Department of Computer Science and Application.

Panjab University, Chandigarh.

(2024-25)

M.Sc. (Hons) in Computer Science (Specialization in Data Science).

ASSIGNMENT

Course Title:	**BIG DATA ANALYTICS**
Course Code:	**MDS-2411**

Submitted by:

Meenu.

Roll no: 07

Submitted to:

Dr. R.K Singla

DCSA

INDEX

Introduction to Big Data

In the era of digital transformation, data has become one of the most valuable resources globally. The exponential growth in the volume and complexity of data has ushered in the age of big data, a phenomenon that transcends mere information management. Big data refers to datasets so vast and intricate that traditional databases and tools are insufficient for handling them. Instead, specialized technologies and methodologies are required to extract meaningful insights, make informed decisions, and drive innovation.

Big data is not just about size; it represents a shift in how we perceive and leverage data. As organizations, governments, and individuals generate data at unprecedented rates, big data serves as the foundation for critical advancements in technology, science, and industry. For example, social media interactions, IoT devices, sensors, and transaction records continuously contribute to the ever-expanding data universe.

Understanding big data is essential because its applications extend to nearly every aspect of modern life. From analyzing consumer behavior to predicting natural disasters, the ability to process large datasets can transform challenges into opportunities. Ultimately, big data drives actionable intelligence, bringing precision, efficiency, and foresight to decision-making processes.

Key Characteristics of Big Data

Big data exhibits unique traits that distinguish it from traditional datasets. These are often referred to as the "5 Vs":

1. **Volume**:
    - Volume refers to the sheer scale of data being produced. In 2025, it is estimated that over 180 zettabytes of data will be generated globally—a staggering figure. This encompasses structured data (such as databases and spreadsheets) and unstructured data (such as videos, social media posts, and emails).
    - Organizations must develop scalable storage and processing solutions, such as distributed systems and cloud computing, to accommodate these vast quantities.
2. **Velocity**:
    - Velocity captures the speed at which data is produced and needs to be processed. Real-time data streams—such as stock market transactions, social media updates, and IoT sensors—demand immediate analysis.
    - Technologies like Apache Kafka and Spark facilitate high-speed data processing, enabling businesses to act on insights almost instantaneously.
3. **Variety**:
    - Variety highlights the diversity of data types, formats, and sources. Data can be structured (traditional databases), semi-structured (JSON or XML files), or unstructured (images, videos, audio files).
    - Processing such heterogeneous data requires advanced tools and algorithms capable of integration and interpretation.

4. **Veracity**:
    - This characteristic focuses on the quality and reliability of data. With big data, inconsistencies, errors, and biases can affect the accuracy of insights.
    - Addressing veracity requires proper data validation and cleansing processes.
5. **Value**:
    - The ultimate goal of analyzing big data is to extract value. Data must translate into actionable insights or benefits for organizations.
    - Examples: Customer preferences for targeted marketing or operational efficiencies in supply chains.

Types of Big Data

Big data can be classified into three primary types based on its structure and origin:

1. **Structured Data**:
    - This type of data is organized in a fixed format, such as rows and columns in a database.
    - It is easy to store, search, and analyze using traditional methods.
    - Examples: Employee records, sales data, and financial transactions.
2. **Semi-Structured Data**:
    - It does not follow a strict format but includes tags or markers to separate elements.
    - Semi-structured data is more flexible than structured data and often requires specialized tools for analysis.
    - Examples: JSON files, XML data, and emails.
3. **Unstructured Data**:
    - This type of data does not have a predefined structure and is the most complex to process and analyze.
    - It constitutes the majority of data generated today.
    - Examples: Images, videos, social media posts, and audio files.
4. **Metadata** (Supplementary Type):
    - Often considered a subset, metadata describes other data, making it easier to categorize, find, and use.

Introduction to Hadoop

Hadoop is an open-source software framework that has revolutionized how organizations handle big data. With the exponential growth of data in recent decades, traditional methods of data processing became inefficient and costly. Enter Hadoop—a solution designed to manage vast datasets, enabling distributed storage and parallel processing across clusters of inexpensive commodity hardware. Its ability to process structured, semi-structured, and unstructured data has made it an essential tool in the big data ecosystem.

History of Hadoop

The story of Hadoop begins with a need to process immense amounts of data. It was developed by Doug Cutting and Mike Cafarella as part of the Nutch search engine project. Inspired by Google's innovations—specifically its 2003 white paper on the Google File System (GFS) and the MapReduce programming model—Hadoop was born. Yahoo! played a significant role in its early adoption and development, making Hadoop a central part of its data operations.

Hadoop's name has an interesting origin too. It was named after a toy elephant owned by Doug Cutting's son, giving it a unique and playful identity in the tech world.

Key Components of Hadoop*:*

1. **HDFS (Hadoop Distributed File System)**: Responsible for storing large datasets in a distributed manner.
2. **MapReduce**: Handles distributed data processing.
3. **YARN (Yet Another Resource Negotiator)**: Manages resources across the Hadoop cluster.
4. **Hadoop Common**: Provides libraries and utilities necessary for the other components to function.

Introduction to HDFS

HDFS is the backbone of the Hadoop ecosystem and serves as its primary storage layer. It is specifically designed to handle the storage of large datasets, overcoming the limitations of traditional file systems. By distributing data across multiple nodes in a cluster, HDFS ensures seamless scalability, fault tolerance, and reliability, making it ideal for big data applications.

Origins and Significance

HDFS was inspired by Google's File System (GFS), which outlined a novel approach to managing data on distributed systems. The creators of Hadoop adopted these principles to develop HDFS, ensuring that it could efficiently manage data across inexpensive commodity hardware. HDFS revolutionized how organizations store and access data, enabling them to harness the power of big data without incurring exorbitant costs.

Key Design Philosophy

At its core, HDFS embraces the principle of *"Write Once, Read Many Times."* This approach is particularly suitable for applications that involve data analysis rather than transactional workloads. Instead of constantly modifying data, users can focus on processing and extracting insights from stable datasets.

HDFS also minimizes the need for data movement by bringing computations closer to the data itself. This strategy significantly reduces latency and enhances processing efficiency, aligning with Hadoop's overarching philosophy.

Why HDFS Stands Out

HDFS addresses critical challenges posed by traditional storage systems:

1. **Handling Large Data Volumes**:
    - Traditional file systems struggle to store large files or datasets. HDFS, however, can seamlessly store terabytes or even petabytes of data, breaking files into smaller blocks and distributing them across nodes.
2. **Fault Tolerance**:
    - HDFS is built to withstand node failures without data loss. The replication mechanism ensures that data is always available, even if a node becomes inoperative.
3. **Cost Efficiency**:
    - Instead of relying on expensive hardware, HDFS operates on low-cost commodity machines, democratizing access to big data storage solutions.
4. **Scalability**:
    - As data needs grow, organizations can add more nodes to their HDFS cluster, ensuring continuous performance without major restructuring.

Future of Hadoop and HDFS

The future of Hadoop and HDFS lies in enhancing efficiency and integrating with emerging technologies like:

1. **Cloud Computing**:
    - Many Hadoop deployments are moving to cloud platforms to improve flexibility and reduce costs.
2. **Artificial Intelligence (AI) and Machine Learning (ML)**:
    - HDFS serves as a foundational storage layer for training and deploying AI/ML models.
3. **Edge Computing**:
    - Combining Hadoop with edge computing will allow real-time data processing closer to the source.

Introduction to Apache Pig

Apache Pig is a high-level platform developed to process and analyze large-scale datasets in a distributed environment. It simplifies the tasks of managing big data by providing a scripting language called **Pig Latin**, which is easy to use and designed to handle complex data flows. Pig is particularly popular among data scientists and developers for its ability to abstract away the complexities of writing extensive MapReduce code.

Created by Yahoo in 2006 and later adopted by the Apache Software Foundation, Pig was named after its capability to "eat and process any kind of data." This makes it highly suitable for the heterogeneous nature of big data, as it can efficiently work with both structured and unstructured data.

Why Use Apache Pig?

1. Simplifies Big Data Processing

Apache Pig eliminates the need to write complex MapReduce programs in Java, which can be time-consuming and require extensive programming expertise. Its scripting language, **Pig Latin**, provides a high-level abstraction for specifying data transformations, making it easier and faster to develop data pipelines.

2. Efficiency in Handling Large Data Sets

Pig is optimized for processing large-scale datasets efficiently. It can handle structured, semi-structured, and unstructured data seamlessly, which is crucial for big data applications where data comes in various forms.

3. Flexibility and Versatility

Pig supports multiple data formats, including plain text files, JSON, XML, and binary files. This versatility makes it a preferred choice for handling the heterogeneous nature of big data.

4. Automatic Optimization

One of Pig's standout features is its ability to optimize scripts automatically. When you write a Pig Latin script, the underlying Pig engine converts it into a series of MapReduce jobs, applying optimization techniques to enhance execution speed and resource utilization.

5. Extensibility with User-Defined Functions (UDFs*)*

Pig enables users to write custom functions, known as **User-Defined Functions (UDFs)**, in Java, Python, or other languages. This extensibility allows users to implement unique data transformations and operations tailored to their specific requirements.

6. Scalability

Pig is highly scalable and performs exceptionally well in distributed environments. It can process massive volumes of data by distributing the workload across multiple nodes in a Hadoop cluster.

How Apache Pig Works

Apache Pig operates as an abstraction layer over MapReduce. It provides a high-level scripting interface called **Pig Latin**, which simplifies the creation of complex data processing tasks. Under the hood, Pig translates these scripts into MapReduce jobs that run on a Hadoop cluster. Here's a detailed breakdown of its functionality:

1. Architecture of Apache Pig

The architecture of Apache Pig comprises the following components:

a. Pig Latin Scripts

- Users write data processing tasks using Pig Latin, a language specifically designed for this purpose. These scripts define data transformations like filtering, grouping, joining, and aggregating.

b. Pig Engine

- The Pig Engine acts as the core of Apache Pig. It parses, compiles, and optimizes the Pig Latin script into a series of MapReduce jobs or equivalent operations.
- The engine consists of several stages:
    - **Parser**: Validates the syntax of the Pig Latin script and generates a logical plan.
    - **Optimizer**: Optimizes the logical plan for efficient execution.
    - **Compiler**: Converts the optimized plan into a physical plan (MapReduce jobs).
    - **Executor**: Executes the physical plan on a Hadoop cluster.

c. Execution Modes

Pig can operate in two distinct modes:

1. **Local Mode**:
    - Processes data on a single machine, without requiring Hadoop.
    - Ideal for testing and smaller datasets.
2. **MapReduce Mode**:
    - Runs on a Hadoop cluster, using HDFS for storage and MapReduce for distributed processing.

Introduction to Apache Hive

Apache Hive is a data warehousing and analytics platform designed for querying and managing large datasets stored in Hadoop's HDFS. With the explosive growth of big data, there arose a need to simplify data processing for analysts and developers who were accustomed to SQL-like systems. Hive bridges this gap by offering a high-level abstraction through Hive Query Language (HiveQL), making big data analytics accessible to users without expertise in low-level programming models like MapReduce.

Hive originated at Facebook in 2007 and quickly gained traction due to its ability to process petabyte-scale datasets efficiently. By 2010, it was donated to the Apache Software Foundation and became an open-source project. Hive is now widely adopted across industries for ETL (Extract, Transform, Load) processes, data summarization, and business intelligence tasks. It stands out as a robust tool for managing structured and semi-structured data, enabling organizations to derive insights and make data-driven decisions.

Architecture of Apache Hive

The architecture of Apache Hive is designed to work seamlessly with Hadoop and its components. It consists of the following major elements:

1. Storage Layer*:*

Hive relies on **HDFS (Hadoop Distributed File System)** for data storage. HDFS provides the scalability and reliability required to handle massive datasets.

2. Metastore:

The **Metastore** is a critical component that stores metadata about Hive tables, such as their schema, location, data types, and partitions. It uses a relational database (such as MySQL or Derby) to maintain this information. The Metastore enables Hive to efficiently query and manage structured data.

3. Driver:

The **Hive Driver** manages the lifecycle of HiveQL queries. It is responsible for:

- Parsing the queries.
- Generating execution plans.
- Coordinating execution across the Hadoop cluster.

4. Compiler:

Hive's **Compiler** converts HiveQL queries into directed acyclic graphs (DAGs) that represent MapReduce or Tez jobs. These jobs are then executed on the Hadoop cluster.

5. Execution Engine:

Hive uses the **Execution Engine** to run compiled queries as MapReduce, Tez, or Spark jobs. Tez and Spark provide better performance compared to traditional MapReduce, especially for iterative processing tasks.

6. HiveQL Interface:

Hive provides an interface for users to submit HiveQL queries and interact with the system. The interface supports SQL-like commands, making it intuitive for database professionals.

7. User Interface:

Hive offers multiple ways to interact:

- **Command Line Interface (CLI)** for executing queries directly.
- **JDBC/ODBC Drivers** for integrating with other applications.
- **Web Interface** for remote access and management.

Advantages of Apache Hive

1. **User-Friendly**:
    - SQL-like interface makes Hive accessible to users from traditional database backgrounds.
2. **Scalable**:
    - Capable of processing petabyte-scale datasets across distributed Hadoop clusters.
3. **Flexible Data Formats**:
    - Supports multiple storage formats like ORC, Parquet, and text files.
4. **Integration**:
    - Works seamlessly with other Hadoop tools, such as Pig, Spark, and HBase.
5. **Data Partitioning**:
    - Hive supports partitioning and bucketing, which improves query performance.

Apache Hive plays a pivotal role in the Hadoop ecosystem, enabling efficient data querying and analytics for massive datasets. Its architecture ensures scalability and flexibility, while its SQL-like HiveQL interface makes it accessible to a wide range of users. Hive is an invaluable tool for organizations looking to bridge the gap between traditional data warehousing systems and the distributed processing capabilities of Hadoop.

Cloudera Quick Start VM Installation

The primary objective of this assignment is to develop a hands-on understanding of installing and configuring Apache Hadoop or Cloudera QuickStart VM on a personal computer.

• Learn the step-by-step installation process for Apache Hadoop and Cloudera QuickStart VM on a Windows-based system.

• Understand the importance of environment variables and their role in software functionality.

• Familiarize themselves with Hadoop's key components, such as HDFS, YARN, and MapReduce, through direct interaction.

• Explore the configurations required for setting up a Hadoop cluster and gain practical knowledge of handling big data tools.

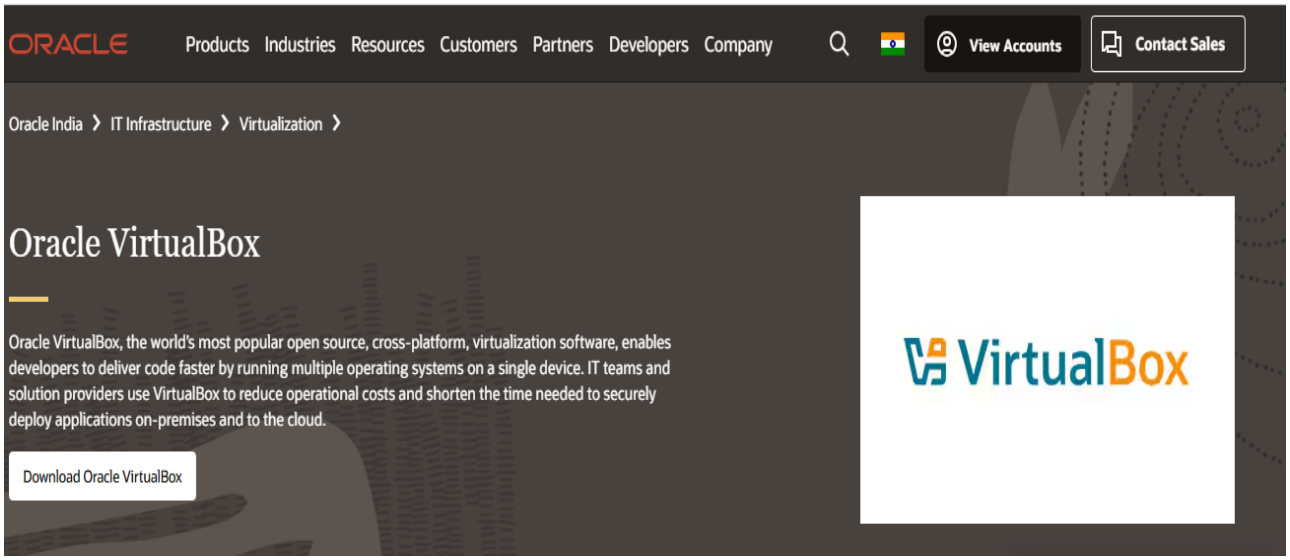
• Verify the successful installation by accessing Hadoop services and executing basic commands.

• Enhance problem-solving skills by troubleshooting common installation errors and challenges.

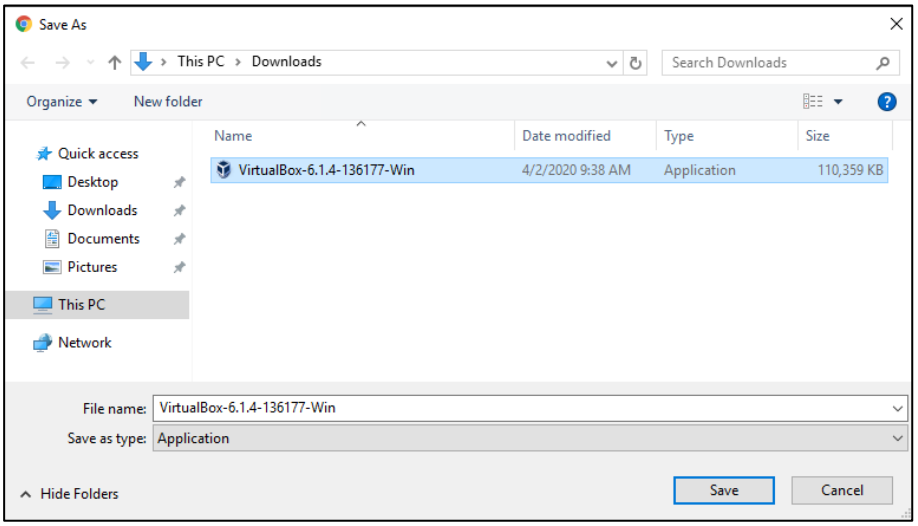
Part1–Oracle Virtual Box Installation

[1] LocatethelatestversionofOracleVirtualBoxforyourcomputer'soperatingsystem from the following site:

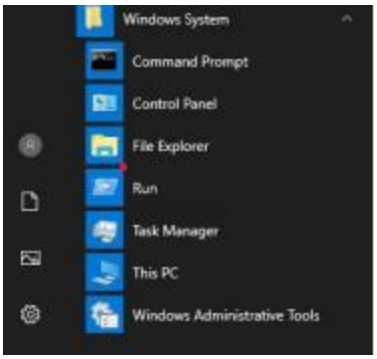
https://www.virtualbox.org/wiki/Downloads



- Click on the applicable link to initiate the download. Depending on how your Windows system is configured, the installation file will either automatically load into the **Downloads** folder or you will be prompted for a location to save the file.

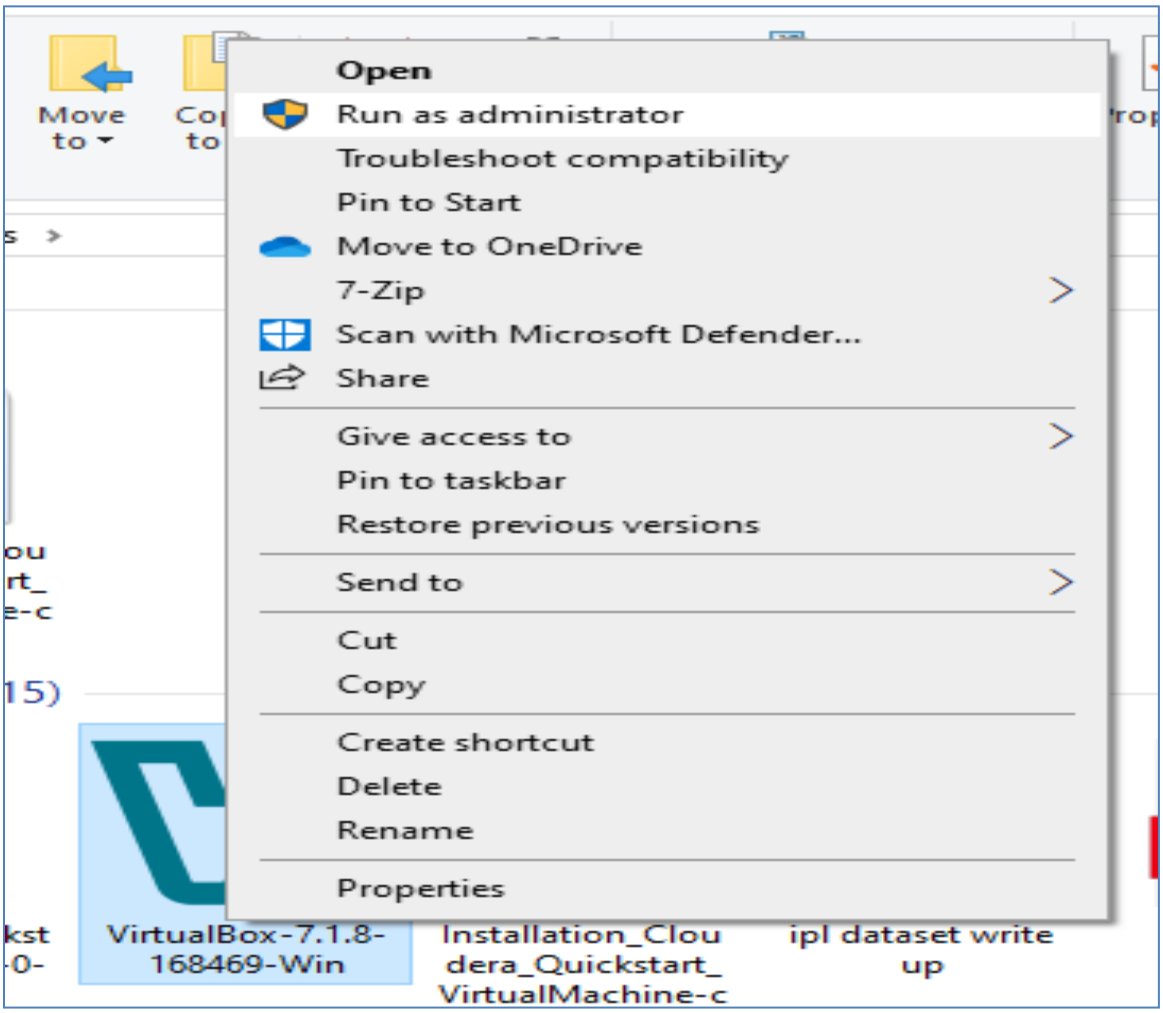


- Please note the Oracle VirtualBox version depicted in this document may not match the version you are downloading since the software is continuously upgraded.



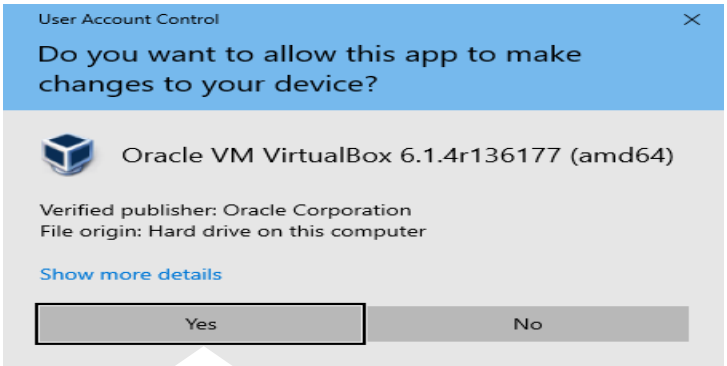
Open the **FileExplorer** application.InWindows10/11,the program is located in **StartMenu-->WindowsSystem-->FileExplorer**

- In **File Explorer**, navigate to the location the Oracle VirtualBox installation file was downloaded to. Right-click on the file with your computer mouse and select the **Run as Administrator** option from the right-click menu.



Step 4

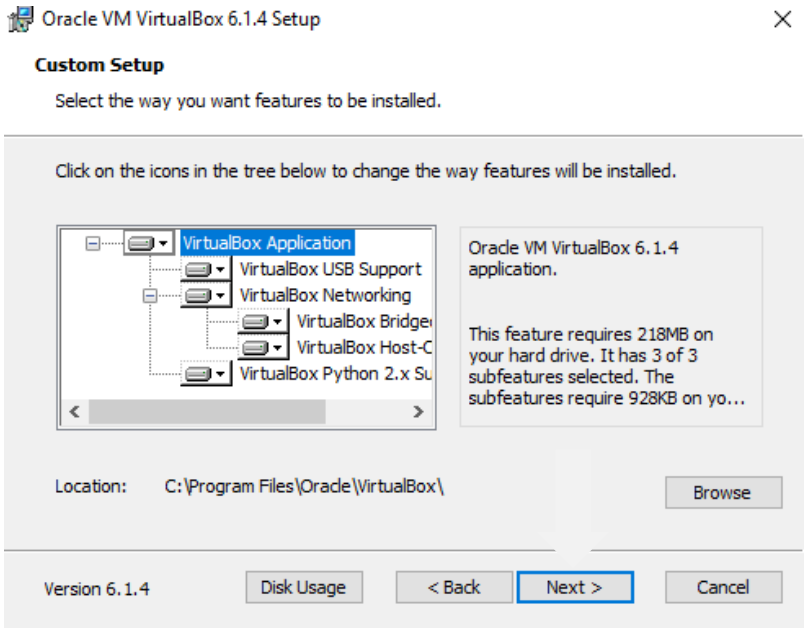
You may see a **User Account Control (UAC)**verification prompt similar to that presented below. Click on the **YES** option to initiate the installation program.

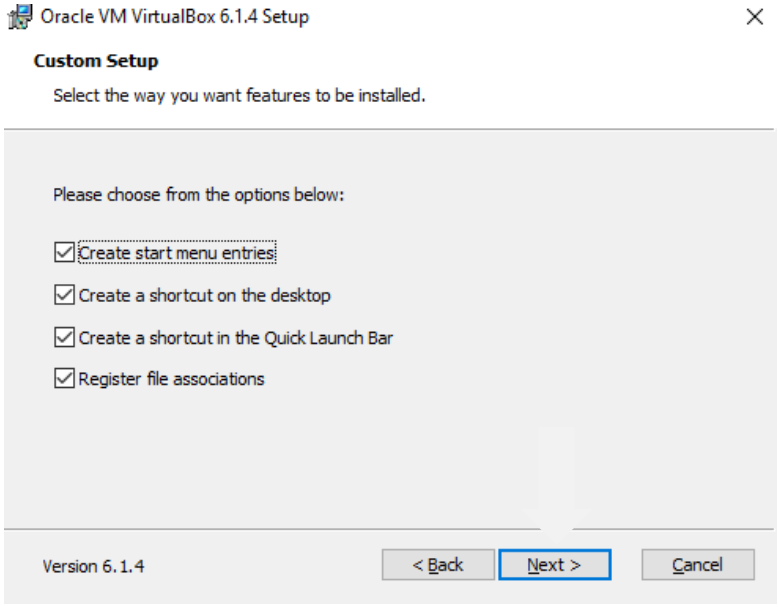


Step -5

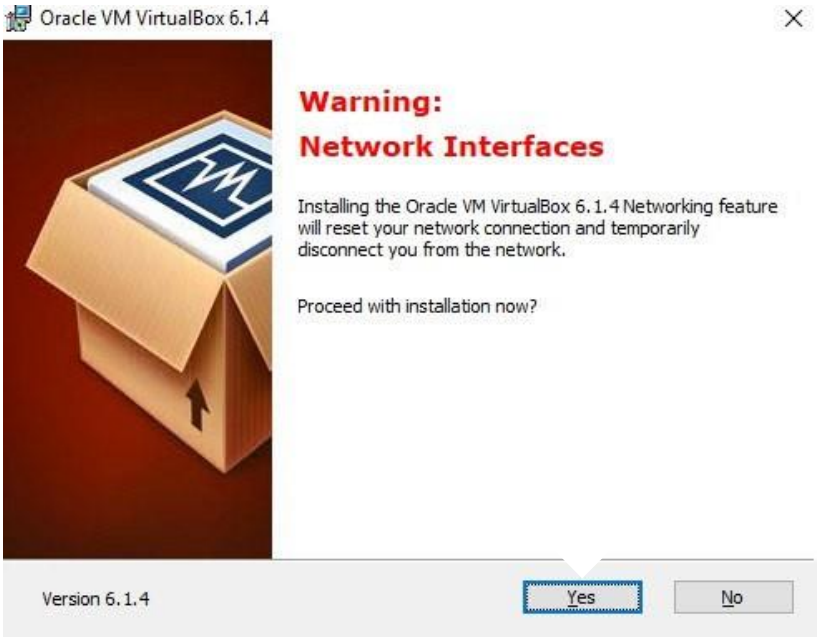
- Click the **NEXT** button when the following window appears:

- The **CustomSetup** window will then appear. Leave the default options as is and click on the **NEXT** button to continue.

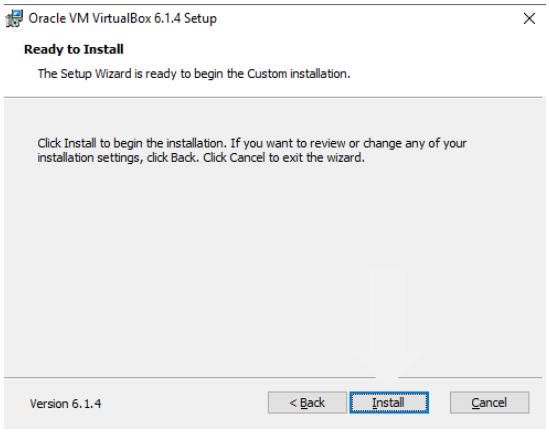




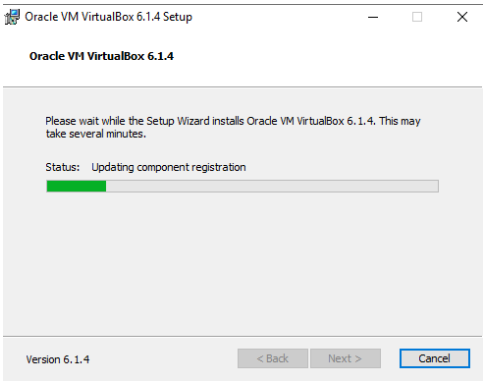
- You'll then be presented with the warning prompt depicted below. Click on the **YES** button to continue.

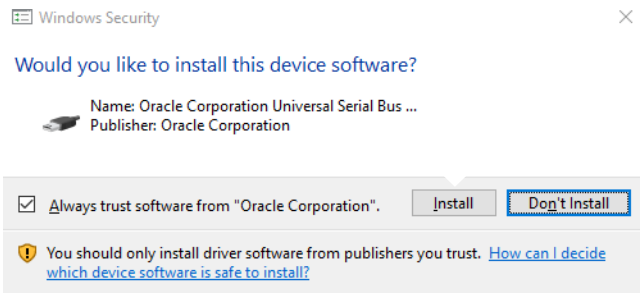


- The **Ready to Install** window will appear. Click on the **INSTALL** button to execute the installation process.



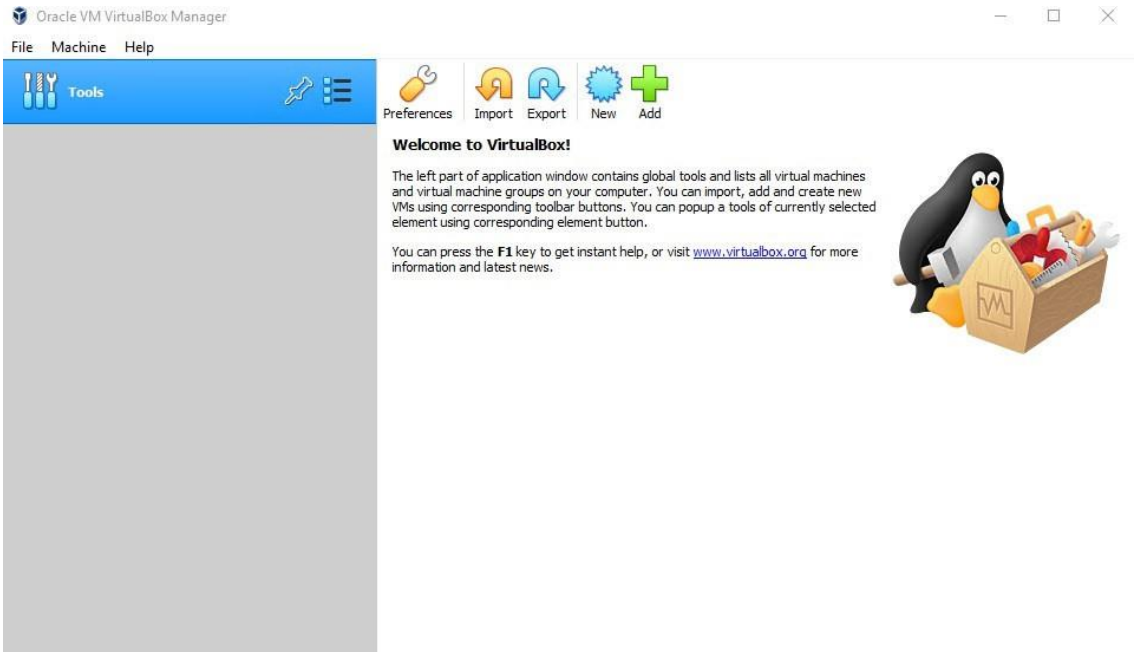
- The installation process will then commence. A status screen similar to that below will appear. You may receive a "**Would you like to install this device software?**" prompt similar to that presented at the bottom of this page. Click on the **INSTALL** button to approve the driver installation.





- You will see the window below once the installation is complete. Check the "**Start Oracle VM VirtualBox. Installation**" item as shown below. Then click on the **FINISH** button to close out the installation program.

- The **Oracle VM VirtualBox Manager** application will then open. You can proceed to Part 2 (next page) of this guidance document.

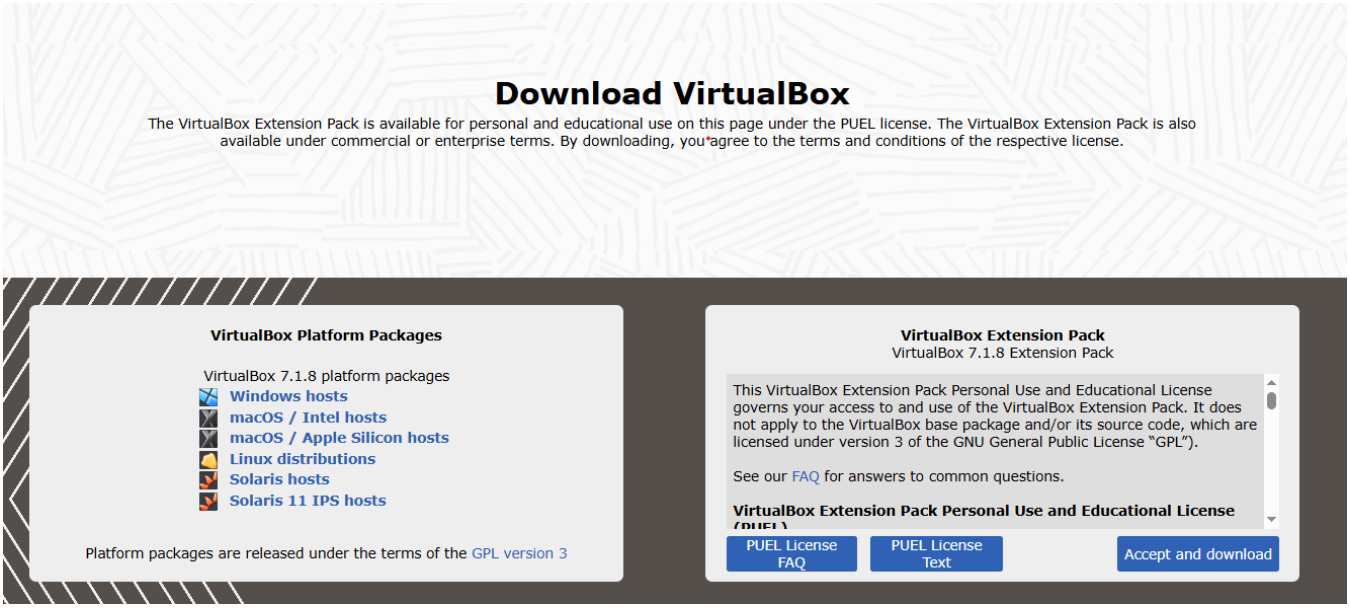


Part2–Oracle VirtualBox Extension Pack Installation

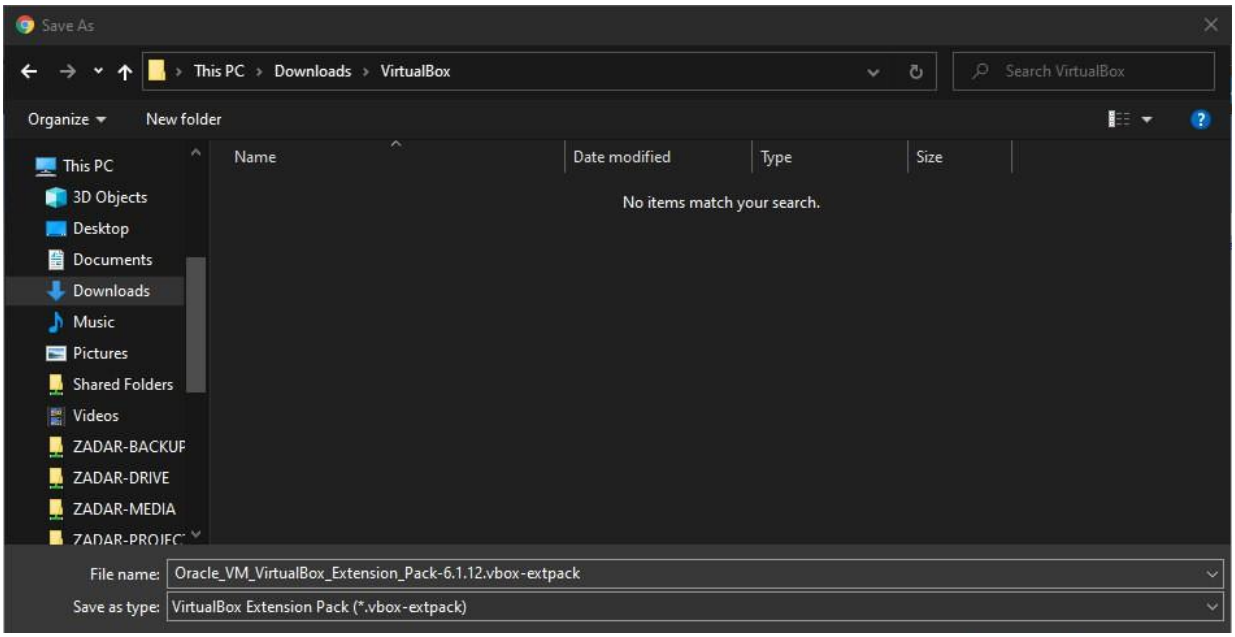
- Locate the Oracle Virtual Box Extension Pack download from the following site:

https://www.virtualbox.org/wiki/Downloads

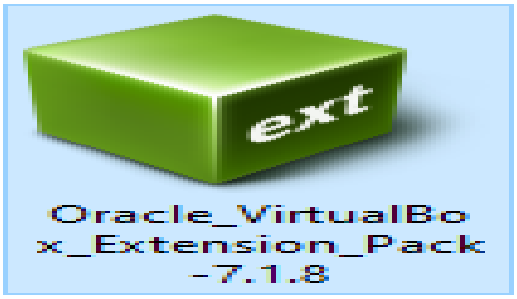
With your computer mouse, click on the "**AllSupportedPlatforms**" link.



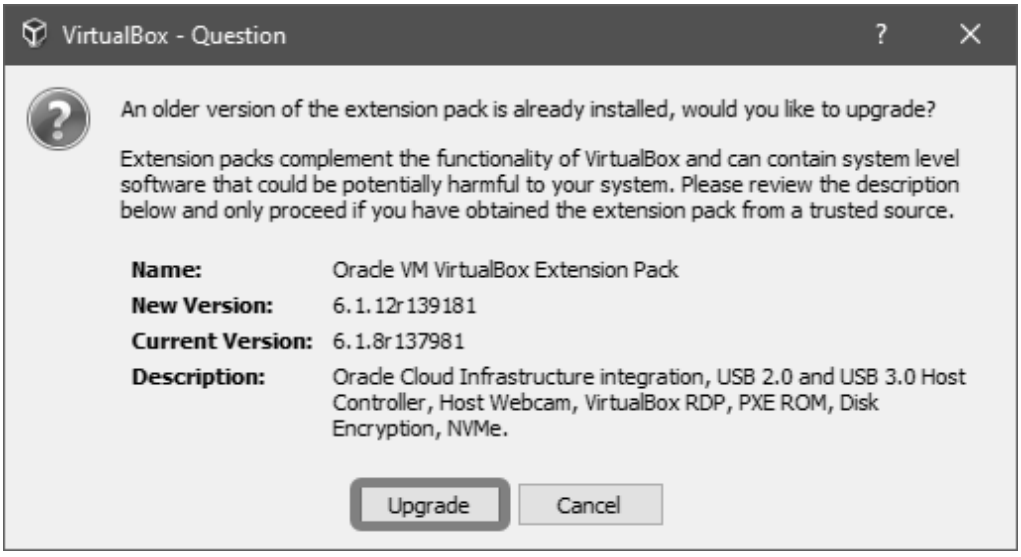
- A “**SaveAs**” prompt will appear. Navigate to an accessible location on your computer and then click on the **SAVE** button to save the extension pack file.



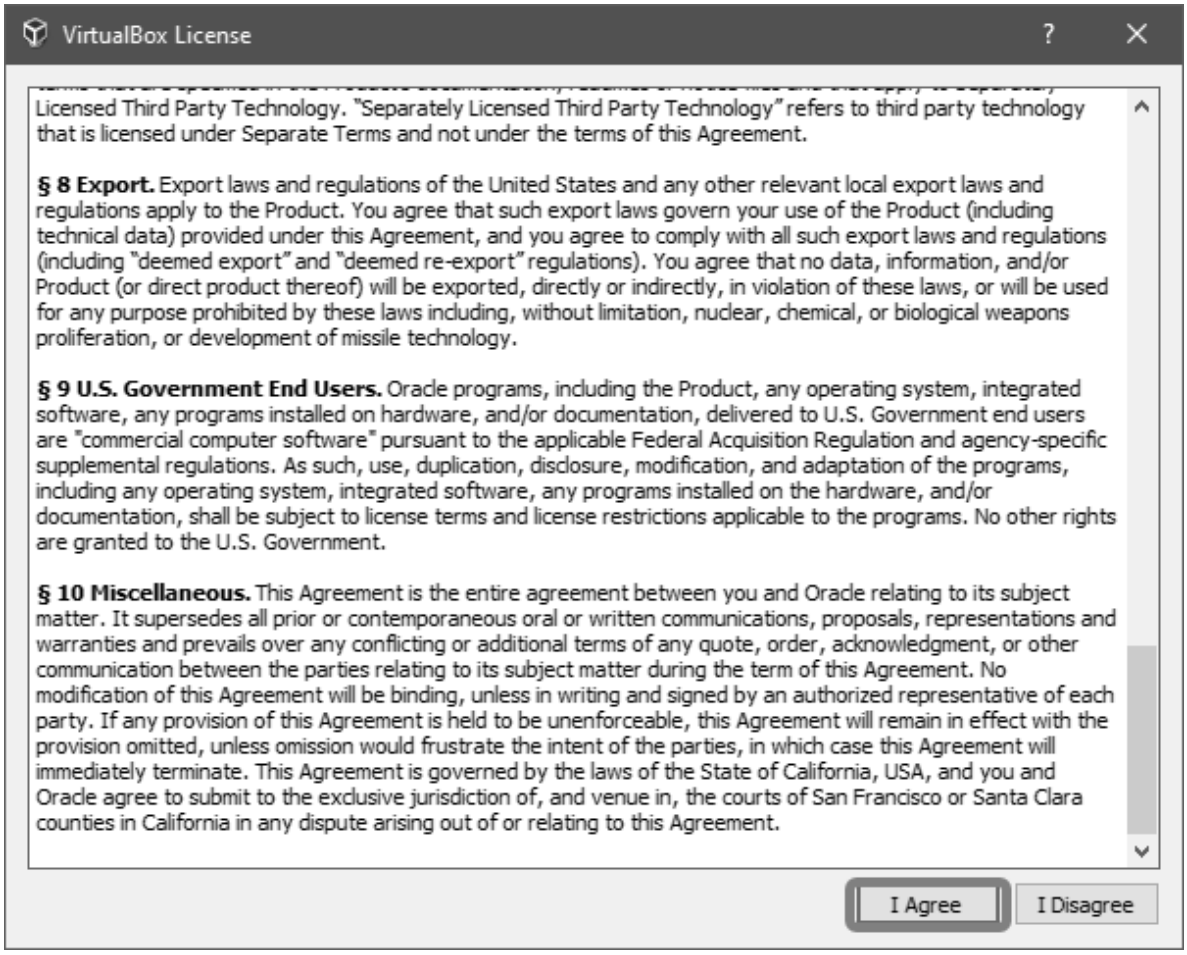
- Access the Windows File Explorer application and navigate to the location of the extension pack file. With your computer mouse, double-click on the Oracle VirtualBox Extension Pack file.



- You may receive the following prompt if you already have Oracle VirtualBox and the Oracle VirtualBox Extension Pack installed. If you receive this prompt, click on the **UPGRADE** button to perform the upgrade.

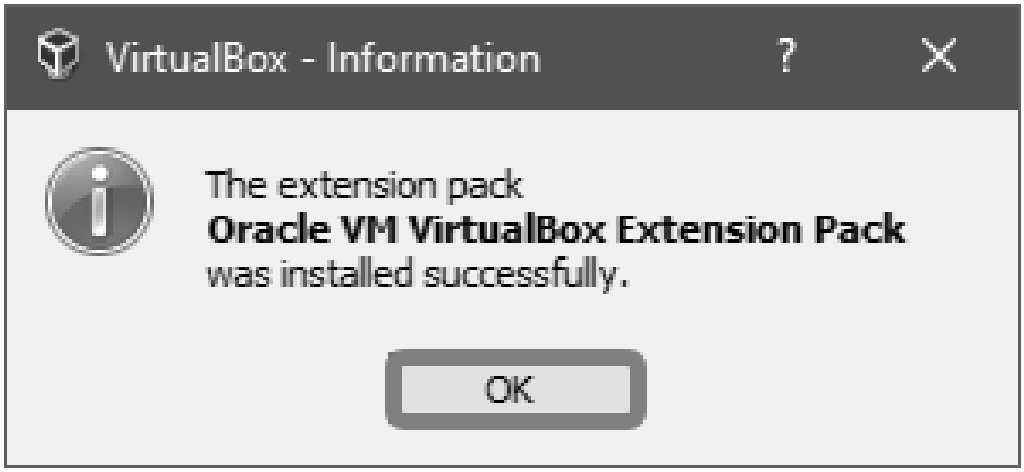


- You'll then see the VirtualBox license agreement screen. Scroll through and review the licensing information. Click on the **I AGREE** button to continue the installation.



- You may then receive a **User Account Control(UAC)** prompt similar to that presented below. Click on the **YES** button to continue the installation.

- You'll receive the following prompt once the installation has finished. Click on the **OK** button to close out the prompt.



Part3–Cloudera QuickStart VM Installation

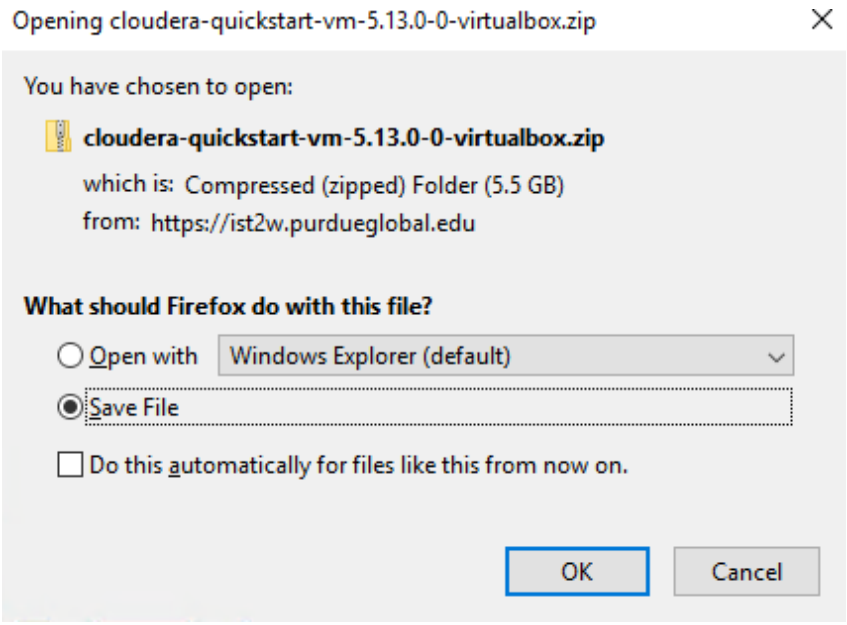
- Using a web browser like Google Chrome or Mozilla Firefox, download the Cloudera QuickStart VM Version5.13 archive file via one of the following links:

  https://downloads.cloudera.com/demo_vm/virtualbox/cloudera-quickstart-vm-5.13.0-0-virtualbox.zip

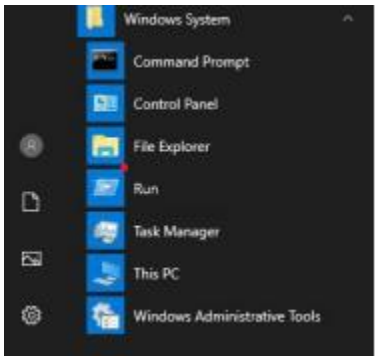
  OR

  https://ist2w.purdueglobal.edu/iso/cloudera-quickstart-vm-5.13.0-0-virtualbox.zip

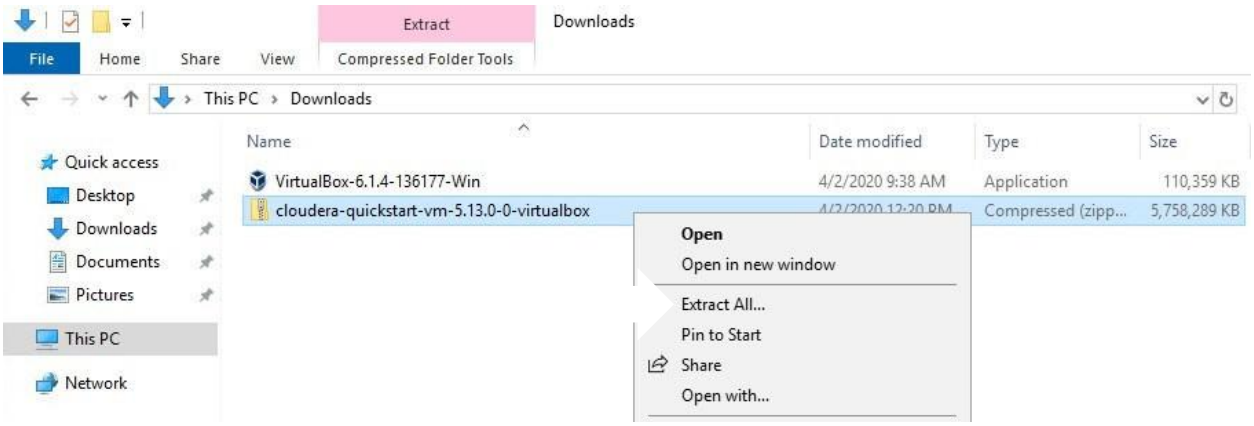
  Depending on how your Windows system is configured, the archive file will either automatically load into the **Downloads** folder or you will be prompted for allocation to save the file. Save the file to an accessible folder location on your computer (e.g., **Documents**, **Downloads**, etc.). The archive file is approximately 6 GB in size. The amount of time it takes for the download to complete is dependent on broadband speed.



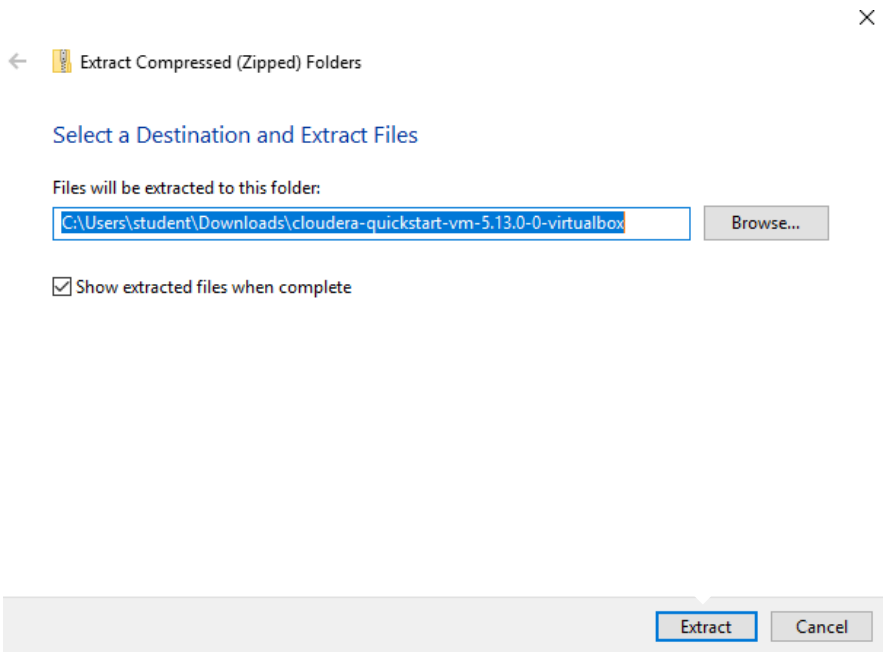
- Open the **FileExplorer** application.InWindows10,theprogramislocatedin **StartMenu-->WindowsSystem-->FileExplorer**.



- In **File Explorer**, navigate to the location the Cloudera QuickStart VM archive file was downloaded to. Right-click on the file (**cloudera-quickstart-vm-5.13.0-0- virtualbox.zip**) with your computer mouse and select the **Extract All...** option from the right-click menu.

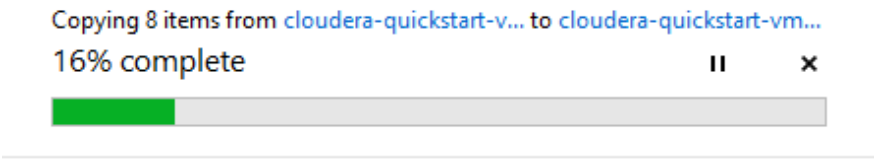


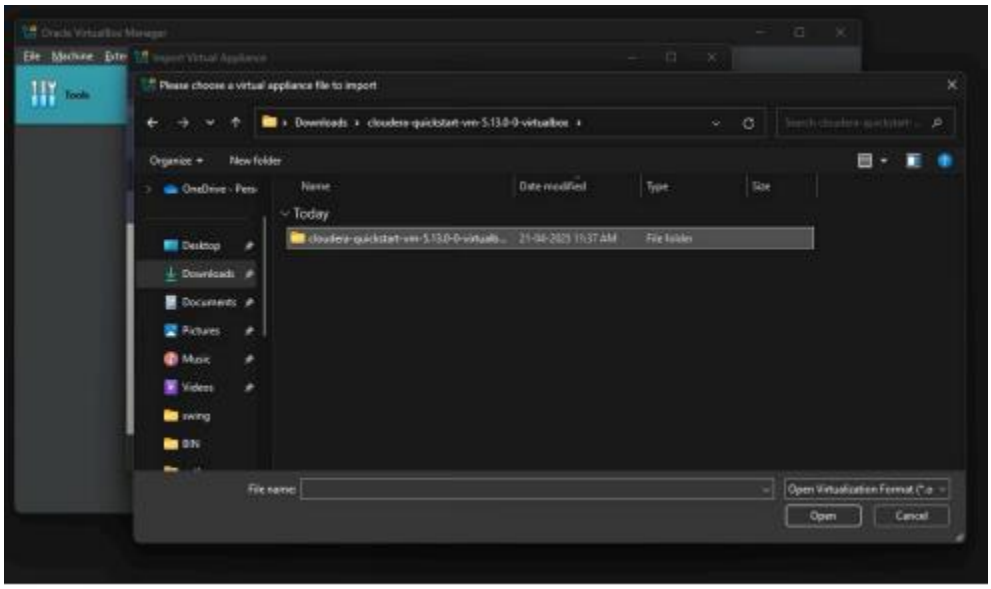
- You'll then be prompted for an extraction location. You can accept the specified default location or change it to a personal preference. Please make sure you note where the archive contents are extracted to. Click on the **EXTRACT** button to initiate the extraction process.



