

Practical 1 : 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

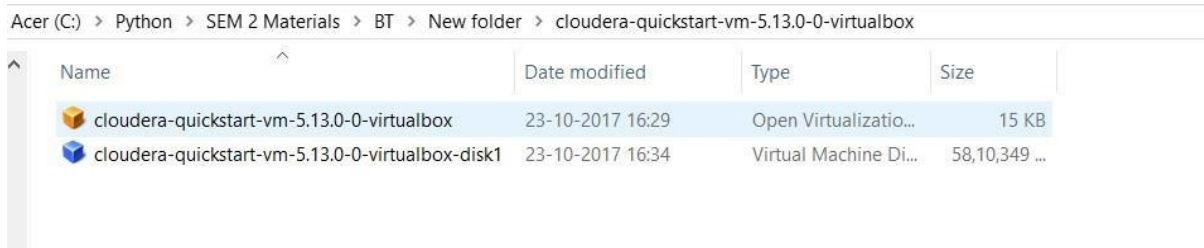
A virtual machine such as Oracle Virtual Box or VMWare

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

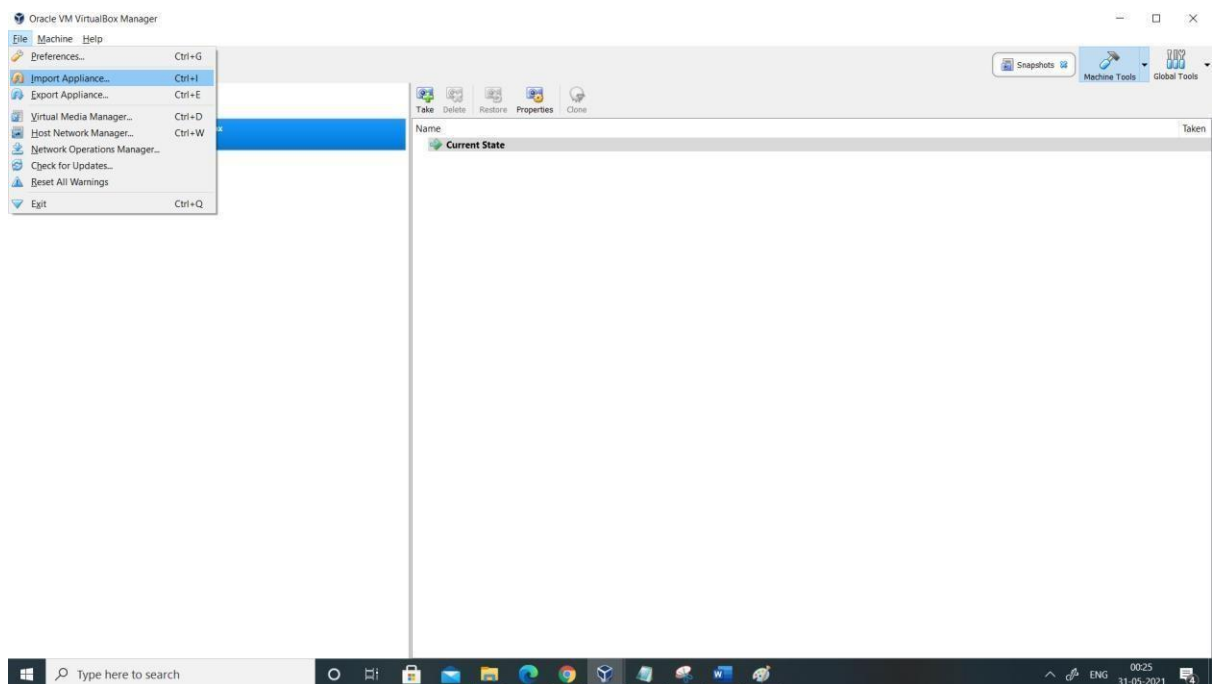
- ✓ The Cloudera QuickStart VMs are openly available as Zip archives in VirtualBox, VMware and KVM formats. To download the VM, search for <https://www.cloudera.com/downloads.html>, and select the appropriate version of CDH that you require.
 - ✓ Click on the 'GET IT NOW' button, and it will prompt you 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.
-



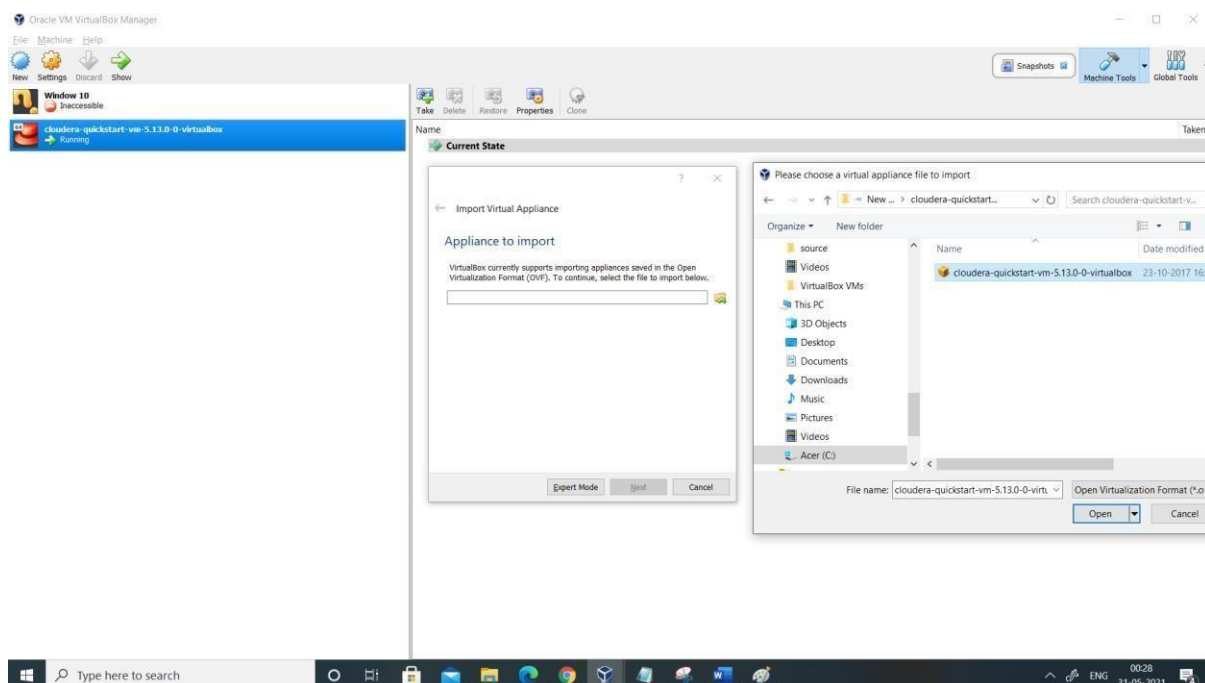
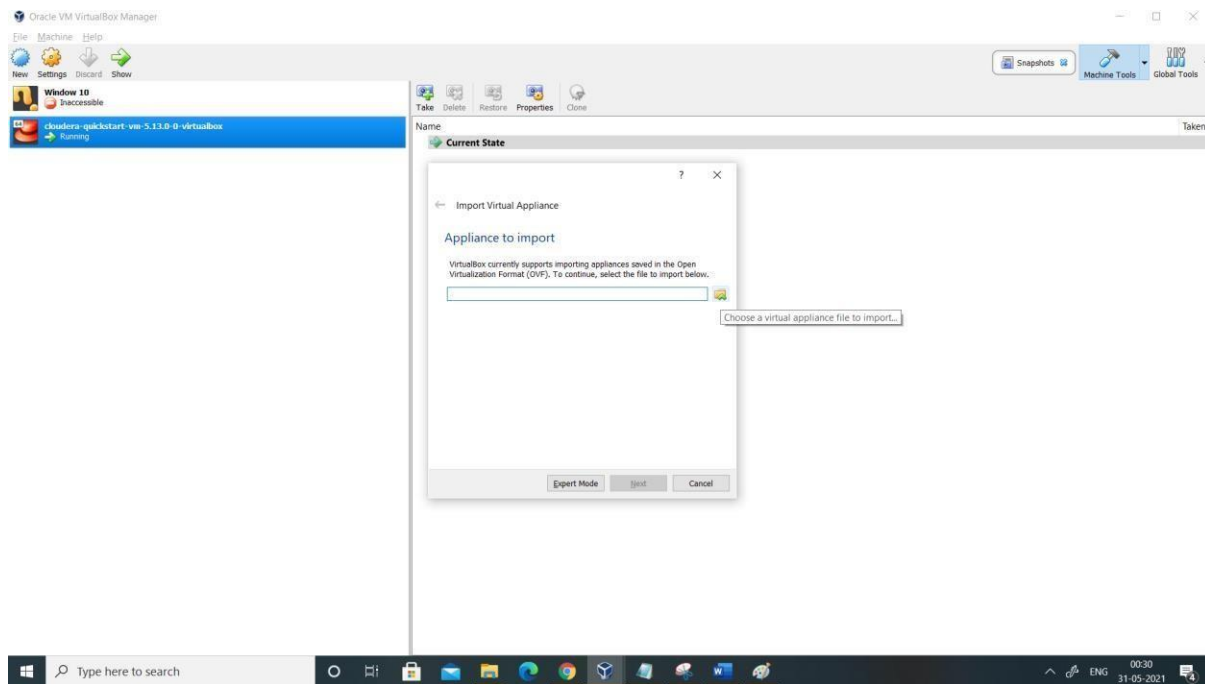
✓ Now that the downloading process is done with, let's move forward with this Cloudera QuickStart VM Installation guide and see the actual process.

