**CHAPTER 1**

**INTRODUCTION TO BIG DATA**

In order to understand 'Big Data', you first need to know:

**1.1 WHAT IS DATA?**

The quantities, characters, or symbols on which operations are performed by a computer, which may be stored and transmitted in the form of electrical signals and recorded on magnetic, optical, or mechanical recording media.

**1.2 WHAT IS BIG DATA?**

Big Data is also data but with a huge size. Big Data is a term used to describe a collection of data that is huge in size and yet growing exponentially with time. In short, such data is so large and complex that none of the traditional data management tools are able to store it or process it efficiently. Big data was originally associated with three key concepts: volume, variety, and velocity. A 2018 definition states "Big data is where parallel computing tools are needed to handle data", and notes, "This represents a distinct and clearly defined change in the computer science used, via parallel programming theories, and losses of some of the guarantees and capabilities made by Codd’s relational model.

The growing maturity of the concept more starkly delineates the difference between

“big data” and “business intelligence”

* Business Intelligence uses applied mathematics tools and descriptive statistics with data with high information density to measure things, detect trends, etc.
* Big data uses mathematical analysis, optimization, inductive statistics and concepts from nonlinear system identification to infer laws (regressions, nonlinear relationships, and causal effects) from large sets of data with low information densityto reveal relationships and dependencies, or to perform predictions of outcomes and behaviors.

# 1.3 EXAMPLES OF BIG DATA

Following are some examples of big data –

* The New York Stock Exchange generates about one terabyte of new trade data per day.



Fig. 1.3.1

* Social Media :-

The statistic shows that 500+terabytes of new data get ingested into the databases of social media site Facebook, every day. This data is mainly generated in terms of photo and video uploads, message exchanges, putting comments etc.

* A single Jet enginecan generate 10+terabytes of data in 30 minutes of flight time. With many thousand flights per day, generation of data reaches up to many Petabytes.

# 1.4 TYPES OF BIG DATA

Big data can be found in 3 forms :

1. Structured
2. Unstructured
3. Semi-structured

## 1.4.1 STRUCTURED

Any data that can be stored, accessed and processed in the form of fixed format is termed as a 'structured' data. Over the period of time, talent in computer science has achieved greater success in developing techniques for working with such kind of data (where the format is well known in advance) and also deriving value out of it. However, nowadays, we are foreseeing issues when a size of such data grows to a huge extent, typical sizes are being in the rage of multiple zettabytes.

**Example of structured data :**

An 'Employee' table in a database is an example of Structured Data

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Emp\_ID** | **Employee\_Name** | **Gender** | **Department** | **Salary\_In\_lacs** |
| 2365 | Rajesh Kulkarni | Male | Finance | 650000 |
| 3398 | Pratibha Joshi | Female | Admin | 650000 |
| 7465 | Shushil Roy | Male | Admin | 500000 |

Table 1.4.1.1

## 1.4.2 UNSTRUCTURED

Any data with unknown form or the structure is classified as unstructured data. In addition to the size being huge, un-structured data poses multiple challenges in terms of its processing for deriving value out of it. A typical example of unstructured data is a heterogeneous data source containing a combination of simple text files, images, videos etc. Now day organizations have wealth of data available with them but unfortunately, they don't know

how to derive value out of it since this data is in its raw form or unstructured format.

**Example of Un-Structured Data :**

The output returned by Google search

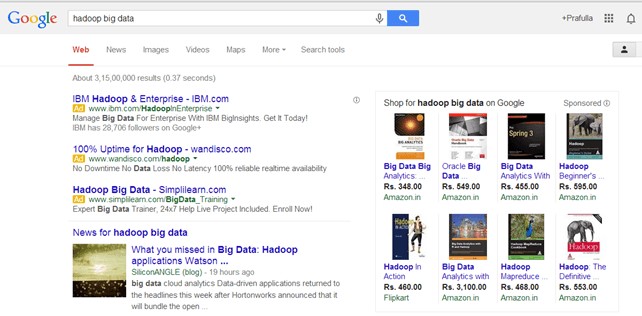


Fig. 1.4.2.1

## 1.4.3 SEMI-STRUCTURED

Semi-structured data can contain both the forms of data. We can see semistructured data as a structured in form but it is actually not defined with e.g. a table definition in relational DBMS. Example of semi-structured data is a data represented in an XML file.

**Examples of Semi-Structured Data :**

Personal data stored in an XML file-

<rec><name>Prashant Rao</name><sex>Male</sex><age>35</age></rec>

<rec><name>Seema R.</name><sex>Female</sex><age>41</age></rec>

<rec><name>Satish Mane</name><sex>Male</sex><age>29</age></rec>

<rec><name>Subroto Roy</name><sex>Male</sex><age>26</age></rec>

<rec><name>Jeremiah J.</name><sex>Male</sex><age>35</age></rec>

## 1.4.4 DATA GROWTH OVER YEARS

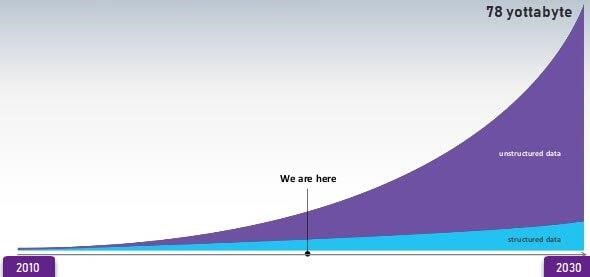


Fig. 1.4.4.1

Please note that web application data, which is unstructured, consists of log files, transaction history files etc. OLTP systems are built to work with structured data wherein data is stored in relations (tables).

# 1.5 CHARACTERSTICS OF BIG DATA

1. Volume – The name Big Data itself is related to a size which is enormous. Size of data plays a very crucial role in determining value out of data. Also, whether a particular data can actually be considered as a Big Data or not, is dependent upon the volume of data. Hence, 'Volume' is one characteristic which needs to be considered while dealing with Big Data.
2. Variety – The next aspect of Big Data is its variety.

Variety refers to heterogeneous sources and the nature of data, both structured and unstructured. During earlier days, spreadsheets and databases were the only sources of data considered by most of the applications. Nowadays, data in the form of emails, photos, videos, monitoring devices, PDFs, audio, etc. are also being considered in the analysis applications. This variety of unstructured data poses certain issues for storage, mining and analyzing data.

1. Velocity – The term 'velocity' refers to the speed of generation of data. How fast the data is generated and processed to meet the demands, determines real potential in the data.

Big Data Velocity deals with the speed at which data flows in from sources like business processes, application logs, networks, and social media sites, sensors, Mobile devices, etc. The flow of data is massive and continuous.

1. Variability – This refers to the inconsistency which can be shown by the data at times, thus hampering the process of being able to handle and manage the data effectively.

# 1.6 BENEFITS OF BIG DATA PROCESSING

Ability to process Big Data brings in multiple benefits, such as-

* Businesses can utilize outside intelligence while taking decisions

Access to social data from search engines and sites like Facebook, Twitter are enabling organizations to fine tune their business strategies.

