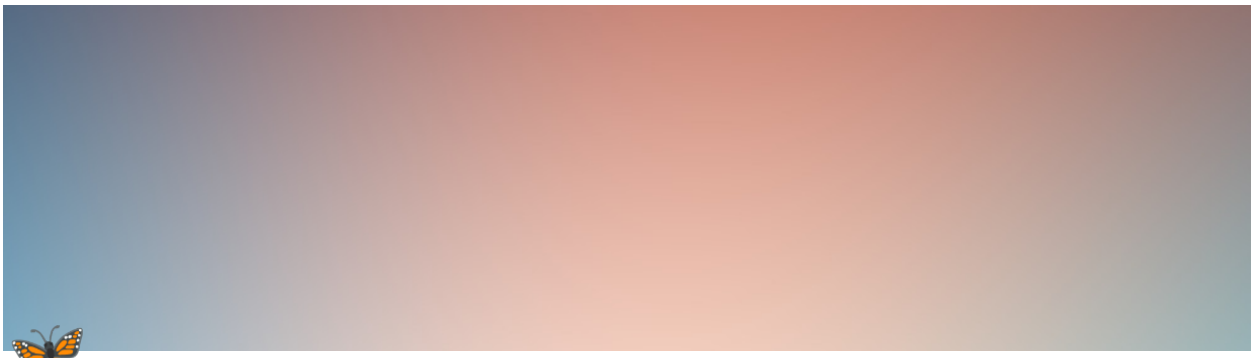


101 - Kafka

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

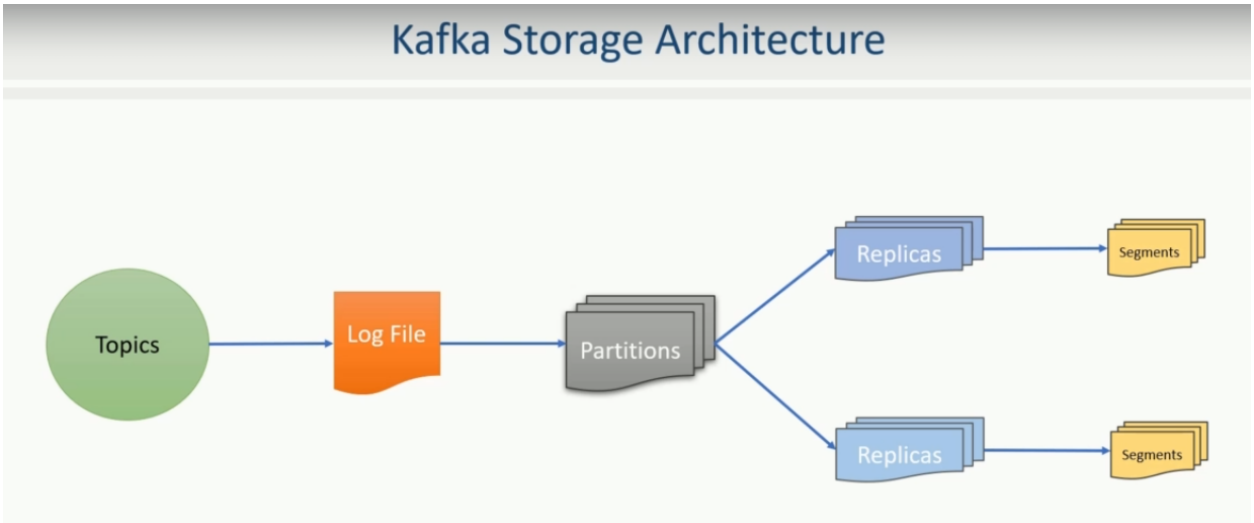
Apache Kafka is a Message Broker

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.**

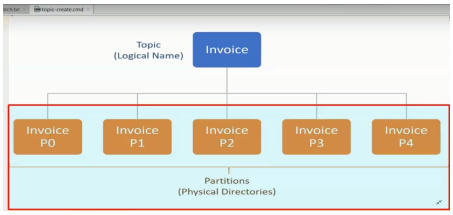
3. Kafka Storage Architecture



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

Replication Factor

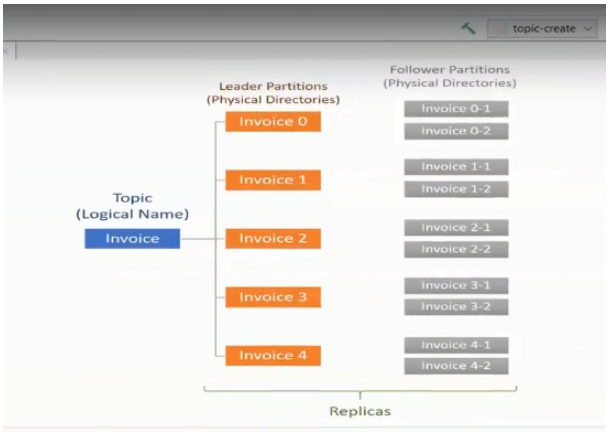
💡 **How many copies we want to maintain for each partition ———> Number of Replicas = partitions * Replication**

```
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

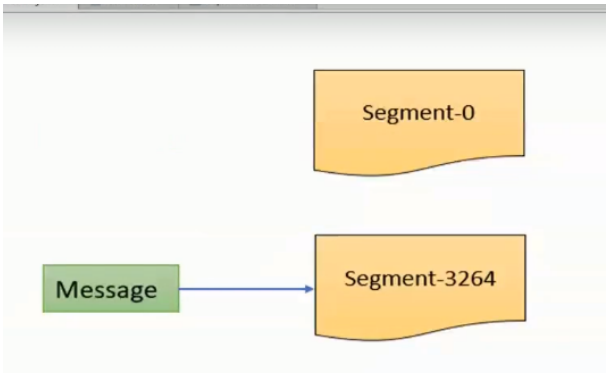
04. Log Segments

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

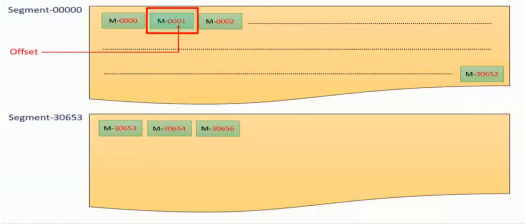
Logic to split the Segment Files

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

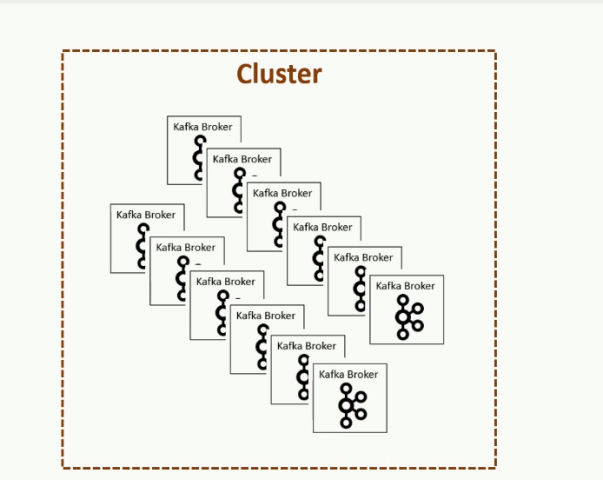
4. Kafka Cluster Architecture

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

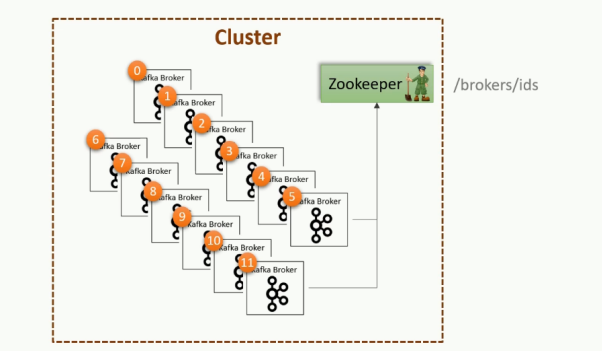
In a typical cluster we have a master node to maintain the list of active cluster members. Master always knows the state of other members.



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

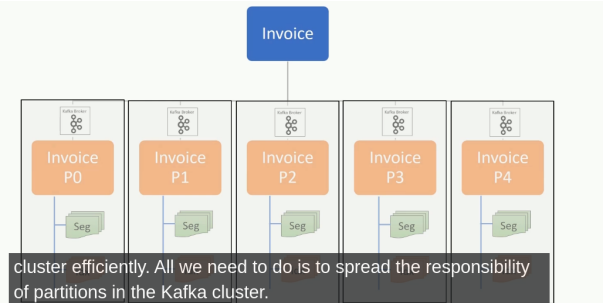
Above activities are maintained by a controller. One of the kafka broker is elected as the Controller to pick some extra responsibilities . If we have a single node cluster it will act as a broker as well as the controller.

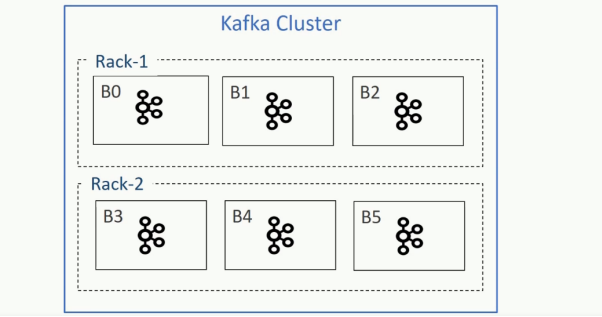
> Only one controller in the cluster at any point in Time

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

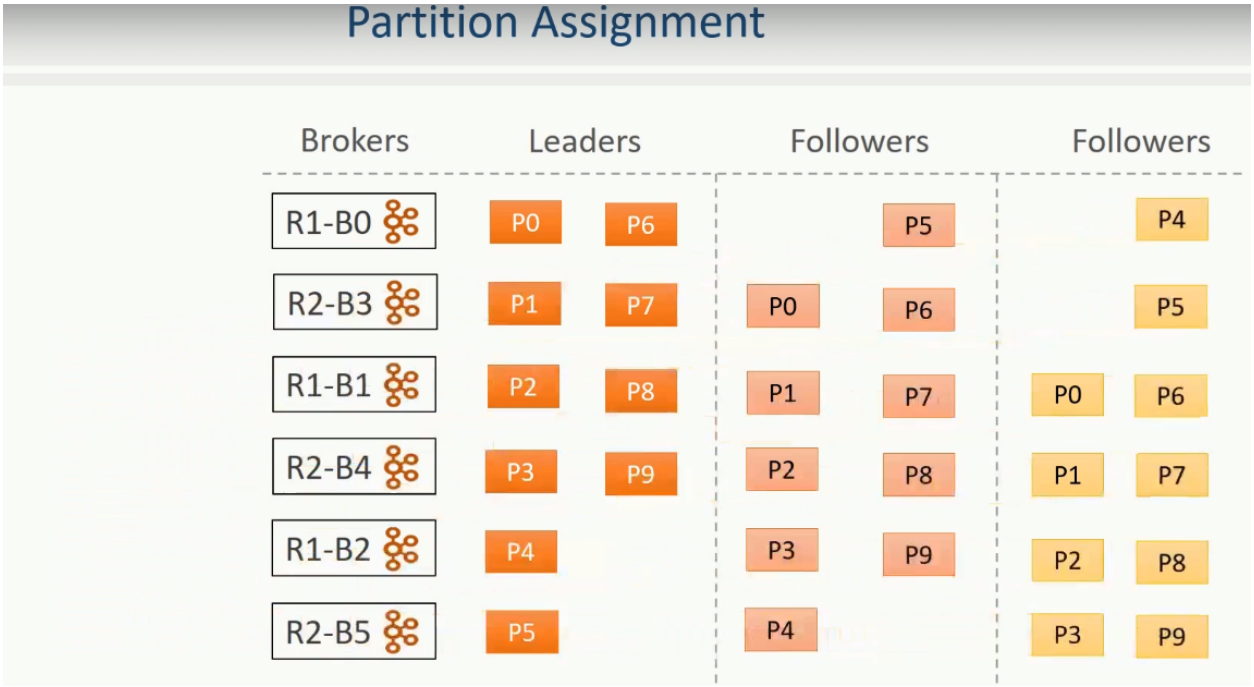
