Mini-Project Report

Title : Data Analytics using Cloudera Hadoop

Dataset Used : Stocks.csv, Dividends.csv

Technologies Used : Hadoop, HDFS, Map-Reduce, Hive, Sqoop, Pig, Spark using Scala

Submitted to : Mr. Prashant Gangwar

Submitted by :

1. Gayatri Gupta
2. Harshit Khare
3. Mahua
4. Mayank Sharma

Objective:

The Mini-Project consists of 5 sub-projects which uses different technologies.

Part 1: Map-Reduce: Finding words present in different files

> *Objective:* to find which words are present in how many files along with the file names using Map-Reduce Framework.

Part 2: Sqoop Task: Loading Data from RDBMS to HDFS

> *Objective:* to load data from RDBMS (MySQL) to HDFS & then load data from HDFS to Hive tables using Apache Sqoop.

Part 3: Stocks Analysis using Hive

> *Objective:* to run Hive queries on the stocks dataset loaded using Sqoop to understand it better and then perform analytics on it.

Part 4: Pig Analytics

> *Objective*: to perform basic queries on 2 datasets: stocks and dividends and then joining them using Pig Latin to understand how Apache Pig Framework works.

Part 5: Twitter's Top 10 popular Hashtag Streaming per second using Apache Spark

> *Objective:* to find Top 10 popular Hashtag on Twitter and perform Web Scraping using Spark & Scala to stream the data on per second basis.

Contents:

Introduction to Big Data

Big data is data sets that are so big and complex that traditional data-processing application software are inadequate to deal with them. Big data challenges include capturing data, data storage, data analysis, search, sharing, transfer, visualization, querying, updating, information privacy and data source. There are a number of concepts associated with big data: originally there were 3 concepts volume, variety, velocity. Other concepts later attributed with big data are veracity (i.e., how much noise is in the data) and value.

Lately, the term "big data" tends to refer to the use of predictive analytics, user behavior analytics, or certain other advanced data analytics methods that extract value from data, and seldom to a particular size of data set. "There is little doubt that the quantities of data now available are indeed large, but that’s not the most relevant characteristic of this new data ecosystem." Analysis of data sets can find new correlations to "spot business trends, prevent diseases, combat crime and so on." Scientists, business executives, practitioners of medicine, advertising and governments alike regularly meet difficulties with large data-sets in areas including Internet search, fintech, urban informatics, and business informatics. Scientists encounter limitations in e-Science work, including meteorology, genomics, complex physics simulations, biology and environmental research.

Characteristics of Big Data

Big data can be described by the following characteristics:

- **Volume**

  The quantity of generated and stored data. The size of the data determines the value and potential insight, and whether it can be considered big data or not.

- **Variety**

  The type and nature of the data. This helps people who analyze it to effectively use the resulting insight. Big data draws from text, images, audio, video; plus, it completes missing pieces through data fusion.

- **Velocity**

In this context, the speed at which the data is generated and processed to meet the demands and challenges that lie in the path of growth and development. Big data is often available in real-time.

- **Veracity**

The data quality of captured data can vary greatly, affecting the accurate analysis.

Why is Big Data Important?

The importance of big data doesn't revolve around how much data you have, but what you do with it. You can take data from any source and analyze it to find answers that enable

1) cost reductions
2) time reductions
3) new product development and optimized offerings
4) smart decision making.

When you combine big data with high-powered analytics, you can accomplish business-related tasks such as:

- Determining root causes of failures, issues and defects in near-real time.
- Generating coupons at the point of sale based on the customer's buying habits.
- Recalculating entire risk portfolios in minutes.
- Detecting fraudulent behavior before it affects your organization.

Benefits of Big Data & Data Analytics

Big data makes it possible for you to gain more complete answers because you have more information.

More complete answers mean more confidence in the data—which means a completely different approach to tackling problems.

Big Data Challenges

While big data holds a lot of promise, it is not without its challenges.

- First, big data is…big. Although new technologies have been developed for data storage, data volumes are doubling in size about every two years. Organizations still struggle to keep pace with their data and find ways to effectively store it.

- But it's not enough to just store the data. Data must be used to be valuable and that depends on curation. Clean data, or data that's relevant to the client and organized in a way that enables meaningful analysis, requires a lot of work. Data scientists spend 50 to 80 percent of their time curating and preparing data before it can actually be used.

- Finally, big data technology is changing at a rapid pace. A few years ago, Apache Hadoop was the popular technology used to handle big data. Then Apache Spark was introduced in 2014. Today, a combination of the two frameworks appears to be the best approach. Keeping up with big data technology is an ongoing challenge

How Big Data works

Big data gives you new insights that open up new opportunities and business models. Getting started involves three key actions:

- **Integrate**

  Big data brings together data from many disparate sources and applications. Traditional data integration mechanisms, such as ETL (extract, transform, and load) generally aren't up to the task. It requires new strategies and technologies to analyze big data sets at terabyte, or even petabyte, scale.
  During integration, you need to bring in the data, process it, and make sure it's formatted and available in a form that your business analysts can get started with.

- **Manage**

  Big data requires storage. Your storage solution can be in the cloud, on premises, or both. You can store your data in any form you want and bring your desired processing requirements and necessary process engines to those data sets on an on-demand basis. Many people choose their storage solution according to where their data is currently residing. The cloud is gradually gaining popularity because it supports your current compute requirements and enables you to spin up resources as needed.

- **Analyze**

  Your investment in big data pays off when you analyze and act on your data. Get new clarity with a visual analysis of your varied data sets. Explore the data further to make new discoveries. Share your findings with others. Build data models with machine learning and artificial intelligence. Put your data to work.

Introduction to Hadoop

Hadoop is an Apache open source framework written in java that allows distributed processing of large datasets across clusters of computers using simple programming models. A Hadoop frame-worked application works in an environment that provides distributed storage and computation across clusters of computers. Hadoop is designed to scale up from single server to thousands of machines, each offering local computation and storage.

Hadoop Architecture

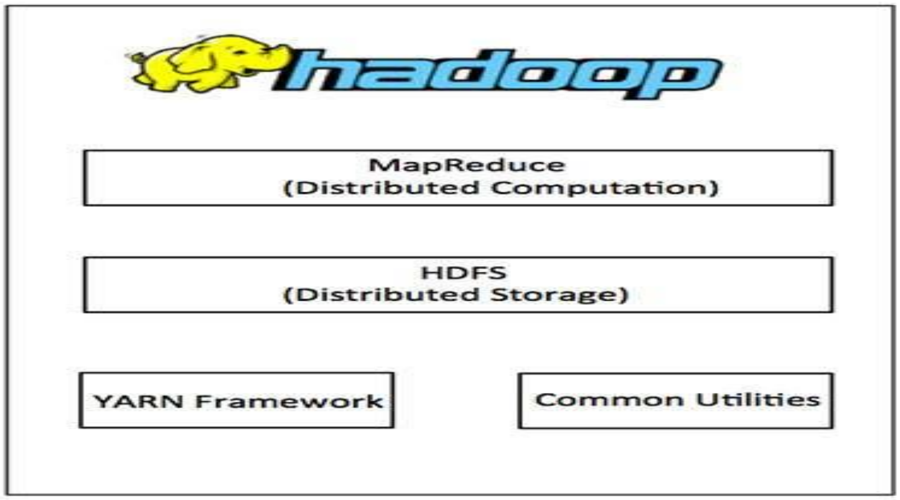
Hadoop framework includes following four modules:

- **Hadoop Common:** These are Java libraries and utilities required by other Hadoop modules. These libraries provide filesystem and OS level abstractions and contains the necessary Java files and scripts required to start Hadoop.

- **Hadoop YARN:** This is a framework for job scheduling and cluster resource management.

- **Hadoop Distributed File System (HDFS):** A distributed file system that provides high-throughput access to application data.

- **Hadoop MapReduce:** This is YARN-based system for parallel processing of large data sets.



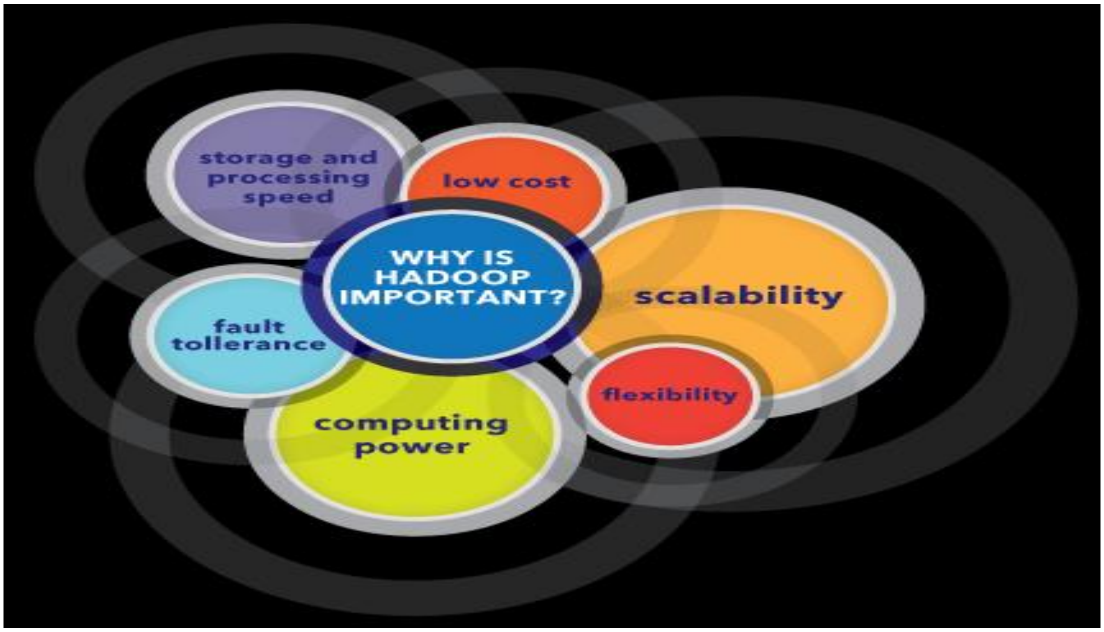
Advantages of Hadoop

- Hadoop framework allows the user to quickly write and test distributed systems. It is efficient, and it automatic distributes the data and work across the machines and in turn, utilizes the underlying parallelism of the CPU cores.

- Hadoop does not rely on hardware to provide fault-tolerance and high availability (FTHA), rather Hadoop library itself has been designed to detect and handle failures at the application layer.

- Servers can be added or removed from the cluster dynamically and Hadoop continues to operate without interruption.
  Another big advantage of Hadoop is that apart from being open source, it is compatible on all the platforms since it is Java based.

Why is Hadoop important?



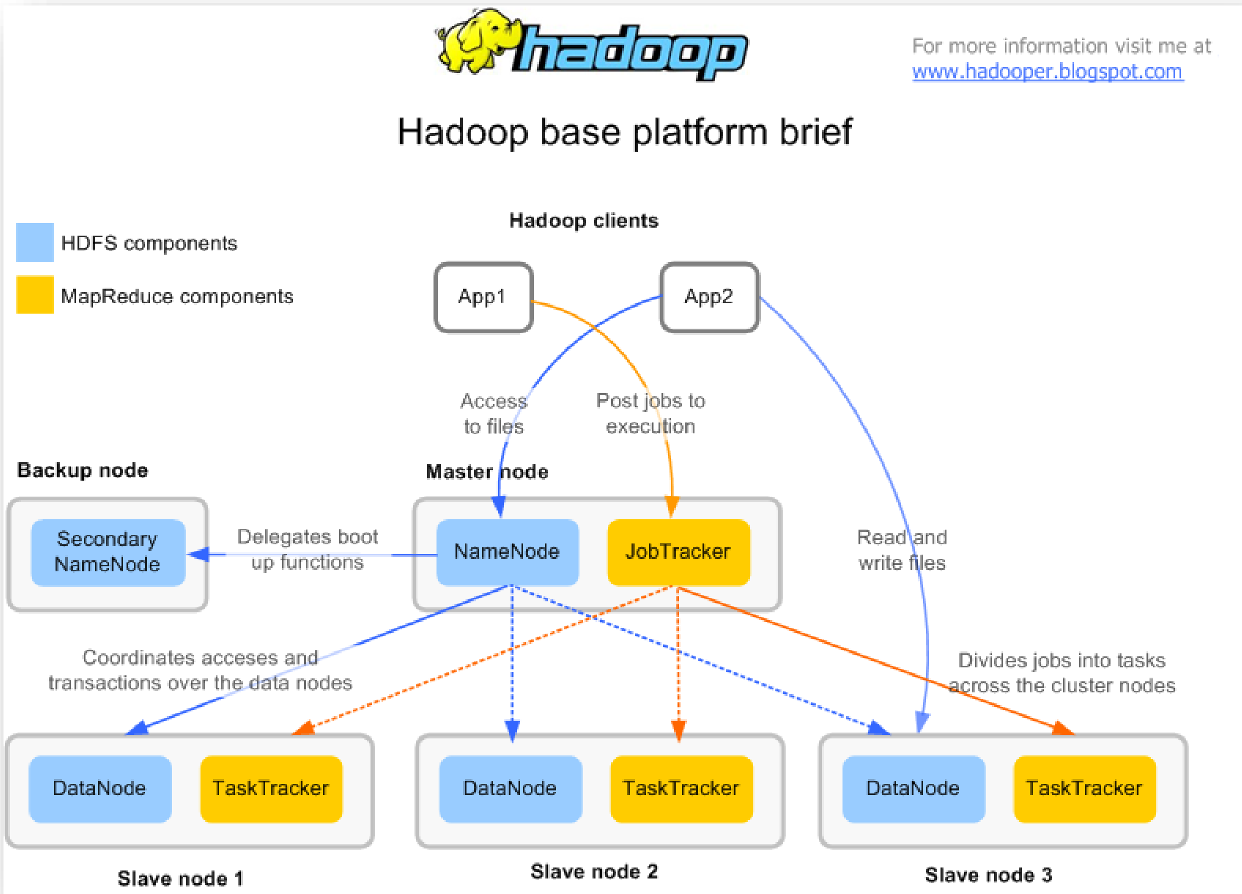
Hadoop Distributed File System

Hadoop can work directly with any mountable distributed file system such as Local FS, HFTP FS, S3 FS, and others, but the most common file system used by Hadoop is the Hadoop Distributed File System (HDFS).

The Hadoop Distributed File System (HDFS) is based on the Google File System (GFS) and provides a distributed file system that is designed to run on large clusters (thousands of computers) of small computer machines in a reliable, fault-tolerant manner.

