Responsible for consumer Configuration KafkaAnnotationDrivenConfiguration.class

If we scroll the Java File we have

- KafkaListnerContainerFactoryConfigurer —> This has the info about the consumer configuration
- kafkaListnerContainerFactory —> By default this was used in the @KafkaListner Annotations
- KafkaConsumerFactory



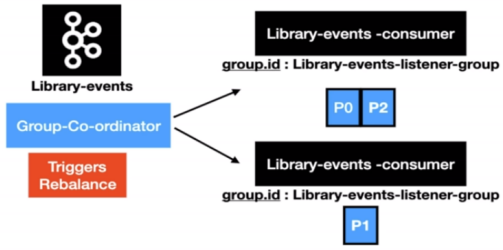
Consumer Groups

Multiple instance of the same application with the same group id. These are the Foundation for scalable message consumption.

Rebalance

Changing the partition ownership from one to another

When One More instance added to the KafkaConsumer Group with the Same Group Id. Group Co-ordinator(Kafka Broker) will receive the Signal from the New consumer and rebalance will be triggered and the partitions will be distributed.



Offset Management

Consumer wont read the record which is already processed

Our Consumer Constantly poll the records. Consumer consumes the Messages and records are processed by the Consumer Application.

Once the records are processed kafka consumer application automatically commits the offsets to *consumser*offsets topics . So once again the Poll loop knows where to pull the data and from what topic id we need to pull the data.

```
The consumer poll() method returns one or more ConsumerRecords. The MessageListener is called for each record. The following lists describe

RECORD: Commit the offset when the listener returns after processing the record.

BATCH: Commit the offset when all the records returned by the poll() have been processed.
```

```
TIME: Commit the offset when all the records returned by the poll() have been processed, as long as the ackTime since the last commit has b

COUNT: Commit the offset when all the records returned by the poll() have been processed, as long as ackCount records have been received si

COUNT_TIME: Similar to TIME and COUNT, but the commit is performed if either condition is true.

MANUAL: The message listener is responsible to acknowledge() the Acknowledgment. After that, the same semantics as BATCH are applied.

MANUAL_IMMEDIATE: Commit the offset immediately when the Acknowledgment.acknowledge() method is called by the listener.
```

Consumer Offset Manually

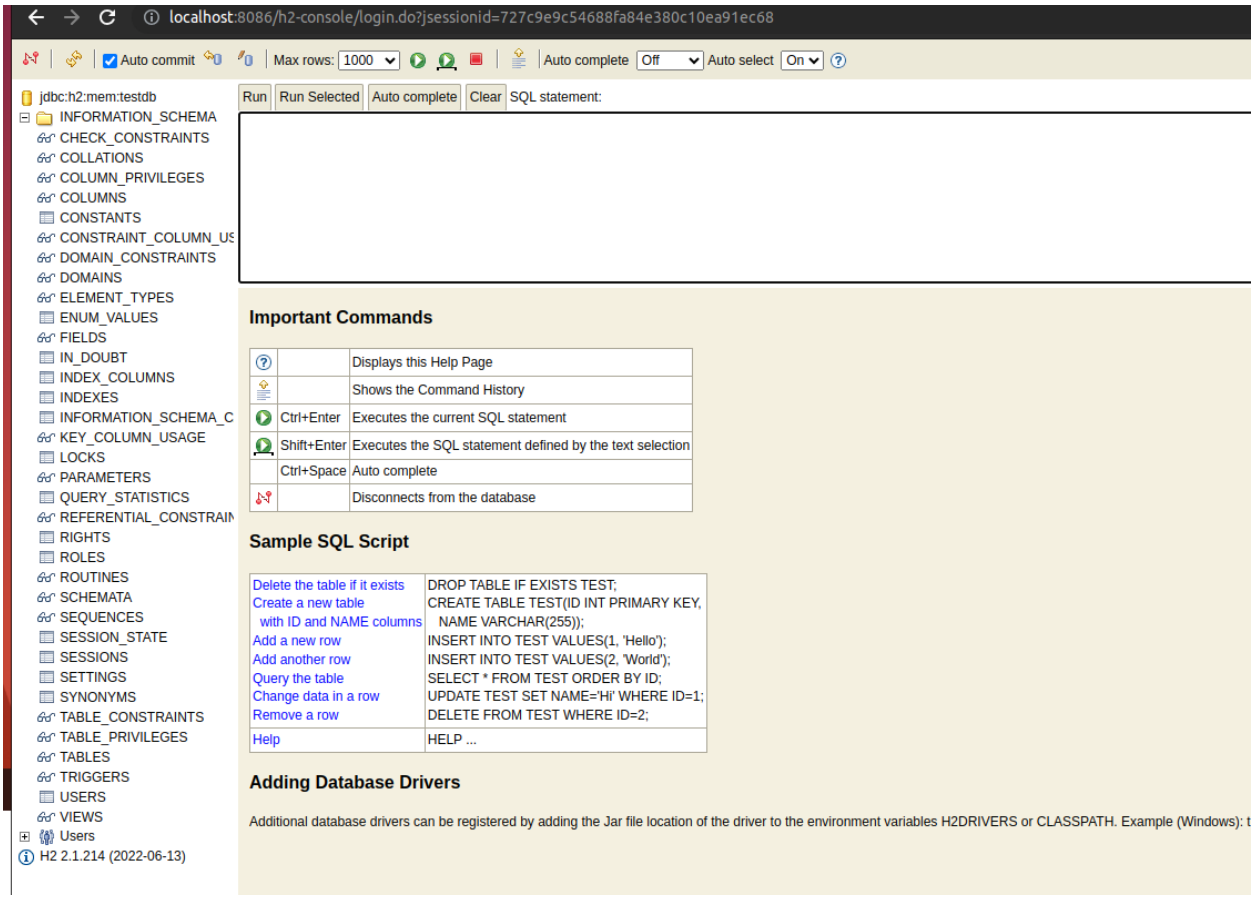
Persist the Records to Database

Step 1 : Add Config File

We are going to persist the records into the H2 InMemory database

```
spring:
  profiles:
    active: local
server:
  port: 8086
---
spring:
  config:
    activate:
      on-profile: local
  kafka:
    consumer:
      bootstrap-servers: localhost:9092
      key-deserializer: org.apache.kafka.common.serialization.IntegerDeserializer
      value-deserializer: org.apache.kafka.common.serialization.StringDeserializer
  datasource:
    url: jdbc:h2:mem:testdb
    driver-class-name: org.h2.Driver
  jpa:
    database: h2
    database-platform: org.hibernate.dialect.H2Dialect
    generate-ddl : true
  h2:
    console:
      enabled: true
```

By adding this database we have enabled the H2 InMemory Database



Step 2 - Build Service Layer

we will create a repository in Spring and using which we can perform CRUD Operation

1. Build the Models

Book

```
package com.learnkafka.Entity;

import jakarta.persistence.Entity;
import jakarta.persistence.Id;
import jakarta.persistence.JoinColumn;
import jakarta.persistence.OneToOne;
import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@AllArgsConstructor
@NoArgsConstructor
@Builder
@Entity
public class Book {
    @Id
    private Integer bookId;
    private String bookName;
    private String bookAuthorName;
```

