# ERODE SENGUNTHAR
## ENGINEERING COLLEGE
**(An Autonomous Institution)**

Approved by AICTE, New Delhi, Permanently Affiliated to Anna University- Chennai,
Accredited by National Board of Accreditation (NBA), New Delhi &
National Assessment and Accreditation Council (NAAC), Bangalore with 'A' Grade

**PERUNDURAI -638 057**

# BONAFIDE CERTIFICATE

Register No……

Certified that this is the Bonafide Record of Work Done

Name of the Student : _____

Branch : _____

Name of the lab : _____

Semester / Year : _____

Faculty Incharge Head of the Department

Submitted for the End Semester Practical Examination

held on _____

Internal Examiner External Examiner

| EX.NO: 1 | DOWNLOAD AND INSTALL HADOOP; UNDERSTAND DIFFERENT HADOOP MODES; STARTUP SCRIPTS, CONFIGURATION FILES |
|---|---|
| DATE: | |

## Aim:

To Download and install Hadoop; Understand different Hadoop modes.Startup scripts, Configuration files.

**Procedure:**

**1)Prerequisites**

1. Java 8 runtime environment (JRE)
2. Apache Hadoop 3.3.0

**Step 1 - Download Hadoop binary package**

The first step is to download Hadoop binaries from the official website.
https://archive.apache.org/dist/hadoop/common/hadoop-3.3.0/hadoop-3.3.0.tar.gz

**Step 2 - Unpack the package**

After finishing the file download, we should unpack the package using 7zip or command line.

```
>>cd Downloads
```

Because I am installing Hadoop in folder Haddop of my C drive (C:\Hadoop) we create the the directory

```
>>mkdir C:\Hadoop
```

then run the following command to unzip:

```
>>tar -xvzf  hadoop-3.3.0.tar.gz -C C:\Hadoop\
```

After the unzip command is completed we have to install the Java.

**Step 3 -Java installation**

You can install Java 8 from the following page
https://www.oracle.com/it/java/technologies/javase/javase8-archive-downloads.html#license-lightbox.
After finishing the file download we open a new command prompt, we should unpack the package

```
>>cd Downloads
```

Because I am installing Java in folder Java of my C drive (C:\Java)

```
>>mkdir C:\Java
```

then run the following command to unzip:

```
>>tar -xvzf  jre-8u361-windows-x64.tar.gz -C C:\Java\
```

**Step 3 - Install Hadoop native IO binary**

Hadoop on Linux includes optional Native IO support. However Native IO is mandatory on Windows and without it you will not be able to get your installation working.

The following repository already pre-built Hadoop Windows native libraries:

https://github.com/ruslanmv/How-to-install-Hadoop-on-Windows/tree/master/winutils/hadoop-3.3.0-YARN-8246/bin

Download all the files in the following location and save them to the bin folder under Hadoop folder.

**Step 5 - Configure environment variables**

Now we've downloaded and unpacked all the artefacts we need to configure two important environment variables.

We configure **JAVA_HOME** environment variable:

Variable name : `JAVA_HOME`
Variable value:  `C:\Java\jre1.8.0_361`

We configure **HADOOP_HOME** environment variable:

Variable name : `HADOOP_HOME`
Variable value:  `C:\Hadoop\hadoop-3.3.0’

Configure PATH environment variable

Once we finish setting up the above two environment variables, we need to add the bin folders to the PATH environment variable. We click on Edit

If PATH environment exists in your system, you can also manually add the following two paths to it:

**%JAVA_HOME%/bin**
**%HADOOP_HOME%/bin**

**Verification of Installation**

Once you complete the installation, Close your terminal window and open a new one and please run the following command  to verify:

>>java -version

**java version "1.8.0_361"**
**Java(TM) SE Runtime Environment (build 1.8.0_361-b09)**
**Java HotSpot(TM) 64-Bit Server VM (build 25.361-b09, mixed mode)**

You should also be able to run the following command:

>>hadoop -version

**Hadoop 3.3.6**
**Source code repository https://github.com/apache/hadoop.git -r 7f7f523c6f9a**
**Compiled by user on 2023-03-08T20:30Z**
**Compiled with protoc 3.7.1**

**Step 6 - Configure Hadoop**

Now we are ready to configure the most important part - Hadoop configurations which involves Core, YARN, MapReduce, HDFS configurations.

**a)Configure core site**

Edit file core-site.xml in %HADOOP_HOME%\etc\hadoop folder.
Replace configuration element with the following:

```
xml
<configuration>
  <property>
    <name>fs.default.name</name>
    <value>hdfs://0.0.0.0:19000</value>
  </property>
</configuration>
```

**b)Configure HDFS**

Edit file hdfs-site.xml in %HADOOP_HOME%\etc\hadoop folder.

Before editing, please correct two folders in your system: one for namenode directory and another for data directory. For my system, I created the following two sub folders:

>>mkdir C:\hadoop\hadoop-3.3.0\data\datanode

>>mkdir C:\hadoop\hadoop-3.3.0\data\namenode

```
xml
<configuration>
  <property>
    <name>dfs.replication</name>
    <value>1</value>
  </property>
  <property>
    <name>dfs.namenode.name.dir</name>
    <value>/hadoop/hadoop-3.3.0/data/namenode</value>
  </property>
  <property>
    <name>dfs.datanode.data.dir</name>
    <value>/hadoop/hadoop-3.3.0/data/datanode</value>
  </property>
</configuration>
```

**Configure MapReduce and YARN site:**

Edit file mapred-site.xml in %HADOOP_HOME%\etc\hadoop folder.

Replace configuration element with the following:

```
xml
<configuration>
   <property>
      <name>mapreduce.framework.name</name>
      <value>yarn</value>
   </property>
   <property>
      <name>mapreduce.application.classpath</name>

<value>%HADOOP_HOME%/share/hadoop/mapreduce/*,%HADOOP_HOME%/sh
are/hadoop/mapreduce/lib/*,%HADOOP_HOME%/share/hadoop/common/*,%HAD
OOP_HOME%/share/hadoop/common/lib/*,%HADOOP_HOME%/share/hadoop/yar
n/*,%HADOOP_HOME%/share/hadoop/yarn/lib/*,%HADOOP_HOME%/share/had
oop/hdfs/*,%HADOOP_HOME%/share/hadoop/hdfs/lib/*</value>
   </property>
</configuration>
```

Edit file yarn-site.xml in %HADOOP_HOME%\etc\hadoop folder.

```
xml
<configuration>
   <property>
      <name>yarn.nodemanager.aux-services</name>
      <value>mapreduce_shuffle</value>
```

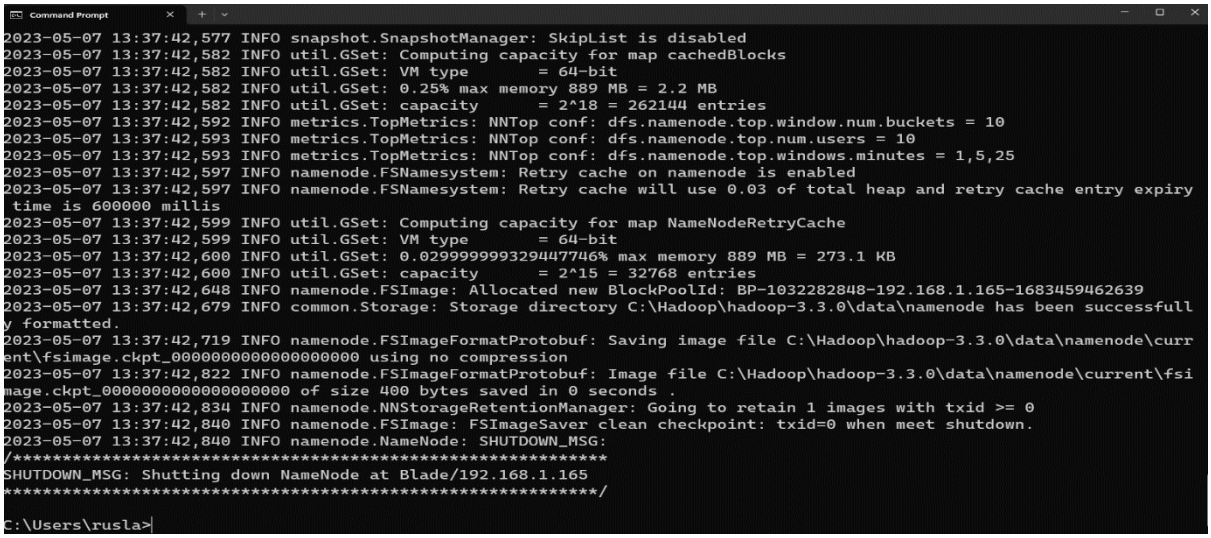
```
  </property>
  <property>
    <name>yarn.nodemanager.env-whitelist</name>

<value>JAVA_HOME,HADOOP_COMMON_HOME,HADOOP_HDFS_HOME,HADOOP_CONF_DIR,CLASSPATH_PREPEND_DISTCACHE,HADOOP_YARN_HOME,HADOOP_MAPRED_HOME</value>
  </property>
</configuration>
```