Cloudera QuickStart VM Installation

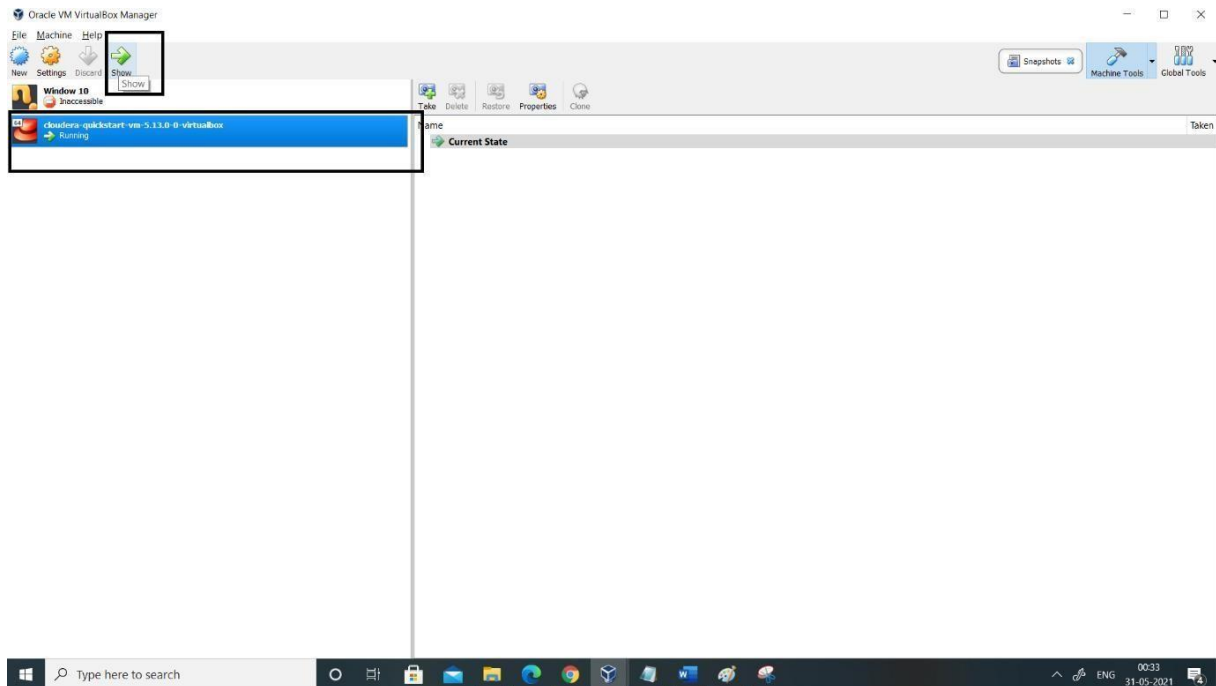
- ✓ Before setting up the Cloudera Virtual Machine, you would need to have a virtual machine such as VMware or Oracle VirtualBox on your system.
- ✓ In this case, we are using Oracle VirtualBox to set up the Cloudera QuickStart VM.
- ✓ In order to download and install the Oracle VirtualBox on your operating 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'.



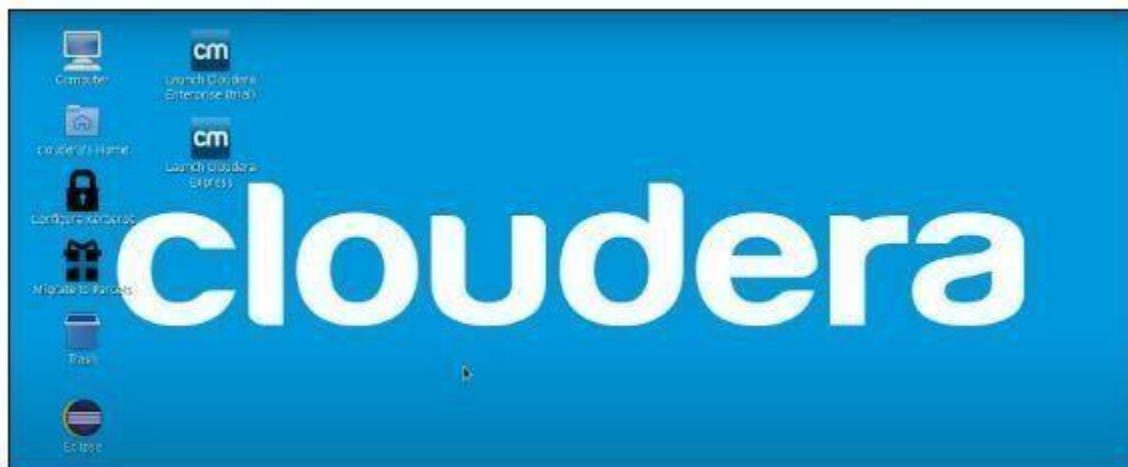
- ✓ Choose the QuickStart VM image by looking into your downloads. Click on 'Open' and then 'Next'. Now you can see the specifications, then click on 'Import'. This will start importing the virtual disk image .vmdk file into your VM box.



