DEPARTMENT

OF

Computer Science and Engineering (AI&ML)

LAB MANUAL

B.TECH

CSE(AI&ML)

VII-SEMESTER

BIG DATA ANALYTICS LAB

(A8807) (VCE-R22)

VARDHAMAN COLLEGE OF ENGINEERING(AUTONOMOUS)

Affiliated to JNTUH, Approved by AICTE, Accredited by NAAC, with A++ Grade, ISO9001:2015 Certified

Shamshabad, Hyderabad–501218

2025- 26

VARDHAMAN COLLEGE OF ENGINEERING

(AUTONOMOUS)

Affiliated to **JNTUH**, Approved by **AICTE**, Accredited by **NAAC**, with **A++** Grade,
ISO9001:2015 Certified
Kacharam, Shamshabad, Hyderabad–501218

CERTIFICATE

This is to certify that the Bonafide of practical work carried out by ______________________________ Roll Number _____________________ of B.Tech in the **Big Data Analytics Laboratory (A8807)** submitted to the Department of Computer Science and Engineering (AI&ML), in partial fulfillment of the requirements for the award of degree of **Bachelor of Technology** in **Computer Science and Engineering (AI&ML)** during the year 2025-26.

No. of Experiments done:

Total no. of Experiments:

Date: HOD Staff Members Incharge

Roll Number:

Submitted for the Practical Exam held on:

Internal Examiner External Examiner

List of Experiments

S.No.	Date	Experiment details	Page No	Marks	Sign
1		**Hadoop Environment setup** • Write the steps to download, install and configure the Hadoop framework on Ubuntu/Linux and Windows operating systems.	12-15		
2		**Hadoop HDFS Commands** • Implement the following file management tasks in Hadoop framework using Cloudera: • Adding files and directories • Retrieving files • Deleting files	16-19		
3		**MapReduce Programming** Develop a WordCount Java program and implement in Hadoop MapReduce framework using Cloudera.	20-22		
4		**MapReduce Programming** • Develop a MapReduce program to search for a specific keyword in a file. • Develop a MapReduce program to sort data by student name (value).	23-26		
5		**Cassandra** • Implement keyspace operations to group column families together for the given application data. • Implement CRUD operations on the given dataset using Cassandra.	27		
6		**Cassandra** • Design a table/column family and perform various collection types Set, List and Map using Cassandra. • Design a table/column family and perform Alter table commands using Cassandra.	28		
7		**MongoDB** • Implement a program with basic commands on databases and collections using MongoDB. • Implement CRUD operations on the given dataset using MongoDB.	29-30		
8		**MongoDB** • Perform Count, Limit, Sort, and Skip operations on the given collections using MongoDB	31		

9		**Pig Latin commands** • Implement Relational operators –Loading and Storing, and Diagnostic operators -Dump, Describe, Illustrate & Explain on the given database in Hadoop Pig framework using Cloudera. • Develop a Pig Latin program to implement Filtering, Sorting operations on the given database.	32-34		
10		**Pig Latin commands** • Implement Grouping, Joining, Combining and Splitting operations on the given database using Pig Latin statements. • Perform Eval Functions on the given dataset. • Develop a WordCount program using Pig Latin statements.	35-40		
11		**Hive commands** • Implement Data Definition Language (DDL) Commands for databases in Hadoop Hive framework using Cloudera. • Implement Data Definition Language (DDL) Commands for tables in Hive.	41-45		
12		**Hive commands** • Implement Data Manipulation Language (DML) Commands for tables in Hive. • Perform data partitioning to split the given larger dataset into more meaningful chunks.	46-48		

VARDHAMAN
COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEEERING (AI&ML)

LAB MANUAL

Big Data Analytics Laboratory

IV Yr I SEM SEM B.TECH CSE (AI & ML)

Course Title	Big Data Analytics Laboratory
Course Type	Laboratory
Category	Core Engineering
Regulation	VCE-R22
Academic Year	2025-2026
Course Code	A8807

Course Instructors	
	CSE (AI&ML)
	Mr. Rama Chandra Rao Meka Ms. Pravallika B

DEPARTMENT VISSION

To be a premier center of excellence with research in artificial intelligence through unique interdisciplinary partnerships and positive contribution to the community, organizations and society as a whole.

DEPARTMENT MISSION

M1: To impart knowledge in cutting edge Artificial Intelligence technologies in par with industrial standards.

M2: To collaborate with industry to uplift innovative research and development in Artificial Intelligence and Machine Learning and its allied fields to serve the needs of society.

M3: To produce successful Computer Science and Engineering graduates with a specialization in AI/ML with personal and professional responsibilities and commitment to lifelong learning.

Program Educational Objectives (PEOs)

PEO1: Graduates will have the ability to adapt, contribute and innovate new technologies and systems in the key domains of Artificial Intelligence and Machine Learning.

PEO2: Graduates will be able to successfully pursue higher education in reputed institutions with AI Specialization.

PEO3: Graduates will have the ability to explore research areas and produce outstanding contribution in various areas of Artificial Intelligence and Machine Learning.

PEO4: Graduates will be ethically and socially responsible solution providers and entrepreneurs in the field of Computer Science and Engineering with AI/ML Specialization.

United Nations Sustainable Development Goals (SDGs)

SDG1: No Poverty - End poverty in all its forms everywhere.

SDG2: Zero Hunger - End hunger, achieve food security and improved nutrition and promote sustainable agriculture.

SDG3: Good Health and Well-Being - Ensure healthy lives and promote well-being for all at all ages.

SDG4: Quality Education -Ensure inclusive and equitable quality education and promote lifelong learning opportunities for all.

SDG5: Gender Equality - Achieve gender equality and empower all women and girls.

SDG6: Clean Water and Sanitation -Ensure availability and sustainable management of water and sanitation for all.

SDG7: Affordable and Clean Energy -Ensure access to affordable, reliable, sustainable and modern energy for all.

SDG8: Decent Work and Economic Growth - Promote sustained, inclusive and sustainable economic growth, full and productive employment and decent work for all.

SDG9: Industry, Innovation and Infrastructure -Build resilient infrastructure, promote inclusive and sustainable industrialization and foster innovation.

SDG10: Reduced Inequalities - Reduce inequality within and among countries.

SDG11: Sustainable Cities and Communities -Make cities and human settlements inclusive, safe, resilient and sustainable.

SDG12: Responsible Consumption and Production - Ensure sustainable consumption and production patterns.

SDG13: Climate Action - Take urgent action to combat climate change and its impacts.

SDG14: Life Below Water -Conserve and sustainably use the oceans, seas and marine resources for sustainable development.

SDG15: Life on Land - Protect, restore and promote sustainable use of terrestrial ecosystems, sustainably manage forests, combat desertification, halt and reverse land degradation, and halt biodiversity loss.

SDG16: Peace, Justice and Strong Institutions -Promote peaceful and inclusive societies for sustainable development, provide access to justice for all and build effective, accountable and inclusive institutions at all levels.

SDG17: Partnerships for the Goals - Strengthen the means of implementation and revitalize the global partnership for sustainable development.

COURSE OVERVIEW

The key objective of this course is to familiarize the students with most important information technologies used in manipulating, storing, and analyzing big data with low latency. The course gives insights of the modern big data tools like Cassandra, MongoDB, Pig and Hive that allows users to make better and faster decisions.

COURSE OUTCOMES (COs)

After the completion of the course, the student will be able to:

CO#	Course Outcomes	POs	PSOs
A8807.1	Implement HDFS commands on file management tasks.	-	-
A8807.2	Use of MapReduce Programming to process massive amounts of data in parallel.	1,5	1,2
A8807.3	Use NoSQL databases like MangoDB and Cassandra to stock log data to be pulled for analysis.	2,5	1,2
A8807.4	Implement Pig programs for complex data flow and analysis.	1,5	1,2
A8807.5	Implement Hive programs for complex data flow and analysis.	3,5	1,2

PROGRAM OUTCOMES (POs)

PO#	Program Specific Outcomes
PO-1	Engineering Knowledge
PO-2	Problem Analysis
PO-3	Design/Development of Solutions
PO-5	Engineering Tool Usage

PROGRAM SPECIFIC OUTCOMES (PSOs)

PSO#	Program Specific Outcomes
PSO-1	Apply the knowledge of Artificial Intelligence to design, develop, and evaluate computational solutions for complex problems in diverse domains, such as healthcare, finance, and automation.
PSO-2	Demonstrate expertise in using advanced ML tools, techniques, and frameworks to develop innovative solutions for data analysis, pattern recognition, and intelligent decision-making system

BLOOM’S LEVEL OF THE COURSE OUTCOMES

CO#	Bloom’s Level					
	Remember (L1)	Understand (L2)	Apply (L3)	Analyze (L4)	Evaluate (L5)	Create (L6)
A8807.1		✓				
A8807.2			✓			
A8807.3			✓			
A8807.4			✓			
A8807.5			✓			

COURSE ARTICULATION MATRIX

CO#/ POs	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2
A8807.1		3												
A8807.2				3`									2	
A8807.3					3								2	
A8807.4					3								2	
A8807.5					3								2	

Note: 1-Low, 2-Medium, 3-High

COURSE ASSESSMENT

S.NO#	EVALUATION METHOD	ASSESSMENT TOOL	Max. Marks	
			Marks	Total
1	Continuous Internal Evaluation (CIE)	Internal practical examination	40	40
2	Semester End Examination (SEE)	Write-up (Program1 & Program2)	20	60
		Evaluation of results (Program1 & Program2)	30	
		Viva-Voce	10	

LIST OF PROGRAMS

S.No	Title of the Experiment	Tools and Techniques	Expected Skills/Ability
1	**Hadoop Environment setup** • Write the steps to download, install and configure the Hadoop framework on Ubuntu/Linux and Windows operating systems.	OS: Windows / Linux Open-source Analytical Tools: Hadoop, Cassandra, MongoDB, Pig, Hive Open-source IDE: Cloudera Web browser: Internet Explorer/ Google Chrome/ Mozilla Firefox	Basic Hadoop Environment
2	**Hadoop HDFS Commands** • Implement the following file management tasks in Hadoop framework using Cloudera: • Adding files and directories • Retrieving files • Deleting files		Basic hdfs - file commands.
3	**MapReduce Programming** • Develop a WordCount Java program and implement in Hadoop MapReduce framework using Cloudera.		Develop programs using MapReduce Programming
4	**MapReduce Programming** • Develop a MapReduce program to search for a specific keyword in a file. • Develop a MapReduce program to sort data by student name (value).		
5	**Cassandra** • Implement keyspace operations to group column families together for the given application data. • Implement CRUD operations on the given dataset using Cassandra.		Implement CRUD operations on the given dataset using Cassandra
6	**Cassandra** • Design a table/column family and perform various collection types Set, List and Map using Cassandra. • Design a table/column family and perform Alter table commands using Cassandra.		

S.No	Title of the Experiment	Tools and Techniques	Expected Skills/Ability
7	**MongoDB** • Implement a program with basic commands on databases and collections using MongoDB. • Implement CRUD operations on the given dataset using MongoDB.	OS: Windows / Linux Open-source Analytical Tools: Hadoop, Cassandra, MongoDB, Pig, Hive Open-source IDE: Cloudera Web browser: Internet Explorer/ Google Chrome/ Mozilla Firefox	Implement CRUD operations on the given dataset using MongoDB
8	**MongoDB** • Perform Count, Limit, Sort, and Skip operations on the given collections using MongoDB		
9	**Pig Latin commands** • Implement Relational operators –Loading and Storing, and Diagnostic operators -Dump, Describe, Illustrate & Explain on the given database in Hadoop Pig framework using Cloudera. • Develop a Pig Latin program to implement Filtering, Sorting operations on the given database.		Implement database operations using Pig tool.
10	**Pig Latin commands** • Implement Grouping, Joining, Combining and Splitting operations on the given database using Pig Latin statements. • Perform Eval Functions on the given dataset. • Develop a WordCount program using Pig Latin statements.		
11	**Hive commands** • Implement Data Definition Language (DDL) Commands for databases in Hadoop Hive framework using Cloudera. • Implement Data Definition Language (DDL) Commands for tables in Hive.		Implement DDL & DML Commands for databases and tables using Hive tool.
12	**Hive commands** • Implement Data Manipulation Language (DML) Commands for tables in Hive. • Perform data partitioning to split the given larger dataset into more meaningful chunks.		

