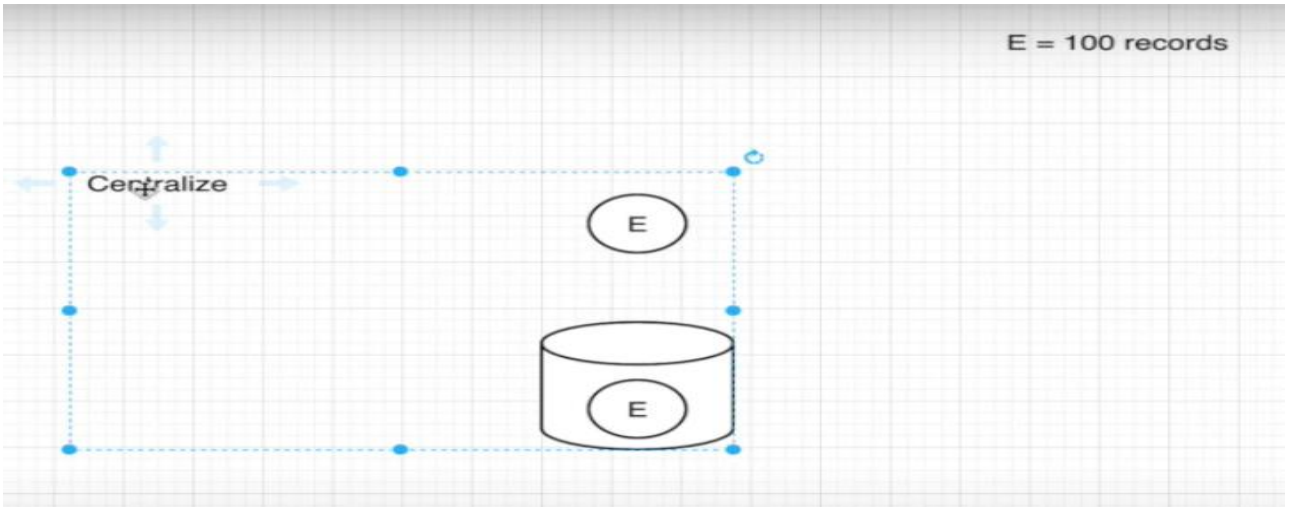
KAFKA:

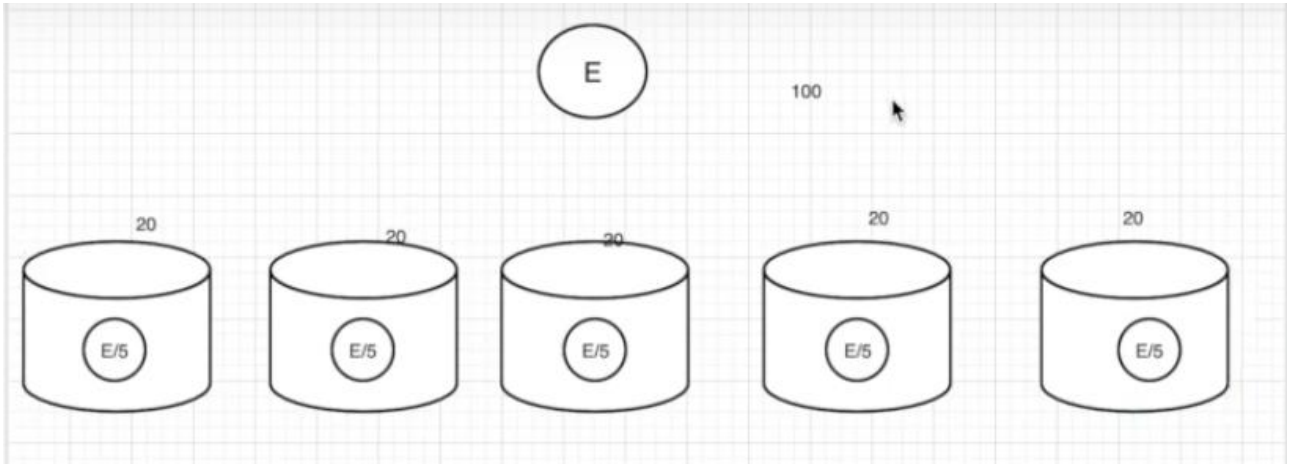

Old Way to store data (Message): Here we are storing the 100 records in one location (centralize system). If anything failed/crashed then we will loss all the data from the location.




To overcome this problem, we came up with distributed system.

## Distributed System:

1. This is the better approach from the old way

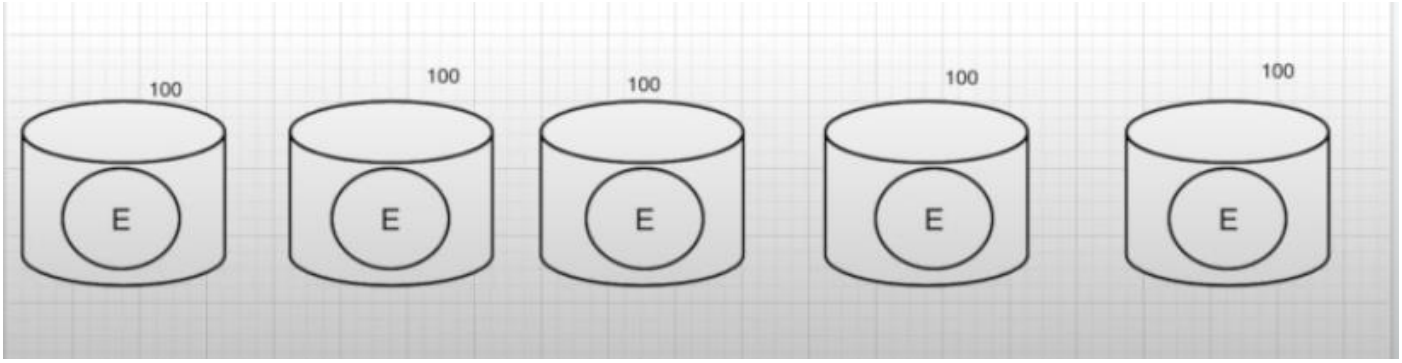



In this system we are storing 100 records in multiple location with equally divided part (20 in each location) as showing in image or as per our convenient.

So, in this approach if any system crashed then we will loss that records only other will be still safe.

To overcome this problem, we came up with the replication approach.

2. In this approach we copy the same entity data in multiple location and that is called data **replication**.

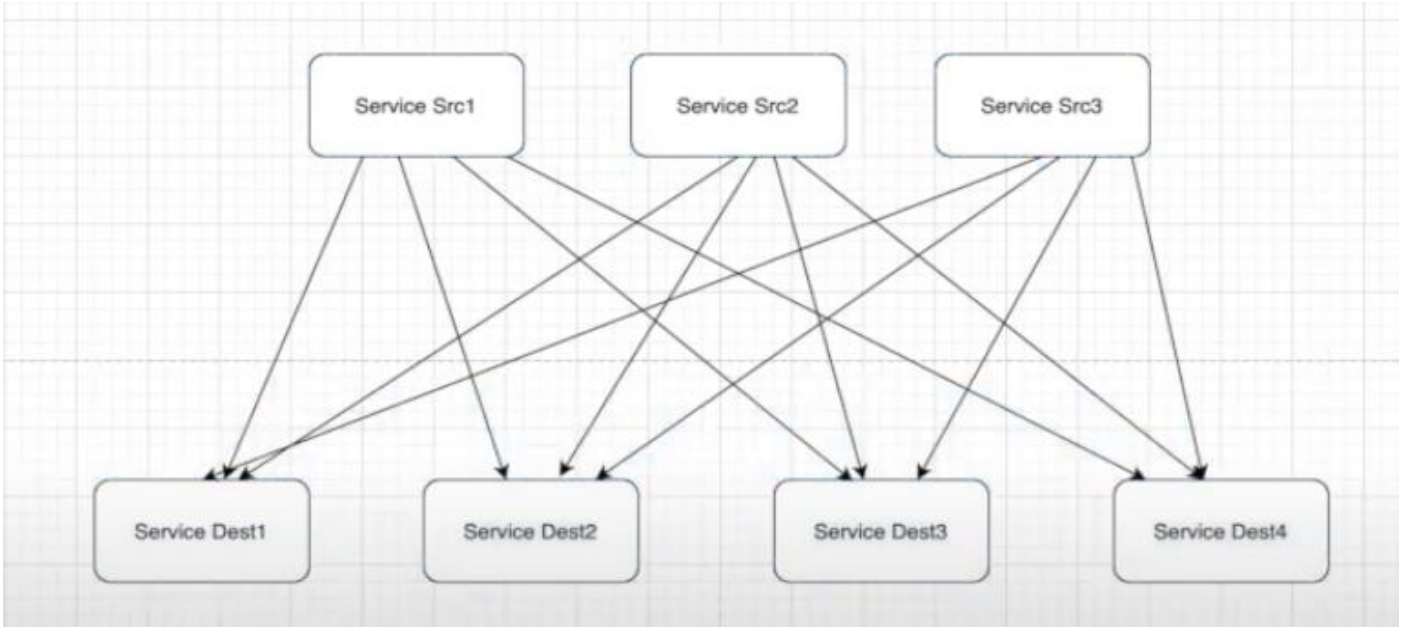



This is having disadvantage like it is consuming much space because same entity is being replicated in multiple location. Data redundancy being increased. But advantage is that any system is crashed then that data will be still available in different location we can access from there.

So, Kafka follow the both distributed approaches and these are configurable.