1) Event Distribution ⇒ Partitions are distributed evenly as much as possible to achieve work load balance

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

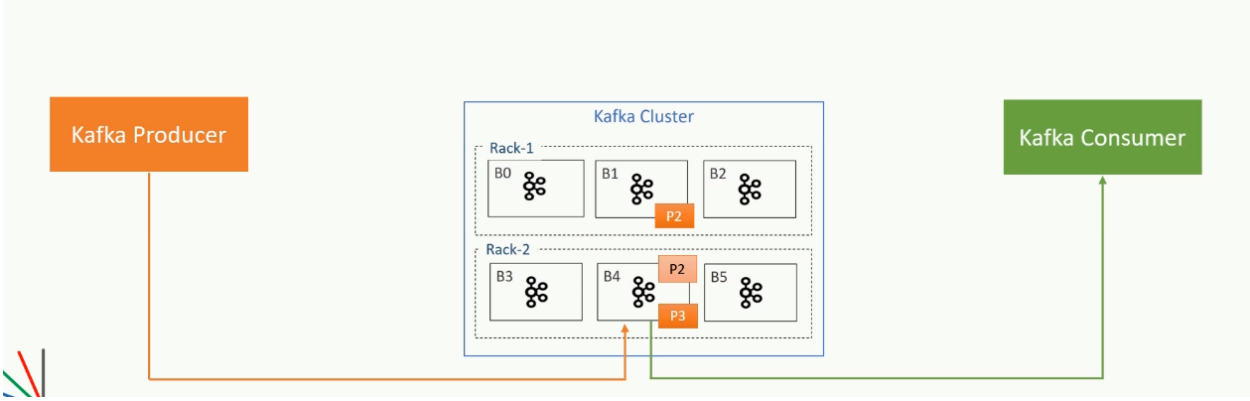
Broker Manages two Types of partitions

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.

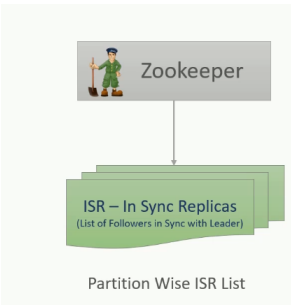
Follower Partition is responsible for copy the messages from the leader to stay update with all the messages. followers will stay sync with the leader .Follower will ask for messages to the leader, this will keep on going in the infinite loop. When the leader dies it may became the leader also it will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

ISR LIST

Some times followers are failed to in sync with Leader

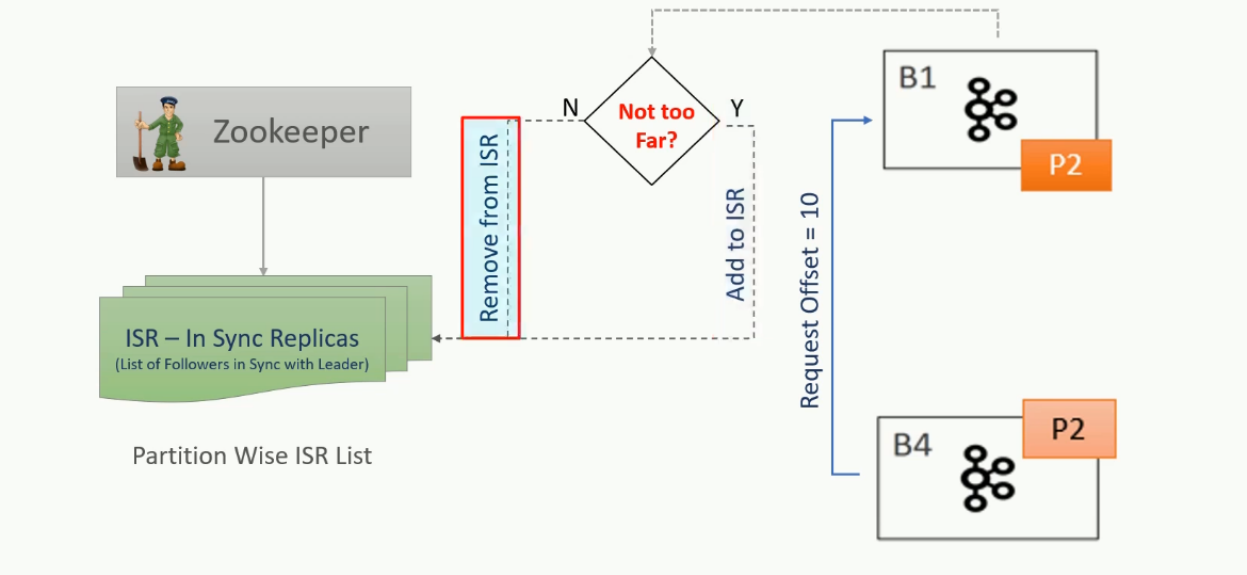
1. Network Congestion ⇒ Due to slow network follower can fall behind
2. Follower Broker Crash/Restart ⇒ since the broker dies all the replicas are fall behind

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



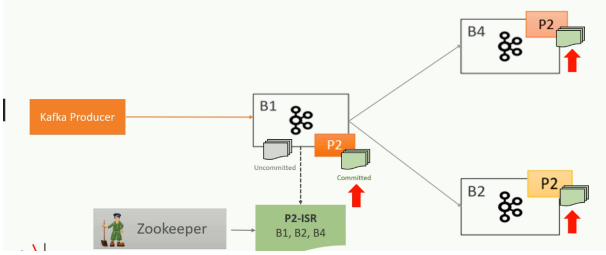
Committed Vs Un-Committed Message

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 .

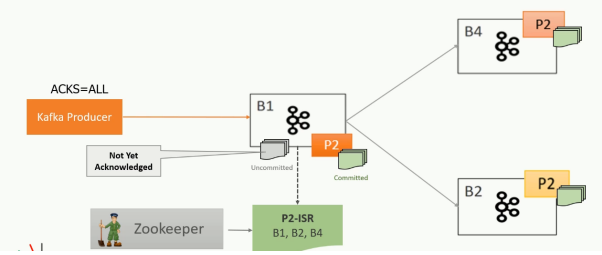
If we loose the Leader Now

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

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

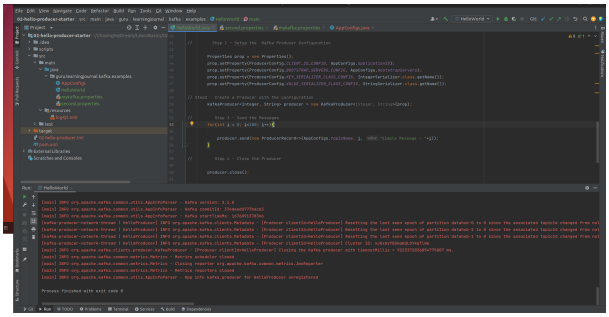
```
esak@esak-PC:~/kafka/bin$ kafka-topics.sh --create  --topic datahub --partitions 3 --bootstrap-server localhost:9092
Created topic datahub.
```

Create a New Producer

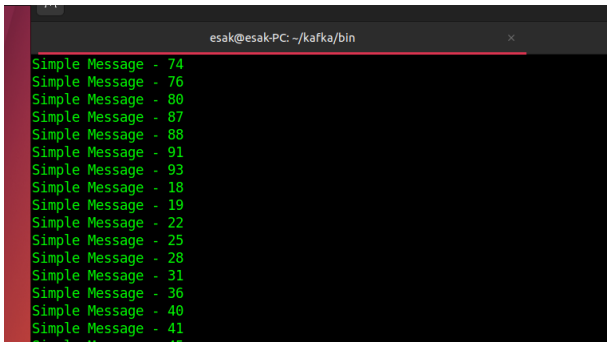
```
package guru.learningjournal.kafka.examples;