* Improved customer service

Traditional customer feedback systems are getting replaced by new systems designed with Big Data technologies. In these new systems, Big Data and natural language processing technologies are being used to read and evaluate consumer responses.

* Early identification of risk to the product/services, if any
* Better operational efficiency

Big Data technologies can be used for creating a staging area or landing zone for new data before identifying what data should be moved to the data warehouse. In addition, such integration of Big Data technologies and data warehouse helps an organization to offload infrequently accessed data.

**CHAPTER 2**

**INTRODUCTION TO HADOOP**

# 2.1 WHAT IS HADOOP

Apache Hadoop is an open source software framework used to develop data processing applications which are executed in a distributed computing environment.

Applications built using HADOOP are run on large data sets distributed across clusters of commodity computers. Commodity computers are cheap and widely available. These are mainly useful for achieving greater computational power at low cost.

Similar to data residing in a local file system of a personal computer system, in

Hadoop, data resides in a distributed file system which is called as a Hadoop Distributed File system. The processing model is based on 'Data Locality' concept wherein computational logic is sent to cluster nodes(server) containing data. This computational logic is nothing, but a compiled version of a program written in a high-level language such as Java. Such a program, processes data stored in Hadoop HDFS.

# 2.2 HADOOP ECOSYSTEM AND COMPONENTS

Below diagram shows various components in the Hadoop ecosystem-

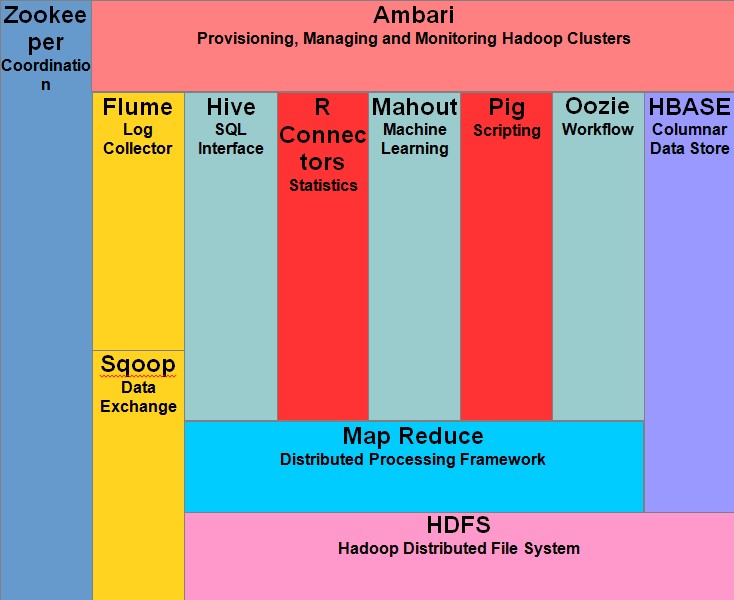


Fig. 2.2.1

Apache Hadoop consists of two sub-projects –

## 2.2.1 Hadoop MapReduce

MapReduce is a computational model and software framework for writing applications which are run on Hadoop. These MapReduce programs are capable of processing enormous data in parallel on large clusters of computation nodes.

## 2.2.2 HDFS (Hadoop Distributed File System)

HDFS takes care of the storage part of Hadoop applications. MapReduce applications consume data from HDFS. HDFS creates multiple replicas of data blocks and distributes them on compute nodes in a cluster. This distribution enables reliable and extremely rapid computations.

Although Hadoop is best known for MapReduce and its distributed file system- HDFS, the term is also used for a family of related projects that fall under the umbrella of distributed computing and large-scale data processing. Other Hadoop-related projects at Apache include are Hive, HBase, Mahout, Sqoop, Flume, and ZooKeeper**.**

# 2.3 HADOOP ARCHITECTURE

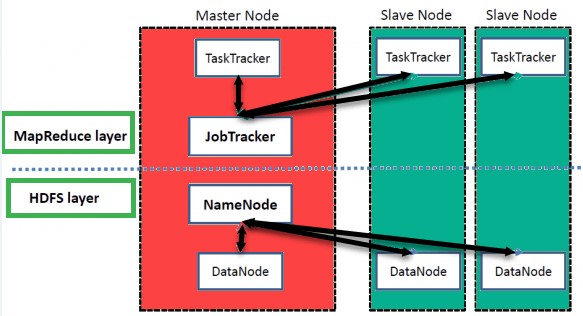


Fig. 2.3.1

Hadoop has a Master-Slave Architecture for data storage and distributed data processing using MapReduce and HDFS methods.

**NameNode:**

NameNode represented every files and directory which is used in the namespace

**DataNode:**

DataNode helps you to manage the state of an HDFS node and allows you to interacts with the blocks

**MasterNode:**

The master node allows you to conduct parallel processing of data using Hadoop MapReduce.

**Slave node:**

The slave nodes are the additional machines in the Hadoop cluster which allows you to store data to conduct complex calculations. Moreover, all the slave node comes with Task Tracker and a DataNode. This allows you to synchronize the processes with the NameNode and Job Tracker respectively.

In Hadoop, master or slave system can be set up in the cloud or on-premise.

# 2.4 FEATURES OF HADOOP

* **Suitable for Big Data Analysis**

As Big Data tends to be distributed and unstructured in nature, HADOOP clusters are best suited for analysis of Big Data. Since it is processing logic (not the actual data) that flows to the computing nodes, less network bandwidth is consumed. This concept is called as data locality concept which helps increase the efficiency of Hadoop based applications.

* **Scalability**

HADOOP clusters can easily be scaled to any extent by adding additional cluster nodes and thus allows for the growth of Big Data. Also, scaling does not require modifications to application logic.

* **Fault Tolerance**

HADOOP ecosystem has a provision to replicate the input data on to other cluster nodes. That way, in the event of a cluster node failure, data processing can still proceed by using data stored on another cluster node.

# 2.5 HDFS ARCHITECTURE

Hdfs cluster primarily consists of a namenode that manages the file system metadata and a data node that stores the actualdata.

* **Namenode:** namenode can be considered as a master of the system. It maintains the file system tree and the metadata for all the files and directories present in the system. Two files 'namespaceimage' and the 'editlog' are used to store metadata information. Name node has knowledge of all the data nodes containing data blocks for a given file, however, it does not store block locations persistently. This information is reconstructed every time from data nodes when the system starts.
* **Datanode:** data nodes are slaves which reside on each machine in a cluster and provide the actual storage. It is responsible for serving, read and write requests for the clients.

Read/write operations in hdfs operate at a block level. Data files in hdfs are broken into block-sized chunks, which are stored as independent units. Default block-size is 64 mb.

## 2.5.1 READ OPERATION IN HDFS

Data read request is served by HDFS, NameNode, and DataNode. Let's call the reader as a 'client'. Below diagram depicts file read operation in Hadoop.

