**PRACTICAL 1: TO STUDY ABOUT BIG DATA ANALYTICS**

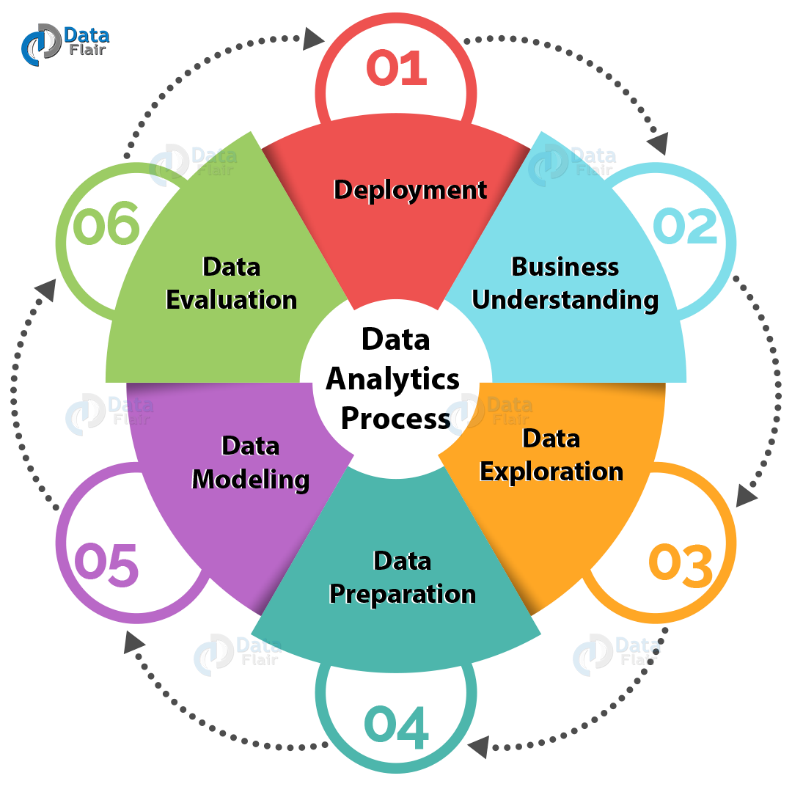
Big Data and Machine Learning have become the reason behind the success of various industries. Both these technologies are becoming popular day by day among all data scientists and professionals. **Big data is a term that is used to describe large, hard-to-manage, structured, and unstructured voluminous data. Whereas, Machine learning is a subfield of Artificial Intelligence that enables machines to automatically learn and improve from experience/past data.**

Big data analytics is the use of advanced analytic techniques against very large, diverse data sets that include structured, semi-structured and unstructured data, from different sources, and in different sizes from terabytes to zettabytes.

Data is information in raw format. With increasing data size, it has become need for inspecting, cleaning, transforming, and modeling data with the goal of finding useful information, making conclusions, and supporting decision making. This process is known as Big Data data analysis.

Data mining is a particular data analysis technique where modeling and knowledge discovery for predictive rather than purely descriptive purposes is focused. Business intelligence covers data analysis that relies heavily on aggregation, focusing on business information. In statistical applications, some people divide business analytics into descriptive statistics, exploratory data analysis (EDA), and confirmatory data analysis (CDA). EDA focuses on discovering new features in the data and CDA focuses on confirming or falsifying existing hypotheses. Predictive analytics does forecasting or classification by focusing on statistical or structural models while in text analytics, statistical, linguistic and structural techniques are applied to extract and classify information from textual sources, a species of unstructured data. All are varieties of data analysis.

**DATA ANALYTICS PROCESS**



**a. Business Understanding**

The very first step consists of business understanding. Whenever any requirement occurs, firstly we need to determine business objective, assess the situation, determine data mining goals and then produce the project plan as per the requirement. Business objectives are defined in this phase.

**b. Data Exploration**

Second step consists of Data understanding. For further process, we need to gather initial data, describe and explore the data and verify data quality to ensure it contains the data we require. Data collected from the various sources is described in terms of its application and need for the project in this phase. This is also known as data exploration. This is necessary to verify the quality of data collected.

**c. Data Preparation**

Next come Data preparation. From the data collected in last step, we need to select data as per the need, clean it, construct it to get useful information and then integrate it all. Finally we need to format the data to get appropriate data. Data is selected, cleaned, and integrated in the format finalized for the analysis in this phase.

**d. Data Modeling**

Once data is gathered, we need to do data modeling. For this, we need to select modeling technique, generate test design, build model and assess the model built. Data model is build to analyze relationships between various selected objects in the data, test cases are built for assessing the model and model is tested and implemented on the data in this phase.

**e. Data Evaluation**

Next come data evaluation where we evaluate the results generated in last step, review the scope of error and determine next steps that need to be performed. Results of the test cases are evaluated and reviewed for the scope of error in this phase.

**f. Deployment**

Final step in analytic process is deployment. Here we need to plan the deployment and monitoring and maintenance, we need to produce final report and review the project. Results of the analysis are deployed in this phase. This is also known as reviewing of the project.

The complete above process is known as business analytics proc

**PRACTICAL 2: TO STUDY ABOUT BIG DATA STORAGE**

Big data storage is a compute-and-storage architecture you can use to collect and manage huge-scale datasets and perform real-time data analyses. These analyses can then be used to generate intelligence from metadata.

Typically, big data storage is composed of hard disk drives due to the media’s lower cost. However, flash storage is gaining popularity due to its decreasing cost. When flash is used, systems can be built purely on flash media or can be built as hybrids of flash and disk storage.

Data within big data datasets is unstructured. To accommodate this, big data storage is usually built with object and file-based storage. These storage types are not restricted to specific capacities and typically volumes scale to terabyte or petabyte sizes.

**BIG DATA STORAGE CHALLENGES**

When configuring and implementing big data storage there are a few common challenges you might encounter. All these challenges take different shape when running on the public cloud vs. on-premises storage.

1. Size and storage costs

Big data grows geometrically, requiring substantial storage space. As data sources are added, these demands increase further and need to be accounted for. When implementing big data storage, you need to ensure that it is capable of scaling at the same rate as your data collection.

1. Data transfer rates

When you need to transfer large volumes of data, high transfer rates are key. In big data environments, data scientists must be able to move data quickly from primary sources to their analysis environment.

1. Security

Big data frequently contains sensitive data, such as personally identifiable information (PII) or financial data. This makes data a prime target for criminals and a liability if left unprotected. Even unintentional corruption of data can have significant consequences.

1. High availability

Regardless of what resources are used, you need to ensure that data remains highly available. You should have measures in place to deal with infrastructure failures. You also need to ensure that you can reliably and efficiently retrieve archived data.

**PRACTICAL 3: TO STUDY ABOUT HDFS/GFS**

**MAIN GOAL OF GFS AND HDFS**

The HDFS and GFS were built to support large files coming from various sources and in a variety of formats. Huge data storage size (Peta bytes) are distributed across thousands of disks attached to commodity hardware. Both HDFS and GFS are designed for data-intensive computing and not for normal end-users[1](https://scialert.net/fulltext/?doi=rjit.2016.66.74#1047083_ja). Data-intensive computing is a class of parallel computing used to process and analyze large amount of data referred to as big data. Data reliability is achieved through distribution architecture, even when failures occur within chunk servers, master or network partitions. There is no limit to the cluster that you can have. The size can be increased anytime as per the need.

**GOOGLE FILE SYSTEM (GFS)**

**Google File System (GFS)** is a scalable distributed file system (DFS) created by Google Inc. and developed to accommodate Google’s expanding data processing requirements. GFS provides fault tolerance, reliability, scalability, availability and performance to large networks and connected nodes. GFS is made up of several storage systems built from low-cost commodity hardware components. It is optimized to accomodate Google's different data use and storage needs, such as its search engine, which generates huge amounts of data that must be stored.

The Google File System capitalized on the strength of off-the-shelf servers while minimizing hardware weaknesses.

The GFS node cluster is a single master with multiple chunk servers that are continuously accessed by different client systems. Chunk servers store data as Linux files on local disks. Stored data is divided into large chunks (64 MB), which are replicated in the network a minimum of three times. The large chunk size reduces network overhead.

GFS is designed to accommodate Google’s large cluster requirements without burdening applications. Files are stored in hierarchical directories identified by path names. Metadata - such as namespace, access control data, and mapping information - is controlled by the master, which interacts with and monitors the status updates of each chunk server through timed heartbeat messages.

GFS features include:

* Fault tolerance
* Critical data replication
* Automatic and efficient data recovery
* High aggregate throughput
* Reduced client and master interaction because of large chunk server size
* Namespace management and locking
* High availability

The largest GFS clusters have more than 1,000 nodes with 300 TB disk storage capacity. This can be accessed by hundreds of clients on a continuous basis.

**HADOOP DISTRIBUTED FILE SYSTEM**

First of all, it should by clearly state that Hadoop has Google origins. Based on three white papers published by Google, which are: "Google file system[2](https://scialert.net/fulltext/?doi=rjit.2016.66.74#24047_con)", "MapReduce: Simplified data processing on large clusters[3](https://scialert.net/fulltext/?doi=rjit.2016.66.74#49521_con)" and "Bigtable: A distributed storage system for structured data[4](https://scialert.net/fulltext/?doi=rjit.2016.66.74#9904_op)", Apache developed Apache HDFS, Apache MapReduce and Apache HBase, respectively. Almost 95% of the architecture described in these three white papers is implemented in Apache projects with some minor differences. Google released these white papers with no code. So, it was up to engineers and scientists at Apache to design and implement the architecture.

**COMPARISION BETWEEN GFS AND HDFS**

**Scalability:** Both HDFS and GFS are considered as cluster based architecture. Each file system runs over machines built from commodity hardware. Each cluster may consist of thousands of nodes with huge data size storage.

**Implementation:** Since GFS is proprietary file system and exclusive to Google only, it can not be used by any other company.

In the other part, HDFS based on Apache Hadoop open-source project can be deployed and used by any company willing to manage and process big data.

Yahoo! might be the most famous example where clusters are managed by Hadoop with HDFS file system inside. Yahoo! has more than 100,000 CPU in 40,000 computers running Hadoop. Their biggest cluster contains around 4500 nodes.

Facebook uses Hadoop to store copies of internal logs and dimension data sources and uses it as a source for reporting/analytics and machine learning. They currently have two major clusters:

|  |  |
| --- | --- |
| • | A 1100-machine cluster with 8800 cores and about 12 Petabytes raw storage |
| • | A 300-machine cluster with 2400 cores and about 3 Petabytes raw storage |

EBay uses Apache on 532-machine cluster with Apache HBase for search optimization and research.

Twitter, LinkedIn, Adobe, A9.com (Amazon) and many other websites use Hadoop to store and process data logs, batch jobs, processes for internal usage and structured data storage on hundreds of clusters with thousands of nodes each.

**File serving:** In GFS, files are divided into units called chunks of fixed size. Chunk size is 64 MB and can be stored on different nodes in cluster for load balancing and performance needs. In Hadoop, HDFS file system divides the files into units called blocks of 128 MB in size[5](https://scialert.net/fulltext/?doi=rjit.2016.66.74#75253_con). Block size can be adjustable based on the size of data.

**Internal communication:** Communication between chucks and clusters within GFS is made through TCP connections. For data transfer, pipelining is used over TCP connections. The same method is in HDFS, but Remote Procedure Call (RPC) are used to conduct external communication between clusters and blocks.

**Cache management:** In GFS, cache metadata are saved in client memory. Chunk server does not need cache file data. Linux system running on the chunk server caches frequently accessed data in memory.

The HDFS has "DistributedCache". DistributedCache is facility provided by the MapReduce to distribute application-specific, large, read-only files efficiently. It also caches files such as text, archives (zip, tar, tgz and tar.gz) and jars needed by applications.

DistributedCache files can be private or public, that determines how they can be shared on the slave nodes.

"Private" DistributedCache files are cached in a local directory private to the user whose jobs need these files.

"Public" DistributedCache files are cached in a global directory and the file access is setup in sucha way that they are publicly visible to all users.

**Files protection and permission:** Suitebriar-Google partner-mentions in its security analysis research that GFS splits files up and stores it in multiple pieces on multiple machines. File names have random names and are not human readable. Files are obfuscated through algorithms that change constantly. The HDFS implements POSIX-like mode permission for files and directories. All files and directories are associated with an owner and a group with separate permissions for users who are owners, for users that are members of the group and for all other users.

**Replication strategy:** The GFS has two replicas: Primary replicas and secondary replicas.

A primary replica is the data chunk that a chunk server sends to a client.

Secondary replicas serve as backups on other chunk servers. User can specify the number of replicas to be maintained.

The HDFS has an automatic replication rack based system. By default two copies of each block are stored by different DataNodes in the same rack and a third copy is stored on a DataNode on a different rack.

**File namespace:**

|  |  |
| --- | --- |
| • | In GFS, files are organized hierarchically in directories and identified by path names |
| • | The GFS is exclusively for Google only |
| • | The HDFS supports a traditional hierarchical file organization |
| • | Users or application can create directories to store files inside |
| • | The HDFS also supports third-party file systems such as CloudStore and Amazon Simple Storage Service (S3) |

**PRACTICAL 4: TO STUDY ABOUT HADOOP AND ITS TOOLS**

Hadoop is **an open source, Java based framework used for storing and processing big data**. The data is stored on inexpensive commodity servers that run as clusters. Its distributed file system enables concurrent processing and fault tolerance. Developed by Doug Cutting and Michael J.

Hadoop tutorial provides basic and advanced concepts of Hadoop. Our Hadoop tutorial is designed for beginners and professionals.

Hadoop is an open source framework. It is provided by Apache to process and analyze very huge volume of data. It is written in Java and currently used by Google, Facebook, LinkedIn, Yahoo, Twitter etc.

Our Hadoop tutorial includes all topics of Big Data Hadoop with HDFS, MapReduce, Yarn, Hive, HBase, Pig, Sqoop etc.

## Modules of Hadoop

1. **HDFS:** Hadoop Distributed File System. Google published its paper GFS and on the basis of that HDFS was developed. It states that the files will be broken into blocks and stored in nodes over the distributed architecture.
2. **Yarn:** Yet another Resource Negotiator is used for job scheduling and manage the cluster.
3. **Map Reduce:** This is a framework which helps Java programs to do the parallel computation on data using key value pair. The Map task takes input data and converts it into a data set which can be computed in Key value pair. The output of Map task is consumed by reduce task and then the out of reducer gives the desired result.
4. **Hadoop Common:** These Java libraries are used to start Hadoop and are used by other Hadoop modules.

# HADOOP ANALYTICS TOOLS FOR BIG DATA

### **1. Apache Spark**

Apache spark in an open-source processing engine that is designed for ease of analytics operations. It is a cluster computing platform that is designed to be fast and made for general purpose uses. Spark is designed to cover various batch applications, [Machine Learning](https://www.geeksforgeeks.org/machine-learning/), streaming data processing, and interactive queries.