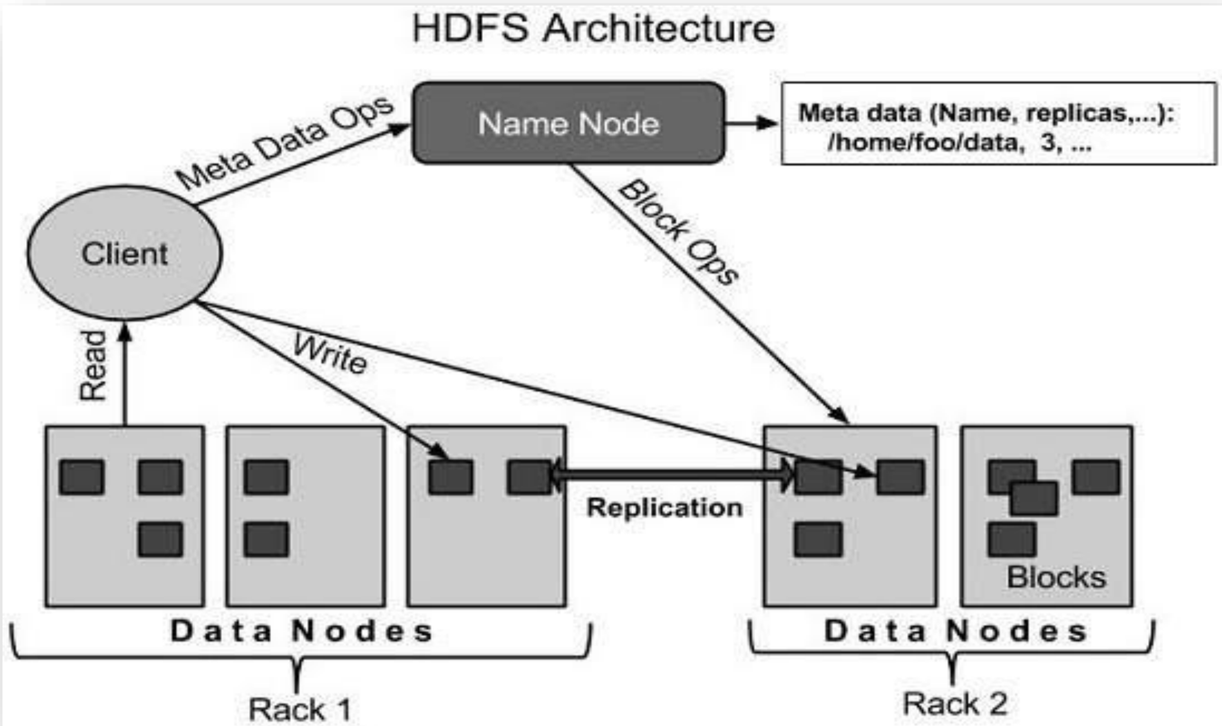
HDFS uses a master/slave architecture where master consists of a single NameNode that manages the file system metadata and one or more slave DataNodes that store the actual data.

A file in an HDFS namespace is split into several blocks and those blocks are stored in a set of DataNodes. The NameNode determines the mapping of blocks to the DataNodes. The DataNodes takes care of read and write operation with the file system. They also take care of block creation, deletion and replication based on instruction given by NameNode. HDFS provides a shell like any other file system and a list of commands are available to interact with the file system.



HDFS Architecture

HDFS follows the master-slave architecture and it has the following elements.



- **Namenode**

The namenode is the commodity hardware that contains the GNU/Linux operating system and the namenode software. It is a software that can be run on commodity hardware. The system having the namenode acts as the master server and it does the following tasks:

- Manages the file system namespace.
- Regulates client's access to files.
- It also executes file system operations such as renaming, closing, and opening files and directories.

- **Datanode**

The datanode is a commodity hardware having the GNU/Linux operating system and datanode software. For every node (Commodity hardware/System) in a cluster, there will be a datanode. These nodes manage the data storage of their system. Datanodes perform read-write operations on the file systems, as per client request.

They also perform operations such as block creation, deletion, and replication according to the instructions of the namenode.

- **Block**

Generally, the user data is stored in the files of HDFS. The file in a file system will be divided into one or more segments and/or stored in individual data nodes. These file segments are called as blocks. In other words, the minimum amount of data that HDFS can read or write is called a Block. The default block size is 64MB, but it can be increased as per the need to change in HDFS configuration.

How Does Hadoop Work?

- **Stage 1**

A user/application can submit a job to the Hadoop (a Hadoop job client) for required process by specifying the following items:

- The location of the input and output files in the distributed file system.
- The java classes in the form of jar file containing the implementation of map and reduce functions.
- The job configuration by setting different parameters specific to the job.

- **Stage 2**

The Hadoop job client then submits the job (jar/executable etc.) and configuration to the JobTracker which then assumes the responsibility of distributing the software/configuration to the slaves, scheduling tasks and monitoring them, providing status and diagnostic information to the job-client.

- **Stage 3**

The TaskTrackers on different nodes execute the task as per MapReduce implementation and output of the reduce function is stored into the output files on the file system.

Map-Reduce

MapReduce is a framework using which we can write applications to process huge amounts of data, in parallel, on large clusters of commodity hardware in a reliable manner.

MapReduce is a processing technique and a program model for distributed computing based on java. The MapReduce algorithm contains two important tasks, namely Map and Reduce. Map takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key/value pairs). Secondly, reduce task, which takes the output from a map as an input and combines those data tuples into a smaller set of tuples. As the sequence of the name MapReduce implies, the reduce task is always performed after the map job.

The major advantage of MapReduce is that it is easy to scale data processing over multiple computing nodes. Under the MapReduce model, the data processing primitives are called mappers and reducers. Decomposing a data processing application into mappers and reducers is sometimes nontrivial. But, once we write an application in the MapReduce form, scaling the application to run over hundreds, thousands, or even tens of thousands of machines in a cluster is merely a configuration change.

Map-Reduce: Finding words present in different files

mapper.py

```
#!/usr/bin/env python

from os import listdir
from os.path import isfile, join

import sys

for mypath in sys.stdin:
    mypath = mypath.rstrip()
    onlyFiles = [join(mypath, f) for f in listdir(mypath) if isfile(join(mypath, f))]
    fileNames = [f for f in listdir(mypath) if isfile(join(mypath, f))]
# print(onlyFiles)
    i = 0
    for file in onlyFiles:
    # print(file)
    # fileName = fileSplit[-1]
        f = open(file, 'r')

        lines = f.readlines()


        for line in lines:
            for word in line.split():
                print fileNames[i] +" "+ word
        i += 1
```

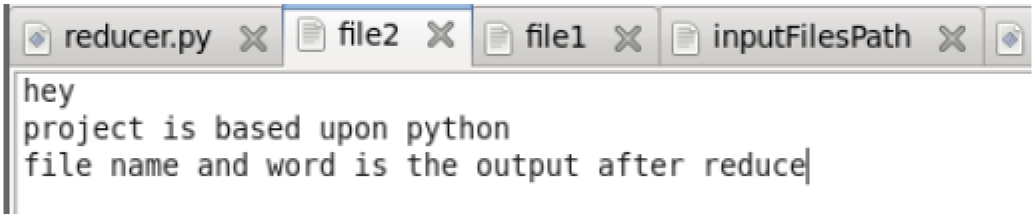
reducer.py

```python
#!/usr/bin/env python

import sys

def removeDuplicates(words):
    removed = []
    for word in words:
        if word not in removed:
            removed.append(word)

    return removed
wordKeyFileValue = {}
final = []
fileAndWord = {}
words = []
for line in sys.stdin:

    line = line.strip()

    file, word = line.split()
    words.append(word)

    if file not in fileAndWord:
        fileAndWord[file] = []
        fileAndWord[file].append(word)
    else:
        if word not in fileAndWord[file]:
            fileAndWord[file].append(word)

    removed = removeDuplicates(words)

    # print(removed)

    # print(fileAndWord)

    #
    for word in removed:
        for key in fileAndWord.keys():
            if word in fileAndWord[key]:
                final.append(key)
        wordKeyFileValue[word] = final
        final = []
    # print(wordKeyFileValue)
    words = sorted(wordKeyFileValue.keys())
    values = wordKeyFileValue.values()
for word, values in zip(words, values):
    print ""
    print word,
    for value in values:
        print value,
    print ""
    # print(fileAndWord)
```

file1.txt

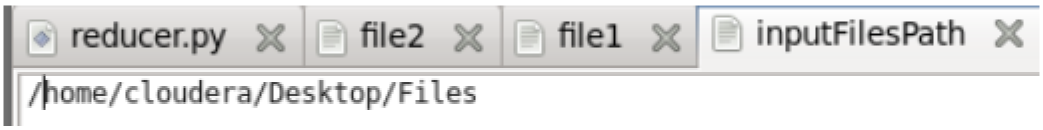
```
: is based on map reduce
has two methods - map and reduce
```

file2.txt



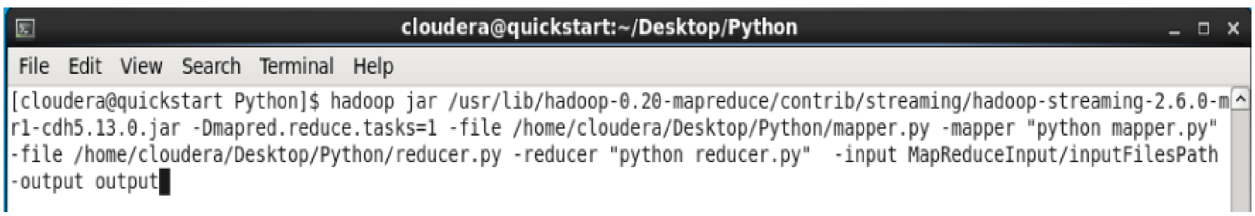

```
hey
project is based upon python
file name and word is the output after reduce
```

inputFilesPath



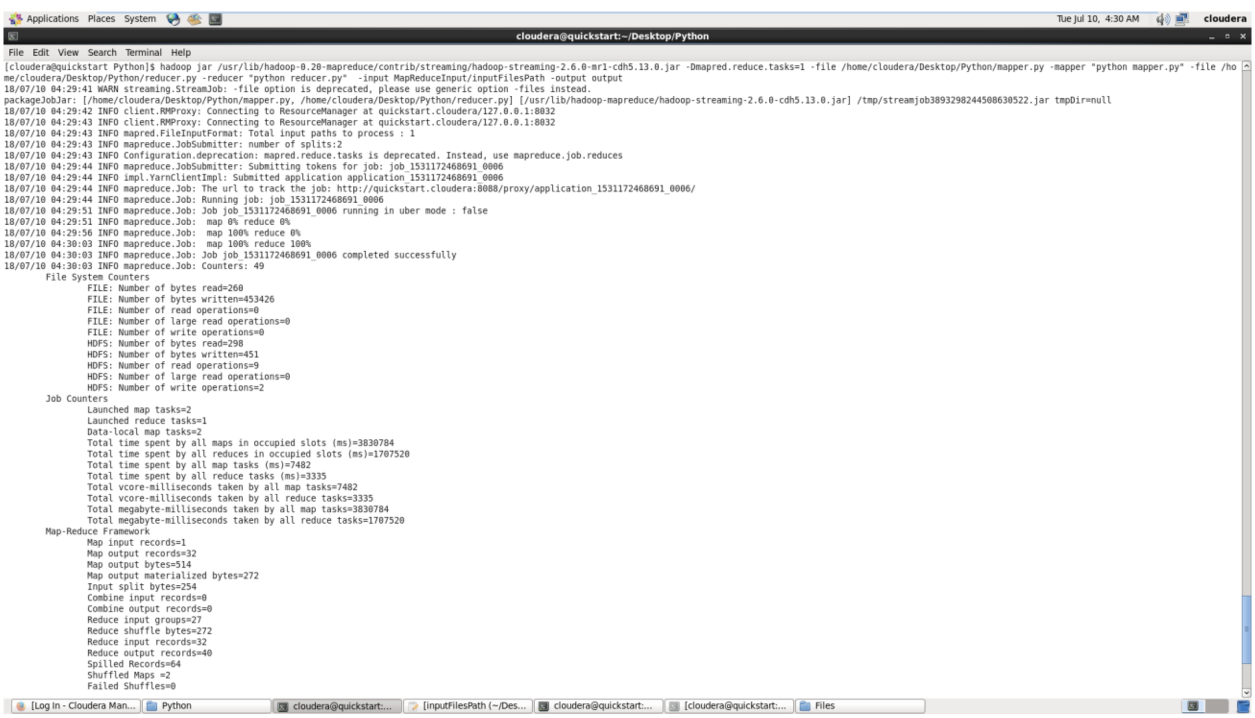

```
/home/cloudera/Desktop/Files
```

Command:




```
[cloudera@quickstart Python]$ hadoop jar /usr/lib/hadoop-0.20-mapreduce/contrib/streaming/hadoop-streaming-2.6.0-mr1-cdh5.13.0.jar -Dmapred.reduce.tasks=1 -file /home/cloudera/Desktop/Python/mapper.py -mapper "python mapper.py" -file /home/cloudera/Desktop/Python/reducer.py -reducer "python reducer.py"  -input MapReduceInput/inputFilesPath -output output
```

Command Running:




```
[cloudera@quickstart Python]$ hadoop jar /usr/lib/hadoop-0.20-mapreduce/contrib/streaming/hadoop-streaming-2.6.0-mr1-cdh5.13.0.jar -Dmapred.reduce.tasks=1 -file /home/cloudera/Desktop/Python/mapper.py -mapper "python mapper.py" -file /home/cloudera/Desktop/Python/reducer.py -reducer "python reducer.py"  -input MapReduceInput/inputFilesPath -output output
18/07/10 04:29:41 WARN streaming.StreamJob: -file option is deprecated, please use generic option -files instead.
packageJobJar: [/home/cloudera/Desktop/Python/mapper.py, /home/cloudera/Desktop/Python/reducer.py] [/usr/lib/hadoop-mapreduce/hadoop-streaming-2.6.0-cdh5.13.0.jar] /tmp/streamjob3893298244508630522.jar tmpDir=null
18/07/10 04:29:42 INFO client.RMProxy: Connecting to ResourceManager at quickstart.cloudera/127.0.0.1:8032
18/07/10 04:29:43 INFO client.RMProxy: Connecting to ResourceManager at quickstart.cloudera/127.0.0.1:8032
18/07/10 04:29:43 INFO mapred.FileInputFormat: Total input paths to process : 1
18/07/10 04:29:43 INFO mapreduce.JobSubmitter: number of splits:2
18/07/10 04:29:43 INFO Configuration.deprecation: mapred.reduce.tasks is deprecated. Instead, use mapreduce.job.reduces
18/07/10 04:29:44 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1531172468691_0006
18/07/10 04:29:44 INFO impl.YarnClientImpl: Submitted application application_1531172468691_0006
18/07/10 04:29:44 INFO mapreduce.Job: The url to track the job: http://quickstart.cloudera:8088/proxy/application_1531172468691_0006/
18/07/10 04:29:44 INFO mapreduce.Job: Running job: job_1531172468691_0006
18/07/10 04:29:51 INFO mapreduce.Job: Job job_1531172468691_0006 running in uber mode : false
18/07/10 04:29:51 INFO mapreduce.Job:  map 0% reduce 0%
18/07/10 04:29:56 INFO mapreduce.Job:  map 100% reduce 0%
18/07/10 04:30:03 INFO mapreduce.Job:  map 100% reduce 100%
18/07/10 04:30:03 INFO mapreduce.Job: Job job_1531172468691_0006 completed successfully
18/07/10 04:30:03 INFO mapreduce.Job: Counters: 49
        File System Counters
                FILE: Number of bytes read=260
                FILE: Number of bytes written=453426
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=298
                HDFS: Number of bytes written=451
                HDFS: Number of read operations=9
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=2
        Job Counters
                Launched map tasks=2
                Launched reduce tasks=1
                Data-local map tasks=2
                Total time spent by all maps in occupied slots (ms)=3830784
                Total time spent by all reduces in occupied slots (ms)=1707520
                Total time spent by all map tasks (ms)=7482
                Total time spent by all reduce tasks (ms)=3335
                Total vcore-milliseconds taken by all map tasks=7482
                Total vcore-milliseconds taken by all reduce tasks=3335
                Total megabyte-milliseconds taken by all map tasks=3830784
                Total megabyte-milliseconds taken by all reduce tasks=1707520
        Map-Reduce Framework
                Map input records=1
                Map output records=32
                Map output bytes=514
                Map output materialized bytes=272
                Input split bytes=254
                Combine input records=0
                Combine output records=0
                Reduce input groups=27
                Reduce shuffle bytes=272
                Reduce input records=32
                Reduce output records=40
                Spilled Records=64
                Shuffled Maps =2
                Failed Shuffles=0
```

Output:

```
[cloudera@quickstart Python]$ hadoop fs -cat output/*0000
- file1.txt file2.txt
after file1.txt
and file1.txt file2.txt
based file1.txt
file file1.txt
has file2.txt
hey file1.txt file2.txt
is file1.txt file2.txt
map file2.txt
methods file1.txt
name file1.txt file2.txt
on file1.txt
output file1.txt file2.txt
project file2.txt
python file2.txt
reduce file1.txt file2.txt
the file1.txt
two file2.txt
upon file2.txt
word file2.txt
[cloudera@quickstart Python]$
```

Introduction to Sqoop

Apache Sqoop is a tool in Hadoop ecosystem which is designed to transfer data between HDFS (Hadoop storage) and relational database servers like MySQL, Oracle RDB, SQLite, Teradata, Netezza, Postgres etc. Apache Sqoop imports data from relational databases to HDFS, and exports data from HDFS to relational databases. It efficiently transfers bulk data between Hadoop and external datastores such as enterprise data warehouses, relational databases, etc.

