UNIT 2 BIG DATA TOOLS I

Big Data Applications using Pig and Hive – Fundamentals of HBase and ZooKeeper – IBM Infosphere Big Insights – Introduction to FLUME – KAFKA

Apache PIG:

Pig is a high-level programming language useful for analyzing large data sets. PIG was a result of the development effort at Yahoo!

In a MapReduce framework, programs need to be translated into a series of Map and Reduce stages. However, this is not a programming model which data analysts are familiar with. So, in order to bridge this gap, an abstraction called Pig was built on top of Hadoop.

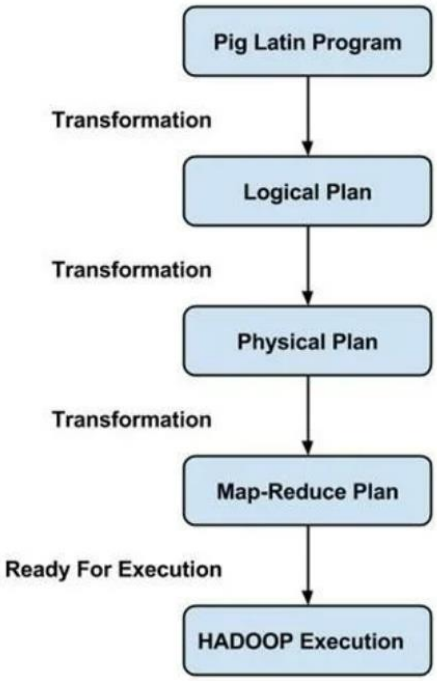
Apache Pig lets people focus more on analyzing bulk data sets and spend less time writing Map-Reduce programs. Like Pigs, who eat anything, the Apache Pig programming language is designed to work on any data.

Pig Architecture:

The Architecture of Pig consists of two components:

1. Pig Latin, which is a language
2. A runtime environment, for running PigLatin programs.

A Pig Latin program consists of a series of operations or transformations which are applied to the input data to produce output. These operations describe a data flow which is translated into an executable representation, by Hadoop Pig execution environment. Underneath, results of these transformations are series of MapReduce jobs which a programmer is unaware of. So, in a way, Pig in Hadoop allows the programmer to focus on data rather than the nature of execution. PigLatin is a relatively stiffened language which uses familiar keywords from data processing e.g., Join, Group and Filter.



PIG Architecture

Execution modes:

Pig in Hadoop has two execution modes:

1. Local mode: In this mode, the Hadoop Pig language runs in a single JVM and makes use of a local file system. This mode is suitable only for the analysis of small datasets using Pig in Hadoop
2. Map Reduce mode: In this mode, queries written in Pig Latin are translated into MapReduce jobs and are run on a Hadoop cluster (cluster may be pseudo or fully distributed). MapReduce mode with the fully distributed cluster is useful for running Pig on large datasets.

Pig Represents Big Data as data flows. Pig is a high-level platform or tool which is used to process large datasets. It provides a high-level of abstraction for processing over the educe. It provides a high-level scripting language, known as *Pig Latin* which is used to develop the data analysis codes. First, to process the data which is stored in the HDFS, the programmers will write the scripts using the Pig Latin Language. Internally *Pig Engine*(a component of Apache Pig) converted all these scripts into a specific map and reduce task. But these are not visible to the programmers in order to provide a high-level of abstraction. Pig Latin and Pig Engine are the two main components of the Apache Pig tool. The result of Pig is always stored in the HDFS.

Need of Pig: One limitation of MapReduce is that the development cycle is very long. Writing the reducer and mapper, compiling packaging the code, submitting the job and retrieving the output is a time-consuming task. Apache Pig reduces the time of development using the multi-query approach. Also, Pig is beneficial than Java. 200 lines of Java code can

be written in only 10 lines using the Pig Latin language. Programmers who have SQL knowledge needed less effort to learn Pig Latin.

- It uses query approach which results in reducing the length of the code.
- Pig Latin is SQL like language.
- It provides many builtIn operators.
- It provides nested data types (tuples, bags, map).

Evolution of Pig: Earlier in 2006, Apache Pig was developed by Yahoo's researchers. At that time, the main idea to develop Pig was to execute the MapReduce jobs on extremely large datasets. In the year 2007, it moved to Apache Software Foundation(ASF) which makes it an open source project. The first version(*0.1*) of Pig came in the year 2008. The latest version of Apache Pig is *0.18* which came in the year 2017.

Features of Apache Pig:

- For performing several operations Apache Pig provides rich sets of operators like the filters, join, sort, etc.
- Easy to learn, read and write. Especially for SQL-programmer, Apache Pig is a boon.
- Apache Pig is extensible so that you can make your own user-defined functions and process.
- Join operation is easy in Apache Pig.
- Fewer lines of code.
- Apache Pig allows splits in the pipeline.
- The data structure is multivalued, nested, and richer.
- Pig can handle the analysis of both structured and unstructured data.

Difference between Pig and MapReduce

Apache Pig	MapReduce
It is a scripting language.	It is a compiled programming language.
Abstraction is at higher level.	Abstraction is at lower level.
It have less line of code as compared to MapReduce.	Lines of code is more.
Less effort is needed for Apache Pig.	More development efforts are required for MapReduce.
Code efficiency is less as compared to MapReduce.	As compared to Pig efficiency of code is higher.
Pig provides built in functions for ordering, sorting and union.	Hard to perform data operations.

Apache Pig	MapReduce
It allows nested data types like map, tuple and bag	It does not allow nested data types

Applications of Apache Pig:

- For exploring large datasets Pig Scripting is used.
- Provides the supports across large data-sets for Ad-hoc queries.
- In the prototyping of large data-sets processing algorithms.
- Required to process the time sensitive data loads.
- For collecting large amounts of datasets in form of search logs and web crawls.
- Used where the analytical insights are needed using the sampling.

Types of Data Models in Apache Pig: It consist of the 4 types of data models as follows:

- **Atom**: It is a atomic data value which is used to store as a string. The main use of this model is that it can be used as a number and as well as a string.
- **Tuple**: It is an ordered set of the fields.
- **Bag**: It is a collection of the tuples.
- **Map**: It is a set of key/value pairs.

Hive

Hive is a data warehouse infrastructure tool to process structured data in Hadoop. It resides on top of Hadoop to summarize Big Data and makes querying and analyzing easy.

Initially, Hive was developed by Facebook, later the Apache Software Foundation took it up and developed it further as an open source under the name Apache Hive. It is used by different companies. For example, Amazon uses it in Amazon Elastic MapReduce.

Hive is not

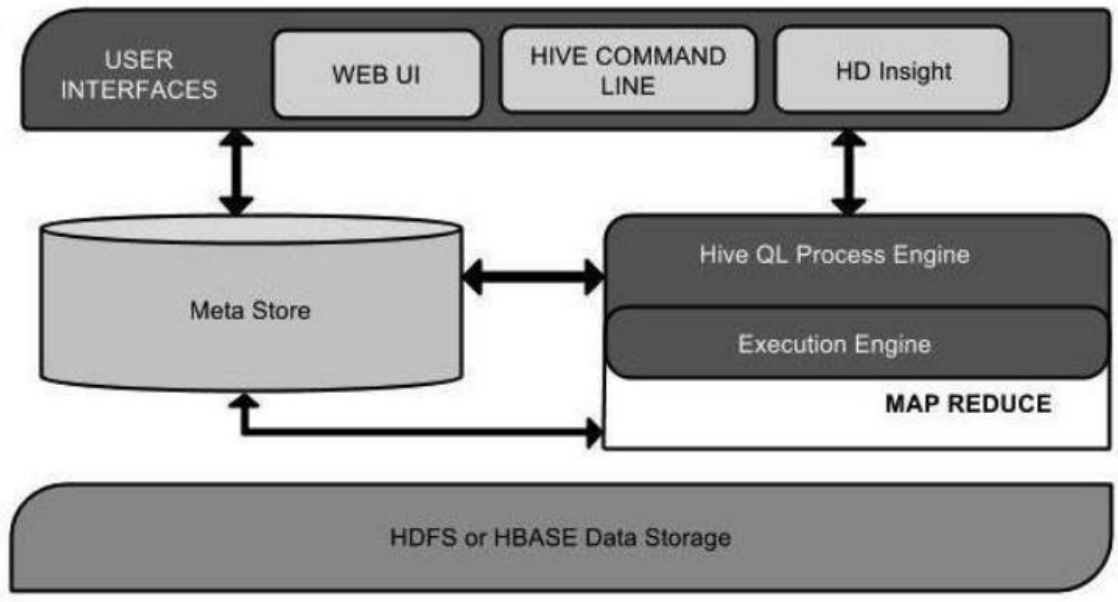
- A relational database
- A design for OnLine Transaction Processing (OLTP)
- A language for real-time queries and row-level updates

Features of Hive

- It stores schema in a database and processes data into HDFS.
- It is designed for OLAP.
- It provides SQL-type language for querying called HiveQL or HQL.
- It is familiar, fast, scalable, and extensible.

