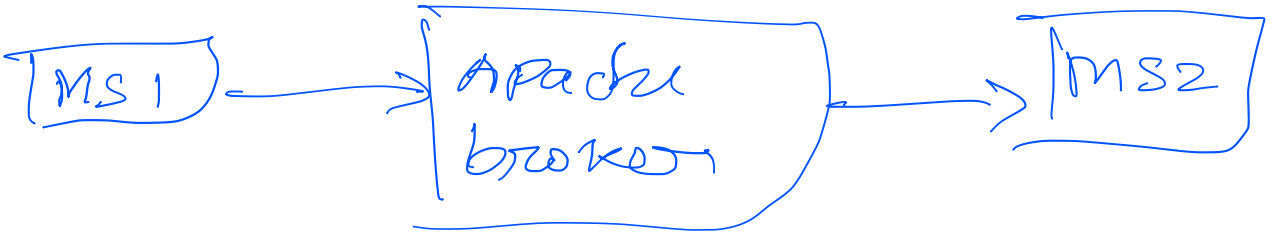
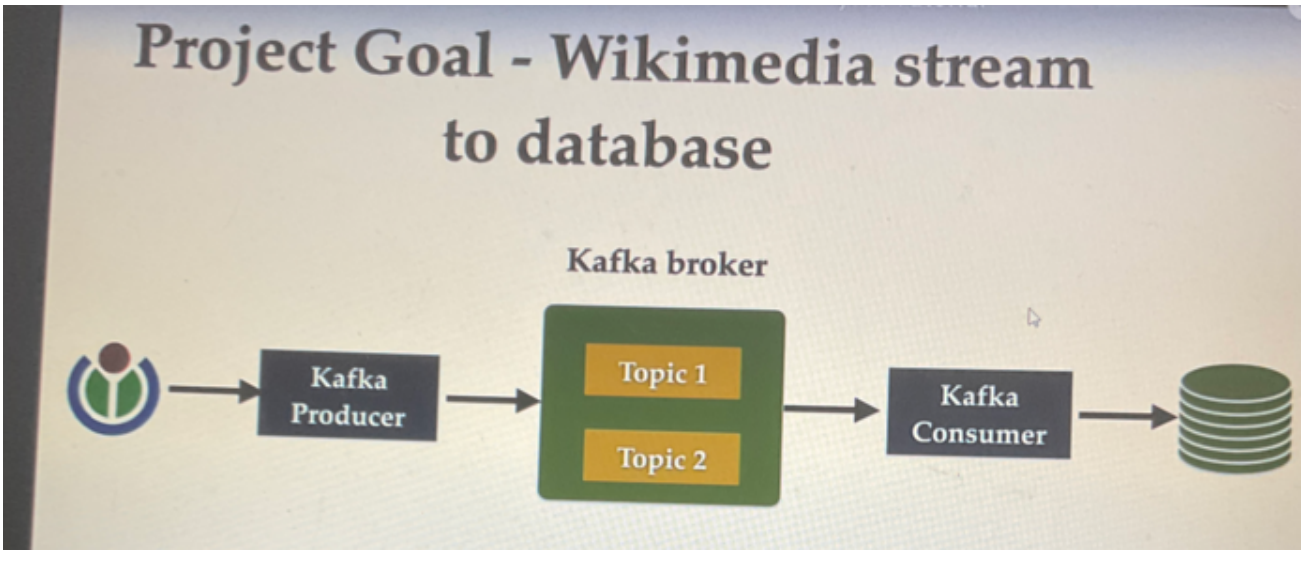
Kafka

Apache kafka

Apache kafka is a messaging System. used to communicate message between two microservices.

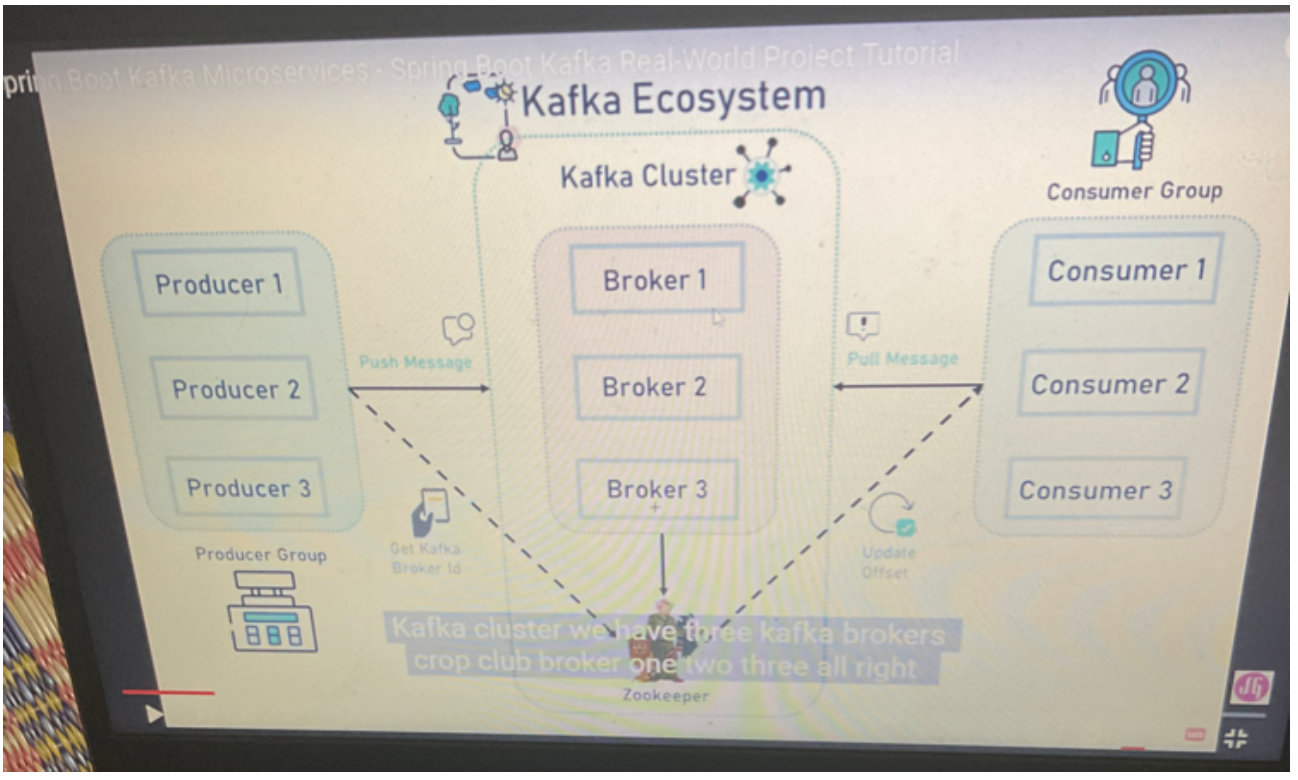




Flow is like It'll consume data from the source using kafka Producer. then after reading the data from kafka Producer. after It'll write the data in kafka broker.

then kafka Consumer - consume the source data from kafka broker and then It'll write the database.

[Apache kafka used for deal with large amount of data]



Zoo Keeper - It is a Service It'll manage the state of all the kafka brokers within a kafka cluster and It'll also manage the configuration of Topic

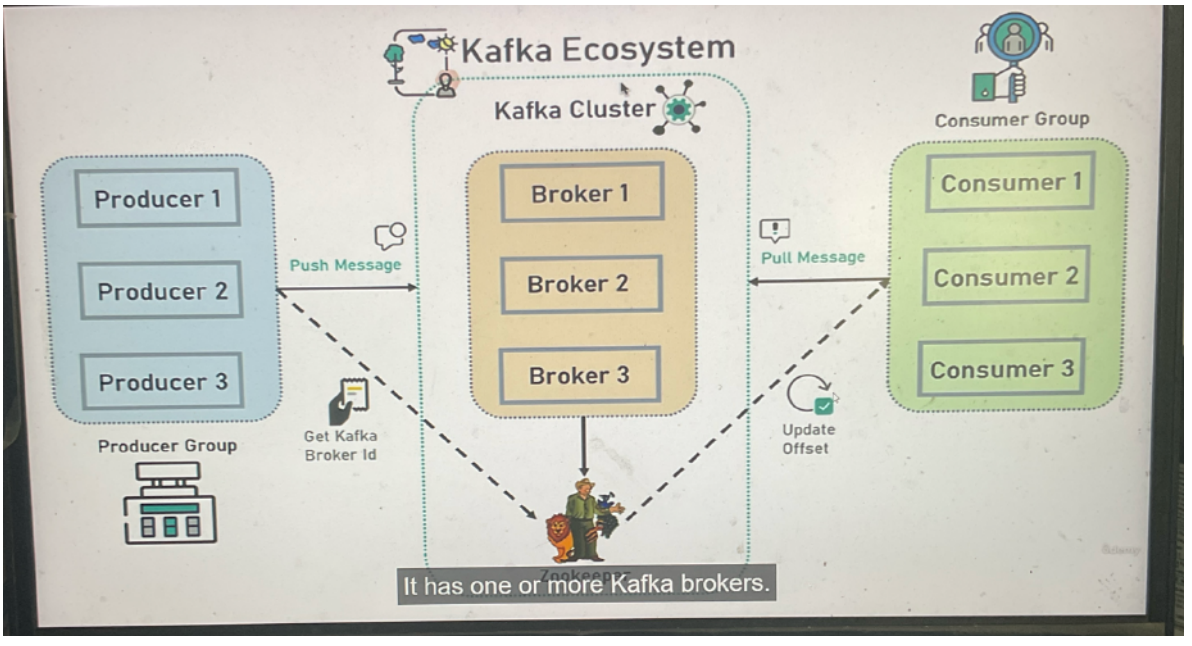
Apache Server run on Port 9092

Dependencies : Spring for Apache Kafka

→ whenever we want to create a multi model maven Project we have to add <Packaging> Pom </packaging>

Apache Kafka

It built by LinkedIn, now managed by Apache Foundation.



Apache kafka is distributed System. So it acts as kafka cluster.

