**Create a topic named:**

~/kafka/bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic TutorialTopic

**Push message**: echo "Hello, World" | ~/kafka/bin/kafka-console-producer.sh --broker-list localhost:9092 --topic TutorialTopic > /dev/null

The following command consumes messages from TutorialTopic. Note the use of the --from-beginning flag, which allows the consumption of messages that were published before the consumer was started:

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

**View list topic**: ./kafka-topics.sh --list --zookeeper localhost:2181

## **Install KafkaT (Optional)**

### **Publish-Subscribe Messaging System**

1. Consumers can subscribe to one or more topic and consume all the messages in that topic

## **What is Kafka?**

* Kafka provides both pub-sub and queue based messaging system in a fast, reliable, persisted and zero downtime manner.
* Kafka is suitable for both offline and online message . Kafka messages are persisted on the disk and replicated within the cluster to prevent data loss. Kafka is built on top of the ZooKeeper synchronization service.
* Producers simply send the message to a topic and consumer can choose any one type of messaging system depending on their need.

**Benefits:**

* Easy that extend − Kafka messaging system easily that change without down time
* Performance − Kafka has high throughput for both publishing and subscribing messages. It maintains stable performance even many TB of messages are stored
* Kafka is very fast and nearly no die time and zero data loss

**Use:**

* Log Aggregation Solution − Kafka can be used across an organization to collect logs from multiple services and make them available in a standard format to multiple con-sumers.
* Stream Processing: Streaming read data from a topic, processes it, and write processed data to a new topic where it becomes available for users and applications. Kafka’s strong durability is also very useful in the context of stream processin

**Topics**

* A stream of messages belonging to a particular category is called a topic. Data is stored in topics.
* Topics are split into partitions. For each topic, Kafka keeps a mini-mum of one partition

**Partition**

* Topics may have many partitions, so it can handle an arbitrary amount of data.

**Partition offset**

* Each partitioned message has a unique sequence id called as offset

**Replicas of partition**

* Replicas are never read or write data. They are used to prevent data loss.

**Brokers**

* Brokers are simple system responsible for maintaining the pub-lished data. Each broker may have zero or more partitions per topic. Assume, if there are N partitions in a topic and N number of brokers, each broker will have one partition.

**Kafka Cluster**

* Kafka’s having more than one broker are called as Kafka cluster. A Kafka cluster can be expanded without downtime. These clusters are used to manage the persistence and replication of message data.

**Producers**

* Producers are the publisher of messages to one or more Kafka topics. Producers send data to Kafka brokers. Every time a producer pub-lishes a message to a broker. Actually, the message will be appended to a partition. Producer can also send messages to a partition of their choice.

**Consumers**

* Consumers read data from brokers. Consumers subscribes to one or more topics and consume published messages by pulling data from the brokers.

**Leader**

* Leader is the node responsible for all reads and writes for the given partition. Every partition has one server acting as a leader.

# Apache Kafka - Cluster Architecture

**Broker**

* Kafka cluster typically consists of multiple brokers to maintain load balance. They use ZooKeeper for maintaining their cluster state. One Kafka broker instance can handle hundreds of thousands of reads and writes per second and each bro-ker can handle TB of messages without performance impact. Kafka broker can be done by ZooKeeper.

**ZooKeeper**

* ZooKeeper is used for managing and coordinating Kafka broker. ZooKeeper service is mainly used to notify producer and consumer about the presence of any new broker in the Kafka system or failure of the broker in the Kafka system.

**Producers**

* Producers push data to brokers. When the new broker is started, all the producers search it and automatically sends a message to that new broker.

**Workflow of Pub-Sub Messaging**

1. Producers send message to a topic in a regular interval.
2. Kafka broker stores all messages in the partitions configured for that particular topic. It ensures the messages are equally shared between partitions. If the producer sends two messages and there are two partitions, Kafka will store one message in the first partition and the second message in the second partition.
3. Consumer subscribes to a specific topic.
4. Once the consumer subscribes to a topic, Kafka will provide the current offset of the topic to the consumer and also saves the offset in the Zookeeper ensemble.
5. Consumer will request the Kafka for new messages.
6. Once Kafka receives the messages from producers, it forwards these messages to the consumers.
7. Consumer will receive the message and process it.
8. Once the messages are processed, consumer will send a ping to the Kafka broker.
9. Once Kafka receives an ping, it changes the offset to the new value and updates it in the Zookeeper.
10. This above flow will repeat until the consumer stops the request.

## **Workflow of Queue Messaging / Consumer Group**

A group of consumers having the same Group ID will subscribe to a topic and the messages are shared among them

1. Producers send message to a topic in a regular interval.
2. Kafka stores all messages in the partitions configured
3. A single consumer subscribes to a specific topic, assume Topic-01 with Group ID as Group-1.
4. Kafka interacts with the consumer in the same way as Pub-Sub Messaging until new consumer subscribes the same topic, Topic-01 with the same Group ID as Group-1.
5. Once the new consumer arrives, Kafka switches its operation to share mode and shares the data between the two consumers. T
6. The number of consumer exceeds the number of partitions, the new consumer will not receive any further message until any one of the existing consumer unsubscribes.