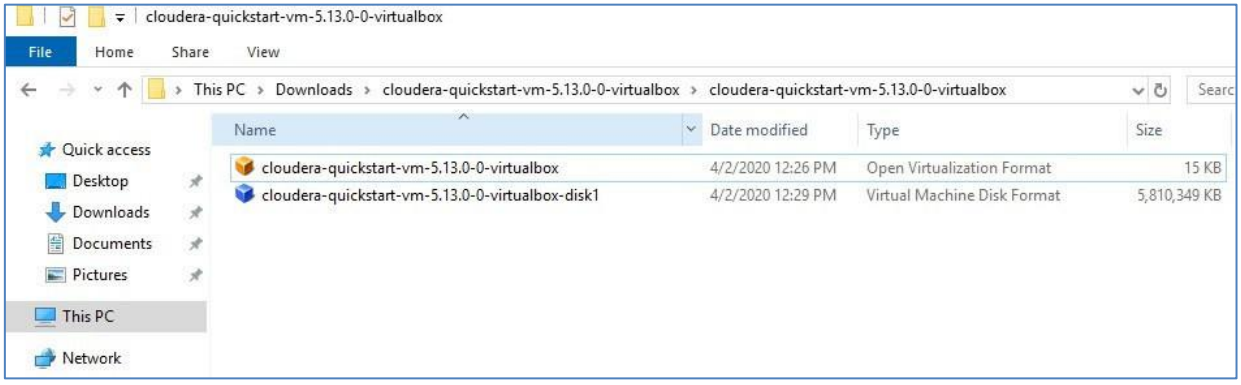
- You'llthenbepresentedwithanextractionprogressbar.Thetotaltimetocompletethe extraction process is dependent on system capabilities.

- Once the extract progress is complete, a new **File Explorer** window will appear showing the extracted contents. Double-click on the **cloudera-quickstart-vm-5.13.0-0- virtualbox** folder to view its contents



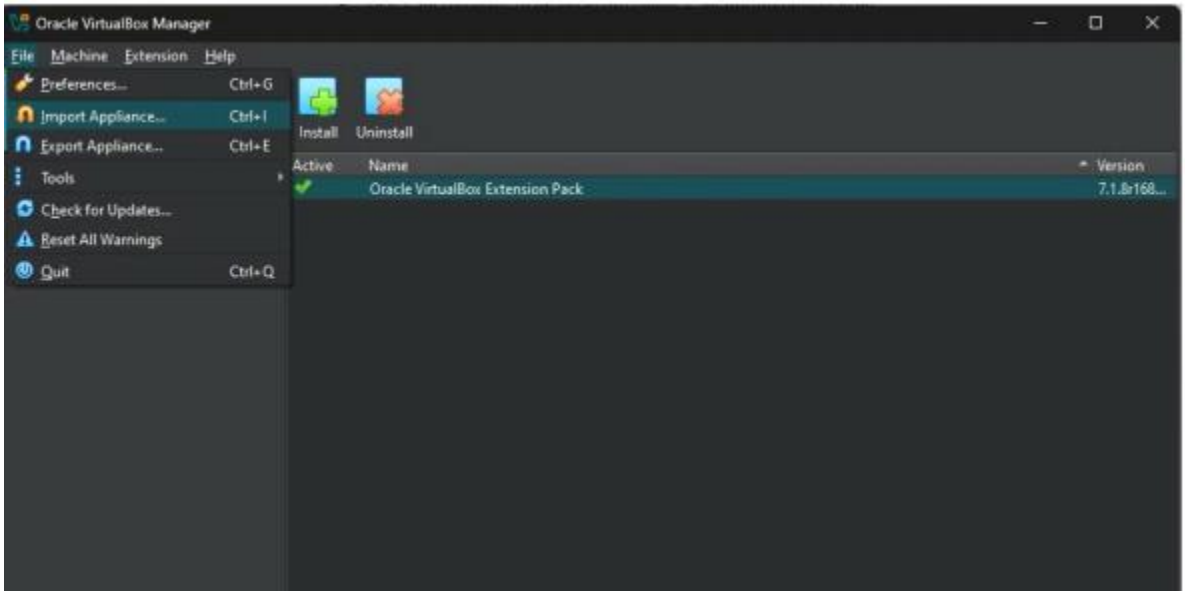


- There should be two files contained within the **cloudera- quickstart-vm-5.13.0-0-virtualbox** folder. The first item is an open virtualization format (OVF) file that contains information on the virtual machine. The second item contains the virtual machine's storage components in virtual machine disk format (VMDF).

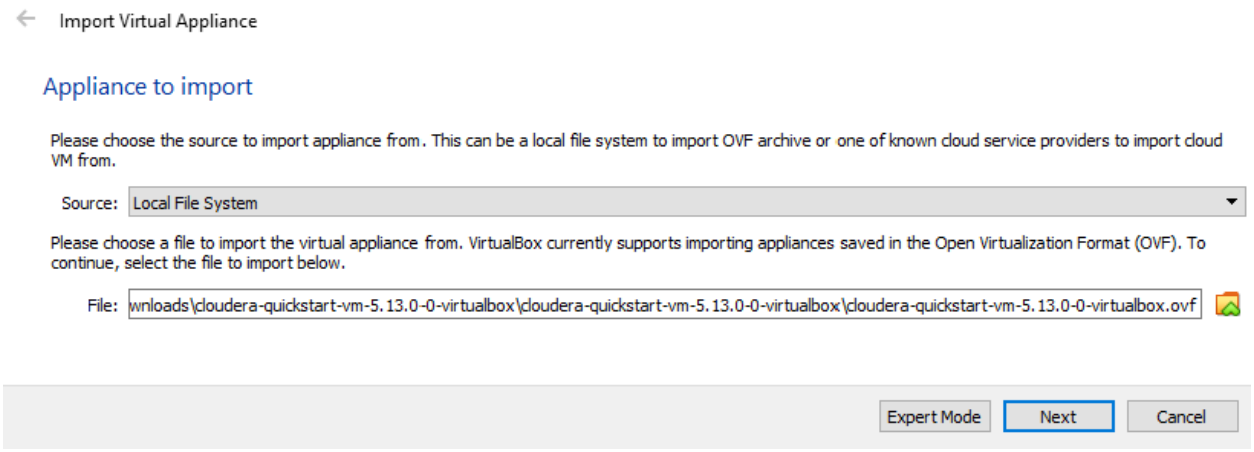


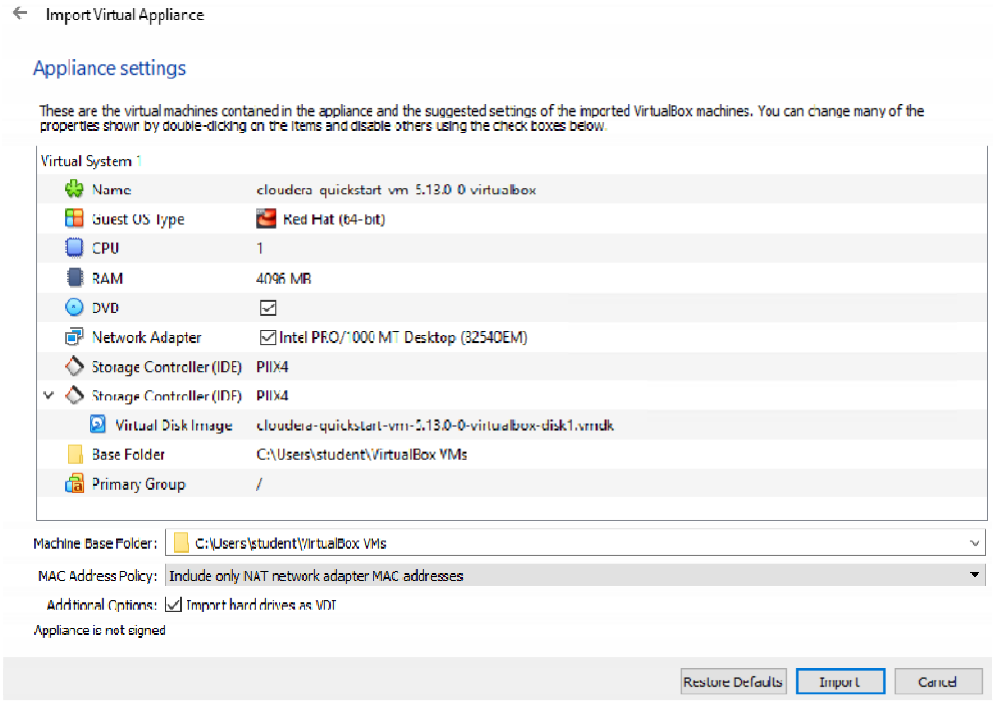
- In Windows10/11,theprogramis located in **Start Menu □ Oracle VM VirtualBox □ Oracle VM VirtualBox**.

In the **Oracle VM Virtual Box Manager** window, click on the **File□ImportAppliance**option from the top menu.

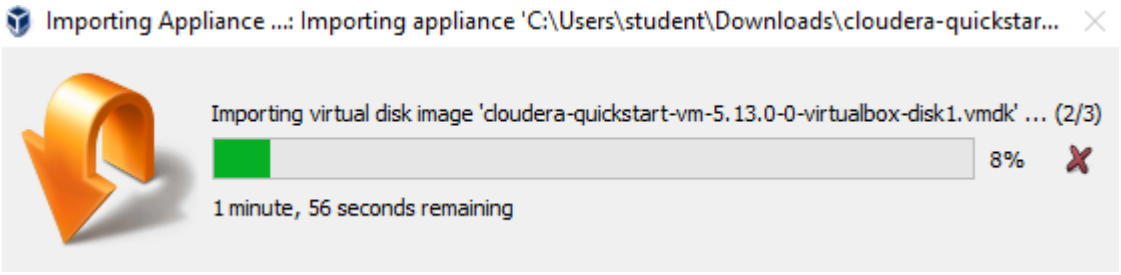


- the **File** text field of the **Import Virtual Appliance** window will now show the OVFfile along with its directory path. Click on the **NEXT** button to continue.

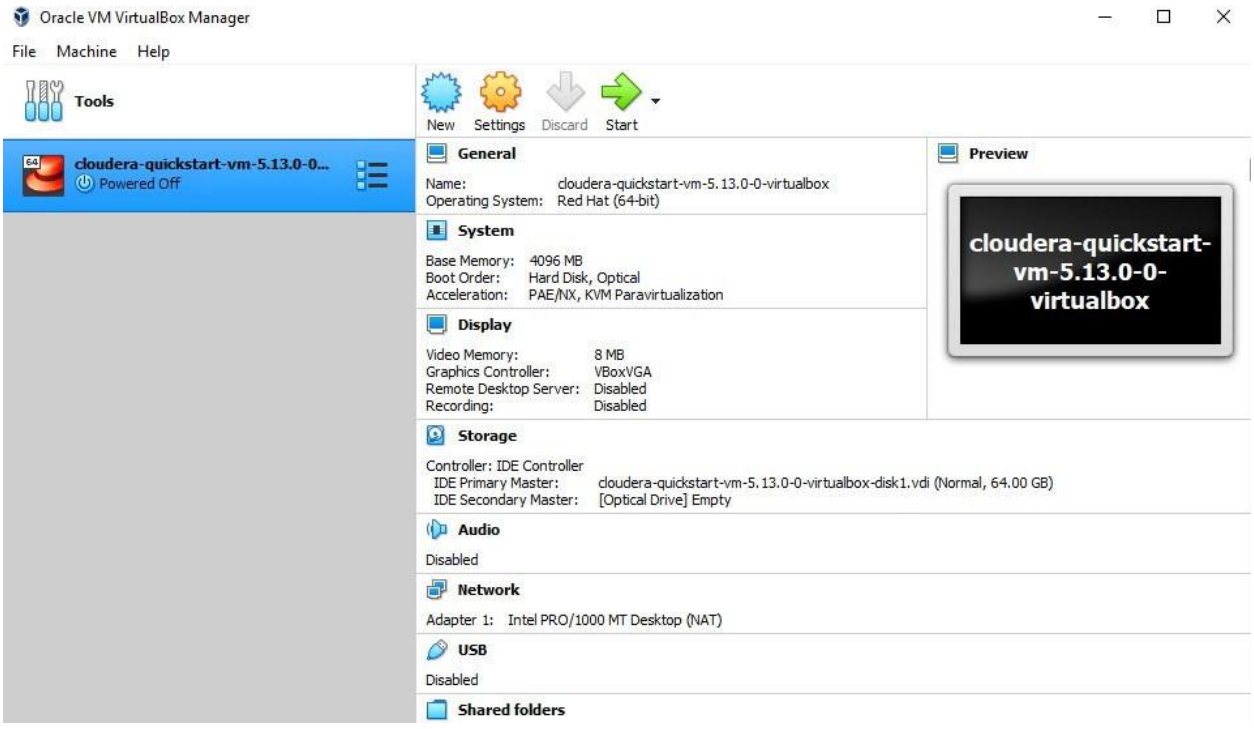




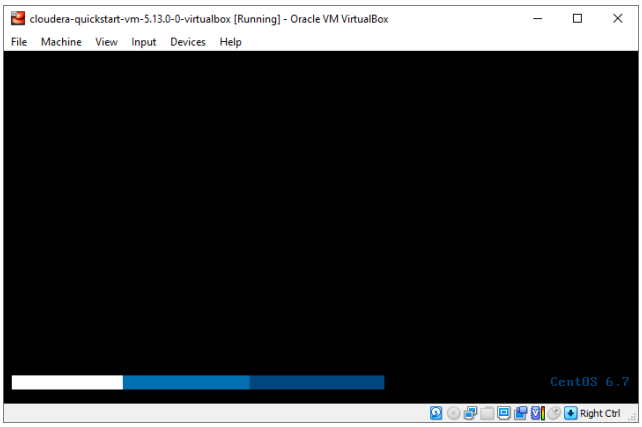
- You'll then be presented with the appliance import progress bar. The total time to complete the import process is dependent on system capabilities. The progress bar will disappear once the import process is complete.



- Following the import process, the **Oracle VM VirtualBox Manager** window will show the newly imported Cloudera QuickStart VM in the inventory list located in the left window pane. Highlight or select the Cloudera QuickStart VM entry and then select the **START** button to boot-up the VM.



- The VM will begin to boot-up in a new window as depicted below.

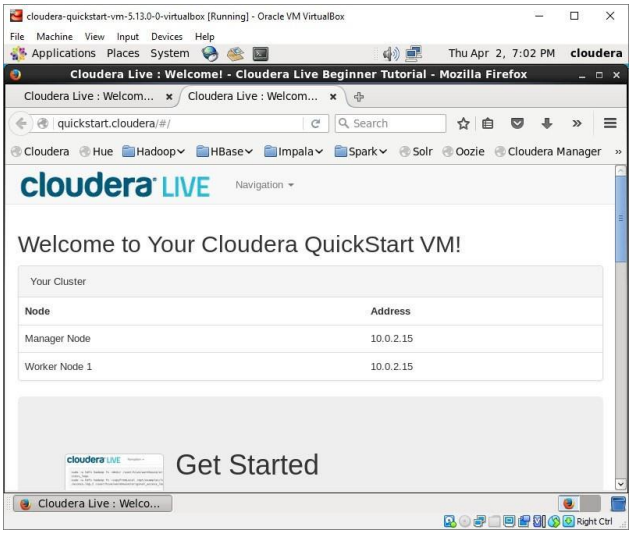


- The VM will automatically login as the "**cloudera**" user. The username/password for applications within the VM are as follows:

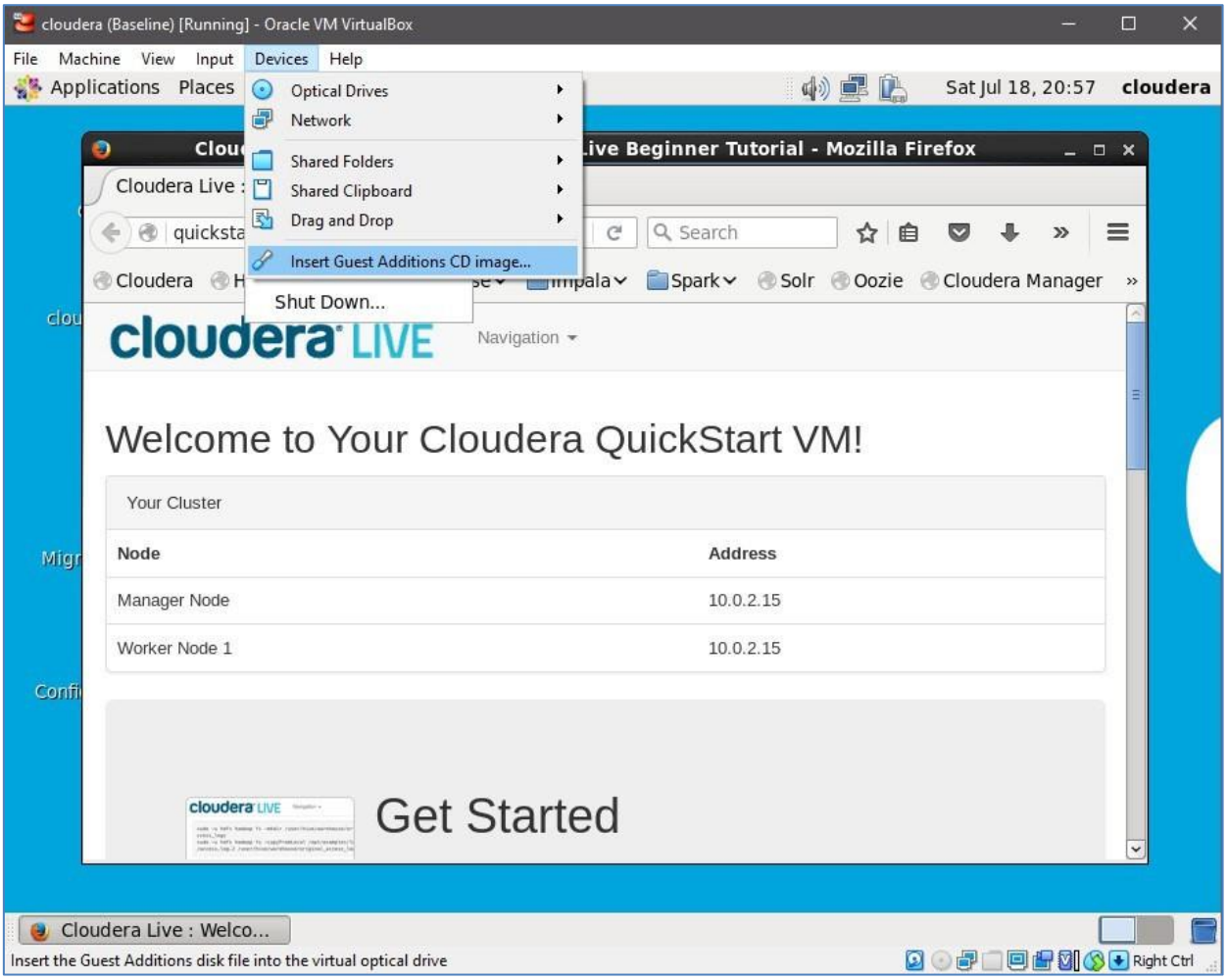
Username :**cloudera**

Password:**cloudera**

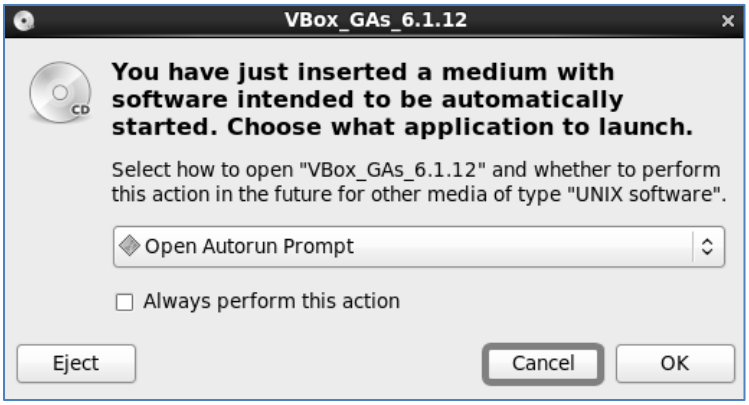
- The Mozilla Firefox browser will automatically open during startup displaying a welcome message similar to that illustrated below. This indicates the VM booted properly.



- You now need to install VirtualBox Guest Additions functionality in the Cloudera virtual machine. In the top menu of the virtual machine window, select **DEVICES**-**INSERT GUEST ADDITIONS CD IMAGE**.



- You may then see the following prompt. Click on the **CANCEL** Button to close out the prompt.



- On the Linux desktop, double click on the drive folder with the name starting with "**VBox_Gas**" and depicted with a CD-ROM icon.

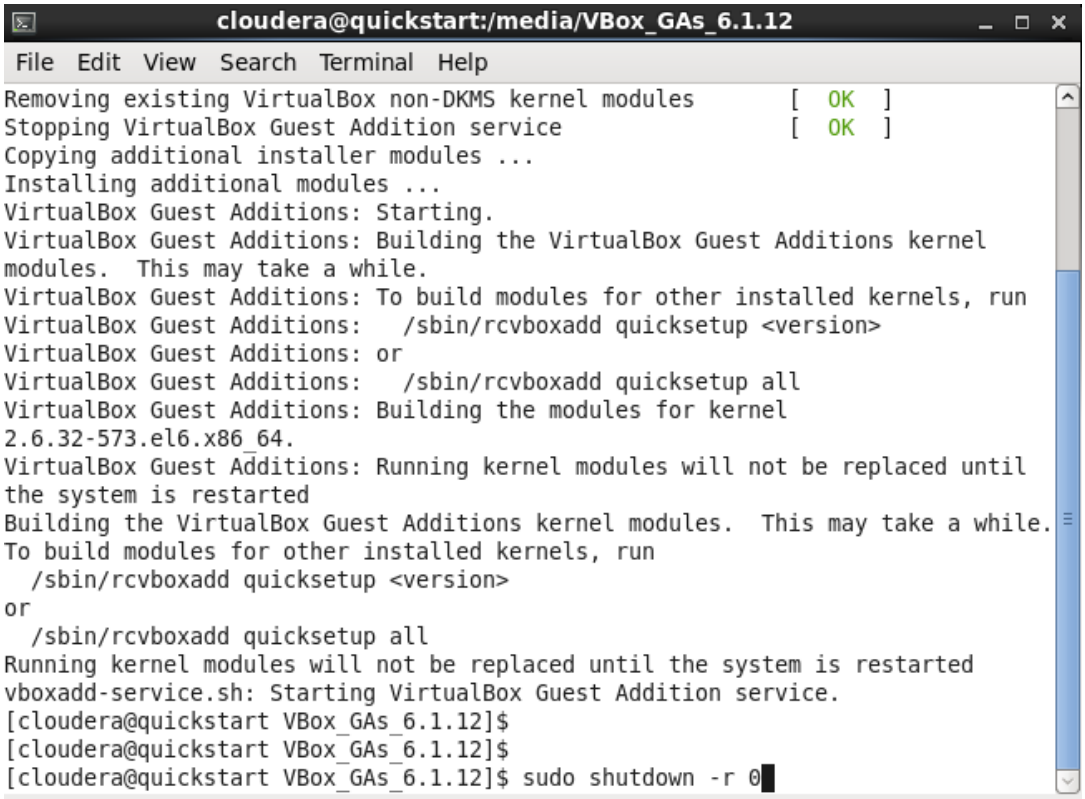
- A Terminal window will then appear. Execute the following statement at the Linux command prompt:

sudo./VBoxLinuxAdditions.run

- The installation process will begin. Accept the defaults if prompted for any options.

Verify the Installation

1. Check the status of services like HDFS, YARN, and Hive in the Cloudera Manager dashboard.
2. Run basic Hadoop commands in the terminal to ensure everything is working:



Installation of Apache Hadoop on Windows 10

Apache Hadoop is an open-source framework used for storing and processing large-scale data across clusters of computers. It is designed to handle big data efficiently, even when the data is unstructured or semi-structured. Hadoop follows a distributed computing model, making it highly scalable and fault-tolerant.

Key Components of Hadoop

1. **Hadoop Distributed File System (HDFS):**
    - HDFS is responsible for storing data.
    - It breaks large files into smaller blocks and distributes them across multiple nodes in a cluster.
    - It ensures fault tolerance by replicating data blocks across different nodes.
2. **MapReduce:**
    - A programming model used for processing large datasets.
    - It consists of two main functions:
        - **Map:** Divides the input data into smaller chunks and processes them in parallel.
        - **Reduce:** Aggregates and summarizes the results.
3. **YARN (Yet Another Resource Negotiator):**
    - Acts as the cluster management tool.
    - It schedules and manages resources for applications running on Hadoop.
4. **Hadoop Common:**
    - Provides essential libraries and utilities needed by other Hadoop modules.

Features of Apache Hadoop

- **Scalability:** Can handle increasing amounts of data by adding more nodes to the cluster.
- **Fault Tolerance:** Automatically recovers from failures by replicating data.
- **Cost-Effective:** Uses commodity hardware to build clusters, making it affordable for businesses.
- **Parallel Processing:** Efficiently processes large datasets by dividing tasks across nodes.

Applications of Hadoop

Hadoop is widely used across industries to manage and analyze big data. Some common applications include:

- **Data Warehousing:** Storing and retrieving massive amounts of data for analysis.
- **Social Media Analytics:** Analyzing user interactions and trends on platforms like Facebook and Twitter.
- **Recommendation Systems:** Platforms like Netflix and Amazon use Hadoop to recommend products and content.
- **Scientific Research:** Processing complex datasets in fields like genomics and climate studies.

Objective:

This assignment aims to guide you through the steps required to install Apache Hadoop on a Windows 10 system and understand its setup and configuration process.

Part 1: Prerequisites

- **Download and Install Java Development Kit (JDK):**

  Visit the Oracle JDK download page.

  Install the JDK and configure the `JAVA_HOME` environment variable.

- **Download and Install WinRAR or 7-Zip:**

  Use this tool to extract Hadoop files efficiently.

Downloading the Java 8

https://www.java.com/en/download/windows_offline.jsp

Part 2: Hadoop Installation

1. **Download Apache Hadoop:**
   - Visit Apache Hadoop Releases and download the latest version.
   - Extract the Hadoop `.tar.gz` file using WinRAR or 7-Zip.

2. **Set Environment Variables:**

- Configure the `HADOOP_HOME` and update the `PATH` environment variable to include `%HADOOP_HOME%\bin`.

Click on Download Button it will start the downloading process

Downloading the Hadoop 3.4.0

https://dlcdn.apache.org/hadoop/common/hadoop-3.4.0/hadoop-3.4.0.tar.gz

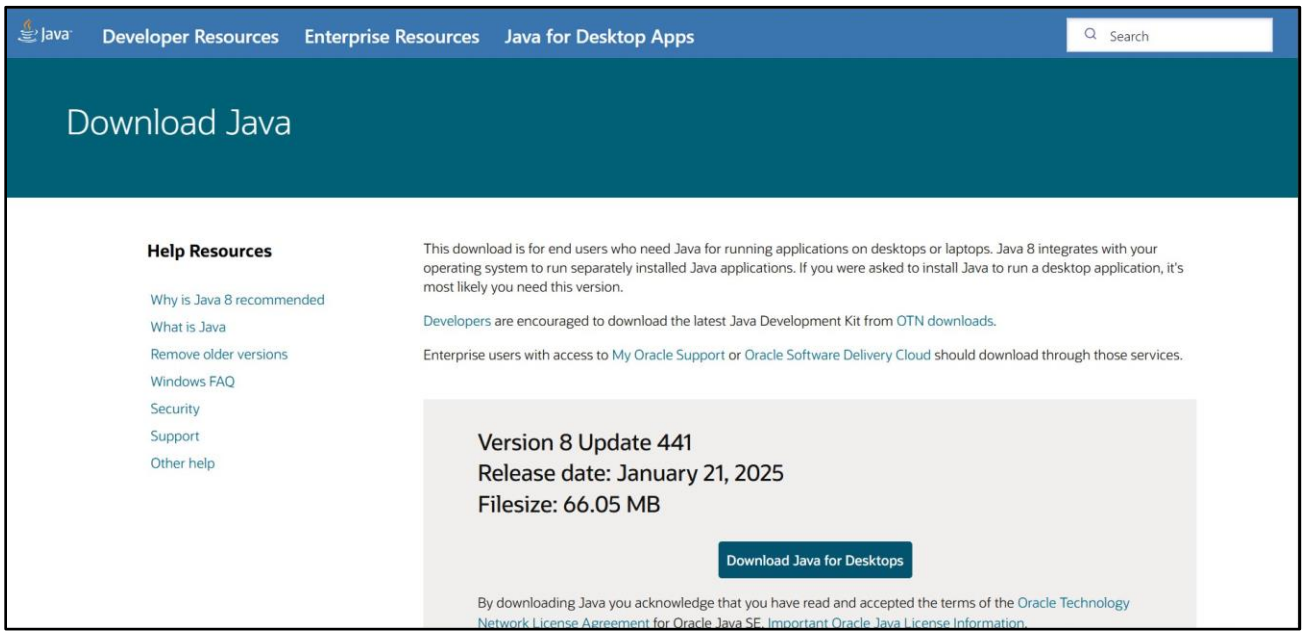
Click on the link it will automatically start the downloading process

Downloading the 7Zip

Downloading the Hadoop I/O Files

https://onedrive.live.com/?id=AB81AF6E7ADDA0B4%213422&cid=AB81AF6E7ADDA0B4

Copy pastes the Hadoop I/O file in the **bin folder** i.e. C:\hadoop-3.4.0\hadoop-3.4.0\bin



Setting up environment variables for both Java and Hadoop:

Step 1: Setting Up the JAVA_HOME Variable

- **Locate the JDK Installation Path:**

  After installing the JDK, note its installation directory.

- **Set the JAVA_HOME Variable:**

  Open *System Properties*:

  Right-click on "This PC" or "My Computer," click on "Properties."

  Go to "Advanced system settings" and click "Environment Variables."

- Under *System Variables*, click "New."

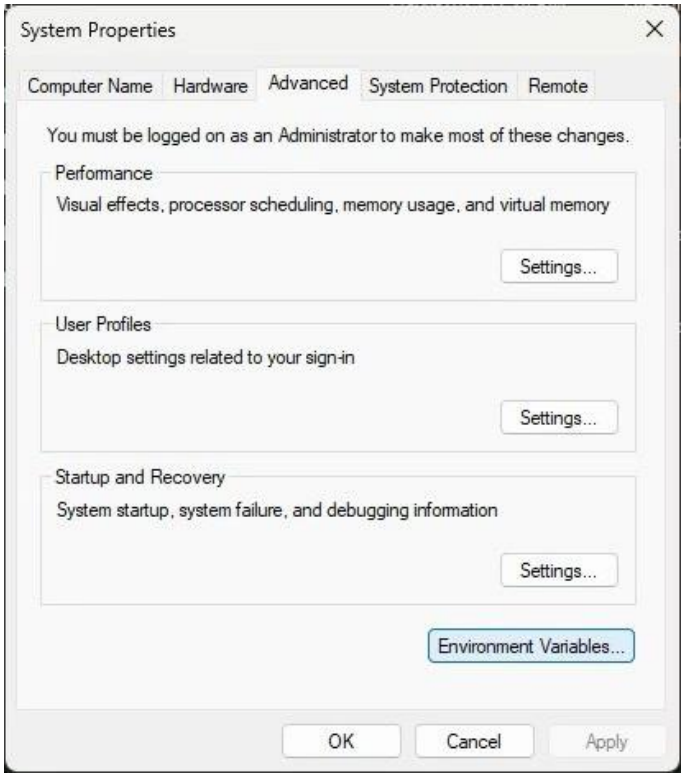
  Variable Name: `JAVA_HOME`

  Variable Value: Path to the JDK installation (e.g., `C:\Program Files\Java\jdk-<version>`).

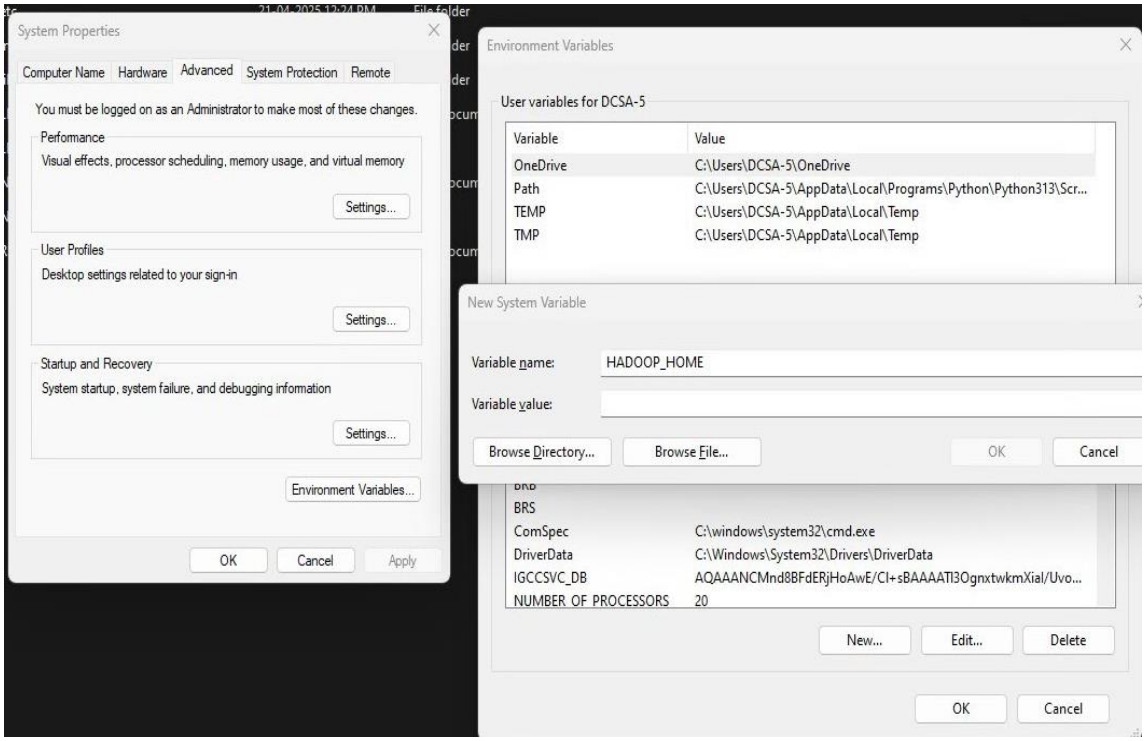
- **Update the PATH Variable:**

  In the same "Environment Variables" window, locate the `Path` variable under *System Variables*.

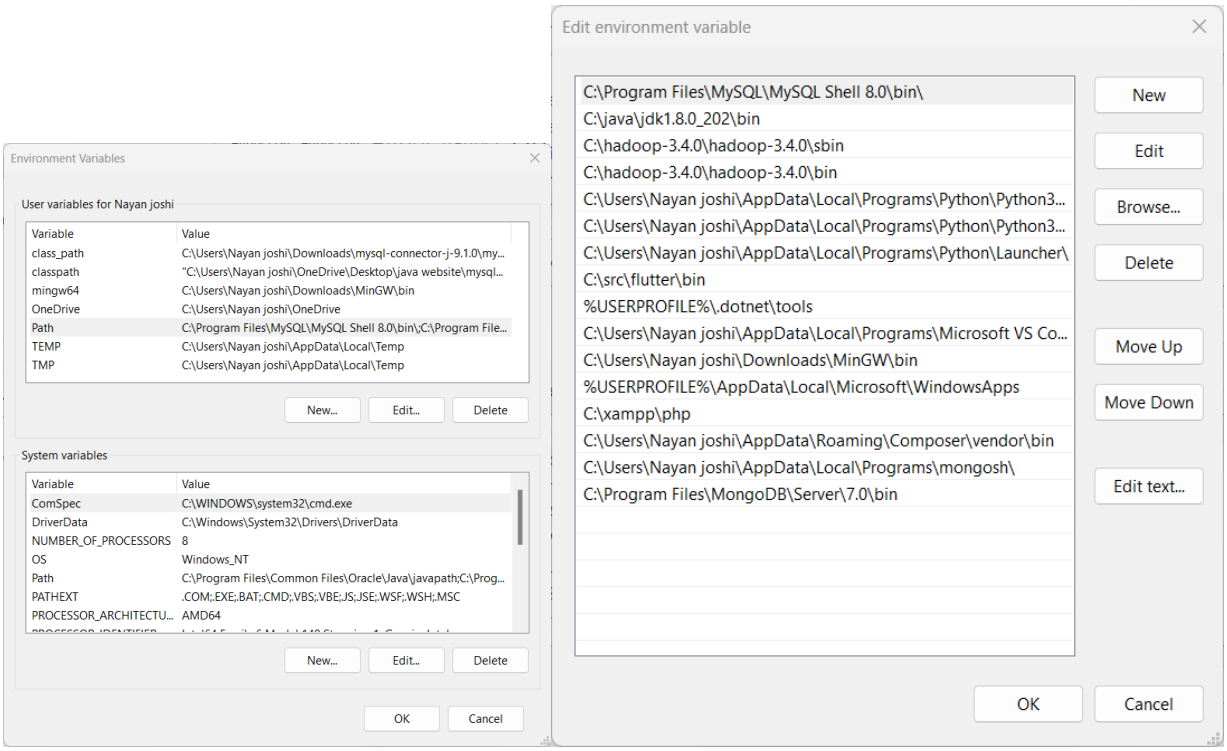
  Click "Edit," then "New," and add:



Click on 'New' under the TextBox 'System Variables' and enter variable name as HADOOP_HOME and value as C:\hadoop-3.4.0\hadoop-3.4.0;



Click on Path > Click on Edit button



Path:

- Java bin: **C:\java\jdk1.8.0_361\bin**
- Hadoop bin: **C:\hadoop-3.4.0\hadoop-3.4.0\bin**
- Hadoop sbin: **C:\hadoop-3.4.0\hadoop-3.4.0\sbin**

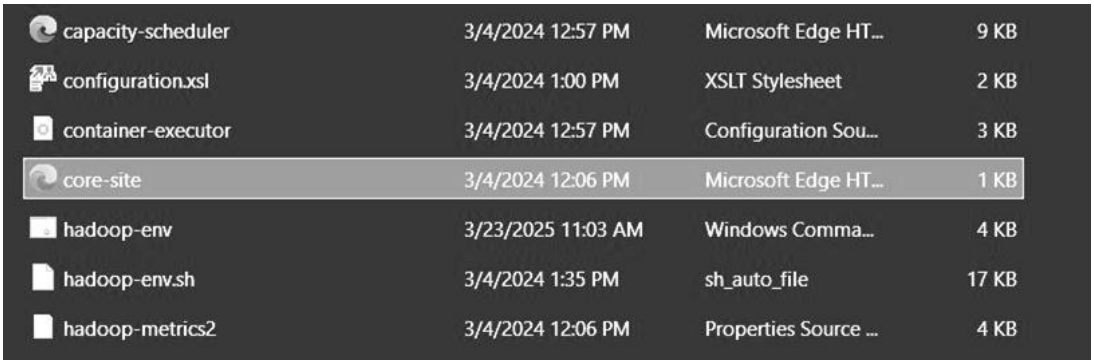
Configure HADOOP Files :

A) Edit hadoop-env.cmd file available in C:\hadoop-3.4.0\hadoop-3.4.0\etc\hadoop

Open the file in notepad > move to line number 25 set the value of **JAVA_HOME = C:\java\jdk1.8.0_361**

```
@rem distributed under the License is distributed on an "AS IS" BASIS,
@rem WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
@rem See the License for the specific language governing permissions and
@rem limitations under the License.

@rem Set Hadoop-specific environment variables here.

@rem The only required environment variable is JAVA_HOME.  All others are
@rem optional.  When running a distributed configuration it is best to
@rem set JAVA_HOME in this file, so that it is correctly defined on
@rem remote nodes.

@rem The java implementation to use.  Required.
set JAVA_HOME = C:\java\jdk1.8.0_361

@rem The jsvc implementation to use. Jsvc is required to run secure datanodes.
@rem set JSVC_HOME=%JSVC_HOME%

@rem set HADOOP_CONF_DIR=
```

B) Edit hadoop-env.cmd file available in C:\hadoop-3.4.0\hadoop-3.4.0\etc\hadoop



Open the file in notepad > copy paste the following code between <configuration> tags

<property>

<name>fs.default.name</name>

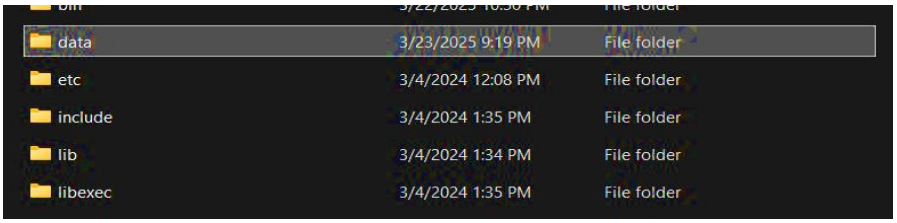
<value>hdfs://localhost:9820</value>

</property>

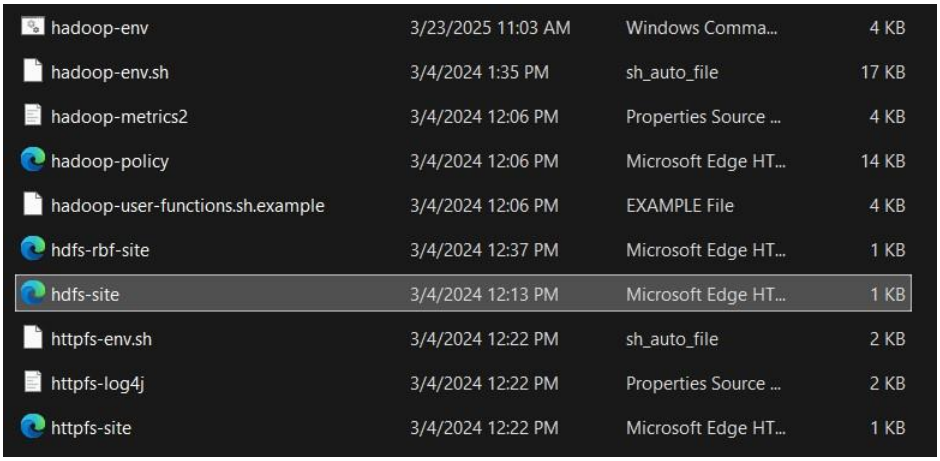
```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!--
  Licensed under the Apache License, Version 2.0 (the "License");
  you may not use this file except in compliance with the License.
  You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

  Unless required by applicable law or agreed to in writing, software
  distributed under the License is distributed on an "AS IS" BASIS,
  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  See the License for the specific language governing permissions and
  limitations under the License. See accompanying LICENSE file.
-->

<!-- Put site-specific property overrides in this file. -->

<configuration>
 <property>
      <name>fs.default.name</name>
      <value>hdfs://localhost:9820</value>
 </property>
</configuration>
```

C) Creating the data in Hadoop folder

Create the folder structure like (data -> dfs -> datanode) and (data -> dfs ->namenode)



D) Edit hdfs-site.xml file available in C:\hadoop-3.4.0\hadoop-3.4.0\etc\hadoop



Open the file in notepad > copy paste the following code between <configuration> tags

<property>

<name>dfs.replication</name>

<value>1</value>

</property>

<property>

<name>dfs.namenode.name.dir</name>

<value>file:///C:/hadoop/data/dfs/namenode</value>

</property>

<property>

<name>dfs.datanode.data.dir</name>

<value>file:///C:/hadoop/data/dfs/datanode</value>

</property>

```
<!-- Put site-specific property overrides in this file. -->

<configuration>
 <property>
     <name>dfs.replication</name>
     <value>1</value>
     </property>
     <property>
     <name>dfs.namenode.name.dir</name>
     <value>file:///C:/hadoop/data/dfs/namenode</value>
     </property>
     <property>
     <name>dfs.datanode.data.dir</name>
     <value>file:///C:/hadoop/data/dfs/datanode</value>
     </property>

</configuration>
```

Repalce the path of namenode and datanode in value tags

```
<configuration>
 <property>
     <name>dfs.replication</name>
     <value>1</value>
     </property>
     <property>
     <name>dfs.namenode.name.dir</name>
     <value>C:\hadoop-3.4.0\hadoop-3.4.0\data\dfs\namenode</value>
     </property>
     <property>
     <name>dfs.datanode.data.dir</name>
     <value>C:\hadoop-3.4.0\hadoop-3.4.0\data\dfs\datanode<value>
     </property>

</configuration>
```

E) Edit mapred-site.xml file available in C:\hadoop-3.4.0\hadoop-3.4.0\etc\hadoop

Open the file in notepad > copy paste the following code between <configuration> tags

<property>

<name>mapreduce.framework.name</name>

<value>yarn</value>

<description>MapReduce framework name</description>

</property>

```
<!-- Put site-specific property overrides in this file. -->

<configuration>
 <property>
     <name>mapreduce.framework.name</name>
     <value>yarn</value>
     <description>MapReduce framework name</description>
     </property>
</configuration>
```

F) Edit mapred-site.xml file available **in C:\hadoop-3.4.0\hadoop-3.4.0\etc\hadoop**

Open the file in notepad > copy paste the following code between <configuration> tags

<property>

<name>yarn.nodemanager.aux-services</name>

<value>mapreduce_shufle</value>

<description>Yarn Node Manager Aux Service</description>

</property>

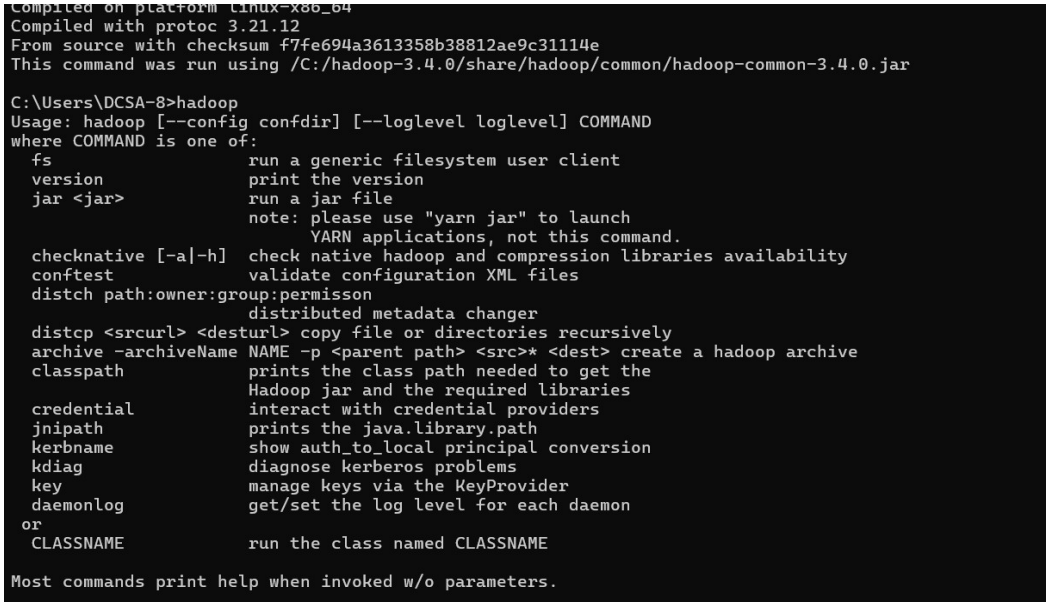
```
<configuration>

<!-- Site specific YARN configuration properties -->
<property>
     <name>yarn.nodemanager.aux-services</name>
     <value>mapreduce_shuffle</value>
     <description>Yarn Node Manager Aux Service</description>
     </property>
</configuration>
```

After this, try the commands:

> Java - version

> hadoop

> hadoop -version



Format File System:

Open the command prompt > paste the command

" hdfs namenode -format" > **press enter key**

```
C:\Users\DCSA-8>  hdfs namenode -format
2025-04-16 12:58:11,180 INFO namenode.NameNode: STARTUP_MSG:
/************************************************************
STARTUP_MSG: Starting NameNode
STARTUP_MSG:   host = DESKTOP-FB1DFNH/192.168.56.1
STARTUP_MSG:   args = [-format]
STARTUP_MSG:   version = 3.4.0
STARTUP_MSG:   classpath = C:\hadoop-3.4.0\etc\hadoop;C:\hadoop-3.4.0\share\hadoop\common;C:\hadoop-3.4.0\share\hadoop\common\lib\animal-sniffer-annotations
-1.17.jar;C:\hadoop-3.4.0\share\hadoop\common\lib\audience-annotations-0.12.0.jar;C:\hadoop-3.4.0\share\hadoop\common\lib\avro-1.9.2.jar;C:\hadoop-3.4.0\sha
re\hadoop\common\lib\bcprov-jdk15on-1.70.jar;C:\hadoop-3.4.0\share\hadoop\common\lib\checker-qual-2.5.2.jar;C:\hadoop-3.4.0\share\hadoop\common\lib\commons-
beanutils-1.9.4.jar;C:\hadoop-3.4.0\share\hadoop\common\lib\commons-cli-1.5.0.jar;C:\hadoop-3.4.0\share\hadoop\common\lib\commons-codec-1.15.jar;C:\hadoop-3
.4.0\share\hadoop\common\lib\commons-collections-3.2.2.jar;C:\hadoop-3.4.0\share\hadoop\common\lib\commons-compress-1.24.0.jar;C:\hadoop-3.4.0\share\hadoop\
common\lib\commons-configuration2-2.8.0.jar;C:\hadoop-3.4.0\share\hadoop\common\lib\commons-daemon-1.0.13.jar;C:\hadoop-3.4.0\share\hadoop\common\lib\common
s-io-2.14.0.jar;C:\hadoop-3.4.0\share\hadoop\common\lib\commons-lang3-3.12.0.jar;C:\hadoop-3.4.0\share\hadoop\common\lib\commons-logging-1.2.jar;C:\hadoop-3
.4.0\share\hadoop\common\lib\commons-math3-3.6.1.jar;C:\hadoop-3.4.0\share\hadoop\common\lib\commons-net-3.9.0.jar;C:\hadoop-3.4.0\share\hadoop\common\lib\c
ommons-text-1.10.0.jar;C:\hadoop-3.4.0\share\hadoop\common\lib\curator-client-5.2.0.jar;C:\hadoop-3.4.0\share\hadoop\common\lib\curator-framework-5.2.0.jar;
C:\hadoop-3.4.0\share\hadoop\common\lib\curator-recipes-5.2.0.jar;C:\hadoop-3.4.0\share\hadoop\common\lib\dnsjava-3.4.0.jar;C:\hadoop-3.4.0\share\hadoop\com
```

START HADOOP

open the command prompt and type the following commands

- start-hdfs
- start-yarn

It will open 4 new windows cmd terminals for 4 daemon processes, namely **namenode datanode, nodemanager**, and **resourcemanager**. Don t close these windows, minimize them. Closing the windows will terminate the daemons. You can run them in the background if you don t like to see these windows.

You can check this by using the command "Jps"

```
C:\Users\DCSA-8>start-dfs

C:\Users\DCSA-8>start-yarn
starting yarn daemons

C:\Users\DCSA-8>jsp
'jsp' is not recognized as an internal or external command,
operable program or batch file.

C:\Users\DCSA-8>jps
4192 ResourceManager
12084 NameNode
9672 NodeManager
8268 Jps
```

Part 5: Verification

Verify installation by visiting:

- HDFS Web UI: **`http://localhost:9870`**
- YARN Web UI**: `http://localhost:8088`**

Creating a Shared Folder in Window and Linux.

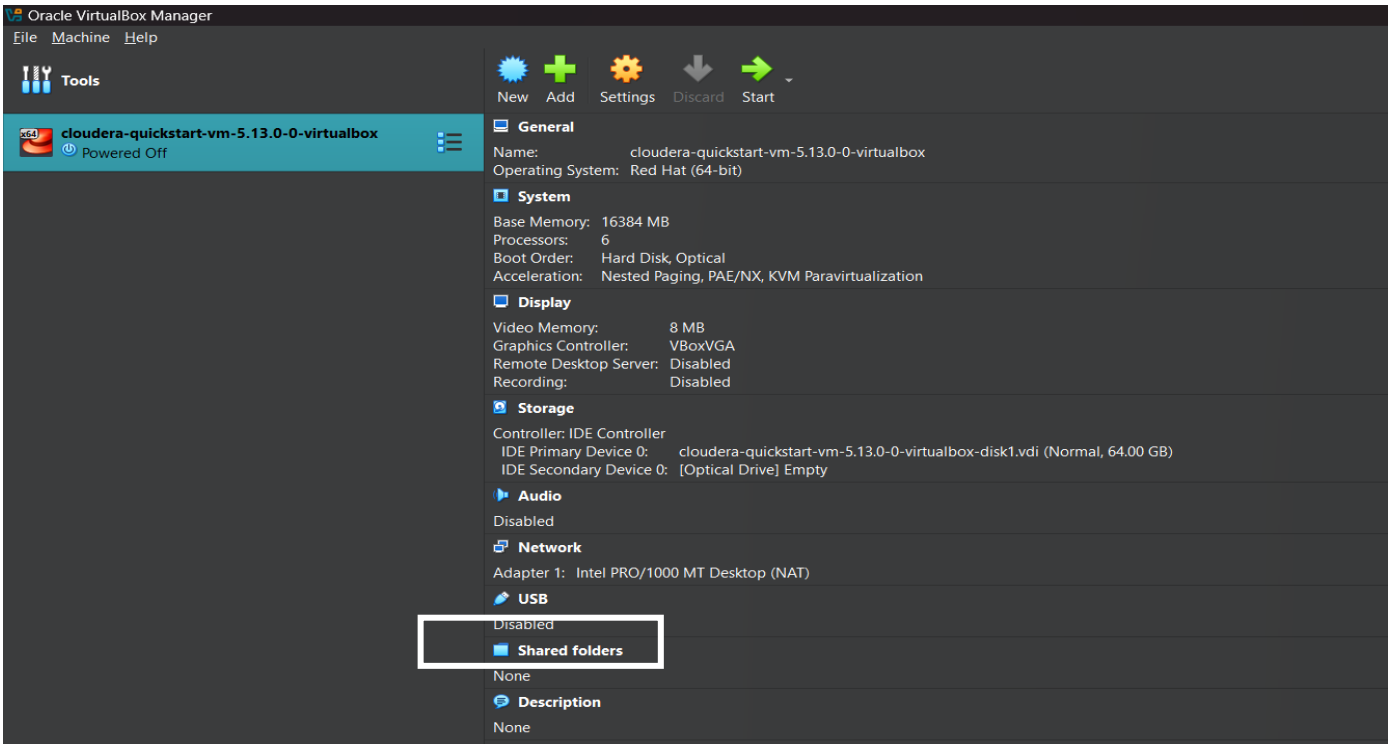
STEP 1

- Create a folder in Host Operating System (Windows)

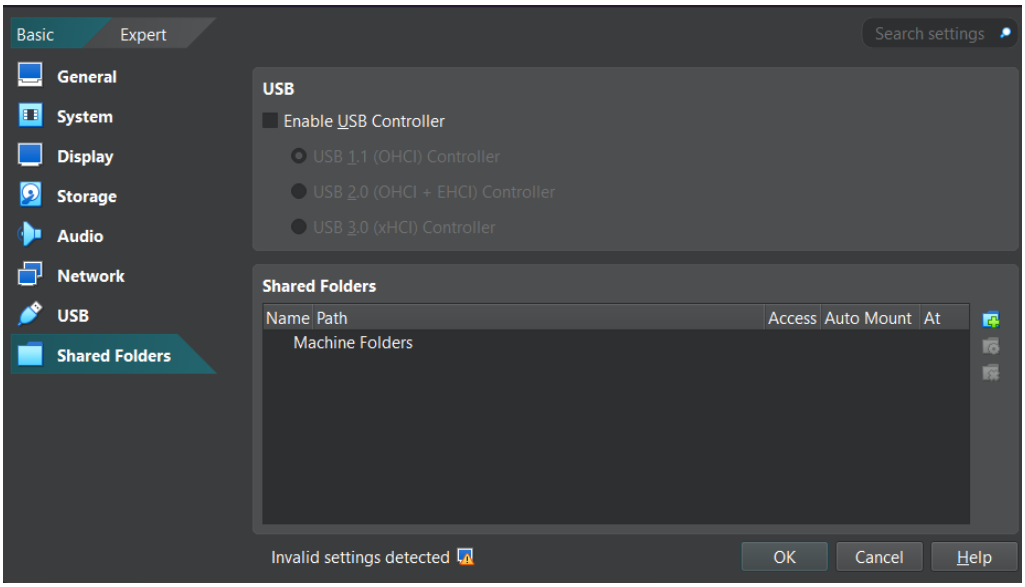
Ctrl +Shift + N >Rename the File or (Press F2).

STEP 2

- Open The **Virtualbox**> Click on **Shared Folder**> Shared Folder Dialog Box Will appear.