Producer : It is nothing but application that Produces the message send that messages to the kafka broker.

any kinda message ⇒
- event
- stream of records
- Stream of data
- array Plaintext
- String Json.

Consumer :

Consumer will basically Consume those message from the Kafka broker.

Kafka Topic :

So basically we Create a topic in a Kafka cluster. So that Consumer can able to subscribe to that Particular topic.

Kafka cluster. It is nothing but a cluster

It has one (or) more kafka brokers.

Zoo keeper :

Zoo keeper can manages the State of all the kafka brokers.

- Zookeeper maintains the state of all the kafka brokers in a kafka cluster.
- As well as, zookeeper managing the Configuration of all the topics Producer and Consumer.

Apach kafka is Full tolerance. means

If one or a broker goes down, then it

can have capability to manage the other nodes

coz the data is distributed among different

kafka brokers.

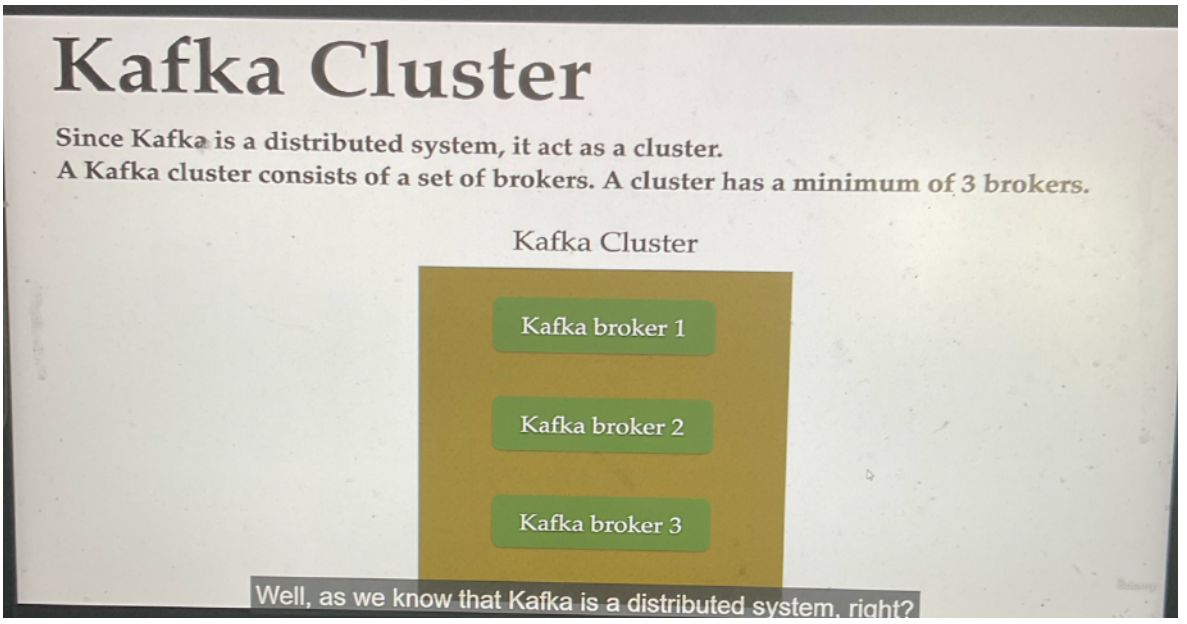
- The data can be stored in apache kafka

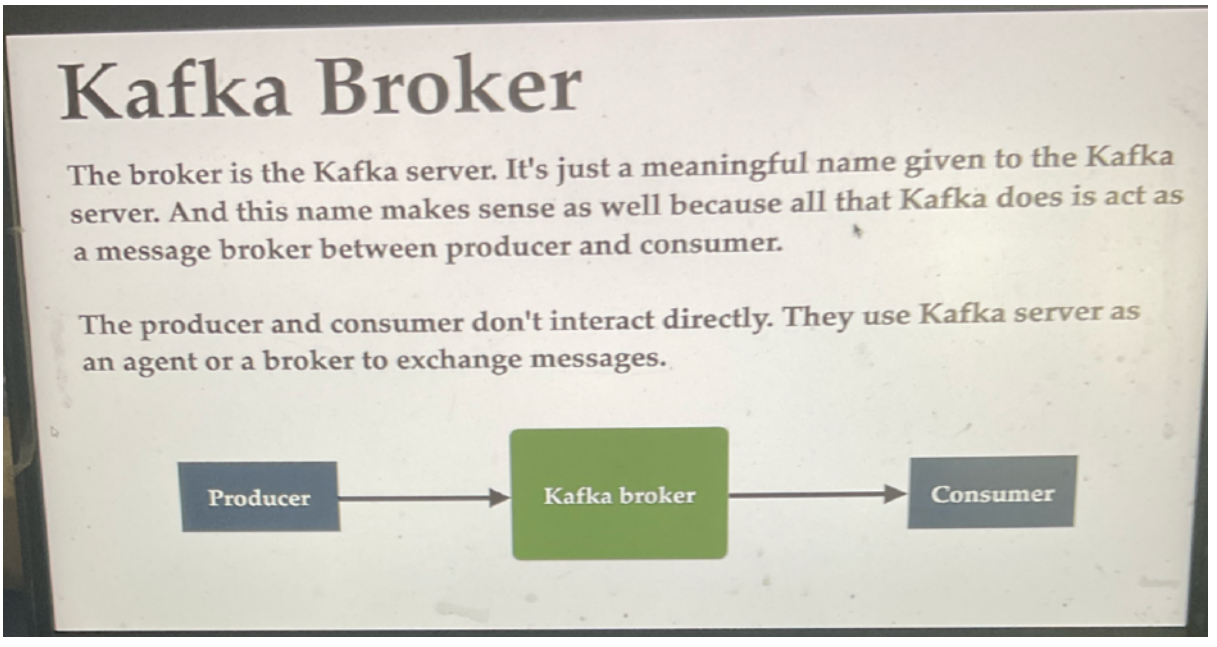
at any duration. So the Consumer can

able to consume the history data from the

Apach broker

Apache Kafka Core Concepts:





Producer

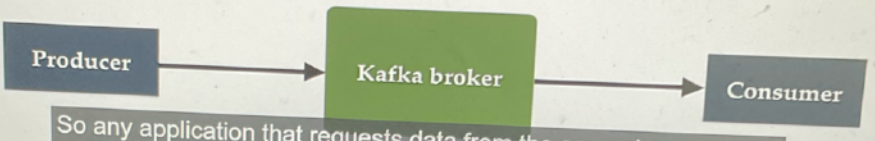
Producer is an application that sends messages. It does not send messages directly to the recipient. It send messages only to the Kafka server.

Just remember, Producer is nothing but an application that produces the messages and sends them to the Kafka

Consumer

Consumer is an application that reads messages from the Kafka server.

If producers are sending data, they must be sending it to someone, right? The consumers are the recipients. But remember that the producers don't send data to a recipient address. They just send it to Kafka server. And anyone who is interested in that data can come forward and take it from Kafka server. So, any application that requests data from a Kafka server is a consumer, and they can ask for data send by any producer provided they have permissions to read it.

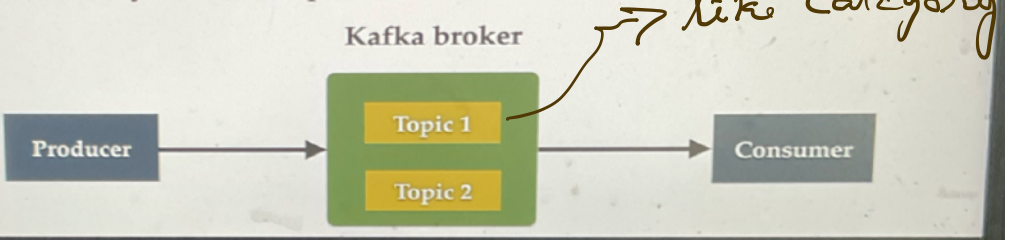


So any application that requests data from the server is a Consumer and they can ask for data sent by

Kafka Topic

We learned that producer sends data to the Kafka broker. Then a consumer can ask for data from the Kafka broker. But the question is, Which data? We need to have some identification mechanism to request data from a broker. There comes the notion of the topic.

- Topic is like a table in database or folder in a file system.
- Topic is identified by a name.
- You can have any number of topics.

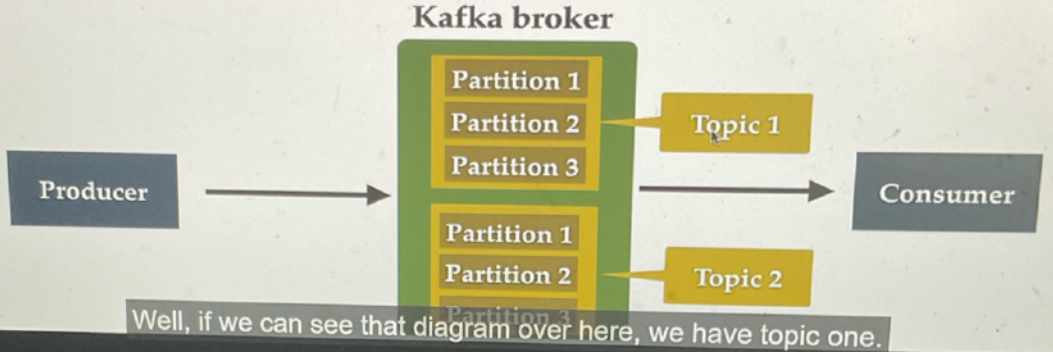


Topic = Category : (categorize message to the Kafka broker). query ur panni message access panna multiple ways like db query.

Kafka Partitions

Kafka topics are divided into a number of partitions, which contain records in an unchangeable sequence.

