U18ISI5202T - BIG DATA TECHNOLOGIES LAB REPORT

Name : Monika M
Roll no : 20BIS025
Course : U18ISI5202T – Big Data Technologies
Date : 28.12.2022

INDEX

Experiment 1: *Perform setting up and Installing Hadoop in its three operating modes: Standalone, Pseudo distributed, Fully distributed*

a) *OBJECTIVE OF THE EXERCISE/EXPERIMENT*

Installation of Hadoop Framework, it's components and study the Hadoop Ecosystem.
Perform setting up and Installing Hadoop in its three operating modes: Standalone, Pseudo distributed, fully distributed.

b) *Facilities/material required to do the exercise/experiment:*

Sl.No.	*Facilities/material required*	*Quantity*
1.	*Hadoop*	*As per strength*

THEORY:

Hadoop is an open-source framework that allows to store and process big data in a distributed environment across clusters of computers using simple programming models.

Hadoop software can be installed in three modes of operation:

Stand Alone Mode: Hadoop is a distributed software and is designed to run on a commodity of machines. However, we can install it on a single node in stand-alone mode. In this mode, Hadoop software runs as a single monolithic java process. This mode is extremely useful for debugging purpose. You can first

test run your Map-Reduce application in this mode on small data, before actually executing it on cluster with big data.
Pseudo Distributed Mode: In this mode also, Hadoop software is installed on a Single Node. Various daemons of Hadoop will run on the same machine as separate java processes. Hence all the daemons namely Name Node, DataNode, SecondaryNameNode, Job Tracker, Task Tracker run on single machine.
Fully Distributed Mode: In Fully Distributed Mode, the daemons Name Node, Job Tracker, SecondaryNameNode (Optional and can be run on a separate node) run on the Master Node. The daemons DataNode and Task Tracker run on the Slave Node.

c)*Procedure for doing the exercise/experiment:*

Installation Steps: – Hadoop Installation: Ubuntu Operating System in stand-alone mode

Step 1: Install OpenJDK on Ubuntu

The Hadoop framework is written in Java, and its services require a compatible Java Runtime Environment (JRE) and Java Development Kit (JDK). Use the following command to update your system before

initiating a new installation:

COMMANDS
1.Sudo apt install *l 2.Sudo apt update*

TERMINAL:

```
[sudo] password for hadoop:
Hit:1 https://download.docker.com/linux/ubuntu jammy InRelease
Get:2 http://security.ubuntu.com/ubuntu jammy-security InRelease [110 kB]
Hit:3 http://in.archive.ubuntu.com/ubuntu jammy InRelease
Get:4 http://in.archive.ubuntu.com/ubuntu jammy-updates InRelease [114 kB]
Get:5 http://in.archive.ubuntu.com/ubuntu jammy-backports InRelease [99.8 kB]
Get:6 http://security.ubuntu.com/ubuntu jammy-security/main amd64 DEP-11 Metada
ta [20.1 kB]
Get:7 http://security.ubuntu.com/ubuntu jammy-security/universe amd64 DEP-11 Me
tadata [13.3 kB]
Get:8 http://in.archive.ubuntu.com/ubuntu jammy-updates/main amd64 Packages [73
0 kB]
```

STEP 2:

In this step, we will install latest version of JDK on the machine.

Apache Hadoop 3.x fully supports Java 8. The OpenJDK 8 package in Ubuntu contains both the runtime environment and development kit.

These commands install the specified version of java on your VM. The "sudo" command enables installation as an administrator. When you use the sudo command, the system asks for your password.
During the installation, the installer identifies the amount of disc space that is required and asks for your permission to continue. Input "y" to continue

Type the following command in your terminal to install OpenJDK 8:

COMMAND
3. sudo apt install openjdk-8-jdk -y

TERMINAL:

```
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following packages were automatically installed and are no longer required:
  chromium-codecs-ffmpeg-extra gstreamer1.0-vaapi i965-va-driver
  intel-media-va-driver libaacs0 libaom3 libass9 libavcodec58 libavformat58
  libavutil56 libbdplus0 libblas3 libbluray2 libbs2b0 libchromaprint1
  libcodec2-1.0 libdav1d5 libflite1 libgme0 libgsm1
  libgstreamer-plugins-bad1.0-0 libigdgmm12 liblilv-0-0 libmfx1 libmysofa1
  libnorm1 libopenmpt0 libpgm-5.3-0 libpostproc55 librabbitmq4 librubberband2
  libserd-0-0 libshine3 libsnappy1v5 libsord-0-0 libsratom-0-0
  libsrt1.4-gnutls libssh-gcrypt-4 libswresample3 libswscale5 libudfread0
  libva-drm2 libva-wayland2 libva-x11-2 libva2 libvdpau1 libvidstab1.1
  libx265-199 libxvidcore4 libzimg2 libzmq5 libzvbi-common libzvbi0
  mesa-va-drivers mesa-vdpau-drivers pocketsphinx-en-us systemd-hwe-hwdb
  va-driver-all vdpau-driver-all
```

STEP 3: Verify Java Installation

To check the installation of Java, you can check the version of the installed Java with the following command.

COMMAND
4. java -version; javac -version

TERMINAL:

```
openjdk version "11.0.17" 2022-10-18
OpenJDK Runtime Environment (build 11.0.17+8-post-Ubuntu-1ubuntu222.04)
OpenJDK 64-Bit Server VM (build 11.0.17+8-post-Ubuntu-1ubuntu222.04, mixed mode
, sharing)
```

It returns "The program java can be found in the following packages". This output verifies that OpenJDK has been successfully installed.

Set Up a Non-Root User for Hadoop Environment

STEP 4:Setup password less ssh

Install Open SSH Server and Open SSH Client

The next step is installation of ssh server which is needed for management of the communication with localhost. The following commands download and install the related files for the ssh server.
We will now setup the password less ssh client with the following command.

COMMAND
5. sudo apt install openssh-server openssh-client -y

TERMINAL:

```
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
openssh-client is already the newest version (1:8.9p1-3).
openssh-server is already the newest version (1:8.9p1-3).
openssh-server set to manually installed.
The following packages were automatically installed and are no longer required:
  chromium-codecs-ffmpeg-extra gstreamer1.0-vaapi i965-va-driver
  intel-media-va-driver libaacs0 libaom3 libass9 libavcodec58 libavformat58
  libavutil56 libbdplus0 libblas3 libbluray2 libbs2b0 libchromaprint1
  libcodec2-1.0 libdav1d5 libflashrom1 libflite1 libftdi1-2 libgme0 libgsm1
  libgstreamer-plugins-bad1.0-0 libigdgmm12 liblilv-0-0 libmfx1 libmysofa1
  libnorm1 libopenmpt0 libpgm-5.3-0 libpostproc55 librabbitmq4 librubberband2
  libserd-0-0 libshine3 libsnappy1v5 libsord-0-0 libsratom-0-0
  libsrt1.4-gnutls libssh-gcrypt-4 libswresample3 libswscale5 libudfread0
  libva-drm2 libva-wayland2 libva-x11-2 libva2 libvdpau1 libvidstab1.1
  libx265-199 libxvidcore4 libzimg2 libzmq5 libzvbi-common libzvbi0
  mesa-va-drivers mesa-vdpau-drivers pocketsphinx-en-us va-driver-all
  vdpau-driver-all
```

STEP 5: Create Hadoop User:

Let's create a separate user for Hadoop so we have isolation between the Hadoop file system and the Unix file system.

This step is optional, but recommended because it gives you flexibility to have a separate account for Hadoop installation by separating this installation from other software installation.

You can use a different group name and user name based on your preferences. Once the user is identified, the system asks for her password and other information including name, last name, etc. Since this is a trial installation, you can enter a simple password and press enter to skip these question

Run the following command to create a new user with name hadoop:

COMMAND
6. sudo adduser hadoop

TERMINAL:

```
Adding user `hadoop' ...
Adding new group `hadoop' (1001) ...
Adding new user `hadoop' (1001) with group `hadoop' ...
Creating home directory `/home/hadoop' ...
Copying files from `/etc/skel' ...
New password:
Retype new password:
passwd: password updated successfully
Changing the user information for hadoop
Enter the new value, or press ENTER for the default
        Full Name []:
        Room Number []:
        Work Phone []:
        Home Phone []:
        Other []:
Is the information correct? [Y/n] y
```

STEP 6: GET A PERMISSION:

The new user needs to have sudo privileges to be able to run the programs as administrator. Once the sudo privilege is set, we can continue with the new user logged in.Also run these commands from an admin privileged user present on the machine.