This is how Sqoop got its name – “SQL to Hadoop & Hadoop to SQL”.

Additionally, Sqoop is used to import data from external datastores into Hadoop ecosystem’s tools like Hive & HBase.

Why Sqoop?

For Hadoop developer, the actual game starts after the data is being loaded in HDFS. They play around this data in order to gain various insights hidden in the data stored in HDFS.

So, for this analysis the data residing in the relational database management systems need to be transferred to HDFS. The task of writing MapReduce code for importing and exporting data from relational database to HDFS is uninteresting & tedious. This is where Apache Sqoop comes to rescue and removes their pain. It automates the process of importing & exporting the data.

Sqoop makes the life of developers easy by providing CLI for importing and exporting data. They just have to provide basic information like database authentication, source, destination, operations etc. It takes care of remaining part.

Sqoop internally converts the command into MapReduce tasks, which are then executed over HDFS. It uses YARN framework to import and export the data, which provides fault tolerance on top of parallelism.

Key Features of Sqoop

Sqoop provides many salient features like:

- **Full Load**: Apache Sqoop can load the whole table by a single command. You can also load all the tables from a database using a single command.

- **Incremental Load:** Apache Sqoop also provides the facility of incremental load where you can load parts of table whenever it is updated.

- **Parallel import/export:** Sqoop uses YARN framework to import and export the data, which provides fault tolerance on top of parallelism.

- **Import results of SQL query**: You can also import the result returned from an SQL query in HDFS.

- **Compression:** You can compress your data by using deflate(gzip) algorithm with –compress argument, or by specifying –compression-codec argument. You can also load compressed table in Apache Hive.

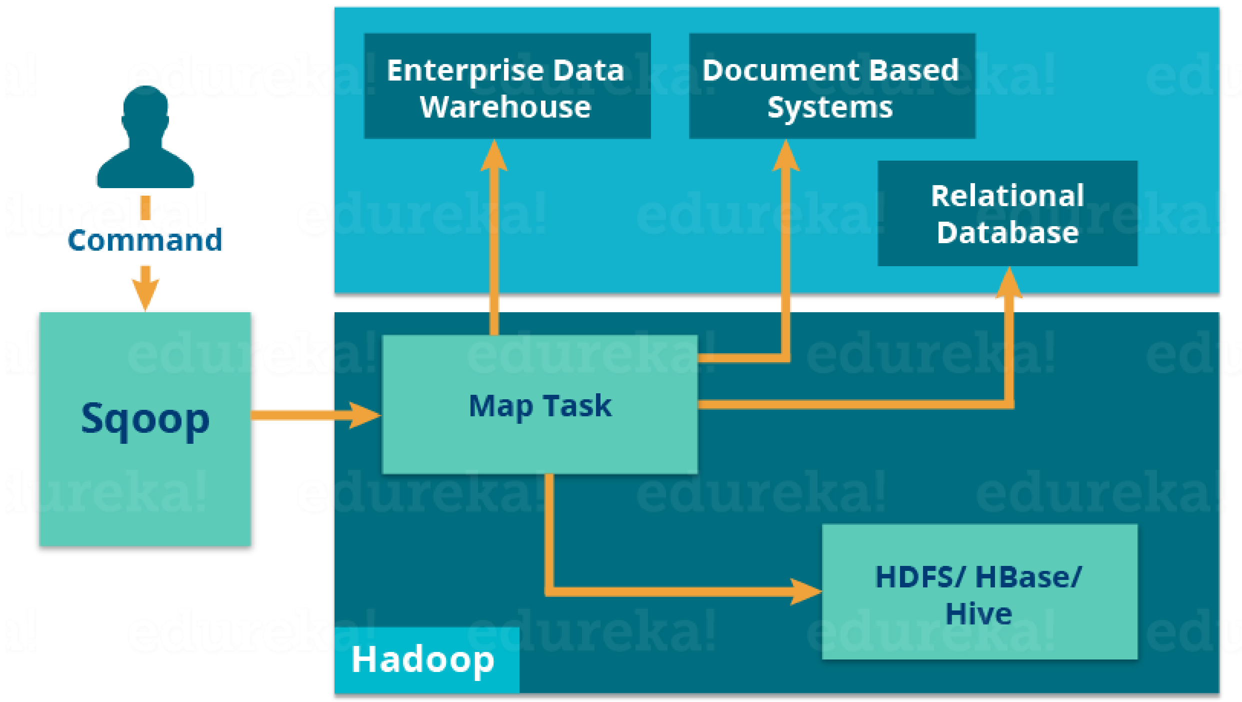
- **Connectors for all major RDBMS Databases:** Apache Sqoop provides connectors for multiple RDBMS databases, covering almost the entire circumference.

- **Kerberos Security Integration:** Kerberos is a computer network authentication protocol which works on the basis of 'tickets' to allow nodes communicating over a non-secure network to prove their identity to one another in a secure manner. Sqoop supports Kerberos authentication.

- **Load data directly into HIVE/HBase:** You can load data directly into Apache Hive for analysis and also dump your data in HBase, which is a NoSQL database.

- **Support for Accumulo:** You can also instruct Sqoop to import the table in Accumulo rather than a directory in HDFS.

Sqoop Architecture



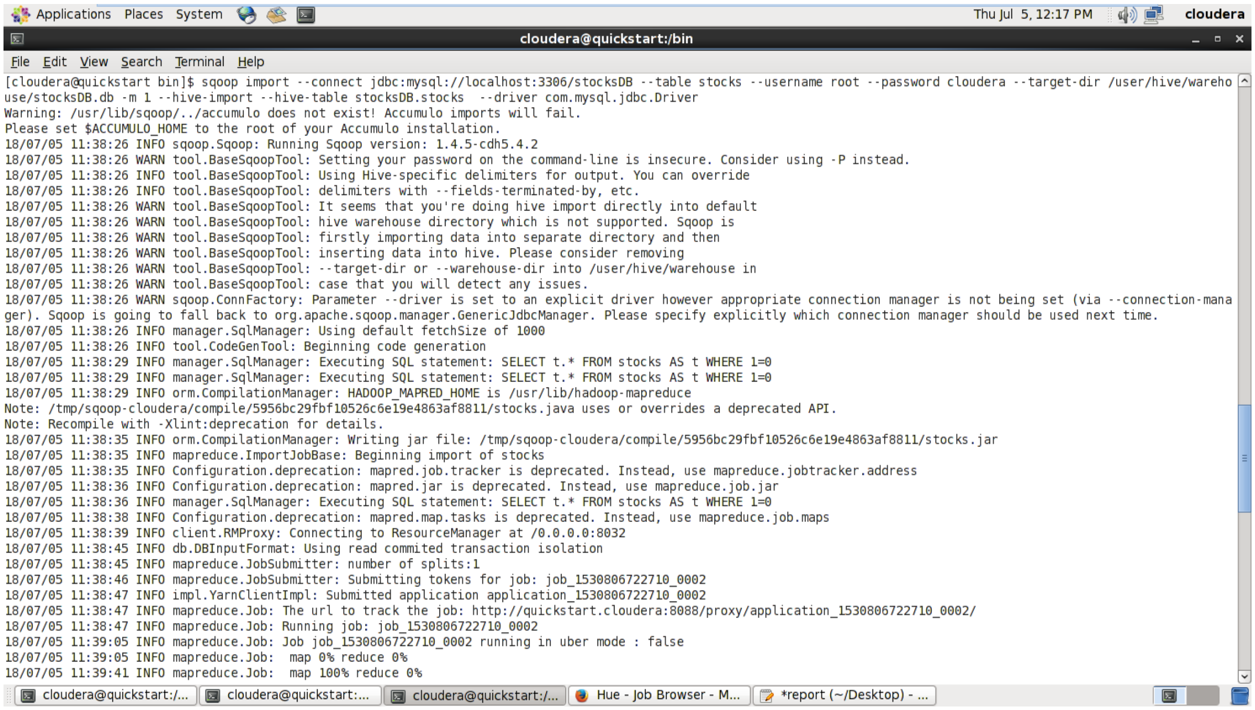
Sqoop Task: Loading Data from RDBMS to HDFS

Dataset used : stocks.csv

Task – to load data from RDBMS (MySQL) to HDFS & then load data from HDFS to Hive tables

Step 1 – Loading data from RDBMS (MySQL) to HDFS

Command :
```
sqoop import --connect jdbc:mysql://localhost:3306/stocksDB
--table stocks --username root --password cloudera --target-dir
/user/hive/warehouse/stocksDB.db -m 1 --hive-import --hive-table
stocksDB.stocks  --driver com.mysql.jdbc.Driver
```



What is 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 :

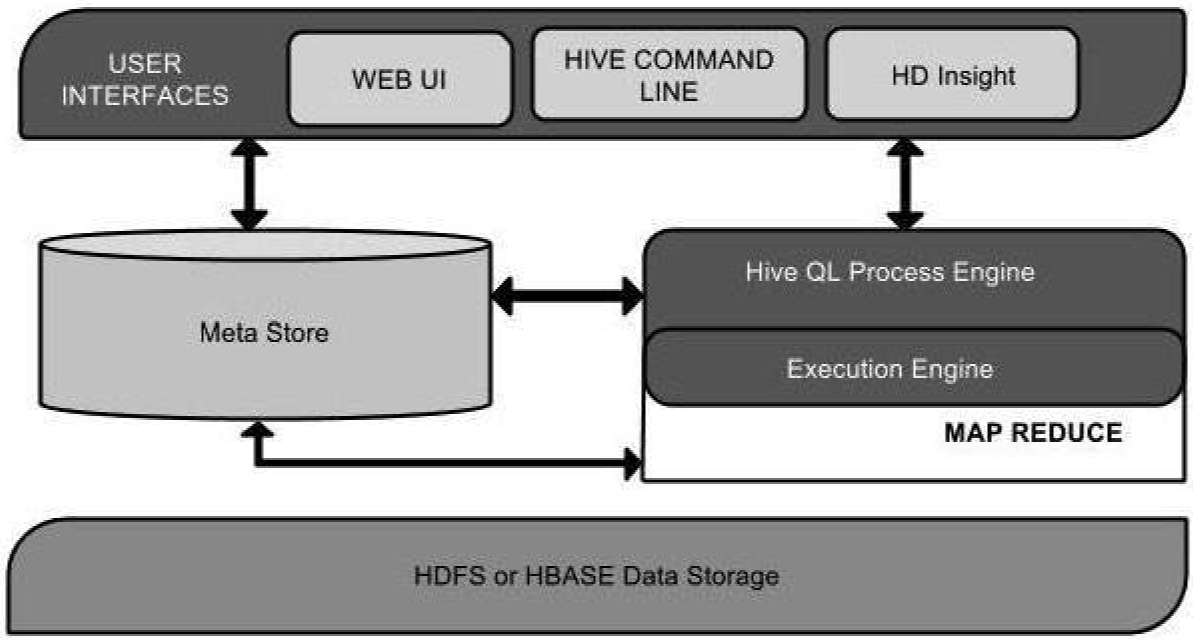
- A relational database
- A design for OnLine Transaction Processing (OLTP)
- A language for real-time queries and row-level updates

Features of Hive

- It stores schema in a database and processed 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:



Stocks Analysis using Hive

Dataset used : Stocks.csv

Step 1 - Opening Hive2 or Thrift Server & make a connection with it.

```
[cloudera@quickstart Desktop]$ cd $HIVE_HOME/bin
[cloudera@quickstart bin]$ beeline
Beeline version 1.1.0-cdh5.13.0 by Apache Hive
beeline> !connect jdbc:hive2://localhost:10000/default
scan complete in 2ms
Connecting to jdbc:hive2://localhost:10000/default
Enter username for jdbc:hive2://localhost:10000/default: cloudera
Enter password for jdbc:hive2://localhost:10000/default: ********
Connected to: Apache Hive (version 1.1.0-cdh5.13.0)
Driver: Hive JDBC (version 1.1.0-cdh5.13.0)
Transaction isolation: TRANSACTION_REPEATABLE_READ
0: jdbc:hive2://localhost:10000/default>
```

Step 2 - Creating & using Database stocks_DB

Command - `create database stocks_DB;`

```
use stocks_DB;
```

```
0: jdbc:hive2://localhost:10000/default> create database stocks_DB;
INFO  : Compiling command(queryId=hive_20180709234747_80246bbf-6c30-4824-ad40-7a688b78926a): create database stocks_DB
INFO  : Semantic Analysis Completed
INFO  : Returning Hive schema: Schema(fieldSchemas:null, properties:null)
INFO  : Completed compiling command(queryId=hive_20180709234747_80246bbf-6c30-4824-ad40-7a688b78926a); Time taken: 0.016 seconds
INFO  : Executing command(queryId=hive_20180709234747_80246bbf-6c30-4824-ad40-7a688b78926a): create database stocks_DB
INFO  : Starting task [Stage-0:DDL] in serial mode
INFO  : Completed executing command(queryId=hive_20180709234747_80246bbf-6c30-4824-ad40-7a688b78926a); Time taken: 0.459 seconds
INFO  : OK
No rows affected (0.503 seconds)
0: jdbc:hive2://localhost:10000/default> use stocks_DB;
INFO  : Compiling command(queryId=hive_20180709234747_e08a8c96-302f-43f8-8697-5ea94d0c8595): use stocks_DB
INFO  : Semantic Analysis Completed
INFO  : Returning Hive schema: Schema(fieldSchemas:null, properties:null)
INFO  : Completed compiling command(queryId=hive_20180709234747_e08a8c96-302f-43f8-8697-5ea94d0c8595); Time taken: 0.021 seconds
INFO  : Executing command(queryId=hive_20180709234747_e08a8c96-302f-43f8-8697-5ea94d0c8595): use stocks_DB
INFO  : Starting task [Stage-0:DDL] in serial mode
INFO  : Completed executing command(queryId=hive_20180709234747_e08a8c96-302f-43f8-8697-5ea94d0c8595); Time taken: 0.014 seconds
INFO  : OK
No rows affected (0.055 seconds)
```