## Message Streaming Platform:

### Old Approach (Non-Messaging System):

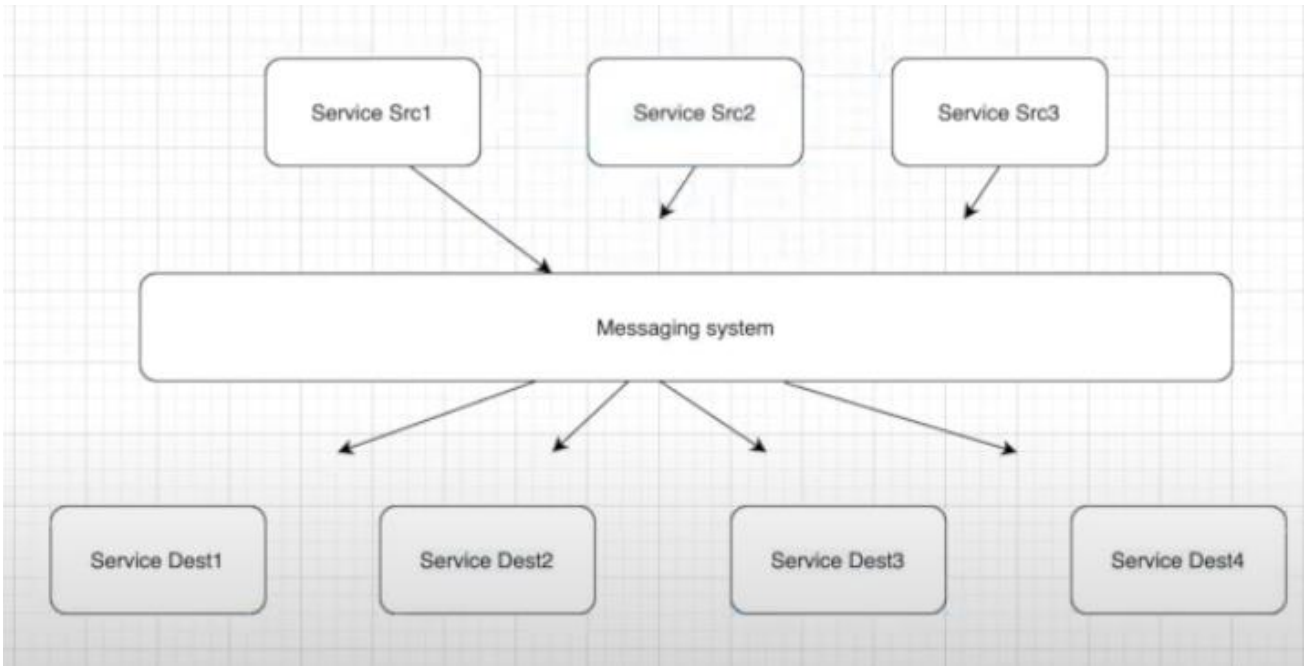



Here multiple services are there which are communicating with each other. One service is communication with different multiple services.so basis of below there are (3*4) 12 connection are established. If there will 1000s of services then number of connections will be increased which will be very difficult to manage. Every connection come with their difficulties:

1. What will be the Data format?
2. What will be the connection type?
3. What will be the data schema?
4. Etc.

To overcome with this problem, we came up with Messaging System.

**Messaging System:**

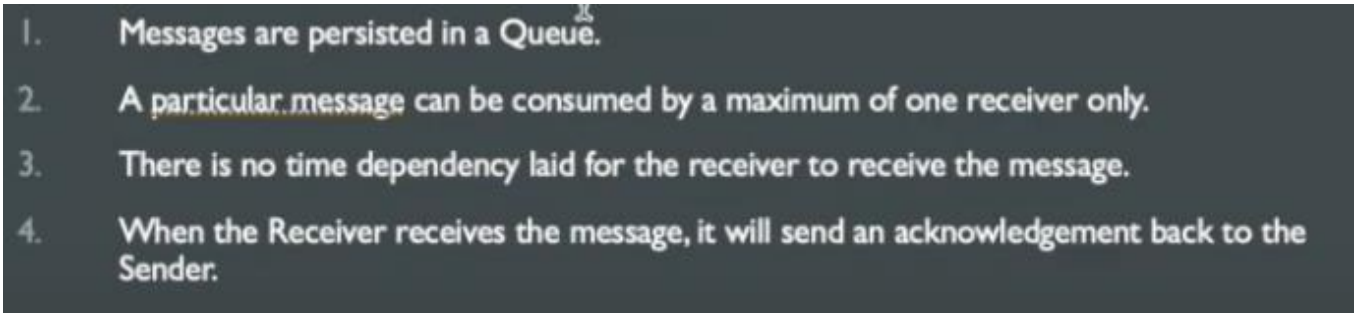


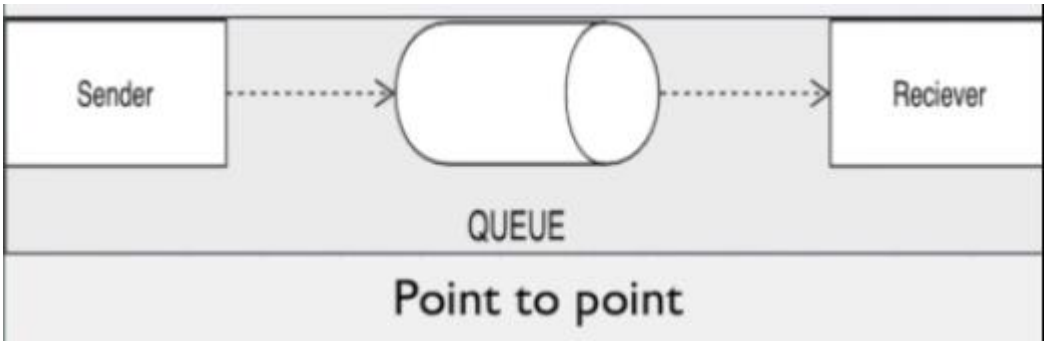

In this approach Services with communicate through messaging system whoever service will be interested in data that can take from messaging system because that data will be already put there by other system. And here (3+4) 7 connection are making only.

We can think like this approach we can handle with databases (we can store the data in data base and query the data from there) also then why we are using messaging system. Messaging system having the extra advantage like if any system who is interested in data, then whenever the data will be available in messaging system then interested parties will get the notification to get the data. This notification are two types
1. Pull notification
2. Push notification

***A Messaging System is responsible for transferring the data from one application to another so the application can focus on the data without getting bogged down on the data transmission and sharing***
It is divided in two types:

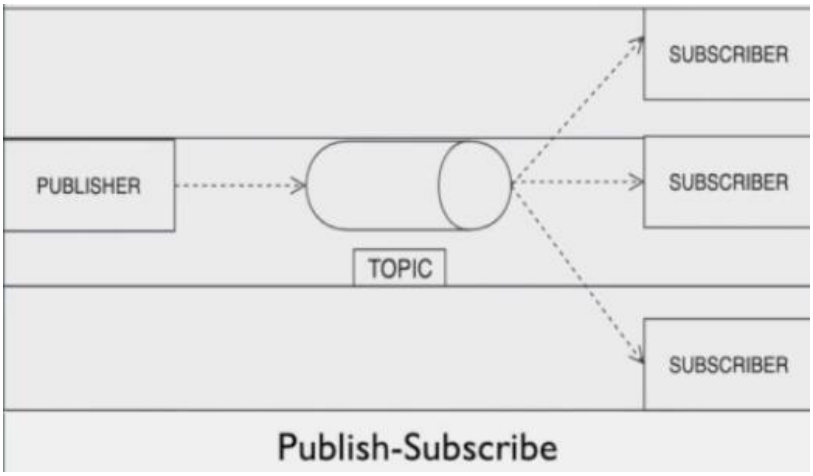
1. **Point to Point Messaging system:**



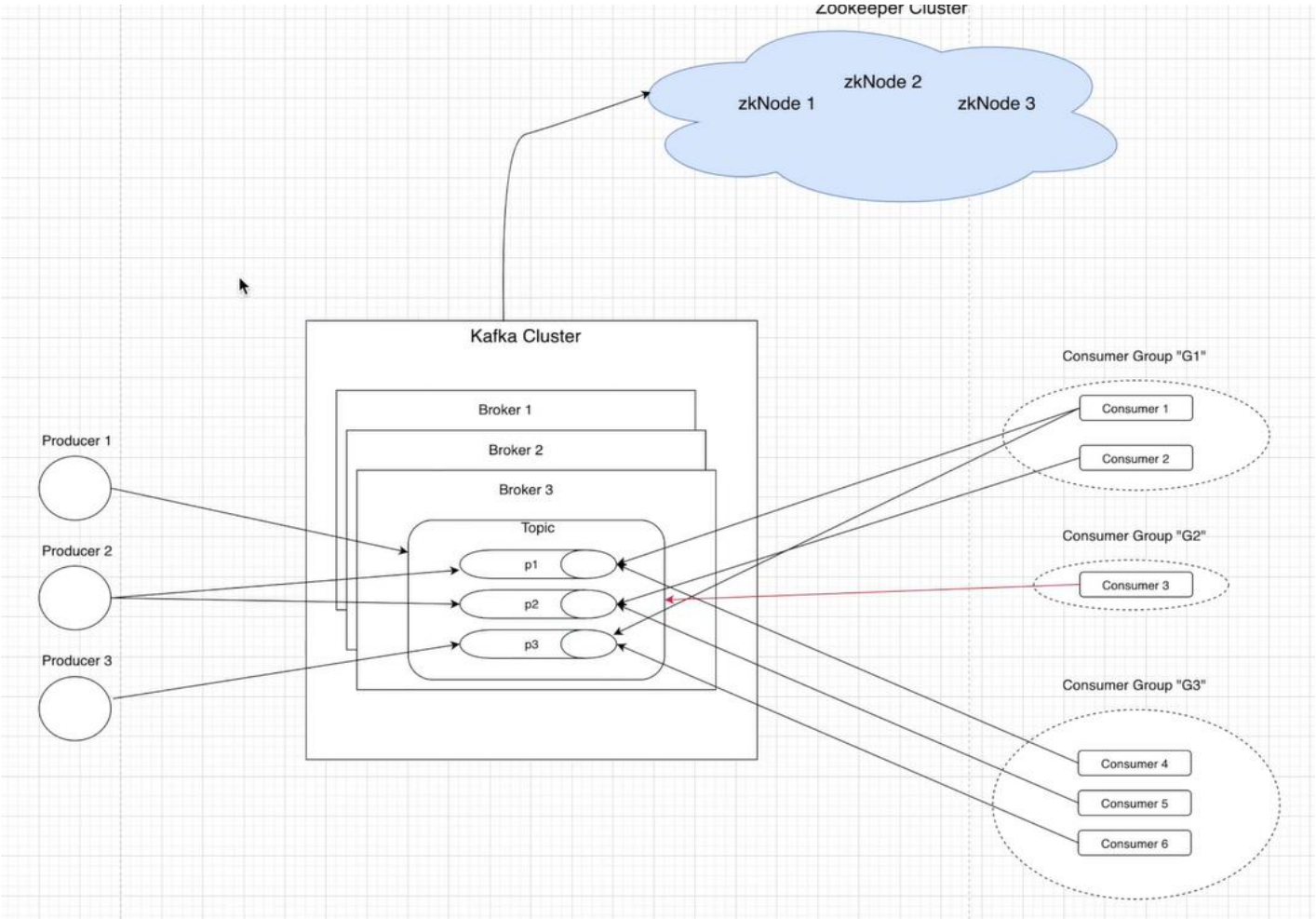



**2. Publish-Subscribe Messaging system:**

1. Messages are persisted in a Topic.
2. A particular message can be consumed by any number of consumers.
3. There is time dependency laid for the consumer to consume the message.
4. When the Subscriber receives the message, it doesn't send an acknowledgement back to the Publisher.

