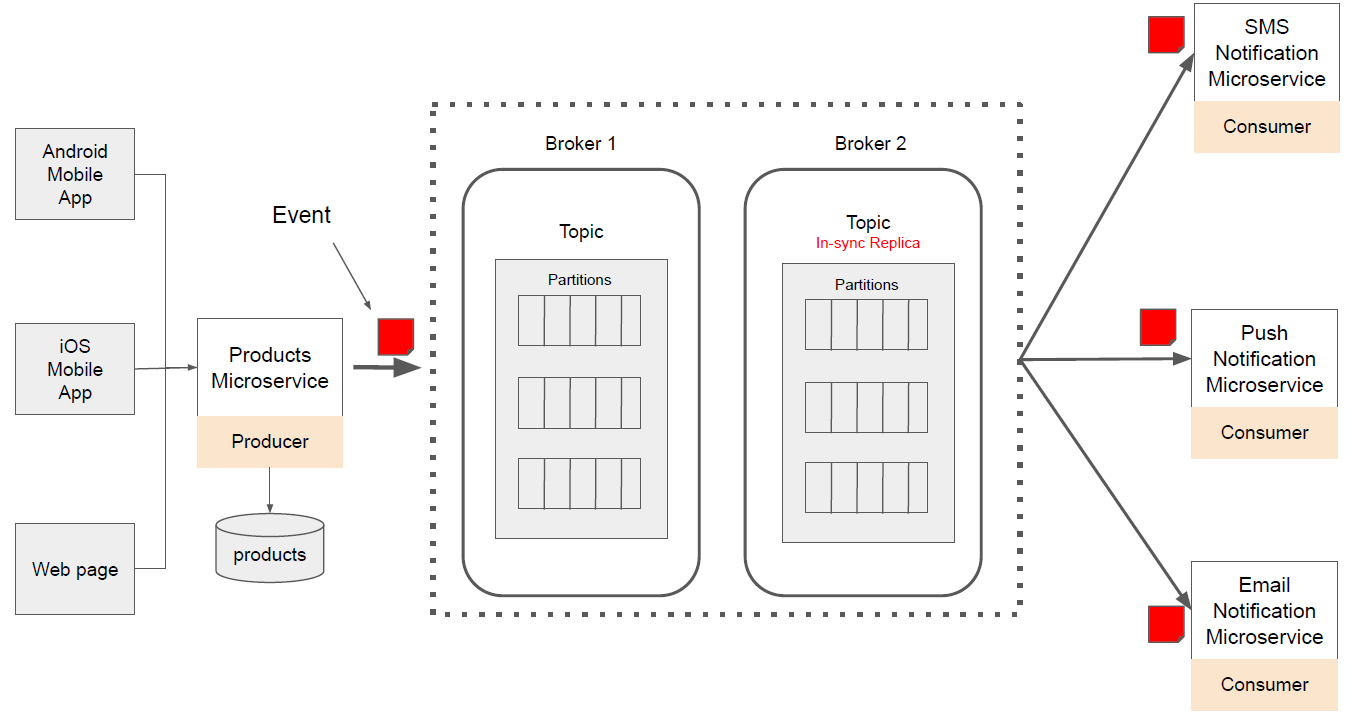
Kafka

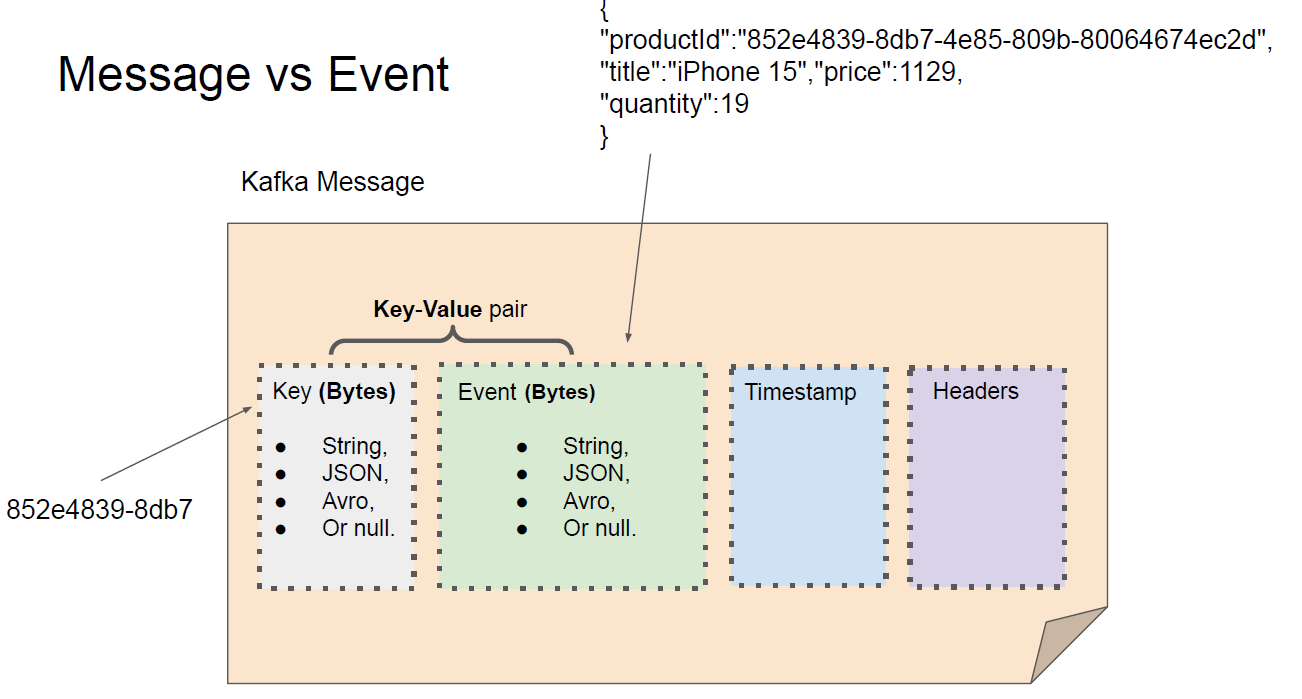
Kafka, also known as **Apache Kafka**, is an open-source, distributed event streaming platform designed to handle real-time data feeds. It is widely used for building real-time data pipelines and streaming applications.