Step 3 - Creating External table stocks

Command –

```
create external table if not exists stocks
(
exch STRING,
symbol STRING,
date STRING,
stocks_price_open FLOAT,
stocks_price_high FLOAT,
stocks_price_low FLOAT,
stocks_price_close FLOAT,
volume INT,
price_adj_close FLOAT
)
COMMENT 'This is External Table for Stocks'
ROW FORMAT DELIMITED
FIELDS TERMINATED BY',';
```

```
0: jdbc:hive2://localhost:10000/default> create external table if not exists stocks
. . . . . . . . . . . . . . . . . . . . . . .> (
. . . . . . . . . . . . . . . . . . . . . . .> exch STRING,
. . . . . . . . . . . . . . . . . . . . . . .> symbol STRING,
. . . . . . . . . . . . . . . . . . . . . . .> date STRING,
. . . . . . . . . . . . . . . . . . . . . . .> stocks_price_open FLOAT,
. . . . . . . . . . . . . . . . . . . . . . .> stocks_price_high FLOAT,
. . . . . . . . . . . . . . . . . . . . . . .> stocks_price_low FLOAT,
. . . . . . . . . . . . . . . . . . . . . . .> stocks_price_close FLOAT,
. . . . . . . . . . . . . . . . . . . . . . .> volume INT,
. . . . . . . . . . . . . . . . . . . . . . .> price_adj_close FLOAT
. . . . . . . . . . . . . . . . . . . . . . .> )
. . . . . . . . . . . . . . . . . . . . . . .> COMMENT 'This is External Table for Stocks'
. . . . . . . . . . . . . . . . . . . . . . .> ROW FORMAT DELIMITED
. . . . . . . . . . . . . . . . . . . . . . .> FIELDS TERMINATED BY',';
```

```
0: jdbc:hive2://localhost:10000/default> show tables in stocks_DB;
INFO  : Compiling command(queryId=hive_20180709234949_0bf57d43-e864-460f-ac11-9d36920d8ade): show tables in stocks_DB
INFO  : Semantic Analysis Completed
INFO  : Returning Hive schema: Schema(fieldSchemas:[FieldSchema(name:tab_name, type:string, comment:from deserializer)], properties:null)
INFO  : Completed compiling command(queryId=hive_20180709234949_0bf57d43-e864-460f-ac11-9d36920d8ade); Time taken: 0.019 seconds
INFO  : Executing command(queryId=hive_20180709234949_0bf57d43-e864-460f-ac11-9d36920d8ade): show tables in stocks_DB
INFO  : Starting task [Stage-0:DDL] in serial mode
INFO  : Completed executing command(queryId=hive_20180709234949_0bf57d43-e864-460f-ac11-9d36920d8ade); Time taken: 0.036 seconds
INFO  : OK
+-----------+--+
| tab_name  |
+-----------+--+
| stocks    |
+-----------+--+
```

Step 4 - Executing Queries

Query 1: Finding top 10 companies according to number of stocks

Command - `select symbol,count(symbol) as count from stocks group by symbol order by count desc limit 10;`

```
0: jdbc:hive2://localhost:10000/default> select symbol,count(symbol) as count from stocks group by symbol order by count desc limit 10;
INFO  : Compiling command(queryId=hive_20180710002323_ca41f7dc-9a7d-4864-bcdb-36e569775482): select symbol,count(symbol) as count from stocks group by symbol
 order by count desc limit 10
INFO  : Semantic Analysis Completed
INFO  : Returning Hive schema: Schema(fieldSchemas:[FieldSchema(name:symbol, type:string, comment:null), FieldSchema(name:count, type:bigint, comment:null)],
 properties:null)
INFO  : Completed compiling command(queryId=hive_20180710002323_ca41f7dc-9a7d-4864-bcdb-36e569775482); Time taken: 0.273 seconds
INFO  : Executing command(queryId=hive_20180710002323_ca41f7dc-9a7d-4864-bcdb-36e569775482): select symbol,count(symbol) as count from stocks group by symbol
 order by count desc limit 10
INFO  : Query ID = hive_20180710002323_ca41f7dc-9a7d-4864-bcdb-36e569775482
INFO  : Total jobs = 2
INFO  : Launching Job 1 out of 2
INFO  : Starting task [Stage-1:MAPRED] in serial mode
INFO  : Number of reduce tasks not specified. Estimated from input data size: 1
INFO  : In order to change the average load for a reducer (in bytes):
INFO  :   set hive.exec.reducers.bytes.per.reducer=<number>
INFO  : In order to limit the maximum number of reducers:
INFO  :   set hive.exec.reducers.max=<number>
INFO  : In order to set a constant number of reducers:
INFO  :   set mapreduce.job.reduces=<number>
INFO  : number of splits:1
INFO  : Submitting tokens for job: job_1531159788380_0006
INFO  : The url to track the job: http://quickstart.cloudera:8088/proxy/application_1531159788380_0006/
INFO  : Starting Job = job_1531159788380_0006, Tracking URL = http://quickstart.cloudera:8088/proxy/application_1531159788380_0006/
INFO  : Kill Command = /usr/lib/hadoop/bin/hadoop job  -kill job_1531159788380_0006
INFO  : Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
INFO  : 2018-07-10 00:23:41,179 Stage-1 map = 0%,  reduce = 0%
INFO  : 2018-07-10 00:23:48,588 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 2.92 sec
INFO  : 2018-07-10 00:23:55,938 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 4.6 sec
INFO  : MapReduce Total cumulative CPU time: 4 seconds 600 msec
INFO  : Ended Job = job_1531159788380_0006
INFO  : Launching Job 2 out of 2
INFO  : Starting task [Stage-2:MAPRED] in serial mode
INFO  : Number of reduce tasks determined at compile time: 1
INFO  : In order to change the average load for a reducer (in bytes):
INFO  :   set hive.exec.reducers.bytes.per.reducer=<number>
INFO  : In order to limit the maximum number of reducers:
INFO  :   set hive.exec.reducers.max=<number>
```

```
INFO  :   set hive.exec.reducers.bytes.per.reducer=<number>
INFO  : In order to limit the maximum number of reducers:
INFO  :   set hive.exec.reducers.max=<number>
INFO  : In order to set a constant number of reducers:
INFO  :   set mapreduce.job.reduces=<number>
INFO  : number of splits:1
INFO  : Submitting tokens for job: job_1531159788380_0007
INFO  : The url to track the job: http://quickstart.cloudera:8088/proxy/application_1531159788380_0007/
INFO  : Starting Job = job_1531159788380_0007, Tracking URL = http://quickstart.cloudera:8088/proxy/application_1531159788380_0007/
INFO  : Kill Command = /usr/lib/hadoop/bin/hadoop job  -kill job_1531159788380_0007
INFO  : Hadoop job information for Stage-2: number of mappers: 1; number of reducers: 1
INFO  : 2018-07-10 00:24:04,053 Stage-2 map = 0%,  reduce = 0%
INFO  : 2018-07-10 00:24:09,438 Stage-2 map = 100%,  reduce = 0%, Cumulative CPU 1.13 sec
INFO  : 2018-07-10 00:24:16,793 Stage-2 map = 100%,  reduce = 100%, Cumulative CPU 2.79 sec
INFO  : MapReduce Total cumulative CPU time: 2 seconds 790 msec
INFO  : Ended Job = job_1531159788380_0007
INFO  : MapReduce Jobs Launched:
INFO  : Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 4.6 sec   HDFS Read: 40998932 HDFS Write: 4994 SUCCESS
INFO  : Stage-Stage-2: Map: 1  Reduce: 1   Cumulative CPU: 2.79 sec   HDFS Read: 10050 HDFS Write: 91 SUCCESS
INFO  : Total MapReduce CPU Time Spent: 7 seconds 390 msec
INFO  : Completed executing command(queryId=hive_20180710002323_ca41f7dc-9a7d-4864-bcdb-36e569775482); Time taken: 43.735 seconds
INFO  : OK
+---------+--------+--+
| symbol  | count  |
+---------+--------+--+
| AA      | 12109  |
| AEP     | 10121  |
| AET     | 8353   |
| AXP     | 8290   |
| ASA     | 8095   |
| AMR     | 7590   |
| APA     | 7091   |
| AVP     | 7085   |
| AVT     | 7084   |
| ALK     | 6831   |
+---------+--------+--+
10 rows selected (44.14 seconds)
```

Query 2: Finding top 10 years according to average volume of stocks

Command - `select year(date),avg(volume) as avgVol from stocks group by year(date) order by avgVol desc limit 10;`

```
0: jdbc:hive2://localhost:10000/default> select year(date),avg(volume) as avgVol from stocks group by year(date) order by avgVol desc limit 10;
INFO  : Compiling command(queryId=hive_20180710002626_ba156c06-867f-45b0-9a1f-7744363c10b2): select year(date),avg(volume) as avgVol from stocks group by yea
r(date) order by avgVol desc limit 10
INFO  : Semantic Analysis Completed
INFO  : Returning Hive schema: Schema(fieldSchemas:[FieldSchema(name:_c0, type:int, comment:null), FieldSchema(name:avgvol, type:double, comment:null)], prop
erties:null)
INFO  : Completed compiling command(queryId=hive_20180710002626_ba156c06-867f-45b0-9a1f-7744363c10b2); Time taken: 0.107 seconds
INFO  : Executing command(queryId=hive_20180710002626_ba156c06-867f-45b0-9a1f-7744363c10b2): select year(date),avg(volume) as avgVol from stocks group by yea
r(date) order by avgVol desc limit 10
INFO  : Query ID = hive_20180710002626_ba156c06-867f-45b0-9a1f-7744363c10b2
INFO  : Total jobs = 2
INFO  : Launching Job 1 out of 2
INFO  : Starting task [Stage-1:MAPRED] in serial mode
INFO  : Number of reduce tasks not specified. Estimated from input data size: 1
INFO  : In order to change the average load for a reducer (in bytes):
INFO  :   set hive.exec.reducers.bytes.per.reducer=<number>
INFO  : In order to limit the maximum number of reducers:
INFO  :   set hive.exec.reducers.max=<number>
INFO  : In order to set a constant number of reducers:
INFO  :   set mapreduce.job.reduces=<number>
INFO  : number of splits:1
INFO  : Submitting tokens for job: job_1531159788380_0008
INFO  : The url to track the job: http://quickstart.cloudera:8088/proxy/application_1531159788380_0008/
INFO  : Starting Job = job_1531159788380_0008, Tracking URL = http://quickstart.cloudera:8088/proxy/application_1531159788380_0008/
INFO  : Kill Command = /usr/lib/hadoop/bin/hadoop job  -kill job_1531159788380_0008
INFO  : Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
INFO  : 2018-07-10 00:26:43,971 Stage-1 map = 0%,  reduce = 0%
INFO  : 2018-07-10 00:26:52,518 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 5.64 sec
INFO  : 2018-07-10 00:26:58,831 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 7.37 sec
INFO  : MapReduce Total cumulative CPU time: 7 seconds 370 msec
INFO  : Ended Job = job_1531159788380_0008
INFO  : Launching Job 2 out of 2
INFO  : Kill Command = /usr/lib/hadoop/bin/hadoop job  -kill job_1531159788380_0009
INFO  : Hadoop job information for Stage-2: number of mappers: 1; number of reducers: 1
INFO  : 2018-07-10 00:27:05,691 Stage-2 map = 0%,  reduce = 0%
INFO  : 2018-07-10 00:27:12,189 Stage-2 map = 100%,  reduce = 0%, Cumulative CPU 1.63 sec
INFO  : 2018-07-10 00:27:18,535 Stage-2 map = 100%,  reduce = 100%, Cumulative CPU 3.34 sec
INFO  : MapReduce Total cumulative CPU time: 3 seconds 340 msec
INFO  : Ended Job = job_1531159788380_0009
INFO  : MapReduce Jobs Launched:
INFO  : Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 7.37 sec   HDFS Read: 40999849 HDFS Write: 1468 SUCCESS
INFO  : Stage-Stage-2: Map: 1  Reduce: 1   Cumulative CPU: 3.34 sec   HDFS Read: 6538 HDFS Write: 235 SUCCESS
INFO  : Total MapReduce CPU Time Spent: 10 seconds 710 msec
INFO  : Completed executing command(queryId=hive_20180710002626_ba156c06-867f-45b0-9a1f-7744363c10b2); Time taken: 43.49 seconds
INFO  : OK
+-------+---------------------+--+
|  _c0  |       avgvol        |
+-------+---------------------+--+
| 2009  | 2144999.0106877508  |
| 2010  | 2009298.4872611465  |
| 2008  | 1941244.2091383133  |
| 2007  | 1337635.7611070648  |
| 1983  | 1087647.1091521909  |
| 2006  | 1059718.7874033398  |
| 1982  | 1021882.4769433466  |
| 2003  | 941810.9683610097   |
| 2005  | 902292.056767634    |
| 2002  | 899578.376947301    |
+-------+---------------------+--+
10 rows selected (43.678 seconds)
```

Query 3: Finding top 10 month & year according to rise in stocks price

Command - `select month(date) as month ,year(date) as year, (max(stocks_price_high) - min(stocks_price_low)) as stocksPriceRise from stocks group by month(date), year(date) order by stocksPriceRise DESC LIMIT 10;`

```
0: jdbc:hive2://localhost:10000/default> select month(date) as month ,year(date) as year, (max(stocks_price_high) - min(stocks_price_low)) as stocksPriceRise
 from stocks group by month(date), year(date) order by stocksPriceRise DESC LIMIT 10;
INFO  : Compiling command(queryId=hive_20180710003131_efdf73b3-2b46-47ff-8cff-525a4040b137): select month(date) as month ,year(date) as year, (max(stocks_pri
ce_high) - min(stocks_price_low)) as stocksPriceRise from stocks group by month(date), year(date) order by stocksPriceRise DESC LIMIT 10
INFO  : Semantic Analysis Completed
INFO  : Returning Hive schema: Schema(fieldSchemas:[FieldSchema(name:month, type:int, comment:null), FieldSchema(name:year, type:int, comment:null), FieldSch
ema(name:stockspricerise, type:float, comment:null)], properties:null)
INFO  : Completed compiling command(queryId=hive_20180710003131_efdf73b3-2b46-47ff-8cff-525a4040b137); Time taken: 0.122 seconds
INFO  : Executing command(queryId=hive_20180710003131_efdf73b3-2b46-47ff-8cff-525a4040b137): select month(date) as month ,year(date) as year, (max(stocks_pri
ce_high) - min(stocks_price_low)) as stocksPriceRise from stocks group by month(date), year(date) order by stocksPriceRise DESC LIMIT 10
INFO  : Query ID = hive_20180710003131_efdf73b3-2b46-47ff-8cff-525a4040b137
INFO  : Total jobs = 2
INFO  : Launching Job 1 out of 2
INFO  : Starting task [Stage-1:MAPRED] in serial mode
INFO  : Number of reduce tasks not specified. Estimated from input data size: 1
INFO  : In order to change the average load for a reducer (in bytes):
INFO  :   set hive.exec.reducers.bytes.per.reducer=<number>
INFO  : In order to limit the maximum number of reducers:
INFO  :   set hive.exec.reducers.max=<number>
INFO  : In order to set a constant number of reducers:
INFO  :   set mapreduce.job.reduces=<number>
INFO  : number of splits:1
INFO  : Submitting tokens for job: job_1531159788380_0010
INFO  : The url to track the job: http://quickstart.cloudera:8088/proxy/application_1531159788380_0010/
INFO  : Starting Job = job_1531159788380_0010, Tracking URL = http://quickstart.cloudera:8088/proxy/application_1531159788380_0010/
INFO  : Kill Command = /usr/lib/hadoop/bin/hadoop job  -kill job_1531159788380_0010
INFO  : Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
INFO  : 2018-07-10 00:31:31,119 Stage-1 map = 0%,  reduce = 0%
INFO  : 2018-07-10 00:31:41,939 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 7.48 sec
INFO  : 2018-07-10 00:31:49,259 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 9.72 sec
INFO  : MapReduce Total cumulative CPU time: 9 seconds 720 msec
INFO  : Ended Job = job_1531159788380_0010
INFO  : Launching Job 2 out of 2
INFO  :   set hive.exec.reducers.bytes.per.reducer=<number>
INFO  : In order to limit the maximum number of reducers:
INFO  :   set hive.exec.reducers.max=<number>
INFO  : In order to set a constant number of reducers:
INFO  :   set mapreduce.job.reduces=<number>
INFO  : number of splits:1
INFO  : Submitting tokens for job: job_1531159788380_0011
INFO  : The url to track the job: http://quickstart.cloudera:8088/proxy/application_1531159788380_0011/
INFO  : Starting Job = job_1531159788380_0011, Tracking URL = http://quickstart.cloudera:8088/proxy/application_1531159788380_0011/
INFO  : Kill Command = /usr/lib/hadoop/bin/hadoop job  -kill job_1531159788380_0011
INFO  : Hadoop job information for Stage-2: number of mappers: 1; number of reducers: 1
INFO  : 2018-07-10 00:31:56,033 Stage-2 map = 0%,  reduce = 0%
INFO  : 2018-07-10 00:32:02,323 Stage-2 map = 100%,  reduce = 0%, Cumulative CPU 1.18 sec
INFO  : 2018-07-10 00:32:09,626 Stage-2 map = 100%,  reduce = 100%, Cumulative CPU 2.88 sec
INFO  : MapReduce Total cumulative CPU time: 2 seconds 880 msec
INFO  : Ended Job = job_1531159788380_0011
INFO  : MapReduce Jobs Launched:
INFO  : Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 9.72 sec   HDFS Read: 41001141 HDFS Write: 14686 SUCCESS
INFO  : Stage-Stage-2: Map: 1  Reduce: 1   Cumulative CPU: 2.88 sec   HDFS Read: 19845 HDFS Write: 147 SUCCESS
INFO  : Total MapReduce CPU Time Spent: 12 seconds 600 msec
INFO  : Completed executing command(queryId=hive_20180710003131_efdf73b3-2b46-47ff-8cff-525a4040b137); Time taken: 46.419 seconds
INFO  : OK
+--------+-------+---------------------+--+
| month  | year  |   stockspricerise   |
+--------+-------+---------------------+--+
| 2      | 2007  | 465.7699890136719   |
| 12     | 2006  | 444.5899963378906   |
| 1      | 2007  | 444.5199890136719   |
| 6      | 2007  | 435.6399841308594   |
| 9      | 2008  | 430.4900207519531   |
| 5      | 2007  | 429.2699890136719   |
| 3      | 2007  | 420.6000061035156   |
| 7      | 2007  | 420.42999267578125  |
| 10     | 2007  | 418.9100036621094   |
| 4      | 2007  | 415.760009765625    |
+--------+-------+---------------------+--+
10 rows selected (46.618 seconds)
```

