INDEX

PRACTICAL NO:1

AIM: To Install Cloudera Quickstart VM on VirtualBox

Theory:
Cloudera is a software that provides a platform for data analytics, data warehousing, and machine learning. Initially, Cloudera started as an open-source Apache Hadoop distribution project, commonly known as Cloudera Distribution for Hadoop or CDH. It contains Apache Hadoop and other related projects where all the components are 100% open-source under Apache License.

Cloudera provides virtual machine images of complete Apache Hadoop clusters, making it easy to get started with Cloudera CDH. The following topics will be covered in this assignment on Cloudera QuickStart VM Installation.

1. What is Cloudera QuickStart VM?
2. Cloudera QuickStart VM Installation - Prerequisites
3. Downloading the Cloudera QuickStart VM
4. Cloudera QuickStart VM Installation on windows

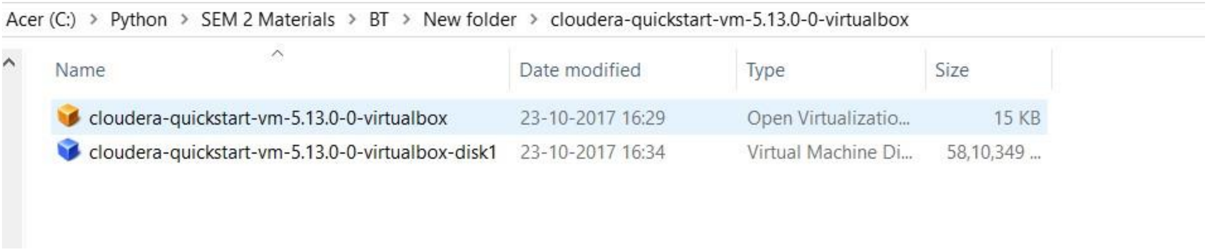
What Is Cloudera QuickStart VM?
Cloudera QuickStart VM includes everything that you would need for using CDH, Impala, Cloudera Search, and Cloudera Manager. The Cloudera QuickStart VM uses a package-based install that allows you to work with or without the Cloudera Manager. It has a sample of Cloudera's platform for "Big Data."

Cloudera QuickStart VM Installation - Prerequisites
A virtual machine such as Oracle Virtual Box or VMWare
RAM of 12+ GB. That is 4+ GB for the operating system and 8+ GB for Cloudera
80GB hard disk

Download Oracle Virtual Box from https://www.virtualbox.org/wiki/Downloads and install it in your system

Downloading the Cloudera QuickStart VM

- ✓ The Cloudera QuickStart VMs are openly available as Zip archives in VirtualBox, VMware and KVM formats. To download the VM, search for https://www.cloudera.com/downloads.html, and select the appropriate version of CDH that you require.
- ✓ Click on the 'GET IT NOW' button, and it will prompt you to fill in your details.
- ✓ Once the file is downloaded, go to the download folder and unzip these files. It can then be used to set up a single node Cloudera cluster.
- ✓ Shown below are the two virtual images of Cloudera QuickStart VM.



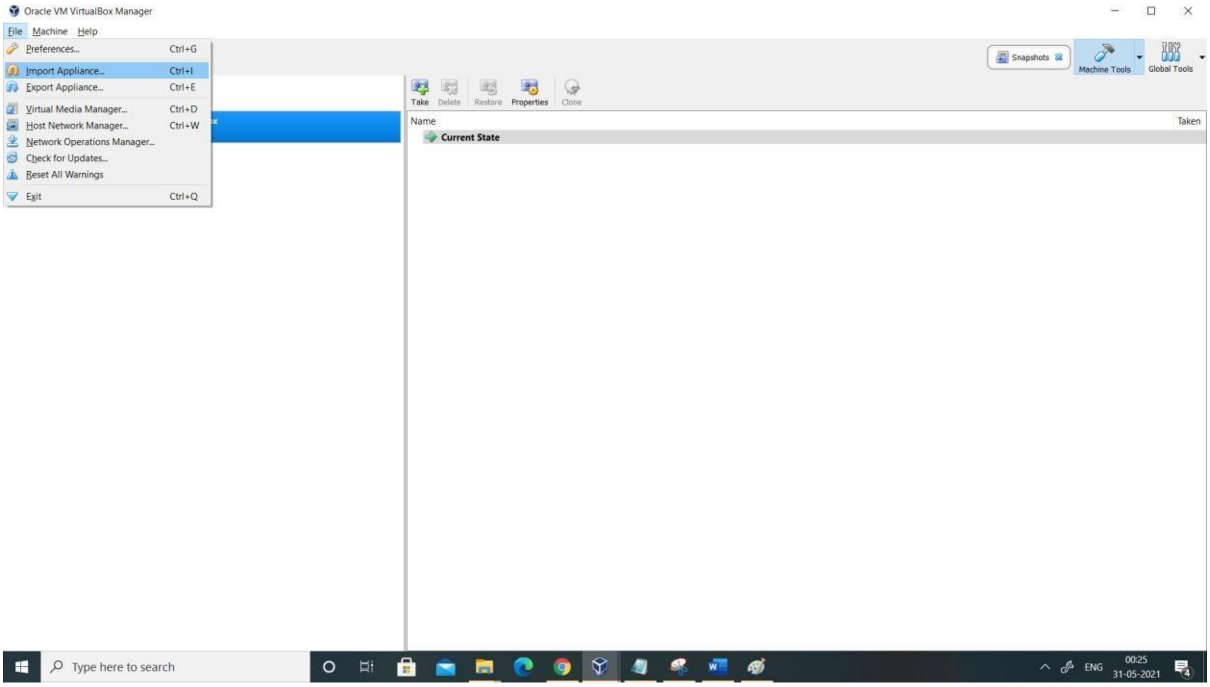
Acer (C:) > Python > SEM 2 Materials > BT > New folder > cloudera-quickstart-vm-5.13.0-0-virtualbox

Name	Date modified	Type	Size
cloudera-quickstart-vm-5.13.0-0-virtualbox	23-10-2017 16:29	Open Virtualizatio...	15 KB
cloudera-quickstart-vm-5.13.0-0-virtualbox-disk1	23-10-2017 16:34	Virtual Machine Di...	58,10,349 ...

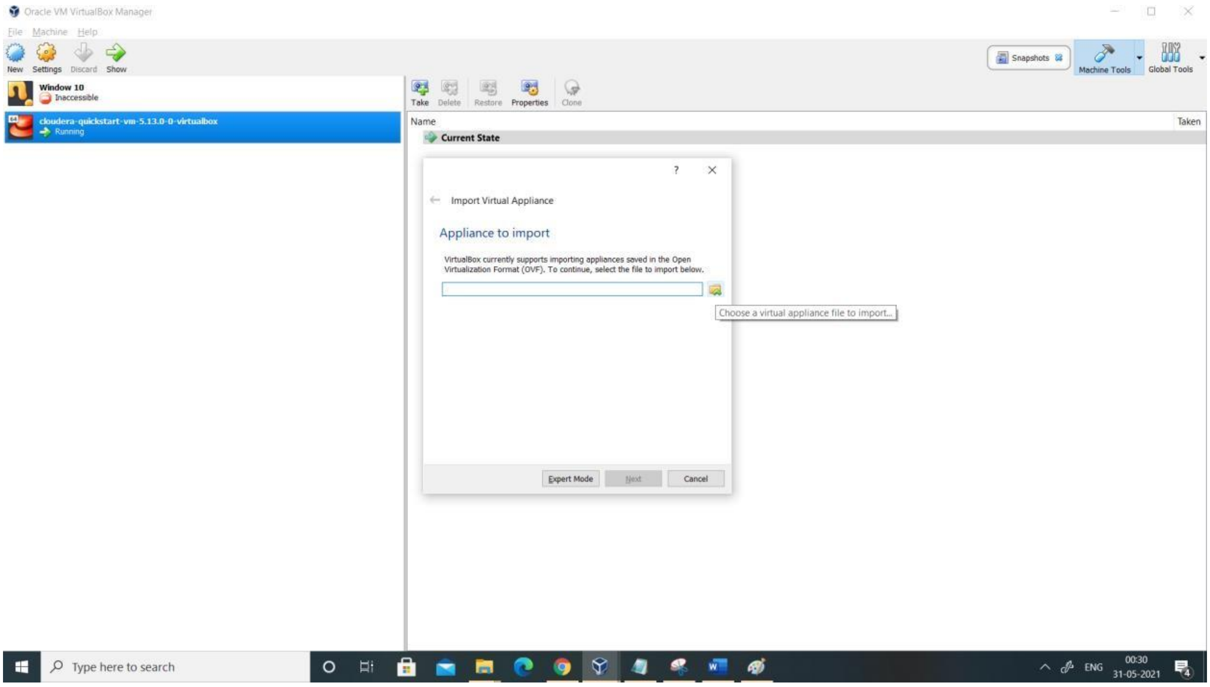
- ✓ Now that the downloading process is done with, let's move forward with this Cloudera QuickStart VM Installation guide and see the actual process.

Cloudera QuickStart VM Installation

- ✓ Before setting up the Cloudera Virtual Machine, you would need to have a virtual machine such as VMware or Oracle VirtualBox on your system.
- ✓ In this case, we are using Oracle VirtualBox to set up the Cloudera QuickStart VM.
- ✓ In order to download and install the Oracle VirtualBox on your operating system, click on the following link: Oracle VirtualBox(https://www.virtualbox.org/wiki/Downloads).
- ✓ To set up the Cloudera QuickStart VM in your Oracle VirtualBox Manager, click on 'File' and then select 'Import Appliance'.



- ✓ Choose the QuickStart VM image by looking into your downloads. Click on 'Open' and then 'Next'. Now you can see the specifications, then click on 'Import'. This will start importing the virtual disk image .vmdk file into your VM box.

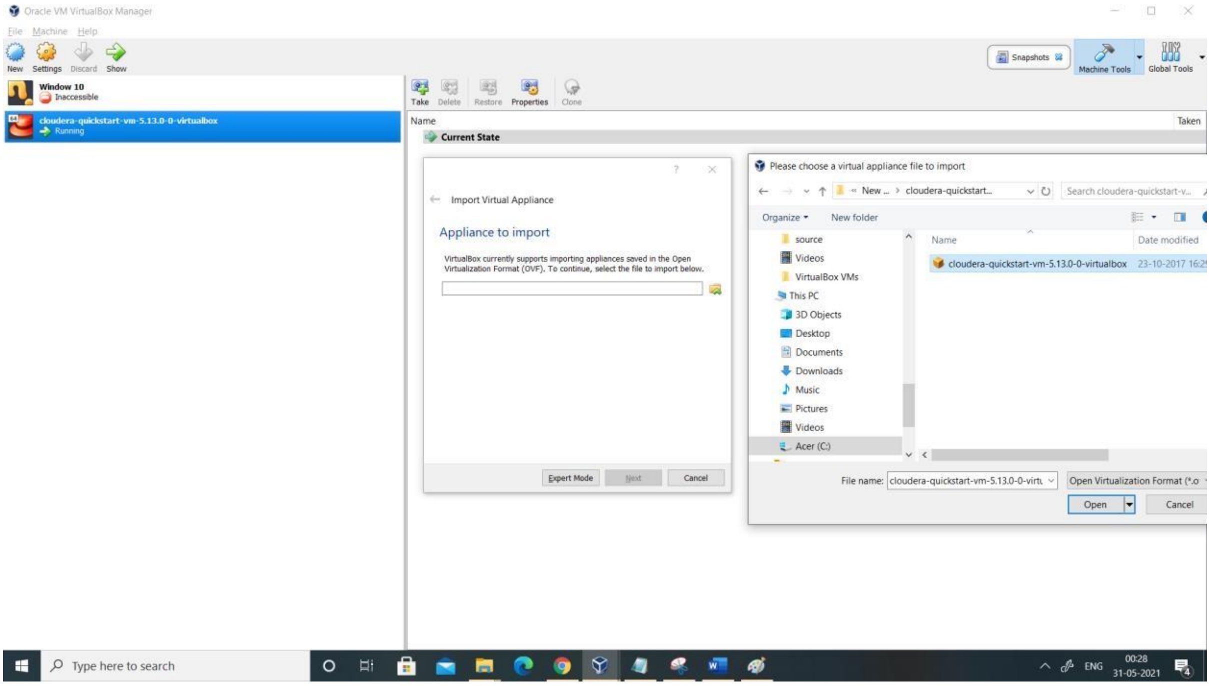


Import Virtual Appliance

Appliance to import

VirtualBox currently supports importing appliances saved in the Open Virtualization Format (OVF). To continue, select the file to import below.