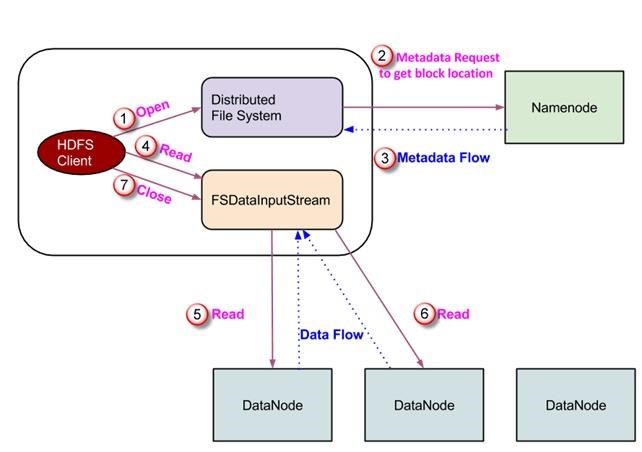


Fig. 2.5.1.1

1. A client initiates read request by calling 'open**()'** method of FileSystem object; it is an object of type DistributedFileSystem.
2. This object connects to namenode using RPC and gets metadata information such as the locations of the blocks of the file. Please note that these addresses are of first few blocks of a file.
3. In response to this metadata request, addresses of the DataNodes having a copy of that block is returned back.
4. Once addresses of DataNodes are received, an object of type FSDataInputStream is returned to the client. FSDataInputStream contains DFSInputStream which takes care of interactions with DataNode and NameNode. In step 4 shown in the above diagram, a client invokes 'read**()'** method which causes DFSInputStream to establish a connection with the first DataNode with the first block of a file.
5. Data is read in the form of streams wherein client invokes 'read**()'** method repeatedly. This process of read**()** operation continues till it reaches the end of block.
6. Once the end of a block is reached, DFSInputStream closes the connection and moves on to locate the next DataNode for the next block
7. Once a client has done with the reading, it calls aclose**()** method.

## 2.5.2 WRITE OPERATION IN HDFS

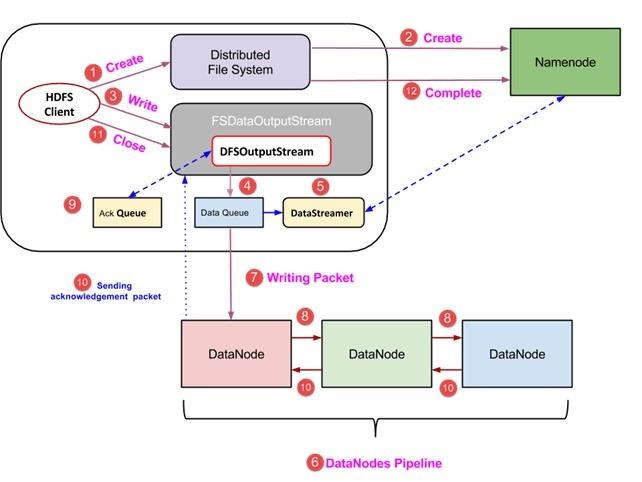


Fig. 2.5.2.1

1. A client initiates write operation by calling 'create()' method of DistributedFileSystem object which creates a new file - Step no. 1 in the above diagram.
2. DistributedFileSystem object connects to the NameNode using RPC call and initiates new file creation. However, this file creates operation does not associate any blocks with the file. It is the responsibility of NameNode to verify that the file (which is being created) does not exist already and a client has correct permissions to create a new file. If a file already exists or client does not have sufficient permission to create a new file, then IOException is thrown to the client. Otherwise, the operation succeeds and a new record for the file is created by the NameNode.
3. Once a new record in NameNode is created, an object of type

FSDataOutputStream is returned to the client. A client uses it to write data into the HDFS. Data write method is invoked (step 3 in the diagram).

1. FSDataOutputStream contains DFSOutputStream object which looks after communication with DataNodes and NameNode. While the client continues writing data, DFSOutputStream continues creating packets with this data. These packets are enqueued into a queue which is called as DataQueue.
2. There is one more component called DataStreamer which consumes this DataQueue. DataStreamer also asks NameNode for allocation of new blocks thereby picking desirable DataNodes to be used for replication. 6. Now, the process of replication starts by creating a pipeline using DataNodes. In our case, we have chosen a replication level of 3 and hence there are 3 DataNodes in the pipeline.
3. The DataStreamer pours packets into the first DataNode in the pipeline.
4. Every DataNode in a pipeline stores packet received by it and forwards the same to the second DataNode in a pipeline.
5. Another queue, 'Ack Queue' is maintained by DFSOutputStream to store packets which are waiting for acknowledgment from DataNodes.
6. Once acknowledgment for a packet in the queue is received from all DataNodes in the pipeline, it is removed from the 'Ack Queue'. In the event of any DataNode failure, packets from this queue are used to reinitiate the operation.
7. After a client is done with the writing data, it calls a close() method (Step 9 in the diagram) Call to close(), results into flushing remaining data packets to the pipeline followed by waiting for acknowledgment.
8. Once a final acknowledgment is received, NameNode is contacted to tell it that the file write operation is complete.

# 2.6 MAP-REDUCE ARCHITECTURE AND WORKING

MapReduce is a programming model suitable for processing of huge data. Hadoop is capable of running MapReduce programs written in various languages: Java, Ruby, Python, and C++. MapReduce programs are parallel in nature, thus are very useful for performing large-scale data analysis using multiple machines in the cluster.

MapReduce programs work in two phases:

1. Map phase 2. Reduce phase.

An input to each phase is key-value pairs. In addition, every programmer needs to specify two functions: map function and reduce function.

## 2.6.1 MAP REDUCE WORKING

The whole process goes through four phases of execution namely, splitting, mapping, shuffling, and reducing.

Let's understand this with an example –

Consider you have following input data for your Map Reduce Program

Welcome to Hadoop Class

Hadoop is good

Hadoop is bad

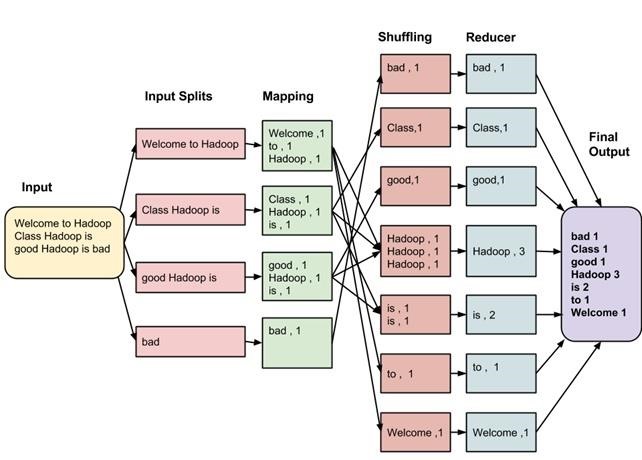


Fig. 2.6.1.1

The final output of the MapReduce task is

|  |  |
| --- | --- |
| bad | 1 |
| Class | 1 |
| good | 1 |
| Hadoop | 3 |
| is | 2 |
| to | 1 |
| Welcome | 1 |

Table 2.6.1.1

The data goes through the following phases

Input Splits:

An input to a MapReduce job is divided into fixed-size pieces called inputsplitsInput split is a chunk of the input that is consumed by a single map Mapping:

This is the very first phase in the execution of map-reduce program. In this phase data in each split is passed to a mapping function to produce output values. In our example, a job of mapping phase is to count a number of occurrences of each word from input splits (more details about input-split is given below) and prepare a list in the form of <word, frequency>

Shuffling:

This phase consumes the output of Mapping phase. Its task is to consolidate the relevant records from Mapping phase output. In our example, the same words are clubed together along with their respective frequency.