**Section :1**

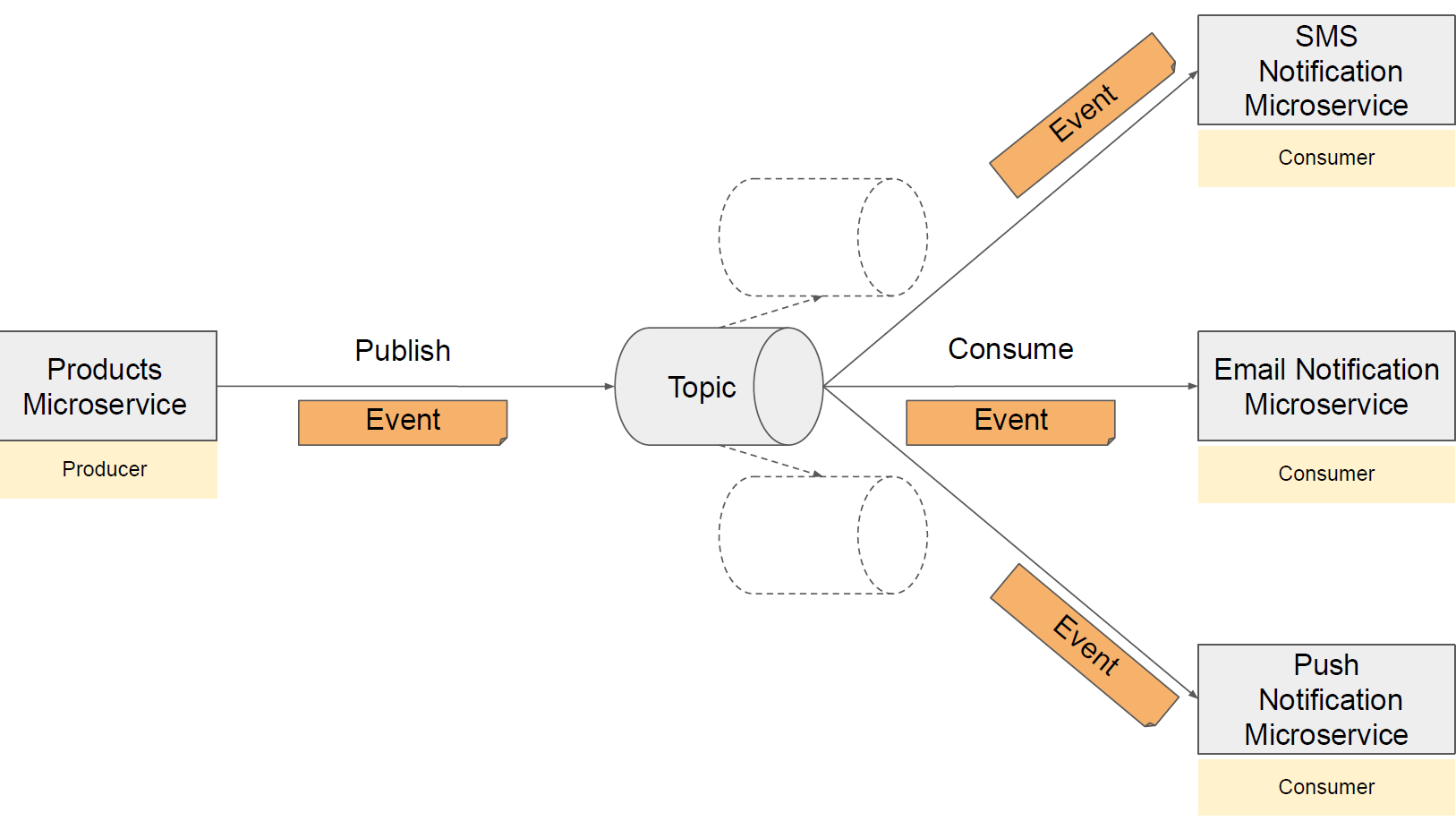
1. **Messages and Events**

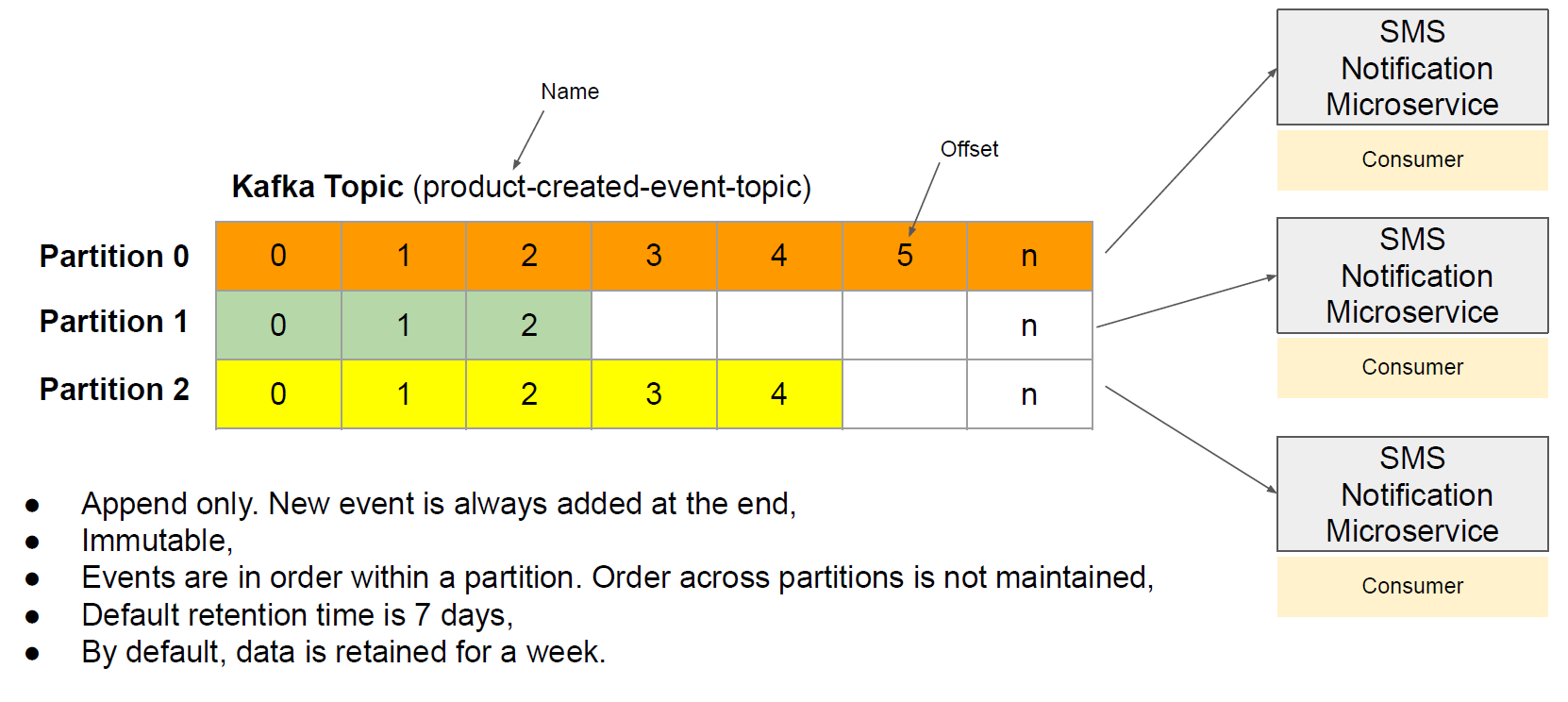
****

What is an Event? In Apache Kafka, an event is an indication that something has happened. ● UserLoggedInEvent ● ProductCreatedEvent, ● OrderCreatedEvent.

