**Apache Kafka & ZooKeeper**

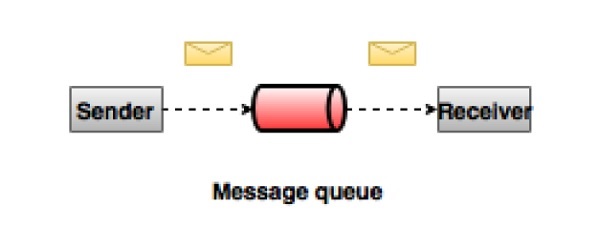
* Apache Kafka was originated at LinkedIn and later became an open sourced Apache project in 2011, then First-class Apache project in 2012.
* Kafka is written in Scala and Java.
* Apache Kafka is **publish-subscribe** (**pub-sub**) based fault tolerant messaging system. It is fast, scalable and distributed by design.
* Kafka is designed for distributed high throughput systems.

# Need of Message system:

* A Messaging System is responsible for transferring data from one application to another, so the applications can focus on data, but not worry about how to share it.
* Distributed messaging is based on the concept of reliable message queuing.
* Messages are queued asynchronously between client applications and messaging system.
* Types of messaging patterns:
  + Point to point and
  + Publish-subscribe (pub-sub) messaging system.
* Most of the messaging patterns follow **pub-sub**.

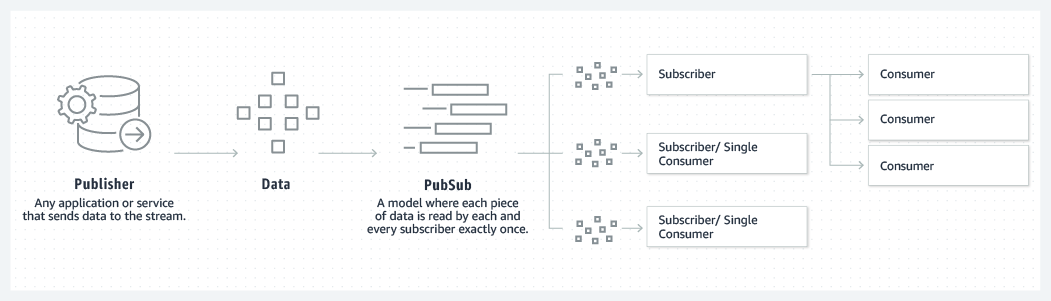
### **Point to Point Messaging System**

* In a point-to-point system, messages are persisted in a queue.
* One or more consumers can consume the messages in the queue, but a particular message can be consumed by a maximum of one consumer only.
* Once a consumer reads a message in the queue, it disappears from that queue.



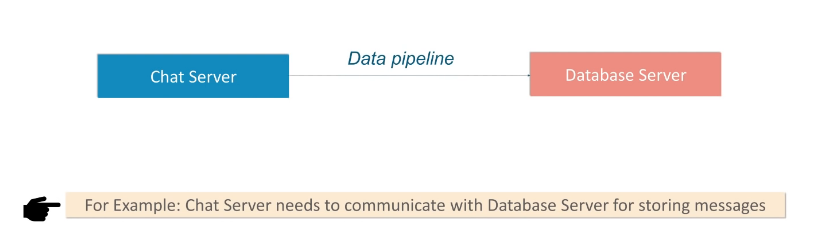
### **Publish-Subscribe Messaging System**

* In the publish-subscribe system, messages are persisted in a **topic**.
* Unlike point-to-point system, consumers can subscribe to one or more topic and consume all the messages in that topic.
* In the Publish-Subscribe system, message producers are called publishers and message consumers are called subscribers.
* **Example** is Dish TV, which publishes different channels like sports, movies, music, etc., and anyone can subscribe to their own set of channels and get them whenever their subscribed channels are available.