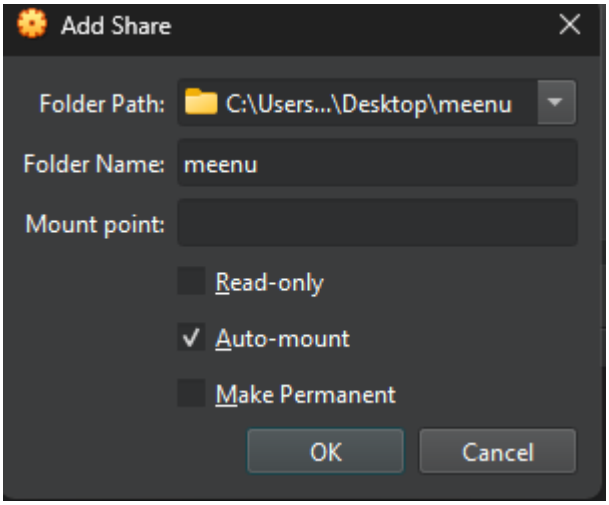


The **dialog box** will appear.



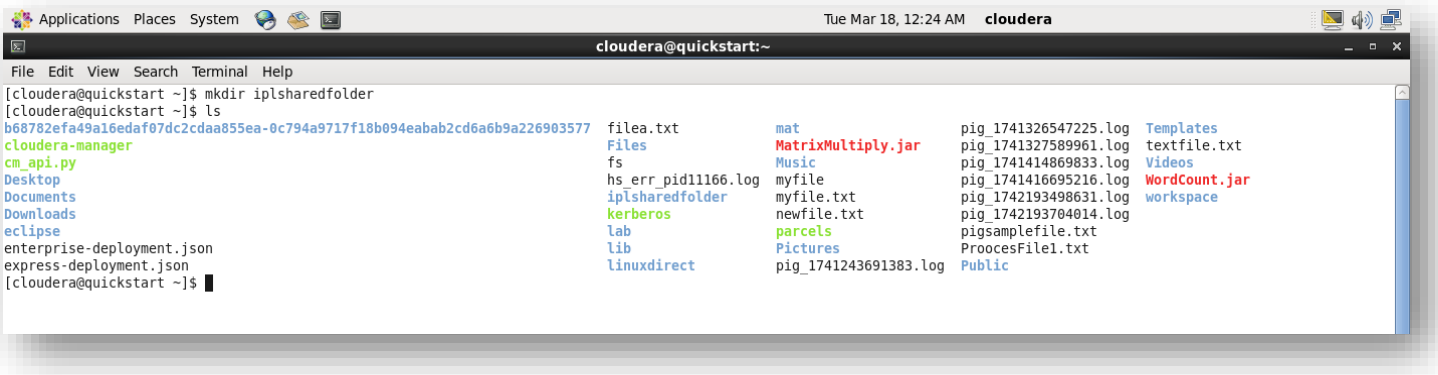
- Click On  Icon.

- Select the Folder > Check the **Auto Mount Option**> Click OK.
- Give the path of your folder.



STEP 3

- Start with your **VM**.
- Open the terminal &Create a Directory in Linux System.



STEP 4

- Change to **Super User** for mounting access by **su** command and enter **Password cloudera.**

```
[cloudera@quickstart ~]$ su
Password:
```

- Type The Following Command.

Syntax Of Command

mount -tvboxsf<Folder Name in Windows>< Folder Name In Linux>

```
[root@quickstart cloudera]# mount -t vboxsf meenu /home/cloudera/iplsharedfolder
```

- Now you are done with the procedures. As in host operating system your folder name **meenu** with stored csv files matches and deliveries is now shared in your virtual operating system (Linux).
- Open the Mounted folder in Linux named with **iplsharedfolder** and check the files stored in it.

```
[root@quickstart cloudera]# ls /home/cloudera/iplsharedfolder
deliveries.csv  matches.csv
```

STEP 5

- Sharing the mounted folder from (linux) to **Hadoop.**
- Using **-put** or **copyFromLocal** command.

```
root@quickstart cloudera]# hdfs dfs -mkdir ipl
root@quickstart cloudera]# hdfs dfs -put /home/cloudera/iplsharedfolder /user/cloudera/ipl
root@quickstart cloudera]# hdfs dfs -ls /user/cloudera/ipl
ound 2 items
rw-r--r--   1 root cloudera   27019953 2025-03-18 02:34 /user/cloudera/ipl/deliveries.csv
rw-r--r--   1 root cloudera     225266 2025-03-18 02:34 /user/cloudera/ipl/matches.csv
root@quickstart cloudera]#
```

STEP 6

- Staring with PIG.
- Enter pig command.

```
[cloudera@quickstart ~]$ pig
```

```
2025-03-27 22:37:36,183 [main] INFO  org.apache.hadoop.conf.Configuration.deprec
ation - fs.default.name is deprecated. Instead, use fs.defaultFS
2025-03-27 22:37:36,183 [main] INFO  org.apache.hadoop.conf.Configuration.deprec
ation - mapred.job.tracker is deprecated. Instead, use mapreduce.jobtracker.addr
ess
2025-03-27 22:37:36,196 [main] INFO  org.apache.hadoop.conf.Configuration.deprec
ation - fs.default.name is deprecated. Instead, use fs.defaultFS
2025-03-27 22:37:36,196 [main] INFO  org.apache.hadoop.conf.Configuration.deprec
ation - mapred.job.tracker is deprecated. Instead, use mapreduce.jobtracker.addr
ess
grunt>
```

MAP-REDUCE WORD COUNT PROGRAM IN CLOUDERA VM

PRE-REQUISITESTOFOLLOW:

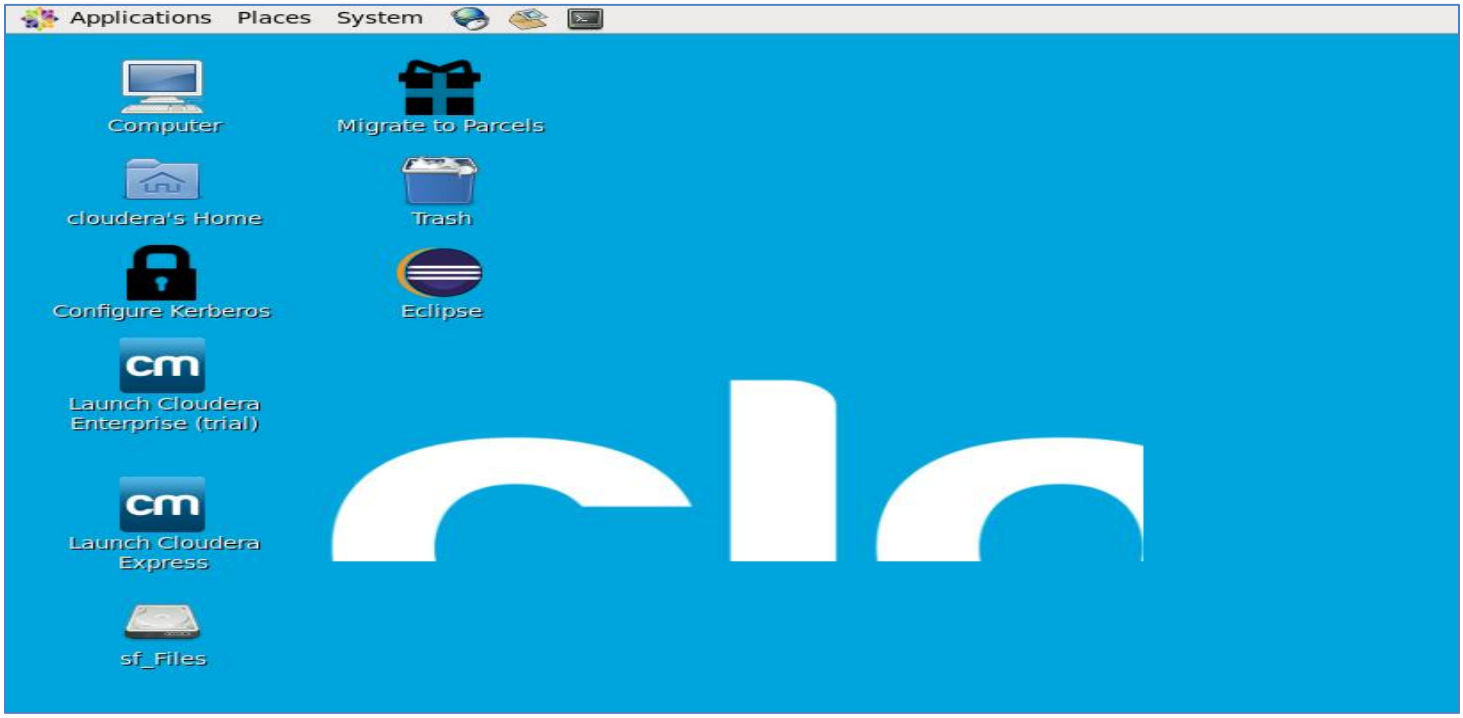
- Hadoop installation or cloudera vm
- Java IDE

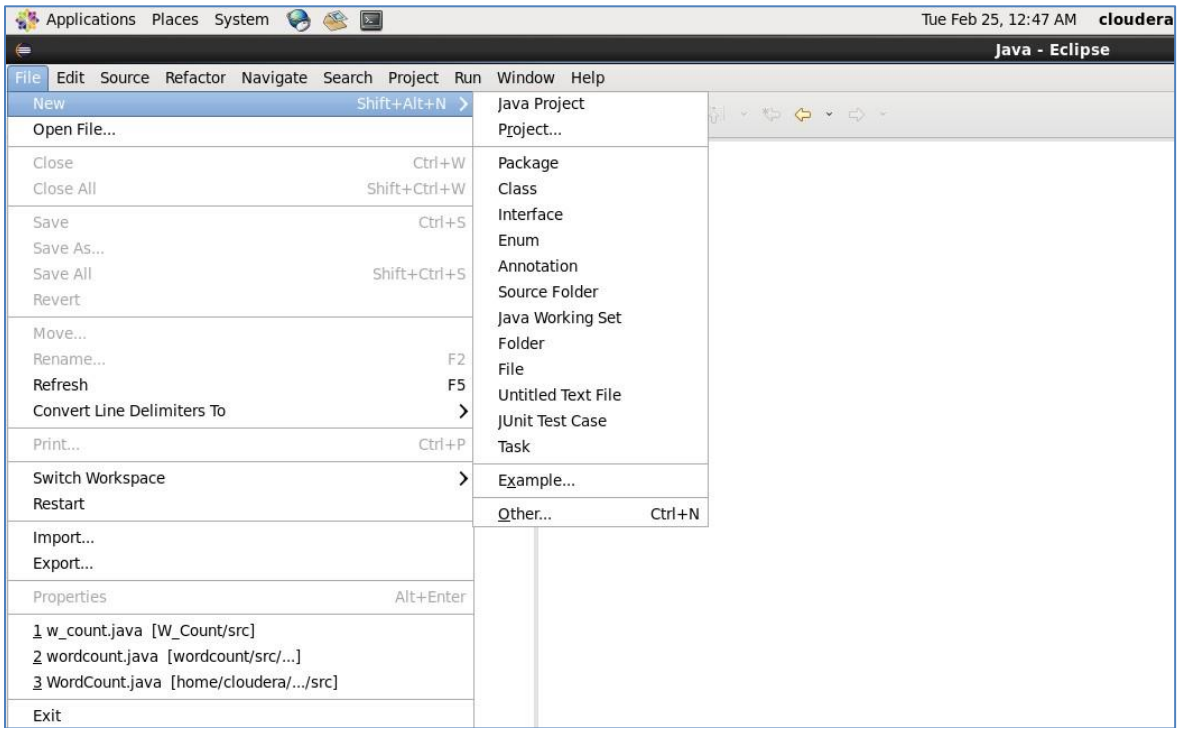
STEPS TO EXECUTE WORD COUNT IN HADOOP ENVIRONMENT.

Step I

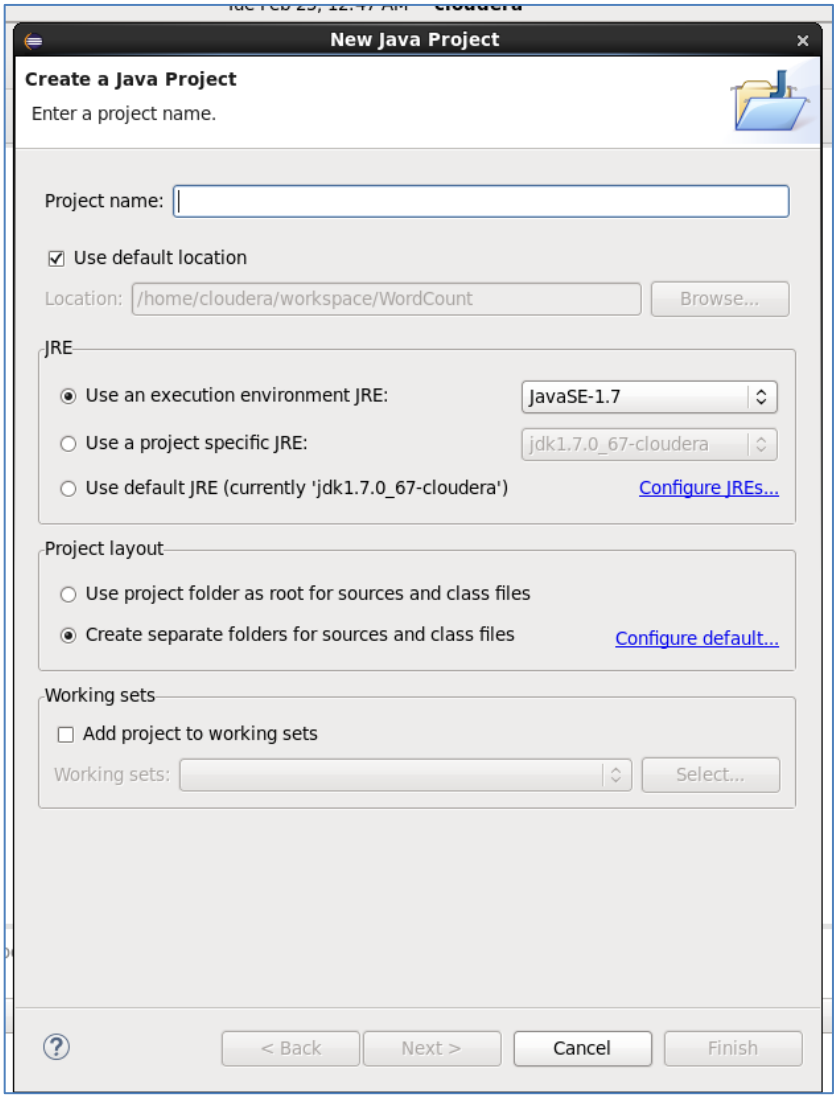
➢ Open VirtualBox, StartClouderaVM

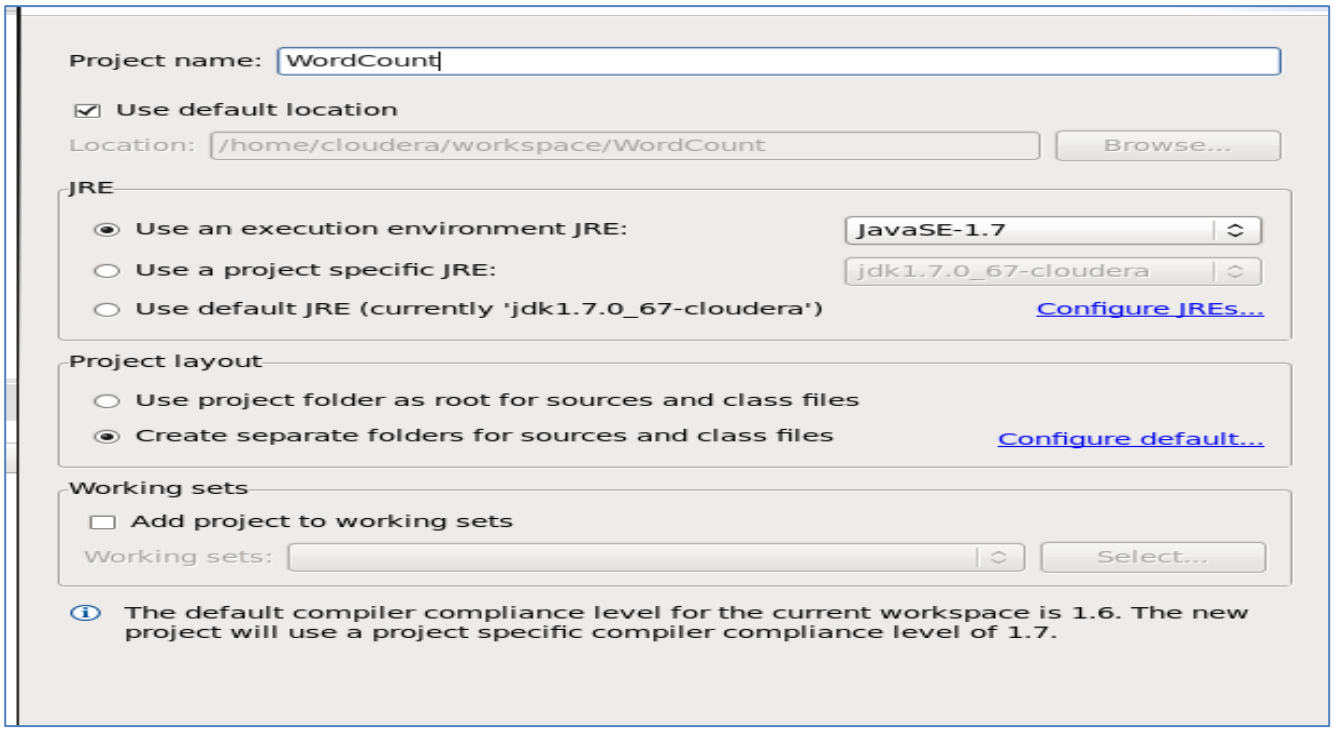
Launch Eclipse>Select File>New>Java Project >Provide Javaproject name.





- Give name to your Project.

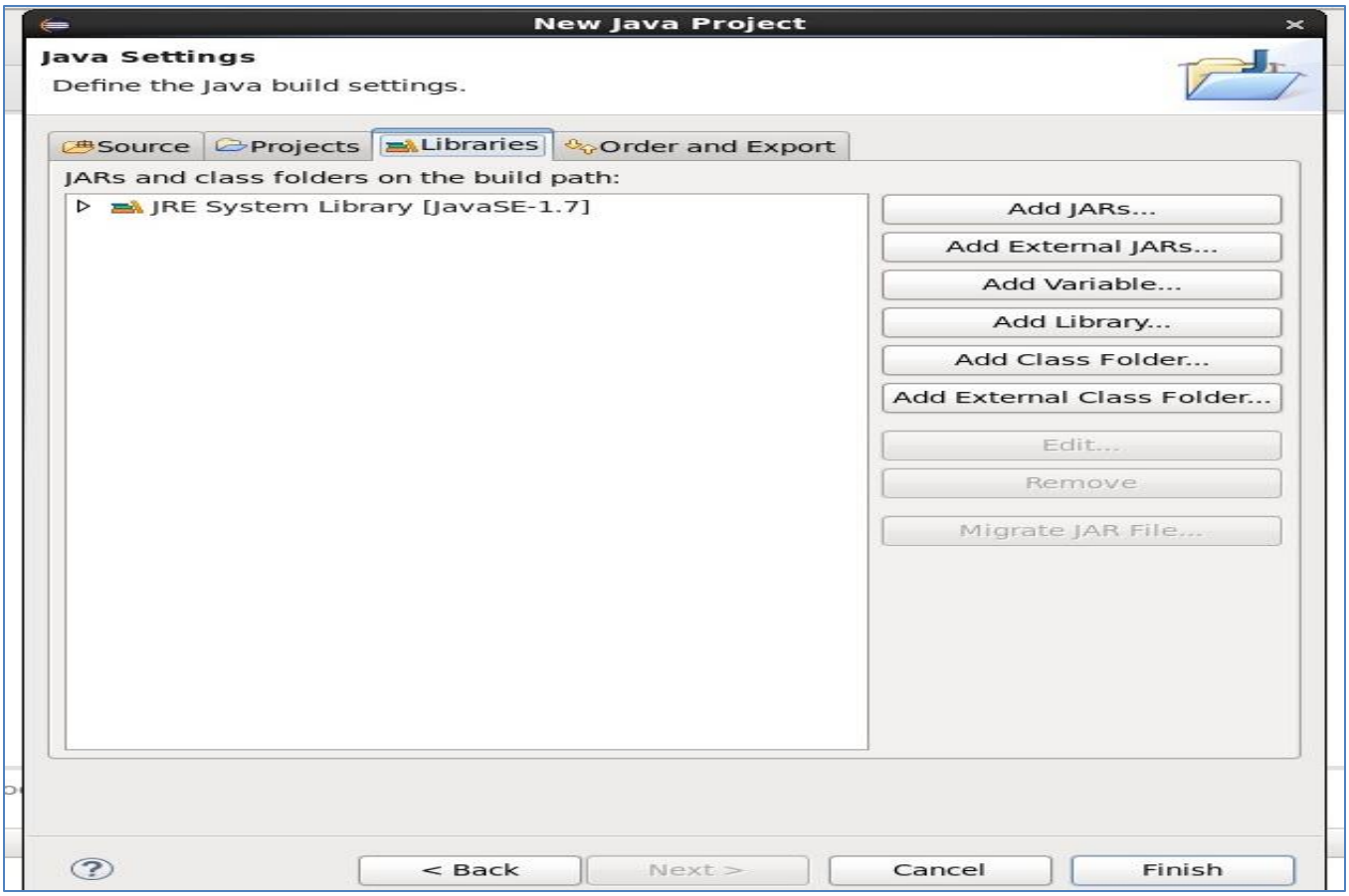




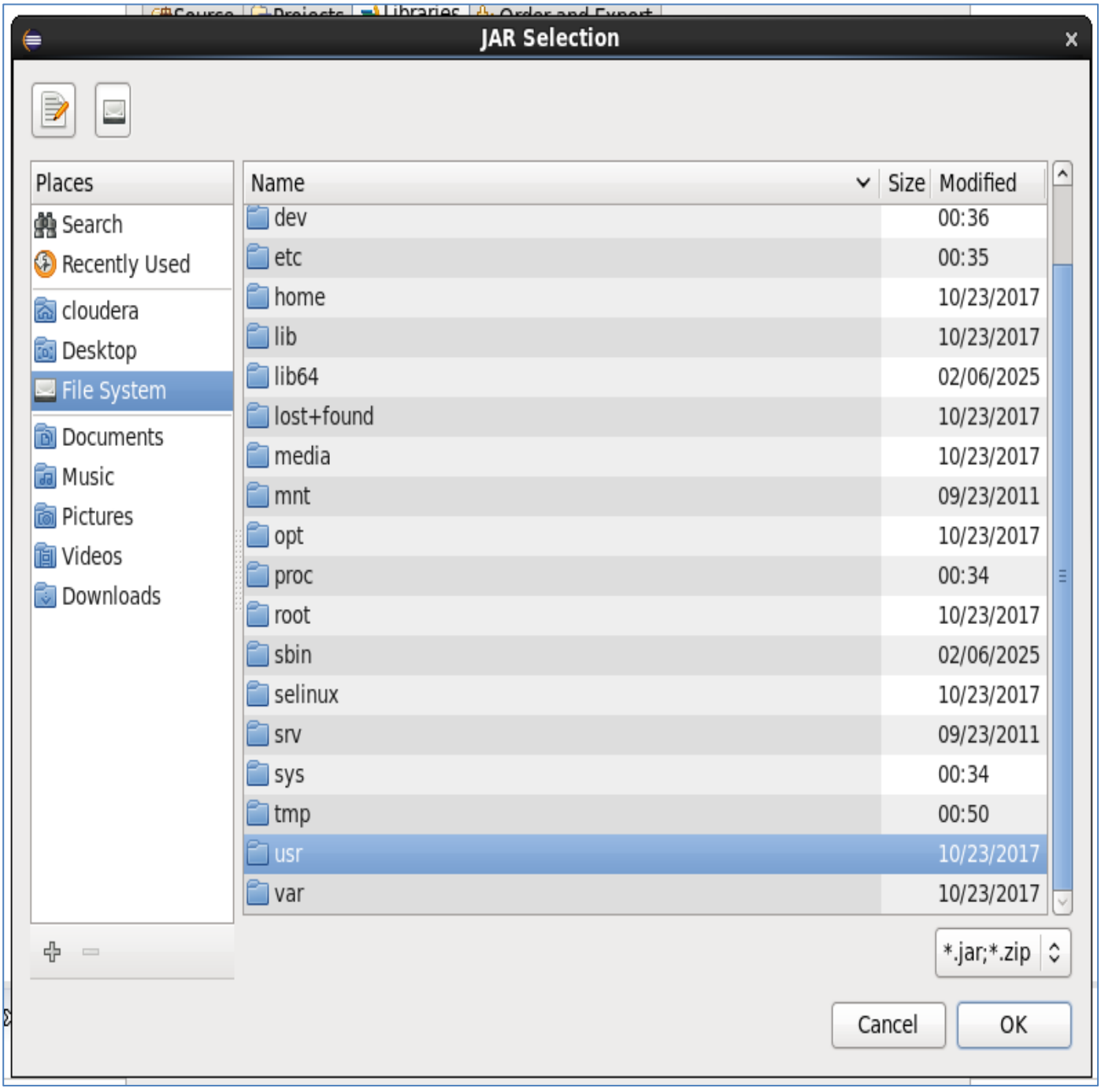
- Click on Next

Step II

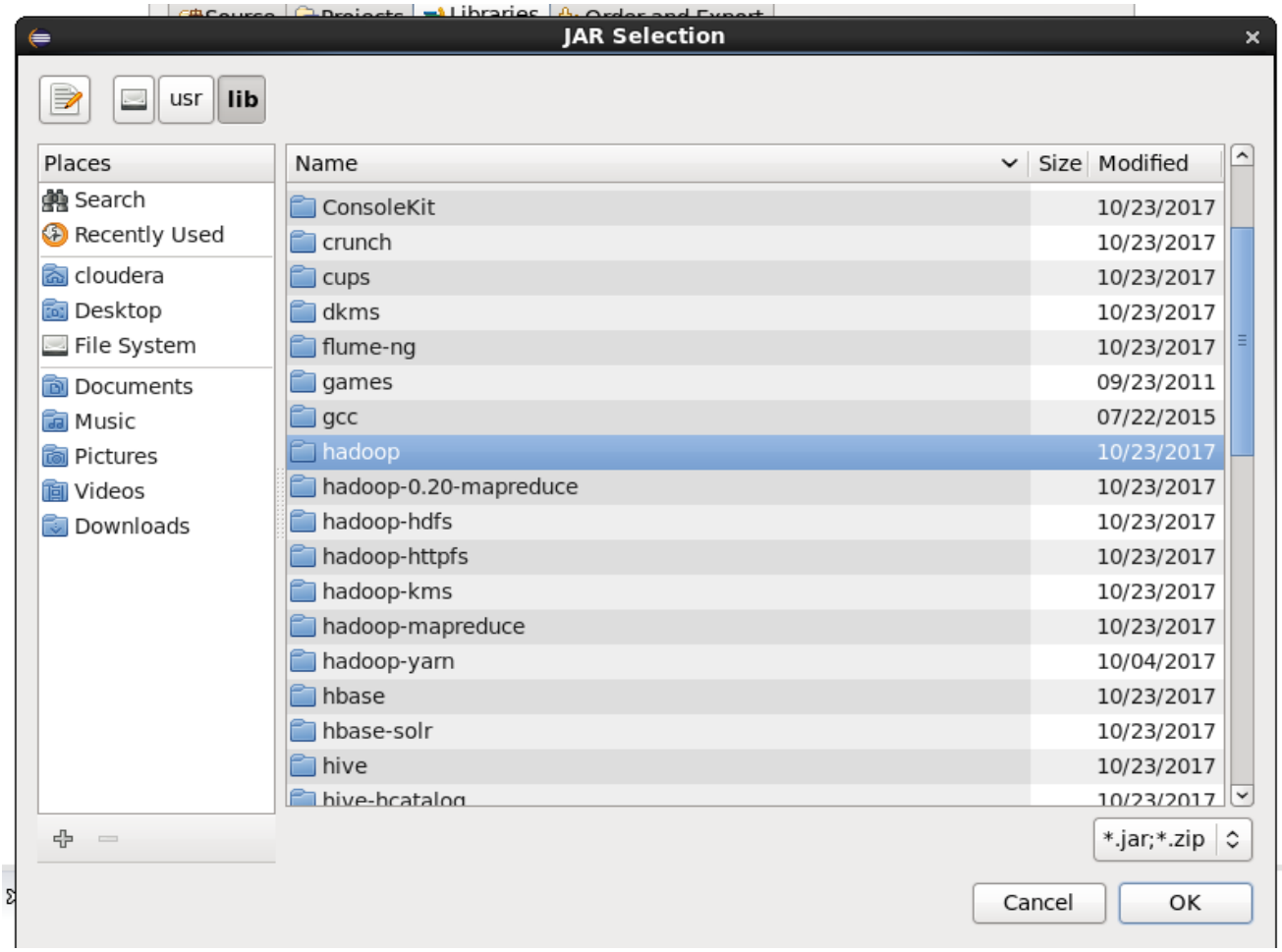
- ADDHADOOPLIBRARIES(jarfiles)

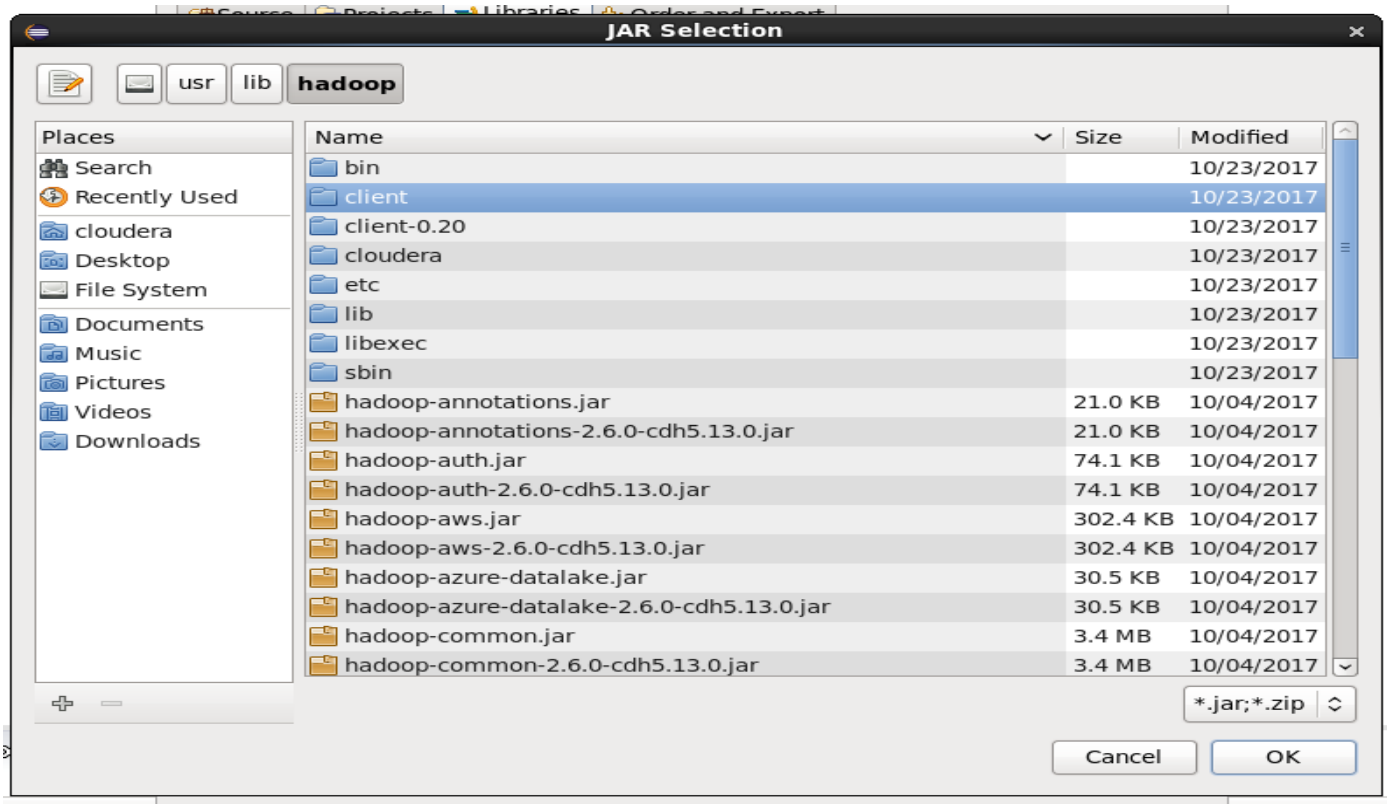


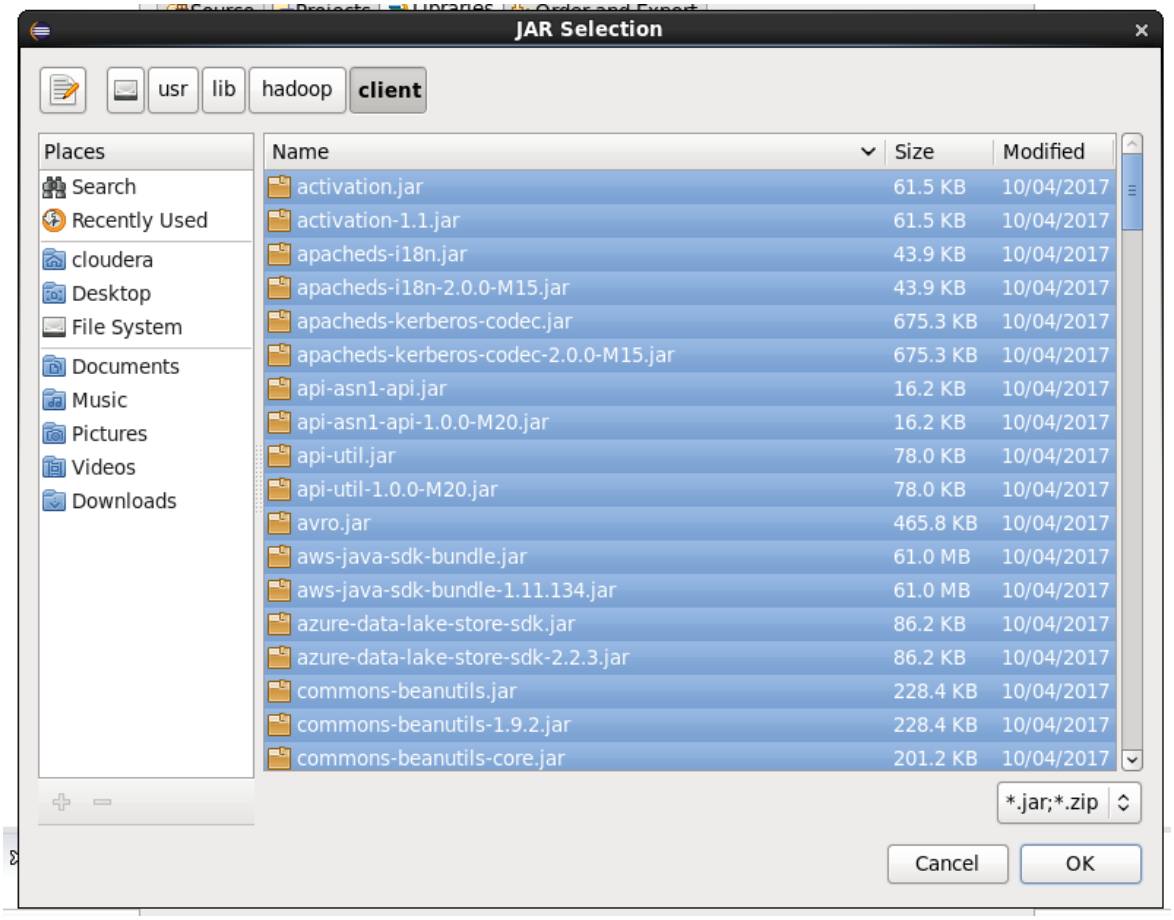
- Go to libraries
- In above figure, you can see the **ADD External JARs** option on the right hand side. Click on it and add hadoop jar files, which consists of **yarn, hdfs , mapreduce** all required jar files.

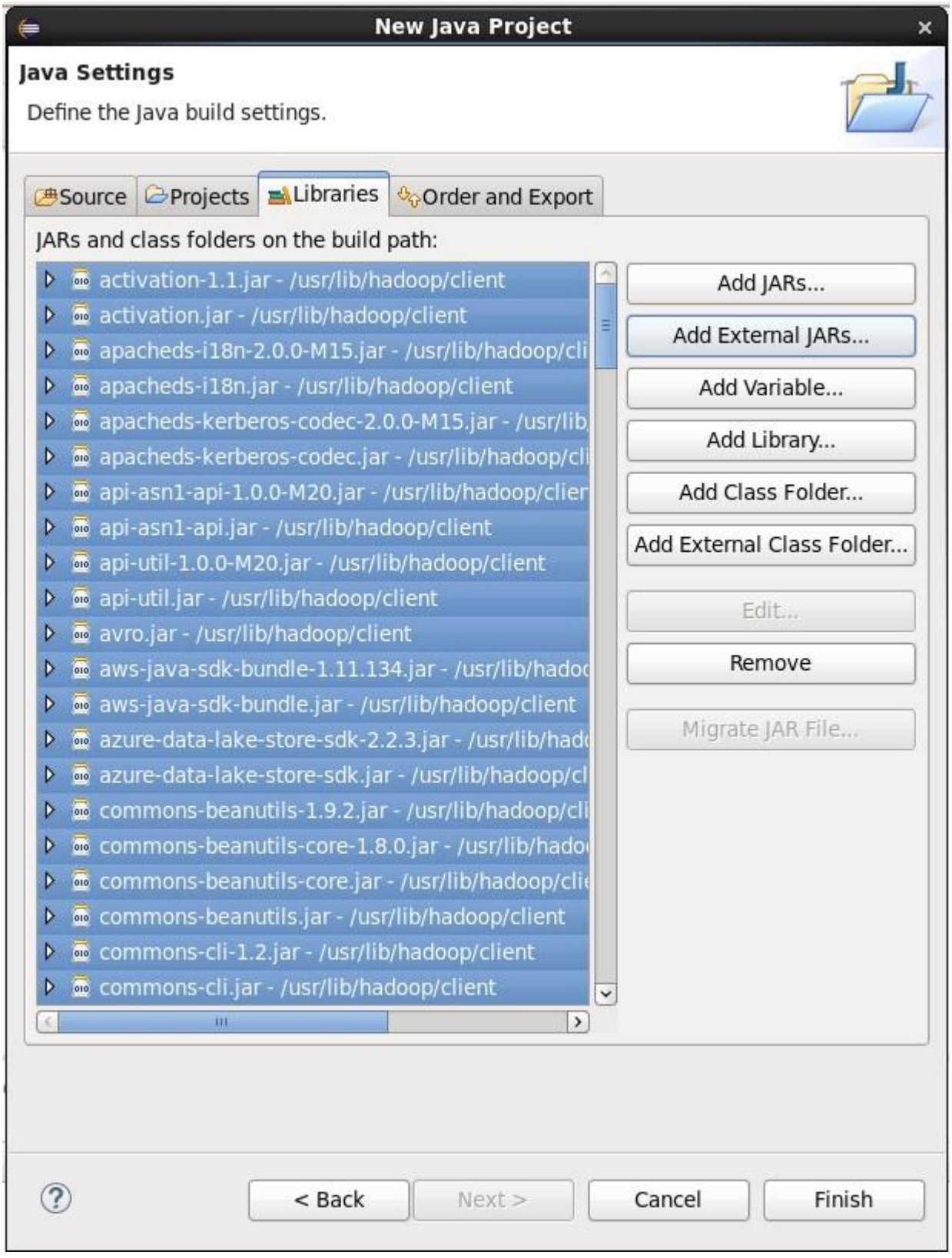


- Click on File System >**usr** >**lib**>**Hadoop**>selectHadoopjarfiles>OK.
- File System >**client**> OK > FINISH.




JAR Selection
usr
lib
hadoop
Places
Search
Recently Used
cloudera
Desktop
File System
Documents
Music
Pictures
Videos
Downloads
Name
Size
Modified
bin 10/23/2017
client 10/23/2017
client-0.20 10/23/2017
cloudera 10/23/2017
etc 10/23/2017
lib 10/23/2017
libexec 10/23/2017
sbin 10/23/2017
hadoop-annotations.jar 21.0 KB 10/04/2017
hadoop-annotations-2.6.0-cdh5.13.0.jar 21.0 KB 10/04/2017
hadoop-auth.jar 74.1 KB 10/04/2017
hadoop-auth-2.6.0-cdh5.13.0.jar 74.1 KB 10/04/2017
hadoop-aws.jar 302.4 KB 10/04/2017
hadoop-aws-2.6.0-cdh5.13.0.jar 302.4 KB 10/04/2017
hadoop-azure-datalake.jar 30.5 KB 10/04/2017
hadoop-azure-datalake-2.6.0-cdh5.13.0.jar 30.5 KB 10/04/2017
hadoop-common.jar 3.4 MB 10/04/2017
hadoop-common-2.6.0-cdh5.13.0.jar 3.4 MB 10/04/2017
.jar;.zip
Cancel
OK

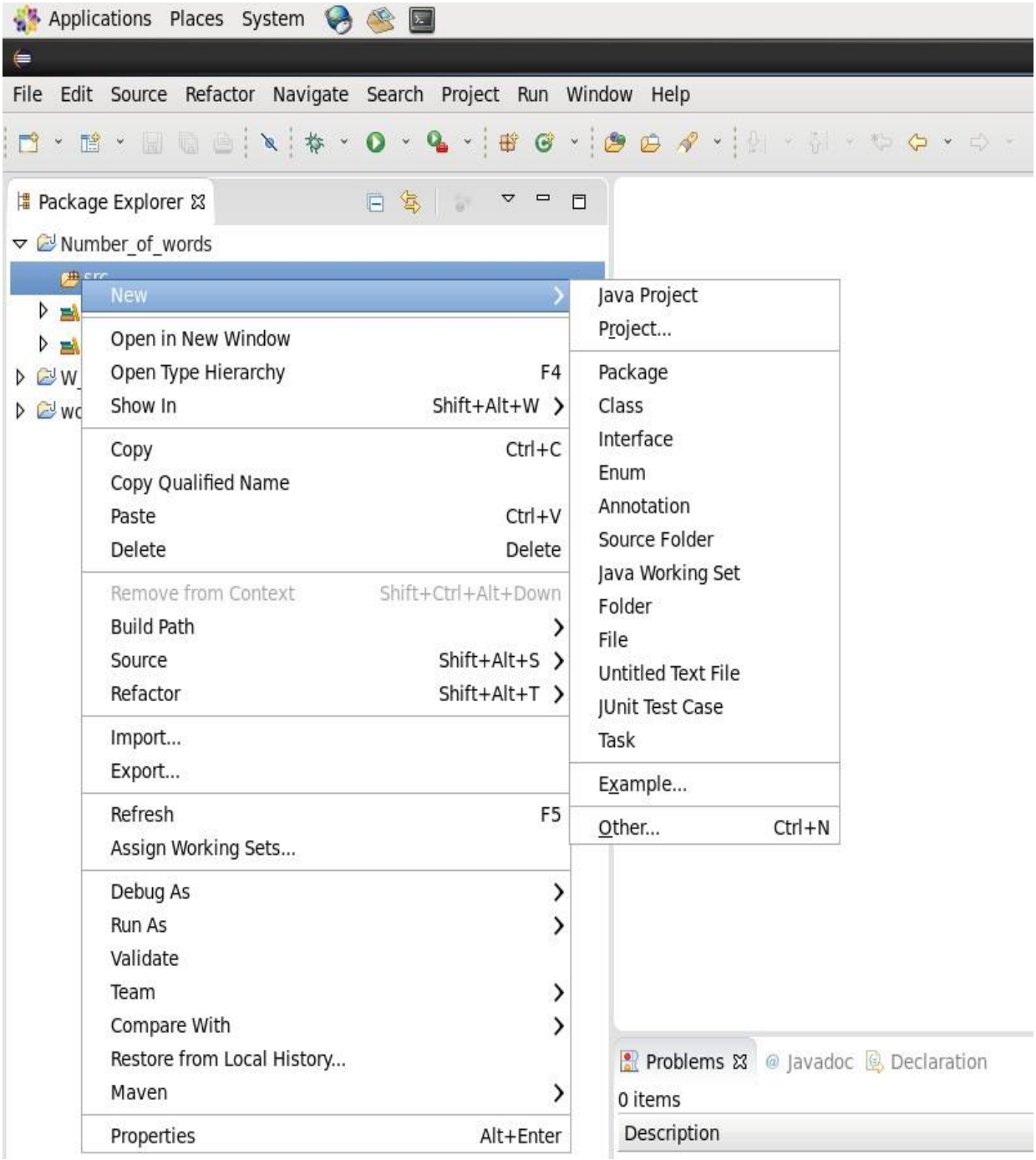

JAR Selection
usr
lib
hadoop
client
Places
Search
Recently Used
cloudera
Desktop
File System
Documents
Music
Pictures
Videos
Downloads
Name
Size
Modified
activation.jar 61.5 KB 10/04/2017
activation-1.1.jar 61.5 KB 10/04/2017
apacheds-i18n.jar 43.9 KB 10/04/2017
apacheds-i18n-2.0.0-M15.jar 43.9 KB 10/04/2017
apacheds-kerberos-codec.jar 675.3 KB 10/04/2017
apacheds-kerberos-codec-2.0.0-M15.jar 675.3 KB 10/04/2017
api-asn1-api.jar 16.2 KB 10/04/2017
api-asn1-api-1.0.0-M20.jar 16.2 KB 10/04/2017
api-util.jar 78.0 KB 10/04/2017
api-util-1.0.0-M20.jar 78.0 KB 10/04/2017
avro.jar 465.8 KB 10/04/2017
aws-java-sdk-bundle.jar 61.0 MB 10/04/2017
aws-java-sdk-bundle-1.11.134.jar 61.0 MB 10/04/2017
azure-data-lake-store-sdk.jar 86.2 KB 10/04/2017
azure-data-lake-store-sdk-2.2.3.jar 86.2 KB 10/04/2017
commons-beanutils.jar 228.4 KB 10/04/2017
commons-beanutils-1.9.2.jar 228.4 KB 10/04/2017
commons-beanutils-core.jar 201.2 KB 10/04/2017
.jar;.zip
Cancel
OK

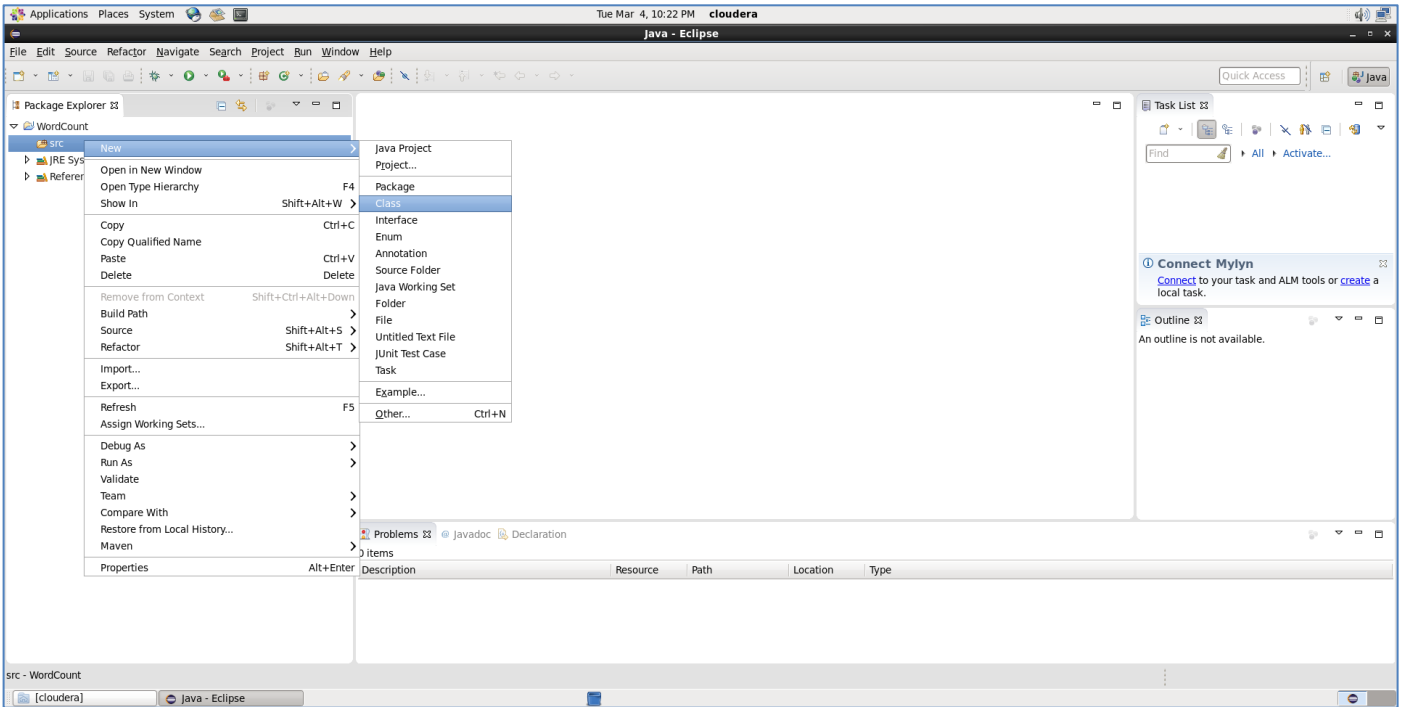

New Java Project
Java Settings
Define the Java build settings.
Source
Projects
Libraries
Order and Export
JARs and class folders on the build path:
activation-1.1.jar - /usr/lib/hadoop/client
activation.jar - /usr/lib/hadoop/client
apacheds-i18n-2.0.0-M15.jar - /usr/lib/hadoop/cli
apacheds-i18n.jar - /usr/lib/hadoop/client
apacheds-kerberos-codec-2.0.0-M15.jar - /usr/lib
apacheds-kerberos-codec.jar - /usr/lib/hadoop/cli
api-asn1-api-1.0.0-M20.jar - /usr/lib/hadoop/clier
api-asn1-api.jar - /usr/lib/hadoop/client
api-util-1.0.0-M20.jar - /usr/lib/hadoop/client
api-util.jar - /usr/lib/hadoop/client
avro.jar - /usr/lib/hadoop/client
aws-java-sdk-bundle-1.11.134.jar - /usr/lib/hadoc
aws-java-sdk-bundle.jar - /usr/lib/hadoop/client
azure-data-lake-store-sdk-2.2.3.jar - /usr/lib/had
azure-data-lake-store-sdk.jar - /usr/lib/hadoop/cl
commons-beanutils-1.9.2.jar - /usr/lib/hadoop/cli
commons-beanutils-core-1.8.0.jar - /usr/lib/hado
commons-beanutils-core.jar - /usr/lib/hadoop/cli
commons-beanutils.jar - /usr/lib/hadoop/client
commons-cli-1.2.jar - /usr/lib/hadoop/client
commons-cli.jar - /usr/lib/hadoop/client
Add JARs...
Add External JARs...
Add Variable...
Add Library...
Add Class Folder...
Add External Class Folder...
Edit...
Remove
Migrate JAR File...
< Back
Next >
Cancel
Finish

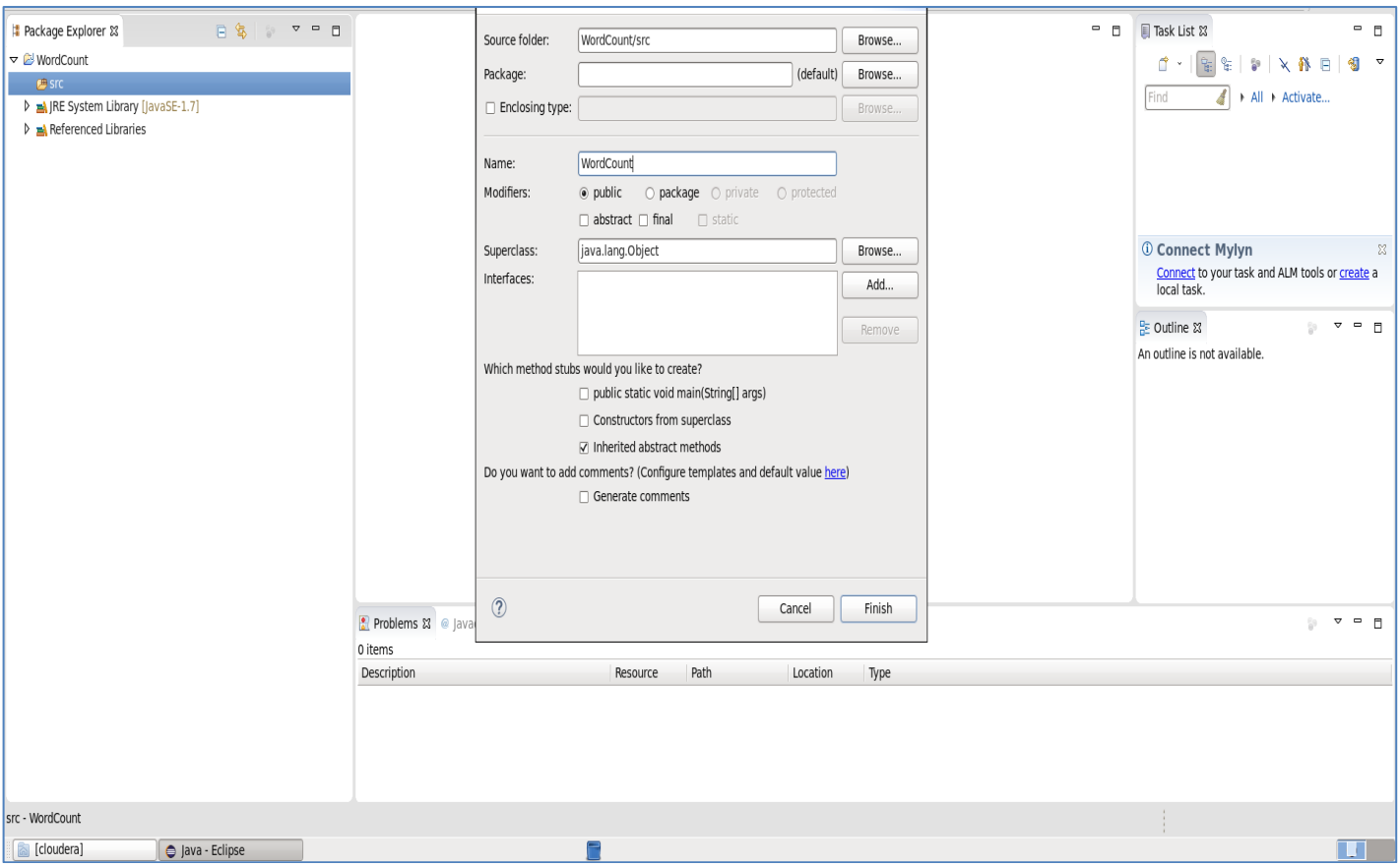
Step III:
CREATING JAVA FILE

- Create a new class that performs word count operation.