****

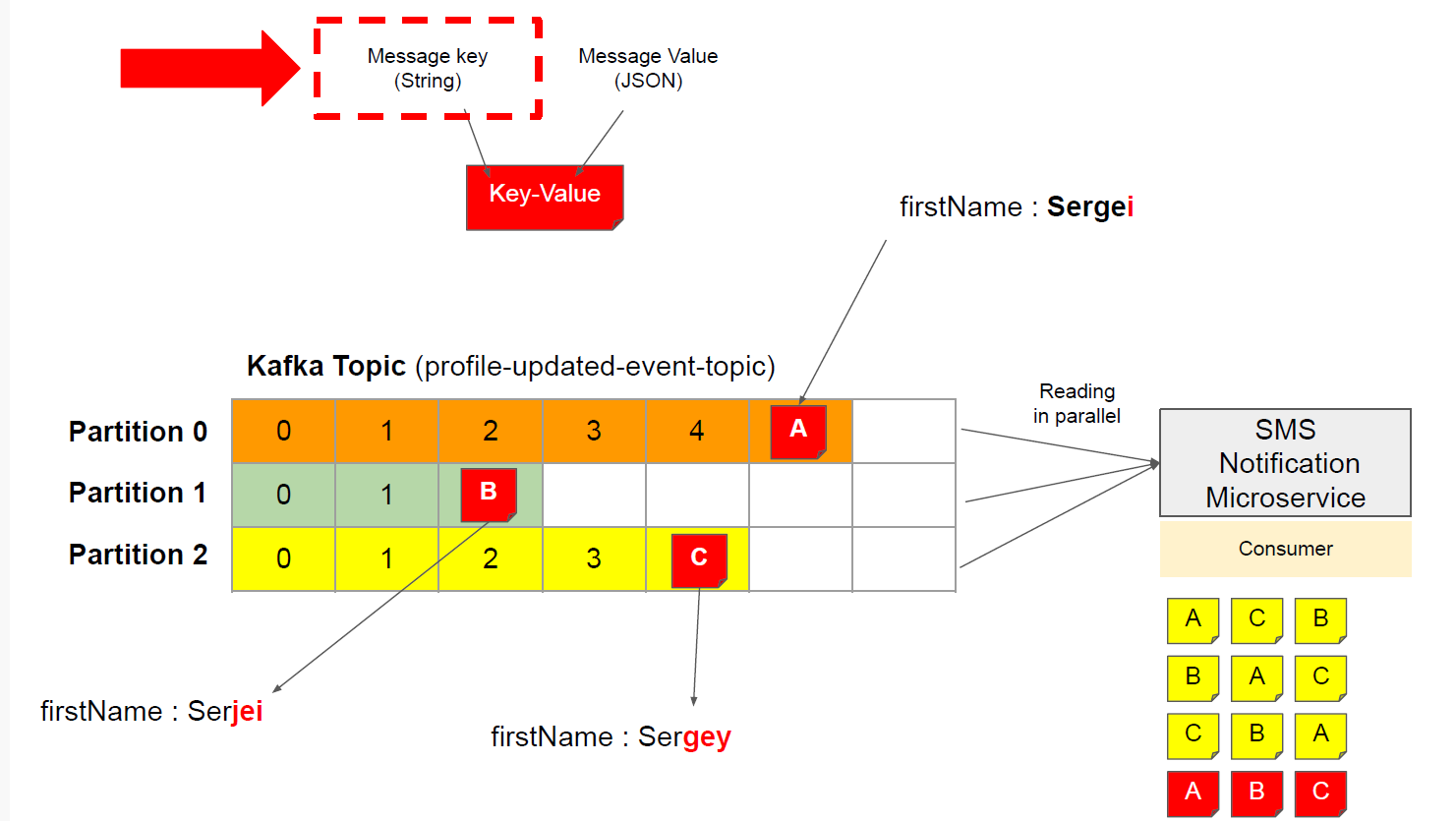
1. **Kafka Topic**

****

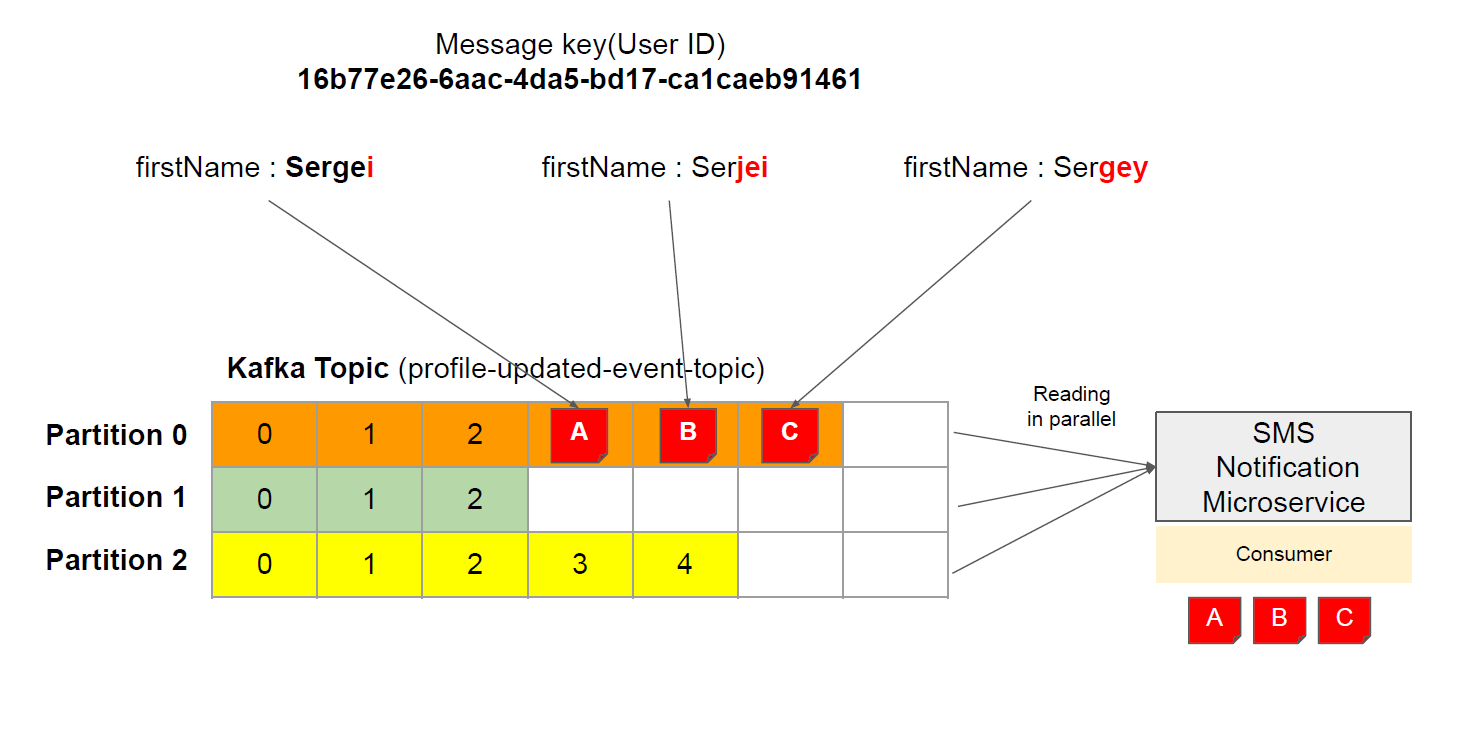


**Note** : topic is logical entity where as partion actually exist.

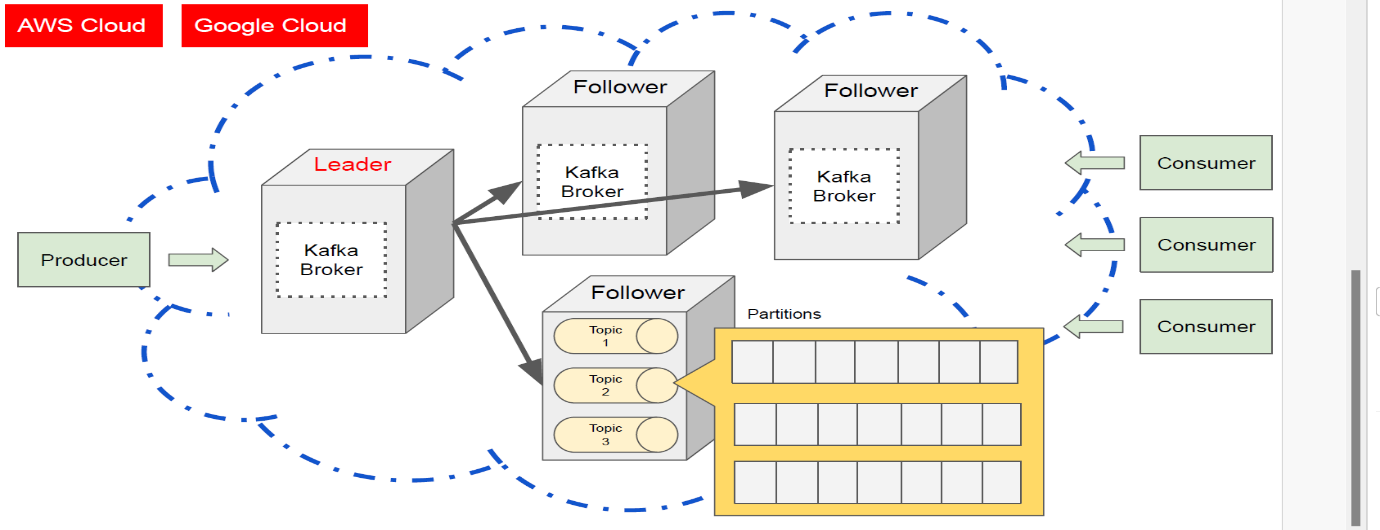
1. **Events Ordering**
2. If key is not present in kafka msg -> order is consumption in not gurrantte