Failure Record

```
package com.learnkafka.Entity;

import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.Id;
import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;

@AllArgsConstructor
@NoArgsConstructor
@Data
@Builder
@Entity
public class FailureRecord {

    @Id
    @GeneratedValue
    private Integer bookId;
    private String topic;
```

```
    @OneToOne
    @JoinColumn(name="libraryEventId")
    private LibraryEvent libraryEvent;

}
```

```
    private Integer key_value;
    private String errorRecord;
    private Integer partition;
    private Long offset_value;
    private String exception;
    private String status;
}
```

Library Event

```
package com.learnkafka.Entity;

import jakarta.persistence.*;
import lombok.*;

@Data
@AllArgsConstructor
@NoArgsConstructor
@Builder
@Entity
public class LibraryEvent {

    @Id
    @GeneratedValue
    public Integer libraryEventId;
    @Enumerated(EnumType.STRING)
    public LibraryEventType libraryEventType;

    @OneToOne(mappedBy = "libraryEvent", cascade = {CascadeType.ALL})
    @ToString.Exclude
    private Book book ;
}
```

LibraryEvent Type

```
package com.learnkafka.Entity;

public enum LibraryEventType {

    NEW,
    UPDATE
}
```

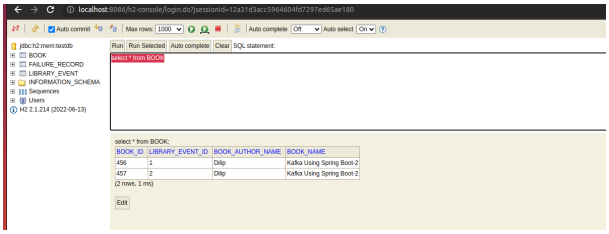
2. Build REpo layers

```
package com.learnkafka.jpa;

import com.learnkafka.Entity.LibraryEvent;
import com.learnkafka.Entity.LibraryEventType;
import org.springframework.data.repository.CrudRepository;

public interface LibraryEventsRepository extends CrudRepository<LibraryEvent, Integer> {
}
```

3. Build Service Layer

```
package com.learnkafka.service;

import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.learnkafka.Entity.LibraryEvent;
import com.learnkafka.jpa.LibraryEventsRepository;
import lombok.extern.slf4j.Slf4j;
import org.apache.kafka.clients.consumer.ConsumerRecord;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

@Service
@Slf4j
public class LibraryEventService {

    @Autowired
    ObjectMapper objectMapper;  ==============> using this objectMapper we are extracting the JSON Values

    @Autowired
```

```
    private LibraryEventsRepository libraryEventsRepository;
    public void ProcessLibraryEvent(ConsumerRecord<Integer, String> consumerRecord) throws JsonProcessingException {
//         Read the LibraryEvent Value from the JSON
        LibraryEvent libraryEvent = objectMapper.readValue(consumerRecord.value(), LibraryEvent.class);
        log.info("INFO --- Extracted Event is {}", libraryEvent );
//        Add the data to the H2

        switch (libraryEvent.getLibraryEventType()) {
            case NEW :
//                Save the Event to Database
                saveEvent(libraryEvent);
                break;
            case UPDATE:
                break;
            default:
                log.info("INFO :: Invalid libraryEvent Type");
        }



    }

    private void saveEvent(LibraryEvent libraryEvent) {
        libraryEvent.getBook().setLibraryEvent(libraryEvent);
        libraryEventsRepository.save(libraryEvent);
        log.info("INFO :: Data stored in to Database sucessfully - {}", libraryEvent);
    }
}
```



If we try to update an Id which is Not Available then error is thrown

```
2023-07-04T08:43:41.198+05:30  INFO 15128 --- [ntainer#0-0-C-1] c.l.consumer.LibraryEventConsumer        : INFO :: Consumer Record :: Consu
2023-07-04T08:43:41.199+05:30  INFO 15128 --- [ntainer#0-0-C-1] c.l.service.LibraryEventService          : INFO --- Extracted Event is Libr
2023-07-04T08:43:41.213+05:30 ERROR 15128 --- [ntainer#0-0-C-1] o.s.kafka.listener.DefaultErrorHandler   : Backoff FixedBackOff{interval=0,

org.springframework.kafka.listener.ListenerExecutionFailedException: Listener method 'public void com.learnkafka.consumer.LibraryEventConsu
  at org.springframework.kafka.listener.KafkaMessageListenerContainer$ListenerConsumer.decorateException(KafkaMessageListenerContainer.java
  at org.springframework.kafka.listener.KafkaMessageListenerContainer$ListenerConsumer.doInvokeOnMessage(KafkaMessageListenerContainer.java
  at org.springframework.kafka.listener.KafkaMessageListenerContainer$ListenerConsumer.invokeOnMessage(KafkaMessageListenerContainer.java:2
  at org.springframework.kafka.listener.KafkaMessageListenerContainer$ListenerConsumer.lambda$doInvokeRecordListener$58(KafkaMessageListene
  at io.micrometer.observation.Observation.lambda$observe$4(Observation.java:544) ~[micrometer-observation-1.11.1.jar:1.11.1]
  at io.micrometer.observation.Observation.observeWithContext(Observation.java:603) ~[micrometer-observation-1.11.1.jar:1.11.1]
  at io.micrometer.observation.Observation.observe(Observation.java:544) ~[micrometer-observation-1.11.1.jar:1.11.1]
  at org.springframework.kafka.listener.KafkaMessageListenerContainer$ListenerConsumer.doInvokeRecordListener(KafkaMessageListenerContainer
  at org.springframework.kafka.listener.KafkaMessageListenerContainer$ListenerConsumer.doInvokeWithRecords(KafkaMessageListenerContainer.ja
  at org.springframework.kafka.listener.KafkaMessageListenerContainer$ListenerConsumer.invokeRecordListener(KafkaMessageListenerContainer.j
  at org.springframework.kafka.listener.KafkaMessageListenerContainer$ListenerConsumer.invokeListener(KafkaMessageListenerContainer.java:22
  at org.springframework.kafka.listener.KafkaMessageListenerContainer$ListenerConsumer.invokeIfHaveRecords(KafkaMessageListenerContainer.ja
  at org.springframework.kafka.listener.KafkaMessageListenerContainer$ListenerConsumer.pollAndInvoke(KafkaMessageListenerContainer.java:151
  at org.springframework.kafka.listener.KafkaMessageListenerContainer$ListenerConsumer.run(KafkaMessageListenerContainer.java:1394) ~[sprin
  at java.base/java.util.concurrent.CompletableFuture$AsyncRun.run(CompletableFuture.java:1804) ~[na:na]
  at java.base/java.lang.Thread.run(Thread.java:833) ~[na:na]
  Suppressed: org.springframework.kafka.listener.ListenerExecutionFailedException: Restored Stack Trace
    at org.springframework.kafka.listener.adapter.MessagingMessageListenerAdapter.invokeHandler(MessagingMessageListenerAdapter.java:391) ~
    at org.springframework.kafka.listener.adapter.RecordMessagingMessageListenerAdapter.onMessage(RecordMessagingMessageListenerAdapter.jav
    at org.springframework.kafka.listener.adapter.RecordMessagingMessageListenerAdapter.onMessage(RecordMessagingMessageListenerAdapter.jav
    at org.springframework.kafka.listener.KafkaMessageListenerContainer$ListenerConsumer.doInvokeOnMessage(KafkaMessageListenerContainer.ja
Caused by: java.lang.IllegalArgumentException: Not a valid libraryeventId
  at com.learnkafka.service.LibraryEventService.validateEvent(LibraryEventService.java:54) ~[classes/:na]
  at com.learnkafka.service.LibraryEventService.ProcessLibraryEvent(LibraryEventService.java:36) ~[classes/:na]
  at com.learnkafka.consumer.LibraryEventConsumer.onMessage(LibraryEventConsumer.java:22) ~[classes/:na]
  at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke0(Native Method) ~[na:na]
```

```
  at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:77) ~[na:na]
  at java.base/jdk.internal.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43) ~[na:na]
  at java.base/java.lang.reflect.Method.invoke(Method.java:568) ~[na:na]
  at org.springframework.messaging.handler.invocation.InvocableHandlerMethod.doInvoke(InvocableHandlerMethod.java:169) ~[spring-messaging-6
  at org.springframework.messaging.handler.invocation.InvocableHandlerMethod.invoke(InvocableHandlerMethod.java:119) ~[spring-messaging-6.0
  at org.springframework.kafka.listener.adapter.HandlerAdapter.invoke(HandlerAdapter.java:56) ~[spring-kafka-3.0.8.jar:3.0.8]
  at org.springframework.kafka.listener.adapter.MessagingMessageListenerAdapter.invokeHandler(MessagingMessageListenerAdapter.java:375) ~[s
  at org.springframework.kafka.listener.adapter.RecordMessagingMessageListenerAdapter.onMessage(RecordMessagingMessageListenerAdapter.java:
  at org.springframework.kafka.listener.adapter.RecordMessagingMessageListenerAdapter.onMessage(RecordMessagingMessageListenerAdapter.java:
  at org.springframework.kafka.listener.KafkaMessageListenerContainer$ListenerConsumer.doInvokeOnMessage(KafkaMessageListenerContainer.java
  ... 14 common frames omitted
```