**Step 7 - Initialise HDFS**

Run the following command in Command Prompt

>>hdfs namenode -format

The following is an example when it is formatted successfully:

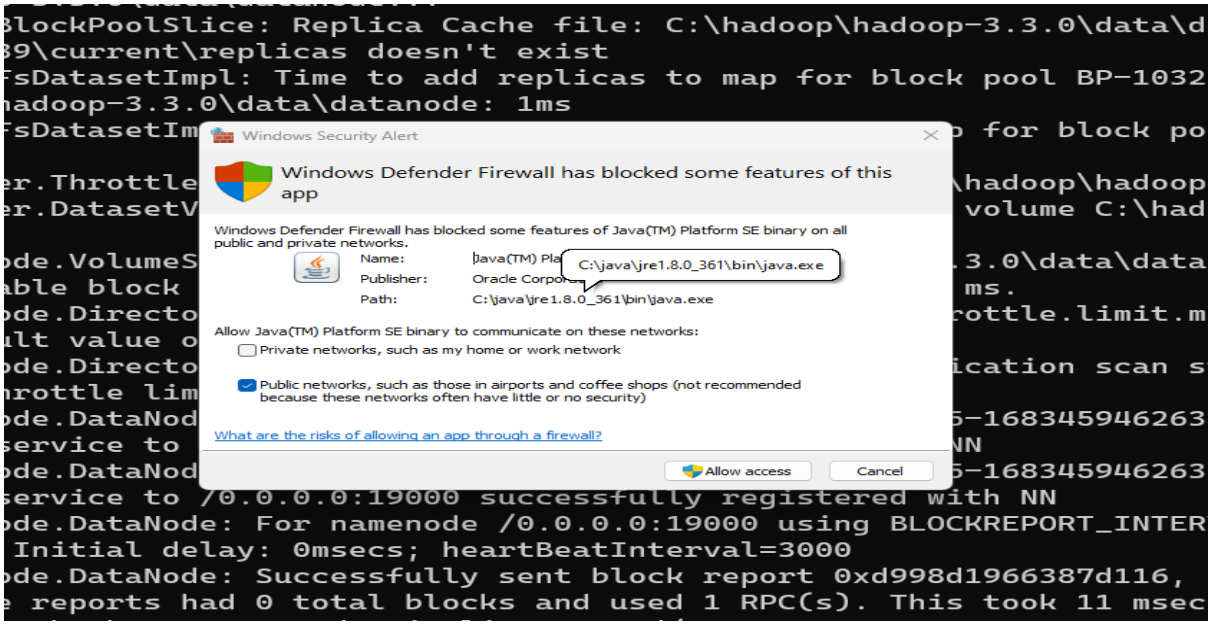



**Step 8 - Start HDFS daemons**

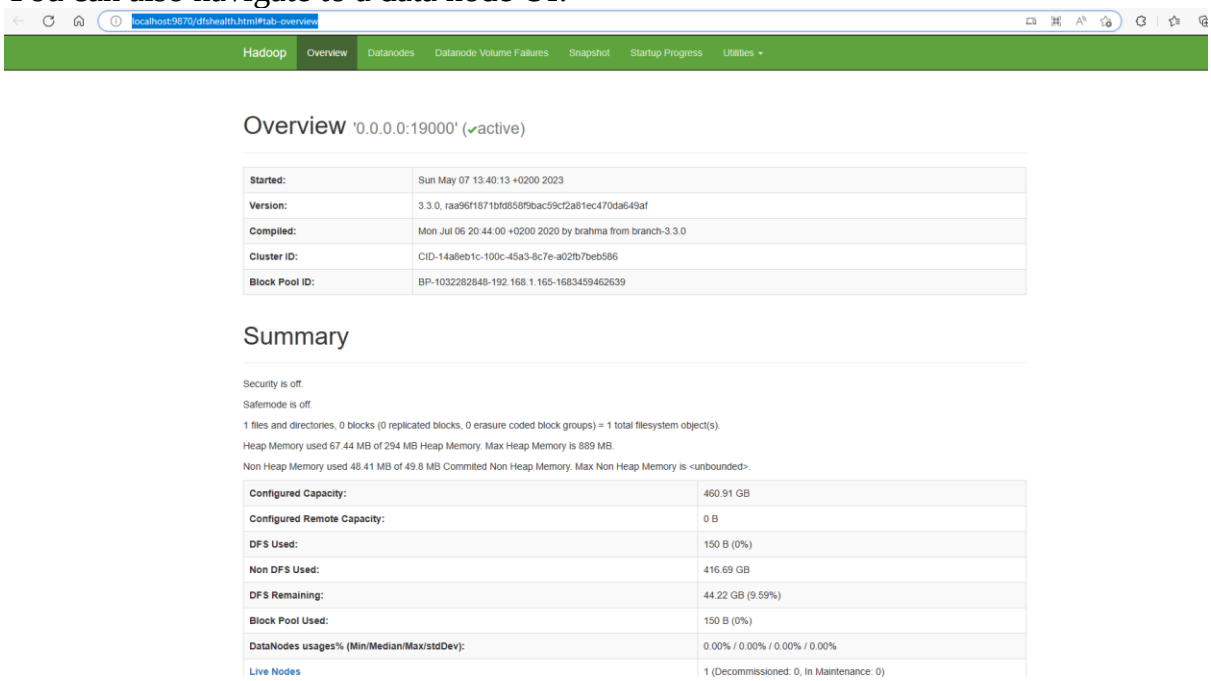
Run the following command to start HDFS daemons in Command Prompt:

>>%HADOOP_HOME%\sbin\start-dfs.cmd

Please click Allow access to the java.




Verify HDFS web portal UI through this link:
http://localhost:9870/dfshealth.html#tab-overview.
You can also navigate to a data node UI:



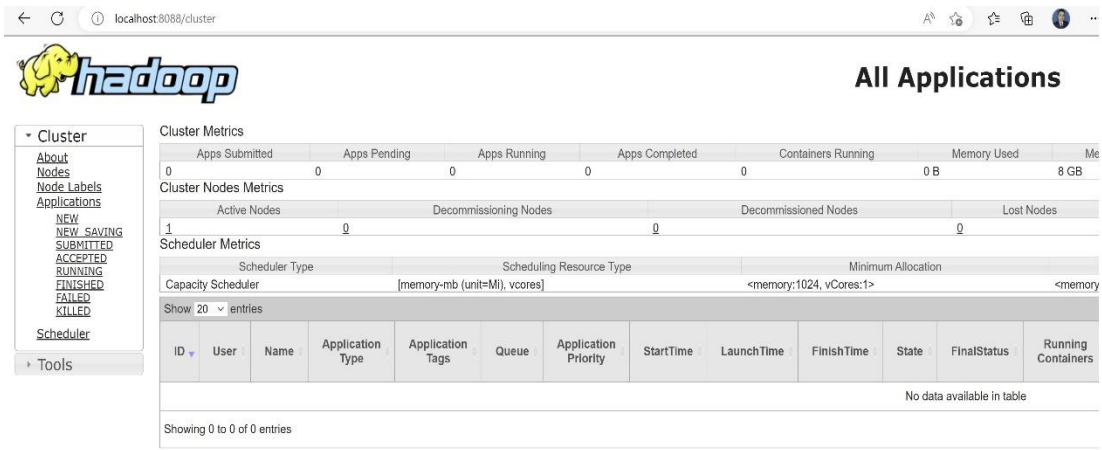

**Step 9 - Start YARN daemons**

Run the following command in an elevated Command Prompt window (Run as administrator) to start YARN daemons:

>>%HADOOP_HOME%\sbin\start-yarn.cmd

You can verify YARN resource manager UI when all services are started successfully.

http://localhost:8088




## Result:

Thus downloading and installing Hadoop &Understanding different Hadoop modes was successfully executed.

| EX.NO: 2 | HADOOP IMPLEMENTATION OF FILE MANAGEMENT TASKS, SUCH AS ADDING FILES AND DIRECTORIES, RETRIEVING FILES AND DELETING FILES |
|---|---|
| DATE: | |

## Aim:

To Implement the following file management tasks in Hadoop: Adding files and directories Retrieving files Deleting Files.