Query 4: Finding top 10 max difference between stock price close & stock price adjustment close

Command - `select price_adj_close-stocks_price_close as adj from stocks order by adj  desc limit 10;`

```
0: jdbc:hive2://localhost:10000/default> select price_adj_close-stocks_price_close as adj from stocks order by adj  desc limit 10;
INFO  : Compiling command(queryId=hive_20180710003838_13317352-ef6d-4663-8703-539d8af33595): select price_adj_close-stocks_price_close as adj from stocks ord
er by adj  desc limit 10
INFO  : Semantic Analysis Completed
INFO  : Returning Hive schema: Schema(fieldSchemas:[FieldSchema(name:adj, type:float, comment:null)], properties:null)
INFO  : Completed compiling command(queryId=hive_20180710003838_13317352-ef6d-4663-8703-539d8af33595); Time taken: 0.101 seconds
INFO  : Executing command(queryId=hive_20180710003838_13317352-ef6d-4663-8703-539d8af33595): select price_adj_close-stocks_price_close as adj from stocks ord
er by adj  desc limit 10
INFO  : Query ID = hive_20180710003838_13317352-ef6d-4663-8703-539d8af33595
INFO  : Total jobs = 1
INFO  : Launching Job 1 out of 1
INFO  : Starting task [Stage-1:MAPRED] in serial mode
INFO  : Number of reduce tasks determined at compile time: 1
INFO  : In order to change the average load for a reducer (in bytes):
INFO  :   set hive.exec.reducers.bytes.per.reducer=<number>
INFO  : In order to limit the maximum number of reducers:
INFO  :   set hive.exec.reducers.max=<number>
INFO  : In order to set a constant number of reducers:
INFO  :   set mapreduce.job.reduces=<number>
INFO  : number of splits:1
INFO  : Submitting tokens for job: job_1531159788380_0013
INFO  : The url to track the job: http://quickstart.cloudera:8088/proxy/application_1531159788380_0013/
INFO  : Starting Job = job_1531159788380_0013, Tracking URL = http://quickstart.cloudera:8088/proxy/application_1531159788380_0013/
INFO  : Kill Command = /usr/lib/hadoop/bin/hadoop job  -kill job_1531159788380_0013
INFO  : Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
INFO  : 2018-07-10 00:38:24,570 Stage-1 map = 0%,  reduce = 0%
INFO  : 2018-07-10 00:38:33,894 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 5.97 sec
INFO  : 2018-07-10 00:38:40,247 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 7.66 sec
INFO  : MapReduce Total cumulative CPU time: 7 seconds 660 msec
INFO  : Ended Job = job_1531159788380_0013
INFO  : MapReduce Jobs Launched:
INFO  : Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 7.66 sec   HDFS Read: 40999660 HDFS Write: 81 SUCCESS
INFO  : Total MapReduce CPU Time Spent: 7 seconds 660 msec
INFO  : Completed executing command(queryId=hive_20180710003838_13317352-ef6d-4663-8703-539d8af33595); Time taken: 22.513 seconds
INFO  : OK
```

```
+---------------------+--+
|         adj         |
+---------------------+--+
| 1843.7900390625     |
| 1810.360107421875   |
| 1800.4000244140625  |
| 1786.3199462890625  |
| 1780.81005859375    |
| 1779.239990234375   |
| 1779.239990234375   |
| 1769.280029296875   |
| 1769.280029296875   |
| 1766.22998046875    |
+---------------------+--+
10 rows selected (22.688 seconds)
```

Query 5: Finding top 10 companies according to volume of stocks

Command - `select symbol,max(volume) as MAXVOL from stocks group by symbol order by MAXVOL desc limit 10;`

```
0: jdbc:hive2://localhost:10000/default> select symbol,max(volume) as MAXVOL from stocks group by symbol order by MAXVOL desc limit 10;
INFO  : Compiling command(queryId=hive_20180710004040_4340ed8b-1934-4883-b939-35bd2daf7fbf): select symbol,max(volume) as MAXVOL from stocks group by symbol
order by MAXVOL desc limit 10
INFO  : Semantic Analysis Completed
INFO  : Returning Hive schema: Schema(fieldSchemas:[FieldSchema(name:symbol, type:string, comment:null), FieldSchema(name:maxvol, type:int, comment:null)], p
roperties:null)
INFO  : Completed compiling command(queryId=hive_20180710004040_4340ed8b-1934-4883-b939-35bd2daf7fbf); Time taken: 0.094 seconds
INFO  : Executing command(queryId=hive_20180710004040_4340ed8b-1934-4883-b939-35bd2daf7fbf): select symbol,max(volume) as MAXVOL from stocks group by symbol
order by MAXVOL desc limit 10
INFO  : Query ID = hive_20180710004040_4340ed8b-1934-4883-b939-35bd2daf7fbf
INFO  : Total jobs = 2
INFO  : Launching Job 1 out of 2
INFO  : Starting task [Stage-1:MAPRED] in serial mode
INFO  : Number of reduce tasks not specified. Estimated from input data size: 1
INFO  : In order to change the average load for a reducer (in bytes):
INFO  :   set hive.exec.reducers.bytes.per.reducer=<number>
INFO  : In order to limit the maximum number of reducers:
INFO  :   set hive.exec.reducers.max=<number>
INFO  : In order to set a constant number of reducers:
INFO  :   set mapreduce.job.reduces=<number>
INFO  : number of splits:1
INFO  : Submitting tokens for job: job_1531159788380_0014
INFO  : The url to track the job: http://quickstart.cloudera:8088/proxy/application_1531159788380_0014/
INFO  : Starting Job = job_1531159788380_0014, Tracking URL = http://quickstart.cloudera:8088/proxy/application_1531159788380_0014/
INFO  : Kill Command = /usr/lib/hadoop/bin/hadoop job  -kill job_1531159788380_0014
INFO  : Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
INFO  : 2018-07-10 00:40:39,328 Stage-1 map = 0%,  reduce = 0%
INFO  : 2018-07-10 00:40:46,659 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 3.52 sec
INFO  : 2018-07-10 00:40:52,928 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 5.1 sec
INFO  : MapReduce Total cumulative CPU time: 5 seconds 100 msec
INFO  : Ended Job = job_1531159788380_0014
```

```
INFO  :   set hive.exec.reducers.bytes.per.reducer=<number>
INFO  : In order to limit the maximum number of reducers:
INFO  :   set hive.exec.reducers.max=<number>
INFO  : In order to set a constant number of reducers:
INFO  :   set mapreduce.job.reduces=<number>
INFO  : number of splits:1
INFO  : Submitting tokens for job: job_1531159788380_0015
INFO  : The url to track the job: http://quickstart.cloudera:8088/proxy/application_1531159788380_0015/
INFO  : Starting Job = job_1531159788380_0015, Tracking URL = http://quickstart.cloudera:8088/proxy/application_1531159788380_0015/
INFO  : Kill Command = /usr/lib/hadoop/bin/hadoop job  -kill job_1531159788380_0015
INFO  : Hadoop job information for Stage-2: number of mappers: 1; number of reducers: 1
INFO  : 2018-07-10 00:40:59,769 Stage-2 map = 0%,  reduce = 0%
INFO  : 2018-07-10 00:41:06,043 Stage-2 map = 100%,  reduce = 0%, Cumulative CPU 1.13 sec
INFO  : 2018-07-10 00:41:12,626 Stage-2 map = 100%,  reduce = 100%, Cumulative CPU 2.87 sec
INFO  : MapReduce Total cumulative CPU time: 2 seconds 870 msec
INFO  : Ended Job = job_1531159788380_0015
INFO  : MapReduce Jobs Launched:
INFO  : Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 5.1 sec   HDFS Read: 40999017 HDFS Write: 5259 SUCCESS
INFO  : Stage-Stage-2: Map: 1  Reduce: 1   Cumulative CPU: 2.87 sec   HDFS Read: 10299 HDFS Write: 135 SUCCESS
INFO  : Total MapReduce CPU Time Spent: 7 seconds 970 msec
INFO  : Completed executing command(queryId=hive_20180710004040_4340ed8b-1934-4883-b939-35bd2daf7fbf); Time taken: 40.643 seconds
INFO  : OK
+---------+------------+--+
| symbol  |   maxvol   |
+---------+------------+--+
| AVP     | 318182200  |
| AA      | 242106500  |
| AMD     | 163101700  |
| AIG     | 148878600  |
| ABK     | 112834200  |
| AXL     | 103526300  |
| AXP     | 90336900   |
| AMR     | 89947900   |
| ACE     | 74437100   |
| AET     | 67642400   |
+---------+------------+--+
10 rows selected (40.809 seconds)
```

Query 6: Finding top 10 times when price adjustment close and price close has positive difference

Command - `select count(price_adj_close-stocks_price_close) as adj from stocks where (price_adj_close-stocks_price_close) > 0 limit 10;`

```
0: jdbc:hive2://localhost:10000/default> select count(price_adj_close-stocks_price_close) as adj from stocks where (price_adj_close-stocks_price_close) !0 li
mit 10;
```

```
INFO  : 2018-07-10 01:47:37,214 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 7.98 sec
INFO  : MapReduce Total cumulative CPU time: 7 seconds 980 msec
INFO  : Ended Job = job_1531165075887_0003
INFO  : MapReduce Jobs Launched:
INFO  : Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 7.98 sec   HDFS Read: 81992182 HDFS Write: 8 SUCCESS
INFO  : Total MapReduce CPU Time Spent: 7 seconds 980 msec
INFO  : Completed executing command(queryId=hive_20180710014646_62180fb6-28ce-4a2a-b00a-fa1b14032a83); Time taken: 38.261 seconds
INFO  : OK
+----------+--+
|   adj    |
+----------+--+
| 1284850  |
+----------+--+
```

Query 7: Finding top 10 dates according to number of stocks of company AVP

Command - `select date, volume from stocks where symbol = "AVP" order by volume desc limit 10;`

```
0: jdbc:hive2://localhost:10000/default> select date, volume from stocks where symbol = "AVP" order by volume desc limit 10;
INFO  : Compiling command(queryId=hive_20180710004343_3b51aa20-1328-470e-8995-d9bb487853e8): select date, volume from stocks where symbol = "AVP" order by vo
lume desc limit 10
INFO  : Semantic Analysis Completed
INFO  : Returning Hive schema: Schema(fieldSchemas:[FieldSchema(name:date, type:string, comment:null), FieldSchema(name:volume, type:int, comment:null)], pro
perties:null)
INFO  : Completed compiling command(queryId=hive_20180710004343_3b51aa20-1328-470e-8995-d9bb487853e8); Time taken: 0.144 seconds
INFO  : Executing command(queryId=hive_20180710004343_3b51aa20-1328-470e-8995-d9bb487853e8): select date, volume from stocks where symbol = "AVP" order by vo
lume desc limit 10
INFO  : Query ID = hive_20180710004343_3b51aa20-1328-470e-8995-d9bb487853e8
INFO  : Total jobs = 1
INFO  : Launching Job 1 out of 1
INFO  : Starting task [Stage-1:MAPRED] in serial mode
INFO  : Number of reduce tasks determined at compile time: 1
INFO  : In order to change the average load for a reducer (in bytes):
INFO  :   set hive.exec.reducers.bytes.per.reducer=<number>
INFO  : In order to limit the maximum number of reducers:
INFO  :   set hive.exec.reducers.max=<number>
INFO  : In order to set a constant number of reducers:
INFO  :   set mapreduce.job.reduces=<number>
INFO  : number of splits:1
INFO  : Submitting tokens for job: job_1531159788380_0016
INFO  : The url to track the job: http://quickstart.cloudera:8088/proxy/application_1531159788380_0016/
INFO  : Starting Job = job_1531159788380_0016, Tracking URL = http://quickstart.cloudera:8088/proxy/application_1531159788380_0016/
INFO  : Kill Command = /usr/lib/hadoop/bin/hadoop job  -kill job_1531159788380_0016
INFO  : Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
INFO  : 2018-07-10 00:43:42,249 Stage-1 map = 0%,  reduce = 0%
INFO  : 2018-07-10 00:43:49,555 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 3.74 sec
INFO  : 2018-07-10 00:43:56,876 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 5.53 sec
INFO  : MapReduce Total cumulative CPU time: 5 seconds 530 msec
INFO  : Ended Job = job_1531159788380_0016
INFO  : MapReduce Jobs Launched:
INFO  : Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 5.53 sec   HDFS Read: 40999895 HDFS Write: 202 SUCCESS
INFO  : Total MapReduce CPU Time Spent: 5 seconds 530 msec
INFO  : Completed executing command(queryId=hive_20180710004343_3b51aa20-1328-470e-8995-d9bb487853e8); Time taken: 22.903 seconds
INFO  : OK
+-------------+------------+--+
|    date     |   volume   |
+-------------+------------+--+
| 1988-05-12  | 318182200  |
| 1988-02-12  | 165642600  |
| 1991-03-14  | 93343200   |
| 1987-11-13  | 87786600   |
| 1988-02-08  | 82144000   |
| 1989-05-03  | 79389800   |
| 1988-02-09  | 77893300   |
| 1989-05-10  | 57950900   |
| 1989-05-18  | 46672000   |
| 1989-05-19  | 38488800   |
+-------------+------------+--+
10 rows selected (23.111 seconds)
```