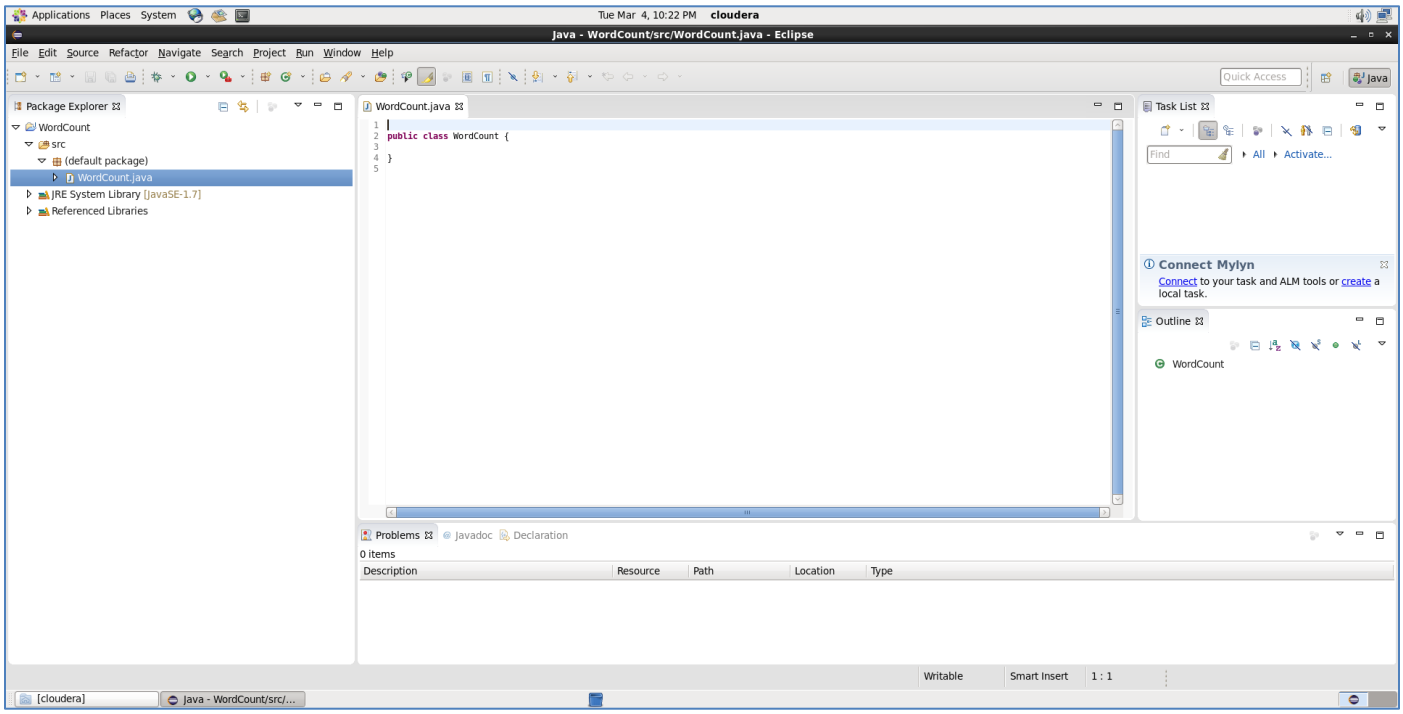
  ProjectName > New > Class > Name it Word Count



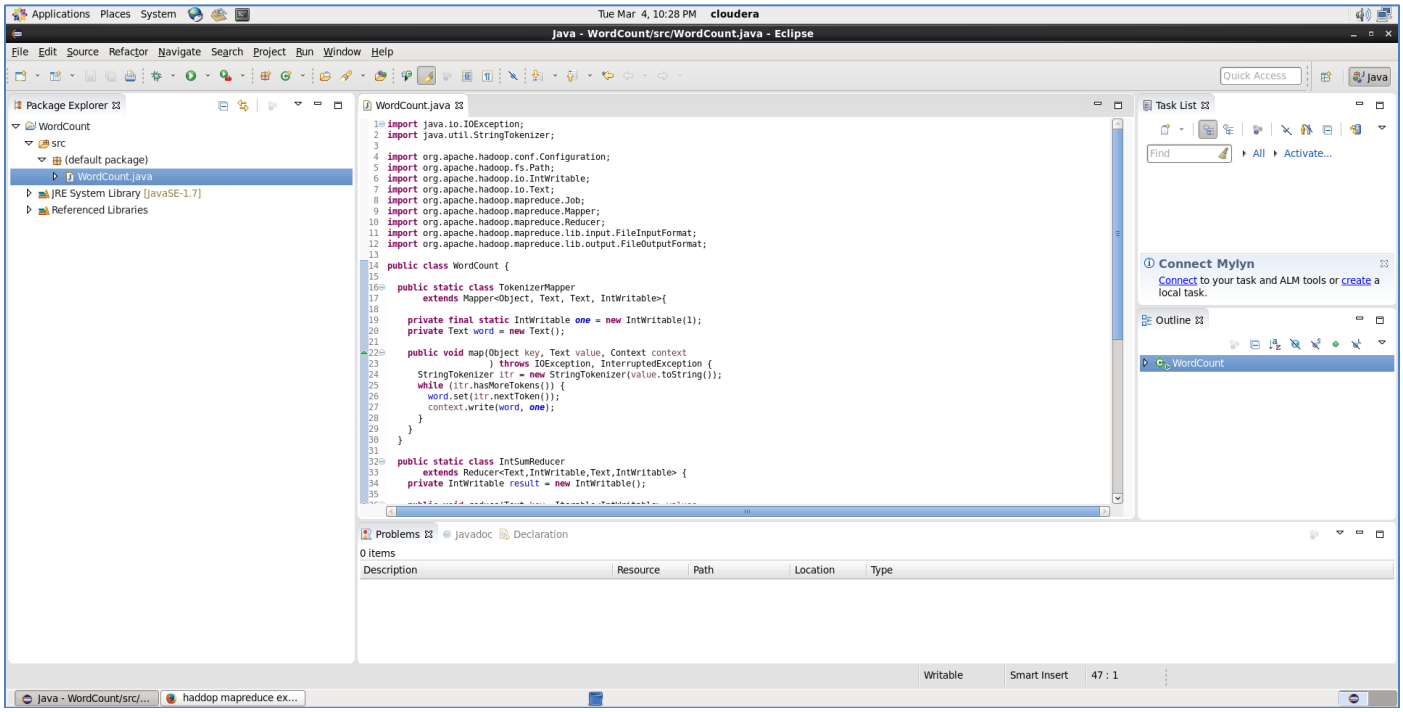

Applications Places System
Tue Mar 4, 10:22 PM cloudera
Java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Java
Package Explorer
WordCount
src
New
Open in New Window
Open Type Hierarchy F4
Show In Shift+Alt+W
Copy Ctrl+C
Copy Qualified Name
Paste Ctrl+V
Delete Delete
Remove from Context Shift+Ctrl+Alt+Down
Build Path
Source Shift+Alt+S
Refactor Shift+Alt+T
Import...
Export...
Refresh F5
Assign Working Sets...
Debug As
Run As
Validate
Team
Compare With
Restore from Local History...
Maven
Properties Alt+Enter
Java Project
Project...
Package
Class
Interface
Enum
Annotation
Source Folder
Java Working Set
Folder
File
Untitled Text File
JUnit Test Case
Task
Example...
Other... Ctrl+N
Task List
Find
All
Activate...
Connect Mylyn
Connect to your task and ALM tools or create a local task.
Outline
An outline is not available.
Problems
Javadoc
Declaration
0 items
Description
Resource
Path
Location
Type
src - WordCount
[cloudera]
Java - Eclipse


Package Explorer
WordCount
src
JRE System Library [JavaSE-1.7]
Referenced Libraries
Source folder: WordCount/src
Browse...
Package:
(default)
Enclosing type:
Name: WordCount
Modifiers: public package private protected
abstract final static
Superclass: java.lang.Object
Interfaces:
Add...
Remove
Which method stubs would you like to create?
public static void main(String[] args)
Constructors from superclass
Inherited abstract methods
Do you want to add comments? (Configure templates and default value here)
Generate comments
Cancel
Finish
Task List
Connect Mylyn
Outline
An outline is not available.
Problems
0 items
Description
Resource
Path
Location
Type
src - WordCount
[cloudera]
Java - Eclipse

- Copy Mapreduce word count program from below link i.e .available on apache org.



- Copy the code from the link and paste.

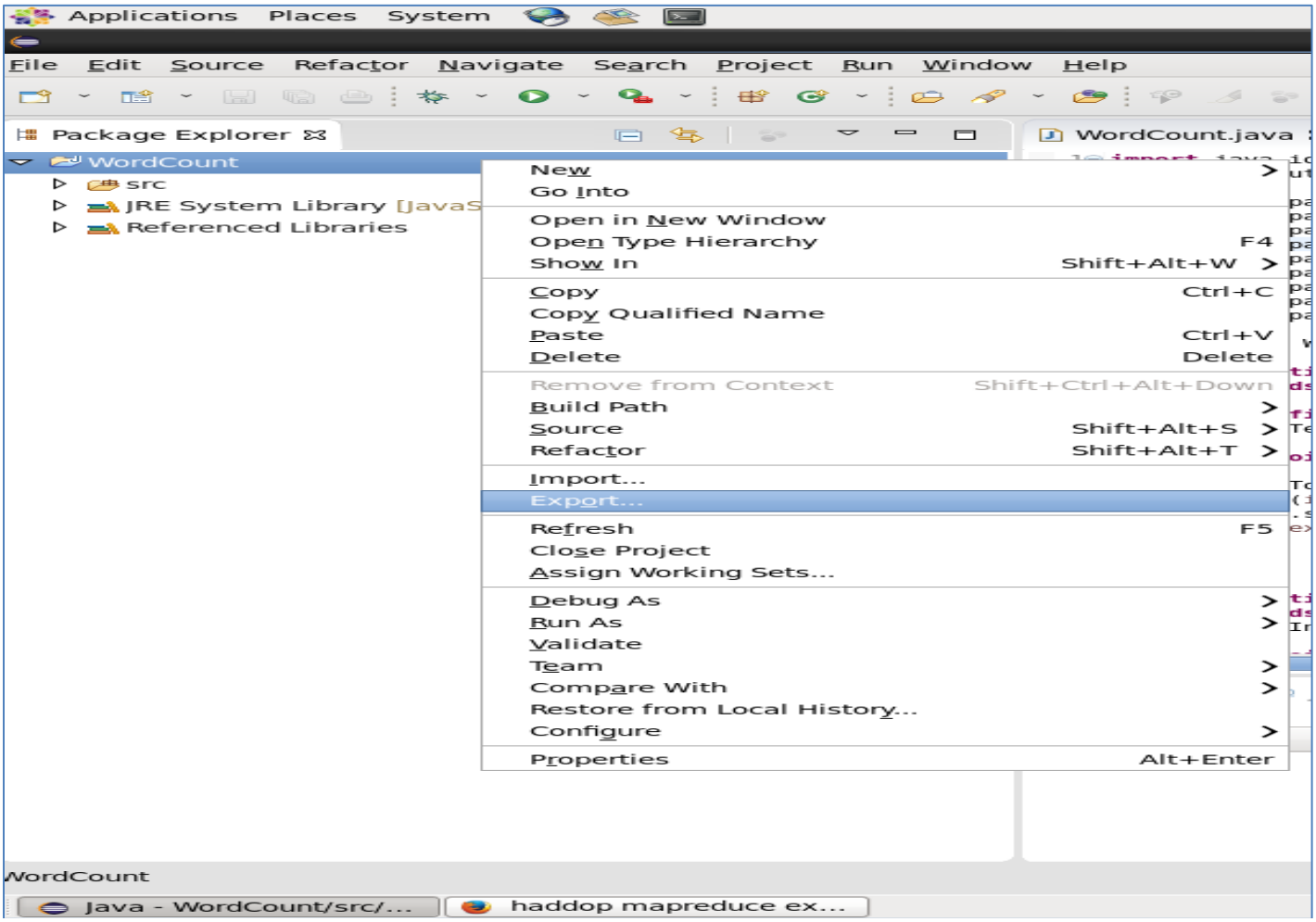


- This program consists of three classes:
  - **Driver class** (this is entry point which has main function)
  - **Tokenizer Mapper** class which extends the public class **Mapper** and implements the **Map** function.
  - **IntSumReducer** class which extends public class **Reducer** and implements the **Reduce** function.

Step IV:

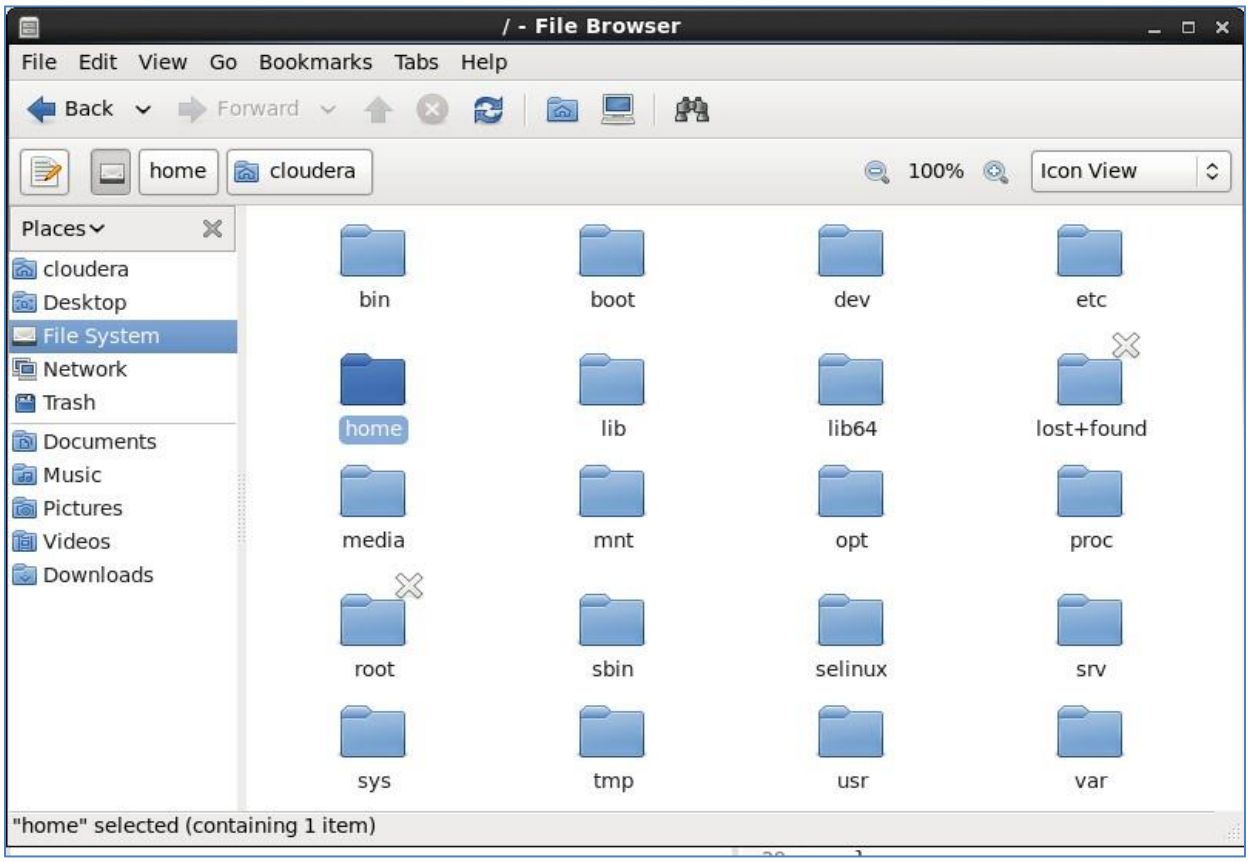
- Make a jar file

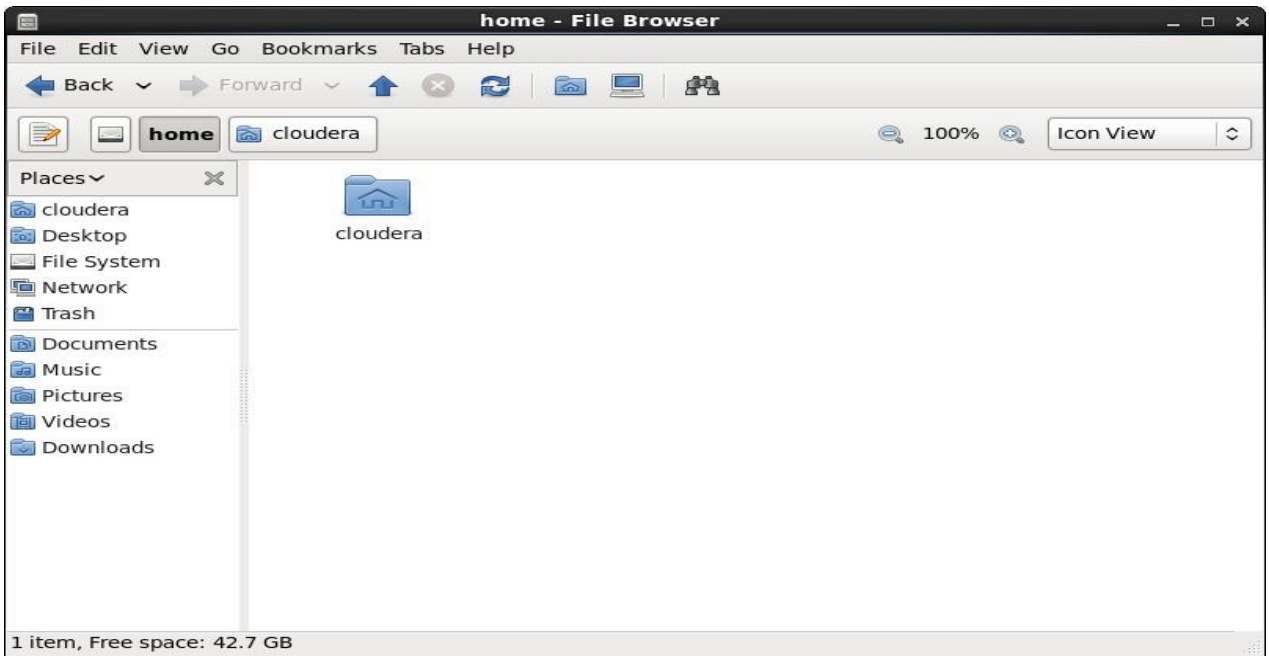
RightclickonProject>Export>**JARFile>**Next>ProvidenameforJARFile(WordCount.jar)>ClickNext>Finish



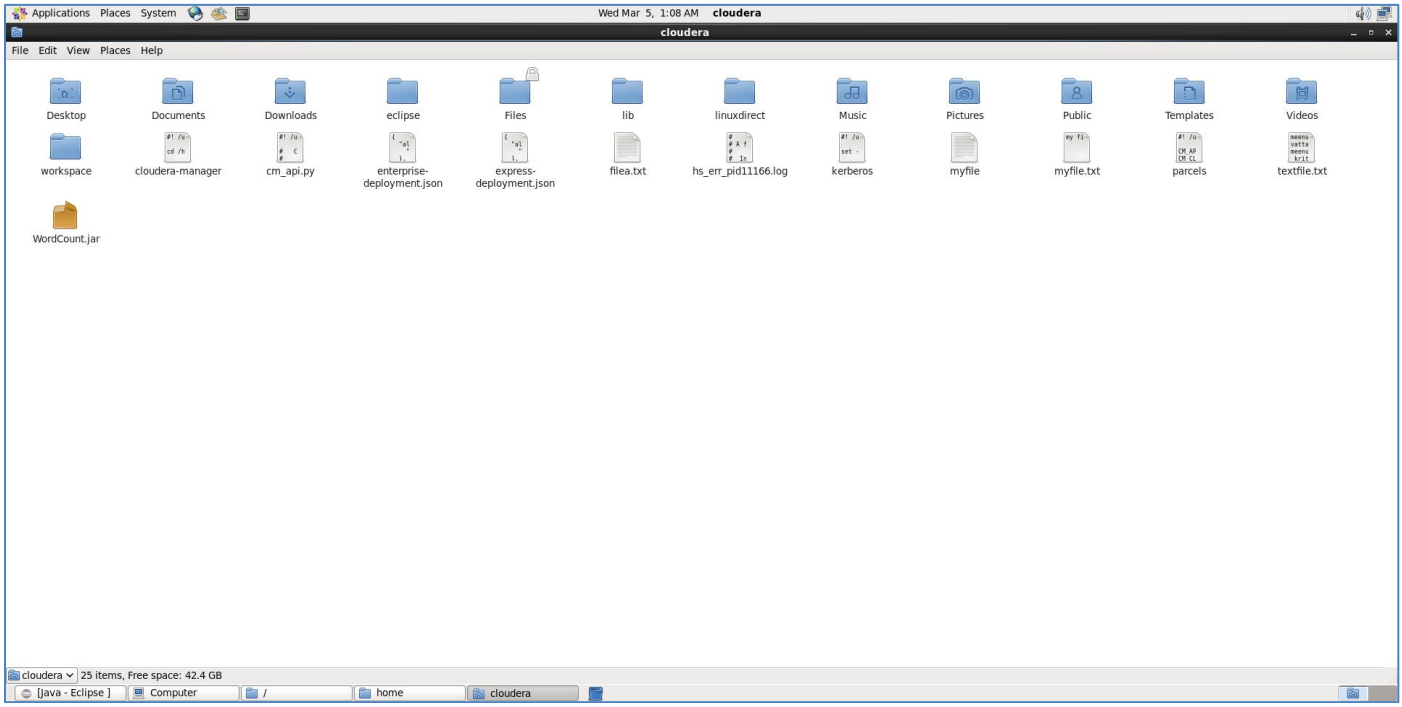
- To check whether jar file is exported or not:

  Go to Applications > File System > Home >cloudera





- If wordcount.jar appears then it is done.



Step V:

- Now open terminal.
- Create a file in local system windows or linux.

  Cat > filename

cat> /home/cloudera/ProocesFile.txt

Add content in the file.

Check it using cat filename

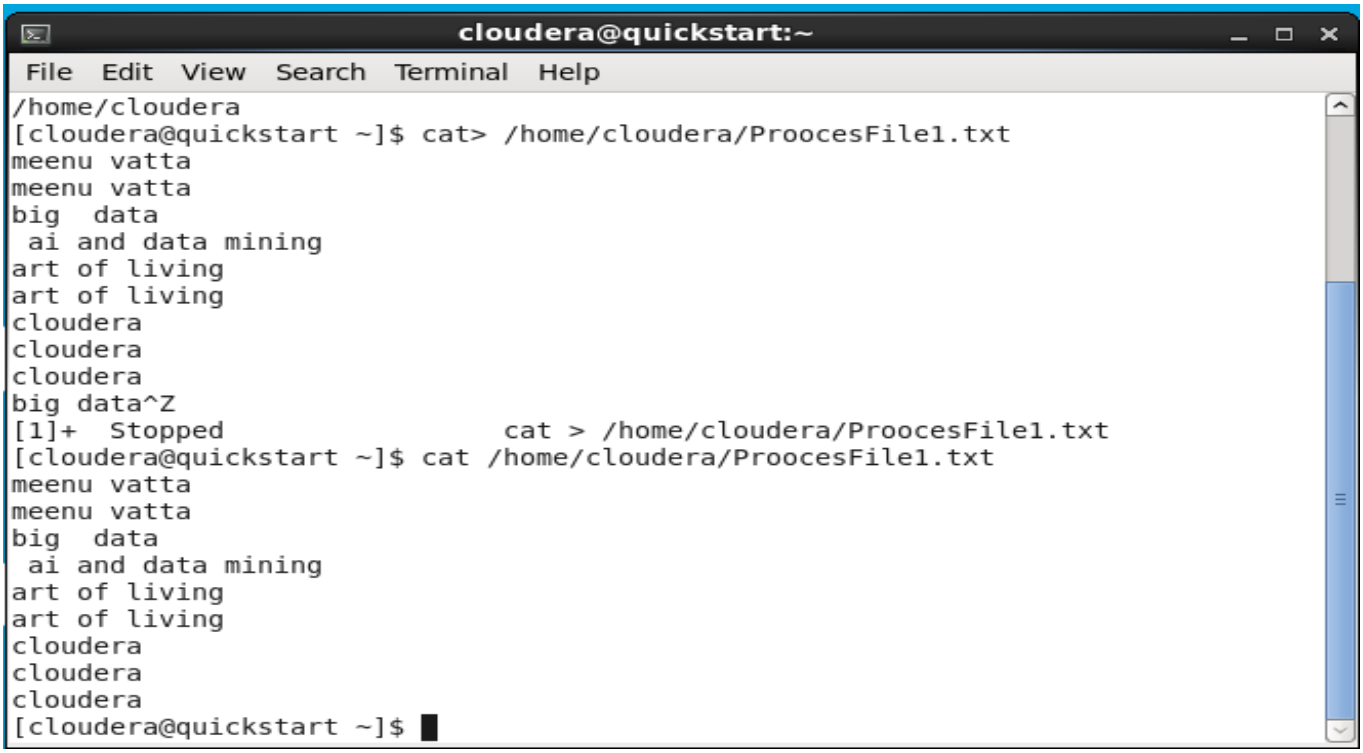
Now create a directory on **hdfs** using**-mkdir** command.

Step VI:

- Copy file into hdfs

To bring file in hdfs, copy that file in hdfs using **-putor-copyFromLocal**command.

hdfs dfs-put inputfile/inputdir


cloudera@quickstart:~
File Edit View Search Terminal Help
/home/cloudera
[cloudera@quickstart ~]$ cat> /home/cloudera/ProocesFile1.txt
meenu vatta
meenu vatta
big data
 ai and data mining
art of living
art of living
cloudera
cloudera
cloudera
big data^Z
[1]+ Stopped   cat > /home/cloudera/ProocesFile1.txt
[cloudera@quickstart ~]$ cat /home/cloudera/ProocesFile1.txt
meenu vatta
meenu vatta
big data
 ai and data mining
art of living
art of living
cloudera
cloudera
cloudera
[cloudera@quickstart ~]$


Applications Places System
Wed Mar 5, 1:27 AM cloudera
cloudera@quickstart:~
File Edit View Search Terminal Help
[cloudera@quickstart ~]$ hdfs dfs -ls
Found 8 items
-rw-r--r--   1 root   cloudera   77 2025-03-04 01:30 employee.txt
-rw-r--r--   1 cloudera cloudera   0 2025-01-16 22:20 file1.txt
-rw-r--r--   1 cloudera cloudera   14 2025-01-16 22:15 filea.txt
-rw-r--r--   1 cloudera cloudera   0 2025-01-16 21:43 hdfsfile.txt
-rw-r--r--   1 cloudera cloudera   0 2025-01-16 22:09 hdfsfilenew.txt
-rw-r--r--   1 cloudera cloudera   14 2025-01-16 21:42 myfile.txt
drwxr-xr-x   - cloudera cloudera   0 2025-03-04 01:35 pig
drwxr-xr-x   - cloudera cloudera   0 2025-02-11 01:31 sample
[cloudera@quickstart ~]$ hdfs dfs -ls /
Found 7 items
drwxrwxrwx   - hdfs   supergroup   0 2017-10-23 09:15 /benchmarks
drwxr-xr-x   - hbase   supergroup   0 2025-03-05 01:02 /hbase
drwxr-xr-x   - cloudera supergroup   0 2025-02-11 02:10 /output
drwxr-xr-x   - solr   solr   0 2017-10-23 09:18 /solr
drwxrwxrwt   - hdfs   supergroup   0 2025-03-04 02:06 /tmp
drwxr-xr-x   - hdfs   supergroup   0 2017-10-23 09:17 /user
drwxr-xr-x   - hdfs   supergroup   0 2017-10-23 09:17 /var
[cloudera@quickstart ~]$ hdfs dfs -mkdir /inputfolder1
[cloudera@quickstart ~]$ hdfs dfs -put /home/cloudera/ProocesFile1.txt /inputfol
der1/
[cloudera@quickstart ~]$ -cat /inputfolder1/ProocesFile1.txt
bash: -cat: command not found
[cloudera@quickstart ~]$ hdfs dfs -cat /inputfolder1/ProocesFile1.txt
meenu vatta
meenu vatta
big data
 ai and data mining
art of living
art of living
cloudera
cloudera
cloudera
[cloudera@quickstart ~]$ hadoop jar /home/cloudera/WordCount.jar WordCount /inpu
tfolder1/ProocesFile1.txt /out1
25/03/05 01:25:35 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0
:8032
25/03/05 01:25:35 WARN mapreduce.JobResourceUploader: Hadoop command-line option
 parsing not performed. Implement the Tool interface and execute your applicatio
n with ToolRunner to remedy this.
25/03/05 01:25:36 INFO input.FileInputFormat: Total input paths to process : 1
25/03/05 01:25:36 INFO mapreduce.JobSubmitter: number of splits:1
25/03/05 01:25:36 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_17
41165367876_0001
25/03/05 01:25:36 INFO impl.YarnClientImpl: Submitted application application_17
41165367876_0001
25/03/05 01:25:36 INFO mapreduce.Job: The url to track the job: http://quickstar
t.cloudera:8088/proxy/application_1741165367876_0001/
25/03/05 01:25:36 INFO mapreduce.Job: Running job: job_1741165367876_0001
[Java - Eclipse]
cloudera@quickstart:~

Step VII:

- Run jarfile:

Execute Hadoop mapreduce word count jarfile.

```
hadoopjar<jarfile_name>classname<inputtextfilepath><output_directorypath>
```

This will start the execution of Mapreduce job.

Wait for a while until file is being executed. After execution, the output will contain number of splits, number of map tasks, number of reducer tasks etc. But output directory not be existing already.

- **Map function reads input data,splits into words and emits key-value pairs.**

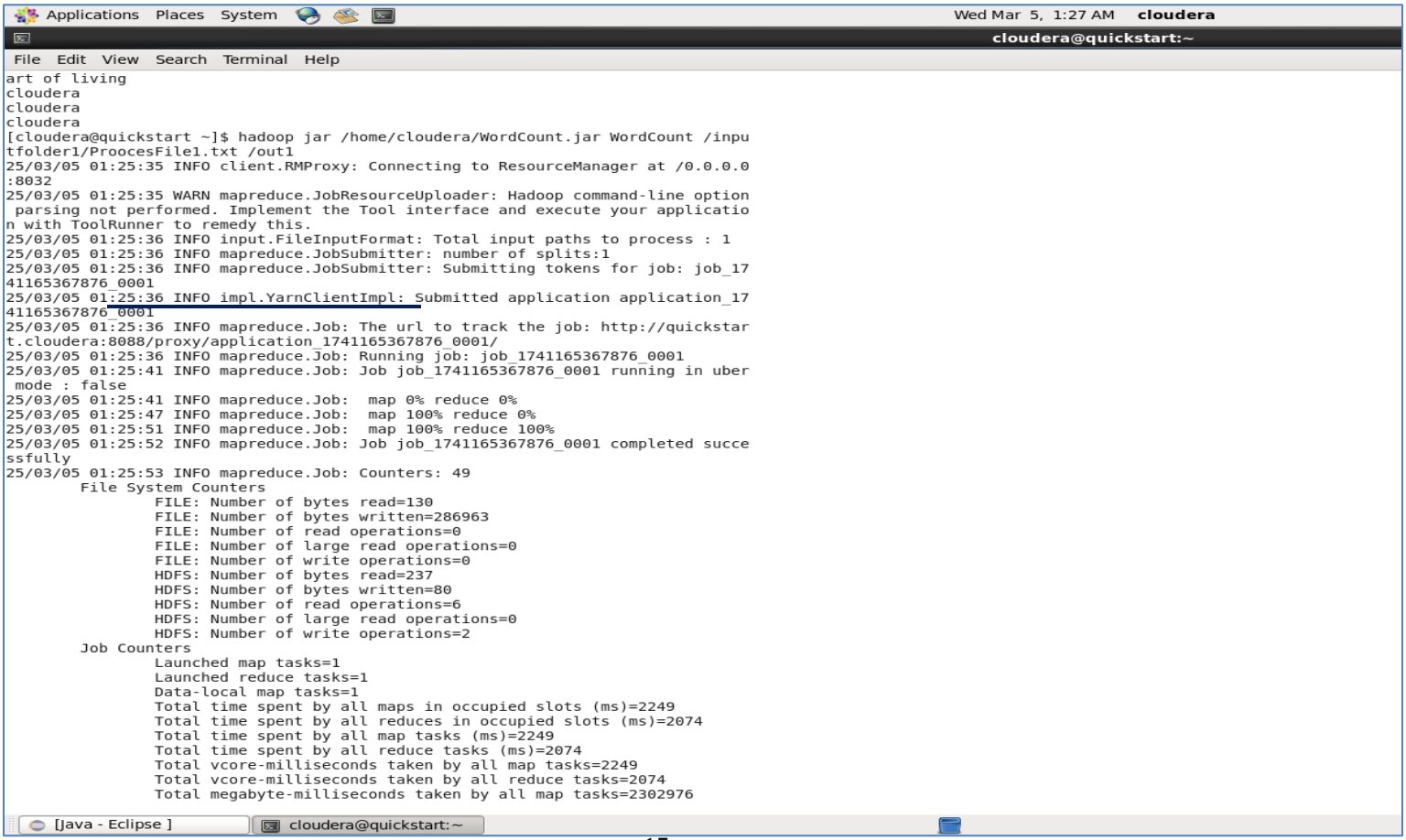
(is,1)

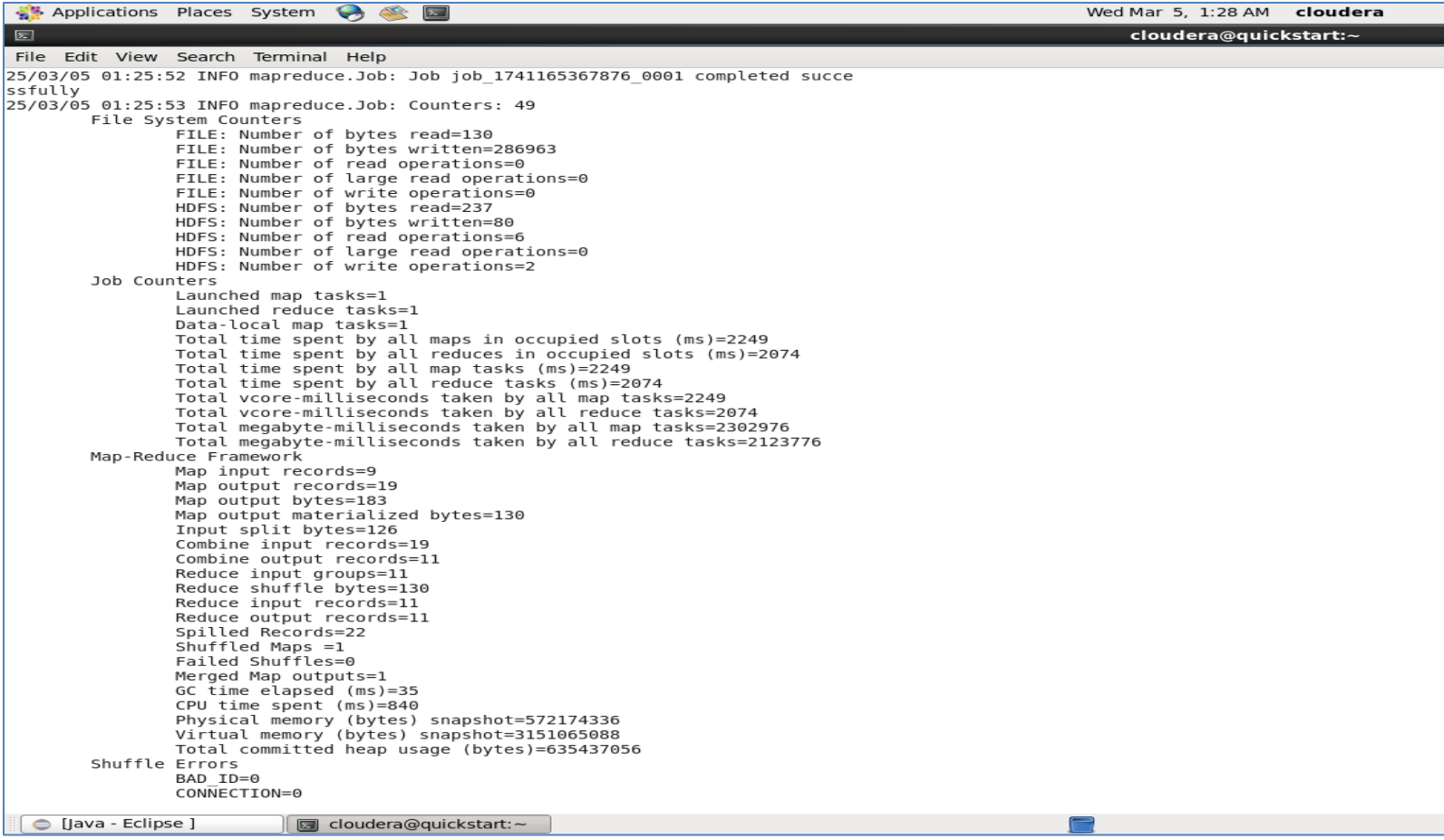
(Hadoop,1)

(Hadoop,1)

At this stage, words are not yet grouped they are just mapped into (key, value) pairs.

- **Groups the same words to gather and sorts the min alphabetic order.**
- **Each reducer gets a list of values for a given key as input and sums up the values for each word.**





Step VIII:

- Open the result:

To see resultant files in output directory with following command:

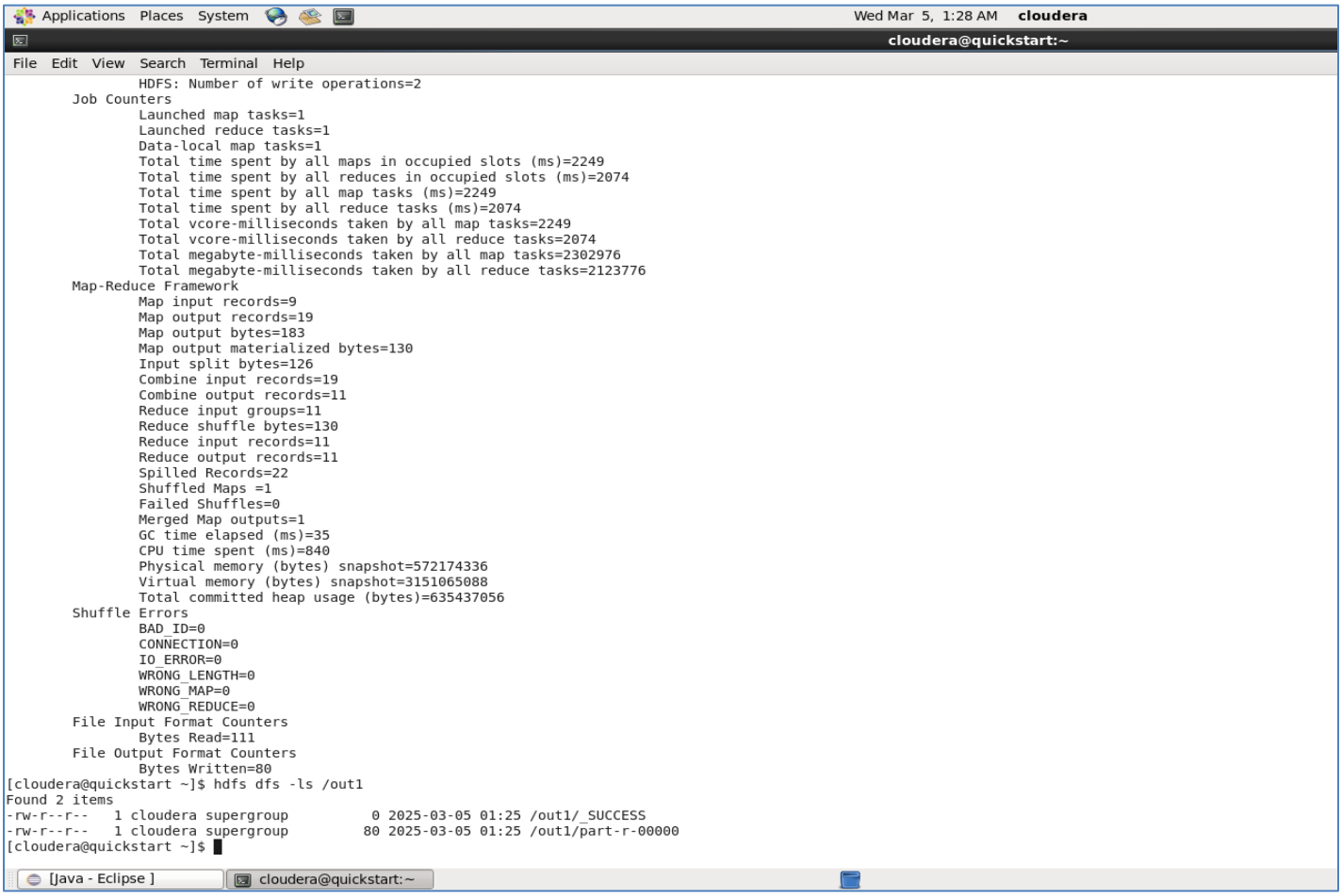
```
hdfsdfs-ls/out1
```

- Twofileswithiniti.e.**_SUCCESS**and**part-r-00000**

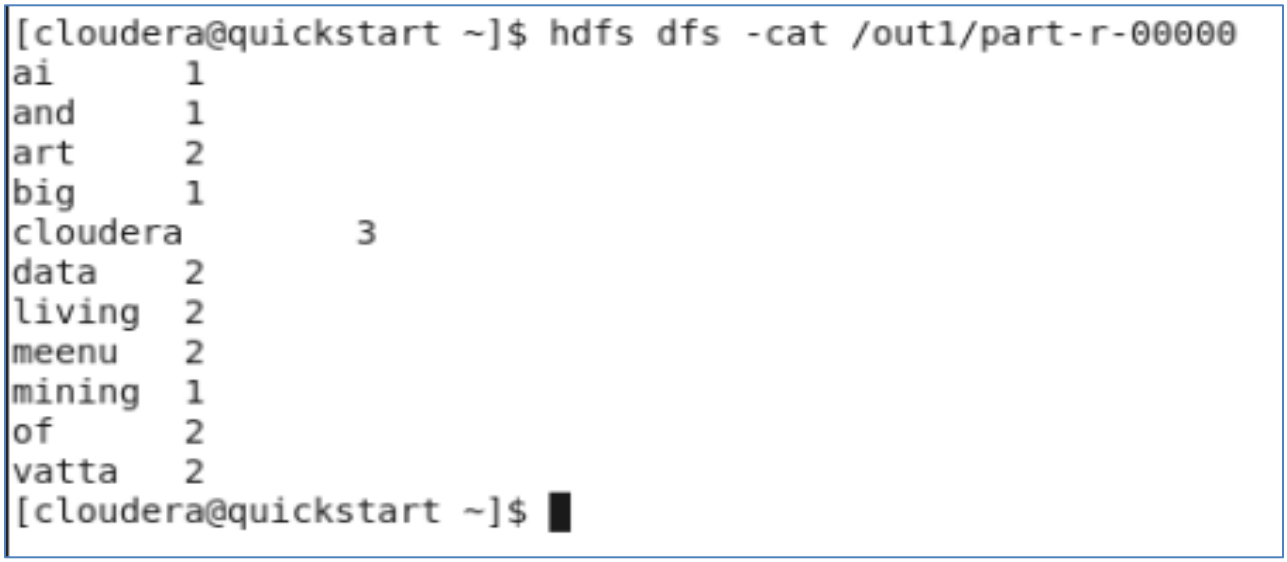
Theoutputofinputfilei.e.countingwordswillbestoredin**part-r-00000**file.This file is generated by HDFS.

To view output run the following command:

```
hdfsdfs-cat/output/part-00000
```



- Opening the file **part-r-00000**. Will result in the wordcount in the File ProocesFile.txt.



```
[cloudera@quickstart ~]$ hdfs dfs -cat /out1/part-r-00000
ai      1
and     1
art     2
big     1
cloudera        3
data    2
living  2
meenu   2
mining  1
of      2
vatta   2
[cloudera@quickstart ~]$ hdfs dfs -cat /inputfolder1/ProocesFile1.txt
meenu vatta
meenu vatta
big  data
 ai and data mining
art of living
art of living
cloudera
cloudera
cloudera
[cloudera@quickstart ~]$
```

Matrix Multiplication using MapReduce in JAVA

Matrix multiplication is a way to combine two matrices, say A and B, to get a new matrix C.

If:

- Matrix A is of size `m x n` (m rows, n columns)
- Matrix B is of size `n x p` (n rows, p columns)

Then the resulting matrix C will be of size `m x p`.

Each element of matrix C, say `C[i][j]`, is calculated by multiplying the **i-th row** of A with the **j-th column** of B:

Why Use MapReduce?

MapReduce is a programming model designed to process **large datasets in parallel** across many machines. Matrix multiplication fits perfectly because:

- Each `C[i][j]` can be computed independently.
- You can split the work of computing parts of the result to multiple machines.
- It works well with sparse matrices (with many zeros) by skipping zeroes.

PRE-REQUISITESTOFOLLOW:

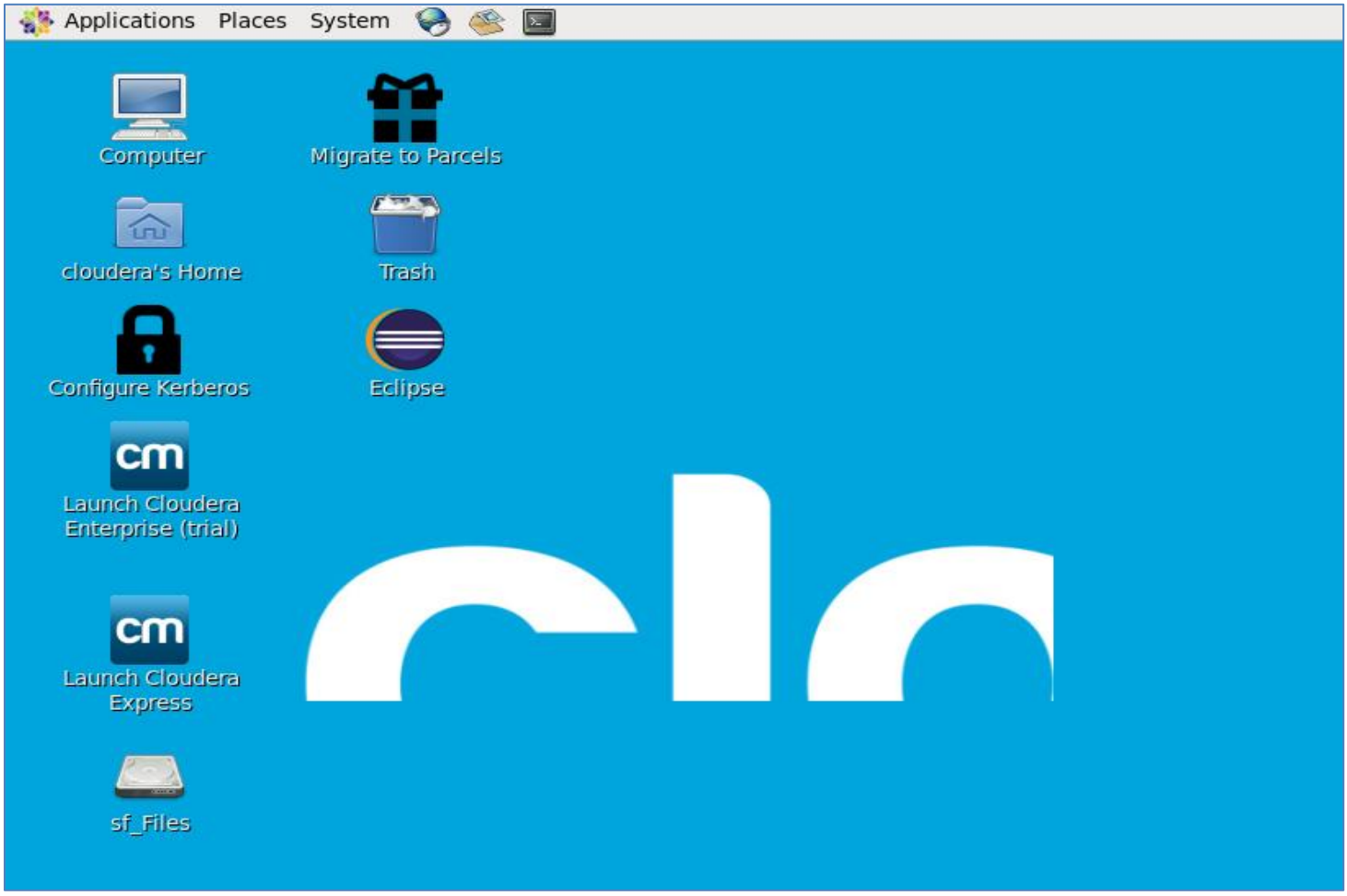
- Hadoop installation or cloudera vm
- Java IDE

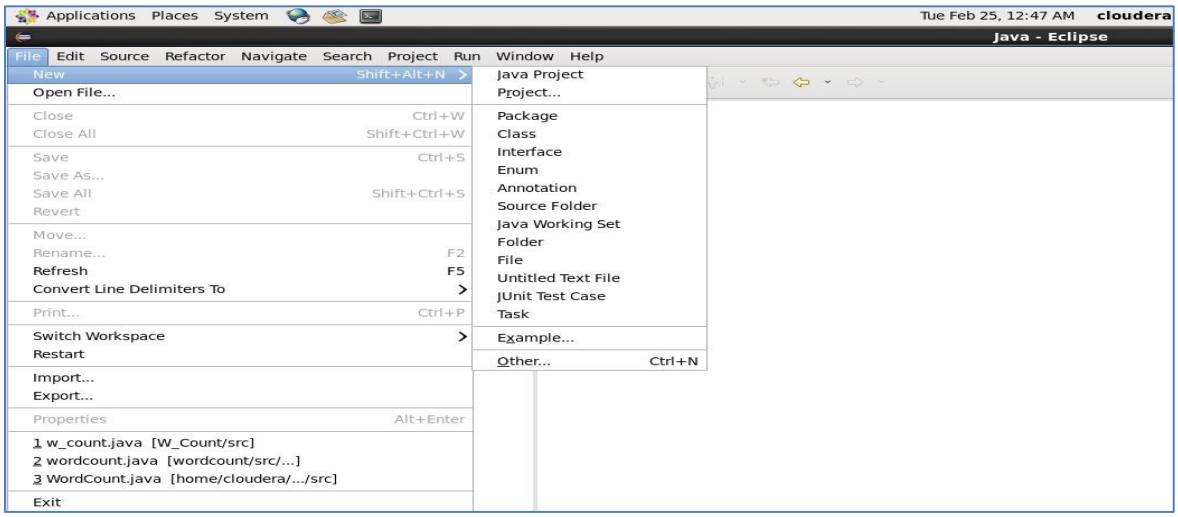
STEPS TO EXECUTE WORD COUNT IN HADOOP ENVIRONMENT.

Step I

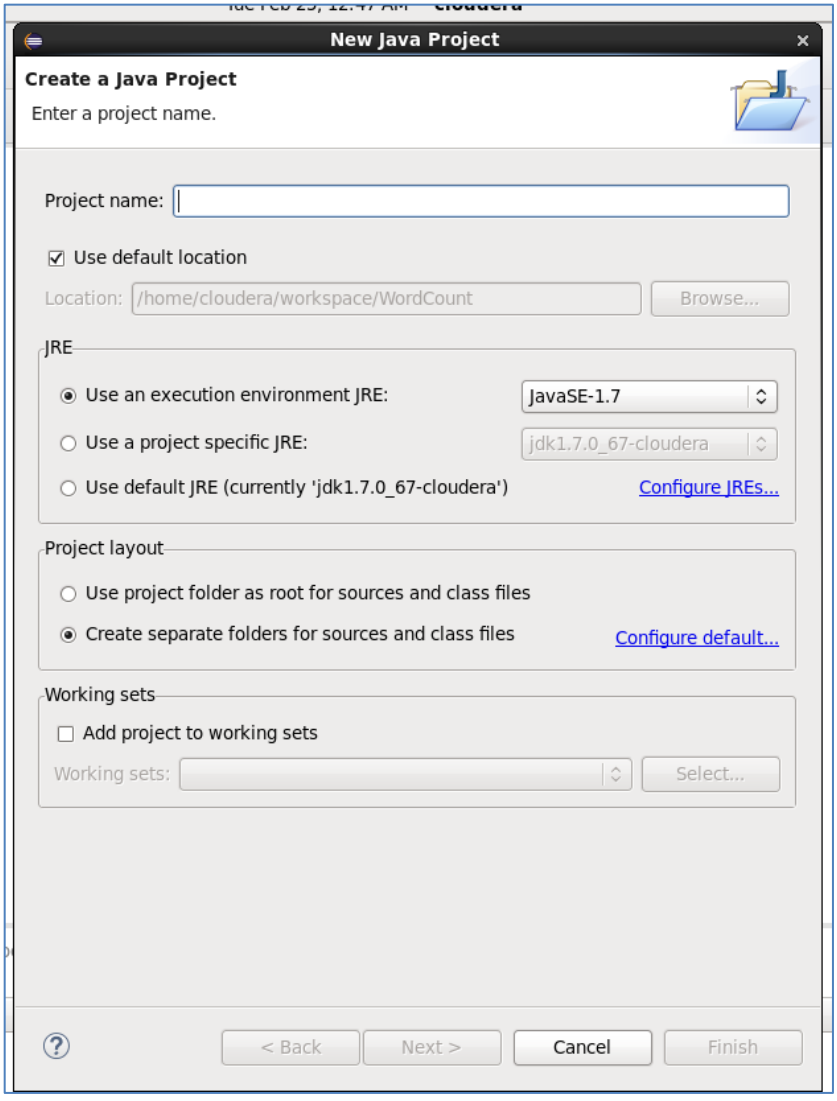
- Open VirtualBox, StartClouderaVM

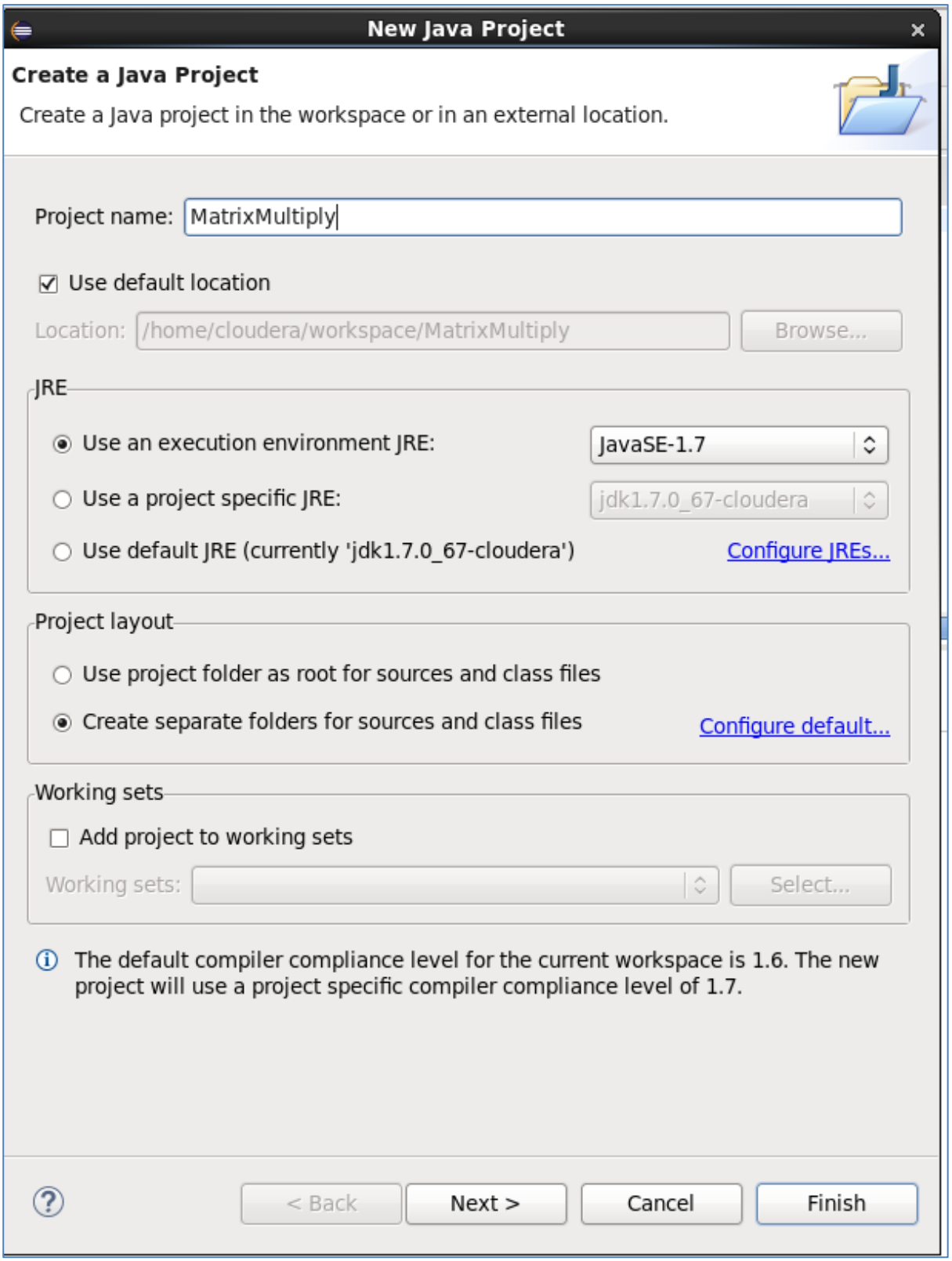
Launch Eclipse>Select File>New>Java Project >Provide Javaproject name.


Applications
Places
System
Computer
Migrate to Parcels
cloudera's Home
Trash
Configure Kerberos
Eclipse
Launch Cloudera Enterprise (trial)
Launch Cloudera Express
sf_Files



- Give name to your Project.

