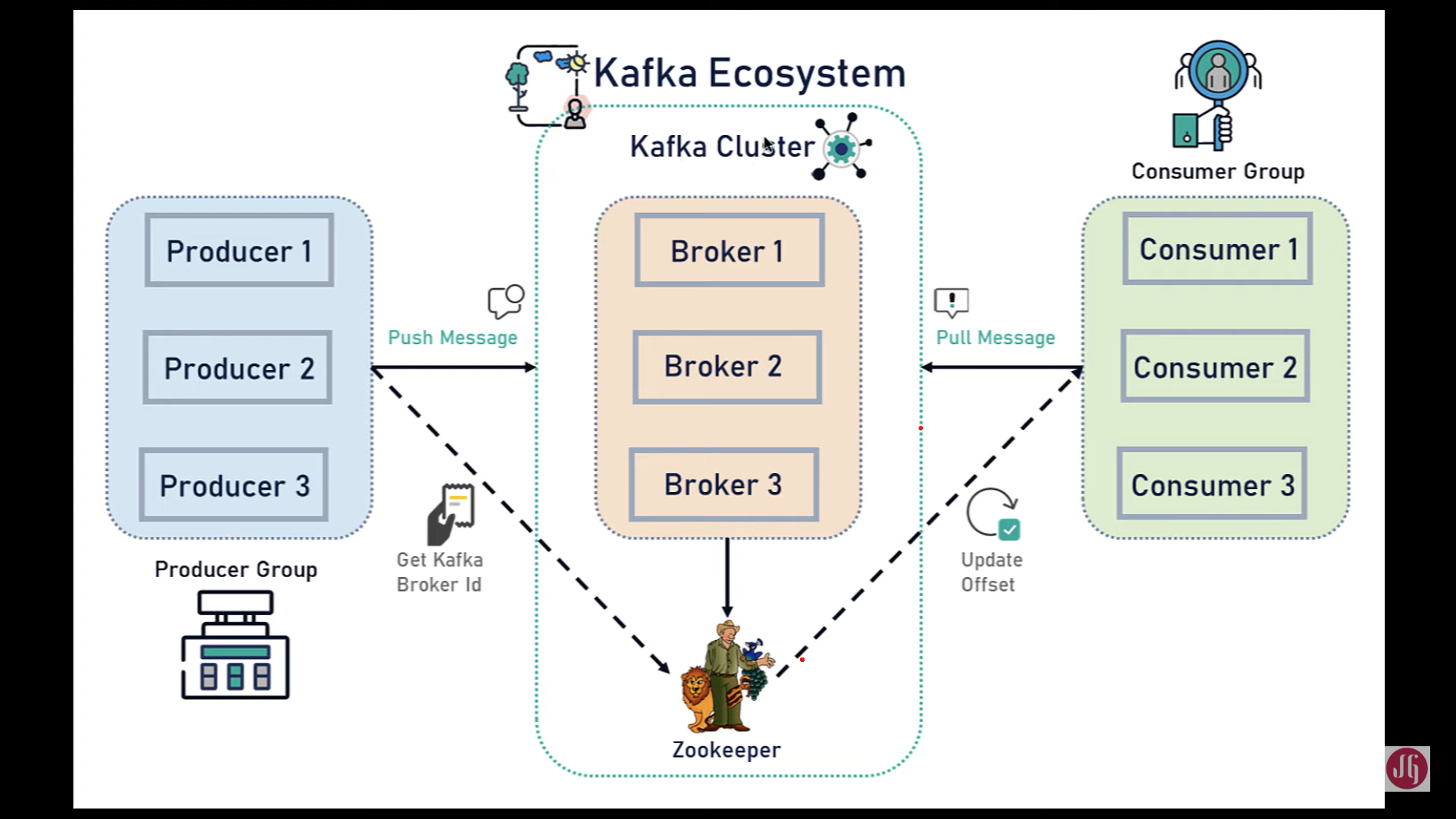
Kafka :

Apache Kafka is an open-source distributed event streaming platform used by thousands of companies for high-performance data pipelines, streaming analytics, data integration, and mission-critical applications.