**Features of Spark:**

* In memory processing
* Tight Integration Of component
* Easy and In-expensive
* The powerful processing engine makes it so fast
* Spark Streaming has high level library for streaming process

### **2. Map Reduce**

MapReduce is just like an Algorithm or a data structure that is based on the YARN framework. The primary feature of MapReduce is to perform the distributed processing in parallel in a Hadoop cluster, which Makes Hadoop working so fast Because when we are dealing with Big Data, serial processing is no more of any use.

**Features of Map-Reduce:**

* Scalable
* Fault Tolerance
* Paraller Processing
* Tunable Replication
* Load Balancing

### **3. Apache Hive**

Apache Hive is a Data warehousing tool that is built on top of the Hadoop, and Data Warehousing is nothing but storing the data at a fixed location generated from various sources. Hive is one of the best tools used for data analysis on Hadoop. The one who is having knowledge of SQL can comfortably use Apache Hive. The query language of high is known as HQL or HIVEQL.

**Features of Hive:**

* Queries are similar to SQL queries.
* Hive has different storage type HBase, ORC, Plain text, etc.
* Hive has in-built function for data-mining and other works.
* Hive operates on compressed data that is present inside Hadoop Ecosystem.

### **4. Apache Pig**

This Pig was Initially developed by Yahoo to get ease in programming. Apache Pig has the capability to process an extensive dataset as it works on top of the Hadoop. Apache pig is used for analyzing more massive datasets by representing them as dataflow. Apache Pig also raises the level of abstraction for processing enormous datasets. Pig Latin is the scripting language that the developer uses for working on the Pig framework that runs on Pig runtime.

**Features of Pig:**

* Easy To Programme
* Rich set of operators
* Ability to handle various kind of data
* Extensibility

### **5. HBase**

HBase is nothing but a non-relational, NoSQL distributed, and column-oriented database. HBase consists of various tables where each table has multiple numbers of data rows. These rows will have multiple numbers of column family’s, and this column family will have columns that contain key-value pairs. HBase works on the top of HDFS(Hadoop Distributed File System). We use HBase for searching small size data from the more massive datasets.

**Features of HBase:**

* HBase has Linear and Modular Scalability
* JAVA API can easily be used for client access
* Block cache for real time data queries

### **6. Apache Sqoop**

Sqoop is a command-line tool that is developed by Apache. The primary purpose of Apache Sqoop is to import structured data i.e., [RDBMS](https://www.geeksforgeeks.org/rdbms-full-form/)(Relational database management System) like MySQL, SQL Server, Oracle to our HDFS(Hadoop Distributed File System). Sqoop can also export the data from our HDFS to RDBMS.

**Features of Sqoop:**

* Sqoop can Import Data To Hive or HBase
* Connecting to database server
* Controlling parallelism

**PRACTICAL 5: TO STUDY ABOUT INSTALLATION OF HADOOP**

### **Step 1:** [Click here](https://goo.gl/ipdJJa) to download the Java 8 Package. Save this file in your home directory.

### **Step 2:** Extract the Java Tar File.

**Command:** tar -xvf jdk-8u101-linux-i586.tar.gz

Untar Java - Install Hadoop - Edureka

Fig: Hadoop Installation – Extracting Java Files

### **Step 3:** Download the Hadoop 2.7.3 Package.

**Command:** wget https://archive.apache.org/dist/hadoop/core/hadoop-2.7.3/hadoop-2.7.3.tar.gz

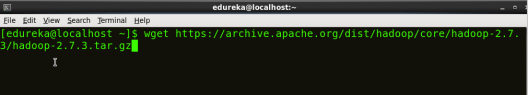


Fig: Hadoop Installation – Downloading Hadoop

### **Step 4:** Extract the Hadoop tar File.

**Command**: tar -xvf hadoop-2.7.3.tar.gz

Extract Hadoop Package - Install Hadoop - Edureka

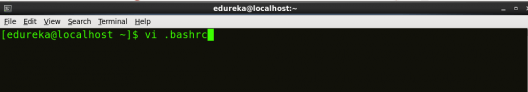
Fig: Hadoop Installation – Extracting Hadoop Files

### **Step 5:** Add the Hadoop and Java paths in the bash file (.bashrc).

Open**.** **bashrc** file. Now, add Hadoop and Java Path as shown below.

Learn more about the Hadoop Ecosystem and its tools with the [Hadoop Certification](https://www.edureka.co/big-data-hadoop-training-certification).

**Command:**  vi .bashrc



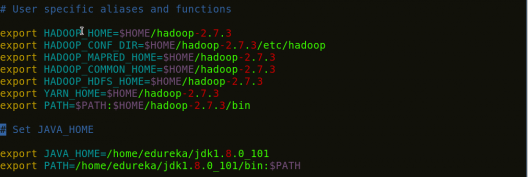


Fig: Hadoop Installation – Setting Environment Variable

Then, save the bash file and close it.

For applying all these changes to the current Terminal, execute the source command.

**Command:** source .bashrc

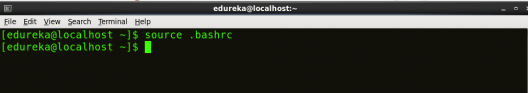


Fig: Hadoop Installation – Refreshing environment variables

To make sure that Java and Hadoop have been properly installed on your system and can be accessed through the Terminal, execute the java -version and hadoop version commands.

**Command:** java -version

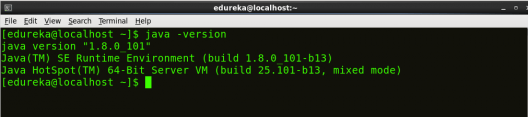


Fig: Hadoop Installation – Checking Java Version

**Command:** hadoop version

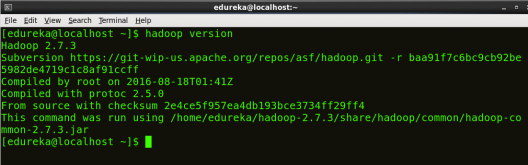


Fig: Hadoop Installation – Checking Hadoop Version

### **Step 6:** Edit the [**Hadoop Configuration files**](https://www.edureka.co/blog/explaining-hadoop-configuration/).

***Command:*** cd hadoop-2.7.3/etc/hadoop/

***Command:*** ls

All the Hadoop configuration files are located in **hadoop-2.7.3/etc/hadoop** directory as you can see in the snapshot below:

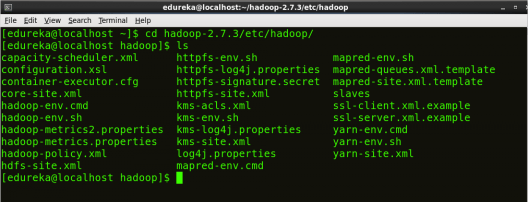
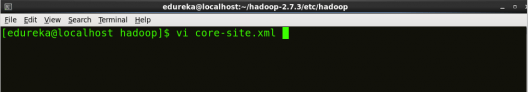


Fig: Hadoop Installation – Hadoop Configuration Files

### **Step 7:** Open core-site.xml and edit the property mentioned below inside configuration tag:

core-site.xml informs Hadoop daemon where NameNode runs in the cluster. It contains configuration settings of Hadoop core such as I/O settings that are common to HDFS & MapReduce.

**Command:** vi core-site.xml



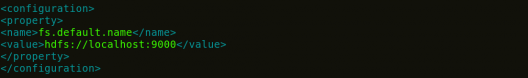


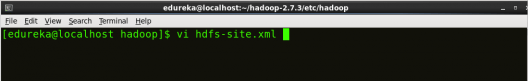
Fig: Hadoop Installation – Configuring core-site.xml

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | <?xml version="1.0" encoding="UTF-8"?>  <?xml-stylesheet type="text/xsl" href="configuration.xsl"?>  <configuration>  <property>  <name>fs.default.name</name>  <value>hdfs://localhost:9000</value>  </property>  </configuration> |

### **Step 8:** Edit hdfs-site.xml and edit the property mentioned below inside configuration tag:

hdfs-site.xml contains configuration settings of HDFS daemons (i.e. NameNode, DataNode, Secondary NameNode). It also includes the replication factor and block size of HDFS.

**Command:** vi hdfs-site.xml



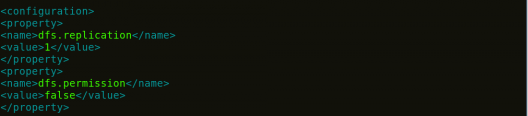


Fig: Hadoop Installation – Configuring hdfs-site.xml

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | <?xml version="1.0" encoding="UTF-8"?>  <?xml-stylesheet type="text/xsl" href="configuration.xsl"?>  <configuration>  <property>  <name>dfs.replication</name>  <value>1</value>  </property>  <property>  <name>dfs.permission</name>  <value>false</value>  </property>  </configuration> |

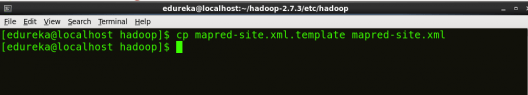
### **Step 9:** Edit the mapred-site.xml file and edit the property mentioned below inside configuration tag:

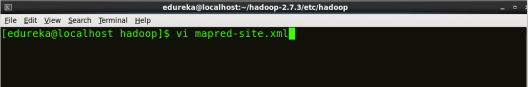
mapred-site.xml contains configuration settings of MapReduce application like number of JVM that can run in parallel, the size of the mapper and the reducer process,  CPU cores available for a process, etc.

In some cases, mapred-site.xml file is not available. So, we have to create the mapred-site.xml file using mapred-site.xml template.

**Command:** cp mapred-site.xml.template mapred-site.xml

**Command:** vi mapred-site.xml.





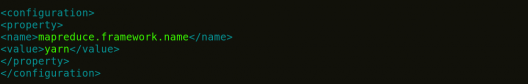


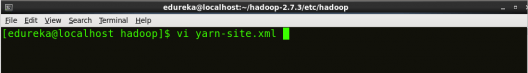
Fig: Hadoop Installation – Configuring mapred-site.xml

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | <?xml version="1.0" encoding="UTF-8"?>  <?xml-stylesheet type="text/xsl" href="configuration.xsl"?>  <configuration>  <property>  <name>mapreduce.framework.name</name>  <value>yarn</value>  </property>  </configuration> |

### **Step 10:** Edit yarn-site.xml and edit the property mentioned below inside configuration tag:

yarn-site.xml contains configuration settings of ResourceManager and NodeManager like application memory management size, the operation needed on program & algorithm, etc.

**Command:** vi yarn-site.xml



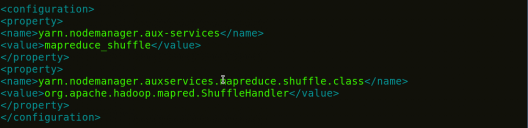


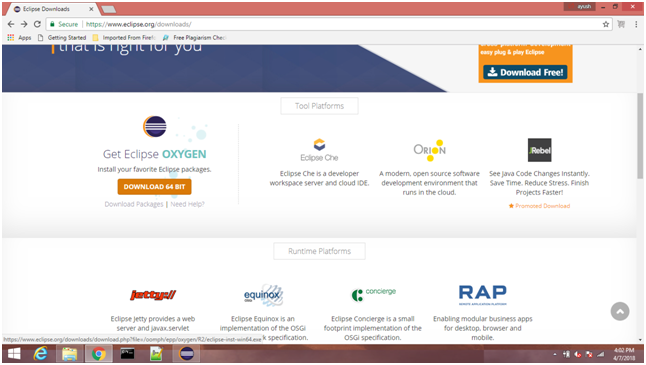
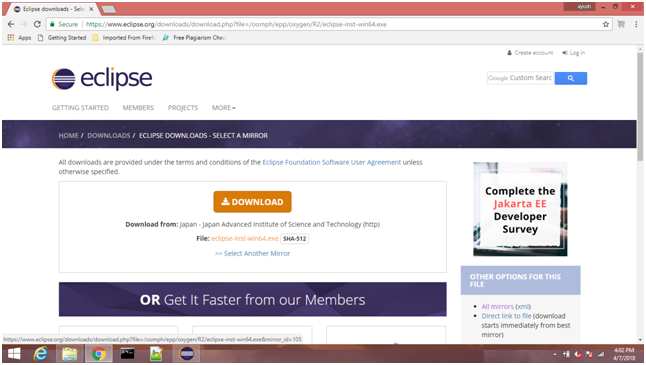
Fig: Hadoop Installation – Configuring yarn-site.xml

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | <?xml version="1.0">  <configuration>  <property>  <name>yarn.nodemanager.aux-services</name>  <value>mapreduce\_shuffle</value>  </property>  <property>  <name>yarn.nodemanager.auxservices.mapreduce.shuffle.class</name>  <value>org.apache.hadoop.mapred.ShuffleHandler</value>  </property>  </configuration> |

**PRACTICAL 6: TO STUDY ABOUT INSTALLATION OF ECLIPSE**

### Step 1: Download the Latest version

Click the link [**Download Eclipse**](https://www.eclipse.org/downloads/) to visit the download page of eclipse. You can download the latest version of eclipse i.e. eclipse oxygen from that page. The opened page will look like following, click on **DOWNLOAD 64 BIT** to proceed the download.

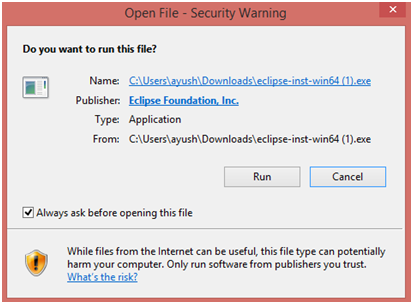
  


Eclipse oxygen 64 bit installer will be installed on our system. Here, we are using Windows operating system therefore the downloaded file will be different from the one which needs to be installed on Linux based systems.

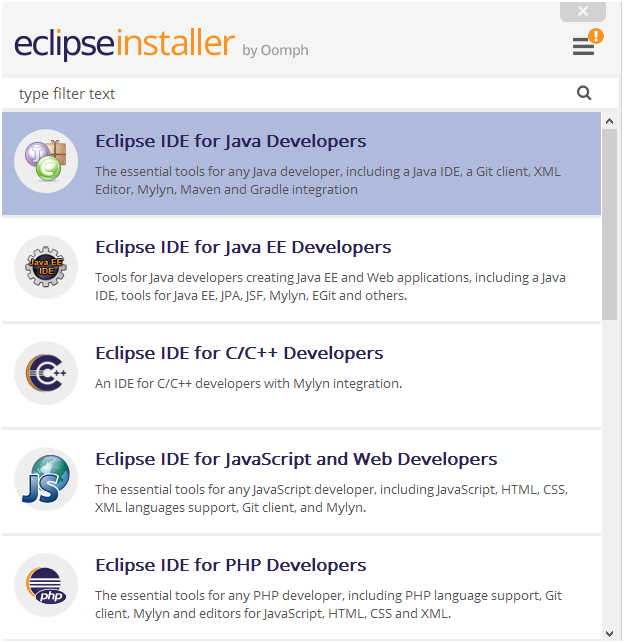
Just Click on **DOWNLOAD** button to download the installer.

