



kafka

With



Spring Boot

First understand, what is kafka?

Apache Kafka is for streaming data in real-time!

This guide explains Kafka's main ideas, from when it started at **LinkedIn** to becoming a widely-used way to communicate worldwide.

The Origin of Kafka at LinkedIn: Addressing a Specific Need

Kafka started off at LinkedIn to tackle the big challenge of managing loads of activity data on their website. With all the clicks, system updates, and stats flying around, they needed a smart system to handle it all smoothly and in real-time.

So, Kafka was born! It was built to be tough, able to handle tons of messages at once without breaking a sweat. As it worked like a charm for LinkedIn, it became a big hit and got picked up by the Apache Software Foundation.

From there, it just kept growing, becoming a favorite tool for lots of different businesses thanks to its awesome streaming powers.

Let's take an example

Example of cab booking app

When a user goes to a cab booking app and books a cab, then a driver is assigned to that particular user. Now that user is getting constant location updates of the driver, so how do you think this type of communication is done?

There are two ways



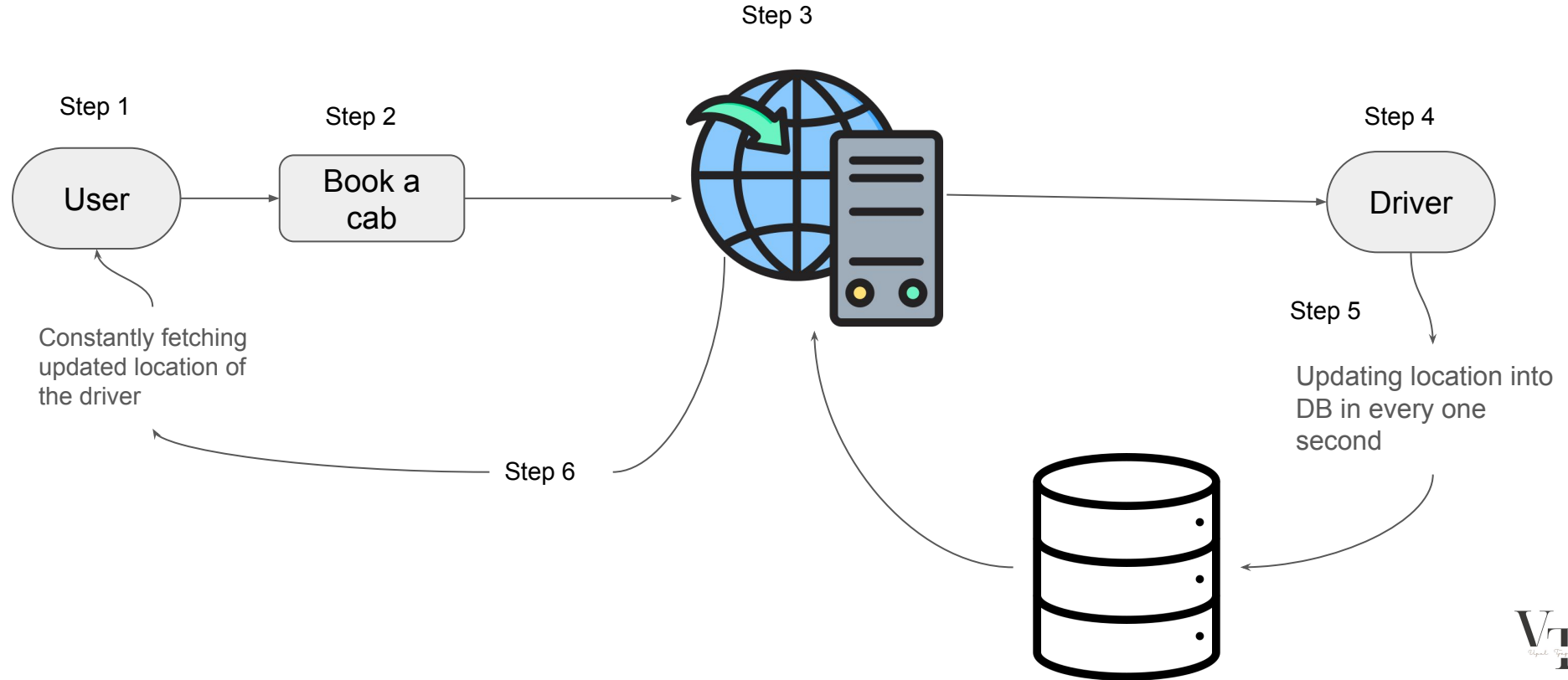
```
graph TD; A[There are two ways] --> B[Using DB to save the locations and then pass to User]; A --> C[Using Kafka to save the locations in a queue and then send it to the user]
```

**Using DB to save the locations
and then pass to User**

Using Kafka to save the locations in a
queue and then send it to the user

I believe question is clear

1. Let's see the DB approach first



After seeing the diagram everything seems fine and neat

Alright, let's put our detective hats on and snoop out the problem in this!

1. Now there is only one user and one driver, but their can be million of them.
2. Every second drivers are pushing their location into DB and every second users are picking the location from the DB → So **DB will not allow these humongous**

Operation

So System will crash....



2. Let's see the Kafka Approach

But, First understand kafka, because this doc is all about kafka

Kafka is an open-source distributed event streaming platform used for building real-time data pipelines and streaming applications

Kafka is designed to handle large volumes of data, allowing seamless communication between systems and applications in real-time.