Introduction to Hadoop Framework

1.1 Hadoop

- **Hadoop** is a collection of open-source software utilities that facilitates using a network of many computers to solve problems involving massive amounts of data and computation.
- The core of Apache Hadoop software framework consists of a storage part, known as **Hadoop Distributed File System (HDFS),** and a processing part which is a **MapReduce** programming model.

1.2 Key Aspects of Hadoop

- Open-source software
- Framework
- Distributed
- Massive storage
- Faster processing

1.3 Hadoop Core Components

1) **HDFS**
   a) Storage component
   b) Distributes data across several nodes
   c) Natively redundant
2) **MapReduce**
   a) Computational framework
   b) Splits a task across multiple nodes
   c) Processes data in parallel

1.4 Hadoop Ecosystem:

Hadoop Ecosystem are support projects to enhance the functionality of Hadoop Core Components.

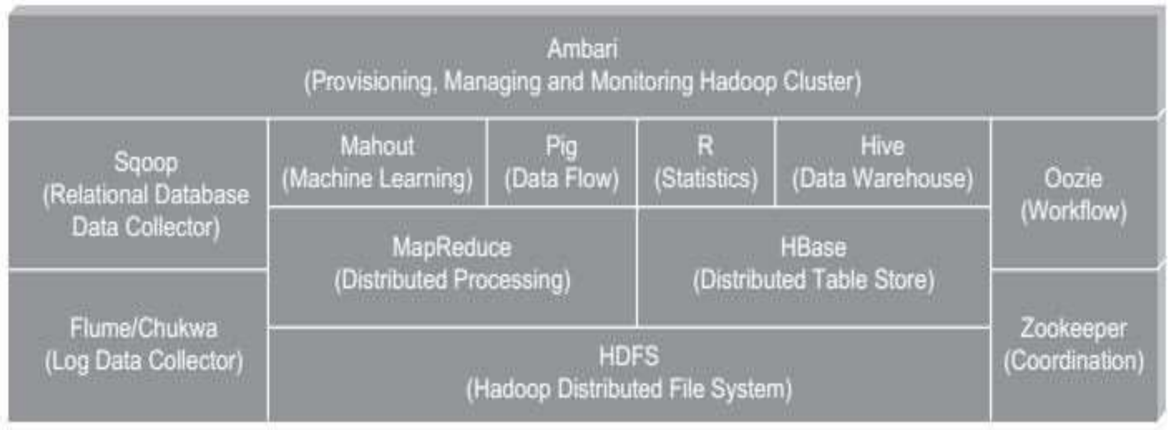


Fig 1a-: Hadoop Ecosystem

1.5 High-Level Architecture of Hadoop

- It is a distributed Master-Slave Architecture.
- Master node is known as NameNode and
- Slave nodes are known as DataNodes.

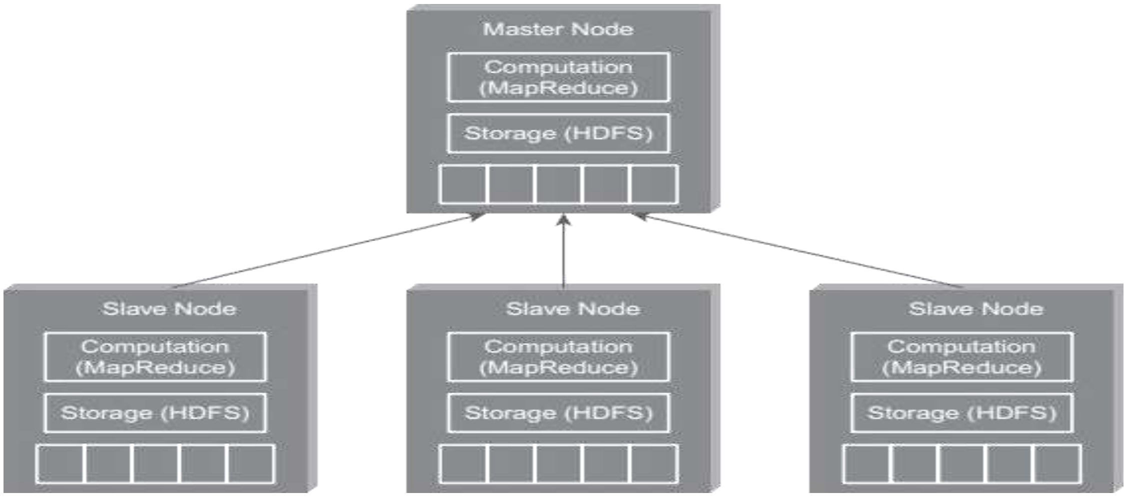
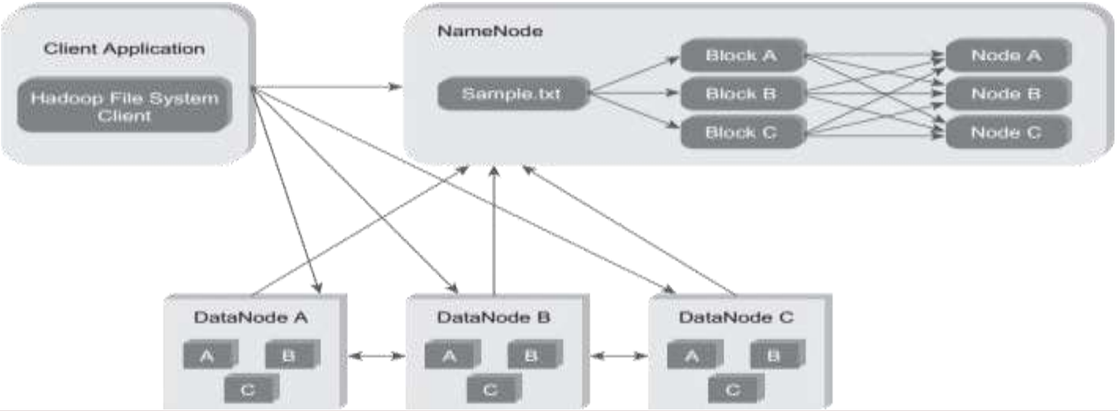


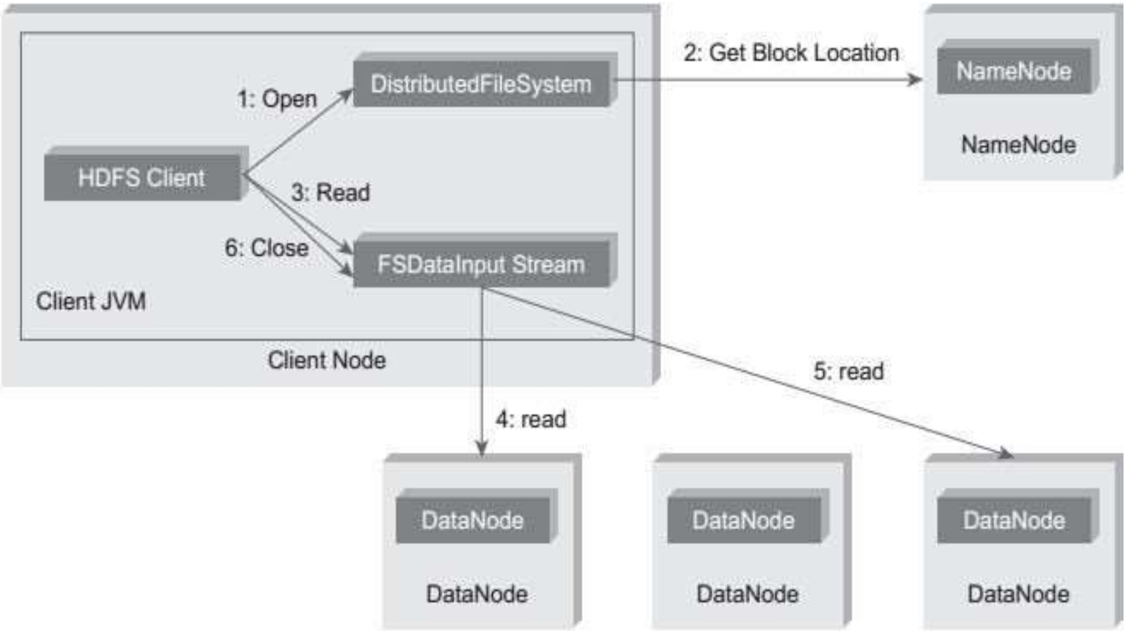
Fig 1b-: High-Level Architecture of Hadoop

1.6 Hadoop Distributed File System Architecture and HDFS Daemons

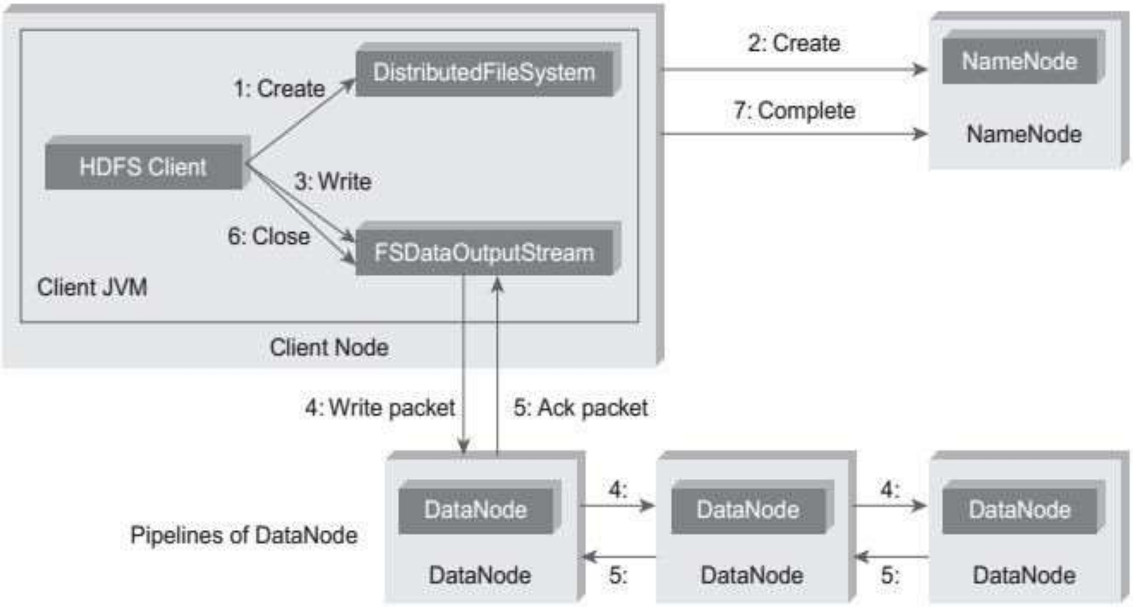
- Client Application interacts with NameNode for metadata related activities and communicates with DataNodes to read and write files.
- **Datanodes** are the slave nodes that divides the input files of varied formats into blocks and store the actual data. DataNodes converse with each other for pipeline reads and writes.
- **NameNode** is the master node that manages the File System Namespace, controlling the client's access to file-related operations such as read, write, create, delete and naming files and directories. When NameNode starts up, it reads FsImage and EditLog from disk.
- If the NameNode has not restarted for months, the **Secondary NameNode** applies edits log on FSImage at regular intervals.
- Hadoop 1.x can configure to 64MB while Hadoop 2.x and Hadoop 3.x cluster can have 64MB/ 128MB / 256MB/ 512 MB. Hadoop Administrator have control over block size to be configured for Cluster.



1.7 Anatomy of File Read



1.8 Anatomy of File Write



Week-1	Hadoop Environment setup: Write the steps to download, install and configure the Hadoop framework on Ubuntu Linux and Windows operating systems.

Problem Statement: Installation of VirtualBox and Cloudera to setup the Hadoop environment and its ecosystems.

Prerequisite: Java JDK of any version should be installed on the system.

Program Steps:

Step1: **Open the VirtualBox website:** Go to https://www.virtualbox.org/ in your computer's Internet browser. This is the website from which you'll download the VirtualBox setup file.

Step2: **Click Download VirtualBox:** It's a blue button in the middle of the page. Doing so will open the downloads page.