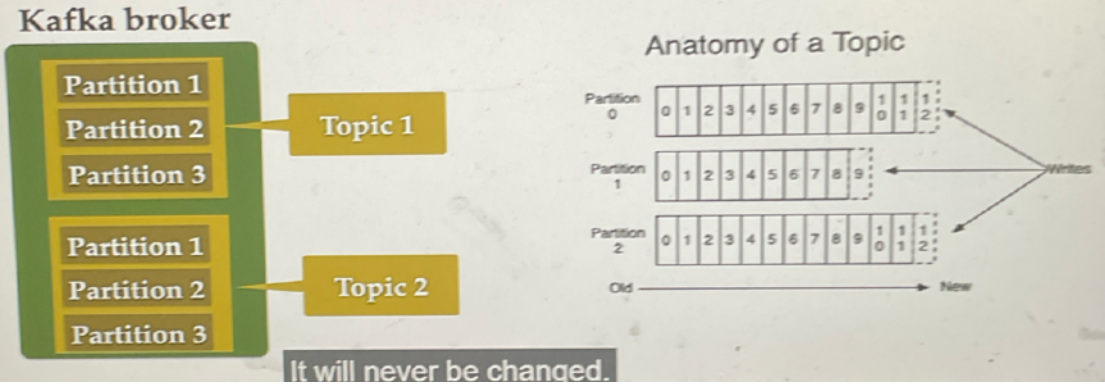
Kafka Brokers will store messages for a topic. But the capacity of data can be enormous and it may not be possible to store in a single computer. Therefore it will partitioned into multiple parts and distributed among multiple computers, since Kafka is a distributed system.



Well, if we can see that diagram over here, we have topic one.

Offsets

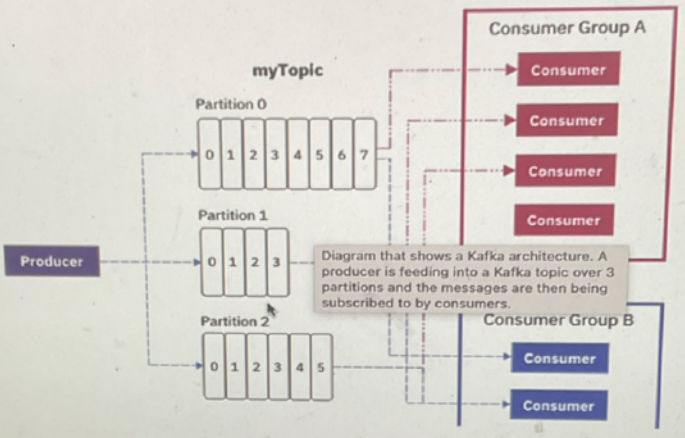
Offset is a sequence of ids given to messages as the arrive at a partition. Once the offset is assigned it will never be changed. The first message gets an offset zero. The next message receives an offset one and so on.



It will never be changed.

Consumer Groups

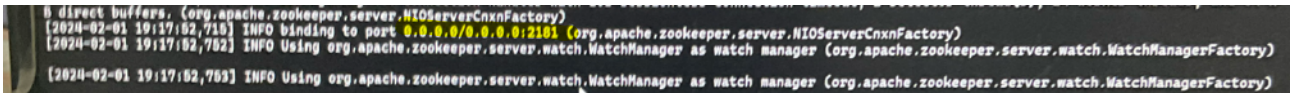
A consumer group contains one or more consumers working together to process the messages.



Kafka Installation :

Zoo keeper :

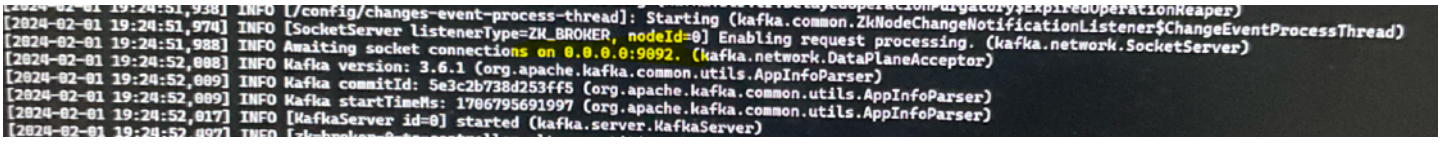
```
D:\installs>cd kafka

D:\installs\kafka>.\bin\windows\zookeeper-server-start.bat .\config\zookeeper.properties
```



means Zoo keeper Started.

Server

```
PS D:\> cd .\installs\
PS D:\installs> cd .\kafka\
PS D:\installs\kafka> .\bin\windows\kafka-server-start.bat .\config\server.properties
```



means Server Started.

Refer github textfile => for Commands

-> Create topic

-> read Event

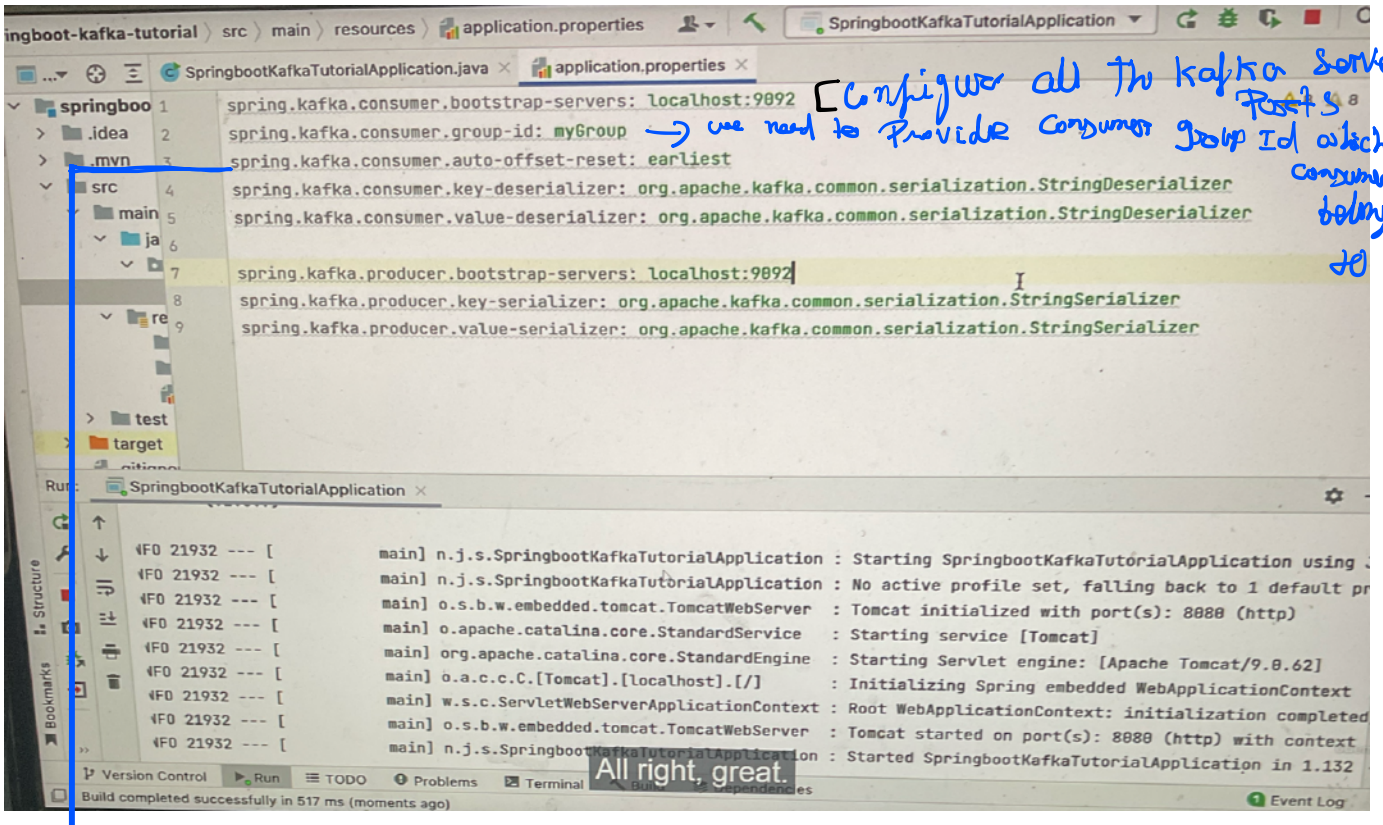
-> Create Event

Dependency :

Spring for Apache Kafka

- without Springboot we have to do lot Configurations for kafka.

Configure kafka Producer and Consumer.



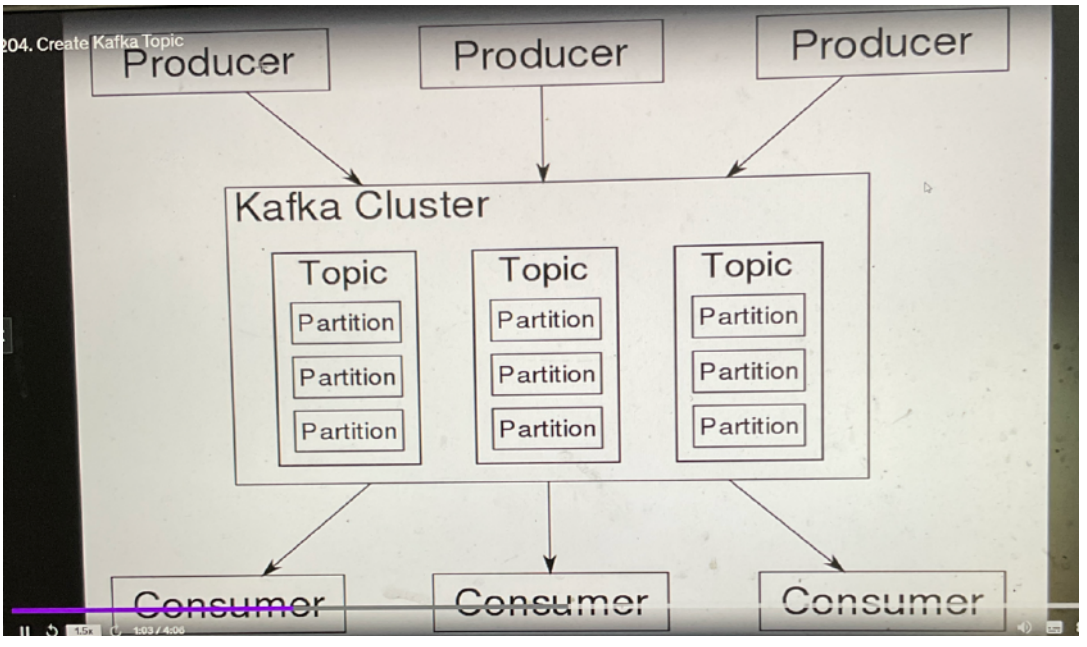
→ It will automatically reset the offset to the earliest offset.