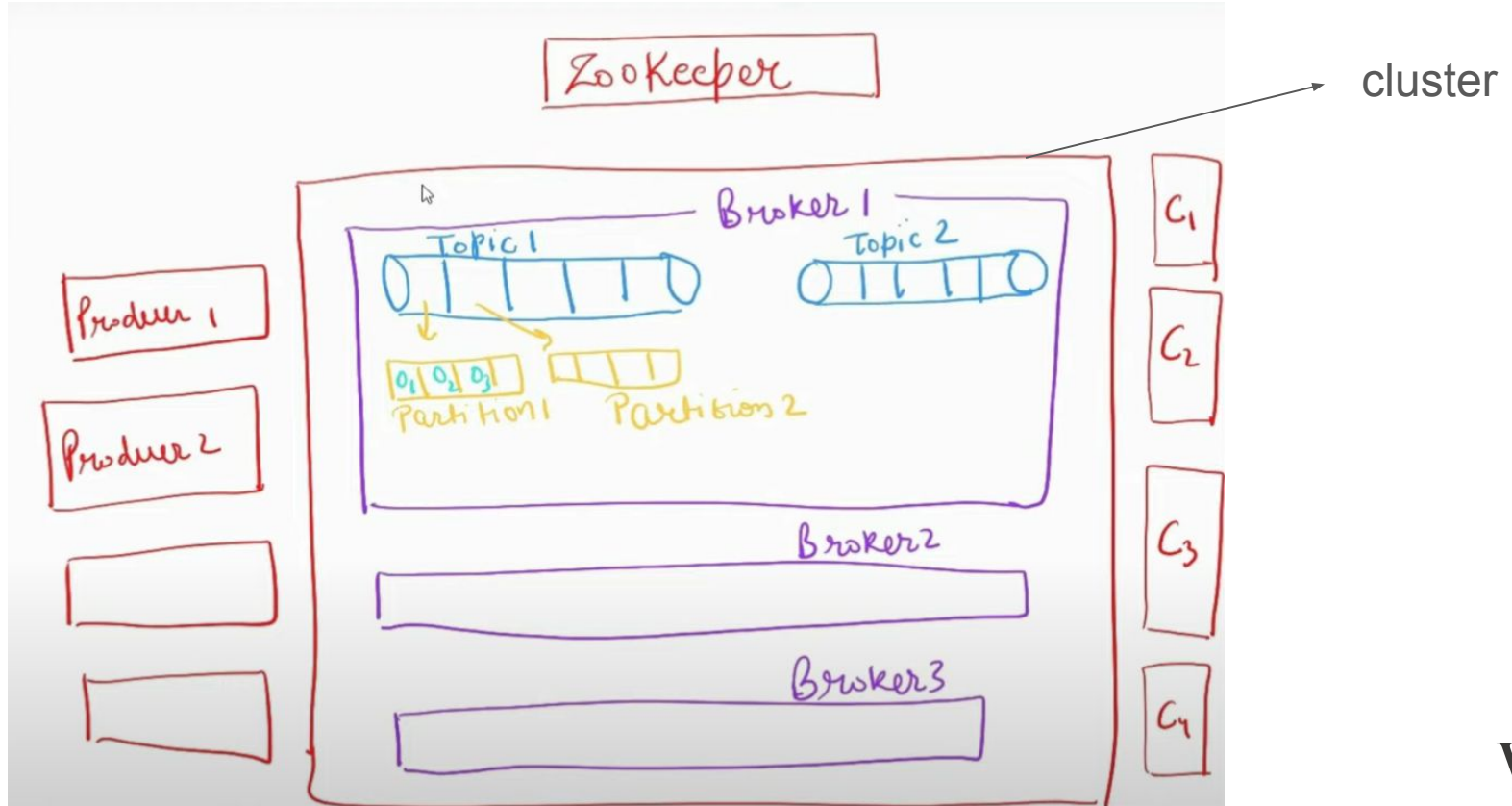
It is characterized by its

1. Scalability
2. Fault-tolerance
3. high throughput

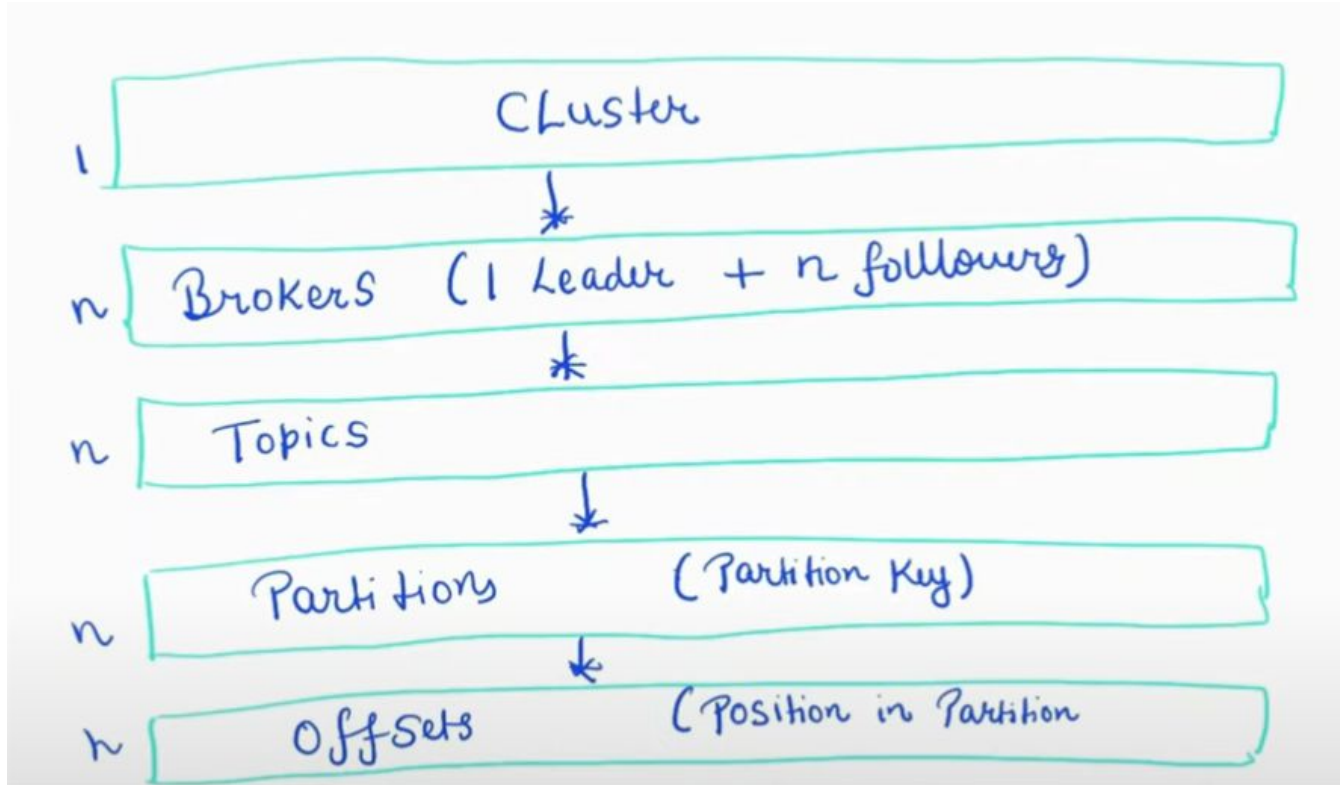
Making it a popular choice for organizations across various industries to manage and process streaming data efficiently.

Kafka is often used for use cases such as log aggregation, real-time analytics, monitoring, and messaging systems.

Architecture of kafka



Mapping inside the cluster

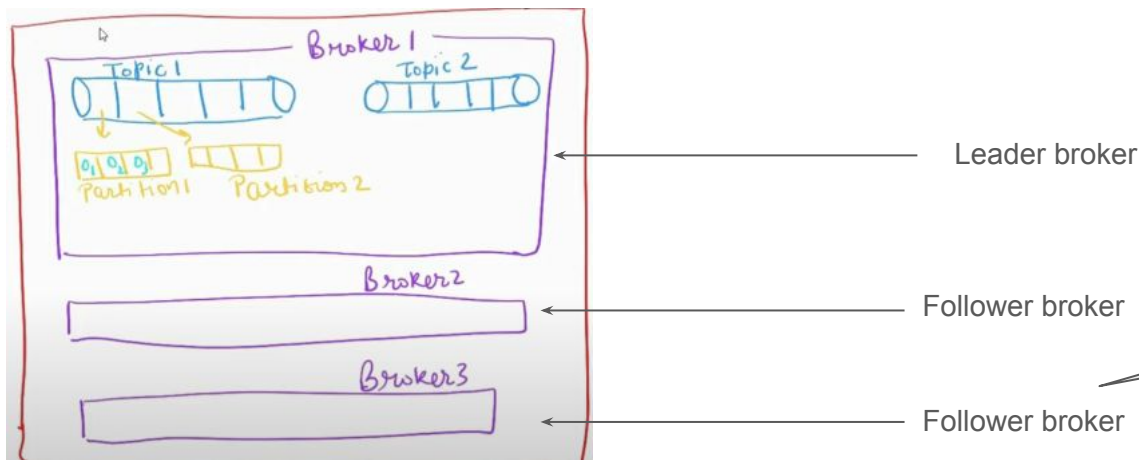


Understand each unit of kafka

Broker

The architecture of Apache Kafka is designed for **distributed**, **fault tolerant**, and **scalable** handling of streaming data.

High level overview there is a cluster containing multiple brokers where one is a leader and the rest is followers there by maintaining replica of data. Now the leader broker host a leader partition where the producer writes and consumer reads. Rest brokers hosts follower partitions and hence will **replicate** the **same data to different disk location for full tolerance**



Follower broker contains the same data for fault tolerance, in case we lost the data from the leader broker

Understand each unit of kafka

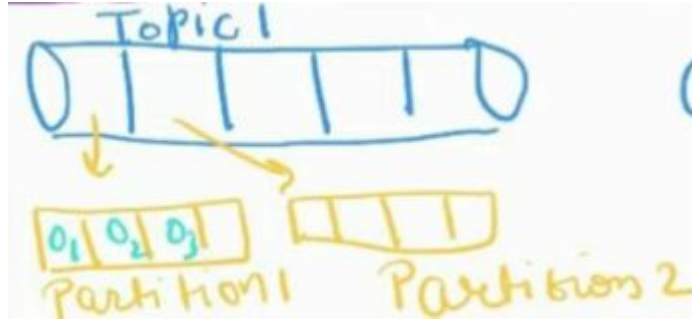
Broker

But still, what is broker?