### Step 2: Install Eclipse

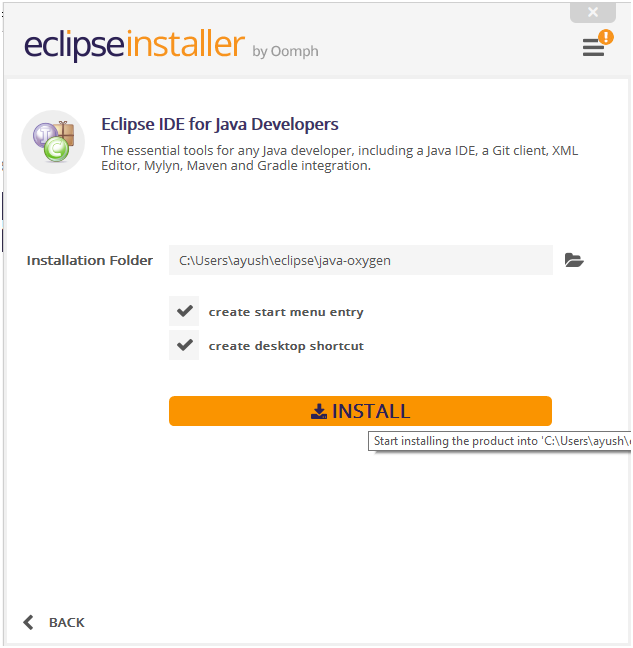
Double click on the **exe** file which has just been downloaded. The screen will look like following. Click **Run** to proceed the installation.



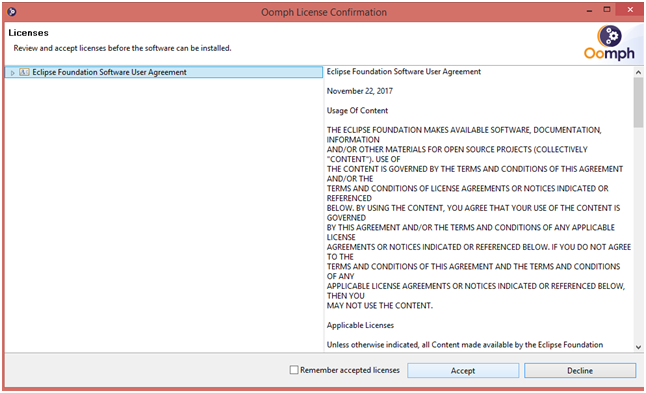
Choose the software suit which you want to install. In our case, we have chosen **Eclipse IDE for Java Developers** which is recommended in our case.



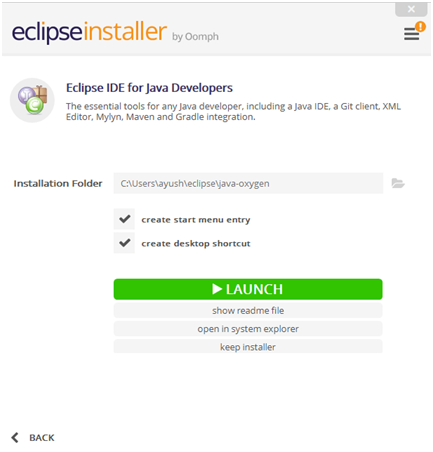
Now, the Set up is ready to install Eclipse oxygen 64 bit in the directory shown in the image. However, we can select any destination folder present on our system. Just click install when you done with the directory selection.



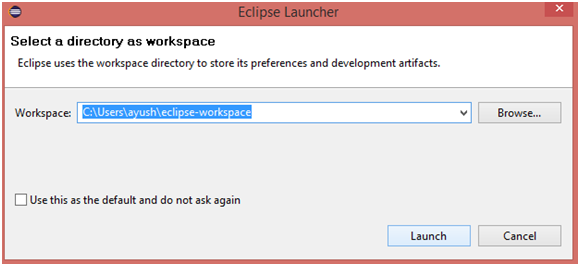
The set up will ask us to accept the Eclipse Foundation Software Agreement. Just click **Accept** to continue.



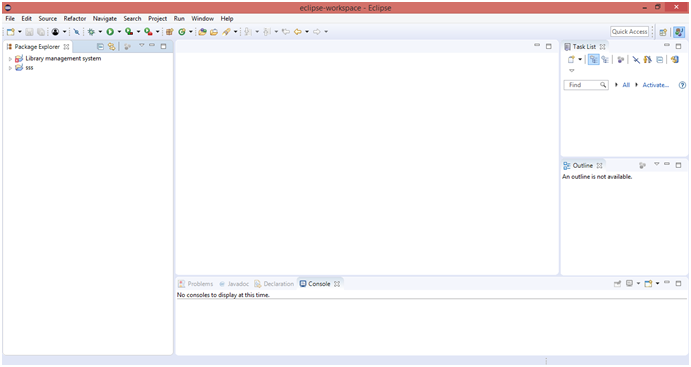
Now, we will have to wait for the time the Eclipse will be installing on our system. Once the installation will be done, the following screen will appear. Just click the **LAUNCH** button to launch eclipse.



Browse the workspace directory, just click the **Launch** button once you done with the process.



We have got the Eclipse IDE opened on our system. However,the screen will appear like following. Now, we are all set to configure Eclipse in order to run the JavaFX application.



**PRACTICAL 7:** **TO STUDY ABOUT INSTALLATION OF CLOUDERA**

1. Prepare to install and configure the Cloudera Manager packages. For more information, see [Configuring Repository](https://docs.cloudera.com/cdp-private-cloud-base/7.1.7/installation/topics/cdpdc-configure-repository.html). Do this if you have not done it already. Confirm that the repo is set up.
2. Install Cloudera Manager Server. For more information on installing Cloudera Manager Server, see [Installing Cloudera Manager](https://docs.cloudera.com/cdp-private-cloud-base/7.1.7/installation/topics/cdpdc-install-cm-server.html).
3. Preconfigure the databases for:
   * Ranger
   * Cloudera Manager Server
   * Cloudera Management Service roles - Reports Manager
   * Data Analytics Studio (DAS) Supported with PostgreSQL only.
   * Hue
   * Each Hive metastore
   * Oozie
   * Schema Registry
   * Streams Messaging Manager .
4. Install Agent on all hosts in the cluster. It is possible to add hosts to Cloudera Manager using the [Installation Wizard](https://docs.cloudera.com/cdp-private-cloud-base/7.1.7/installation/topics/cdpdc-installation-wizard.html).
5. Start Cloudera Manager Server and Cloudera Manager agent on all hosts. For more information, see [Cloudera Manager Agent](https://docs.cloudera.com/cdp-private-cloud-base/7.1.7/managing-clusters/topics/cm_agents.html) and [Cloudera Manager Server](https://docs.cloudera.com/cdp-private-cloud-upgrade/latest/upgrade-cdp/topics/ug_cm_upgrade_backup_start_management_service.html).
6. Install Cloudera Manager User licence. For more information, see [Installation Wizard](https://docs.cloudera.com/cdp-private-cloud-base/7.1.7/installation/topics/cdpdc-installation-wizard.html). (Upload the license file and exit the cluster setup by clicking the Cloudera Manager icon )

caution

* + Do not **set up a cluster using the Wizard (Step 7)** .
  + Do not **proceed to Welcome (Add Cluster - Installation)**.

1. Add Hosts to a cluster. To add hosts to a cluster, see [Adding Hosts to a cluster](https://docs.cloudera.com/cdp-private-cloud-base/7.1.7/managing-clusters/topics/cm-add-hosts.html).
2. Add Cloudera Manager management service to the cluster. To add services to the cluster, see [Select Services](https://docs.cloudera.com/cdp-private-cloud-base/7.1.7/installation/topics/cdpdc-select-services.html).
3. Set up Kerberos. If you have a Kerberos cluster, then you must add the KDC details in the Administration>Security>Kerberos Credentials>Setup KDC for Cloudera Manager page using Cloudera Manager. For more information on Kerberos and Active Directory, see Enabling Kerberos authentication for CDP

Note:

* + You must set the value of Maximum Renewable Life for Principals to 0 or to the value that was provided for Ambari KDC. If Maximum Renewable Life for Principal and Ambari KDC do not match, keytabs generation fails in Cloudera Manager.
  + If you are not upgrading from CDH or HDP to CDP Private Cloud Base, you can follow the [Production Installation](https://docs.cloudera.com/cdp-private-cloud-base/7.1.7/installation/topics/cdpdc-installing-cm-runtime.html) instructions to perform a fresh installation in your production environment.

**PRACTICAL 8: TO EXECUTE WORD COUNT PROGRAM**

public class CountNumberOfWordsInStringMain {

    public static void main(String[] args) {

        String str = "TO EXECUTE A WORD COUNT PROGRAM";

        int count = 1;

        for (int i = 0; i < str.length() - 1; i++)

        {

            if ((str.charAt(i) == ' ') && (str.charAt(i + 1) != ' '))

            {

                count++;

            }

        }

        System.out.println("Number of words in a string : " + count);

    }

}

**OUTPUT**

Number of words in a string : 6

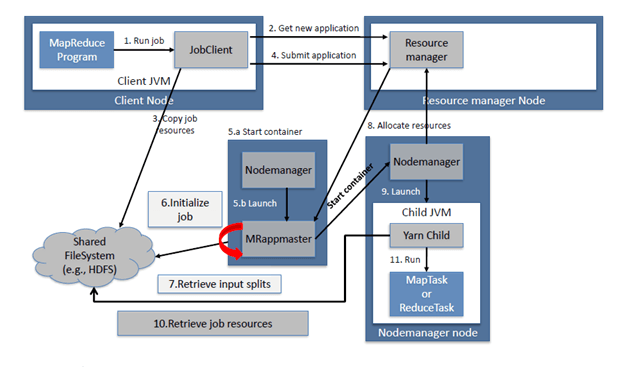
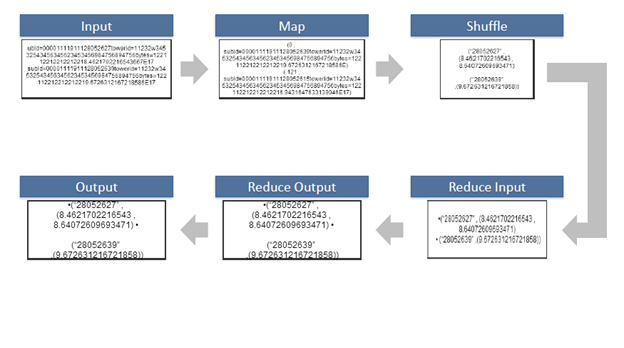
**PRACTICAL 9: TO STUDY ABOUT INTRODUCTION TO MAP REDUCER**

A MapReduce is a data processing tool which is used to process the data parallelly in a distributed form. It was developed in 2004, on the basis of paper titled as "MapReduce: Simplified Data Processing on Large Clusters," published by Google.

The MapReduce is a paradigm which has two phases, the mapper phase, and the reducer phase. In the Mapper, the input is given in the form of a key-value pair. The output of the Mapper is fed to the reducer as input. The reducer runs only after the Mapper is over. The reducer too takes input in key-value format, and the output of reducer is the final output.

## Steps in Map Reduce

* The map takes data in the form of pairs and returns a list of <key, value> pairs. The keys will not be unique in this case.
* Using the output of Map, sort and shuffle are applied by the Hadoop architecture. This sort and shuffle acts on these list of <key, value> pairs and sends out unique keys and a list of values associated with this unique key <key, list(values)>.
* An output of sort and shuffle sent to the reducer phase. The reducer performs a defined function on a list of values for unique keys, and Final output <key, value> will be stored/displayed.



## **Sort and Shuffle**

The sort and shuffle occur on the output of Mapper and before the reducer. When the Mapper task is complete, the results are sorted by key, partitioned if there are multiple reducers, and then written to disk. Using the input from each Mapper <k2,v2>, we collect all the values for each unique key k2. This output from the shuffle phase in the form of <k2, list(v2)> is sent as input to reducer phase.

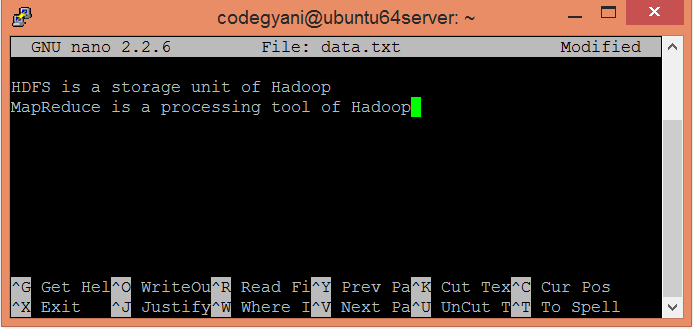
## **Usage of** **MapReduce**

* It can be used in various application like document clustering, distributed sorting, and web link-graph reversal.
* It can be used for distributed pattern-based searching.
* We can also use MapReduce in machine learning.
* It was used by Google to regenerate Google's index of the World Wide Web.
* It can be used in multiple computing environments such as multi-cluster, multi-core, and mobile environment.