Execute the command

COMMAND
7. sudo adduser hadoop sudo

STEP 7: Switches the user from current user to the new user created.

To login with gokilanm, the system asks for its password which we identified in previous step.Switch to the newly created user and enter the corresponding password:

COMMAND
8. su - hadoop

STEP 8: Generate Public & Private Key

Pairs Enable Passwordless SSH for

Hadoop User

Once the user is added, generate SS key pair for the user.Generate an SSH key pair and define the location is to be stored in:

COMMANDS
9. ssh-keygen -t rsa -P '' -f ~/.ssh/id_rsa

You will be asked to enter the filename. Just press Enter to complete the process and Create a ssh key for passwordless ssh configuration

TERMINAL:

```
Generating public/private rsa key pair.
Enter file in which to save the key (/home/hadoop/.ssh/id_rsa):
Created directory '/home/hadoop/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/hadoop/.ssh/id_rsa
Your public key has been saved in /home/hadoop/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:dr7r/taS5pUE/71/hMbx4eNJvBV9Eph8s/E2AhunO7s hadoop@swetha
The key's randomart image is:
+---[RSA 3072]----+
|           . o   |
|           o+.=  |
|            B. *.|
|           o +++=|
|        S . ..**=|
|       . o o .+B=|
|          . o+=.B|
|           o=..+o|
|         o=E+. .+|
+----[SHA256]-----+
```

The system proceeds to generate and save the SSH key pair.

STEP 9: Enable SSH access to local machine using this key.

Use the cat command to store the public key as authorized keys in the ssh directory:

COMMAND
10. cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys

STEP 10:

Set the permissions for your user with the chmod command:

COMMANDS
11.chmod 0600 ~/.ssh/authorized_keys

STEP 11:

The new user is now able to SSH without needing to enter a password every time. Verify everything is set up correctly by using the hdoop user to SSH to localhost:

Let's verify key based login.

COMMANDS
12.ssh localhost

STEP 12:INSTALL HADOOP:

Software:

Hadoop

Version:3.2.2

Download link(s): https://www.apache.org/dyn/closer.cgi/hadoop/common/hadoop-3.2.2/hadoop- 3.2.2.tar.gz

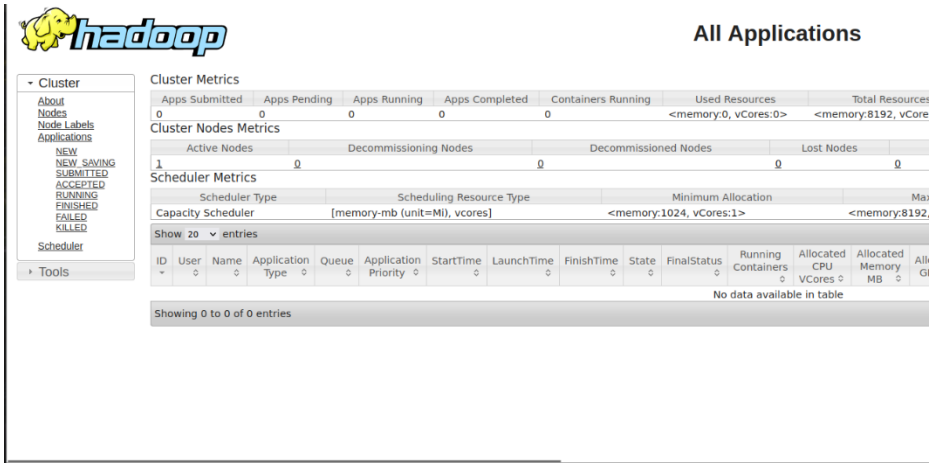
File size 210 MB

Install size Variable Requirements

Virtualbox :Ubuntu 10.04 LTS or

higher

Visit the official Apache Hadoop project page, and select the version of Hadoop you want to implement.



