

# APACHE KAFKA

Apache Kafka, yüksek veri akış hızlarıyla büyük ölçekte gerçekleşen veri iletimi için kullanılan bir dağıtılmış akış işleme platformudur. Genellikle büyük veri işleme, veri entegrasyonu, gerçek zamanlı veri akışı ve olay günlüğü yönetimi gibi senaryolarda kullanılır. Kafka'nın temel amacı, verileri farklı uygulamalar, sistemler veya bileşenler arasında güvenilir ve yüksek hızda iletmektir.

## Kafka Temel Bileşenler

- **Broker**

Kafka Cluster halinde çalışır ve yatay ölçeklenebilirdir.

Broker'lar da Kafka Clusterları oluşturan sunuculardır. Her Broker birer sayıdan oluşan ID ile tanımlıdır.

Kafka dağıtık bir yapıda olduğu için tek bir broker Topic'in tamamını alamaz, belirli partitionları alır.

- **Topic**

Topic, verilerin – mesajların gönderilip alındığı, kullanıcı tanımlı kategori ismidir.

İlişkisel veritabanlarındaki tablolara benzetebiliriz, bir Kafka Cluster içerisinde binlerce topic olabilir.

Bir topic oluştururken içerisindeki Partition ve Replication Factor sayısını belirtiriz.

Kafka bir **veritabanı değildir**, veriler sınırlı zamanda hataya dayanıklı bir şekilde saklanır, varsayılan süre **bir haftadır**.

- **Partition**

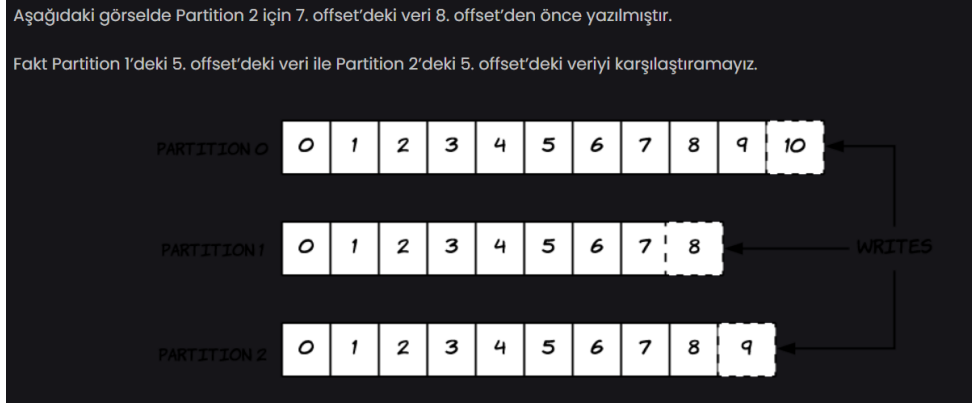
Tüm veriyi tek bir dosyadan okumaktansa farklı dosyalardan okumak hem daha performanslı hem de hataya karşı daha toleranslı.

Bu sebeple topicler bir veya birden fazla bölümden (Partition) meydana gelirler. Partition sayısına topic oluşturulurken karar verilir, sonradan ihtiyaca göre değiştirilebilir.

Mesajlar partition içerisine **sıralı** ve **değiştirilemez** olarak eklenirler. Artan şekilde bir id alırlar, buna **offset** denir.

Yani kafkaya gelen her mesajın partition numarası ve offset numarası oluşur.

Partition'lar **kendi içinde sıralıdır**.



Topiclerin bu partition özelliği sayesinde yazma ve okuma işlemleri paralel bir şekilde yapılabilir.

## Producer

Kafkaya mesaj gönderen uygulama

Burada mesaj: Bir tablo ya da metin dosyasının bir satırı olarak düşünülebilir.

Topice mesaj gönderirken key ve value göndermek mümkündür. Eğer key atanırsa, aynı keye sahip mesajlar aynı partitiona gönderilir.

Eğer key atanmazsa mesaj topic içerisinde rastgele bir partitiona eklenir.

## Consumer – Consumer Group

Topic'lerdeki veriyi okurlar.

Consumerlar aynı Partition içerisindeki mesajları sıralı bir şekilde okur.

Consumer'lar verileri **Consumer Group**lar halinde okur. Tek de olsa her consumer'ın bir grubu vardır. Kullanıcı tarafından grup oluşturulmazsa Kafka bunu otomatik oluşturur.

Bir Consumer Group birden fazla Partition'dan okuyabilir.

Bir Partition'ı ise bir Consumer Group içerisinde sadece bir consumer okuyabilir.

Gruptaki her consumer belli partition'lardan okuma yapar. Mesela topicte 3 partition varsa:

- **3 consumer durumu:** Her consumer birer partition'dan okuma yapar.
- **2 consumer durumu:** Biri tek partition'dan, biri kalan iki partition'dan okuma yapar.
- **4 consumer durumu:** Üçü birer partition'dan okuma yapar. Kalan deaktif olur.

Gruptaki bir consumer'ın çökmesi gibi durumlara karşı bazı durumlarda partition sayısından fazla consumer kullanılabilir.