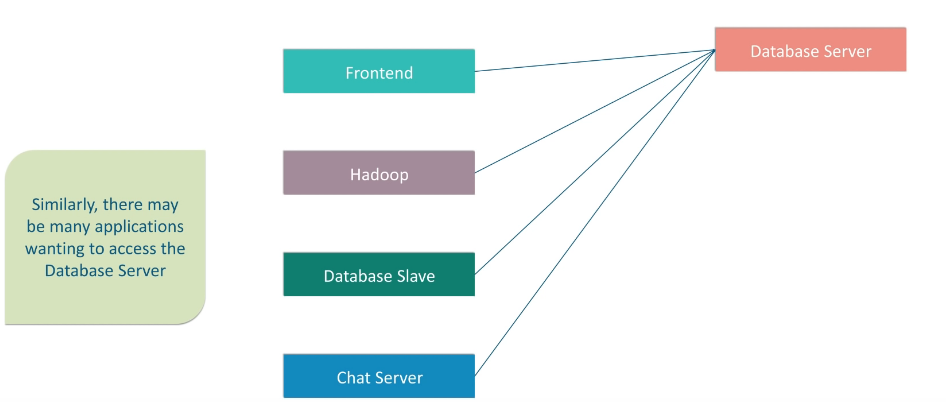


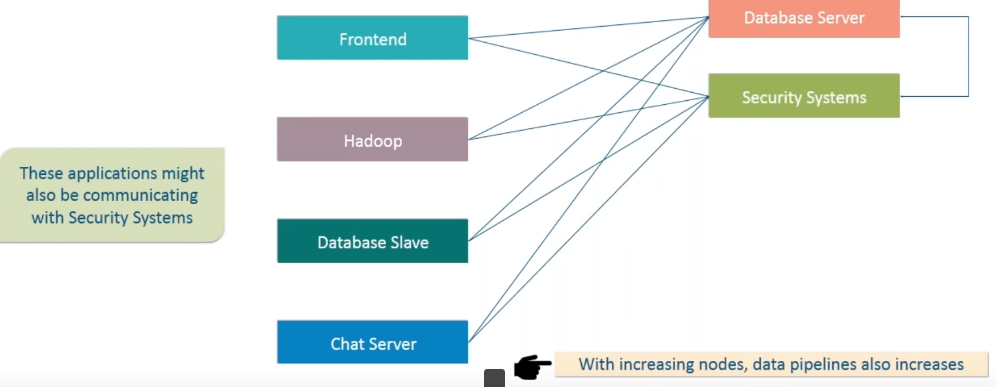
## **1.1 Data Pipeline:**

* Communication between different systems which is done by using data pipeline

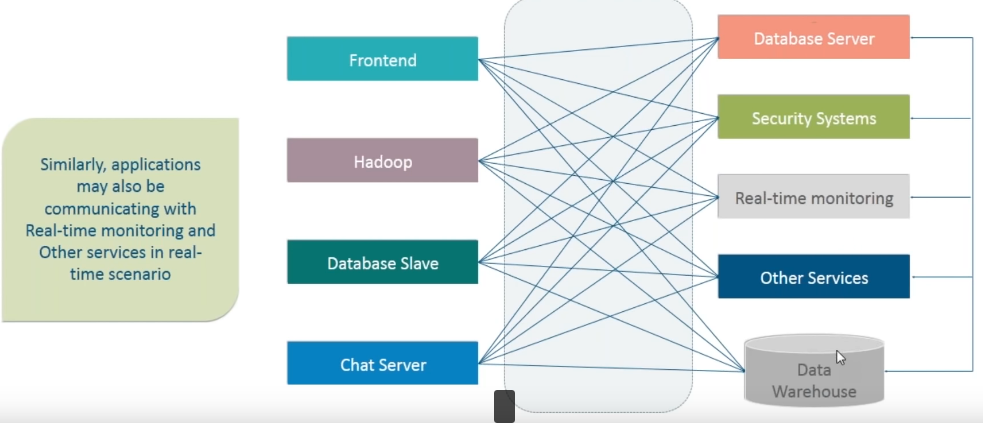


* Increase no.of nodes

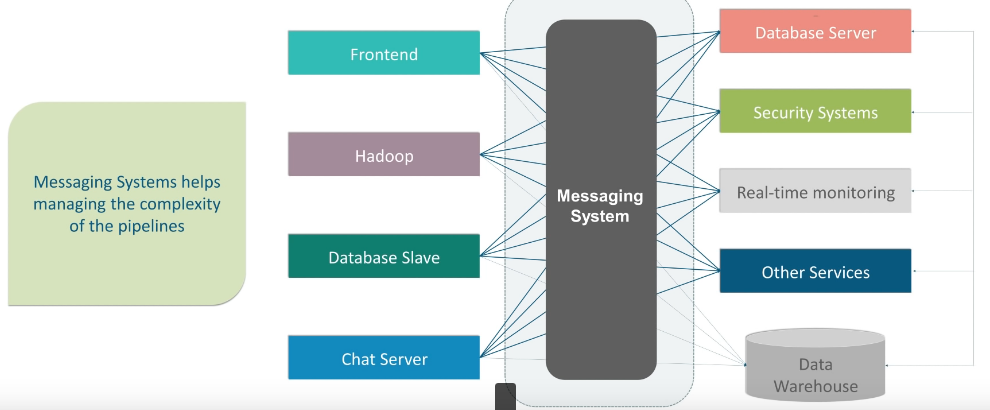




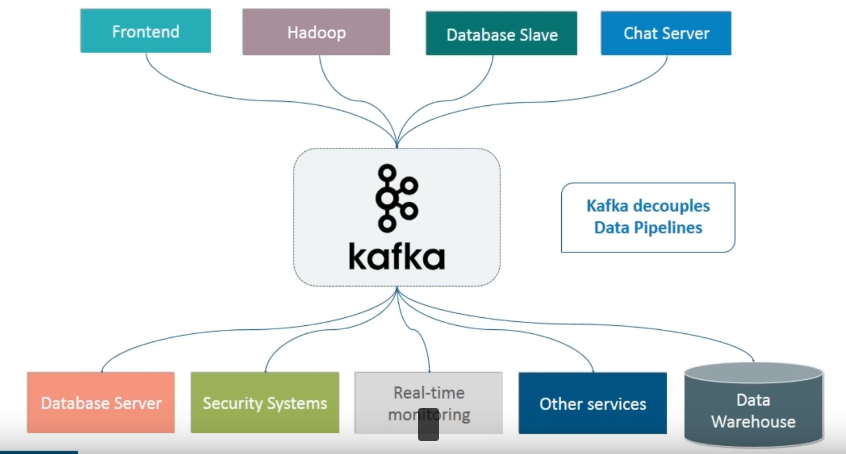
* Complex data pipeline



* Solution to the Complex data pipelines



* Kafka Decouples Data pipelines

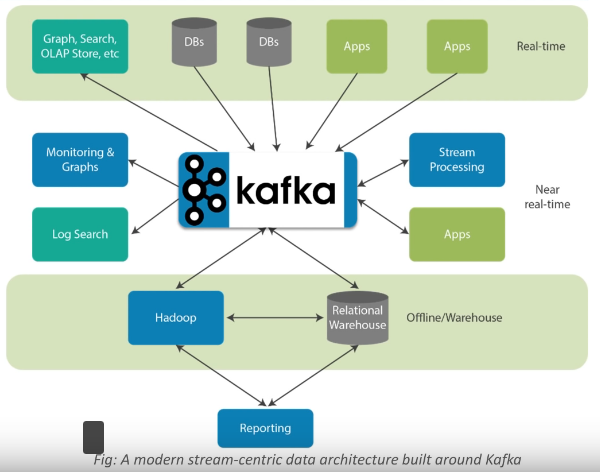


# What is Kafka:

* Apache Kafka is a **distributed publish-subscribe** messaging system and a robust queue that can handle a high volume of data and enables you to pass messages from one end-point to another.
* Kafka is suitable for both offline and online message consumption.
* 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**.
* It integrates very well with Apache Storm and Spark for real-time streaming data analysis.

### **Benefits**

* **Reliability** − Kafka is distributed, partitioned, replicated and fault tolerance.
* **Scalability** − Kafka messaging system scales easily without down time..
* **Durability** − Kafka uses Distributed commit log which means messages persists on disk as fast as possible, hence it is durable..
* **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 guarantees **zero downtime** and **zero data loss**.