Choose a virtual appliance file to import...



Import Virtual Appliance

Appliance to import

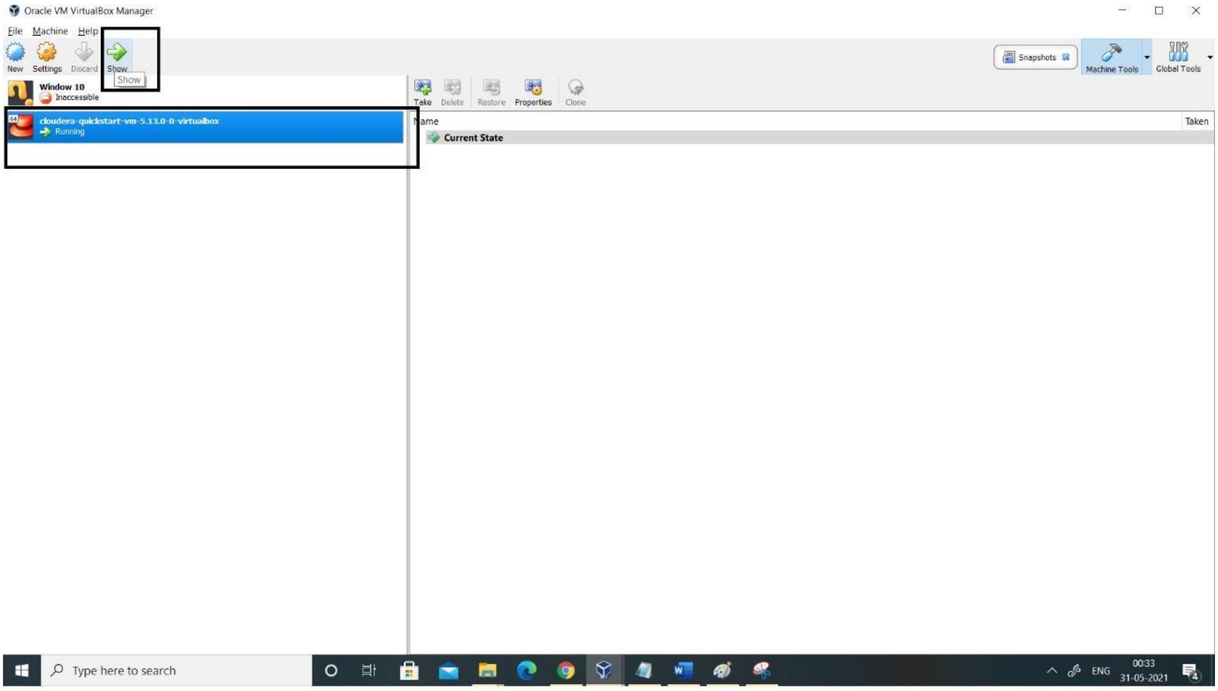
VirtualBox currently supports importing appliances saved in the Open Virtualization Format (OVF). To continue, select the file to import below.

Please choose a virtual appliance file to import

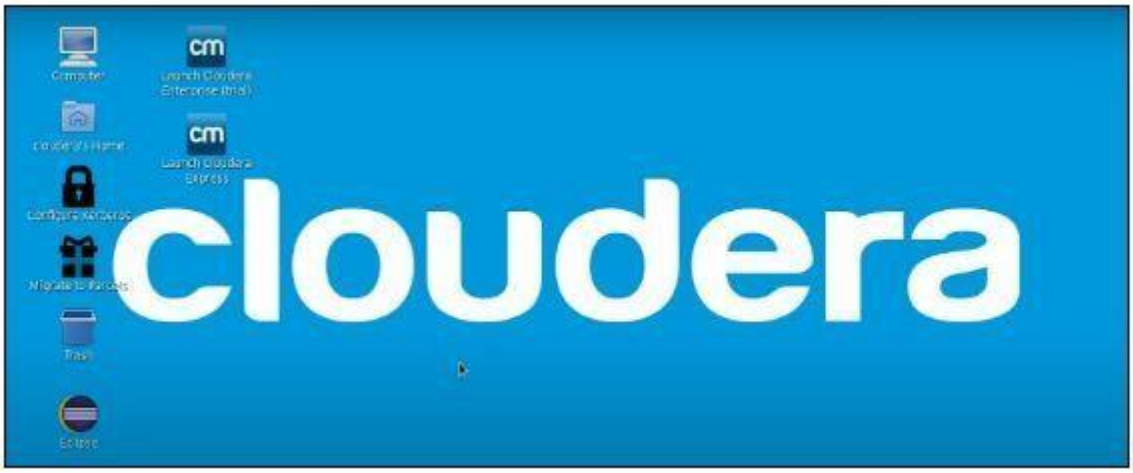
File name: cloudera-quickstart-vm-5.13.0-0-virtu

Open Virtualization Format (*.o

- ✓ Click on "Open" and Wait for a while, as the importing finishes.
- ✓ The next step is to go ahead and set up a Cloudera QuickStart VM for practice. Once the importing is complete, you can see the Cloudera QuickStart VM on the left side panel.



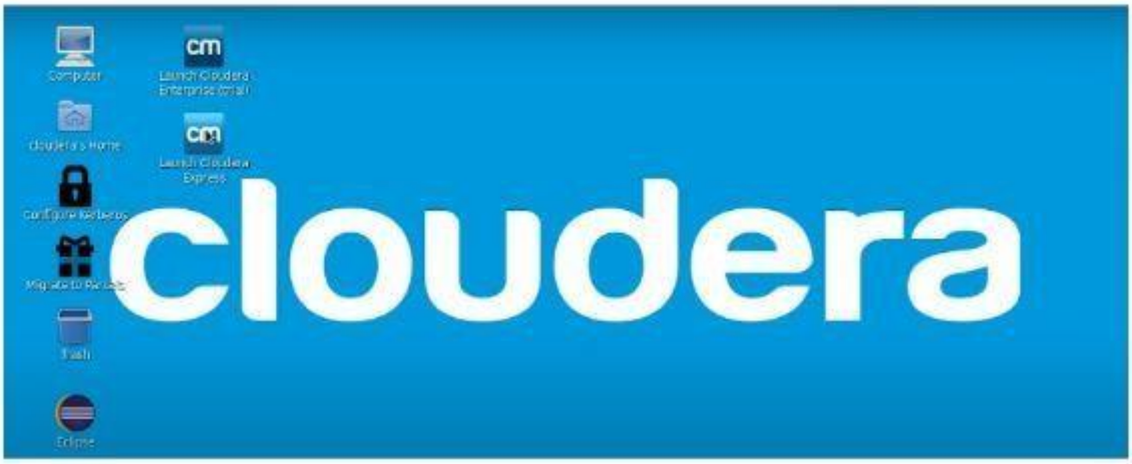
- ✓ The next step will be going ahead and starting the machine by clicking the 'Start' symbol on top.
- ✓ Once your machine comes on, it will look like this:



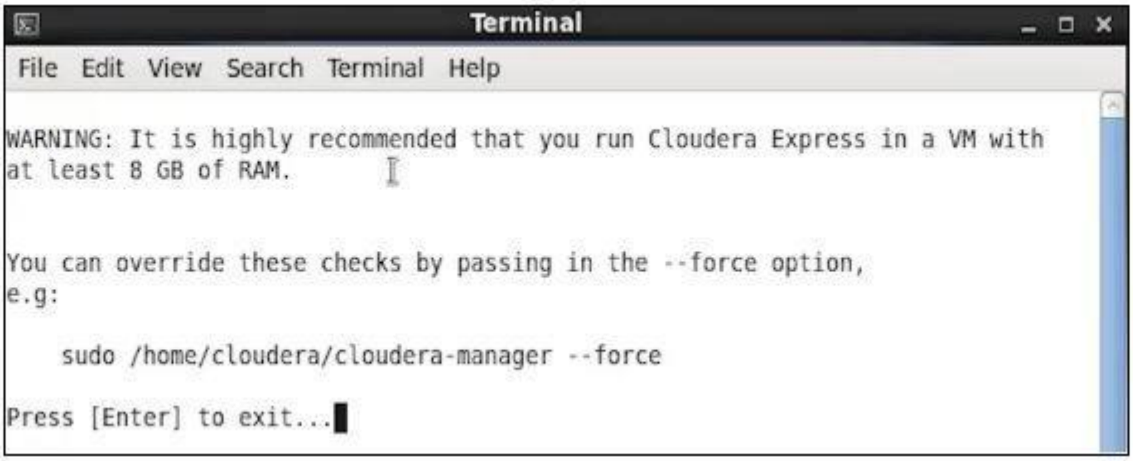
- ✓ Next, we have to follow a few steps to gain admin console access. You need to click on the terminal present on top of the desktop screen, and type in the following:

1. hostname # This shows the hostname which will be quickstart.cloudera
2. hdfs dfs -ls / # Checks if you have access and if your cluster is working. It displays what exists on your HDFS location by default
3. service cloudera-scm-server status # Tells what command you have to type to use cloudera express free
4. su - #Login as root
5. service cloudera-scm-server status # The password for root is cloudera

- ✓ Once you see that your HDFS access is working fine, you can close the terminal. Then, you have to click on the following icon that says ‘Launch Cloudera Express’.