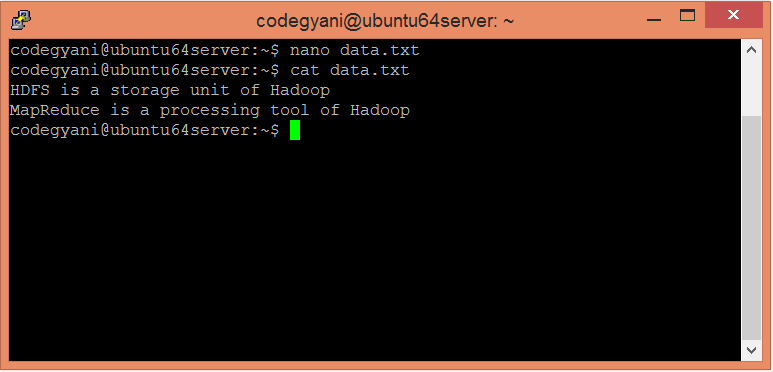
**PRACTICAL 10: TO EXECUTE WORD COUNT PROGRAM USING MAP REDUCER**

## Steps to execute MapReduce word count example

* Create a text file in your local machine and write some text into it.  
  $ nano data.txt

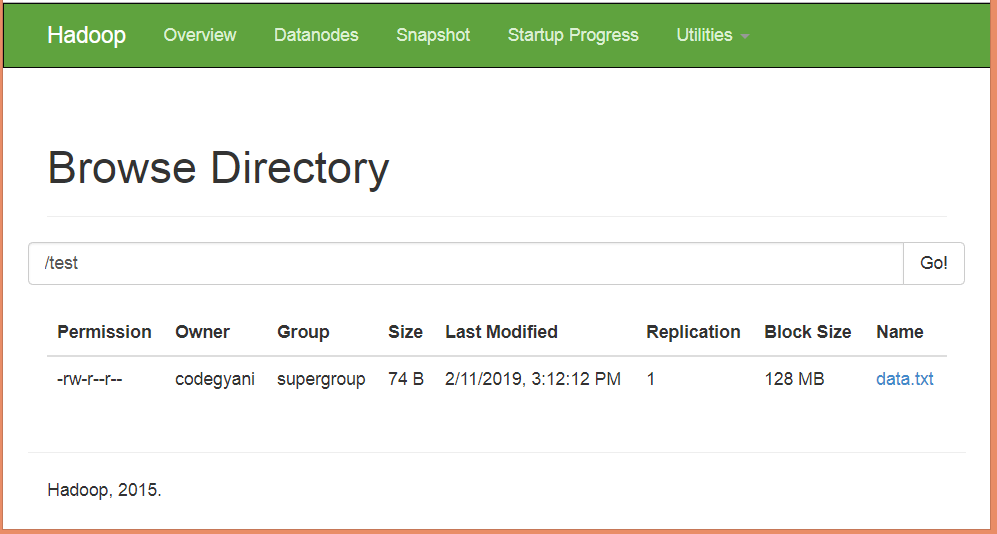


* Check the text written in the data.txt file.   
  $ cat data.txt



In this example, we find out the frequency of each word exists in this text file.

* Create a directory in HDFS, where to kept text file.  
  $ hdfs dfs -mkdir /test
* Upload the data.txt file on HDFS in the specific directory.  
  $ hdfs dfs -put /home/codegyani/data.txt /test



* Write the MapReduce program using eclipse.

### File: WC\_Mapper.java

1. package com.javatpoint;
3. import java.io.IOException;
4. import java.util.StringTokenizer;
5. import org.apache.hadoop.io.IntWritable;
6. import org.apache.hadoop.io.LongWritable;
7. import org.apache.hadoop.io.Text;
8. import org.apache.hadoop.mapred.MapReduceBase;
9. import org.apache.hadoop.mapred.Mapper;
10. import org.apache.hadoop.mapred.OutputCollector;
11. import org.apache.hadoop.mapred.Reporter;
12. public class WC\_Mapper extends MapReduceBase implements Mapper<LongWritable,Text,Text,IntWritable>{
13. private final static IntWritable one = new IntWritable(1);
14. private Text word = new Text();
15. public void map(LongWritable key, Text value,OutputCollector<Text,IntWritable> output,
16. Reporter reporter) throws IOException{
17. String line = value.toString();
18. StringTokenizer  tokenizer = new StringTokenizer(line);
19. while (tokenizer.hasMoreTokens()){
20. word.set(tokenizer.nextToken());
21. output.collect(word, one);
22. }
23. }
25. }

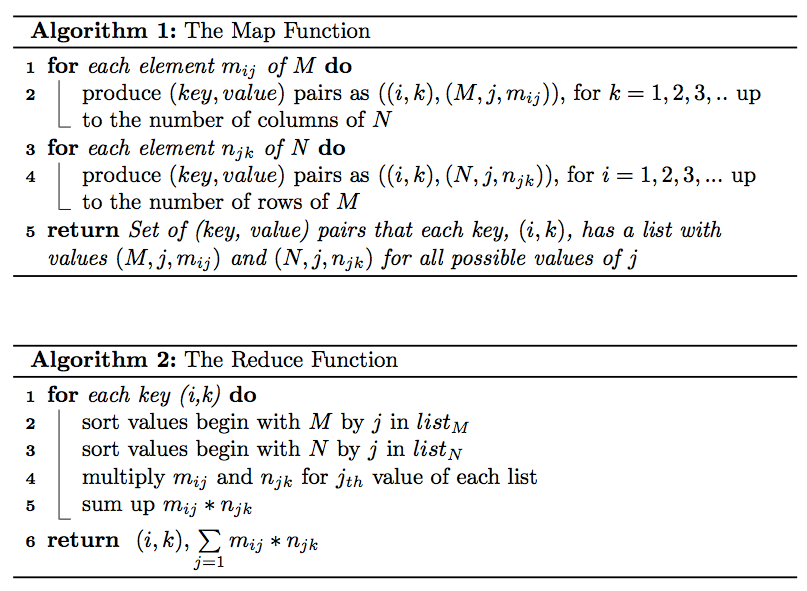
### File: WC\_Reducer.java

1. package com.javatpoint;
2. import java.io.IOException;
3. import java.util.Iterator;
4. import org.apache.hadoop.io.IntWritable;
5. import org.apache.hadoop.io.Text;
6. import org.apache.hadoop.mapred.MapReduceBase;
7. import org.apache.hadoop.mapred.OutputCollector;
8. import org.apache.hadoop.mapred.Reducer;
9. import org.apache.hadoop.mapred.Reporter;
11. public class WC\_Reducer  extends MapReduceBase implements Reducer<Text,IntWritable,Text,IntWritable> {
12. public void reduce(Text key, Iterator<IntWritable> values,OutputCollector<Text,IntWritable> output,
13. Reporter reporter) throws IOException {
14. int sum=0;
15. while (values.hasNext()) {
16. sum+=values.next().get();
17. }
18. output.collect(key,new IntWritable(sum));
19. }
20. }

### File: WC\_Runner.java

1. package com.javatpoint;
3. import java.io.IOException;
4. import org.apache.hadoop.fs.Path;
5. import org.apache.hadoop.io.IntWritable;
6. import org.apache.hadoop.io.Text;
7. import org.apache.hadoop.mapred.FileInputFormat;
8. import org.apache.hadoop.mapred.FileOutputFormat;
9. import org.apache.hadoop.mapred.JobClient;
10. import org.apache.hadoop.mapred.JobConf;
11. import org.apache.hadoop.mapred.TextInputFormat;
12. import org.apache.hadoop.mapred.TextOutputFormat;
13. public class WC\_Runner {
14. public static void main(String[] args) throws IOException{
15. JobConf conf = new JobConf(WC\_Runner.class);
16. conf.setJobName("WordCount");
17. conf.setOutputKeyClass(Text.class);
18. conf.setOutputValueClass(IntWritable.class);
19. conf.setMapperClass(WC\_Mapper.class);
20. conf.setCombinerClass(WC\_Reducer.class);
21. conf.setReducerClass(WC\_Reducer.class);
22. conf.setInputFormat(TextInputFormat.class);
23. conf.setOutputFormat(TextOutputFormat.class);
24. FileInputFormat.setInputPaths(conf,new Path(args[0]));
25. FileOutputFormat.setOutputPath(conf,new Path(args[1]));
26. JobClient.runJob(conf);
27. }
28. }

**PRACTICAL 11: TO IMPLEMENT MATRIX MULTIPLICATION USING MAP REDUCER**

We will write Map and Reduce functions to process input files. Map and reduce functions will implement the following algorithms: [](https://github.com/marufaytekin/MatrixMultiply/blob/main/img/algorithms.png)

As it seen in MatrixMultiply.java code below, in main method the configuration parameters are being set as well as the input/output directories of MapReduce job [2].

|  |
| --- |
| [**MATRIXMULTIPLY.JAVA**](https://github.com/marufaytekin/MatrixMultiply/blob/main/src/main/java/com/lendap/hadoop/MatrixMultiply.java)  package com.lendap.hadoop; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| import org.apache.hadoop.conf.\*; |
|  |

|  |
| --- |
| import org.apache.hadoop.fs.Path; |
|  |

|  |
| --- |
| import org.apache.hadoop.io.\*; |
|  |

|  |
| --- |
| import org.apache.hadoop.mapreduce.\*; |
|  |

|  |
| --- |
| import org.apache.hadoop.mapreduce.lib.input.FileInputFormat; |
|  |

|  |
| --- |
| import org.apache.hadoop.mapreduce.lib.input.TextInputFormat; |
|  |

|  |
| --- |
| import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat; |
|  |

|  |
| --- |
| import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| public class MatrixMultiply { |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| public static void main(String[] args) throws Exception { |
|  |

|  |
| --- |
| if (args.length != 2) { |
|  |

|  |
| --- |
| System.err.println("Usage: MatrixMultiply <in\_dir> <out\_dir>"); |
|  |

|  |
| --- |
| System.exit(2); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| Configuration conf = new Configuration(); |
|  |

|  |
| --- |
| // M is an m-by-n matrix; N is an n-by-p matrix. |
|  |

|  |
| --- |
| conf.set("m", "1000"); |
|  |

|  |
| --- |
| conf.set("n", "100"); |
|  |

|  |
| --- |
| conf.set("p", "1000"); |
|  |

|  |
| --- |
| @SuppressWarnings("deprecation") |
|  |

|  |
| --- |
| Job job = new Job(conf, "MatrixMultiply"); |
|  |

|  |
| --- |
| job.setJarByClass(MatrixMultiply.class); |
|  |

|  |
| --- |
| job.setOutputKeyClass(Text.class); |
|  |

|  |
| --- |
| job.setOutputValueClass(Text.class); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| job.setMapperClass(Map.class); |
|  |

|  |
| --- |
| job.setReducerClass(Reduce.class); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| job.setInputFormatClass(TextInputFormat.class); |
|  |

|  |
| --- |
| job.setOutputFormatClass(TextOutputFormat.class); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| FileInputFormat.addInputPath(job, new Path(args[0])); |
|  |

|  |
| --- |
| FileOutputFormat.setOutputPath(job, new Path(args[1])); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| job.waitForCompletion(true); |
|  |

|  |
| --- |
| } |
|  |

}

Mapper class extends org.apache.hadoop.mapreduce.Mapper class and implements the map task described in Algorithm 1 and creates the key,valuepairs from the input files as it shown in the code below:

**MAP.JAVA**

|  |
| --- |
| package com.lendap.hadoop; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| import org.apache.hadoop.conf.\*; |
|  |

|  |
| --- |
| import org.apache.hadoop.io.LongWritable; |
|  |

|  |
| --- |
| import org.apache.hadoop.io.Text; |
|  |

|  |
| --- |
| import org.apache.hadoop.mapreduce.Mapper; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| import java.io.IOException; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| public class Map |
|  |

|  |
| --- |
| extends org.apache.hadoop.mapreduce.Mapper<LongWritable, Text, Text, Text> { |
|  |

|  |
| --- |
| @Override |
|  |

|  |
| --- |
| public void map(LongWritable key, Text value, Context context) |
|  |

|  |
| --- |
| throws IOException, InterruptedException { |
|  |

|  |
| --- |
| Configuration conf = context.getConfiguration(); |
|  |

|  |
| --- |
| int m = Integer.parseInt(conf.get("m")); |
|  |

|  |
| --- |
| int p = Integer.parseInt(conf.get("p")); |
|  |

|  |
| --- |
| String line = value.toString(); |
|  |

|  |
| --- |
| // (M, i, j, Mij); |
|  |

|  |
| --- |
| String[] indicesAndValue = line.split(","); |
|  |

|  |
| --- |
| Text outputKey = new Text(); |
|  |

|  |
| --- |
| Text outputValue = new Text(); |
|  |

|  |
| --- |
| if (indicesAndValue[0].equals("M")) { |
|  |

|  |
| --- |
| for (int k = 0; k < p; k++) { |
|  |

|  |
| --- |
| outputKey.set(indicesAndValue[1] + "," + k); |
|  |

|  |
| --- |
| // outputKey.set(i,k); |
|  |

|  |
| --- |
| outputValue.set(indicesAndValue[0] + "," + indicesAndValue[2] |
|  |

|  |
| --- |
| + "," + indicesAndValue[3]); |
|  |

|  |
| --- |
| // outputValue.set(M,j,Mij); |
|  |

|  |
| --- |
| context.write(outputKey, outputValue); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| } else { |
|  |

|  |
| --- |
| // (N, j, k, Njk); |
|  |

|  |
| --- |
| for (int i = 0; i < m; i++) { |
|  |

|  |
| --- |
| outputKey.set(i + "," + indicesAndValue[2]); |
|  |

|  |
| --- |
| outputValue.set("N," + indicesAndValue[1] + "," |
|  |

|  |
| --- |
| + indicesAndValue[3]); |
|  |

|  |
| --- |
| context.write(outputKey, outputValue); |
| }}}} |

Reducer class, extends org.apache.hadoop.mapreduce.Reducer class and implements the reduce task described in Algorithm 2 and creates the key,valuepairs for the product matrix then writes its output on HDFS as it shown in the code below:

**REDUCE.JAVA**

|  |
| --- |
| package com.lendap.hadoop; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| import org.apache.hadoop.io.Text; |
|  |

|  |
| --- |
| import org.apache.hadoop.mapreduce.Reducer; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| import java.io.IOException; |
|  |

|  |
| --- |
| import java.util.HashMap; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| public class Reduce |
|  |

|  |
| --- |
| extends org.apache.hadoop.mapreduce.Reducer<Text, Text, Text, Text> { |
|  |

|  |
| --- |
| @Override |
|  |

|  |
| --- |
| public void reduce(Text key, Iterable<Text> values, Context context) |
|  |

|  |
| --- |
| throws IOException, InterruptedException { |
|  |

|  |
| --- |
| String[] value; |
|  |

|  |
| --- |
| //key=(i,k), |
|  |

|  |
| --- |
| //Values = [(M/N,j,V/W),..] |
|  |

|  |
| --- |
| HashMap<Integer, Float> hashA = new HashMap<Integer, Float>(); |
|  |

|  |
| --- |
| HashMap<Integer, Float> hashB = new HashMap<Integer, Float>(); |
|  |

|  |
| --- |
| for (Text val : values) { |
|  |

|  |
| --- |
| value = val.toString().split(","); |
|  |

|  |
| --- |
| if (value[0].equals("M")) { |
|  |

|  |
| --- |
| hashA.put(Integer.parseInt(value[1]), Float.parseFloat(value[2])); |
|  |

|  |
| --- |
| } else { |
|  |

|  |
| --- |
| hashB.put(Integer.parseInt(value[1]), Float.parseFloat(value[2])); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| int n = Integer.parseInt(context.getConfiguration().get("n")); |
|  |

|  |
| --- |
| float result = 0.0f; |
|  |

|  |
| --- |
| float m\_ij; |
|  |

|  |
| --- |
| float n\_jk; |
|  |

|  |
| --- |
| for (int j = 0; j < n; j++) { |
|  |

|  |
| --- |
| m\_ij = hashA.containsKey(j) ? hashA.get(j) : 0.0f; |
|  |

|  |
| --- |
| n\_jk = hashB.containsKey(j) ? hashB.get(j) : 0.0f; |
|  |

|  |
| --- |
| result += m\_ij \* n\_jk; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| if (result != 0.0f) { |
|  |

|  |
| --- |
| context.write(null, |
|  |

|  |
| --- |
| new Text(key.toString() + "," + Float.toString(result))); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| } |
|  |

}

**PRACTICAL 12: ANALYSIS AND IMPLEMENTATION OF NOSQL**

NoSQL databases (aka "not only SQL") are non-tabular databases and store data differently than relational tables. NoSQL databases come in a variety of types based on their data model. The main types are document, key-value, wide-column, and graph. They provide flexible schemas and scale easily with large amounts of data and high user loads.