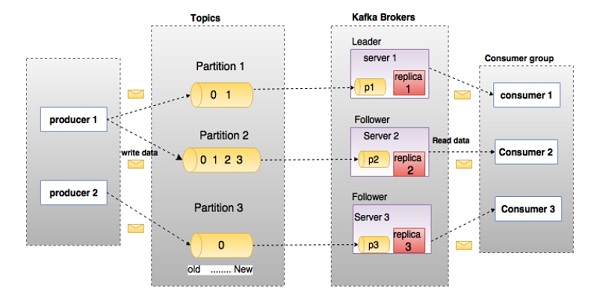


# Kafka Feature:

* **Metrics** − Kafka is often used for operational monitoring data. This involves aggregating statistics from distributed applications to produce centralized feeds of operational data.
* **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** − Popular frameworks such as Storm and Spark 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 processing.
* Kafka is a unified platform for handling all the **real-time data feeds**.
* Kafka supports **low latency message delivery** and gives guarantee for fault tolerance in the presence of machine failures.
* It has the ability to handle a large number of diverse consumers.
* Kafka is very fast, performs **2 million writes/sec**.
* Kafka persists all data to the disk, which essentially means that all the writes go to the page cache of the OS (RAM). This makes it very efficient to transfer data from page cache to a network socket.

# Kafka Components:

* The main components are
  + **Topics**
  + **Brokers**
  + **Producers and**
  + **Consumers.**

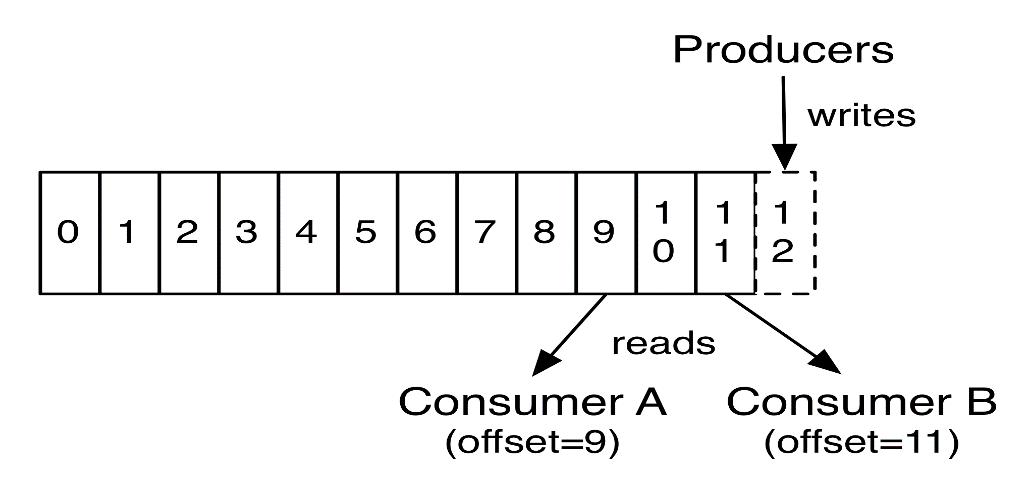


* In the above diagram, a topic is configured into three partitions. Partition 1 has two offset factors 0 and 1. Partition 2 has four offset factors 0, 1, 2, and 3. Partition 3 has one offset factor 0. The id of the replica is same as the id of the server that hosts it.
* Assume, if the replication factor of the topic is set to 3, then Kafka will create 3 identical replicas of each partition and place them in the cluster to make available for all its operations. To balance a load in cluster, each broker stores one or more of those partitions. Multiple producers and consumers can publish and retrieve messages at the same time.

## **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. Each such partition contains messages in an immutable ordered sequence. A partition is implemented as a set of segment files of equal sizes.





## **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 nothing but **backups of a 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.
* Assume if there are N partitions in a topic and more than N brokers (n + m), the first N broker will have one partition and the next M broker will not have any partition for that particular topic.
* Assume if there are N partitions in a topic and less than N brokers (n-m), each broker will have one or more partition sharing among them. This scenario is not recommended due to unequal load distri-bution among the broker.