**Kafka cluster:** Kafka cluster consist of set of brokers. Cluster has minimum of three brokers. If any broker goes down, it will have the capability to manage the entire cluster.

**Producer:** Producer is the application that will produce the message and send that message into the Kafka Cluster. Here message can be a event or stream of data or stream of records or JSON.

**Consumer**: Consumers will consume those messages from the Kafka brokers. Here consumers are the recipients.

**Zookeeper**: Zookeeper will manage the state of all the Kafka brokers and maintain the configuration of all the topics producer and consumers.

**1. Kafka Cluster**

Since Kafka is a distributed system, it acts as a cluster. A Kafka cluster consists of a set of brokers. A cluster has a minimum of 3 brokers.

The following diagram shows Kafka cluster with three Kafka brokers:

**[](https://blogger.googleusercontent.com/img/a/AVvXsEi9hrIWpPIyRZ5rVEc0K3UZ7_s2IUxnZvF5DB0W5Pp9mfZQFKuiwVsasWAlUOueWCQv9z8qR8DcoZqIxImgjEpmDrQHAne1y7odpkE87e-QGzwWCarB2k8ZA_ZH07AMft9ZPj7xLZXz8cu4ugCDU5c2XCeJTChNwJ-8zb9MWxdvkzAx-Z1wEPqZK2JZ)**

**2. Kafka Broker**

The broker is the Kafka server. It's just a meaningful name given to the Kafka server. And this name makes sense as well because all that Kafka does is act as a message broker between producer and consumer.

The producer and consumer don't interact directly. They use the Kafka server as an agent or a broker to exchange messages.

The following diagram shows a Kafka broker, it acts as an agent or broker to exchange messages between Producer and Consumer:

**[](https://blogger.googleusercontent.com/img/a/AVvXsEiBTAZMBDniTdXHsiadf7DJbypCOFO0WjQhj5PnWAYTF58saqG6sbQu6CJBXFuB3mvbtm_D29IXzwScxUwyXGyuT3Sq3XCzymEGVUBReiEWfn7EcV-mPpRRXCIvIiSs6FBGLkcjb7YJRV3ysSoUMc5unD7G5867FY77MLxk95I_LeHYi1cH71TOPe8w)**

**3. Kafka Producer**

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

The following diagram shows Producer sends messages directly to Kafka broker:

**[](https://blogger.googleusercontent.com/img/a/AVvXsEiBTAZMBDniTdXHsiadf7DJbypCOFO0WjQhj5PnWAYTF58saqG6sbQu6CJBXFuB3mvbtm_D29IXzwScxUwyXGyuT3Sq3XCzymEGVUBReiEWfn7EcV-mPpRRXCIvIiSs6FBGLkcjb7YJRV3ysSoUMc5unD7G5867FY77MLxk95I_LeHYi1cH71TOPe8w)**

**4. 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 the Kafka server.

Anyone who is interested in that data can come forward and take it from the Kafka server. So, any application that requests data from a Kafka server is a consumer, and they can ask for data sent by any producer provided they have permission to read it.

The following diagram shows Producer sends messages directly to the Kafka broker and the Consumer consumes or reads messages from the Kafka broker:

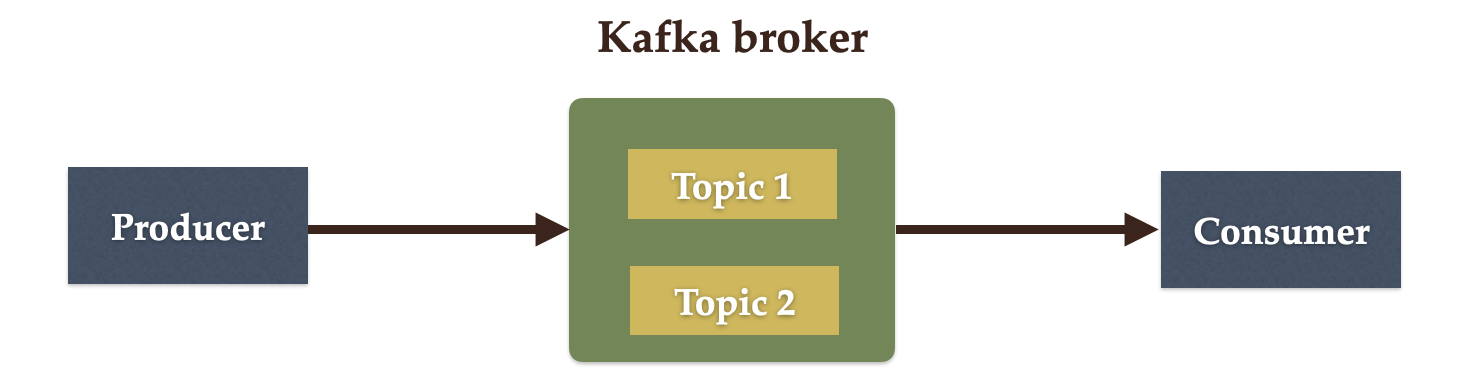
**[](https://blogger.googleusercontent.com/img/a/AVvXsEiBTAZMBDniTdXHsiadf7DJbypCOFO0WjQhj5PnWAYTF58saqG6sbQu6CJBXFuB3mvbtm_D29IXzwScxUwyXGyuT3Sq3XCzymEGVUBReiEWfn7EcV-mPpRRXCIvIiSs6FBGLkcjb7YJRV3ysSoUMc5unD7G5867FY77MLxk95I_LeHYi1cH71TOPe8w)**

**5. 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 Kafka topic.

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

The following diagram shows two Topics are created in a Kafka broker:

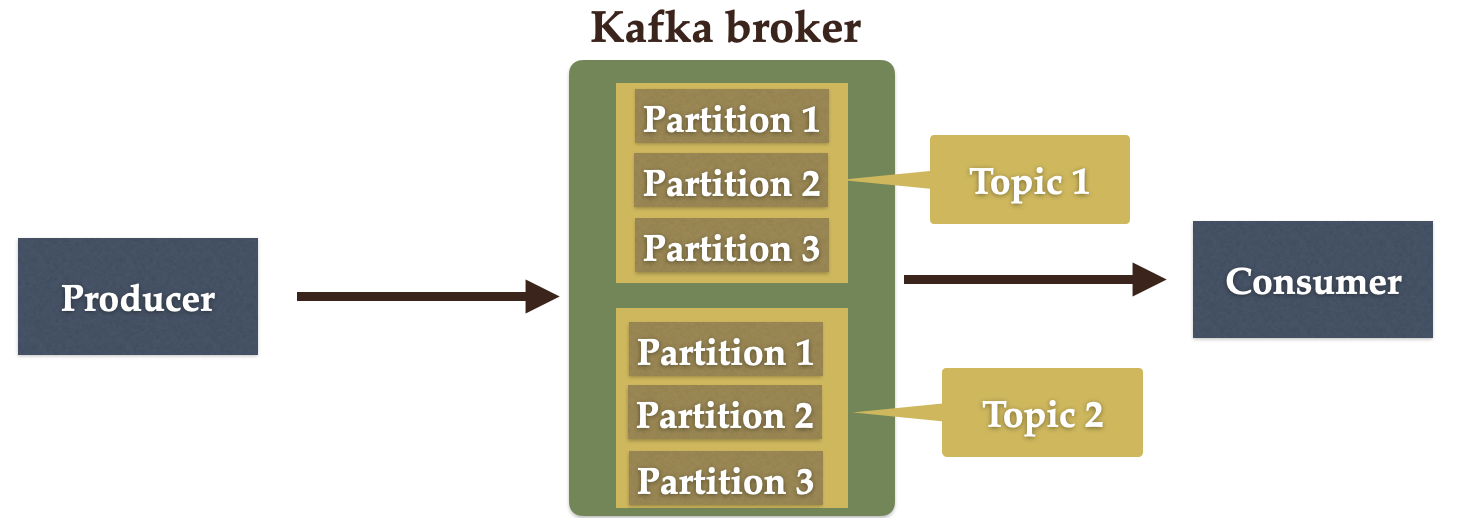
**[](https://blogger.googleusercontent.com/img/a/AVvXsEhlKR0H-usK8tqw4l6ZBotEdj0GlufBMIYymVIarTvZdq0mRwm4tmB0zZIuSFFEGh2r_c5oK2J2TYVxbzufIs3leif1K0ZOOG6gTZPI7_5vsGcSlvEHMBJVGM-WjSudn17PTFwcu4BBqb2Ip8Q9vZke57l7DmVOqOJXYyRaPGhjpX4yw47yOagYN_ws)**