Error Handling

By default spring kafka consumer retries for 9 times and we are going to override this Configuration

```
Adding a method  which changes the no of times to 2

public DefaultErrorHandler errorHandler () {
        var fixedbackoff = new FixedBackOff(1000L, 2);

        return new DefaultErrorHandler(fixedbackoff);
    }


call this Method inside the Factory
@Bean
    ConcurrentKafkaListenerContainerFactory<?,?> kafkaListenerContainerFactory (
            ConcurrentKafkaListenerContainerFactoryConfigurer configurer,
            ConsumerFactory<Object, Object> kafkaConsumerFactory
    ) {

        ConcurrentKafkaListenerContainerFactory<Object, Object> factory = new ConcurrentKafkaListenerContainerFactory<>();
        configurer.configure(factory, kafkaConsumerFactory);

//         It sets the concurrency to 3 threads
        factory.setConcurrency(3);
//        factory.getContainerProperties().setAckMode(ContainerProperties.AckMode.MANUAL);

//         Adding Error Handler
        factory.setCommonErrorHandler(errorHandler());
        return factory;
    }
```

Retry Listener

We make sure and monitor what happens in each and every retry

```
public DefaultErrorHandler errorHandler () {
        var fixedbackoff = new FixedBackOff(1000L, 2);

        var errorHandler = new DefaultErrorHandler(fixedbackoff);

        // Adding Code for the Retry Listner - What to do
        errorHandler.setRetryListeners((consumerRecord, e, i) -> {
            log.info("Failed Record in Retry Listener exception {} ", e.getMessage());
            log.info("Failed Number of  Retry Listener exception {} ", i);
        });

        return errorHandler;

    }


Every Time when we do the Retry above Code will be executed
```

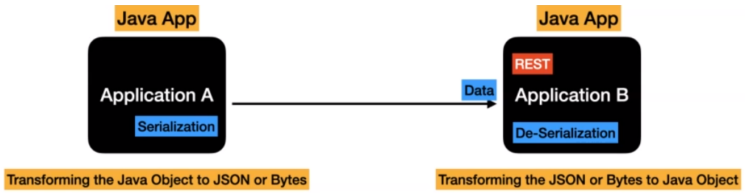
Retry for particular Exception

```
```

Recovery

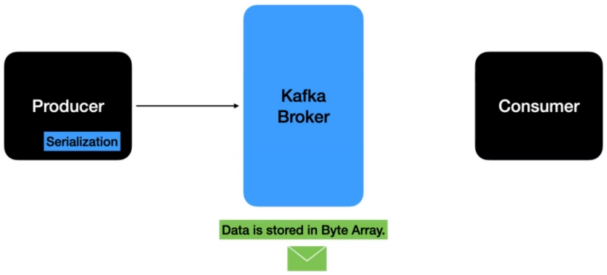
101 - AVRO SCHEMA

Producer and Consumer are decoupled and independent of each other. Producer decides the data contract or structure of the data. Consumers are indirectly coupled with the data format that's been sent by the producer.

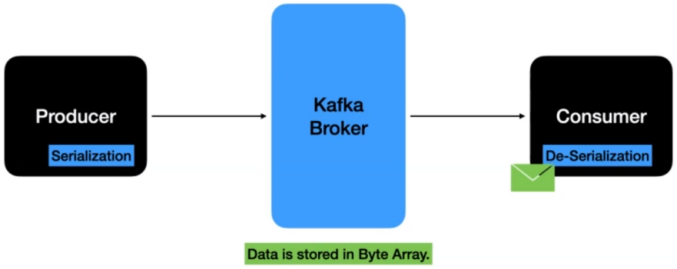


Data serialisation is the process of translating a data structure or object state into a format that can be stored in a file or memory or transmitted and reconstructed later in a different environment.

Producer converts the Java Object into a byte Array via the Serialisation Process and Stored in the Kafka Broker



When Consumer Pulls the data it converts the byte Array into Java Objects via de-serialization



2. Serialization Format

we have two serialization Format

1. Binary Serialization Format
2. plaintext serialization Format

Binary Serialization	Plaintext Serialization
• This serializes the data to byte array • Not Human readable • This is more efficient because the data is compact and less memory overhead • Serialization in generally faster	• This serializes the data into an ecoded text. • Human Readable • Data is verbose and it can be inefficient • Serialization in slower compared to Binary Serialization

Binary Serialization	Plaintext Serialization
• AVRO • ProtocolBuf • Thrift	• JSON • XML

Using Binary Serialization we can define the Schema IDL which basically define a SChema for Data Structure and make sure the data is always aligned to that Schema.

Why Avro for Kafka Data? | Confluent

Confluent is building the foundational platform for data in motion so any organization can innovate and win in a digital-first world.

https://www.confluent.io/blog/avro-kafka-data/

2. Intro To Avro

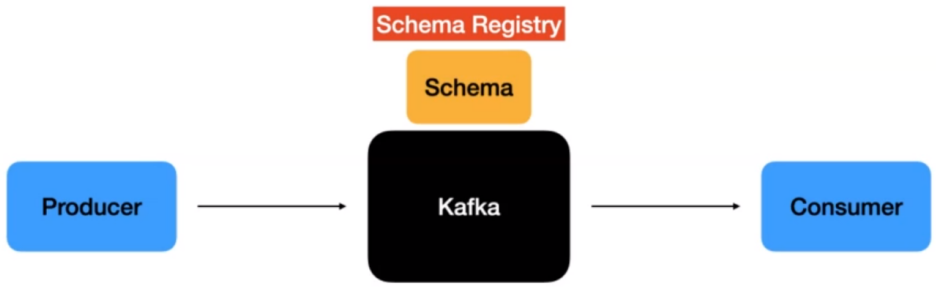
Avro is a data serialization System and it helps to exchange data between two systems using the binary serialization format.