## **Kafka Cluster**

* Kafka cluster typically consists of **multiple brokers** to maintain **load balance**.
* A Kafka cluster can be expanded without downtime. These clusters are used to **manage the persistence and replication of message data**.
* Kafka brokers are stateless, so 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 leader election can be done by ZooKeeper.

## **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, the broker simply appends the message to the last segment file. Actually, the message will be appended to a partition.
* **Producer can also send messages to a partition of their choice**.
* Kafka producer doesn’t wait for acknowledgements from the broker and sends messages as fast as the broker can handle.

## **Consumers**

* Consumers **read data from brokers**. Consumers **subscribes** to one or more topics and consume published messages **by pulling data from the brokers**.
* Consumer has to maintain how many messages have been consumed by using partition offset.
* If the consumer acknowledges a particular message offset, it implies that the consumer has consumed all prior messages.
* The consumer issues an asynchronous pull request to the broker to have a buffer of bytes ready to consume.
* The consumers can rewind or skip to any point in a partition simply by supplying an offset value.
* Consumer offset value is notified by ZooKeeper.

## **Leader**

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

## **Follower**

* **Node which follows leader instructions** are called as follower.
* If the leader fails, one of the follower will automatically become the new leader.
* A follower acts as normal consumer, pulls messages and up-dates its own data store.

## **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.
* As per the notification received by the Zookeeper regarding presence or failure of the broker then pro-ducer and consumer takes decision and starts coordinating their task with some other broker.

# Kafka Architecture:



# Kafka - WorkFlow

## **6.1 Workflow of Pub-Sub Messaging**

1. Producers send message to a topic at regular intervals.
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 in a regular interval (like 100 Ms) 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 an acknowledgement to the Kafka broker.
9. Once Kafka receives an acknowledgement, it changes the offset to the new value and updates it in the Zookeeper. Since offsets are maintained in the Zookeeper, the consumer can read next message correctly even during server outrages.
10. This above flow will repeat until the consumer stops the request.
11. Consumer has the option to rewind/skip to the desired offset of a topic at any time and read all the subsequent messages.

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

* In a queue messaging system instead of a single consumer, a group of consumers having the same Group ID will subscribe to a topic.
* Example, consumers subscribing to a topic with same Group ID are considered as a single group and the messages are shared among them. Let us check the actual workflow of this system.

1. Producers send message to a topic in a regular interval.
2. Kafka stores all messages in the partitions configured for that particular topic similar to the earlier scenario.
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.



1. Once the new consumer arrives, Kafka switches its operation to share mode and shares the data between the two consumers. This sharing will go on until the number of con-sumers reach the number of partition configured for that particular topic.
2. Once 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. This scenario arises because each consumer in Kafka will be assigned a minimum of one partition and once all the partitions are assigned to the existing consumers, the new consumers will have to wait.
3. This feature is also called as Consumer Group. In the same way, Kafka will provide the best of both the systems in a very simple and efficient manner.

# ZooKeeper

* Zookeeper is a service that Kafka uses to manage its cluster state and configurations.
* 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 (queue readers) and so on.
* 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.
* **Kafka will restore the state, once the Zookeeper restarts**. This gives zero downtime for Kafka.
* The leader election between the Kafka broker is also done by using Zookeeper in the event of leader failure.

# Installing Apache Kafka

## **Step-1: Prerequisites**

* Our server and a non-root user with sudo privileges.
* At least **4GB of RAM** on the server. Installations without this amount of RAM may cause the Kafka service to fail, with the [Java virtual machine (JVM)](https://en.wikipedia.org/wiki/Java_virtual_machine) throwing an “**Out Of Memory**” exception during startup.
* [OpenJDK](http://openjdk.java.net/) 8 installed on your server.