**6. Kafka Partitions**

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

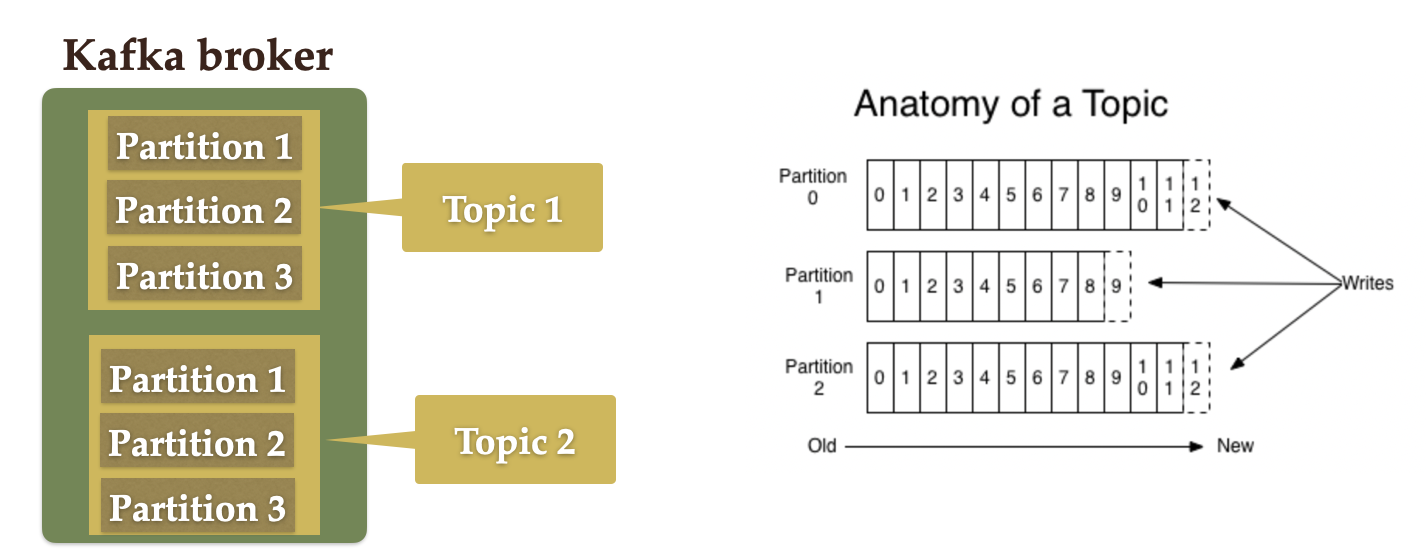
Kafka Brokers will store messages for a topic. But the capacity of data can be enormous and it may not be possible to store in a single computer. Therefore it will be partitioned into multiple parts and distributed among multiple computers since Kafka is a distributed system.