Reducing:

In this phase, output values from the Shuffling phase are aggregated. This phase combines values from Shuffling phase and returns a single output value. In short, this phase summarizes the complete dataset.

In our example, this phase aggregates the values from Shuffling phase i.e., calculates total occurrences of each word.

## 2.6.2 MAP REDUCE ARCHITECTURE

* One map task is created for each split which then executes map function for each record in the split.
* It is always beneficial to have multiple splits because the time taken to process a split is small as compared to the time taken for processing of the whole input. When the splits are smaller, the processing is better to load balanced since we are processing the splits in parallel.
* However, it is also not desirable to have splits too small in size. When splits are too small, the overload of managing the splits and map task creation begins to dominate the total job execution time.
* For most jobs, it is better to make a split size equal to the size of an HDFS block (which is 64 MB, by default).
* Execution of map tasks results into writing output to a local disk on the respective node and not to HDFS.
* Reason for choosing local disk over HDFS is, to avoid replication which takes place in case of HDFS store operation.
* Map output is intermediate output which is processed by reduce tasks to produce the final output.
* Once the job is complete, the map output can be thrown away. So, storing it in HDFS with replication becomes overkill.
* In the event of node failure, before the map output is consumed by the reduce task, Hadoop reruns the map task on another node and re-creates the map output.
* Reduce task doesn't work on the concept of data locality. An output of every map task is fed to the reduce task. Map output is transferred to the machine where reduce task is running.
* On this machine, the output is merged and then passed to the user-defined reduce function.
* Unlike the map output, reduce output is stored in HDFS (the first replica is stored on the local node and other replicas are stored on off-rack nodes). So, writing the reduce output.

## 2.6.3 WORK ORGANIZATION BY MAP REDUCE

Hadoop divides the job into tasks. There are two types of tasks:

1. Maptasks (Splits & Mapping)
2. Reducetasks (Shuffling, Reducing) as mentioned above.

The complete execution process (execution of Map and Reduce tasks, both) is controlled by two types of entities called a

1. Job tracker: Acts like a master (responsible for complete execution of submitted job)
2. MultipleTaskTrackers: Acts like slaves**,** each of them performing the job

For every job submitted for execution in the system, there is one Job tracker that resides on Namenode and there are multipletask trackers which reside on Datanode.

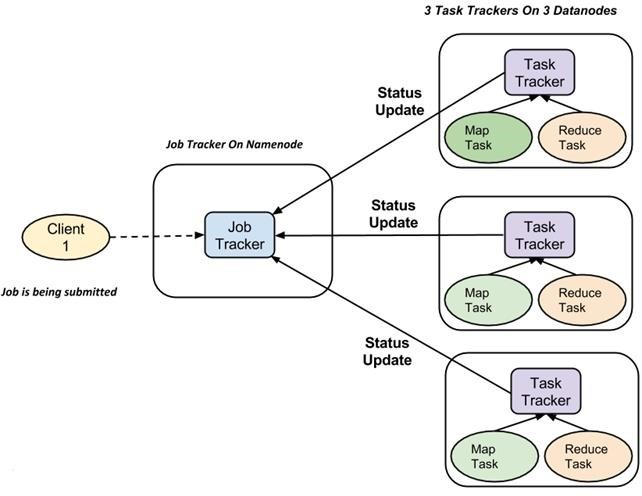


Fig. 2.6.3.1

* A job is divided into multiple tasks which are then run onto multiple data nodes in a cluster.
* It is the responsibility of job tracker to coordinate the activity by scheduling tasks to run on different data nodes.
* Execution of individual task is then to look after by task tracker, which resides on every data node executing part of the job.
* Task tracker's responsibility is to send the progress report to the job tracker.
* In addition, task tracker periodically sends 'heartbeat' signal to the Job tracker so as to notify him of the current state of the system.
* Thus, job tracker keeps track of the overall progress of each job. In the event of task failure, the job tracker can reschedule it on a different task tracker.

**CHAPTER 3**

**APACHE FLUME**

# 3.1 APACHE FLUME INTRODUCTION

Apache Flume is a system used for moving massive quantities of streaming data into HDFS. Collecting log data present in log files from web servers and aggregating it in HDFS for analysis, is one common example use case of Flume.

Flume supports multiple sources like –

* 'tail' (which pipes data from a local file and write into HDFS via Flume, similar to Unix command 'tail')
* System logs
* Apache log4j (enable Java applications to write events to files in HDFS via Flume).

## 3.1.1 FLUME ARCHITECTURE

A Flume agent is a JVM process which has 3 components -Flume Source, Flume Channel and Flume Sink- through which events propagate after initiated at an external source.

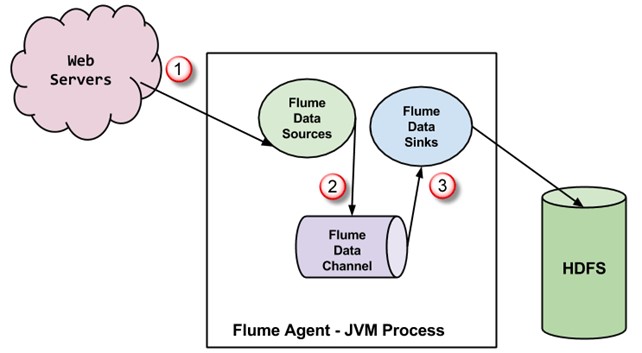


Fig. 3.1.1.1

1. In the above diagram, the events generated by external source (Webserver) are consumed by Flume Data Source. The external source sends events to Flume source in a format that is recognized by the target source.
2. Flume Source receives an event and stores it into one or more channels. The channel acts as a store which keeps the event until it is consumed by the flume sink. This channel may use a local file system in order to store these events.
3. Flume sink removes the event from a channel and stores it into an external repository like e.g., HDFS. There could be multiple flume agents, in which case flume sink forwards the event to the flume source of next flume agent in the flow.