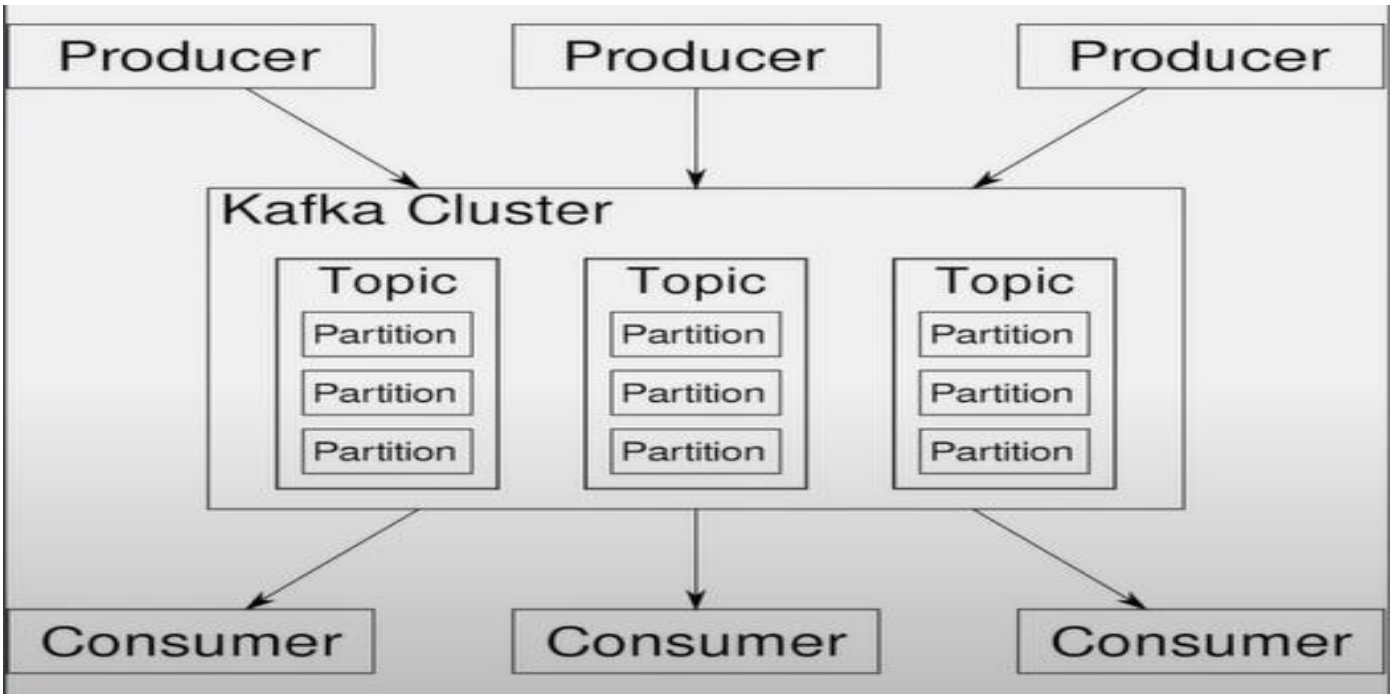



## KAFKA Architecture:



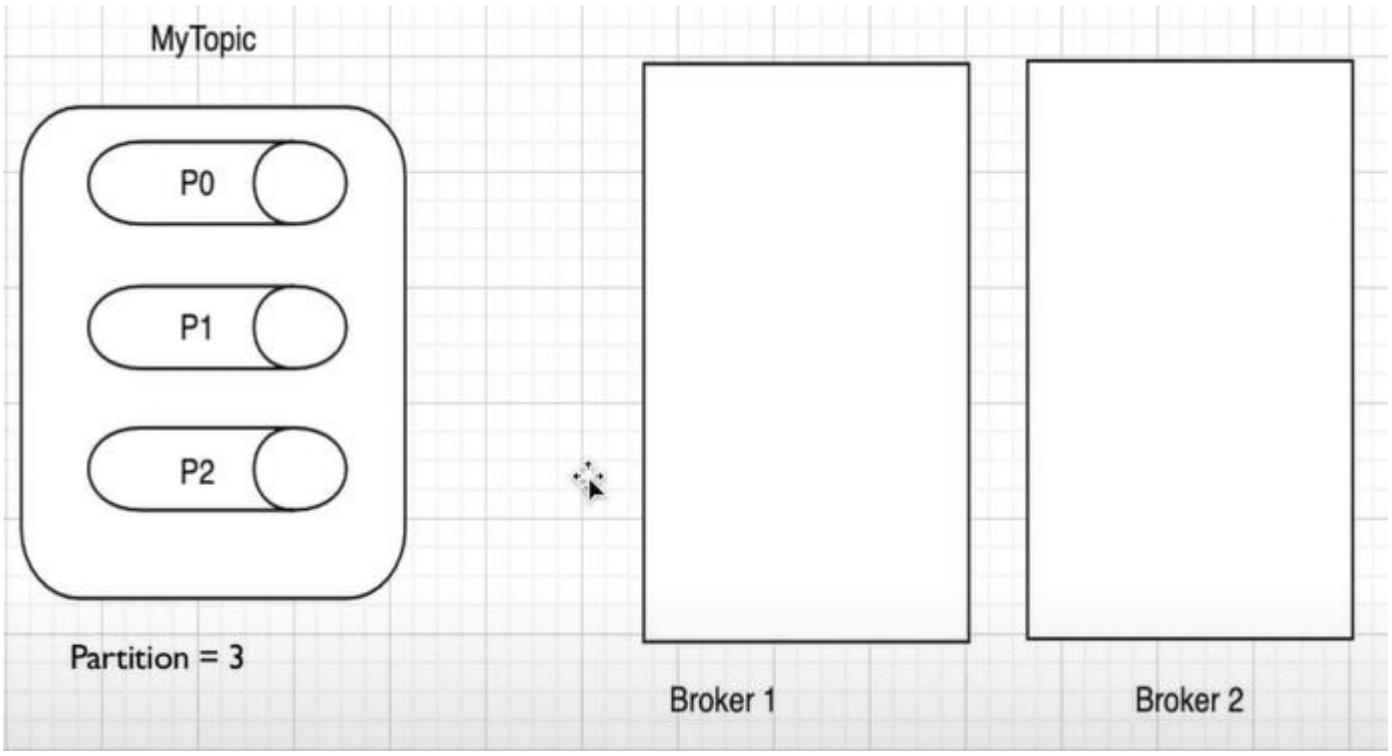
## Topics:

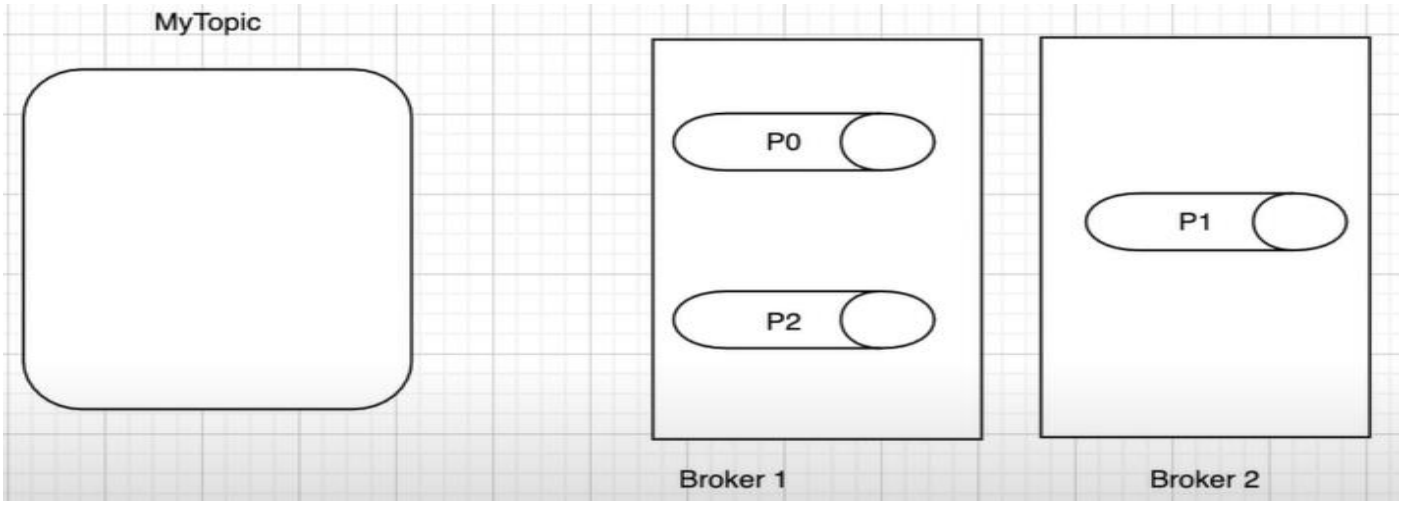
- A Stream of Message belonging to a particular category is called a topic**.**
- It is a logical feed name to which records are published.
- Similar to a table in a database (records are considered messages here)
- Unique Identifier pf a topic is its NAME.
- We can create as many topics as we want.




**PARTITIONS:**

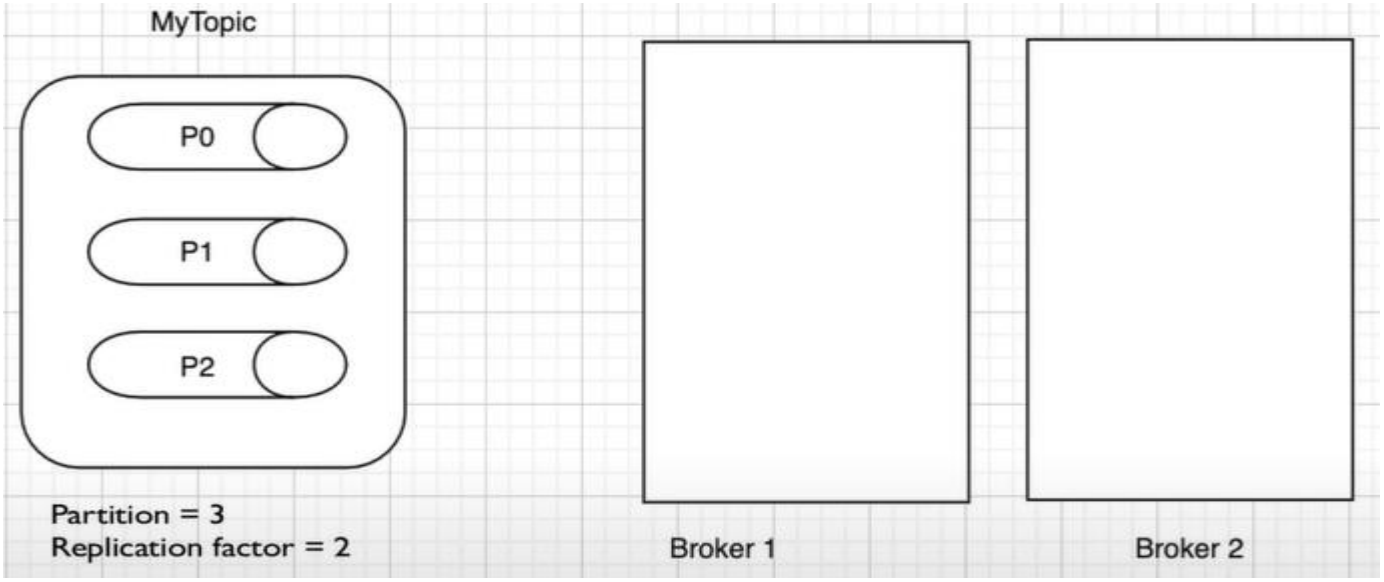
- Topics are split in partitions.
- All the messages within a partition are ordered and immutable.
- Each message within a partition has a unique ID associated known as **Offset.**