4th and 5th : DeserializerKey, DeserializerValue } Deserialize Value and key in a message

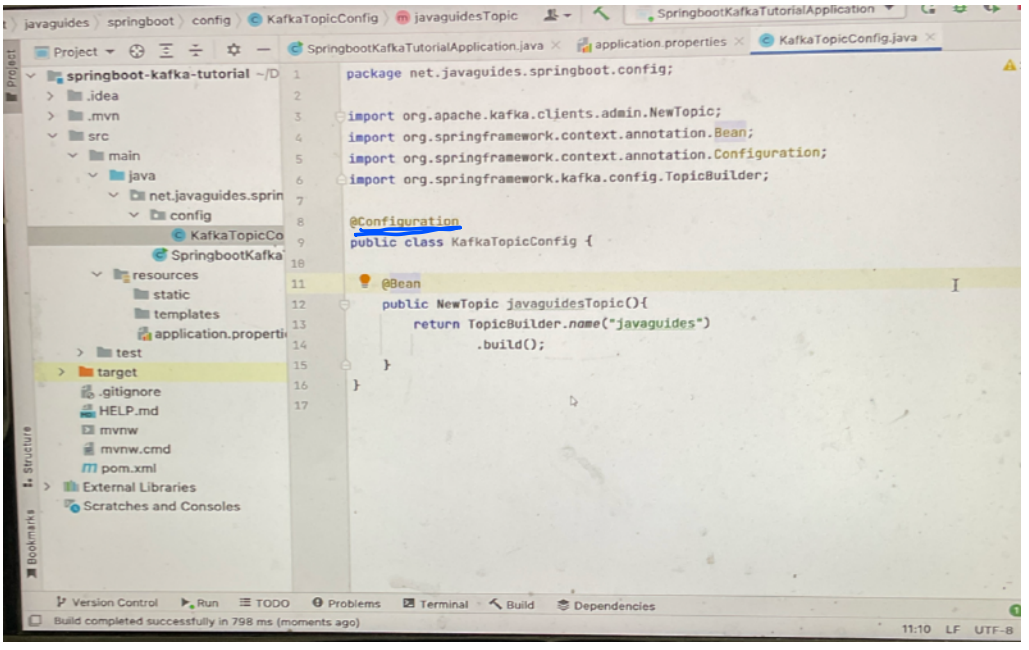
6th : Producer Port number

7th : Serializer, Serializer } Serialized Value and key in a message.

Create Kafka topic

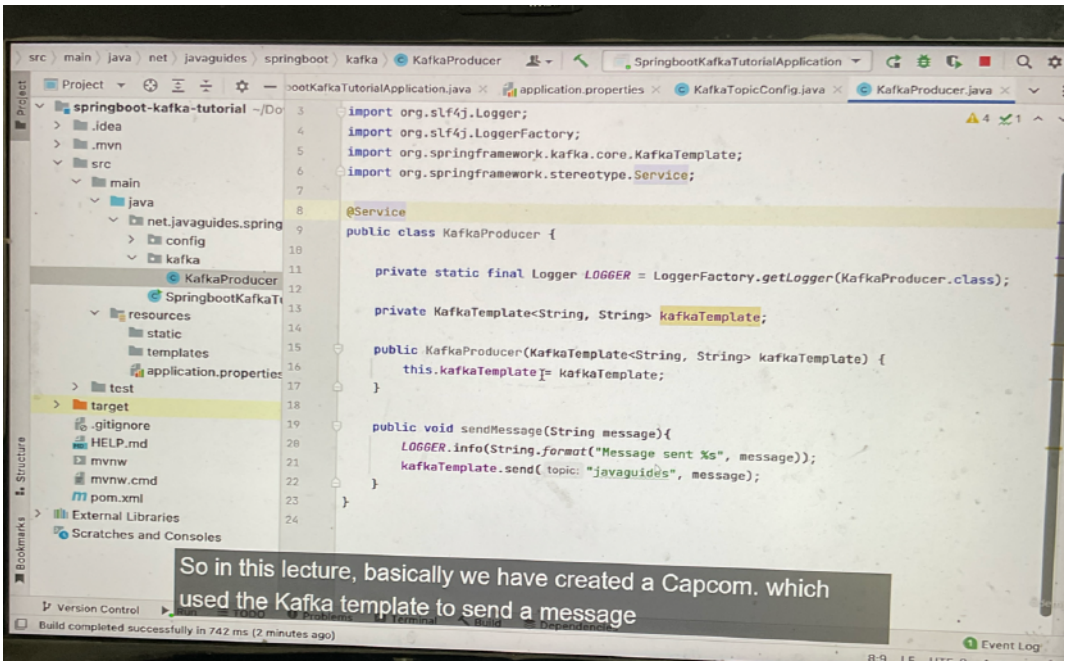


Create kafka config Package. In That Package write one class and Inside one method and anotate with @Bean.

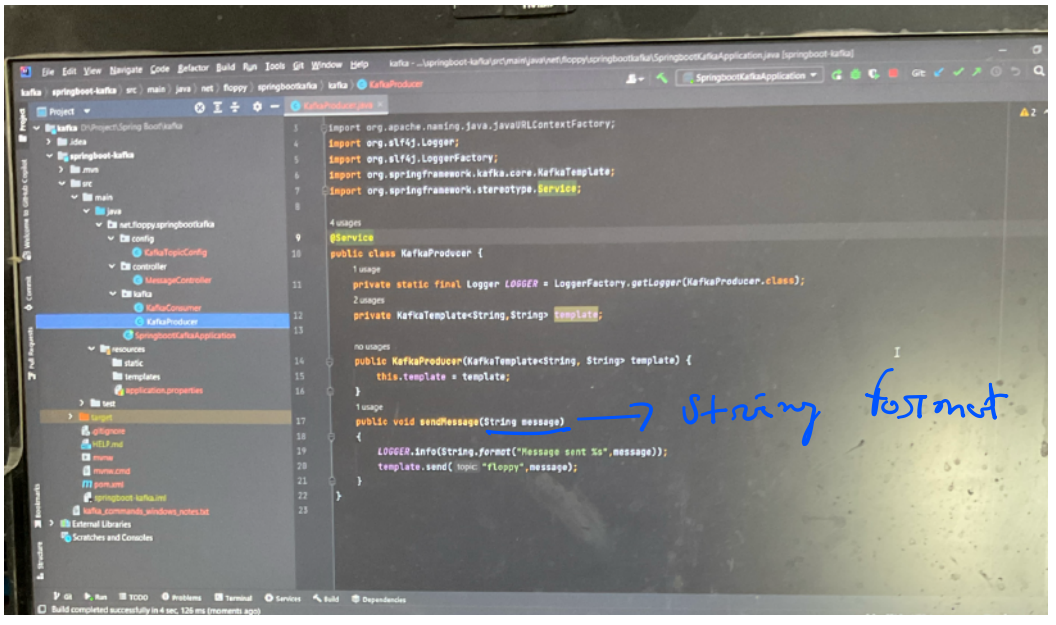


Create kafka Producer:

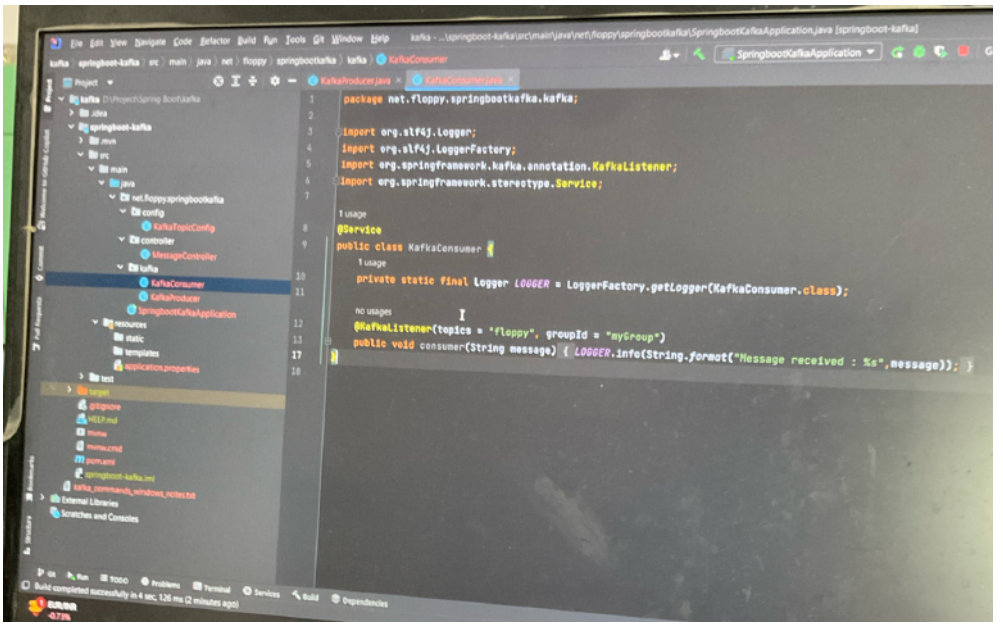
Create kafka Producer to write a message to the topic.