Step3: **Click Windows hosts:** You'll see this link below the "VirtualBox 6.1.32 platform packages" heading. The VirtualBox EXE file will begin downloading onto your computer.

Step4: **Open the VirtualBox EXE file:** Go to the location to which the EXE file downloaded and double-click the file. Doing so will open the VirtualBox installation window.

Step5: **Navigate through the installation prompts:** Do the following:

- Click **Next** on the first three pages.
- Click **Yes** when prompted.
- Click **Install**
- Click **Yes** when prompted.

Step6: **Click Install when prompted:** Doing so will allow VirtualBox to begin installing on your computer.

Step7: **Click Finish when prompted:** It's in the lower-right side of the window. Doing so will close the installation window and open VirtualBox. Now that you've installed and opened VirtualBox, you can create a virtual machine to run any operating system on your PC.

Step8: **Open Virtual Box window:** After completion of installation process, the VirtualBox window gets opened.

Step9: To **set up the Cloudera QuickStart VM** in your Oracle VirtualBox Manager, click on 'File' and then select 'Import Appliance'.

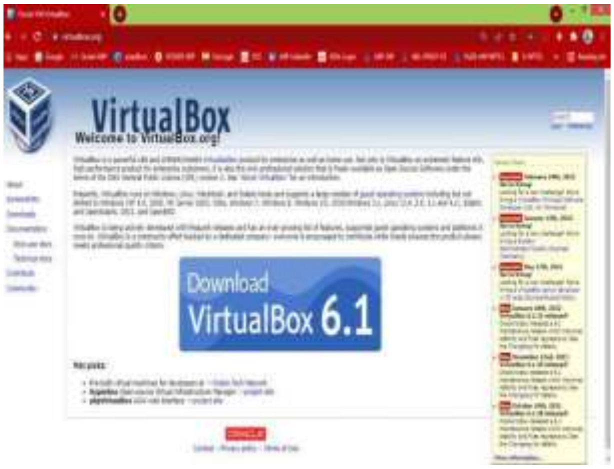
Step10: **Import Cloudera QuickStart VM image:** Choose the QuickStart VM image by looking into your downloads. Click on 'Open' and then 'Next'. Now you can see the specifications, then click on 'Import'. This will start importing the virtual disk image .vmdk file into your VM box. Importing takes several minutes.

Step11: **Cloudera QuickStart VM for practice in VirtualBox:** Once the importing is complete, you can see the Cloudera QuickStart VM on the left side panel.

Step12: **Ready with Cloudera QuickStart VM:** Now you are required to start the machine by clicking the 'Start' symbol on top and brings up the Cloudera QuickStart VM Desktop Environment.

Conclusion: The installation of VirtualBox and Cloudera to setup the Hadoop environment and its ecosystems is successfully done.

Screenshots for Installation of VirtualBox and Cloudera to setup Hadoop

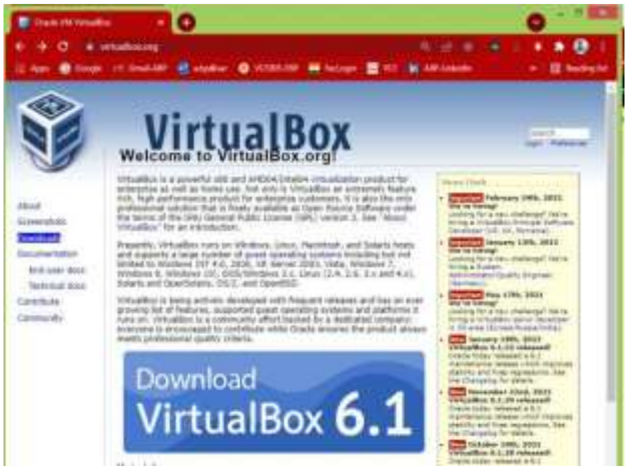


VirtualBox

Welcome to VirtualBox.org!

Download VirtualBox 6.1

Step1: Open the VirtualBox website:

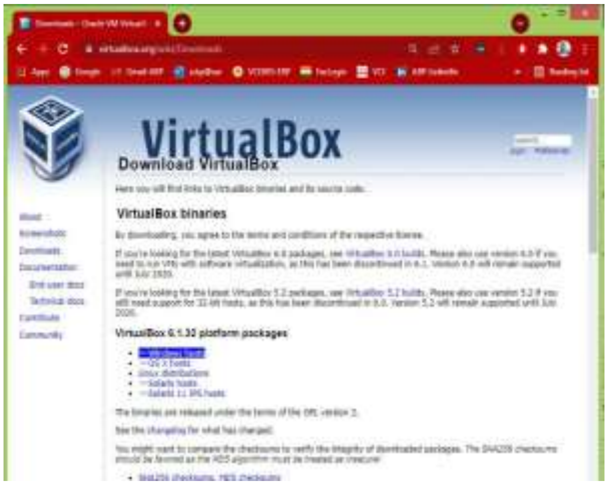


VirtualBox

Welcome to VirtualBox.org!

Download VirtualBox 6.1

Step2: Click Download VirtualBox:



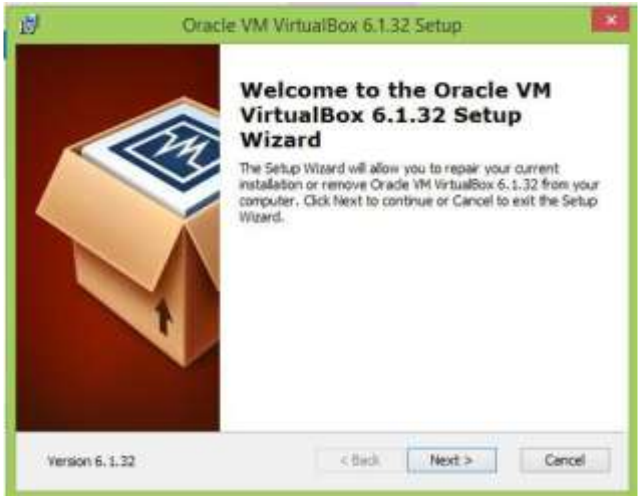
VirtualBox

Download VirtualBox

VirtualBox binaries

VirtualBox 6.1.32 platform packages

Step3: Click Windows hosts:



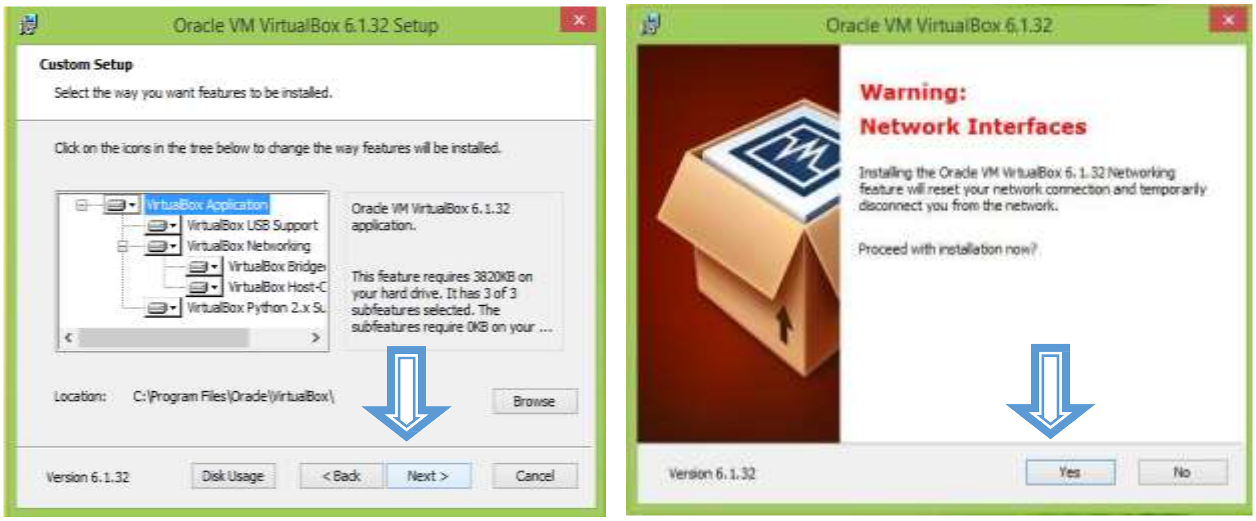
Oracle VM VirtualBox 6.1.32 Setup

Welcome to the Oracle VM VirtualBox 6.1.32 Setup Wizard

The Setup Wizard will allow you to repair your current installation or remove Oracle VM VirtualBox 6.1.32 from your computer. Click Next to continue or Cancel to exit the Setup Wizard.

Version 6.1.32

Step4: Open the VirtualBox EXE file



Oracle VM VirtualBox 6.1.32 Setup

Custom Setup

Select the way you want features to be installed.

Click on the icons in the tree below to change the way features will be installed.

Oracle VM VirtualBox 6.1.32 application.

This feature requires 3820KB on your hard drive. It has 3 of 3 subfeatures selected. The subfeatures require 0KB on your ...

Location: C:\Program Files\Oracle\VirtualBox\

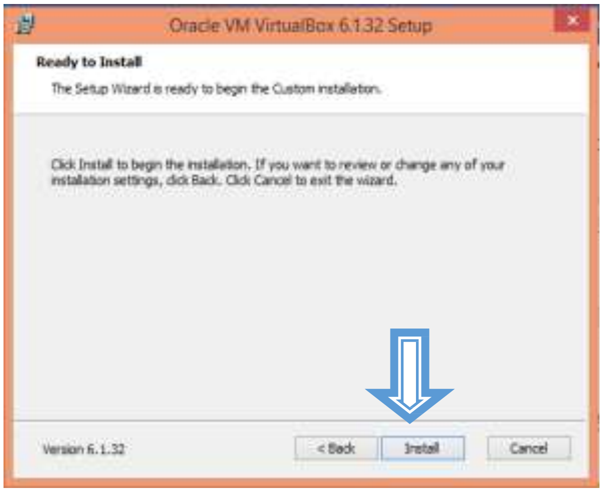
Oracle VM VirtualBox 6.1.32

Warning: Network Interfaces

Installing the Oracle VM VirtualBox 6.1.32 Networking feature will reset your network connection and temporarily disconnect you from the network.

Proceed with installation now?

Step5: Navigate through the installation prompts

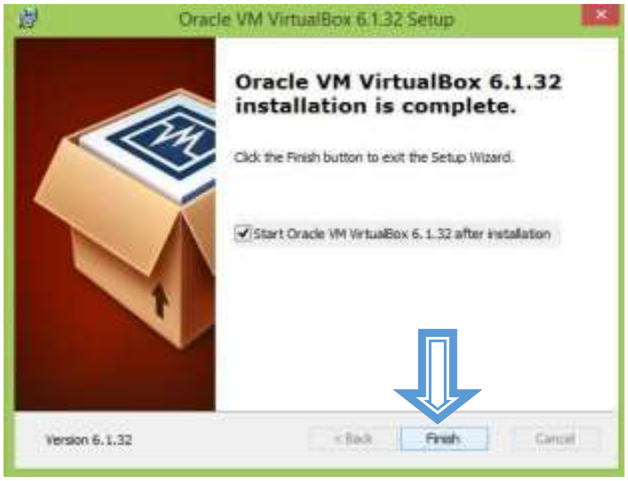


Oracle VM VirtualBox 6.1.32 Setup

Ready to Install

The Setup Wizard is ready to begin the Custom installation.

Click Install to begin the installation. If you want to review or change any of your installation settings, click Back. Click Cancel to exit the wizard.



Oracle VM VirtualBox 6.1.32 Setup

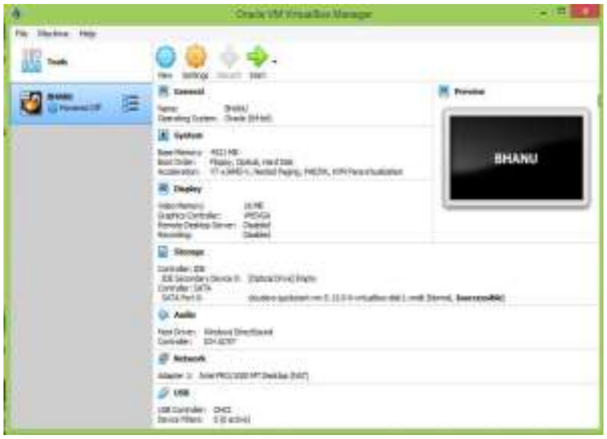
Oracle VM VirtualBox 6.1.32 installation is complete.