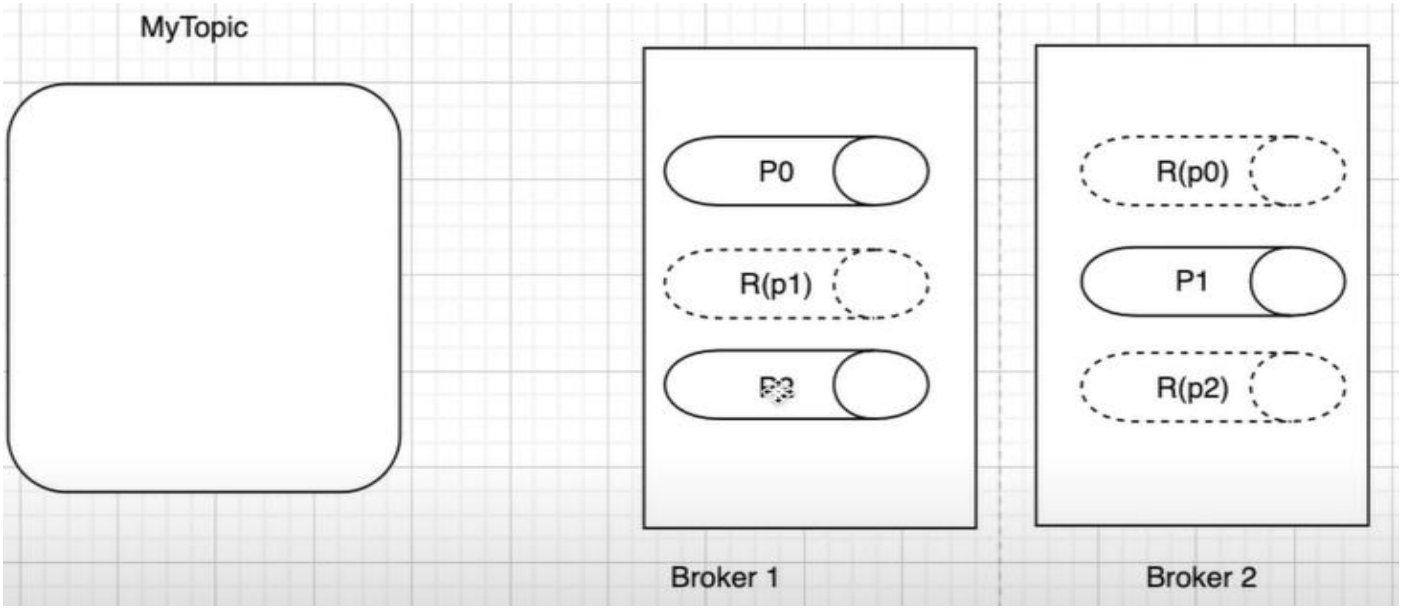





**REPLICA AND REPLICATION:**

- Replica are backup of partition.
- Replica are never read or write the data
- They are used to prevent data loss. (Fault Tolerance)



## Producer:

- Producers are applications which write/publish data to the topics within a cluster using producing APIs
- Producers can write data either on the topic level (All the partitions of the topic) or specific partitions of the topic.

## Consumer:

- Consumers are applications which read/consume data from the topics with in a cluster using the consuming APIs
- Consumers can read the data either on the topic level (All the partitions of the topic) or specific partitions of the topic.
- Consumers are always associated with exactly one **Consumer Group.**
- **A** Consumer Group is a group of related consumers that perform the task.

## Broker:

- Brokers are simple software processes who maintain and manage the published messages.
- Also Known as **Kafka servers**.
- Brokers also manage the consumer-offsets and are responsible for the delivery of message to the right consumers.
- A set of brokers who are communicating with each other to perform the management and maintenance task are collectively known as **Kafka Cluster**.
- We can add more brokers in an already running Kafka cluster without any down time.

## Zookeeper:

- Zookeeper is used to monitor Kafka Cluster and co-ordinate with each broker.
- Keeps all the metadata information related to Kafka cluster in the form of a key-value pair.
- Metadata includes: 1. Configuration information 2. Health Status of each Broker
- It is used for the controller election within Kafka Cluster.
- A set of Zookeepers nodes working together to manage other distributed system is known as Zookeeper Cluster Or "**Zookeeper Ensemble**"

## Kafka Feature:

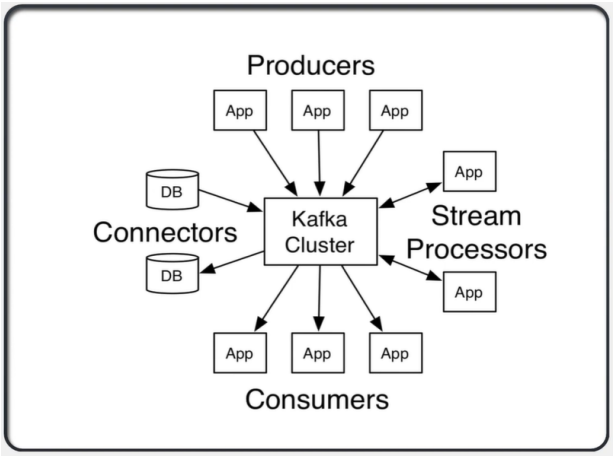
**Benefits of Kafka over other Messaging Systems:**

1. **Scalable:** Horizontal scaling is done by adding new brokers to the existing clusters.
2. **Fault Tolerance:** Kafka cluster can handle failures because of its distributed nature.
3. **Durable:** Kafka uses **"Distributed Commit Log"** Which means messages persists on disk as fast as possible.
4. **Performance:** Kafka has **high throughput** for both publishing and subscribing messages.
5. **No Data Loss:** It configure no data loss if we configure its properly.
6. **Zero Down Time:** It ensure zero down time when required number of brokers are present in the cluster.
7. **Reliability:** Kafka is reliable because it provides above feature.

## Kafka Provides:

### Core APIs:

- Producer APIs
- Consumer APIs
- Stream APIs
- Connector APIs
- Admin APIs



# Apache Kafka - Installation Steps

## Downloading the Required Files

- Download Server JRE according to your OS and CPU architecture from http://www.oracle.com/technetwork/java/javase/downloads/jre8-downloads-2133155.html
- Download and install 7-zip from http://www.7-zip.org/download.html
- Download and extract ZooKeeper using 7-zip from http://zookeeper.apache.org/releases.html
- Download and extract Kafka using 7-zip from http://kafka.apache.org/downloads.html

For this tutorial, we are assuming that ZooKeeper and Kafka are unzipped in the C: drive, but you can unzip them in any location.

Here, we are using full-fledged ZooKeeper and not the one packaged with Kafka because it will be a single-node ZooKeeper instance. If you want, you can run Kafka with a packaged ZooKeeper located in a Kafka package inside the *\kafka\bin\windows* directory.

## Step 1 - Verifying Java Installation

1. Start the JRE installation and hit the "Change destination folder" checkbox, then click 'Install.'
2. Change the installation directory to any path without spaces in the folder name. E.g., C:\Java\jre1.8.0_xx\. (By default, it will be C:\Program Files\Java\jre1.8.0_xx), then click 'Next.'
3. Now open the system environment variables dialogue by opening Control Panel -> System -> Advanced system settings -> Environment Variables.
4. Hit the New User Variable button in the User variables section, then type JAVA_HOME in *Variable name* and give your jre path in the *Variable value*
5. Now click OK.
6. Search for a Path variable in the "System Variable" section in the "Environment Variables" dialogue box you just opened.
7. Edit the path and type ";%JAVA_HOME%\bin" at the end of the text already written there
8. To confirm the Java installation, just open cmd and type "*java –version.*" You should be able to see the version of Java you just installed.

## Step 2 - ZooKeeper Framework Installation

1. Go to your ZooKeeper config directory. For me its C:\zookeeper-3.4.7\conf
2. Rename file "zoo_sample.cfg" to "zoo.cfg"
3. Open zoo.cfg in any text editor, like Notepad; I prefer Notepad++.
4. Find and edit dataDir=/tmp/zookeeper to :\zookeeper-3.4.7\data
5. Add an entry in the System Environment Variables as we did for Java.
   a. Add ZOOKEEPER_HOME = C:\zookeeper-3.4.7 to the System Variables.
   b. Edit the System Variable named "Path" and add ;%ZOOKEEPER_HOME%\bin;
6. You can change the default Zookeeper port in zoo.cfg file (Default port 2181).
7. Run ZooKeeper by opening a new cmd and type zkserver.

### Configuring the ZooKeeper Properties

The installer creates a configuration file named zoo.cfg, that contains the ZooKeeper properties. You can customize the default properties of ZooKeeper based on your environment and requirement.
For more information about the ZooKeeper properties, see the ZooKeeper documentation.

- Open the **zoo.cfg** file from the following directory:
  On Windows: <MDM Registry Edition Installation Directory>\bin
  On UNIX. <MDM Registry Edition Installation Directory>/bin

- Customize the following properties based on your requirement:
  **tickTime:** Length of a single tick, in milliseconds. ZooKeeper uses tick as the basic time unit to regulate timeouts. Default is 2000.
  **initLimit:** Number of ticks after which the ZooKeeper server times out during the synchronization process. Default is 10.
  **syncLimit:** Maximum number of ticks for the followers to wait to synchronize with the leader before the

followers time out. Default is 5.

**dataDir:** Absolute path for the ZooKeeper to store the in-memory snapshots of the data and the transactional log of updates to the database.

**clientPort:** Port on which the ZooKeeper server listens for client connections.

**maxClientCnxns:** Maximum number of concurrent connections that a single client can make to a single member of the ZooKeeper ensemble. Default is 10.

- If you use an ensemble of ZooKeeper servers, perform the following tasks:
  A. Add the following server entries to the zoo.cfg file:

```
server.1=<Host Name1>:<Port Number1>:<Port Number2>
server.2=<Host Name1>:<Port Number1>:<Port Number2>
server.3=<Host Name1>:<Port Number1>:<Port Number2>
...
server.N=<Host Name1>:<Port Number1>:<Port Number2>
```

Host Name refers to the host name of the ZooKeeper server, and Port Numbers 1 and 2 refers to the ports that the ZooKeeper uses to communicate between the ZooKeeper servers.

For example, the following entries indicate an ensemble of three ZooKeeper servers:

```
server.1=PriHost:18046:18047
server.2=SecHost:18049:18050
server.3=TerHost:18052:18053
```

B. Create a file named myid in the directory that you configure for the dataDir property in the zoo.cfg file.
C. Open the myid file, specify the ID of the server based on the server.<Number> entry in the zoo.cfg file, and save the myid file.
For example, if the host name is specified as server.2 in the zoo.cfg file, specify 2 in the myid file.
D. In the other ZooKeeper servers, add the same server entries in the zoo.cfg file, create the myid file, and enter the corresponding server ID to the myid file.