### NoSQL database features

Each NoSQL database has its own unique features. At a high level, many NoSQL databases have the following features:

* [Flexible schemas](https://docs.mongodb.com/manual/core/data-modeling-introduction/#flexible-schema)
* [Horizontal scaling](https://www.mongodb.com/basics/scaling)
* [Fast queries due to the data model](https://docs.mongodb.com/manual/core/data-modeling-introduction/#document-structure)
* [Ease of use for developers](https://www.mongodb.com/why-use-mongodb)

### Types of NoSQL databases

Over time, four major [types of NoSQL databases](https://www.mongodb.com/scale/types-of-nosql-databases) emerged: document databases, key-value databases, wide-column stores, and graph databases.

* **Document databases** store data in documents similar to JSON (JavaScript Object Notation) objects. Each document contains pairs of fields and values. The values can typically be a variety of types including things like strings, numbers, booleans, arrays, or objects.
* **Key-value databases** are a simpler type of database where each item contains keys and values.
* **Wide-column stores** store data in tables, rows, and dynamic columns.
* **Graph databases** store data in nodes and edges. Nodes typically store information about people, places, and things, while edges store information about the relationships between the nodes.

**IMPLEMENTATION OF NOSQL USING MONGO DB**

The data model we design for a NoSQL database will depend on the type of NoSQL database we choose. Let's consider how to store the same information about a user and their hobbies in a document database like MongoDB.

{

"\_id": 1,

"first\_name": "Leslie",

"last\_name": "Yepp",

"cell": "8125552344",

"city": "Pawnee",

"hobbies": ["scrapbooking", "eating waffles", "working"]

}

In order to retrieve all of the information about a user and their hobbies, a single document can be retrieved from the database. No joins are required, resulting in faster queries.

**PRACTICAL 13: INTRODUCTION TO MONGO DB**

MongoDB is a NoSQL database which stores the data in form of key-value pairs. It is an **Open Source**, **Document Database** which provides high performance and scalability along with data modelling and data management of huge sets of data in an enterprise application.

MongoDB also provides the feature of Auto-Scaling. Since, MongoDB is a cross platform database and can be installed across different platforms like Windows, Linux etc.

A record in MongoDB is a document, which is a data structure composed of field and value pairs. MongoDB documents are similar to JSON objects. The values of fields may include other documents, arrays, and arrays of documents.

A Document is nothing but a data structure with name-value pairs like in JSON. It is very easy to map any custom Object of any programming language with a MongoDB Document. For example : Student object has attributes name, rollno and subjects, where subjects is a List.

Document for Student in MongoDB will be like:

{

name : "Stduytonight",

rollno : 1,

subjects : ["C Language", "C++", "Core Java"]

}

We can see, Documents are actually JSON representation of custom Objects. Also, excessive JOINS can be avoided by saving data in form of Arrays and Documents(Embedded) inside a Document.

## Key Features of MongoDB

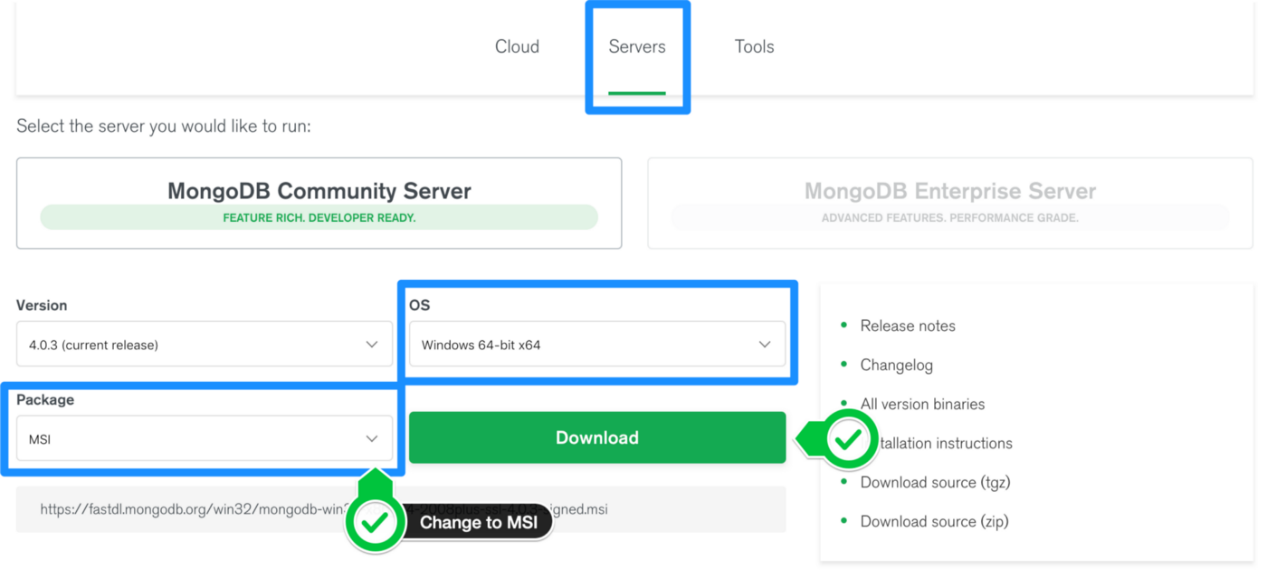
Apart from most of the NoSQL default features, MongoDB does bring in some more, very important and useful features :

1. MongoDB provides high performance. Input/Output operations are lesser than relational databases due to support of embedded documents(data models) and Select queries are also faster as Indexes in MongoDB supports faster queries.
2. MongoDB has a rich Query Language, supporting all the major CRUD operations. The Query Language also provides good Text Search and Aggregation features.
3. **Auto Replication** feature of MongoDB leads to High Availability. It provides an automatic failover mechanism, as data is restored through backup(replica) copy if server fails.
4. Sharding is a major feature of MongoDB. Horizontal Scalability is possible due to sharding.
5. MongoDB supports multiple Storage Engines. When we save data in form of documents(NoSQL) or tables(RDBMS) who saves the data? It's the Storage Engine. Storage Engines manages how data is saved in memory and on disk.

**PRACTICAL 14: TO STUDY ABOUT INSTALLATION OF MONGO DB**

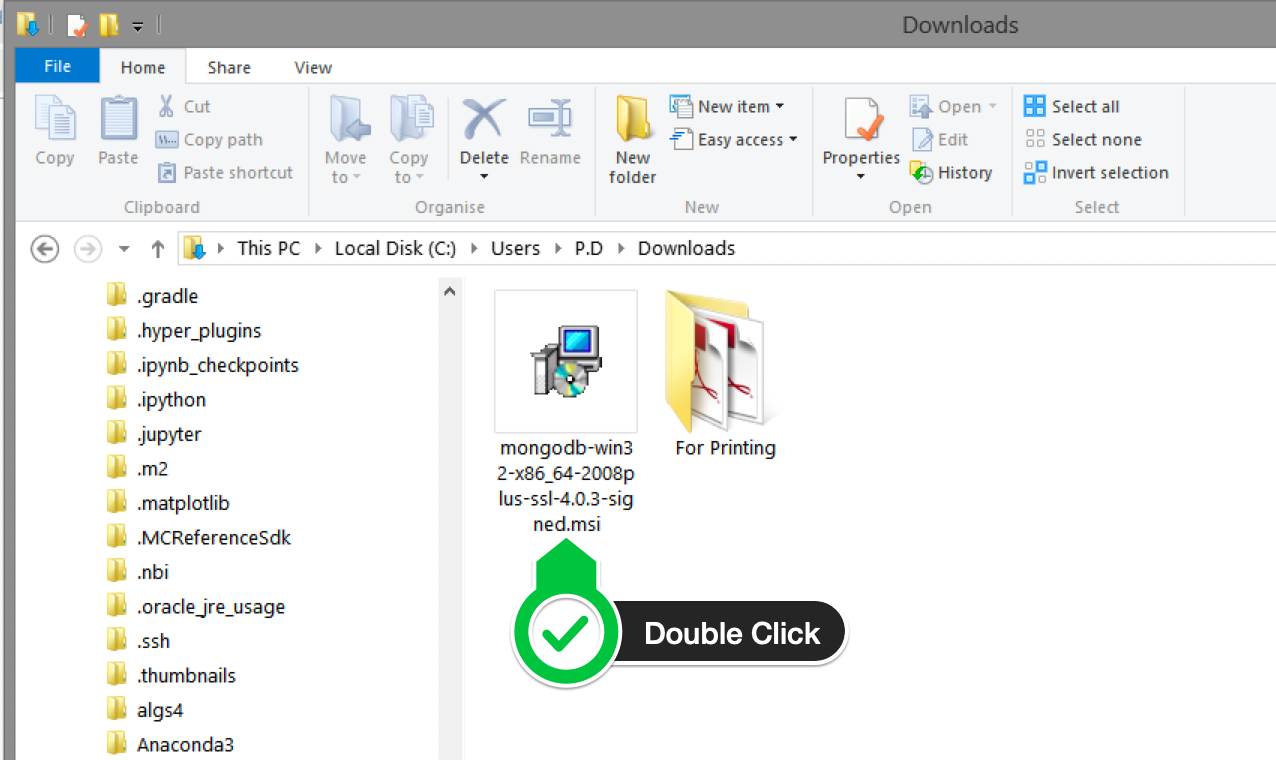
**Step 1 — Download the MongoDB MSI Installer Package**

Head over [here](https://www.mongodb.com/download-center/community) and download the current version of MongoDB. Make sure you **select MSI** as the package you want to download.

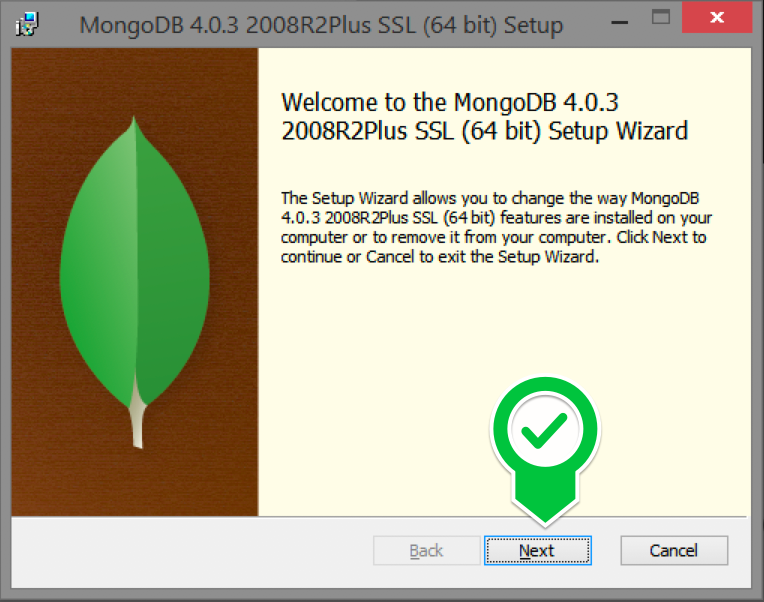


**Step 2 — Install MongoDB with the Installation Wizard**

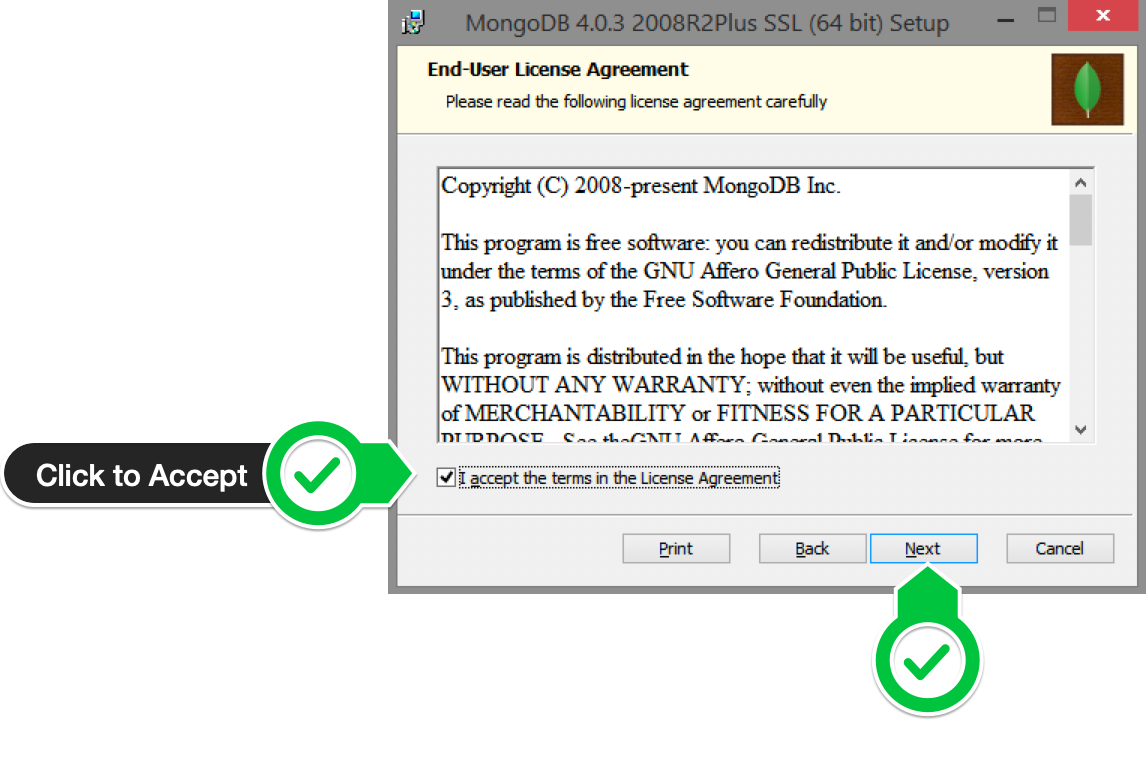
A. Make sure you are logged in as a user with Admin privileges. Then navigate to your downloads folder and double click on the .msi package you just downloaded. This will launch the installation wizard.