## **Role of ZooKeeper**

* A critical dependency of Apache Kafka is Apache Zookeeper, which is a distributed configuration and synchronization service. Zookeeper serves as the coordination interface between the Kafka brokers and consumers. The Kafka servers share information via a Zookeeper cluster. Kafka stores basic metadata in Zookeeper such as information about topics, brokers, consumer offsets
* Since all the critical information is stored in the Zookeeper and it normally replicates this data across its ensemble, failure of Kafka broker / Zookeeper does not affect the state of the Kafka cluster.

Error: port already in use

sudo lsof -i :2181

sudo kill -9 PID

## **Single Node-Single Broker Configuration**

## **Start ZooKeeper**

Open a new terminal and type the following command −

./bin/zookeeper-server-start.sh config/zookeeper.properties

To start Kafka Broker, type the following command −

./bin/kafka-server-start.sh config/server.properties

## **List of Topics**

./bin/kafka-topics.sh --list --zookeeper localhost:2181

**Creating a Kafka Topic:**

bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1

--partitions 1 --topic topic-name

Once the topic has been created, you can get the notification in Kafka broker terminal window and the log for the created topic specified in “/tmp/kafka-logs/“ in the config/server.properties file.\

### **Start Producer to Send Messages**

./bin/kafka-console-producer.sh --broker-list localhost:9092 --topic topic-name

### **Start Consumer to Receive Messages**

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

## **Single Node-Multiple Brokers Configuration**

The Describe command is used to check which broker is listening on the current created topic as shown below −

bin/kafka-topics.sh --describe --zookeeper localhost:2181

--topic Multibrokerappli-cation

### **Modifying a Topic**

./bin/kafka-topics.sh —zookeeper localhost:2181 --alter --topic topic\_name --parti-tions count

**Apache Kafka**

* tốc độ nhanh: Với một máy đơn cài đặt Kafka có thể xử lý số lượng dữ liệu từ việc đọc và ghi lên tới hàng trăm megabyte trong một giây từ hàng ngàn máy khách.
* Khả năng mở rộng: Kafka được thiết kế cho phép dễ dàng được mở rộng và trong suốt với người dùng (nghĩa là không có thời gian chết – ngừng hoạt động trong khi thêm một nút máy chủ mới vào cụm). Khi Kafka chạy trên một cụm, luồng dữ liệu sẽ được phân chia và được vận chuyển tới các nút trong cụm, do đó cho phép trung chuyển các dữ liệu mà có khối lượng lớn hơn nhiều so với sức chứa của một máy đơn.
* Độ tin cậy: Dữ liệu vào hàng đợi sẽ được lưu trữ trên ổ đĩa và được sao chép tới các nút khác trong cụm để ngăn ngừa việc mất dữ liệu, như vậy Kafka đảm bảo tính chịu lỗi cao.

## Topic

Mỗi partition là một **chuỗi log**, **có thứ tự** và **không thể thay đổi** (immutable).

Mỗi message trong partition sẽ có id tăng dần , gọi là offset

Kafka cluster sẽ lưu lại mọi message đã được published, cho dù message đó đã được/chưa được sử dụng (consume). Thời gian lưu message có thể tuỳ chỉnh được thông qua log retention.

Một điểm thú vị là Consumer sẽ điều khiển những gì mình muốn đọc thông qua offset của message, hay thậm chí là thứ tự đọc. Consumer có thể reset lại vị trí của offset để re-process lại một vài message nào đó.

## Producer

Như đã nói ở trên, producer nhằm mục đích chính là ném message vào topic. Cụ thể hơn là producer có nhiệm vụ là **chọn** message nào, để **ném vào** partition nào trong topic. Nhiệm vụ này rất quan trọng, giúp cho kafka có khả năng "scale" tốt

* Queue: Một message sẽ được xử lý bời một consumer
* Pub/Sub: Một message sẽ được xử lý bởi **một vài** consumer thích hợp, tuỳ theo topic

[Kafka Manager](https://github.com/yahoo/kafka-manager).

#### Nên bật bao nhiêu broker trong 1 cụm kafka?

Càng nhiều broker thì khả năng chịu lỗi sẽ càng cao vì số lượng replicate cho 1 partition sẽ phụ thuộc vào số lượng broker. Nhưng đồng thời thì cũng tốn chi phí hơn và control phức tạp hơn. Giả sử bạn đang mong muốn một cụm brokers có khả năng lưu trữ 20T thì nên chia đều mỗi broker có cùng 1 lượng lưu trữ, giả sử như 4T. Vậy ta sẽ cần 5 broker.

#### 2. Khi tạo topic thì nên set số lượng partition và replicate là bao nhiêu?

* Số partition sẽ quyết định khả năng xử lý song song của bạn, càng nhiều xử lý càng nhanh. Nhưng nó cũng đẻ ra nhiều vấn đề. Ae có thể đọc tại link này: <https://www.confluent.io/blog/how-choose-number-topics-partitions-kafka-cluster> nó dài quá mình không thể tóm lược được
* Số replicate càng cao thì tăng khả năng chịu lỗi càng lớn, nhưng nó lại bị phụ thuộc vào số broker. Replicate càng lớn thì có thể tăng về số RAM và disk sử dụng của server Kafka.