1. If key is present in kafka msg -> order of consumption is gurrantted

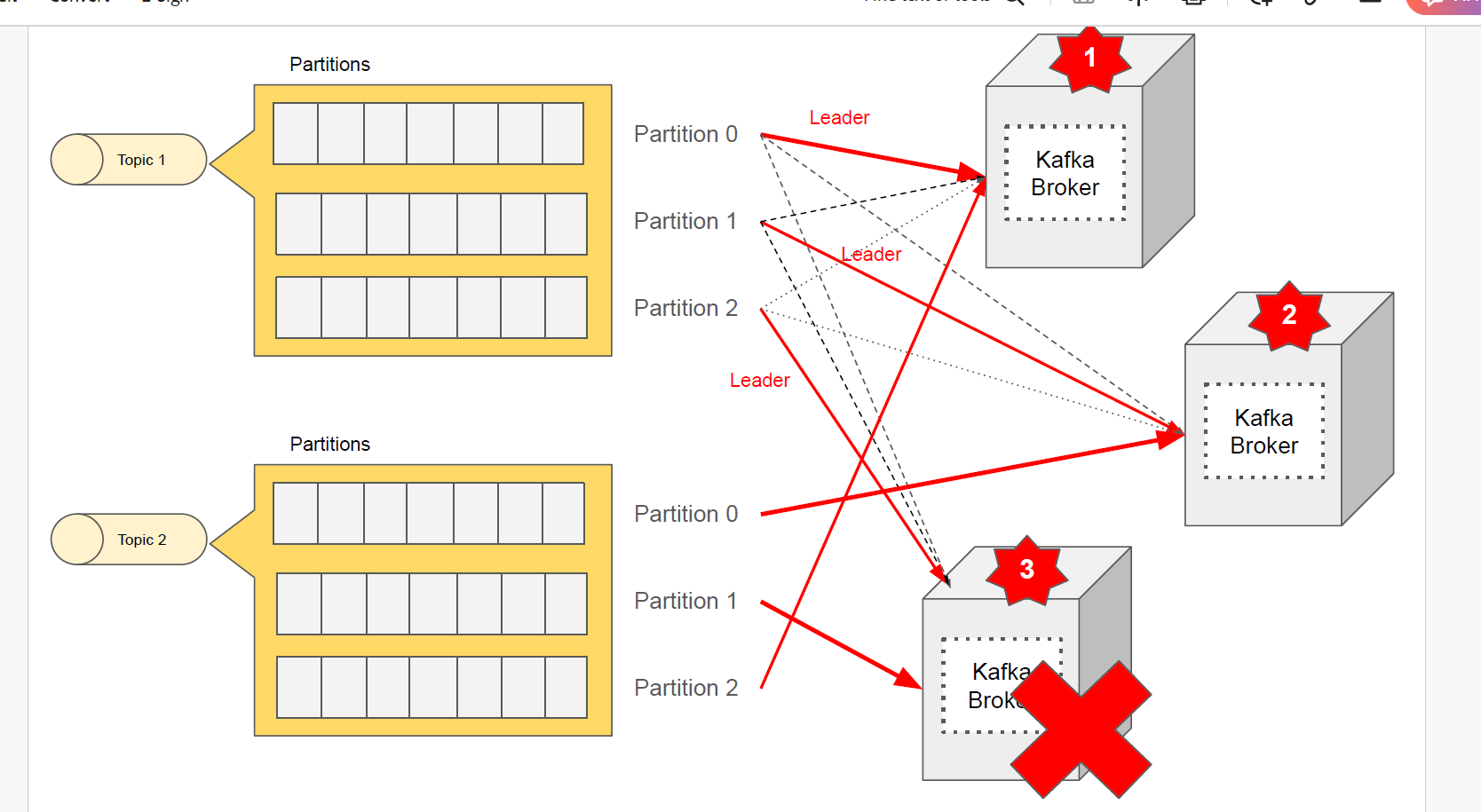
****

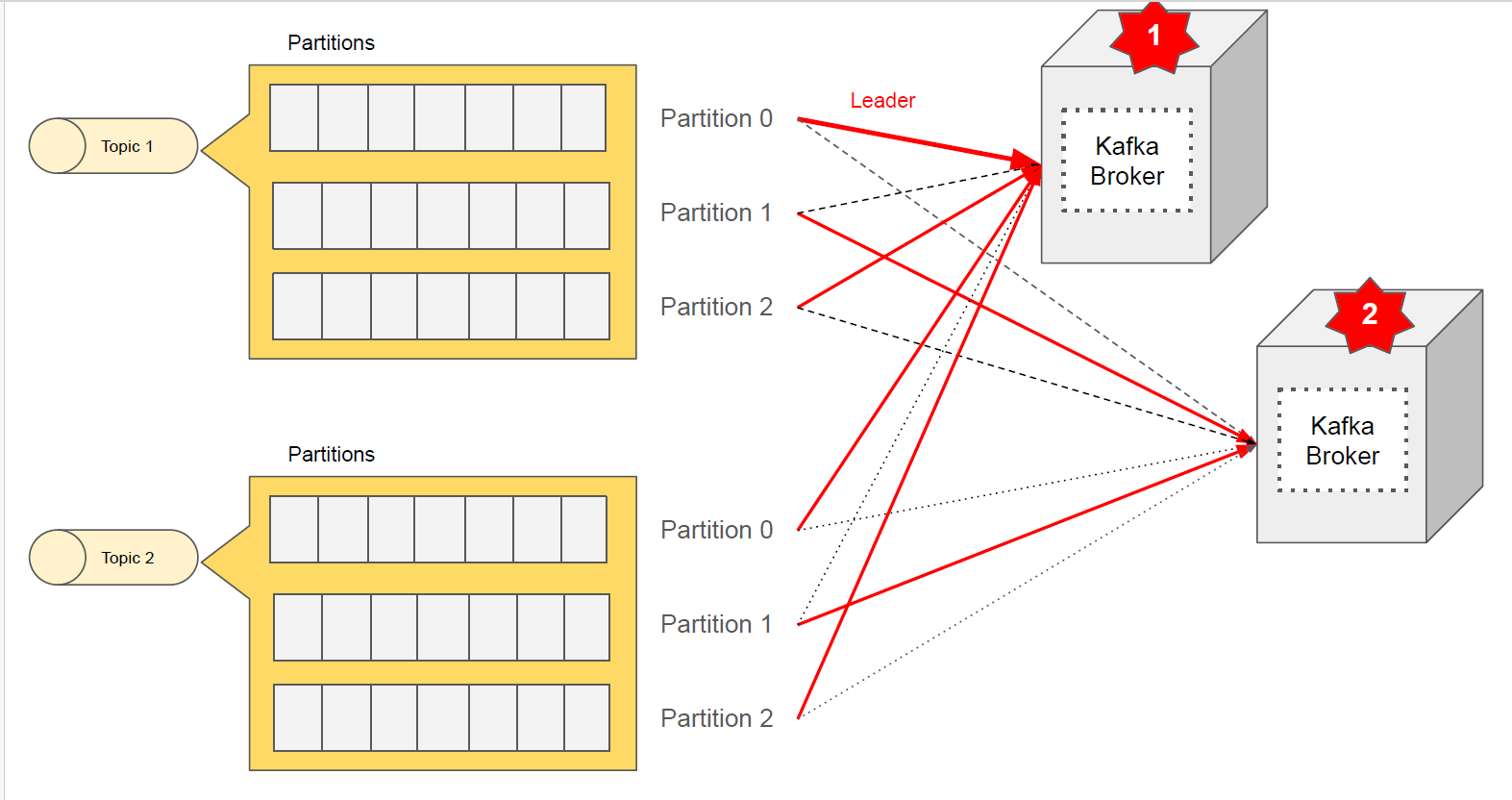
**Kafka broker**

**Section :2**

****

**In kafka leadership getting shifted or getting when required and hance it avoids bottleneck for the same**

**Image shows each partion working with other broker as leader**

**it gives durability due to replicating msg across broker**

**when broker is lost leadership managed by kafka**

**2 ways to start apache kafka**

1. **Zookeeper**
2. **Kraft -> kafka raft -> it s protocol**

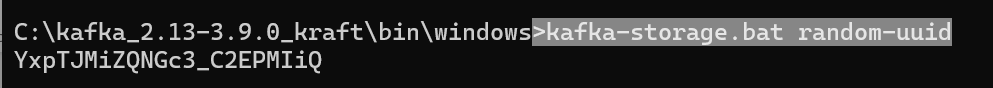
**Kraft**

**C:\kafka\_2.13-3.9.0\_kraft\config\kraft**

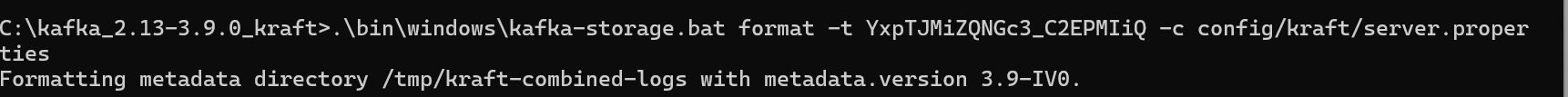
**3 properties file:**

1. **Broker.properties ->contains configuration for server which acts as broker which is responsible storing and serving data from partitions.**
2. **Controller.properties-. contains data for server that acts as server which is responsible managing cluster metadata and co-ordinating leader election for partitions**
3. **Server.properties -> stores data for both controller and broker**

**Step 1: Generate a Cluster UUID**

****

**Step 2: Format Log Directories**

****

**Step 3: Start the Kafka Server**

****

**What are log directories?**

**Log directories are the folders where Kafka stores its data as ordered sequences of messages called logs. Each log is divided into smaller files called segments, which contain the actual messages and some index files to help locate them. Log directories are organized by topics and partitions, which are logical groups of logs that belong to a specific use case and provide parallelism and fault tolerance.**

**Why do we need to format log directories?**

**We need to format log directories before we can start Kafka in KRaft mode. Formatting log directories means assigning a unique identifier to the cluster and initializing the metadata for the partitions. This prevents accidental configuration changes and ensures compatibility with KRaft mode.**

**What is server.properties file?**

**The server.properties is a configuration file that specifies various settings for the Kafka server, such as the broker id, the listeners, the log directories, the number of controllers, the ports, and the log retention policies. You can modify this file to change the default values according to your needs and preferences. You can also override some of the properties by passing them as command-line arguments when starting the Kafka server.**

**Step 4: Test the Kafka Server**

**# Create a topic named product-created-events-topic with one partition and one replica**

bin/kafka-topics.sh --create --topic product-created-events-topic --partitions 3 --replication-factor 1 --bootstrap-server localhost:9092

**# Produce some messages to the topic**

bin/kafka-console-producer.sh --topic product-created-events-topic --bootstrap-server localhost:9092

**# Consume the messages from the topic**

bin/kafka-console-consumer.sh --topic product-created-events-topic --from-beginning --bootstrap-server localhost:9092

Starting more than 1 kafka server:

1. Need to have 3 server.properties

Need to change below property to unique data

**a)node.id**

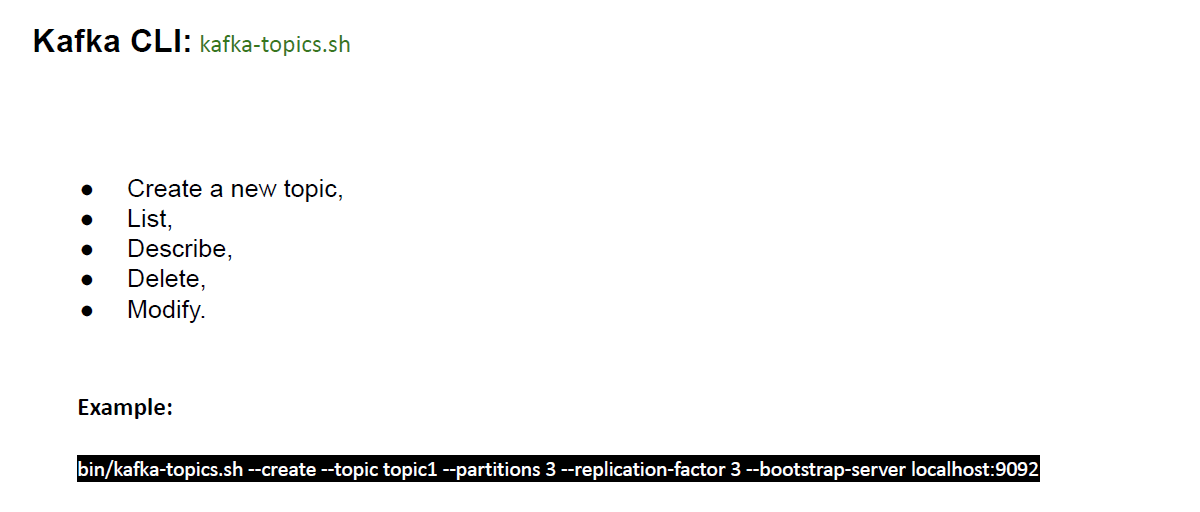
**b)** **listeners :pair of ports**

**b) advertised.listener :**

**c) controller.quorum.voters**

**e)** **log.dirs -> to maintain logs for each broker**

**Kafka Cli Topics**

**Section 3:**

**bin/kafka-topics.sh --create --topic topic1 --partitions 3 --replication-factor 3 --bootstrap-server localhost:9092**

**What is an Apache Kafka Topic?**

**A Kafka Topic is a logical collection of messages that are produced and**

**consumed by Kafka applications.**

**A Kafka Topic has a unique name that identifies it in a Kafka cluster. A Kafka cluster is a group of servers, called brokers, that store and manage the topics and messages. A Kafka application can connect to a Kafka cluster and interact with the topics and messages.**