**Procedure:**

**Step-1: Adding Files and Directories to HDFS**

Before you can run Hadoop programs on data stored in HDFS, you'll need to put the data into
HDFS first.
Let's create a directory and put a file in it. HDFS has a default working directory of /user/$USER, where $USER is your login username. This directory isn't automatically created
for you, so let's create it using the mkdir command.
For the purpose of illustration, we use the username chuck.
You should substitute your own username in the example commands.

```
>>hdfs dfs -mkdir /user/chuck
>>hdfs dfs -put example.txt /user/chuck
```

**Step-2: Retrieving Files from HDFS**
The Hadoop get command copies files from HDFS back to the local filesystem. To retrieve example.txt, run the following command:

```
>>hdfs dfs -cat /user/chuck/example.txt
```

**Step-3: Deleting Files from HDFS**
To delete a file from HDFS, use the rm command: hdfs dfs -rm /user/chuck/example.txt To delete a directory recursively, use the -rm -r or -rmr option:

```
>> hdfs dfs -rm -r /user/chuck
```

**Step-4: Copying Data from NFS (Local File System) to HDFS**
Use the copyFromLocal command to copy data from a local directory to HDFS:

```
>>hdfs dfs -copyFromLocal /home/lendi/Desktop/shakes/glossary
/lendi_english/
```

**Step-5: Viewing File Contents in HDFS**
Use the cat command to view the contents of a file stored in HDFS:

```
>>hdfs dfs -cat /lendi_english/glossary
```

**Step-6: Listing Items in HDFS**
Use the ls command to list files and directories in HDFS:

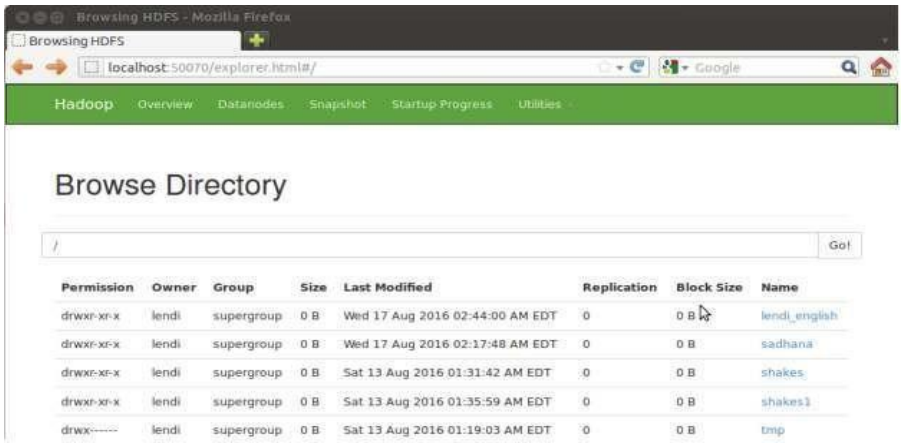
```
>>hdfs dfs -ls hdfs://localhost:9000/
```

**Step-7: Deleting Files or Directories in HDFS**
Use the rm or rmr command to remove files or directories:

```
>>hdfs dfs -rmr /kartheek
```

Note: rmr is deprecated; prefer -rm -r in recent Hadoop versions.

**OUTPUT:**




**Result:**
Thus, Hadoop Implementation of file management tasks, such Adding files and directories, retrieving files and Deleting files has been successfully completed.

| EX.NO: 3 | IMPLEMENT OF MATRIX MULTIPLICATION WITH HADOOP MAP REDUCE |
|---|---|
| DATE: | |

## Aim:

To Implement of Matrix Multiplication with Hadoop Map Reduce.

## Algorithm:

### For Map Function:

for each element m_ij of M do
  produce (key, value) pairs as ((i, k), (M, j, m_ij))
  for k = 1, 2, 3, … up to the number of columns of N
end for

for each element n_jk of N do
  produce (key, value) pairs as ((i, k), (N, j, n_jk))
  for i = 1, 2, 3, … up to the number of rows of M
end for

return set of (key, value) pairs such that each key (i, k)
has a list with values (M, j, m_ij) and (N, j, n_jk)
for all possible values of j.

### For Reduce Function:

for each key (i, k) do
  sort values beginning with M by j → listM
  sort values beginning with N by j → listN
end for

## Program:

**Step 1. Download the hadoop jar files with these links.**

Download Hadoop Common Jar files: https://goo.gl/G4MyHp 5 wget https://goo.gl/G4MyHp -O hadoop- common-2.2.0.jar
Download Hadoop Mapreduce Jar File: https://goo.gl/KT8yf

$ wget https://goo.gl/KT8yfB-O hadoop-mapreduce-client-core-2.7.1.jar

**Step 2. Creating Mappy file for Matrix Multiplication.**

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

import java.io.IOException;

public class MatrixMapper extends Mapper<LongWritable, Text, Text, Text> {

   @Override
   public void map(LongWritable key, Text value, Context context) throws
IOException, InterruptedException {
      Configuration conf = context.getConfiguration();

      int m = Integer.parseInt(conf.get("m")); // rows of M
      int p = Integer.parseInt(conf.get("p")); // columns of N

      String line = value.toString();
      // Input format: MatrixName,i,j,value
      // Example: M,0,1,2 or N,1,0,3
      String[] indicesAndValue = line.split(",");

      String matrixName = indicesAndValue[0];
      int row = Integer.parseInt(indicesAndValue[1]);
      int col = Integer.parseInt(indicesAndValue[2]);
      int val = Integer.parseInt(indicesAndValue[3]);

      Text outputKey = new Text();
      Text outputValue = new Text();

      if (matrixName.equals("M")) {
         // For each element M[i][j], emit pairs for all k (columns of N)
         for (int k = 0; k < p; k++) {
            outputKey.set(row + "," + k);   // (i, k)
            outputValue.set("M," + col + "," + val);  // (M, j, mij)
            context.write(outputKey, outputValue);
         }
      } else {
         // For each element N[j][k], emit pairs for all i (rows of M)
         for (int i = 0; i < m; i++) {
            outputKey.set(i + "," + col);   // (i, k)
            outputValue.set("N," + row + "," + val);  // (N, j, njk)
            context.write(outputKey, outputValue);
         }
```

```
        }
    }
}
```

**Step 3. Creating Reducer.java file for Matrix Multiplication.**

```
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

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

public class MatrixReducer extends Reducer<Text, Text, Text, IntWritable> {

    @Override
    public void reduce(Text key, Iterable<Text> values, Context context)
            throws IOException, InterruptedException {

        // key: (i, k)
        // values: [(M, j, mij), (N, j, njk), …]

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

        for (Text val : values) {
            String[] value = val.toString().split(",");
            if (value[0].equals("M")) {
                // (M, j, mij)
                int j = Integer.parseInt(value[1]);
                float mVal = Float.parseFloat(value[2]);
                hashA.put(j, mVal);
            } else {
                // (N, j, njk)
                int j = Integer.parseInt(value[1]);
                float nVal = Float.parseFloat(value[2]);
                hashB.put(j, nVal);
            }
        }

        // Multiply matching indices (sum over j)
        float result = 0;
        for (int j : hashA.keySet()) {
            if (hashB.containsKey(j)) {
                result += hashA.get(j) * hashB.get(j);
            }
```

```
        }

        context.write(key, new IntWritable((int) result));
    }
}
```

**Step 4. Creating MatrixMultiply.java file**

```
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.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

public class MatrixMultiply {

    public static void main(String[] args) throws Exception {
        if (args.length != 2) {
            System.err.println("Usage: MatrixMultiply <in_dir> <out_dir>");
            System.exit(2);
        }

        Configuration conf = new Configuration();

        // Matrix dimensions (example values, must match input format)
        conf.set("m", "1000"); // number of rows in M
        conf.set("n", "100");  // number of columns in M / rows in N
        conf.set("p", "1000"); // number of columns in N
        Job job = Job.getInstance(conf, "MatrixMultiply");
        job.setJarByClass(MatrixMultiply.class);
        // Mapper and Reducer
        job.setMapperClass(MatrixMapper.class);
        job.setReducerClass(MatrixReducer.class);
        // Output types
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(Text.class);  // Mapper emits <Text, Text>
        // Input and Output formats
        job.setInputFormatClass(TextInputFormat.class);
        job.setOutputFormatClass(TextOutputFormat.class);
        // Input and Output paths
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

        // Exit after completion
        System.exit(job.waitForCompletion(true) ? 0 : 1);
```

```
    }
}
```

**Step 5. Compiling the program in particular folder named as operation**

5.1) Compile Mapper

javac -cp hadoop-common-2.2.0.jar:hadoop-mapreduce-client-core-2.7.1.jar:. -d operation/ MatrixMapper.java

5.2) Compile Reducer

javac -cp hadoop-common-2.2.0.jar:hadoop-mapreduce-client-core-2.7.1.jar:. -d operation/ MatrixReducer.java

5.3) Compile Driver

javac -cp hadoop-common-2.2.0.jar:hadoop-mapreduce-client-core-2.7.1.jar:operation/:. -d operation/ MatrixMultiply.java

**Step 6. Let's retrieve the directory after compilation.**

```
$ ls -R operation/
operation/:
www
operation/www:
ehadoopinfo
operation/www/ehadoopinfo:
com
operation/www/ehadoopinfo/com:
Map.class
MatrixMultiply.class
Reduce.class
```

**Step 7. Creating Jar file for the Matrix Multiplication.**

```
$ jar -cvf MatrixMultiply.jar -C operation/ .