- ✓ 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:

```
Terminal
File Edit View Search Terminal Help

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...

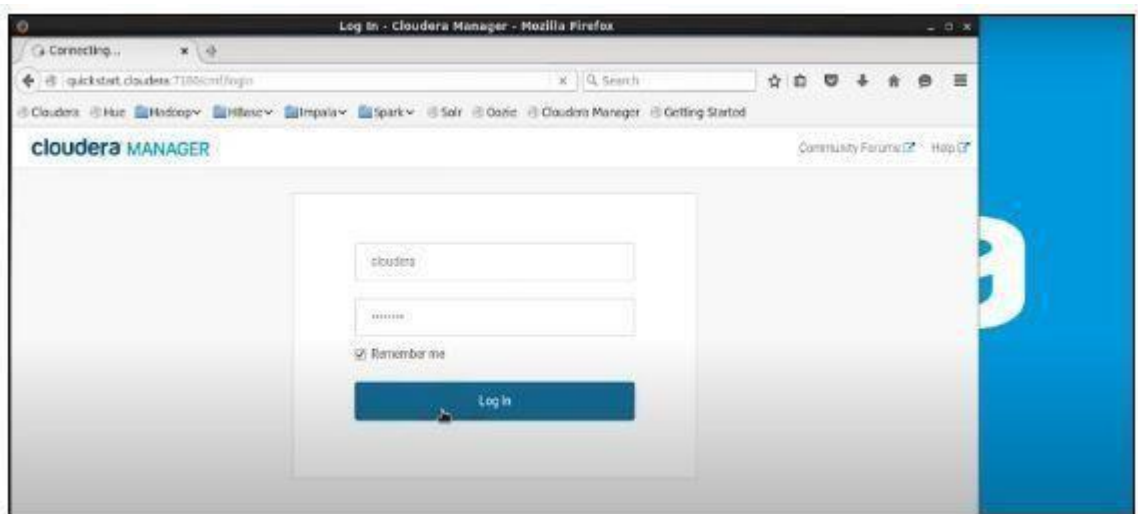
Success! You can now log into Cloudera Manager from the QuickStart VM's browser:

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.



- ✓ 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 2 - Implementing Map-Reduce Program for Word Count problem.

SERVICES TO START-

HDFS

HBASE

If name node error occurs , restart HDFS

Steps to Perform the Practical:

Step 1: Open Cloudera

- Start your Cloudera environment and log in.

Step 2: Go to Eclipse

- Launch Eclipse from the Cloudera environment.

Step 3: Create a new Java Project

- Go to `File > New > Java Project`.

Step 4: Name the Project and Configure Libraries

- Give the project a name, for example, "WordCount."
- Click "Next" and navigate to the "Libraries" tab.
- Click "Add External JARs."

Step 5: Add Hadoop JARs

- Navigate to the Hadoop library directory, typically located at `/usr/lib/hadoop`.
- Select all the JAR files and click "OK."

Step 6: Add Hadoop Client JARs

- Click "Add External JARs" again.
- Navigate to `/usr/lib/hadoop/client` and select all the JAR files.
- Click "OK."

Step 7: Add Hadoop Client 0.20 JARs

- Once more, click "Add External JARs."
- Navigate to `/usr/lib/hadoop/client0.20` and select all the JAR files.
- Click "OK."

Step 8: Finish Project Configuration

- Click "Finish" to complete the project setup.

Step 9: Create a New Java Class

- Under the newly created Java project, right-click on `src > New > Class`.
- Give the class the same name as the project, e.g., "WordCount," and click "Finish."

Step 10: Write Word Count Code

- Write the Java code for WordCount in the newly created class.

```
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, "word count");
        job.setJarByClass(WordCount.class);
        job.setMapperClass(TokenizerMapper.class);
        job.setCombinerClass(IntSumReducer.class);
        job.setReducerClass(IntSumReducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}

```

- Save the file using `Ctrl + S`.

Step 11: Navigate to the project name, right-click on it, then choose "Export" > "Java" > "JAR file." Click "Next," set the path for the JAR file (e.g., cloudera), and provide a name matching the project. Finally, click "Finish" to complete the export process.

OPEN CLOUDERA TERMINAL

Step 1: Check existing files in HDFS

```

-----
[cloudera@quickstart ~]$ hdfs dfs -ls
Found 2 items
drwx----- - cloudera cloudera      0 2024-01-10 03:09 .staging
-rw-r--r--  1 cloudera cloudera     81 2024-01-10 02:59 inputdirectory
-----

```

Step 2: Create a directory /mapreduce in HDFS

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

Step 3: Check the contents of the root directory in HDFS

```
[cloudera@quickstart ~]$ hdfs dfs -ls /  
Found 8 items  
drwxrwxrwx - hdfs supergroup 0 2017-10-23 10:29 /benchmarks  
drwxr-xr-x - hbase supergroup 0 2024-02-16 03:36 /hbase  
drwxr-xr-x - hdfs supergroup 0 2024-01-06 06:52 /inputdirectory  
drwxr-xr-x - hdfs supergroup 0 2024-02-18 03:08 /mapreduce  
drwxr-xr-x - solr solr 0 2017-10-23 10:32 /solr  
drwxrwxrwt - hdfs supergroup 0 2024-01-06 06:56 /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
```

Step 4: Change permissions of the /mapreduce directory to 777

```
[cloudera@quickstart ~]$ sudo -u hdfs hadoop fs -chmod -R 777 /mapreduce  
[cloudera@quickstart ~]$ hdfs dfs -ls / mapreduce  
Found 8 items  
drwxrwxrwx - hdfs supergroup 0 2017-10-23 10:29 /benchmarks  
drwxrwxrwx - hbase supergroup 0 2024-02-16 03:36 /hbase  
drwxrwxrwx - hdfs supergroup 0 2024-01-06 06:52 /inputdirectory  
drwxrwxrwx - hdfs supergroup 0 2024-02-18 03:08 /mapreduce  
drwxrwxrwx - solr solr 0 2017-10-23 10:32 /solr  
drwxrwxrwt - hdfs supergroup 0 2024-01-06 06:56 /tmp  
drwxrwxrwx - hdfs supergroup 0 2017-10-23 10:31 /user  
drwxrwxrwx - hdfs supergroup 0 2017-10-23 10:31 /var
```

Step 5: Create a local file /home/cloudera/input.txt

```
[cloudera@quickstart ~]$ cat > /home/cloudera/input.txt  
myself OM Panchal  
om panchal  
^Z  
[1]+ Stopped cat > /home/cloudera/input.txt
```

Step 6: Verify the content of the local file

```
[cloudera@quickstart ~]$ cat > /home/cloudera/input.txt  
myself OM Panchal  
om panchal  
^Z  
[1]+  Stopped                  cat > /home/cloudera/input.txt
```

Step 7: Upload the local file to HDFS under the /mapreduce directory

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

Step 8: Verify the uploaded file in HDFS

```
[cloudera@quickstart ~]$ hdfs dfs -ls /mapreduce  
Found 1 items  
-rw-r--r--  1 hdfs supergroup      29 2024-02-18 03:15 /mapreduce/input.txt
```

Step 9: Run the WordCount job

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

```

24/02/18 03:19:20 INFO mapreduce.Job: map 100% reduce 0%
24/02/18 03:19:38 INFO mapreduce.Job: map 100% reduce 100%
24/02/18 03:19:39 INFO mapreduce.Job: Job job_1708253660300_0001 completed successfully
24/02/18 03:19:40 INFO mapreduce.Job: Counters: 49
  File System Counters
    FILE: Number of bytes read=77
    FILE: Number of bytes written=294519
    FILE: Number of read operations=0
    FILE: Number of large read operations=0
    FILE: Number of write operations=0
    HDFS: Number of bytes read=145
    HDFS: Number of bytes written=39
    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)=12290048
    Total time spent by all reduces in occupied slots (ms)=7835136
    Total time spent by all map tasks (ms)=24004
    Total time spent by all reduce tasks (ms)=15303
    Total vcore-milliseconds taken by all map tasks=24004
    Total vcore-milliseconds taken by all reduce tasks=15303
    Total megabyte-milliseconds taken by all map tasks=12290048
    Total megabyte-milliseconds taken by all reduce tasks=7835136
  Map-Reduce Framework
    Map input records=2
    Map output records=5
    Map output bytes=49
    Map output materialized bytes=73
    Input split bytes=116
    Combine input records=5
    Combine output records=5
    Reduce input groups=5
    Reduce shuffle bytes=73
    Reduce input records=5
    Reduce output records=5
    Spilled Records=10
    Shuffled Maps =1
    Failed Shuffles=0
    Merged Map outputs=1
    GC time elapsed (ms)=514
    CPU time spent (ms)=1120
    Physical memory (bytes) snapshot=224993280
    Virtual memory (bytes) snapshot=1410572288
    Total committed heap usage (bytes)=101449728

```

Step 10: Check the output in HDFS

```

[cloudera@quickstart ~]$ hdfs dfs -ls /mapreduce/output
Found 2 items
-rw-r--r--  1 cloudera supergroup          0 2024-02-18 03:19 /mapreduce/output/_SUCCESS
-rw-r--r--  1 cloudera supergroup        39 2024-02-18 03:19 /mapreduce/output/part-r-00000

```

Step 11: View the contents of the output file