### Starting and Stopping a ZooKeeper Server

- the MDM Registry Edition servers, the ZooKeeper server stops.
- You can run the `zkServer` script located in the following directory to manually start or stop the ZooKeeper Server:
  On Windows: <MDM Registry Edition Installation Directory>\bin
  On UNIX. <MDM Registry Edition Installation Directory>/bin
- Use the following command to manually start the ZooKeeper server:
  **`zkServer start`**

**NOTE:** If you use an ensemble of ZooKeeper servers, manually start all the ZooKeeper servers before you start the MDM Registry Edition servers.

- Use the following command to manually stop the ZooKeeper server:
  **`zkServer stop`**

## Step 3 - Setting Up Kafka

1. Go to your Kafka config directory. For me its C:\kafka_2.11-0.9.0.0\config
2. Edit the file "server.properties."
3. Find and edit the line log.dirs=/tmp/kafka-logs" to "log.dir= C:\kafka_2.11-0.9.0.0\kafka-logs.
4. If your ZooKeeper is running on some other machine or cluster you can edit "zookeeper.connect:2181" to your custom IP and port. For this demo, we are using the same machine so there's no need to change. Also, the Kafka port and broker.id are configurable in this file. Leave other settings as it is.
5. Your Kafka will run on default port 9092 and connect to ZooKeeper's default port, 2181.

## Step 4 - Running a Kafka Server

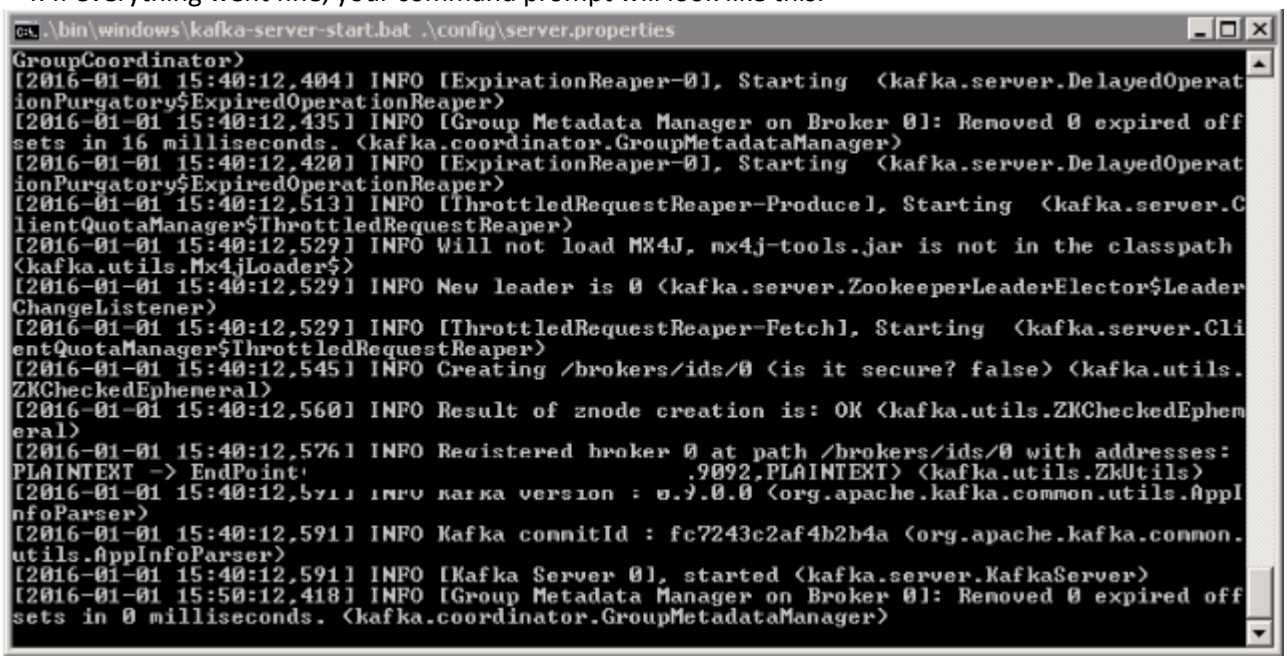
Important: Please ensure that your ZooKeeper instance is up and running before starting a Kafka server.

1. Go to your Kafka installation directory: C:\kafka_2.11-0.9.0.0\

2. Open a command prompt here by pressing Shift + right click and choose the "Open command window here" option).
3. Now type .\bin\windows\kafka-server-start.bat .\config\server.properties and press Enter.

```
.\bin\windows\kafka-server-start.bat .\config\server.properties
```


4. If everything went fine, your command prompt will look like this:

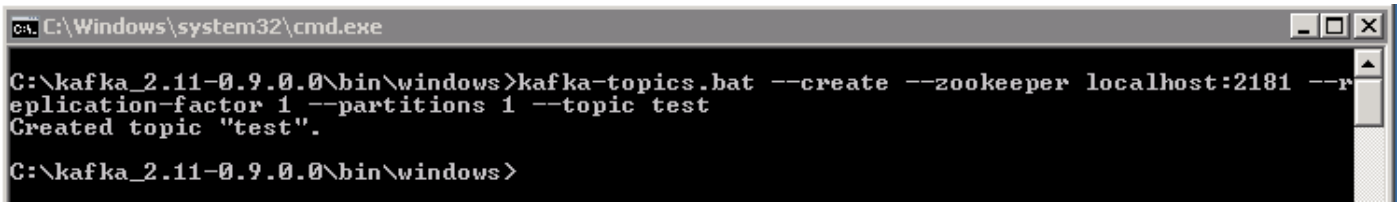



5. Now your Kafka Server is up and running, you can create topics to store messages. Also, we can produce or consume data from Java or Scala code or directly from the command prompt.

## Step 5 - Creating Topics

1. Now create a topic with the name "test" and a replication factor of 1, as we have only one Kafka server running. If you have a cluster with more than one Kafka server running, you can increase the replication-factor accordingly, which will increase the data availability and act like a fault-tolerant system.
2. Open a new command prompt in the location C:\kafka_2.11-0.9.0.0\bin\windows.
3. Type the following command and hit Enter:

```
kafka-topics.bat --create --zookeeper localhost:2181 --replication-factor 1 --
partitions 1 --topic test
```




## Step 6 - Creating a Producer and Consumer to Test Server

1. Open a new command prompt in the location C:\kafka_2.11-0.9.0.0\bin\windows
2. To start a producer type the following command:

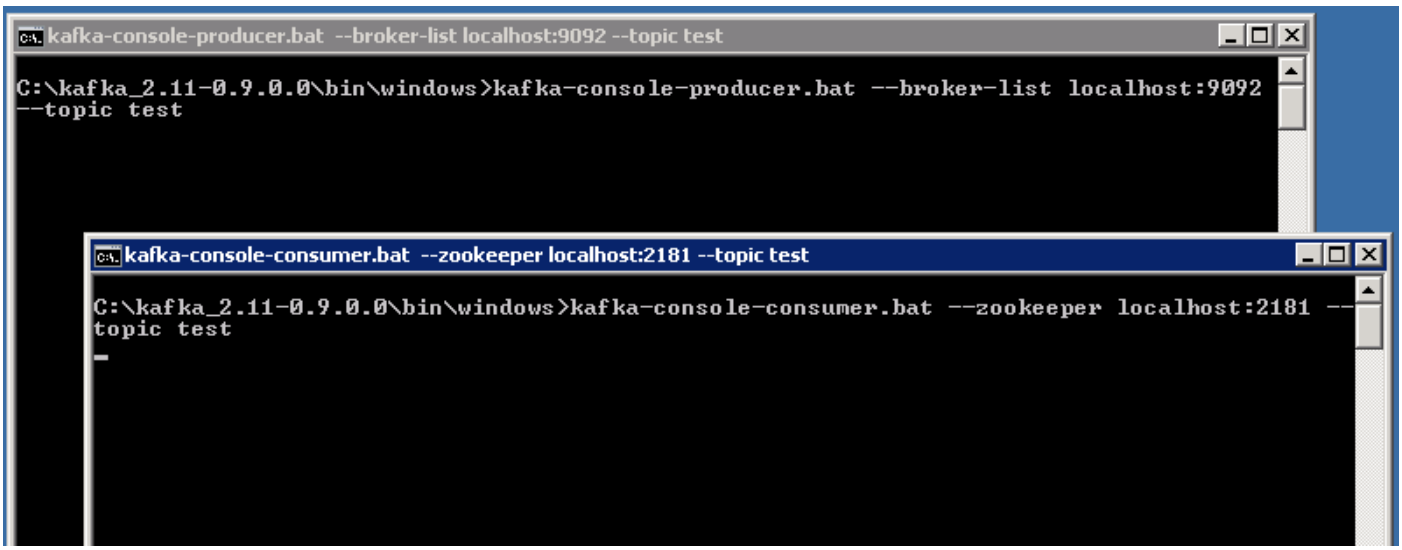
```
kafka-console-producer.bat --broker-list localhost:9092 --topic test
```

3. Again open a new command prompt in the same location as C:\kafka_2.11-0.9.0.0\bin\windows

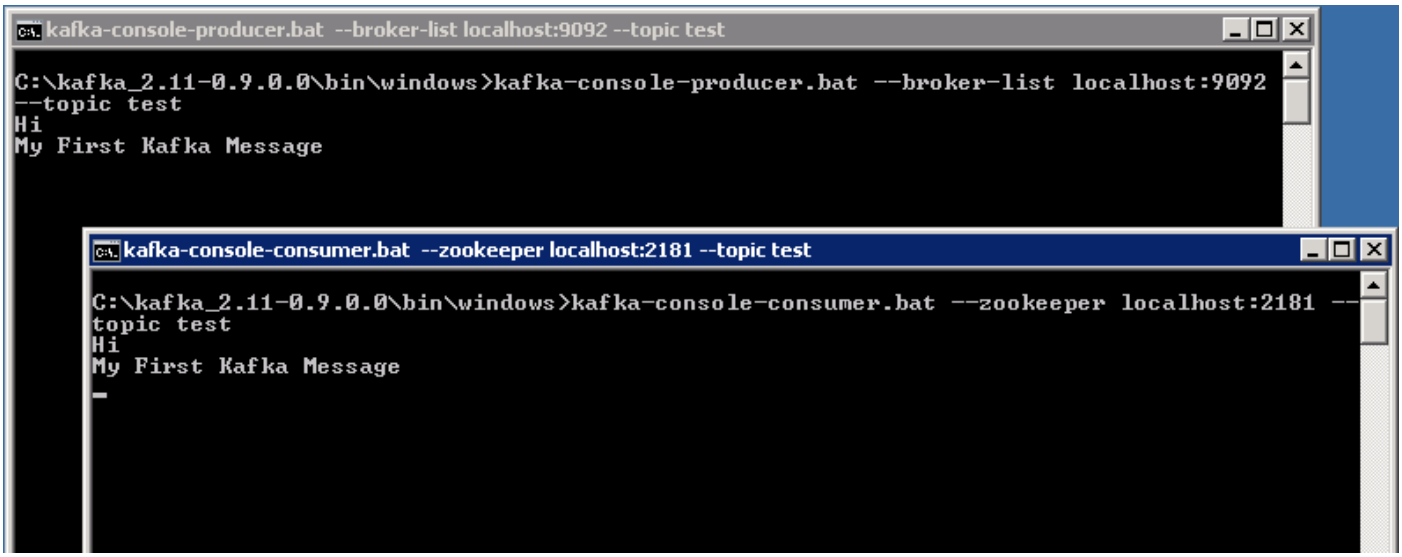
4. Now start a consumer by typing the following command:
   After kafka version 2.0 (>= 2.0):

```
kafka-console-consumer.bat --bootstrap-server localhost:9092 --topic test
```

5. Now you will have two command prompts, like the image below:




6. Now type anything in the producer command prompt and press Enter, and you should be able to see the message in the other consumer command prompt.