Architecture of Hive

The following component diagram depicts the architecture of Hive:



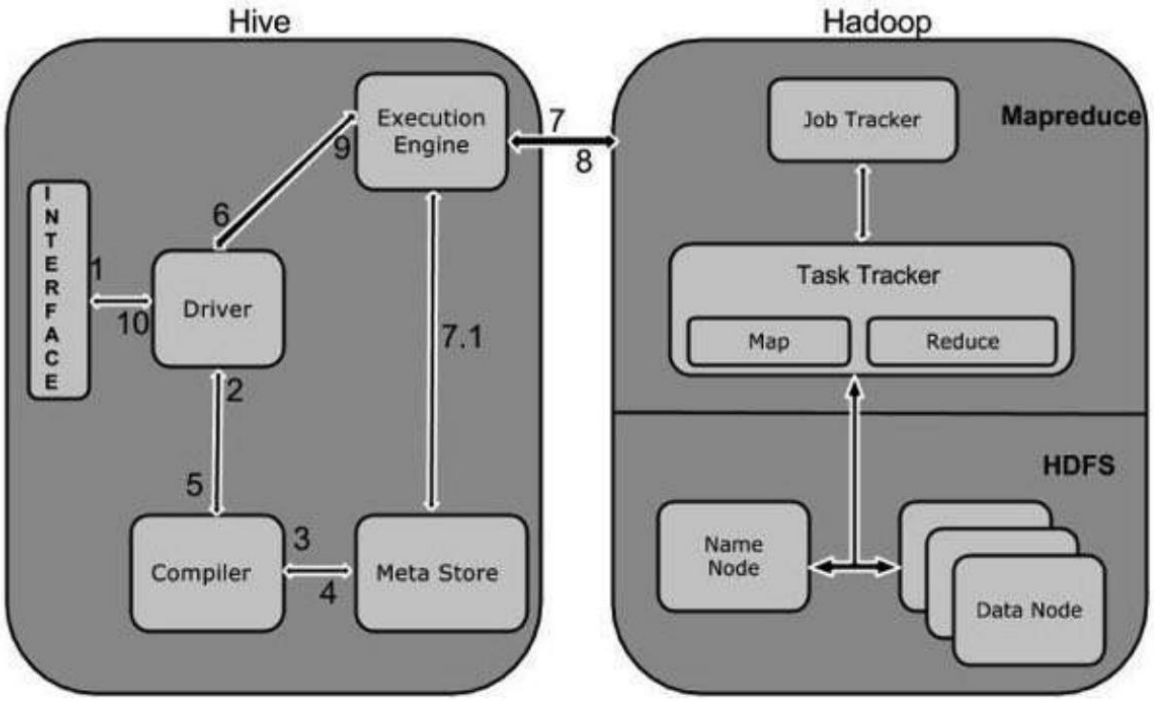
This component diagram contains different units. The following table describes each unit:

Unit Name	Operation
User Interface	Hive is a data warehouse infrastructure software that can create interaction between users and HDFS. The user interfaces that Hive supports are Hive Web UI, Hive command line, and Hive HD Insight (In Windows server).
Meta Store	Hive chooses respective database servers to store the schema or Metadata of tables, databases, columns in a table, their data types, and HDFS mapping.
HiveQL Process Engine	HiveQL is similar to SQL for querying schema info on the Metastore. It is one of the replacements for the traditional approach for the MapReduce program. Instead of writing a MapReduce program in Java, we can write a query for the MapReduce job and process it.
Execution Engine	The conjunction part of HiveQL process Engine and MapReduce is Hive Execution Engine. The execution engine processes the query and generates results as same as MapReduce results. It uses the flavor of MapReduce.

HDFS or HBase	Hadoop distributed file systems or HBASE are the data storage techniques to store data in the file system.

Working of Hive

The following diagram depicts the workflow between Hive and Hadoop.



The following table defines how Hive interacts with the Hadoop framework:

S No.	Operation
1	**Execute Query** The Hive interface such as Command Line or Web UI sends query to Driver (any database driver such as JDBC, ODBC, etc.) to execute.
2	**Get Plan** The driver takes the help of query compiler that parses the query to check the syntax and query plan or the requirement of query.
3	**Get Metadata** The compiler sends metadata request to Metastore (any database).
4	**Send Metadata** Metastore sends metadata as a response to the compiler.
5	**Send Plan** The compiler checks the requirement and resends the plan to the driver. Up

	to here, the parsing and compiling of a query is complete.
6	**Execute Plan** The driver sends the execute plan to the execution engine.
7	**Execute Job** Internally, the process of execution job is a MapReduce job. The execution engine sends the job to JobTracker, which is in Name node and it assigns this job to TaskTracker, which is in Data node. Here, the query executes MapReduce job.
7.1	**Metadata Ops** Meanwhile in execution, the execution engine can execute metadata operations with Metastore.
8	**Fetch Result** The execution engine receives the results from Data nodes.
9	**Send Results** The execution engine sends those resultant values to the driver.
10	**Send Results** The driver sends the results to Hive Interfaces.

Hive - Data Types

his chapter takes you through the different data types in Hive, which are involved in the table creation. All the data types in Hive are classified into four types, given as follows:

- Column Types
- Literals
- Null Values
- Complex Types

Column Types

Column types are used as column data types of Hive. They are as follows:

Integral Types

Integer type data can be specified using integral data types, INT. When the data range exceeds the range of INT, you need to use BIGINT and if the data range is smaller than the INT, you use SMALLINT. TINYINT is smaller than SMALLINT.

The following table depicts various INT data types:

Type	Postfix	Example
TINYINT	Y	10Y

SMALLINT	S	10S
INT	-	10
BIGINT	L	10L

String Types

String type data types can be specified using single quotes (' ') or double quotes (" "). It contains two data types: VARCHAR and CHAR. Hive follows C-type escape characters.

The following table depicts various CHAR data types:

Data Type	Length
VARCHAR	1 to 65355
CHAR	255

Timestamp

It supports traditional UNIX timestamps with optional nanosecond precision. It supports java.sql.Timestamp format "YYYY-MM-DD HH:MM: SS.fffffffff" and format "yyyy-mm-dd hh:mm:ss.fffffffff".

Dates

DATE values are described in year/month/day format in the form {{YYYY-MM-DD}}.

Decimals

The DECIMAL type in Hive is as same as Big Decimal format of Java. It is used for representing immutable arbitrary precision. The syntax and example is as follows:

```
DECIMAL(precision, scale)
decimal(10,0)
```

Union Types

Union is a collection of heterogeneous data types. You can create an instance using **create union**. The syntax and example is as follows:

```
UNIONTYPE<int, double, array<string>, struct<a:int,b:string>>

{0:1}
{1:2.0}
{2:["three","four"]}
{3:{"a":5,"b":"five"}}
{2:["six","seven"]}
{3:{"a":8,"b":"eight"}}
{0:9}
{1:10.0}
```

AD

Literals

The following literals are used in Hive:

Floating Point Types

Floating point types are nothing but numbers with decimal points. Generally, this type of data is composed of DOUBLE data types.

Decimal Type

Decimal-type data is nothing but floating point values with a higher range than DOUBLE data types. The range of decimal type is approximately -10^{-308} to 10^{308}.

Null Value

Missing values are represented by the special value NULL.

Complex Types

The Hive complex data types are as follows:

Arrays

Arrays in Hive are used the same way they are used in Java.

```
Syntax: ARRAY<data_type>
```

Maps

Maps in Hive are similar to Java Maps.

```
Syntax: MAP<primitive_type, data_type>
```

Structs

Structs in Hive is similar to using complex data with comment.

```
Syntax: STRUCT<col_name : data_type [COMMENT col_comment], ...>
```

HBase

Limitations of Hadoop

Hadoop can perform only batch processing, and data will be accessed only in a sequential manner. That means one must search the entire dataset for the simplest of jobs.

A huge dataset when processed results in another huge data set, which should also be processed sequentially. At this point, a new solution is needed to access any point of data in a single unit of time (random access).

Hadoop Random Access Databases

Applications such as HBase, Cassandra, CouchDB, Dynamo, and MongoDB are some databases that store huge amounts of data and access the data randomly.

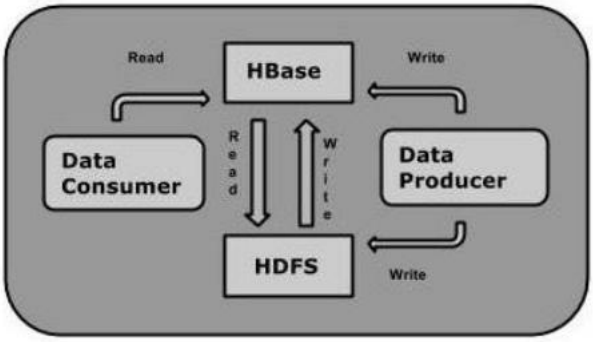
What is HBase?

HBase is a distributed column-oriented database built on top of the Hadoop file system. It is an open-source project and is horizontally scalable.

HBase is a data model that is similar to Google's big table and designed to provide quick random access to huge amounts of structured data. It leverages the fault tolerance provided by the Hadoop File System (HDFS).