Kafka broker are individual server within the kafka cluster. They store and manage data handle producer and consumer requests, and participate in the replication and distribution of the data

Understand each unit of kafka

Topic

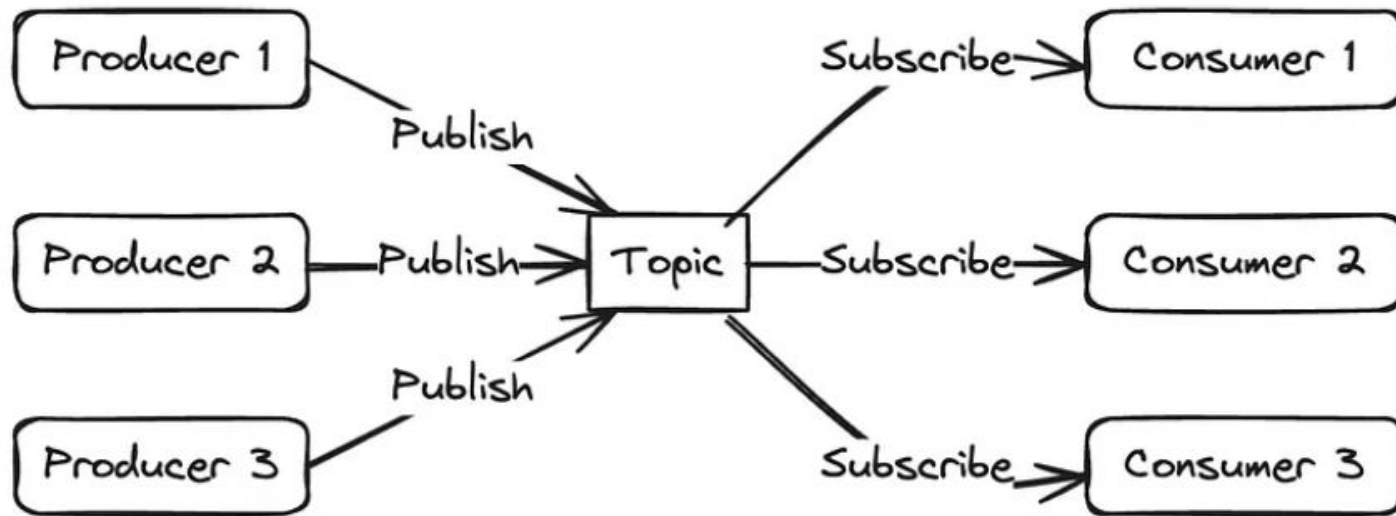


Topics are logical channels or categories to which messages are published. Topics can be divided into partitions for scalability and parallelism.

Understand each unit of kafka

Topic

Topics are logical channels or categories to which messages are published. Topics can be divided into partitions for scalability and parallelism.



Understand each unit of kafka

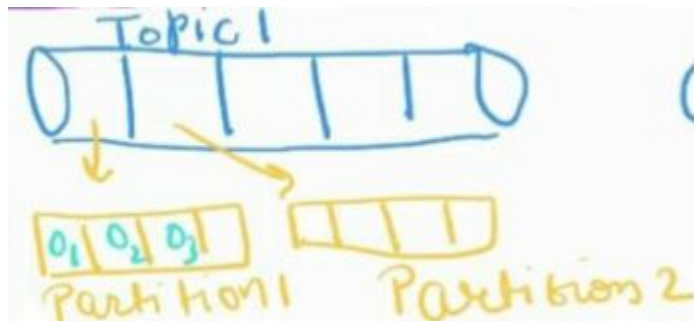
Topic

In Apache Kafka, a topic is like a unique folder or category where messages are stored and organized. It's where producers put their messages, and consumers come to get them. Each topic has its own name within Kafka. Producers send messages to specific topics, and consumers **subscribe** to get messages from those topics.

Topics are cool because they let lots of **multi-subscriber**(consumers) read messages at their own speed without bothering each other. This setup makes Kafka great for big projects where many apps or services need to work with the same messages.

Understand each unit of kafka

Partition



Partitions are a fundamental aspect of Kafka's scalability and fault tolerance. A topic in Kafka is divided into one or more partitions. These partitions allow the messages for a topic to be spread across multiple nodes in the Kafka cluster, enabling horizontal scaling of processing. Each partition can be placed on a different server, thus spreading the data load and allowing for more messages to be processed concurrently.

Understand each unit of kafka

Partition

Partitions also play a crucial role in fault tolerance. In Kafka, each partition can be replicated across multiple nodes, meaning that copies of the partition's data are maintained on different servers. This replication ensures that if a node fails, the data can be retrieved from another node that has a replica of the partition, thereby minimizing data loss and downtime.

The number of partitions in a topic directly influences the degree of parallelism in data processing, as more partitions mean more parallel consumers can read from a topic concurrently. However, the partition count should be chosen wisely, as too many partitions can lead to overhead in management and decreased performance due to the increased number of file handles and network connections.

Understand each unit of kafka

Partition

In summary, topics and partitions are central to Kafka's architecture, providing the mechanisms for organizing messages and enabling the system to scale out and handle failures gracefully, thus ensuring reliable data delivery at scale

Understand each unit of kafka

Zookeeper



Understand each unit of kafka

Zookeeper



Let's first, understand the zookeeper

In real life, a zookeeper is responsible for the care and management of animals in a zoo, ensuring their well-being, safety, and maintaining their habitats. Similarly, in the context of Apache Kafka, Zookeeper is a crucial component responsible for **managing and coordinating the Kafka cluster**.

Here are some similarities between a real-life zookeeper and Kafka Zookeeper:

Management and Coordination: Just as a real-life zookeeper manages and coordinates the activities and needs of animals in a zoo, Kafka Zookeeper manages and coordinates the activities of Kafka brokers (servers) and clients (producers and consumers).