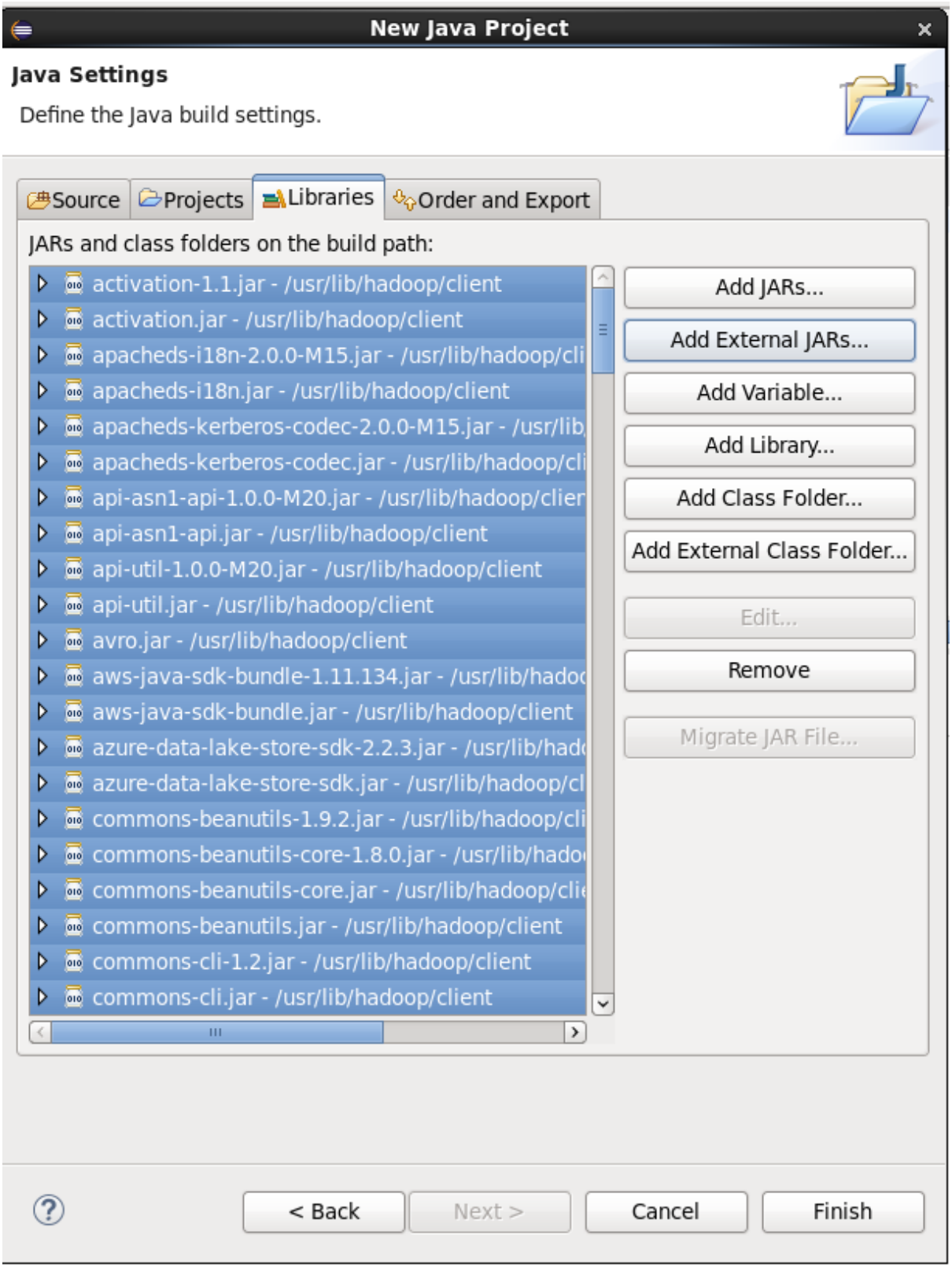




- Click on Next

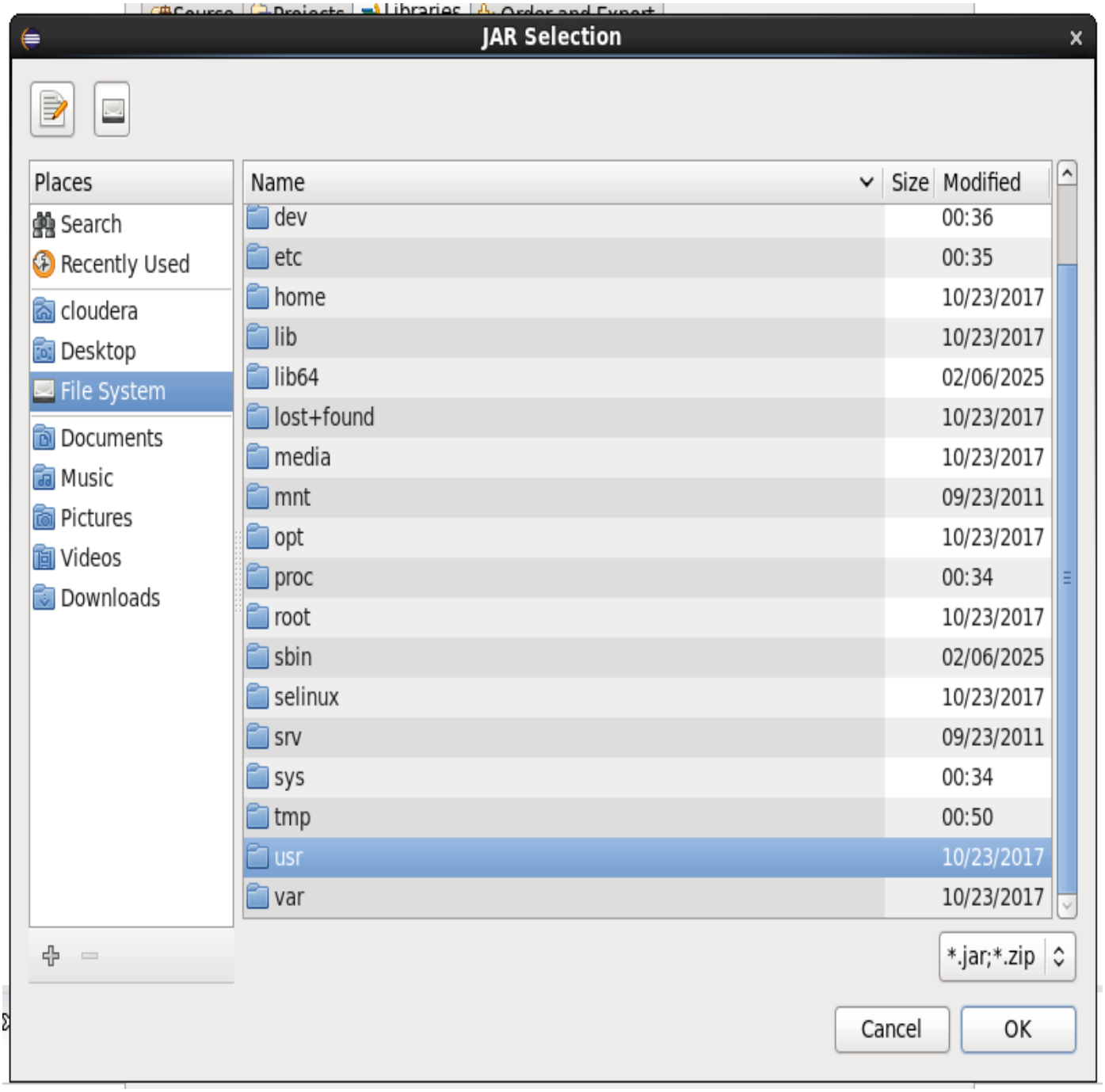
Step II

- ADD HADOOP LIBRARIES (jarfiles)

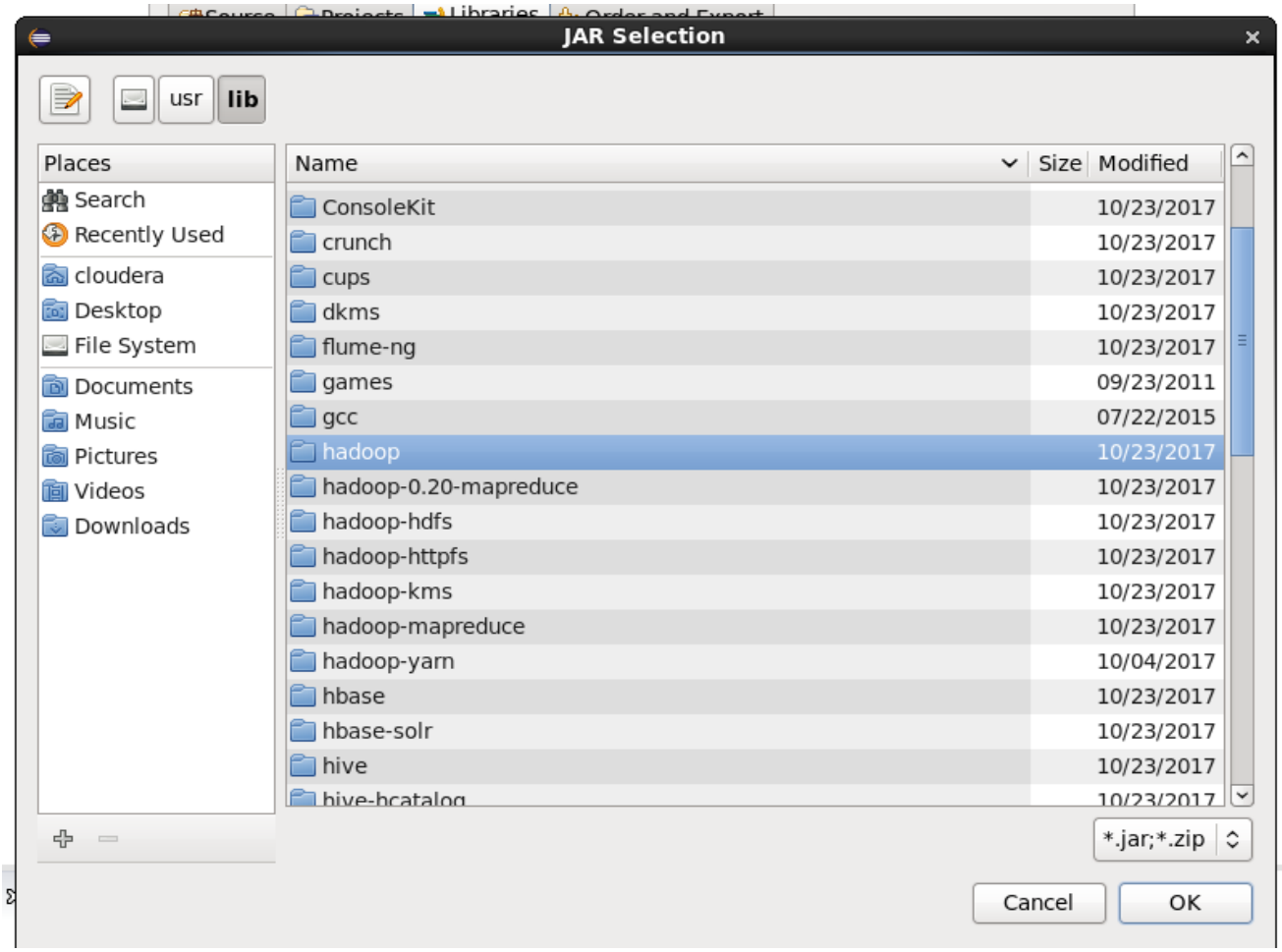


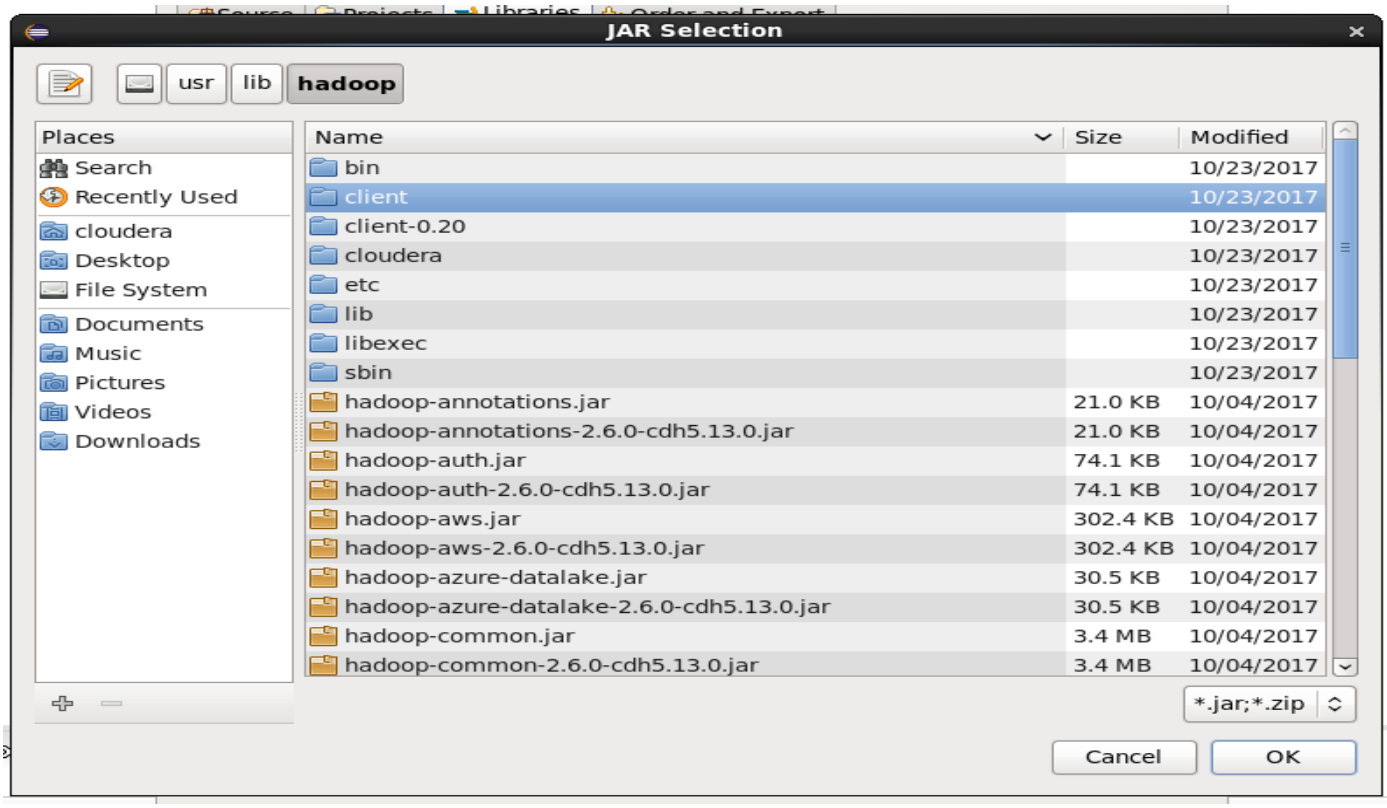
- Go to libraries

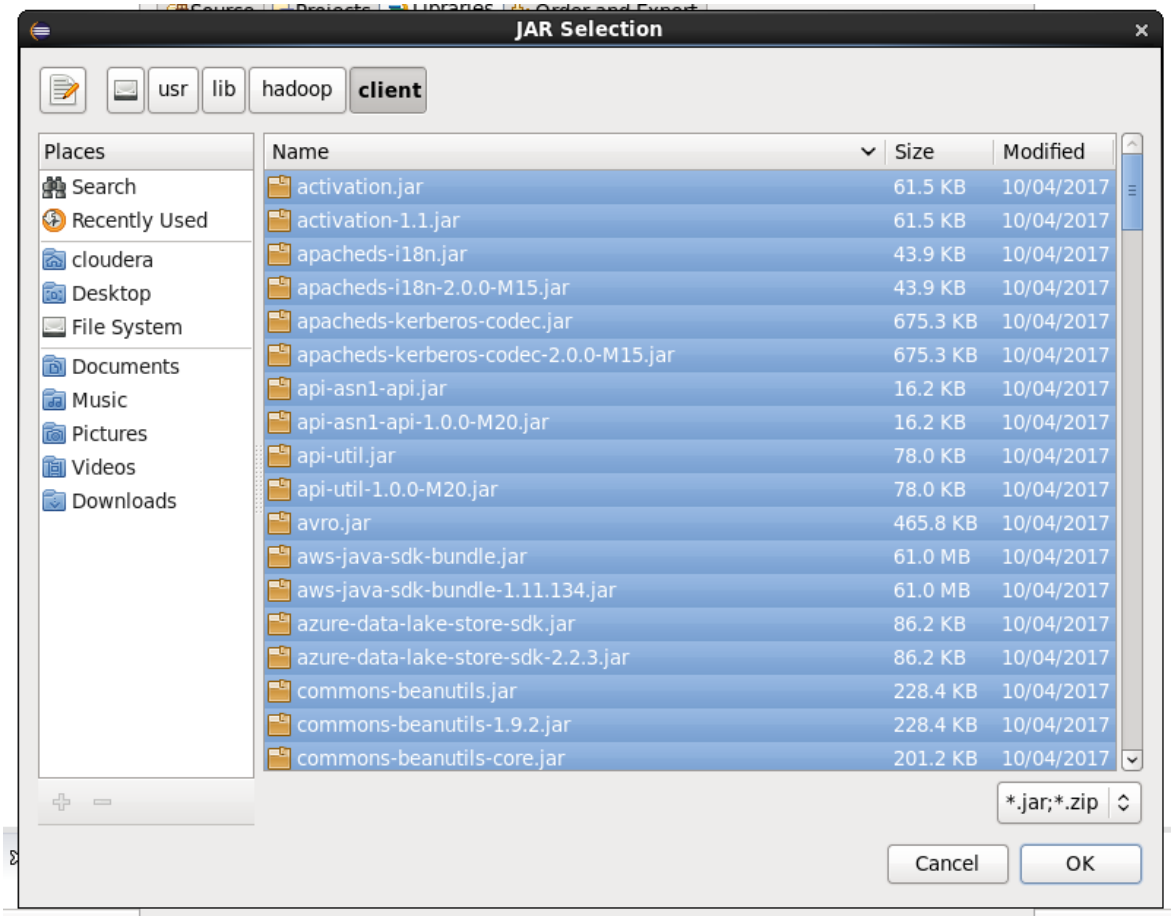
- In above figure, you can see the **ADD External JARs** option on the right hand side. Click on it and add hadoop jar files, which consists of **yarn, hdfs , mapreduce** all required jar files.

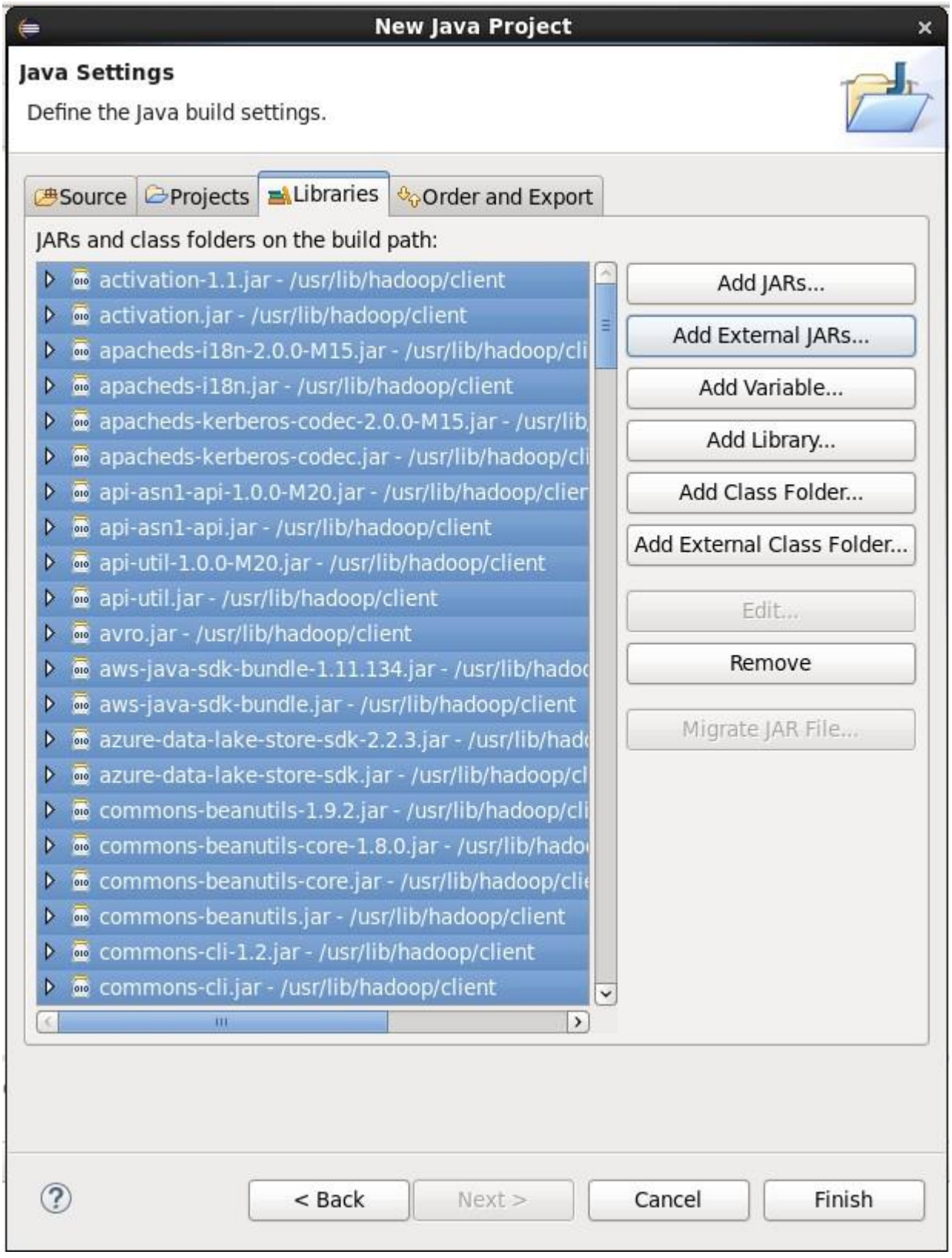


- Click on File System >**usr** >**lib>Hadoop**>selectHadoopjarfiles>OK.
- File System >**client**> OK > FINISH.




JAR Selection
usr
lib
hadoop
Places
Search
Recently Used
cloudera
Desktop
File System
Documents
Music
Pictures
Videos
Downloads
Name
Size
Modified
bin 10/23/2017
client 10/23/2017
client-0.20 10/23/2017
cloudera 10/23/2017
etc 10/23/2017
lib 10/23/2017
libexec 10/23/2017
sbin 10/23/2017
hadoop-annotations.jar 21.0 KB 10/04/2017
hadoop-annotations-2.6.0-cdh5.13.0.jar 21.0 KB 10/04/2017
hadoop-auth.jar 74.1 KB 10/04/2017
hadoop-auth-2.6.0-cdh5.13.0.jar 74.1 KB 10/04/2017
hadoop-aws.jar 302.4 KB 10/04/2017
hadoop-aws-2.6.0-cdh5.13.0.jar 302.4 KB 10/04/2017
hadoop-azure-datalake.jar 30.5 KB 10/04/2017
hadoop-azure-datalake-2.6.0-cdh5.13.0.jar 30.5 KB 10/04/2017
hadoop-common.jar 3.4 MB 10/04/2017
hadoop-common-2.6.0-cdh5.13.0.jar 3.4 MB 10/04/2017
.jar;.zip
Cancel
OK

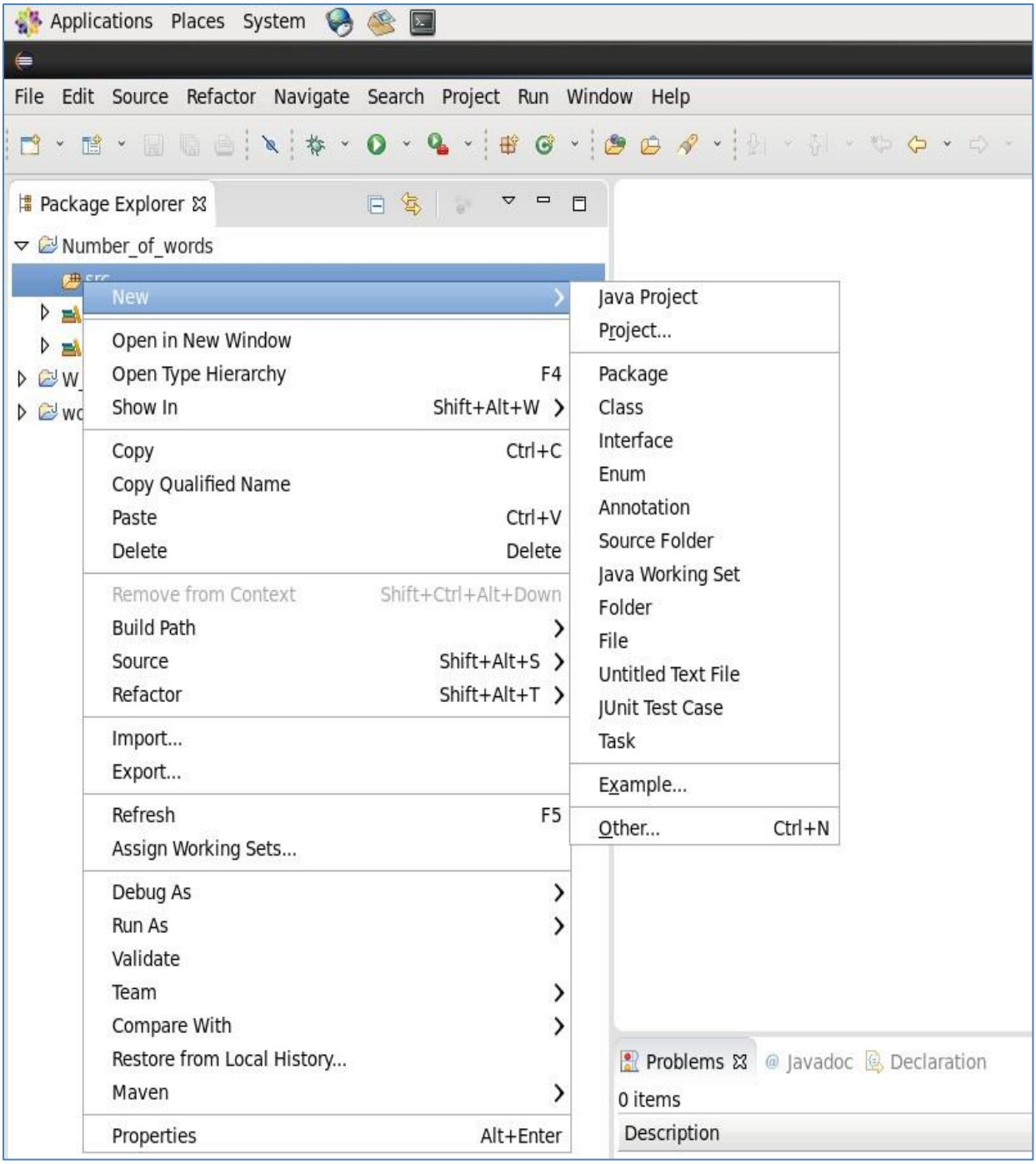

JAR Selection
usr
lib
hadoop
client
Places
Search
Recently Used
cloudera
Desktop
File System
Documents
Music
Pictures
Videos
Downloads
Name
Size
Modified
activation.jar 61.5 KB 10/04/2017
activation-1.1.jar 61.5 KB 10/04/2017
apacheds-i18n.jar 43.9 KB 10/04/2017
apacheds-i18n-2.0.0-M15.jar 43.9 KB 10/04/2017
apacheds-kerberos-codec.jar 675.3 KB 10/04/2017
apacheds-kerberos-codec-2.0.0-M15.jar 675.3 KB 10/04/2017
api-asn1-api.jar 16.2 KB 10/04/2017
api-asn1-api-1.0.0-M20.jar 16.2 KB 10/04/2017
api-util.jar 78.0 KB 10/04/2017
api-util-1.0.0-M20.jar 78.0 KB 10/04/2017
avro.jar 465.8 KB 10/04/2017
aws-java-sdk-bundle.jar 61.0 MB 10/04/2017
aws-java-sdk-bundle-1.11.134.jar 61.0 MB 10/04/2017
azure-data-lake-store-sdk.jar 86.2 KB 10/04/2017
azure-data-lake-store-sdk-2.2.3.jar 86.2 KB 10/04/2017
commons-beanutils.jar 228.4 KB 10/04/2017
commons-beanutils-1.9.2.jar 228.4 KB 10/04/2017
commons-beanutils-core.jar 201.2 KB 10/04/2017
.jar;.zip
Cancel
OK

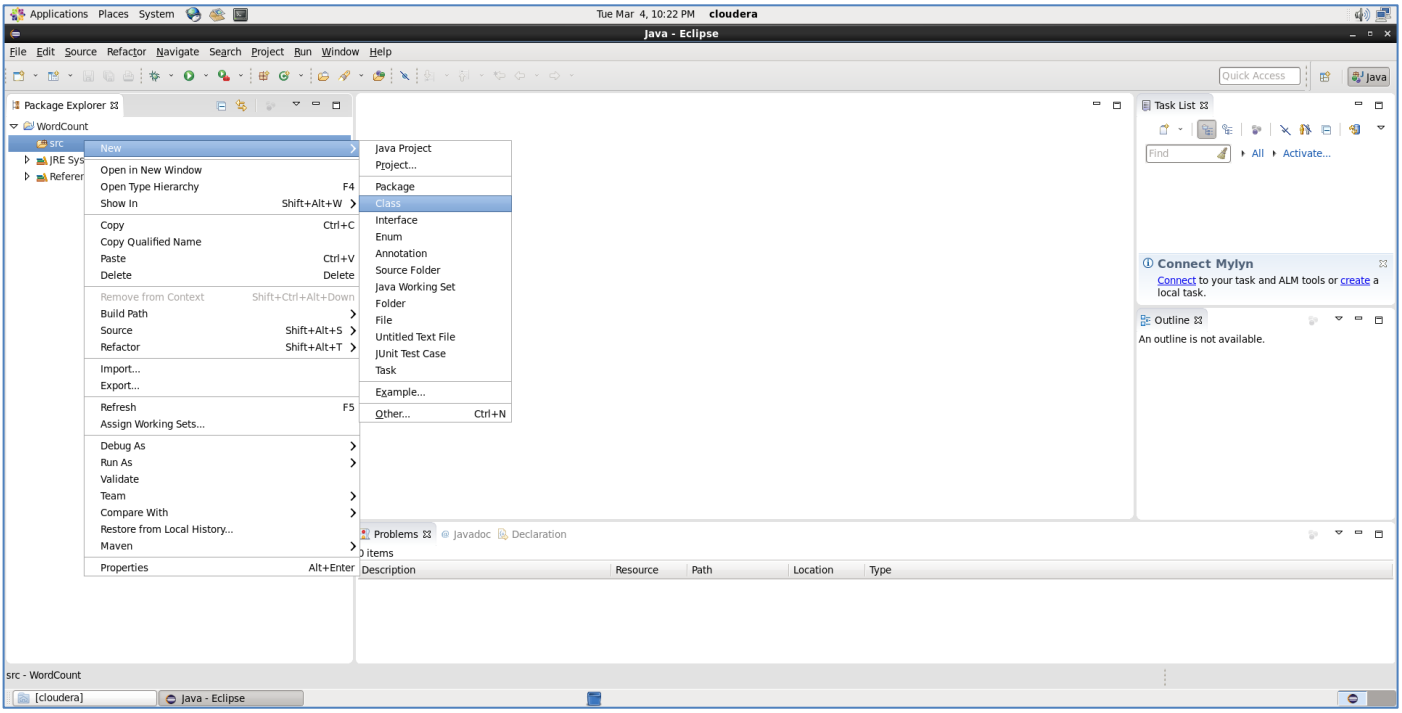

New Java Project
Java Settings
Define the Java build settings.
Source
Projects
Libraries
Order and Export
JARs and class folders on the build path:
activation-1.1.jar - /usr/lib/hadoop/client
activation.jar - /usr/lib/hadoop/client
apacheds-i18n-2.0.0-M15.jar - /usr/lib/hadoop/cli
apacheds-i18n.jar - /usr/lib/hadoop/client
apacheds-kerberos-codec-2.0.0-M15.jar - /usr/lib
apacheds-kerberos-codec.jar - /usr/lib/hadoop/cli
api-asn1-api-1.0.0-M20.jar - /usr/lib/hadoop/clier
api-asn1-api.jar - /usr/lib/hadoop/client
api-util-1.0.0-M20.jar - /usr/lib/hadoop/client
api-util.jar - /usr/lib/hadoop/client
avro.jar - /usr/lib/hadoop/client
aws-java-sdk-bundle-1.11.134.jar - /usr/lib/hadoo
aws-java-sdk-bundle.jar - /usr/lib/hadoop/client
azure-data-lake-store-sdk-2.2.3.jar - /usr/lib/had
azure-data-lake-store-sdk.jar - /usr/lib/hadoop/cl
commons-beanutils-1.9.2.jar - /usr/lib/hadoop/cli
commons-beanutils-core-1.8.0.jar - /usr/lib/hado
commons-beanutils-core.jar - /usr/lib/hadoop/cli
commons-beanutils.jar - /usr/lib/hadoop/client
commons-cli-1.2.jar - /usr/lib/hadoop/client
commons-cli.jar - /usr/lib/hadoop/client
Add JARs...
Add External JARs...
Add Variable...
Add Library...
Add Class Folder...
Add External Class Folder...
Edit...
Remove
Migrate JAR File...
< Back
Next >
Cancel
Finish

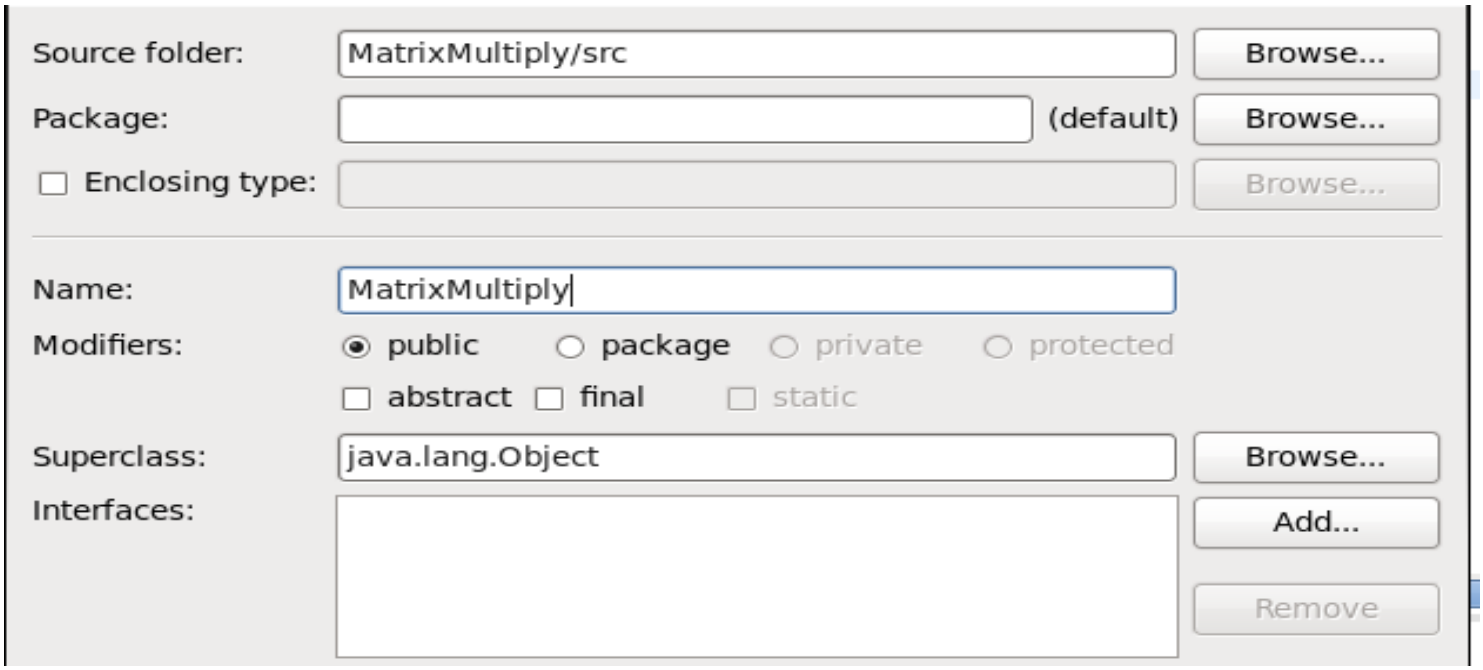
Step III:
CREATING JAVA FILE

- Create a new class that performs Matrix multiplication operation.

  ProjectName > New > Class > Name it Matrixmultiply




Applications Places System
Tue Mar 4, 10:22 PM cloudera
Java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Java
Package Explorer
WordCount
src
New
Open in New Window
Open Type Hierarchy F4
Show In Shift+Alt+W
Copy Ctrl+C
Copy Qualified Name
Paste Ctrl+V
Delete Delete
Remove from Context Shift+Ctrl+Alt+Down
Build Path
Source Shift+Alt+S
Refactor Shift+Alt+T
Import...
Export...
Refresh F5
Assign Working Sets...
Debug As
Run As
Validate
Team
Compare With
Restore from Local History...
Maven
Properties Alt+Enter
Java Project
Project...
Package
Class
Interface
Enum
Annotation
Source Folder
Java Working Set
Folder
File
Untitled Text File
JUnit Test Case
Task
Example...
Other... Ctrl+N
Task List
Find
All
Activate...
Connect Mylyn
Connect to your task and ALM tools or create a local task.
Outline
An outline is not available.
Problems
Javadoc
Declaration
0 items
Description
Resource
Path
Location
Type
src - WordCount
[cloudera]
Java - Eclipse


Source folder:
MatrixMultiply/src
Browse...
Package:
(default)
Browse...
Enclosing type:
Browse...
Name:
MatrixMultiply
Modifiers:
public
package
private
protected
abstract
final
static
Superclass:
java.lang.Object
Browse...
Interfaces:
Add...
Remove

- Copy Mapreduce MM program from below link i.e .available on apache org.

StubDriver.java *MatrixMultiply.java

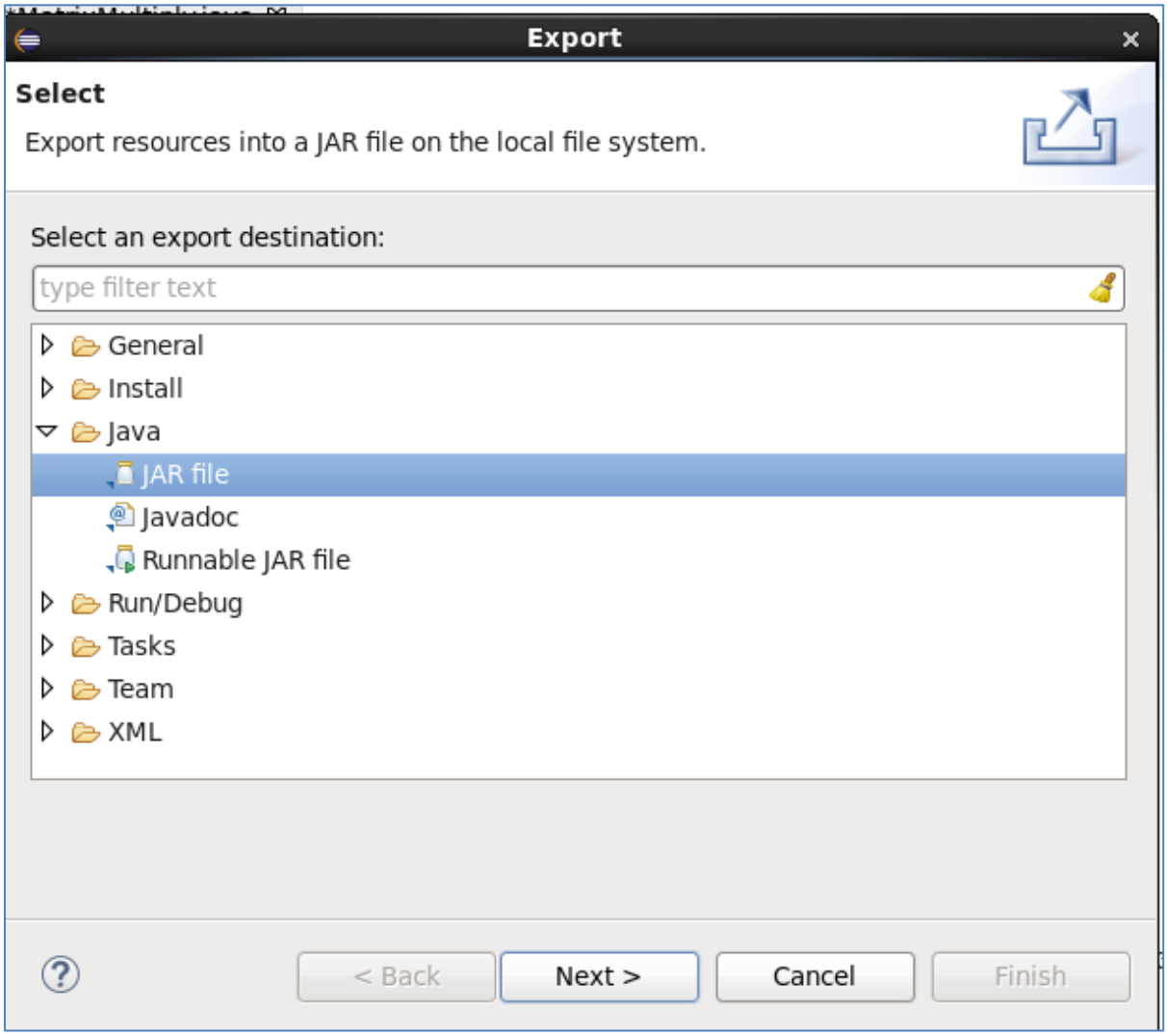
```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

import java.io.IOException;
import java.util.HashMap;

public class MatrixMultiply {

    public static class Map extends 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();
            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);
                    outputValue.set(indicesAndValue[0] + "," + indicesAndValue[2] + "," + indicesAndValue[3]);
                    context.write(outputKey, outputValue);
                }
            } else {
                for (int i = 0; i < m; i++) {
                    outputKey.set(i + "," + indicesAndValue[2]);
                    outputValue.set("N," + indicesAndValue[1] + "," + indicesAndValue[3]);
                    context.write(outputKey, outputValue);
                }
            }
        }
```

- Copy the code from the link and paste.
- This program consists of three classes:
  - **Driver class** (this is entry point which has main function)
  - **Tokenizer Mapper** class which extends the public class **Mapper** and implements the **Map** function.
  - **IntSumReducer** class which extends public class **Reducer** and implements the **Reduce** function.

Step IV:

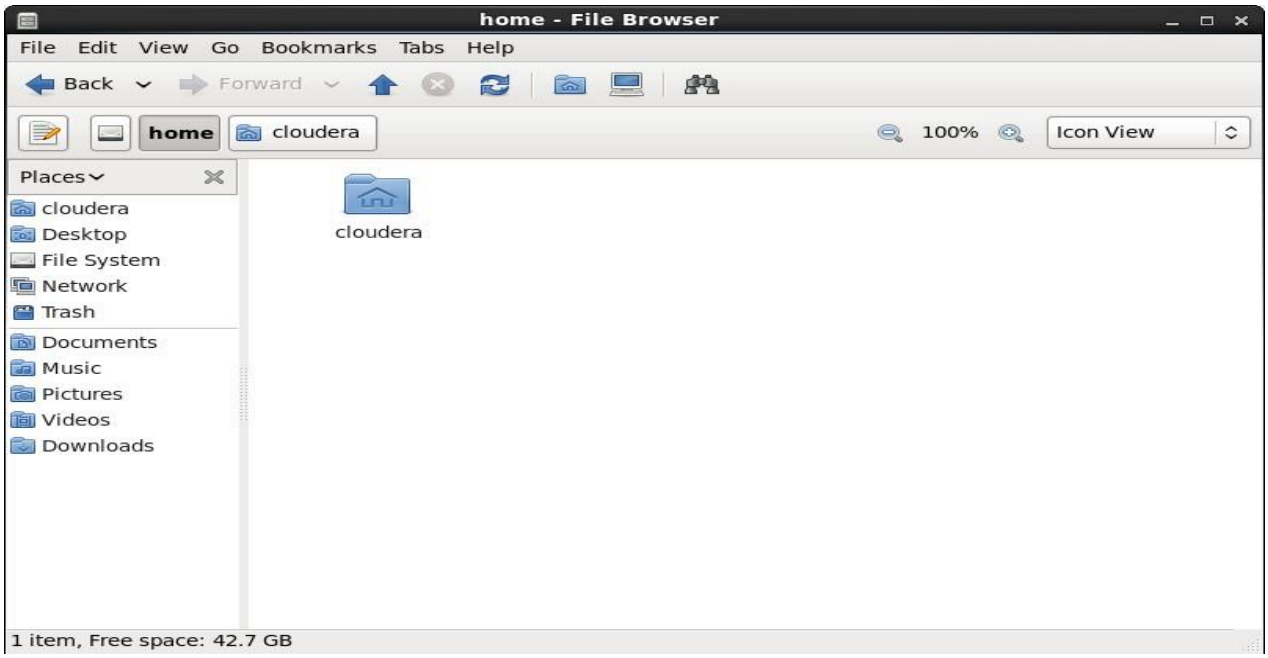
- Make a jar file

RightclickonProject>Export>JARFile>Next>ProvidenameforJARFile(Matrixmultiply.jar)>ClickNext>Finish

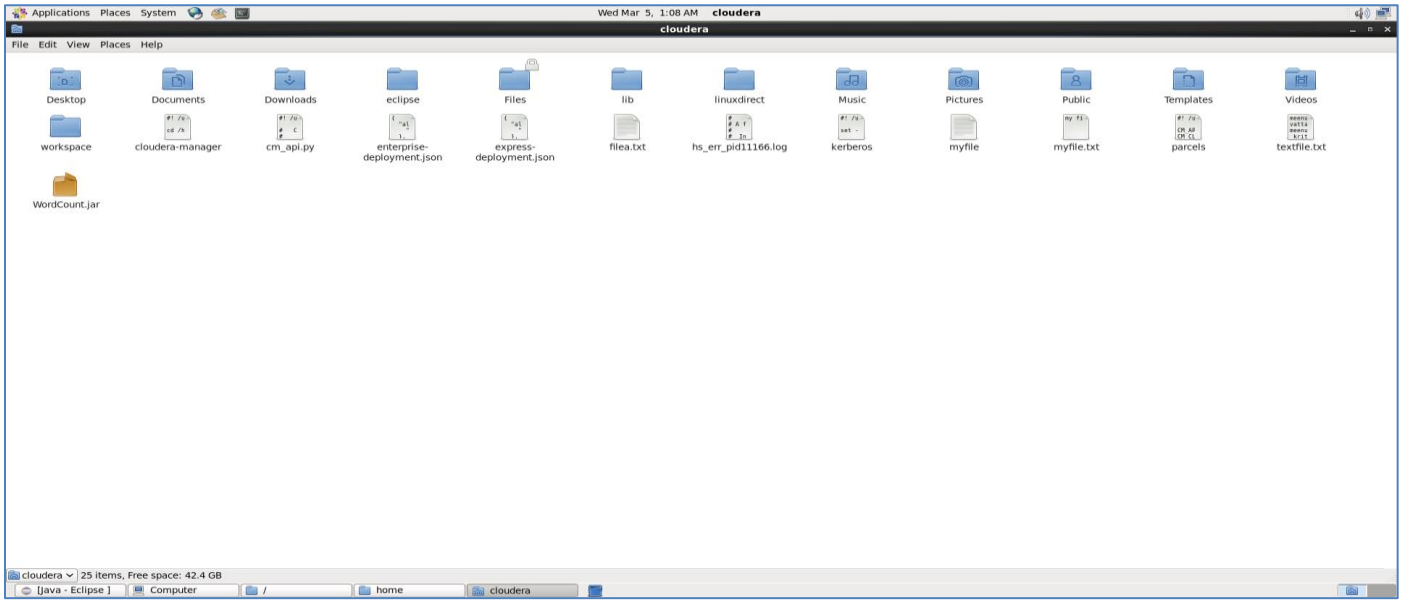


- To check whether jar file is exported or not:

  Go to Applications > File System > Home >cloudera



- If **MatrixMultiply.jar** appears then it is done.



Step V:

- Now open terminal.
- Make directory

```
[root@quickstart cloudera]# hdfs dfs -mkdir /user/cloudera/mmmeenu
```

- Using Shared Folder move M.txt and F.txt file in Linux and then copy it in Hadoop using the –put command .

```
[root@quickstart cloudera]# hdfs dfs -mkdir /user/cloudera/mmmeenu
[root@quickstart cloudera]# hdfs dfs -put /home/cloudera/matrix/M.txt /user/cloude
ra/mmmeenu
```

Matrix Data Model for MapReduce

We represent matrix M as a relation $M(I, J, V)$, with tuples (i, j, m_{ij}), and matrix N as a relation $N(J, K, W)$, with tuples (j, k, n_{jk}). Most matrices are sparse so large amount of cells have value zero. When we represent matrices in this form, we do not need to keep entries for the cells that have values of zero to save large amount of disk space. As input data files, we store matrix M and N on HDFS in following format:

M, i, j, m_{ij}

```
M,0,0,10.0
M,0,2,9.0
M,0,3,9.0
M,1,0,1.0
M,1,1,3.0
M,1,2,18.0
M,1,3,25.2
....
```

N, j, k, n_{jk}

```
N,0,0,1.0
N,0,2,3.0
N,0,4,2.0
N,1,0,2.0
N,3,2,-1.0
N,3,6,4.0
N,4,6,5.0
N,4,0,-1.0
....
```

Step VII:

- Run jarfile:

  Execute Hadoop mapreduce matrixmultiply jarfile.

```
hadoopjar<jarfile_name>classname<inputtextfilepath><output_directorypath>
```

This will start the execution of Mapreduce job.

MapReduce

We will write Map and Reduce functions to process input files. Map function will produce $key, value$ pairs from the input data as it is described in Algorithm 1. Reduce function uses the output of the Map function and performs the calculations and produces $key, value$ pairs as described in Algorithm 2. All outputs are written to HDFS.

Algorithm 1: The Map Function

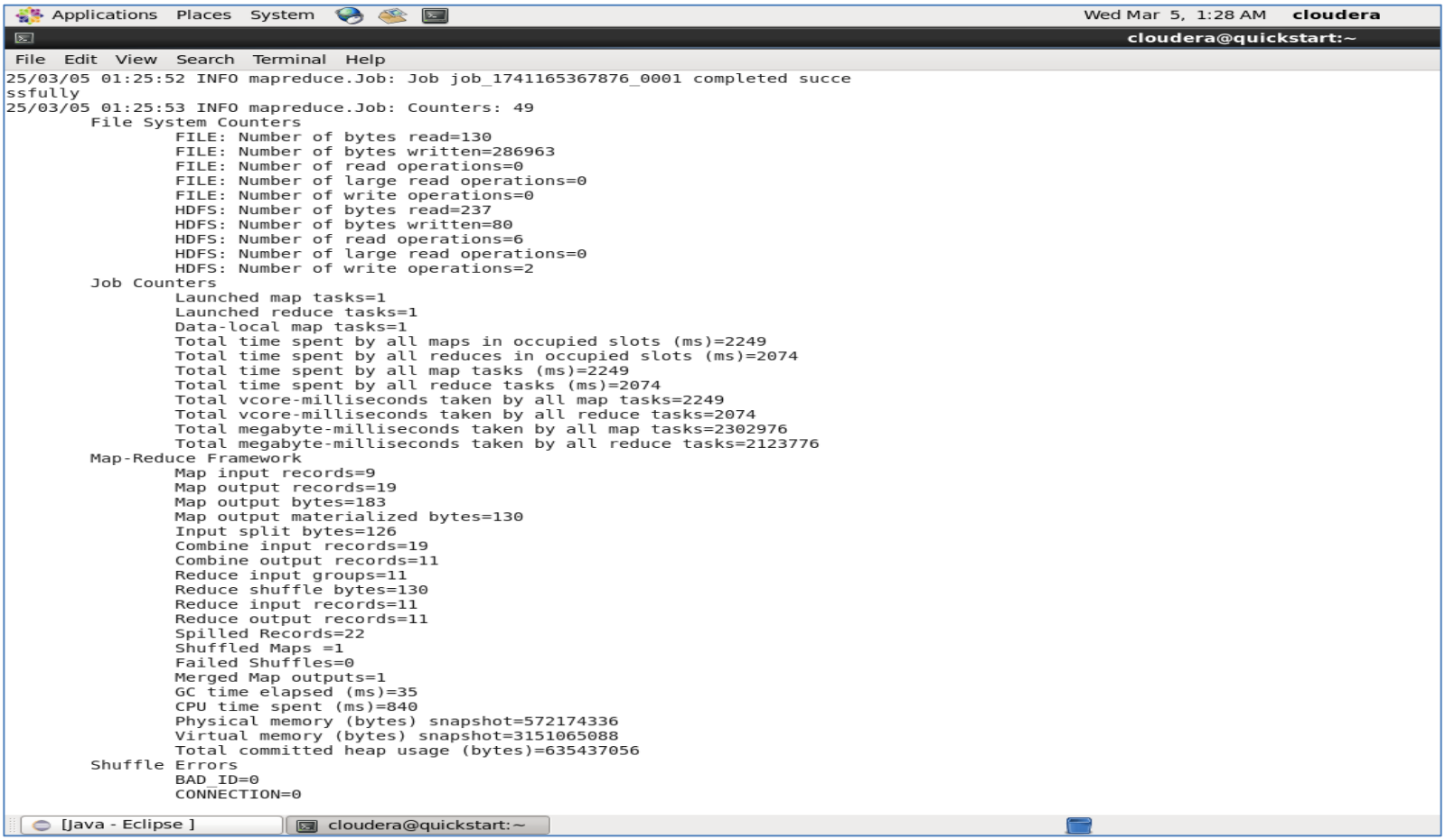
1 **for** *each element* m_{ij} *of* M **do**
2 produce $(key, value)$ pairs as $((i,k),(M,j,m_{ij}))$, for $k = 1,2,3,..$ up to the number of columns of N
3 **for** *each element* n_{jk} *of* N **do**
4 produce $(key, value)$ pairs as $((i,k),(N,j,n_{jk}))$, for $i = 1,2,3,...$ up to the number of rows of M
5 **return** *Set of (key, value) pairs that each key,* (i,k), *has a list with values* (M,j,m_{ij}) *and* (N,j,n_{jk}) *for all possible values of* j

Algorithm 2: The Reduce Function

1 **for** *each key (i,k)* **do**
2 sort values begin with M by j in $list_M$
3 sort values begin with N by j in $list_N$
4 multiply m_{ij} and n_{jk} for j_{th} value of each list
5 sum up $m_{ij} * n_{jk}$
6 **return** $(i,k), \sum_{j=1} m_{ij} * n_{jk}$

The Mapper class extends org.apache.hadoop.mapreduce.Mapper class and implements the map task described in Algorithm 1 and creates the *key,value* pairs from the input files.
Reducer class extends org.apache.hadoop.mapreduce.Reducer class and implements the reduce task described in Algorithm 2 and creates the *key,value* pairs for the product matrix then writes its output to HDFS.

```
[cloudera@quickstart ~]$ hadoop jar MatrixMultiply.jar MatrixMultiply /user/cloudera/Data_file /success_dir
25/03/26 01:11:38 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
25/03/26 01:11:38 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the
25/03/26 01:11:38 INFO input.FileInputFormat: Total input paths to process : 2
25/03/26 01:11:39 INFO mapreduce.JobSubmitter: number of splits:2
25/03/26 01:11:39 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1742975946277_0002
25/03/26 01:11:39 INFO impl.YarnClientImpl: Submitted application application_1742975946277_0002
25/03/26 01:11:39 INFO mapreduce.Job: The url to track the job: http://quickstart.cloudera:8088/proxy/application_174
25/03/26 01:11:39 INFO mapreduce.Job: Running job: job_1742975946277_0002
25/03/26 01:11:45 INFO mapreduce.Job: Job job_1742975946277_0002 running in uber mode : false
25/03/26 01:11:45 INFO mapreduce.Job:  map 0% reduce 0%
25/03/26 01:12:03 INFO mapreduce.Job:  map 25% reduce 0%
25/03/26 01:12:10 INFO mapreduce.Job:  map 42% reduce 0%
```


Applications Places System
Wed Mar 5, 1:28 AM cloudera
cloudera@quickstart:~
File Edit View Search Terminal Help
25/03/05 01:25:52 INFO mapreduce.Job: Job job_1741165367876_0001 completed succe
ssfully
25/03/05 01:25:53 INFO mapreduce.Job: Counters: 49
File System Counters
FILE: Number of bytes read=130
FILE: Number of bytes written=286963
FILE: Number of read operations=0
FILE: Number of large read operations=0
FILE: Number of write operations=0
HDFS: Number of bytes read=237
HDFS: Number of bytes written=80
HDFS: Number of read operations=6
HDFS: Number of large read operations=0
HDFS: Number of write operations=2
Job Counters
Launched map tasks=1
Launched reduce tasks=1
Data-local map tasks=1
Total time spent by all maps in occupied slots (ms)=2249
Total time spent by all reduces in occupied slots (ms)=2074
Total time spent by all map tasks (ms)=2249
Total time spent by all reduce tasks (ms)=2074
Total vcore-milliseconds taken by all map tasks=2249
Total vcore-milliseconds taken by all reduce tasks=2074
Total megabyte-milliseconds taken by all map tasks=2302976
Total megabyte-milliseconds taken by all reduce tasks=2123776
Map-Reduce Framework
Map input records=9
Map output records=19
Map output bytes=183
Map output materialized bytes=130
Input split bytes=126
Combine input records=19
Combine output records=11
Reduce input groups=11
Reduce shuffle bytes=130
Reduce input records=11
Reduce output records=11
Spilled Records=22
Shuffled Maps =1
Failed Shuffles=0
Merged Map outputs=1
GC time elapsed (ms)=35
CPU time spent (ms)=840
Physical memory (bytes) snapshot=572174336
Virtual memory (bytes) snapshot=3151065088
Total committed heap usage (bytes)=635437056
Shuffle Errors
BAD_ID=0
CONNECTION=0
[Java - Eclipse]
cloudera@quickstart:~

Step VIII:

- Open the result:

  To see resultant files in output directory with following command:

  ```
  hdfsdfs-ls/out1
  ```

- Two files within it i.e.**_SUCCESS**and**part-r-00000**

  Theoutputofinputfilei.e.countingwordswillbestoredin**part-r-00000**file.This file is generated by HDFS.

