Data Distribution and Analyst

Hadoop

Introduction

Hadoop is a collection of open-source software. It uses a network of many computers to analyst and compute huge data problems. It contains three core components: Hadoop Distributed File System(HDFS), Hadoop YARN, and HADOOP MapReduce.

Hadoop History

(Hadoop Wiki)

"According to its co-founders, Doug Cutting and Mike Cafarella, the genesis of Hadoop was the Google File System paper that was published in October 2003. This paper spawned another one from Google – "MapReduce: Simplified Data Processing on Large Clusters". Development started on the Apache Nutch project, but was moved to the new Hadoop subproject in January 2006. Doug Cutting, who was working at Yahoo! at the time, named it after his son's toy elephant. The initial code that was factored out of Nutch consisted of about 5,000 lines of code for HDFS and about 6,000 lines of code for MapReduce.
The first committer to add to the Hadoop project was Owen O'Malley (in March 2006); Hadoop 0.1.0 was released in April 2006. It continues to evolve through contributions that are being made to the project."

Hadoop Advantages

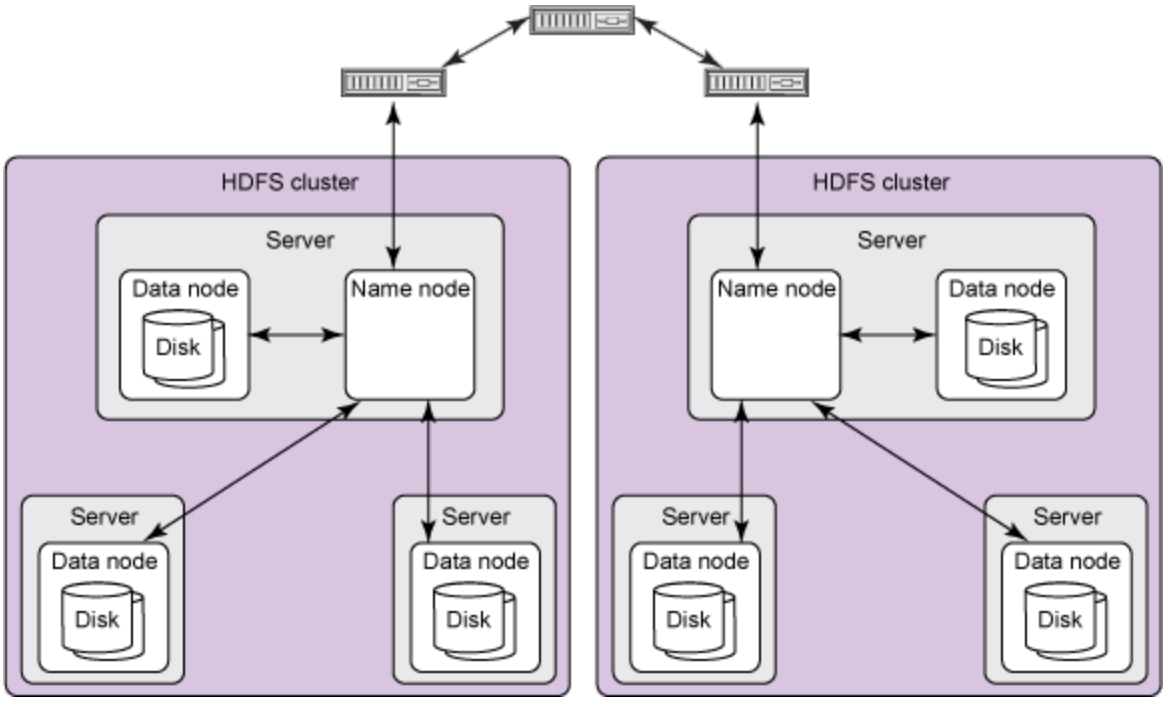
(Tutorialspoint of hadoop)

- Hadoop framework allows the user to quickly write and test distributed systems. It is efficient, and it automatic distributes the data and work across the machines and in turn, utilizes the underlying parallelism of the CPU cores.
- Hadoop does not rely on hardware to provide fault-tolerance and high availability (FTHA), rather Hadoop library itself has been designed to detect and handle failures at the application layer.
- Servers can be added or removed from the cluster dynamically and Hadoop continues to operate without interruption.
- Another big advantage of Hadoop is that apart from being open source, it is compatible on all the platforms since it is Java based.