Understand each unit of kafka

Zookeeper



Let's first, understand the zookeeper

Ensuring Stability and Safety: Both types of zookeepers ensure stability and safety. A real-life zookeeper ensures the safety and well-being of animals, while Kafka Zookeeper ensures the stability and reliability of the Kafka cluster by keeping track of its state and configuration.

Maintaining Order and Organization: Zookeepers in both contexts maintain order and organization. In a zoo, this involves organizing feeding schedules, cleaning habitats, and ensuring proper care routines. In Kafka, Zookeeper maintains the metadata about Kafka topics, brokers, and partitions, ensuring that the cluster remains organized and

Understand each unit of kafka

Zookeeper



Let's first, understand the zookeeper

Handling Failures: Both types of zookeepers handle failures. In a zoo, this might involve responding to medical emergencies or resolving conflicts among animals. In Kafka, Zookeeper handles failures by **electing a leader among Kafka brokers and ensuring that the cluster can continue to operate even if some nodes fail.**

Communication and Coordination: Zookeepers in both contexts facilitate communication and coordination. A real-life zookeeper communicates with other staff members to ensure that tasks are carried out efficiently. Kafka Zookeeper facilitates communication and coordination among Kafka brokers and clients by providing a **centralized coordination** service.

Understand each unit of kafka

Zookeeper



Here are the main roles of ZooKeeper in Kafka:

Cluster Coordination:

Leader Election

Broker Registration and management

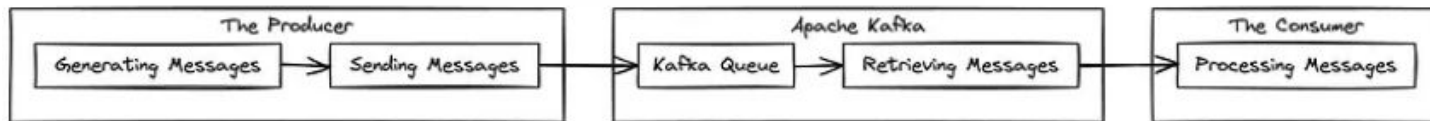
Topic and Partition Information

Consumer Group Management

Broker and Topic Health Monitoring

Understand each unit of kafka

Producer



Producers are application that published messages to kafka topics. they create producer record like this

```
@Override
public boolean updateLocation(String location){

    kafkaTemplate.send(Constant.CAB_LOCATION,location);
    return true;
}
```

```
public CompletableFuture<SendResult<K, V>> send(String topic, @Nullable V data) {
    ProducerRecord<K, V> producerRecord = new ProducerRecord(topic, data);
    return this.observeSend(producerRecord);
}
```

Understand each unit of kafka

Consumer

A consumer in Apache Kafka is an application or process that subscribes to one or more topics and reads messages from them. Consumers pull data from Kafka topics; they do not receive data pushed to them. This ***pull model*** allows consumers to control the pace at which they process messages, enabling them to handle varying loads of data efficiently.

In Kafka, consumers team up to handle data from different parts of a topic. This teamwork helps spread out the work and make things faster. Each consumer in the team gets assigned specific parts of the topic to focus on. Every message from those parts goes to just one consumer in the team. Kafka keeps track of who's doing what, moving things around as needed when new members join or others leave.

Understand each unit of kafka

Consumer

Two main work of Kafka consumer



Consumers keep track of the messages they have processed using an **offset**, which is a sequential id that uniquely identifies each message within a partition.

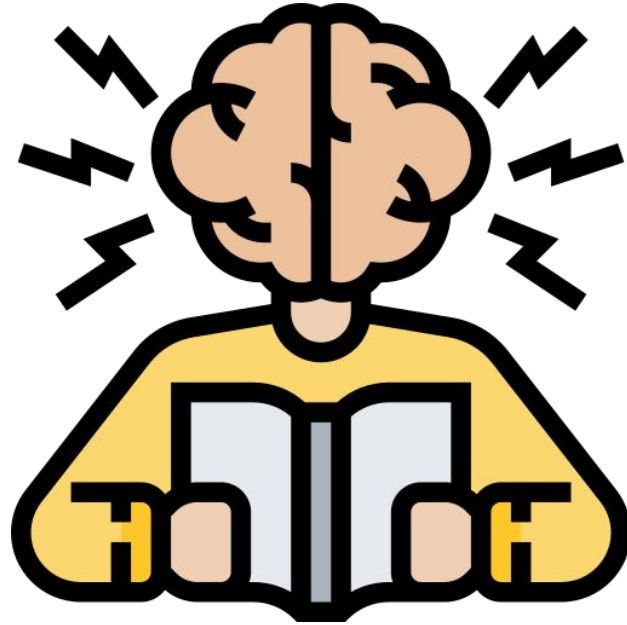
Consumers can commit these **offsets** to Kafka, which allows them to **resume** reading from where they left off in case of a failure or restart, ensuring no message is lost or processed twice.

What are Offsets?

They are just the unique id each message is given to.

Kafka has its own Offsets management(ZOOKEEPER), it knows that if the offsets are full. now product messages will be written from the next offset only.

Enough with theory of Beloved Kafka 🥹! :)



Let's implement this theory

Some steps to setup kafka in local

- Goto → <https://kafka.apache.org/quickstart>
- Download file like this → [kafka_2.13-3.7.0.tgz](#)
- Unzip this file in your C- drive
- Install java 8 or 8+ version in your Local machine (if you have it then skip this step)
- Now go to this folder like this **kafka_2.13-3.7.0** in your C-drive and then goto **Bin** folder then **window**(If you are using window only)
- Open this path in your Command Prompt it looks like this

C:\kafka_2.13-3.7.0\bin\windows>

Setup is done, phewww!