Visit this URL and choose one of the mirror sites. You can copy the download link and also use "wget" to download it from command prompt:

COMMANDS
13.wget https://www.apache.org/dyn/closer.cgi/hadoop/common/hadoop- 3.3.0/hadoop-3.3.0.tar.gz

```
--2022-11-04 19:56:55--  https://archive.apache.org/dist/hadoop/common/hadoop-3.
3.0/hadoop-3.3.0.tar.gz
Resolving archive.apache.org (archive.apache.org)... 138.201.131.134, 2a01:4f8:1
72:2ec5::2
Connecting to archive.apache.org (archive.apache.org)|138.201.131.134|:443... co
nnected.
HTTP request sent, awaiting response... 200 OK
Length: 500749234 (478M) [application/x-gzip]
Saving to: ‘hadoop-3.3.0.tar.gz’

hadoop-3.3.0.tar.gz 100%[===================>] 477.55M  5.82MB/s    in 1m 47s

2022-11-04 19:58:42 (4.48 MB/s) - ‘hadoop-3.3.0.tar.gz’ saved [500749234/5007492
34]
```

STEP 14: Untar the Tarball

Once downloaded, extract the downloaded file:

COMMANDS
14.tar xzf hadoop-3.3.0.tar.gz

STEP 15:Single Node Hadoop Deployment (Pseudo-Distributed Mode)

Hadoop excels when deployed in a fully distributed mode on a large cluster of networked servers. However, if you are new to Hadoop and want to explore basic commands or test applications, you can configure Hadoop on a single node.

This setup, also called pseudo-distributed mode, allows each Hadoop daemon to run as a single Java process. A Hadoop environment is configured by editing a set of configuration files:

bashrc
hadoop-env.sh
core-site.xml
hdfs-site.xml
mapred-site-xml
yarn-site.xml

Configure Hadoop

We can add only the minimum property in the Hadoop configuration. The user can add more properties to it.

a) Setting Up the environment variables

Step 16) Modify ~/.bashrc file

COMMANDS
15.sudo nano .bashrc

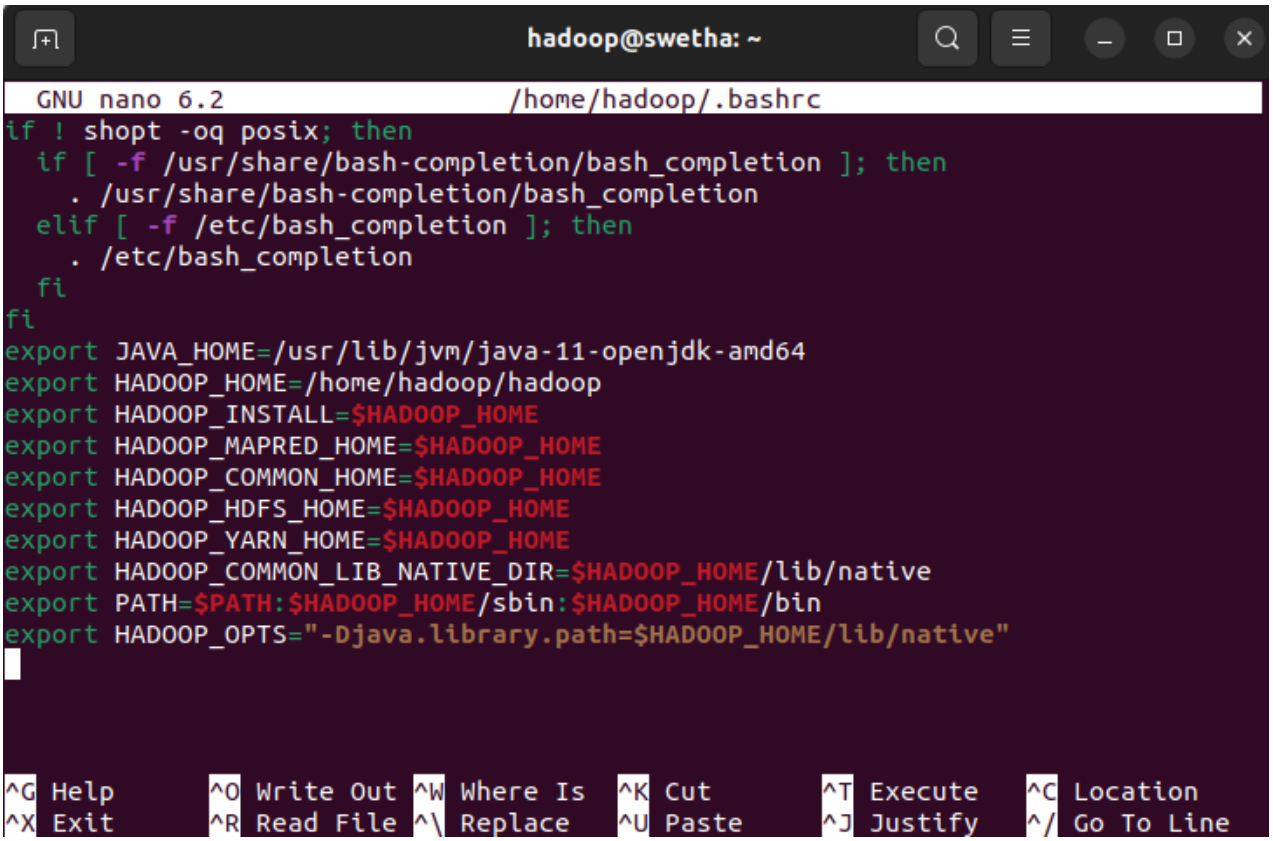
```
hadoop@swetha:~$ mv hadoop-3.3.0 hadoop
hadoop@swetha:~$ nano ~/.bashrc
```