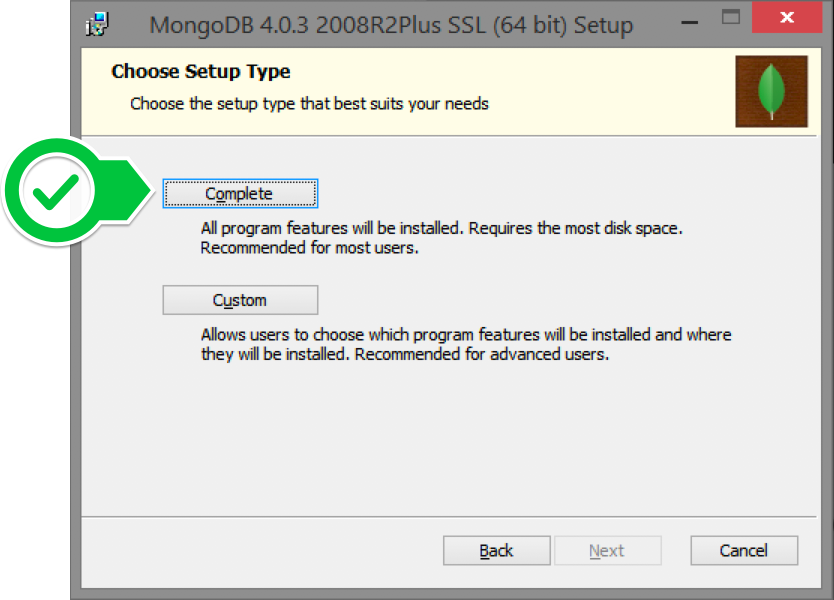
B. Click Next to start installation.



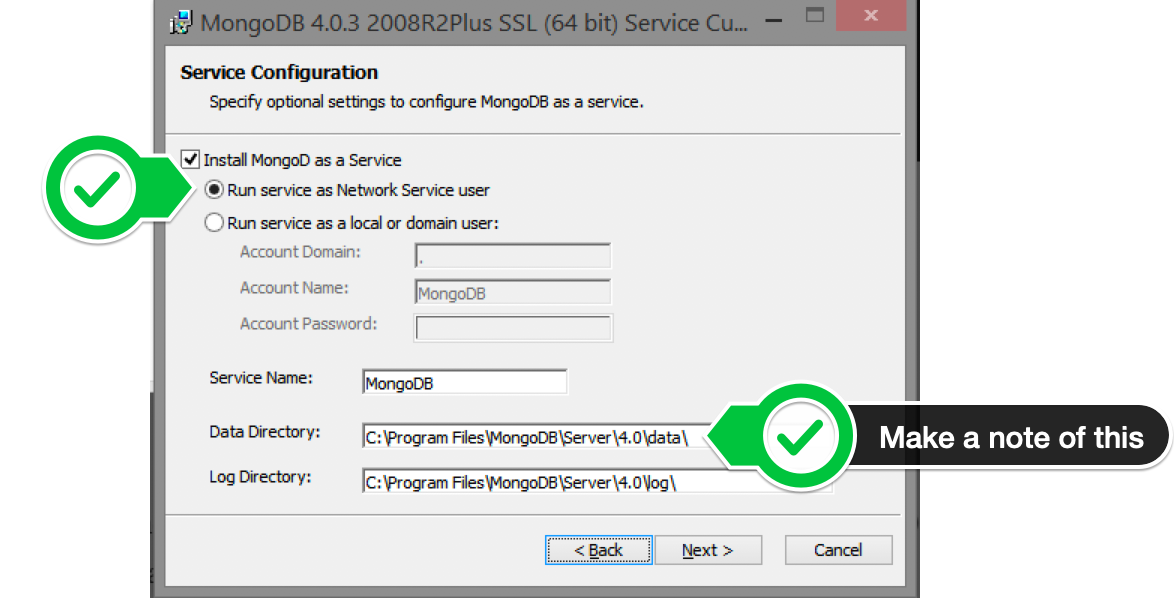
C. Accept the licence agreement then click Next.



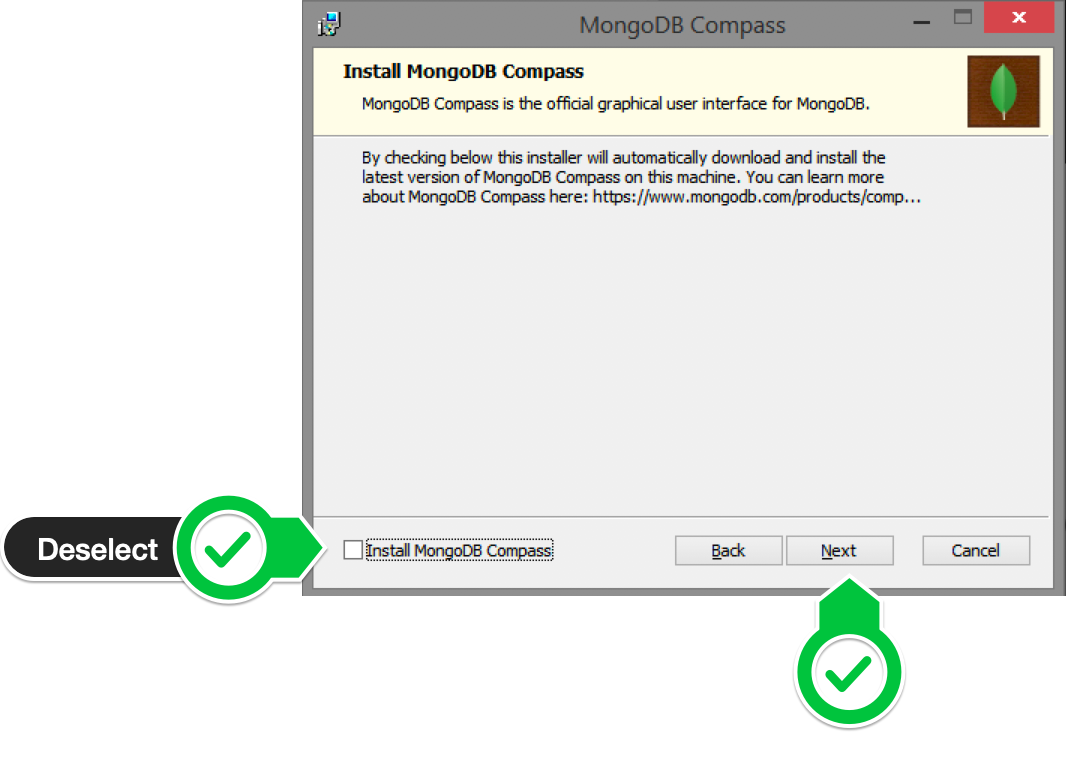
D. Select the Complete setup.



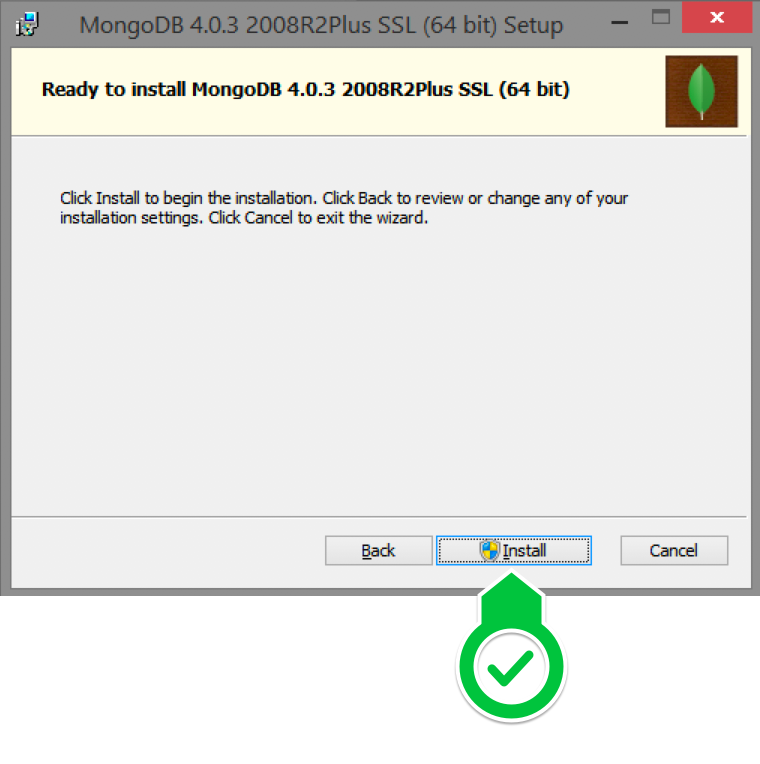
E. Select “Run service as Network Service user” and make a note of the data directory, we’ll need this later.



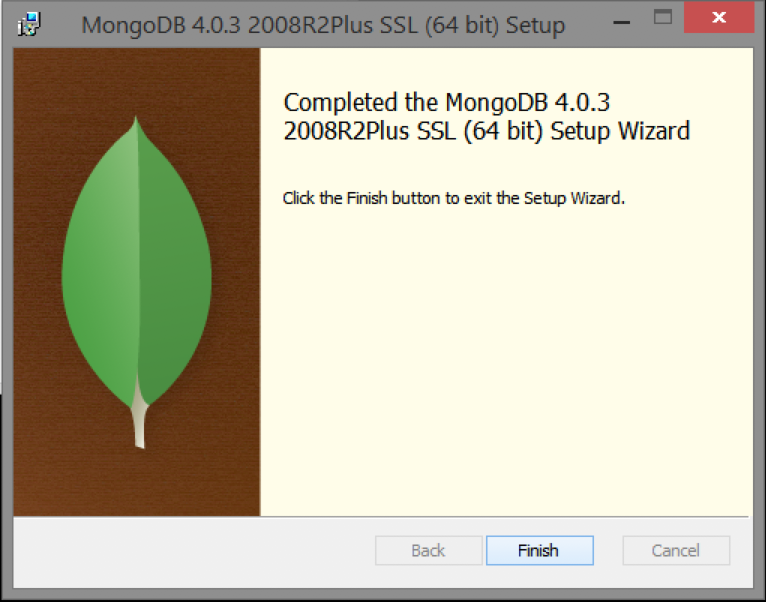
F. We won’t need Mongo Compass, so deselect it and click Next.



G. Click Install to begin installation.

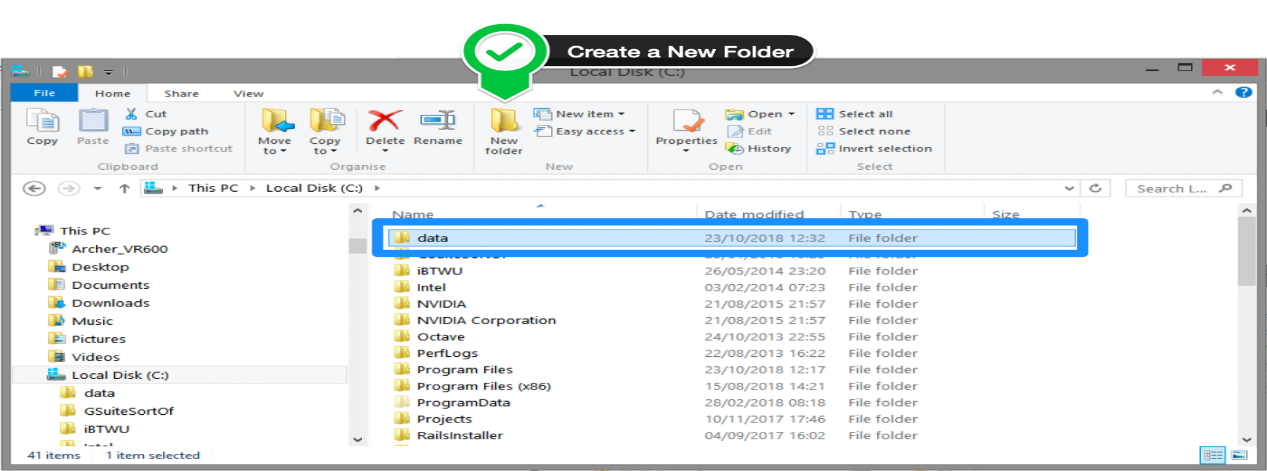


F. Hit Finish to complete installation.

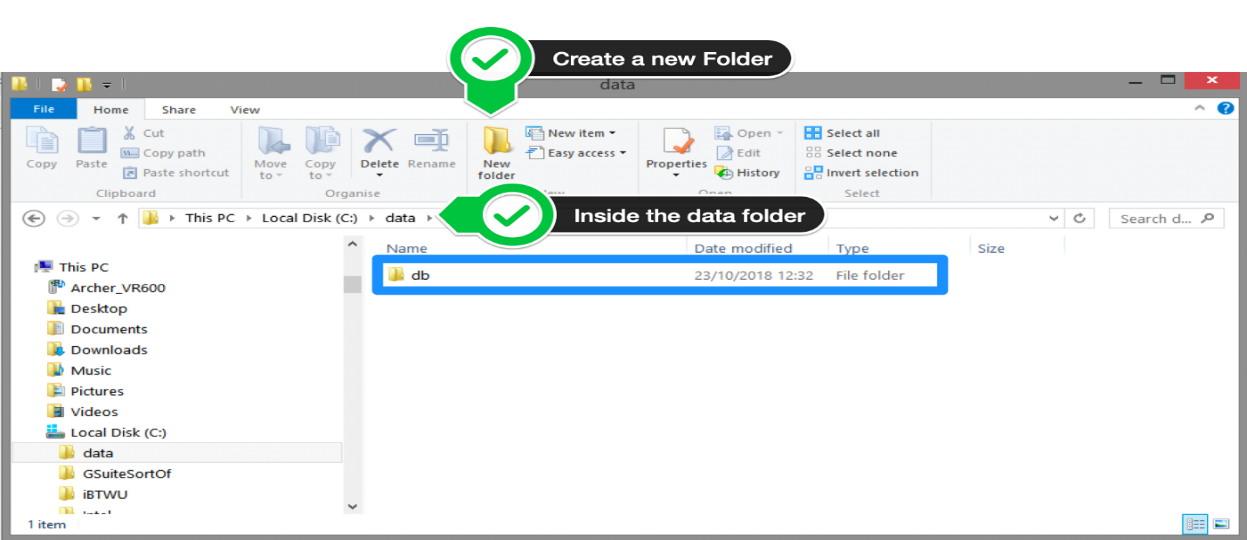


**Step 3— Create the Data Folders to Store our Databases**

A. Navigate to the **C Drive** on your computer using Explorer and create a new folder called **data** here.



B. Inside the **data** folder you just created, create another folder called **db**.



**Step 4 — Setup Alias Shortcuts for Mongo and Mongod**

Once installation is complete, we’ll need to set up MongoDB on the local system.