Step 3 - Creating External table stocks

Command –

```
create table stocks_partition(
exch STRING,
symbol STRING,
date STRING,
stocks_price_close FLOAT,
stocks_price_high FLOAT,
stocks_price_low FLOAT,
stocks_price_open FLOAT,
volume INT,
price_adj_close FLOAT)
PARTITIONED BY (exch_name STRING,yr STRING, sym STRING)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';
```

```
0: jdbc:hive2://localhost:10000/default> create table stocks_partition(
. . . . . . . . . . . . . . . . . . . . .> exch STRING,
. . . . . . . . . . . . . . . . . . . . .> symbol STRING,
. . . . . . . . . . . . . . . . . . . . .> date STRING,
. . . . . . . . . . . . . . . . . . . . .> stocks_price_close FLOAT,
. . . . . . . . . . . . . . . . . . . . .> stocks_price_high FLOAT,
. . . . . . . . . . . . . . . . . . . . .> stocks_price_low FLOAT,
. . . . . . . . . . . . . . . . . . . . .> stocks_price_open FLOAT,
. . . . . . . . . . . . . . . . . . . . .> volume INT,
. . . . . . . . . . . . . . . . . . . . .> price_adj_close FLOAT)
. . . . . . . . . . . . . . . . . . . . .> PARTITIONED BY (exch_name STRING,yr STRING, sym STRING)
. . . . . . . . . . . . . . . . . . . . .> ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';
```

```
0: jdbc:hive2://localhost:10000/default> set hive.exec.dynamic.partition = true;
No rows affected (0.013 seconds)
0: jdbc:hive2://localhost:10000/default> set hive.exec.max.dynamic.partitions = 2000;
No rows affected (0.005 seconds)
0: jdbc:hive2://localhost:10000/default> set hive.exec.max.dynamic.partitions.pernode = 1000;
No rows affected (0.004 seconds)
```

Query 8: Loading data into partitions

Command –

```
INSERT OVERWRITE TABLE stocks_partition
PARTITION (exch_name = 'NYSE',yr, sym)
SELECT *, year(date), symbol
FROM stocks WHERE year(date) IN ('2010','2009','2008','2007','2006')
AND symbol LIKE 'A%';
```

Applications Places System Sat Jul 7, 5:09 AM cloudera

cloudera@quickstart:/bin

File Edit View Search Terminal Help

```
1: jdbc:hive2://localhost:10000/default> INSERT OVERWRITE TABLE stocks_partition
1: jdbc:hive2://localhost:10000/default> PARTITION (exch_name = 'NYSE',yr, sym)
1: jdbc:hive2://localhost:10000/default> SELECT *, year(date), symbol
1: jdbc:hive2://localhost:10000/default> FROM stocks WHERE year(date) IN ('2010','2009','2008','2007','2006') AND symbol LIKE 'A%';
INFO  : Number of reduce tasks is set to 0 since there's no reduce operator
WARN  : Hadoop command-line option parsing not performed. Implement the Tool interface and execute your application with ToolRunner to remedy this.
INFO  : Starting Job = job_1530898074895_0013, Tracking URL = http://quickstart.cloudera:8088/proxy/application_1530898074895_0013/
INFO  : Kill Command = /usr/lib/hadoop/bin/hadoop job  -kill job_1530898074895_0013
INFO  : Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 0
INFO  : 2018-07-06 12:11:29,728 Stage-1 map = 0%,  reduce = 0%
INFO  : 2018-07-06 12:12:26,520 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 38.09 sec
INFO  : 2018-07-06 12:13:27,997 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 55.49 sec
INFO  : MapReduce Total cumulative CPU time: 55 seconds 490 msec
INFO  : Ended Job = job_1530898074895_0013
INFO  : Stage-4 is selected by condition resolver.
INFO  : Stage-3 is filtered out by condition resolver.
INFO  : Stage-5 is filtered out by condition resolver.
INFO  : Moving data to: hdfs://quickstart.cloudera:8020/user/hive/warehouse/stocksdb.db/stocks_partition/exch_name=NYSE/.hive-staging_hive_2018-07-06_12-11-04_519_54014
33050848256017-1/-ext-10000 from hdfs://quickstart.cloudera:8020/user/hive/warehouse/stocksdb.db/stocks_partition/exch_name=NYSE/.hive-staging_hive_2018-07-06_12-11-04_
519_5401433050848256017-1/-ext-10002
INFO  : Loading data to table stocksdb.stocks_partition partition (exch_name=NYSE, yr=null, sym=null) from hdfs://quickstart.cloudera:8020/user/hive/warehouse/stocksdb.
db/stocks_partition/exch_name=NYSE/.hive-staging_hive_2018-07-06_12-11-04_519_5401433050848256017-1/-ext-10000

[1]+  Stopped                 beeline
[cloudera@quickstart bin]$ beeline
Beeline version 1.1.0-cdh5.4.2 by Apache Hive
beeline> !connect jdbc:hive2://localhost 10000/default
scan complete in 9ms
Connecting to jdbc:hive2://localhost
Enter password for jdbc:hive2://localhost:
Error: Could not open client transport with JDBC Uri: jdbc:hive2://localhost: Cannot open without port. (state=08S01,code=0)
0: jdbc:hive2://localhost (closed)> !connect jdbc:hive2://localhost:10000/default
Connecting to jdbc:hive2://localhost:10000/default
Enter username for jdbc:hive2://localhost:10000/default: cloudera
Enter password for jdbc:hive2://localhost:10000/default: ********
Connected to: Apache Hive (version 1.1.0-cdh5.4.2)
Driver: Hive JDBC (version 1.1.0-cdh5.4.2)
Transaction isolation: TRANSACTION_REPEATABLE_READ
1: jdbc:hive2://localhost:10000/default> show partitions stocks_partition;
```

*report (~/Desktop) - ... | cloudera@quickstart:/... | [cloudera@quickstart:... | [Hue - Job Browser - M...

Applications Places System Sat Jul 7, 5:09 AM cloudera

cloudera@quickstart:/bin

File Edit View Search Terminal Help

```
1: jdbc:hive2://localhost:10000/default> show partitions stocks_partition;
Error: Error while compiling statement: FAILED: SemanticException [Error 10001]: Table not found stocks_partition (state=42S02,code=10001)
1: jdbc:hive2://localhost:10000/default> use stocksDB;
No rows affected (0.294 seconds)
1: jdbc:hive2://localhost:10000/default> show partitions stocks_partition;
+---------------------------------+--+
|            partition            |
+---------------------------------+--+
| exch_name=NYSE/yr=2006/sym=AAV  |
| exch_name=NYSE/yr=2006/sym=AB   |
| exch_name=NYSE/yr=2006/sym=ABB  |
| exch_name=NYSE/yr=2006/sym=ABG  |
| exch_name=NYSE/yr=2006/sym=ABR  |
| exch_name=NYSE/yr=2006/sym=ACC  |
| exch_name=NYSE/yr=2006/sym=ADX  |
| exch_name=NYSE/yr=2006/sym=ADY  |
| exch_name=NYSE/yr=2006/sym=AEC  |
| exch_name=NYSE/yr=2006/sym=AED  |
| exch_name=NYSE/yr=2006/sym=AEH  |
| exch_name=NYSE/yr=2006/sym=AEL  |
| exch_name=NYSE/yr=2006/sym=AEM  |
| exch_name=NYSE/yr=2006/sym=AES  |
| exch_name=NYSE/yr=2006/sym=AFE  |
| exch_name=NYSE/yr=2006/sym=AFG  |
| exch_name=NYSE/yr=2006/sym=AFL  |
| exch_name=NYSE/yr=2006/sym=AGD  |
| exch_name=NYSE/yr=2006/sym=AGM  |
| exch_name=NYSE/yr=2006/sym=AGP  |
| exch_name=NYSE/yr=2006/sym=AGU  |
| exch_name=NYSE/yr=2006/sym=AHD  |
| exch_name=NYSE/yr=2006/sym=AIQ  |
| exch_name=NYSE/yr=2006/sym=AIZ  |
| exch_name=NYSE/yr=2006/sym=AKF  |
| exch_name=NYSE/yr=2006/sym=AKS  |
| exch_name=NYSE/yr=2006/sym=ALE  |
| exch_name=NYSE/yr=2006/sym=ALG  |
| exch_name=NYSE/yr=2006/sym=ALU  |
| exch_name=NYSE/yr=2006/sym=ALZ  |
| exch_name=NYSE/yr=2006/sym=AM   |
```

*report (~/Desktop) - ... | cloudera@quickstart:/... | [cloudera@quickstart:... | [Hue - Job Browser - M...

Query 9: Finding maximum stock price in AVP company partition

Command –

```
SELECT MAX(stocks_price_high) as max_high FROM stocks_partition
WHERE sym = 'AVP';
```

```
1: jdbc:hive2://localhost:10000/default> SELECT sym,MAX(stocks_price_high) FROM stocks_partition
1: jdbc:hive2://localhost:10000/default> WHERE sym = 'AVP';
Error: Error while compiling statement: FAILED: SemanticException [Error 10025]: Line 1:7 Expression not in GROUP BY key 'sym' (state=42000,code=10025)
1: jdbc:hive2://localhost:10000/default> SELECT MAX(stocks_price_high) FROM stocks_partition
1: jdbc:hive2://localhost:10000/default> WHERE sym = 'AVP';
INFO  : Number of reduce tasks determined at compile time: 1
INFO  : In order to change the average load for a reducer (in bytes):
INFO  :   set hive.exec.reducers.bytes.per.reducer=<number>
INFO  : In order to limit the maximum number of reducers:
INFO  :   set hive.exec.reducers.max=<number>
INFO  : In order to set a constant number of reducers:
INFO  :   set mapreduce.job.reduces=<number>
WARN  : Hadoop command-line option parsing not performed. Implement the Tool interface and execute your application with ToolRunner to remedy this.
INFO  : Starting Job = job_1530898074895_0020, Tracking URL = http://quickstart.cloudera:8088/proxy/application_1530898074895_0020/
INFO  : Kill Command = /usr/lib/hadoop/bin/hadoop job  -kill job_1530898074895_0020
INFO  : Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
INFO  : 2018-07-07 11:58:48,034 Stage-1 map = 0%,  reduce = 0%
INFO  : 2018-07-07 11:59:00,248 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 3.72 sec
INFO  : 2018-07-07 11:59:13,213 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 8.63 sec
INFO  : MapReduce Total cumulative CPU time: 8 seconds 630 msec
INFO  : Ended Job = job_1530898074895_0020
+--------------------+--+
|        _c0         |
+--------------------+--+
| 45.34000015258789  |
+--------------------+--+
```

Query 10: Finding minimum stock price in AVP company partition

Command –

```
SELECT MIN(stocks_price_low) as min_low FROM stocks_partition
WHERE sym = 'AVP';
```

```
1: jdbc:hive2://localhost:10000/default> SELECT MIN(stocks_price_low) as min_low FROM stocks_partition
1: jdbc:hive2://localhost:10000/default> WHERE sym = 'AVP';
INFO  : Number of reduce tasks determined at compile time: 1
INFO  : In order to change the average load for a reducer (in bytes):
INFO  :   set hive.exec.reducers.bytes.per.reducer=<number>
INFO  : In order to limit the maximum number of reducers:
INFO  :   set hive.exec.reducers.max=<number>
INFO  : In order to set a constant number of reducers:
INFO  :   set mapreduce.job.reduces=<number>
WARN  : Hadoop command-line option parsing not performed. Implement the Tool interface and execute your application with ToolRunner to remedy this.
INFO  : Starting Job = job_1530898074895_0022, Tracking URL = http://quickstart.cloudera:8088/proxy/application_1530898074895_0022/
INFO  : Kill Command = /usr/lib/hadoop/bin/hadoop job  -kill job_1530898074895_0022
INFO  : Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
INFO  : 2018-07-07 12:01:58,545 Stage-1 map = 0%,  reduce = 0%
INFO  : 2018-07-07 12:02:10,499 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 3.6 sec
INFO  : 2018-07-07 12:02:22,458 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 8.0 sec
INFO  : MapReduce Total cumulative CPU time: 8 seconds 0 msec
INFO  : Ended Job = job_1530898074895_0022
+---------------------+--+
|       min_low       |
+---------------------+--+
| 14.399999618530273  |
+---------------------+--+
```

What is Apache Pig?

Apache Pig is an abstraction over MapReduce. It is a tool/platform which is used to analyze larger sets of data representing them as data flows. Pig is generally used with Hadoop; we can perform all the data manipulation operations in Hadoop using Apache Pig.

To write data analysis programs, Pig provides a high-level language known as Pig Latin. This language provides various operators using which programmers can develop their own functions for reading, writing, and processing data.

To analyze data using Apache Pig, programmers need to write scripts using Pig Latin language. All these scripts are internally converted to Map and Reduce tasks. Apache Pig has a component known as Pig Engine that accepts the Pig Latin scripts as input and converts those scripts into MapReduce jobs.

Why Do We Need Apache Pig?

Programmers who are not so good at Java normally used to struggle working with Hadoop, especially while performing any MapReduce tasks. Apache Pig is a boon for all such programmers.

- Using Pig Latin, programmers can perform MapReduce tasks easily without having to type complex codes in Java.
- Apache Pig uses multi-query approach, thereby reducing the length of codes. For example, an operation that would require you to type 200 lines of code (LoC) in Java can be easily done by typing as less as just 10 LoC in Apache Pig. Ultimately Apache Pig reduces the development time by almost 16 times.
- Pig Latin is SQL-like language and it is easy to learn Apache Pig when you are familiar with SQL.
- Apache Pig provides many built-in operators to support data operations like joins, filters, ordering, etc. In addition, it also provides nested data types like tuples, bags, and maps that are missing from MapReduce.

Features of Pig

Apache Pig comes with the following features −

- **Rich set of operators** − It provides many operators to perform operations like join, sort, filer, etc.

- **Ease of programming −** Pig Latin is similar to SQL and it is easy to write a Pig script if you are good at SQL.

- **Optimization opportunities −** The tasks in Apache Pig optimize their execution automatically, so the programmers need to focus only on semantics of the language.

- **Extensibility −** Using the existing operators, users can develop their own functions to read, process, and write data.

- **UDF's −** Pig provides the facility to create User-defined Functions in other programming languages such as Java and invoke or embed them in Pig Scripts.

- **Handles all kinds of data −** Apache Pig analyzes all kinds of data, both structured as well as unstructured. It stores the results in HDFS.

Apache Pig Vs Hive

Both Apache Pig and Hive are used to create MapReduce jobs. And in some cases, Hive operates on HDFS in a similar way Apache Pig does. In the following table, we have listed a few significant points that set Apache Pig apart from Hive.

Apache Pig	Hive
Apache Pig uses a language called **Pig Latin**. It was originally created at **Yahoo**.	Hive uses a language called **HiveQL**. It was originally created at **Facebook**.
Pig Latin is a data flow language.	HiveQL is a query processing language.
Pig Latin is a procedural language and it fits in pipeline paradigm.	HiveQL is a declarative language.
Apache Pig can handle structured, unstructured, and semi-structured data.	Hive is mostly for structured data.