```

[cloudera@quickstart ~]$ hdfs dfs -cat /mapreduce/output/part-r-00000
OM      1
Panchal 1
myself  1
om      1
panchal 1

```

PRACTICAL 3 - WRITE A SPARK CODE TO HANDLE THE STREAMING OF DATA USING RDD AND DATA FRAME.

ENTER IN THE SPARK SHELL USING SPARK-SHELL , START THE HADOOP SERVICES (HDFS AND YARN , SPARK) BEFORE LAUNCHING THE SPARK SHELL.

STEP 1: IMPORT THE NECESSARY LIBRARIES

```
scala> import org.apache.spark.graphx.Edge
import org.apache.spark.graphx.Edge

scala> import org.apache.spark.graphx.Graph
import org.apache.spark.graphx.Graph

scala> import org.apache.spark.graphx.lib._
import org.apache.spark.graphx.lib._
```

STEP 2: DEFINE THE VERTEX AND EDGE DATA

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

scala> 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)
  | )
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))
```

STEP 3: CREATE THE SPARK RDDS AND THE GRAPH

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

scala> val edgeRDD = sc.parallelize(edgeArray)
edgeRDD: org.apache.spark.rdd.RDD[org.apache.spark.graphx.Edge[Int]] = ParallelCollectionRDD[1] at parallelize at <console>:34

scala> val graph = Graph(verRDD, edgeRDD)
graph: org.apache.spark.graphx.Graph[(String, Int),Int] = org.apache.spark.graphx.impl.GraphImpl@5a4fc46f
```

STEP 4: FILTER VERTICES BY POPULATION

```
scala> graph.vertices.filter {
  |   case (id, (city, population)) => population > 50000
  | }.collect.foreach {
  |   case (id, (city, population)) =>
  |     println(s"The population of $city is $population")
  | }
[Stage 0:>          (0 + 0) / 2][Stage 1:>          (0 + 0) / 2]24/04/12 22:45:50 WARN cluster.YarnScheduler: Initial job has not accepted an
y resources; check your cluster UI to ensure that workers are registered and have sufficient resources
The population of Wilmington is 70851
The population of Scranton is 76089
The population of Baltimore is 620961
The population of Philadelphia is 1580863
The population of New York is 8175133
```

STEP 5: TRAVERSE THE GRAPH EDGES

```
scala> for (triplet <- graph.triplets.collect) {
  |   println(s"The distance between ${triplet.srcAttr._1} and ${triplet.dstAttr._1} is ${triplet.attr} kilometers")
  | }
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 Philadelphia and New York is 130 kilometers
The distance between Philadelphia and Scranton is 168 kilometers
The distance between Harrisburg and Scranton is 162 kilometers
The distance between Wilmington and Philadelphia is 39 kilometers
The distance between New York and Scranton is 159 kilometers
```

STEP 6: FILTER EDGES BY DISTANCE

```
THE DISTANCE BETWEEN NEW YORK AND SCRANTON IS 159 KILOMETERS
```

```
scala> 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 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 1 and 5 is 130
The distance between 4 and 1 is 39
```

Practical 4 - Write a Spark code to Handle the Streaming of data using RDD and Data frame.

START SERVICES – SPARK IN CLUDERA MANAGER

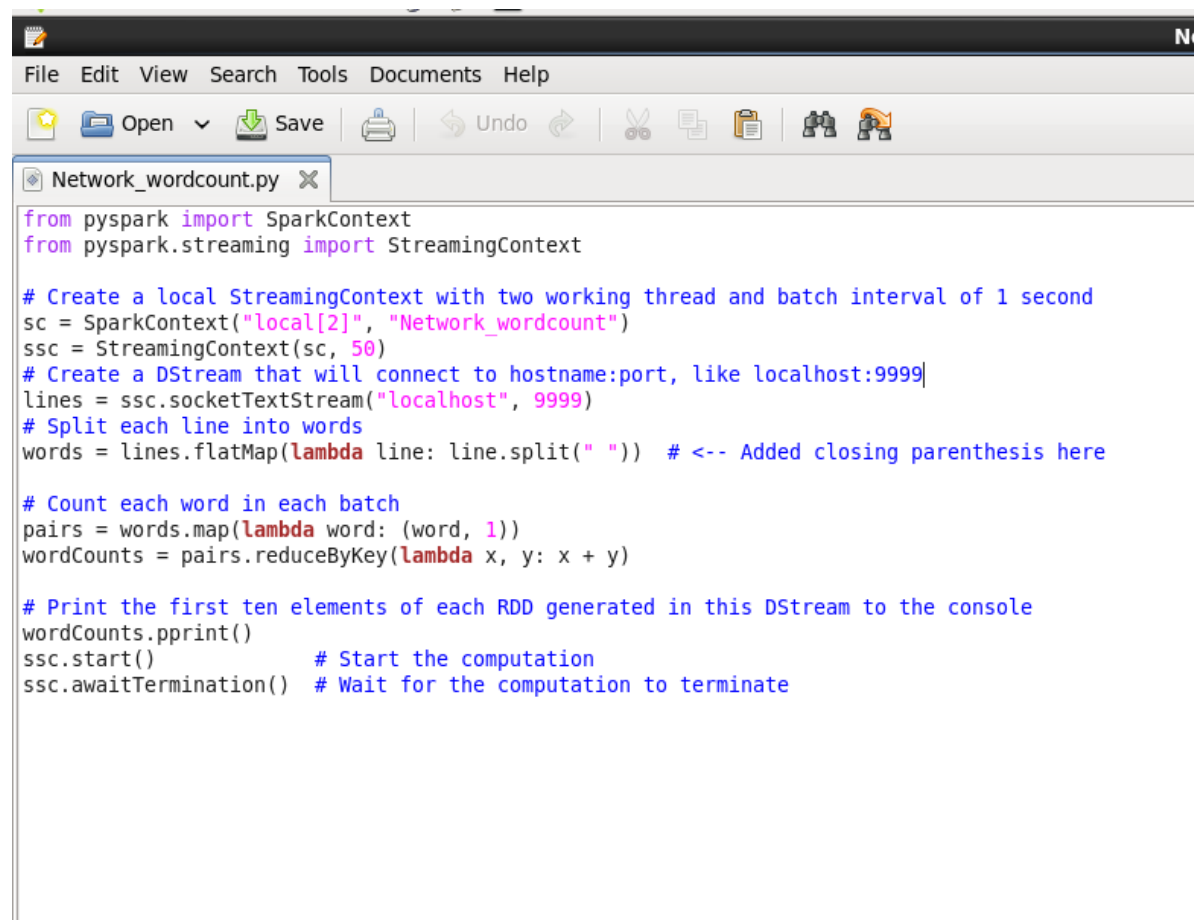
<https://spark.apache.org/docs/latest/streaming-programming-guide.html>

STEP 1 – Write Your Python Code:

Inside the text editor, write your Python code (Network_wordcount.py).

PS – save the file in Home/Cloudera/Documents

Below is the code



```
from pyspark import SparkContext
from pyspark.streaming import StreamingContext

# Create a local StreamingContext with two working thread and batch interval of 1 second
sc = SparkContext("local[2]", "Network_wordcount")
ssc = StreamingContext(sc, 50)
# Create a DStream that will connect to hostname:port, like localhost:9999
lines = ssc.socketTextStream("localhost", 9999)
# Split each line into words
words = lines.flatMap(lambda line: line.split(" ")) # <-- Added closing parenthesis here

# Count each word in each batch
pairs = words.map(lambda word: (word, 1))
wordCounts = pairs.reduceByKey(lambda x, y: x + y)