This command opens a window. Navigate to the end of the window and paste the following lines to it. Then hold CTRL+x to exit. Type "y" and press enter to save the file.

Define the Hadoop environment variables by adding the following content to the end of the file:

```
#Hadoop Related Options
export HADOOP_HOME=/home/gokilanm/hadoop-3.2.2
export HADOOP_INSTALL=$HADOOP_HOME
export HADOOP_MAPRED_HOME=$HADOOP_HOME
export HADOOP_COMMON_HOME=$HADOOP_HOME
export HADOOP_HDFS_HOME=$HADOOP_HOME
export YARN_HOME=$HADOOP_HOME
export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_HOME/lib/native
export PATH=$PATH:$HADOOP_HOME/sbin:$HADOOP_HOME/bin
export HADOOP_OPTS="-
Djava.library.path=$HADOOP_HOME/lib/native"
```

TERMINAL:



Once you add the variables, save and exit the .bashrc file.

It is vital to apply the changes to the current running environment by using the following command:

COMMANDS
16.source ~/.bashrc

STEP:17 Configurations related to HDFS

The hadoop-env.sh file serves as a master file to configure YARN, HDFS, MapReduce, and Hadoop-related project settings.

When setting up a single node Hadoop cluster, you need to define which Java implementation is to be utilized. Use the previously created $HADOOP_HOME variable to access the hadoop-env.sh file:

COMMANDS
17. sudo nano $HADOOP_HOME/etc/hadoop/hadoop-env.sh

Uncomment the $JAVA_HOME variable (i.e., remove the # sign) and add the full path to the OpenJDK installation on your system. If you have installed the same version as presented in the first part of this tutorial, add the following line:

```
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
```

The path needs to match the location of the Java installation on your system

If you need help to locate the correct Java path, run the following command in your terminal window:

COMMANDS
18. which javac

The resulting output provides the path to the Java binary directory.

```
hadoop@swetha:~$ which javac
/usr/bin/javac
```

Use the provided path to find the OpenJDK directory with the following command:

COMMANDS
19. readlink -f /usr/bin/javac

```
hadoop@swetha:~$ readlink -f /usr/bin/javac
/usr/lib/jvm/java-11-openjdk-amd64/bin/javac
hadoop@swetha:~$
```

The section of the path just before the /bin/javac directory needs to be assigned to the $JAVA_HOME variable.

STEP 18:Edit core-site.xml File

The core-site.xml file defines HDFS and Hadoop core properties.

To set up Hadoop in a pseudo-distributed mode, you need to specify the URL for your NameNode, and the temporary directory Hadoop uses for the map and reduce process.