added manifest
adding: www/ (in = 0) (out = 0) (stored 0%)
adding: www/ehadoopinfo/ (in = 0) (out = 0) (stored 0%)
adding: www/ehadoopinfo/com/ (in = 0) (out = 0) (stored 0%)
adding: www/ehadoopinfo/com/Reduce.class (in = 2919) (out = 1271) (deflated 56%)
adding: www/ehadoopinfo/com/MatrixMultiply.class (in = 1815) (out = 932) (deflated 48%)
adding: www/ehadoopinfo/com/Map.class (in = 2353) (out = 993) (deflated 57%)
```

**Step 8. Uploading the M, N file which contains the matrix multiplication data to HDFS.**

```
$ cat M
M,0,0,1
M,0,1,2
M,1,0,3
M,1,1,4
```

```
$ cat N
N,0,0,5
N,0,1,6
N,1,0,7
N,1,1,8
```

```
$ hadoop fs -mkdir Matrix/
$ hadoop fs -copyFromLocal M Matrix/
$ hadoop fs -copyFromLocal N Matrix/
```

**OUTPUT:**

```
S hadoop
fs-cat
result/part-
r-00000
0.0.19.0
0.1.22.0
1.0.43.0
1,1.50,0
```

**Result:**

Thus matrix multiplication using map reduce was executed successfully.

# INDEX

| S.NO | DATE | EXCERCISES | PAGE NO | MARKS | SIGN |
|---|---|---|---|---|---|
| 1. | | DOWNLOADING AND INSTALLING HADOOP; UNDERSTANDING DIFFERENT HADOOP MODES. STARTUP SCRIPTS, CONFIGURATION FILES. | | | |
| 2. | | HADOOP IMPLEMENTATION OF FILE MANAGEMENT TASKS, SUCH AS ADDING FILES AND DIRECTORIES, RETRIEVING FILES AND DELETING FILES | | | |
| 3. | | IMPLEMENT OF MATRIX MULTIPLICATION WITH HADOOP MAP REDUCE | | | |
| 4. | | RUN A BASIC WORD COUNT MAP REDUCE PROGRAM TO UNDERSTAND MAP REDUCE PARADIGM. | | | |
| 5. | | INSTALLATION OF HIVE ALONG WITH PRACTICE EXAMPLES. | | | |
| 6. | | INSTALLATION OF HBASE, INSTALLING THRIFT ALONG WITH PRACTICE EXAMPLES | | | |
| 7. | | PRACTICE IMPORTING AND EXPORTING DATA FROM VARIOUS DATABASES. | | | |
| 8. | | JAVA APPLICATION TO FIND THE MAXIMUM TEMPERATURE USING SPARK | | | |

| EX.NO: 4 | RUN A BASIC WORD COUNT MAP REDUCE PROGRAM TO UNDERSTAND MAP REDUCE PARADIGM |
|---|---|
| DATE: | |

## Aim:

To implement a Hadoop MapReduce program in Java that counts the number of occurrences of each word in a given input text file.

## Prerequisites:

1. **Java JDK**
    - Install JDK from Oracle or OpenJDK.
    - Set environment variable JAVA_HOME.
2. **Hadoop**
    - Install Hadoop.
    - Start HDFS with:
    - start-dfs.sh

## Algorithm:

- Start the program and initialize Hadoop configuration.
- Set the input and output paths for the job.
- Define a Mapper to read input lines, split them into words, and emit each word as a key with value 1.
- Define a Reducer to sum the counts for each word and emit the final count.
- Configure the job (set mapper, reducer, input/output formats and paths) and submit it to the cluster.
- Wait for job completion and terminate.

## Program:

The program is presented in small, numbered program-steps to match the lab format.

### Imports

Purpose: bring required Hadoop and Java classes into scope

```
package com.mapreduce.wc;

import java.io.IOException;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import org.apache.hadoop.util.GenericOptionsParser;
```

### Driver

Purpose: job configuration and submission

```
public class WordCount {
   public static void main(String[] args) throws Exception {
      Configuration conf = new Configuration();

      String[] files = new GenericOptionsParser(conf, args).getRemainingArgs();

      Path input = new Path(files[0]);
      Path output = new Path(files[1]);

      Job job = Job.getInstance(conf, "wordcount");
      job.setJarByClass(WordCount.class);

      job.setMapperClass(MapForWordCount.class);
      job.setReducerClass(ReduceForWordCount.class);
```

```
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);

        FileInputFormat.addInputPath(job, input);
        FileOutputFormat.setOutputPath(job, output);

        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
```

**Mapper**

Purpose: transform raw input lines into intermediate key-value pairs

```
    // Mapper Class
    public static class MapForWordCount extends Mapper<LongWritable, Text, Text,
IntWritable> {
        private final static IntWritable one = new IntWritable(1);
        private Text word = new Text();

        @Override
        public void map(LongWritable key, Text value, Context context) throws
IOException, InterruptedException {
            String line = value.toString();
            String[] words = line.split("\\s+");

            for (String w : words) {
                if (!w.trim().isEmpty()) {
                    word.set(w.toUpperCase().trim());
                    context.write(word, one);
                }
            }
        }
    }
```

**Reducer**

Purpose: aggregate intermediate counts into final results

```
    // Reducer Class
    public static class ReduceForWordCount extends Reducer<Text, IntWritable, Text,
IntWritable> {
        @Override
        public void reduce(Text key, Iterable<IntWritable> values, Context context)
                throws IOException, InterruptedException {
            int sum = 0;
            for (IntWritable val : values) {
```

```
            sum += val.get();
        }
        context.write(key, new IntWritable(sum));
    }
  }
}
```

**Output:**

Input file (input.txt):

```
>>hello world
>>hello hadoop
>>world of mapreduce
```

Execution command:

```
>>>hadoop jar WordCount.jar com.mapreduce.wc.WordCount input output
```

Expected output file (part-r-00000):