## 3.1.2 FEATURES OF FLUME

* Flume has a flexible design based upon streaming data flows. It is fault tolerant and robust with multiple failovers and recovery mechanisms. Flume has different levels of reliability to offer which includes 'best-effort delivery' and an 'end-to-enddelivery'. Best-effort delivery does not tolerate any Flume node failure whereas 'end-to-end delivery' mode guarantees delivery even in the event of multiple node failures.
* Flume carries data between sources and sinks. This gathering of data can either be scheduled or event-driven. Flume has its own query processing engine which makes it easy to transform each new batch of data before it is moved to the intended sink.
* Possible Flume sinks include HDFS and HBase. Flume can also be used to transport event data including but not limited to network traffic data, data generated by social media websites and email messages.

## 3.1.3 LOADING TWITTER DATA USING FLUME

* Step 1) Go to the directory containing source code files in it.
* Step 2) Set CLASSPATH to contain <Flume Installation Dir>/lib/\* and ~/FlumeTutorial/flume/mytwittersource/\*

export CLASSPATH="/usr/local/apache-flume-1.4.0-bin/lib/\*:~/FlumeT

utorial/flume/mytwittersource/\*"

* Step 3) Compile source code using the command-

javac -d . MyTwitterSourceForFlume.java MyTwitterSource.java

* Step 4)Create a jar

First, create Manifest.txt file using a text editor of your choice and add below line in it-

Main-Class: flume.mytwittersource.MyTwitterSourceForFlume

.. here flume.mytwittersource.MyTwitterSourceForFlume is the name of the main class. Please note that you have to hit enter key at end of this line. Now, create JAR **'**MyTwitterSourceForFlume.jar'as-

jar cfm MyTwitterSourceForFlume.jar Manifest.txt flume/mytwitterso urce/\*.class

* Step 5) Copy this jar to <Flume Installation Directory>/lib/

sudo cp MyTwitterSourceForFlume.jar <Flume Installation Directory>

/lib/

* Step 6) Go to the configuration directory of Flume, <Flume Installation

Directory>/conf

If flume.conf does not exist, then copy flume-conf.properties.template and rename it to flume.conf

sudo cp flume-conf.properties.template flume.conf

If flume-env.sh does not exist, then copy flume-env.sh.template and rename it to flume-env.sh