**What is Topic Partition?**

**A Kafka Topic is divided into one or more partitions, each of which stores a subset of messages in an ordered sequence. A partition is a physical unit of storage and processing in a Kafka broker. Each partition has a unique identifier, called a partition ID, that is assigned by the broker.**

**The number of partitions affects the scalability, parallelism, fault-tolerance, and ordering guarantees of a topic.**

**● Scalability: The more partitions a topic has, the more messages it can store and handle. A topic with many partitions can be distributed across multiple brokers, which increases the storage capacity and performance of the topic.**

**● Parallelism: The more partitions a topic has, the more producers and consumers can interact with it concurrently. A topic with many partitions can have multiple producers sending messages to ifferent partitions, and multiple consumers receiving messages**

**from different partitions. This increases the throughput and efficiency of the topic.**

**● Fault-tolerance: The more partitions a topic has, the more resilient it is to failures. A topic with many partitions can have replicas, which are copies of the partitions stored on different brokers. If a broker fails, the replicas can take over and continue serving the messages. This increases the availability and reliability of the topic.**

**● Ordering guarantees: The more partitions a topic has, the less strict the ordering guarantees are. A topic with many partitions can only guarantee the order of messages within each partition, but not across partitions. This means that messages from different partitions may be delivered out of order to the consumers. This may or may not be acceptable, depending on the use case of the topic.**

**How to create topic ?**

**To create a topic, we need to specify the following parameters:**

**● --bootstrap-server: The address of one or more brokers in the**

**Kafka cluster that we want to connect to. For example,**

**localhost:9092.**

**● --create: The flag that indicates that we want to create a topic.**

**● --topic: The name of the topic that we want to create. For**

**example, product-created-events-topic.**

**● --partitions: The number of partitions that we want to create**

**for the topic. For example, 3.**

**● --replication-factor: The number of replicas that we want to**

**create for each partition of the topic. For example, 2.**

**For example, the following command creates a topic named**

**product-created-events-topic with 3 partitions and 2 replicas in a**

**Kafka cluster with a broker at localhost:9092:**

kafka-topics.sh --bootstrap-server localhost:9092 --create --topic

product-created-events-topic --partitions 3 --replication-factor 2

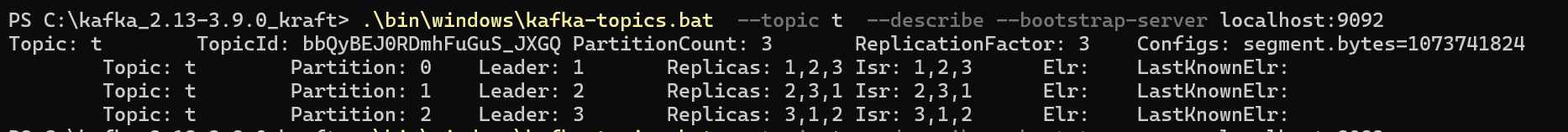
If the topic is created successfully, the tool will print a confirmation message, such as:

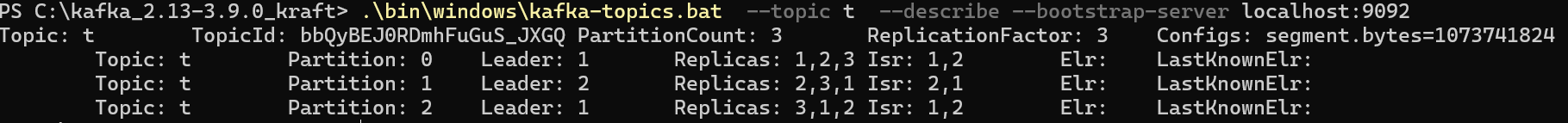
Created topic product-created-events-topic.

If the topic already exists, or if there is an error, the tool will print an error message, such as:

Topic 'product-created-events-topic' already exists.

Case : 3 brokers up : 3 partition: 3 replication



I shutdown my 1 broker that is server 3 after leader will be changes  


…………………………………………………..XXXXXXXXXXXXXXXXXXXX……………………………………..

**How to create topic with partitions?**

kafka-topics.sh --bootstrap-server localhost:9092 --create –topic topic1 --**partitions** 3

………………………………………………..XXXXXXXXXXXXXXXXXXXXXXX…………………………………

**How to update topic?**

kafka-topics.sh --bootstrap-server localhost:9092 --**alter** –topic topic1 --config retention.ms=43200000

**updating topic partition:**

kafka-topics.sh --bootstrap-server localhost:9092 --**alter** –topic topic1 --partitions 6

**updating replication factor:**

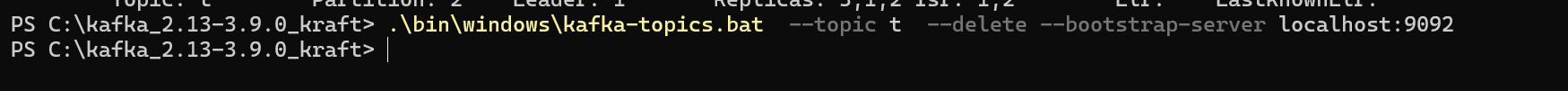
kafka-topics.sh --bootstrap-server localhost:9092 --**alter** –topic topic1 --replication-factor 3

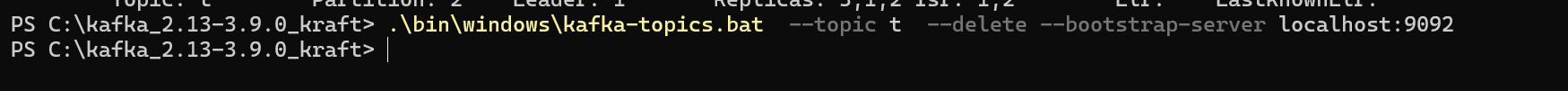
**Note**: Note that you can only increase the replication factor for a topic, but not decrease it. This is because reducing the replication factor would cause data loss and inconsistency. If you want to reduce the replication factor for a topic, you have to delete the topic and recreate it with the desiredreplication factor.

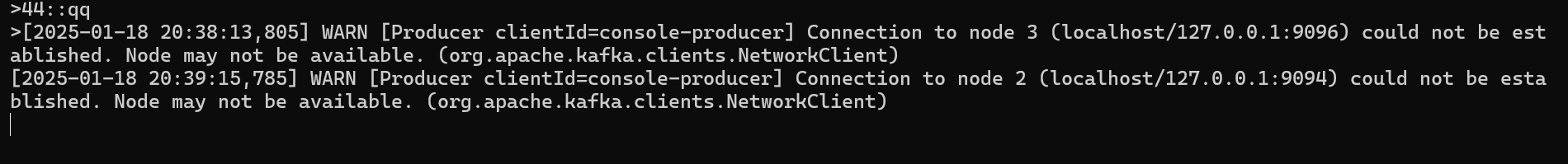
………………………………………………..XXXXXXXXXXXXXXXXXXXXXXX…………………………………

**How to delete topic?**

kafka-topics.sh --bootstrap-server localhost:9092 --delete –topic topic1

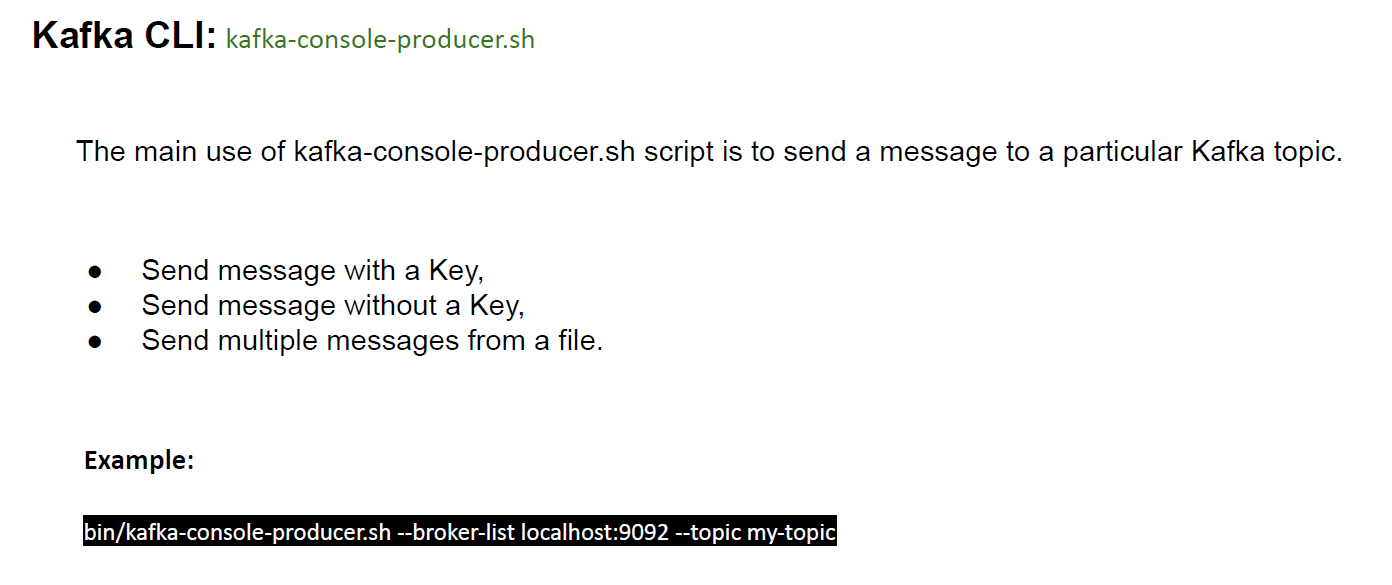
  
after deleiting producer and consumer leastening to this topis will start throwing exception:





**Kafka Producer**

Section 4



**How to produce messages to Kafka topics without a key?**

Key insures sequence gurantte of msg consumption

./kafka-console-producer.sh --bootstrap-server localhost:9092 --topic topic1

**To consume msg from consumer:**

./kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic topic --from-beginning

………………………………………….XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX…………………………………………….

**How to produce messages to Kafka topics with a key**

./kafka-console-producer.sh --bootstrap-server localhost:9092 --topic topic1 **--property "parse.key=true"**

**--property "key.separator=:"**

**To consume msg from consumer:**

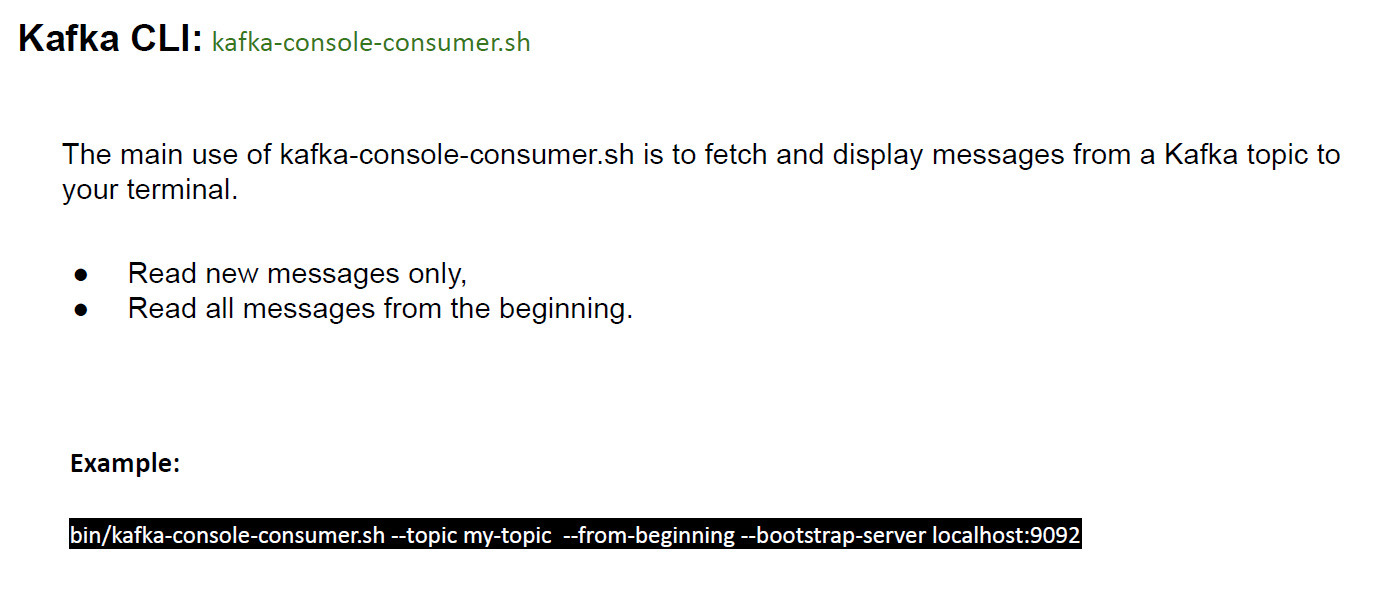
./kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic topic1 --from-beginning **--property**

**"print.key=true" --property "key.separator=:"**

**Note :**The print.key=true property tells the consumer to print both the key and the value of each message, separated by the key.separator character.

**Kafka Consumer**

**Section 5:**

****

We have 2 different properties for each consumer and producer as well  
For Producer:

**--property "parse.key=true"**

**--property "key.separator=:"**

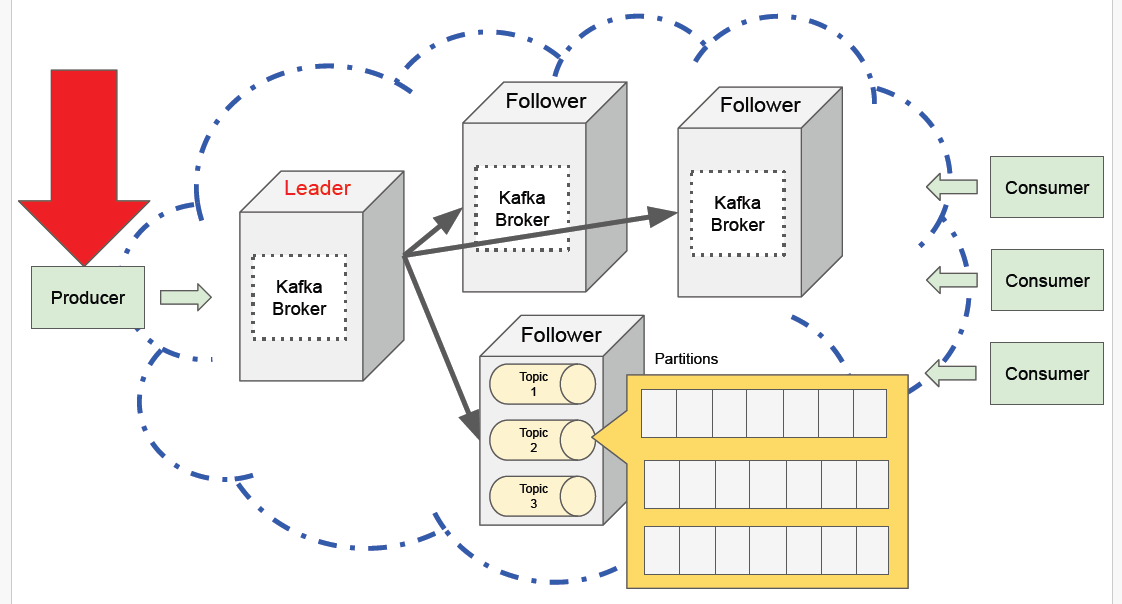
For consumer:

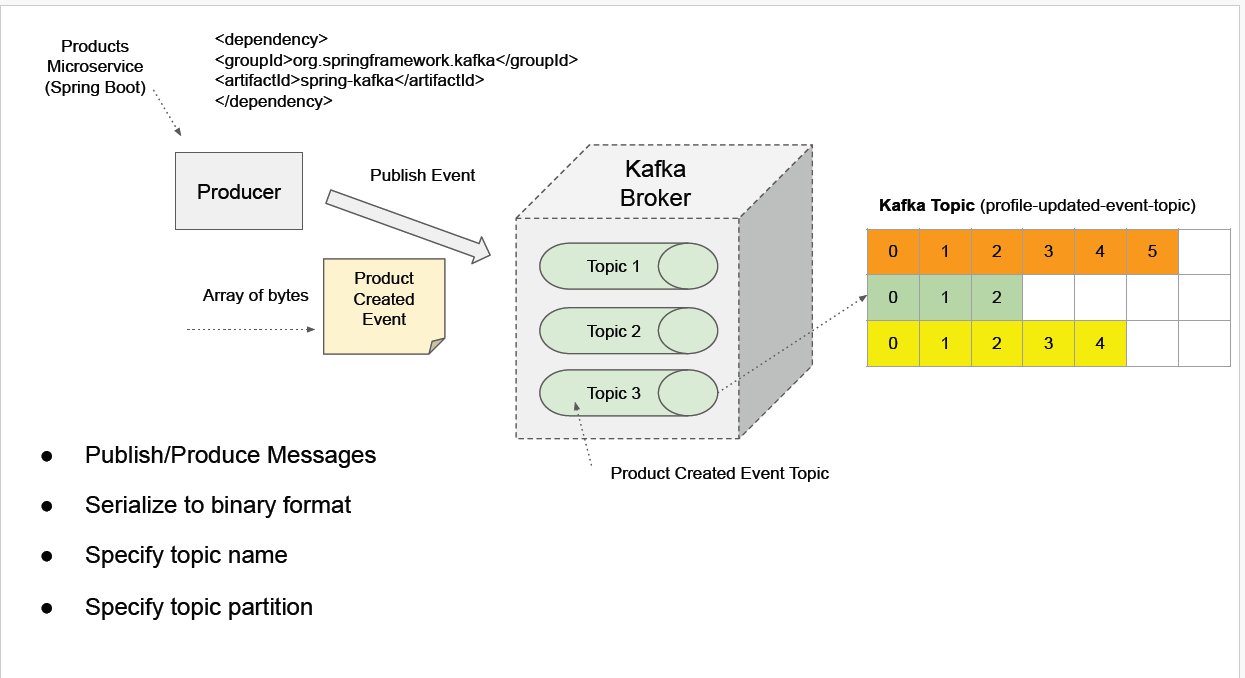
**--property print.key=true**

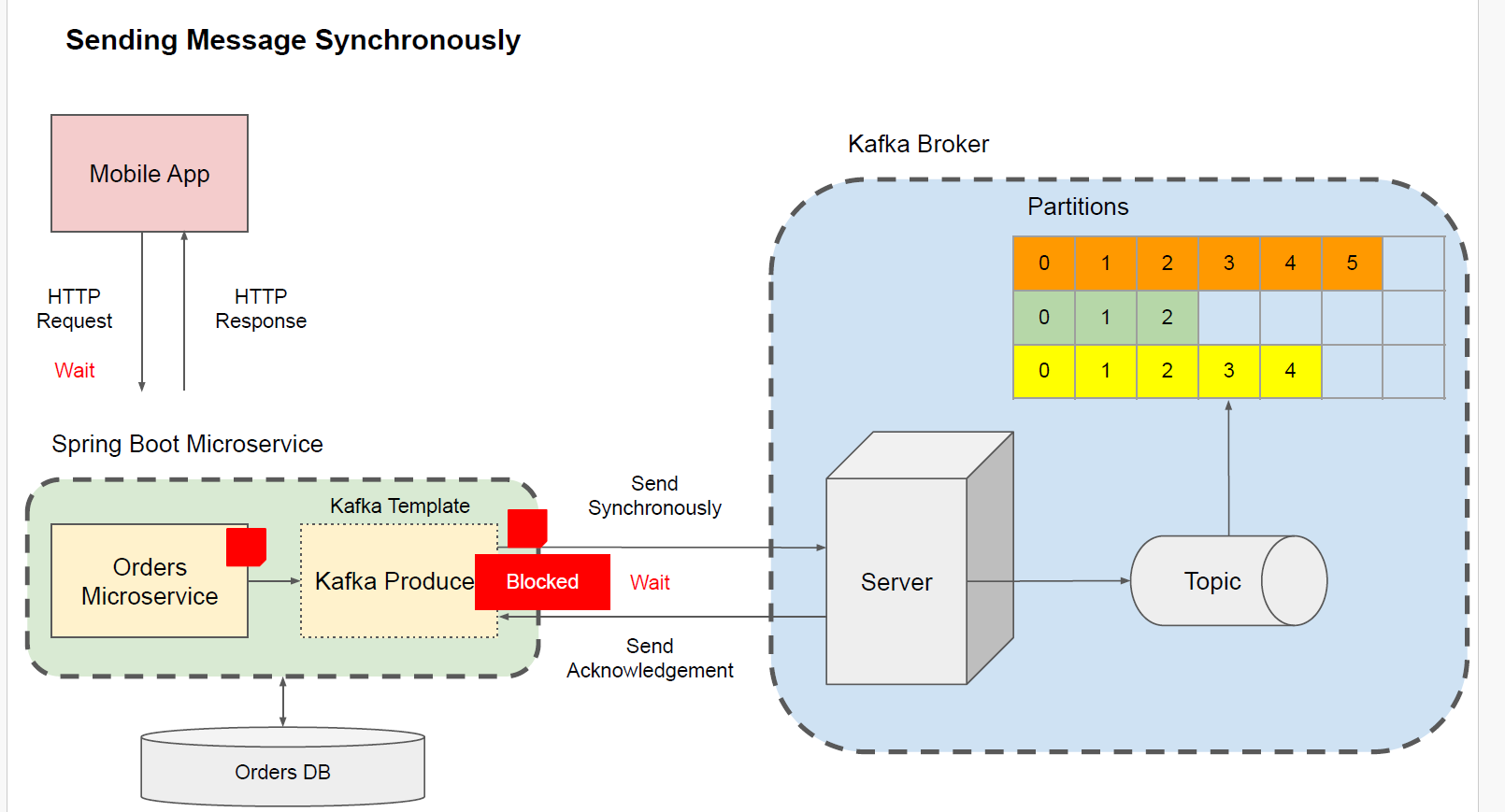
**--property print.value=true //default is true**

**--property key.separator=::**

**--from-beginning**

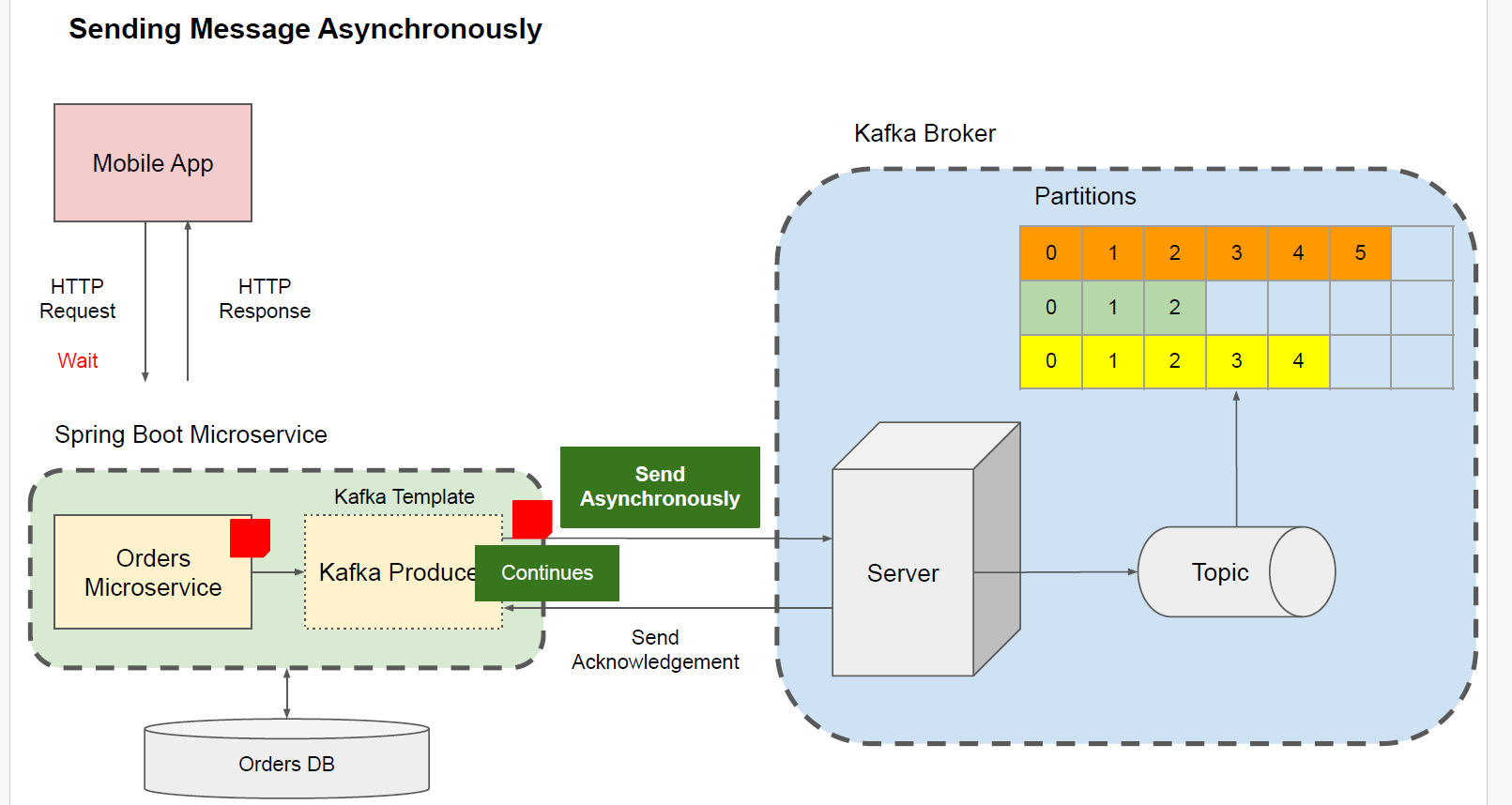
Spring Boot project : Doc **Kafka Producer**