AVRO is a compact and fast binary data format

- it takes up less space
- it has a direct impact on memory and speed of transfer of data
- it has the support of most popular programming languages
- It has the Support for all primitive types and complex Types

- Primitive Types
  - String, bytes, int ,long, float, double, boolean and null
- Complex Types
  - enum
  - arrays
  - maps
  - record - This type is normally used to hold multiple complex types.
  - union - This is used to represent a field can hold multiple types. [String , null]
  - fixed - This is used to represent a field can be of a fixed size, specifying the number of bytes of the value

- Data Owner defines a Schema in JSON format for the data structure that they would want to communicate to the other system



Consumer can read the schema and expects the message to be in this Format

3. Build Simple AVRO Schema

```
Simple greetings.avsc Schema File

{
    "type": "record",
    "name": "greetings",
    "fields": [
        {
            "name": "greetings",
            "type": "string"
        }
    ]
}
```

Inorder to generate avro File from the avro schema Files we need the Following dependency

```
<dependency>
            <groupId>org.apache.avro</groupId>
            <artifactId>avro</artifactId>
            <version>1.10.1</version>
        </dependency>
```

For using Schema registry we need the Following

```
<dependency>
            <groupId>io.confluent</groupId>
            <artifactId>kafka-avro-serializer</artifactId>
            <version>5.5.1</version>
        </dependency>
```

This Kafka-avro is not part of maven central repository we have to download it from the confluent repo

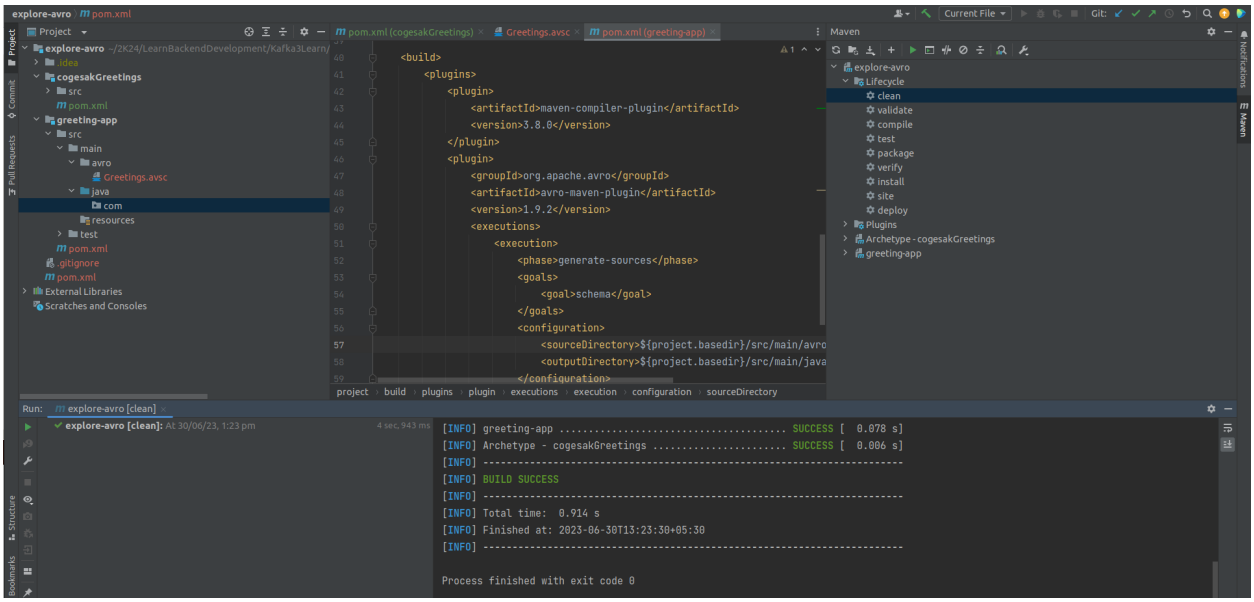
```
<repository>
            <id>confluent</id>
            <name>Confluent</name>
            <url>https://packages.confluent.io/maven/</url>
        </repository>
```

Create the .avsc File

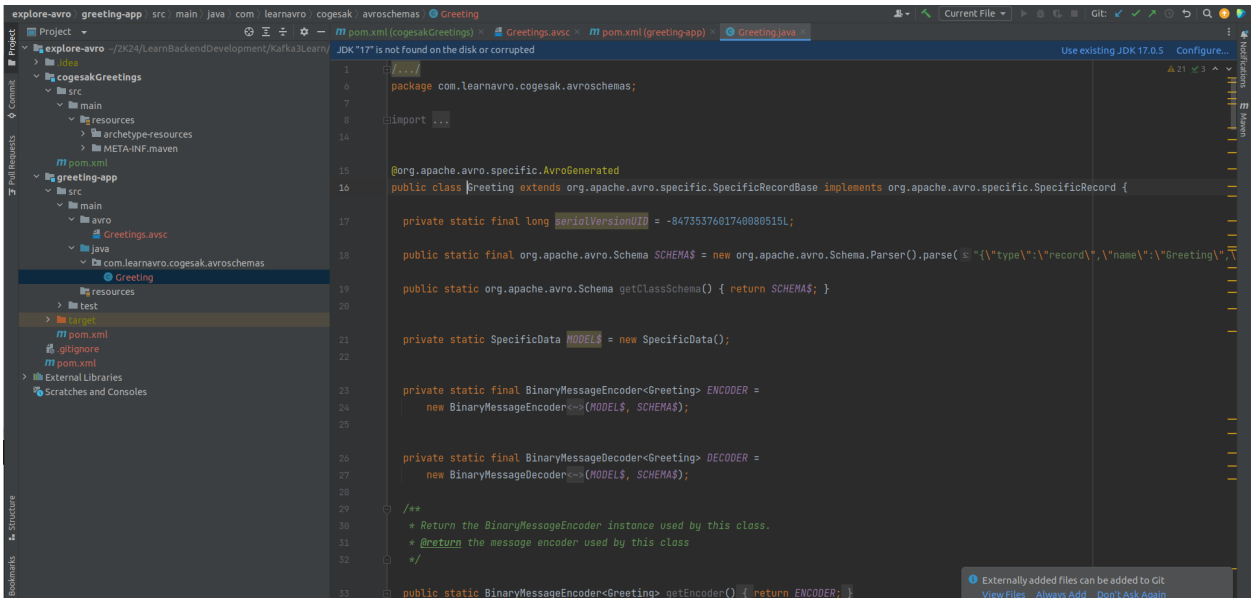
```
{
  "name":"Greeting",
  "namespace": "com.learnavro.cogesak.avroschemas",
  "type": "record",
  "fields": [
    {
      "name":"greeting",
      "type": "string"
    }
  ]
}
```