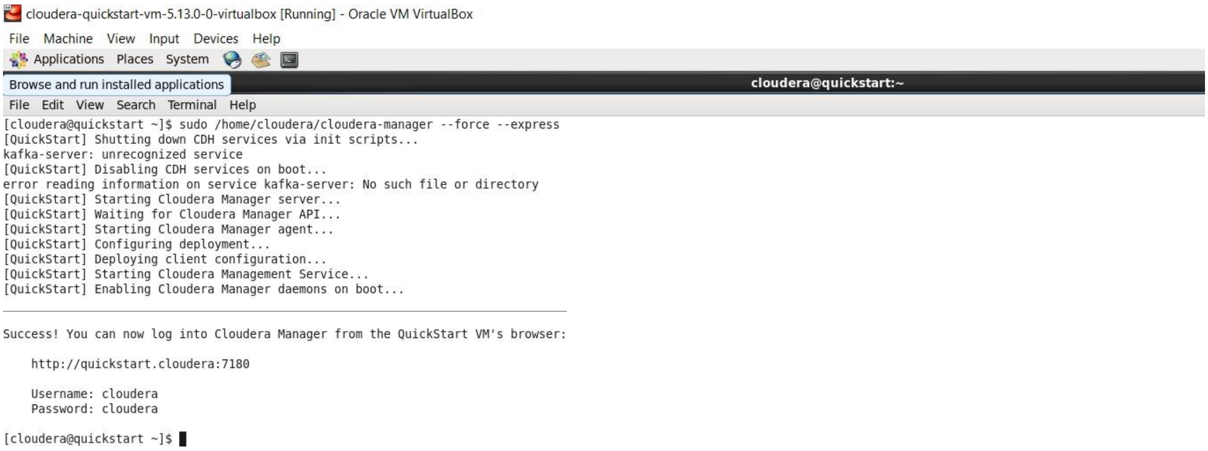


- ✓ Once you click on the express icon, a screen will appear with the following command:



```
WARNING: It is highly recommended that you run Cloudera Express in a VM with
at least 8 GB of RAM.

You can override these checks by passing in the --force option,
e.g:

   sudo /home/cloudera/cloudera-manager --force

Press [Enter] to exit...
```

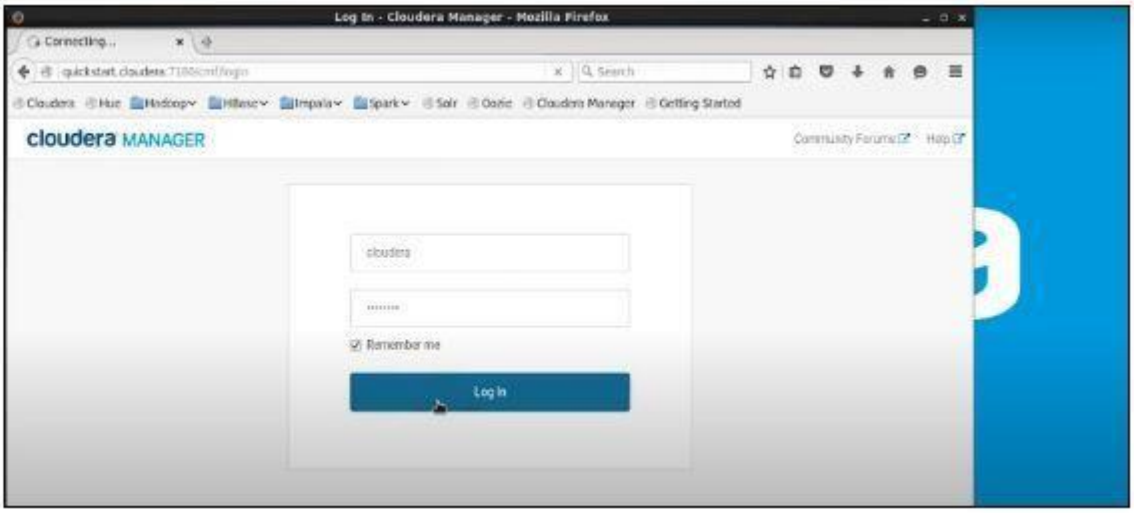
- ✓ You are required to copy the command, and run it on a separate terminal. Hence, open a new terminal, and use the below command to close the Cloudera based services. It will restart the services, after which you can access your admin console.


cloudera-quickstart-vm-5.13.0-0-virtualbox [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Applications Places System
Browse and run installed applications
cloudera@quickstart:~
File Edit View Search Terminal Help

```
[cloudera@quickstart ~]$ sudo /home/cloudera/cloudera-manager --force --express
[QuickStart] Shutting down CDH services via init scripts...
kafka-server: unrecognized service
[QuickStart] Disabling CDH services on boot...
error reading information on service kafka-server: No such file or directory
[QuickStart] Starting Cloudera Manager server...
[QuickStart] Waiting for Cloudera Manager API...
[QuickStart] Starting Cloudera Manager agent...
[QuickStart] Configuring deployment...
[QuickStart] Deploying client configuration...
[QuickStart] Starting Cloudera Management Service...
[QuickStart] Enabling Cloudera Manager daemons on boot...
________________________________________________________________________________

Success! You can now log into Cloudera Manager from the QuickStart VM's browser:

    http://quickstart.cloudera:7180

    Username: cloudera
    Password: cloudera

[cloudera@quickstart ~]$ 
```

- ✓ Now that our deployment has been configured, client configurations have also been deployed. Additionally, it has restarted the Cloudera Management Service, which gives access to the Cloudera QuickStart admin console with the help of a username and password.

- ✓ Go on and open up the browser and change the port number to 7180.
- ✓ You can log in to the Cloudera Manager by providing your username and password.



cloudera

········

☑ Remember me

- ✓ You can go ahead and restart the services now. It will ensure that the cluster becomes accessible either by Hue as a web interface or Cloudera QuickStart Terminal, where you can write your commands.

PRACTICAL NO:2

AIM: Implementing Map-Reduce Program for Word Count Problem

Theory:
MapReduce is a programming model and processing framework designed for processing and generating large datasets that can be parallelized across a distributed cluster of computers. It was originally developed by Google and popularized by their 2004 paper titled "MapReduce: Simplified Data Processing on Large Clusters." MapReduce has been widely used in the big data processing field and has influenced the development of various distributed data processing systems, including Apache Hadoop.

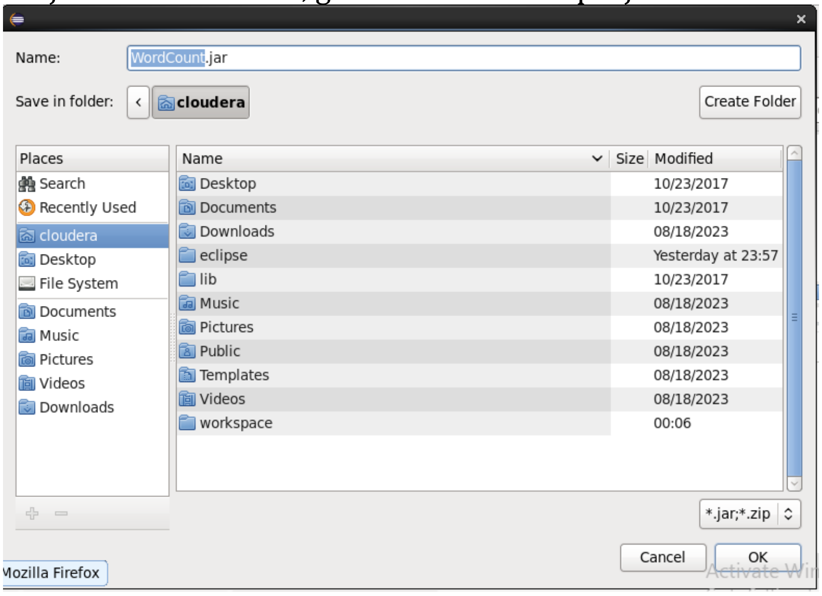
MapReduce is designed to be fault-tolerant and scalable, making it suitable for processing large datasets across distributed clusters of commodity hardware. It abstracts away many of the complexities of distributed computing, allowing developers to focus on writing the mapping and reducing functions for their specific data processing tasks.

While MapReduce has been instrumental in the world of big data processing, more modern data processing frameworks like Apache Spark have gained popularity due to their improved performance and support for a broader range of data processing tasks. Nonetheless, MapReduce remains an important concept in the history of distributed data processing.

Steps to perform the practical:
Step1: Open Cloudera.
Step2: Goto eclipse
Step3: Goto file>new>java project
Step4: Give name as WordCount, click on next and goto libraries click on add external jar
Step5: Goto file system>usr>lib>hadoop and select all the jar files and click on ok.
Step6: Again, click add external jar> file system>usr>lib>hadoop>client and select all the jar files and click on ok.
Step7: Again, click add external jar> file system>usr>lib>hadoop>client0.20 and select all the jar files and click on ok.
Step8: click on finish
Step9: Under java project right click on src>new>class give name same as project name,click finish
Step10: Write the code for word count, cntl+s to save file

```
import java.io.IOException;
import java.util.StringTokenizer;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class WordCount {
 public static class TokenizerMapper extends Mapper<Object, Text, Text, IntWritable>
{
  private final static IntWritable one = new IntWritable(1);
  private Text word = new Text();

  public void map(Object key, Text value, Context context) throws IOException,
InterruptedException {
   StringTokenizer itr = new StringTokenizer(value.toString());
   while (itr.hasMoreTokens()) {
    word.set(itr.nextToken());
    context.write(word, one);
   }
  }
 }
 public static class IntSumReducer extends Reducer<Text, IntWritable, Text,
IntWritable> {
  private IntWritable result = new IntWritable();

  public void reduce(Text key, Iterable<IntWritable> values, Context context) throws
IOException, InterruptedException {
   int sum = 0;
   for (IntWritable val : values) {
    sum += val.get();
   }
   result.set(sum);
   context.write(key, result);
  }
 }

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

Step11: Goto project name right click on it>export>java>jar file>click next >give path for jar file i.e cloudera, give name same as project name>click on>finish

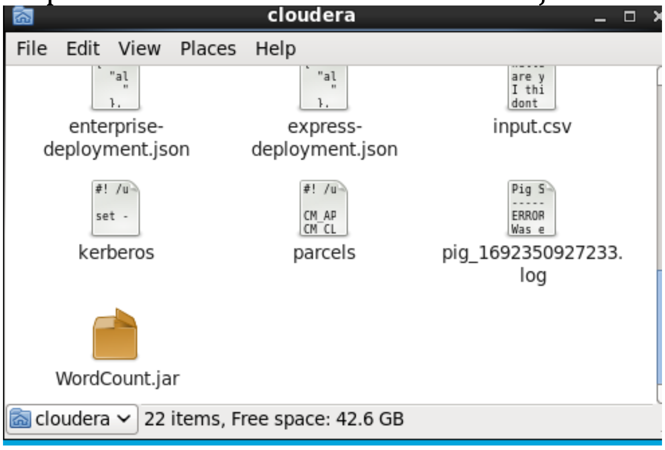


Name: WordCount.jar

Save in folder: cloudera

Name	Size	Modified
Desktop		10/23/2017
Documents		10/23/2017
Downloads		08/18/2023
eclipse		Yesterday at 23:57
lib		10/23/2017
Music		08/18/2023
Pictures		08/18/2023
Public		08/18/2023
Templates		08/18/2023
Videos		08/18/2023
workspace		00:06

Step12: Goto cloudera home and check jar created or not.



22 items, Free space: 42.6 GB

Step13: Open terminal fire ls command

Step14: To check list of files under Hadoop.

```
[cloudera@quickstart ~]$ hdfs dfs -ls
Found 1 items
drwx------   - cloudera cloudera          0 2023-09-01 02:43 .staging
```

Step15: Create folder with name inputdirectory.

```
[cloudera@quickstart ~]$ sudo -u hdfs hadoop fs -mkdir /mapreduce
```

Step16: To check folder created or not under Hadoop.

```
[cloudera@quickstart ~]$ hdfs dfs -ls /
Found 8 items
drwxrwxrwx   - hdfs  supergroup          0 2017-10-23 10:29 /benchmarks
drwxr-xr-x   - hbase supergroup          0 2023-09-29 23:23 /hbase
drwxr-xr-x   - hdfs  supergroup          0 2023-09-30 00:15 /mapreduce
```

Step17: Give permission to /inputdirectory folder.