sudo cp flume-env.sh.template flume-env.sh

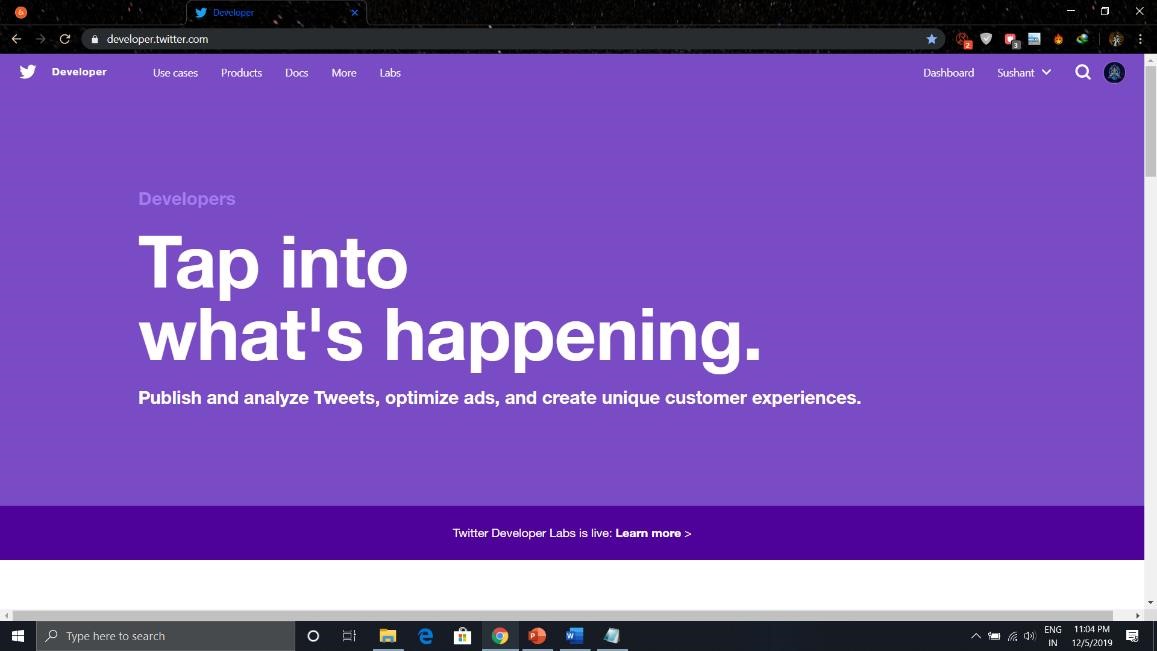
**CHAPTER 4**

**TWITTER API**

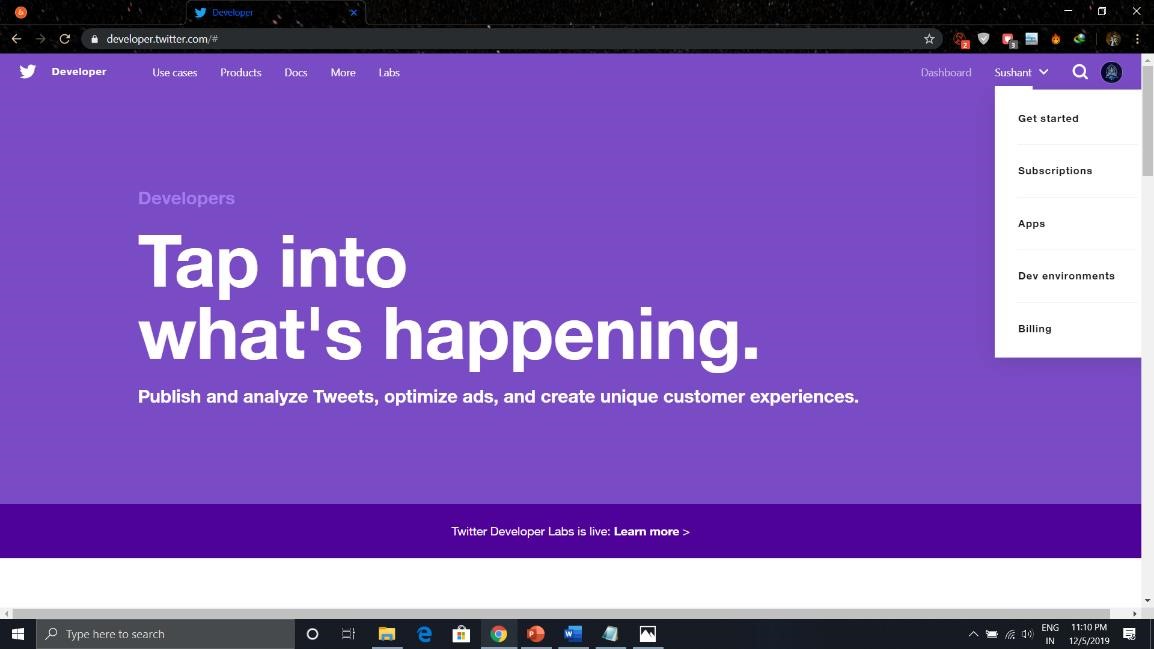
# 4.1 CREATING A TWITTER APPLICATION

* Step 1) Create a Twitter application by signing in to

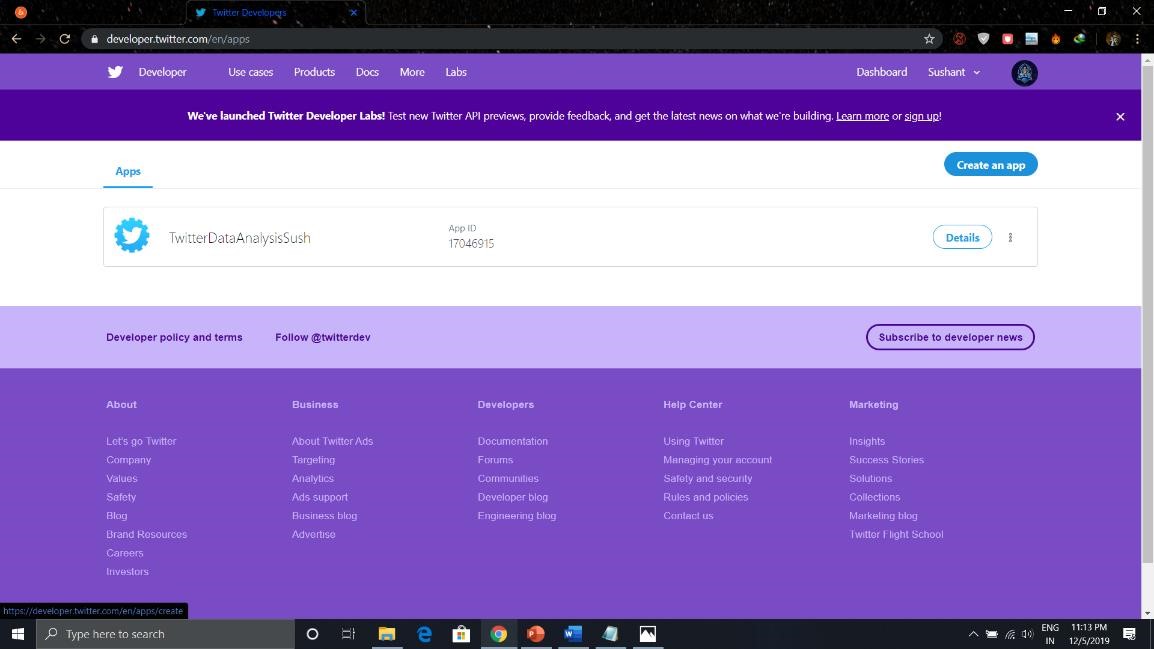
[www.developer.twitter.com](http://www.developer.twitter.com/)



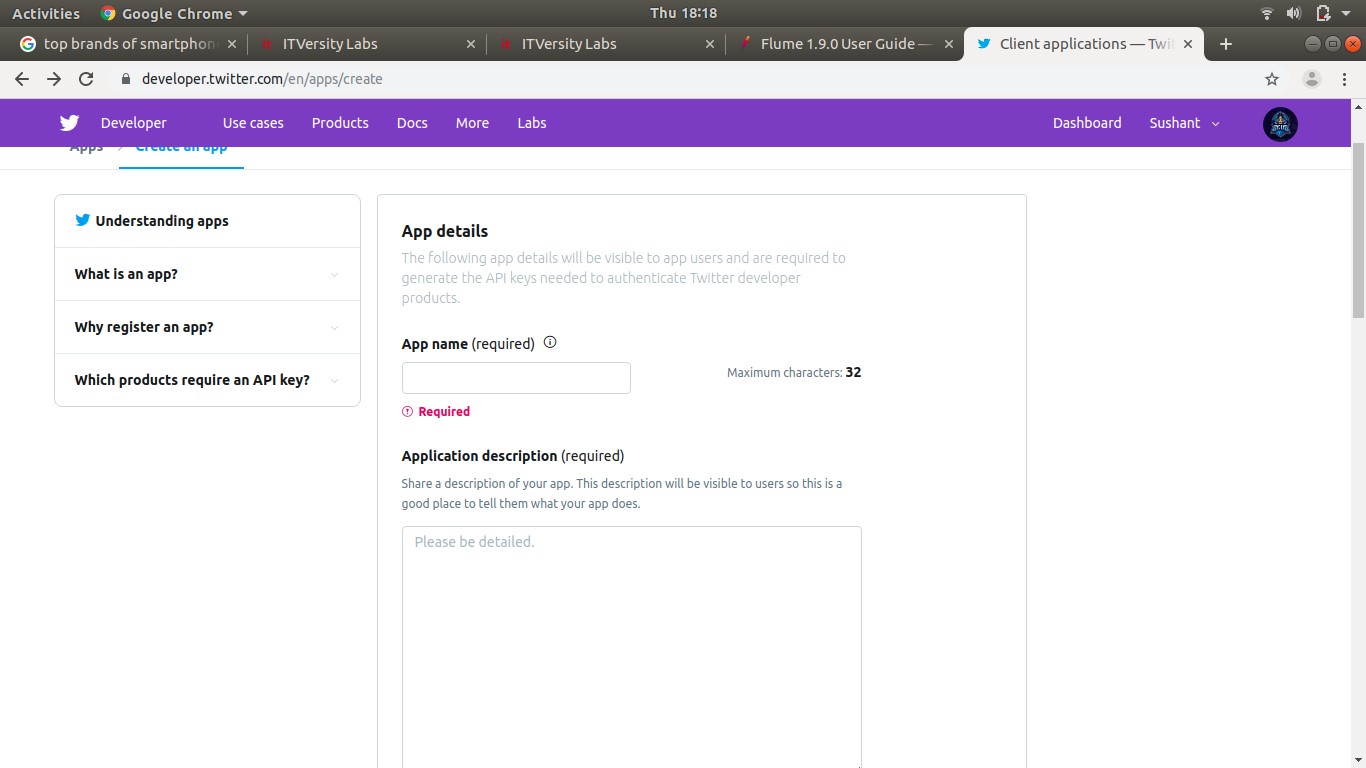
Step 2) Go to “apps” (This option gets dropped down when profile namebutton at the top right corner is clicked)