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



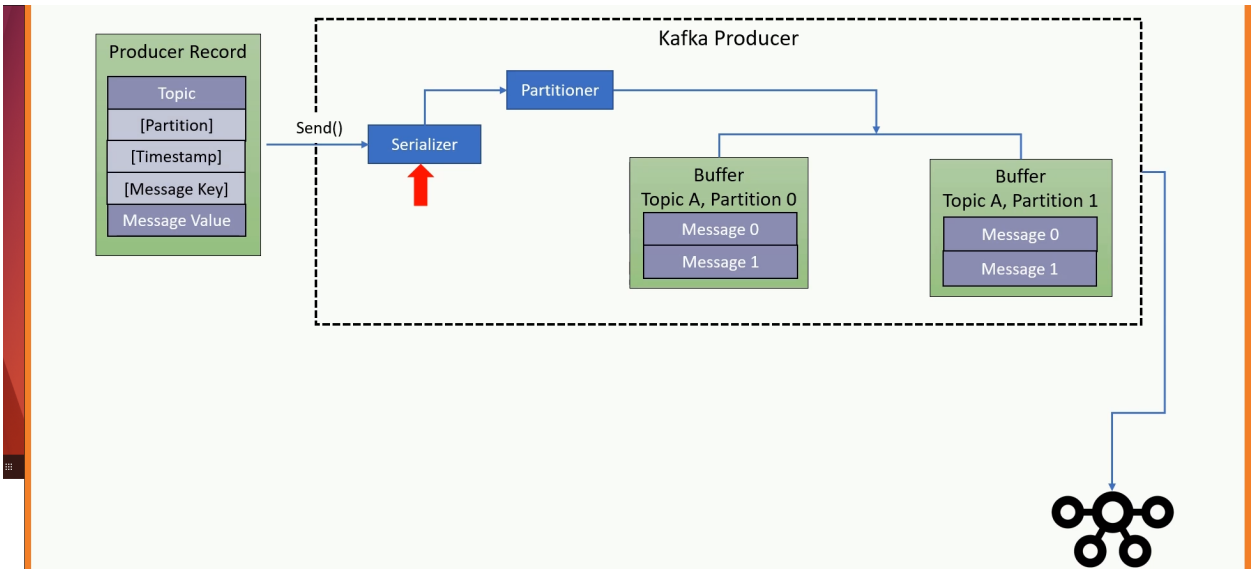
Producer Records

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
```

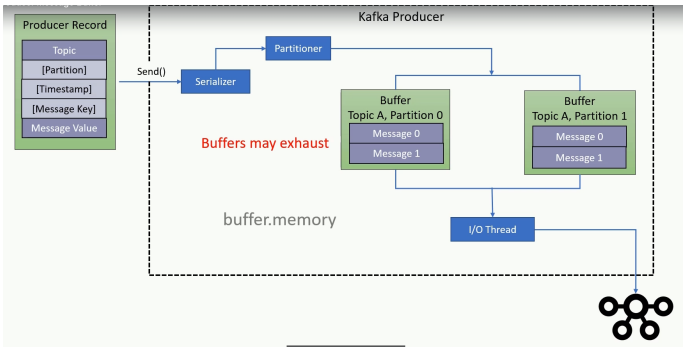
Message Buffer

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.

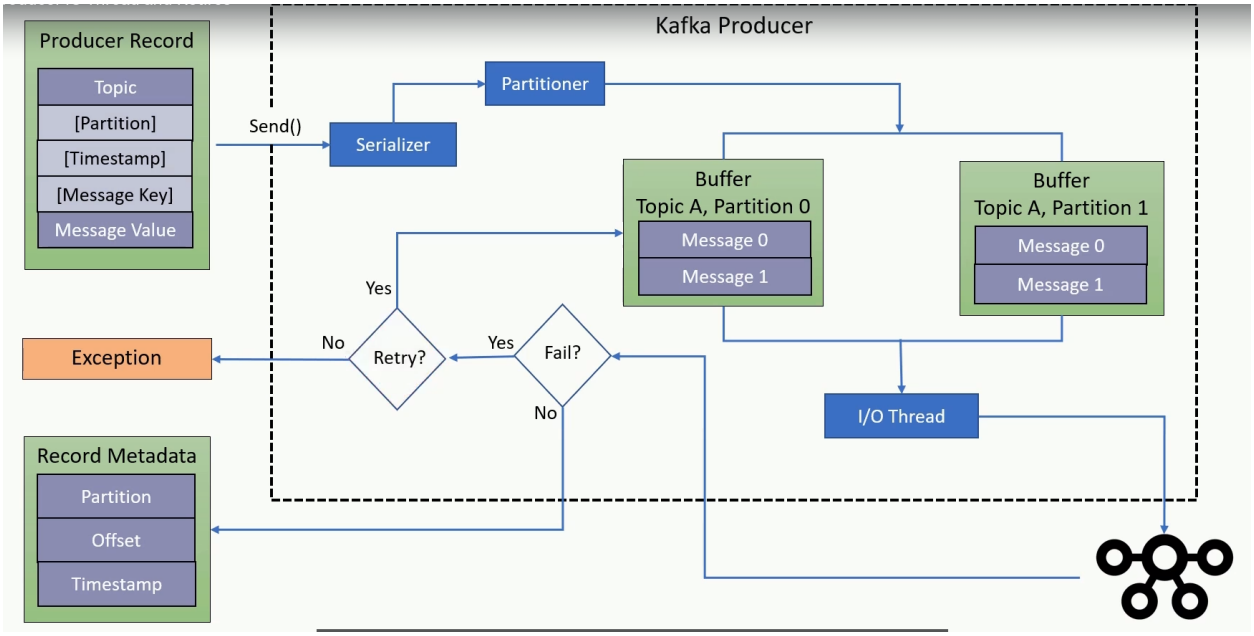