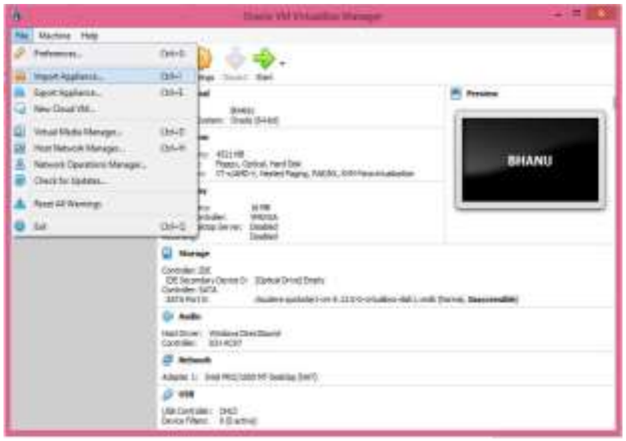
Click the Finish button to exit the Setup Wizard.

☑ Start Oracle VM VirtualBox 6.1.32 after installation

Step6: Click Install when prompted

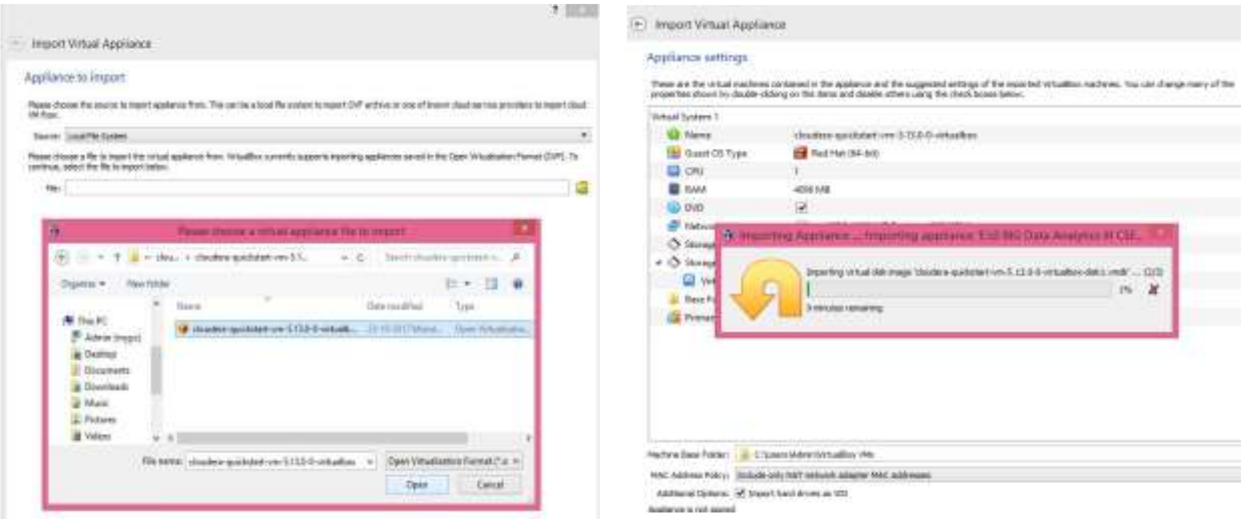
Step7: Click Finish when prompted





Step8: Open Virtual Box window

Step9: Set up the Cloudera QuickStart VM


Import Virtual Appliance

Appliance to import

Source: Local File System

Please choose a virtual appliance file to import

File name: cloudera-quickstart-vm-5.13.0-0-virtualbox

Open Virtualization Format

Open Cancel

Import Virtual Appliance

Appliance settings

Virtual System 1

Name cloudera-quickstart-vm-5.13.0-0-virtualbox

Guest OS Type Red Hat (64-bit)

CPU 1

RAM 4096 MB

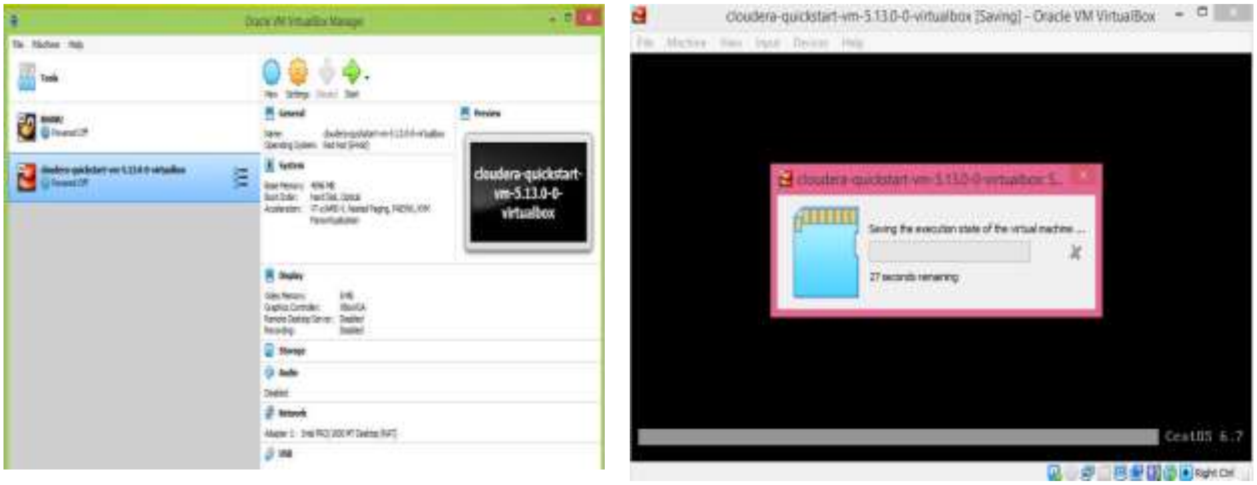
DVD ☑

Importing Appliance ... Importing appliance

1%

Machine Base Folder: C:\Users\Admin\VirtualBox VMs

Step10: Import Cloudera QuickStart VM image


Oracle VM VirtualBox Manager

cloudera-quickstart-vm-5.13.0-0-virtualbox

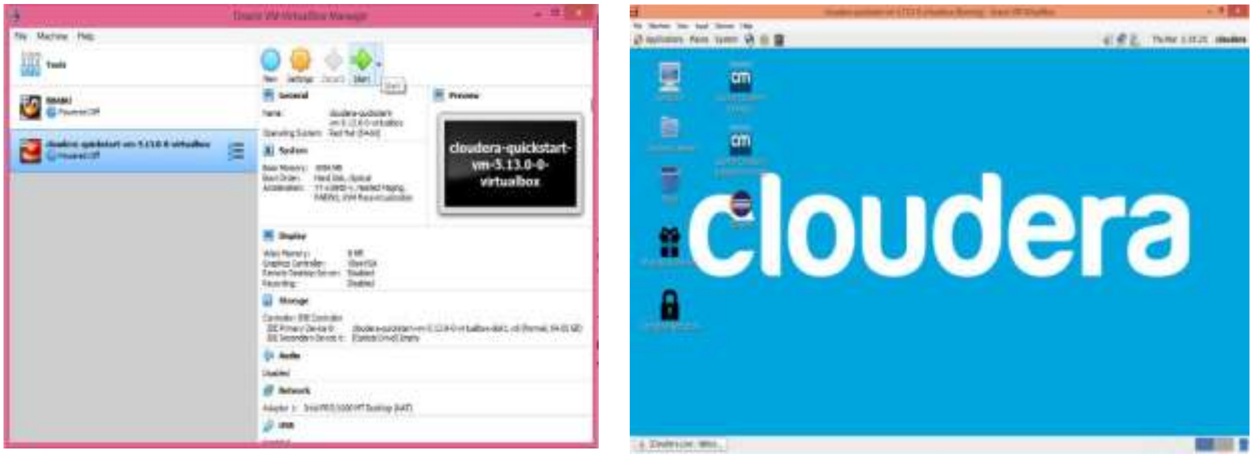
cloudera-quickstart-vm-5.13.0-0-virtualbox [Saving] - Oracle VM VirtualBox

Saving the execution state of the virtual machine ...

27 seconds remaining

CentOS 6.7

Step11: Cloudera QuickStart VM inVirtualBox


Oracle VM VirtualBox Manager

cloudera-quickstart-vm-5.13.0-0-virtualbox

Step12: Ready with Cloudera QuickStart VM

Week-2

Hadoop HDFS Commands

Implement the following file management tasks in Hadoop framework using Cloudera:

- Adding files and directories
- Retrieving files
- Deleting files

Prerequisite: VirtualBox, Cloudera QuickStart.

Basic Information:

- HDFS is a scalable distributed filesystem designed to scale to petabytes of data while running on top of the underlying filesystem of the operating system.
- HDFS keeps track of where the data resides in a network by associating the name of its rack (or network switch) with the dataset. Hadoop provides a set of command line utilities that work similarly to the Linux file commands and serve as your primary interface with HDFS.
- We‘re going to have a look into HDFS by interacting with it from the command line.
- We will take a look at the most common file management tasks in Hadoop, which include:
  - Adding files and directories to HDFS
  - Retrieving files from HDFS to local filesystem
  - Deleting files from HDFS
- There are many more commands in **"$HADOOP_HOME/bin/hadoop fs**" than are demonstrated here, although these basic operations will get you started.
- Running **./bin/hadoop dfs** with no additional arguments will list all the commands that can be run with the FsShell system.
- **$HADOOP_HOME/bin/hadoop fs -help commandName** will display a short usage summary for the operation.
- The following conventions are used for parameters:
  - **"<path>"** means any file or directory name.
  - **"<path>..."** means one or more file or directory names.
  - **"<file>"** means any filenam e.
  - **"<src>"** and **"<dest>"** are path names in a directed operation.
  - **"<localSrc>"** and **"<localDest>"** are paths as above, but on the local file system.
  - **“<hdfsSrc>"** and **"<hdfsDest>"** are paths as above, but on the Hadoop distributed file system.

hadoop fs <args>

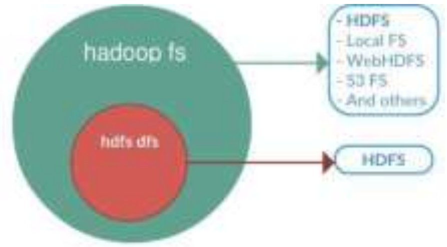
fs is used for a generic **F**ile **S**ystem and it can point to any file system such as a local file system, HDFS, WebHDFS, S3FS, etc.

hdfs dfs <args>

dfs points to the **D**istributed **F**ile **S**ystem and it is specific to execute operations on HDFS. **Note: hadoop dfs** is deprecated and it will be directed to use hdfs dfs.