  To view output run the following command:

  ```
  hdfsdfs-cat/output/part-00000
  ```

```
-rw-r--r--   1 cloudera supergroup          0 2025-03-26 01:13 /success_dir/_SUCCESS
-rw-r--r--   1 cloudera supergroup   13814981 2025-03-26 01:13 /success_dir/part-r-00000
[cloudera@quickstart ~]$
```

- Opening the file **part-r-00000**.

```
[cloudera@quickstart ~]$ hdfs dfs -cat /success_dir/part-r-00000
```

```
999,982,551.0
999,983,479.0
999,984,1038.0
999,985,564.0
999,986,909.0
999,987,623.0
999,988,458.0
999,989,692.0
999,99,604.0
999,990,557.0
999,991,877.0
999,992,252.0
999,993,689.0
999,994,659.0
```

Word Count in Apache Pig.

Objective

The goal is to count the number of times each word appears in a text file using Apache Pig, a high-level platform for processing large datasets on Hadoop.

Step 1: Create Directory and File Locally.

- Create a Directory:
  using command **mkdir dir_name.**

```
[cloudera@quickstart ~]$ mkdir labmeenu
[cloudera@quickstart ~]$ mkdir labmeenu/pig
```

- Create a Text File:
  using command **mkdir dir_name/file_name.**

```
[cloudera@quickstart ~]$ touch labmeenu/pig/learning.txt
```

- Add Sample Text:
  using command **cat> mkdir dir_name/file_name.**

```
[cloudera@quickstart ~]$ cat labmeenu/pig/learing.txt
i am learning hive.
iam meenu
i am learning big data
iam meenu
i am learning C.
iam meenu
i am learning hadoop.
iam meenu
i am learning pig.
iam meenu
```

Step 2: Upload File to HDFS

- Create HDFS Directory:
  using command **hdfs dfs -mkdir dir_name**

```
[cloudera@quickstart ~]$ hdfs dfs -mkdir labmeenu/pig
```

- Upload the File:
  using command **hdfs dfs -put filepath_in_local to_hdfs_dir**

```
[cloudera@quickstart ~]$ hdfs dfs -put /home/cloudera/learning.txt  /user/cloudera/labwc/pig
[cloudera@quickstart ~]$ hdfs dfs -cat  /user/cloudera/labwc/pig/learning.txt
i am learning hive.
 iam meenu
i am learning big data
 iam meenu
i am learning C.
 iam meenu
i am learning hadoop.
 iam meenu
i am learning pig.
 iam meenu
[cloudera@quickstart ~]$
```

Step 3: Launch Pig (in Local or MapReduce Mode)

- **Local Mode** (runs on local file system):

  pig -x local

- **MapReduce Mode** (on Hadoop cluster):

  pig

```
[cloudera@quickstart ~]$ pig
log4j:WARN No appenders could be found for logger (org.apache.hadoop.util.Shell).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
2025-04-21 01:19:52,462 [main] INFO  org.apache.pig.Main - Apache Pig version 0.12.0-cdh5.13.0 (rexported) compiled Oct 04 2017, 11:09:03
2025-04-21 01:19:52,462 [main] INFO  org.apache.pig.Main - Logging error messages to: /home/cloudera/pig_1745223592454.log
2025-04-21 01:19:52,472 [main] INFO  org.apache.pig.impl.util.Utils - Default bootup file /home/cloudera/.pigbootup not found
2025-04-21 01:19:52,761 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is deprecated. Instead, use mapreduce.jobtr
2025-04-21 01:19:52,761 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.defaultFS
2025-04-21 01:19:52,761 [main] INFO  org.apache.pig.backend.hadoop.executionengine.HExecutionEngine - Connecting to hadoop file system at: hdfs://quic
2025-04-21 01:19:53,333 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is deprecated. Instead, use mapreduce.jobtr
2025-04-21 01:19:53,333 [main] INFO  org.apache.pig.backend.hadoop.executionengine.HExecutionEngine - Connecting to map-reduce job tracker at: localho
2025-04-21 01:19:53,333 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.defaultFS
2025-04-21 01:19:53,352 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.defaultFS
2025-04-21 01:19:53,353 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is deprecated. Instead, use mapreduce.jobtr
2025-04-21 01:19:53,379 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.defaultFS
2025-04-21 01:19:53,380 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is deprecated. Instead, use mapreduce.jobtr
2025-04-21 01:19:53,409 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.defaultFS
2025-04-21 01:19:53,410 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is deprecated. Instead, use mapreduce.jobtr
2025-04-21 01:19:53,427 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.defaultFS
2025-04-21 01:19:53,427 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is deprecated. Instead, use mapreduce.jobtr
2025-04-21 01:19:53,440 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.defaultFS
2025-04-21 01:19:53,441 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is deprecated. Instead, use mapreduce.jobtr
2025-04-21 01:19:53,453 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.defaultFS
2025-04-21 01:19:53,453 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is deprecated. Instead, use mapreduce.jobtr
2025-04-21 01:19:53,465 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.defaultFS
2025-04-21 01:19:53,465 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is deprecated. Instead, use mapreduce.jobtr
2025-04-21 01:19:53,482 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.defaultFS
2025-04-21 01:19:53,483 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is deprecated. Instead, use mapreduce.jobtr
grunt>
```

Step 4: Pig Script for Word Count

Pig Commands:

: Now load this file stored in HDFS (space delimited file) into pig grunt shell:

Lines = LOAD '/user/cloudera/lab/pig/learning.txt' AS (line:chararray);

dump Lines;

```
grunt> lines = load '/user/cloudera/labmeenu/pig/learning.txt' AS (line : chararray);
grunt> dump lines;
```

```
Input(s):
Successfully read 10 records (543 bytes) from: "/user/cloudera/labwc/pig/learning.txt"

Output(s):
Successfully stored 10 records (196 bytes) in: "hdfs://quickstart.cloudera:8020/tmp/temp-1480431573/tmp367581047"

Counters:
Total records written : 10
Total bytes written : 196
Spillable Memory Manager spill count : 0
Total bags proactively spilled: 0
Total records proactively spilled: 0

Job DAG:
job_1743139950833_0041


2025-04-21 01:41:05,697 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Success!
2025-04-21 01:41:05,697 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.defaultFS
2025-04-21 01:41:05,697 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is deprecated. Instead, use mapreduce.jobtrac
2025-04-21 01:41:05,697 [main] INFO  org.apache.pig.data.SchemaTupleBackend - Key [pig.schematuple] was not set... will not generate code.
2025-04-21 01:41:05,723 [main] INFO  org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input paths to process : 1
2025-04-21 01:41:05,723 [main] INFO  org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths to process : 1
(i am learning hive.)
 iam meenu)
(i am learning big data)
 iam meenu)
(i am learning C.)
 iam meenu)
(i am learning hadoop.)
 iam meenu)
(i am learning pig.)
 iam meenu)
```

- Flatten the words in each line

Words = FOREACH Lines GENERATE FLATTEN (TOKENIZE (line)) as word;

dump Words;

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

```
grunt> dump words;
```

```
Input(s):
Successfully read 10 records (543 bytes) from: "/user/cloudera/labwc/pig/learning.txt"

Output(s):
Successfully stored 31 records (254 bytes) in: "hdfs://quickstart.cloudera:8020/tmp/temp-1480431573/tmp-1831132846"

Counters:
Total records written : 31
Total bytes written : 254
Spillable Memory Manager spill count : 0
Total bags proactively spilled: 0
Total records proactively spilled: 0

Job DAG:
job_1743139950833_0042


2025-04-21 01:44:13,485 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Success!
2025-04-21 01:44:13,485 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.defaultFS
2025-04-21 01:44:13,485 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is deprecated. Instead, use mapreduce.jobtrac
2025-04-21 01:44:13,486 [main] INFO  org.apache.pig.data.SchemaTupleBackend - Key [pig.schematuple] was not set... will not generate code.
2025-04-21 01:44:13,499 [main] INFO  org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input paths to process : 1
2025-04-21 01:44:13,499 [main] INFO  org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths to process : 1
(i)
(am)
(learning)
(hive.)
(iam)
(meenu)
(i)
(am)
(learning)
(big)
(data)
(iam)
(meenu)
(i)
(am)
(learning)
(C.)
(iam)
(meenu)
(i)
(am)
(learning)
(hadoop.)
(iam)
(meenu)
(i)
(am)
```

- Group the same words

grouped = GROUP Words BY word;

dump grouped;

describe grouped;

```
grunt> grouped = GROUP words by word;
grunt> dump grouped;
```

result:

```
Input(s):
Successfully read 10 records (543 bytes) from: "/user/cloudera/labwc/pig/learning.txt"

Output(s):
Successfully stored 11 records (364 bytes) in: "hdfs://quickstart.cloudera:8020/tmp/temp-1480431573/tmp1489941602"

Counters:
Total records written : 11
Total bytes written : 364
Spillable Memory Manager spill count : 0
Total bags proactively spilled: 0
Total records proactively spilled: 0

Job DAG:
job_1743139950833_0043


2025-04-21 01:46:54,995 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Success!
2025-04-21 01:46:54,995 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.defaultFS
2025-04-21 01:46:54,995 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is deprecated. Instead, use mapreduce.jobtrack
2025-04-21 01:46:54,995 [main] INFO  org.apache.pig.data.SchemaTupleBackend - Key [pig.schematuple] was not set... will not generate code.
2025-04-21 01:46:55,019 [main] INFO  org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input paths to process : 1
2025-04-21 01:46:55,019 [main] INFO  org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths to process : 1
(i,{(i),(i),(i),(i),(i)})
(C.,{(C.)})
(am,{(am),(am),(am),(am),(am)})
(big,{(big)})
(iam,{(iam),(iam),(iam),(iam),(iam)})
(data,{(data)})
(pig.,{(pig.)})
(hive.,{(hive.)})
(meenu,{(meenu),(meenu),(meenu),(meenu),(meenu)})
(hadoop.,{(hadoop.)})
(learning,{(learning),(learning),(learning),(learning),(learning)})
grunt>
```

```
grunt> describe grouped;
grouped: {group: chararray,words: {(word: chararray)}}
```

- Now, do the word count

wordcount = FOREACH grouped GENERATE group, COUNT(words);

dump wordcount;

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

```
Success!

Job Stats (time in seconds):
JobId   Maps    Reduces MaxMapTime      MinMapTIme      AvgMapTime      MedianMapTime   MaxReduceTime   MinReduceTime   AvgReduceTime   MedianReducetime
job_1743139950833_0044  1       1       1       1       1       1       1       1       1       grouped,lines,wordcount,words   GROUP_BY,COMBINER

Input(s):
Successfully read 10 records (543 bytes) from: "/user/cloudera/labwc/pig/learning.txt"

Output(s):
Successfully stored 11 records (114 bytes) in: "hdfs://quickstart.cloudera:8020/tmp/temp-1480431573/tmp-2102893147"

Counters:
Total records written : 11
Total bytes written : 114
Spillable Memory Manager spill count : 0
Total bags proactively spilled: 0
Total records proactively spilled: 0

Job DAG:
job_1743139950833_0044


2025-04-21 01:49:46,739 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Success!
2025-04-21 01:49:46,739 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.defaultFS
2025-04-21 01:49:46,739 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is deprecated. Instead, use mapreduce.jobtracke
2025-04-21 01:49:46,739 [main] INFO  org.apache.pig.data.SchemaTupleBackend - Key [pig.schematuple] was not set... will not generate code.
2025-04-21 01:49:46,748 [main] INFO  org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input paths to process : 1
2025-04-21 01:49:46,748 [main] INFO  org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths to process : 1
(i,5)
(C.,1)
(am,5)
(big,1)
(iam,5)
(data,1)
(pig.,1)
(hive.,1)
(meenu,5)
(hadoop.,1)
(learning,5)
```

Driver Analysis Using Apache Pig

Objective

To analyze driver-related data (such as names, trips, ratings, and earnings) using Apache Pig, and perform tasks like:

- Loading and processing driver data
- Calculating total earnings
- Calculating average ratings
- Counting total trips
- Filtering high-performing drivers

Step 1: Transferring files.

- Start by transferring the files we will be working with to the **linux file system** of our vm

$ wget https://cse.sc.edu/~rose/587/CSV/drivers.csv

$ wget https://cse.sc.edu/~rose/587/CSV/timesheet.csv

```
[cloudera@quickstart ~]$ wget --no-check-certificate https://cse.sc.edu/~rose/587/CSV/drivers.csv
--2025-04-21 02:24:23--  https://cse.sc.edu/~rose/587/CSV/drivers.csv
Resolving cse.sc.edu... 129.252.138.13
Connecting to cse.sc.edu|129.252.138.13|:443... connected.
WARNING: cannot verify cse.sc.edu's certificate, issued by "/C=US/O=Let's Encrypt/CN=E6":
  Unable to locally verify the issuer's authority.
HTTP request sent, awaiting response... 200 OK
Length: 2043 (2.0K) [text/csv]
Saving to: "drivers.csv"

100%[======================================================================================================

2025-04-21 02:24:24 (300 MB/s) - "drivers.csv" saved [2043/2043]

[cloudera@quickstart ~]$ wget --no-check-certificate https://cse.sc.edu/~rose/587/CSV/timesheet.csv
--2025-04-21 02:24:05--  https://cse.sc.edu/~rose/587/CSV/timesheet.csv
Resolving cse.sc.edu... 129.252.138.13
Connecting to cse.sc.edu|129.252.138.13|:443... connected.
WARNING: cannot verify cse.sc.edu's certificate, issued by "/C=US/O=Let's Encrypt/CN=E6":
  Unable to locally verify the issuer's authority.
HTTP request sent, awaiting response... 200 OK
Length: 26205 (26K) [text/csv]
Saving to: "timesheet.csv"

100%[======================================================================================================

2025-04-21 02:24:06 (96.8 KB/s) - "timesheet.csv" saved [26205/26205]
```

Step 2: Upload File to HDFS

$ hadoop fs -put drivers.csv /user/cloudera/lab/pig

$ hadoop fs –put timesheet.csv /user/cloudera/lab/pig

```
[cloudera@quickstart ~]$ hdfs dfs -put  drivers.csv /user/cloudera/meenu/pig
[cloudera@quickstart ~]$ hdfs dfs -put  timesheet.csv /user/cloudera/meenu/pig
```

Step 3: Launch Pig Grunt Shell

Invoke grunt (the pig interactive environment):

$ pig

grunt>

```
[cloudera@quickstart ~]$ pig
log4j:WARN No appenders could be found for logger (or
log4j:WARN Please initialize the log4j system properl
log4j:WARN See http://logging.apache.org/log4j/1.2/fa
2025-04-21 02:27:59,518 [main] INFO  org.apache.pig.M
```

Determine the current working directory:

grunt>*pwd*

Verify that the files we will be working with are in the current working directory:

grunt>*ls*

- Step 4: Pig Script for Driver Analysis

  Load Data:

  grunt>*drivers = LOAD 'drivers.csv' USING PigStorage(',');*

  By using this command the **drivers.csv** file is uploaded in alias named **drivers**.

Always specify the full path to the file when loading.

```
grunt> drivers = LOAD '/user/cloudera/meenu/pig/drivers.csv' USING PigStorage(',');
grunt> dump drivers;
```

Now dump **drivers** to the console. This should display the content of the file in the Pig shell.

```
(driverId,name,ssn,location,certified,wage-plan)
(10,George Vetticaden,621011971,244-4532 Nulla Rd.,N,miles)
(11,Jamie Engesser,262112338,366-4125 Ac Street,N,miles)
(12,Paul Coddin,198041975,Ap #622-957 Risus. Street,Y,hours)
(13,Joe Niemiec,139907145,2071 Hendrerit. Ave,Y,hours)
(14,Adis Cesir,820812209,Ap #810-1228 In St.,Y,hours)
(15,Rohit Bakshi,239005227,648-5681 Dui- Rd.,Y,hours)
(16,Tom McCuch,363303105,P.O. Box 313- 962 Parturient Rd.,Y,hours)
(17,Eric Mizell,123808238,P.O. Box 579- 2191 Gravida. Street,Y,hours)
(18,Grant Liu,171010151,Ap #928-3159 Vestibulum Av.,Y,hours)
(19,Ajay Singh,160005158,592-9430 Nonummy Avenue,Y,hours)
(20,Chris Harris,921812303,883-2691 Proin Avenue,Y,hours)
(21,Jeff Markham,209408086,Ap #852-7966 Facilisis St.,Y,hours)
(22,Nadeem Asghar,783204269,154-9147 Aliquam Ave,Y,hours)
(23,Adam Diaz,928312208,P.O. Box 260- 6127 Vitae Road,Y,hours)
(24,Don Hilborn,254412152,4361 Ac Road,Y,hours)
(25,Jean-Philippe Playe,913310051,P.O. Box 812- 6238 Ac Rd.,Y,hours)
(26,Michael Aube,124705141,P.O. Box 213- 8948 Nec Ave,Y,hours)
(27,Mark Lochbihler,392603159,8355 Ipsum St.,Y,hours)
(28,Olivier Renault,959908181,P.O. Box 243- 6509 Erat. Avenue,Y,hours)
(29,Teddy Choi,185502192,P.O. Box 106- 7003 Amet Rd.,Y,hours)
(30,Dan Rice,282307061,Ap #881-9267 Mollis Avenue,Y,hours)
(31,Rommel Garcia,858912101,P.O. Box 945- 6015 Sociis St.,Y,hours)
(32,Ryan Templeton,290304287,765-6599 Egestas. Av.,Y,hours)
(33,Sridhara Sabbella,967409015,Ap #477-2507 Sagittis Avenue,Y,hours)
(34,Frank Romano,391407216,Ap #753-6814 Quis Ave,Y,hours)
(35,Emil Siemes,971401151,321-2976 Felis Rd.,Y,hours)
(36,Andrew Grande,245303216,Ap #685-9598 Egestas Rd.,Y,hours)
(37,Wes Floyd,190504074,P.O. Box 269- 9611 Nulla Street,Y,hours)
(38,Scott Shaw,386411175,276 Lobortis Road,Y,hours)
(39,David Kaiser,967706052,9185 At Street,Y,hours)
(40,Nicolas Maillard,208510217,1027 Quis Rd.,Y,hours)
(41,Greg Phillips,308103116,P.O. Box 847- 5961 Arcu. Road,Y,hours)
(42,Randy Gelhausen,853302254,145-4200 In- Avenue,Y,hours)
(43,Dave Patton,977706052,3028 A- St.,Y,hours)
```

- Filter Drivers with driver id >15

grunt>*raw_drivers = FILTER drivers BY $0>15;*

```
grunt> raw_drivers = FILTER drivers BY $0>15;
2025-04-21 02:53:25,358 [main] WARN  org.apache.p
grunt> dump raw_drivers;
```

```
(16,Tom McCuch,363303105,P.O. Box 313- 962 Parturient Rd.,Y,hours)
(17,Eric Mizell,123808238,P.O. Box 579- 2191 Gravida. Street,Y,hours)
(18,Grant Liu,171010151,Ap #928-3159 Vestibulum Av.,Y,hours)
(19,Ajay Singh,160005158,592-9430 Nonummy Avenue,Y,hours)
(20,Chris Harris,921812303,883-2691 Proin Avenue,Y,hours)
(21,Jeff Markham,209408086,Ap #852-7966 Facilisis St.,Y,hours)
(22,Nadeem Asghar,783204269,154-9147 Aliquam Ave,Y,hours)
(23,Adam Diaz,928312208,P.O. Box 260- 6127 Vitae Road,Y,hours)
(24,Don Hilborn,254412152,4361 Ac Road,Y,hours)
(25,Jean-Philippe Playe,913310051,P.O. Box 812- 6238 Ac Rd.,Y,hours)
(26,Michael Aube,124705141,P.O. Box 213- 8948 Nec Ave,Y,hours)
(27,Mark Lochbihler,392603159,8355 Ipsum St.,Y,hours)
(28,Olivier Renault,959908181,P.O. Box 243- 6509 Erat. Avenue,Y,hours)
(29,Teddy Choi,185502192,P.O. Box 106- 7003 Amet Rd.,Y,hours)
(30,Dan Rice,282307061,Ap #881-9267 Mollis Avenue,Y,hours)
(31,Rommel Garcia,858912101,P.O. Box 945- 6015 Sociis St.,Y,hours)
(32,Ryan Templeton,290304287,765-6599 Egestas. Av.,Y,hours)
(33,Sridhara Sabbella,967409015,Ap #477-2507 Sagittis Avenue,Y,hours)
(34,Frank Romano,391407216,Ap #753-6814 Quis Ave,Y,hours)
(35,Emil Siemes,971401151,321-2976 Felis Rd.,Y,hours)
(36,Andrew Grande,245303216,Ap #685-9598 Egestas Rd.,Y,hours)
(37,Wes Floyd,190504074,P.O. Box 269- 9611 Nulla Street,Y,hours)
(38,Scott Shaw,386411175,276 Lobortis Road,Y,hours)
(39,David Kaiser,967706052,9185 At Street,Y,hours)
(40,Nicolas Maillard,208510217,1027 Quis Rd.,Y,hours)
(41,Greg Phillips,308103116,P.O. Box 847- 5961 Arcu. Road,Y,hours)
(42,Randy Gelhausen,853302254,145-4200 In- Avenue,Y,hours)
(43,Dave Patton,977706052,3028 A- St.,Y,hours)
```

- **Extracts** the first two columns from the **`raw_drivers`** dataset and assigns them the names **`driverId`** and `name` respectively.

grunt>*drivers_details = FOREACH raw_drivers GENERATE $0 AS driverId, $1 AS name;*

```
grunt> drivers_detials = FOREACH raw_drivers GENERATE $0 AS driverId , $1 AS name;
```

Let's look at the resulting table; column one is the ***driverID*** and column two is the **driver's name**:
grunt>*dump drivers_details;*

```
grunt> dump drivers_detials;
```

```
(16,Tom McCuch)
(17,Eric Mizell)
(18,Grant Liu)
(19,Ajay Singh)
(20,Chris Harris)
(21,Jeff Markham)
(22,Nadeem Asghar)
(23,Adam Diaz)
(24,Don Hilborn)
(25,Jean-Philippe Playe)
(26,Michael Aube)
(27,Mark Lochbihler)
(28,Olivier Renault)
(29,Teddy Choi)
(30,Dan Rice)
(31,Rommel Garcia)
(32,Ryan Templeton)
(33,Sridhara Sabbella)
(34,Frank Romano)
(35,Emil Siemes)
(36,Andrew Grande)
(37,Wes Floyd)
(38,Scott Shaw)
(39,David Kaiser)
(40,Nicolas Maillard)
(41,Greg Phillips)
(42,Randy Gelhausen)
(43,Dave Patton)
```

Loading the second dataset the **timesheets.**

Step 1 LOAD:

grunt>*timesheet = LOAD 'timesheet.csv' USING PigStorage(',');*

```
grunt> timesheet = load '/user/cloudera/meenu/pig/timesheet.csv' USING PigStorage(',');
```

grunt>*dump timesheet;*

```
(43,15,53,2579)
(43,16,56,2519)
(43,17,54,2584)
(43,18,47,2665)
(43,19,55,2511)
(43,20,60,2677)
(43,21,52,2585)
(43,22,60,2719)
(43,23,48,2655)
(43,24,48,2641)
(43,25,53,2512)
(43,26,48,2612)
(43,27,58,2614)
(43,28,60,2551)
(43,29,55,2682)
(43,30,49,2504)
(43,31,51,2701)
(43,32,57,2554)
(43,33,52,2730)
(43,34,54,2783)
(43,35,51,2681)
(43,36,51,2655)
(43,37,46,2629)
(43,38,58,2739)
(43,39,47,2535)
(43,40,50,2512)
(43,41,51,2701)
(43,42,55,2538)
(43,43,58,2775)
(43,44,56,2545)
(43,45,46,2671)
(43,46,57,2680)
(43,47,50,2572)
(43,48,52,2517)
(43,49,56,2743)
(43,50,59,2665)
(43,51,58,2593)
(43,52,48,2764)
grunt>
```

- filter out the first row which contains column headings

  grunt>*raw_timesheet = FILTER timesheet by $0>1;*

  ```
  grunt> raw_sheet = FILTER timesheet by $0>1;
  ```

  We now select a subset of columns from **raw_timesheet** to create **timesheet_logged**: Create column 1 from the first column of **raw_timesheets** and label it **driverId.**

  Create column 2 from the third column of **raw_timesheets** and label it **hours_logged**

  Create column 3 from the second column of **raw_timesheets** and label it

miles_logged.

grunt>*timesheet_logged = FOREACH raw_timesheet GENERATE $0 AS driverId, $2 AS hours_logged, $3 AS miles_logged;*

```
grunt> timesheet_log = FOREACH raw_sheet GENERATE $0 AS driverId , $2 as hours_log , $3 AS miles_log;
```

Let's take a peek at **timesheet_logged**. We should have a row for every week of every drivers timesheet data:

grunt>*dump timesheet_logged;*

```
(43,50,2528)
(43,57,2721)
(43,51,2722)
(43,59,2681)
(43,52,2683)
(43,46,2663)
(43,53,2579)
(43,56,2519)
(43,54,2584)
(43,47,2665)
(43,55,2511)
(43,60,2677)
(43,52,2585)
(43,60,2719)
(43,48,2655)
(43,48,2641)
(43,53,2512)
(43,48,2612)
(43,58,2614)
(43,60,2551)
(43,55,2682)
(43,49,2504)
(43,51,2701)
(43,57,2554)
(43,52,2730)
(43,54,2783)
(43,51,2681)
(43,51,2655)
(43,46,2629)
(43,58,2739)
(43,47,2535)
(43,50,2512)
(43,51,2701)
(43,55,2538)
(43,58,2775)
(43,56,2545)
(43,46,2671)
(43,57,2680)
(43,50,2572)
(43,52,2517)
(43,56,2743)
(43,59,2665)
(43,58,2593)
(43,48,2764)
grunt>
```

- Create a new structure from **timesheet_logged** in which we group all of the rows for a given **driverId** as a single tuple. We will end up with one tuple per **driverId**.

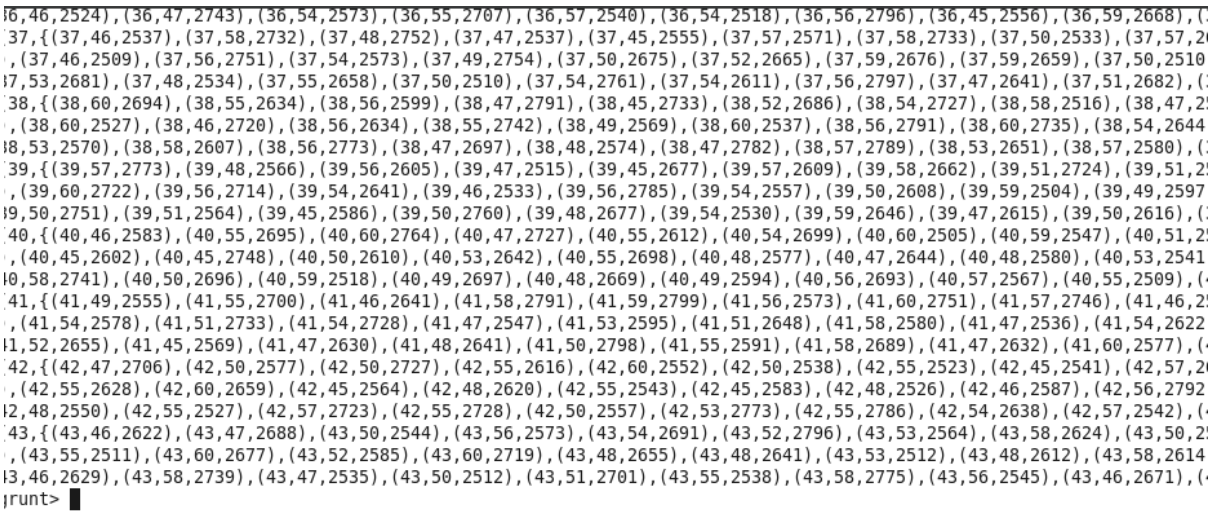
  grunt>*grp_logged=GROUP timesheet_logged by driverId;*

  ```
  grunt> grp_log = group timesheet_log by driverId;
  ```

  Let's take a peek at **grp_logged** to verify our expectation:

grunt>*dump grp_logged;*

```
grunt> dump grp_log;
```



- Now we are ready to sum up the hours and miles in each tuple to create **sum_logged** which has column1as the **driverId**, column 2 as the **sum of hours logged**, and column 3 as the **sum of miles logged.**

  grunt>*sum_logged = FOREACH grp_logged GENERATE group as driverId, SUM(timesheet_logged.hours_logged) as sum_hourslogged, SUM(timesheet_logged.miles_logged) as sum_mileslogged;*

```
grunt> sum_log = foreach grp_log generate group as driverId, SUM(timesheet_log.hours_log) as sum_hourslog, SUM(timesheet_log.miles_log) as sum_mileslog;
```

- Let's take a peek at **sum_logged** to verify our expectation:

  grunt>*dump sum_logged;*

```
(22,2733.0,137550.0)
(23,2750.0,137980.0)
(24,2647.0,134461.0)
(25,2723.0,139180.0)
(26,2730.0,137530.0)
(27,2771.0,137922.0)
(28,2723.0,137469.0)
(29,2760.0,138255.0)
(30,2773.0,137473.0)
(31,2704.0,137057.0)
(32,2736.0,137422.0)
(33,2759.0,139285.0)
(34,2811.0,137728.0)
(35,2728.0,138727.0)
(36,2795.0,138025.0)
(37,2694.0,137223.0)
(38,2760.0,137464.0)
(39,2745.0,138788.0)
(40,2700.0,136931.0)
(41,2723.0,138407.0)
(42,2697.0,136673.0)
(43,2750.0,136993.0)
grunt>
```

- Now we create a new table by joining the **sum_logged** and **drivers_details** tables by matching rows that have the same **driverId**.

grunt>*join_sum_logged = JOIN sum_logged by driverId, drivers_details by driverId;*

```
grunt> join_sum_log = JOIN sum_log by driverId, drivers_detials by driverId;
```

Let's take a peek at **join_sum_logged** to verify our expectation:

grunt>*dump join_sum_logged;*

```
(16,2746.0,137205.0,16,Tom McCuch)
(17,2701.0,135992.0,17,Eric Mizell)
(18,2654.0,137834.0,18,Grant Liu)
(19,2738.0,137968.0,19,Ajay Singh)
(20,2644.0,134564.0,20,Chris Harris)
(21,2751.0,138719.0,21,Jeff Markham)
(22,2733.0,137550.0,22,Nadeem Asghar)
(23,2750.0,137980.0,23,Adam Diaz)
(24,2647.0,134461.0,24,Don Hilborn)
(25,2723.0,139180.0,25,Jean-Philippe Playe)
(26,2730.0,137530.0,26,Michael Aube)
(27,2771.0,137922.0,27,Mark Lochbihler)
(28,2723.0,137469.0,28,Olivier Renault)
(29,2760.0,138255.0,29,Teddy Choi)
(30,2773.0,137473.0,30,Dan Rice)
(31,2704.0,137057.0,31,Rommel Garcia)
(32,2736.0,137422.0,32,Ryan Templeton)
(33,2759.0,139285.0,33,Sridhara Sabbella)
(34,2811.0,137728.0,34,Frank Romano)
(35,2728.0,138727.0,35,Emil Siemes)
(36,2795.0,138025.0,36,Andrew Grande)
(37,2694.0,137223.0,37,Wes Floyd)
(38,2760.0,137464.0,38,Scott Shaw)
(39,2745.0,138788.0,39,David Kaiser)
(40,2700.0,136931.0,40,Nicolas Maillard)
(41,2723.0,138407.0,41,Greg Phillips)
(42,2697.0,136673.0,42,Randy Gelhausen)
(43,2750.0,136993.0,43,Dave Patton)
grunt>
```

- Finally, create a new table called **join_data**, by selecting columns 1, 5, 2, and 3 from the table
  join_sum_logged.

grunt>*join_data = FOREACH join_sum_logged GENERATE $0 as driverId, $4 as name, $1 as hours_logged, $2 as miles_logged;*

```
grunt> join_data = foreach join_sum_log generate $0 as driverId, $4 as name , $1 as hours_log, $2 as miles_log;
```

Finally, we have the overall solution, a table with a row for each driver listing the **driverId, driver name, total hours, and total miles**. Let's take a peek at **join_data** to verify our expectation:

grunt>*dump join_data*

```
(16,Tom McCuch,2746.0,137205.0)
(17,Eric Mizell,2701.0,135992.0)
(18,Grant Liu,2654.0,137834.0)
(19,Ajay Singh,2738.0,137968.0)
(20,Chris Harris,2644.0,134564.0)
(21,Jeff Markham,2751.0,138719.0)
(22,Nadeem Asghar,2733.0,137550.0)
(23,Adam Diaz,2750.0,137980.0)
(24,Don Hilborn,2647.0,134461.0)
(25,Jean-Philippe Playe,2723.0,139180.
(26,Michael Aube,2730.0,137530.0)
(27,Mark Lochbihler,2771.0,137922.0)
(28,Olivier Renault,2723.0,137469.0)
(29,Teddy Choi,2760.0,138255.0)
(30,Dan Rice,2773.0,137473.0)
(31,Rommel Garcia,2704.0,137057.0)
(32,Ryan Templeton,2736.0,137422.0)
(33,Sridhara Sabbella,2759.0,139285.0)
(34,Frank Romano,2811.0,137728.0)
(35,Emil Siemes,2728.0,138727.0)
(36,Andrew Grande,2795.0,138025.0)
(37,Wes Floyd,2694.0,137223.0)
(38,Scott Shaw,2760.0,137464.0)
(39,David Kaiser,2745.0,138788.0)
(40,Nicolas Maillard,2700.0,136931.0)
(41,Greg Phillips,2723.0,138407.0)
(42,Randy Gelhausen,2697.0,136673.0)
(43,Dave Patton,2750.0,136993.0)
grunt>
```

Apache Pig Example

Apache Pig is a tool used to analyze large amounts of data by representing them as data flows. Using the **PigLatin** scripting language operations like ETL (Extract, Transform and Load), data anlaysis and iterative processing can be easily achieved.

Pig is an abstraction over MapReduce. In other words, all Pig scripts internally are converted into Map and Reduce tasks to get the task done. Pig was built to make programming MapReduce applications easier. Before Pig, Java was the only way to process the data stored on HDFS.

Pig was first built in Yahoo! and later became a top level Apache project. In this write up, we will walk through the different features of pig using a sample dataset.

Dataset

The dataset is a simple text (movies_data.csv) file lists movie names and its details like movie-ID, Name-of-movie, release-year, rating and duration**.**

You can download the Dataset from:
https://rksingla.puchd.ac.in/documents/movies_data.csv
https://rksingla.puchd.ac.in/documents/movies_with_duplicate.csv

A sample of the dataset is as follows:

1,The Nightmare Before Christmas,1993,3.9,4568
2,The Mummy,1932,3.5,4388
3,Orphans of the Storm,1921,3.2,9062
4,The Object of Beauty,1991,2.8,6150
5,Night Tide,1963,2.8,5126
6,One Magic Christmas,1985,3.8,5333
7,Muriel's Wedding,1994,3.5,6323
8,Mother's Boys,1994,3.4,5733
9,Nosferatu: Original Version,1929,3.5,5651
10,Nick of Time,1995,3.4,5333
The file has a total of 49590 records.

Pig can be started in one of the following two modes:

- **Local Mode**
- **Cluster/MapReduce Mode**

Using the '**-x local'** options starts pig in the local mode whereas executing the pig command without any options starts in Pig in the cluster mode. When in local mode, pig can access files on the local file system. In cluster mode, pig can access files on HDFS.

Open your terminal and execute the pig command as follows:

- To start in Local Mode:
  $ pig -x local
  grunt>
- To start in Cluster Mode:
  $ pig
  grunt>

```
[cloudera@quickstart ~]$ pig
```

This command presents you with a grunt shell. The grunt shell allows you to execute PigLatin statements to quickly test out data flows on your data step by step without having to execute complete scripts**.**

Pig Latin

To learn Pig Latin, let's question the data. Before we start asking questions, we need the data to be accessible in Pig.

To learn Pig Latin, let's question the data. Let's practice pig using a sample movie dataset. The file has a total of 49590 records. Before we start asking questions, we need the data to be accessible in Pig.

Apache Pig Analysis on Movies Dataset

Introduction

Apache Pig is a high-level platform for creating MapReduce programs used with Hadoop. It provides a scripting language, Pig Latin, which simplifies the process of analyzing large datasets. In this project, we use Apache Pig to perform exploratory data analysis (EDA) on a movies dataset. This dataset typically includes attributes such as movie title, genre, director, release year, rating, and revenue.

Mount the Directory Containing the Dataset

Once you've downloaded the dataset, you'll need to mount the directory where it's stored so that Apache Pig can access it. Here's how you can do it:

1. **Determine the Directory Path**
   Identify the full path to the directory where the dataset is saved. For example:
   `/home/username/datasets/movies/`
2. **Mount the folder**

   Create the folder which contain the both csv files and mount the folder to linux by using super user. After the mounting put the folder in hadoop FS by using –put command.

```
[root@quickstart cloudera]# mount -t vboxsf movies /home/cloudera/meenu/pig
```

Run pig in cluster/ MapReduce mode to start grunt shell and load the data:

Use the following command to load the data:

grunt> ***movies = LOAD '/user/cloudera/lab/pig/movies_data.csv' USING PigStorage(',') as (id, name, year, rating, duration);***

```
grunt> movie = load '/user/cloudera/mee' using PigStorage(',') as (id, name, year, rating, duration);
```

The above statement is made up of two parts. The part to the left of "=" is called the relation or alias. It looks like a variable but you should note that this is not a variable. When this statement is executed, no MapReduce task is executed.

Since our dataset has records with fields separated by a comma we use the keyword USING **PigStorage(',')**. Another thing we have done in the above statement is giving the names to the fields

using the **'as'** keyword.

Now, let's test to see if the alias has the data we loaded.

grunt> ***DUMP movies;***

```
grunt> dump movie;
```

```
(49559,Mitt (Trailer),2013,3.0,138)
(49560,My Hope America with Billy Graham: The Cross,2013,,1706)
(49561,The Square (Trailer),2014,3.6,154)
(49562,My Hope America with Billy Graham,2013,3.9,)
(49563,My Hope America with Billy Graham: Defining Moments,2013,,1791)
(49564,My Hope America with Billy Graham: Lose to Gain,2013,,1400)
(49565,American Addict,2013,3.5,5377)
(49566,My Hope America with Billy Graham,2013,3.9,)
(49567,El Fuente: 2997 MP10,2013,2.7,470)
(49568,El Fuente: 25 MP10,2013,2.9,464)
(49569,El Fuente: 24 MP10,2013,2.8,484)
(49570,El Fuente: 23976 MP10,2013,2.9,484)
(49571,The Short Game (Trailer),2013,4.1,156)
(49572,El Fuente: 5994 MP10,2013,2.8,471)
(49573,El Fuente: 50 MP10,2013,2.9,464)
(49574,El Fuente: 30 MP10,2013,2.8,470)
(49575,Greg Fitzsimmons: Life on Stage,2013,3.3,3671)
(49576,Dave Foley: Relatively Well,2013,3.2,3446)
(49577,Barbie: Life in the Dreamhouse: Barbie Life in the Dreamhouse: Best of Family,2013
(49578,Barbie: Life in the Dreamhouse: Barbie Life in the Dreamhouse: Best of Friends,201
(49579,Transformers Prime Beast Hunters: Predacons Rising,2013,4.2,3950)
(49580,Underground: The Julian Assange Story,2012,3.7,5665)
(49581,Curious George: A Very Monkey Christmas,2009,3.8,3438)
(49582,Mumfie's White Christmas,1996,2.4,1350)
(49583,Lady Gaga &#38; The Muppets' Holiday Spectacular,2013,3.1,3496)
(49584,Sunset Strip,2012,3.0,5770)
(49585,Silver Bells,2013,3.5,5287)
(49586,Winter Wonderland,2013,2.8,1812)
(49587,Top Gear: Series 19: Africa Special,2013,,6822)
(49588,Fireplace For Your Home: Crackling Fireplace with Music,2010,,3610)
(49589,Kate Plus Ei8ht,2010,2.7,)
(49590,Kate Plus Ei8ht: Season 1,2010,2.7,)
```

We will see lot of text on the screen. It is only after the DUMP statement that a Map-Reduce job is initiated. As we see our data in the output, we confirm that the data has been loaded successfully.

Now, since we have the data in Pig, let's start with the questions.

- ***List the movies that having a rating greater than 4***

grunt> ***movies_greater_than_four = FILTER movies BY (float)rating>4.0;***

```
grunt> dump movies_greater_than_four;
```

movies_greater_than_four = FILTER movies BY rating > 4.0;

- **FILTER**: This command is used to select only those records (rows) from a relation (dataset) that meet a certain condition.
- **`movies`**: This is the original relation (dataset) you previously loaded. It contains records of movies with various fields like `movie_id`, `title`, `genre`, `director`, `release_year`, `rating`, and `revenue_millions`.
- **`rating > 4.0`**: The condition used to filter the movies. Only those movies with a rating strictly greater than 4.0 will be selected.

grunt> *DUMP movies_greater_than_four;*

```
(48436,Beyblade: Metal Fury,2012,4.2,)
(48802,Warren Buffett: Bloomberg Game Changers,2013,4.1,2843)
(48829,Video Game High School,2012,4.2,)
(48839,Video Game High School: Season 2,2013,4.2,)
(48846,Gator Boys,2011,4.1,)
(48850,Video Game High School: Season 1,2012,4.2,)
(48855,Pit Bulls &#38; Parolees,2009,4.3,)
(48860,Too Cute!,2011,4.3,)
(48863,Aziz Ansari: Buried Alive,2013,4.1,4780)
(48867,Alaska: The Last Frontier,2011,4.1,)
(48875,Brew Masters,2010,4.1,)
(49026,Cake Boss: Next Great Baker,2010,4.1,)
(49154,Gator Boys: Season 2,2012,4.1,)
(49194,Stephen Hawking's Grand Design: Season 2,2012,4.1,)
(49316,Aziz Ansari: Buried Alive (Trailer),2013,4.1,105)
(49327,Top Gear: Series 19,2013,4.2,)
(49383,Stephen Hawking's Grand Design,2012,4.1,)
(49486,Max Steel: Season 1,2013,4.1,)
(49504,Lilyhammer: Season 2 (Trailer),2013,4.5,106)
(49505,Life With Boys,2011,4.1,)
(49546,Bo Burnham: what.,2013,4.1,3614)
(49549,Life With Boys: Season 1,2011,4.1,)
(49554,Max Steel,2013,4.1,)
(49556,Lilyhammer: Season 1 (Recap),2013,4.2,194)
(49571,The Short Game (Trailer),2013,4.1,156)
(49579,Transformers Prime Beast Hunters: Predacons Rising,2013,4.2,3950)
grunt>
```

The above statement filters the alias *movies* and stores the results in a new alias ***movies_greater_than_four***. The *movies_greater_than_four* alias will have only records of movies where the rating is greater than 4.

The DUMP command is only used to display information onto the standard output. If you need to store the data to a file you can use the following command**:**

grunt> *store movies_greater_than_four into '/user/cloudera/lab/pig/movies_greater_than_four';*

```
grunt> store movies_greater_than_four into '/user/cloudera/mee/movies_greater_than
_four';
```

STORE

- This command saves (writes) the contents of a relation to a file or directory in the Hadoop Distributed File System (HDFS).
- It's like exporting your data from Pig to HDFS, so you can use it later or with other tools.

Let's have a quick look at the FILTER command:

grunt> *movies_greater_than_four = FILTER movies BY (float)rating>4.0;*

Here, we see a (float) keyword placed before the column 'rating'. This is done to tell Pig that the column we are working on is of type, float. Pig was not informed about the type of the column when the data was loaded.

Following is the command we used to load the data**:**

grunt> *movies = LOAD '/user/cloudera/lab/pig/movies_data.csv' USING PigStorage(',') as (id, name, year, rating, duration);*

Here, the load command specified only the column names. We can modify the statement as follows to include the data type of the columns:

grunt> *movies = LOAD /user/cloudera/lab/pig/movies_data.csv' USING PigStorage(',') as (id:int, name:chararray, year:int, rating:double, duration:int);*

In the above statement, name is chararray (string), rating is a double and fields such as id, year and duration are integers. If the data was loaded using the above statement, we would not need to cast the column during filtering. The data types used in the above statement are called scalar data types. The other scalar types are long, double and bytearray.

To get better at using filters, let's ask the data a few more questions:

- *List the movies that were released between 1950 and 1960*

grunt> movies*_between_50_60 = FILTER movies by year>1950 and year<1960;*

```
grunt> movies_starting_between_50_60 = FILTER movie by year> 1950 and year<1960;
```

```
(29776,Felix the Cat: Golden Anniversary Edition,1958,3.7,)
(29792,Crusade in the Pacific: America at War,1951,3.6,)
(35562,Leave It to Beaver: Season 1: Lonesome Beaver,1957,,1546)
(35563,Leave It to Beaver: Season 1: The Perfect Father,1957,,1553)
(35566,Leave It to Beaver: Season 2: Beaver's Hero,1958,,1558)
(35567,Leave It to Beaver: Season 3: Beaver's Fortune,1959,,1498)
(35568,Leave It to Beaver: Season 1: The Broken Window,1957,,1548)
(35569,Leave It to Beaver: Season 1: Train Trip,1957,,1547)
(35570,Leave It to Beaver: Season 3: Wally's Play,1959,,1533)
(35572,Leave It to Beaver: Season 3: Pet Fair,1959,,1556)
(35573,Leave It to Beaver: Season 3: Beaver and Violet,1959,,1521)
(35616,Alfred Hitchcock Presents: Season 3: The Glass Eye,1957,,1543)
(35617,Alfred Hitchcock Presents: Season 3: Heart of Gold,1957,,1570)
(35618,Alfred Hitchcock Presents: Season 3: The Deadly,1957,,1553)
(35619,Alfred Hitchcock Presents: Season 3: Miss Paisley's Cat,1957,,1519)
(35620,Alfred Hitchcock Presents: Season 3: The Percentage,1957,,1554)
(35621,Alfred Hitchcock Presents: Season 3: Together,1957,,1527)
(35622,Alfred Hitchcock Presents: Season 3: Miss Bracegirdle Does Her Duty,1957,,1556)
(35623,Alfred Hitchcock Presents: Season 3: On the Nose,1957,,1564)
(35624,Alfred Hitchcock Presents: Season 3: The Right Kind of House,1957,,1565)
(35625,Alfred Hitchcock Presents: Season 3: Foghorn,1957,,1516)
(35626,Alfred Hitchcock Presents: Season 3: Death Sentence,1957,,1563)
(35627,Alfred Hitchcock Presents: Season 3: The Crocodile Case,1957,,1566)
(35628,Alfred Hitchcock Presents: Season 3: The Canary Sedan,1957,,1570)
(35635,Alfred Hitchcock Presents: Season 3: Impromptu Murder,1957,,1553)
(44641,Isn't Life Wonderful!,1954,2.9,4788)
(44723,Disney Animation Collection: Vol. 4: The Tortoise and the Hare: Paul Bunyan,1958,,1020)
(44767,Disney Animation Collection: Vol. 2: Three Little Pigs: Lambert the Sheepish Lion,1952,,494)
```

- *List the movies that start with the Alpahbet A*

grunt> *movies_starting_with_A = FILTER movies by name matches 'A.*';*

```
grunt> movies_starting_with_A = filter movie by name matches 'A.*'
```

```
(49002,Auction Kings: Season 3: Redneck Chariot Japanese Antique Daggers,2012,,
5)
(49003,Auction Kings: Season 3: Wild West Memorabilia NFL Helmet,2012,,1314)
(49031,A Magic Puppy,2012,3.3,4825)
(49056,A Country Christmas,2013,3.8,5463)
(49057,Another Zero in the System,2013,3.0,4960)
(49227,Alaska: Ice Cold Killers: Season 1,2012,3.6,)
(49269,Alaska: The Last Frontier: Season 1: Something's Fishy,2011,,2620)
(49270,Alaska: The Last Frontier: Season 1: Fall Feast,2011,,2620)
(49271,Alaska: The Last Frontier: Season 1: Spring Has Sprung,2011,,2619)
(49272,Alaska: The Last Frontier: Season 1: Cattle Drive,2011,,2620)
(49273,Alaska: The Last Frontier: Season 1: Range Riding,2011,,2620)
(49274,Alaska: The Last Frontier: Season 1: The River Wild,2011,,2619)
(49275,Alaska: The Last Frontier: Season 1: Before the Freeze,2011,,2558)
(49276,Alaska: The Last Frontier: Season 1: Fueling the Fire,2011,,2619)
(49277,Alaska: The Last Frontier: Season 1: Snow  Cold and Darkness,2011,,2556)
(49278,Alaska: The Last Frontier: Season 1: Dead of Winter,2011,,2560)
(49279,Alaska: Ice Cold Killers: Season 1: Frozen Terror,2012,,2619)
(49280,Alaska: Ice Cold Killers: Season 1: Mountain Man,2012,,2619)
(49281,Alaska: Ice Cold Killers: Season 1: Hunting Humans,2012,,2619)
(49316,Aziz Ansari: Buried Alive (Trailer),2013,4.1,105)
(49322,Al Murray: The Pub Landlord Live - The Only Way Is Epic,2012,3.0,3986)
(49381,Alaska: Ice Cold Killers,2012,3.6,)
(49393,Arnez J: Racially Motivated,2013,3.9,3552)
(49395,A Liga,2010,3.8,)
(49565,American Addict,2013,3.5,5377)
```

- *List the movies that have duration greater than 2 hours*

grunt> *movies_duration_2_hrs = FILTER movies by duration > 7200;*

```
grunt> movies_duration_2_hrs = filter movie by duration > 7200;
```

```
(42646,56 Up,2012,4.0,8617)
(42668,Die Nibelungen: Kriemhild's Revenge,1924,3.1,7878)
(42669,Siegfried,1924,3.2,9005)
(42877,The Tower,2012,3.6,7302)
(42989,Molly's Girl,2012,2.7,7354)
(43228,Day of the Falcon,2011,3.8,7810)
(43336,We Steal Secrets: The Story of WikiLeaks,2013,3.9,7787)
(44551,WWE: The Best of Raw &amp;#38; SmackDown 2012: Vol. 2,2012,,8036)
(44552,WWE: The Best of Raw &amp;#38; SmackDown 2012: Vol. 3,2012,,10664)
(44554,WWE: The Best of Raw &amp;#38; SmackDown 2012: Vol. 1,2012,,7773)
(44562,New World,2013,3.9,8113)
(44599,Perlasca,2002,3.9,7455)
(44657,Makers Our Story,2011,1.6,8162)
(45030,Salinger,2013,4.0,7425)
(45076,The Horde,2012,3.0,7776)
(45106,Elephants Dream 4 Hour,2006,3.1,11780)
(45108,Settai,2013,2.8,7390)
(46832,WWE for All Mankind: The Life &amp;#38; Career of Mick Foley,2013,4.2,8066)
(46949,Example Short XII 23976 2 Hour,2010,2.6,7370)
(48714,WWE: The Top 25 Rivalries in Wrestling History,2013,3.7,10228)
(49055,Theeya Velai Seyyanum Kumaru,2013,3.3,8239)
```

- *List the movies that have rating between 3 and 4*

grunt> *movies_rating_3_4 = FILTER movies BY rating>3.0 and rating<4.0;*

```
grunt> movies_rating_3_4 = filter movie by rating >3.0 and rating <4.0;
```

```
(48436,Beyblade: Metal Fury,2012,4.2,)
(48802,Warren Buffett: Bloomberg Game Changers,2013,4.1,2843)
(48829,Video Game High School,2012,4.2,)
(48839,Video Game High School: Season 2,2013,4.2,)
(48846,Gator Boys,2011,4.1,)
(48850,Video Game High School: Season 1,2012,4.2,)
(48855,Pit Bulls &amp;#38; Parolees,2009,4.3,)
(48860,Too Cute!,2011,4.3,)
(48863,Aziz Ansari: Buried Alive,2013,4.1,4780)
(48867,Alaska: The Last Frontier,2011,4.1,)
(48875,Brew Masters,2010,4.1,)
(49026,Cake Boss: Next Great Baker,2010,4.1,)
(49154,Gator Boys: Season 2,2012,4.1,)
(49194,Stephen Hawking's Grand Design: Season 2,2012,4.1,)
(49316,Aziz Ansari: Buried Alive (Trailer),2013,4.1,105)
(49327,Top Gear: Series 19,2013,4.2,)
(49383,Stephen Hawking's Grand Design,2012,4.1,)
(49486,Max Steel: Season 1,2013,4.1,)
(49504,Lilyhammer: Season 2 (Trailer),2013,4.5,106)
(49505,Life With Boys,2011,4.1,)
(49546,Bo Burnham: what.,2013,4.1,3614)
(49549,Life With Boys: Season 1,2011,4.1,)
(49554,Max Steel,2013,4.1,)
(49556,Lilyhammer: Season 1 (Recap),2013,4.2,194)
(49571,The Short Game (Trailer),2013,4.1,156)
(49579,Transformers Prime Beast Hunters: Predacons Rising,2013,4.2,3950)
```

Let us try few more commands**:**

1. ***DESCRIBE***: The schema of a relation/alias can be viewed using the DESCRIBE command.

grunt> *DESCRIBE movies;*
movies: {id: int, name: chararray, year: int, rating: double, duration: int}

```
grunt> describe movie;
movie: {id: bytearray,name: bytearray,year: bytearray,rating: bytearray,duration:
bytearray}
```

2. *ILLUSTRATE*: To view the step-by-step execution of a sequence of statements you can use the ILLUSTRATE command.

grunt> *ILLUSTRATE movies_duration_2_hrs;*

```
| movie     | id:bytearray     | name:bytearray          | year:bytearray     | ra
ting:bytearray     | duration:bytearray     |
-------------------------------------------------------------------------------
-------------------------------------------
|           | 975              | Possession              | 2002               | 3.
3                  | 6120                   |
|           | 6934             | Thoda Pyaar Thoda Magic | 2008               | 3.
7                  | 7881                   |
-------------------------------------------------------------------------------
-------------------------------------------
-------------------------------------------------------------------------------
-------------------------------------------------------------
| movies_duration_2_hrs     | id:bytearray     | name:bytearray          | year:by
tearray     | rating:bytearray     | duration:bytearray     |
-------------------------------------------------------------------------------
-------------------------------------------------------------
|                           | 6934             | Thoda Pyaar Thoda Magic | 2008
            | 3.7                  | 7881                   |
-------------------------------------------------------------------------------
-------------------------------------------------------------
```

DESCRIBE and *ILLUSTRATE* are really useful for debugging.

Complex Types

Pig supports three different complex types to handle data, namely, tuples, bags and maps. It is important that you understand these types properly as they will be used very often when working with data.

Tuples: A tuple is just like a row in a **table. It is comma separated list of fields.**

(49539, 'The Magic Crystal', 2013, 3.7, 4561)
The above tuple has five fields. A tuple is surrounded by brackets.

Bags**:** A bag is an unordered collection of tuples.

{ (49382, 'Final Offer'), (49385, 'Delete') }
The above bag is has two tuples. Each tuple has two fields, Id and movie name.

Maps**: A** map is a **<key, value> store**. The key and value are joined together using **#.**

['name'#'The Magic Crystal', 'year'#2013]
The above map has two keys, namely, name and year and have values 'The Magic Crystal' and 2013. The first value is a chararray and the second one is an integer.

We will be using the above complex type quite often in our future examples.

3. ***FOREACH:*** FOREACH gives a simple way to apply transformations based on columns. Let's understand this with an example**.**

- ***List the movie names and their duration in minutes***

grunt> ***movie_duration = FOREACH movies GENERATE name, (double)(duration/60);***

```
grunt> movie_duration = foreach movie generate name, (double)(duration/60);
```

The above statement generates a new alias that has the list of movies and their duration in minutes.
You can check the results using the DUMP command:
dump movie_duration;

```
(El Fuente: 2997 MP10,7.0)
(El Fuente: 25 MP10,7.0)
(El Fuente: 24 MP10,8.0)
(El Fuente: 23976 MP10,8.0)
(The Short Game (Trailer),2.0)
(El Fuente: 5994 MP10,7.0)
(El Fuente: 50 MP10,7.0)
(El Fuente: 30 MP10,7.0)
(Greg Fitzsimmons: Life on Stage,61.0)
(Dave Foley: Relatively Well,57.0)
(Barbie: Life in the Dreamhouse: Barbie Life in the Dreamhouse: Best of Family,23.
0)
(Barbie: Life in the Dreamhouse: Barbie Life in the Dreamhouse: Best of Friends,24
.0)
(Transformers Prime Beast Hunters: Predacons Rising,65.0)
(Underground: The Julian Assange Story,94.0)
(Curious George: A Very Monkey Christmas,57.0)
(Mumfie's White Christmas,22.0)
(Lady Gaga &#38; The Muppets' Holiday Spectacular,58.0)
(Sunset Strip,96.0)
(Silver Bells,88.0)
(Winter Wonderland,30.0)
(Top Gear: Series 19: Africa Special,113.0)
(Fireplace For Your Home: Crackling Fireplace with Music 60.0)
```

4. ***GROUP:*** The GROUP keyword is used to group fields in a relation.