```
we can increase the Producer memory
buffer.memory=2G
```

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



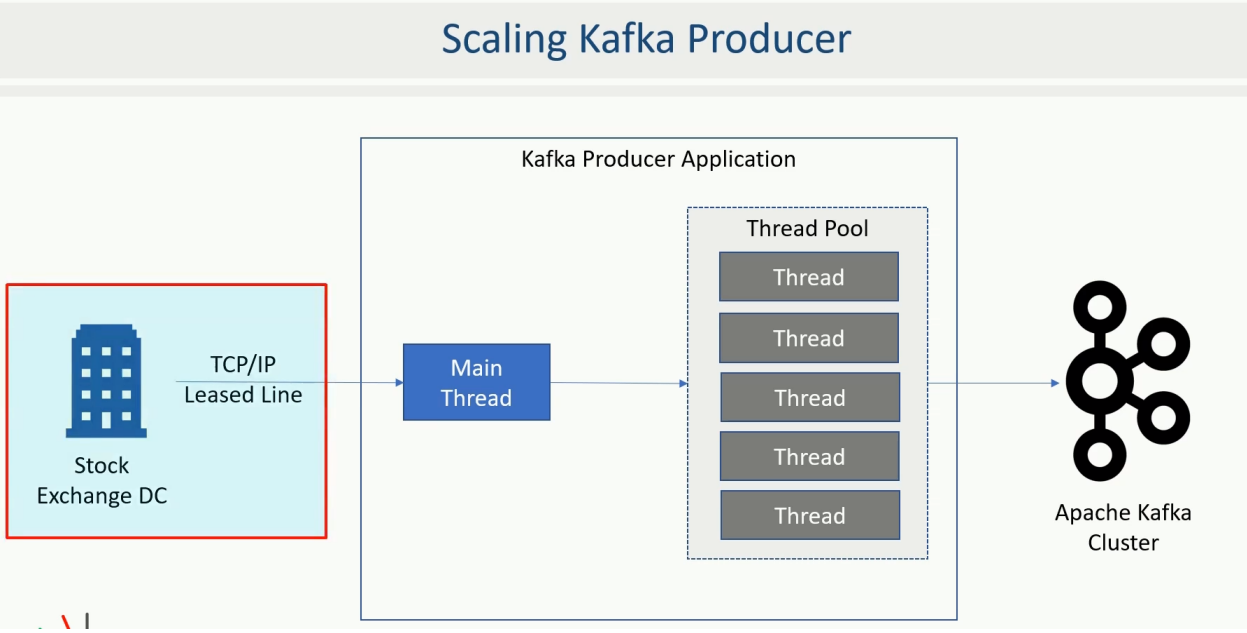
ADVANCED Producers

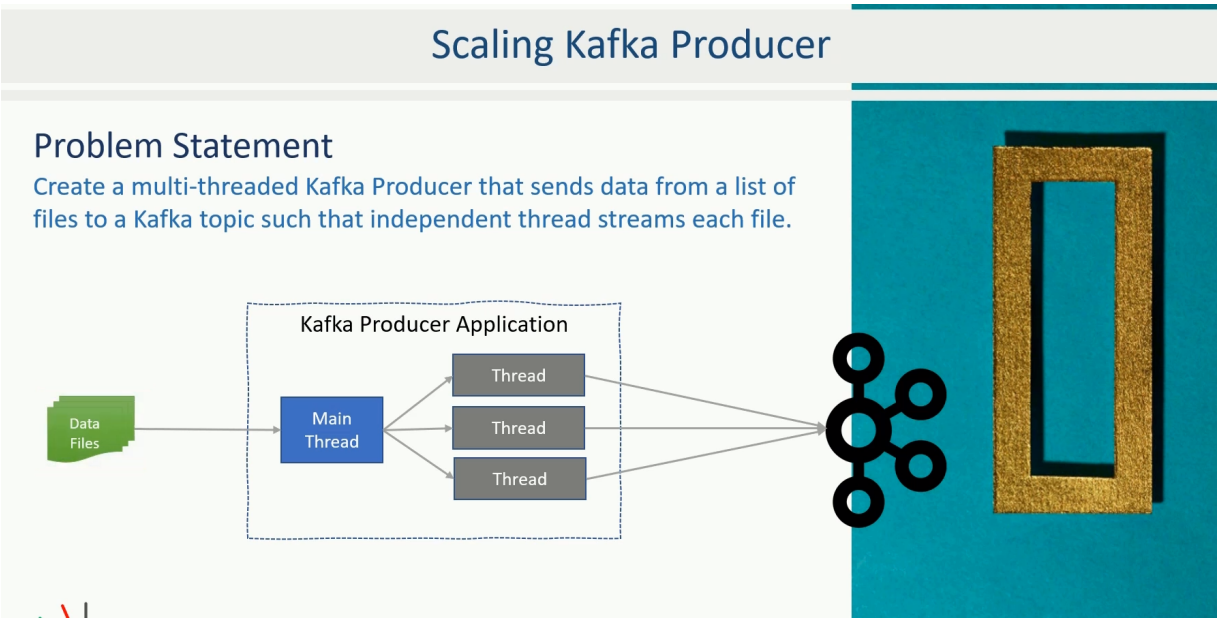
Multi Thread Producer

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 {
```

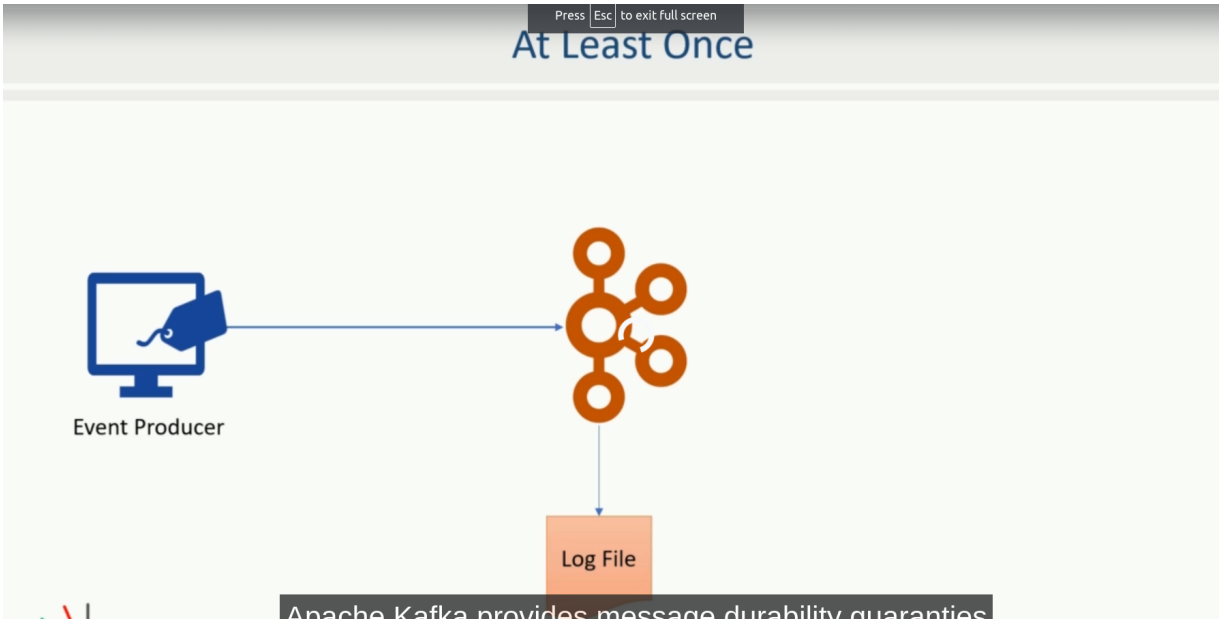
```
            producer.close();
        }



    }
}
```