8. If you are able to push and see your messages on the consumer side, you are done with Kafka setup.

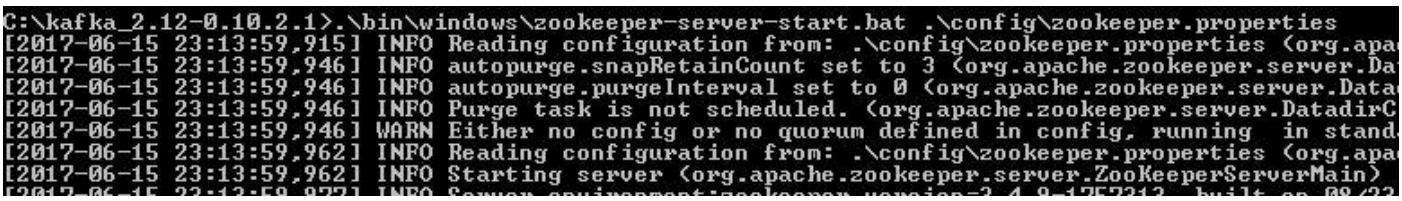
## Some Other Useful Commands

1. List Topics: kafka-topics.bat --list --zookeeper localhost:2181
2. Describe Topic: kafka-topics.bat --describe --zookeeper localhost:2181 --topic [Topic Name]
3. Read messages from the beginning
    a. Before version < 2.0: kafka-console-consumer.bat --zookeeper localhost:2181 --topic [Topic Name] --from-beginning
    b. After version > 2.0: kafka-console-consumer.bat --bootstrap-server localhost:9092 --topic [Topic Name] --from-beginn
4. Delete Topic: kafka-run-class.bat kafka.admin.TopicCommand --delete --topic [topic_to_delete] --zookeeper localhost:2181

## We will now start Apache Kafka-

- This Kafka installation comes with an inbuilt zookeeper. Zookeeper is mainly used to track status of nodes present in Kafka cluster and also to keep track of Kafka topics, messages, etc.
  Open a command prompt and start the Zookeeper-

```
C:\kafka_2.12-0.10.2.1>.\bin\windows\zookeeper-server-start.bat
.\config\zookeeper.properties
```




- Open a new command prompt and start the Apache Kafka-

```
C:\kafka_2.12-0.10.2.1>.\bin\windows\kafka-server-start.bat .\config\server.properties
```


- Open a new command prompt and create a topic with name javainuse-topic, that has only one partition & one replica.

```
C:\kafka_2.12-0.10.2.1>.\bin\windows\kafka-topics.bat --create --zookeeper
localhost:2181 --replication-factor 1 --partitions 1 --topic javainuse-topic
```


- Next Open a new command prompt and create a producer to send message to the above created javainuse-topic and send a message - Hello World Javainuse to it-

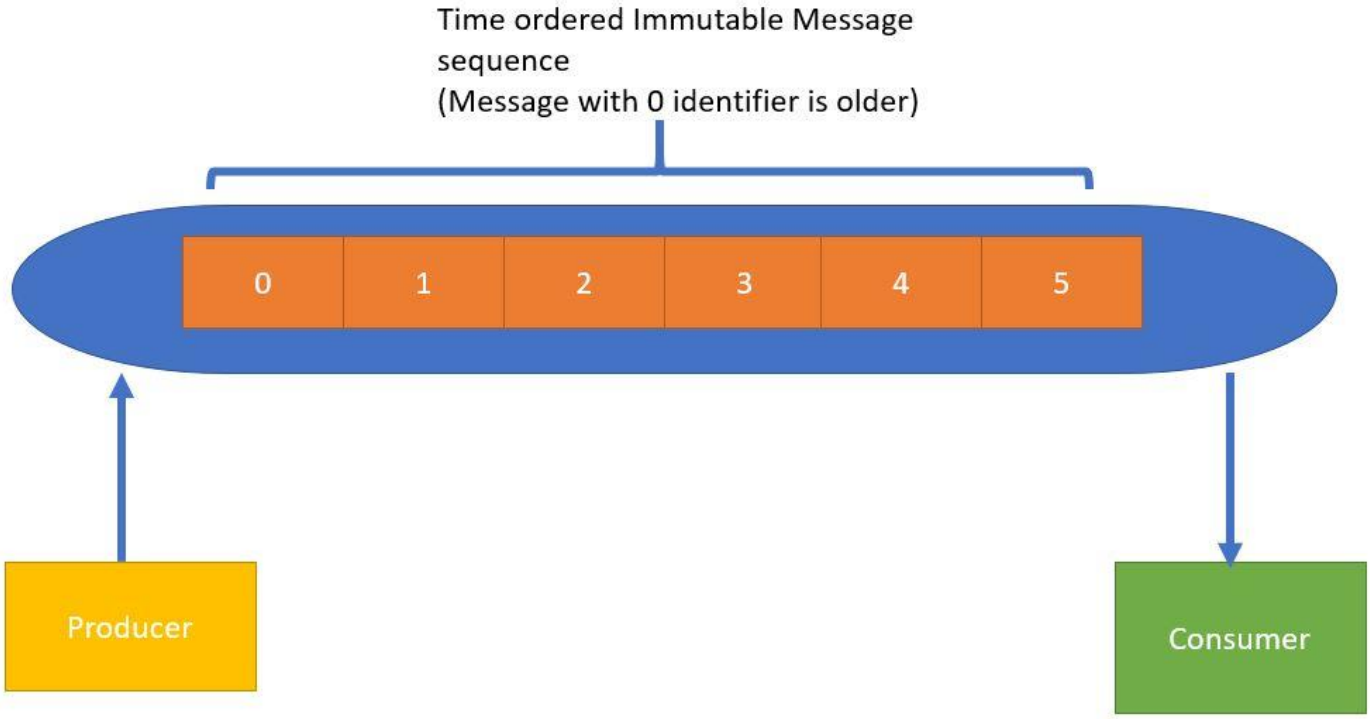
```
C:\kafka_2.12-0.10.2.1>.\bin\windows\kafka-console-producer.bat --broker-list
localhost:9092 --topic javainuse-topic
Hello World Javainuse
```


- Finally Open a new command prompt and start the consumer which listens to the topic javainuse-topic we just created above. We will get the message we had sent using the producer

```
C:\kafka_2.12-0.10.2.1>.\bin\windows\kafka-console-consumer.bat --
```

## Internal Working of Apache Kafka

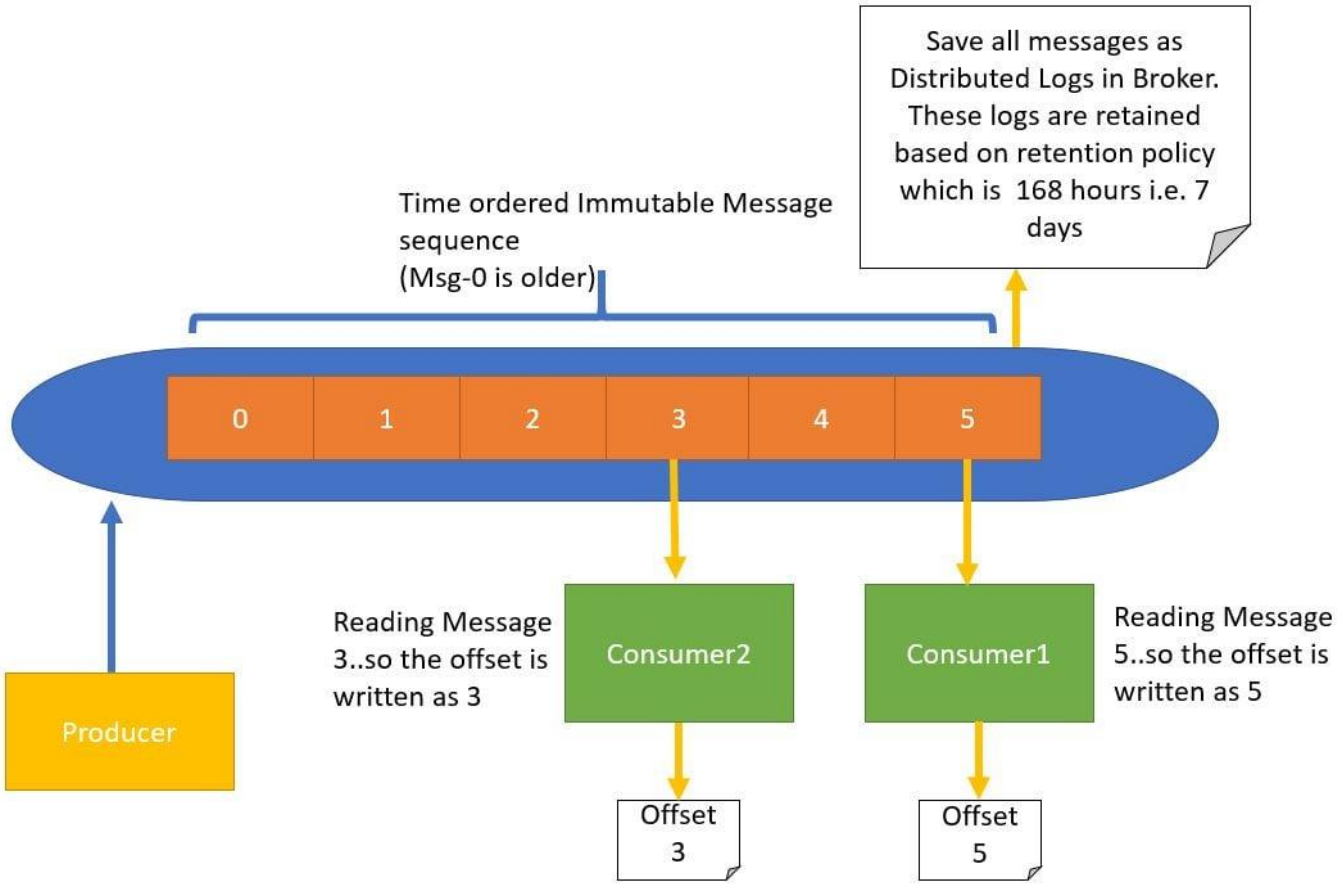
Topics are the base abstraction of where data lives within Kafka. They can be considered similar to the concept of table in a database. Each topic is backed by logs which are partitioned and distributed.