# sudo yum install java-1.8.0-openjdk

# sudo yum install java-1.8.0-openjdk-devel

## **Step-2: Creating a kafka user**

* Logged in as your non-root sudo user, create a user called **kafka** with the useradd command:

# sudo useradd kafka -m

* The -m flag ensures that a home directory will be created for the user. This home directory, /home/kafka, will act as our workspace directory for executing commands in the sections below.
* Set the password using passwd:

# sudo passwd kafka

* Add the **kafka** user to the wheel group with the adduser command

# sudo usermod -aG wheel kafka

* Your **kafka** user is now ready. Log into this account using su:

# su -l kafka

## **Step 3 - ZooKeeper Framework Installation**

* 1. Download the ZooKeeper:

$ cd /opt/

$ wget http://mirrors.estointernet.in/apache/zookeeper/zookeeper-3.5.6/apache-zookeeper-3.5.6-bin.tar.gz

* 1. Extract tar file

$ tar -zxf zookeeper-3.4.6.tar.gz

$ cd zookeeper-3.4.6

$ mkdir data

* 1. Create Configuration file

$ vi conf/zoo.cfg

tickTime=2000

dataDir=/path/to/zookeeper/data

clientPort=2181

initLimit=5

syncLimit=2

* 1. Start Zookeeper Server

$ bin/zkServer.sh start

(or)

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

**Output**: $ JMX enabled by default

$ Using config: /Users/../zookeeper-3.4.6/bin/../conf/zoo.cfg

$ Starting zookeeper ... STARTED

* 1. Start CLI

$ bin/zkCli.sh

**Output**: Connecting to localhost:2181

................

................

................

Welcome to ZooKeeper!

................

................

WATCHER::

WatchedEvent state:SyncConnected type: None path:null

[zk: localhost:2181(CONNECTED) 0]

* 1. Stop Zookeeper Server

$ bin/zkServer.sh stop

## **Step-4: Downloading and Extracting the Kafka Binaries**

* create a directory in /home/kafka called Downloads to store your downloads:

# mkdir ~/Downloads

# curl "https://www.apache.org/dist/kafka/2.1.1/kafka\_2.11-2.1.1.tgz" -o ~/Downloads/kafka.tgz

(or)

# curl “https://www.apache.org/dyn/closer.cgi?path=/kafka/2.3.0/kafka\_2.12-2.3.0.tgz” –o ~/Downloads/kafka.tgz

# mkdir ~/kafka && cd ~/kafka

# tar -xvzf ~/Downloads/kafka.tgz --strip 1

* We specify the --strip 1 flag to ensure that the archive’s contents are extracted in ~/kafka/ itself and not in another directory (such as ~/kafka/kafka\_2.11-2.1.1/) inside of it.

## **Step 5 — Configuring the Kafka Server**

* Kafka’s configuration options are specified in server.properties

# vi ~/kafka/config/server.properties

* Let’s add a setting that will allow us to delete Kafka topics.

delete.topic.enable = true

## **Step 6 — Creating Systemd Unit Files and Starting the Kafka Server**

* 1. We will create [systemd unit files](https://www.digitalocean.com/community/tutorials/understanding-systemd-units-and-unit-files) for the Kafka service. This will help us perform common service actions such as starting, stopping, and restarting Kafka in a manner consistent with other Linux services.
  2. Create the unit file for zookeeper:

$ sudo vi /etc/systemd/system/zookeeper.service

[Unit]

Requires=network.target remote-fs.target

After=network.target remote-fs.target

[Service]

Type=simple

User=kafka

ExecStart=/home/kafka/kafka/bin/zookeeper-server-start.sh /home/kafka/kafka/config/zookeeper.properties

ExecStop=/home/kafka/kafka/bin/zookeeper-server-stop.sh

Restart=on-abnormal

[Install]

WantedBy=multi-user.target

**NOTE**:

The [Unit] section specifies that Zookeeper requires networking and the filesystem to be ready before it can start.

The [Service] section specifies that systemd should use the zookeeper-server-start.sh and zookeeper-server-stop.sh shell files for starting and stopping the service. It also specifies that Zookeeper should be restarted automatically if it exits abnormally.

* 1. Start the Server

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

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

* 1. create the systemd service file for kafka:

$ sudo vi /etc/systemd/system/kafka.service

[Unit]

Requires=zookeeper.service

After=zookeeper.service

[Service]

Type=simple

User=kafka

ExecStart=/bin/sh -c '/home/kafka/kafka/bin/kafka-server-start.sh /home/kafka/kafka/config/server.properties > /home/kafka/kafka/kafka.log 2>&1'

ExecStop=/home/kafka/kafka/bin/kafka-server-stop.sh

Restart=on-abnormal

[Install]

WantedBy=multi-user.target

**NOTE**:

The [Unit] section specifies that this unit file depends on zookeeper.service. This will ensure that zookeeper gets started automatically when the kafa service starts.