It is a part of the Hadoop ecosystem that provides random real-time read/write access to data in the Hadoop File System.

One can store the data in HDFS either directly or through HBase. Data consumer reads/accesses the data in HDFS randomly using HBase. HBase sits on top of the Hadoop File System and provides read and write access.



HBase and HDFS

HDFS	HBase
HDFS is a distributed file system suitable for storing large files.	HBase is a database built on top of the HDFS.
HDFS does not support fast individual record lookups.	HBase provides fast lookups for larger tables.
It provides high-latency batch processing; no concept of batch processing.	It provides low-latency access to single rows from billions of records (Random access).
It provides only sequential access to data.	HBase internally uses Hash tables and provides random access, and it stores the data in indexed HDFS files for faster lookups.

Storage Mechanism in HBase

HBase is a **column-oriented database** and the tables in it are sorted by row. The table schema defines only column families, which are the key-value pairs. A table has multiple column families; each can have a number of columns. Subsequent column values are stored contiguously on the disk. Each cell value of the table has a timestamp. In short, in an HBase:

- Table is a collection of rows.
- Row is a collection of column families.
- Column family is a collection of columns.

- Column is a collection of key-value pairs.

  Given below is an example schema of a table in HBase.

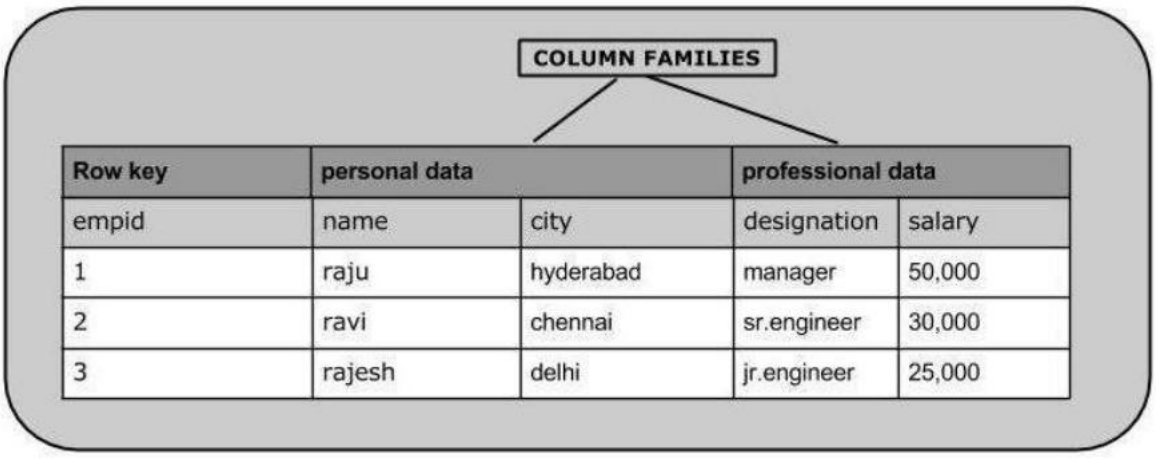
Row id	Column Family			Column Family			Column Family			Column Family		
	col1	col2	col3	col1	col2	col3	col1	col2	col3	col1	col2	col3
1												
2												
3												

Column Oriented and Row Oriented

Column-oriented databases are those that store data tables as sections of columns of data, rather than as rows of data. Shortly, they will have column families.

Row-Oriented Database	Column-Oriented Database
It is suitable for Online Transaction Process (OLTP).	It is suitable for Online Analytical Processing (OLAP).
Such databases are designed for a small number of rows and columns.	Column-oriented databases are designed for huge tables.

The following image shows column families in a column-oriented database:



HBase and RDBMS

HBase	RDBMS

HBase is schema-less, it doesn't have the concept of fixed columns schema; defines only column families.	An RDBMS is governed by its schema, which describes the whole structure of tables.
It is built for wide tables. HBase is horizontally scalable.	It is thin and built for small tables. Hard to scale.
No transactions are there in HBase.	RDBMS is transactional.
It has de-normalized data.	It will have normalized data.
It is good for semi-structured as well as structured data.	It is good for structured data.

Features of HBase

- HBase is linearly scalable.
- It has automatic failure support.
- It provides consistent reading and writing.
- It integrates with Hadoop, both as a source and a destination.
- It has easy java API for clients.
- It provides data replication across clusters.

Where to Use HBase

- Apache HBase is used to have random, real-time read/write access to Big Data.
- It hosts very large tables on top of clusters of commodity hardware.
- Apache HBase is a non-relational database modeled after Google's Big Table. Bigtable acts up on Google File System, likewise, Apache HBase works on top of Hadoop and HDFS.

Applications of HBase

- It is used whenever there is a need to write heavy applications.
- HBase is used whenever we need to provide fast random access to available data.
- Companies such as Facebook, Twitter, Yahoo, and Adobe use HBase internally.

HBase History

Year	Event
Nov 2006	Google released the paper on BigTable.
Feb 2007	The initial HBase prototype was created as a Hadoop contribution.
Oct 2007	The first usable HBase along with Hadoop 0.15.0 was released.

Jan 2008	HBase became the sub-project of Hadoop.
Oct 2008	HBase 0.18.1 was released.
Jan 2009	HBase 0.19.0 was released.
Sept 2009	HBase 0.20.0 was released.
May 2010	HBase became Apache top-level project.

Zookeeper:

- Zookeeper can be defined as a highly available service for coordinating processes of distributed applications.
- It is a centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services, which of these are used widely by distributed applications.
- The ZooKeeper framework was originally built at "Yahoo!" for accessing their applications in an easy and robust manner. Later, Apache ZooKeeper became a standard for organized service used by Hadoop, HBase, and other distributed frameworks. For example, Apache HBase uses ZooKeeper to track the status of distributed data.
- The development of zookeeper was mainly driven to face the application needs of distributed systems.
- Addition to availability, the nodes are also used to track server failures or network partitions.

Distributed systems:

- Distributed application can run on multiple systems in a network at a given time simultaneously by coordinating among themselves to complete a particular task in a fast and efficient manner.
- Normally, complex and time-consuming tasks, which will take hours to complete by a non-distributed application (running in a single system) can be done in minutes by a distributed application by using computing capabilities of all the system involved.
- Usually a group of systems in which a distributed application is running is called a Cluster and each machine running in a cluster is called a Node. A distributed application has two parts, Server and Client application. Server applications are actually distributed and have a common interface.

Usual stumbling blocks of distributed computing:

- The network is mostly not reliable.
- Delay occurs almost in every operation.
- The structure of the network keeps changing and the environment is never homogeneous.

Need for zookeeper:

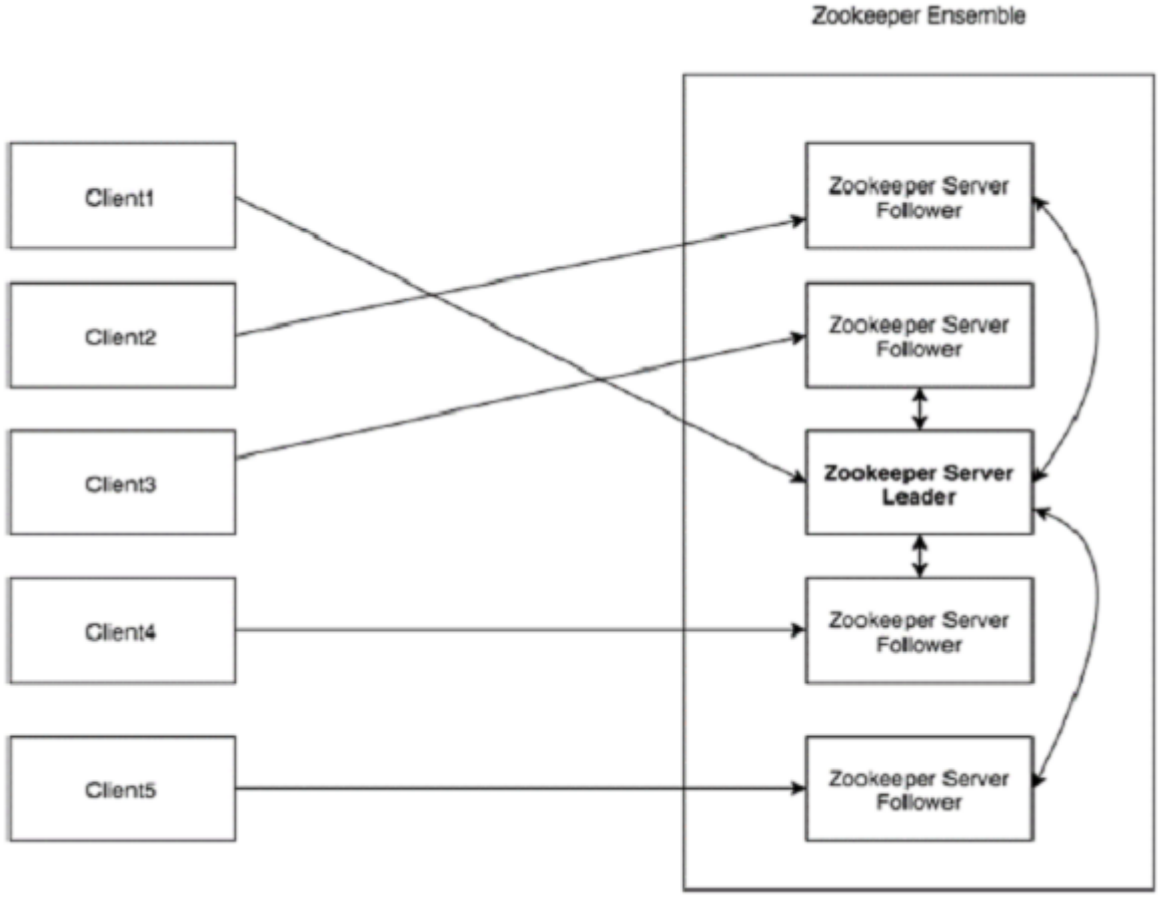
- In the past there were only programs ,which were usually homogeneous used to run in a single system using single cpu. But today, we always deal with applications consisting of independent programs running in a changing environment. Zookeeper is especially designed to relieve developers from these tedious situations.
- Zookeeper is an API that enables application developers to implement their own primitives easily, where only specific primitives can be implemented by the rest on the server side.