```
[cloudera@quickstart ~]$ sudo -u hdfs hadoop fs -chmod -R 777 / mapreduce
```

Step18:To check permission granted or not under Hadoop.

```
[cloudera@quickstart ~]$ hdfs dfs -ls /
Found 8 items
drwxrwxrwx   - hdfs  supergroup        0 2017-10-23 10:29 /benchmarks
drwxr-xr-x   - hbase supergroup        0 2023-09-29 23:23 /hbase
drwxrwxrwx   - hdfs  supergroup        0 2023-09-30 00:15 / mapreduce
```

Step19: Check if directory contains any data (no output since directory is empty)

```
[cloudera@quickstart ~]$ hdfs dfs -ls / mapreduce
```

Step20: Create file in local storage, enter content and cntrl+z

```
[cloudera@quickstart ~]$ cat > /home/cloudera/input.txt
This salma salma
^Z
[1]+  Stopped                 cat > /home/cloudera/Processfile1.txt
```

Step21: To check the content in Processfile1.txt

```
[cloudera@quickstart ~]$ cat /home/cloudera/input.txt
This salma salma
```

Step22: Transfer the file into hdfs from cloudera home

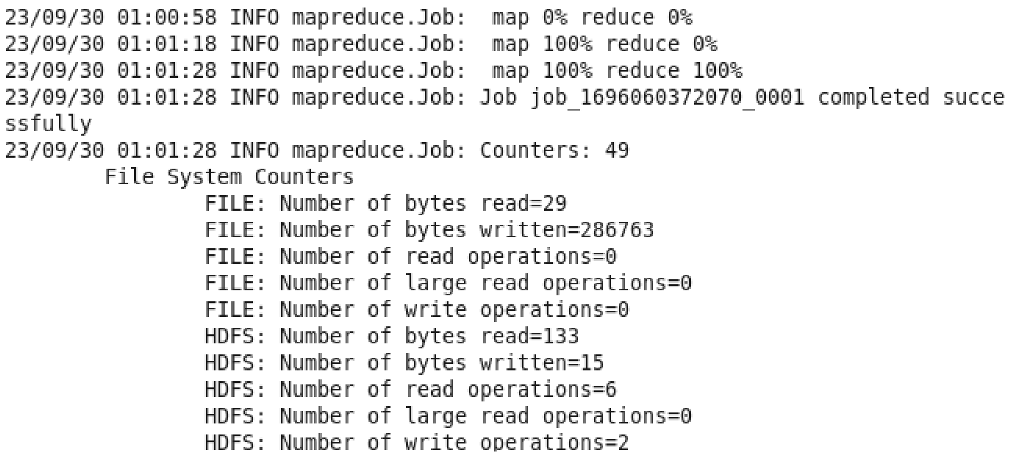
```
[cloudera@quickstart ~]$ sudo -u hdfs hadoop fs -put /home/cloudera/input.txt
/mapreduce
```

Step23: Check again if inputdirectory shows the file or not

```
[cloudera@quickstart ~]$ hdfs dfs -ls /mapreduce
Found 1 items
-rw-r--r--   1 hdfs supergroup        120 2023-09-30 00:22 /mapreduce/input.txt
```

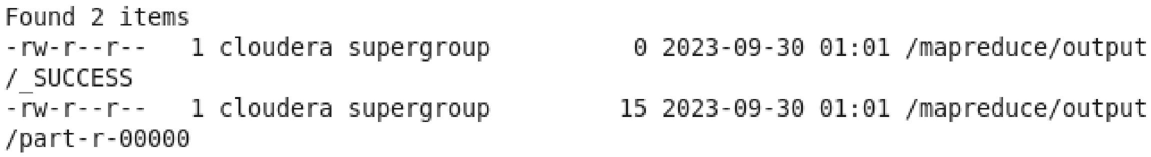
Step24: Run the wordcount jar file

```
[cloudera@quickstart ~]$ hadoop jar /home/cloudera/WordCount.jar WordCount
/mapreduce/input.txt /mapreduce/output
```



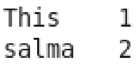
```
23/09/30 01:00:58 INFO mapreduce.Job:  map 0% reduce 0%
23/09/30 01:01:18 INFO mapreduce.Job:  map 100% reduce 0%
23/09/30 01:01:28 INFO mapreduce.Job:  map 100% reduce 100%
23/09/30 01:01:28 INFO mapreduce.Job: Job job_1696060372070_0001 completed succe
ssfully
23/09/30 01:01:28 INFO mapreduce.Job: Counters: 49
        File System Counters
                FILE: Number of bytes read=29
                FILE: Number of bytes written=286763
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=133
                HDFS: Number of bytes written=15
                HDFS: Number of read operations=6
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=2
```

Step25: Check output folder
[cloudera@quickstart ~]$ hdfs dfs -ls /mapreduce/output



```
Found 2 items
-rw-r--r--   1 cloudera supergroup          0 2023-09-30 01:01 /mapreduce/output
/_SUCCESS
-rw-r--r--   1 cloudera supergroup         15 2023-09-30 01:01 /mapreduce/output
/part-r-00000
```

Step26: Check output result
[cloudera@quickstart ~]$ hdfs dfs -cat /mapreduce/output/part-r-00000



```
This    1
salma   2
```

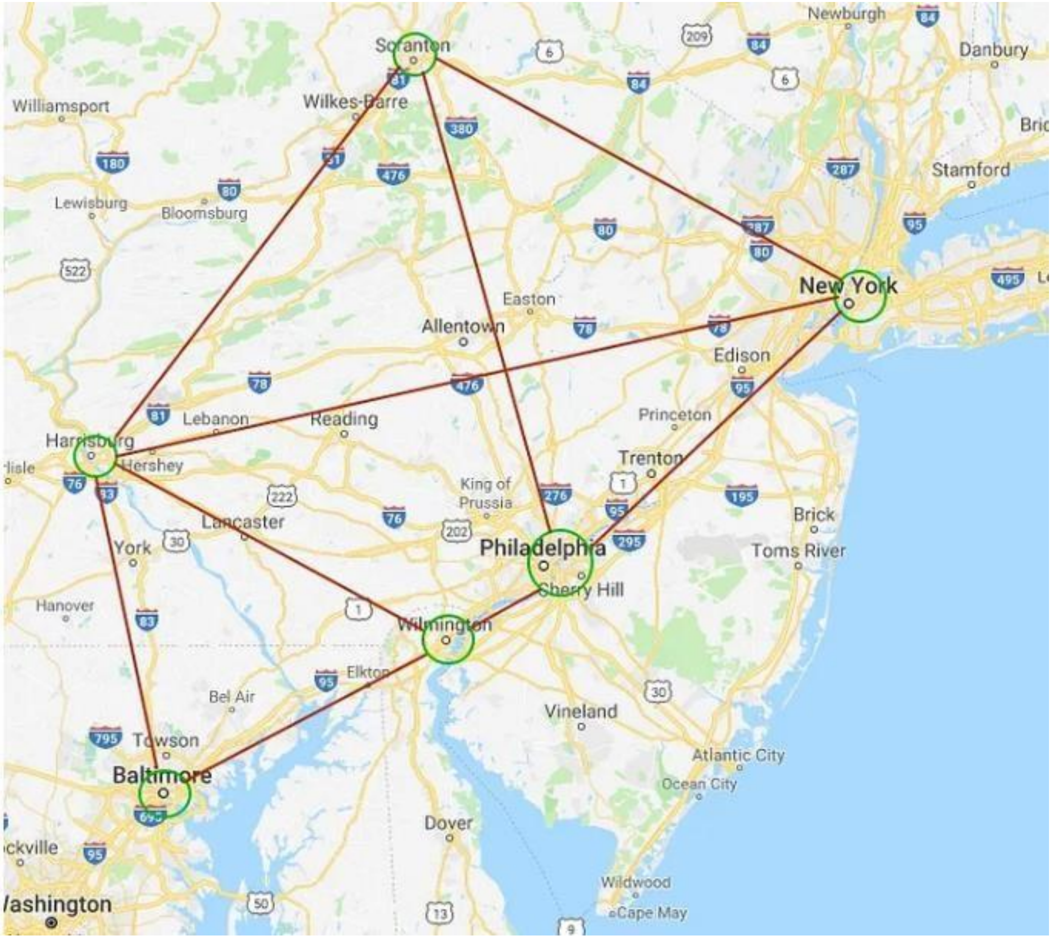
PRACTICAL NO:3

AIM: Download and install Spark. Create Graphical data and access the graphical data using Spark.

Theory:
Apache Spark is an open-source, distributed computing system that is designed for big data processing and analytics. It is a powerful framework for processing large volumes of data in parallel across a cluster of computers. Spark was developed to address some of the limitations of earlier big data processing frameworks like Hadoop MapReduce.

Apache Spark has gained widespread adoption in the big data and data analytics domain due to its speed, versatility, and ease of use. It is used by organizations for various data processing and analysis tasks, from batch processing large datasets to real-time data processing and machine learning applications.

Steps to perform the practical:
We will work on the below graph to understand the creation of graphical data and access of graphical data using spark.



To work with above graph, we need to launch Spark shell.

Step1: Start Cloudera , type in the terminal:

```
spark-shell
```

Step2: Now, we have to make some imports:

```
import org.apache.spark.graphx.Edge  #working with edge attribute

import org.apache.spark.graphx.Graph #analyzing and processing large graphs in a distributed fashion

import org.apache.spark.graphx.lib._     #Various analytics functions for graphs.
```

Step3: To create property graph we should firstly create an array of vertices and an array of edges. For vertices array, type in your spark shell:

```
val verArray = Array(
(1L, (“Philadelphia”, 1580863)),
(2L, (“Baltimore”, 620961)),
(3L, (“Harrisburg”, 49528)),
(4L, (“Wilmington”, 70851)),
(5L, (“New York”, 8175133)),
(6L, (“Scranton”, 76089)))
```

The attributes of the vertices mean the city name and population, respectively.

As an output you will see the following:

```
verArray: Array[(Long, (String, Int))] = Array((1,(Philadelphia,1580863)), (2,(Baltimore,620961)), (3,(Harrisburg,49528)), (4,(Wilmington,70851)), (5,(New York,8175133)), (6,(Scranton,76089)))
```

Step4: To create edges array, type in the spark shell:

```
val edgeArray = Array(
Edge(2L, 3L, 113),
Edge(2L, 4L, 106),
Edge(3L, 4L, 128),
Edge(3L, 5L, 248),
Edge(3L, 6L, 162),
Edge(4L, 1L, 39),
Edge(1L, 6L, 168),
Edge(1L, 5L, 130),
Edge(5L, 6L, 159))
```

The first and the second arguments indicate the source and the destination vertices identifiers and the third argument means the edge property which, in our case, is the distance between corresponding cities in kilometres.

The above-mentioned input will give us the following output:

```
edgeArray: Array[org.apache.spark.graphx.Edge[Int]] = Array(Edge(2,3,113),
Edge(2,4,106), Edge(3,4,128), Edge(3,5,248), Edge(3,6,162), Edge(4,1,39),
Edge(1,6,168), Edge(1,5,130), Edge(5,6,159))
```

Step5: Next, we will create RDDs from the vertices and edges arrays by using the sc.parallelize()command:

Note: Resilient Distributed Dataset (RDD) is the fundamental data structure of Spark. They are immutable Distributed collections of objects of any type.