Kafka Producer:



Kafka Consumer:



Json Serializer & Deserializer

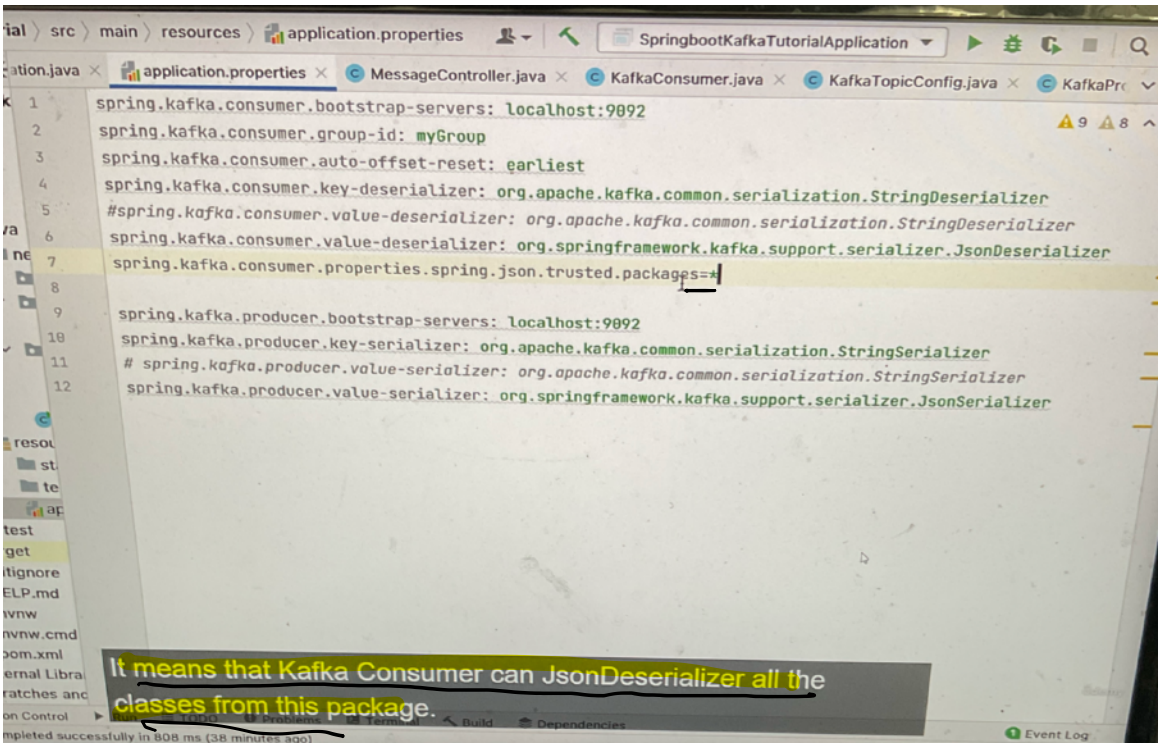
Configure Kafka Producer and Consumer for JSON Serializer and Deserializer

How to send and receive a Java Object as a JSON byte[] to and from Apache Kafka.

Apache Kafka stores and transports byte[]. There are a number of built-in serializers and deserializers but it doesn't include any for JSON. Spring Kafka created a JsonSerializer and JsonDeserializer which we can use to convert Java Objects to and from JSON.

We'll send a Java Object as JSON byte[] to a Kafka Topic using a JsonSerializer. Afterward, we'll configure how to receive a JSON byte[] and automatically convert it to a Java Object using a JsonDeserializer.

you know, it provides JsonSerializer and JsonDeserializer



RS

Difference between Synchronous and Asynchronous Communication 0

Rahul · Lecture 210 · 1 month ago

Hey Ramesh, I want to know when to use Synchronous and Asynchronous Communication, like we learnt in starting about Rest Template and WebClient to send DTOs from one service to another. And here also we will be sending something similar I guess. Can you help me with the difference like, when do we use what type of communication, which type is preferred over other, if we have to only send DTOs?

2 replies **Follow replies**

Ramesh — Instructor 0

1 month ago

Synchronous Communication:

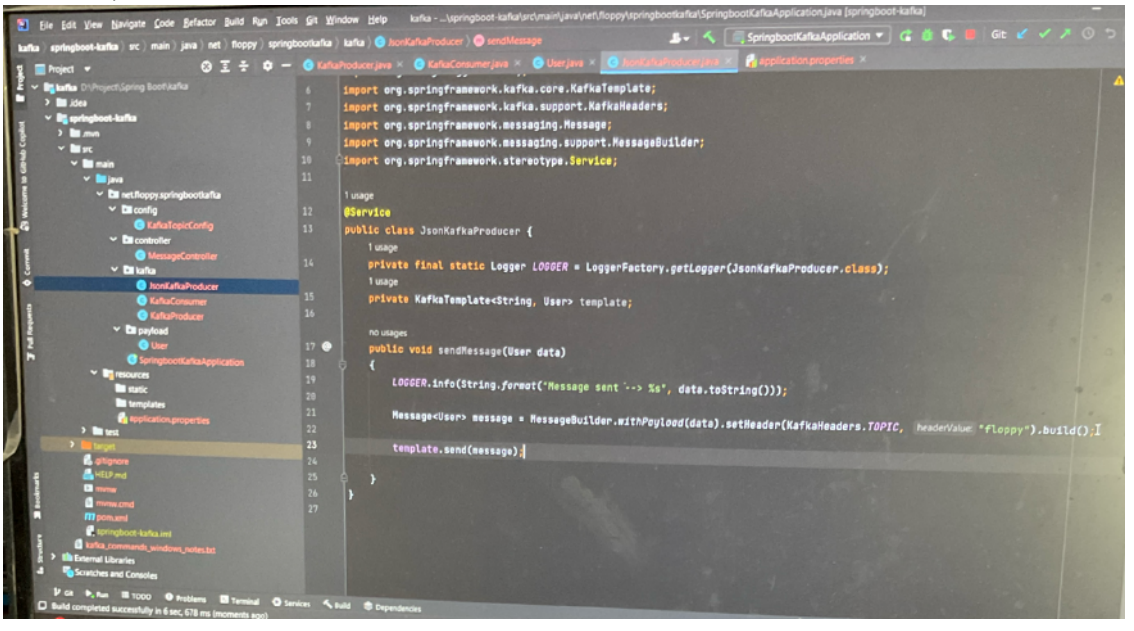
- Blocking operations; the sender waits for a response.
- Used when immediate feedback is needed, such as CRUD operations in web services.
- Preferred for simple request-response interactions.

Asynchronous Communication:

- Non-blocking; the sender doesn't wait for a response.
- Ideal for long-running operations or when the response isn't immediately needed.
- Better for scalability and handling high-volume traffic.

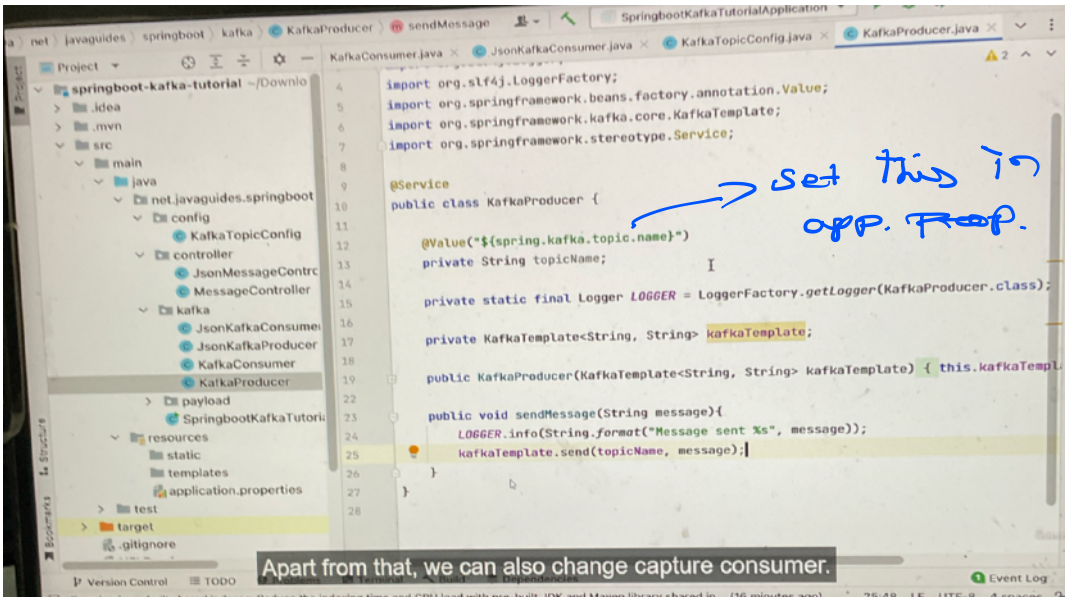
Choose synchronous for direct, immediate interactions like form submissions or data fetches. Use asynchronous for background tasks, notifications, or when dealing with operations that don't require an instant response.