- Apache ZooKeeper is a service used by a cluster (group of nodes) to coordinate between themselves and maintain shared data with robust synchronization techniques. ZooKeeper is itself a distributed application providing services for writing a distributed application.

The common services provided by ZooKeeper are as follows

- **Naming service**: Identifying the nodes in a cluster by name. It is similar to DNS, but for nodes.
- **Configuration management**: Latest and up-to-date configuration information of the system for a joining node.
- **Cluster management**: Joining / leaving of a node in a cluster and node status at real time.
- **Leader election**: Electing a node as leader for coordination purpose.
- **Locking and synchronization service**: Locking the data while modifying it. This mechanism helps you in automatic fail recovery while connecting other distributed applications like Apache HBase.
- **Highly reliable data registry**: Availability of data even when one or a few nodes are down.

Architecture of Zookeeper:



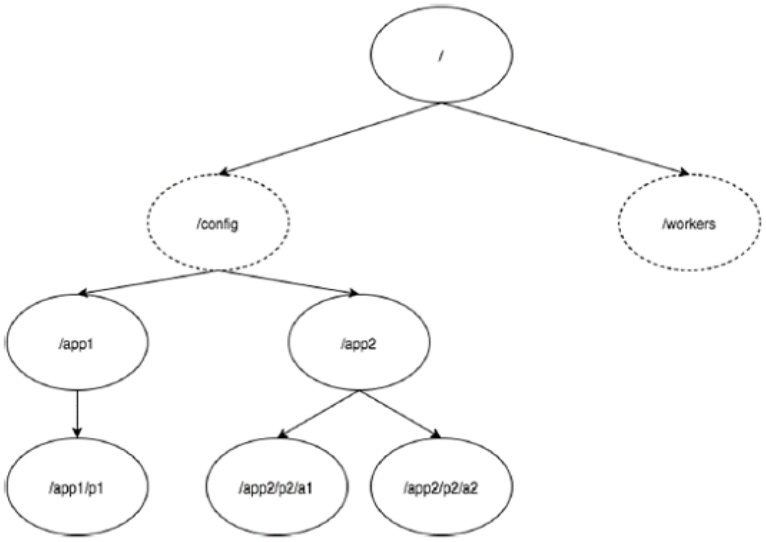
The components of the zookeeper architecture are explained in detail below.

- **Client:** These are the nodes in our distributed cluster, which accesses information from the server. For a particular time interval, every client sends a message to the server to let the sever know that the client is alive. Similarly, the server sends an acknowledgement when a client connects. If there is no response from the connected server, the client automatically redirects the message to another server.
- **Server:** Server is one of the nodes in our ZooKeeper ensemble which provides all the services to clients and gives acknowledgement to client to inform that the server is alive.
- **Ensemble:** It is a group of zookeeper servers . The minimum number of nodes that in needed to form an ensemble.
- **Leader:** It is the server node that performs automatic recovery in case of failure of any connected node. These leaders are elected during service startup.
- **Follower:** These are the server node that follows leader's instructions.

Hierarchical Namespace:

The following diagram depicts the tree structure of ZooKeeper file system used for memory representation. ZooKeeper node is referred as znode. Every znode is identified by a name and separated by a sequence of path (/).

- In the diagram, first you have a root znode separated by "/". Under root, you have two logical namespaces config and workers.
- The config namespace is used for centralized configuration management and the workers namespace is used for naming.
- Under config namespace, each znode can store upto 1MB of data. This is similar to UNIX file system except that the parent znode can store data as well. The main purpose of this structure is to store synchronized data and describe the metadata of the znode.
- This structure is called as ZooKeeper Data Model



Every znode in the ZooKeeper data model maintains a stat structure. A stat simply provides the metadata of a znode. It consists of Version number, Action control list (ACL), Timestamp, and Data length.

- **Version number**: Every znode has a version number, which means every time the data associated with the znode changes, its corresponding version number would also increased. The use of version number is important when multiple zookeeper clients are trying to perform operations over the same znode.
- **Action Control List (ACL)** : ACL is basically an authentication mechanism for accessing the znode. It governs all the znode read and write operations.
- **Timestamp**: Timestamp represents time elapsed from znode creation and modification. It is usually represented in milliseconds. ZooKeeper identifies every change to the znodes from "Transaction ID" (zxid). **Zxid** is unique and maintains time for each transaction so that you can easily identify the time elapsed from one request to another request.
- **Data length** :Total amount of the data stored in a znode is the data length. You can store a maximum of 1MB of data.

Types of Znodes

Znodes are categorized as persistence, sequential, and ephemeral.

- **Persistence znode**: Persistence znode is alive even after the client, which created that particular znode, is disconnected. By default, all znodes are persistent unless otherwise specified.
- **Ephemeral znode**:Ephemeral znodes are active until the client is alive. When a client gets disconnected from the ZooKeeper ensemble, then the ephemeral znodes get deleted automatically.
- **Sequential znode**: Sequential znodes can be either persistent or ephemeral. When a new znode is created as a sequential znode, then ZooKeeper sets the path of the znode by attaching a 10 digit sequence number to the original name.

  For example, if a znode with path **/myapp** is created as a sequential znode, ZooKeeper will change the path to **/myapp0000000001** and set the next sequence number as 0000000002. If two sequential znodes are created concurrently, then ZooKeeper never uses the same number for each znode.

- Sequential znodes play an important role in Locking and Synchronization.

Sessions and Watches:

- Sessions are very important for the operation of ZooKeeper. Requests in a session are executed in FIFO order. Once a client connects to a server, the session will be established and a **session id** is assigned to the client.
- The client sends **heartbeats** at a particular time interval to keep the session valid. If the ZooKeeper ensemble does not receive heartbeats from a client for more than the period (session timeout) specified at the starting of the service, it decides that the client died.
- Session timeouts are usually represented in milliseconds. When a session ends for any reason, the ephemeral znodes created during that session also get deleted.
- Watches are a simple mechanism for the client to get notifications about the changes in the ZooKeeper ensemble. Clients can set watches while reading a particular znode. Watches send a notification to the registered client for any of the znode (on which client registers) changes.
- Znode changes are modification of data associated with the znode or changes in the znode's children. Watches are triggered only once. If a client wants a notification again, it must be

done through another read operation. When a connection session is expired, the client will be disconnected from the server and the associated watches are also removed.

Zookeeper workflow:

Once a ZooKeeper ensemble starts, it will wait for the clients to connect. Clients will connect to one of the nodes in the ZooKeeper ensemble. It may be a leader or a follower node. Once a client is connected, the node assigns a session ID to the particular client and sends an acknowledgement to the client. If the client does not get an acknowledgment, it simply tries to connect another node in the ZooKeeper ensemble. Once connected to a node, the client will send heartbeats to the node in a regular interval to make sure that the connection is not lost.

- **If a client wants to read a particular znode,** it sends a **read request** to the node with the znode path and the node returns the requested znode by getting it from its own database. For this reason, reads are fast in ZooKeeper ensemble.
- **If a client wants to store data in the ZooKeeper ensemble**, it sends the znode path and the data to the server. The connected server will forward the request to the leader and then the leader will reissue the writing request to all the followers. If only a majority of the nodes respond successfully, then the write request will succeed and a successful return code will be sent to the client. Otherwise, the write request will fail. The strict majority of nodes is called as **Quorum**.

Nodes in a ZooKeeper Ensemble

Let us analyze the effect of having different number of nodes in the ZooKeeper ensemble.

- If we have **a single node**, then the ZooKeeper ensemble fails when that node fails. It contributes to "Single Point of Failure" and it is not recommended in a production environment.
- If we have **two nodes** and one node fails, we don't have majority as well, since one out of two is not a majority.
- If we have three nodes and one node fails, we have majority and so, it is the minimum requirement. It is mandatory for a ZooKeeper ensemble to have at least three nodes in a live production environment.
- If we have **four nodes** and two nodes fail, it fails again and it is similar to having three nodes. The extra node does not serve any purpose and so, it is better to add nodes in odd numbers, e.g., 3, 5, 7.