We are using the Maven Plugin to generate the Avro Files

```
<plugin>
                <groupId>org.apache.avro</groupId>
                <artifactId>avro-maven-plugin</artifactId>
                <version>1.9.2</version>
                <executions>
                    <execution>
                        <phase>generate-sources</phase>
                        <goals>
                            <goal>schema</goal>
                        </goals>
                        <configuration>
                            <sourceDirectory>${project.basedir}/src/main/avro/</sourceDirectory>
                            <outputDirectory>${project.basedir}/src/main/java/</outputDirectory>
                        </configuration>
                    </execution>
                </executions>
            </plugin>
```



Compile - Avro Files has been Generated Automatically



Lets Create a Simple Java Producer

```
package com.learnavro.cogesak.producer;

import com.learnavro.cogesak.avroschemas.Greeting;
import org.apache.kafka.clients.producer.KafkaProducer;
import org.apache.kafka.clients.producer.ProducerConfig;
import org.apache.kafka.clients.producer.ProducerRecord;
import org.apache.kafka.common.serialization.ByteArraySerializer;
import org.apache.kafka.common.serialization.StringSerializer;

import java.io.IOException;
import java.util.Properties;
import java.util.concurrent.ExecutionException;

public class greetingProducer {

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

         final String APP_TOPIC_NAME = "thirdparty_greeting";

        Properties prop = new Properties();
        prop.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:9092");
        prop.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG, StringSerializer.class.getName());
        prop.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG, ByteArraySerializer.class.getName());


        KafkaProducer<String, byte[]> producer = new KafkaProducer<String, byte[]>(prop);

        Greeting greeting = buildGreeting("Hello Fpm thirdparty data hub !! ");
        byte[] result;
        try {
            result = greeting.toByteBuffer().array();
        } catch (IOException e) {
            throw new RuntimeException(e);
        }

        ProducerRecord<String, byte[]> producerrecord = new ProducerRecord<>(APP_TOPIC_NAME, result);

        var appMetaData = producer.send(producerrecord).get();
        System.out.println(appMetaData.toString());
    }
```

```
    private static Greeting buildGreeting(String s) {

        return Greeting.newBuilder()
                .setGreeting(s)
                .build();
    }
}
```

SImple Java Consumer

```
package com.learnavro.cogesak.consumer;

import com.learnavro.cogesak.avroschemas.Greeting;
import org.apache.kafka.clients.consumer.ConsumerConfig;
import org.apache.kafka.clients.consumer.ConsumerRecord;
import org.apache.kafka.clients.consumer.ConsumerRecords;
import org.apache.kafka.clients.consumer.KafkaConsumer;
import org.apache.kafka.clients.producer.ProducerConfig;
import org.apache.kafka.common.serialization.ByteArrayDeserializer;
import org.apache.kafka.common.serialization.ByteArraySerializer;
import org.apache.kafka.common.serialization.StringDeserializer;
import org.apache.kafka.common.serialization.StringSerializer;

import java.io.IOException;
import java.nio.ByteBuffer;
import java.time.Duration;
import java.util.Collections;
import java.util.Properties;

public class greetingsConsumer {
    private static final String APP_TOPIC_NAME = "thirdparty_greeting";

    public static void main(String[] args) {

        Properties prop = new Properties();
        prop.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:9092");
        prop.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG, StringDeserializer.class.getName());
        prop.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG, ByteArrayDeserializer.class.getName());
        prop.put(ConsumerConfig.GROUP_ID_CONFIG, "greeting.Consumer");


        KafkaConsumer<String, byte[]> consumer = new KafkaConsumer<String, byte[]>(prop);
        consumer.subscribe(Collections.singletonList(APP_TOPIC_NAME));

        while (true) {
            ConsumerRecords<String, byte[]> records = consumer.poll(Duration.ofMillis(100));

            for(ConsumerRecord<String, byte[]> rec: records) {
                try {
                    Greeting greeting = decodeAvroGreeting(rec.value());
                    System.out.println(greeting.toString());
                } catch (IOException e) {
                    throw new RuntimeException(e);
                }
            }


        }
    }

    private static Greeting decodeAvroGreeting(byte[] value) throws IOException {
            return Greeting.fromByteBuffer(ByteBuffer.wrap(value));
    }


}
```

Coffee Shop Avro Java Producer

Step 1 - Generate Avro Schemas

Main Schema

```
{
  "type": "record",
  "namespace": "org.coffeeshop.domain.generated",
  "fields": [
    {
      "name": "id",
      "type": "int",
    },
    {
      "name": "name",
      "type": "string"
    }, {
      "name": "nickName",
      "type": "string",
      "default": "",
      "doc": "Optional Field represent the nickname of the user"
    }, {
      "name": "store",
      "type": "Store",
    },
    {
      "name": "orderLineItems",
      "type": {
        "type": "array",
        "items": {
          "name": "orderLineItem",
          "type": "orderLineItem"
        }
      }
    },
   {
      "name": "status",
      "type": "string"
    }
  ]
}
```

Store Schema

```
{
  "name": "Store",
  "type": "record",
  "namespace": "org.coffeeshop.domain.generated",
  "fields": [
    {
      "name": "id",
      "type": "int"
    },
    {
      "name": "address",
      "type": "Address"
    }
  ]
}
```

Address Schema

```
{
"name": "Address",
  "type": "record",
  "namespace": "org.coffeeshop.domain.generated",
  "fields": [
    {
      "name": "addressLine1",
      "type": "string"
    },
    {
      "name": "city",
      "type": "string"
```

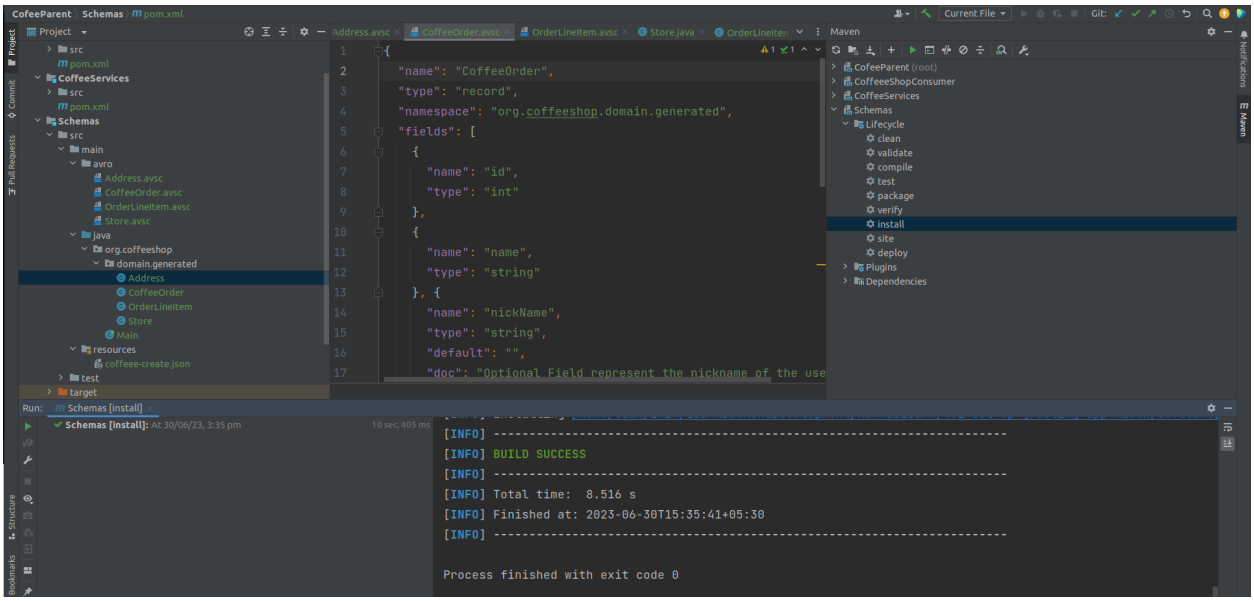
```
    },
    {
      "name": "state_province",
      "type": "string"
    },
    {
      "name": "country",
      "type": "string"
    },
    {
      "name": "zip",
      "type": "string"
    },

  ]
}
```