```
HADOOP    1
HELLO     2
MAPREDUCE 1
OF        1
WORLD     2
```

**Result:**

Thus the implementation of a Hadoop MapReduce program in Java to count the number of occurrences of each word in a given input text file was executed and verified successfully.

| EXP NO: 5 | INSTALLATION OF HIVE ALONG WITH PRACTICE EXAMPLES |
| --- | --- |
| DATE: | |

**Aim:**

To Install Hive tool with examples.

**Procedure:**

1) **Installing Hadoop**

To install Apache Hive, you must have a Hadoop Cluster installed and running: You can refer to our previously published step-by-step guide to install Hadoop 3.2.1 on Windows 10.

2) **Apache Derby**

In addition, Apache Hive requires a relational database to create its Metastore (where all metadata will be stored). In this guide, we will use the Apache Derby database 4.

Since we have Java 8 installed, we must install Apache Derby 10.14.2.0 version (check downloads page) which can be downloaded from the following link.

Once downloaded, we must extract twice *(using 7zip: the first time we extract the .tar.gz file, the second time we extract the .tar file)* the content of the db-derby-10.14.2.0- bin.tar.gz archive into the desired installation directory. Since in the previous guide we have installed Hadoop within ―E:\hadoop-env\hadoop-3.2.1\‖ directory, we will extract Derby into
―E:\hadoop-env\db-derby-10.14.2.0\‖ directory.

3) **Cygwin**

Since there are some Hive 3.1.2 tools that aren‘t compatible with Windows (such as schematool). We will need the Cygwin tool to run some Linux commands.

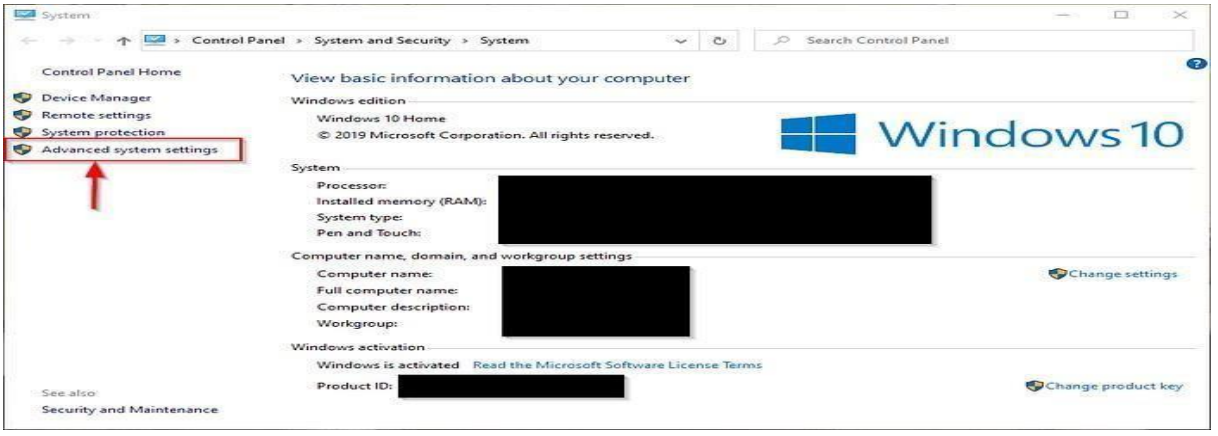
### 4) Downloading Apache Hive binaries

In order to download Apache Hive binaries, you should go to the following website: https://downloads.apache.org/hive/hive-3.1.2/. Then, download the apache-hive3.1.2.- bin.tar.gz file.


When the file download is complete, we should extract twice *(as mentioned above)* the apache-hive.3.1.2-bin.tar.gz archive into ―E:\hadoop-env\apache-hive-3.1.2‖ directory (Since we decided to use E:\hadoop-env\‖ as the installation directory for all technologies used in the previous guide.

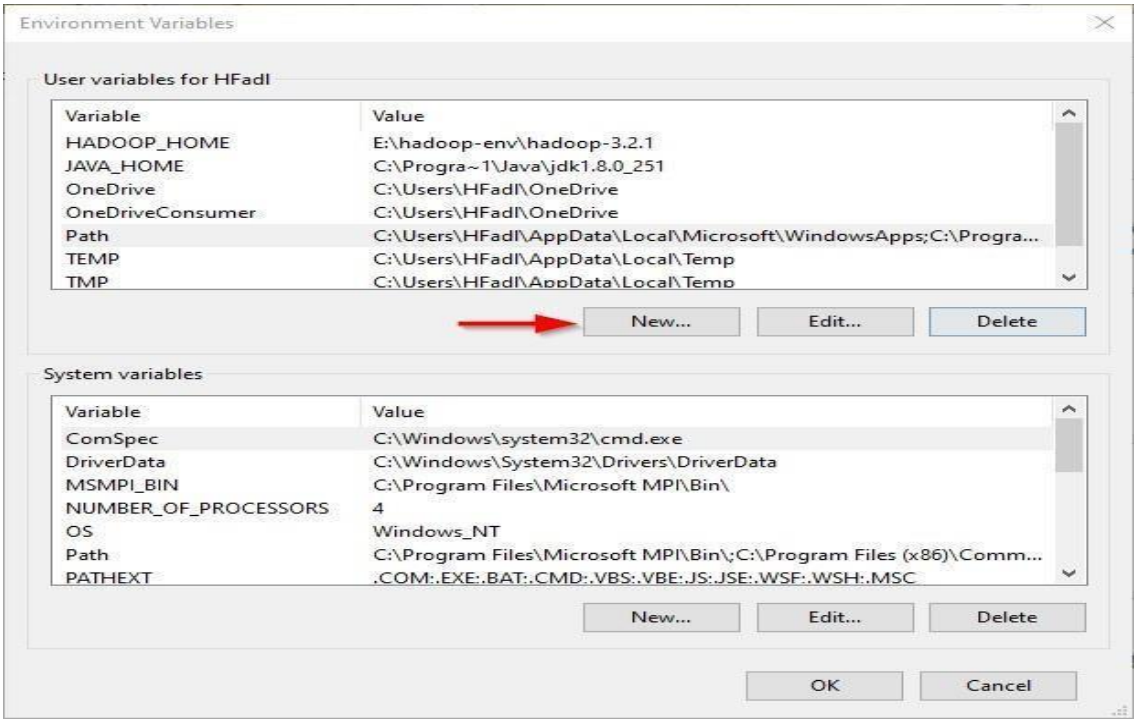
### 5) Setting environment variables

After extracting Derby and Hive archives, we should go to Control Panel > System and Security > System. Then Click on ―Advanced system settings‖.




In the advanced system settings dialog, click on ―Environment variables‖

button. we should add the following user variables:



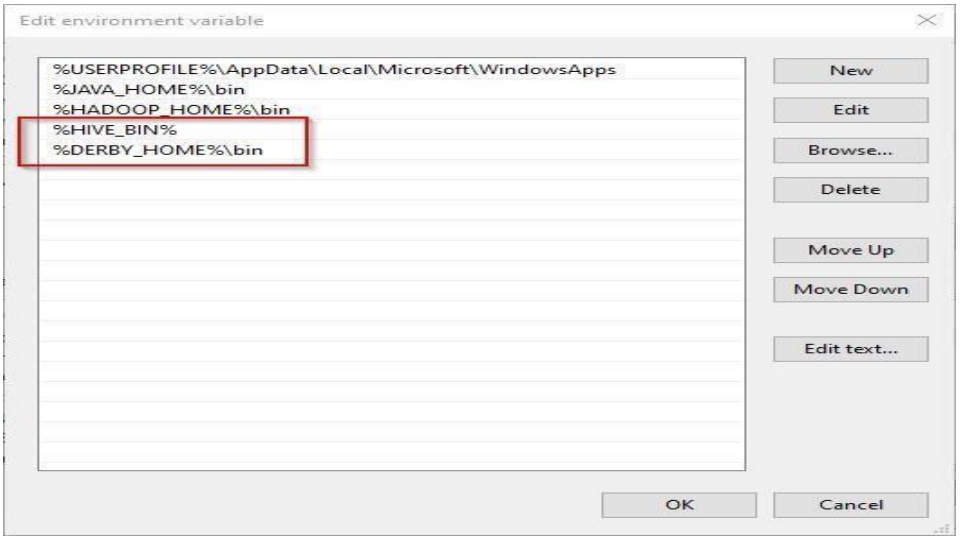

- HIVE_HOME: ―E:\hadoop-env\apache-hive-3.1.2\‖
- DERBY_HOME: ―E:\hadoop-env\db-derby-10.14.2.0\‖
- HIVE_LIB: ―%HIVE_HOME%\lib‖
- HIVE_BIN: ―%HIVE_HOME%\bin‖
- HADOOP_USER_CLASSPATH_FIRST:

Besides, we should add the following system variable:

- HADOOP_USER_CLASSPATH_FIRST

Now, we should edit the Path user variable to add the following paths:

- %HIVE_BIN%
- %DERBY_HOME%\bin



### 6) Configuring Hive

Now, we should go to the Derby libraries directory (E:\hadoop-env\db-derby-10.14.2.0\lib) and copy all *.jar files.
Then, we should paste them within the Hive libraries directory

(E:\hadoopenv\apache- hive- 3.1.2\lib).

### 7) Configuring hive-site.xml

Now, we should go to the Apache Hive configuration directory (E:\hadoop-env\apache-hive)

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration><property>  <name>javax.jdo.option.ConnectionURL</name>
<value>jdbc:derby://localhost:1527/metastore_db;create=true</value>
<description>JDBC connect string for a JDBC metastore</description>
</property><property>
<name>javax.jdo.option.ConnectionDriverName</name>
<value>org.apache.derby.jdbc.ClientDriver</value>
<description>Driver class name for a JDBC
metastore</description> </property> <property>
<name>hive.server2.enable.doAs</name>
<description>Enable user impersonation for HiveServer2</description>

<value>true</value>
</property>
<property>
<name>hive.server2.authentication</name>
<value>NONE</value>
<description> Client authentication types. NONE: no authentication check LDAP:
LDAP/AD based authentication KERBEROS: Kerberos/GSSAPI authentication
CUSTOM: Custom authentication provider (Use with property
hive.server2.custom.authentication.class)
</description>
</property>
<property>
<name>datanucleus.autoCreateTables</nam>e
<value>True</value>
</property>
</configuration>
```

## STARTING SERVICES:

### 1) Hadoop Services

To start Apache Hive, open the command prompt utility as administrator. Then, start the Hadoop services using start-dfs and start-yarn commands (as illustrated in the Hadoop installation guide).

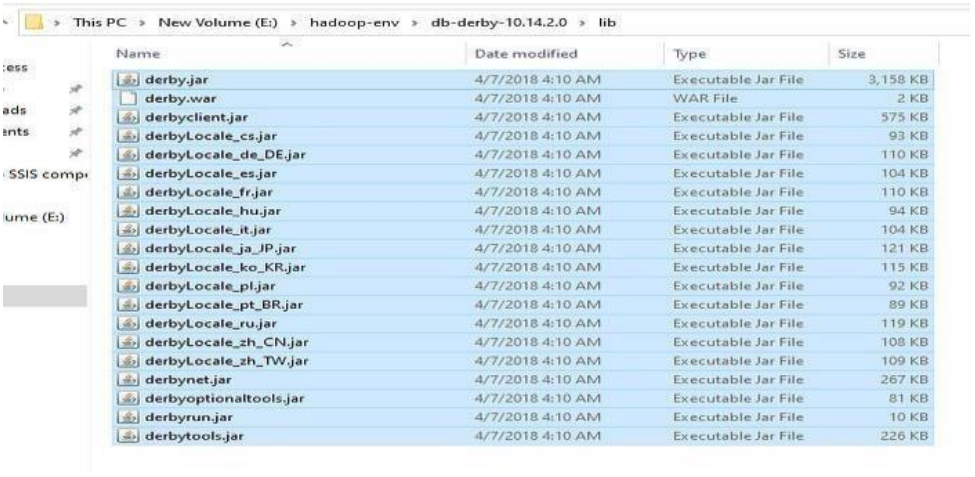
### 2) Derby Network Server

Then, we should start the Derby network server on the localhost using the following command:

```
E:\hadoop-env\db-derby-10.14.2.0\bin\StartNetworkServer-h  0.0.0.0
```

### 3) Starting Apache Hive

Now, let try to open a command prompt tool and go to the Hive binaries directory (E:\hadoopenv\apache- hive-3.1.2\bin) and execute the following command:




We will receive the following error:

```
'hive' is not recognized as an internal or external command, operable program or batch file.
```

This error is thrown since the Hive 3.x version is not built for Windows (only in some Hive 2.x versions). To get things working, we should download the necessary *.cmd files from the following link: https://svn.apache.org/repos/asf/hive/trunk/bin/. Note that, you should keep the folder hierarchy (bin\ext\util).