Pig Analytics tasks

Dataset used : NYSE_stocks.csv, NYSE_dividends.csv

Step 1: Opening Pig in Local mode

```
[cloudera@quickstart bin]$ pig -x local
log4j:WARN No appenders could be found for logger (org.apache.hadoop.util.Shell).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
614  [main] WARN  org.apache.pig.Main  - Cannot write to log file: /bin/pig_1531166349819.log
2018-07-10 01:29:09,836 [main] INFO  org.apache.pig.Main - Apache Pig version 0.12.0-cdh5.13.0 (rexported) compiled Oct 04 2017, 11:09:03
2018-07-10 01:29:09,859 [main] INFO  org.apache.pig.impl.util.Utils - Default bootup file /home/cloudera/.pigbootup not found
2018-07-10 01:29:10,034 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.defaultFS
2018-07-10 01:29:10,035 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is deprecated. Instead, use mapreduce.jobtracker.a
2018-07-10 01:29:10,040 [main] INFO  org.apache.pig.backend.hadoop.executionengine.HExecutionEngine - Connecting to hadoop file system at: file:///
2018-07-10 01:29:10,336 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - io.bytes.per.checksum is deprecated. Instead, use dfs.bytes-per-check
2018-07-10 01:29:10,431 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.defaultFS
2018-07-10 01:29:10,434 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - io.bytes.per.checksum is deprecated. Instead, use dfs.bytes-per-check
2018-07-10 01:29:10,529 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.defaultFS
2018-07-10 01:29:10,533 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - io.bytes.per.checksum is deprecated. Instead, use dfs.bytes-per-check
2018-07-10 01:29:10,606 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.defaultFS
2018-07-10 01:29:10,609 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - io.bytes.per.checksum is deprecated. Instead, use dfs.bytes-per-check
2018-07-10 01:29:10,662 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.defaultFS
2018-07-10 01:29:10,665 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - io.bytes.per.checksum is deprecated. Instead, use dfs.bytes-per-check
2018-07-10 01:29:10,709 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.defaultFS
2018-07-10 01:29:10,711 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - io.bytes.per.checksum is deprecated. Instead, use dfs.bytes-per-check
2018-07-10 01:29:10,749 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.defaultFS
```

Step 2: Loading Data from NYSE_stocks.csv dataset into Pig

```
grunt> stocks = LOAD '/home/cloudera/NYSE_stocks.csv' USING PigStorage(',') as (Exch: CHARARRAY, Symbol:CHARARRAY, Date:DATETIME, Stocks_Open:FLOAT, Stocks_H
OAT)
>> ;
```

Step 3**:** Loading Data from NYSE_dividends.csv dataset into Pig

```
grunt> dividends = LOAD '/home/cloudera/NYSE_dividends.csv' USING PigStorage(',') AS (EXCH: CHARARRAY,Symbol:CHARARRAY,Date:DATETIME,Dividents:FLOAT);
```

Step 4: Grouping Stocks Data by symbol

```
grunt> grp_sym = GROUP stocks BY Symbol;
```

Step 5: Generating Percentage return per stock

```
grunt> return = FOREACH grp_sym GENERATE group,MAX(stocks.Stocks_Close)-MIN(stocks.Stocks_Open) / MIN(stocks.Stocks_Open) * 100 as PCT_RETURN;
```

Step 6: Ordering the generated data

```
grunt> order_return = ORDER return BY PCT_RETURN DESC;
```

Step 7: Putting limit to number of generated outputs to show

```
grunt> limit_return = LIMIT order_return 10;
```

Step 8: Showing output

```
grunt> dump limt_return;
```

Step 9: Joining the two datasets

```
grunt> joining = JOIN stocks BY (Symbol,Date), dividends BY (Symbol,Date);
2018-07-09 01:55:41,864 [main] WARN  org.apache.pig.PigServer - Encountered Warning IMPLICIT_CAST_TO_FLOAT 1 time(s).
grunt> DESCRIBE joining;
joining: {stocks::Exch: chararray,stocks::Symbol: chararray,stocks::Date: datetime,stocks::Stocks_Open: float,stocks::Stocks_High: float,s
float,dividends::Exch: chararray,dividends::Symbol: chararray,dividends::Date: datetime,dividends::Dividends: float}
```

What are the challenges of using Hadoop?

- MapReduce programming is not a good match for all problems. It’s good for simple information requests and problems that can be divided into independent units, but it's not efficient for iterative and interactive analytic tasks. MapReduce is file-intensive. Because the nodes don’t intercommunicate except through sorts and shuffles, iterative algorithms require multiple map-shuffle/sort-reduce phases to complete. This creates multiple files between MapReduce phases and is inefficient for advanced analytic computing.
- There’s a widely acknowledged talent gap. It can be difficult to find entry-level programmers who have sufficient Java skills to be productive with MapReduce. That's one reason distribution providers are racing to put relational (SQL) technology on top of Hadoop. It is much easier to find programmers with SQL skills than MapReduce skills. And, Hadoop administration seems part art and part science, requiring low-level knowledge of operating systems, hardware and Hadoop kernel settings.
- Data security. Another challenge centers around the fragmented data security issues, though new tools and technologies are surfacing. The Kerberos authentication protocol is a great step toward making Hadoop environments secure.
- Full-fledged data management and governance. Hadoop does not have easy-to-use, full-feature tools for data management, data cleansing, governance and metadata. Especially lacking are tools for data quality and standardization.

Apache Spark

Apache Spark is a lightning-fast cluster computing technology, designed for fast computation. It is based on Hadoop MapReduce and it extends the MapReduce model to efficiently use it for more types of computations, which includes interactive queries and stream processing. The main feature of Spark is its in-memory cluster computing that increases the processing speed of an application.

Spark is designed to cover a wide range of workloads such as batch applications, iterative algorithms, interactive queries and streaming. Apart from supporting all these workload in a respective system, it reduces the management burden of maintaining separate tools.

Features of Apache Spark

Apache Spark has following features.

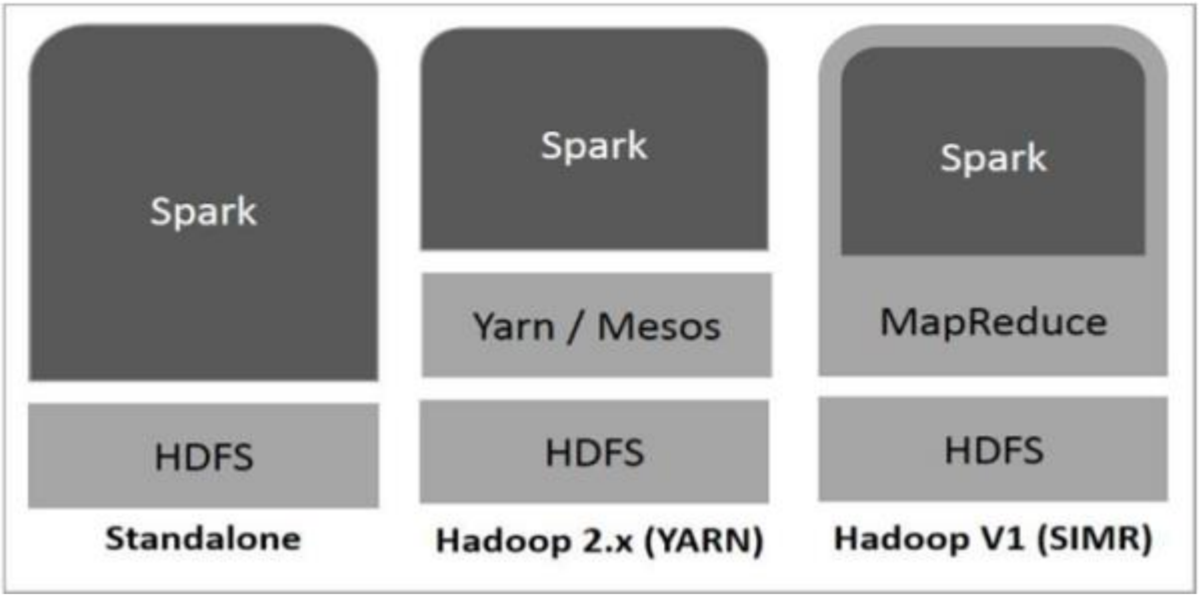
- **Speed** − Spark helps to run an application in Hadoop cluster, up to 100 times faster in memory, and 10 times faster when running on disk. This is possible by reducing number of read/write operations to disk. It stores the intermediate processing data in memory.

- **Supports multiple languages** − Spark provides built-in APIs in Java, Scala, or Python. Therefore, you can write applications in different languages. Spark comes up with 80 high-level operators for interactive querying.

- **Advanced Analytics** − Spark not only supports 'Map' and 'reduce'. It also supports SQL queries, Streaming data, Machine learning (ML), and Graph algorithms.

Spark Built on Hadoop

The following diagram shows three ways of how Spark can be built with Hadoop components.



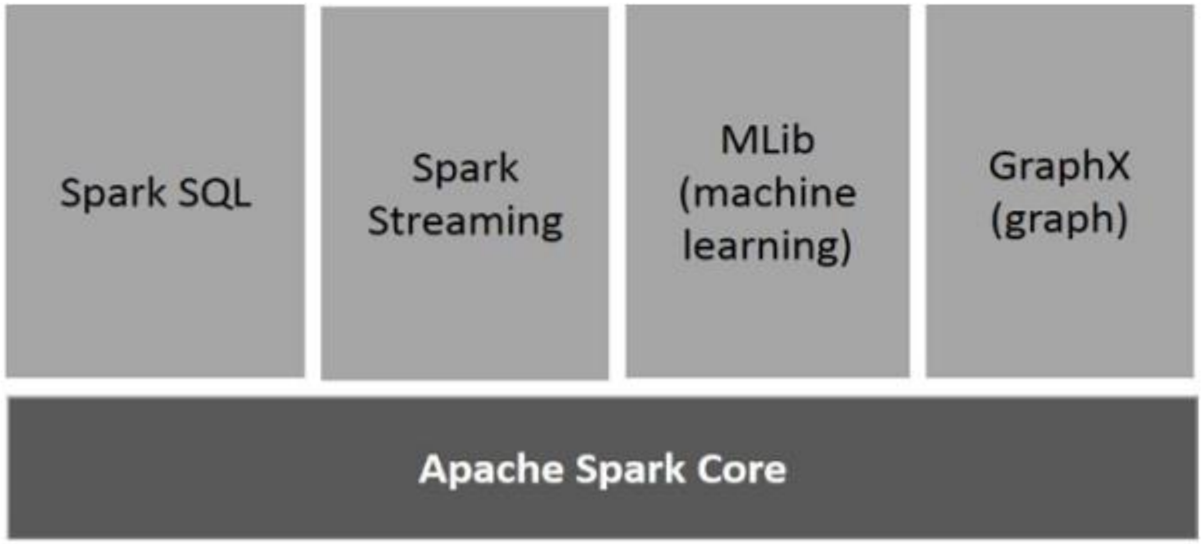
Standalone − Spark Standalone deployment means Spark occupies the place on top of HDFS (Hadoop Distributed File System) and space is allocated for HDFS, explicitly. Here, Spark and MapReduce will run side by side to cover all spark jobs on cluster.

Hadoop Yarn − Hadoop Yarn deployment means, simply, spark runs on Yarn without any pre-installation or root access required. It helps to integrate Spark into Hadoop ecosystem or Hadoop stack. It allows other components to run on top of stack.

Spark in MapReduce (SIMR) − Spark in MapReduce is used to launch spark job in addition to standalone deployment. With SIMR, user can start Spark and uses its shell without any administrative access.

Components of Spark

The following illustration depicts the different components of Spark.



Apache Spark Core

Spark Core is the underlying general execution engine for spark platform that all other functionality is built upon. It provides In-Memory computing and referencing datasets in external storage systems.

Spark SQL

Spark SQL is a component on top of Spark Core that introduces a new data abstraction called SchemaRDD, which provides support for structured and semi-structured data.

Spark Streaming

Spark Streaming leverages Spark Core's fast scheduling capability to perform streaming analytics. It ingests data in mini-batches and performs RDD (Resilient Distributed Datasets) transformations on those mini-batches of data.

MLlib (Machine Learning Library)

MLlib is a distributed machine learning framework above Spark because of the distributed memory-based Spark architecture. It is, according to benchmarks, done by the MLlib developers against the Alternating Least Squares (ALS) implementations. Spark MLlib is nine times as fast as the Hadoop disk-based version of Apache Mahout (before Mahout gained a Spark interface).

GraphX

GraphX is a distributed graph-processing framework on top of Spark. It provides an API for expressing graph computation that can model the user-defined graphs by using Pregel abstraction API. It also provides an optimized runtime for this abstraction.

Resilient Distributed Datasets (RDD)

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. RDDs can contain any type of Python, Java, or Scala objects, including user-defined classes.

Formally, an RDD is a read-only, partitioned collection of records. RDDs can be created through deterministic operations on either data on stable storage or other RDDs. RDD is a fault-tolerant collection of elements that can be operated on in parallel.

There are two ways to create RDDs − parallelizing an existing collection in your driver program, or referencing a dataset in an external storage system, such as a shared file system, HDFS, HBase, or any data source offering a Hadoop Input Format.

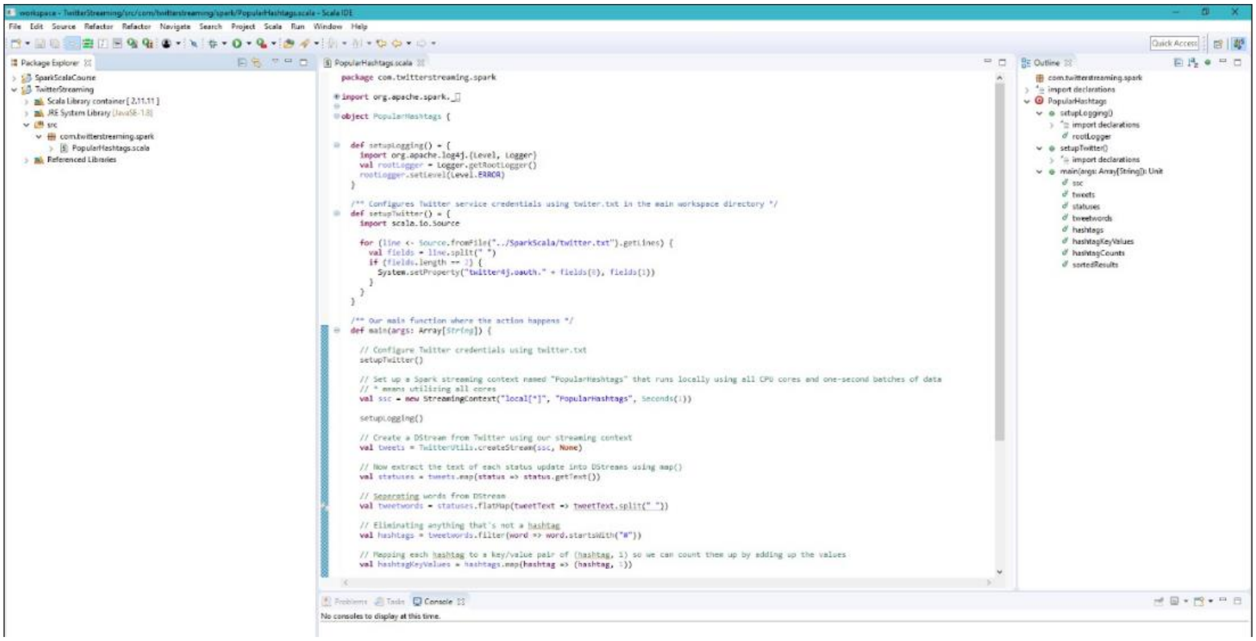
Spark makes use of the concept of RDD to achieve faster and efficient MapReduce operations. Let us first discuss how MapReduce operations take place and why they are not so efficient.