Zookeeper leader election:

Let us analyze how a leader node can be elected in a ZooKeeper ensemble. Consider there are N number of nodes in a cluster. The process of leader election is as follows −

- All the nodes create a sequential, ephemeral znode with the same path, /app/leader_election/guid_.
- ZooKeeper ensemble will append the 10-digit sequence number to the path and the znode created will be /app/leader_election/guid_0000000001, /app/leader_election/guid_0000000002, etc.

- For a given instance, the node which creates the smallest number in the znode becomes the leader and all the other nodes are followers.
- Each follower node watches the znode having the next smallest number. For example, the node which creates znode /app/leader_election/guid_0000000008 will watch the znode /app/leader_election/guid_0000000007 and the node which creates the znode /app/leader_election/guid_0000000007 will watch the znode /app/leader_election/guid_0000000006.
- If the leader goes down, then its corresponding znode /app/leader_electionN gets deleted.
- The next in line follower node will get the notification through watcher about the leader removal.
- The next in line follower node will check if there are other znodes with the smallest number. If none, then it will assume the role of the leader. Otherwise, it finds the node which created the znode with the smallest number as leader.
- Similarly, all other follower nodes elect the node which created the znode with the smallest number as leader.

Leader election is a complex process when it is done from scratch. But ZooKeeper service makes it very simple. Let us move on to the installation of ZooKeeper for development purpose in the next chapter.

Benefits of ZooKeeper:

- Simple coordination process distributed.
- Ordered Messages.
- Reliability throughout the process.
- **Synchronization**: Mutual exclusion and co-operation between server processes. This process helps in Apache HBase for configuration management.
- **Serialization**: Encode the data according to specific rules. Ensure your application runs consistently. This approach can be used in MapReduce to coordinate queue to execute running threads.
- **Atomicity** − Data transfer either succeed or fail completely, but no transaction is partial.

IBM INFOSPHERE BIG INSIGHTS:

InfoSphere BigInsight is a software platform for discovering, analyzing, and visualizing data from disparate sources. You use this software to help process and analyze the volume, variety, and velocity of data that continually enters your organization every day.

MAIN FEATURES OF IBM INFOSPHERE BIG INSIGHTS:

BigInsights allows organizations to cost-effectively analyze a wide variety and large volume of data to gain insights that were not previously possible.

- BigInsights is focused on providing enterprises with the capabilities they need to meet critical business requirements while maintaining compatibility with the Hadoop project.
- BigInsights includes a variety of IBM technologies that enhance and extend the value of open-source Hadoop software to facilitate faster time-to-value, including application

accelerators, analytical facilities, development tools, platform improvements and enterprise software integration.

- While BigInsights offers a wide range of capabilities that extend beyond the Hadoop functionality, IBM has taken an optin approach: you can use the IBM extensions to Hadoop based on your needs rather than being forced to use the extensions that come with InfoSphere BigInsights.
- In addition to core capabilities for installation, configuration and management, InfoSphere BigInsights includes advanced analytics and user interfaces for the non-developer business analyst.
- It is flexible to be used for unstructured or semi-structured information; the solution does not require schema definitions or data preprocessing and allows for structure and associations to be added on the fly across information types.
- The platform runs on commonly available, low-cost hardware in parallel, supporting linear scalability; as information grows, we simply add more commodity hardware.

ADVANTAGES OF IBM INFOSPHERE BIG INSIGHTS:

- Hadoop applications can exist within the System z security perimeter.
- Clients can use mainframe technologies, including IBM HiperSockets™, to securely access production data, and move that data to and from Hadoop for processing.
- Clients can realize the management advantages of running Hadoop on a private cloud infrastructure, providing configuration flexibility and virtualized storage, and avoiding need to deploy and manage discrete cluster nodes and a separate network infrastructure.
- Clients can extend System z governance to hybrid Hadoop implementations.

OPEN SOURCE UTILITIES IN INFOSPHEREBIGINSIGHTS:

InfoSphereBigInsights is 100% compatible with open source Hadoop. open source utilities in InfoSphereBigInsights:

- PIG
- Hive / HCatalog
- Oozie
- HBASE
- Zookeeper
- Flume
- Avro

ADVANCED SOFTWARE CAPABILITIES OF IBM INFOSPHERE:

- Big SQL: Big SQL is a rich, ANSI-compliant SQL implementation.
- SQL language compatibility
- Support for native data sources
- Performance
- Federation
- Security

IBM INFOSPHERE INTERFACES:

- Big R: Big R is a set of libraries that provide end-to-end integration with the popular R programming language that is included in InfoSphereBigInsights. Big R provides a familiar environment for developers and data scientists proficient with the R language.
- Big Sheets: Big Sheets is a spreadsheet style data manipulation and visualization tool that allows business users to access and analyze data in Hadoop without the need to be knowledgeable in Hadoop scripting languages or MapReduce programming. The BigSheets interface is shown in Figure 3. Using built-in line readers, BigSheets can import data in

multiple formats. In this example, it is importing data that is stored in Hive.

ACCELERATORS IN IBM INFOSPHERE BIG INSIGHTS:

WHAT IS AN ACCELERATOR?

Tools to easily import and analyze social data at scale from multiple online sources, including tweets, boards, and blogs.

TYPES:

- **Application Accelerators**: IBM InfoSphereBigInsights extends the capabilities of open source Hadoop with accelerators that use pre-written capabilities for common big data use cases to build quickly high-quality applications. Here are some of the accelerators that are included in InfoSphereBigInsights:
- **Text Analytics Accelerators:** A set of facilities for developing applications that analyze text across multiple spoken languages
- **Machine Data Accelerators**: Tools that are aimed at developers that make it easy to develop applications that process log files, including web logs, mail logs, and various specialized file formats.

WHEN TO USE BIGINSIGHTS ON A CLOUD BASED SERVICE

- Clients do not want to be bothered with acquiring and maintain a Hadoop on cluster on their own premises.
- Data that is stored and processed on Hadoop comes largely from external sources as opposed to local data sources (such as cloud-based social media aggregators).
- Clients want to retain the flexibility to alter the size of clusters up and down based on changing requirements.
- IBM Infosphere:

APACHE FLUME:

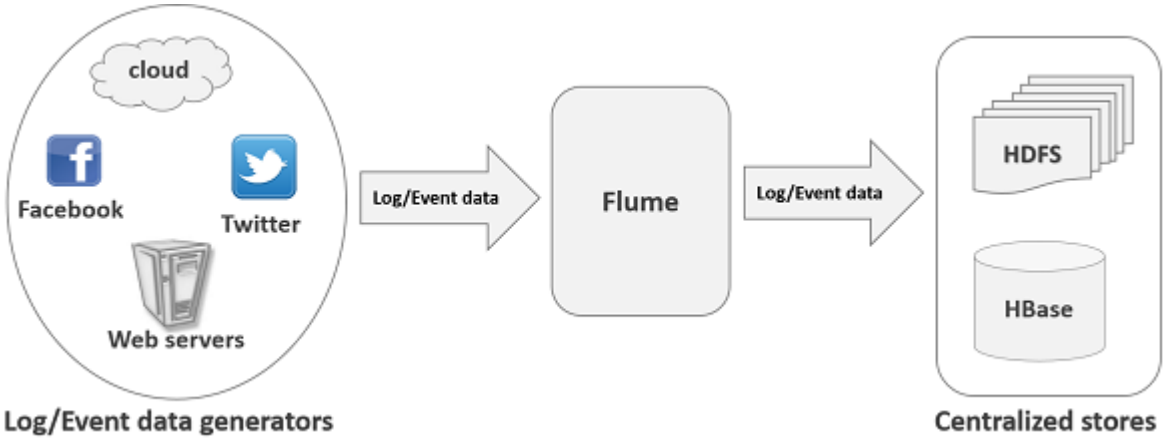
Apache Flume is a tool/service/data ingestion mechanism for collecting aggregating and transporting large amounts of streaming data such as log files, events (etc...) from various sources to a centralized data store.

Flume is a standard, simple, robust, flexible, and extensible tool for data ingestion from various data producers (webservers) into Hadoop. In this tutorial, we will be using simple and illustrative example to explain the basics of Apache Flume and how to use it in practice.

WHAT IS FLUME?

Apache Flume is a tool/service/data ingestion mechanism for collecting aggregating and transporting large amounts of streaming data such as log files, events (etc...) from various sources to a centralized data store.

Flume is a highly reliable, distributed, and configurable tool. It is principally designed to copy streaming data (log data) from various web servers to HDFS.



ADVANTAGES OF FLUME:

Here are the advantages of using Flume −

Using Apache Flume we can store the data in to any of the centralized stores (HBase, HDFS).

When the rate of incoming data exceeds the rate at which data can be written to the destination, Flume acts as a mediator between data producers and the centralized stores and provides a steady flow of data between them.

Flume provides the feature of **contextual routing**.

The transactions in Flume are channel-based where two transactions (one sender and one receiver) are maintained for each message. It guarantees reliable message delivery.