- ***List the years and the number of movies released each year.***

grunt> ***grouped_by_year = group movies by year;***

```
grunt> grouped_by_year = group movie by year;
```

grunt> ***count_by_year = FOREACH grouped_by_year GENERATE group, COUNT(movies);***

```
grunt> count_by_year = foreach grouped_by_year GENERATE group, COUNT(movie);
```

You can check the results by dumping the count_by_year relation on the screen.

We know in advance that the total number of movies in the dataset is 49590. We can check to see if our GROUP operation is correct by verifying the total of the COUNT field. If he sum of the count field is 49590, we can be confident that our grouping has worked correctly.

grunt> ***group_all = GROUP count_by_year ALL;***

```
grunt> group_all = group count_by_year ALL;
```

grunt> ***sum_all = FOREACH group_all GENERATE SUM(count_by_year.$1);***

```
grunt> sum_all = foreach group_all generate SUM(count_by_year.$1);
```

grunt> ***DUMP sum_all;***

```
(49590)
grunt>
```

From the above three statements, the first statement, *GROUP ALL*, groups all the tuples to one group. This is very useful when we need to perform aggregation operations on the entire set.
The next statement, performs a FOREACH on the grouped relation *group_all* and applies the SUM function to the field in position 1 (positions start from 0). Here field in position 1, are the counts of movies for each year. One execution of the DUMP statement the MapReduce program kicks off and gives us the following result:

(49590)
The above value matches to our known fact that the dataset has 49590 movies. So we can conclude that our GROUP operation worked successfully**.**

5. ***ORDER BY:*** Let us question the data to illustrate the ORDER BY operation.

- ***List all the movies in the ascending order of year.***

grunt> ***asc_movies_by_year = ORDER movies BY year ASC;***

```
grunt> asc_movies_by_year = order movie by year asc;
```

grunt> ***DUMP asc_movies_by_year;***

```
(46990,The Adventures of Fish 'N Chips: Totems  Truces &#38; Three Yellow Hairs,20
13,,3259)
(44442,Home Sweet Home,2013,3.1,4892)
(44443,Infected,2013,2.7,5698)
(49554,Max Steel,2013,4.1,)
(44444,The Book of Esther,2013,3.9,5365)
(44445,Walter Latham's Comedy After Dark,2013,3.5,3605)
(49555,Saving Santa,2013,3.8,5038)
(44486,Craig Ferguson: I'm Here to Help,2013,3.9,4864)
(39342,Chennai Express,2013,3.6,8443)
(37174,The Following: Season 1,2013,4.1,)
(38307,House of Cards: Season 1: Chapter 10,2013,,3133)
(45573,Caillou: Season 5: Big Time Caillou &#38; Other Stories: Caillou Explores,2
013,,1551)
(38308,House of Cards: Season 1: Chapter 11,2013,,3231)
(45575,Caillou: Season 5: Big Time Caillou &#38; Other Stories: Blast Off to Space
,2013,,1551)
(44561,InAPPropriate Comedy,2013,2.6,5020)
(49561,The Square (Trailer),2014,3.6,154)
```

- ***List all the movies in the descending order of year.***

grunt> ***desc_movies_by_year = ORDER movies by year DESC;***

```
grunt> desc_movies_by_year = Order movie by year DESC;
```

grunt> ***DUMP desc_movies_by_year;***

```
20060,Charlie Chaplin Collection: Shorts: One A.M.   The Pawnshop,1914,,2712)
20057,Charlie Chaplin Collection: Shorts: By the Sea   His Regeneration   Mabel a
20062,Charlie Chaplin Collection: Shorts: A Woman   Rival Mashers,1914,,3060)
20063,Charlie Chaplin Collection: Shorts: Shanghaied   Triple Trouble,1914,,3013)
20064,Charlie Chaplin Collection: Shorts: The Tramp   Police,1914,,3119)
20065,Charlie Chaplin Collection: Shorts: The Fireman  The Adventurer,1914,,2842)
20066,Charlie Chaplin Collection: Shorts: The Count   The Cure,1914,,2862)
29801,Charlie Chaplin Collection,1914,3.8,)
20067,Charlie Chaplin Collection: Shorts,1914,3.8,)
20068,Charlie Chaplin Collection: Shorts: The Rink   The Floorwalker,1914,,2851)
20061,Charlie Chaplin Collection: Shorts: Work   In The Park  Good For Nothing,19
20056,Charlie Chaplin Collection: Shorts: Behind the Screen   The Knockout   Betw
42672,Fantômas IV: Fantômas vs. Fantômas,1914,2.5,3590)
20055,Charlie Chaplin Collection: Shorts: His New Job   A Jitney Elopement,1914,,
20054,Charlie Chaplin Collection: Shorts: The Immigrant   Easy Street,1914,,2845)
20053,Charlie Chaplin Collection: Shorts: The Champion   The Vagabond,1914,,3400)
610,Cabiria,1914,2.9,7684)
20058,Charlie Chaplin Collection: Shorts: A Night In the Show    The Bank,1914,,2
42673,Fantômas V: The False Magistrate,1914,2.4,4247)
14328,Fantômas III: The Murderous Corpse,1913,2.6,5432)
42665,Fantômas I: In the Shadow of the Guillotine,1913,2.9,3268)
42671,Fantômas II: Juve vs. Fantômas,1913,2.7,3718)
grunt>
```

6. ***DISTINCT:*** The DISTINCT statement is used to remove duplicate records. It works only on entire records, not on individual fields. Let's illustrate this with an example**:**

grunt> ***movies_with_dups = LOAD '/user/cloudera/lab/pig/movies_with_duplicate.csv' USING PigStorage(',') as (id:int, name:chararray, year:int, rating:double, duration:int);***

```
grunt> mov_with_dups =LOAD '/user/cloudera/dup' using PigStorage(',') as (id:int,
name:chararray, year:int, rating:double, duration:int);
```

grunt> ***DUMP movies_with_dups;***

(1,The Nightmare Before Christmas,1993,3.9,4568)
(1,The Nightmare Before Christmas,1993,3.9,4568)
(1,The Nightmare Before Christmas,1993,3.9,4568)
(2,The Mummy,1932,3.5,4388)
(3,Orphans of the Storm,1921,3.2,9062)
(4,The Object of Beauty,1991,2.8,6150)
(5,Night Tide,1963,2.8,5126)
(5,Night Tide,1963,2.8,5126)
(5,Night Tide,1963,2.8,5126)

(6,One Magic Christmas,1985,3.8,5333)
(7,Muriel's Wedding,1994,3.5,6323)
(8,Mother's Boys,1994,3.4,5733)
(9,Nosferatu: Original Version,1929,3.5,5651)
(10,Nick of Time,1995,3.4,5333)
(9,Nosferatu: Original Version,1929,3.5,5651)

```
(1,The Nightmare Before Christmas,1993,3.9,4568)
(1,The Nightmare Before Christmas,1993,3.9,4568)
(1,The Nightmare Before Christmas,1993,3.9,4568)
(2,The Mummy,1932,3.5,4388)
(3,Orphans of the Storm,1921,3.2,9062)
(4,The Object of Beauty,1991,2.8,6150)
(5,Night Tide,1963,2.8,5126)
(5,Night Tide,1963,2.8,5126)
(5,Night Tide,1963,2.8,5126)
(6,One Magic Christmas,1985,3.8,5333)
(7,Muriel's Wedding,1994,3.5,6323)
(8,Mother's Boys,1994,3.4,5733)
(9,Nosferatu: Original Version,1929,3.5,5651)
(10,Nick of Time,1995,3.4,5333)
(9,Nosferatu: Original Version,1929,3.5,5651)
grunt>
```

We see that there are duplicates in this data set. Now let us list the distinct records present in ***movies_with_dups***:

grunt> ***no_dups = DISTINCT movies_with_dups;***

```
grunt> no_dups= DISTINCT mov_with_dups;
```

grunt> ***DUMP no_dups;***

(1,The Nightmare Before Christmas,1993,3.9,4568)
(2,The Mummy,1932,3.5,4388)
(3,Orphans of the Storm,1921,3.2,9062)
(4,The Object of Beauty,1991,2.8,6150)
(5,Night Tide,1963,2.8,5126)
(6,One Magic Christmas,1985,3.8,5333)
(7,Muriel's Wedding,1994,3.5,6323)
(8,Mother's Boys,1994,3.4,5733)
(9,Nosferatu: Original Version,1929,3.5,5651)
(10,Nick of Time,1995,3.4,5333)

```
(1,The Nightmare Before Christmas,1993,3.9,4568)
(2,The Mummy,1932,3.5,4388)
(3,Orphans of the Storm,1921,3.2,9062)
(4,The Object of Beauty,1991,2.8,6150)
(5,Night Tide,1963,2.8,5126)
(6,One Magic Christmas,1985,3.8,5333)
(7,Muriel's Wedding,1994,3.5,6323)
(8,Mother's Boys,1994,3.4,5733)
(9,Nosferatu: Original Version,1929,3.5,5651)
(10,Nick of Time,1995,3.4,5333)
```

7. ***LIMIT:*** Use the LIMIT keyword to get only a limited number for results from relation.

grunt> ***top_10_movies = LIMIT movies 10;***

grunt> ***DUMP top_10_movies;***

```
grunt> top_10_movie = limit movie 10;
grunt> dump top_10_movie;
```

(1,The Nightmare Before Christmas,1993,3.9,4568)
(2,The Mummy,1932,3.5,4388)
(3,Orphans of the Storm,1921,3.2,9062)
(4,The Object of Beauty,1991,2.8,6150)
(5,Night Tide,1963,2.8,5126)
(6,One Magic Christmas,1985,3.8,5333)
(7,Muriel's Wedding,1994,3.5,6323)
(8,Mother's Boys,1994,3.4,5733)
(9,Nosferatu: Original Version,1929,3.5,5651)
(10,Nick of Time,1995,3.4,5333)

```
(1,The Nightmare Before Christmas,1993,3.9,4568)
(2,The Mummy,1932,3.5,4388)
(3,Orphans of the Storm,1921,3.2,9062)
(4,The Object of Beauty,1991,2.8,6150)
(5,Night Tide,1963,2.8,5126)
(6,One Magic Christmas,1985,3.8,5333)
(7,Muriel's Wedding,1994,3.5,6323)
(8,Mother's Boys,1994,3.4,5733)
(9,Nosferatu: Original Version,1929,3.5,5651)
(10,Nick of Time,1995,3.4,5333)
```

8. ***SAMPLE:*** Use the sample keyword to get sample set from your data.

grunt> ***sample_10_percent = sample movies 0.1;***

grunt> ***dump sample_10_percent;***

```
grunt> sample_10_percent = sample movie 0.1;
grunt> dump sample_10_percent;
```

Here, 0.1 = 10%. As we already know that the file has 49590 records. We can check to see the count of records in the relation.

```
(49406,Mexican Fighter,2013,3.7,3684)
(49423,Casper's Scare School: Season 2,2011,3.5,)
(49429,The Cleveland Show: Season 4: Grave Danger,2012,,1297)
(49431,Casper's Scare School: Season 2: Dream Team Curse of the Ring,2011,,1401)
(49457,Glee: Season 4: Glee  Actually,2012,,2639)
(49467,Casper's Scare School: Season 2: Jack Out of the Box Casper Meets Super Cho
c,2011,,1401)
(49468,Casper's Scare School: Season 2: Frankengymteacher's Monster Radio Blodge,2
011,,1401)
(49473,Glee: Season 4: The Role You Were Born to Play,2012,,2639)
(49483,Glee: Season 4: Thanksgiving,2012,,2639)
(49484,Prison Girls,1972,3.0,5681)
(49492,Max Steel: Season 1: C.Y.T.R.O. Attacks!,2013,,1323)
(49497,Max Steel: Season 1: Supermania,2013,,1323)
(49515,Female Convict Scorpion,2008,2.8,6051)
(49537,Karate Girl,2011,3.4,5505)
(49542,Life With Boys: Season 1: In the Principal's Office with Boys,2011,,1368)
(49543,El Fuente: 5994 MP,2013,3.5,471)
(49545,El Fuente: 50 MP,2013,3.3,465)
(49559,Mitt (Trailer),2013,3.0,138)
(49562,My Hope America with Billy Graham,2013,3.9,)
(49564,My Hope America with Billy Graham: Lose to Gain,2013,,1400)
```

grunt> ***sample_group_all = GROUP sample_10_percent ALL;***
grunt> ***sample_count = FOREACH sample_group_all GENERATE COUNT(sample_10_percent.$0);***
grunt> ***dump sample_count;***

```
grunt> sample_group_all = group sample_10_percent ALL;
grunt> sample_count = foreach sample_group_all generate COUNT(sample_10_percent.$0
);
grunt> dump sample_count;
```

The output is (4937) which is approximately 10% for 49590.

```
(4933)
```

IPL Dataset Analysis using Hadoop Pig

Data source

We downloaded the IPL dataset from Kaggle — a popular platform for datasets in CSV format. The dataset contained two primary files:

- **matches.csv** – Contains match-level information (date, teams, venue, etc.)
- **deliveries.csv** – Contains ball-by-ball data for every IPL match (including bowler, batsman, extras, and type of delivery)

Data transfer process by mounting the data to Hadoop.

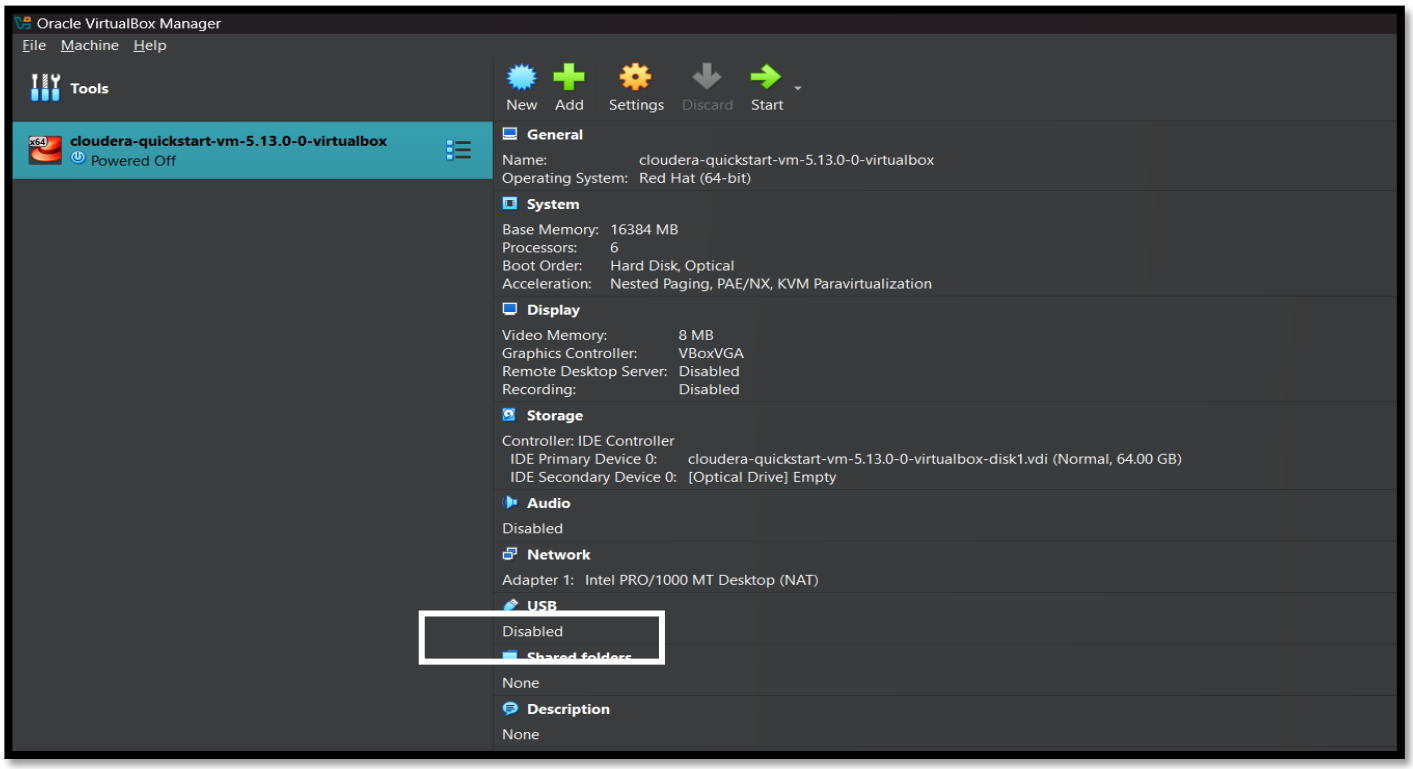
STEP 1

- Create afolder in Host Operating System (Windows)

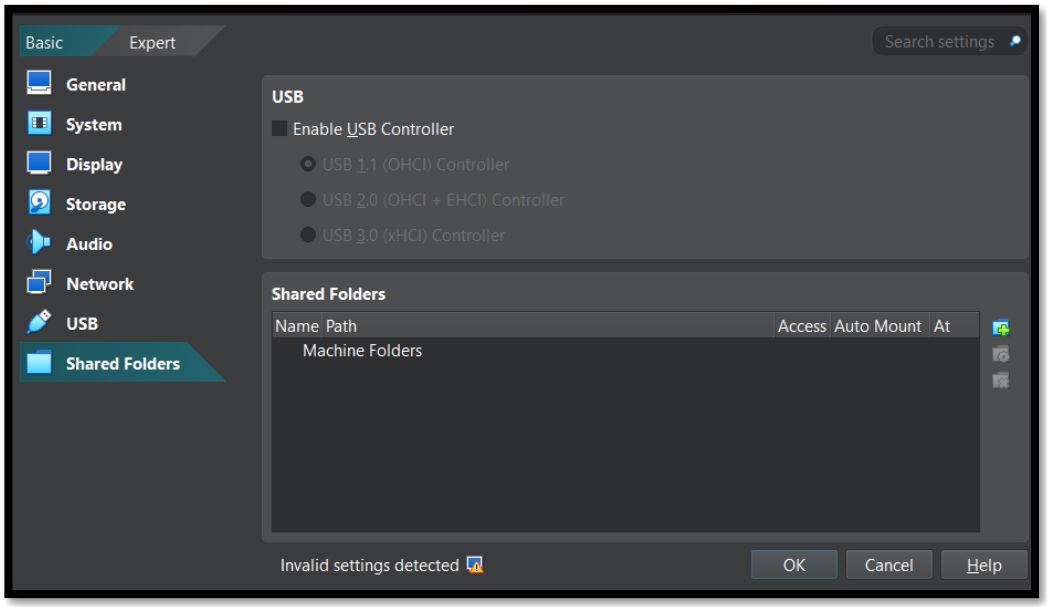
Ctrl +Shift + N >Rename the File or (Press F2).

STEP 2

- Open The **Virtualbox**> Click on **Shared Folder**> Shared Folder Dialog Box Will appear.



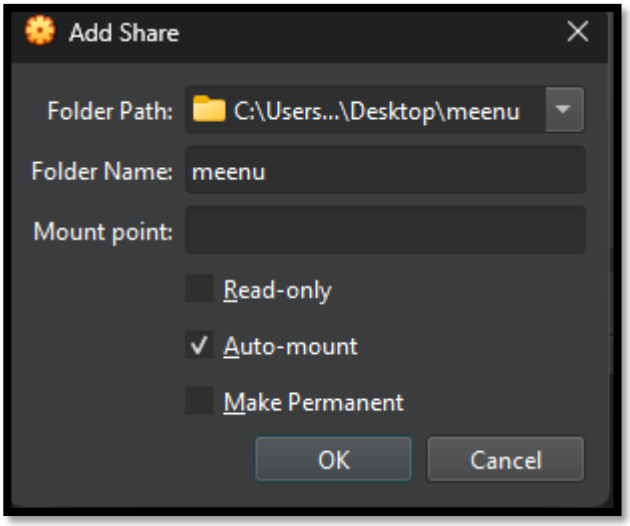
The **dialog box** will appear.



- Click On  Icon.

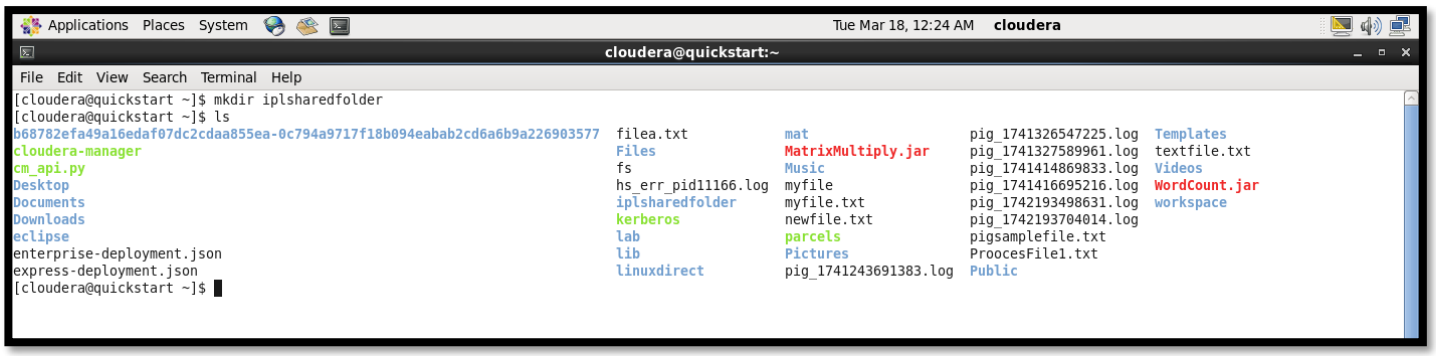
- Select the Folder > Check the **Auto Mount Option**> Click OK.

- Give the path of your folder.



STEP 3

- Start with your **VM**.
- Open the terminal &Create a Directory in Linux System.



STEP 4

- Change to **Super User** for mounting access by **su** command and enter **Password cloudera.**

```
[cloudera@quickstart ~]$ su
Password:
```

- Type The Following Command.

Syntax Of Command

mount -tvboxsf<Folder Name in Windows>< Folder Name In Linux>

```
[root@quickstart cloudera]# mount -t vboxsf meenu /home/cloudera/iplsharedfolder
```

Now you are done with the procedures. As in host operating system your folder name **meenu** with stored csv files matches and deliveries is now shared in your virtual operating system (Linux).

- Open the Mounted folder in Linux named with **iplsharedfolder** and check the files stored in it.

```
[root@quickstart cloudera]# ls /home/cloudera/iplsharedfolder
deliveries.csv  matches.csv
```

STEP 5

- Sharing the mounted folder from (linux) to **Hadoop.**
- Using **-put** or **copyFromLocal** command.

```
root@quickstart cloudera]# hdfs dfs -mkdir ipl
root@quickstart cloudera]# hdfs dfs -put /home/cloudera/iplsharedfolder /user/cloudera/ipl
root@quickstart cloudera]# hdfs dfs -ls /user/cloudera/ipl
ound 2 items
rw-r--r--   1 root cloudera   27019953 2025-03-18 02:34 /user/cloudera/ipl/deliveries.csv
rw-r--r--   1 root cloudera     225266 2025-03-18 02:34 /user/cloudera/ipl/matches.csv
root@quickstart cloudera]#
```

STEP 6

Pig can be started in one of the following two modes:

- **Local Mode**
- **Cluster/MapReduce Mode**

Using the'-x local' options start's pig in the local mode whereas executing the pig command without any options starts in Pig in the cluster mode. When in local mode, pig can access files on the local file system. In cluster mode, pig can access files on HDFS.

- Open your terminal and execute the pig command as follows:

To start in Local Mode: **$ pig -x local**
grunt>

To start in Cluster Mode: **$ pig**
grunt>

This command presents you with a grunt shell. The grunt shell allows you to execute **PigLatin** statements to quickly test

out data flows on your data step by step without having to execute complete scripts.

```
[cloudera@quickstart ~]$ pig
```

- Invoke grunt (the pig interactive environment):

$ pig

grunt>

```
2025-03-27 22:37:36,183 [main] INFO  org.apache.hadoop.conf.Configuration.deprec
ation - fs.default.name is deprecated. Instead, use fs.defaultFS
2025-03-27 22:37:36,183 [main] INFO  org.apache.hadoop.conf.Configuration.deprec
ation - mapred.job.tracker is deprecated. Instead, use mapreduce.jobtracker.addr
ess
2025-03-27 22:37:36,196 [main] INFO  org.apache.hadoop.conf.Configuration.deprec
ation - fs.default.name is deprecated. Instead, use fs.defaultFS
2025-03-27 22:37:36,196 [main] INFO  org.apache.hadoop.conf.Configuration.deprec
ation - mapred.job.tracker is deprecated. Instead, use mapreduce.jobtracker.addr
ess
grunt>
```

STEP 7

QUERY 1

- Use the following command to load the data:

```
grunt> match = load '/user/cloudera/ipl/matches.csv' using PigStorage(',') as(id,season,city,date,match_type,player_of_match,venue,team1,team2,toss_winner,toss_decision,win
ner,result,result_margin,target_runs,target_overs,super_over,method,umpire1,umpire2);
```

```
grunt> deliveries = load '/user/cloudera/ipl/deliveries.csv' using PigStorage(',') as(match_id,inning,batting_team,bowling_team,over,ball,batter,bowler,non_striker,batsman_
runs,extra_runs,total_runs,extras_type,is_wicket,player_dismissed,dismissal_kind,fielder);
```

The above statement is made up of two parts. The part to the left of "=" is called the **relation or alias**. It looks like a variable but you should note that this is not a variable. When this statement is executed, no MapReduce task is executed.

Since our dataset has records with fields separated by a comma we use the keyword **USING PigStorage(',')**. Another thing we have done in the above statement is giving the names to the fields using the **'as'** keyword.

- Now, let's test to see if the alias has the data we loaded**.**

grunt>*DUMP match;*

We will see lot of text on the screen. It is only after the **DUMP** statement that a Map-Reduce job is initiated. As we see our data in the output, we confirm that the data has been loaded successfully.

- Now, let's store the relation in a directory named r.

Using store command.

```
grunt> Store matche into '/user/cloudera/r' using PigStorage(',');
```

```
grunt> Store deliveries into '/user/cloudera/s' using PigStorage(',');
```

STEP 8

Now, since we have the data in Pig. let's start with the questions**.**

QUERY 2

- **NUMBER OF MATCHES WON BY EACH TEAM**.

```
grunt> matches_count = FOREACH matches_group  GENERATE group AS team, COUNT(match) AS wins;
grunt> matches_group = GROUP match  BY winner;
grunt> matches_count = FOREACH matches_group  GENERATE group AS team, COUNT(match) AS wins;
grunt> dump match_count;
```

OUTPUT:

```
(NA,4)
(bat,127)
(field,242)
(winner,1)
(Punjab Kings,9)
(Gujarat Lions,11)
(Pune Warriors,9)
(Delhi Capitals,24)
(Gujarat Titans,3)
(Mumbai Indians,112)
(Deccan Chargers,22)
(Kings XI Punjab,59)
(Delhi Daredevils,59)
(Rajasthan Royals,70)
(Chennai Super Kings,80)
(Sunrisers Hyderabad,52)
(Kochi Tuskers Kerala,6)
(Lucknow Super Giants,3)
(Kolkata Knight Riders,98)
(Rising Pune Supergiant,9)
(Rising Pune Supergiants,4)
(Royal Challengers Bangalore,90)
(Royal Challengers Bengaluru,2)
```

QUERY 3

- **Count total matches per season.**

```
grunt> grouped_season = GROUP matche BY season;
grunt> season_match_count = FOREACH grouped_season GENERATE group AS season, COUNT(matche) AS total_matches;
```

```
grunt> dump sesaon_match_count;
```

OUTPUT:

```
(2009,57)
(2011,73)
(2012,74)
(2013,76)
(2014,60)
(2015,59)
(2016,60)
(2017,59)
(2018,60)
(2019,60)
(2021,60)
(2022,74)
(2023,74)
(2024,71)
(season,1)
(2007/08,58)
(2009/10,60)
(2020/21,60)
```

QUERY 4

- **Most "player of the match" Awards**

```
grunt> grouped_player = GROUP matche BY player_of_match;
```

```
grunt> player_awards = FOREACH grouped_player GENERATE group  AS player, COUNT(matche) AS awards;
grunt> sorted_awareds = ORDER player_awards BY awards DESC;
grunt> dump sorted_awards;
```

```
(AB de Villiers,25)
(CH Gayle,22)
(RG Sharma,19)
(V Kohli,18)
(DA Warner,18)
(MS Dhoni,17)
(SR Watson,16)
(YK Pathan,16)
(RA Jadeja,16)
(SP Narine,15)
(AD Russell,15)
(SK Raina,14)
(KA Pollard,14)
(KL Rahul,13)
(AM Rahane,13)
(G Gambhir,13)
(JC Buttler,13)
(Rashid Khan,12)
(MEK Hussey,12)
(S Dhawan,12)
(A Mishra,12)
(SV Samson,11)
(RD Gaikwad,11)
(DR Smith,11)
(V Sehwag,11)
(Shubman Gill,10)
(AT Rayudu,10)
(F du Plessis,10)
(UT Yadav,10)
(JH Kallis,10)
(JJ Bumrah,10)
(SE Marsh,9)
(GJ Maxwell,9)
(MP Stoinis,9)
(SA Yadav,9)
(Harbhajan Singh,8)
(AR Patel,8)
(HH Pandya,8)
(RR Pant,8)
(SR Tendulkar,8)
(AC Gilchrist,7)
(Kuldeep Yadav,7)
```

QUERY 5

- **Total runs scored by each batsman.**

```
grunt> grouped_batsman = GROUP deliveries BY batter;
```

```
grunt> batsman_runs = FOREACH grouped_batsman GENERATE group AS batter, SUM(deliveries.batsman_runs) AS total_runs;
grunt> sorted_batsman= ORDER batsman_runs BY total_runs DESC;
```

```
grunt> DUMP sorted_batsman;
```

OUTPUT:

```
(D Wiese,)
(CA Lynn,)
(C Nanda,)
(C Munro,)
(C Madan,)
(C Green,)
(BR Dunk,)
(BB Sran,)
(B Kumar,)
(B Akhil,)
(AS Raut,)
(AP Tare,)
(AP Dole,)
(AD Nath,)
(A Zampa,)
(A Tomar,)
(A Singh,)
(A Nehra,)
(batter,)
```

QUERY 6

- **Which batsman has played how many deliveries.**

```
grunt> grouped_by_batsman = GROUP deliveries BY batsman;
grunt> batsman_deliveries = FOREACH grouped_by_batsman
GENERATE group AS batsman, COUNT(deliveries) AS
deliveries_faced;
grunt> DUMP batsman_deliveries;
```

```
(MV Boucher,318)
(UBT Chand,314)
(JR Hopes,314)
(S Sohal,311)
(Azhar Mahmood,311)
(JEC Franklin,309)
(SP Goswami,307)
(KV Sharma,306)
(P Negi,301)
(Lalit Yadav,301)
(R Bhatia,290)
(R Vinay Kumar,289)
(AP Tare,285)
(KR Mayers,276)
(Anuj Rawat,276)
(DJ Mitchell,274)
(Shashank Singh,272)
(Y Nagar,271)
(VVS Laxman,271)
(DB Das,267)
(N Wadhera,261)
(M Kaif,258)
(B Chipli,256)
(R Powell,254)
(LA Pomersbach,251)
(DJG Sammy,251)
(PK Garg,247)
```

QUERY 7

- **COUNT THE NUMBER OF NO-BALLS BOWLED BY EACH BOWLER IN DESCENDING ORDER.**

```
grunt> grouped_by_bowler = GROUP no_balls BY bowler;
grunt> no_ball_count = FOREACH grouped_by_bowler GENERATE group AS bowler,
COUNT(no_balls) AS total_no_balls;
grunt> ordered_no_balls = ORDER no_ball_count BY total_no_balls DESC;
grunt> DUMP ordered_no_balls;
```

```
(AD Russell,67)
(DR Smith,67)
(KK Nair,66)
(SC Ganguly,65)
(IK Pathan,65)
(SS Tiwary,64)
(NV Ojha,62)
(DJ Hooda,62)
(EJG Morgan,61)
(KH Pandya,59)
(MP Stoinis,55)
(BJ Hodge,54)
(CA Lynn,54)
(AC Gilchrist,53)
(Harbhajan Singh,53)
(DJ Hussey,52)
(R Tewatia,51)
(Tilak Varma,51)
(JA Morkel,50)
(Y Venugopal Rao,50)
(STR Binny,50)
(KC Sangakkara,49)
(KM Jadhav,49)
(SO Hetmyer,49)
(N Pooran,49)
(S Badrinath,48)
(R Parag,48)
(VR Iyer,48)
(S Dube,48)
(M Vohra,47)
```

QUERY 8

- **COUNT THE NUMBER OF WIDE BOWLED BY EACH BOWLER IN DESCENDING ORDER**

```
grunt> grouped_by_bowler = GROUP wide_deliveries BY bowler_name;
grunt> bowler_wide_count = FOREACH grouped_by_bowler GENERATE group AS bowler_name, COUNT(wide_deliveries) AS
wide_count;
grunt> sorted_wides = ORDER bowler_wide_count BY wide_count DESC;
grunt> DUMP sorted_wides
```

```
(Y Venugopal Rao,48)
(R Ashwin,46)
(IK Pathan,46)
(N Pooran,45)
(SP Narine,45)
(HH Gibbs,45)
(M Vohra,44)
(Tilak Varma,44)
(KM Jadhav,44)
(AK Markram,43)
(ML Hayden,42)
(Shakib Al Hasan,42)
(B Sai Sudharsan,41)
(Abhishek Sharma,40)
(R Parag,40)
(MC Henriques,40)
(Shahbaz Ahmed,40)
(RK Singh,39)
(AM Nayar,39)
(JD Ryder,39)
(JA Morkel,38)
(EJG Morgan,38)
(CL White,37)
(LMP Simmons,37)
(CA Lynn,37)
(GC Smith,36)
(LRPL Taylor,36)
(A Badoni,36)
(LS Livingstone,35)
(PP Chawla,35)
(TL Suman,34)
```

QUERY 9

- **FINDING TOTAL EXTRAS GIVEN BY EACH PLAYER**

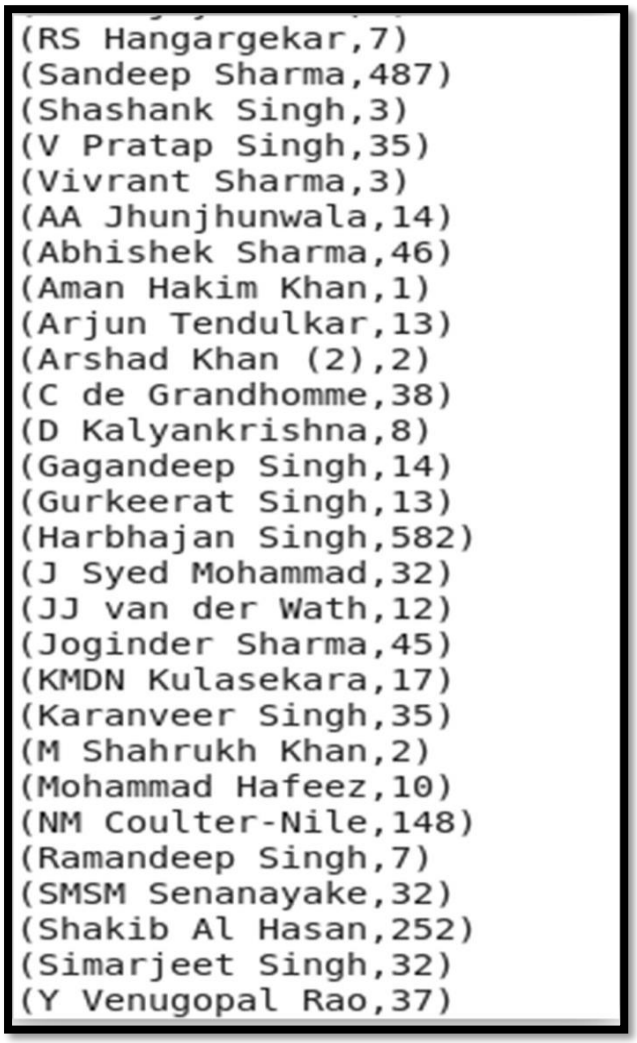
```
grunt> group_extras = GROUP deliveries BY bowler;
grunt> total_extras = FOREACH group_extras GENERATE group AS bowler, SUM(deliveries.extra_runs) AS extras;
grunt> DUMP total_extras;
```

```
(Sandeep Sharma,153)
(Shashank Singh,1)
(V Pratap Singh,12)
(Vivrant Sharma,0)
(AA Jhunjhunwala,1)
(Abhishek Sharma,11)
(Aman Hakim Khan,0)
(Arjun Tendulkar,4)
(Arshad Khan (2),0)
(C de Grandhomme,7)
(D Kalyankrishna,2)
(Gagandeep Singh,3)
(Gurkeerat Singh,5)
(Harbhajan Singh,161)
(J Syed Mohammad,10)
(JJ van der Wath,3)
(Joginder Sharma,13)
(KMDN Kulasekara,6)
(Karanveer Singh,14)
(M Shahrukh Khan,0)
(Mohammad Hafeez,2)
(NM Coulter-Nile,52)
(Ramandeep Singh,6)
(SMSM Senanayake,9)
(Shakib Al Hasan,71)
(Simarjeet Singh,10)
(Y Venugopal Rao,7)
(Fazalhaq Farooqi,6)
(Mashrafe Mortaza,1)
(Mujeeb Ur Rahman,20)
(Mukesh Choudhary,16)
(PM Sarvesh Kumar,1)
(RE van der Merwe,25)
```

QUERY 10

- **NUMBER OF OVERS BOWLED BY EACH PLAYER**

```
grunt> group_bowler = GROUP deliveries BY bowler;
grunt> balls_bowled = FOREACH group_bowler GENERATE group AS bowler,COUNT(deliveries) AS balls;
grunt> overs_bowled = FOREACH balls_bowled GENERATE bowler, balls/6 AS overs;
2025-03-27 22:37:34,094 [main] WARN  org.apache.pig.PigServer - Encountered Warning IMPLICIT_CAST_TO_LONG 1 time(s).
grunt> DUMP overs_bowled;
```


(RS Hangargekar,7)
(Sandeep Sharma,487)
(Shashank Singh,3)
(V Pratap Singh,35)
(Vivrant Sharma,3)
(AA Jhunjhunwala,14)
(Abhishek Sharma,46)
(Aman Hakim Khan,1)
(Arjun Tendulkar,13)
(Arshad Khan (2),2)
(C de Grandhomme,38)
(D Kalyankrishna,8)
(Gagandeep Singh,14)
(Gurkeerat Singh,13)
(Harbhajan Singh,582)
(J Syed Mohammad,32)
(JJ van der Wath,12)
(Joginder Sharma,45)
(KMDN Kulasekara,17)
(Karanveer Singh,35)
(M Shahrukh Khan,2)
(Mohammad Hafeez,10)
(NM Coulter-Nile,148)
(Ramandeep Singh,7)
(SMSM Senanayake,32)
(Shakib Al Hasan,252)
(Simarjeet Singh,32)
(Y Venugopal Rao,37)

Employee Dataset Analysis Using Apache Pig

Objective:

The objective of this assignment is to perform basic analysis on an employee dataset using **Apache Pig**, a high-level platform for creating MapReduce programs used with Hadoop. This includes data loading, filtering, grouping, and summarizing key metrics like average salary by department, gender distribution, and identifying top earners.

Dataset:

- Download csv file that contain the datasets.
- Upload to HDFS

```
[cloudera@quickstart ~]$ su
Password:
[root@quickstart cloudera]# mount -t vboxsf pigemp /home/cloudera/
[root@quickstart cloudera]# ls
b68782efa49a16edaf07dc2cdaa855ea-0c794a9717f18b094eabab2cd6a6b9a226903577  emp_dept                  hive1
cloudera-manager                                                          emp_dept.tar.gz           hs_err_pid1116
cm_api.py                                                                 emp_dept.tar.gz.1         ipl
Desktop                                                                   enterprise-deployment.json iplsharedfolde
Documents                                                                 express-deployment.json   kerberos
Downloads                                                                 filea.txt                 lab
drivers.csv                                                               Files                     labmeenu
eclipse                                                                   fs                        learning.txt
[root@quickstart cloudera]# hdfs dfs -mkdir empdataset
[root@quickstart cloudera]# hdfs dfs -put /home/cloudera/emp_dept /user/cloudera/empdataset
[root@quickstart cloudera]# hdfs dfs -ls /user/cloudera/empdataset
Found 3 items
-rw-r--r--   1 root cloudera         80 2025-04-22 01:36 /user/cloudera/empdataset/dept.csv
-rw-r--r--   1 root cloudera        596 2025-04-22 01:36 /user/cloudera/empdataset/emp.csv
-rw-r--r--   1 root cloudera         59 2025-04-22 01:36 /user/cloudera/empdataset/salgrade.csv
```

Load the Dataset

- Open terminal and enter the pig shell.
- Create the alias named emp and load the csv dataset file.

emp = load '/data/emp_dept/emp.csv' using PigStorage('\t') as (empno: int, ename:chararray, job: chararray, mgr:int, hiredate: datetime, sal:float, deptno:int);

```
grunt> emp = load '/user/cloudera/empdataset/emp.csv' using PigStorage('\t') as (empno: int, ename: chararray, job:chararray, mgr: int , hiredate: datetime, sal: float, deptno : int);
```

dept = load '/data/emp_dept/dept.csv' using PigStorage('\t') as (deptno:int,dname:chararray,loc : chararray);

```
grunt> dep = load '/user/cloudera/empdataset/dept.csv' using  PigStorage('\t') as (deptno: int, danme: chararray,loc:chararray);
```

salgrade = load '/data/emp_dept/salgrade.csv' using PigStorage('\t') as (grade : int , losal: int , hisal:int);

```
grunt> salgrade= load '/user/cloudera/empdataset/salgrade.csv' using PigStorage('\t') as (grade: int, losal: int, hisal:int);
```

Analysis

1. Write down the Apache Pig statement(s) to get Smith's employment date.

smith = FILTER emp BY ename == 'SMITH' ;
hireDate = FOREACH smith GENERATE ename, hiredate ;
DUMP hireDate ;

```
grunt> smith = filter emp by ename =='SMITH';
grunt> hiredate = foreach smith generate ename, hiredate;
grunt> dump hiredate;
```

```
(SMITH,1980-12-17T00:00:00.000-08:00)
```

2. Write down the Apache Pig statement(s) to get Ford's job title.

ford = FILTER emp BY ename == 'FORD' ;

```
grunt> ford = filter emp  by ename=='FORD';
```

fordTitle = FOREACH ford GENERATE ename, job ;
DUMP fordTitle ;

```
grunt> fordtitle = foreach ford generate ename ,job;
grunt> dump fordtitle;
```

```
(FORD,ANALYST)
```

3. Write down the Apache Pig statement(s) to get the first employee (by the hiredate).

joinDate1 = FOREACH emp GENERATE hiredate ;
joinDate2 = ORDER joinDate1 BY hiredate ASC ;
earliestJoinDate = LIMIT joinDate2 1 ;

```
grunt> joindate1 = foreach emp generate hiredate;
grunt> joindate2= order joindate1 by hiredate ASC;
grunt> earliestJOindate= LIMIT joindate2 1;
```

earliestJoinEmp = JOIN emp BY hiredate, earliestJoinDate BY hiredate ;

```
grunt> earliestJoimemp= join emp by hiredate, earliestJOindate by hiredate;
```

x = FOREACH earliestJoinEmp GENERATE $0, $1, $2, $4;
DUMP x;

```
grunt> meenu= Foreach earliestJoimemp GENERATE $0,$1,$2,$4;
grunt> dump meenu;
```

```
(7369,SMITH,CLERK,1980-12-17T00:00:00.000-08:00)
```

4. Write down the Apache Pig statement(s) to get the number of employees in each department.
empDept = GROUP emp BY deptno ;

```
grunt> empdept = group emp by deptno;
```

deptEmp = FOREACH empDept GENERATE group AS deptno, COUNT(emp) AS empCnt ;

```
grunt> deptemp = foreach empdept generate group as deptno , COUNT(emp) AS empcut;
```

DUMP deptEmp ;

```
grunt> dump deptemp;
```

```
(10,3)
(20,5)
(30,6)
```

5. Write down the Apache Pig statement(s) to get the number of employees in each city.
empDept = GROUP emp BY deptno ;
deptEmp = FOREACH empDept GENERATE group AS deptno, COUNT(emp) AS empCnt ;
empCity = JOIN deptEmp BY deptno, dept BY deptno ;
city = GROUP empCity BY loc ;

```
grunt> empdept= group emp by deptno;
grunt> deptemp = foreach empdept GENERATE group AS deptno, COUNT(emp) AS empcut;
grunt> empcity= join deptemp by deptno, dep by deptno;
grunt> city= group empcity by loc;
```

cityEmp = FOREACH city GENERATE group AS city, SUM(empCity.empCnt) AS cityEmpCnt ;
DUMP cityEmp;

```
grunt> cityemp = foreach city GENERATE group as city, SUM(empcity.empcut) AS cityempcut;
grunt> dump cityemp;
```

```
(DALLAS,5)
(CHICAGO,6)
(NEW YORK,3)
```

6. Try the following data outputs:
   (1) The average salary in each city.
   a = join emp by deptno ,dept by deptno ;
   b= foreach a generate sal ,loc ;

```
grunt> aa =join emp by deptno , dep by deptno;
grunt> bb = foreach aa generate sal, loc;
```

c= group b by loc;

```
grunt> cc= group bb by loc;
```

d= foreach c generate group ,AVG(b.sal);

```
grunt> dd = foreach cc GENERATE group , AVG(bb.sal);
```

dump d;

```
grunt> dump dd;
```

```
(DALLAS,2175.0)
(CHICAGO,1566.6666666666667)
(NEW YORK,2916.6666666666665)
```

(2) The highest paid employee in each department

a = foreach emp generate sal,deptno;
b= group a by deptno;

```
grunt> aa = foreach emp generate sal,deptno;
grunt> bb = group aa by deptno;
```

c= foreach b generate group ,MAX(a.sal);
dump c;

```
grunt> cc = foreach bb generate group,  MAX(aa.sal);
grunt> dump cc;
```

```
(10,5000.0)
(20,3000.0)
(30,2850.0)
```

(3) The managers whose subordinates have at least one subordinate

a= foreach emp generate empno ;
b= foreach emp generate mgr ;
c = join a by $0 left , b by $0 ;

```
grunt> aa= foreach emp generate empno;
grunt> bb= foreach emp generate mgr;
grunt> cc= join aa by $0 left, bb by$0;
```

d= foreach c generate $0, {$1} ;
e= foreach d generate $0,COUNT($1) ;
f= filter e by $1 == 0;

```
grunt> dd= foreach cc generate $0, {$1};
grunt> ee = foreach dd generate $0, COUNT($1);
grunt> ff = filter  ee by $1==0;
```

g = foreach f generate $0 ;

```
grunt> gg = foreach ff generate $0;
```

h = join emp by empno ,g by $0 ;

```
grunt> hh = join emp by empno, gg by $0;
```

i = group h by mgr ;

```
grunt> ii = group hh by mgr;
```

j = foreach i generate group ,COUNT(h);

```
grunt> jj =  foreach ii generate group, COUNT(hh);
```

k = filter j by $1 >= 1 ;

```
grunt> kk = filter jj by $1>=1;
```

l = join emp by empno ,k by $0 ;

```
grunt> ll= join emp by empno, kk by $0;
```

m = filter l by job != 'MANAGER' ;

```
grunt> mmm = filter ll by job!='MANAGER';
```

n = group h by mgr ;

```
grunt> nn = group hh by mgr;
```

o = foreach n generate group;

```
grunt> oo = foreach nn generate group;
```

p= join emp by empno ,o by $0 ;

```
grunt> pp = join emp by empno , oo by $0;
```

dump p;

```
(7698,BLAKE,MANAGER,7839,1991-05-01T00:00:00.000-07:00,2850.0,30,7698)
(7782,CLARK,MANAGER,7839,1981-06-09T00:00:00.000-07:00,2450.0,10,7782)
(7788,SCOTT,ANALYST,7655,1987-03-21T00:00:00.000-08:00,3000.0,20,7788)
(7902,FORD,ANALYST,7655,1981-12-03T00:00:00.000-08:00,3000.0,20,7902)
```

(4) The number of employees for each hiring year

a= foreach emp generate empno ,GetYear(hiredate) ;

```
grunt> aa = foreach emp generate empno, GetYear(hiredate);
```

b= group a by $1 ;

```
grunt> bb = group aa by $1;
```

c= foreach b generate group , COUNT(a.empno);

```
grunt> c = foreach bb generate group , COUNT(aa.empno);
```

dump c ;

```
grunt> dump c;
```

```
(1980,1)
(1981,10)
(1987,2)
(1991,1)
```

Working with Online Social Networks Data

A table named “famous” has two columns called user id and follower id. It represents each user ID has a particular follower ID. These follower IDs are also users of Facebook / Meta. Then, find the famous percentage of each user.

Note: **famous Percentage = number of followers a user has / total number of users on the platform.**

Counting the number of "followers" per Twitter user

Problem Statement: For each user, Calculate the total number of followers of that user.

- Create a table with name Famous having two fields User Id => Integer& Follower Id => Integer in Linux.

```
[cloudera@quickstart ~]$ touch famous
[cloudera@quickstart ~]$ cat> famous
123 14
189 15
890 16
167 17
568 19
145 20 ^Z
[1]+  Stopped                 cat > famous
```

- Put it into hdfs home directory. Then run pig so as grunt shell appears.

```
[cloudera@quickstart ~]$ hdfs dfs -put famous /user/cloudera
[cloudera@quickstart ~]$ pig
```

- Load the dataset from hdfs into pig.

```
grunt> data = LOAD '/user/cloudera/famous' USING PigStorage (' ') as (id:long, fr:long);
2025-04-20 23:15:51,546 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation -
2025-04-20 23:15:51,546 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation -
.jobtracker.address
grunt> dump data;
```

```
(123,14)
(189,15)
(890,16)
(167,17)
(568,19)
```

- Check if user IDs are valid (e.g. not null) and clean the dataset. Organize data such that each node ID is associated to a list of neighbors.

```
grunt> SPLIT data into good_data if id is not null and fr is not null , bad_data OTHERWISE;
grunt> nodes = GROUP good_data BY id;
grunt> dump nodes;
```

```
(123,{(123,14)})
(167,{(167,17)})
(189,{(189,15)})
(568,{(568,19)})
(890,{(890,16)})
```

- For each node ID generate an output relation consisting of the node ID and the number of "friends".

```
grunt> friends = FOREACH nodes GENERATE group, COUNT(good_data) AS followers;
grunt> dump friends;
```

```
(123,1)
(167,1)
(189,1)
(568,1)
(890,1)
```

- Count the following:

```
grunt> nodes2 = GROUP good_data BY fr;
grunt> following = FOREACH nodes2 GENERATE group, COUNT(good_data);
grunt> dump following;
```

```
(14,1)
(15,1)
(16,1)
(17,1)
(19,1)
```

- Find the outliers.

```
grunt> outliers = FILTER friends BY followers <3;
2025-04-20 23:27:33,753 [main] WARN  org.apache.pi
grunt> dump outliers;
```

```
(16,2)
(18,1)
(19,1)
```

Word Count Problem in Hive.

Objective

The goal of the Word Count problem is to count the number of times each word appears in a given text dataset.

1. Problem Setup

Suppose we have a text dataset where each row contains a line of text. Our task is to break these lines into individual words and count how many times each word appears.

Step 1: **Create a Word Count File Locally**

- Create a directory named **WordCountMeenu** and create a file in this directory named **WC.txt.**

```
[cloudera@quickstart ~]$ mkdir WordcountMeenu
```

```
[cloudera@quickstart ~]$ touch /home/cloudera/WordcountMeenu/WC.txt
[cloudera@quickstart ~]$ cat /home/cloudera/WordcountMeenu/WC.txt
```

- Add content to your file and save it.

```
[cloudera@quickstart ~]$ cat>  /home/cloudera/WordcountMeenu/WC.txt
Meenu
vatta
vatta
meenu
hive
pig
pig
hive
hive
hive
big data
^Z
[4]+  Stopped                 cat > /home/cloudera/WordcountMeenu/WC.txt
```

Step 2: Create Hive Table for Word Count.

First, we create a Hive table to store the text data. This table will have a single column called **`line`** of type **`STRING`.**

```
hive> create table meenuinputfile (text_line String);
OK
Time taken: 0.098 seconds
```

Explanation:

- `CREATE TABLE`: Defines a new table.
- `texts`: Table name.
- `line STRING`: A column named `line` that will store each line of text as a string.

Step 3. Loading the data in the table.

- From Local to Linux file system.

  load data local inpath '/home/cloudera/direct_name/file_name' into table table_name;

LOAD DATA

This is a Hive command used to load data into a table.

LOCAL

This keyword tells Hive that the file is located **on the local file system** (not HDFS).
If you **remove LOCAL**, Hive will look for the file in **HDFS** instead.

INPATH '**/home/cloudera/direct_name/file_name**'

This specifies the **full local path** to the file you want to load.

- **/home/cloudera/direct_name/file_name** → path to the actual file on your local machine.

Make sure the file exists at this path before running the command.

INTO TABLE table_name

This tells Hive to load the data **into the specified Hive table**.

- **table_name** should already be created before this command is run.
- The format of the file should match the table's schema (e.g., number of columns, delimiters).

```
hive> load data local inpath '/home/cloudera/WordcountMeenu/WC.txt' into table meenuinputfile;
Loading data to table default.meenuinputfile
Table default.meenuinputfile stats: [numFiles=1, totalSize=0]
OK
Time taken: 0.494 seconds
```

```
hive> LOAD DATA LOCAL INPATH 'WordcountMeenu/WC.txt' into table meenuinputfile;
Loading data to table default.meenuinputfile
Table default.meenuinputfile stats: [numFiles=2, totalSize=61]
OK
Time taken: 0.584 seconds
```

- From HDFS.
  load data inpath 'direct_name/file_name' into table table_name;

Step 4: Word Count Query in Hive

Now that the table is ready, let's write the query to count words.

- To display the all content stored in the table.
  select * from tablename;

```
hive> select * from meenuinputfile;
OK
Meenu
vatta
vatta
meenu
hive
pig
pig
hive
hive
hive
big data
Time taken: 0.184 seconds, Fetched: 11 row(s)
```

- To display the fields present in table;
  DESCRIBE tablename;

```
hive> describe meenuinputfile;
OK
text_line               string
Time taken: 0.058 seconds, Fetched: 1 row(s)
```

- To Count the occurence of word in file.
  CREATE table wordcount as select word , COUNT(*) from tablename lateral view explode(split(text_line, ' ')) 1table as word GROUP BY word;

```
hive> CREATE table wordcount as SELECT word, COUNT(*) from meenuinputfile lateral view explode(split(text_line, ' ')) 1table as word GROUP BY word;
Query ID = cloudera_20250421001515_a43ce60b-f780-4312-b731-a223bd34095f
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks not specified. Estimated from input data size: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1743139950833_0038, Tracking URL = http://quickstart.cloudera:8088/proxy/application_1743139950833_0038/
Kill Command = /usr/lib/hadoop/bin/hadoop job  -kill job_1743139950833_0038
Hadoop job information for Stage-1: number of mappers: 2; number of reducers: 1
2025-04-21 00:15:34,068 Stage-1 map = 0%,  reduce = 0%
2025-04-21 00:15:41,027 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 1.6 sec
2025-04-21 00:15:45,136 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 2.29 sec
MapReduce Total cumulative CPU time: 2 seconds 290 msec
Ended Job = job_1743139950833_0038
Moving data to: hdfs://quickstart.cloudera:8020/user/hive/warehouse/wordcount
Table default.wordcount stats: [numFiles=1, numRows=7, totalSize=50, rawDataSize=43]
MapReduce Jobs Launched:
Stage-Stage-1: Map: 2  Reduce: 1   Cumulative CPU: 2.29 sec   HDFS Read: 14301 HDFS Write: 123 SUCCESS
Total MapReduce CPU Time Spent: 2 seconds 290 msec
OK
Time taken: 18.603 seconds
```

Explanation:

`CREATE TABLE wordcount AS`

- This command creates a new Hive table named `wordcount`.
- The structure (columns and data types) of the new table is inferred from the result of the `SELECT` query.
- The data resulting from the `SELECT` is immediately inserted into the newly created table.

`SELECT word, COUNT(*)`

- **`word`**: Represents each individual word extracted from the input data.
- **`COUNT(*)`**: Counts the total number of occurrences of each unique word.
- This clause selects the word and the number of times it appears in the dataset.

FROM tablename

- Specifies the source table that contains the raw text data.
- This table must have a column (e.g., `text_line`) containing text strings.

LATERAL VIEW explode(split(text_line, ' ')) t1 AS word

- **split(text_line, ' '):** Splits the text in **text_line** column into an array of words, using space as the delimiter.
- **explode(...):** Converts the resulting array into multiple rows, each containing a single word.
- **LATERAL VIEW**: A Hive construct that allows the use of `explode` to transform complex types (like arrays) into individual rows.
- **t1**: An alias for the virtual table created by the lateral view.
- **AS word**: Assigns a name to the exploded value (each individual word).