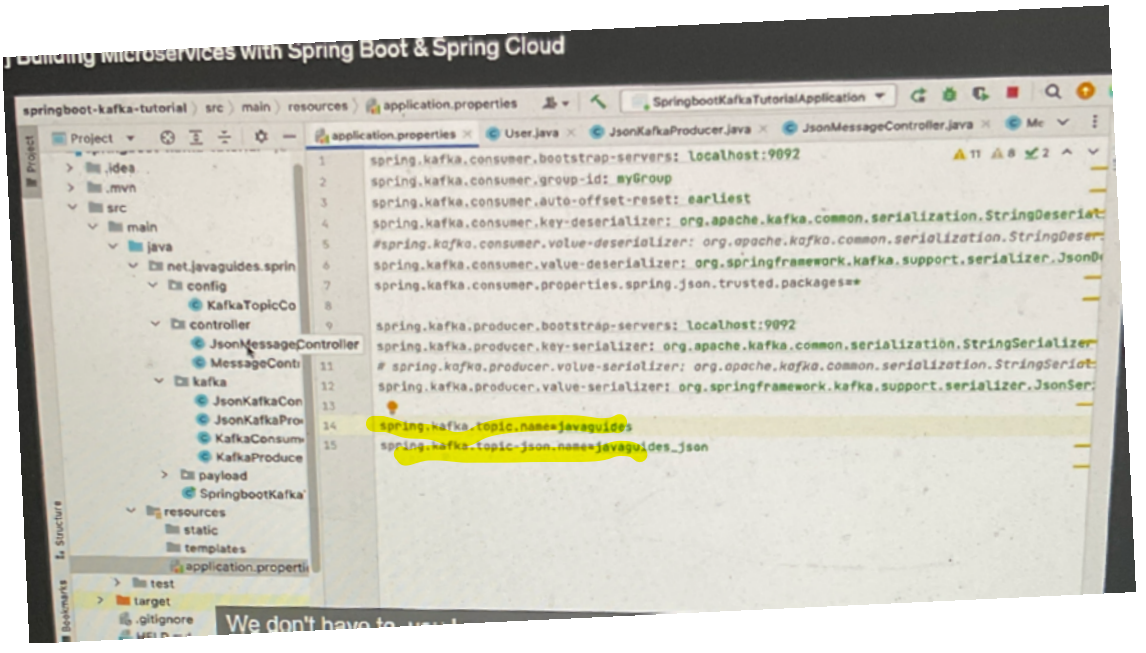
JSON Producer



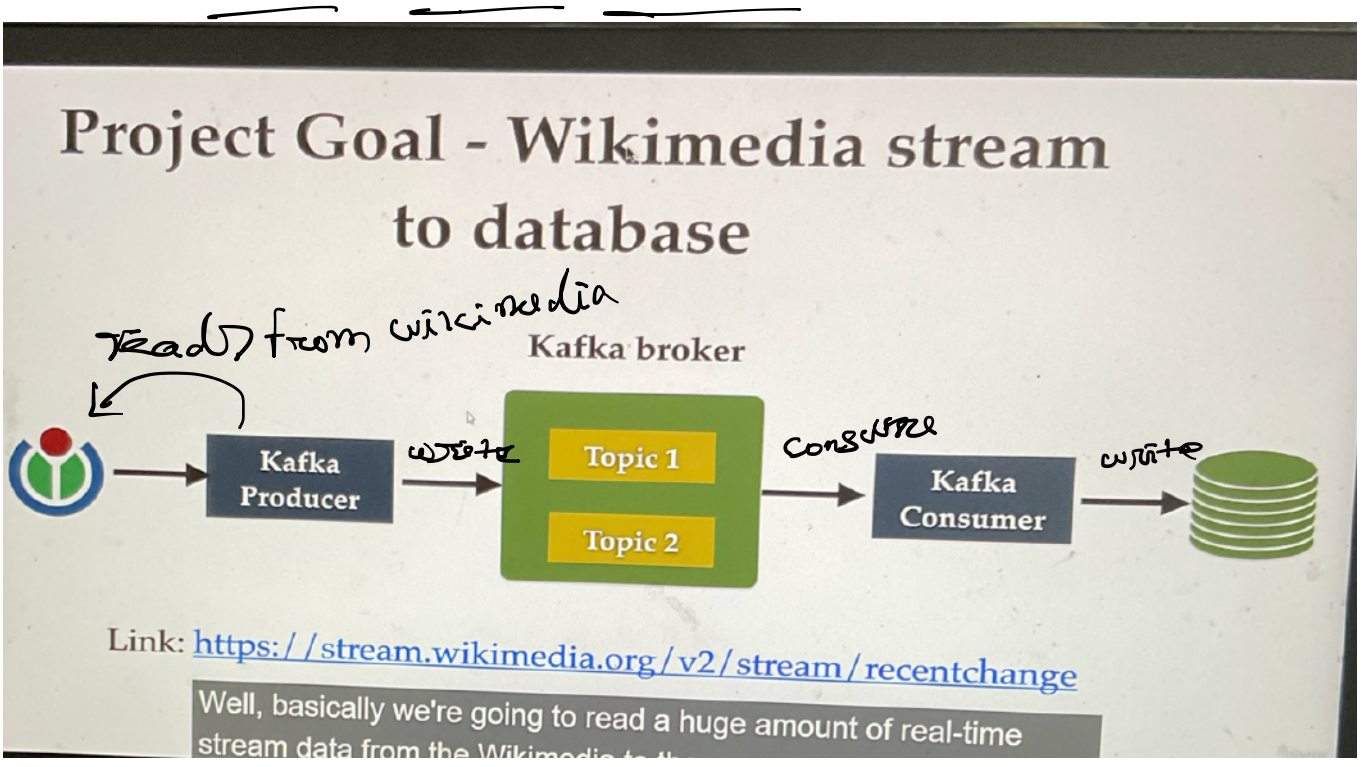
It is not a good manner to user hardcoded String in Program.

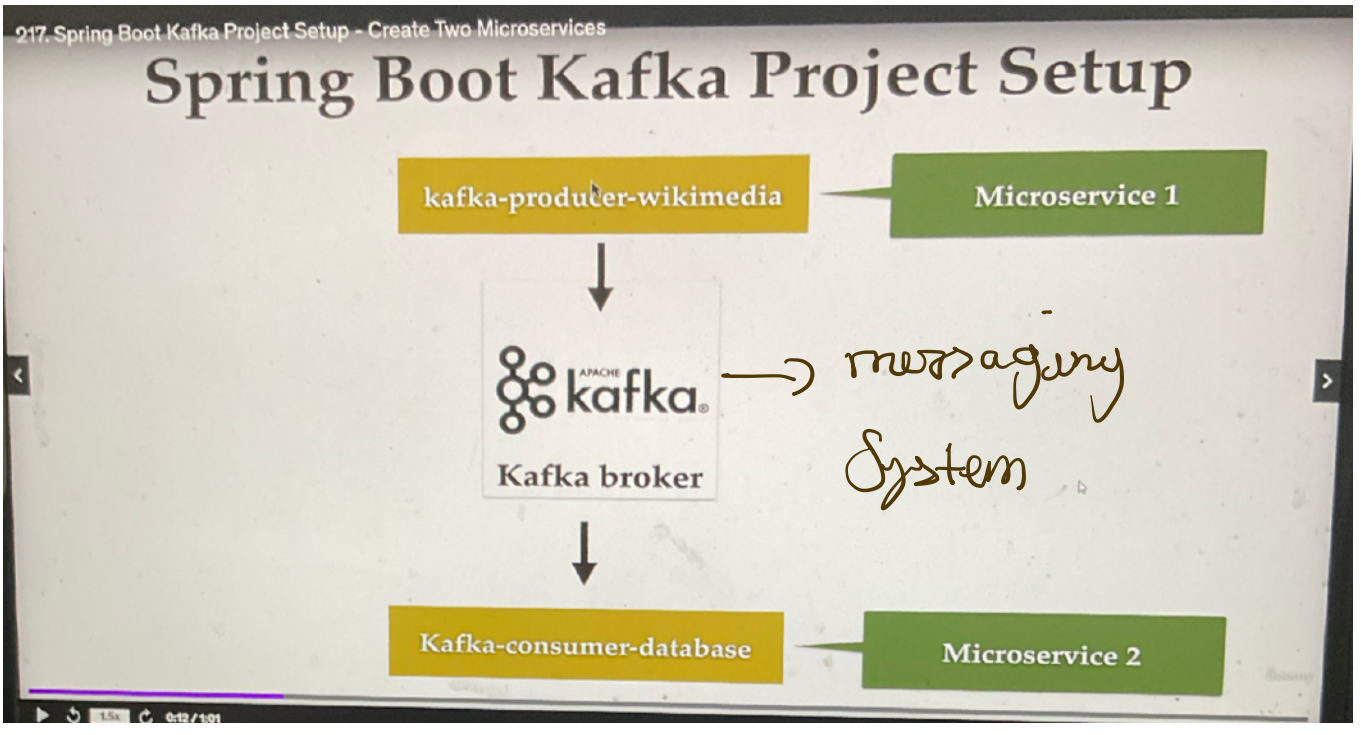
So,


springboot-kafka-tutorial
KafkaProducer.java
KafkaConsumer.java
JsonKafkaConsumer.java
KafkaTopicConfig.java
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.kafka.core.KafkaTemplate;
import org.springframework.stereotype.Service;
@Service
public class KafkaProducer {
@Value("${spring.kafka.topic.name}")
private String topicName;
private static final Logger LOGGER = LoggerFactory.getLogger(KafkaProducer.class);
private KafkaTemplate<String, String> kafkaTemplate;
public KafkaProducer(KafkaTemplate<String, String> kafkaTemplate) { this.kafkaTempl
public void sendMessage(String message){
LOGGER.info(String.format("Message sent %s", message));
kafkaTemplate.send(topicName, message);
}
}
Set this in app. prop.
Apart from that, we can also change capture consumer.