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

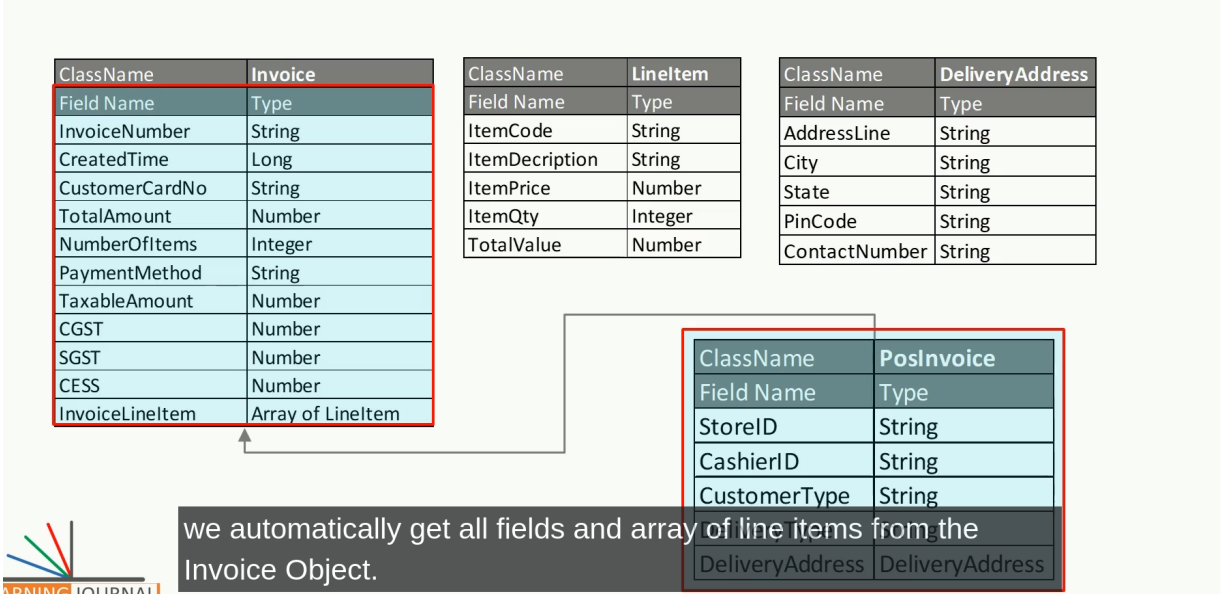
Step 2 - Serialise Java Types

we can use

1. Json Serialization
2. Avro Serialization

JSON SERIALIZATION

Lets Say we want to model an invoice event line below



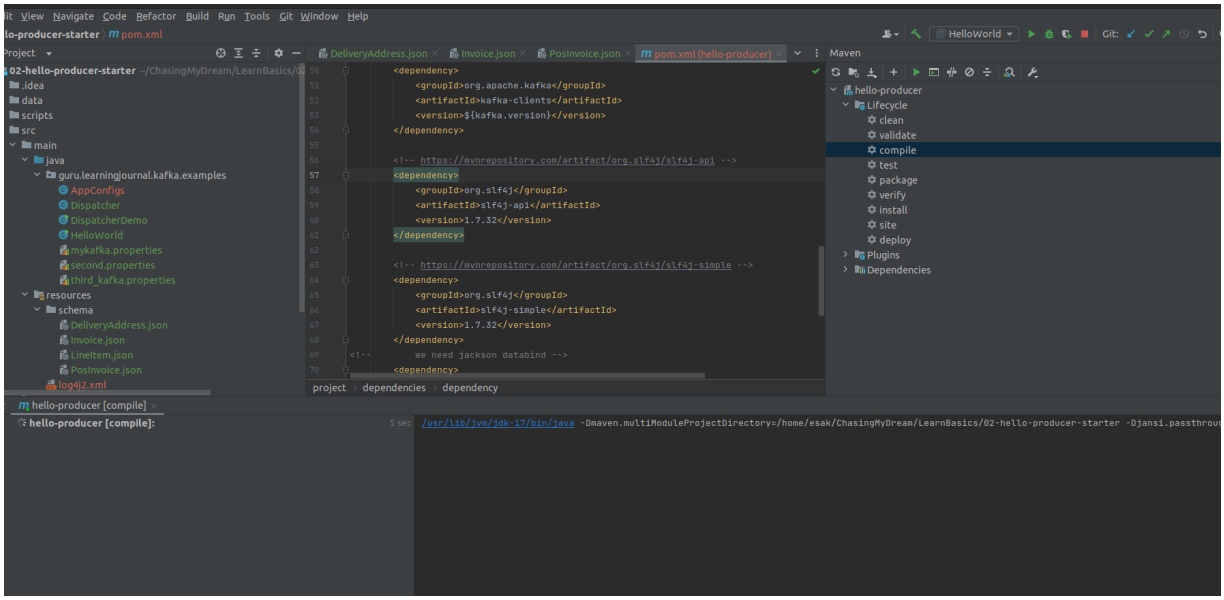
Lets Create Our First Schema

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

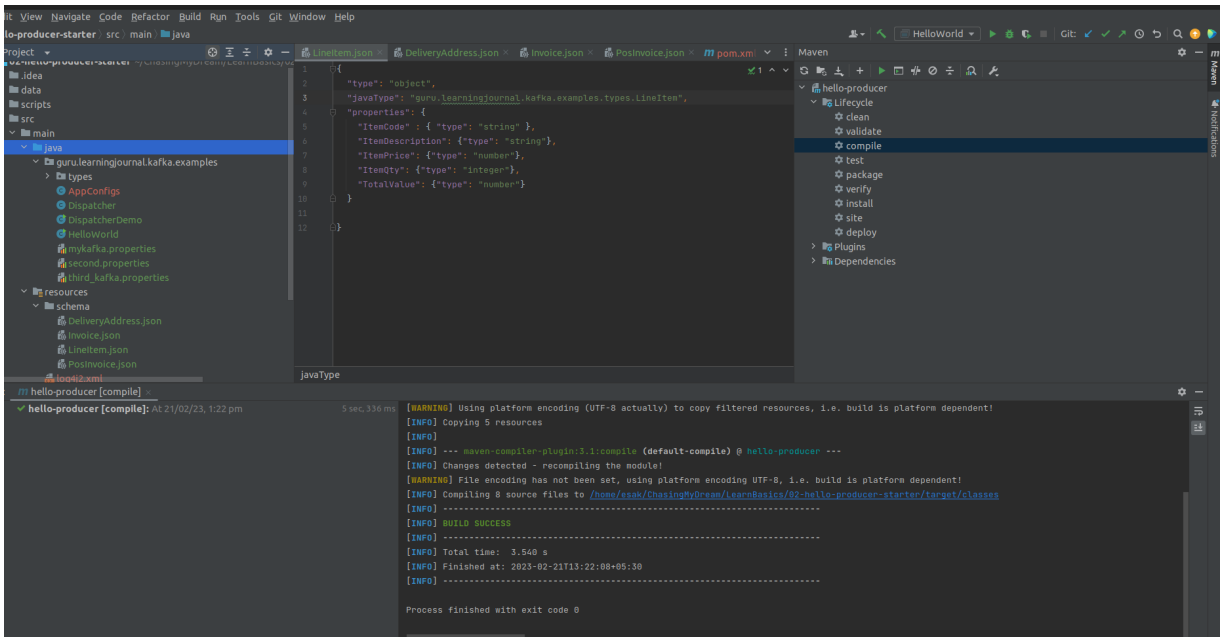
```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <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"]}
  ]
}
```