# Print the first ten elements of each RDD generated in this DStream to the console
wordCounts.pprint()
ssc.start() # Start the computation
ssc.awaitTermination() # Wait for the computation to terminate
```

Step 2: Save and Exit

Save the file (in python extension) and exit the text editor.

Step 3: Open a New Terminal Window

Open a new terminal window or tab to perform the following steps simultaneously.

Step 4: Start Netcat

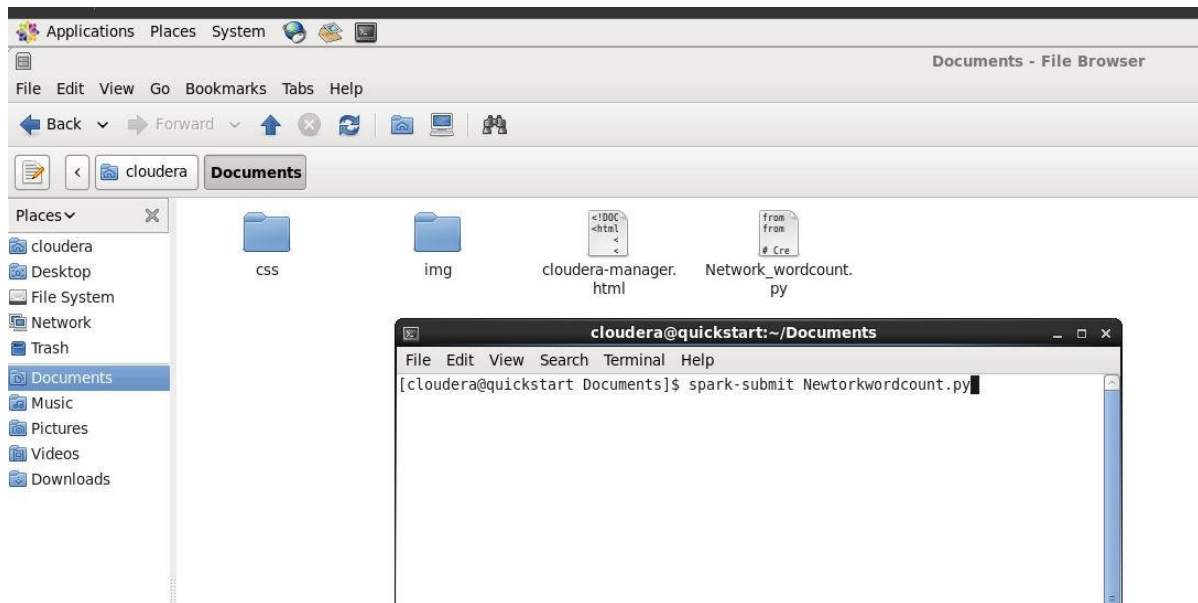
In the new terminal, start Netcat in listening mode with the specified port (9999).



```
cloudera@quickstart:~/Documents
File Edit View Search Terminal Help
[cloudera@quickstart Documents]$ nc -lk 9999
Om PAnchal
OM
ANYaa
eren
eren
abcd
efghf
kwakwf
^Z
[1]+  Stopped                  nc -lk 9999
[cloudera@quickstart Documents]$
```


Step 5: Submit the Spark Job

Switch to terminal and submit the Spark job using spark-submit.



RUN BOTH TERMINALS

Step 6: Verify Results

Check the terminal where Spark Streaming is running. You should see word counts printed as batches are processed.

Step 7: Stop Spark Streaming

Terminate the Spark Streaming application when you are done.(CTRL +Z)

OUTPUT –

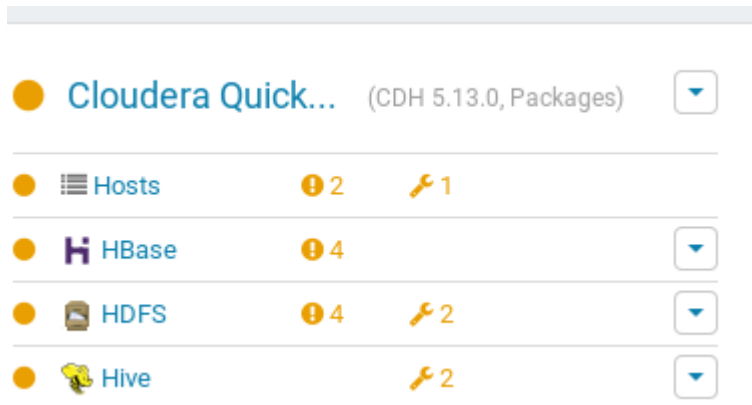
```
24/02/21 22:15:04 INFO python.PythonRunner: Times: total = 33, boot = -356, init = 389, finish = 0
24/02/21 22:15:04 INFO python.PythonRunner: Times: total = 52, boot = -228, init = 280, finish = 0
24/02/21 22:15:04 INFO executor.Executor: Finished task 0.0 in stage 4.0 (TID 4). 1257 bytes result sent to driver
24/02/21 22:15:05 INFO scheduler.DAGScheduler: ResultStage 4 (runJob at PythonRDD.scala:393) finished in 0.165 s
24/02/21 22:15:05 INFO scheduler.DAGScheduler: Job 2 finished: runJob at PythonRDD.scala:393, took 0.244368 s
24/02/21 22:15:05 INFO scheduler.TaskSetManager: Finished task 0.0 in stage 4.0 (TID 4) in 166 ms on localhost (executor driver) (1/1)
24/02/21 22:15:05 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 4.0, whose tasks have all completed, from pool
-----
Time: 2024-02-21 22:15:00
-----
(u'abcd', 1)
(u'', 1)
(u'Om', 1)
(u'PAnchal', 1)
(u'OM', 1)
(u'eren', 2)
(u'kwakwf', 1)
(u'ANYaa', 1)
(u'efghf', 1)

24/02/21 22:15:05 INFO scheduler.JobScheduler: Finished job streaming job 1708582500000 ms.0 from job set of time 1708582500000 ms
24/02/21 22:15:05 INFO scheduler.JobScheduler: Total delay: 5.032 s for time 1708582500000 ms (execution: 4.157 s)
24/02/21 22:15:05 INFO scheduler.ReceivedBlockTracker: Deleting batches ArrayBuffer()
24/02/21 22:15:05 INFO scheduler.InputInfoTracker: remove old batch metadata:
24/02/21 22:15:50 INFO scheduler.JobScheduler: Added jobs for time 1708582550000 ms
24/02/21 22:15:50 INFO scheduler.JobScheduler: Starting job streaming job 1708582550000 ms.0 from job set of time 1708582550000 ms
-----
```


Practical 5 - Install Hive and use Hive Create and store structured databases.

Step 1: Start Cloudera

Ensure that your Cloudera services HIVE , HDFS is up and running.



Step 2: Create Employee File

Create a file named employee.txt with the following content:

```
[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^Z
_ _ _ _ _
```

Step 3: View the Create file

```
[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
```

Step 4: List All Files in HDFS

```
[cloudera@quickstart ~]$ hdfs dfs -ls /  
Found 9 items  
drwxrwxrwx - hdfs supergroup 0 2017-10-23 10:29 /benchmarks  
drwxrwxrwx - hbase supergroup 0 2024-02-21 22:53 /hbase  
drwxrwxrwx - hdfs supergroup 0 2024-02-22 00:50 /inputdirectory  
drwxrwxrwx - hdfs supergroup 0 2024-02-18 03:18 /mapreduce  
drwxrwxrwx - solr solr 0 2017-10-23 10:32 /solr  
drwxr-xr-x - cloudera supergroup 0 2024-02-21 22:41 /sqoop_import_data  
drwxrwxrwt - hdfs supergroup 0 2024-01-06 06:56 /tmp  
drwxrwxrwx - hdfs supergroup 0 2017-10-23 10:31 /user  
drwxrwxrwx - hdfs supergroup 0 2017-10-23 10:31 /var
```

Step 5: Create HDFS Directory(in my case already created)

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

Step 6: List HDFS Directory(input directory should be present)

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

Step 7: Set Permissions for the Directory

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

Step 8: Verify Permissions(ignore)

HDFS DFS -LS /

Step 9: Move the File to HDFS

```
[cloudera@quickstart ~]$ sudo -u hdfs hadoop fs -put /home/cloudera/employee.txt  
/inputdirectory  
put: `/inputdirectory/employee.txt': File exists
```

Step 10: Check File in HDFS

```
[cloudera@quickstart ~]$ hdfs dfs -ls /inputdirectory
Found 1 items
-rwxrwxrwx  1 hdfs supergroup      123 2024-02-22 00:50 /inputdirectory/employee.txt
```

Step 11: Read File in HDFS

```
[cloudera@quickstart ~]$ hadoop fs -cat /inputdirectory/employee.txt
1~Sachine~Pune~Product Engineering~100000~Big Data
2~Gaurav~Bengalore~Sales~90000~CRM
3~Manish~Chennai~Recruiter~125000~HR
[cloudera@quickstart ~]$ hdfs dfs -ls /
```

Step 12: Enter Hive Shell