You can download all *.cmd files from the following GitHub.

```
com.google.common.base.Preconditions.checkArgument(ZLjava/lang/String;Ljava/lang/O
bj ect;)
V
at org.apache.hadoop.conf.Configuration.set(Configuration.java:1357) at
org.apache.hadoop.conf.Configuration.set(Configuration.java:1338) at
org.apache.hadoop.mapred.JobConf.setJar(JobConf.java:518) at
org.apache.hadoop.mapred.JobConf.setJarByClass(JobConf.java:536) at
org.apache.hadoop.mapred.JobConf.<init>(JobConf.java:430) at
org.apache.hadoop.hive.conf.HiveConf.initialize(HiveConf.java:5141) at
org.apache.hadoop.hive.conf.HiveConf.<init>(HiveConf.java:5104) at
org.apache.hive.beeline.HiveSchemaTool.<init>(HiveSchemaTool.java:96) at
org.apache.hive.beeline.HiveSchemaTool.main(HiveSchemaTool.java:1473) at
sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method) at
sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62
) at
sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.jav
a:4 3) at java.lang.reflect.Method.invoke(Method.java:498) at
org.apache.hadoop.util.RunJar.run(RunJar.java:318) at
org.apache.hadoop.util.RunJar.main(RunJar.java:232)
```

E:\hadoop-env\hadoop-3.2.1\share\hadoop\hdfs\lib‖.

*Note: This file is also uploaded to the GitHub repository mentioned above.*
Now, if we run hive command again, then Apache Hive will start successfully.

### 4) Initializing Hive

After ensuring that the Apache Hive started successfully. We may not be able to run any HiveQL command. This is because the Metastore is not initialized yet. Besides HiveServer2 service must be running.
To initialize Metastore, we need to use schematool utility which is not compatible with windows. To solve this problem, we will use Cygwin utility which allows executing Linux command from windows.

### 5) Creating symbolic links

First, we need to create the following directories:

- E:\cygdrive
- C:\cygdrive

Now, open the command prompt as administrator and execute the following commands:

```
mklink /J E:\cygdrive\e\
E:\ mklink /J
```

These symbolic links are needed to work with Cygwin utility properly since Java maycause some problems.

### 6) Initializing Hive Metastore

Open Cygwin utility and execute the following commands to define the environment variables:

```
export HADOOP_HOME='/cygdrive/e/hadoop-env/hadoop-3.2.1' export PATH=$PATH:$HADOOP_HOME/bin
export HIVE_HOME='/cygdrive/e/hadoop-env/apache-hive-3.1.2'
export PATH=$PATH:$HIVE_HOME/bin export HADOOP_CLASSPATH=$HADOOP_CLASSPATH:$HIVE_HOME/lib/*.jar
```

We can add these lines to the ―~/.bashrc‖ file then you don't need to write them each time you open Cygwin.

Now, we should use the schematool utility to initialize the Metastore:

```
$HIVE_HOME/bin/schematool -dbType derby -initSchema
```

### 7) Starting HiveServer2 service

Now, open a command prompt and run the following command:

```
hive --service hiveserver2 start
```

We should leave this command prompt open, and open a new one where we should start Apache Hive using the following command:

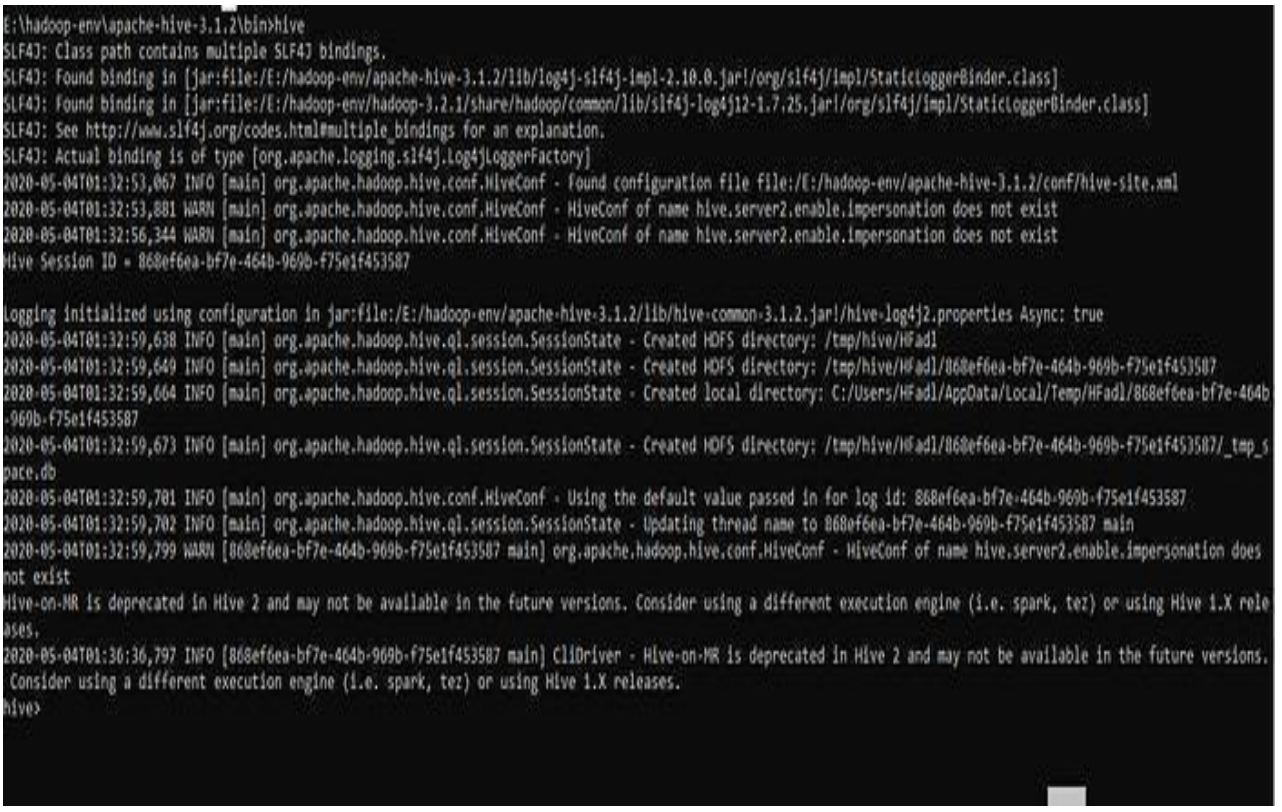
```
hive
```

### 8) Starting WebHCat Service (Optional)

In the project we are working on, we need to execute HiveQL statement from SQL Server Integration Services which can access Hive from the WebHCat server. To start the WebHCat server, we should open the Cygwin utility and execute the following command:

```
$HIVE_HOME/hcatalog/sbin/webhcat_server.sh start
```

**Output:**




**Result:**

Thus the installation of hive was successfully implemented.

| EX.NO: 6 | INSTALLATION OF HBASE, INSTALLING THRIFT ALONG WITH PRACTICE EXAMPLES |
|---|---|
| DATE: | |

## Aim:

To install and configure **Apache HBase** along with **Thrift support**, ensuring compatibility with **Hadoop 3.3.0** on Windows.

## Prerequisites:

1. **Install Java JDK**
    - Download from Oracle JDK.
    - Ensure environment variable JAVA_HOME is set properly.
2. **Install Hadoop 3.3.0 (Windows)**
    - Ensure Hadoop Already installed
    - Ensure Hadoop services are running (namenode, datanode, etc.).
3. **Download HBase (Compatible Version)**
    - Go to HBase Downloads.
    - Download a version **compatible with Hadoop 3.3.0** (recommended: **HBase 2.4.x or later**).