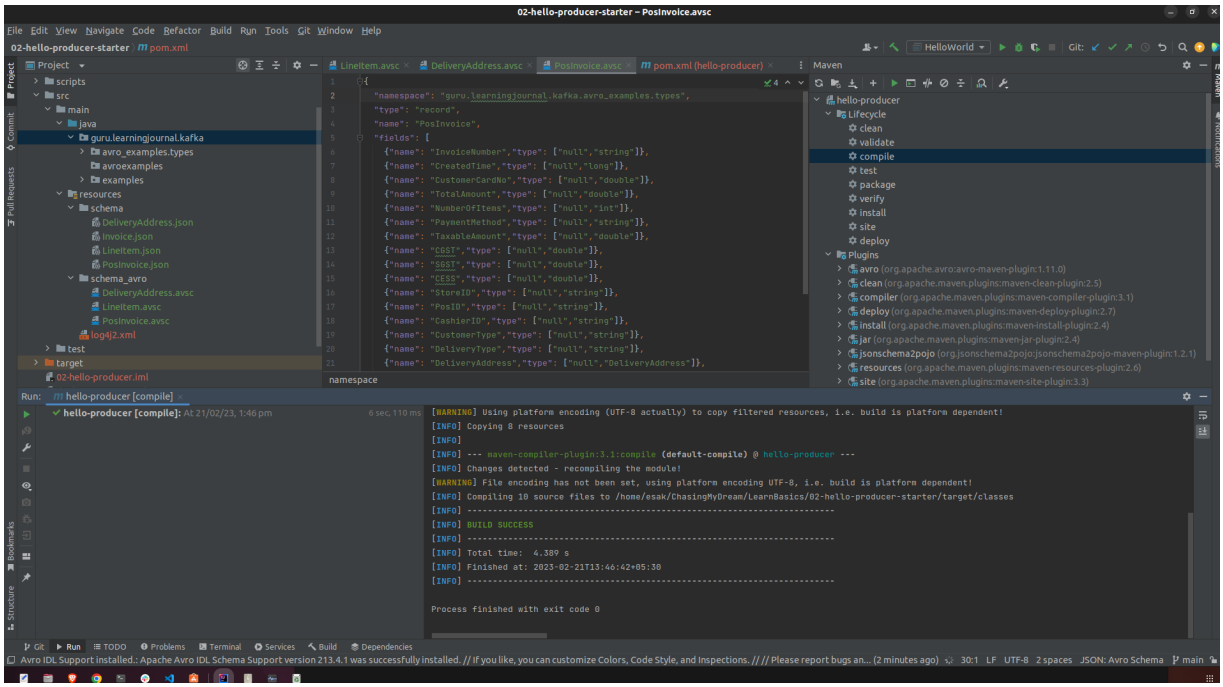
Lets Create the Delivery Address

```
{
  "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

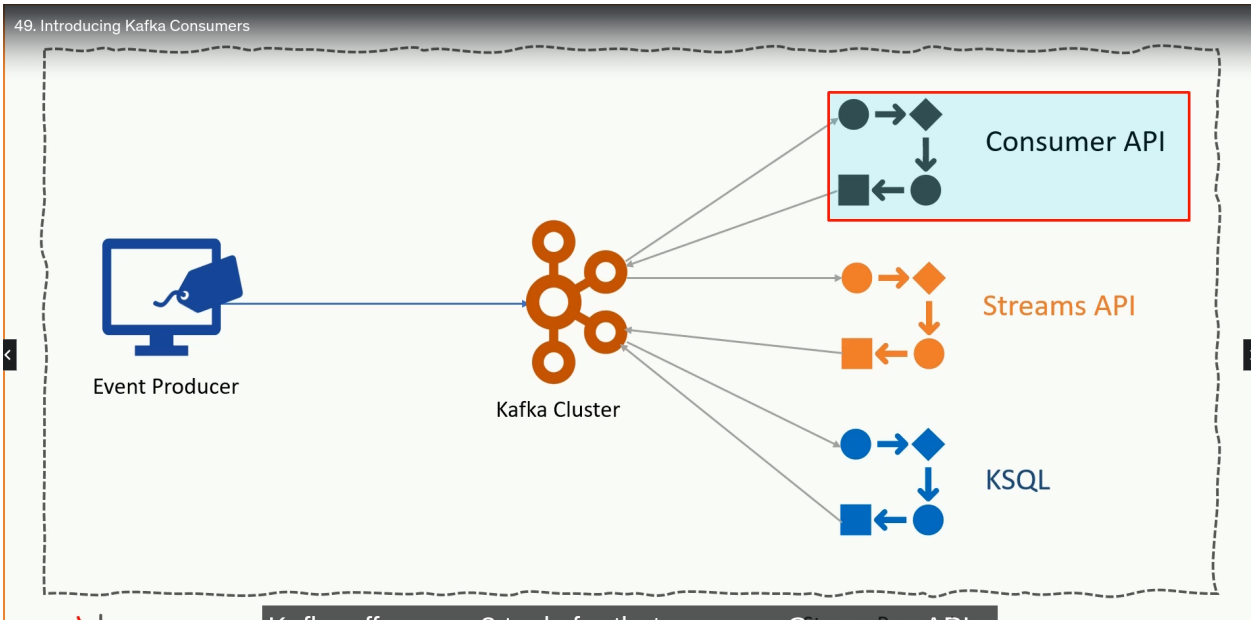
We want to produce records and send it to Kafka Broker

Create Simple Data Producer with Avro Schemas

Create Simple Data with Multi Thread producer

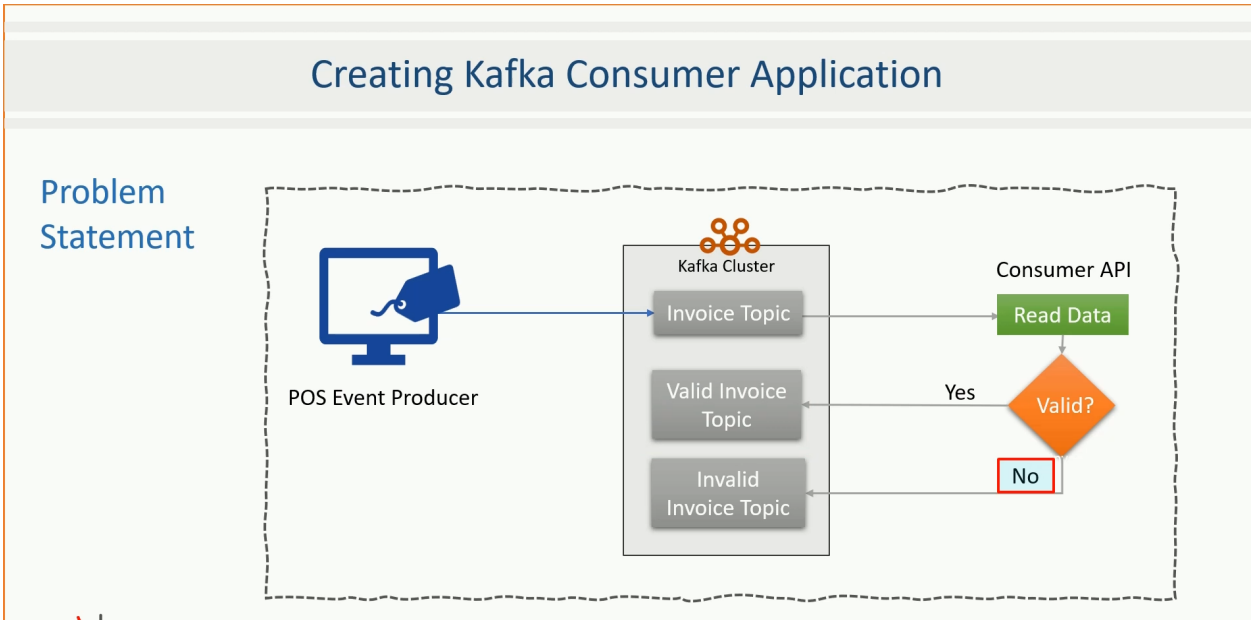
Consumers

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

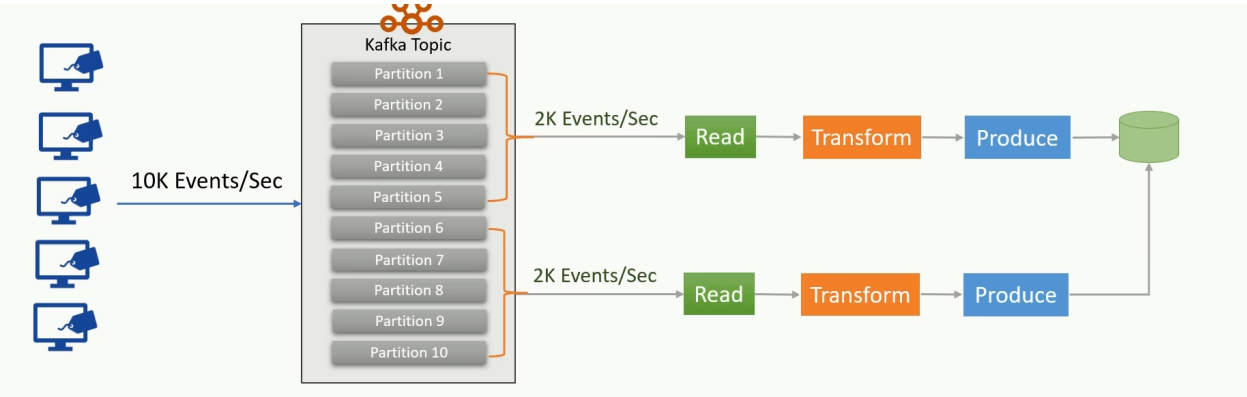
```
while(true) {
            ConsumerRecords<String, String> records = consumer.poll(Duration.ofMillis(100));
            for(ConsumerRecord<String, String> rec : records) {
                System.out.println(rec.key() + "-------"+rec.value());
            }
        }
```

OverAll Code

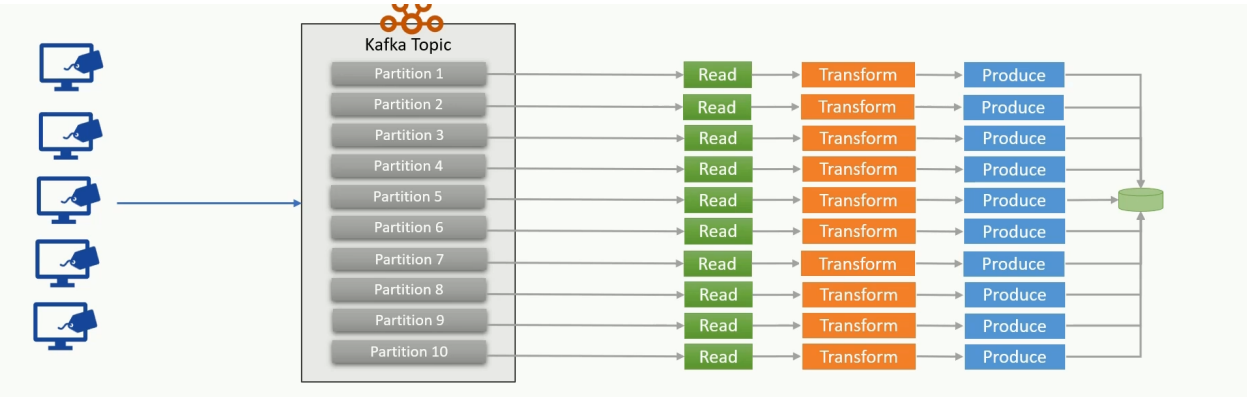
```
package org.example;