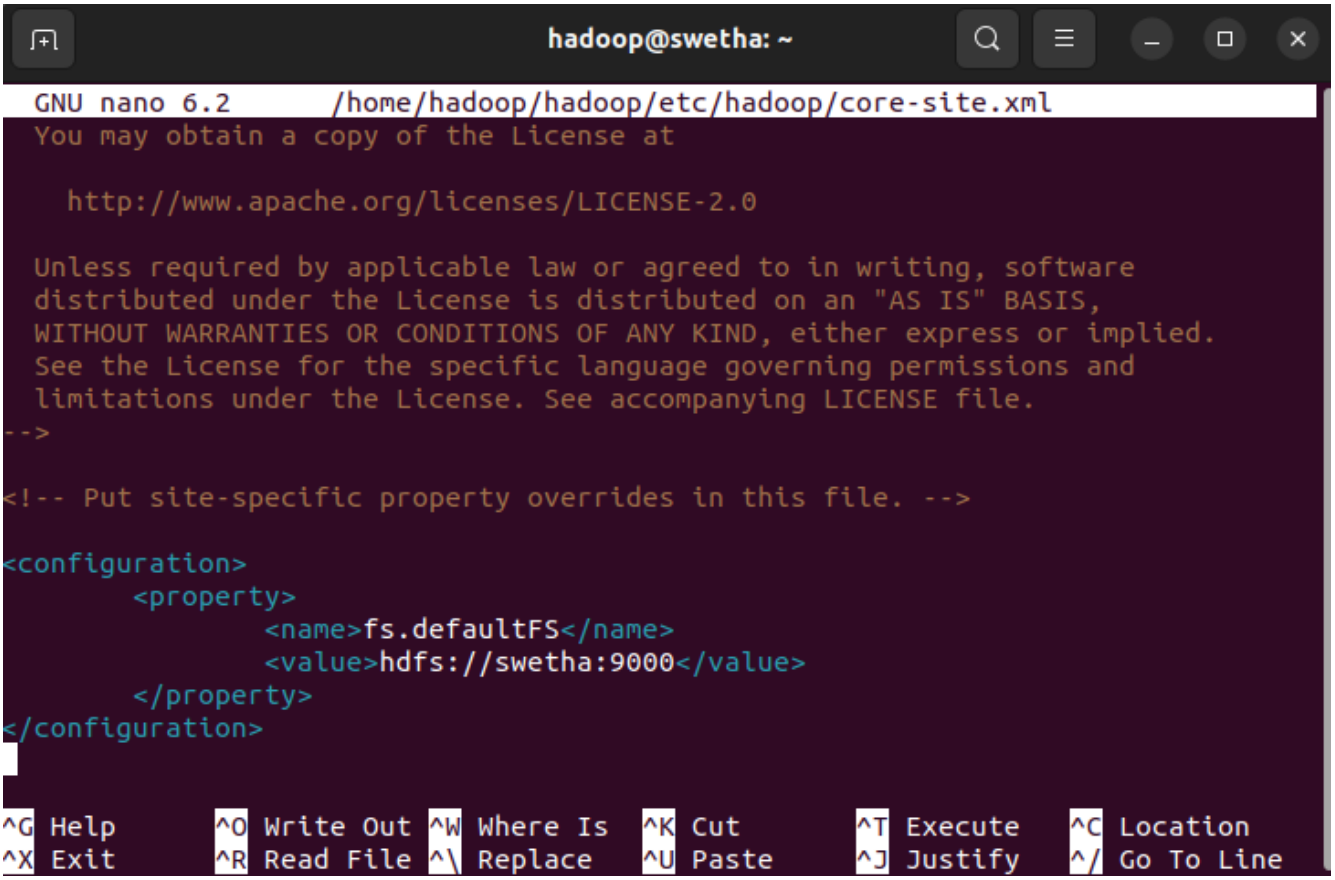
COMMANDS
20. sudo nano $HADOOP_HOME/etc/hadoop/core-site.xml

Open the core-site.xml file in a text editor:

Add the following configuration to override the default values for the temporary directory and add your HDFS URL to replace the default local file system setting:

```
<configuration>
<property>
  <name>hadoop.tmp.dir</name>
  <value>/home/gokilanm/tmpdata</value>
</property>
<property>
  <name>fs.default.name</name>
  <value>hdfs://127.0.0.1:9000</value>
</property>
</configuration>
```

TERMINAL:



This example uses values specific to the local system. You should use values that match your systems requirements. The data needs to be consistent throughout the configuration process.