It is was very hard, right?

Now we will going to run our kafka in our local machine

There is **two** step to start kafka

1. Start hardworking Zookeeper



Command to start zookeeper

```
zookeeper-server-start.bat ../../config/zookeeper.properties
```

Paste the above command in your first cmd window like this

```
C:\Windows\System32\cmd.e  X  +  v

Microsoft Windows [Version 10.0.22631.3374]
(c) Microsoft Corporation. All rights reserved.

C:\kafka_2.13-3.7.0\bin\windows>zookeeper-server-start.bat ../../config/zookeeper.properties
```

Now we will going to run our kafka in our local machine

There is **two** step to start kafka

1. Start Kafka cluster



Command to start kafka cluster

```
kafka-server-start.bat ../../config/server.properties
```

Paste the above command in your 2nd cmd window like this

```
C:\Windows\System32\cmd.e  X  +  v
Microsoft Windows [Version 10.0.22631.3374]
(c) Microsoft Corporation. All rights reserved.

C:\kafka_2.13-3.7.0\bin\windows>kafka-server-start.bat ../../config/server.properties
```

Note- Do not close those two cmd windows

Now we will gonna create a topic with name “**vipul-D-topic**”

Open new cmd window

```
kafka-topics.bat --create --topic vipul-D-topic --bootstrap-server localhost:9092
```

Paste the above command in your 3rd cmd window like this

```
C:\kafka_2.13-3.7.0\bin\windows>kafka-topics.bat --create --topic vipul-D-topic --bootstrap-server localhost:9092  
Created topic vipul-D-topic.
```

To see topics

ALL :-

```
kafka-topics.bat --describe --bootstrap-server localhost:9092
```

Specific :-

```
kafka-topics.bat --describe --topic vipul-D-topic --bootstrap-server localhost:9092
```


Open two more CMD one for producer and one for consumer

Producer

```
kafka-console-producer.bat --broker-list localhost:9092 --topic vipul-D-topic  
>Message 1  
>Message 2  
>
```

Consumer

```
kafka-console-consumer.bat --bootstrap-server localhost:9092 --topic vipul-D-topic --from-beginning  
O/P  
>message1  
>message2  
>message3
```

Output - as soon we send data from producer it will be consume by a consumer

Producer

```
C:\Windows\System32\cmd.e x + v
Microsoft Windows [Version 10.0.22631.3374]
(c) Microsoft Corporation. All rights reserved.

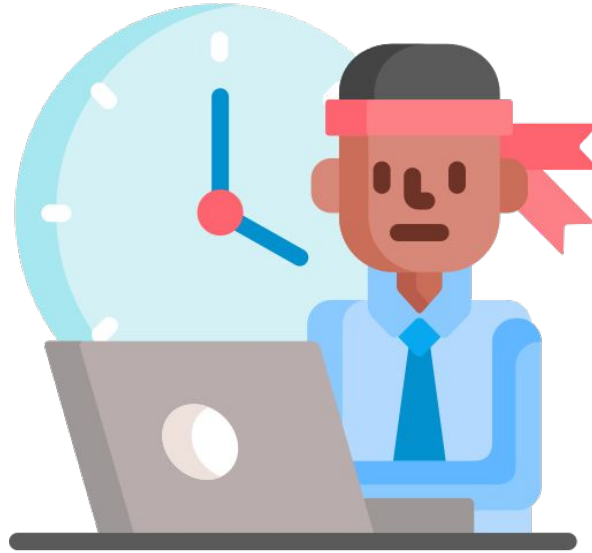
C:\kafka_2.13-3.7.0\bin\windows>kafka-console-producer.bat --broker-list loc
alhost:9092 --topic vipul-D-topic
>message1
>vipul
>roh
>dee
>ani
>messageX
>|
```

Consumer

```
C:\Windows\System32\cmd.e x + v
Microsoft Windows [Version 10.0.22631.3374]
(c) Microsoft Corporation. All rights reserved.

C:\kafka_2.13-3.7.0\bin\windows>kafka-console-consumer.bat --bootstrap-serve
r localhost:9092 --topic vipul-D-topic --from-beginning
message1
vipul
roh
dee
ani
messageX
|
```

That's all from Kafka in local



Let's take an example

Example of cab booking app

When a user go to cab booking app and book a cab then a driver is assigned to that particular user, now that user is getting constant location update of the driver, so how you think this type of communication is done?