## Procedure:

### Step 1 – Extract Files

- Extract the downloaded **HBase archive** into a suitable directory (e.g., C:\hbase).
- Inside this folder, create two subfolders:
    - hbase
    - zookeeper

### Step 2 – Edit hbase.cmd

- Navigate to C:\hbase\bin.
- Open **hbase.cmd** in a text editor.
- Search for the line:
- %HEAP_SETTINGS%

**Step 3 – Configure hbase-env.cmd**

- Go to C:\hbase\conf.
- Open **hbase-env.cmd** in a text editor.
- Add the following lines **after the comment section**:

```
set JAVA_HOME=%JAVA_HOME%
set HBASE_CLASSPATH=%HBASE_HOME%\lib\client-facing-thirdparty\*
set HBASE_HEAPSIZE=8000
set HBASE_OPTS="-XX:+UseConcMarkSweepGC" "-
Djava.net.preferIPv4Stack=true"
set SERVER_GC_OPTS="-verbose:gc" "-XX:+PrintGCDetails" "-
XX:+PrintGCDateStamps" %HBASE_GC_OPTS%
set HBASE_USE_GC_LOGFILE=true

set HBASE_JMX_BASE="-Dcom.sun.management.jmxremote.ssl=false" "-
Dcom.sun.management.jmxremote.authenticate=false"

set HBASE_MASTER_OPTS=%HBASE_JMX_BASE% "-
Dcom.sun.management.jmxremote.port=10101"
set HBASE_REGIONSERVER_OPTS=%HBASE_JMX_BASE% "-
Dcom.sun.management.jmxremote.port=10102"
set HBASE_THRIFT_OPTS=%HBASE_JMX_BASE% "-
Dcom.sun.management.jmxremote.port=10103"
set HBASE_ZOOKEEPER_OPTS=%HBASE_JMX_BASE% "-
Dcom.sun.management.jmxremote.port=10104"

set HBASE_REGIONSERVERS=%HBASE_HOME%\conf\regionservers
set HBASE_LOG_DIR=%HBASE_HOME%\logs
set HBASE_IDENT_STRING=%USERNAME%
set HBASE_MANAGES_ZK=true
```

**Step 4 – Configure hbase-site.xml**

- Open **hbase-site.xml** in C:\hbase\conf.
- Add the following inside the <configuration> tag:

```
<property>
  <name>hbase.rootdir</name>
  <value>hdfs://localhost:9000/hbase</value>
</property>

<property>
```

```
  <name>hbase.zookeeper.property.dataDir</name>
  <value>C:/hbase/zookeeper</value>
</property>

<property>
  <name>hbase.zookeeper.quorum</name>
  <value>localhost</value>
</property>
```

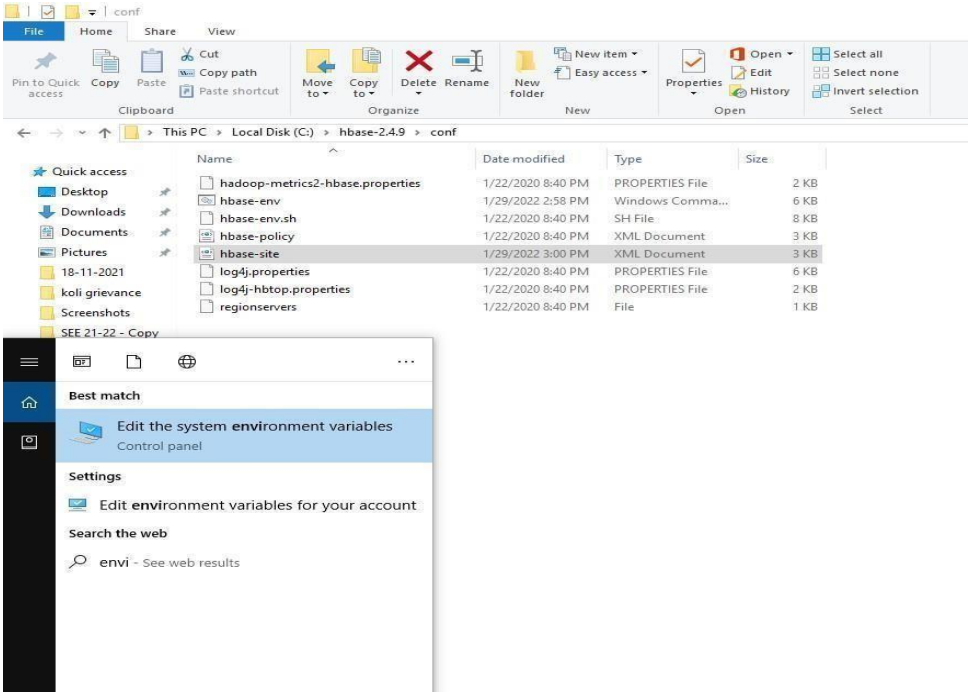
**Step 5 – Set Environment Variables**

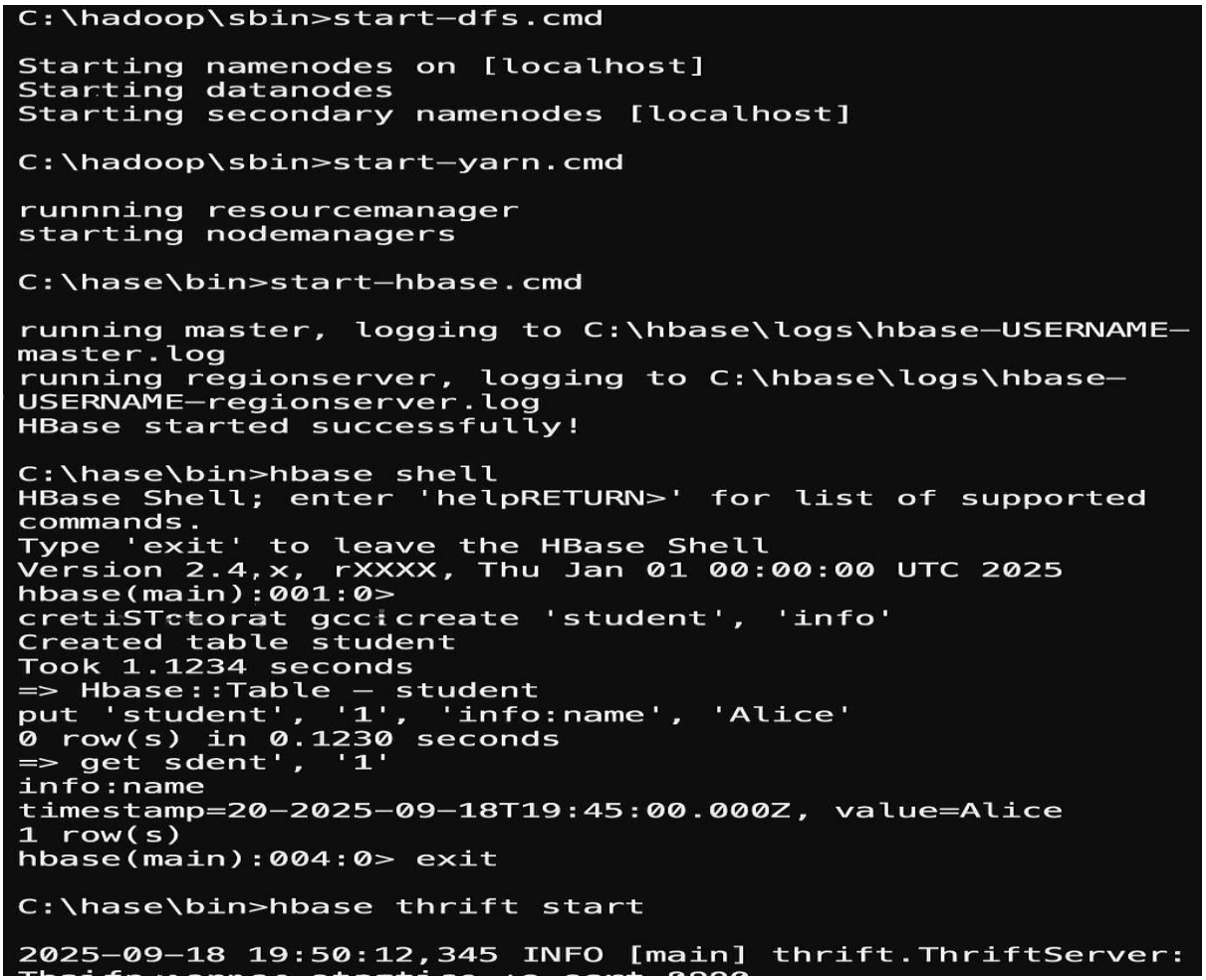
1. Open **System Properties → Environment Variables**.
2. Add a new variable:
    - **Variable name**: HBASE_HOME
    - **Variable value**: C:\hbase
3. Edit the **Path** variable → Add:
4. %HBASE_HOME%\bin

**Step 6 – Start HBase and Thrift**

1. Open Command Prompt → Start Hadoop services (start-dfs.cmd, start-yarn.cmd).
2. Start HBase:
3. start-hbase.cmd
4. Verify HBase shell:
5. hbase shell
    - Try creating a table:
    - create 'student', 'info'
    - put 'student', '1', 'info:name', 'Alice'
    - get 'student', '1'
6. Start Thrift service:
7. hbase thrift start
    - This enables HBase clients (Python, Java, etc.) to connect via **Thrift API**.