A. Open up your Hyper terminal running Git Bash.

B. Change directory to your home directory with the following command: cd ~

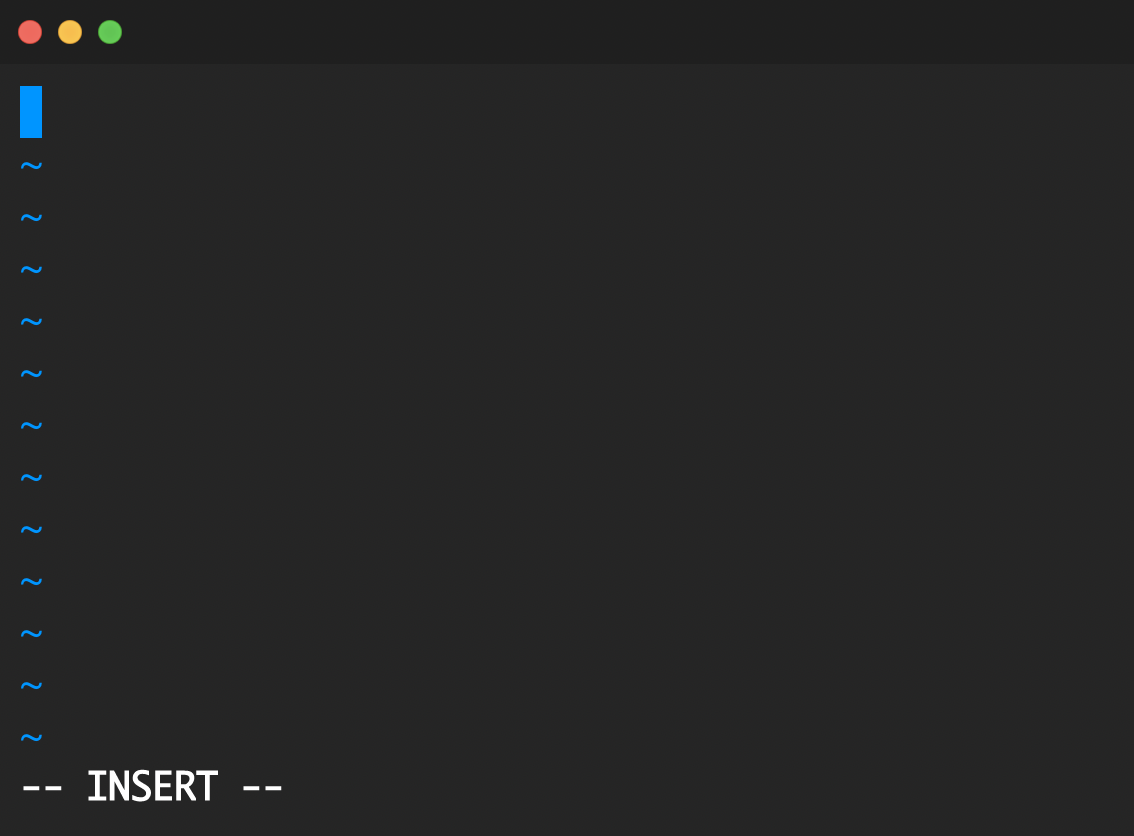
C. Here, we’re going to create a file called .bash\_profile using the following command:

touch .bash\_profile

D. Open the newly created .bash\_profile with vim using the following command:

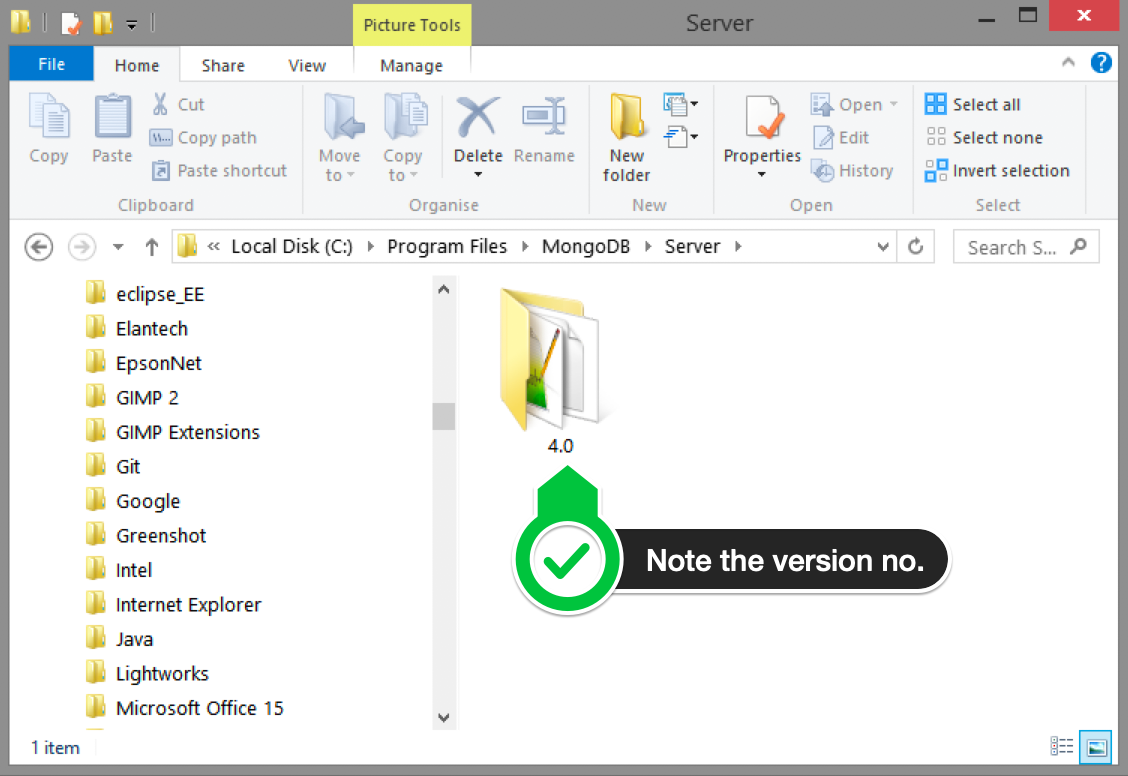
vim .bash\_profile

E. In vim, hit the **I** key on the keyboard to enter insert mode.



F. In your explorer go to C → Program Files → MongoDB → Server

Now you should see the version of your MongoDB.



G. Paste in the following code into vim, make sure your replace the 4.0 with your version that you see in explorer

alias mongod="/c/Program\ files/MongoDB/Server/4.0/bin/mongod.exe"  
alias mongo="/c/Program\ Files/MongoDB/Server/4.0/bin/mongo.exe"

F. Hit the Escape key on your keyboard to exit the insert mode. Then type :wq!

to save and exit Vim.



**Step 5 — Verify That Setup was Successful**

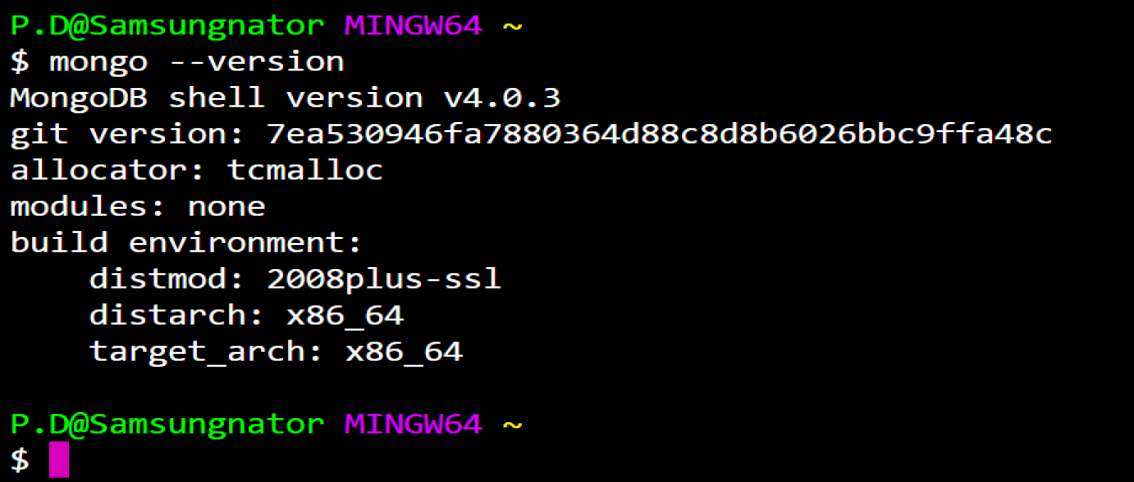
A. Close down the current Hyper terminal and quit the application.

B. Re-launch Hyper.

C. Type the following commands into the Hyper terminal:

mongo --version

Once you’ve hit enter, you should see something like this:



This means that you have successfully installed and setup MongoDB on your local system!

**PRACTICAL 15: TO STUDY ABOUT HIVE WITH COMMANDS**

## **DDL COMMANDS ON DATABASES IN HIVE**

### **Create Database in Hive**.

CREATE (DATABASE) [IF NOT EXISTS] database\_name

  [COMMENT database\_comment]

  [LOCATION hdfs\_path]

  [WITH DBPROPERTIES (property\_name=property\_value, ...)];

In the above syntax for create database command, the values mentioned in square brackets [] are optional.

#### **Usage of Create Database Command in Hive**

hive> create database if not exists firstDB comment "This is my first demo" location '/user/hive/warehouse/newdb' with DBPROPERTIES ('createdby'='abhay','createdfor'='dezyre');

OK

Time taken: 0.092 seconds

### **Drop Database in Hive**

DROP (DATABASE) [IF EXISTS] database\_name [RESTRICT|CASCADE];

#### **Usage of Drop Database Command in Hive**

hive> drop database if exists firstDB CASCADE;

OK

Time taken: 0.099 seconds

### **Describe Database Command in Hive**

### This command is used to check any associated metadata for the databases.

Describe Database Command in Hive

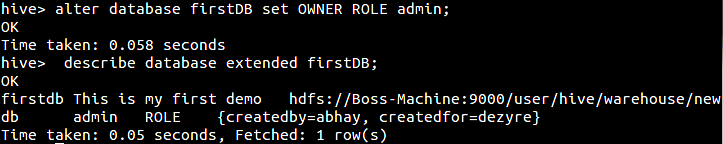
### **Alter Database Command in Hive**

### Whenever the developers need to change the metadata of any of the databases, alter hive DDL command can be used as follows –

ALTER (DATABASE) database\_name SET DBPROPERTIES (property\_name=property\_value, ...);

**Usage of ALTER database command in Hive**

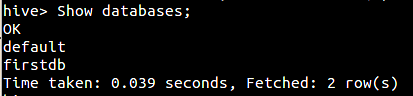
Let’s use the Alter command to modify the OWNER property and specify the role for the owner –ALTER (DATABASE) database\_name SET OWNER [USER|ROLE] user\_or\_role;



### **Show Database Command in Hive**

### Programmers can view the list of existing databases in the current schema.

#### **Usage of Show Database Command**



### **Use Database Command in Hive**

### This hive command is used to select a specific database for the session on which hive queries would be executed.

#### **Usage of Use Database Command in Hive**

Use Database Hive Command

### **DDL COMMANDS ON TABLES IN HIVE**

#### **Create Table Command in Hive**

#### Hive create table command is used to create a table in the existing database that is in use for a particular session.

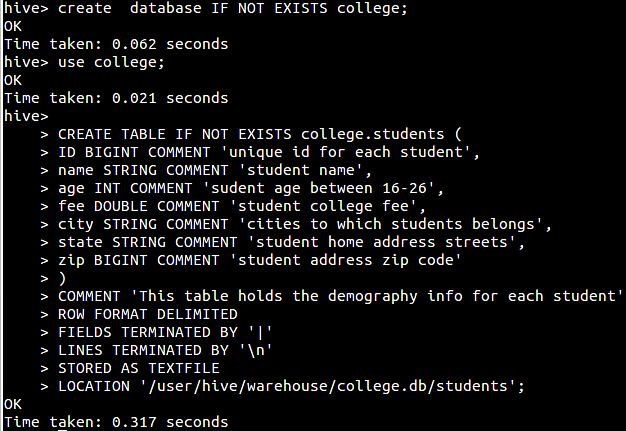
CREATE  TABLE [IF NOT EXISTS] [db\_name.]table\_name    --

  [(col\_name data\_type [COMMENT col\_comment], ...)]

  [COMMENT table\_comment]

   [LOCATION hdfs\_path]

### **Hive Create Table Usage**



In the above step, we have created a hive table named Students in the database college with various fields like ID, Name, fee, city, etc. Comments have been mentioned for each column so that anybody referring to the table gets an overview about what the columns mean.

### **DROP Table Command in Hive**

### Drops the table and all the data associated with it in the Hive metastore.

DROP TABLE [IF EXISTS] table\_name [PURGE];

**Usage of DROP Table command in Hive**

Drop Table Hive Command

DROP table command removes the metadata and data for a particular table. Data is usually moved to .Trash/Current directory if Trash is configured. If PURGE option is specified then the table data will not go to the trash directory and there will be no scope to retrieve the data in case of erroneous DROP command execution.

### **TRUNCATE Table Command in Hive**

### This hive command is used to truncate all the rows present in a table i.e. it deletes all the data from the Hive meta store and the data cannot be restored.

TRUNCATE TABLE [db\_name].table\_name

#### **Usage of TRUNCATE Table in Hive**

Truncate Table Hive Command

### **ALTER Table Command in Hive**

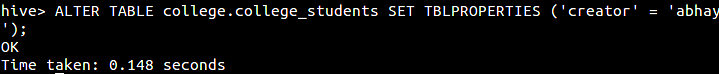
### Using ALTER Table command, the structure and metadata of the table can be modified even after the table has been created. Let’s try to change the name of an existing table using the ALTER command –

ALTER TABLE [db\_name].old\_table\_name RENAME TO [db\_name].new\_table\_name;

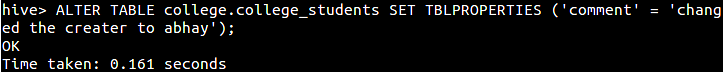
Alter Table Command in Hive

### **Syntax to ALTER Table Properties**

ALTER TABLE [db\_name].tablename SET TBLPROPERTIES (‘property\_key’=’property\_new\_value’)

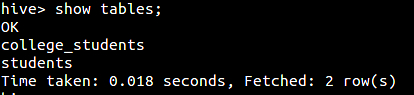


In the above step, we have set the creator attribute for the table and similarly we can later or modify other table properties also.



### **Show Table Command in Hive**

### Gives the list of existing tables in the current database schema.



## **DML COMMANDS IN HIVE**

DML (Data Manipulation Language) commands in Hive are used for inserting and querying the data from hive tables once the structure and architecture of the database has been defined using the DDL commands listed above.