There is two ways

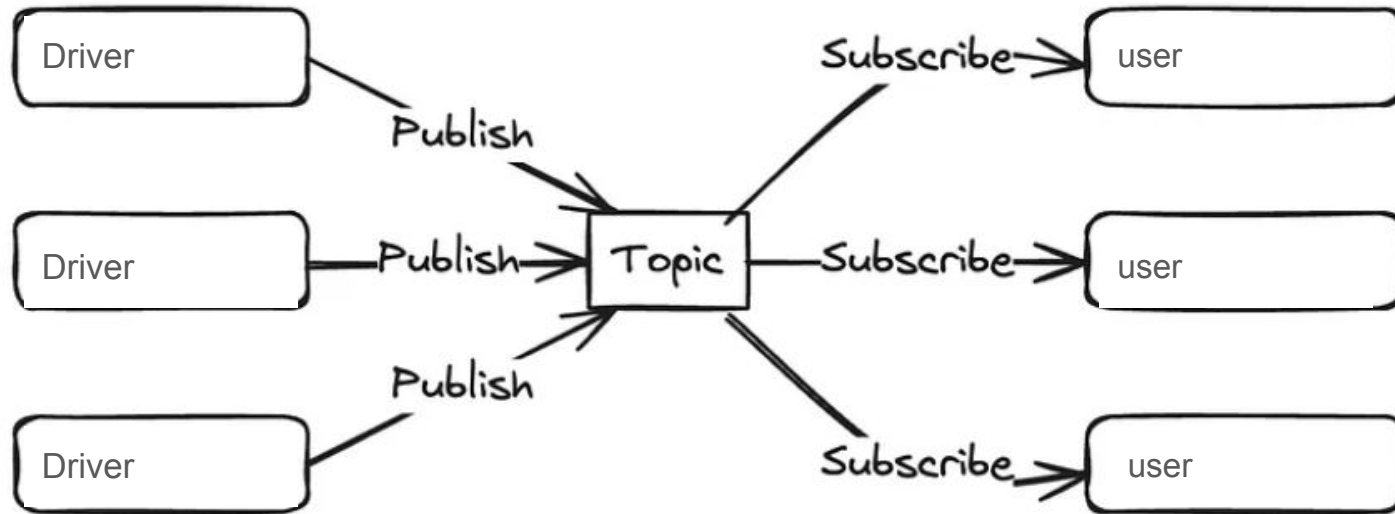


```
graph TD; A[There is two ways] --> B[Using DB to save the locations and then pass to User]; A --> C[Using Kafka to save the locations in a queue and then send it to the user];
```

Using DB to save the locations
and then pass to User

**Using Kafka to save the locations in
a queue and then send it to the user**

Back to our 2nd solution: Using Kafka



Let's implement this is using Spring Boot



Spring Boot

Create two spring boot project

Cab-book-driver

Cab-book-user

Add the same dependency in both project

Maven:

```
<dependency>  
  <groupId>org.springframework.kafka</groupId>  
  <artifactId>spring-kafka</artifactId>  
</dependency>
```


Create a configuration class in Cab-book-driver project

```
package com.vip.config;

import com.vip.common.Constant;
import org.apache.kafka.clients.admin.NewTopic;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.kafka.config.TopicBuilder;
```

```
@Configuration
```

```
public class KafkaConfig {
```

```
    @Bean
```

```
    public NewTopic topic(){
```

```
        return TopicBuilder.name(Constant.CAB_LOCATION).build();
```

```
    }
```

```
}
```

This is our topic name
In our case

```
public static final String CAB_LOCATION = "cab-location"
```

Controller

```
@RestController
@RequestMapping("/location")
public class CabLocationController {

    1 usage
    @Autowired
    private CabLocationService cabLocationService;

    @PostMapping("")
    public ResponseEntity updateLocation() throws InterruptedException {

        int range=100;
        while(range>0){
            //mock data
            cabLocationService.updateLocation(Math.random()+" , "+Math.random());
            // creating a delay in pushing so that location get pushed to kafka in every second
            Thread.sleep( millis: 1000);
            range--;
        }

        return new ResponseEntity<>(Map.of( k1: "message", v1: "Location Updated"), HttpStatus.OK);
    }
}
```

Service

4 usages 1 implementation

```
public interface CabLocationService {
```

1 usage 1 implementation

```
    boolean updateLocation(String location);
```

```
}
```

Service Implementation

`@Service`

```
public class CabLocationServiceImpl implements CabLocationService {
```

1 usage

`@Autowired`

```
private KafkaTemplate<String, Object> kafkaTemplate;
```

1 usage

`@Override`

```
public boolean updateLocation(String location){
```

```
    // This is where we push our data to kafka --> PRODUCER
```

```
    kafkaTemplate.send(Constant.CAB_LOCATION, location);
```

```
    return true;
```

```
}
```

```
}
```

1. Start the zookeeper and kafka in your local using these two command as we have done before

Zookeeper →

```
zookeeper-server-start.bat ../../config/zookeeper.properties
```