OrderLIne Schema

```
{
  "name": "OrderLineItem",
  "type": "record",
  "namespace": "org.coffeeshop.domain.generated",
  "fields": [
    {
      "name": "name",
      "type": "string"
    },
    {
      "name": "size",
      "type": {
        "type": "enum",
        "name": "size",
        "symbols": ["SMALL", "MEDIUM", "LARGE", "X-LARGE"]
      }
    },
    {
      "name": "quantity",
      "type": "int"
    }
  ]
}
```

Step 2 - Generate Classes using Maven Tool

```
mvn clean
mvn compile

check the java Classes

update the Configuration to Include other Imports


<configuration>
                            <sourceDirectory>${project.basedir}/src/main/avro/</sourceDirectory>
                            <outputDirectory>${project.basedir}/src/main/java/</outputDirectory>
                            <imports>
                                <import>${project.basedir}/src/main/avro/Store.avsc</import>
                                <import>${project.basedir}/src/main/avro/Address.avsc</import>
                                <import>${project.basedir}/src/main/avro/OrderLineItem.avsc</import>
                            </imports>
                        </configuration>
```



Step 3 - Create the Producer

```
package org.coffeeshop.Producer;

import org.apache.kafka.clients.producer.KafkaProducer;
import org.apache.kafka.clients.producer.ProducerConfig;
import org.apache.kafka.clients.producer.ProducerRecord;
import org.apache.kafka.common.serialization.ByteArraySerializer;
import org.apache.kafka.common.serialization.StringSerializer;
import org.coffeeshop.domain.generated.CoffeeOrder;

import java.io.IOException;
import java.util.Properties;
import java.util.concurrent.ExecutionException;

import static org.coffeeshop.util.CofeeOrderUtil.buildnewCoffeeOrder;

public class coffeeProducer {

    private static final String APP_TOPIC_NAME = "thirdparty_coffee";

    public static void main(String[] args) throws IOException, ExecutionException, InterruptedException {
        Properties prop = new Properties();
        prop.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:9092");
        prop.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG, StringSerializer.class.getName());
        prop.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG, ByteArraySerializer.class.getName());

        KafkaProducer<String, byte[]> producer = new KafkaProducer<String, byte[]>(prop);

        CoffeeOrder coffeeorder = buildnewCoffeeOrder();
        System.out.println(coffeeorder.toString());

        byte[] result = coffeeorder.toByteBuffer().array();

        ProducerRecord<String, byte[]> producerrecord = new ProducerRecord<String, byte[]> (APP_TOPIC_NAME, result);
        var appmetadata = producer.send(producerrecord).get();

        System.out.println("Metadata is " + appmetadata.toString());



    }

}
```

Util Class

```
package org.coffeeshop.util;

import org.coffeeshop.domain.generated.*;

import java.util.List;
import java.util.Random;

public class CofeeOrderUtil {

    public static int randomId(){
        Random random = new Random();
        return random.nextInt(1000);
    }
    private static Address buildAddress() {

        return Address.newBuilder()
                .setAddressLine1("1234 Address Line 1")
                .setCity("Chicago")
                .setStateProvince("IL")
                .setZip("12345")
                .setCountry("INDIA")
                .build();

    }

    private static Store generateStore(){

        return  Store.newBuilder()
                .setId(randomId())
                .setAddress(buildAddress())
                .build();
    }

    private static List<OrderLineItem> generateOrderLineItems() {

        var orderLineItem = OrderLineItem.newBuilder()
                .setName("Caffe Latte")
                .setQuantity(1)
                .setSize(Size.MEDIUM)
                .build();

        return List.of(orderLineItem);

    }


    public static CoffeeOrder buildnewCoffeeOrder() {
        return CoffeeOrder.newBuilder().
                setId(randomId()).
                setName("Esakki Thangappapillai").
                setNickName("Esak")
                .setStore(generateStore())
                .setOrderLineItems(generateOrderLineItems())
                .setStatus("NEW")
                .build();
    }
}
```

Consumer Code

```
package org.coffeeshop;

import com.fasterxml.jackson.databind.cfg.ConfigFeature;
import org.apache.kafka.clients.consumer.ConsumerConfig;
import org.apache.kafka.clients.consumer.ConsumerRecord;
import org.apache.kafka.clients.consumer.ConsumerRecords;
import org.apache.kafka.clients.consumer.KafkaConsumer;
import org.apache.kafka.clients.producer.ProducerConfig;
import org.apache.kafka.common.serialization.ByteArrayDeserializer;
import org.apache.kafka.common.serialization.StringDeserializer;
import org.coffeeshop.domain.generated.CoffeeOrder;
```

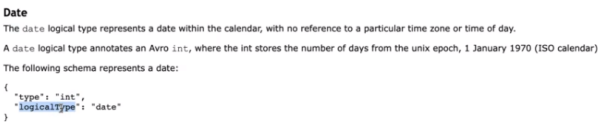
```
import java.io.IOException;
import java.nio.ByteBuffer;
import java.time.Duration;
import java.util.Collections;
import java.util.Properties;

public class coffeeConsumer {
    public static final String APP_TOPIC_NAME = "thirdparty_coffee";

    public static void main(String[] args) {

        Properties prop = new Properties();
        prop.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:9092");
        prop.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG, StringDeserializer.class.getName());
        prop.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG, ByteArrayDeserializer.class.getName());
        prop.put(ConsumerConfig.GROUP_ID_CONFIG, "greeting.Consumer");
        prop.put(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG,  "earliest");

        KafkaConsumer<String, byte[]> consumer = new KafkaConsumer<String, byte[]>(prop);
        consumer.subscribe(Collections.singletonList(APP_TOPIC_NAME));


        while (true) {
            ConsumerRecords<String, byte[]> records = consumer.poll(Duration.ofMillis(100));

            for(ConsumerRecord<String, byte[]> rec: records) {
                try {
                    CoffeeOrder coffee = decodeAvroGreeting(rec.value());
                    System.out.println(coffee.toString());
                } catch (IOException e) {
                    throw new RuntimeException(e);
                }
            }


        }

    }

    private static CoffeeOrder decodeAvroGreeting(byte[] value) throws IOException {
        return CoffeeOrder.fromByteBuffer(ByteBuffer.wrap(value));
    }


}
```

#. More Avro Types

Logical Types are used to represent Decimals , UUID , Date, Timestamp , Time

AVRO uses additional Attributes to represent a particular logical type

Here we represent decimal data type



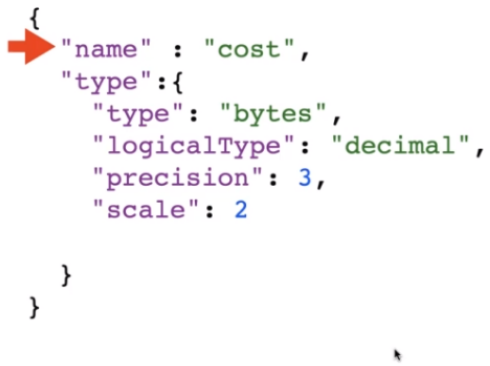
Date

The `date` logical type represents a date within the calendar, with no reference to a particular time zone or time of day.