## Output:


conf
File
Home
Share
View
Pin to Quick access
Copy
Paste
Cut
Copy path
Paste shortcut
Clipboard
Move to
Copy to
Delete
Rename
Organize
New folder
New item
Easy access
New
Properties
Open
Edit
History
Open
Select all
Select none
Invert selection
Select
This PC
Local Disk (C:)
hbase-2.4.9
conf
Quick access
Desktop
Downloads
Documents
Pictures
18-11-2021
koli grievance
Screenshots
SEE 21-22 - Copy
Name
Date modified
Type
Size
hadoop-metrics2-hbase.properties 1/22/2020 8:40 PM PROPERTIES File 2 KB
hbase-env 1/29/2022 2:58 PM Windows Comma... 6 KB
hbase-env.sh 1/22/2020 8:40 PM SH File 8 KB
hbase-policy 1/22/2020 8:40 PM XML Document 3 KB
hbase-site 1/29/2022 3:00 PM XML Document 3 KB
log4j.properties 1/22/2020 8:40 PM PROPERTIES File 6 KB
log4j-hbtop.properties 1/22/2020 8:40 PM PROPERTIES File 2 KB
regionservers 1/22/2020 8:40 PM File 1 KB
Best match
Edit the system environment variables
Control panel
Settings
Edit environment variables for your account
Search the web
envi - See web results




**Result:**

Thus the Apache HBase along with Thrift support, ensuring compatibility with Hadoop 3.3.0 on Windows was installed successfully.

| EX.NO: 7 | PRACTICE IMPORTING AND EXPORTING DATA FROM VARIOUS DATABASES |
|---|---|
| DATE: | |

## Aim:

To install and configure Cassandra, Hadoop (HDFS), Pig, Hive, and HBase and perform data importing and exporting operations using their respective tools and client libraries.

## Prerequisites:

### 1. Java JDK

- Install JDK from Oracle or OpenJDK.
- Set environment variable JAVA_HOME.

### 2. Hadoop

- Install Hadoop.
- Start HDFS with:
- start-dfs.sh

### 3. Cassandra

- Install Apache Cassandra from Cassandra Downloads.
- Start Cassandra:
- cassandra -f
- Use cqlsh to interact.

### 4. Pig

- Download Apache Pig.
- Start Pig shell:
  >>>Pig

### 5. Hive

- Download Apache Hive.
- Start Hive shell:
  >>>Hive

6. **HBase**
    - Install Apache HBase (compatible with Hadoop 3.3.0).
    - Start HBase:
        >>>start-hbase.sh

## Procedure:

### 1. Cassandra

- Start Cassandra with:
- cassandra -f
- Use Python Cassandra driver for practice:

```
from cassandra.cluster import Cluster

# Connect to Cassandra cluster
cluster = Cluster(['localhost'])
session = cluster.connect()

# Create keyspace and table
session.execute("CREATE KEYSPACE IF NOT EXISTS test WITH REPLICATION
= {'class': 'SimpleStrategy', 'replication_factor': 1}")
session.execute("CREATE TABLE IF NOT EXISTS test.users (id UUID PRIMARY
KEY, name TEXT, age INT)")

# Insert data
session.execute("INSERT INTO test.users (id, name, age) VALUES (uuid(), 'John
Doe', 30)")
session.execute("INSERT INTO test.users (id, name, age) VALUES (uuid(), 'Jane
Smith', 25)")

# Query data
rows = session.execute("SELECT * FROM test.users")
for row in rows:
    print(row.id, row.name, row.age)

# Close
session.shutdown()
cluster.shutdown()
```

## 2. Hadoop (HDFS)

- Upload and download files using HDFS Python client:

```
from hdfs import InsecureClient

# Connect to HDFS
hdfs_client = InsecureClient('http://localhost:50070', user='your_username')

# Upload file
hdfs_client.upload('/user/your_username/data',  'local_file.txt')

# Download file
hdfs_client.download('/user/your_username/data/local_file.txt',  'downloaded_file.txt')

# List files
files = hdfs_client.list('/user/your_username/data')
print(files)
```

## 3. Pig

- Example Pig script to calculate revenue:

```
sales = LOAD 'sales.csv' USING PigStorage(',') AS (product:chararray, quantity:int,
price:double);
revenue = FOREACH sales GENERATE product, quantity * price AS total_revenue;
STORE revenue INTO 'revenue.csv' USING PigStorage(',');
```

## 4. Hive

- Connect to Hive using Python:

```
from pyhive import hive

# Connect to Hive
conn = hive.connect(host='localhost', port=10000, auth='NONE', database='default')

with conn.cursor() as cursor:
    # Create table
```

```
    cursor.execute("CREATE TABLE IF NOT EXISTS test_table (id INT, name
STRING, age INT)")
    # Insert data
    cursor.execute("INSERT INTO test_table VALUES (1, 'John Doe', 30)")
    cursor.execute("INSERT INTO test_table VALUES (2, 'Jane Smith', 25)")
    # Query data
    cursor.execute("SELECT * FROM test_table")
    results = cursor.fetchall()
    for row in results:
        print(row)

conn.close()
```

### 5. HBase

- Start HBase shell:
- hbase shell
- Create table and load CSV data:

```
create 'products', 'details'
put 'products', '101', 'details:product_name', 'Widget A'
put 'products', '101', 'details:category', 'Electronics'
put 'products', '101', 'details:price', '25.99'

put 'products', '102', 'details:product_name', 'Widget B'
put 'products', '102', 'details:category', 'Electronics'
put 'products', '102', 'details:price', '19.99'

put 'products', '103', 'details:product_name', 'Widget C'
put 'products', '103', 'details:category', 'Home'
put 'products', '103', 'details:price', '12.99'

scan 'products'
```

**Output:**

1.Cassandra

| id | name | age |
|---|---|---|
| 123e4567-e89b-12d3-a456-426614174000 | John Doe | 30 |
| 223e4567-e89b-12d3-a456-426614174001 | Jane Smith | 25 |

2.HDFS

- Upload → File appears in /user/your_username/data.
- Download → File contents same as local file.

3. Pig (revenue.csv)

Apple,6.0
Banana,2.4
Orange,1.8

4. Hive

(1, 'John Doe', 30)
(2, 'Jane Smith', 25)

5. HBase

| ROW | COLUMN+CELL | |
|---|---|---|
| 101 | details:product_name | Widget A |
| | details:category | Electronics |
| | details:price | 25.99 |
| 102 | details:product_name | Widget B |
| | details:category | Electronics |
| | details:price | 19.99 |
| 103 | details:product_name | Widget C |
| | details:category | Home |
| | details:price | 12.99 |

**Result:**

Thus the installation and configuration of Cassandra, Hadoop (HDFS), Pig, Hive, and HBase and to perform data importing and exporting operations using their respective tools and client libraries was executed and verified successfully.

| EX.NO: 8 | JAVA APPLICATION TO FIND THE MAXIMUM TEMPERATURE USING SPARK |
|---|---|
| DATE: | |

## Aim:

To develop a Java Application to find the maximum temperature using Spark.

## Algorithm:

1) **Initialize Spark Configuration**
   -Create a `SparkConf` object with:
   -Master set to `local[2]` (2 threads for local execution)
   -Application name set to `"testfilter"`

2) **Create Spark Context**
   -Instantiate `SparkContext` using the configuration object.

3) **Set Hadoop Directory (Windows-specific**
   -Set system property `hadoop.home.dir` to `"c://winutil//"` to avoid Hadoop-related errors on Windows.

4) **Load Input File**
   -Read the text file located at `D://sparkprog//temp//stats.txt` into an RDD named `input`.

5) **Split Each Line**
   -Transform each line in `input` by splitting it into an array of strings (tokens), resulting in RDD `line`.

6) **Extract City and Temperature**
   -From each tokenized line, extract:
      -City name from index `3`
      -Temperature from index `4`
   -Combine them into a single string and store in RDD `city`.

7) **Split Combined City-Temperature Strings**
   -Split each string in `city` again to separate city and temperature, resulting in RDD `rdd3`.

8) **Create Key-Value Pairs**
   -Map each element in `rdd3` to a tuple:
      -Key: city name
      -Value: temperature

-Store in RDD `maintemp`.

9) **Group Temperatures by City**
-Group all temperature values by city using `groupByKey()`, resulting in RDD `grp`.

10) **Find Maximum Temperature per City**
-For each city group:
Convert the iterable of temperatures to a list
Find the maximum temperature
-Store the result in RDD `main`.

11) **Print Results:**
-Iterate over each element in `main`
-Print the city and its maximum temperature

**Program:**

```
import org.apache.spark.*;

object testfilter extends App {

	val conf-new SparkConf().setMaster("local[2]").setAppName("testfilter");

	val sc = new SparkContext(conf); System.setProperty("hadoop.home.dir",
	"c://winutil//");

	val input sc.textFile("file:///D://sparkprog//temp//stats.txt");
	val line input.map(x=>x.split();
	val city line.map(x=>(x(3)+"\x(4)));
	val rdd3-city.map(x=>x.split()
	val maintemp-rdd3.map(x=((x(0)x(1))));
	val grp maintemp.groupByKey();
	val main grp.map/case (x.iter) (xiter.toList.max));


	for (i<=main)
	{
		print(i) print("\n")
	}
}
```

**Output:**

```
>>> (Jammu and Kashmir,20)
    (Madhya Pradesh,32)
    (Bihar,31)
```

**Result:**

Thus the Java Application to find the maximum temperature using Spark was Executed and verified successfully.