```
[cloudera@quickstart ~]$ hive

Logging initialized using configuration in jar:file:/usr/lib/hive/lib/hive-common-1.1.0-cdh5.13.0.jar!/hive-log4j.properties
WARNING: Hive CLI is deprecated and migration to Beeline is recommended.
hive> █
```

Step 13: Show Databases in Hive

```
hive> show databases;
OK
default
organization
Time taken: 1.541 seconds, Fetched: 2 row(s)
```

Step 14: Create Database

```
hive> create database organisation;
OK
```

Step 15: Show Databases Again

```
hive> show databases;
OK
default
organisation
organization
Time taken: 0.021 seconds, Fetched: 3 row(s)
```

Step 16: Change Database

```
hive> use organisation;  
OK  
--
```

Step 17: Create Table in Hive

```
hive> CREATE TABLE employee (  
  >   name STRING,  
  >   city STRING,  
  >   department STRING,  
  >   salary INT,  
  >   domain STRING  
  > ) ROW FORMAT DELIMITED  
  > FIELDS TERMINATED BY '~';  
OK  
Time taken: 0.402 seconds
```

Step 18: Show Tables in Hive

```
hive> show tables;  
OK  
employee  
Time taken: 0.078 seconds, Fetched: 1 row(s)  
--
```

Step 19: Read Content from Employee Table (Empty)

```
hive> select * from employee;  
OK  
Time taken: 0.455 seconds
```

Step 20: Load Data into Employee Table

```
hive> load data inpath '/inputdirectory/employee.txt' overwrite into table employee;  
Loading data to table organisation.employee  
Table organisation.employee stats: [numFiles=1, numRows=0, totalSize=123, rawDataSize=0]  
OK  
Time taken: 0.497 seconds
```

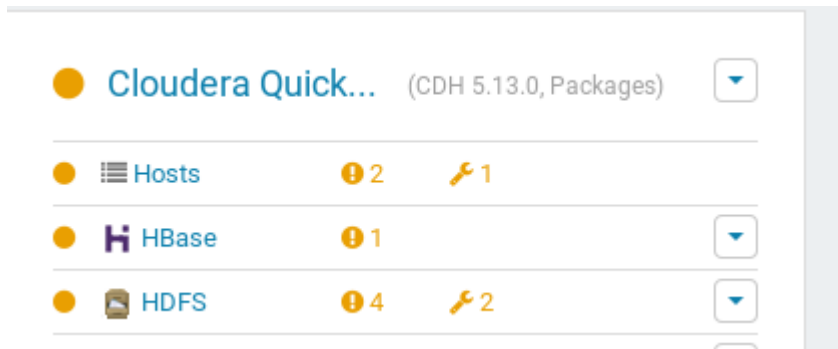
Step 21: Read Content from Employee Table (With Data)

```
hive> select * from employee;
OK
1      Sachine Pune      NULL      100000
2      Gaurav  Bangalore      NULL      90000
3      Manish  Chennai NULL      125000
Time taken: 0.078 seconds, Fetched: 3 row(s)
hive> █
```


Practical 6 - Install HBase and use the HBase Data model to store and retrieve data

Step 1: Start Cloudera

Start your Cloudera and start services HBase, HDFS



Step 2: Start the HBase Shell

Open a terminal and start the HBase shell:

```
[cloudera@quickstart ~]$ hbase shell
24/02/21 23:14:14 INFO Configuration.deprecation: hadoop.native.lib is deprecated. Instead, use io.native.lib.available
HBase Shell; enter 'help<RETURN>' for list of supported commands.
Type "exit<RETURN>" to leave the HBase Shell
Version 1.2.0-cdh5.13.0, rUnknown, Wed Oct 4 11:16:18 PDT 2017
```

Step 3: Check HBase Status, Version, and User Information

Execute the following commands to check the HBase status, version, and user information:

```
hbase(main):026:0> whoami
cloudera (auth:SIMPLE)
  groups: cloudera, default

hbase(main):027:0> status
1 active master, 0 backup masters, 1 servers, 0 dead, 4.0000 average load

hbase(main):028:0> version
1.2.0-cdh5.13.0, rUnknown, Wed Oct 4 11:16:18 PDT 2017
```

Status should be active, if not restart HBase in Cloudera manager

Step 4: Create 'employee' Table

Create the 'employee' table with columns 'Name', 'ID', 'Designation', 'Salary', and 'Department':

```
hbase(main):029:0> create 'employee','Name','ID','Designation','Salary','Department'
```

Step 5: Verify 'employee' Table Creation

List the tables to verify the 'employee' table creation:

```
hbase(main):004:0> list
TABLE
employee
1 row(s) in 0.0610 seconds
```

Step 6: Scan the 'employee' Table

Scan the 'employee' table

```
hbase(main):006:0> scan 'employee'
ROW          COLUMN+CELL
0 row(s) in 0.2730 seconds
```

Step 6: Create 'student' Table

Create the 'student' table with columns 'name', 'age', and 'course':

```
hbase(main):007:0> create 'student', 'name', 'age', 'course'
0 row(s) in 1.2710 seconds
```

Step 07: Insert Data into 'student' Table

Insert data for three students into the 'student' table:

```
hbase(main):008:0> put 'student','om','name:fullname','Om Panchal'
0 row(s) in 0.1740 seconds

hbase(main):009:0> put 'student','om','age:presentage','22'
0 row(s) in 0.0240 seconds

hbase(main):010:0> put 'student','om','course:pursuing','Big Data'
0 row(s) in 0.0270 seconds

hbase(main):011:0> put 'student','srushti','name:fullname','Srushti K'
0 row(s) in 0.0720 seconds

hbase(main):012:0> put 'student','srushti','age:presentage','25'
0 row(s) in 0.0430 seconds

hbase(main):013:0> put 'student','srushti','course:pursuing','Machine Learning'
0 row(s) in 0.0200 seconds

hbase(main):014:0> put 'student','meghana','name:fullname','Meghana N'
0 row(s) in 0.0430 seconds

hbase(main):015:0> put 'student','meghana','age:presentage','23'
0 row(s) in 0.0490 seconds

hbase(main):016:0> put 'student','meghana','course:pursuing','Artificial Intelli
gence'
0 row(s) in 0.0340 seconds
```

Step 08: Retrieve Data from 'student' Table

Retrieve and display data for all students in the 'student' table:

```
hbase(main):017:0> get 'student','om'
COLUMN      CELL
age:presentage    timestamp=1708586414443, value=22
course:pursuing   timestamp=1708586424069, value=Big Data
name:fullname     timestamp=1708586403699, value=Om Panchal
3 row(s) in 0.0150 seconds

hbase(main):018:0> get 'student','srushti'
COLUMN      CELL
age:presentage    timestamp=1708586446979, value=25
course:pursuing   timestamp=1708586457550, value=Machine Learning
name:fullname     timestamp=1708586435456, value=Srushti K
3 row(s) in 0.0080 seconds

hbase(main):019:0> get 'student','meghana'
COLUMN      CELL
age:presentage    timestamp=1708586479058, value=23
course:pursuing   timestamp=1708586494027, value=Artificial Intelligence
name:fullname     timestamp=1708586469353, value=Meghana N
3 row(s) in 0.0260 seconds
```

Step 09: Scan and Count the 'student' Table

Scan and count the rows in the 'student' table:

```
hbase(main):020:0> scan 'student'
ROW          COLUMN+CELL
meghana      column=age:presentage, timestamp=1708586479058, value=23
meghana      column=course:pursuing, timestamp=1708586494027, value=Artificial Intelligence
meghana      column=name:fullname, timestamp=1708586469353, value=Meghana N
om           column=age:presentage, timestamp=1708586414443, value=22
om           column=course:pursuing, timestamp=1708586424069, value=Big Data
om           column=name:fullname, timestamp=1708586403699, value=Om Panchal
srushti      column=age:presentage, timestamp=1708586446979, value=25
srushti      column=course:pursuing, timestamp=1708586457550, value=Machine Learning
srushti      column=name:fullname, timestamp=1708586435456, value=Srushti K
3 row(s) in 0.0450 seconds
```

Step 10: Alter the 'student' Table

Alter the 'student' table to set the maximum number of versions for the 'name' column to 5:

```
hbase(main):021:0> alter 'student', NAME=>'name', VERSIONS=>5
Updating all regions with the new schema...
0/1 regions updated.
1/1 regions updated.
Done.
0 row(s) in 3.0520 seconds
```

Step 11: Put Altered Values for 'srushti'

Put altered values for the 'name' column for 'srushti':

```
hbase(main):022:0> put 'student','srushti','name:fullname','Srushti Kulkarni'
0 row(s) in 0.0120 seconds
```

Step 18: Scan 'student' Table to Check Alteration

Scan the 'student' table to check if the 'name' alteration has taken effect:

```
hbase(main):023:0> scan 'student'
ROW          COLUMN+CELL
meghana      column=age:presentage, timestamp=1708586479058, value=23
meghana      column=course:pursuing, timestamp=1708586494027, value=Artificial Intelligence
meghana      column=name:fullname, timestamp=1708586469353, value=Meghana N
om           column=age:presentage, timestamp=1708586414443, value=22
om           column=course:pursuing, timestamp=1708586424069, value=Big Data
om           column=name:fullname, timestamp=1708586403699, value=Om Panchal
srushti      column=age:presentage, timestamp=1708586446979, value=25
srushti      column=course:pursuing, timestamp=1708586457550, value=Machine Learning
srushti      column=name:fullname, timestamp=1708586618426, value=Srushti Kulkarni
3 row(s) in 0.0250 seconds
```

Step 12: Delete 'name' Column for 'meghana'

Delete the 'name' column for 'meghana':

```
hbase(main):024:0> delete 'student','meghana','name:fullname'
0 row(s) in 0.0370 seconds
```

Step 13: Check if 'name' Column is Deleted for 'meghana'

Check if the 'name' column has been deleted for 'meghana':

```
hbase(main):025:0> get 'student','meghana'
COLUMN                                CELL
age:presentage                        timestamp=1708586479058, value=23
course:pursuing                       timestamp=1708586494027, value=Artificial Intelligence
2 row(s) in 0.0090 seconds
```

Practical 7 - Perform importing and exporting of data between SQL and Hadoop using Sqoop

Before running Sqoop commands, it's essential to ensure that Sqoop services are started on the Cloudera manager.

Step 1: Connect to MySQL

Open a terminal and connect to MySQL using the following command:

PASSWORD - CLOUDERA

```
[cloudera@quickstart ~]$ mysql -D retail_db -u retail_dba -p
Enter password:
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 15
Server version: 5.1.73 Source distribution

Copyright (c) 2000, 2013, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> █
```

Step 2: List Tables in MySQL

List the tables in the retail_db database:

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

Step 3: Select Data from MySQL Table

Select data from the customers table to verify the existing data:

```
mysql> select * from customers;
```

Step 4: Open a New Terminal and Run Sqoop Import

Open a new terminal and run the Sqoop import command to transfer data from MySQL to Hadoop. The command connects to MySQL, specifies the table, username, password, and target directory:

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

sqoop import --connect jdbc:mysql://localhost/retail_db --table customers --username retail_dba --password cloudera --target-dir /sqoop_import_data -m 1
```

Step 5: View Imported Data in Hadoop

Check if the data has been successfully transferred from MySQL to Hadoop. Use the following command to view the content of the imported data file:

```
[cloudera@quickstart ~]$ hadoop fs -cat /sqoop_import_data/part-m-000000
```