Building Microservices With Spring Boot & Spring Cloud
springboot-kafka-tutorial
application.properties
spring.kafka.consumer.bootstrap-servers: localhost:9092
spring.kafka.consumer.group-id: myGroup
spring.kafka.consumer.auto-offset-reset: earliest
spring.kafka.consumer.key-deserializer: org.apache.kafka.common.serialization.StringDeserial
spring.kafka.consumer.value-deserializer: org.springframework.kafka.support.serializer.JsonD
spring.kafka.consumer.properties.spring.json.trusted.packages=*
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.JsonSer
spring.kafka.topic.name=javaguides
spring.kafka.topic-json.name=javaguides_json
We don't have to

Real World Project

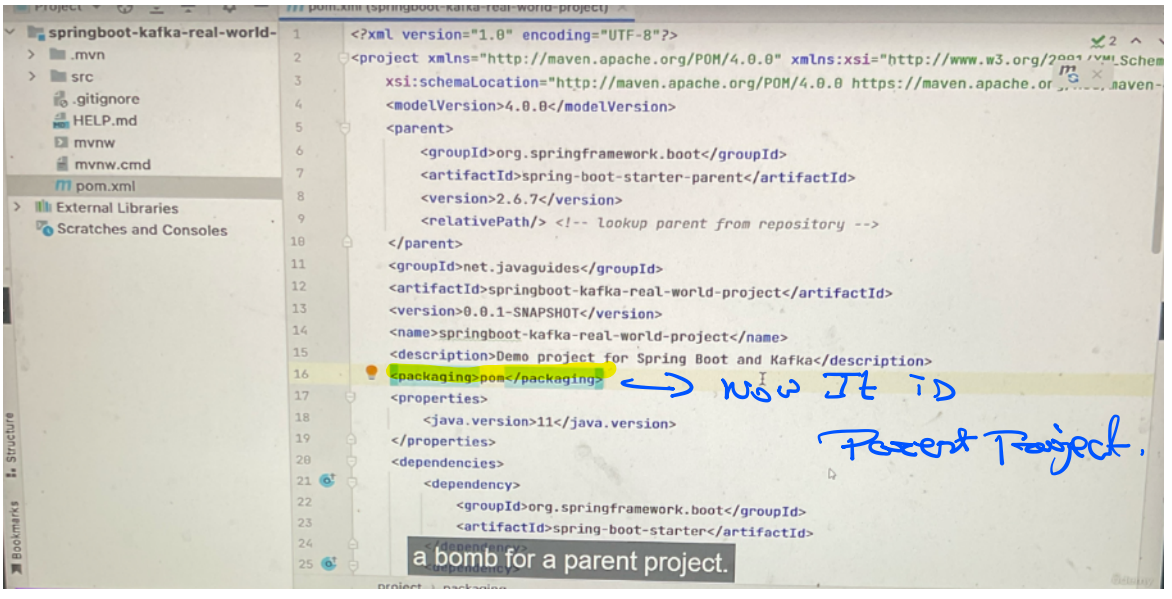




Wikimedia Producer Setup

To make multi module maven Project.

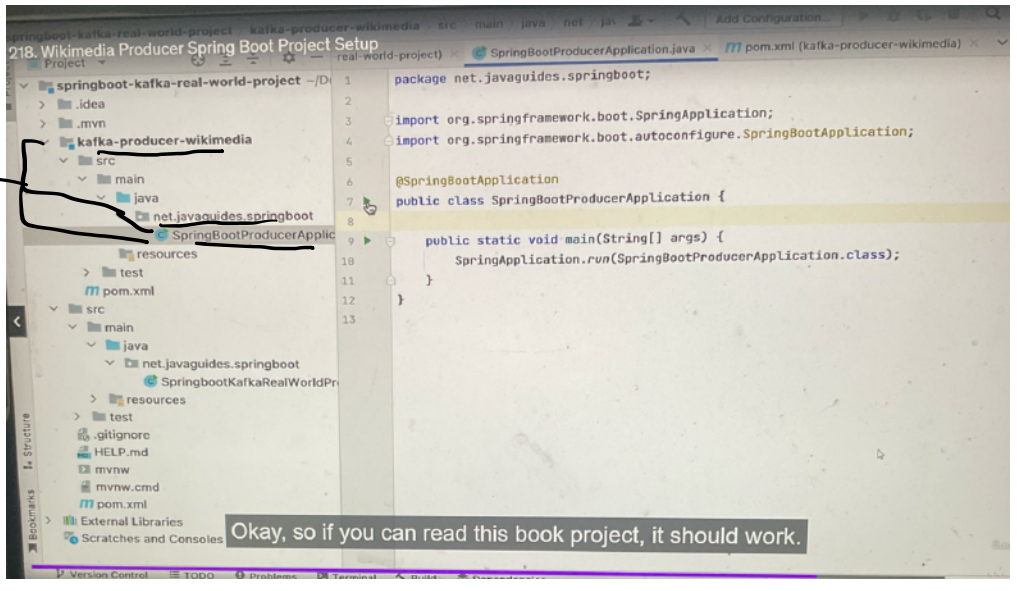
The neccesary to make, first Project as Parent Project. To make that, ...

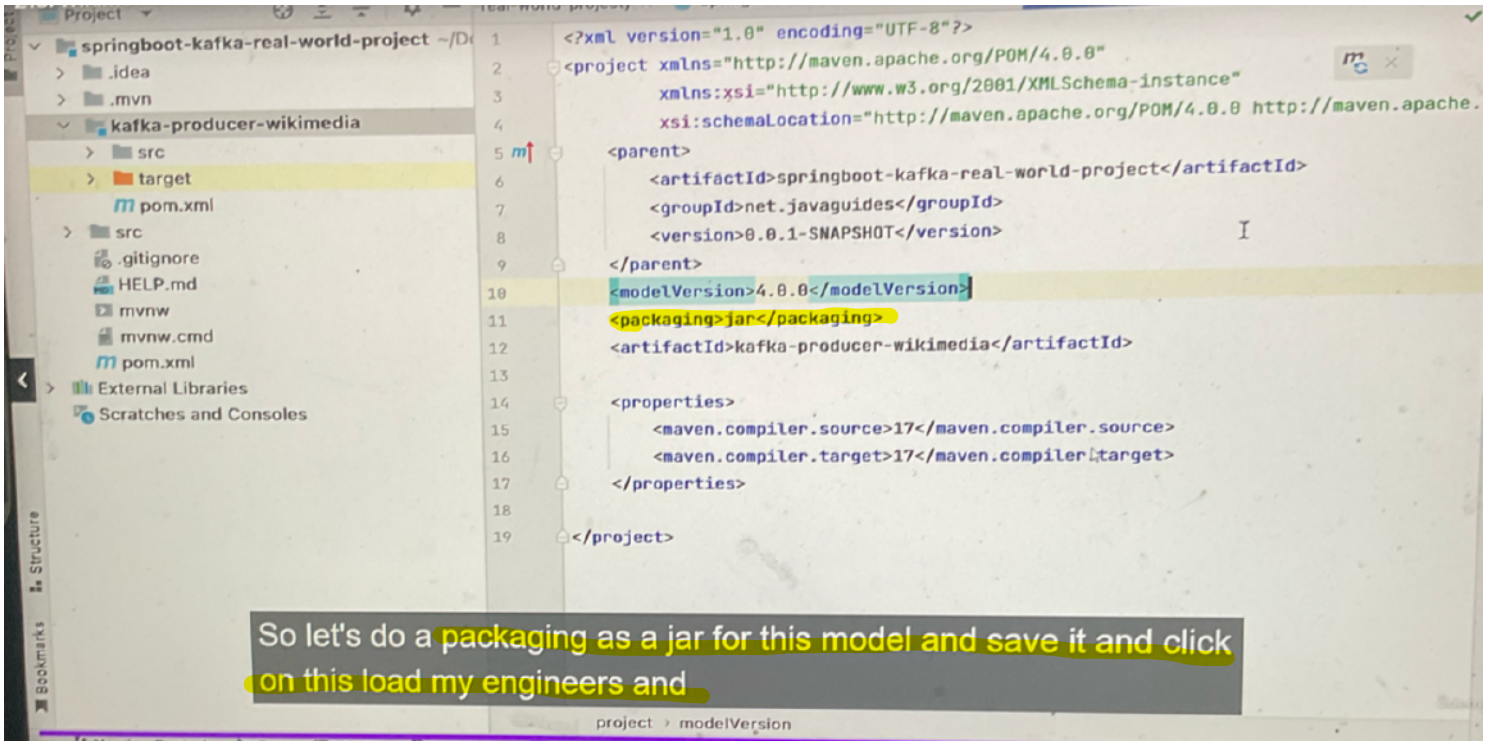


→ Now It is Parent Project.

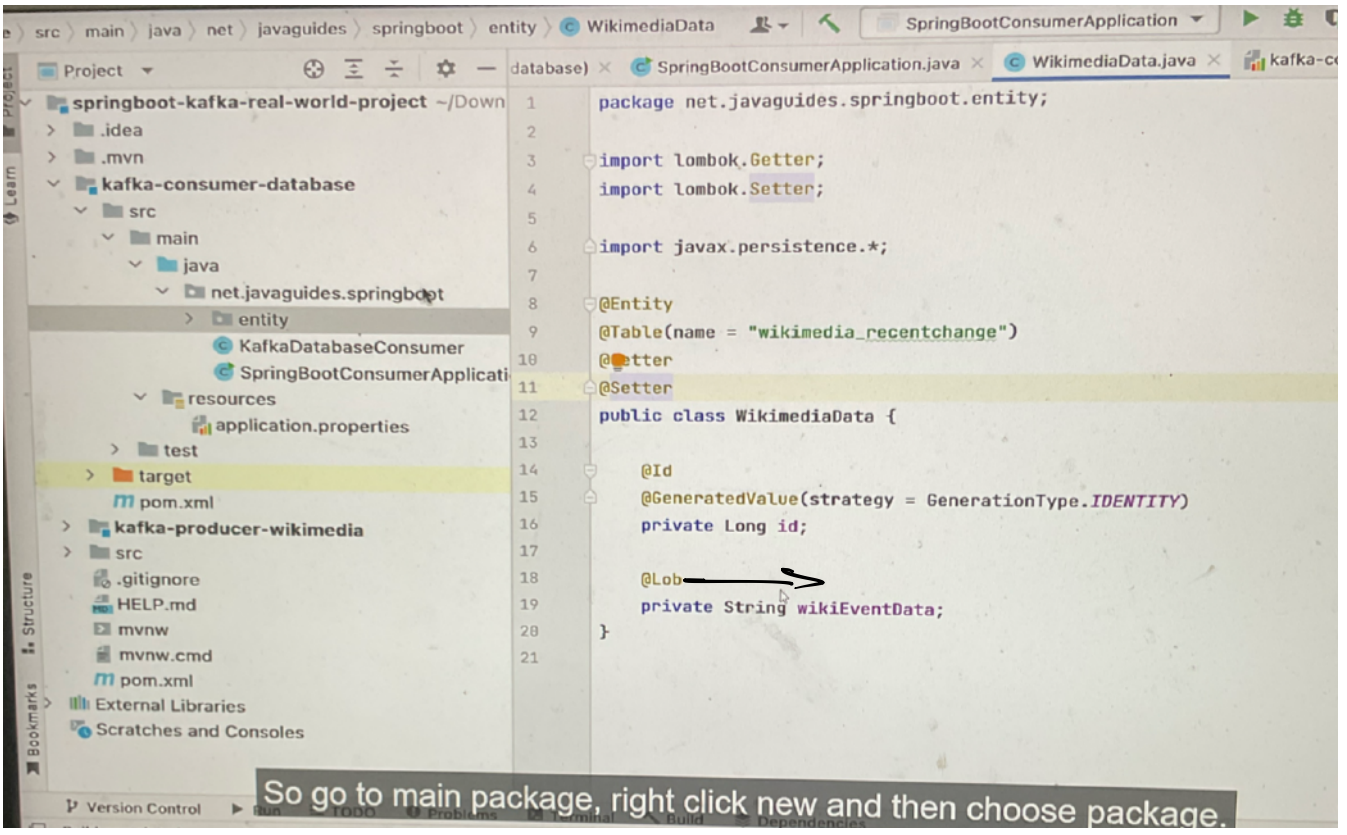
Create Sub module :