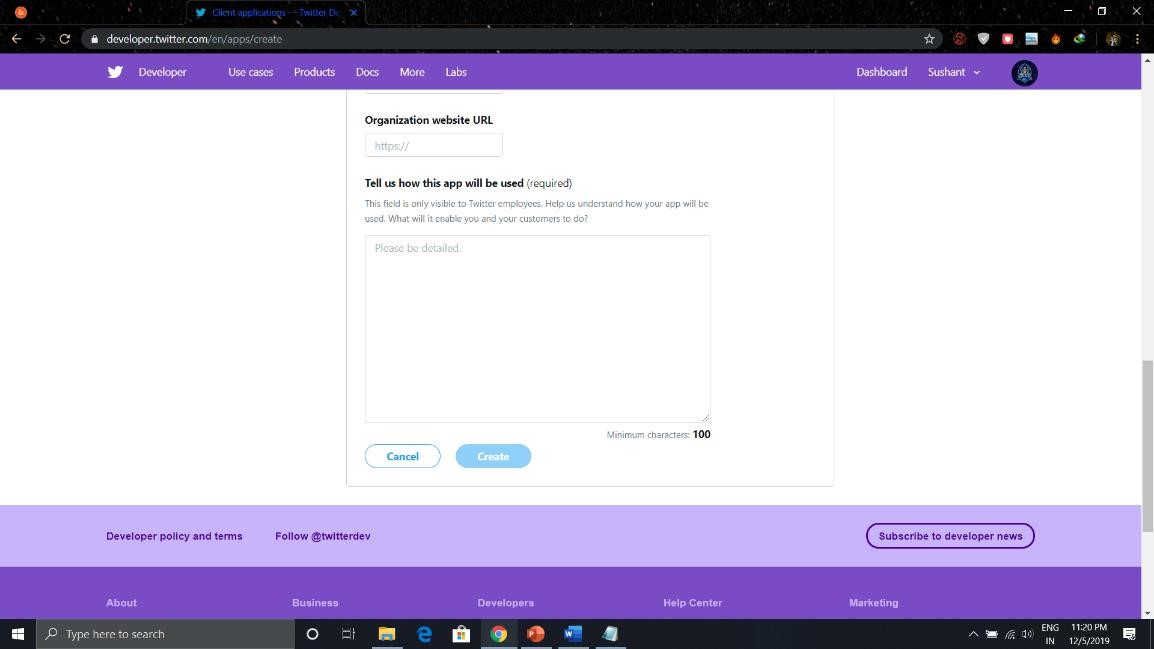
* Step 3) Create a new application by clicking 'Create New App'



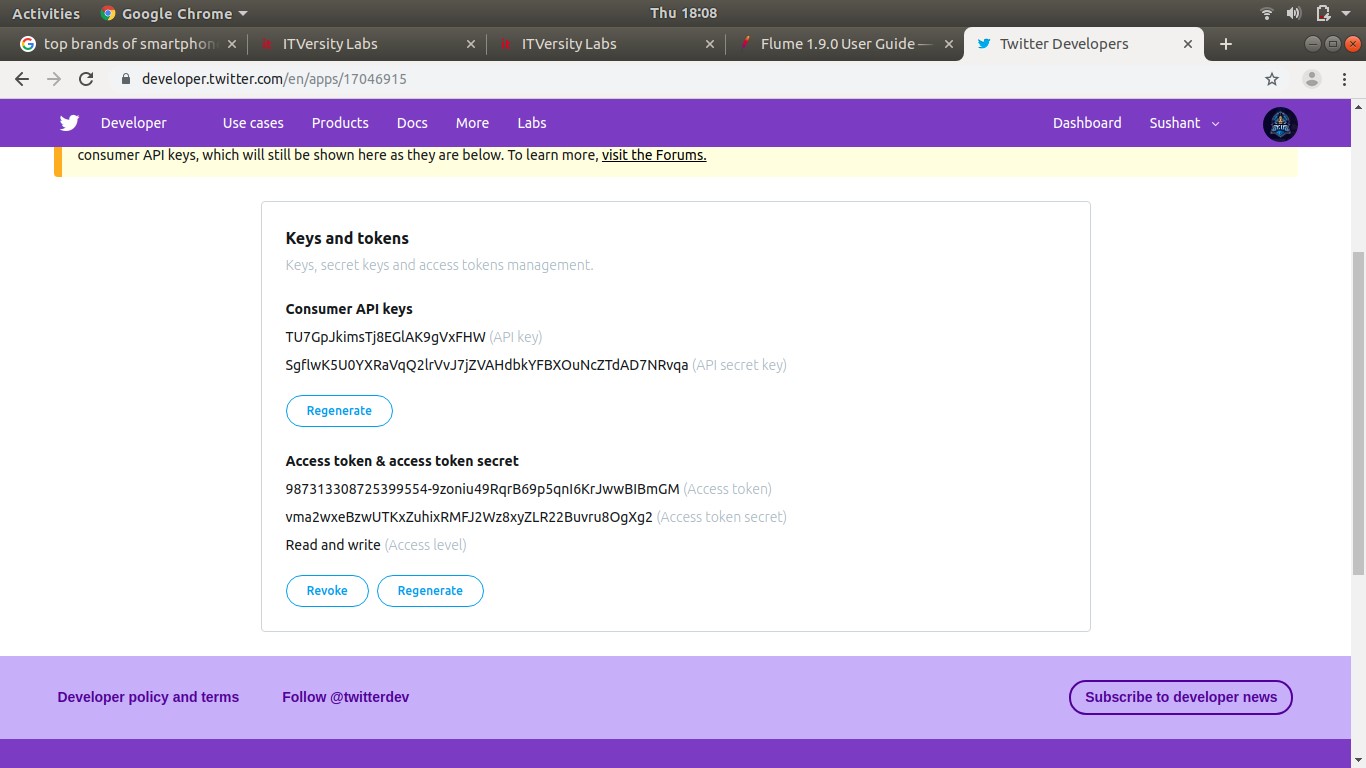
Step 4) Fill up application details by specifying the name of application, description, and website. You may refer to the notes given underneath each input box.



* Step 5) Scroll down the page and accept terms by marking 'Yes, I agree' and click on button 'Create'



Step 6) On the window of a newly created application, go to the tab, 'API Keys' scroll down the page and click button 'Create my access token'.



We need to copy Consumer key, Consumer secret, Access token and Access token secret to updating 'twitter\_4.conf'.