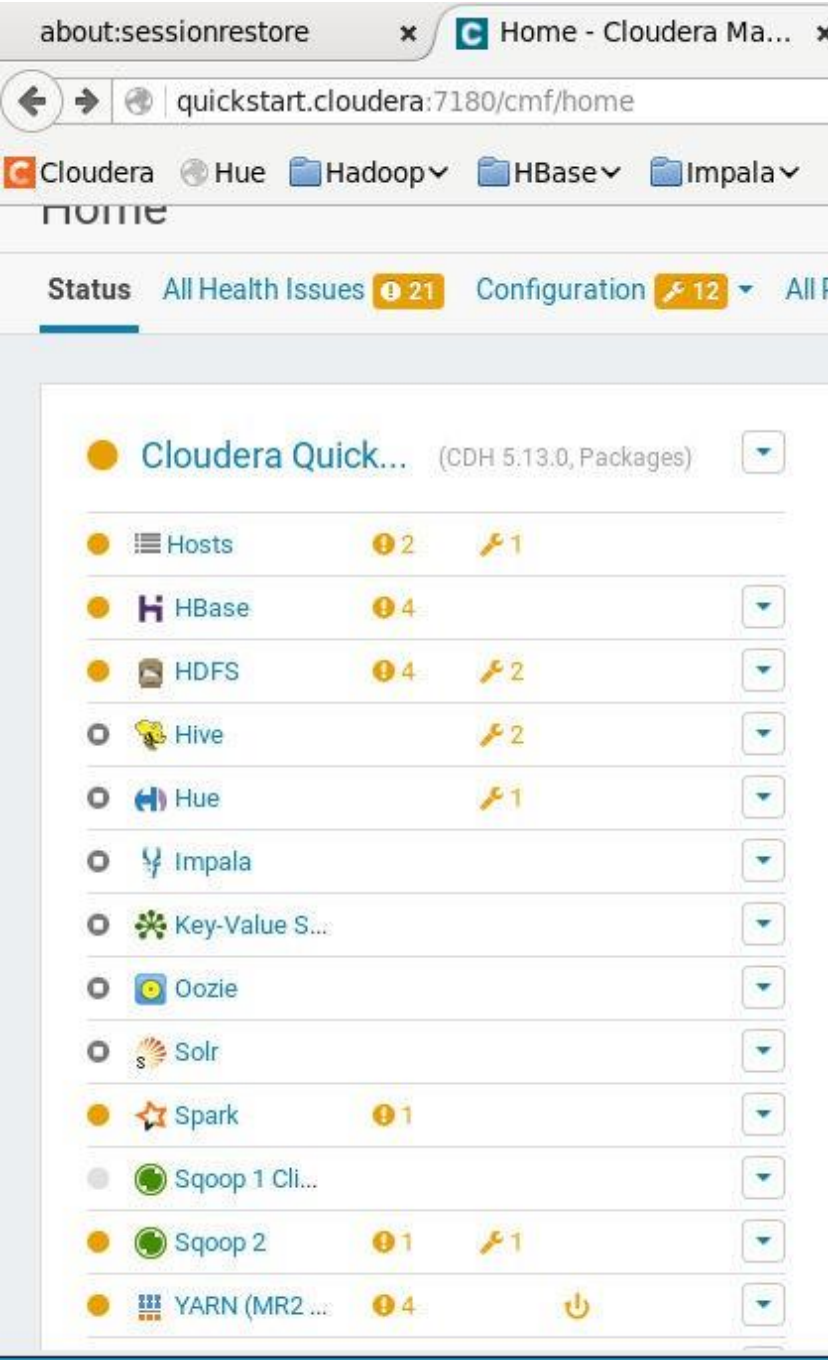

This command will display the contents of the imported data file. Ensure that the data has been transferred correctly.

```
12273,Mary,Dunn,XXXXXXXX,XXXXXXXX,1020 Misty Hills Round, Caguas, PR, 00723
12274,Joyce,Smith,XXXXXXXX,XXXXXXXX,5492 Blue View, Staten Island, NY, 10314
12275,Mary,Dudley,XXXXXXXX,XXXXXXXX,4212 Honey Trail, Modesto, CA, 95355
12276,Martha,Smith,XXXXXXXX,XXXXXXXX,4385 Round Place, Caguas, PR, 00725
12277,Mary,Baxter,XXXXXXXX,XXXXXXXX,629 Grand Inlet, Pacoima, CA, 91331
12278,Karen,Smith,XXXXXXXX,XXXXXXXX,3453 Honey Woods, Philadelphia, PA, 19104
12279,Mary,Dunn,XXXXXXXX,XXXXXXXX,5860 Grand Limits, Caguas, PR, 00725
12280,John,Smith,XXXXXXXX,XXXXXXXX,1550 Rustic Grove Farms, Tallahassee, FL, 32308
12281,Scott,Miller,XXXXXXXX,XXXXXXXX,8620 Burning Isle, Louisville, KY, 40214
12282,Mary,Wilson,XXXXXXXX,XXXXXXXX,2781 Emerald Thicket, Caguas, PR, 00725
12283,Willie,Pena,XXXXXXXX,XXXXXXXX,6654 Green Crest, San Juan, PR, 00926
12284,Mary,Smith,XXXXXXXX,XXXXXXXX,1185 Rustic River Thicket, New Haven, CT, 06511
12285,Patrick,Mcgee,XXXXXXXX,XXXXXXXX,2059 Sunny Cape, Caguas, PR, 00725
12286,Mary,Smith,XXXXXXXX,XXXXXXXX,5284 Quiet Field, Baltimore, MD, 21224
12287,Robert,Lee,XXXXXXXX,XXXXXXXX,5387 Amber Campus, Caguas, PR, 00725
12288,Karen,Smith,XXXXXXXX,XXXXXXXX,1726 Velvet Thicket, Algonquin, IL, 60102
12289,Andrea,Price,XXXXXXXX,XXXXXXXX,6471 Dusty Limits, Caguas, PR, 00725
12290,Mary,Hawkins,XXXXXXXX,XXXXXXXX,3060 Emerald Crossing, Enfield, CT, 06082
12291,Jennifer,Sims,XXXXXXXX,XXXXXXXX,3289 Thunder Prairie Inlet, Caguas, PR, 00725
12292,Doris,May,XXXXXXXX,XXXXXXXX,7291 Thunder Hills Knoll, Spring Valley, CA, 91977
12293,Mary,Smith,XXXXXXXX,XXXXXXXX,6764 Harvest View Point, Las Vegas, NV, 89117
12294,Theresa,Reid,XXXXXXXX,XXXXXXXX,3236 Grand Carrefour, Mission Viejo, CA, 92692
12295,Mary,Fischer,XXXXXXXX,XXXXXXXX,275 Heather Rabbit Parkway, Caguas, PR, 00725
12296,Anna,Smith,XXXXXXXX,XXXXXXXX,7596 Silent Mews, Orlando, FL, 32822
12297,Mary,Simpson,XXXXXXXX,XXXXXXXX,8428 Thunder Private, Lenoir, NC, 28645
12298,Mary,Harper,XXXXXXXX,XXXXXXXX,2078 Little Forest Corner, Caguas, PR, 00725
12299,Randy,Smith,XXXXXXXX,XXXXXXXX,9640 Golden Hill, Chicago, IL, 60660
12300,Mary,Lindsey,XXXXXXXX,XXXXXXXX,577 Old Branch Jetty, Roswell, GA, 30075
12301,John,Smith,XXXXXXXX,XXXXXXXX,4393 Noble Creek Beach, Caguas, PR, 00725
12302,Donald,Sampson,XXXXXXXX,XXXXXXXX,8550 Red Oak Bank, Austin, TX, 78753
12303,Mary,Smith,XXXXXXXX,XXXXXXXX,2357 Cinder Anchor Route, Caguas, PR, 00725
12304,David,Tran,XXXXXXXX,XXXXXXXX,2578 Amber Barn Turnabout, Pasadena, MD, 21122
12305,Stephanie,Leblanc,XXXXXXXX,XXXXXXXX,164 Dewy Nook, Caguas, PR, 00725
12306,Juan,Smith,XXXXXXXX,XXXXXXXX,2484 Golden Wagon Pointe, Goose Creek, SC, 29445
12307,Mary,Glover,XXXXXXXX,XXXXXXXX,6940 Pleasant Zephyr Via, Caguas, PR, 00725
12308,Joseph,Smith,XXXXXXXX,XXXXXXXX,355 Wishing Rise Ledge, Chicago, IL, 60644
12309,Barbara,Pena,XXXXXXXX,XXXXXXXX,9414 Fallen Hickory Wood, Chicago, IL, 60610
12310,Ralph,Nielsen,XXXXXXXX,XXXXXXXX,6639 Clear Fawn Grounds, Caguas, PR, 00725
12311,Mary,Smith,XXXXXXXX,XXXXXXXX,7368 Blue Swale, Los Angeles, CA, 90066
12312,Jessica,Figueroa,XXXXXXXX,XXXXXXXX,1757 Hidden Beacon Highway, Gaithersburg, MD, 20878
12313,Andrea,Wang,XXXXXXXX,XXXXXXXX,5717 Quiet Dell, Chandler, AZ, 85224
12314,Bruce,Smith,XXXXXXXX,XXXXXXXX,9283 Hidden Elk Terrace, Caguas, PR, 00725
12315,Mary,Jones,XXXXXXXX,XXXXXXXX,1321 Easy Embers Lane, Santa Fe, NM, 87505
12316,Kathy,Garcia,XXXXXXXX,XXXXXXXX,3544 Velvet Crescent, Chicago, IL, 60615
12317,Shawn,Cross,XXXXXXXX,XXXXXXXX,1981 Honey Nectar Line, Caguas, PR, 00725
12318,Mary,Smith,XXXXXXXX,XXXXXXXX,5945 UMBER Crossing, Saint Paul, MN, 55106
12319,Laura,Woodward,XXXXXXXX,XXXXXXXX,6231 Fallen Range, Caguas, PR, 00725
12320,Mary,Smith,XXXXXXXX,XXXXXXXX,1811 Hazy Oak Subdivision, Elyria, OH, 44035
```

Practical 8 - Write a Pig Script for solving counting problems.

Step 1: Start Necessary Services

Before starting the practical, ensure that the necessary services, including HDFS,YARN are running on the Cloudera cluster.



Step 2: Create Input File input.csv

Open a new terminal and create a CSV file named input.csv:

```
[cloudera@quickstart ~]$ cat > /home/cloudera/input.csv
People die when they are killed.
Don't talk, it makes you sound stupid.
The ocean is so salty because everyone pees in it.
If you see a stranger, follow him.^Z
[1]+  Stopped                  cat > /home/cloudera/input.csv
```

View the data –

```
[cloudera@quickstart ~]$ cat /home/cloudera/input.csv
People die when they are killed.
Don't talk, it makes you sound stupid.
The ocean is so salty because everyone pees in it.
[cloudera@quickstart ~]$ █
```

Step 3: View Data Through Pig

To view the data through Pig, enter the Pig shell:

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

Step 4: Pig Shell Prompt

After the above command, your prompt will change to (grunt>).

```
grunt> █
```

Step 5: Pig Code for Word Count

Enter the following Pig 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 word;
grunt> grouped = GROUP words BY word;
grunt> wordcount = FOREACH grouped GENERATE group, COUNT(words);
grunt> DUMP wordcount;
```

Step 6: View the Output

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

```
HadoopVersion  PigVersion  UserId  StartedAt  FinishedAt  Features
2.6.0-cdh5.13.0 0.12.0-cdh5.13.0  cloudera  2024-02-22 00:32:13  2024-02-22 00:32:27  GROUP_BY

Success!

Job Stats (time in seconds):
JobId  Alias  Feature Outputs
job_local1793176200_0001  grouped,lines,wordcount,words  GROUP_BY,COMBINER  file:/tmp/temp443132024/tmp2122179645,

Input(s):
Successfully read records from: "/home/cloudera/input.csv"

Output(s):
Successfully stored records in: "file:/tmp/temp443132024/tmp2122179645"

Job DAG:
job_local1793176200_0001
```

```
(in,1)
(is,1)
(it,1)
(so,1)
(The,1)
(are,1)
(die,1)
(it.,1)
(you,1)
(pees,1)
(talk,1)
(they,1)
(when,1)
(makes,1)
(ocean,1)
(salty,1)
(sound,1)
(People,1)
(Don't,1)
(because,1)
(killed.,1)
(stupid.,1)
(everyone,1)
grunt> █
```