```
val verRDD = sc.parallelize(verArray)
```

You will see:

```
verRDD: org.apache.spark.rdd.RDD[(Long, (String, Int))] = ParallelCollectionRDD[0] at
parallelize at <console>:34
```

```
val edgeRDD = sc.parallelize(edgeArray)
```

You will see:

```
edgeRDD:org.apache.spark.rdd.RDD[org.apache.spark.graphx.Edge[Int]] =
ParallelCollectionRDD[1] at parallelize at <console>:34
```

Step6: We are ready to build a property graph. The basic property graph constructor takes an RDD of vertices and an RDD of edges and builds a graph.

```
val graph = Graph(verRDD, edgeRDD)
```

You will see:

```
graph: org.apache.spark.graphx.Graph[(String, Int),Int] =
org.apache.spark.graphx.impl.GraphImpl@79ce6829
```

Step7: Let’s see some basic operations with graphs such as filtration by vertices, filtration by edges and operations with triplets.

A) Filtration by vertices

To illustrate the filtration by vertices let’s find the cities with population more than 50000.

To implement this, we will use the filter operator:

```
graph.vertices.filter {
case (id, (city, population)) => population > 50000
}.collect.foreach {
case (id, (city, population)) =>
println(s”The population of $city is $population”)
}
```

And this is the result we get:

```
The population of Scranton is 76089
The population of Wilmington is 70851
The population of Philadelphia is 1580863
The population of New York is 8175133
The population of Baltimore is 620961
```

B) Triplets

One of the core functionalities of GraphX is exposed through the triplets RDD. There is one triplet for each edge which contains information about both the vertices and the edge information. Let’s take a look through graph.triplets.collect.

As an example of working with triplets, we will find the distances between the connected cities:

```
for (triplet <- graph.triplets.collect) {
println(s”””The distance between ${triplet.srcAttr._1} and
${triplet.dstAttr._1} is ${triplet.attr} kilometers”””)
}
```

As a result, you should see:

```
The distance between Baltimore and Harrisburg is 113 kilometers
The distance between Baltimore and Wilmington is 106 kilometers
The distance between Harrisburg and Wilmington is 128 kilometers
The distance between Harrisburg and New York is 248 kilometers
The distance between Harrisburg and Scranton is 162 kilometers
The distance between Wilmington and Philadelphia is 39 kilometers
The distance between Philadelphia and New York is 130 kilometers
The distance between Philadelphia and Scranton is 168 kilometers
The distance between New York and Scranton is 159 kilometers
```

C) Filtration by edges

Now, let’s consider another type of filtration, namely filtration by edges. For this purpose, we want to find the cities, the distance between which is less than 150 kilometers. If we type in the spark shell,

```
graph.edges.filter {
case Edge(city1, city2, distance) => distance < 150
}.collect.foreach {
case Edge(city1, city2, distance) =>
println(s”The distance between $city1 and $city2 is $distance”)
}
```

The result will be:

```
The distance between 2 and 3 is 113
The distance between 2 and 4 is 106
The distance between 3 and 4 is 128
The distance between 4 and 1 is 39
The distance between 1 and 5 is 130
```

PRACTICAL NO:4

AIM: Write a Spark code to Handle the Streaming of data using RDD and Data frame.

Theory:
Streaming data refers to a continuous flow of data that is generated, processed, and transmitted in real-time or near-real-time. Unlike traditional batch processing, where data is collected and processed in predefined chunks or batches, streaming data is constantly produced and processed as it becomes available. This data can come from various sources, including sensors, social media feeds, log files, financial transactions, and more.

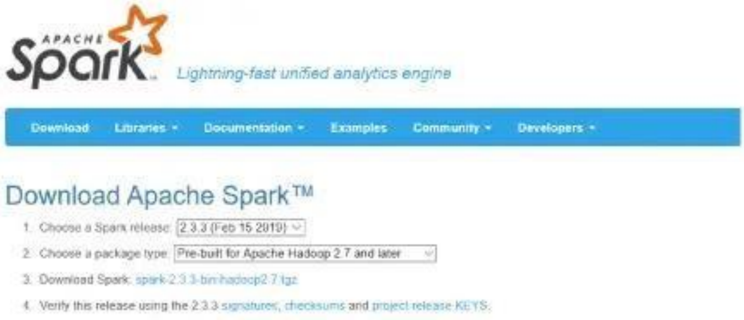
RDD stands for Resilient Distributed Dataset, and it is a fundamental data structure in Apache Spark, a popular open-source distributed computing framework for big data processing. RDDs are designed to provide fault-tolerant, parallelized data processing capabilities in a distributed computing environment.

A DataFrame is a data structure commonly used in data analysis and manipulation, particularly in the Python programming language with libraries like Pandas. It is a two-dimensional, tabular data structure that resembles a spreadsheet or a SQL table. In a DataFrame, data is organized into rows and columns, where each column can contain data of different types, such as numbers, strings, or dates.

Software Requirements:

1) Apache Hadoop 3.3 or higher
2) Python 3.11 or higher
3) Java Development Kit (JDK) 8 or higher

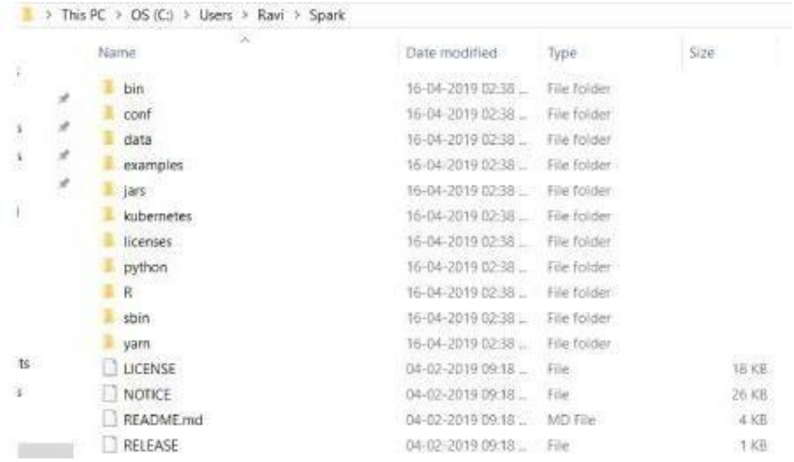
Installation Procedure:
Step 1: Go to Apache Spark's official download page and choose the latest release. For the package type, choose 'Pre-built for Apache Hadoop'.
The page will look like the one below:



Download Apache Spark™

1. Choose a Spark release: 2.3.3 (Feb 15 2019)
2. Choose a package type: Pre-built for Apache Hadoop 2.7 and later
3. Download Spark: spark-2.3.3-bin-hadoop2.7.tgz
4. Verify this release using the 2.3.3 signatures, checksums and project release KEYS.

Step 2: Once the download is completed, unzip the file, unzip the file using WinZip or WinRAR, or 7-ZIP.
Step 3: Create a folder called Spark under your user Directory like below and copy and paste the content from the unzipped file.
C:\Users\<USER>\Spark
It looks like the below after copy-pasting into the Spark directory.

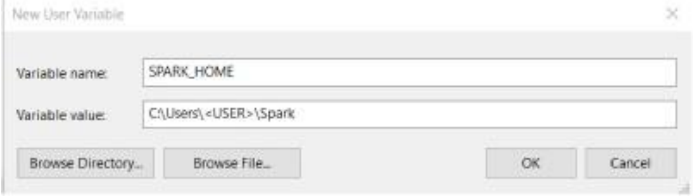


This PC > OS (C:) > Users > Ravi > Spark

Name	Date modified	Type	Size
bin	16-04-2019 02:38 ...	File folder	
conf	16-04-2019 02:38 ...	File folder	
data	16-04-2019 02:38 ...	File folder	
examples	16-04-2019 02:38 ...	File folder	
jars	16-04-2019 02:38 ...	File folder	
kubernetes	16-04-2019 02:38 ...	File folder	
licenses	16-04-2019 02:38 ...	File folder	
python	16-04-2019 02:38 ...	File folder	
R	16-04-2019 02:38 ...	File folder	
sbin	16-04-2019 02:38 ...	File folder	
yarn	16-04-2019 02:38 ...	File folder	
LICENSE	04-02-2019 09:18 ...	File	18 KB
NOTICE	04-02-2019 09:18 ...	File	26 KB
README.md	04-02-2019 09:18 ...	MD File	4 KB
RELEASE	04-02-2019 09:18 ...	File	1 KB

Step 4: Go to the conf folder and open the log file called log4j.properties. template. Change INFO to WARN (It can be an ERROR to reduce the log).

Step 5: Now, we need to configure the path.
Go to Control Panel -> System and Security -> System -> Advanced Settings -> Environment Variables
Add below new user variable (or System variable) (To add a new user variable, click on the New button under User variable for <USER>)

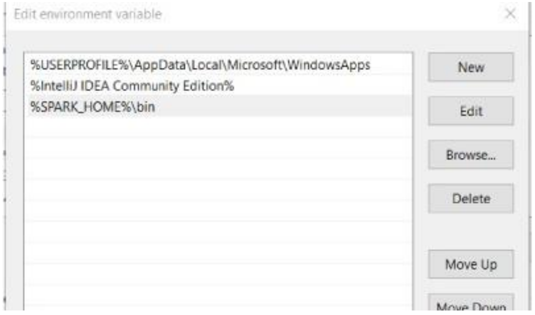


New User Variable

Variable name: SPARK_HOME

Variable value: C:\Users\<USER>\Spark

Click OK.
Add %SPARK_HOME%\bin to the path variable.



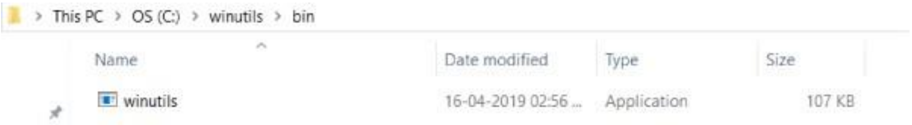
Edit environment variable