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



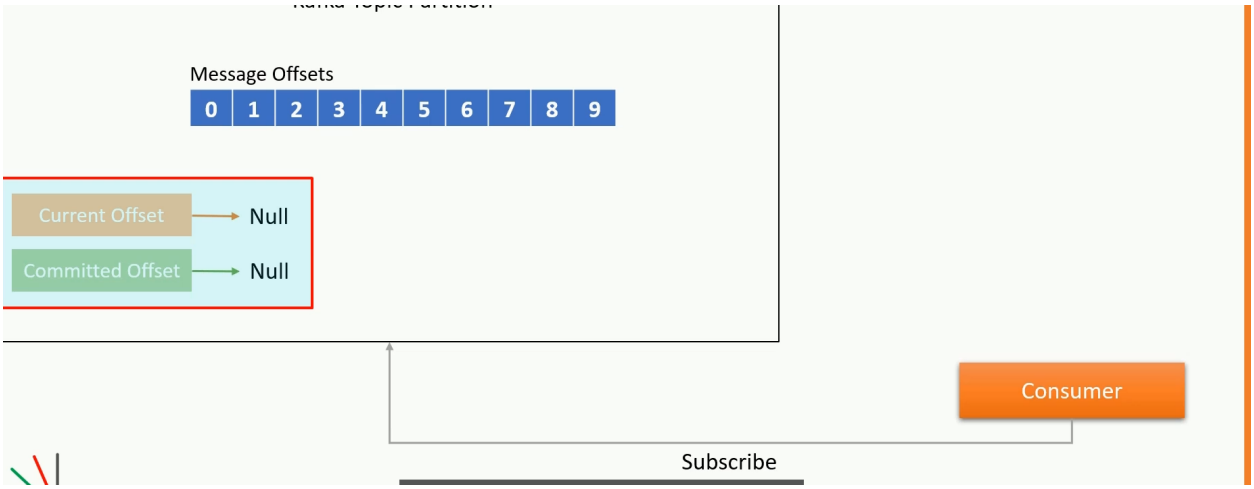
Max number of consumers are equal to the maximum number of partitions



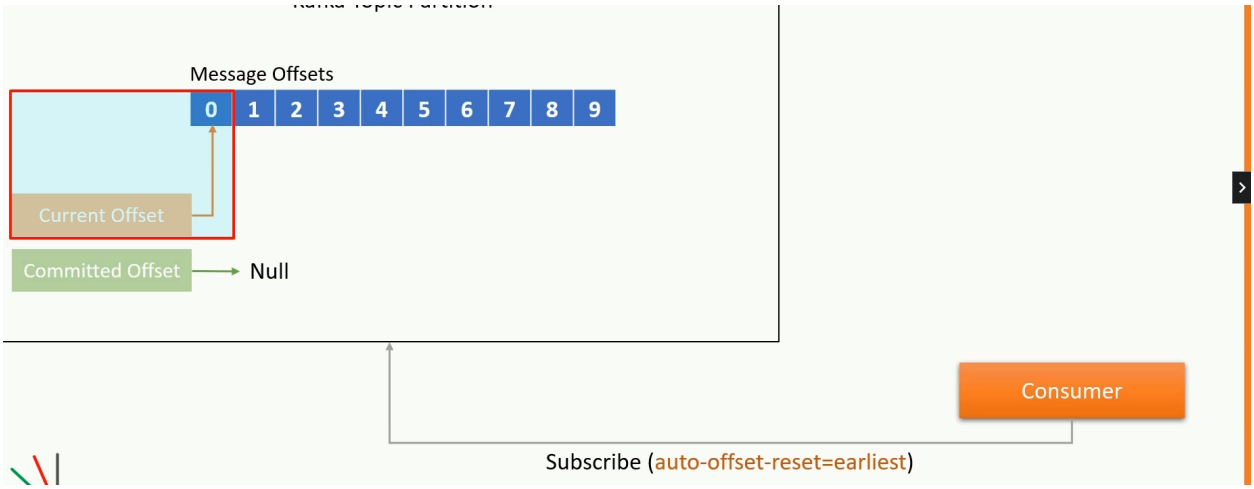
Current Offsets Vs Committed Offsets

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.

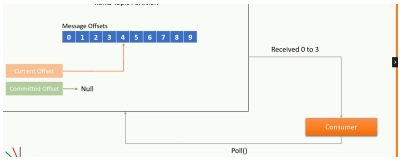
Offset is used to identify the message in the partitions.Kafka maintains 2 more offsets for each partitions



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

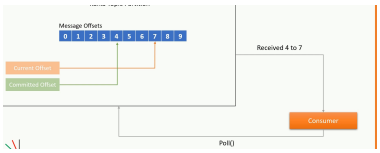


Once we start to Poll the Messages , lets Say poll 3 messages

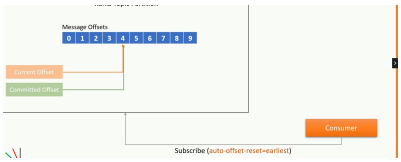


Current Offset is live with the Consumer session , once the Consumer Dies it move back to the earliest offset position

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

2.02 Create a Simple String Kafka Producer

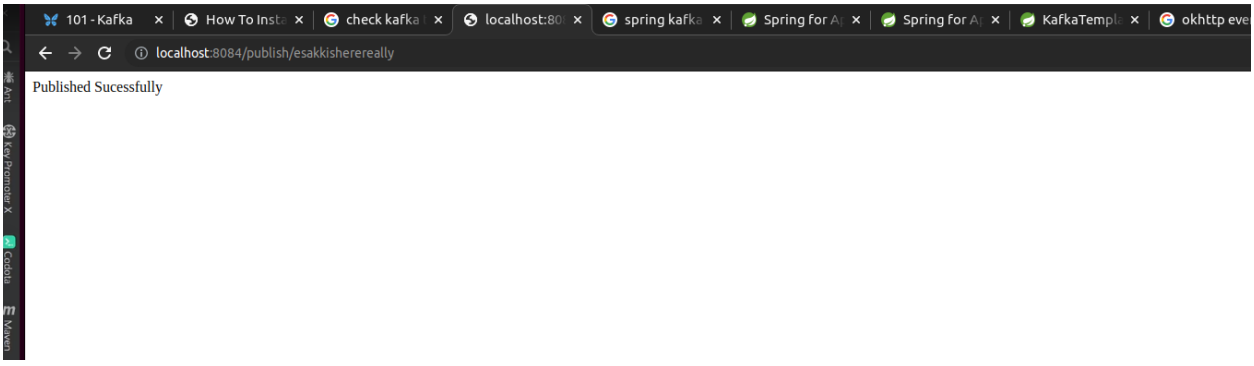
Main Class

```
@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

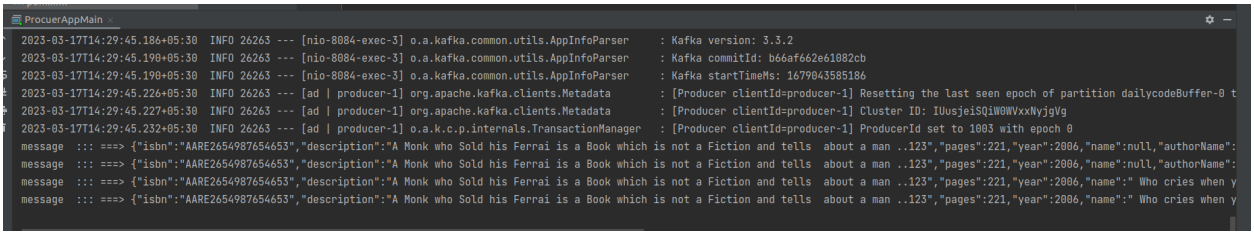
```
package com.cogesak.dailyKafka;