Below is the list categorized as **hdfs** commands:

namenode| secondarynamenode| datanode| dfs| dfsadmin| fsck| balancer| fetchdt| oiv| dfsgroups

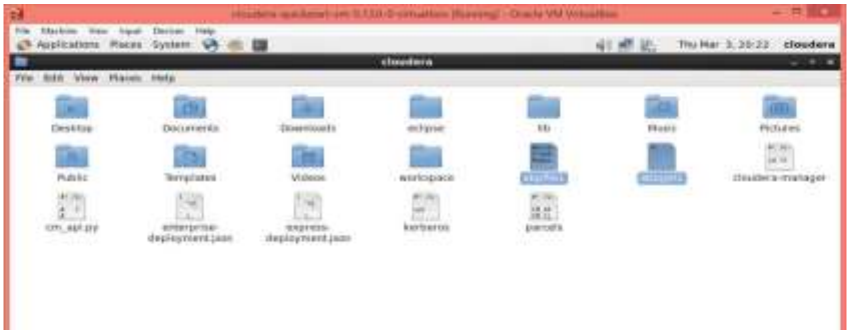


Program Steps:

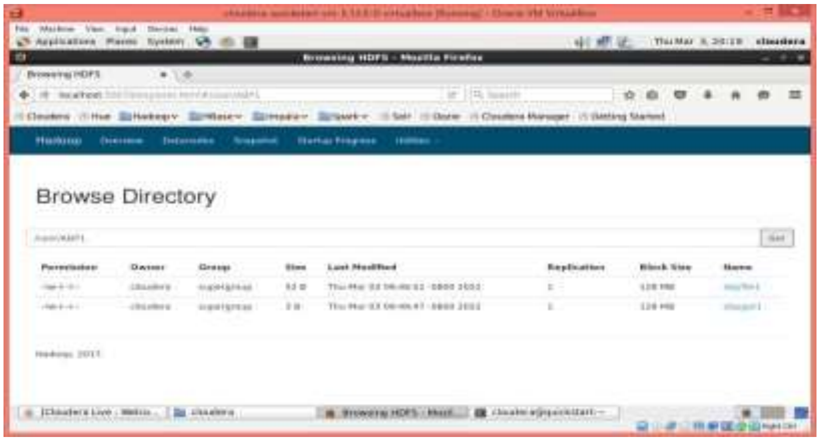
S.No.	**Create files and directories**
	➢ Before you can run Hadoop programs on data stored in HDFS, you‘ll need to put the data into HDFS first. ➢ Let‘s create a directory and put a file in it. HDFS has a default working directory of /user/$USER, where $USER is your login user name. ➢ This isn‘t automatically created for you, though, so let‘s create it with the mkdir command directory. ➢ For the purpose of illustration, we use chuck. You should substitute your user name in the example commands.
1.	**–mkdir <path/FolderName>** To create a named directory in given path of HDFS. In Hadoop dfs there is no home directory by default. ➢ hdfs dfs –mkdir /ABP ➢ hdfs dfs –mkdir /ABP/abpsubdir =creates sub directory ➢ hdfs dfs –mkdir /user/ABP1 = creates sub directory in user
2.	**Create new file with content in local file system (files present on OS). gedit abpfile1** Create a file with name **“abpfile1”**, type the content in it. **Similarly create abpget1 file.** To view files of local file system, go to **cloudera** desktop double click **cloudera’s home** icon. Its path for programming is **/home/cloudera/**
A)	**Adding files and directories to HDFS**
3.	**–put <localSrc> <hdfsDest>** To copy file/folder from local file system to HDFS store. Both files exist ➢ hdfs dfs –put /home/cloudera/abpfile1 /user/ABP1/ =file copied and exists in both locations ➢ hdfs dfs –ls –R user/ABP1/ =check for destination location file

4.	**–copyFromLocal <localSrc> <hdfsDest> (Identical to –put)** To copy a file/folder from local file system to HDFS store. Both files exist ➢ hdfs dfs –copyFromLocal /home/cloudera/abpfile1 /user/ABP1/
5.	**–moveFromLocal <localSrc> <hdfsDest>** To move a file/folder from local file system to HDFS store. Works like –put, but deletes moved file/folder from lfs, and exists only in HDFS store. ➢ hdfs dfs –moveFromLocal /home/cloudera/abpget1 /user/ABP1/
B)	**Retrieving files from HDFS to local filesystem**
6.	**–get <hdfsSr> <localDest>** To copy a file/folder from HDFS to local file system. Both files exist. ➢ hdfs dfs –get /user/ABP1/abpfile1 /home/cloudera/abpfile2
7.	**–copyToLocal <hdfsSrc> <localDest> (Identical to –get)** To copy a file/folder from HDFS to local file system. Both files exist ➢ hdfs dfs –copyToLocal /user/ABP1/abpget1 /home/cloudera/
8.	**–moveToLocal <hdfsSrc> <localDest>** To move a file/folder from HDFS store to local file system. Works like -get, but deletes moved file/folder from HDFS store, and exists only in lfs. ➢ hdfs dfs –moveToLocal /user/ABP1/abpget1 /home/cloudera/
9.	**–cp <hdfsSrc> <hdfsDest>** To copy the file or directory from given source to destination within HDFS. ➢ hdfs dfs –cp /A123 /user/
10.	**–mv <hdfsSrc> <hdfsDest>** To move the file from the specified source to destination within HDFS. ➢ hdfs dfs –mv /user/ABP1/abpfile1 /A123
11.	**–cat <filen-ame>** To display the contents of an HDFS file on console. ➢ hdfs dfs –cat /home/cloudera/abpget1
C)	**Deleting files from**
12.	**–rm -r <hdfsFilename/hdfsDirectoryName>** Deletes a file from HDFS recursively. hdfs dfs –rm /user/A123/abpfile1 =removes file hdfs dfs –rm -r /user/ABP1 = recursively removes directory and all its contents

Screenshots for Hadoop File Management Tasks


cloudera-quickstart-vm-5.13.0-0-virtualbox [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Applications Places System
Thu Mar 3, 20:22 cloudera
cloudera
File Edit View Places Help
Desktop
Documents
Downloads
eclipse
lib
Music
Pictures
Public
Templates
Videos
workspace
abpfile1
abpget1
cloudera-manager
cm_api.py
enterprise-deployment.json
express-deployment.json
kerberos
parcels

Adding Files to HDFS gedit abpfile1
gedit abpget1

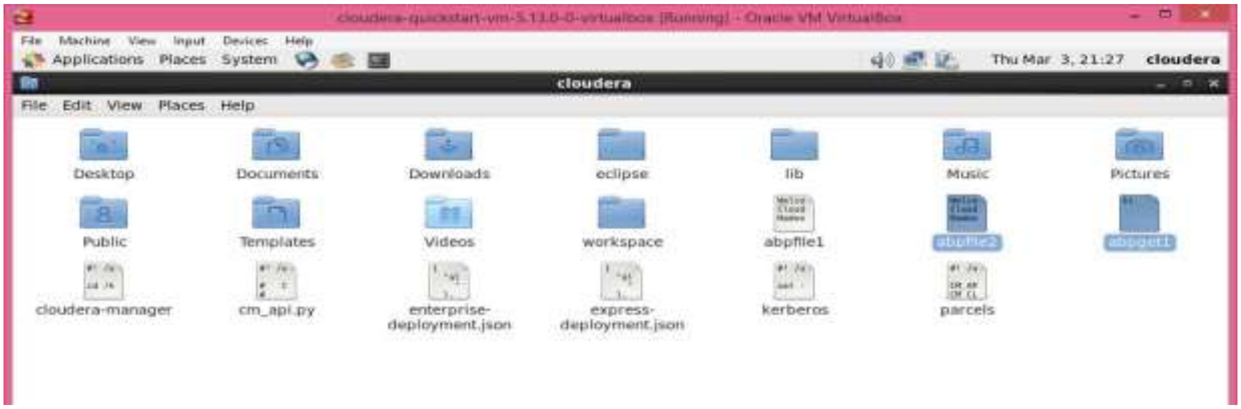

cloudera-quickstart-vm-5.13.0-0-virtualbox [Running] - Oracle VM VirtualBox
Browsing HDFS - Mozilla Firefox
Browsing HDFS
Cloudera Hue Hadoop HBase Impala Spark Solr Oozie Cloudera Manager Getting Started
Hadoop Overview Datanodes Snapshot Startup Progress Utilities
Go!
Hadoop, 2017.

Browse Directory

/user/ABP1

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r--	cloudera	supergroup	32 B	Thu Mar 03 06:46:12 -0800 2022	1	128 MB	abpfile1
-rw-r--r--	cloudera	supergroup	3 B	Thu Mar 03 06:46:47 -0800 2022	1	128 MB	abpget1

Adding Files to HDFS hdfs dfs –put /home/cloudera/abpfile1 /user/ABP1/
hdfs dfs –copyFromLocal /home/cloudera/abpget1 /user/ABP1/


cloudera-quickstart-vm-5.13.0-0-virtualbox [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Applications Places System
Thu Mar 3, 21:27 cloudera
cloudera
File Edit View Places Help
Desktop
Documents
Downloads
eclipse
lib
Music
Pictures
Public
Templates
Videos
workspace
abpfile1
abpfile2
abpget1
cloudera-manager
cm_api.py
enterprise-deployment.json
express-deployment.json
kerberos
parcels

Retrieving files from HDFS to local filesystem
hdfs dfs –get /user/ABP1/abpfile1 /home/cloudera/abpfile2 hdfs dfs
–copyToLocal /user/ABP1/abpget1 /home/cloudera/

Week-3	MapReduce Programming • Develop a WordCount Java program and implement in Hadoop MapReduce framework using Cloudera.

Program Steps:

Step-1: Open Virtual box and then start **cloudera quickstart**

Step-2: Open **Eclipse** present on the cloudera desktop

Step-3: Creating Java Project

3.1 : File-> New -> Project -> **Java Project** -> Next

("WordCount" is the Project name)

Step-4: Adding the Hadoop libraries to the Project

4.1 : Click on **Add External Jars** button, then, File File System > usr > lib> **Hadoop**, select all the libraray (jar) files and then click OK button

4.2 : Now Click on Add External Jars button, then, File File System > usr > lib> Hadoop>**client**, select all the libraray (jar) files and then click OK button

4.3 : Click **finish** button

Step-5: Create Java MapperReduce program

5.1 : In the explorer panel Right click on "src" folder of the project WordCount

New> Class> **Name** textfield give as "WordCount" and click Finish button.

5.2 : Type the code for WordCount program with import files, Mapper class, Reducer class Driver class with main method

Step-6: Export the project as JAR

6.1 : Right click WordCount project and select "Export" >> Java >> Jar file >> Next>> in the JAR file textfield give as /home/cloudera/WordCount.jar, click Finish button>> OK

6.2 : Open **cloudera@quickstart** terminal and verify the jar file using **ls** command Step-

7: Create the input file for the MapReduce program by typing command

cat > /home/cloudera/inputFile.txt

Verify the data contents by **cat /home/cloudera/inputFile.txt**

(if not available, it will be created. Type the text of words in quick start.

Step-8: Move the input file created in local system to hdfs store by **Hdfs**

dfs -put /home/cloudera/inputFile.txt /WCInput/ View the

contents of the file moved to hdfs by typing command **hdfs dfs -**

cat /WCInput/ inputFile.txt

Step-9: Run MapReduce program on Hadoop by typing command

```
hadoop jar /home/cloudera/WordCount.jar WordCount /WCInput/inputFile.txt
/WCOutput
```

Each time you run the above command; you need to give different name for the output directory.

Step-10: View the output directory content by hdfs dfs -ls /WCOutput of the program/job executed,

```
hdfs dfs -cat /WCOutput/part-r-00000
```

Program Source Code:

```
//word count java program code using mapreduce in hadoop framework
import java.io.IOException;
import java.util.StringTokenizer;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class WordCount          //driver class
{   //Mapper class
    public static class TokenizerMapper extends Mapper<Object, Text, Text, IntWritable>
    { private final static IntWritable one = new IntWritable(1); private
      Text word = new Text();
      public void map(Object key, Text value, Context context) throws IOException,
      InterruptedException
      { StringTokenizer itr = new StringTokenizer(value.toString()); while
        (itr.hasMoreTokens())
        {   word.set(itr.nextToken());
            context.write(word, one);
       }
      }
    } //End Mapper class
    //Reducer class
    public static class IntSumReducer extends Reducer<Text,IntWritable,Text,IntWritable>
    {  private IntWritable result = new IntWritable();
      public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException,
      InterruptedException
      {  int sum = 0;
        for (IntWritable val : values)
        {      sum += val.get();
        }
        result.set(sum);
        context.write(key, result);
      }
    } //End Reducer class


    //Driver class main()
    public static void main(String[] args) throws Exception
    {  Configuration conf = new Configuration();  Job
      job = Job.getInstance(conf, "word count");
      job.setJarByClass(WordCount.class);
      job.setMapperClass(TokenizerMapper.class);
      job.setCombinerClass(IntSumReducer.class);
      job.setReducerClass(IntSumReducer.class);
      job.setOutputKeyClass(Text.class);
      job.setOutputValueClass(IntWritable.class);
      FileInputFormat.addInputPath(job, new Path(args[0]));
      FileOutputFormat.setOutputPath(job, new Path(args[1]));
      System.exit(job.waitForCompletion(true) ? 0 : 1);
    } //End main()
} //End WordCount class
```

Week-4	**MapReduce Programming** • Develop a MapReduce program to search for a specific keyword in a file. • Develop a MapReduce program to sort data by student name (value).