```
%USERPROFILE%\AppData\Local\Microsoft\WindowsApps
%IntelliJ IDEA Community Edition%
%SPARK_HOME%\bin
```

Click OK.

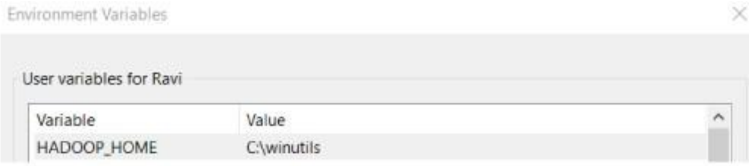
Step 6: Spark needs a piece of Hadoop to run. For Hadoop 3.3, you need to install winutils.exe.
You can find winutils.exe on this link
https://github.com/steveloughran/winutils/tree/master You can download it for your ease.

Step 7: Create a folder called winutils in C drive and create a folder called bin inside. Then, move the downloaded winutils file to the bin folder.
C:\winutils\bin

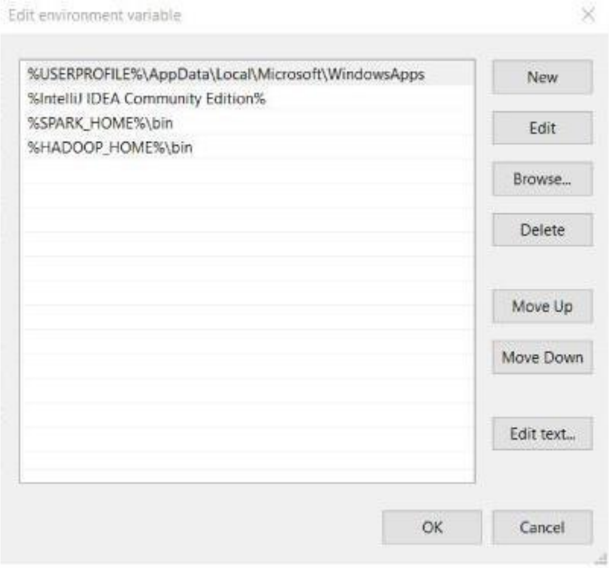


Name	Date modified	Type	Size
winutils	16-04-2019 02:56 ...	Application	107 KB

Add the user (or system) variable %HADOOP_HOME% like SPARK_HOME.



Variable	Value
HADOOP_HOME	C:\winutils

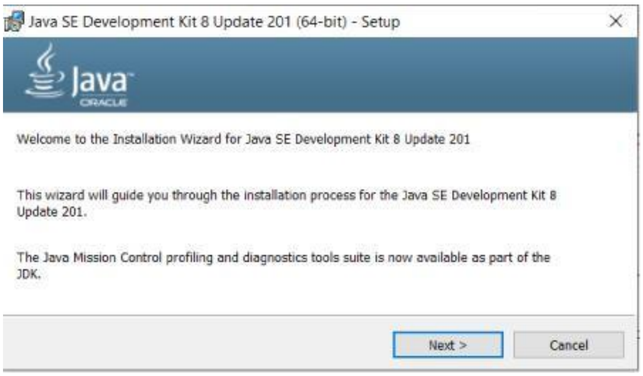


%USERPROFILE%\AppData\Local\Microsoft\WindowsApps
%IntelliJ IDEA Community Edition%
%SPARK_HOME%\bin
%HADOOP_HOME%\bin

Click OK.

Step 8: To install Apache Spark, Java should be installed on your computer. If you don't have java installed on your system. Please follow the below process

Java Installation Steps:
1. Go to the official Java site mentioned below the page.
Accept Licence Agreement for Java SE Development Kit 8u201
2. Download jdk-8u201-windows-x64.exe file
3. Double Click on the Downloaded .exe file, and you will see the window is shown below.



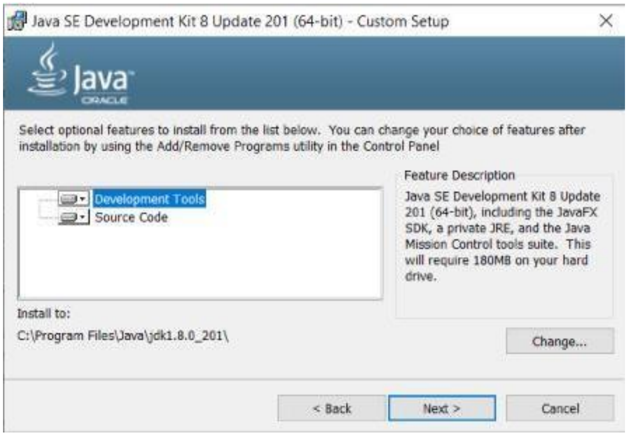
Java SE Development Kit 8 Update 201 (64-bit) - Setup

Welcome to the Installation Wizard for Java SE Development Kit 8 Update 201

This wizard will guide you through the installation process for the Java SE Development Kit 8 Update 201.

The Java Mission Control profiling and diagnostics tools suite is now available as part of the JDK.

4. Click Next.
5. Then below window will be displayed.



Java SE Development Kit 8 Update 201 (64-bit) - Custom Setup

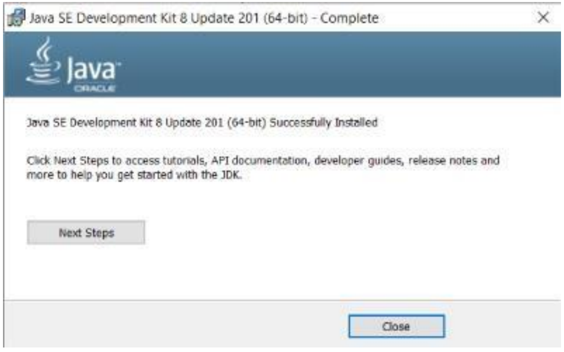
Select optional features to install from the list below. You can change your choice of features after installation by using the Add/Remove Programs utility in the Control Panel

Development Tools
Source Code

Feature Description
Java SE Development Kit 8 Update 201 (64-bit), including the JavaFX SDK, a private JRE, and the Java Mission Control tools suite. This will require 180MB on your hard drive.

Install to:
C:\Program Files\Java\jdk1.8.0_201\

6. Click Next.
7. Below window will be displayed after some process.



Java SE Development Kit 8 Update 201 (64-bit) - Complete

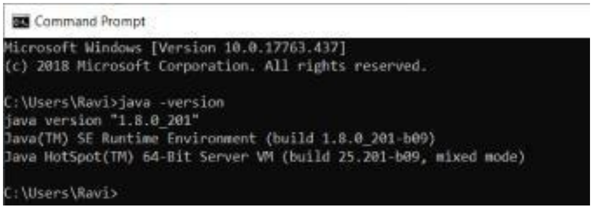
Java SE Development Kit 8 Update 201 (64-bit) Successfully Installed

Click Next Steps to access tutorials, API documentation, developer guides, release notes and more to help you get started with the JDK.

8. Click Close.

Test Java Installation

Open Command Line and type java -version, then it should display the installed version of Java

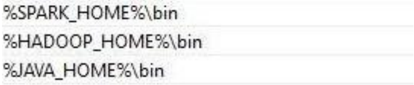


Command Prompt

```
Microsoft Windows [Version 10.0.17763.437]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\Ravi>java -version
java version "1.8.0_201"
Java(TM) SE Runtime Environment (build 1.8.0_201-b09)
Java HotSpot(TM) 64-Bit Server VM (build 25.201-b09, mixed mode)

C:\Users\Ravi>
```

You should also check JAVA_HOME and the path of %JAVA_HOME%\bin included in user variables (or system variables)

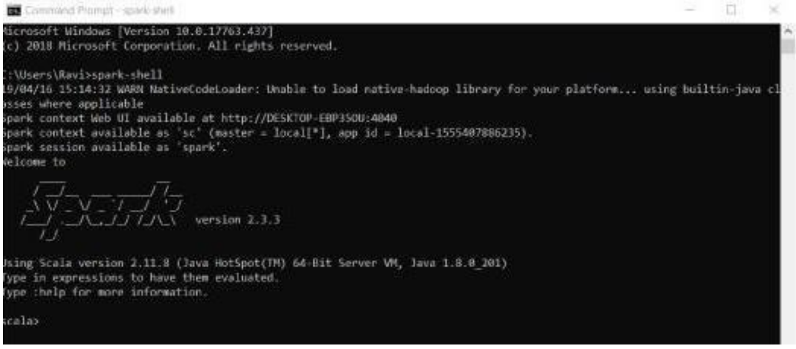
1. In the end, the environment variables have 3 new paths (if you need to add a Java path, otherwise SPARK_HOME and HADOOP_HOME).



```
%SPARK_HOME%\bin
%HADOOP_HOME%\bin
%JAVA_HOME%\bin
```

Test Installation

Open the command line and type spark-shell, and you will get the result below.



```
icrosoft Windows [Version 10.0.17763.437]
(c) 2018 Microsoft Corporation. All rights reserved.

:\Users\Ravi>spark-shell
19/04/16 15:14:32 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java cl
asses where applicable
park context Web UI available at http://DESKTOP-EBP3SOU:4040
park context available as 'sc' (master = local[*], app id = local-1555407886235).
park session available as 'spark'.
elcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 2.3.3
      /_/

sing Scala version 2.11.8 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_201)
ype in expressions to have them evaluated.
ype :help for more information.

cala>
```

We have completed the spark installation on the Windows system.

Creating and Accessing Graphical data in Spark:

Creating and accessing a bar graph in using pyspark:

To create a bar graph and access that bar graph in PySpark, you can use the Matplotlib library to create the bar graph and then use Spark to work with the data that you want to visualize in the bar graph. Here's a step-by-step guide:

1) Install Dependencies:

You'll need to install both PySpark and Matplotlib if you haven't already. You can use pip for installation:

```
pip install pyspark
pip install matplotlib
```

2) Create a SparkSession:

Initialize a SparkSession to work with PySpark:

```
from pyspark.sql import SparkSession
spark=SparkSession.builder.appName("BarGraphExample").getOrCreate()
```

3) Prepare Data:

Prepare your data in a format that can be used to create a bar graph. For example, you can read data from a CSV file into a DataFrame:

```
from pyspark.sql import Row
# Sample data
data = [Row(category="A", value=10),
        Row(category="B", value=20),
        Row(category="C", value=15),
        Row(category="D", value=30)]

# Create a DataFrame
df = spark.createDataFrame(data)
```

4) Create the Bar Graph:
Use Matplotlib to create the bar graph based on the data in the DataFrame:

```
matplotlib.pyplot as plt
# Extract data from the DataFrame
categories = df.select("category").rdd.flatMap(lambda x: x).collect()
values = df.select("value").rdd.flatMap(lambda x: x).collect()
# Create the bar graph
plt.bar(categories, values)
plt.xlabel("Categories")
plt.ylabel("Values")
plt.title("Bar Graph Example")
# Display the graph
plt.show()
```

This code will create a simple bar graph using Matplotlib and allow you to access and analyze your data using PySpark. You can customize the graph and data preparation based on your specific requirements and dataset.

5) Access and Analyze Data:
You can still use Spark to perform operations on your data even after creating the bar graph. For example, you can use Spark to aggregate, filter, or manipulate the data and then update the bar graph accordingly.

Let's say you have a DataFrame with sales data for different product categories, and you've already created a bar graph. Now, you want to update the bar graph to show the total sales for each category and add a title to the graph. Here's an example:

```
from pyspark.sql import SparkSession
import matplotlib.pyplot as plt

# Create a SparkSession
Spark=SparkSession.builder.appName("BarGraphExample").getOrCreate()
# Sample data
```

```
data = [("Electronics", 1000),
        ("Clothing", 1500),
        ("Books", 800)
       ("Toys", 1200)]

# Create a DataFrame
columns = ["Category", "Sales"]
df = spark.createDataFrame(data, columns)

# Create the initial bar graph
categories = df.select("Category").rdd.flatMap(lambda x: x).collect()
sales = df.select("Sales").rdd.flatMap(lambda x: x).collect()
plt.bar(categories, sales)
plt.xlabel("Categories")
plt.ylabel("Sales")
plt.title("Bar Graph Example (Initial)")

# Display the initial graph
plt.show()

# Continue data analysis
# Calculate total sales per category
total_sales = df.groupBy("Category").sum("Sales").withColumnRenamed("sum(Sales)",
"TotalSales")

# Update the bar graph with total sales
updated_categories = total_sales.select("Category").rdd.flatMap(lambda x: x).collect()
updated_sales = total_sales.select("TotalSales").rdd.flatMap(lambda x: x).collect()

# Create an updated bar graph with total sales
plt.bar(updated_categories, updated_sales)
plt.xlabel("Categories")
plt.ylabel("Total Sales")
plt.title("Bar Graph Example (Updated)")