Flume is reliable, fault tolerant, scalable, manageable, and customizable.

FEATURES OF FLUME

Some of the notable features of Flume are as follows −

Flume ingests log data from multiple web servers into a centralized store (HDFS, HBase) efficiently.

Using Flume, we can get the data from multiple servers immediately into Hadoop.

Along with the log files, Flume is also used to import huge volumes of event data produced by social networking sites like Facebook and Twitter, and e-commerce websites like Amazon and Flipkart.

Flume supports a large set of sources and destinations types.

Flume supports multi-hop flows, fan-in fan-out flows, contextual routing, etc.

Flume can be scaled horizontally.

COMPONENTS OF FLUME:

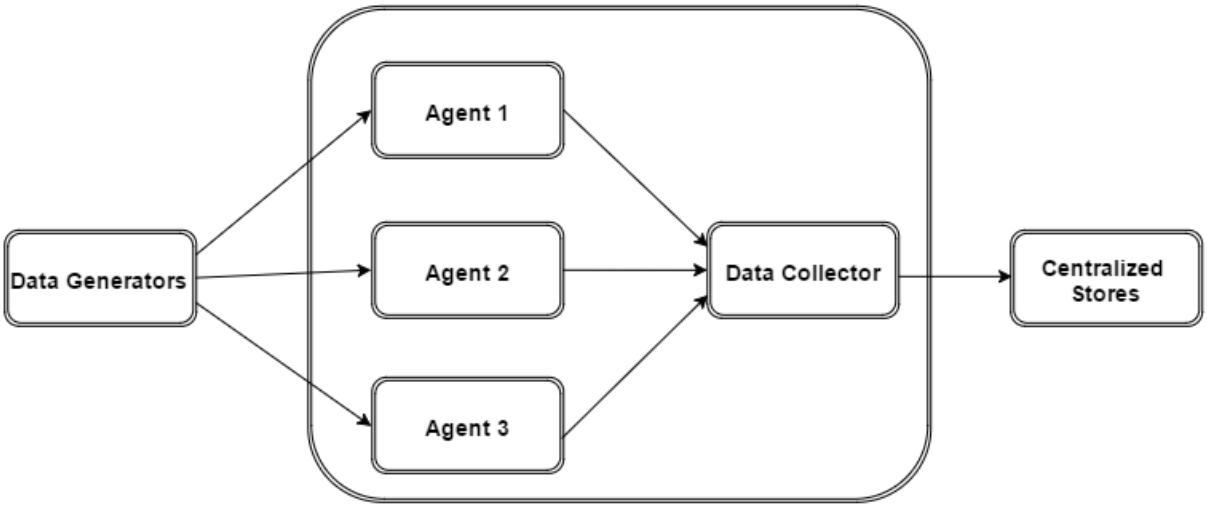
The flume agent has 3 components: source, sink and channel.

- **Source**: It accepts the data from the incoming streamline and stores the data in the channel.
- **Channel**: In general, the reading speed is faster than the writing speed. Thus, we need some buffer to match the read & write speed difference. Basically, the buffer acts as a intermediary storage that stores the data being transferred temporarily and therefore prevents data loss. Similarly, channel acts as the local storage or a temporary storage between the source of data and persistent data in the HDFS.
- **Sink**: Then, our last component i.e. Sink, collects the data from the channel and commits or writes the data in the HDFS permanently.

- **Interceptors**:Interceptors are used to alter/inspect flume events which are transferred between source and channel.
- **Channel Selector:**These are used to determine which channel is to be opted to transfer the data in case of multiple channels. There are two types of channel selectors .
- **Default channel selectors** − These are also known as replicating channel selectors they replicates all the events in each channel.
- **Multiplexing channel selectors** − These decides the channel to send an event based on the address in the header of that event.
- **Sink Processors**:These are used to invoke a particular sink from the selected group of sinks. These are used to create failover paths for your sinks or load balance events across multiple sinks from a channel.

ARCHITECTURE OF FLUME:

The following illustration depicts the basic architecture of Flume. As shown in the illustration, **data generators** (such as Facebook, Twitter) generate data which gets collected by individual Flume **agents** running on them. Thereafter, a **data collector** (which is also an agent) collects the data from the agents which is aggregated and pushed into a centralized store such as HDFS or HBase.

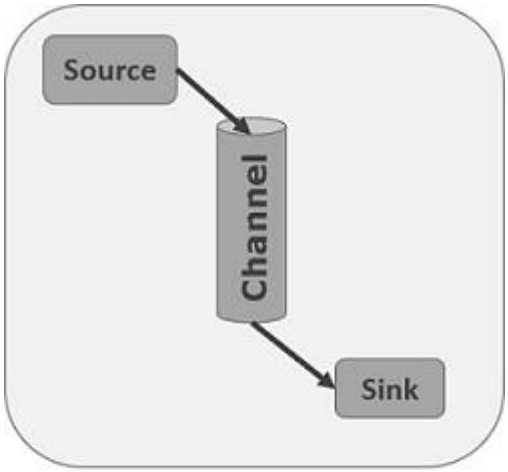


FLUME EVENT

An **event** is the basic unit of the data transported inside **Flume**. It contains a payload of byte array that is to be transported from the source to the destination accompanied by optional headers. A typical Flume event would have the following structure.

FLUME AGENT

An agent is an independent daemon process (JVM) in Flume. It receives the data (events) from clients or other agents and forwards it to its next destination (sink or agent). Flume may have more than one agent. Following diagram represents a Flume Agent.



Flume Agent

SOURCE:

A **source** is the component of an Agent which receives data from the data generators and transfers it to one or more channels in the form of Flume events.Apache Flume supports several types of sources and each source receives events from a specified data generator.

Example − Avro source, Thrift source, twitter 1% source etc.

CHANNEL:

A **channel** is a transient store which receives the events from the source and buffers them till they are consumed by sinks. It acts as a bridge between the sources and the sinks.

These channels are fully transactional and they can work with any number of sources and sinks.

Example − JDBC channel, File system channel, Memory channel, etc.

Sink:

A **sink** stores the data into centralized stores like HBase and HDFS. It consumes the data (events) from the channels and delivers it to the destination. The destination of the sink might be another agent or the central stores.

Example − HDFS sink

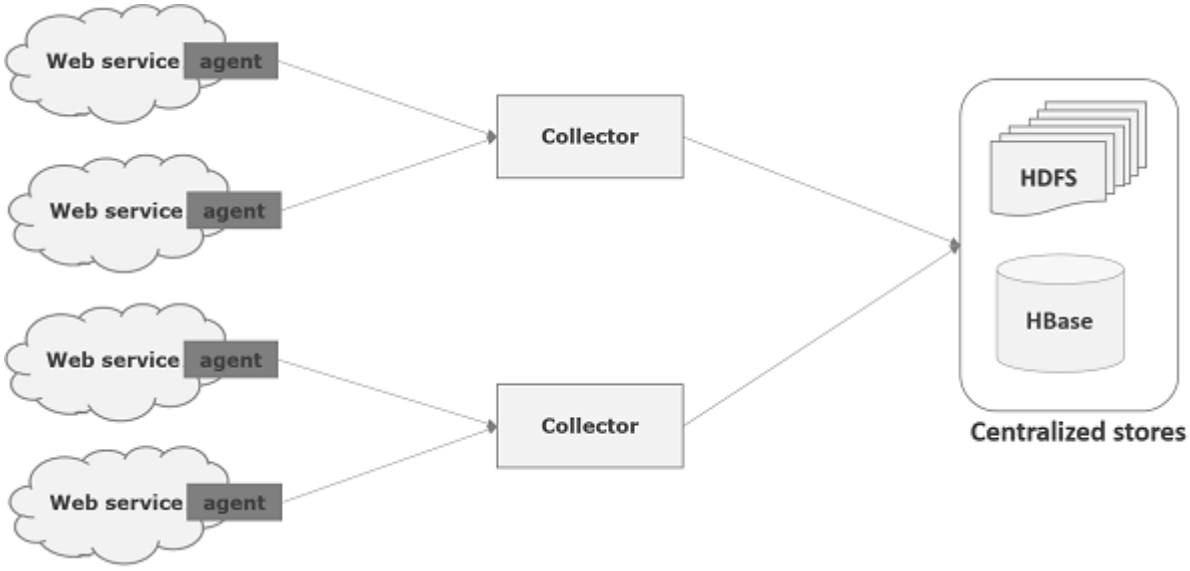
Note − A flume agent can have multiple sources, sinks and channels. We have listed all the supported sources, sinks, channels in the Flume configuration chapter of this tutorial.

APACHE FLUME DATA FLOW:

Flume is a framework which is used to move log data into HDFS. Generally events and log data are generated by the log servers and these servers have Flume agents running on them. These agents receive the data from the data generators.

The data in these agents will be collected by an intermediate node known as **Collector**. Just like agents, there can be multiple collectors in Flume.

Finally, the data from all these collectors will be aggregated and pushed to a centralized store such as HBase or HDFS. The following diagram explains the data flow in Flume.



-

Multi-hop Flow

Within Flume, there can be multiple agents and before reaching the final destination, an event may travel through more than one agent. This is known as **multi-hop flow**.