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



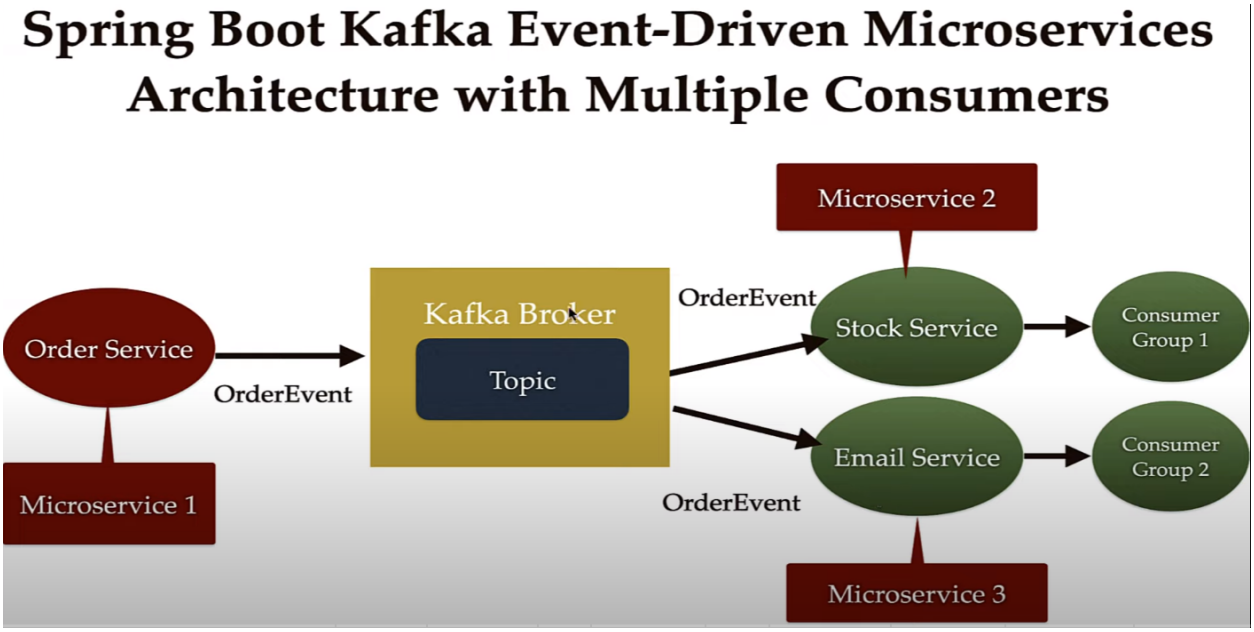

```
@Bean
    public ConsumerFactory<String, String> consumerfactory() {
        Map<String, Object> consumerconfig = new HashMap<>();
        consumerconfig.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:9092");
        consumerconfig.put(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG, "beginning");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

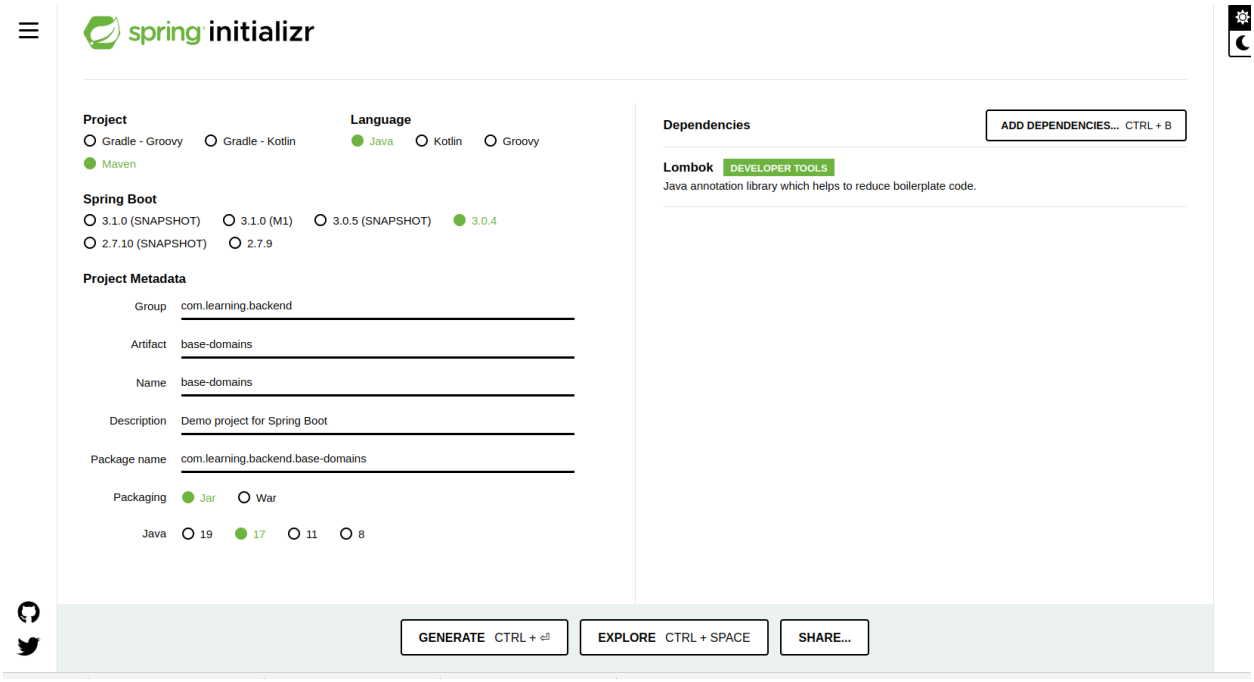
```
package com.cogesak.dailyKafka;

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

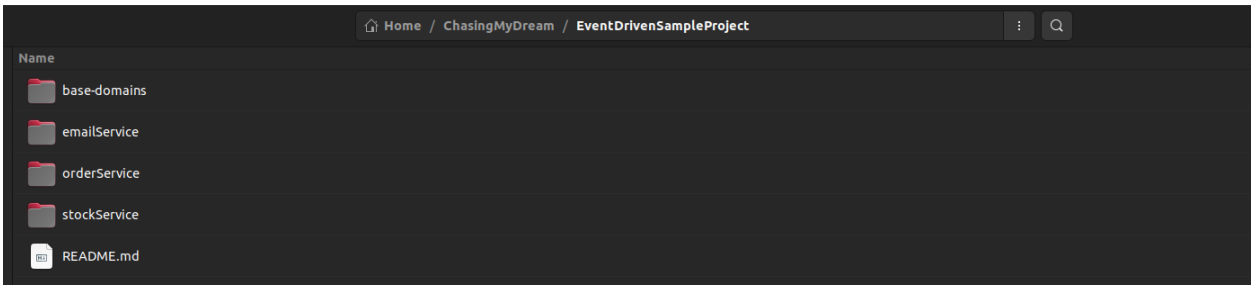
Create a simple Project where we use kafka as a message Broker



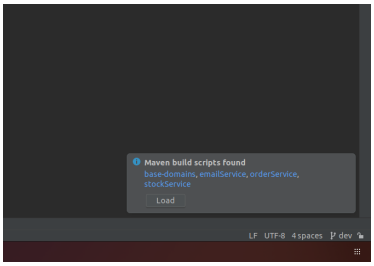
Create OrderService , StockService, EmailService, baseDomains in Start.spring.io websites

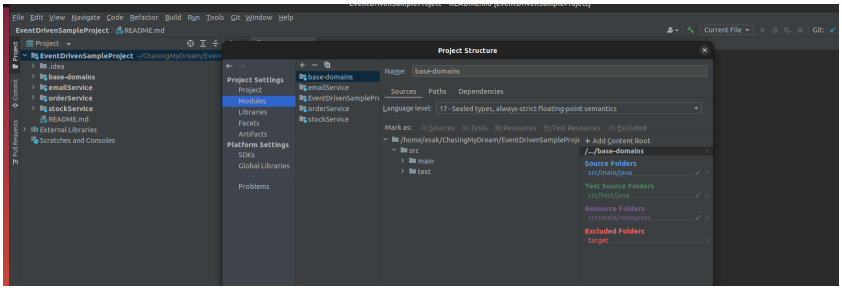


Create a Base Project and Move all the Extracted Folders to the base root Folder



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());

            }
        });


    }
}
```

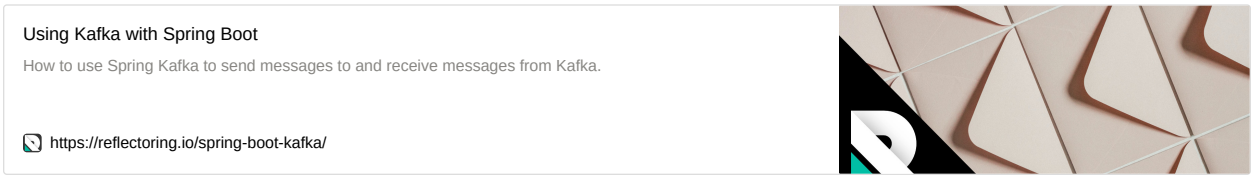
Working with StockService

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



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("Orde

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