The [Service] section specifies that systemd should use the kafka-server-start.sh and kafka-server-stop.sh shell files for starting and stopping the service. It also specifies that Kafka should be restarted automatically if it exits abnormally.

* 1. Now that the units have been defined, start Kafka and While we have started the kafka service, if we were to reboot our server with the following command:

$ sudo systemctl start kafka

$ sudo systemctl enable kafka

* 1. To ensure that the server has started successfully, check the journal logs for the kafka unit:

$ journalctl -u kafka

o/p: Jul 17 18:38:59 kafka-centos systemd[1]: Started kafka.service.

* You now have a Kafka server listening on port 9092.

## **Step 7— Testing the Installation**

* Let’s publish and consume a “Hello World” message to make sure the Kafka server is behaving correctly.
* Publishing messages in Kafka requires:
  + A producer, which enables the publication of records and data to topics.
  + A consumer, which reads messages and data from topics.

1. **Create a topic** named TutorialTopic by typing:

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

Output: Created topic "TutorialTopic".

1. **Create a producer** from the command line using the kafka-console-producer.sh script. It expects the Kafka server’s hostname, port, and a topic name as arguments.

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

1. Create a Kafka consumer using the kafka-console-consumer.sh script. It expects the ZooKeeper server’s hostname and port, along with a topic name as arguments.

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

1. If there are no configuration issues, you should see Hello, World in your terminal
2. The script will continue to run, waiting for more messages to be published to the topic. Feel free to open a new terminal and start a producer to publish a few more messages. You should be able to see them all in the consumer’s output.
3. When you are done testing, press CTRL+C to stop the consumer script.

## **Step 8 — Setting Up a Multi-Node Cluster (Optional)**

* You should repeat Step 1, Step 4, and Step 5 on each of the new machines. Additionally, you should make the following changes in the server.properties file for each:
* First we make a config file for each of the brokers

cp config/server.properties config/server-1.properties

cp config/server.properties config/server-2.properties

* The value of the broker.id property should be changed such that it is unique throughout the cluster. This property uniquely identifies each server in the cluster and can have any string as its value. For example, "server1", "server2", etc.

config/server-1.properties:

broker.id=1

listeners=PLAINTEXT://:9093

log.dirs=/tmp/kafka-logs-1

config/server-2.properties:

broker.id=2

listeners=PLAINTEXT://:9094

log.dirs=/tmp/kafka-logs-2

* The value of the zookeeper.connect property should be changed such that all nodes point to the same ZooKeeper instance. This property specifies the Zookeeper instance’s address and follows the <HOSTNAME/IP\_ADDRESS>:<PORT> format. For example, "203.0.113.0:2181","203.0.113.1:2181" etc.
* need to start the two new nodes

bin/kafka-server-start.sh config/server-1.properties &

bin/kafka-server-start.sh config/server-2.properties &

# Reference:

* What Is Kafka? Everything You Need to Know: <https://dzone.com/articles/what-is-kafka>.
* <https://www.tutorialspoint.com/apache_kafka/apache_kafka_introduction.htm>
* <https://kafka.apache.org/>
* Kinesis Vs Kafka: <http://cloudurable.com/blog/kinesis-vs-kafka/index.html>
* What is Apache Kafka: <http://cloudurable.com/blog/what-is-kafka/index.html>
* Introduction to Apache Kafka: <https://dzone.com/articles/introduction-to-apache-kafka-1>
* Apache Kafka Turotial: <https://www.youtube.com/watch?v=hyJZP-rgooc>
* Understanding of Kafka: <https://www.youtube.com/watch?v=k-7lz6Ex354>
* GitHub: <https://github.com/apache/Kafka>
* How To Install Apache Kafka on CentOS 7: <https://www.digitalocean.com/community/tutorials/how-to-install-apache-kafka-on-centos-7>