Kafka →

```
kafka-server-start.bat ../../config/server.properties
```

2. After that, build and run the producer(*Cab-book-driver*) project

Create a consumer class in Cab-book-user project

```
@Service
public class LocationService {

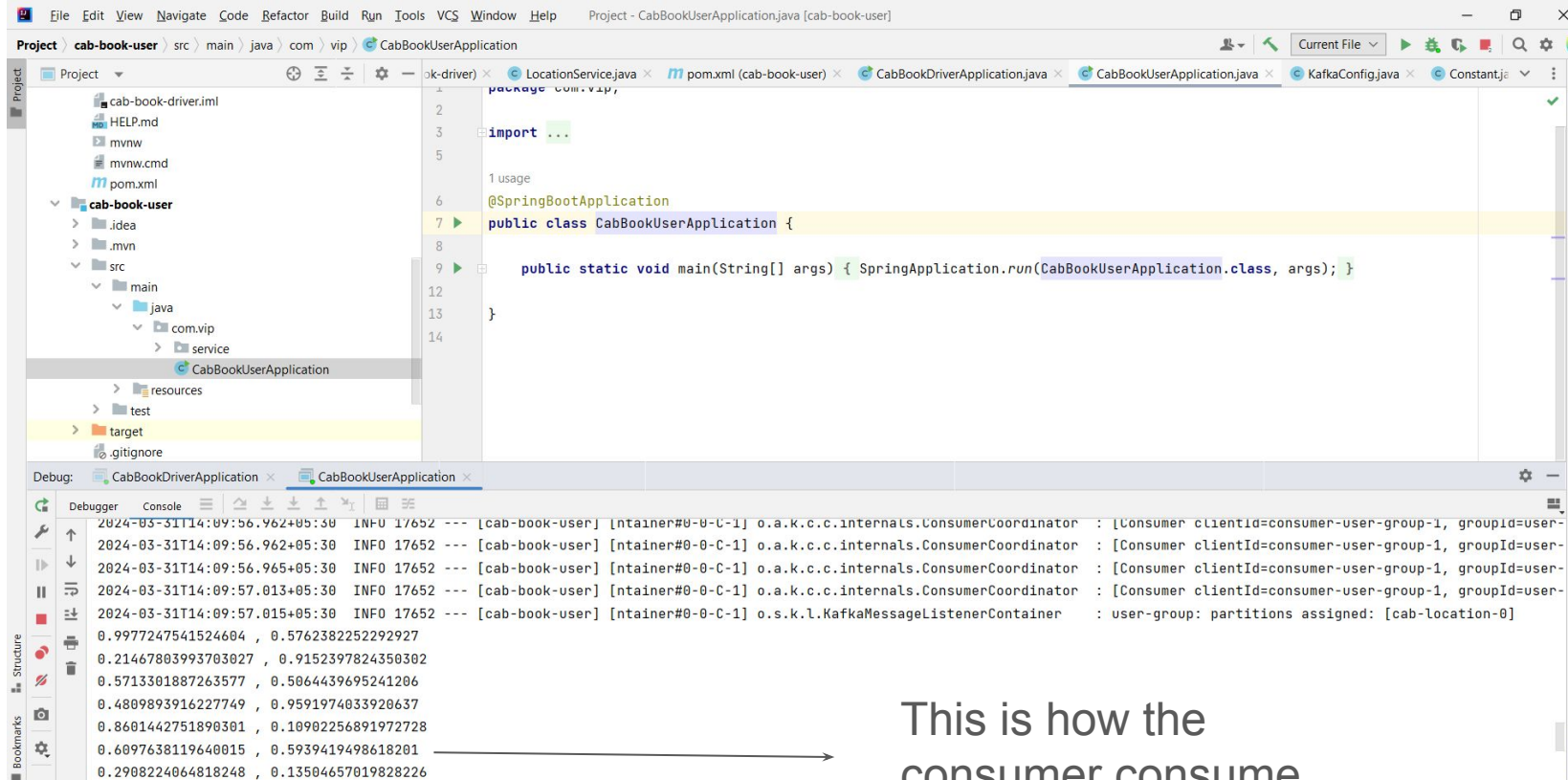
    @KafkaListener(topics = "cab-location", groupId = "user-group")
    public void cabLocation(String location){

        System.out.println(location);

    }
}
```

Here we are only printing the locations which were sent by our driver.

After that, build and run the consumer(Cab-book-user) project



The screenshot displays an IDE window for the 'cab-book-user' project. The left sidebar shows the project structure, with the 'target' directory highlighted. The main editor shows the 'CabBookUserApplication.java' file, which contains the following code:

```
package com.vip;  
  
import ...  
  
1 usage  
2  
3 @SpringBootApplication  
4  
5 public class CabBookUserApplication {  
6  
7     public static void main(String[] args) { SpringApplication.run(CabBookUserApplication.class, args); }  
8  
9 }  
10  
11  
12  
13  
14
```

The bottom panel shows the 'Debugger' tab with the following logs:

```
2024-03-31T14:09:56.962+05:30 INFO 17652 --- [cab-book-user] [ntainer#0-0-C-1] o.a.k.c.c.internals.ConsumerCoordinator : [Consumer clientId=consumer-user-group-1, groupId=user-  
2024-03-31T14:09:56.965+05:30 INFO 17652 --- [cab-book-user] [ntainer#0-0-C-1] o.a.k.c.c.internals.ConsumerCoordinator : [Consumer clientId=consumer-user-group-1, groupId=user-  
2024-03-31T14:09:57.013+05:30 INFO 17652 --- [cab-book-user] [ntainer#0-0-C-1] o.a.k.c.c.internals.ConsumerCoordinator : [Consumer clientId=consumer-user-group-1, groupId=user-  
2024-03-31T14:09:57.015+05:30 INFO 17652 --- [cab-book-user] [ntainer#0-0-C-1] o.s.k.l.KafkaMessageListenerContainer : user-group: partitions assigned: [cab-location-0]  
0.9977247541524604 , 0.5762382252292927  
0.21467803993703027 , 0.9152397824350302  
0.5713301887263577 , 0.5064439695241206  
0.4809893916227749 , 0.9591974033920637  
0.8601442751890301 , 0.10902256891972728  
0.6097638119640015 , 0.5939419498618201  
0.2908224064818248 , 0.13504657019828226
```

An arrow points from the log entry 'user-group: partitions assigned: [cab-location-0]' to the text 'This is how the consumer consume the location from kafka topic'.

This is how the
consumer consume
the location from
kafka topic

That's a wrap! All thanks to our beloved Kafka, the data rockstar! 🙌

Thanks for your time. Detecting Errors, Experiencing Keen Scrutiny Helps. Addressing suggestions for further improvement, I kindly request that you correct me rather than making fun of the mistakes .

Your constructive feedback is greatly appreciated :-)

Project Git repo : <https://github.com/vipultyagi07/Kafka.git>