The following diagram shows Kafka's topic is further divided into a number of partitions:

**[](https://blogger.googleusercontent.com/img/a/AVvXsEjy5qEhC7_hrTlauMSEdhjOkybBZVp44eVVIpASv-s3-llmp_lqf9MqArwSu-LhIq6qovoOBXwcYz0VZK7fjsniTCjxUi09UlhhO4-R6VYZSpA9ZJyskfGXZaOJQTnt93fvcUdaoY9KEyph6QfeTQegZ5Os5Y_tVoV-UwPl1Ca-jrTpx89SnmWkh3L3)**

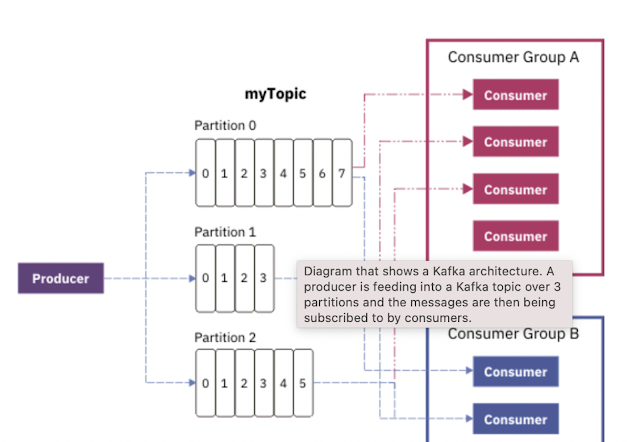
**7. Kafka Offsets**

Offset is a sequence of ids given to messages as they 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.

**[](https://blogger.googleusercontent.com/img/a/AVvXsEgLjNHk8K7-N07M7jGV7a_pSMB94bH0e1S4iWghXORilNuVztnJiKMZsbHLt9rQvjoEY8tz4vCafPTwBecVEwO1z5lDXqdI2qOGsnxWwVEjNoL1U_SYGcQdsigcSuRHYzBxNaG4yF0G7cU82nxHRaIA6YtNueBwVyTonbD8t8cmrJ_SK-NppnLXuqoZ)**