Twitter’s Top 10 popular Hashtag Streaming per second using Apache Spark

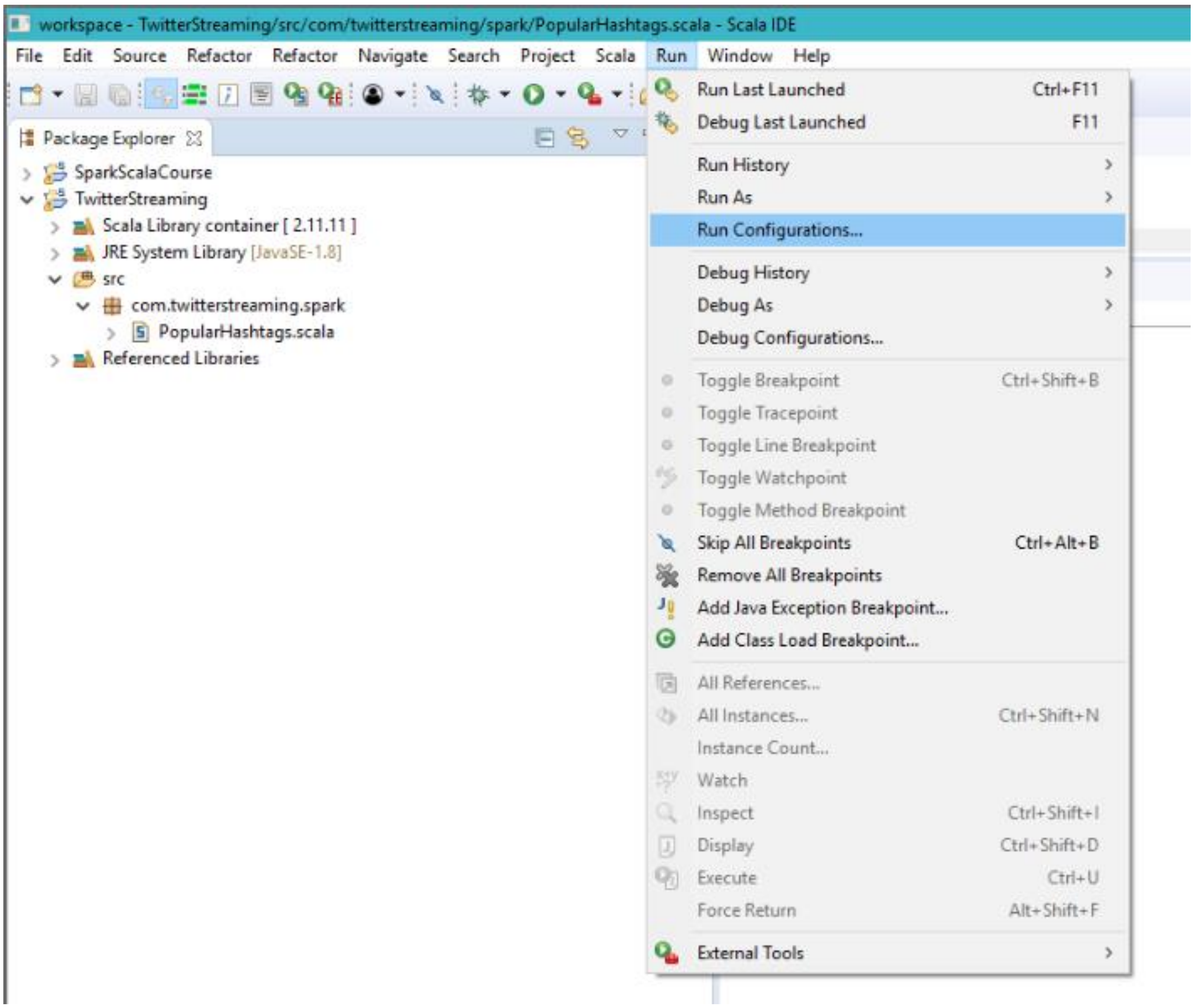
Technologies used –

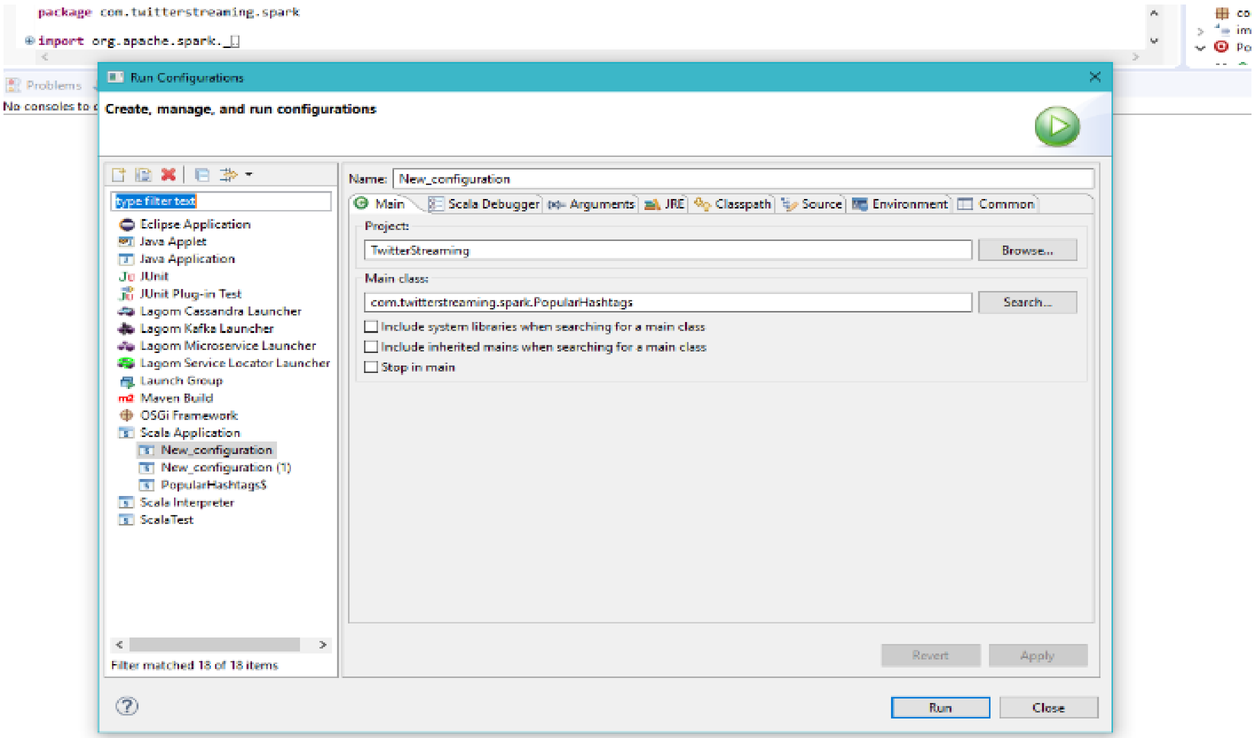
- Spark
- Scala 2.11.11
- Libraries used : dsstream-twitter_2.11-0.1.0-SNAPSHOT, twitter4j-core-4.0.4, twitter4j-stream-4.0.4
- Eclipse Scala IDE

Scala Code:



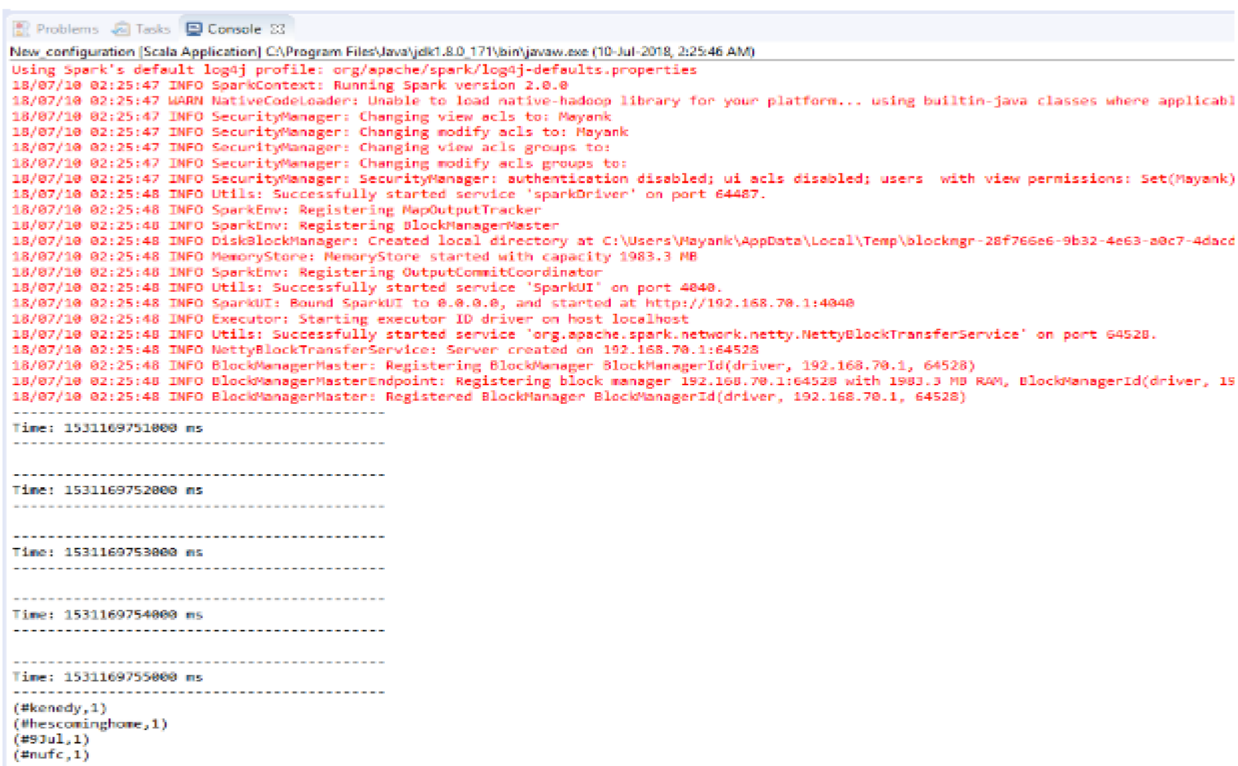
Running Script:





package com.twitterstreaming.spark
import org.apache.spark._
Run Configurations
Create, manage, and run configurations
type filter text
Eclipse Application
Java Applet
Java Application
JUnit
JUnit Plug-in Test
Lagom Cassandra Launcher
Lagom Kafka Launcher
Lagom Microservice Launcher
Lagom Service Locator Launcher
Launch Group
Maven Build
OSGi Framework
Scala Application
New_configuration
New_configuration (1)
PopularHashtags$
Scala Interpreter
ScalaTest
Filter matched 18 of 18 items
Name: New_configuration
Main
Scala Debugger
Arguments
JRE
Classpath
Source
Environment
Common
Project:
TwitterStreaming
Browse...
Main class:
com.twitterstreaming.spark.PopularHashtags
Search...
Include system libraries when searching for a main class
Include inherited mains when searching for a main class
Stop in main
Revert
Apply
Run
Close

Output:



Problems
Tasks
Console
New_configuration [Scala Application] C:\Program Files\Java\jdk1.8.0_171\bin\javaw.exe (10-Jul-2018, 2:25:46 AM)
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
18/07/10 02:25:47 INFO SparkContext: Running Spark version 2.0.0
18/07/10 02:25:47 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicabl
18/07/10 02:25:47 INFO SecurityManager: Changing view acls to: Mayank
18/07/10 02:25:47 INFO SecurityManager: Changing modify acls to: Mayank
18/07/10 02:25:47 INFO SecurityManager: Changing view acls groups to:
18/07/10 02:25:47 INFO SecurityManager: Changing modify acls groups to:
18/07/10 02:25:47 INFO SecurityManager: SecurityManager: authentication disabled; ui acls disabled; users with view permissions: Set(Mayank)
18/07/10 02:25:48 INFO Utils: Successfully started service 'sparkDriver' on port 64487.
18/07/10 02:25:48 INFO SparkEnv: Registering MapOutputTracker
18/07/10 02:25:48 INFO SparkEnv: Registering BlockManagerMaster
18/07/10 02:25:48 INFO DiskBlockManager: Created local directory at C:\Users\Mayank\AppData\Local\Temp\blockmgr-28f766e6-9b32-4e63-a0c7-4dacd
18/07/10 02:25:48 INFO MemoryStore: MemoryStore started with capacity 1983.3 MB
18/07/10 02:25:48 INFO SparkEnv: Registering OutputCommitCoordinator
18/07/10 02:25:48 INFO Utils: Successfully started service 'SparkUI' on port 4040.
18/07/10 02:25:48 INFO SparkUI: Bound SparkUI to 0.0.0.0, and started at http://192.168.70.1:4040
18/07/10 02:25:48 INFO Executor: Starting executor ID driver on host localhost
18/07/10 02:25:48 INFO Utils: Successfully started service 'org.apache.spark.network.netty.NettyBlockTransferService' on port 64528.
18/07/10 02:25:48 INFO NettyBlockTransferService: Server created on 192.168.70.1:64528
18/07/10 02:25:48 INFO BlockManagerMaster: Registering BlockManager BlockManagerId(driver, 192.168.70.1, 64528)
18/07/10 02:25:48 INFO BlockManagerMasterEndpoint: Registering block manager 192.168.70.1:64528 with 1983.3 MB RAM, BlockManagerId(driver, 19
18/07/10 02:25:48 INFO BlockManagerMaster: Registered BlockManager BlockManagerId(driver, 192.168.70.1, 64528)

Time: 1531169751000 ms

Time: 1531169752000 ms

Time: 1531169753000 ms

Time: 1531169754000 ms

Time: 1531169755000 ms

(#kenedy,1)
(#hescominghome,1)
(#9Jul,1)
(#nufc,1)

```
(#PachaMama,1)
(#happygrandpa,1)
(#weareacmilan,1)
...

-------------------------------------------
Time: 1531169768000 ms
-------------------------------------------
(#Brexit!,1)
(#ForzaMilan,1)
(#deixaelastrabalhar,1)
(#LunesDeGanarSeguidores,1)
(#Ty,1)
(#Dominos,1)
(#طيف_الورد_للدعم,1)
(#PachaMama,1)
(#happygrandpa,1)
(#weareacmilan,1)
...

-------------------------------------------
Time: 1531169769000 ms
-------------------------------------------
(#MasterChef,2)
(#Brexit!,1)
(#ForzaMilan,1)
(#deixaelastrabalhar,1)
(#LunesDeGanarSeguidores,1)
(#Ty,1)
(#Dominos,1)
(#طيف_الورد_للدعم,1)
(#PachaMama,1)
(#happygrandpa,1)
...

-------------------------------------------
Time: 1531169770000 ms
-------------------------------------------
(#MasterChef,2)
(#Brexit!,1)
(#ForzaMilan,1)
(#deixaelastrabalhar,1)
(#□□□□,1)
(#LunesDeGanarSeguidores,1)
(#Ty,1)
(#Dominos,1)
```

Twitter.txt file:

twitter - Notepad

File Edit Format View Help

```
consumerKey 061iS5x7[redacted]
consumerSecret i[redacted]WbwerFYVDIKAyCfovs8Y
accessToken [redacted]czxGSvzCKd6kDOFEcaYjMtz1p
accessTokenSecret [redacted]kreoo9m
```