Data can be loaded into Hive tables using –

* LOAD command
* Insert command

### **Usage of LOAD Command for Inserting Data Into Hive Tables**

#### **Syntax for Load Command in Hive**

LOAD DATA [LOCAL] INPATH 'hdfsfilepath/localfilepath' [OVERWRITE] INTO TABLE existing\_table\_name

#### **Example**

Let’s load a structured file that contains information about different students.

Let’s take a look at the data present in the file –

ID|name|age|fee|city|state |zip

1|Kendall|22|25874|Kulti-Barakar|WB|451333

2|Mikayla|25|35367|Jalgaon|Maharastra|710179

3|Raven|20|49103|Rewa|Madhya Pradesh|392423

4|Carla|19|27121|Pilibhit|UP|769853

5|Edward|21|32053|Tuticorin|Tamil Nadu|368262

6|Wynter|21|43956|Surendranagar|GJ|457441

7|Patrick|19|19050|Mumbai|MH|580220

8|Hayfa|18|15590|Amroha|UP|470705

9|Raven|16|37836|Cuddalore|TN|787001

The file is a ‘|’ delimited file where each row  can be inserted as a table record.

First let’s create a table student based on the contents in the file –

* The **ROW FORMAT DELIMITED** must appear before any of the other clauses, with the exception of the STORED AS … clause.
* The clause **ROW FORMAT DELIMITED FIELDS TERMINATED BY '|** means  I character will be used as field separator by hive.
* The clause **LINES TERMINATED BY ‘\n'** means that the line delimiter will be new line.
* The clause **LINES TERMINATED BY ‘\n'** and **STORED AS …** do not require the ROW FORMAT DELIMITED keywords.

hive> CREATE TABLE IF NOT EXISTS college.students (

> ID BIGINT COMMENT 'unique id for each student',

> name STRING COMMENT 'student name',

> age INT COMMENT 'sudent age between 16-26',

> fee DOUBLE COMMENT 'student college fee',

> city STRING COMMENT 'cities to which students belongs',

> state STRING COMMENT 'student home address state s',

> zip BIGINT COMMENT 'student address zip code'

> )

> COMMENT 'This table holds the demography info for each student'

> ROW FORMAT DELIMITED

> FIELDS TERMINATED BY '|'

> LINES TERMINATED BY '\n'

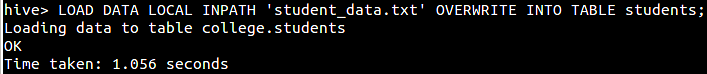
> STORED AS TEXTFILE

> LOCATION '/user/hive/warehouse/college.db/students';

OK

Time taken: 0.112 seconds

Let’s  load the file into the student table –



**PRACTICAL 16: TO STUDY ABOUT PIG AND EXECUTR ITS PROGRAM**

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

**Need of Pig:** One limitation of MapReduce is that the development cycle is very long. Writing the reducer and mapper, compiling packaging the code, submitting the job and retrieving the output is a time-consuming task. Apache Pig reduces the time of development using the multi-query approach. Also, Pig is beneficial for programmers who are not from [Java](https://www.geeksforgeeks.org/java/) background. 200 lines of Java code can be written in only 10 lines using the Pig Latin language. Programmers who have SQL knowledge needed less effort to learn Pig Latin.

**Features of Apache Pig:**

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

## **Apache Pig Execution Modes**

## **Local Mode**

### In this mode, all the files are installed and run from your local host and local file system. There is no need of Hadoop or HDFS. This mode is generally used for testing purpose.

### **MapReduce Mode**

### MapReduce mode is where we load or process the data that exists in the Hadoop File System (HDFS) using Apache Pig. In this mode, whenever we execute the Pig Latin statements to process the data, a MapReduce job is invoked in the back-end to perform a particular operation on the data that exists in the HDFS.

## **Executing Pig Script in Batch mode**

While executing Apache Pig statements in batch mode, follow the steps given below.

### Step 1

Write all the required Pig Latin statements in a single file. We can write all the Pig Latin statements and commands in a single file and save it as **.pig** file.

### Step 2

Execute the Apache Pig script. You can execute the Pig script from the shell (Linux) as shown below.

|  |  |
| --- | --- |
| **Local mode** | **MapReduce mode** |
| $ pig -x local **Sample\_script.pig** | $ pig -x mapreduce **Sample\_script.pig** |

You can execute it from the Grunt shell as well using the exec command as shown below.

grunt> exec /sample\_script.pig

### Executing a Pig Script from HDFS

We can also execute a Pig script that resides in the HDFS. Suppose there is a Pig script with the name **Sample\_script.pig** in the HDFS directory named **/pig\_data/**. We can execute it as shown below.

$ pig -x mapreduce hdfs://localhost:9000/pig\_data/Sample\_script.pig

### **Example**

Assume we have a file **student\_details.txt** in HDFS with the following content.

**student\_details.txt**

001,Rajiv,Reddy,21,9848022337,Hyderabad

002,siddarth,Battacharya,22,9848022338,Kolkata

003,Rajesh,Khanna,22,9848022339,Delhi

004,Preethi,Agarwal,21,9848022330,Pune

005,Trupthi,Mohanthy,23,9848022336,Bhuwaneshwar

006,Archana,Mishra,23,9848022335,Chennai

007,Komal,Nayak,24,9848022334,trivendram

008,Bharathi,Nambiayar,24,9848022333,Chennai

We also have a sample script with the name **sample\_script.pig**, in the same HDFS directory. This file contains statements performing operations and transformations on the **student** relation, as shown below.

student = LOAD 'hdfs://localhost:9000/pig\_data/student\_details.txt' USING PigStorage(',')

as (id:int, firstname:chararray, lastname:chararray, phone:chararray, city:chararray);

student\_order = ORDER student BY age DESC;

student\_limit = LIMIT student\_order 4;

Dump student\_limit;

* The first statement of the script will load the data in the file named **student\_details.txt** as a relation named **student**.
* The second statement of the script will arrange the tuples of the relation in descending order, based on age, and store it as **student\_order**.
* The third statement of the script will store the first 4 tuples of **student\_order** as **student\_limit**.
* Finally the fourth statement will dump the content of the relation **student\_limit**.

Let us now execute the **sample\_script.pig** as shown below.

$./pig -x mapreduce hdfs://localhost:9000/pig\_data/sample\_script.pig

Apache Pig gets executed and gives you the output with the following content.

(7,Komal,Nayak,24,9848022334,trivendram)

(8,Bharathi,Nambiayar,24,9848022333,Chennai)

(5,Trupthi,Mohanthy,23,9848022336,Bhuwaneshwar)

(6,Archana,Mishra,23,9848022335,Chennai)

2015-10-19 10:31:27,446 [main] INFO org.apache.pig.Main - Pig script completed in 12

minutes, 32 seconds and 751 milliseconds (752751 ms)

**PRACTICAL 17: TO STUDY ABOUT SPARK**

Apache Spark is an open-source, distributed processing system used for big data workloads. It utilizes in-memory caching and optimized query execution for fast queries against data of any size. Simply put, Spark is a **fast and general engine for large-scale data processing**.

The **fast** part means that it’s faster than previous approaches to work with Big Data like classical [MapReduce](https://www.ibm.com/analytics/hadoop/mapreduce). The secret for being faster is that Spark runs on memory (RAM), and that makes the processing much faster than on disk drives.

The **general** part means that it can be used for multiple things like running distributed SQL, creating data pipelines, ingesting data into a database, running Machine Learning algorithms, working with graphs or data streams, and much more.

**WORKING**

Hadoop MapReduce is a programming model for processing big data sets with a parallel, distributed algorithm. Developers can write massively parallelized operators, without having to worry about work distribution, and fault tolerance. However, a challenge to MapReduce is the sequential multi-step process it takes to run a job. With each step, MapReduce reads data from the cluster, performs operations, and writes the results back to HDFS. Because each step requires a disk read, and write, MapReduce jobs are slower due to the latency of disk I/O.

Spark was created to address the limitations to MapReduce, by doing processing in-memory, reducing the number of steps in a job, and by reusing data across multiple parallel operations. With Spark, only one-step is needed where data is read into memory, operations performed, and the results written back—resulting in a much faster execution. Spark also reuses data by using an in-memory cache to greatly speed up machine learning algorithms that repeatedly call a function on the same dataset. Data re-use is accomplished through the creation of DataFrames, an abstraction over Resilient Distributed Dataset (RDD), which is a collection of objects that is cached in memory, and reused in multiple Spark operations. This dramatically lowers the latency making Spark multiple times faster than MapReduce, especially when doing machine learning, and interactive analytics.

## **Basic Syntax**

**Create a Rdd**

# 1. Read external data in Spark as Rdd  
rdd = sc.textFile("path")# 2. Create rdd from a list  
rdd = sc.parallelize(["id","name","3","5"])# 3. Dataframe to rdd  
rdd = df.rdd

**Create a dataframe**

# 1. Read in Spark as Dataframe directly   
# header and schema are optionaldf = sqlContext.read.csv("path", header = True/False, schema=df\_schema)# 2.1 rdd to dataframe with column names  
df = spark.createDataFrame(rdd,["name","age"])# 2.2 rdd to dataframe with schemafrom pyspark.sql.types import \*df\_schema = StructType([  
**...**  StructField("name", StringType(), True),  
**...**  StructField("age", IntegerType(), True)])df = spark.createDataFrame(rdd,df\_schema)

**Transformations:**

# Basic transformations:# 1. select. Index column by df.column\_name or use "column\_name"  
df.select(df.name)  
df.select("name")# 2. filter/where they are the same  
 df.filter(df.age>20)  
 df.filter("age>20")  
 df.where("age>20")  
 df.where(df.age>20)# 3. sort/orderBy   
 df.sort("age",ascending=False)  
 df.sort(df.age.desct())# 4. groupBy and agg  
df.groupBy("gender").agg(count("name"),avg("age"))# 5. join  
df1.join(df.2, (df1.x1 == df2.x1) & (df1.x2 == df2.x2),'left')# 6. create a new column from existing column  
df.withColumn("first\_name",split(name,'\_')[0])

**Write SQL in Spark**

1. Run sqlContext=sqlContext(sc) before create a dataframe  
2. Create a dataframe called df  
3. Run df.createOrReplaceTempView("table1") to create a temp table  
4. talbe2=sqlContext("select id, name from table1")If you are writing multiple lines, use """ like this:  
talbe2=sqlContext.sql("""  
select id, name   
from table1  
where age>20  
""")

**Actions:**

1. Show: show first n rows of a dataframe (but not rdd) without cells being truncated  
df.show(5, truncate=False)2. Take: display a list of first few rows of dataframe or rdd  
df.take(5)3. collect: collect all the data of a dataframe or rdd  
df.collect()4. count: count number of rows  
df.count()6. printSchema: show column names, data types and whether they are nullable.  
df.printSchema()7. cache : cache the data in memory if it's going to be reused a lot. Use unpersist() to uncache data and free memory.  
df.cache()  
df.unpersist()# Transformations and actions can be connected and executed in sequence from left to right.   
df1.filter("age>10").join(df2,df1.x==df2.y).sort("age").show()

## **Takeaways:**

1. Use and look for functions that used on dataframe instead of rdd if possible, as it’s faster on dataframe
2. Be careful when using collect() unless you need to download the whole data; use show/take
3. Cache the data if you are going to use it a lot later.

**PRACTICAL 18: TO EXECUTE SPARK IMPLEMENTED PROGRAM**

To illustrate RDD basics, consider the simple program below:

val lines = sc.textFile("data.txt")

val lineLengths = lines.map(s => s.length)

val totalLength = lineLengths.reduce((a, b) => a + b)

The first line defines a base RDD from an external file. This dataset is not loaded in memory or otherwise acted on: lines is merely a pointer to the file. The second line defines lineLengths as the result of a map transformation. Again, lineLengths is not immediately computed, due to laziness. Finally, we run reduce, which is an action. At this point Spark breaks the computation into tasks to run on separate machines, and each machine runs both its part of the map and a local reduction, returning only its answer to the driver program.

If we also wanted to use lineLengths again later, we could add:

lineLengths.persist()

before the reduce, which would cause lineLengths to be saved in memory after the first time it is computed.

### Passing Functions to Spark

Spark’s API relies heavily on passing functions in the driver program to run on the cluster. There are two recommended ways to do this:

* [Anonymous function syntax](http://docs.scala-lang.org/tour/basics.html#functions), which can be used for short pieces of code.
* Static methods in a global singleton object. For example, you can define object MyFunctions and then pass MyFunctions.func1, as follows:

object MyFunctions {

def func1(s: String): String = { ... }

}

myRdd.map(MyFunctions.func1)

Note that while it is also possible to pass a reference to a method in a class instance (as opposed to a singleton object), this requires sending the object that contains that class along with the method. For example, consider:

class MyClass {

def func1(s: String): String = { ... }

def doStuff(rdd: RDD[String]): RDD[String] = { rdd.map(func1) }

}

Here, if we create a new MyClass instance and call doStuff on it, the map inside there references the func1 method *of that MyClass instance*, so the whole object needs to be sent to the cluster. It is similar to writing rdd.map(x => this.func1(x)).

In a similar way, accessing fields of the outer object will reference the whole object:

class MyClass {

val field = "Hello"

def doStuff(rdd: RDD[String]): RDD[String] = { rdd.map(x => field + x) }

}

is equivalent to writing rdd.map(x => this.field + x), which references all of this. To avoid this issue, the simplest way is to copy field into a local variable instead of accessing it externally:

def doStuff(rdd: RDD[String]): RDD[String] = {

val field\_ = this.field

rdd.map(x => field\_ + x)

}

**EXAMPLE**

## Word Count Application

Once you have Spark installed and have it up and running, you can run the data analytics queries using Spark API.

These are simple commands to read the data from a text file and process it. We’ll look at advanced use cases of using Spark framework in the future articles in this series.

First, let’s use Spark API to run the popular Word Count example. Open a new Spark Scala Shell if you don’t already have it running. Here are the commands for this example.

import org.apache.spark.SparkContext

import org.apache.spark.SparkContext.\_

val txtFile = "README.md"

val txtData = sc.textFile(txtFile)

txtData.cache()

We call the cache function to store the RDD created in the above step in the cache, so Spark doesn’t have to compute it every time we use it for further data queries. Note that cache() is a lazy operation. Spark doesn’t immediately store the data in memory when we call cache. It actually takes place when an action is called on an RDD.

Now, we can call the count function to see how many lines are there in the text file.

txtData.count()

Now, we can run the following commands to perform the word count. The count shows up next to each word in the text file.

val wcData = txtData.flatMap(l => l.split(" ")).map(word => (word, 1)).reduceByKey(\_ + \_)

wcData.collect().foreach(println)