# 4.2 MODIFYING ‘ twitter\_4.conf ’ FILE

* Step 1) Open 'twitter\_4.conf'in write mode and set values for below parameters-

sudo gedit flume.conf

* Step 2)Copy the keys in the following code-

MyTwitAgent.sources = Twitter

MyTwitAgent.channels = MemChannel

MyTwitAgent.sinks = HDFS

MyTwitAgent.sources.Twitter.type = flume.mytwittersource.MyTwitterSourceF orFlume

MyTwitAgent.sources.Twitter.channels = MemChannel

MyTwitAgent.sources.Twitter.consumerKey = <Copy consumer key value from T witter App>

MyTwitAgent.sources.Twitter.consumerSecret = <Copy consumer secret value from Twitter App>

MyTwitAgent.sources.Twitter.accessToken = <Copy access token value from T witter App>

MyTwitAgent.sources.Twitter.accessTokenSecret = <Copy access token secret value from Twitter App>

MyTwitAgent.sources.Twitter.keywords = guru99

MyTwitAgent.sinks.HDFS.channel = MemChannel

MyTwitAgent.sinks.HDFS.type = hdfs

MyTwitAgent.sinks.HDFS.hdfs.path = hdfs://localhost:54310/user/hduser/flu me/tweets/

MyTwitAgent.sinks.HDFS.hdfs.fileType = DataStream

MyTwitAgent.sinks.HDFS.hdfs.writeFormat = Text

MyTwitAgent.sinks.HDFS.hdfs.batchSize = 1000

MyTwitAgent.sinks.HDFS.hdfs.rollSize = 0

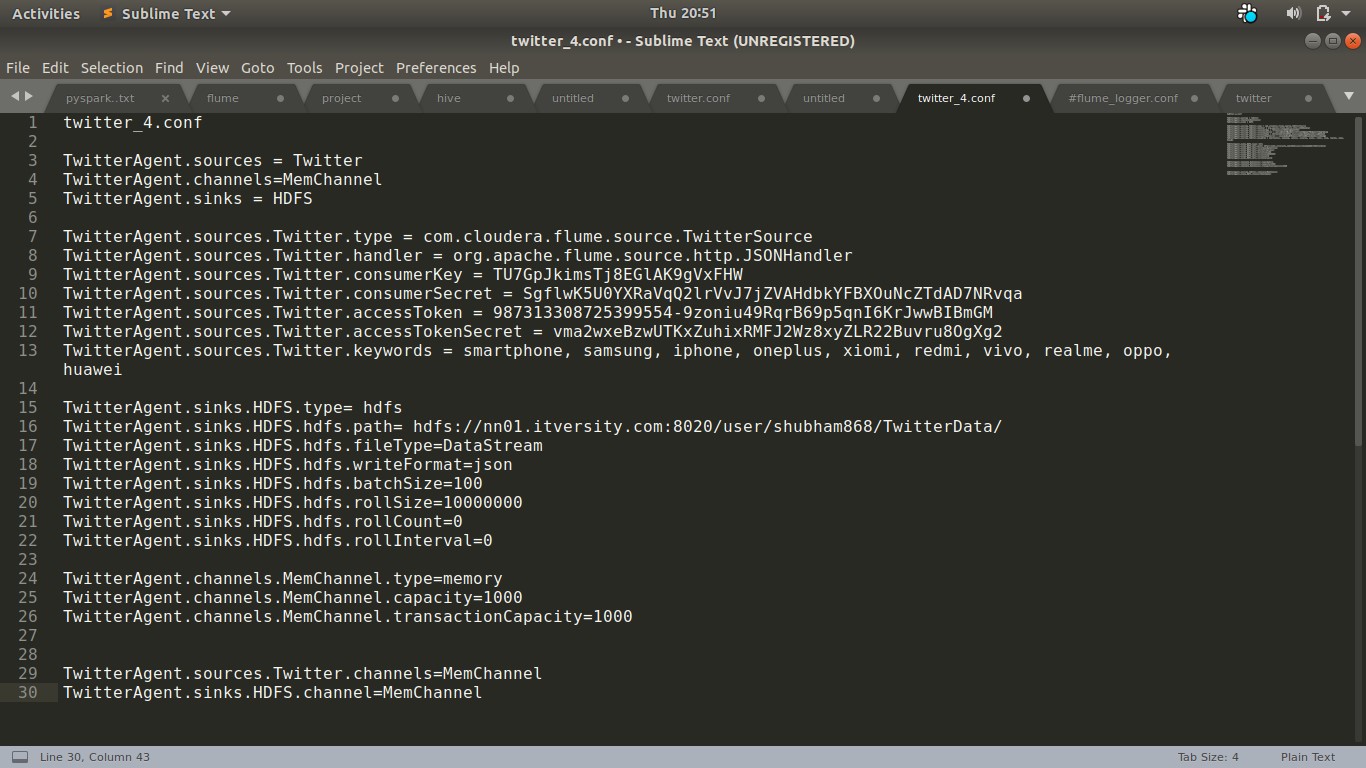
MyTwitAgent.sinks.HDFS.hdfs.rollCount = 10000

MyTwitAgent.channels.MemChannel.type = memory

MyTwitAgent.channels.MemChannel.capacity = 10000

MyTwitAgent.channels.MemChannel.transactionCapacity = 1000

* Step 3)The code after copying the keys looks like-



* Step 4) Also, set TwitterAgent.sinks.HDFS.hdfs.path as below,

TwitterAgent.sinks.HDFS.hdfs.path = hdfs://<Host Name>:<Port

Number>/<HDFS Home Directory>/flume/tweets/

To know <Host Name>**,** <Port Number> and <HDFS Home Directory> , see value of parameter 'fs.defaultFS' set in $HADOOP\_HOME/etc/hadoop/core-site.xml

* Step 5) In order to flush the data to HDFS, as an when it comes, delete below entry if it exists,

TwitterAgent.sinks.HDFS.hdfs.rollInterval = 600

**CHAPTER 5**

**BIG DATA TOOLS**

# 5.1 HIVE

Hive is developed on top of Hadoop. It is a data warehouse framework for querying and analysis of data that is stored in HDFS. Hive is an open source-software that lets programmers analyze large data sets on Hadoop.

The size of data sets being collected and analyzed in the industry for business intelligence is growing and, in a way, it is making traditional data warehousing solutions more expensive. Hadoop with MapReduce framework, is being used as an alternative solution for analyzing data sets with huge size. Though, Hadoop has proved useful for working on huge data sets, its MapReduce framework is very low level and it requires programmers to write custom programs which are hard to maintain and reuse. Hive comes here for rescue of programmers.

Hive provides SQL-like declarative language, called HiveQL, which is used for expressing queries. Using Hive-QL users associated with SQL are able to perform data analysis very easily.

Hive engine compiles these queries into Map-Reduce jobs to be executed on Hadoop. In addition, custom Map-Reduce scripts can also be plugged into queries. Hive operates on data stored in tables which consists of primitive data types and collection data types like arrays and maps.

**Features:**

* It Supports SQL like query language for interaction and Data modeling
* It compiles language with two main tasks map, and reducer
* It allows defining these tasks using Java or Python
* Hive designed for managing and querying only structured data
* Hive's SQL-inspired language separates the user from the complexity of Map Reduce programming
* It offers Java Database Connectivity (JDBC) interface

# 5.2 SPARK

Apache Spark is a powerful open source big data analytics tool. It offers over 80 high-level operators that make it easy to build parallel apps. It is used at a wide range of organizations to process large datasets.

**Features:**

* It helps to run an application in Hadoop cluster, up to 100 times faster in memory, and ten times faster on disk
* It offers lighting Fast Processing
* Support for Sophisticated Analytics
* Ability to Integrate with Hadoop and Existing Hadoop Data
* It provides built-in APIs in Java, Scala, or Python

# Reference

www.itversity.com www.developer.twitter.com www.dgadiraju.com www towardsdatascience.com