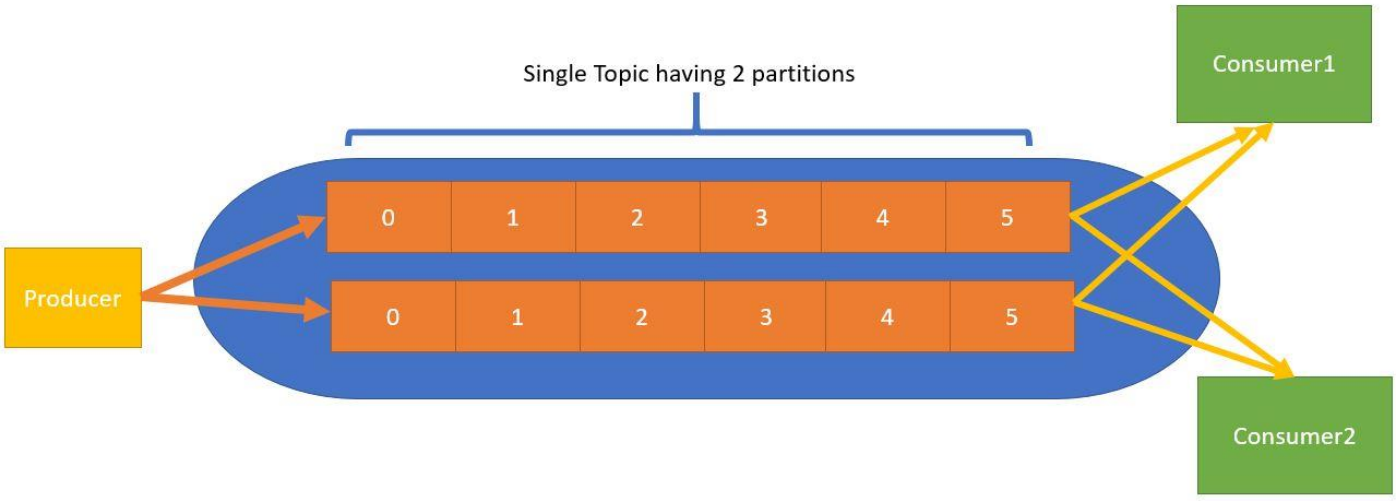

Each Message has a timestamp Identifier Binary Payload


Message offset is last read message position. Consumer is responsible of keeping

track of message consumed. Consumer keeps track of the successfully processed message. apache kafka retains all published messages. The default is 168




Each Topic has one or more partitions Partition allows - Scale Fault tolerant Achieve higher level of throughput



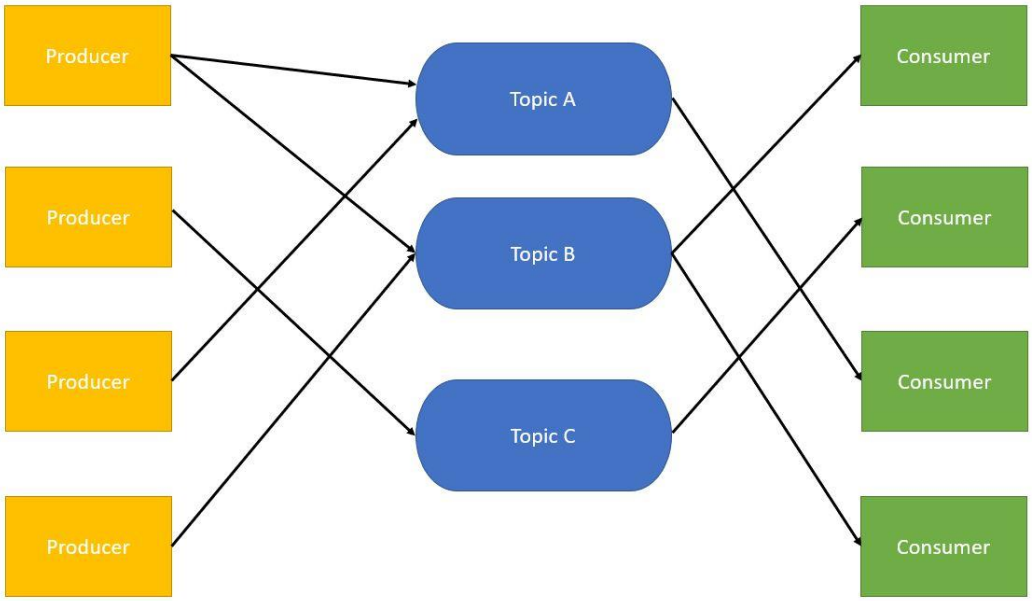

Each topic should have minimum 1 partition

## Understanding Apache Kafka Architecture

# Apache Kafka Topic

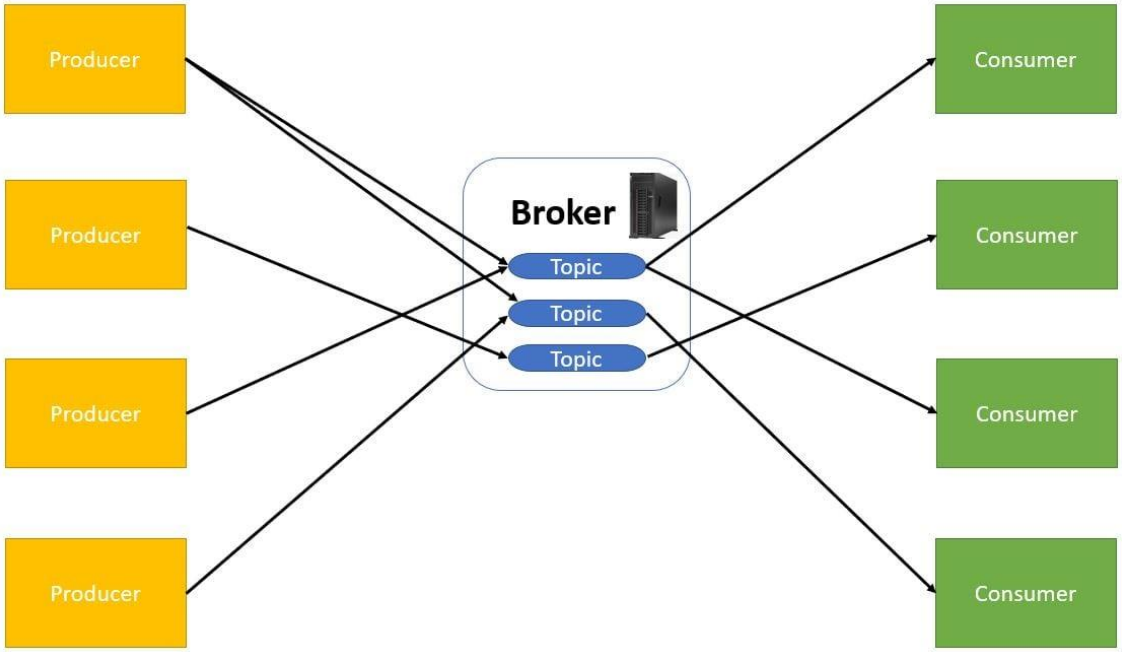
Apache Kafka is a messaging system where messages are sent by producers and these messages are consumed by one or more consumers. Producers send the messages to Apache Kafka Topics. From the topics these messages are then consumed by the consumers. Topics have unique names which are used by producers and consumers for sending/consuming messages.

Topics are the base abstraction of where data lives within Kafka. They can be considered similar to the concept of table in a database. Each topic is backed by logs which are partitioned and distributed.

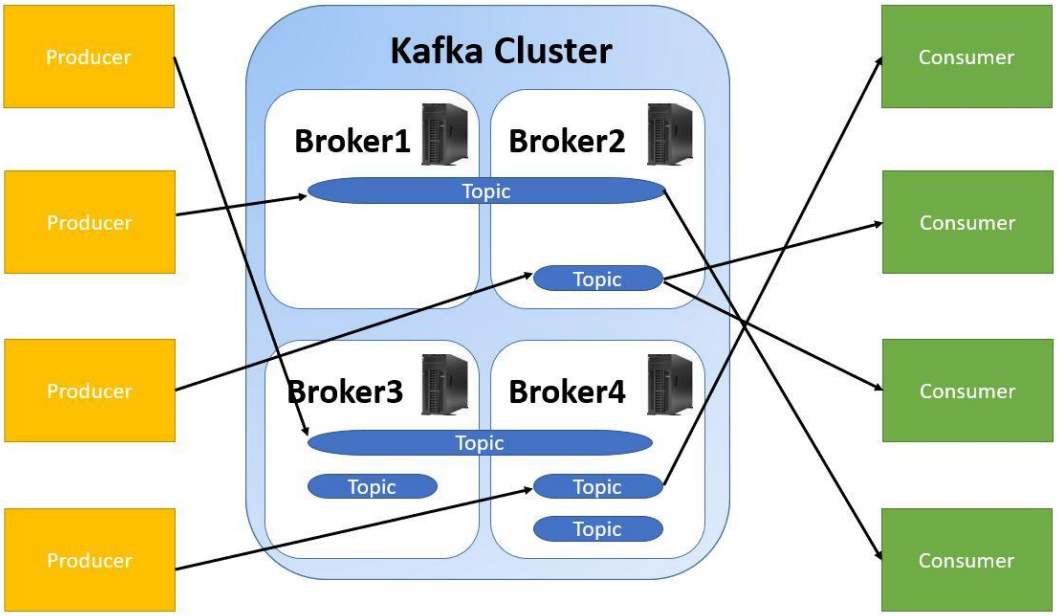



# Apache Kafka Broker

The physical/virtual machines or servers where topics reside are called brokers. Kafka Broker is a software process that runs on machine. Broker has access to resources such as file systems where logs are maintained. A single topic can have multiple partitions running on different brokers.



We can add more brokers to scale the Kafka Application.