A `date` logical type annotates an Avro `int`, where the int stores the number of days from the unix epoch, 1 January 1970 (ISO calendar)

The following schema represents a date:

```
{
  "type": "int",
  "logicalType": "date"
}
```



Lets Add a Timestamp to the Coffee Order

```
{
  "name": "CoffeeOrder",
  "type": "record",
  "namespace": "org.coffeeshop.domain.generated",
  "fields": [
    {
      "name": "id",
      "type": "int"
    },
    {
      "name": "name",
      "type": "string"
    }, {
      "name": "nickName",
      "type": "string",
      "default": "",
      "doc": "Optional Field represent the nickname of the user"
    }, {
      "name": "store",
      "type": "Store"
    },
    {
      "name": "orderLineItems",
      "type": {
        "type": "array",
        "items": {
          "name": "orderLineItem",
          "type": "OrderLineItem"
        }
      }
    },
    {
      "name": "ordered_time",
      "type": {
        "type": "long",
        "logicalType": "timestamp-millis"
      }
    },
   {
      "name": "status",
      "type": "string"
    }
  ]
}
```

Adding UUID

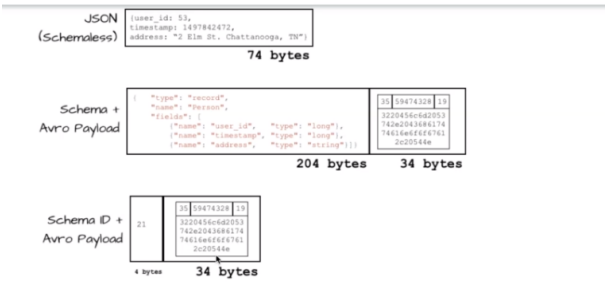
lets update the Schema File

```
Lets Add the build File

<enableDecimalLogicalType>true</enableDecimalLogicalType>
<customConversions>org.apache.avro.Conversion$UUIDConversion</customConversions>


Update the avsc File

"fields": [
    {
      "name": "id",
      "type": {
        "type": "string",
        "logicalType": "uuid"
      }
    },


Update the Fields as Date

{
      "name": "ordered_data",
      "type": {
        "type": "int",
        "logicalType": "date"
      }
    },
```

In Avro Messages we are sending the Schema and data For Every Event we have to send schema + data

1. This pattern Makes message bulkier
2. It also leads to inefficient network bandwidth and adds storage overhead

To overcome this we use the Schema Registry



Schemas, Contracts, and Compatibility | Confluent

This blog post explores the similarities between schemas and APIs, and the importance of being able to modify schemas without the risk of breaking consumer

https://www.confluent.io/blog/schemas-contracts-compatibility/?_ga=2.205203091.2005966880.1650190514-1685861233.1648224453&_gac=1.154929994.1648739558.CjwKCAjwopWSBhB6EiwAjxmqDdc7q5-nqdT-Zx3DI64gxdYDjq4-Ile0txJr4rgDFYY4HAytwrpGZRoCNgUQAvD_BwE

Schema Registry

its a Product from confluent

It serves two purposes

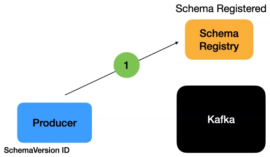
1. Enforcing Data Contracts
2. Handling Schema Evolution

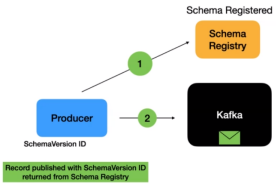
Before Producer records published into kafka , a call will be made to the schema

Then Message will be send to Kafka Topic

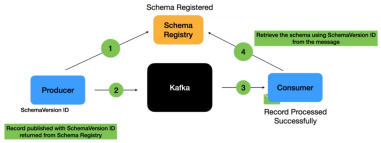
When Consumer Polls for the data , First it calls to the schema registry using the

registry and schema we are trying to publish will be registered first. Schema registry sends the version id as resposne





schema id in the Message from kafka. It will be verified against the message and the message will be processed.

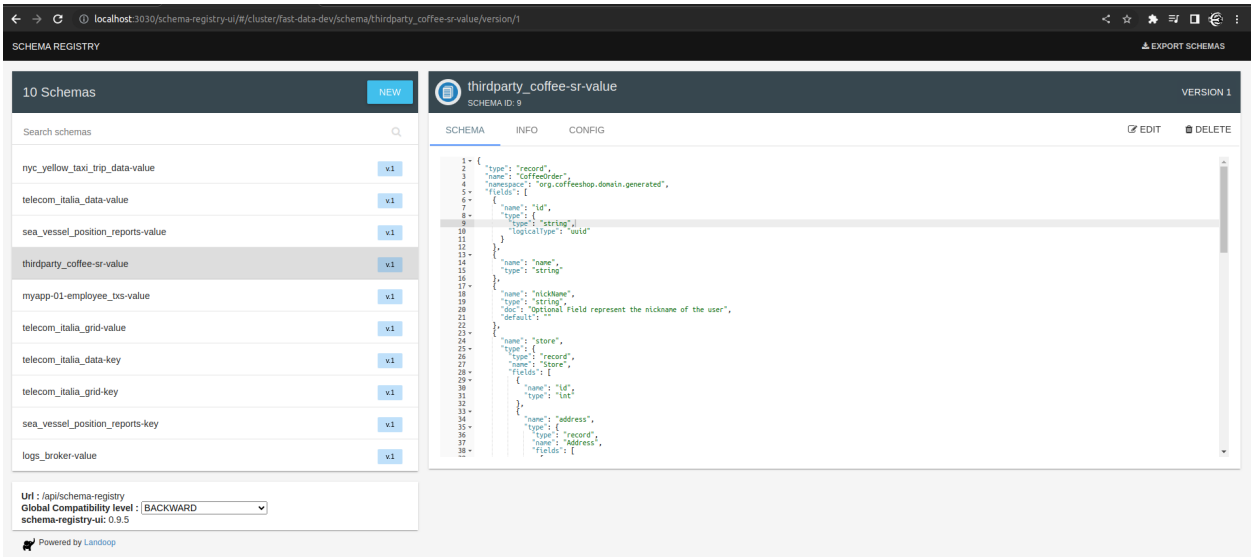


2. Lets Build the Producer

```
KafkaAvroSerializer --> will interact with the schema REgistry and get the SchemaID . we dont need to add any code changes
```