# Display the updated graph
plt.show()
```

6) Close Spark Session:
Don't forget to stop the Spark session when you're done:

```
spark.stop()
```

In this example, we start by creating a bar graph that shows the initial sales data for each category. Afterward, we continue the data analysis using PySpark, calculate the total sales for each category, and update the bar graph to display the total sales. The two bar graphs are displayed sequentially, showing the initial and updated data visualizations.

This demonstrates how you can use PySpark to perform data analysis, calculate new values, and update your visualizations accordingly.

PRACTICAL NO: 5

AIM: Install Hive and use Hive to Create and store structured databases.

Theory:
"Hive" refers to Apache Hive, which is an open-source data warehouse infrastructure and query language that is used for managing and querying large datasets stored in distributed storage systems like Hadoop Distributed File System (HDFS). Hive was originally developed by Facebook and later open-sourced as part of the Apache Software Foundation.

Apache Hive provides a high-level, SQL-like interface for working with big data in Hadoop clusters, making it easier for data analysts and engineers to access and analyze large datasets without the need for complex programming in MapReduce or other lower-level technologies.

Steps to perform the practical:
Step1: Start cloudera
Step2: Open new terminal and create cat file as

```
[cloudera@quickstart ~]$ cat > /home/cloudera/employee.txt
1~Sachine~Pune~Product Engineering~100000~Big Data
2~Gaurav~Bengalore~Sales~90000~CRM
3~Manish~Chennai~Recruiter~125000~HR
4~Bhushan~Hyderabad~Developer~50000~BFSI
```

Step3: View the created file with cat command

```
[cloudera@quickstart ~]$ cat /home/cloudera/employee.txt
1~Sachine~Pune~Product Engineering~100000~Big Data
2~Gaurav~Bengalore~Sales~90000~CRM
3~Manish~Chennai~Recruiter~125000~HR
4~Bhushan~Hyderabad~Developer~50000~BFSI
```

Step4: List all the file directory under HDFS

```
[cloudera@quickstart ~]$ hdfs dfs -ls /
drwxrwxrwx   - hdfs  supergroup          0 2017-10-23 10:29 /benchmarks
drwxr-xr-x   - hbase supergroup          0 2023-08-18 02:00 /hbase
drwxr-xr-x   - solr  solr                0 2017-10-23 10:32 /solr
drwxrwxrwt   - hdfs  supergroup          0 2023-08-18 02:12 /tmp
drwxr-xr-x   - hdfs  supergroup          0 2017-10-23 10:31 /user
drwxr-xr-x   - hdfs  supergroup          0 2017-10-23 10:31 /var
```

Step5: Create HDFS file directory with name inputdirectory

```
[cloudera@quickstart ~]$ sudo -u hdfs hadoop fs -mkdir /inputdirectory
```

Step6: List the file directory to check if your directory has been created or not

```
[cloudera@quickstart ~]$ hdfs dfs -ls /
drwxrwxrwx   - hdfs  supergroup          0 2017-10-23 10:29 /benchmarks
drwxr-xr-x   - hbase supergroup          0 2023-08-18 02:00 /hbase
drwxr-xr-x   - hdfs  supergroup          0 2023-08-25 02:14 /inputdirectory
drwxr-xr-x   - solr  solr                0 2017-10-23 10:32 /solr
drwxrwxrwt   - hdfs  supergroup          0 2023-08-18 02:12 /tmp
drwxr-xr-x   - hdfs  supergroup          0 2017-10-23 10:31 /user
drwxr-xr-x   - hdfs  supergroup          0 2017-10-23 10:31 /var
[cloudera@quickstart ~]$
```

Step7: Give the permission of read, write and execution to the input directory

```
[cloudera@quickstart ~]$ sudo -u hdfs hadoop fs -chmod -R 777 /inputdirectory
```

Step8:Check if the permissions has been granted to your directory or not, you will see drwxrwxrwx for your directory if it has been granted

```
[cloudera@quickstart ~]$ hdfs dfs -ls /
drwxrwxrwx   - hdfs  supergroup          0 2017-10-23 10:29 /benchmarks
drwxr-xr-x   - hbase supergroup          0 2023-08-18 02:00 /hbase
drwxrwxrwx   - hdfs  supergroup          0 2023-08-25 02:14 /inputdirectory
drwxr-xr-x   - solr  solr                0 2017-10-23 10:32 /solr
drwxrwxrwt   - hdfs  supergroup          0 2023-08-18 02:12 /tmp
drwxr-xr-x   - hdfs  supergroup          0 2017-10-23 10:31 /user
drwxr-xr-x   - hdfs  supergroup          0 2017-10-23 10:31 /var
```

Step9: Now, shift the file which you have created earlier with cat command from cloudera to HDFS directory

```
[cloudera@quickstart ~]$ sudo -u hdfs hadoop fs -put /home/cloudera/employee.txt
/inputdirectrory
```

Step10: View the inputdirectory whether your file has been added or not

```
[cloudera@quickstart ~]$ hdfs dfs -ls /inputdirectrory
-rw-r--r--   1 hdfs supergroup        165 2023-08-25 02:21 /inputdirectrory/employee.txt
```

Step11: Read file in HDFS as,

```
[cloudera@quickstart ~]$ hadoop fs -cat /inputdirectrory/employee.txt
1~Sachine~Pune~Product Engineering~100000~Big Data
2~Gaurav~Bengalore~Sales~90000~CRM
3~Manish~Chennai~Recruiter~125000~HR
4~Bhushan~Hyderabad~Developer~50000~BFSI
```

Step12: Now enter into hive shell to create and store structured data

```
[cloudera@quickstart ~]$ hive
hive>
```

Step13: List all the databases present under hive

```
hive> show databases;
OK
default
```

Step14: Create database named organization

```
hive>create database organization;
OK
```

Step15: Check if database has been created or not

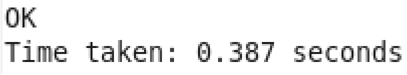
```
hive> show databases;
OK
default
organization
```

Step16: Change database from default to organization

```
hive> use organization;
OK
```

Step17: Create table named employee

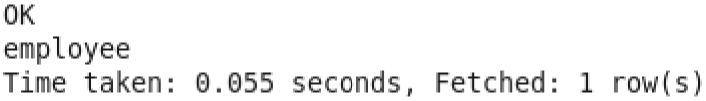
```
hive> create table employee(
   > id int,
   > name string,
   > city string,
   > department string,
   > salary int,
   > domain string)
   > row format delimited
   > fields terminated by '~';
```



```
OK
Time taken: 0.387 seconds
```

Step18: List all the tables to check whether your table has been created or not

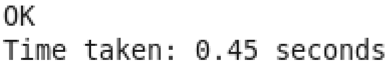
```
hive>show tables;
```



```
OK
employee
Time taken: 0.055 seconds, Fetched: 1 row(s)
```

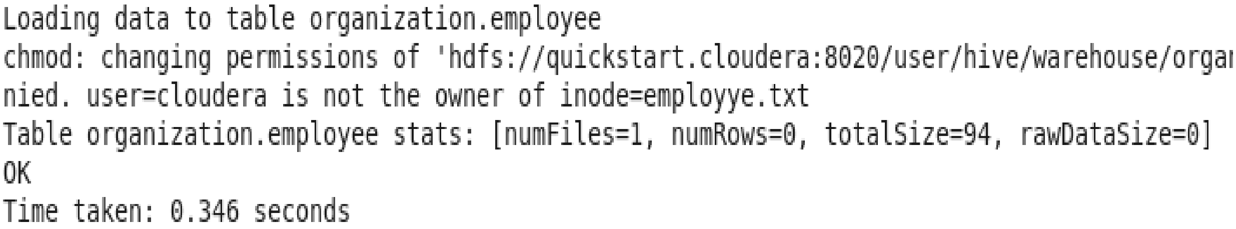
Step19: Read content from employee table, nothing will be printed because only columns has been created without data

```
hive> select * from employee;
```



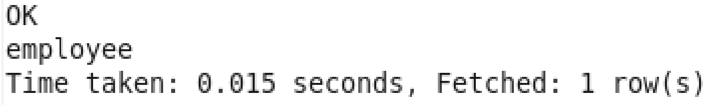
```
OK
Time taken: 0.45 seconds
```

Step20: Load the employee data from HDFS to employee table
hive>load data inpath '/inputdirectory/employee.txt' overwrite into table employee;



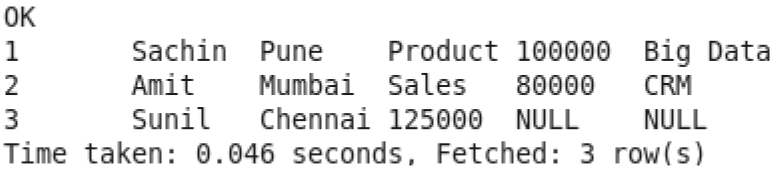
```
Loading data to table organization.employee
chmod: changing permissions of 'hdfs://quickstart.cloudera:8020/user/hive/warehouse/orga
nied. user=cloudera is not the owner of inode=employye.txt
Table organization.employee stats: [numFiles=1, numRows=0, totalSize=94, rawDataSize=0]
OK
Time taken: 0.346 seconds
```

Step21: Read the content of employee table again,this time you will get the data because we have shifted the data from HDFS to employee table
hive>show tables;



```
OK
employee
Time taken: 0.015 seconds, Fetched: 1 row(s)
```

hive>select * from employee;



```
OK
1       Sachin  Pune    Product 100000  Big Data
2       Amit    Mumbai  Sales   80000   CRM
3       Sunil   Chennai 125000  NULL    NULL
Time taken: 0.046 seconds, Fetched: 3 row(s)
```

PRACTICAL NO: 6

AIM: Install HBase and use the HBase Data model to store and retrieve data

Theory:
HBase is a distributed, scalable, and open-source NoSQL database system designed to handle large amounts of data in a distributed computing environment. It is part of the Apache Hadoop ecosystem and is often used for managing and storing vast amounts of semi-structured or unstructured data.

HBase is a valuable tool in the big data landscape, offering a distributed and scalable solution for managing and querying large volumes of data with low-latency requirements. It is particularly well-suited for use cases where traditional relational databases may struggle to handle the scale and complexity of the data.

Steps to perform the practical:
Step1: Start Cloudera
Step2: start the hbase shell as:

```
[cloudera@quickstart ~]$ hbase shell
hbase(main):001:0>
```

Step3: Check status,version,table_help, whoami

```
hbase(main):001:0> status
1 active master, 0 backup masters, 1 servers, 0 dead, 2.0000 average load

hbase(main):002:0> version
1.2.0-cdh5.13.0, rUnknown, Wed Oct  4 11:16:18 PDT 2017

hbase(main):003:0> table_help
Help for table-reference commands.

hbase(main):004:0> whoami
cloudera (auth:SIMPLE)
   groups: cloudera, default
```

Step4: Create table employee

```
hbase(main):005:0> create 'employee','Name','ID','Designation','Salary','Department'
0 row(s) in 1.3730 seconds
=> Hbase::Table - employee
```

Step5: Verify if table is created or not

```
hbase(main):006:0> list
TABLE
employee
```

```
=> ["employee"]
```

Step6:Disable the table employee

```
hbase(main):007:0> disable 'employee'
0 row(s) in 2.3730 seconds
```

Step7:Scan the table employee

```
hbase(main):008:0> scan 'employee'
ROW                COLUMN+CELL
ERROR: employee is disabled.
```

Step8: Check if table employee is disabled or not