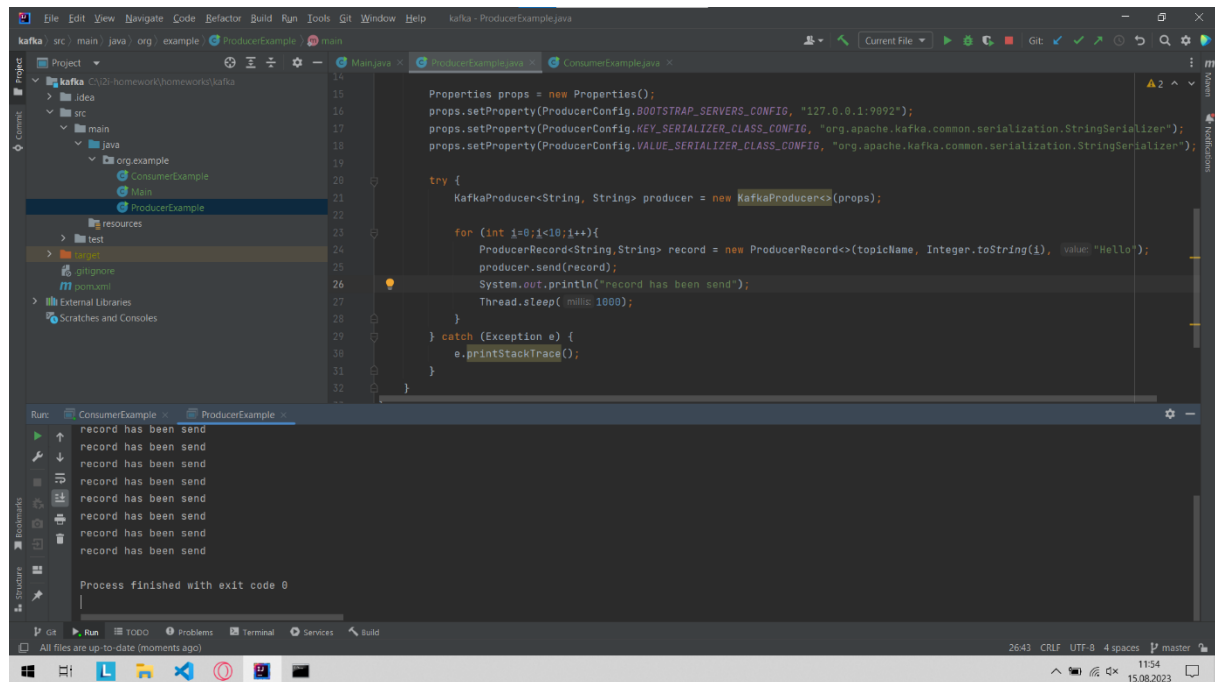
Bir mesaja erişebilmek için Topic, Partition, Offset bilgilerini bilmek gerekiyor.

Kendisine bırakılan mesajı olduğu gibi iletir, değiştirmez.

Mesaj okuma işleminden sonra kaybolmaz, tekrar erişim mümkündür.

## Kafka Practice

### Producer



```
File Edit View Navigate Code Refactor Build Run Tools Git Window Help kafka - ProducerExample.java
Project: kafka / src / main / java / org / example / ProducerExample.java
Project: kafka / src / main / java / org / example / ConsumerExample.java
14
15 Properties props = new Properties();
16 props.setProperty(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, "127.0.0.1:9092");
17 props.setProperty(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG, "org.apache.kafka.common.serialization.StringSerializer");
18 props.setProperty(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG, "org.apache.kafka.common.serialization.StringSerializer");
19
20 try {
21     KafkaProducer<String, String> producer = new KafkaProducer<>(props);
22
23     for (int i=0; i<10; i++){
24         ProducerRecord<String, String> record = new ProducerRecord<>(topicName, Integer.toString(i), "Hello");
25         producer.send(record);
26         System.out.println("record has been send");
27         Thread.sleep(1000);
28     }
29 } catch (Exception e) {
30     e.printStackTrace();
31 }
32 }
```

Run: ConsumerExample x ProducerExample x

```
record has been send
record has been send
record has been send
record has been send
record has been send
record has been send
record has been send
record has been send
record has been send
record has been send

Process finished with exit code 0
```

2643 CRLF UTF-8 4 spaces master 11:54 15.08.2023

# Consumer

```
20 props.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG, StringDeserializer.class.getName());
21
22 // Create kafka consumer
23 KafkaConsumer<String, String> consumer = new KafkaConsumer<>(props);
24
25 // Topic subscribe
26 consumer.subscribe(Collections.singletonList(topicName));
27
28 try {
29     while (true) {
30         ConsumerRecords<String, String> records = consumer.poll(1000);
31         for (ConsumerRecord<String, String> record : records) {
32             System.out.println(record.key() + ". received message: " + record.value());
33         }
34     }
35 } catch (Exception e) {
36     e.printStackTrace();
37 }
38 }
```

Run: ConsumerExample x ProducerExample x

```
0. received message: Hello
1. received message: Hello
2. received message: Hello
3. received message: Hello
4. received message: Hello
5. received message: Hello
6. received message: Hello
7. received message: Hello
8. received message: Hello
9. received message: Hello
```

20:103 CRLF UTF-8 4 spaces master

11:34 15.08.2023