Hadoop Distributed File System(HDFS)

It is a distributed file-system. It is a master/slave architecture.

HDFS Overviews



HDFS Important Terms

(Hadoop Documentation)

Namenode: An HDFS cluster consists of a single NameNode. NameNode executes file system namespace operations like opening, closing, and renaming files and directories. It also determines the mapping of blocks to DataNodes. The DataNodes are responsible for serving read and write requests from the file system's clients.

Datanode: DataNode also perform block creating, deletion, and replication upon instruction from the NameNode.

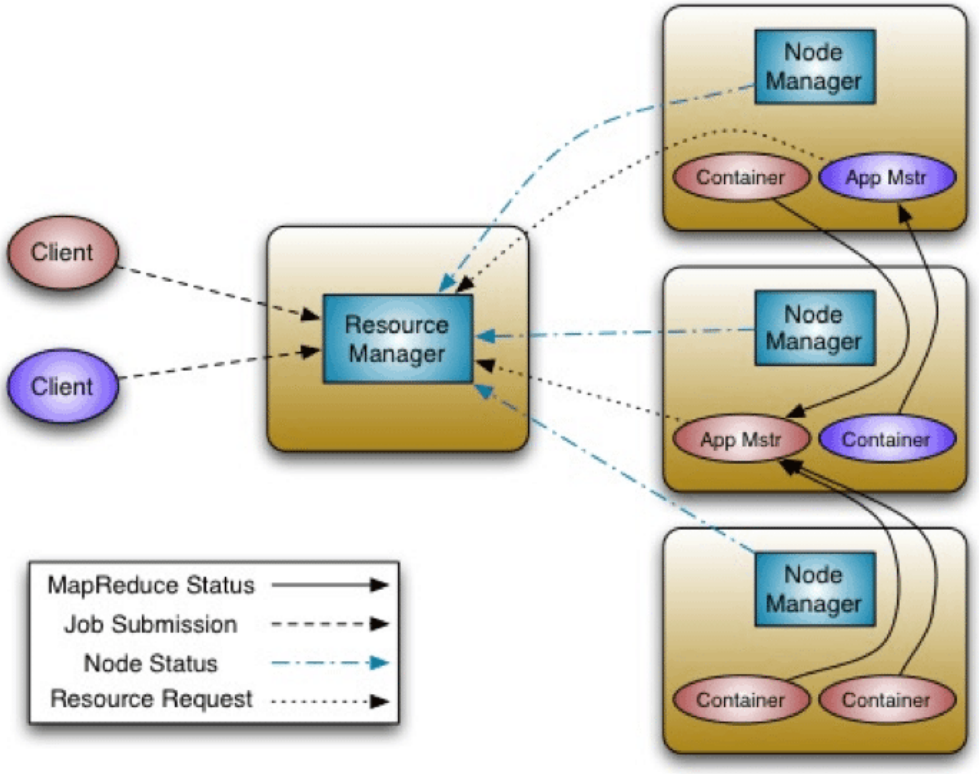
Cluster Rebalancing: The HDFS architecture is compatible with data rebalancing schemes. A scheme might automatically move data from one DataNode to another if the free space on a DataNode falls below a certain threshold. In the event of a sudden high demand for a particular file, a scheme might dynamically create additional replicas and rebalance other data in the cluster. These types of data rebalancing schemes are not yet implemented.

Data Replication: HDFS is designed to reliably store very large files across machines in a large cluster. It stores each file as a sequence of blocks. The blocks of a file are replicated for fault tolerance. The block size and replication factor are configurable per file.
All blocks in a file except the last block are the same size, while users can start a new block without filling out the last block to the configured block size after the support for variable length block was added to append and hsync.
An application can specify the number of replicas of a file. The replication factor can be specified at file creation time and can be changed later. Files in HDFS are write-once (except for appends and truncates) and have strictly one writer at any time.
The NameNode makes all decisions regarding replication of blocks. It periodically receives a Heartbeat and a Blockreport from each of the DataNodes in the cluster. Receipt of a Heartbeat implies that the DataNode is functioning properly. A Blockreport contains a list of all blocks on a DataNode.