GROUP BY word

- Groups the dataset by each unique word.
- Necessary for the aggregation **(COUNT(*))** to compute the frequency of each word.

Result

- A new table `wordcount` is created.
- It contains two columns:
  - **word:** the unique word.
  - **count**: the number of times the word appears in the dataset.

➢ Display the result stored in wordcount table.

SELECT * from wordcount;

```
hive> select * from wordcount;
OK
Meenu   1
big     1
data    1
hive    4
meenu   1
pig     2
vatta   2
Time taken: 0.045 seconds, Fetched: 7 row(s)
```

IPL Dataset Analysis using Hadoop Hive

The Indian Premier League (IPL) is one of the most popular T20 cricket leagues in the world, with rich data that can be analyzed to uncover insights on player performances, team strategies, and match outcomes. In this project, we performed a comprehensive analysis of IPL data using **Hadoop Hive**, leveraging its powerful SQL-like interface to handle large-scale data efficiently.

Dataset Description

We used two primary tables in our analysis:

1. **`matches` table**: Contains metadata about each IPL match, such as the teams playing, venue, toss details, and match results.
2. **`deliveries` table**: Contains ball-by-ball information, including batsman, bowler, runs scored, extras, dismissals, etc.

Prepare IPL Dataset

You will need the IPL dataset, which should ideally consist of two tables: `matches` and `deliveries`.

1 Get IPL Data

- Download IPL datasets (you can find IPL data from sources like **Kaggle** or other open data repositories).
- The two tables are:
  - **Matches**: Contains metadata of each IPL match (e.g., teams, date, winner).
  - **Deliveries**: Contains ball-by-ball data (e.g., runs scored, batsman, bowler, dismissal).

2 Upload Data to HDFS

```
[root@quickstart cloudera]# ls ipl
deliveries.csv  matches.csv
[root@quickstart cloudera]# hdfs dfs -put /home/cloudera/ipl /user/cloudera/ipldata
25/04/19 01:18:08 WARN hdfs.DFSClient: Caught exception
java.lang.InterruptedException
        at java.lang.Object.wait(Native Method)
        at java.lang.Thread.join(Thread.java:1281)
        at java.lang.Thread.join(Thread.java:1355)
        at org.apache.hadoop.hdfs.DFSOutputStream$DataStreamer.closeResponder(DFSOutput
Stream.java:967)
        at org.apache.hadoop.hdfs.DFSOutputStream$DataStreamer.endBlock(DFSOutputStream
.java:705)
        at org.apache.hadoop.hdfs.DFSOutputStream$DataStreamer.run(DFSOutputStream.java
:894)
[root@quickstart cloudera]# hdfs dfs -ls /user/cloudera/ipldata
Found 2 items
-rw-r--r--   1 root cloudera   27019953 2025-04-19 01:18 /user/cloudera/ipldata/deliver
ies.csv
-rw-r--r--   1 root cloudera     225266 2025-04-19 01:18 /user/cloudera/ipldata/matches
.csv
```

3. Create Hive Tables

Now that the data is in HDFS, you can create external Hive tables that reference these CSV files, making them queryable.

- Create a database to store all files and tables and activate the database for following queries

  hive> create database ipl_db;

  hive> use ipl_db;

```
hive> create database meenu_ipl_db;
OK
Time taken: 1.099 seconds
hive> use meenu_ipl_db;
OK
Time taken: 0.053 seconds
```

- ***Create the `matches` Table***

  In Hive, create an external table for the `matches` dataset.

hive> CREATE TABLE matches_ipl (id STRING, season STRING, city STRING, date_of_match STRING, team1 STRING, team2 STRING, toss_winner STRING, toss_decision STRING, result STRING, dl_applied STRING, winner STRING, win_by_runs STRING, win_by_wickets STRING, player_of_match STRING, venue STRING, umpire1 STRING, umpire2 STRING, umpire3 STRING) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n';

```
hive> create table matches_ipl(id String, season String, city String ,date_of_match Str
ing,team1 String , team2 String , toss_winner String, toss_decisio String,result String
, dl_applied String, winner String,win_by_runs String, win_by_wickets String , player_o
f_match String, venue String, umpire1 String, umpire2 String , umpire3 String) ROW FORM
AT DELIMITED FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n';
OK
Time taken: 0.66 seconds
hive>
```

- ***Create the `deliveries` Table***

  Similarly, create an external table for the `deliveries` dataset:

hive> CREATE TABLE deliveries_ipl (matchid STRING, inning STRING, batting_team STRING, bowling_team STRING, overs STRING, ball STRING, batsman STRING, non_striker STRING, bowler STRING, is_super_over STRING, wide_runs STRING, bye_runs STRING, leg_bye_runs STRING, no_ball_runs STRING, penalty_runs STRING, batsman_runs STRING, extra_runs STRING, total_runs STRING, player_dismissed STRING, dismissal_kind STRING, fielder STRING) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n';

```
hive> create table deliv_ipl(matchid String, inning String, batting_team string,is_supe
r_over String , wide_runs String, leg_bye_runs String, no_balls_runs String, penalty_ru
ns String , batsman_runs String, extra_runs String, total_runs String, player_dismissed
 String, dismissal_kind String, fielder String)ROW FORMAT DELIMITED FIELDS TERMINATED B
Y ',' LINES TERMINATED BY '\n';
OK
Time taken: 0.476 seconds
```

- ***Query IPL Data Using HiveQL***

Once the tables are created, you can start querying the IPL data using **HiveQL**, which is similar to SQL.

- *Loading the csv file into blank table created above.*

hive> LOAD DATA LOCAL INPATH '/home/cloudera/Downloads/matches.csv' OVERWRITE INTO TABLE matches_ipl;

```
hive> LOAD DATA LOCAL INPATH '/home/cloudera/ipl/matches.csv' OVERWRITE INTO TABLE matc
hes_ipl;
Loading data to table meenu_ipl_db.matches_ipl
Table meenu_ipl_db.matches_ipl stats: [numFiles=1, numRows=0, totalSize=225266, rawData
Size=0]
OK
Time taken: 1.31 seconds
```

hive> LOAD DATA LOCAL INPATH '/home/cloudera/Downloads/deliveries.csv' OVERWRITE INTO TABLE deliveries_ipl;

```
hive> LOAD DATA LOCAL INPATH '/home/cloudera/ipl/deliveries.csv' OVERWRITE INTO TABLE deliv_ipl;
Loading data to table meenu_ipl_db.deliv_ipl
Table meenu_ipl_db.deliv_ipl stats: [numFiles=1, numRows=0, totalSize=27019953, rawDataSize=0]
OK
Time taken: 1.553 seconds
```

- Display table names in the database

  hive> show tables;

```
hive> show tables;
OK
deliv_ipl
matches_ipl
Time taken: 0.059 seconds, Fetched: 2 row(s)
```

- Display top 6 entries of the table

  hive> select * from matches_ipl limit 6;

```
hive> select * from matches_ipl limit 6;
OK
id      season  city    date    match_type      player_of_match venue   team1   team2 t
oss_winner      toss_decision   winner  result  result_margin   target_runs     target_
overs   super_over      method
335982  2007/08 Bangalore       2008-04-18      League  BB McCullum     M Chinnaswamy S
tadium  Royal Challengers Bangalore     Kolkata Knight Riders   Royal Challengers Banga
lore    field   Kolkata Knight Riders   runs    140     223     20      N       NA
335983  2007/08 Chandigarh      2008-04-19      League  MEK Hussey      "Punjab Cricket
 Association Stadium     Mohali"        Kings XI Punjab Chennai Super Kings     Chennai
 Super Kings    bat     Chennai Super Kings     runs    33      241     20      N
335984  2007/08 Delhi   2008-04-19      League  MF Maharoof     Feroz Shah Kotla      D
elhi Daredevils Rajasthan Royals        Rajasthan Royals        bat     Delhi Daredevil
s       wickets 9       130     20      N       NA
335985  2007/08 Mumbai  2008-04-20      League  MV Boucher      Wankhede Stadium      M
umbai Indians   Royal Challengers Bangalore     Mumbai Indians  bat     Royal Challenge
rs Bangalore    wickets 5       166     20      N       NA
335986  2007/08 Kolkata 2008-04-20      League  DJ Hussey       Eden Gardens    Kolkata
 Knight Riders  Deccan Chargers Deccan Chargers bat     Kolkata Knight Riders   wickets
5       111     20      N       NA
Time taken: 0.49 seconds, Fetched: 6 row(s)
```

hive> select * from deliveries_ipl limit 6;

```
hive> select * from deliv_ipl limit 6;
OK
match_id       inning  batting_team    bowling_team    over    ball    batter  bowler  non_striker     batsman_runs    extra_runs      total_runs      extras_type     i
s_wicket
335982  1       Kolkata Knight Riders   Royal Challengers Bangalore     0       1       SC Ganguly      P Kumar BB McCullum     0       1       1       legbyes 0
335982  1       Kolkata Knight Riders   Royal Challengers Bangalore     0       2       BB McCullum     P Kumar SC Ganguly      0       0       0               0
335982  1       Kolkata Knight Riders   Royal Challengers Bangalore     0       3       BB McCullum     P Kumar SC Ganguly      0       1       1       wides   0
335982  1       Kolkata Knight Riders   Royal Challengers Bangalore     0       4       BB McCullum     P Kumar SC Ganguly      0       0       0               0
335982  1       Kolkata Knight Riders   Royal Challengers Bangalore     0       5       BB McCullum     P Kumar SC Ganguly      0       0       0               0
Time taken: 0.135 seconds, Fetched: 6 row(s)
```

- Gives table of overall total no. of matches played in IPL

hive> CREATE TABLE total_match as SELECT COUNT(*) as matches FROM matches_ipl;

```
hive> create table total_match as SELECT COUNT(*) as macthes from matches_ipl;
Query ID = root_20250419031111_fc437bc5-b63e-455d-8370-4d8d1dd4d99c
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1745044038944_0001, Tracking URL = http://quickstart.cloudera:8088/p
roxy/application_1745044038944_0001/
Kill Command = /usr/lib/hadoop/bin/hadoop job  -kill job_1745044038944_0001
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2025-04-19 03:12:13,157 Stage-1 map = 0%,  reduce = 0%
2025-04-19 03:12:27,661 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 1.88 sec
2025-04-19 03:12:41,398 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 3.99 sec
MapReduce Total cumulative CPU time: 3 seconds 990 msec
Ended Job = job_1745044038944_0001
Moving data to: hdfs://quickstart.cloudera:8020/user/hive/warehouse/meenu_ipl_db.db/total_match
Table meenu_ipl_db.total_match stats: [numFiles=1, numRows=1, totalSize=5, rawDataSize=4]
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 3.99 sec   HDFS Read: 234343 HDFS Write: 85 SUCCESS
Total MapReduce CPU Time Spent: 3 seconds 990 msec
OK
Time taken: 63.855 seconds
```

```
hive> select * from total_match;
OK
1096
Time taken: 0.092 seconds, Fetched: 1 row(s)
```

- Gives table of toss winning team and corresponding match winning team per match

hive> CREATE TABLE toss_win as SELECT toss_winner, winner from matches_ipl;

```
hive> CREATE TABLE toss_win as SELECT toss_winner, winner from matches_ipl;
```

```
hive> SELECT * FROM toss_win;
```

```
"Arun Jaitley Stadium
"Bharat Ratna Shri Atal Bihari Vajpayee Ekana Cricket Stadium
"Narendra Modi Stadium
"MA Chidambaram Stadium
"Eden Gardens
"Bharat Ratna Shri Atal Bihari Vajpayee Ekana Cricket Stadium
"MA Chidambaram Stadium
"Rajiv Gandhi International Stadium
"Wankhede Stadium
"M Chinnaswamy Stadium
"Himachal Pradesh Cricket Association Stadium
"Bharat Ratna Shri Atal Bihari Vajpayee Ekana Cricket Stadium
"Wankhede Stadium
"Arun Jaitley Stadium
"Rajiv Gandhi International Stadium
"Himachal Pradesh Cricket Association Stadium
"Narendra Modi Stadium
"Eden Gardens
"MA Chidambaram Stadium
"M Chinnaswamy Stadium
"Arun Jaitley Stadium
"Barsapara Cricket Stadium
"Wankhede Stadium
"M Chinnaswamy Stadium
"Rajiv Gandhi International Stadium
"Narendra Modi Stadium
"Narendra Modi Stadium
"MA Chidambaram Stadium
"MA Chidambaram Stadium
Time taken: 0.109 seconds, Fetched: 1096 row(s)
```

- Shows how many players have won man of the match award

hive> SELECT COUNT(DISTINCT(player_of_match)) FROM matches_ipl;

```
hive> SELECT COUNT(DISTINCT(playerofmatch)) FROM matches_ipl;
```

```
OK
293
```

- Details of top 12 matches won by most runs

hive> CREATE TABLE WINS_BY_MOST_RUNS AS SELECT * FROM matches_ipl ORDER BY win_by_runs DESC LIMIT 12;

```
hive> create table wins_by_most_runs as select * from matches_ipl order by win_by_runs
DESC LIMIT 12;
Query ID = root_20250419033535_e1f93d5f-4686-464f-a3b3-0dc4bca2bc56
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1745044038944_0003, Tracking URL = http://quickstart.cloudera:8088/p
roxy/application_1745044038944_0003/
Kill Command = /usr/lib/hadoop/bin/hadoop job  -kill job_1745044038944_0003
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2025-04-19 03:35:35,696 Stage-1 map = 0%,  reduce = 0%
2025-04-19 03:35:49,721 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 2.55 sec
2025-04-19 03:36:05,745 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 5.0 sec
MapReduce Total cumulative CPU time: 5 seconds 0 msec
Ended Job = job_1745044038944_0003
Moving data to: hdfs://quickstart.cloudera:8020/user/hive/warehouse/meenu_ipl_db.db/win
s_by_most_runs
Table meenu_ipl_db.wins_by_most_runs stats: [numFiles=1, numRows=12, totalSize=2367, ra
wDataSize=2355]
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 5.0 sec   HDFS Read: 238720 HDFS Wri
te: 2455 SUCCESS
Total MapReduce CPU Time Spent: 5 seconds 0 msec
OK
Time taken: 47.248 seconds
```

hive> Select * from wins_by_most_runs;

```
nternational Stadium     Uppal" Sunrisers Hyderabad     Rising Pune Supergiants Rising
Pune Supergiants        field   Rising Pune Supergiants runs    34      61      11    N
980953  2016    Hyderabad       2016-04-30      League  DA Warner       "Rajiv Gandhi I
nternational Stadium     Uppal" Sunrisers Hyderabad     Royal Challengers Bangalore   R
oyal Challengers Bangalore      field   Sunrisers Hyderabad     runs    15      195   2
0       N
980967  2016    Hyderabad       2016-05-06      League  B Kumar "Rajiv Gandhi Internati
onal Stadium     Uppal" Sunrisers Hyderabad     Gujarat Lions   Sunrisers Hyderabad   f
ield    Sunrisers Hyderabad     wickets 5       127     20      N
980971  2016    Chandigarh      2016-05-07      League  MP Stoinis      "Punjab Cricket
 Association IS Bindra Stadium   Mohali"        Kings XI Punjab Delhi Daredevils      D
elhi Daredevils field   Kings XI Punjab runs    9       182     20      N
980977  2016    Chandigarh      2016-05-09      League  SR Watson       "Punjab Cricket
 Association IS Bindra Stadium   Mohali"        Kings XI Punjab Royal Challengers Banga
lore    Kings XI Punjab field   Royal Challengers Bangalore     runs    1       176   2
0       N
980983  2016    Hyderabad       2016-05-12      League  CH Morris       "Rajiv Gandhi I
nternational Stadium     Uppal" Sunrisers Hyderabad     Delhi Daredevils        Delhi D
aredevils       field   Delhi Daredevils        wickets 7       147     20      N
1082591 2017    Hyderabad       2017-04-05      League  Yuvraj Singh    "Rajiv Gandhi I
nternational Stadium     Uppal" Sunrisers Hyderabad     Royal Challengers Bangalore   R
oyal Challengers Bangalore      field   Sunrisers Hyderabad     runs    35      208   2
0       N
1082596 2017    Hyderabad       2017-04-09      League  Rashid Khan     "Rajiv Gandhi I
nternational Stadium     Uppal" Sunrisers Hyderabad     Gujarat Lions   Sunrisers Hyder
abad    field   Sunrisers Hyderabad     wickets 9       136     20      N
1082609 2017    Hyderabad       2017-04-17      League  B Kumar "Rajiv Gandhi Internati
onal Stadium     Uppal" Sunrisers Hyderabad     Kings XI Punjab Kings XI Punjab field S
unrisers Hyderabad      runs    5       160     20      N
Time taken: 0.132 seconds, Fetched: 12 row(s)
```

- Total number of extra runs given in IPL

hive> SELECT SUM(extra_runs) FROM deliveries_ipl;

```
hive> select sum(extra_runs) from deliv_ipl;
Query ID = root_20250419044545_0b5322f9-b0d7-49d7-b7c0-7d598c5be39e
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1745044038944_0011, Tracking URL = http://quickstart.cloudera:8088/p
roxy/application_1745044038944_0011/
Kill Command = /usr/lib/hadoop/bin/hadoop job  -kill job_1745044038944_0011
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2025-04-19 04:45:19,181 Stage-1 map = 0%,  reduce = 0%
2025-04-19 04:45:32,572 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 3.51 sec
2025-04-19 04:45:46,103 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 5.81 sec
MapReduce Total cumulative CPU time: 5 seconds 810 msec
Ended Job = job_1745044038944_0011
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 5.81 sec   HDFS Read: 27029300 HDFS
Write: 9 SUCCESS
Total MapReduce CPU Time Spent: 5 seconds 810 msec
OK
330064.0
```

- average runs by which a team batting first wins the match

hive> select avg(win_by_runs) from matches_ipl;

```
hive> select avg(win_by_runs)from matches_ipl;
```

- total runs scored by MS Dhoni in IPL

hive> select sum(batsman_runs) from deliveries_ipl where batsman='MS Dhoni';

```
hive> SELECT SUM(batsman_run) FROM deliveries_ipl WHERE batsman= 'MS Dhoni';
```

RESULT:

```
5243
```

- total boundaries hit in IPL

hive> select count(total_runs) from deliveries_ipl where total_runs=4 or total_runs=6;

```
hive> SELECT COUNT(total_run) FROM deliveries_ipl WHERE total_run=4 or total_run=6;
```

```
43185
```

- season wise most runs by which a team won a match--

hive> Create table most_runs_win_seasonwise as select season,max(win_by_runs) from matches_ipl group by season order by season;

```
hive> create table most_runs_win_seasonwise AS select season, max(result_marg
in)from matches_ipl group by season order by season;
```

```
hive> SELECT * FROM most_runs_win_seasonwise;
OK
```

```
2015    138
2016    144
2017    146
2018    102
2019    118
2020/21 97
2021    86
2022    NULL
2023    NULL
2024    NULL
season  NULL
Time taken: 0.039 seconds, Fetched: 18 row(s)
```

- boundaries hit per team in IPL

hive> create table boundaries_per_team as select batting_team, count(total_runs) as no_of_boundaries from deliveries_ipl where total_runs=4 or total_runs=6 group by batting_team order by no_of_boundaries desc;

```
hive> create table boundaries_per_team as select batting_team, COUNT(total_runs) as no_of_boundaries from deliv_ipl Where total_runs = 4 or total_runs =6 group by batti
ng_team order by no_of_boundaries DESC;
```

```
hive> select * from boundaries_per_team;
OK
Kolkata Knight Riders   71
Mumbai Indians  64
Kings XI Punjab 57
Royal Challengers Bangalore     54
Chennai Super Kings     54
Rajasthan Royals        44
Sunrisers Hyderabad     39
Delhi Daredevils        30
Delhi Capitals  19
Deccan Chargers 17
Gujarat Titans  13
Lucknow Super Giants    12
Punjab Kings    9
Pune Warriors   7
Rising Pune Supergiant  4
Royal Challengers Bengaluru     3
Gujarat Lions   3
Rising Pune Supergiants 2
Kochi Tuskers Kerala    2
```

- top 10 bowlers who conceded most extra runs.

hive> create table most_extras_per_bowler as select bowler, sum(extra_runs) as total_extra_runs from deliveries_ipl group by bowler order by total_extra_runs desc limit 5;

```
hive> create table most_extras_per_bowler AS select bowler, SUM(extra_runs) a
s total_extra_runs from deliv_ipl group by bowler  order by total_extra_runs
desc limit 5;
```

```
hive> SELECT * FROM most_extra_per_bowler;
OK
B Kumar 307
SL Malinga      292
DJ Bravo        258
R Ashwin        257
UT Yadav        257
```

hive> create table most_runs as select a.season, b.batsman, b.total_runs, b.inning from matches_ipl a left outer join deliveries_ipl b on (a.id=b.matchid);

```
hive> create table most_runs as select a.season, b.total_runs, b.inning from matches_ipl a LEFT OUTER JOIN deliv_ipl b on (a.id= b.matchid);
Query ID = cloudera_20250421004545_cd0cc4b4-e2f6-4a25-a25f-db0512b60885
```

```
2016    0       2
2016    0       2
2016    1       2
2016    0       2
2016    0       2
2016    0       2
2016    0       2
2016    0       2
2016    1       2
2016    0       2
2016    0       2
season  NULL    NULL
Time taken: 0.077 seconds, Fetched: 260921 row(s)
```

hive> CREATE TABLE most_runs_1 AS SELECT season, batsman, sum(total_runs) as sum_runs FROM most_runs GROUP BY SEASON, BATSMAN;

```
hive> create table most_runs_1 as select season, batsman, sum(total_runs) as
sum_runs from most_runs GROUP by season, batsman;
```

```
hive> SELECT * FROM most_runs_1;
```

```
2024    Saurav Chauhan  19
2024    Shahbaz Ahmed   223
2024    Shashank Singh  370
2024    Shivam Singh    2
2024    Shubman Gill    438
2024    Sikandar Raza   45
2024    Sumit Kumar     23
2024    Swapnil Singh   39
2024    T Kohler-Cadmore        49
2024    T Stubbs        393
2024    TA Boult        14
```

- total dismissals in IPL

hive> SELECT COUNT(player_dismissed) AS total_wickets FROM deliveries_ipl WHERE player_dismissed != “ “;

```
hive> select count(player_dismissed) as total_wickets from deliv_ipl where pl
ayer_dismissed != " ";
```

- **top 10 most wicket taking bowlers**

hive> SELECT bowler, COUNT(player_dismissed) as total_wickets FROM deliveries_ipl WHERE player_dismissed != '' GROUP BY bowler ORDER BY total_wickets desc LIMIT 10;

```
hive> select bowler, count(player_dismissed) as total_wickets from deliv_ipl
where player_dismissed is not null Group by bowler order by total_wickets des
c limit 10 ";
```

SL Malinga	**188**
DJ Bravo	168
A Mishra	165
Harbhajan Singh	161
PP Chawla	156
B Kumar	141
R Ashwin	138
SP Narine	137
UT Yadav	136
R Vinay Kumar	127

```
R Ashwin        4679
SP Narine       4146
B Kumar 4060
RA Jadeja       3895
PP Chawla       3895
YS Chahal       3628
Harbhajan Singh 3496
A Mishra        3444
DJ Bravo        3296
UT Yadav        3190
Time taken: 33.756 seconds, Fetched: 10 row(s)
```

- shows all tables in current database

hive> show tables ;

boundaries_per_team
deliveries_ipl
ipl_data
matches_ipl
most_extras_per_bowler
most_runs
most_runs_1
most_runs_win_seasonwise
top_scorer_seasonwise
toss_win
total_match
wins_by_most_runs

```
hive> show tables;
OK
boundaries_per_team
deliv_ipl
macthes_ipl
matches_ipl
most_runs
most_runs_1
most_runs_wins_seasonwise
top_scorer_seasonwise
toss_win
total_match
wins_by_most_run
wins_by_most_runs
Time taken: 0.026 seconds, Fetched: 12 row(s)
```

Tutorial 1

- Create ‘émployee’ file with following data:
  $cat> emp

```
[cloudera@quickstart ~]$ ls data
data
[cloudera@quickstart ~]$ cat data
1~meenu~pune~engr~1000~bigdata
2~riya~goa~dev~2000~bfsi
3~pooja~bnglr~rec~3000~hr
4~priya~chennai~sales~5000
```

- Put the file created in linux to Hadoop file system.

```
[cloudera@quickstart ~]$ hdfs dfs -put data /user/cloudera
[cloudera@quickstart ~]$
```

- Open one more terminal for hive and enter hive command.

```
[cloudera@quickstart ~]$ hive

Logging initialized using configuration in file:/etc/hive/conf.dist/hive-log4j.
roperties
WARNING: Hive CLI is deprecated and migration to Beeline is recommended.
hive>
```

- Let us create a logical schema for a emp table

```
hive> show databases;
OK
default
lab1
lab1hive
Time taken: 0.337 seconds, Fetched: 3 row(s)
hive> create database org;
OK
Time taken: 0.539 seconds
hive> use org;
OK
Time taken: 0.033 seconds
hive> show tables;
OK
Time taken: 0.026 seconds
hive> !clear
```

- Create a table of **data** in **org** database with attributes used in the data while entering the data in a format.

```
hive> create table data(
    > id int,
    > name string,
    > city string,
    > department string,
    > salary int,
    > domain string)
    > row format delimited
    > fields terminated by '~' ;
OK
Time taken: 0.158 seconds
```

```
hive> show tables;
OK
data
Time taken: 0.018 seconds, Fetched: 1 row(s)
```

- Now, select all data from the table created in org but no data will be show as data is not yet loaded and the data is in /user/cloudera/data

```
hive> select * from data;
OK
Time taken: 0.23 seconds
```

- Load data record in the table data of org database.

```
hive> load data inpath '/user/cloudera/data' overwrite into table data;
Loading data to table org.data
chgrp: changing ownership of 'hdfs://quickstart.cloudera:8020/user/hive/warehous
Table org.data stats: [numFiles=1, numRows=0, totalSize=109, rawDataSize=0]
OK
Time taken: 0.358 seconds
```

- Now, select all from the table data.

```
hive> select   * from data;
OK
1       meenu   pune    engr    1000    bigdata
2       riya    goa     dev     2000    bfsi
3       pooja   bnglr   rec     3000    hr
4       priya   chennai sales   5000    NULL
Time taken: 0.039 seconds, Fetched: 4 row(s)
```

Rows are loaded to the table in org database where table data exist.

- Describe your table **$describe tablename**;

```
hive> DESCRIBE data;
OK
id                      int
name                    string
city                    string
department              string
salary                  int
domain                  string
```

- Select name and city from table.

```
hive> select name, city from data;
OK
meenu   pune
riya    goa
pooja   bnglr
priya   chennai
```

TUTORIAL 2

- Applying sum(), max(), min(), count(), avg().

1. Max()

```
hive> select max(salary) from data;
OK
5000
```

2. Min()

```
hive> select min(salary) from data;
1000
```

3. COUNT()

```
hive> select COUNT(*) from data;
OK
4
```

4. AVG()

```
hive> select AVG(salary) from data;
2750.0
```

Tutorial 3

- Select first two rows from table;

```
hive> select * from data limit 2;
OK
1       meenu   pune    engr    1000    bigdata
2       riya    goa     dev     2000    bfsi
Time taken: 0.05 seconds, Fetched: 2 row(s)
```

- Create a new table named emp with attributes id, name and city only.

```
hive> create table emp
    > as
    > select id, name, city from data;
```

$Show tables;
$describe emp;

```
hive> show tables;
OK
data
emp
Time taken: 0.01 seconds, Fetched: 2 row(s)
hive> describe emp;
OK
id                      int
name                    string
city                    string
Time taken: 0.037 seconds, Fetched: 3 row(s)
```

- **Select all from emp table.**

```
hive> select * from emp;
OK
1       meenu   pune
2       riya    goa
3       pooja   bnglr
4       priya   chennai
```

- **Rename the new table created.**

```
hive> alter table emp rename to new_emp;
OK
Time taken: 0.081 seconds
hive> show tables;
OK
data
new_emp
```

- **Create a table with same schema but no data record in it.**

```
hive> create table updated_emp like data;
OK
Time taken: 0.071 seconds
hive> show tables;
OK
data
new_emp
updated_emp
Time taken: 0.012 seconds, Fetched: 3 row(s)
hive> describe updated_emp;
OK
id                      int
name                    string
city                    string
department              string
salary                  int
domain                  string
Time taken: 0.05 seconds, Fetched: 6 row(s)
hive> select * from updated_emp;
OK
Time taken: 0.053 seconds
```

The new table is created with with same schema as pf data table but this new table has no data record as it is empty.

TUTORIAL 4

- **Select all from data table and order by id in descending order.**

```
hive> select * from data order by id desc;
```

```
4       priya   chennai sales   5000    NULL
3       pooja   bnglr   rec     3000    hr
2       riya    goa     dev     2000    bfsi
1       meenu   pune    engr    1000    bigdata
```

- **Select all from data table and SORT by name in descending order.**

```
hive> select * from data sort by name desc;
```

```
2       riya    goa     dev     2000    bfsi
4       priya   chennai sales   5000    NULL
3       pooja   bnglr   rec     3000    hr
1       meenu   pune    engr    1000    bigdata
```

- **Select all from data table and SORT by name in Ascending order.**

```
hive> select * from data sort by name asc;
```

```
1       meenu   pune    engr    1000    bigdata
3       pooja   bnglr   rec     3000    hr
4       priya   chennai sales   5000    NULL
2       riya    goa     dev     2000    bfsi
```

- **TUTORIAL 5(INNER JOIN)**

- **Create two tables named sales and product and load the data record in the tables using tutorial 1,2 and 3.**

```
hive> create table sales(cname string, pid int) row format delimited fields terminated by ' ';
```

```
hive> create table product(pid int, pname string) row format delimited fields terminated by ' ';
```

Applying inner join

- **$select t1.*, t2.* from t1 JOIN t2 on (t1.common_col = t2.common_col);**

```
hive> select sale.* , products.* from sales JOIN products ON (sales.pid = products.pid);
```

output

```
rahul   1       1       laptop
ajit    2       2       bat
meenu   3       3       book
```

Tutorial 6 (Outer JOIN)

- **Apply outer join to product and sales table**
- **$select t1.*, t2.* from t1 left outer JOIN t2 on (t1.common_col = t2.common_col);**

```
hive> select sales.*, product.* from sales left outer join product on (sales.pid = product.pid);
```

```
rahul   1       1       laptop
ajit    2       2       bat
meenu   3       3       book
riya    NULL    NULL    NULL
```

- **$select t1.*, t2.* from t1 rightouter JOIN t2 on (t1.common_col = t2.common_col);**

```
hive> select sales.*, product.* from sales right outer join product on (sales.pid = product.pid);
```

```
rahul   1       1       laptop
ajit    2       2       bat
meenu   3       3       book
```

- **$select t1.*, t2.* from t1 full outer JOIN t2 on (t1.common_col = t2.common_col);**

```
hive> select sales.*, product.* from sales full outer join product on (sales.pid = product.pid);
```

```
riya    NULL    NULL    NULL
rahul   1       1       laptop
ajit    2       2       bat
meenu   3       3       book
```

ADVANCE DATATYPES IN HIVE

Tutorial 7

- **Primitive data types: INT, STRING, FLOAT, DOUBLE.**
- **Adv data types: ARRAY, MAP, STRUCT.**
- **Create a table schema with advance data type as shown.**

```
hive> create table per_inf(name string, lfriends array<string>,mobile map<string,bigint>, othr_info struct<company:string, pin:int, married:string, salary:int>)
    > row format delimited
    > fields terminated by '\t'
    > collection items terminated by ','
    > map keys terminated by ':'
    > lines terminated by '\n';
OK
```

- **Create a file in linux with respect to the above schema in linux.**

```
Meenu~sachin,rahul,vishal,gaurav~pers:5678,off:3456~pu,160014,no,100000
Shivani~riya,visha,ajay~pers:6789,off:56677~aiims,15756,no,107890
```

- **Load your text file data record in the table named per_info.**

```
hive> load data inpath '/user/cloudera/info' overwrite into table pers_inf;
```

```
hive> show tables;
OK
data
new_emp
per_inf
per_info
product
sales
updated_emp
```

```
hive> describe per_inf;
OK
name                    string
lfriends                array<string>
mobile                  map<string,bigint>
othr_info               struct<company:string,pin:int,married:string,salary:int>
```

- **Select all from pers_inf;**

```
Meenu   ["sachin","rahul","vishal","gaurav"]    {"pers":5678,"off":3456}        {"company":"pu","pin":160014,"married":"no","salary":100000}
Shivani ["riya","visha","ajay"] {"pers":6789,"off":56677}       {"company":"aiims","pin":15756,"married":"no","salary":107890}
```

- **Select only name from pers_inf;**

```
hive> select name from pers_inf;
OK
Meenu
Shivani
```

- **Select name and office from mobile from table;**

```
hive> select name, mobile['off'] from pers_inf;
OK
Meenu   3456
Shivani 56677
```

- **Select name and friends at index 0 of array from table;**

```
hive> select name, lfriends[0] from pers_inf;
OK
Meenu   sachin
Shivani riya
```

- **Select name and from other_info show the company name.**

```
hive> select name, othr_info.company from pers_inf;
OK
Meenu   pu
Shivani aiims
```

PARTIONED TABLE in HIVE

- **Create a table with defined schema.**

```
hive> create table  ppl(
    > fname string,
    > lname string,
    > state string)
    > row format delimited
    > fields terminated by ',';
OK
```

- **Load data in ppl table**

```
hive> load data inpath '/user/cloudera/new.txt' overwrite into table ppl;
```

- **Select * from table;**

```
hive> select* from ppl;
OK
fname   lname   state
Olivia  Johnson CA
Liam    Smith   NY
Emma    Williams        TX
Noah    Brown   FL
Ava     Jones   IL
Elijah  Garcia  PA
Sophia  Miller  OH
James   Davis   GA
Isabella        Rodriguez       NC
Benjamin        Martinez        MI
Mia     Hernandez       AZ
Lucas   Lopez   NJ
Charlotte       Gonzalez        WA
Henry   Wilson  VA
Amelia  Anderson        MA
Jack    Thomas  IN
Harper  Taylor  TN
Ethan   Moore   MO
Evelyn  Jackson WI
Alexander       Martin  CO
Abigail Lee     MN
Sebastian       Perez   SC
Emily   Thompson        AL
```

- **Select * from table limit 5;**

```
hive> select* from ppl limit 5;
OK
fname   lname   state
Olivia  Johnson CA
Liam    Smith   NY
Emma    Williams        TX
Noah    Brown   FL
```

- **Select distinct state from table;**

```
hive> select state from ppl;
OK
state
CA
NY
TX
FL
IL
PA
OH
GA
NC
MI
AZ
NJ
WA
VA
MA
IN
TN
```

- **Create a partition table**

```
hive> create table part_ppl_state(fname string, lname string) PARTITIONED  by (state string);
```

```
hive> SHOW TABLES;
OK
data
new_emp
part_ppl_state
per_inf
per_info
pers_inf
ppl
product
sales
updated_emp
```

TUTORIAL 9

- Alter table In hive we will use product table of org
- Adding column to table product

```
hive> alter table product
    > add columns(pexydate string);
OK
Time taken: 0.064 seconds
hive> desc product
    > ;
OK
pid                     int
pname                   string
pexydate                string
```

```
hive> alter  table product
    > add columns(prcat string);
OK
Time taken: 0.074 seconds
hive> desc product;
OK
pid                     int
pname                   string
pexydate                string
pcat                    string
prcat                   string
```

- **Moving the column after pname:**

```
hive> alter table product
    > change column pcat pcat string after pname;
OK
Time taken: 0.062 seconds
hive> desc product;
OK
pid                     int
pname                   string
pcat                    string
pexydate                string
prcat                   string
```

TUTORIAL 10

- **Changing column name lname to suname;**

```
hive> desc people ;
OK
fname                   string
lname                   string
state                   string
Time taken: 0.096 seconds, Fetched: 3 row(s)
hive> alter table people change column lname surname string ;
OK
Time taken: 0.258 seconds
hive> desc people ;
OK
fname                   string
surname                 string
state                   string
Time taken: 0.057 seconds, Fetched: 3 row(s)
```

Tutorial 11

Difference between SORT BY & ORDER BY in hive

1. ORDER BY:

- **Global Sorting**: `ORDER BY` sorts the entire result set in a global order, which means it orders the rows across all partitions of the data.
- **Single Reducer**: `ORDER BY` guarantees that the output will be sorted as per the given column(s). However, because the data needs to be processed by a single reducer, `ORDER BY` can be slow for large datasets and may lead to performance bottlenecks.
- **Use Case**: It's generally used when you need the final result to be completely ordered.

```
hive> select id, salary from data order by salary desc;
```

```
4       5000
3       3000
2       2000
1       1000
```

SORT BY:

- **Partition-Level Sorting**: `SORT BY` sorts the data within each reducer (or partition). This means that the data is sorted only at the level of the individual reducer, not globally.
- **Multiple Reducers**: The sorting happens independently within each reducer, so it is more scalable and can be faster than `ORDER BY` in large datasets because the data is distributed and not handled by a single reducer.
- **Use Case**: It's used when you want sorted data within each partition but not necessarily in a global order.

```
hive> select id salary from data sort by salary desc;
```

```
4       5000
3       3000
2       2000
1       1000
```

Tutorial 12

BUCKETING IN HIVE

In Hive, **bucketing** is a technique used to decompose large tables into more manageable parts, called **buckets**. It helps in improving query performance by allowing Hive to load only a subset of data during a query, instead of scanning the entire table. Bucketing is similar to partitioning but is based on the value of a column (or multiple columns).

- Creating table

```
hive> create table emp_data (fname string ,lname string ,phone bigint , email string , city string ,country string ) row format delimited fields terminated  by ',';
OK
Time taken: 0.185 seconds
```

- **Loading data,**

```
hive> load data local inpath '/data/emp_data.txt' into table emp_data ;
Loading data to table org.emp_data
Table org.emp_data stats: [numFiles=1, totalSize=614]
OK
Time taken: 0.697 seconds
```

- **Buckting**

```
hive> set hive.enforce.buckting = true;
```

- **Fetching all data.**

```
hive> select * from emp_data ;
OK
John    Doe     1234567890      john.doe@example.com    New York        USA
Jane    Smith   9876543210      jane.smith@example.com  London  UK
Raj     Patel   919876543210    raj.patel@example.com   Mumbai  India
Maria   Gonzalez        34123456789     maria.gonzalez@example.com      New York        USA
Wei     Chen    8613812345678   wei.chen@example.com    Beijing China
Fatima  Zahra   212612345678    fatima.zahra@example.com        London  UK
Ahmed   Khan    923001234567    ahmed.khan@example.com  Mumbai  India
Emily   Brown   61312345678     emily.brown@example.com New York        USA
Oliver  Martin  331234567890    oliver.martin@example.com       Beijing China
Liam    Nguyen  84901234567     liam.nguyen@example.com New York        USA
```

- Creating buckets

```
create table bucket (fname string,lname string , phone bigint,email string , city string ,country string ) clustered by (country) into 4 buckets row format delimited fields terminated by ',' ;
```

```
hive> create table data(fname string,lname string,phone string,email string,city string, country string)  row format delimited
    > fields terminated by ' ';
OK
Time taken: 0.063 seconds
hive> load data inpath '/user/cloudera/bucket' into table data;
Loading data to table func.data
Table func.data stats: [numFiles=1, totalSize=445]
OK
Time taken: 0.14 seconds
```

```
hive> insert into table bucket
    > select* from data;
Query ID = cloudera_20250422001313_eab62846
Total jobs = 1
Launching Job 1 out of 1
```

```
hive> select*from bucket;
```

```
Kartik  Goenka  65432   kr@gg.com       rajasthan       india
Reyansh Khurana 98653   ry@sh.com       mumbai  india
Radha   Rani    5643    rd@hi.com       vrindavan       india
```

```
hive> desc formatted bucket;
OK
# col_name              data_type               comment

fname                   string
lname                   string
phone                   string
email                   string
city                    string
country                 string

# Detailed Table Information
Database:               func
Owner:                  cloudera
CreateTime:             Tue Apr 22 00:06:12 PDT 2025
LastAccessTime:         UNKNOWN
Protect Mode:           None
Retention:              0
Location:               hdfs://quickstart.cloudera:8020/user/hive/warehouse/func.db/bucket
Table Type:             MANAGED_TABLE
Table Parameters:
        COLUMN_STATS_ACCURATE   true
        numFiles                4
        numRows                 10
        rawDataSize             435
        totalSize               445
        transient_lastDdlTime   1745306051
```

```
hive> dfs -ls hdfs://quickstart.cloudera:8020/user/hive/warehouse/func.db/bucket;
Found 4 items
-rwxrwxrwx   1 cloudera supergroup         96 2025-04-22 00:14 hdfs://quickstart.cloudera:8020/user/hive/warehouse/func.db/bucket/000000_0
-rwxrwxrwx   1 cloudera supergroup         88 2025-04-22 00:14 hdfs://quickstart.cloudera:8020/user/hive/warehouse/func.db/bucket/000001_0
-rwxrwxrwx   1 cloudera supergroup         43 2025-04-22 00:14 hdfs://quickstart.cloudera:8020/user/hive/warehouse/func.db/bucket/000002_0
-rwxrwxrwx   1 cloudera supergroup        218 2025-04-22 00:14 hdfs://quickstart.cloudera:8020/user/hive/warehouse/func.db/bucket/000003_0
hive>
```

```
hive> dfs -cat hdfs://quickstart.cloudera:8020/user/hive/warehouse/func.db/bucket/000000_0;
Percy Frenadez 65433 pr@ec.com manilla phillipnes
Keifer Watson 87653 kf@wi.com cebu phillipnes
hive> dfs -cat hdfs://quickstart.cloudera:8020/user/hive/warehouse/func.db/bucket/000002_0;
Inayat Khan 43255 in@rt.com toronto canada
hive>
```

Tutorial 16

HIVE FUNCTION

- **Gives of second elapsed from 1-1-1970 to 04-10-2024.**

```
hive> SELECT UNIX_TIMESTAMP();
unix_timestamp(void) is deprecated. Use current_timestamp instead.
OK
1745227717
```

- Converts second elapsed to actual date and time.

```
hive> SELECT FROM_UNIXTIME(UNIX_TIMESTAMP());
unix_timestamp(void) is deprecated. Use current_timestamp instead.
OK
```

- UTC to IST coversion.

```
hive> SELECT FROM_UNIXTIME(UNIX_TIMESTAMP());
unix_timestamp(void) is deprecated. Use current_timestamp instead.
OK
2025-04-21 02:29:45
Time taken: 0.047 seconds, Fetched: 1 row(s)
```

- IST to UTC

```
hive> SELECT TO_UTC_TIMESTAMP('2025-04-10','IST');
OK
2025-04-09 18:30:00
Time taken: 0.059 seconds, Fetched: 1 row(s)
```

- General date function
- Extracs date only

```
hive> SELECT TO_DATE('2024-08-11 11:32:20');
OK
2024-08-11
Time taken: 0.044 seconds, Fetched: 1 row(s)
```

- 13th week of year

```
hive> SELECT WEEKOFYEAR('2024-08-11 11:32:20');
OK
32
Time taken: 0.05 seconds, Fetched: 1 row(s)
```

- Diff btw 2 dates

```
hive> SELECT DATEDIFF('2024-08-11', '2024-08-11');
OK
0
Time taken: 0.432 seconds, Fetched: 1 row(s)
hive> SELECT DATEDIFF('2024-08-11', '2020-04-11');
OK
1583
```

- Date_add

```
hive> SELECT DATE_ADD('2024-08-11',9);
OK
2024-08-20
Time taken: 0.045 seconds, Fetched: 1 row(s)
```

- Date_sub

```
hive> SELECT DATE_SUB('2024-08-11',9);
OK
2024-08-02
Time taken: 0.05 seconds, Fetched: 1 row(s)
```

TUTORIAL 17

- DISPLAY Column name in hive output.

```
hive> select* from ppl;
OK
```

```
hive> set hive.cli.print.header=true;
```

```
hive> select* from ppl limit 5;
OK
fname   lname   state
```

Tutorial 18

Execute hive queries from file

1.

```
hive> [root@quickstart cloudera]# vi test_hive.hql
```

2. Creating table

```
CREATE DATABASE soft_engg;
USE soft_engg;
CREATE TABLE tech_tools(id INT, lang STRING, paradigm STRING);
INSERT INTO tech_tools values(1, 'C Lang', 'POP'),
(2, 'c++', 'OOP'),
(3, 'Java', 'OOP'),
(4, 'LISP', 'FOP'),
(5, 'MySql', 'SQL');
SELECT * FROM tech_tools;
```

```
[root@quickstart cloudera]# hive
```

```
hive> SHOW DATABASES;
OK
default
emp
employee
ipl_db
lab1
personal
Time taken: 0.154 seconds, Fetched: 6 row(s)
```

No entry of soft_engg

- Enter below given command and queries will run in sequence.

```
[root@quickstart cloudera]# hive -f test_hive.hql;
```

- Show databases.

```
hive> SHOW DATABASES;
OK
default
emp
employee
ipl_db
lab1
personal
soft_engg
Time taken: 0.271 seconds, Fetched: 7 row(s)
```

- Display all data

```
1       C Lang  POP
2       c++     OOP
3       Java    OOP
4       LISP    FOP
5       MySql   SQL
Time taken: 0.15 seconds, Fetched: 5 row(s)
```

- Show table

```
hive> SHOW TABLES;
OK
tech_tools
Time taken: 0.01 seconds, Fetched: 1 row(s)
```

TUTORIAL 19

- **RUNING HDFS COMMANDS IN HIVE**
- Running linux command.

```
hive> !ls;
cloudera-manager
cm_api.py
Desktop
Documents
Downloads
driver
eclipse
emp_dept
employee
enterprise-deployment.json
express-deployment.json
famous
function
ipl
kerberos
learn.txt
lib
```

- Running HDFS command

```
hive> !hadoop fs -ls /user
    > ;
Found 8 items
drwxr-xr-x   - cloudera cloudera            0 2025-04-21 23:02 /user/cloudera
drwxr-xr-x   - mapred   hadoop              0 2017-10-23 09:15 /user/history
drwxrwxrwx   - hive     supergroup          0 2017-10-23 09:17 /user/hive
drwxrwxrwx   - hue      supergroup          0 2017-10-23 09:16 /user/hue
drwxrwxrwx   - jenkins  supergroup          0 2017-10-23 09:15 /user/jenkins
drwxrwxrwx   - oozie    supergroup          0 2017-10-23 09:16 /user/oozie
drwxrwxrwx   - root     supergroup          0 2025-03-31 21:05 /user/root
drwxr-xr-x   - hdfs     supergroup          0 2017-10-23 09:17 /user/spark
hive>
```

Tutorial 20

Store hive query output in a file on your linux file systeem

1.

```
hive> [root@quickstart cloudera]# vi ret_data.hql
```

2.

```
[cloudera@quickstart ~]$ ls -ltr
total 496
```

3.

```
-rw-r--r-- 1 root     root         37 Apr 22 02:37 ret_data.hql
-rw-rw-r-- 1 cloudera cloudera    445 Apr 21 23:26 bucket
-rw-rw-r-- 1 cloudera cloudera    158 Apr 22 01:47 info
-rw-rw-r-- 1 cloudera cloudera      0 Apr 22 02:15 test
-rw-rw-r-- 1 cloudera cloudera    209 Apr 22 02:19 test.hql
```

4.

```
[root@quickstart cloudera]# hive -f ret_data.hql>>techtools.csv

Logging initialized using configuration in file:/etc/hive/conf.dist/hive-log4j.properties
OK
Time taken: 0.52 seconds, Fetched: 5 row(s)
```

5.

```
[root@quickstart cloudera]# ls -ltr
total 764
```

6.

```
-rw-r--r-- 1 root     root        228 Apr 22 02:39 techtools.csv
```

TUTORIAL – 21

HIVE explode function|lateral vivew|hive array Explode()

- **Show databases and create table dev_tools**

```
hive> Show databases;
OK
default
emp
employee
ipl_db
lab1
personal
soft_engg
Time taken: 0.274 seconds, Fetched: 7 row(s)
hive> use soft_engg;
OK
Time taken: 0.016 seconds
hive> CREATE TABLE dev_tools(
    > category STRING,
    > technology array<string>)
    > ROW FORMAT DELIMITED
    > FIELDS TERMINATED BY '\t'
    > COLLECTION ITEMS TERMINATED BY '.';
OK
Time taken: 0.091 seconds
```

- **Make data file in linux named devtools**

```
hive> [root@quickstart cloudera]# cat > devtools
programming_language    Java.Python.C
SQL     Mysql.Oracle.DB2
^Z
[28]+  Stopped                 cat > devtools
```

- **Describe table**

```
hive> DESC dev_tools;
OK
category                string
technology              array<string>
Time taken: 0.171 seconds, Fetched: 2 row(s)
```

- **Load the file data in tne table**

```
hive> LOAD DATA LOCAL INPATH 'devtools' INTO TABLE dev_tools;
Loading data to table soft_engg.dev_tools
Table soft_engg.dev_tools stats: [numFiles=1, totalSize=56]
OK
Time taken: 0.546 seconds
```

- **Display all**

```
hive> select * from dev_tools;
OK
programming_language    ["Java","Python","C"]
SQL     ["Mysql","Oracle","DB2"]
Time taken: 0.201 seconds, Fetched: 2 row(s)
```

- **For lateral view**

```
hive> SELECT category, techno
    > from dev_tools
    > LATERAL VIEW explode(technology) teachnoView
    > AS techno;
OK
programming_language    Java
programming_language    Python
programming_language    C
SQL     Mysql
SQL     Oracle
SQL     DB2
Time taken: 0.07 seconds, Fetched: 6 row(s)
```

TUTORIAL – 23

DELETE & REPLACEC columns in HIVE

- **Select table employee**

```
hive> select * from employee;
OK
```

```
1       meenu   pune    engr    1000    bigdata
2       riya    goa     dev     2000    bfsi
3       pooja   bnglr   rec     3000    hr
4       priya   chennai sales   5000    NULL
Time taken: 0.039 seconds, Fetched: 4 row(s)
```

- **Describe emp**

```
hive> desc employee;
OK
id                      int
name                    string
city                    string
department              string
salary                  int
domain                  string
Time taken: 0.031 seconds, Fetched: 6 row(s)
```

- Alter the table

```
hive> alter table employee replace columns(
    > id INT,
    > name STRING,
    > department STRING,
    > salary INT,
    > domain STRING);
OK
Time taken: 0.097 seconds
```

- **Replace eid & ename**

```
hive> ALTER TABLE employee replace columns(
    > eid INT,
    > ename STRING);
OK
Time taken: 0.06 seconds
```

- **Result**

```
hive> desc employee;
OK
eid                     int
ename                   string
Time taken: 0.029 seconds, Fetched: 2 row(s)
```

TUTORIAL – 24

Collection function in HIVE|SIZE|MAP_KEYS|MAP_KEYS|MAP_VALUES|CONATIN|SORT

- **Create the following file in Linux.**

```
hive> [root@quickstart cloudera]# cat > datafile.txt
1, Sachin|Saurabh|Rahul, branch:IT|year:SE
2, Vinay|Ajinkya|Rohan, branch:Comp|year:TE
3, Ganesh, branch:MECH|year:FE
4, Gaurav|Satish|Bhima, branch:ETC|year:SE
5, ,year:BE
^Z
[26]+  Stopped                 cat > datafile.txt
```

- Create table if not exist

```
hive> CREATE TABLE IF NOT EXISTS course_info(
    > id INT,
    > names array<string>,
    > course_details map<string, string>
    > )
    > ROW FORMAT DELIMITED
    > FIELDS TERMINATED BY ','
    > COLLECTION ITEMS TERMINATED BY '|'
    > MAP KEYS TERMINATED BY ':';
OK
Time taken: 0.833 seconds
```

- Load the file in the above table

```
hive> LOAD DATA LOCAL INPATH 'datafile.txt' INTO table course_info;
Loading data to table default.course_info
Table default.course_info stats: [numFiles=1, totalSize=173]
OK
Time taken: 0.384 seconds
```

- Show data

```
hive> SELECT * FROM course_info;
OK
1       [" Sachin","Saurabh","Rahul"]   {" branch":"IT","year":"SE"}
2       [" Vinay","Ajinkya","Rohan"]    {" branch":"Comp","year":"TE"}
3       [" Ganesh"]     {" branch":"MECH","year":"FE"}
4       [" Gaurav","Satish","Bhima"]    {" branch":"ETC","year":"SE"}
5       [" "]   {"year":"BE"}
Time taken: 0.251 seconds, Fetched: 5 row(s)
```

- **Select all from course info**

```
hive> SELECT names,size(names) as size FROM course_info;
OK
[" Sachin","Saurabh","Rahul"]   3
[" Vinay","Ajinkya","Rohan"]    3
[" Ganesh"]     1
[" Gaurav","Satish","Bhima"]    3
[" "]   1
Time taken: 0.064 seconds, Fetched: 5 row(s)
```

-

```
hive> SELECT course_details,size(course_details) as size FROM course_info;
OK
{" branch":"IT","year":"SE"}    2
{" branch":"Comp","year":"TE"}  2
{" branch":"MECH","year":"FE"}  2
{" branch":"ETC","year":"SE"}   2
{"year":"BE"}   1
Time taken: 0.053 seconds, Fetched: 5 row(s)
```

```
hive> SELECT course_details,map_keys(course_details) as map_keys FROM course_info;
OK
{" branch":"IT","year":"SE"}    [" branch","year"]
{" branch":"Comp","year":"TE"}  [" branch","year"]
{" branch":"MECH","year":"FE"}  [" branch","year"]
{" branch":"ETC","year":"SE"}   [" branch","year"]
{"year":"BE"}   ["year"]
Time taken: 0.048 seconds, Fetched: 5 row(s)
```

```
hive> SELECT course_details,map_values(course_details) as map_values FROM course_info;
OK
{" branch":"IT","year":"SE"}    ["IT","SE"]
{" branch":"Comp","year":"TE"}  ["Comp","TE"]
{" branch":"MECH","year":"FE"}  ["MECH","FE"]
{" branch":"ETC","year":"SE"}   ["ETC","SE"]
{"year":"BE"}   ["BE"]
Time taken: 0.049 seconds, Fetched: 5 row(s)
```

```
hive> SELECT names,array_contains(names, 'Saurabh') FROM course_info;
OK
[" Sachin","Saurabh","Rahul"]   true
[" Vinay","Ajinkya","Rohan"]    false
[" Ganesh"]     false
[" Gaurav","Satish","Bhima"]    false
[" "]   false
Time taken: 0.048 seconds, Fetched: 5 row(s)
```

```
hive> SELECT sort_array(names) FROM course_info;
OK
[" Sachin","Rahul","Saurabh"]
```

TUTORIAL – 25

Handling null values in hive|is null(), isnot null() | null()

- **Table creation**

```
hive> CREATE TABLE IF NOT EXISTS employees(
    > id INT,
    > name STRING,
    > age INT,
    > gender STRING,
    > salary DECIMAL
    > )
    > ROW FORMAT DELIMITED
    > FIELDS TERMINATED BY ',';
OK
Time taken: 0.078 seconds
```

- Create data file in linux env.

```
hive> [root@quickstart cloudera]# cat> employee.txt
1, Santosh, 30, M, 10000
2, Vaibhav, 40, F, 20000
3, Rohan, 41, M, 30000
4, Akshay, 20, F, 40000
5, Sam, 30, M
7, Ana,,,
^Z
[27]+  Stopped                 cat > employee.txt
```

- Load data

```
hive> LOAD DATA LOCAL INPATH 'employee.txt' INTO TABLE employees;
Loading data to table default.employees
Table default.employees stats: [numFiles=1, totalSize=121]
OK
Time taken: 0.599 seconds
```

- Apply queries.

```
hive> SELECT name, isnull(age) from employees;
OK
 Santosh        true
 Vaibhav        true
 Rohan  true
 Akshay true
 Sam    true
 Ana    true
Time taken: 0.227 seconds, Fetched: 6 row(s)
```

```
hive> SELECT name, isnotnull(age) from employees;
OK
 Santosh        false
 Vaibhav        false
 Rohan  false
 Akshay false
 Sam    false
 Ana    false
Time taken: 0.631 seconds, Fetched: 6 row(s)
```

```
hive> SELECT name from employees where isnull(salary);
OK
 Santosh
 Vaibhav
 Rohan
 Akshay
 Sam
 Ana
Time taken: 0.061 seconds, Fetched: 6 row(s)
```

```
hive> SELECT name, nvl(age,-99) from employees;
OK
 Santosh        -99
 Vaibhav        -99
 Rohan  -99
 Akshay -99
 Sam    -99
 Ana    -99
Time taken: 0.048 seconds, Fetched: 6 row(s)
```

```
hive> SELECT name, nvl(age,-9999) from employees;
OK
 Santosh        -9999
 Vaibhav        -9999
 Rohan  -9999
 Akshay -9999
 Sam    -9999
 Ana    -9999
Time taken: 0.78 seconds, Fetched: 6 row(s)
```