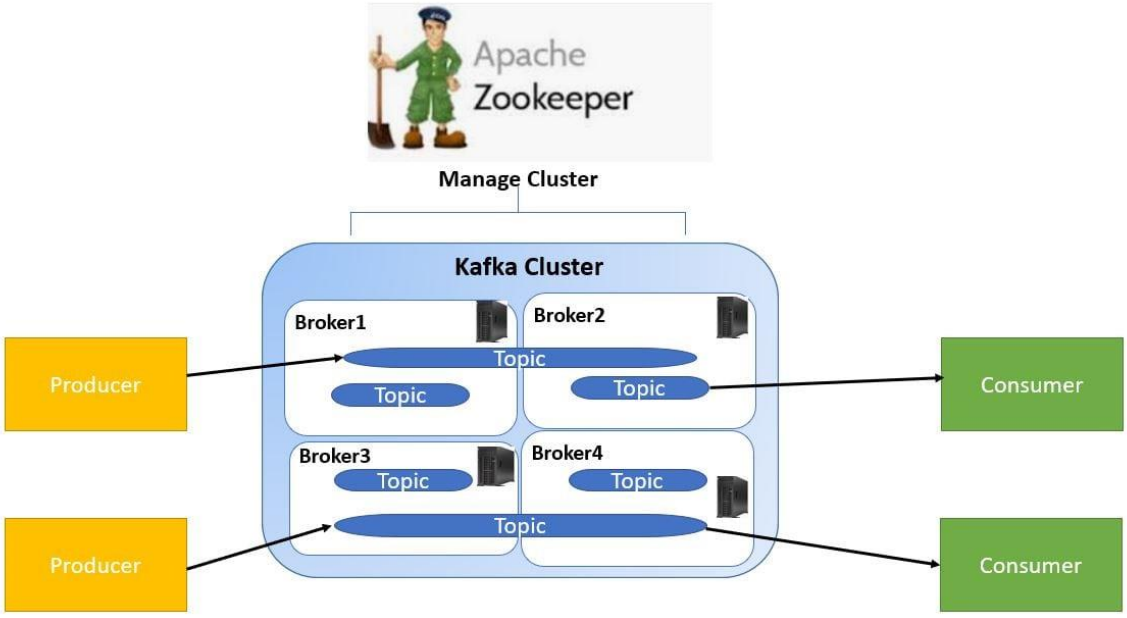



The advantages of using Apache Kafka Cluster are as follows -

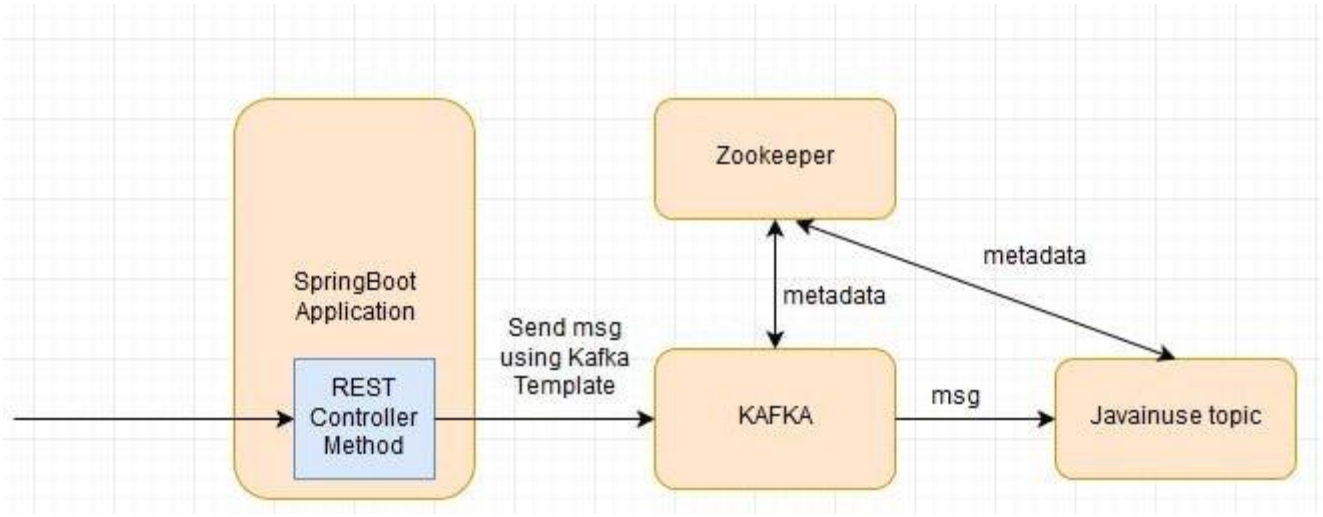
- **Clustering -** Apache Kafka has a clustered set of brokers that work in unison.
- **Distributed -** The data in Apache Kafka is distributed in all the brokers. This is done by making use of partitions which are distributed across multiple brokers.
- **Fault Tolerant -** Apache Kafka maintains replicated copies of data. So if any broker in the cluster goes down, it does not affect the working of Apache Kafka Cluster. This is done by setting the replication value to greater than 1.
- **Application scaling -** Apache Kafka can be scaled horizontally. This increases the throughput.

# Apache Zookeeper

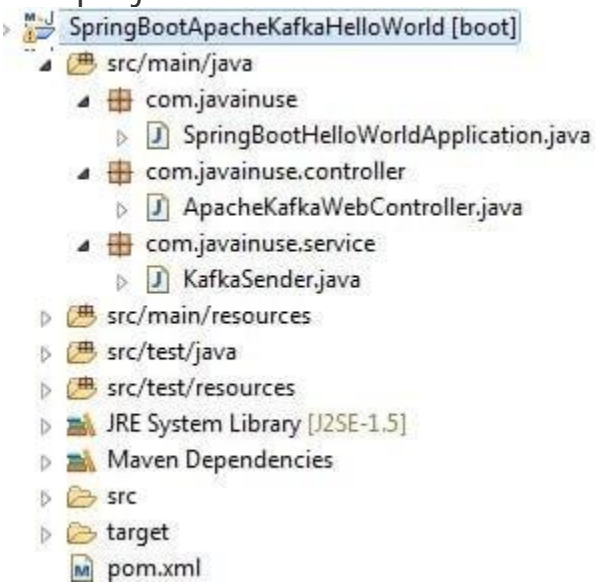
To manage the cluster, we make use of Apache Zookeeper. Apache Zookeeper is a coordination service for distributed application that enables synchronization across a cluster. Zookeeper can be viewed as centralized repository where distributed applications can put data and get data out of it. It is used to keep the distributed system functioning together as a single unit, using its synchronization, serialization and coordination goals, selecting leader node.



# Spring Boot + Apache Kafka Hello World Example




The project will be as follows-




Define the pom.xml as follows- Add the **spring-kafka** dependency.

```
<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.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.javainuse</groupId>
  <artifactId>SpringBootApacheKafkaHelloWorld</artifactId>
  <version>0.0.1-SNAPSHOT</version>

  <packaging>jar</packaging>


	<parent>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-parent</artifactId>
		<version>1.5.3.RELEASE</version>
		<relativePath/>
	</parent>

	<properties>
		<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
		<project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
		<java.version>1.8</java.version>
```

```
	</properties>

	<dependencies>
		<dependency>
			<groupId>org.springframework.kafka</groupId>
			<artifactId>spring-kafka</artifactId>
		</dependency>
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-web</artifactId>
		</dependency>
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-test</artifactId>
			<scope>test</scope>
		</dependency>
	</dependencies>

	<build>
		<plugins>
			<plugin>
				<groupId>org.springframework.boot</groupId>
				<artifactId>spring-boot-maven-plugin</artifactId>
			</plugin>
		</plugins>
	</build>

</project>
```

Define the KafkaSender class to send message to the kafka topic named as java_in_use-topic.

```
package com.javainuse.service;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.kafka.core.KafkaTemplate;
import org.springframework.stereotype.Service;

@Service
public class KafkaSender {

	@Autowired
	private KafkaTemplate<String, String> kafkaTemplate;

	String kafkaTopic = "java_in_use_topic";

	public void send(String message) {

	    kafkaTemplate.send(kafkaTopic, message);
	}
}
```

Define a Controller which will pass the message and trigger the send message to the Kafka Topic using the KafkaSender class.

```
package com.javainuse.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

import com.javainuse.service.KafkaSender;
```

```
@RestController
@RequestMapping(value = "/javainuse-kafka/")
public class ApacheKafkaWebController {

	@Autowired
	KafkaSender kafkaSender;

	@GetMapping(value = "/producer")
	public String producer(@RequestParam("message") String message) {
		kafkaSender.send(message);

		return "Message sent to the Kafka Topic java_in_use_topic Successfully";
	}

}
```

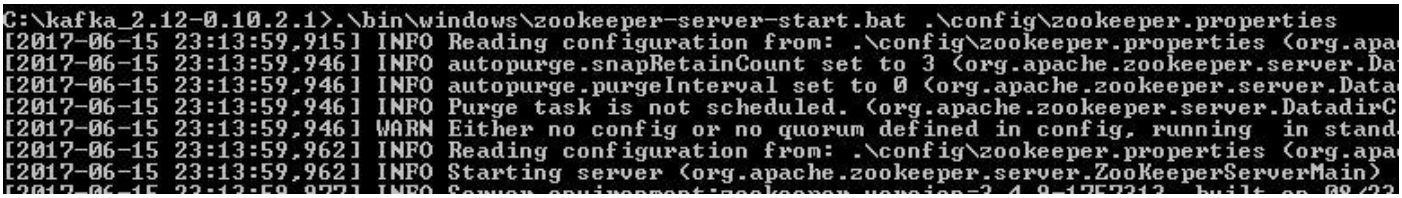
Finally Define the Spring Boot Class with @SpringBootApplication annotation

```
package com.javainuse;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class SpringBootHelloWorldApplication {

	public static void main(String[] args) {

		SpringApplication.run(
				new Object[] { SpringBootHelloWorldApplication.class }, args)
;
	}
}
```

We are done with the required Java code. Now lets start Apache Kafka. As we had explained in detail in the Getting started with Apache Kafka perform the following.

- **Start Apache Zookeeper-**

```
C:\kafka_2.12-0.10.2.1>.\bin\windows\zookeeper-server-start.bat .\config\zookeeper.prope
rties
```



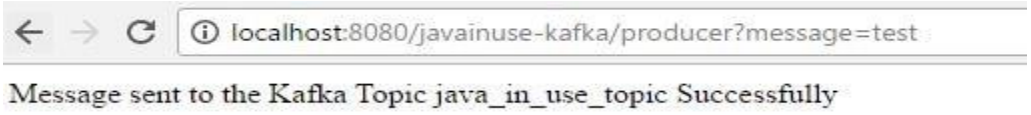

- **Start Apache Kafka-**

```
C:\kafka_2.12-0.10.2.1>.\bin\windows\kafka-server-start.bat .\config\server.properties
```


- Next start the Spring Boot Application by running it as a Java Application.
- Also Start the consumer listening to the java_in_use_topic-

```
C:\kafka_2.12-0.10.2.1>.\bin\windows\kafka-console-consumer.bat --bootstrap-server local
host:9092 --topic java_in_use_topic --from-beginning
```

- Finally hit the url as follows- http://localhost:8080//javainuse-kafka/producer?message=test




- This will trigger the message to be sent to the java_in_use_topic. We can see in the consumer started the message is receive