Created on our own. we need to Create



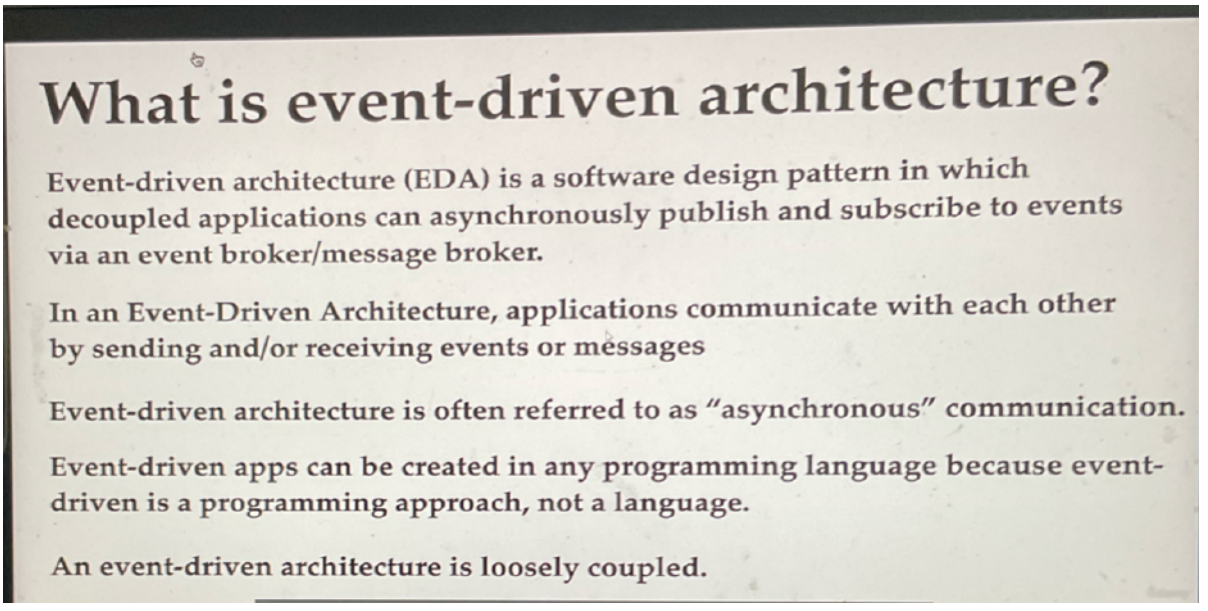

springboot-kafka-real-world-project
.idea
.mvn
kafka-producer-wikimedia
src
target
pom.xml
src
.gitignore
HELP.md
mvnw
mvnw.cmd
pom.xml
External Libraries
Scratches and Consoles
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.
<parent>
<artifactId>springboot-kafka-real-world-project</artifactId>
<groupId>net.javaguides</groupId>
<version>0.0.1-SNAPSHOT</version>
</parent>
<modelVersion>4.0.0</modelVersion>
<packaging>jar</packaging>
<artifactId>kafka-producer-wikimedia</artifactId>
<properties>
<maven.compiler.source>17</maven.compiler.source>
<maven.compiler.target>17</maven.compiler.target>
</properties>
</project>
So let's do a packaging as a jar for this model and save it and click on this load my engineers and
project › modelVersion

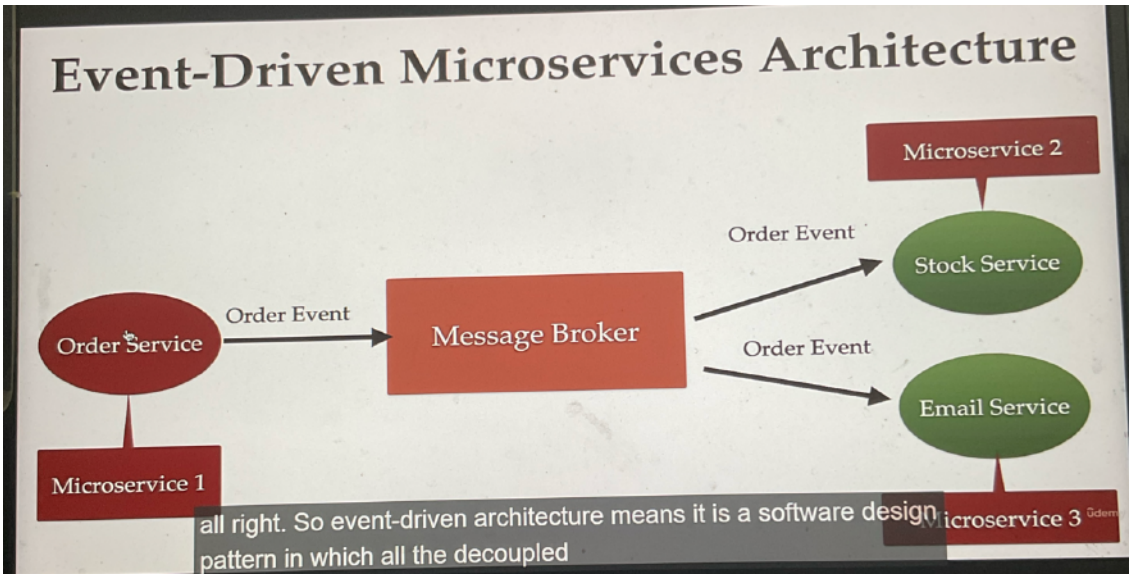
⇒ check git



In order to accept large amount of data. we use @Lob

[action] Event Driven Architecture





whenever customer Place an Order, the order Service will basically Create an order event and it will Publish that Order event to the message broker and Stock Servic

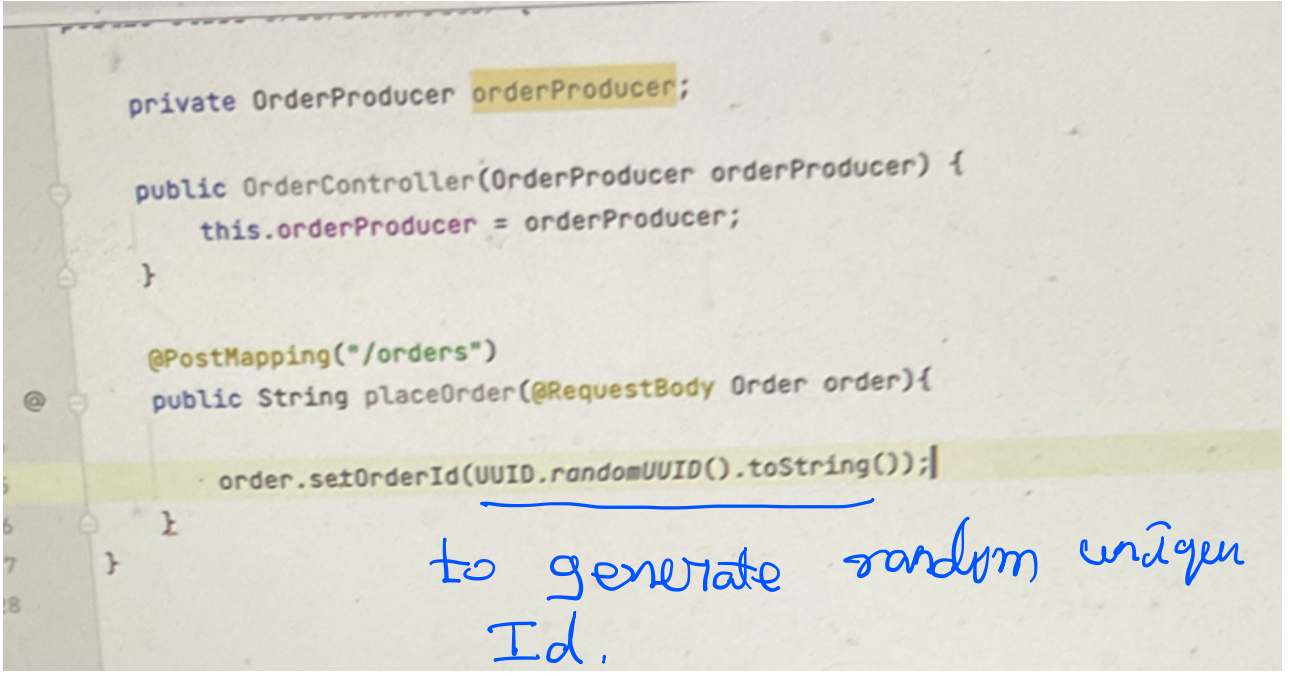
and email service. So there are the basically consumers who subscribed to this message broker. So as soon as order service Published order event to the message broker, then two microservice will receiver that order

Stock service: responsible for receiver the order event from message broker. and it will update the order details in the database.

Email Service: responsible for receiver order and send mail to Customer.

Advantages

* Improved flexibility and maintainability.
* High Scalability.
* Improved Availability.


private OrderProducer orderProducer;
public OrderController(OrderProducer orderProducer) {
this.orderProducer = orderProducer;
}
@PostMapping("/orders")
public String placeOrder(@RequestBody Order order){
order.setOrderId(UUID.randomUUID().toString());
}
}
to generate random unique Id.