Fan-out Flow

The dataflow from one source to multiple channels is known as **fan-out flow**. It is of two types −

- **Replicating** − The data flow where the data will be replicated in all the configured channels.
- **Multiplexing** − The data flow where the data will be sent to a selected channel which is mentioned in the header of the event.

Fan-in Flow

The data flow in which the data will be transferred from many sources to one channel is known as **fan-in flow**.

Failure Handling

In Flume, for each event, two transactions take place: one at the sender and one at the receiver. The sender sends events to the receiver. Soon after receiving the data, the

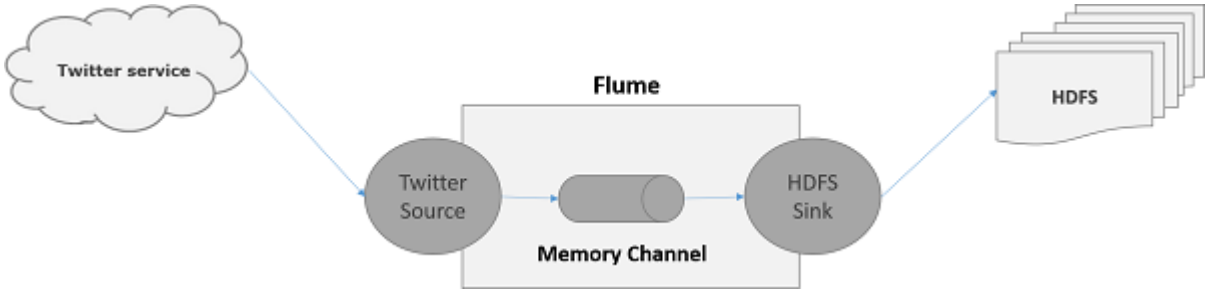
receiver commits its own transaction and sends a “received” signal to the sender. After receiving the signal, the sender commits its transaction. (Sender will not commit its transaction till it receives a signal from the receiver.)

APACHE FLUME-FETCHING FLUME DATA:

Using Flume, we can fetch data from various services and transport it to centralized stores (HDFS and HBase). This chapter explains how to fetch data from Twitter service and store it in HDFS using Apache Flume.

As discussed in Flume Architecture, a webserver generates log data and this data is collected by an agent in Flume. The channel buffers this data to a sink, which finally pushes it to centralized stores.

In the example provided in this chapter, we will create an application and get the tweets from it using the experimental twitter source provided by Apache Flume. We will use the memory channel to buffer these tweets and HDFS sink to push these tweets into the HDFS.



To fetch Twitter data, we will have to follow the steps given below −

- Create a twitter Application
- Install / Start HDFS
- Configure Flume

APPLICATIONS OF FLUME:

Assume an e-commerce web application wants to analyze the customer behavior from a particular region. To do so, they would need to move the available log data in to Hadoop for analysis. Here, Apache Flume comes to our rescue.

Flume is used to move the log data generated by application servers into HDFS at a higher speed.

KAFKA:

Introduction:

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 based fault tolerant messaging system. It is fast, scalable and distributed by design.

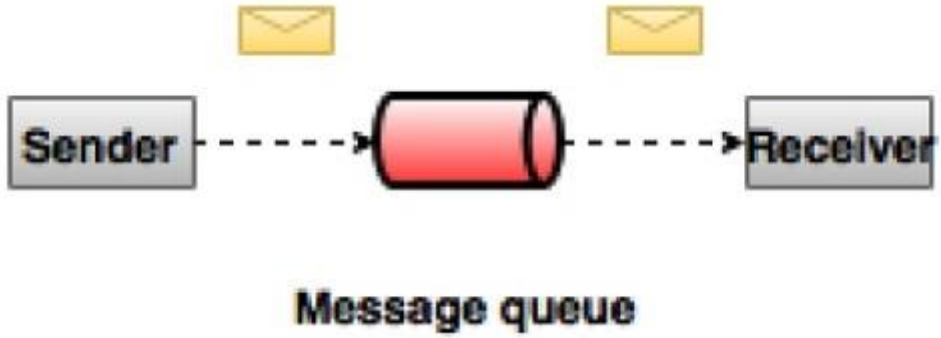
Kafka tends to work very well as a replacement for a more traditional message broker. In comparison to other messaging systems, Kafka has better throughput, built-in partitioning, replication and inherent fault-tolerance, which makes it a good fit for large-scale message processing applications.

A Messaging 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. Two types of messaging patterns are available − one is point to point and the other is publish-subscribe (pub-sub) messaging system. Most of the messaging patterns follow pub-sub.

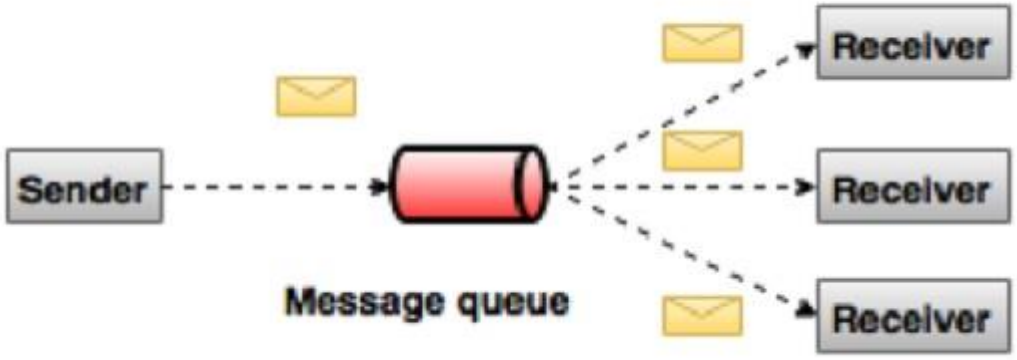
Point to Point Messaging System

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. The typical example of this system is an Order Processing System, where each order will be processed by one Order Processor, but Multiple Order Processors can work as well at the same time. The following diagram depicts the structure.



Publish-Subscribe 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. A real-life 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.



What is Kafka?

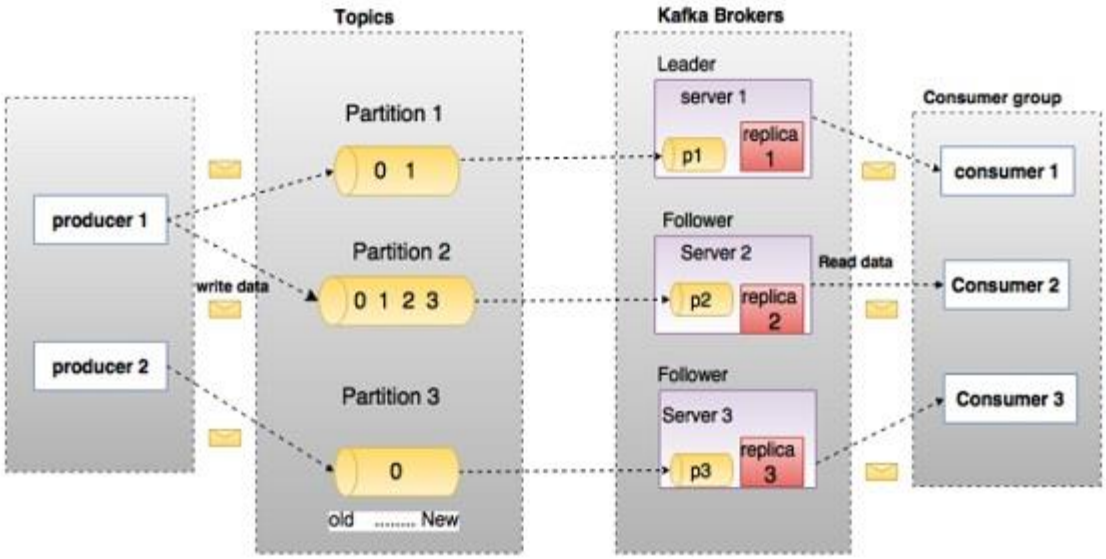
Apache Kafka is a distributed publish-subscribe messaging system and a robust queue that can handle a high volume of data and enables you to pass messages from one end-point to another. Kafka is suitable for both offline and online message consumption. Kafka messages are persisted on the disk and replicated 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.

Need for Kafka

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.

Apache Kafka – Fundamentals

Before moving deep into the Kafka, you must aware of the main terminologies such as topics, brokers, producers and consumers. The following diagram illustrates the main terminologies and the table describes the diagram components in detail.



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.

Components of KAFKA:

TOPIC: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 partitions, so it can handle an arbitrary amount of data.

PARTITON 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 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.