This Will Create the Schema in the Schema Registry

```
package org.coffeeshop.Producer;

import io.confluent.kafka.serializers.KafkaAvroSerializer;
import io.confluent.kafka.serializers.KafkaAvroSerializerConfig;
import org.apache.kafka.clients.producer.KafkaProducer;
import org.apache.kafka.clients.producer.ProducerConfig;
import org.apache.kafka.clients.producer.ProducerRecord;
import org.apache.kafka.common.serialization.ByteArraySerializer;
import org.apache.kafka.common.serialization.StringSerializer;
import org.coffeeshop.domain.generated.CoffeeOrder;

import java.io.IOException;
import java.util.Properties;
import java.util.concurrent.ExecutionException;

import static org.coffeeshop.util.CofeeOrderUtil.buildnewCoffeeOrder;

public class coffeeProducerSchemaRegistry {

    private static final String APP_TOPIC_NAME = "thirdparty_coffee-sr";

    public static void main(String[] args) throws IOException, ExecutionException, InterruptedException {
        Properties prop = new Properties();
        prop.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:9092");
        prop.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG, StringSerializer.class.getName());
        prop.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG, KafkaAvroSerializer.class.getName());
        prop.put(KafkaAvroSerializerConfig.SCHEMA_REGISTRY_URL_CONFIG, "http://localhost:8081");

        KafkaProducer<String, CoffeeOrder> producer = new KafkaProducer<String, CoffeeOrder>(prop);

        CoffeeOrder coffeeorder = buildnewCoffeeOrder();
        System.out.println(coffeeorder.toString());

//        byte[] result = coffeeorder.toByteBuffer().array();

        ProducerRecord<String, CoffeeOrder> producerrecord = new ProducerRecord<String, CoffeeOrder> (APP_TOPIC_NAME, coffeeorder);
        var appmetadata = producer.send(producerrecord).get();

        System.out.println("Metadata is " + appmetadata.toString());



    }

}
```



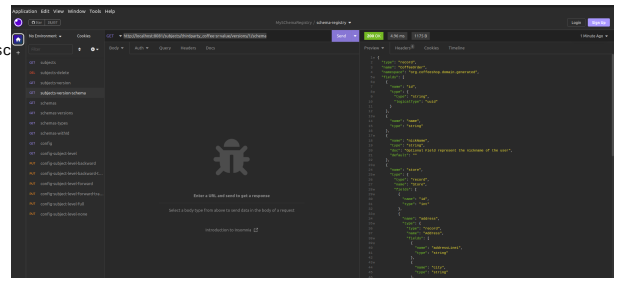
Consumer Code

```
package org.coffeeshop;

import io.confluent.kafka.serializers.KafkaAvroDeserializer;
import io.confluent.kafka.serializers.KafkaAvroDeserializerConfig;
import io.confluent.kafka.serializers.KafkaAvroSerializer;
import org.apache.kafka.clients.consumer.ConsumerConfig;
import org.apache.kafka.clients.consumer.ConsumerRecord;
import org.apache.kafka.clients.consumer.ConsumerRecords;
import org.apache.kafka.clients.consumer.KafkaConsumer;
import org.apache.kafka.clients.producer.ProducerConfig;
import org.apache.kafka.common.serialization.ByteArrayDeserializer;
import org.apache.kafka.common.serialization.StringDeserializer;
import org.coffeeshop.domain.generated.CoffeeOrder;

import java.io.IOException;
import java.nio.ByteBuffer;
import java.time.Duration;
import java.util.Collections;
import java.util.Properties;

public class coffeeConsumerSchemaRegistry {
    public static final String APP_TOPIC_NAME = "thirdparty_coffee-sr";

    public static void main(String[] args) {

        Properties prop = new Properties();
        prop.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:9092");
        prop.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG, StringDeserializer.class.getName());
        prop.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG, KafkaAvroDeserializer.class.getName());
        prop.put(ConsumerConfig.GROUP_ID_CONFIG, "greeting.Consumer");
        prop.put(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG,  "earliest");
        prop.put(KafkaAvroDeserializerConfig.SCHEMA_REGISTRY_URL_CONFIG,  "http://localhost:8081");
        prop.put(KafkaAvroDeserializerConfig.SPECIFIC_AVRO_READER_CONFIG,  true);


        KafkaConsumer<String, CoffeeOrder> consumer = new KafkaConsumer<String, CoffeeOrder>(prop);
        consumer.subscribe(Collections.singletonList(APP_TOPIC_NAME));


        while (true) {
            ConsumerRecords<String, CoffeeOrder> records = consumer.poll(Duration.ofMillis(100));

            for(ConsumerRecord<String, CoffeeOrder> rec: records) {
                try {
//                    CoffeeOrder coffee = decodeAvroGreeting(rec.value());
                    System.out.println(rec.toString());
                } catch (Exception e) {
```

```
                    throw new RuntimeException(e);
                }
            }


        }

    }

    private static CoffeeOrder decodeAvroGreeting(byte[] value) throws IOException {
        return CoffeeOrder.fromByteBuffer(ByteBuffer.wrap(value));
    }


}
```

we can use rest client to Interact with the Schema registry

```
http://localhost:8081/schemas
http://localhost:8081/subjects/thirdparty_coffee-sr-value/versions/1/sc
```



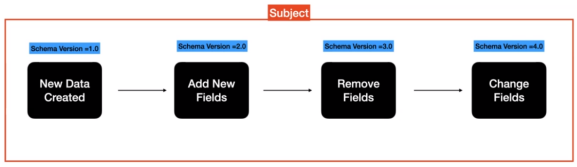
Data Evolution using Schema Registry

Data will continuously evolve with changing business requirements. When the data is Evolved which means

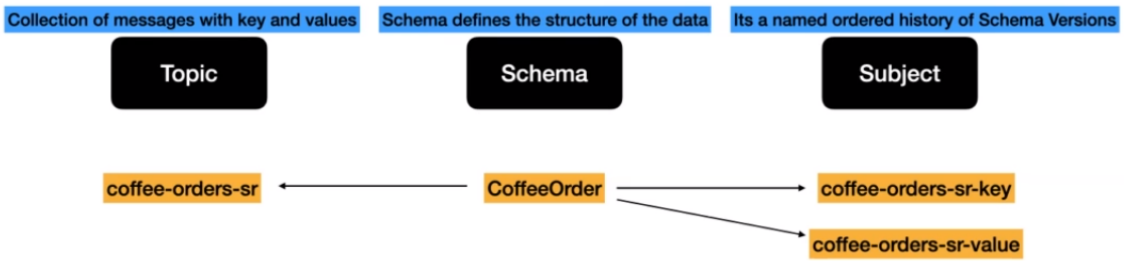
- we may remove Fields
- we may meed to add new Fields
- We may change Fields data type or Field Name

When we update the schema schema continuous to Change.

Anytime data/schema evolves the downstream Consumers should be able to handle the old and new schema seamlessly.



Actual Strategy to define the subject name was using this technique , which is {topic_name}-key and {topic_name}-value



Different Subject Naming Strategies

TopicNameStrategy

- Use it when single type of message should be published in a topic
- This means that the schemas of all messages in the topic must be compatible with each other.
- Subject name is derived using this technique:
  - {topic-name}-key

RecordNameStrategy

- Use it when you have use case where multiple types of related events can be published into the topic and the ordering of events needs to be maintained.
- Thus, the schema registry checks the compatibility for a particular record type, regardless of topic.
- Subject name is derived from the fully qualified record name.
  - **com.learnavro.domain.generated.CoffeeOrder**
- **Usecase**:
  - CoffeeOrder and the related update events

TopicRecordNameStrategy

- Use it when you have use case where multiple types of related events can be published into the topic.
- Schema registry checks compatibility to the **current topic** only
- Subject name is derived from the topic and fully qualified record name.