Hadoop YARN

It's a platform responsible for managing computing resources in clusters and using them for scheduling user's applications.

Hadoop YARN Overviews



(https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/yarn_architecture.gif)

Hadoop YARN Important Terms

(Hadoop Documentation)
ResourceManager: It contains Scheduler and ApplicationsManager.

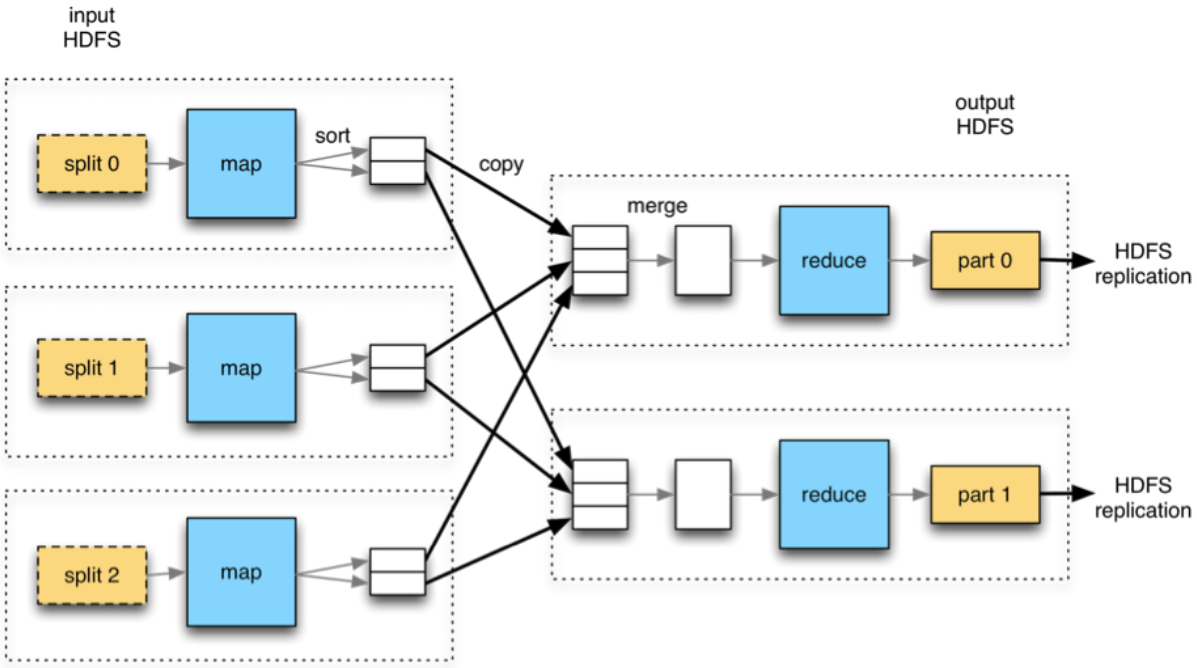
Scheduler: The Scheduler has a pluggable policy which is responsible for partitioning the cluster resources among the various queues, applications etc. The current schedulers such as the CapacityScheduler and the FairScheduler would be some examples of plug-ins.

ApplicationManager: The ApplicationsManager is responsible for accepting job-submissions, negotiating the first container for executing the application specific ApplicationMaster and provides the service for restarting the ApplicationMaster container on failure. The per-application ApplicationMaster has the responsibility of negotiating appropriate resource containers from the Scheduler, tracking their status and monitoring for progress.