Develop a MapReduce program to search for a specific keyword in a file.

Input Data:

```
1001,John,45

1002,Jack,39

1003,Alex,44

1004,Smith,38 1005,Bob,33
```

WordSearcher.java

```
import java.io.IOException;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Job;

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 WordSearcher

{ public static void main(String[] args) throws IOException,

                                                InterruptedException, ClassNotFoundException

  { Configuration conf = new Configuration(); Job

job = new Job(conf);

job.setJarByClass(WordSearcher.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(Text.class);

job.setMapperClass(WordSearchMapper.class);       job.setReducerClass(WordSearchReducer.class);
job.setInputFormatClass(TextInputFormat.class); job.setOutputFormatClass(TextOutputFormat.class);
job.setNumReduceTasks(1);
job.getConfiguration().set("keyword", "Jack");
FileInputFormat.setInputPaths(job, new Path("/mapreduce/student.csv"));
FileOutputFormat.setOutputPath(job, new Path("/mapreduce/output/search"));
System.exit(job.waitForCompletion(true)? 0: 1);
}
}
```

WordSearchMapper.java

```
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.InputSplit;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.lib.input.FileSplit;
public class WordSearchMapper extends Mapper<LongWritable, Text, Text, Text>
{ static String keyword;
static int pos = 0;
protected void setup(Context context) throws IOException, InterruptedException
{ Configuration configuration = context.getConfiguration(); keyword =
configuration.get("keyword");
}
protected void map(LongWritable key, Text value, Context context) throws IOException,
InterruptedException
{  InputSplit i = context.getInputSplit();// Get the input split for this map. FileSplit f =
(FileSplit) i;
String fileName = f.getPath().getName(); Integer
wordPos; pos++;
if (value.toString().contains(keyword))
   { wordPos = value.find(keyword);
context.write(value, new Text(fileName + ","+ new IntWritable(pos). toString() +", "+
wordPos.toString()));
 }
 }
 }
```

WordSearchReducer.java import

```
java.io.IOException; import
org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;
public class WordSearchReducer extends Reducer<Text, Text, Text, Text>
{ protected void reduce(Text key, Text value, Context context) throws IOException,
InterruptedException { context.write(key, value);
}
}
```

OUTPUT

```
1002,Jack,39                student.csv, 2, 5
```

Develop a MapReduce program to sort data by student name (value).

Input Data:

```
1001,John,45
1002,Jack,39
1003,Alex,44
1004,Smith,38 1005,Bob,33
```

```
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
public class SortStudNames
{ public static class SortMapper extends Mapper<LongWritable, Text, Text, Text>
   { protected void map(LongWritable key, Text value, Context context) throws
                                                IOException, InterruptedException
```

```
        { String[] token = value.toString().split(",");
        context.write(new Text(token[1]), new Text(token[0]+" - "+token[1]));
        }
} // Here, value is sorted...
        public static class SortReducer extends Reducer<Text, Text, NullWritable, Text>
        {  public void reduce(Text key, Iterable<Text> values, Context context) throws
                                                 IOException, InterruptedException
{ for (Text details: values)
{ context.write(NullWritable.get(), details);
}
}
}
public static void main(String[] args) throws IOException, InterruptedException,
ClassNotFoundException
{ Configuration conf = new Configuration(); Job
job = new Job(conf);
job.setJarByClass(SortEmpNames.class);
job.setMapperClass(SortMapper.class);
job.setReducerClass(SortReducer.class);
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(Text.class);
FileInputFormat.setInputPaths(job, new Path("/mapreduce/student.csv"));
FileOutputFormat.setOutputPath(job, new Path("/mapreduce/output/sorted/"));
System.exit(job.waitForCompletion(true)? 0: 1);
}
}
```

OUTPUT

```
1003,Alex,44
1005,Bob,33
1002,Jack,39
1001,John,45
1004,Smith,38
```

Week-5	Cassandra
	- Implement keyspace operations to group column families together for the given application data. - Implement CRUD operations on the given dataset using Cassandra.

To create a keyspace by the name “Students”. CREATE

KEYSPACE Students WITH REPLICATION = {

'class':'SimpleStrategy',

'replication_factor':1

};

To describe all the existing keyspaces. DESCRIBE

KEYSPACES;

To create a column family or table by the name “student_info”.

```
CREATE TABLE Student_Info (
        RollNo int PRIMARY KEY,
        StudName text,
        DateofJoining timestamp,
        LastExamPercent double );
```

CRUD (CREATE, READ, UPDATE, AND DELETE) OPERATIONS

```
BEGIN BATCH
INSERT INTO student_info (RollNo,StudName,DateofJoining,LastExamPercent)
VALUES (1,'Michael Storm','2012-03-29', 69.6)
INSERT INTO student_info (RollNo,StudName,DateofJoining,LastExamPercent)
VALUES (2,'Stephen Fox','2013-02-27', 72.5)
INSERT INTO student_info (RollNo,StudName,DateofJoining,LastExamPercent)
VALUES (3,'David Flemming','2014-04-12', 81.7)
INSERT INTO student_info (RollNo,StudName,DateofJoining,LastExamPercent)
VALUES (4,'Ian String','2012-05-11', 73.4)
APPLY BATCH;
```

To view the data from the table “student_info”. SELECT

* FROM student_info;

Week-6

Cassandra

- Design a table/column family and perform various collection types Set, List and Map using Cassandra.
- Design a table/column family and perform Alter table commands using Cassandra.

Collections (SET and LIST)

Objective: To create a table “users” with an “emails” column.The type of this column “emails” is “set”.

```
CREATE TABLE users (
        user_id text PRIMARY KEY,
        first_name text,
        last_name text,
        emails set<text> );
```

Objective: To insert values into the “emails” column of the “users” table.

Note: Set values must be unique.

```
INSERT INTO users (user_id, first_name, last_name, emails)
                VALUES('AB', 'Albert', 'Baggins', {'a@baggins.com', 'baggins@gmail.com'});
```

Using Map: Key, Value Pair

Objective: To alter the “users” table to add a map column “todo”. Act: ALTER TABLE users ADD todo map<timestamp, text>;

Objective: To alter the “users” table to add a column, “top_places” of type list.

```
ALTER TABLE users ADD top_places list<text>;
```

Week-7

MongoDB

- Implement a program with basic commands on databases and collections using MongoDB.
- Implement CRUD operations on the given dataset using MongoDB.

MongoDB use DATABASE_NAME is used to create database. The command will create a new database if it doesn't exist, otherwise it will return the existing database.

```
use DATABASE_NAME
>use mydb
switched to db mydb
>db
mydb

>show dbs
local 0.78125GB
test 0.23012GB

>db.movie.insert({"name":"s point"})
>show dbs
local 0.78125GB
mydb 0.23012GB test
0.23012GB
db.dropDatabase()

db.createCollection(name, options)
>use test
switched to db test
>db.createCollection("mycollection")
{ "ok" : 1 }
>
>show collections
mycollection
system.indexes
```

```
> db.createCollection("mycol", { capped : true, autoIndexID : true, size : 6142800, max : 10000 } ){ "ok" : 0,
"errmsg" : "BSON field 'create.autoIndexID' is an unknown field.",
"code" : 40415,
"codeName" : "Location40415" }
> db.COLLECTION_NAME.drop()
```

Week-8	MongoDB - Perform Count, Limit, Sort, and Skip operations on the given collections using MongoDB

Objective: To find the number of documents in the Students collection. Act:

```
db.Students.count()
```

Objective: To find the number of documents in the Students collection wherein the Grade is VII.

```
db.Students.count({Grade:"VII"});
```

Outcome:

Objective: To retrieve the first 3 documents from the Students collection wherein the Grade is VII.

```
db.Students.find({Grade:"VII"}).limit(3).pretty();
```

Objective: To sort the documents from the Students collection in the ascending order of StudName.

```
db.Students.find().sort({StudName:1}).pretty();
```

Week-9

Pig Latin commands

- **Implement Relational operators –Loading and Storing, and Diagnostic operators -Dump, Describe, Illustrate & Explain on the given database in Hadoop Pig framework using Cloudera.**
- **Develop a Pig Latin program to implement Filtering, Sorting operations on the given database.**

For the given Student dataset and Employee dataset, perform Relational operations like Loading, Storing, Diagnostic Operations (Dump, Describe, Illustrate & Explain) in Hadoop Pig framework using Cloudera

Student ID	First Name	Age	City	CGPA
001	Jagruthi	21	Hyderabad	9.1
002	Praneeth	22	Chennai	8.6
003	Sujith	22	Mumbai	7.8
004	Sreeja	21	Bengaluru	9.2
005	Mahesh	24	Hyderabad	8.8
006	Rohit	22	Chennai	7.8
007	Sindhu	23	Mumbai	8.3

Employee ID	Name	Age	City
001	Angelina	22	LosAngeles
002	Jackie	23	Beijing
003	Deepika	22	Mumbai
004	Pawan	24	Hyderabad
005	Rajani	21	Chennai
006	Amitabh	22	Mumbai

Step-1: **Create a Directory** in HDFS with the name **pigdir** in the required path using **mkdir**:

```
$ hdfs dfs -mkdir /bdalab/pigdir
```

Step-2: The input file of Pig contains each tuple/record in individual lines with the entities separated by a delimiter (“,”).

In the local file system, **create an input file student_data.txt** containing data as shown below.	In the local file system, **create an input file employee_data.txt** containing data as shown below.
001,Jagruthi,21,Hyderabad,9.1	001,Angelina,22,LosAngeles
002,Praneeth,22,Chennai,8.6	002,Jackie,23,Beijing
003,Sujith,22,Mumbai,7.8	003,Deepika,22,Mumbai
004,Sreeja,21,Bengaluru,9.2	004,Pawan,24,Hyderabad
005,Mahesh,24,Hyderabad,8.8	005,Rajani,21,Chennai
006,Rohit,22,Chennai,7.8	006,Amitabh,22,Mumbai
007,Sindhu,23,Mumbai,8.3	

Step-3: **Move the file** from the local file system to HDFS using **put (Or) copyFromLocal command and verify using -cat command**

```
$ hdfs dfs -put  /home/cloudera/pigdir/student_data    /bdalab/pigdir/
$ hdfs dfs -cat  /bdalab/pigdir/student_data
$ hdfs dfs -put /home/cloudera/pigdir/employee_data   /bdalab/pigdir/
$ hdfs dfs -cat /bdalab/pigdir/employee_data
```

Step-4: **Apply Relational Operator – LOAD to load the data** from the file student_data.txt into Pig by executing the following Pig Latin statement in the **Grunt shell**. Relational Operators are **NOT case sensitive.**

```
$ pig          => will direct to       grunt> shell
grunt> student = LOAD '/bdalab/pigdir/student_data.txt'        USING PigStorage(',')
as ( id:int, name:chararray, age:int, city:chararray, cgpa:double );
grunt>    employee    =    LOAD    '/bdalab/pigdir/employee_data.txt'    USING
PigStorage(',') as ( id:int, name:chararray, age:int, city:chararray);
```

Step-5: **Apply Relational Operator – STORE** to **Store the relation** in the HDFS directory **"/pig_output/" as shown below.**

```
grunt> STORE student INTO '/bdalab/pigdir/pig_output/ ' USING PigStorage (',');
grunt> STORE employee INTO ' /bdalab/pigdir/pig_output/ ' USING PigStorage
(',');
```

Step-6: **Verify the stored data** as shown below

```
$ hdfs dfs -ls /bdalab/pigdir/pig_output/
```

```
$ hdfs dfs -cat /bdalab/pigdir/pig_output/part-m-00000
```

Step-7: **Apply Relational Operator – Diagnostic Operator – DUMP to Print the contents of the relation**.

```
grunt>   Dump   student

grunt> Dump employee
```

Step-8: **Apply Relational Operator – Diagnostic Operator – DESCRIBE to View the schema of a relation**.

```
grunt>   Describe   student

grunt> Describe employee
```

Step-9: **Apply Relational Operator – Diagnostic Operator – EXPLAIN to Display the logical, physical, and MapReduce execution plans** of a relation using **Explain** operator

```
grunt>   Explain   student

grunt> Explain employee
```