**8. Kafka Consumer Group**

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

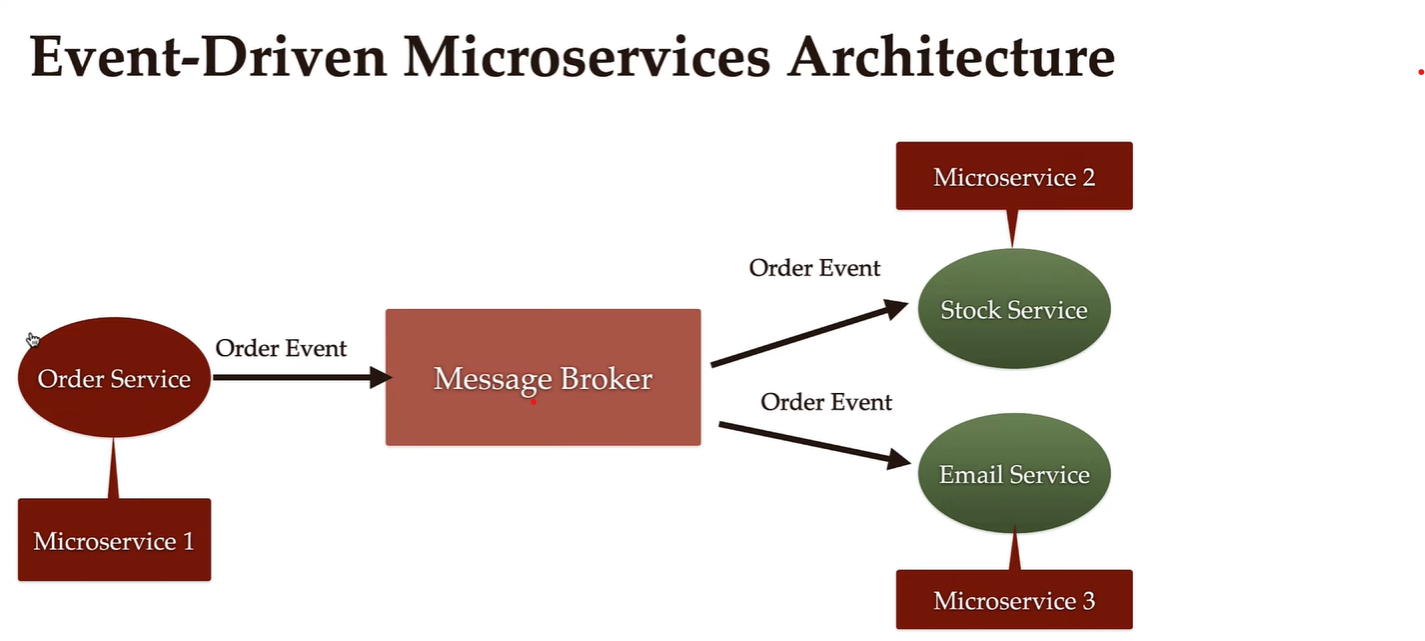
**[](https://blogger.googleusercontent.com/img/a/AVvXsEipRZwK4ijG2kIS4YhFXs0tSxXxRneQwp_BID8Kr5uPRQTWL6ps8F2z-bukwkGDJVTE-EBK8u2poCdJ-He4_O0Sie_KOPChIOZACHBA6A-1T4lqSVe_qO2064m-A93pPK4y6YhJYhyEXxInPw8I04qLxFFrrUcfV8q2ZNl7LrmFXB3XHc_D1lupVKwP)**

**Event-Driven Architecture:**

In this event-Driven microservices architecture all the microservices are loosely coupled they are independent to each other.

Event-Driven Architecture is a software design pattern. In which all the decoupled application will use message broker for Asynchronous communication.

Event-Driven Architecture is often referred to as an Asynchronous communication.



Once the order service will create the event and publish those events to the message broker. Then it don’t have to wait for the response. This is the Asynchronous communication.

**Advantages:**

1.Improves flexibility and maintainability.

2.High Scalability.

3.Improve the availability.

**Oder Service:** Here order service is responsible for collecting the orders and creating the order events and place those order events in the message broker.

**Stock Service:** It’s responsible for updating the order details in the DB.

**Email Service:** It’s responsible for sending the emails.

Here all 3 microservices are independent to each other. Here we are using message broker for communication.

#Start the Zookeeper

**$** .\bin\zkServer.cmd

# Start the ZooKeeper service

$ .\zookeeper-server-start.bat ..\..\config\zookeeper.properties

# Start the Kafka broker service

$ .\kafka-server-start.bat ..\..\config\server.properties

# Create Topics

$ .\bin\windows\kafka-topics.bat --create --topic topic example --bootstrap-server localhost:9092

# We have used producer to write the Events

$ .\bin\windows\kafka-console-producer.bat --topic topic example --bootstrap-server localhost:9092

(Create events name like – helloword,demo1,demo2 etc.)

#We have used consumer to read the events

$ .\bin\windows\kafka-console-consumer.bat --topic topic example --from-beginning --bootstrap-server localhost:9092

**Dependency:**

<dependency>

<groupId>org.springframework.kafka</groupId>

<artifactId>spring-kafka</artifactId>

<version>3.0.7</version>

</dependency>

**Links:**

Website Link - [Apache Kafka](https://kafka.apache.org/)

Spring Kafka Configuration link - [Spring for Apache Kafka](https://docs.spring.io/spring-kafka/reference/html/#introduction)

Realtime Wikimedia Data link - [stream.wikimedia.org/v2/stream/recentchange](https://stream.wikimedia.org/v2/stream/recentchange)