Hadoop MapReduce

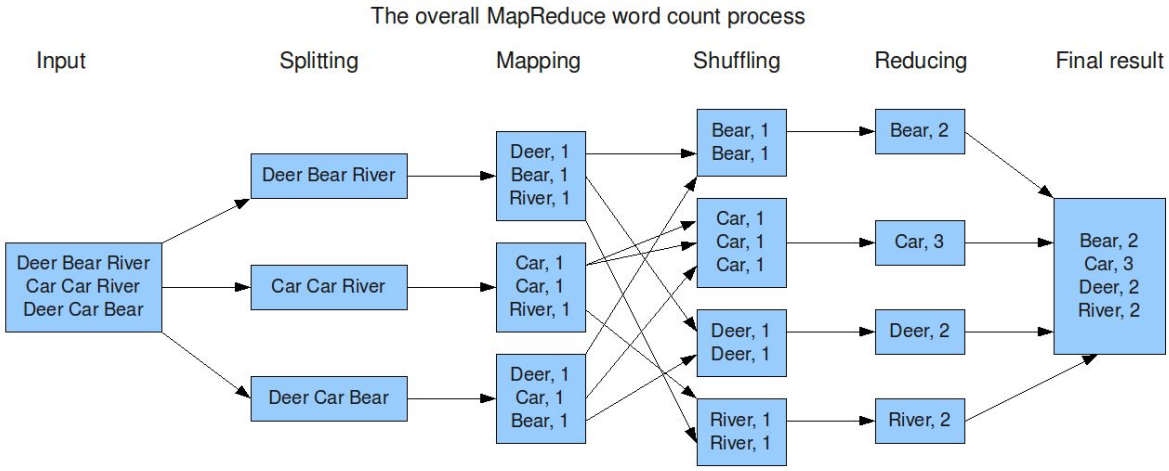
It’s a implementation for huge data processing based on map-reduce model.

Hadoop MapReduce Overviews



(https://www.supinfo.com/articles/resources/207908/2807/3.png)

MapReduce Example - WordCount



(https://www.google.com/url?sa=i&source=images&cd=&cad=rja&uact=8&ved=2ahUKEwj0sp24waLfAhVQ_oMKHRPbDsoQjRx6BAgBEAU&url=http%3A%2F%2Fwww.todaysoftmag.com%2Farticle%2F1358%2Fhadoop-mapreduce-deep-diving-and-tuning&psig=AOvVaw0nb70ThZrcX_fcGERNWNdK&ust=1544986417473621)

Hadoop Installation and configuration

Software requirements(MacOS)

JDK1.8.0_181、ssh、pdsh, hadoop 3.1.1

JDK1.8

// Download
brew tap caskroom/versions brew cask install java8
// Set jdk1.8 as default version.
export JAVA_HOME=`/usr/libexec/java_home -v 1.8`

SSH

// Install ssh.

yum install ssh

PDSH

// Install pdsh
yum install pdsh

Hadoop 3.1.1

// Find a home file for hadoop.
cd /usr/local
// Download
wget http://mirrors.shu.edu.cn/apache/hadoop/common/hadoop-3.1.1/hadoop-3.1.1.tar.gz

Hadoop Environment Setup

// Decompression
tar zxvf hadoop-3.1.1/hadoop-3.1.1.tar.gz
// Go to hadoop directory.
cd hadoop-3.1.1

// Set JAVA_HOME path.
export JAVA_HOME=/usr/local/jdk1.8.0_181

// Test
bin/hadoop

Setting up a single node cluster

etc/hadoop/core-site.xml

```
<configuration>
        <property>
                <name>fs.defaultFS</name>
                <value>hdfs://localhost:9000</value>
```

```
        </property>
</configuration>
```

etc/hadoop/hdfs-site.xml

```
<configuration>
        <property>
                <name>dfs.replication</name>
                <value>1</value>
        </property>
</configuration>
```

Add SSH

1. ssh localhost
2. ssh-keygen -t rsa -P '' -f ~/.ssh/id_rsa
   cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
   chmod 0600 ~/.ssh/authorized_keys

Formatting

```
bin/hdfs namenode -format
```

Start executing

```
sbin/start-dfs.sh
```

Stop

```
sbin/stop-yarn.sh
```

Extract Tips

jps

It will print namenode process.