Step-10: Apply Relational Operator – Diagnostic Operator – ILLUSTRATE to give the step-by-step execution of a sequence of statements

```
grunt>   Illustrate   student

grunt> Illustrate employee
```

Week-10	**Pig Latin commands** - **Implement Grouping, Joining, Combining and Splitting operations on the given database using Pig Latin statements.** - **Perform Eval Functions on the given dataset.** - **Develop a WordCount program using Pig Latin statements.**

The GROUP **operator is used to group the data in one or more relations. It collects the data having the same key.**

```
grunt> Group_data = GROUP Relation_name BY Key;
```

Step-1: **Group the records/tuples** in the relation by age using **GROUP** command and verify.

```
grunt> group_std = GROUP student BY age;
grunt> Dump group_std;
grunt> group_emp = GROUP employee BY city; grunt>
Dump group_emp;
```

Step-2: View **Schema of the table after grouping** the data using the describe command as shown below.

```
grunt> Describe group_std;
group_std: {group: int,student: {(id:int, name:chararray, age:int, city:chararray,
cgpa:float)}}
grunt> Describe group_emp;
group_emp: {group: int,employee: {(id: int,name: chararray,age:int,city:
chararray)}}
```

Step-3: **Group by multiple columns** of the relation by age and city and verify the content.

```
grunt> groupmultiple_std = GROUP student BY (age, city); grunt>
Dump groupmultiple_std
grunt> groupmultiple_emp = GROUP employee BY (age, city);
grunt> Dump groupmultiple_emp
```

Step-4: Group by All columns **of the relation and verify the content. grunt>**

```
groupall_std = GROUP student All;
grunt> Dump groupall_std
grunt> groupall_emp = GROUP employee All;
grunt> Dump groupall_emp
```

Step-5: **Combinedly Group the records/tuples** of the relations student_data and employee_data with the key age and then verify the result.

```
grunt> cogroup_stdemp = COGROUP student_data by age, employee_data by age;

grunt> Dump cogroup_stdemp
```

The JOIN **operator is used to combine records from two or more relations. While performing a join operation, we declare one (or a group of) tuple(s) from each relation, as keys. When these keys match, the two tuples are matched, else the records are dropped. Joins can be of the following types −**

- SELF-Join
- INNER-Join
- OUTER-Join − LEFT Join, RIGHT Join, and FULL Join

Step-6: SELF-JOIN**, we will load the same data multiple times, under different aliases (names).**

```
grunt> std1 = LOAD ' /bdalab/pigdir/student_data ' USING PigStorage(',') as (id:int, name:chararray, age:int, city:chararray, cgpa:float);

grunt> std2 = LOAD ' /bdalab/pigdir/student_data ' USING PigStorage(',') as (id:int, name:chararray, age:int, city:chararray, cgpa:float );

grunt> selfjoin_std_data = JOIN students1 BY id, students2 BY id;

grunt> dump selfjoin_std_data;

(1,Jagruthi,21,Hyderabad,9.1,1,Jagruthi,21,Hyderabad,9.1)
(2,Praneeth,22,Chennai,8.6,2,Praneeth,22,Chennai,8.6)
(3,Sujith,22,Mumbai,7.8,3,Sujith,22,Mumbai,7.8)
(4,Sreeja,21,Bengaluru,9.2,4,Sreeja,21,Bengaluru,9.2)
(5,Mahesh,24,Hyderabad,8.8,5,Mahesh,24,Hyderabad,8.8)
(6,Rohit,22,Chennai,7.8,6,Rohit,22,Chennai,7.8)
(7,Sindhu,23,Mumbai,8.3,7,Sindhu,23,Mumbai,8.3)
```

Step-7: INNER JOIN - EQUI JOIN **creates a new relation by combining column values of two relations based upon the join-predicate. It returns rows when there is a match in both tables.**

```
grunt> innerjoin_data_att = JOIN std_data BY id, std_att BY id;

grunt> dump innerjoin_data_att;

(1,Jagruthi,21,Hyderabad,9.1,1,Jagruthi,joined,9:10:10)
(4,Sreeja,21,Bengaluru,9.2,4,Sreeja,joined,9:10:24)
(6,Rohit,22,Chennai,7.8,6,Rohit,joined,9:11:15)
(7,Sindhu,23,Mumbai,8.3,7,Sindhu,joined,9:12:25)
```

OUTER JOIN **returns all the rows from at least one of the relations. An outer join operation is carried out in three ways –**

- LEFT OUTER JOIN
- RIGHT OUTER JOIN
- FULL OUTER JOIN

Step-8: LEFT OUTER JOIN **operation returns all rows from the left table, even if there are no matches in the right relation.**

Note: Student_data is LEFT

```
grunt> outerleft_data_att = JOIN std_data BY id LEFT, std_att BY id; grunt>

DUMP outerleft_data_att

(1,Jagruthi,21,Hyderabad,9.1,1,Jagruthi,joined,9:10:10) (2,Praneeth,22,Chennai,8.6,,,,)
(3,Sujith,22,Mumbai,7.8,,,,)
(4,Sreeja,21,Bengaluru,9.2,4,Sreeja,joined,9:10:24)
(5,Mahesh,24,Hyderabad,8.8,,,,)
(6,Rohit,22,Chennai,7.8,6,Rohit,joined,9:11:15)
(7,Sindhu,23,Mumbai,8.3,7,Sindhu,joined,9:12:25)
```

Note: Student_att is LEFT

```
grunt> outerleft_att_data = JOIN  std_att BY id LEFT, std_data BY id; grunt>

DUMP outerleft_att_data;

(1,Jagruthi,joined,9:10:10,1,Jagruthi,21,Hyderabad,9.1)
(4,Sreeja,joined,9:10:24,4,Sreeja,21,Bengaluru,9.2)
(6,Rohit,joined,9:11:15,6,Rohit,22,Chennai,7.8)
(7,Sindhu,joined,9:12:25,7,Sindhu,23,Mumbai,8.3)
(8,Sai,joined,9.14:18,,,,,)
(9,Meghana,joined,9.15:25,,,,,)
```

Step-9: RIGHT OUTER JOIN **operation returns all rows from the right table, even if there are no matches in the left table.**

```
grunt> outerright_data_att = JOIN std_data BY id RIGHT, std_att BY id;

grunt> DUMP outerright_data_att;

(1,Jagruthi,21,Hyderabad,9.1,1,Jagruthi,joined,9:10:10)
(4,Sreeja,21,Bengaluru,9.2,4,Sreeja,joined,9:10:24)
(6,Rohit,22,Chennai,7.8,6,Rohit,joined,9:11:15)
(7,Sindhu,23,Mumbai,8.3,7,Sindhu,joined,9:12:25)
```

```
(,,,,,8,Sai,joined,9.14:18)
(,,,,,9,Meghana,joined,9.15:25)
```

Step-10: FULL OUTER JOIN **operation returns rows when there is a match in one of the relations.**

```
grunt> outerfull_data_att = JOIN std_data BY id FULL, std_att BY id; grunt>

DUMP outerfull_data_att;

(1,Jagruthi,21,Hyderabad,9.1,1,Jagruthi,joined,9:10:10) (2,Praneeth,22,Chennai,8.6,,,,)
(3,Sujith,22,Mumbai,7.8,,,,)
(4,Sreeja,21,Bengaluru,9.2,4,Sreeja,joined,9:10:24)
(5,Mahesh,24,Hyderabad,8.8,,,,)
(6,Rohit,22,Chennai,7.8,6,Rohit,joined,9:11:15)
(7,Sindhu,23,Mumbai,8.3,7,Sindhu,joined,9:12:25)
(,,,,,8,Sai,joined,9.14:18)
(,,,,,9,Meghana,joined,9.15:25)
```

Step-11: FILTER **operator is used to select the required tuples from a relation based on a condition**

```
grunt> filter_std = FILTER std_data BY city == 'Hyderabad'; grunt>

DUMP filter_std;

(1,Jagruthi,21,Hyderabad,9.1)
(5,Mahesh,24,Hyderabad,8.8)
```

Step-12: SPLIT **operator is used to split a relation into two or more relations**

```
grunt> SPLIT std_data INTO split_std1 IF age<23, split_std2 IF (age>22 AND
age<25);

grunt> DUMP split_std1;

(1,Jagruthi,21,Hyderabad,9.1)
(2,Praneeth,22,Chennai,8.6)
(3,Sujith,22,Mumbai,7.8)
(4,Sreeja,21,Bengaluru,9.2)
(6,Rohit,22,Chennai,7.8)

grunt> DUMP split_std2;

(5,Mahesh,24,Hyderabad,8.8)
(7,Sindhu,23,Mumbai,8.3)
```

The input file of Pig contains each tuple/record in individual lines with the entities separated by a delimiter (“,”).

Step-1: **Create a Directory** in HDFS with the name **pigdir** in the required path using mkdir**:**

```
$ hdfs dfs -mkdir /bdalab/pigdir
```

Step-2: In the local file system, **create an input file wordcount** containing data as shown below.

```
Deer,Bear,River
Car,Car,River
River,Car,River
Deer,River,Bear
```

Step-3: **Move the file** from the local file system to HDFS using **put (Or) copyFromLocal command and verify using -cat command**

```
$ hdfs dfs -put /home/cloudera/pigdir/wordcount_data   /bdalab/pigdir/
```

```
$ hdfs dfs -cat / bdalab/pigdir/ wordcount_data
```

Step-4: Open Pig in Grunt shell **and execute the following Pig Latin statement.**

$ pig => will direct to **grunt>**

Convert Each line to each tuple.

Apply Relational Operator – LOAD **to load the data into Relation** lines **from the file** wordcount_data.

```
grunt> lines = LOAD '/bdalab/pigdir/wordcount_data'  AS  (line:chararray);
```

```
grunt> DUMP lines;
```

```
(Deer,Bear,River)
(Car,Car,River)
(River,Car,River)
(Deer,River,Bear)
```

Step-5: **Convert Each line tuple to each word tuple TOKENIZE**

splits the line into a field for each word.

FLATTEN **will take the collection of records returned by TOKENIZE and produce a separate record for each one, calling the single field in the record word.**

FOREACH **operator is used to generate specified data transformations based on the column data.**

```
grunt> words = FOREACH lines GENERATE FLATTEN(TOKENIZE(line,’%’)) as word;
```

```
grunt> dump words;
```

```
(Deer)
(Bear)
(River)
(Car)
(Car)
(River)
(River)
(Car)
(River)
(Deer)
(River)
(Bear)
```

Step-6: **Group all similar words into each tuple** grunt>

```
groupword = GROUP words by word; grunt>

dump groupword;

(Car,{(Car),(Car),(Car)})
(Bear,{(Bear),(Bear)})
(Deer,{(Deer),(Deer)})
(River,{(River),(River),(River),(River),(River)})
```

Step-6: **Count each grouped word and display**

```
grunt> wordcount = FOREACH groupword GENERATE group, COUNT(words); grunt>

dump wordcount;

(Car,3)

(Bear,2)

(Deer,2)

(River,5)
```

Week-11	Hive commands • Implement Data Definition Language (DDL) Commands for databases in Hadoop Hive framework using Cloudera. • Implement Data Definition Language (DDL) Commands for tables in Hive.

- Open Virtual box and then start **cloudera quickstart Terminal** and type “hive” to launch hive shell

DDL Commands for Databases

1) **CREATE** database Statement is used to create a database in Hive. A database in Hive is a namespace or a collection or catalog of tables.

Syntax: **CREATE DATABASE|SCHEMA [IF NOT EXISTS] database_name**

[COMMENT database_comment]

[LOCATION hdfs_path]

[WITH DBPROPERTIES (property_name=property_value, ...)];

[ ] are optional clauses. We can use SCHEMA in place of DATABASE in this command. The following query is executed to create a database named employee. If everything went good, you will see a ‘OK’ message, else you will see relevant error message.

Simple creation

```
hive> CREATE DATABASE facultycse; OK

Time taken: 0.033 seconds

hive> CREATE DATABASE facultyece;
```

Full creation

```
hive> CREATE DATABASE IF NOT EXISTS employee COMMENT ‘this is employee database’
      LOCATION ‘/user/hive/warehouse/hivedir/’ WITH DBPROPERTIES (‘creator’=‘Bhanu’,
      ‘date’=‘2020-12-07’);
```

2) **SHOW** databases statement lists all the databases present in the metastore.