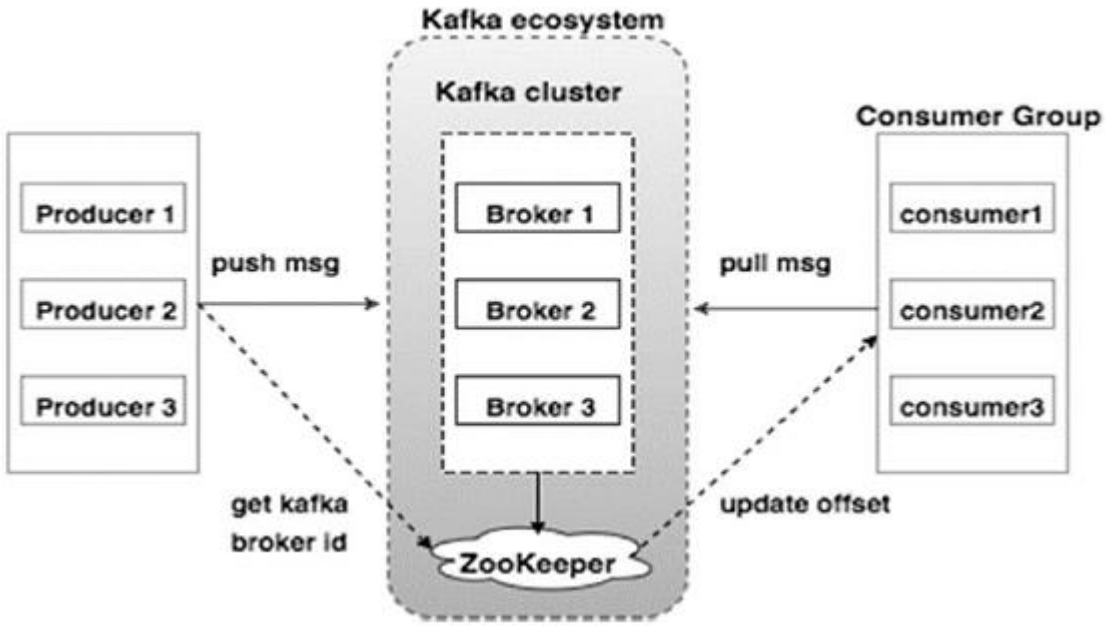
PRODUCERS: Producers are the publisher of messages to one or more Kafka topics. Producers send data to Kafka brokers. Every time a producer pub-lishes a message to a broker, the broker simply appends the message to the last segment file. Actually, the message will be appended to a partition. Producer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

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 up-dates its own data store.

APACHE KAFKA CLUSTER ARCHITECTURE:



BROKER: Kafka cluster typically consists of multiple brokers to maintain load balance. 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.

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.

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. Kafka producer doesn't wait for acknowledgements from the broker and sends messages as fast as the broker can handle.

CONSUMERS: Since Kafka brokers are stateless, which means that the 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.

WORKFLOW OF PUB-SUB MESSAGING

Following is the step wise workflow of the Pub-Sub Messaging

- Producers send message to a topic at regular intervals.
- 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.
- Consumer subscribes to a specific topic.
- 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.
- Consumer will request the Kafka in a regular interval (like 100 Ms) for new messages.
- Once Kafka receives the messages from producers, it forwards these messages to the consumers.
- Consumer will receive the message and process it.
- Once the messages are processed, consumer will send an acknowledgement to the Kafka broker.
- 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.
- This above flow will repeat until the consumer stops the request.
- Consumer has the option to rewind/skip to the desired offset of a topic at any time and read all the subsequent messages.

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. In simple terms, 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.

- Producers send message to a topic in a regular interval.
- Kafka stores all messages in the partitions configured for that particular topic similar to the earlier scenario.
- A single consumer subscribes to a specific topic, assume Topic-01 with Group ID as Group-1.
- 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.

- 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.
- 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.
- 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.

ROLE OF ZOOKEEPER IN KAFKA:

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.

APACHE KAFKA-INTEGRATION WITH STORM:

ABOUT STORM:

Storm was originally created by Nathan Marz and team at BackType. In a short time, Apache Storm became a standard for distributed real-time processing system that allows you to process a huge volume of data. Storm is very fast and a benchmark clocked it at over a million tuples processed per second per node. Apache Storm runs continuously, consuming data from the configured sources (Spouts) and passes the data down the processing pipeline (Bolts). Com-bined, Spouts and Bolts make a Topology.

INTEGRATION WITH STORM:

Kafka and Storm naturally complement each other, and their powerful cooperation enables real-time streaming analytics for fast-moving big data. Kafka and Storm integration is to make easier for developers to ingest and publish data streams from Storm topologies.

CONCEPTUAL FLOW:

A spout is a source of streams. For example, a spout may read tuples off a Kafka Topic and emit them as a stream. A bolt consumes input streams, process and possibly emits new streams. Bolts can do anything from running functions, filtering tuples, do streaming aggregations, streaming joins, talk to databases, and more. Each node in a Storm topology executes in parallel. A topology runs indefinitely until you terminate it. Storm will automatically reassign any failed tasks. Additionally, Storm guarantees that there will be no data loss, even if the machines go down and messages are dropped.

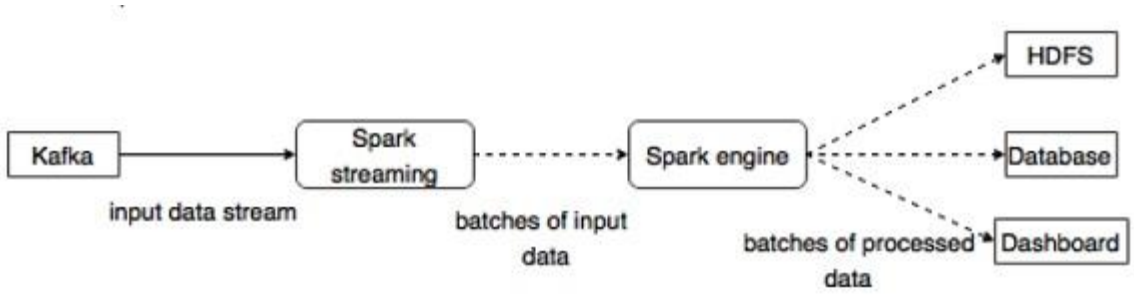
APACHE KAFKA-INTEGRATION WITH SPARK:

ABOUT SPARK

Spark Streaming API enables scalable, high-throughput, fault-tolerant stream processing of live data streams. Data can be ingested from many sources like Kafka, Flume, Twitter, etc., and can be processed using complex algorithms such as high-level functions like map, reduce, join and window. Finally, processed data can be pushed out to filesystems, databases, and live dash-boards. Resilient Distributed Datasets (RDD) is a fundamental data structure of Spark. It is an immutable distributed collection of objects. Each dataset in RDD is divided into logical partitions, which may be computed on different nodes of the cluster.

INTEGRATION WITH SPARK

Kafka is a potential messaging and integration platform for Spark streaming. Kafka act as the central hub for real-time streams of data and are processed using complex algorithms in Spark Streaming. Once the data is processed, Spark Streaming could be publishing results into yet another Kafka topic or store in HDFS, databases or dashboards. The following diagram depicts the conceptual flow.



APACHE KAFKA TOOLS:

SYSTEM TOOLS:

System tools can be run from the command line using the run class script. The syntax is as follows .

Kafka Migration Tool : This tool is used to migrate a broker from one version to an-other.
Mirror Maker : This tool is used to provide mirroring of one Kafka cluster to another.
Consumer Offset Checker : This tool displays Consumer Group, Topic, Partitions, Off-set, logSize, Owner for the specified set of Topics and Consumer Group.

REPLICATION TOOL:

Kafka replication is a high level design tool. The purpose of adding replication tool is for stronger durability and higher availability. Some of the replication tools are mentioned below.

Create Topic Tool : This creates a topic with a default number of partitions, replication factor and uses Kafka's default scheme to do replica assignment.
List Topic Tool : This tool lists the information for a given list of topics. If no topics are provided in the command line, the tool queries Zookeeper to get all the topics and lists the information for them. The fields that the tool displays are topic name, partition, leader and replicas.

Add Partition Tool : Creation of a topic, the number of partitions for topic has to be specified. Later on, more partitions may be needed for the topic, when the volume of the topic will increase. This tool helps to add more partitions for a specific topic and also allows manual replica assignment of the added partitions.

BENEFITS OF KAFKA

Following are a few benefits of Kafka −

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 -APPLICATIONS:

Twitter

Twitter is an online social networking service that provides a platform to send and receive user tweets. Registered users can read and post tweets, but unregistered users can only read tweets. Twitter uses Storm-Kafka as a part of their stream processing infrastructure.

LinkedIn

Apache Kafka is used at LinkedIn for activity stream data and operational metrics. Kafka mes-saging system helps LinkedIn with various products like LinkedIn Newsfeed, LinkedIn Today for online message consumption and in addition to offline analytics systems like Hadoop. Kafka's strong durability is also one of the key factors in connection with LinkedIn.

Netflix

Netflix is an American multinational provider of on-demand Internet streaming media. Netflix uses Kafka for real-time monitoring and event processing.

Mozilla

Mozilla is a free-software community, created in 1998 by members of Netscape. Kafka will soon be replacing a part of Mozilla current production system to collect performance and usage data from the end-user's browser for projects like Telemetry, Test Pilot, etc.

Oracle

Oracle provides native connectivity to Kafka from its Enterprise Service Bus product called OSB (Oracle Service Bus) which allows developers to leverage OSB built-in mediation capabilities to implement staged data pipelines.