Overview and Summary of Hadoop jobs

http://localhost:9870

Setting up a single node YARN

etc/hadoop/mapred-site.xml

```
<!-- Put site-specific property overrides in this file. →
<configuration>
        <property>
                <name>mapreduce.framework.name</name>
                <value>yarn</value>
        </property>
        <property>
        <name>mapreduce.application.classpath</name>
<value>$HADOOP_MAPRED_HOME/share/hadoop/mapreduce/*:$HADOOP_MAPRED_HO
ME/share/hadoop/mapreduce/lib/*</value>
         </property>
</configuration>
~
```

etc/hadoop/yarn-site.xml

```
<configuration>
   <property>
      <name>yarn.nodemanager.aux-services</name>
      <value>mapreduce_shuffle</value>
   </property>
   <property>
```

<name>yarn.nodemanager.env-whitelist</name>

<value>JAVA_HOME,HADOOP_COMMON_HOME,HADOOP_HDFS_HOME,HADOOP_CONF_DIR,CLASSPATH_PREPEND_DISTCACHE,HADOOP_YARN_HOME,HADOOP_MAPRED_HOME</value>
</property>
</configuration>

Stop

sbin/stop-yarn.sh

Setting up Mapreduce

Create jar file

./hadoop com.sun.tools.javac.Main /Users/wanqianhuang/IdeaProjects/LinearRegression/src/main/java/lg.java

Specify main class for that jar file

jar cf lg25.jar lg*.class

Execute Jar files

./hadoop jar /Users/wanqianhuang/IdeaProjects/LinearRegression/src/main/java/lg25.jar lg /user/huang.wanqian/input/test0.txt /user/huang.wanqian/lg25_all_out6 /user/huang.wanqian/input/test0.txt

Change Java heap size

mapred-site.xml

```
<configuration>
 <property>
   <name>mapreduce.framework.name</name>
   <value>yarn</value>
 </property>
 <property>
   <name>mapreduce.map.memory.mb </name>
   <value>2048</value>
   <description> Memory </description>
 </property>
 <property>
   <name>mapreduce.application.classpath</name>

<value>$HADOOP_MAPRED_HOME/share/hadoop/mapreduce/*:$HADOOP_MAPRED_HO
ME/share/hadoop/mapreduce/lib/*</value>
 </property>
</configuration>
```

MapReduce Linear Regression Performance

1. Linear Regression on 1000000 numbers. Compare the time costing between MapReduce and normal java program:

   MapReduce:

   Time spent by all map tasks is 5790ms;
   Time spent by all reduce tasks is 2463 ms;
   In total, it spent 8.253 s

   Total time spent by all map tasks (ms)=5790
   Total time spent by all reduce tasks (ms)=2463

   Java:

   I wait about 5 minutes, and it's still running.

2. Linear Regression on 100000 numbers. Each number is added a random number between 50 and -50.

   MapReduce:

   Total time spent by all map tasks (ms)=7148

Total time spent by all reduce tasks (ms)=2475

In total, it spent 9.623 second.

Java:

It spend around 6.20 second.

3. Linear Regression on 1000000 numbers. Each number is added a random number between 50 and -50.

MapReduce:

Total time spent by all map tasks (ms)=7306

Total time spent by all reduce tasks (ms)=2615

In total, it spent 9.921 second.

Java:

I wait about 5 minutes, and it's still running.