```
hbase(main):010:0> is_disabled 'employee'
true
0 row(s) in 0.0250 seconds
```

Step9:Try to disable all the table starting with letter e

```
hbase(main):011:0> disable_all 'e.*'
employee

Disable the above 1 tables (y/n)?
y
1 tables successfully disabled
```

Step10: Enable the employee table again

```
hbase(main):012:0> enable 'employee'
0 row(s) in 1.2720 seconds
```

Step11: Check if employee table is enabled or not

```
hbase(main):013:0> is_enabled 'employee'
true
0 row(s) in 0.0220 seconds
```

Step12: Create new table student

```
hbase(main):014:0> create 'student','name','age','course'
0 row(s) in 1.2410 seconds
=> Hbase::Table - student
hbase(main):015:0>
```

Step13: Insert two students data into the student table

```
hbase(main):015:0> put 'student','sharath','name:fullname','sharath kumar'
0 row(s) in 0.0490 seconds

hbase(main):016:0> put 'student','sharath','age:presentage','24'
0 row(s) in 0.0060 seconds
```

```
hbase(main):017:0> put 'student','sharath','course:pursuing','Hadoop'
0 row(s) in 0.0030 seconds

hbase(main):018:0> put 'student','shashank','name:fullname','shashank kamble'
0 row(s) in 0.0050 seconds

hbase(main):019:0> put 'student','shashank','age:presentage','23'
0 row(s) in 0.0020 seconds

hbase(main):020:0> put 'student','shashank','course:pursuing','Java'
0 row(s) in 0.0030 seconds
```

Step14: Try to retrieve the students data from table student

```
hbase(main):022:0> get 'student','shashank'
COLUMN              CELL
 age:presentage       timestamp=1693559656275, value=23
 course:pursuing      timestamp=1693559672356, value=Java
 name:fullname        timestamp=1693559641013, value=shashank kamble

hbase(main):023:0> get 'student','sharath'
COLUMN              CELL
 age:presentage       timestamp=1693559577914, value=24
 course:pursuing      timestamp=1693559608940, value=Hadoop
 name:fullname        timestamp=1693559511147, value=sharath kumar

hbase(main):024:0> get 'student','sharath','course'
COLUMN              CELL
 course:pursuing      timestamp=1693559608940, value=Hadoop

hbase(main):025:0> get 'student','shashank','course'
COLUMN              CELL
 course:pursuing      timestamp=1693559672356, value=Java

hbase(main):026:0> get 'student','sharath','name'
COLUMN              CELL
 name:fullname        timestamp=1693559511147, value=sharath kumar

hbase(main):027:0> get 'student','sharath','age'
COLUMN              CELL
 age:presentage       timestamp=1693559577914, value=24
```

Step15: Scan and count the student table

```
hbase(main):028:0> scan 'student'
ROW                COLUMN+CELL
```

```
 sharath              column=age:presentage, timestamp=1693559577914, value=24
sharath              column=course:pursuing, timestamp=1693559608940,
value=Hadoop
 sharath              column=name:fullname, timestamp=1693559511147, value=sharath
kumar
 shashank              column=age:presentage, timestamp=1693559656275, value=23
shashank              column=course:pursuing, timestamp=1693559672356, value=Java
shashank              column=name:fullname, timestamp=1693559641013,
value=shashank kamble
hbase(main):029:0> count 'student'
2 row(s) in 0.0150 seconds
```

Step16:Try to alter the student table

```
hbase(main):042:0> alter 'student',NAME=>'name',VERSION=>5
```

Step17: Put the altered value as:

```
hbase(main):043:0> put 'student','shashank','name:fullname','shashank rao'
0 row(s) in 0.0030 seconds
```

Step18: Scan student and check if name is altered or not

```
hbase(main):044:0> scan 'student'
ROW                    COLUMN+CELL
 sharath                    column=age:presentage, timestamp=1693559577914, value=24
sharath                    column=course:pursuing, timestamp=1693559608940,
value=Hadoop
 sharath                    column=name:fullname, timestamp=1693559511147,
value=sharath kumar
 shashank                    column=age:presentage, timestamp=1693559656275,
value=23
 shashank                    column=course:pursuing, timestamp=1693559672356,
value=Java
 shashank                    column=name:fullname, timestamp=1693560273772,
value=shashank rao
```

Step19: Delete the column name of student shashank

```
hbase(main):045:0> delete 'student','shashank','name:fullname'
0 row(s) in 0.0250 seconds
```

Step20: Check if the name column has been deleted or not

```
hbase(main):046:0> get 'student','shashank'
COLUMN                    CELL
 age:presentage                timestamp=1693559656275, value=23
 course:pursuing                timestamp=1693559672356, value=Java
```

PRACTICAL NO:7

AIM: Perform importing and exporting of data between SQL and Hadoop using Sqoop

Theory:
Sqoop is a tool commonly used in the context of Big Data to facilitate the transfer of data between Apache Hadoop and relational databases. The name "Sqoop" is a combination of "SQL" (Structured Query Language) and "Hadoop." Sqoop is designed to simplify the process of importing data from a relational database (such as MySQL, Oracle, or SQL Server) into Hadoop and exporting data from Hadoop back to a relational database.

Steps to perform the practical:
Step 1: Open terminal and type the below common to connect to mysql

```
[cloudera@quickstart ~]$ mysql -D retail_db -u retail_dba -p
Enter password: cloudera
mysql>
```

Step2: List the table in mysql

```
mysql> show tables;
+---------------------+
| Tables_in_retail_db |
+---------------------+
| categories          |
| customers           |
| departments         |
| order_items         |
| orders              |
| products            |
+---------------------+
6 rows in set (0.00 sec)
```

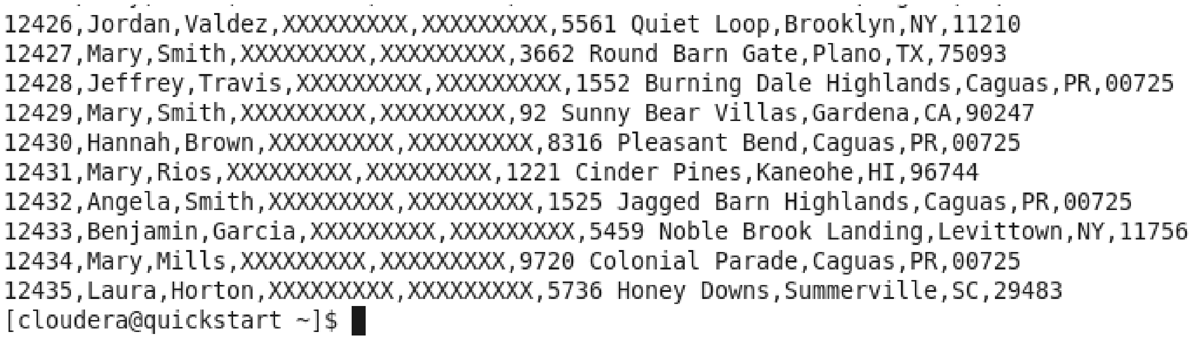
Step3: Select the data from customers table

```
mysql> select * from customers;

12435 rows in set (0.01 sec)
```

Step4: Open new terminal and type below command

```
[cloudera@quickstart ~]$ sqoop import --connect jdbc:mysql://localhost/retail_db --table customers --username retail_dba --password cloudera --target-dir /sqoop_import_data -m 1
```

Step5: View if data has been transferred from sql to hadoop or not
[cloudera@quickstart ~]$ hadoop fs -cat /sqoop_import_data/part-m-00000



```
12426,Jordan,Valdez,XXXXXXXXX,XXXXXXXXX,5561 Quiet Loop,Brooklyn,NY,11210
12427,Mary,Smith,XXXXXXXXX,XXXXXXXXX,3662 Round Barn Gate,Plano,TX,75093
12428,Jeffrey,Travis,XXXXXXXXX,XXXXXXXXX,1552 Burning Dale Highlands,Caguas,PR,00725
12429,Mary,Smith,XXXXXXXXX,XXXXXXXXX,92 Sunny Bear Villas,Gardena,CA,90247
12430,Hannah,Brown,XXXXXXXXX,XXXXXXXXX,8316 Pleasant Bend,Caguas,PR,00725
12431,Mary,Rios,XXXXXXXXX,XXXXXXXXX,1221 Cinder Pines,Kaneohe,HI,96744
12432,Angela,Smith,XXXXXXXXX,XXXXXXXXX,1525 Jagged Barn Highlands,Caguas,PR,00725
12433,Benjamin,Garcia,XXXXXXXXX,XXXXXXXXX,5459 Noble Brook Landing,Levittown,NY,11756
12434,Mary,Mills,XXXXXXXXX,XXXXXXXXX,9720 Colonial Parade,Caguas,PR,00725
12435,Laura,Horton,XXXXXXXXX,XXXXXXXXX,5736 Honey Downs,Summerville,SC,29483
[cloudera@quickstart ~]$ 
```

PRACTICAL NO:8

AIM: Write a Pig Script for solving counting problems.

Theory:
Apache Pig is a platform and high-level scripting language designed to simplify the process of writing complex data processing tasks for large datasets in a Hadoop ecosystem. It was developed by Yahoo! and later became an open-source project under the Apache Software Foundation.

Apache Pig provides a way to express data transformations and analysis in a more abstract, SQL-like language called Pig Latin. Pig Latin scripts are then compiled into a series of MapReduce jobs, which can be executed on a Hadoop cluster. This abstraction helps data analysts and engineers work with large datasets more easily without having to write low-level MapReduce code.

Steps to open the Cloudera:
Step1: Open Vmware workstation and drag the cloudera file in vmware
Step2: Click on Cloudera manager if it doesn't opens then do the step 3
Step3: Click on Launch Cloudera Express and copy the sudo code listed there.
Step4: Open the new terminal, paste the copied code followed with - - express and press enter. After it's executed, you will get the username and password, go to cloud manager again and it will now ask for username and password.
Step5: Enter username and password. Once it is done you are good to go for practical execution.

Steps to perform the practical:
Step1: Open the new terminal and create one file csv as input.csv

```
[cloudera@quickstart ~]$ cat > /home/cloudera/input.csv
hello...How are you?
are you okay?
I think need some break.
dont overthink
you can do it
You and I are best friends
```

Step2: After writing the code enter cntl+z to exit. Then view the data as:

```
[cloudera@quickstart ~]$ cat /home/cloudera/input.csv
hello...How are you?
are you okay?
I think need some break.
dont overthink
you can do it
```

Step3: To view the data through pig enter into the pig shell.

```
[cloudera@quickstart ~]$ pig -x local
```

Step4: After the above command your prompt will change into (grunt>)

Step5: Enter the below code to get the word count of your file

```
grunt> lines = load '/home/cloudera/input.csv' as (line:chararray);
grunt>words = foreach lines GENERATE FLATTEN(TOKENIZE(line)) as woed;
grunt> grouped = GROUP words by woed;
grunt> wordcount = foreach grouped GENERATE group, COUNT(words);
grunt> dump wordcount;
```

Step6: Once you enter after writing the above code you will see the output as:

```
(I,1)
(do,1)
(it,1)
(are,2)
(can,1)
(you,2)
(dont,1)
(need,1)
(some,1)
(you?,1)
(okay?,1)
(think,1)
(break.,1)
(overthink,1)
(hello...How,1)
```