Syn: **SHOW (DATABASES/SCHEMAS) [LIKE ‘wildcards'];**

- Wildcards in the regular expression can only be '*' for any character(s) or '|' for a choice. Examples are 'employees', 'emp*', 'emp*|*ees', all of which will match the database named 'employees’:

hive> SHOW DATABASES; default employee facultycse facultyece	hive> SHOW DATABASES LIKE ‘*ee’; employee
	hive> SHOW DATABASES LIKE ‘fac*’; facultycse facultyece

3) **DESCRIBE** database statement in Hive shows the name of Database in Hive, its comment (if set), its location, its owner name, owner type and its properties.

Syn: DESCRIBE DATABASE/SCHEMA [EXTENDED] db_name;

- EXTENDED can be used to get the database properties.

```
hive>DESCRIBE DATABASE facultycse;

facultycse  hdfs://quickstart.cloudera:8020/user/hive/warehouse/faculty.db  cloudera
USER

hive>DESCRIBE DATABASE EXTENDED employee;
employee this is employee database
hdfs://quickstart.cloudera:8020/user/hive/warehouse/ cloudera USER {date=2020- 12-07,
creator=Bhanu};
```

4) **USE** database statement in Hive is used to select the specific database for a session on which all subsequent HiveQL statements would be executed.

Syn: **USE db_name;**

```
hive> USE employee;

OK
```

5) **DROP** database statement in Hive is used to Drop (delete) the database. The default behavior is RESTRICT which means that the database is dropped only when it is empty. To drop the database with tables, we can use CASCADE.

Syn: DROP (DATABASE|SCHEMA) [IF EXISTS] db_name [RESTRICT|CASCADE];

```
hive> DROP DATABASE facultyece; OK

 hive> DROP DATABASE IF EXISTS facultycse CASCADE;
```

OK

6) **ALTER** database statement in Hive is used to change the metadata associated with the database in Hive.

Syntax for changing Database Properties:

ALTER (DATABASE|SCHEMA) db_name SET DBPROPERTIES **(property_name=property_value, ...);**

hive> ALTER DATABASE employee SET DBPROPERTIES ('creator'='Bhanu Prasad', 'date'='07-12-2020');

employee this is employee database hdfs://quickstart.cloudera:8020 /user/hive/warehouse/hivedir/ cloudera USER {date= **07-12-2020**, creator=**Bhanu Prasad**};

Syn for changing Database owner:

ALTER (DATABASE|SCHEMA) database_name SET OWNER [USER|ROLE] **user_or_role;**

hive> ALTER DATABASE employee SET OWNER USER client**;**

employee this is employee database hdfs://quickstart.cloudera:8020 /user/hive/warehouse/hivedir/ client USER **{date= 07-12-2020, creator=Bhanu Prasad};**

hive> ALTER DATABASE employee SET OWNER ROLE Admin**;**

employee this is employee database hdfs://quickstart.cloudera:8020 /user/hive/warehouse/hivedir/ Admin ROLE **{date= 07-12-2020, creator=Bhanu Prasad};**

DDL Commands for Tables

1) **CREATE TABLE** statement in Hive is used to create a table with the given name. If a table or view already exists with the same name, then the error is thrown. We can use IF NOT EXISTS to skip the error.

Syn: CREATE TABLE [IF NOT EXISTS] [db_name.] table_name [(col_name data_type [COMMENT col_comment], ... [COMMENT col_comment])]

[COMMENT table_comment]

[ROW FORMAT row_format]

[STORED AS file_format]

[LOCATION hdfs_path];

```
hive> CREATE TABLE IF NOT EXISTS employee.emptable   (emp_id STRING COMMENT ‘This
is Employee ID’, emp_name STRING COMMENT ‘This is Employee Name’, emp_sal FLOAT
COMMENT ‘This is Employee Salary’)
COMMENT ‘This table contains Employees Data’ ROW
FORMAT DELIMITED
FIELDS TERMINATED BY ‘,’
STORED AS TEXTFILE;
```

2) **SHOW** tables statement in Hive lists all the base tables and views in the current database.

Syn: **SHOW TABLES [IN database_name];**

```
hive> SHOW TABLES IN employee; OK
emptable
```

3) **DESCRIBE** table statement in Hive shows the lists of columns for the specified table.

Syn: DESCRIBE [EXTENDED|FORMATTED] [db_name.] table_name[.col_name ([.field_name])];

```
hive> DESCRIBE employee.emptable; emp_id
string This is Employee ID emp_name string
This is Employee Name emp_sal float This is
Employee Salary
hive> DESCRIBE EXTENDED employee.emptable;
hive> DESCRIBE FORMATTED employee.emptable;
```

4) **ALTER** table statement in Hive enables you to change the structure of an existing table, rename the table, add columns to the table, change the table properties, etc.

Syntax for Rename a table:

```
ALTER TABLE table_name RENAME TO new_table_name;
hive> ALTER TABLE employee.emptable RENAME TO employee.facultytable;
```

Syn to Add columns to a table:

ALTER TABLE table_name ADD COLUMNS (column1, column2) ;

hive> ALTER TABLE employee.facultytable ADD COLUMNS (emp_post string COMMENT 'This is employee post', emp_age INT COMMENT 'This is employee age');

Syn to set table properties:

ALTER TABLE table_name SET TBLPROPERTIES **('property_key'='property_new_value');**

hive> ALTER TABLE employee.facultytable SET TBLPROPERTIES ('table for'='faculty data');

5) **DROP** table statement in Hive deletes the data for a particular table and remove all metadata associated with it from Hive metastore.

- If PURGE is not specified, then the data is actually moved to the .Trash/current directory.
- If PURGE is specified, then data is lost completely.

Syn: **DROP TABLE [IF EXISTS] table_name [PURGE];**

hive> DROP TABLE IF EXISTS employee.emptable PURGE; OK

6) **TRUNCATE** table statement in Hive removes all the rows from the table or partition. Syn:

TRUNCATE TABLE table_name;

hive> TRUNCATE TABLE employee.emptable; OK

Week-12	**Hive commands** • Implement Data Manipulation Language (DML) Commands for tables in Hive. • Perform data partitioning to split the given larger dataset into more meaningful chunks.

- Open Virtual box and then start **cloudera quickstart Terminal** and type “hive” to launch hive shell

DML Commands for Tables

1) **LOAD** statement in Hive is used to copy/move data files into the locations corresponding to Hive tables.

Syn: LOAD DATA [LOCAL] INPATH 'filepath' [OVERWRITE] INTO TABLE **tablename [PARTITION (partcol1=val1, partcol2=val2 ...)];**

LOCAL keyword = file path in the local filesystem. LOCAL

not specified = file path in the hdfs

OVERWRITE contents of the target table (or partition) will be deleted and replaced by the files otherwise contents are added to the table

```
hive> LOAD DATA LOCAL INPATH ‘/home/cloudera/HiveDir/emptextdata' INTO TABLE employee.facultytable;

OK
```

emptextdata contents

```
1,bob,25000.00,asstprof,35,male

2,mary,35000.00,assocprof,38,female

3,mike,50000.00,prof,45,male
```

2) **SELECT** statement in Hive is similar to the SELECT statement in SQL used for retrieving data from the database.

Syn: **SELECT * FROM tablename;** //displays all records

```
hive> SELECT * FROM employee.facultytable;
1   bob     25000.00    asstprof                35      male
2   mary    35000.00    assocprof       38      female
3   mike    50000.00    prof            45      male
```

SELECT col1,col2 FROM tablename; //Retrieves only specified columns data

```
hive> SELECT emp_name,emp_salary FROM employee.facultytable;
bob         25000.00
mary        35000.00
mike        50000.00
```

3) **a) INSERT INTO** statement appends the data into existing data in the table or partition. Syn:

INSERT INTO TABLE tablename [PARTITION (partcol1=val1, partcol2=val2

...)] VALUES (col1value,col2value,...)

```
hive> INSERT INTO TABLE employee.facultytable VALUES (4, ‘jessy’, 45000.00,
‘assocprof’, 40, ‘female’);
hive> SELECT * FROM employee.facultytable;
4   jessy   45000.00    assocprof       40      female
1   bob     25000.00    asstprof                35      male
2   mary   35000.00     assocprof       38      female
3   mike    50000.00    prof            45      male
```

b) INSERT OVERWRITE table overwrites the existing data in the table or partition.

Syn: INSERT OVERWRITE TABLE tablename1 [PARTITION (partcol1=val1, ..) [IF NOT EXISTS]] select_statement FROM from_statement;

4) **DELETE** statement in Hive deletes the table data. If the WHERE clause is specified, then it deletes the rows that satisfy the condition in where clause.

Syn: DELETE FROM tablename [WHERE expression];

```
hive> DELETE FROM employee.facultytable WHERE emp_age=38; hive>
SELECT * FROM employee.facultytable;
4   jessy   45000.00    assocprof       40      female
1   bob     25000.00    asstprof                35      male
3   mike    50000.00    prof            45      male
```

5) **UPDATE** statement in Hive deletes the table data. If the WHERE clause is specified, then it updates the column of the rows that satisfy the condition in WHERE clause. Partitioning and Bucketing columns cannot be updated.

Syn: **UPDATE tablename SET column = value [, column = value ...] [WHERE expression];**

hive> UPDATE employee.facultytable SET emp_name = 'mike tyson' WHERE emp_age=45;

hive> SELECT * FROM employee.facultytable;

```
4   jessy   45000.00     assocprof    40     female
1   bob     25000.00     asstprof            35     male
3   mike tyson     50000.00     prof         45     male
```

6) **EXPORT** statement exports the table or partition data along with the metadata to the specified output location in the HDFS. Metadata is exported in a _metadata file, and data is exported in a subdirectory 'data.'

Syn: EXPORT TABLE tablename [PARTITION (part_column="value"[, ...])] TO 'export_target_path' [FOR replication('eventid')];

hive> EXPORT TABLE employee.drivertable TO '/user/hive/warehouse';

7) **IMPORT** command imports the data from a specified location to a new table or already existing table.

Syn: IMPORT [[EXTERNAL] TABLE new_or_original_tablename [PARTITION (part_column="value"[, ...])]] FROM 'source_path' [LOCATION 'import_target_path'];

hive> IMPORT TABLE employee.importedtable FROM '/user/hive/warehouse';

CONTENT BEYOND SYLLUBUS

S.No	Name of the Experiment/Prototype/Task/Program
1	Create a data file for below schemas: • **Order**: CustomerId, ItemId, ItemName, OrderDate, DeliveryDate • **Customer**: CustomerId, CustomerName, Address, City, State, Country 1) Create a table for Order and Customer Data. 2) Write a HiveQL to find number of items bought by each customer.
2	Create a data file for below schemas: • **Order**: CustomerId, ItemId, ItemName, OrderDate, DeliveryDate • **Customer**: CustomerId, CustomerName, Address, City, State, Country 1) Load Order and Customer Data. 2) Write a Pig Latin Script to determine number of items bought by each customer.
3	COMPLEX DATA TYPE – BAG 1) Create a file which contains bag dataset as shown below. User ID From To . user1001 user1001@sample.com {(user003@sample.com), (user004@sample.com), (user006@sample.com)} user1002 user1002@sample.com {(user005@sample.com), (user006@sample.com)} user1003 user1003@sample.com {(user001@sample.com),(user005@sample.com)} 2) Write a Pig Latin statement to display the names of all users who have sent emails and also a list of all the people that they have sent the email to. 3) Store the result in a file.
4	Write the insert method to store the following document in MongoDB. Name: “Stephen More” Address: { “City”: “Bangalore”, “Street”: “Electronics City”, “Affiliation”: “XYZ Ltd” } Hobbies: Chess, Lawn Tennis, Base ball

5	To practice import, export, and aggregation in MongoDB. Step 1: Pick any public dataset from the site www.kdnuggets.com. Convert it into CSV format. Make sure that you have at least two numeric columns. Step 2: Use MongoImport to import data from the CSV format file into MongoDB collection, “MongoDBHandsOn” in test database. Step 3: Identify a grouping column. Step 4: Compute the sum of the values in the first numeric column. Step 5: Compute the average of the values in the second numeric column.