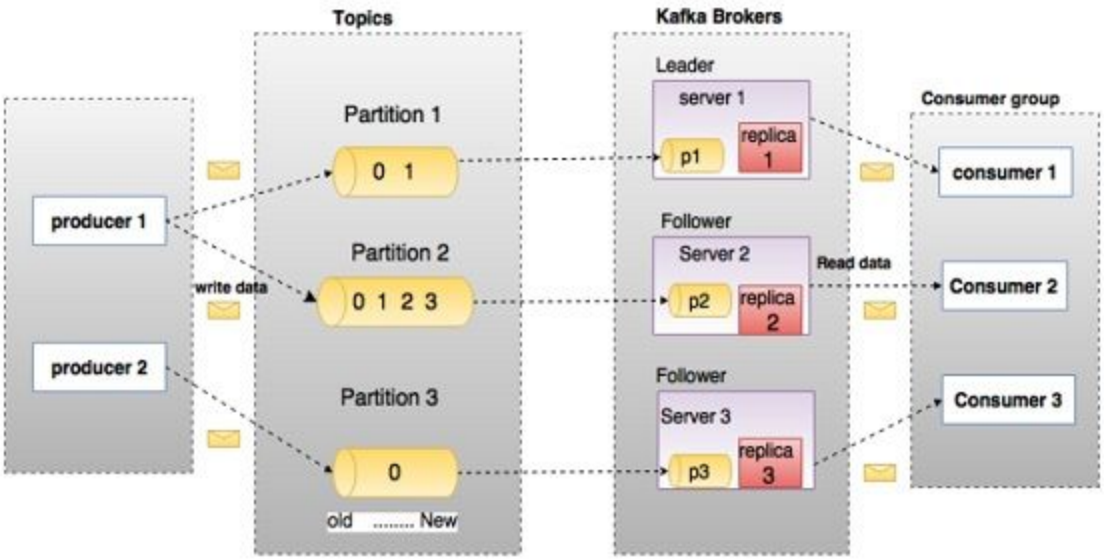
Kafka

Apache Kafka is an open-source stream-processing software platform developed by the Apache Software Foundation. It's written in Scala and Java.

Kafka History

"Apache Kafka was originally developed by LinkedIn, and was subsequently open sourced in early 2011. Graduation from the Apache Incubator occurred on 23 October 2012. In 2014, Jun Rao, Jay Kreps, and Neha Narkhede, who had worked on Kafka at LinkedIn, created a new company named Confluent with a focus on Kafka. According to a Quora post from 2014, Kreps chose to name the software after the author Franz Kafka because it is "a system optimized for writing", and he liked Kafka's work."

Kafka Overview



(https://www.tutorialspoint.com/apache_kafka/apache_kafka_fundamentals.htm)

Kafka Important Terms

([Tutorialspoint of Kafka])

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 minimum 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 published 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 distribution among the broker.

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 publishes 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.

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.

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 updates 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 producer and consumer takes decision and starts coordinating their task with some other broker.

Kafka Installation and commands

Software requirements(MacOS)

JDK1.8.0_181、kafka_2.11-2.1.0.tgz

JDK1.8

// Download
brew tap caskroom/versions brew cask install java8
// Set jdk1.8 as default version.
export JAVA_HOME=`/usr/libexec/java_home -v 1.8`

Kafka_2.11-2.1.0

// Download
> tar -xzf kafka_2.11-2.1.0.tgz

Simple Commands

Start ZooKeeper Server

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

Start the Kafka Server

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

Create a topic

> bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic **topic1**

Print topic names

> bin/kafka-topics.sh --list --zookeeper localhost:2181
topic1

Send messages

> bin/kafka-console-producer.sh --broker-list localhost:9092 --topic **topic1**
Hello World!
How are you!

Start a consumer

> bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic **topic1** --from-beginning
Hello World!
How are you!

Kafka and HDFS connector

Introduction

The HDFS connector allows you to export data from Kafka topics to HDFS files in a variety of formats and integrates with Hive to make data immediately available for querying with HiveQL.

The connector periodically polls data from Kafka and writes them to HDFS. The data from each Kafka topic is partitioned by the provided partitioner and divided into chunks.

The following link contains more information you may want to look.
https://www.confluent.io/connector/kafka-connect-hdfs/

Shares

This is a good hadoop tutorial for a beginner. It contains a lot graphics and real-life examples to explain basic idea of hdfs, yarn and mapreduce.
https://www.youtube.com/watch?v=mafw2-CVYnA&list=PL9ooVrP1hQOFrYxqxb0NJCdCABPZNo0pD&index=3

Hadoop documentation:
https://hadoop.apache.org/docs/r3.1.0/

Tutorialspoint of Hadoop:
https://www.tutorialspoint.com/hadoop/

Kafka documentation:
https://kafka.apache.org/documentation/

Tutorialspoint of Hadoop:
https://www.tutorialspoint.com/apache_kafka/apache_kafka_introduction.htm