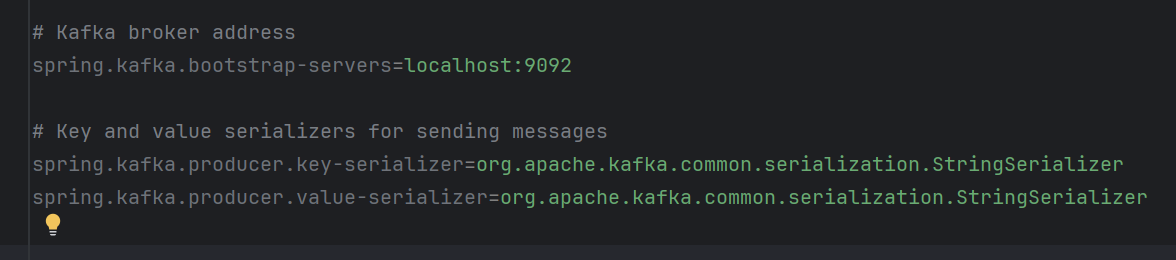
**Synchronous Communication Style**

\

**ASynchronous Communication Style**

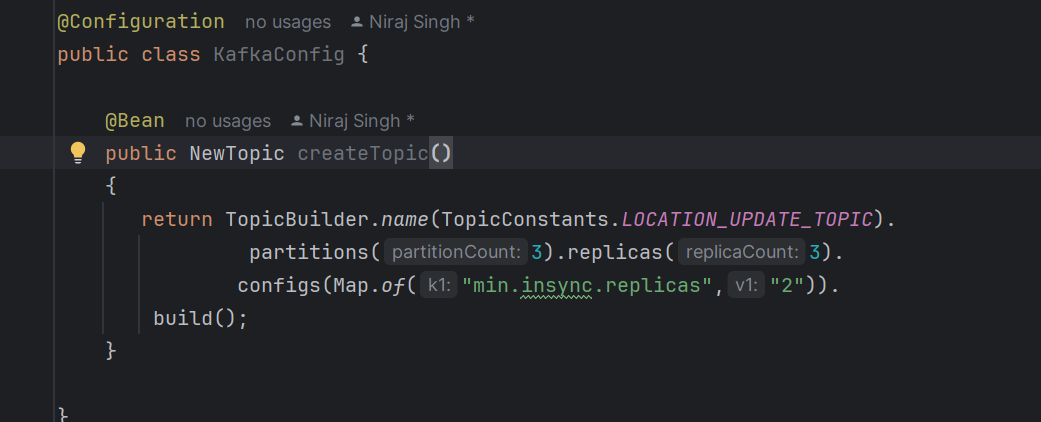


Producer Spring boot application

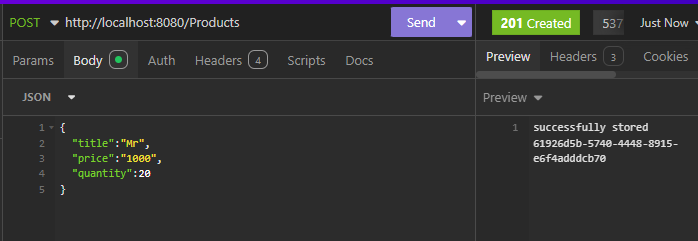
Application properties:  


1. KafkaConfig.java -> to create topic when spring boot application get up  
   @Configuratoin 🡪 class level

@Bean 🡪 Method level



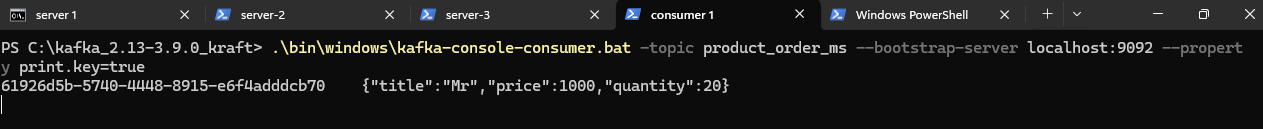
Producer:



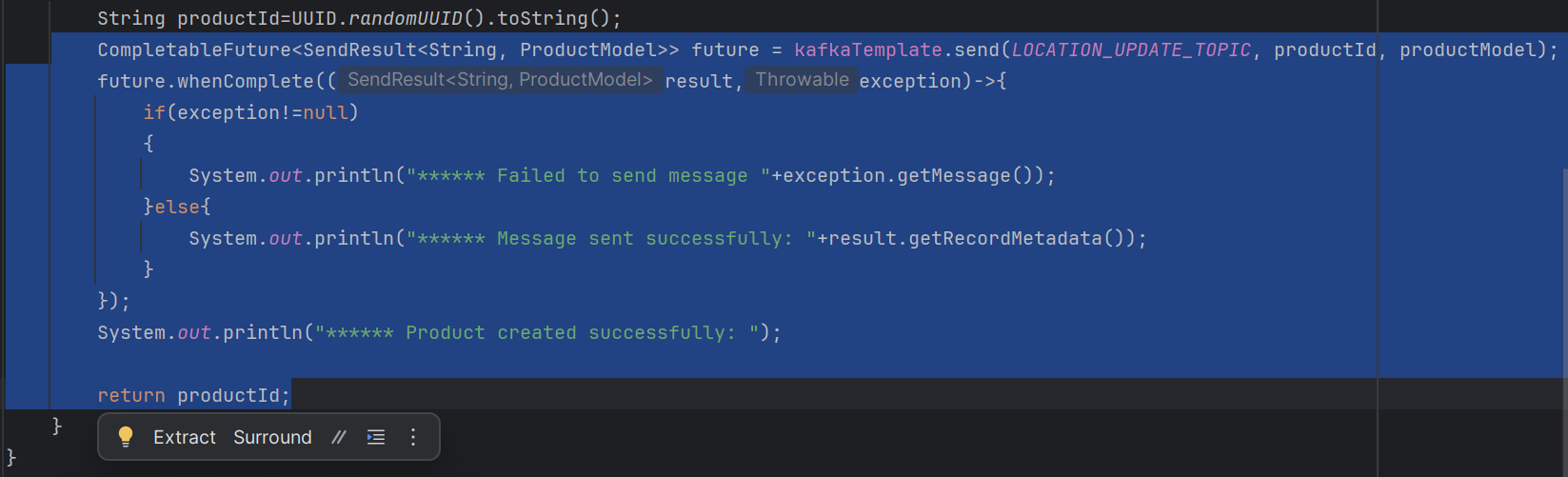
Producer produced msg on partion 0



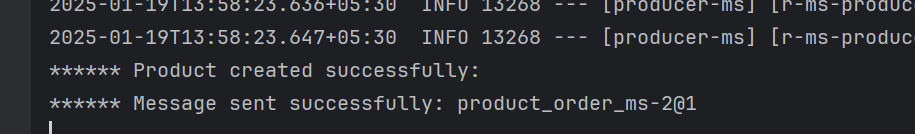
Consumer received msg which was produced by spring (here we have 3 brokers running)



**Asynchronous: As per code**



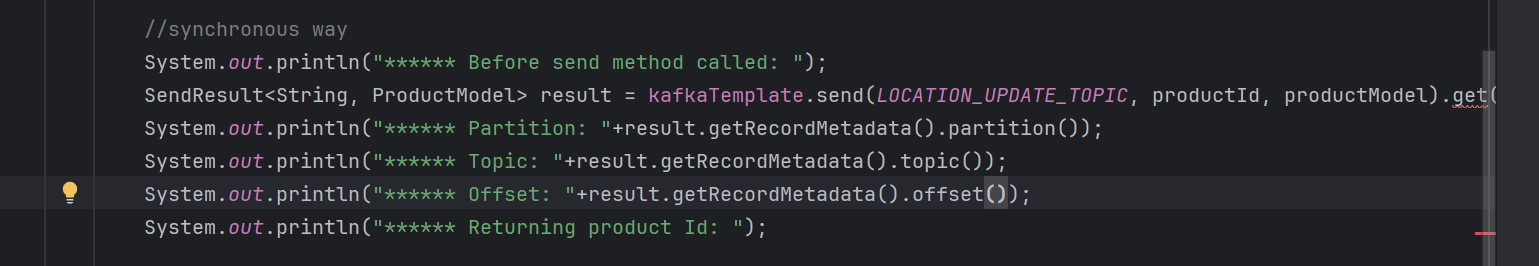
After executing order of print statement proves that it is working asynchronously.(order of printing varied from actually code written as kafka response delayed)

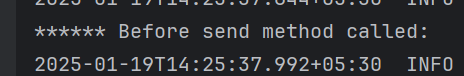


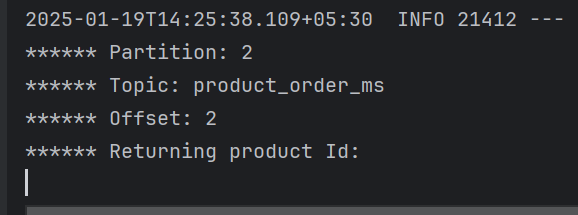
**Synchronous->**

**1)second approach**

**.get()**

****

****

****

**Kafka Producer Acknowledgement and retries**

**Section 7:**

Important points:

1. **listeners** is how Kafka brokers know where to wait for connections, and **advertised.listeners** is how clients know where to find the broker.
2. **No of partition should be >= no of consumer (other wise no sense)**
3. **Replication factor should not be greater than no of brokers.**
4. **Message key ensures sequence of deliver of msg as it get stores in same partion.**
5. **Kafka manager or kafka cruise ui based project control n monitor kafka brokers and all**cloned the gradle project :

**Started it : .\gradlew.bat build**