STEP 19: Edit hdfs-site.xml File

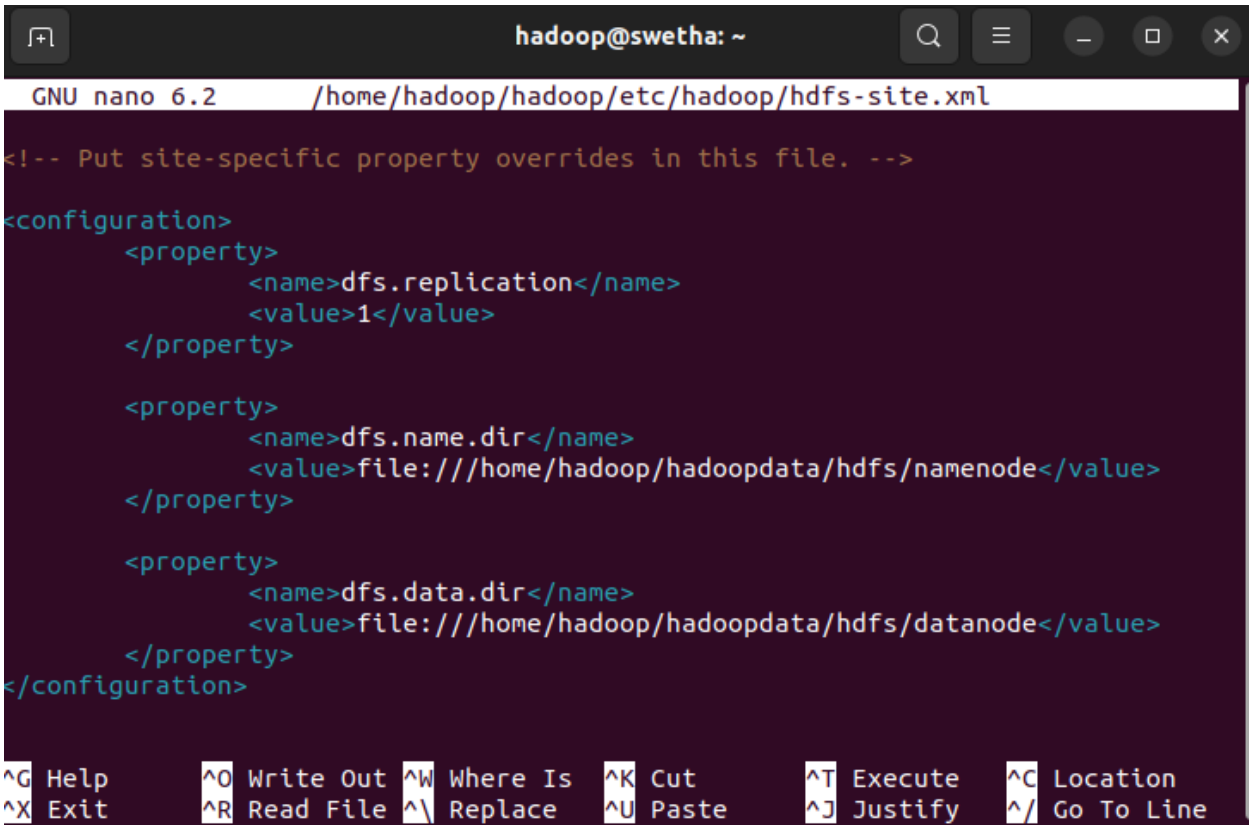
The properties in the hdfs-site.xml file govern the location for storing node metadata, fsimage file, and edit log file. Configure the file by defining the NameNode and DataNode storage directories.Additionally, the default dfs.replication value of 3 needs to be changed to 1 to match the single node setup.Use the following command to open the hdfs-site.xml file for editing:

COMMANDS
21. sudo nano $HADOOP_HOME/etc/hadoop/hdfs-site.xml

Add the following configuration to the file and, if needed, adjust the NameNode and DataNode directories to your custom locations:

```
<configuration>
<property>
  <name>dfs.data.dir</name>
  <value>/home/gokilanm/dfsdata/namenode</value>
</property>
<property>
  <name>dfs.data.dir</name>
  <value>/home/gokilanm/dfsdata/datanode</value>
</property>
<property>
  <name>dfs.replication</name>
  <value>1</value>
</property>
</configuration>
```

If necessary, create the specific directories you defined for the dfs.data.dir value.



STEP 20: Edit mapred-site.xml File

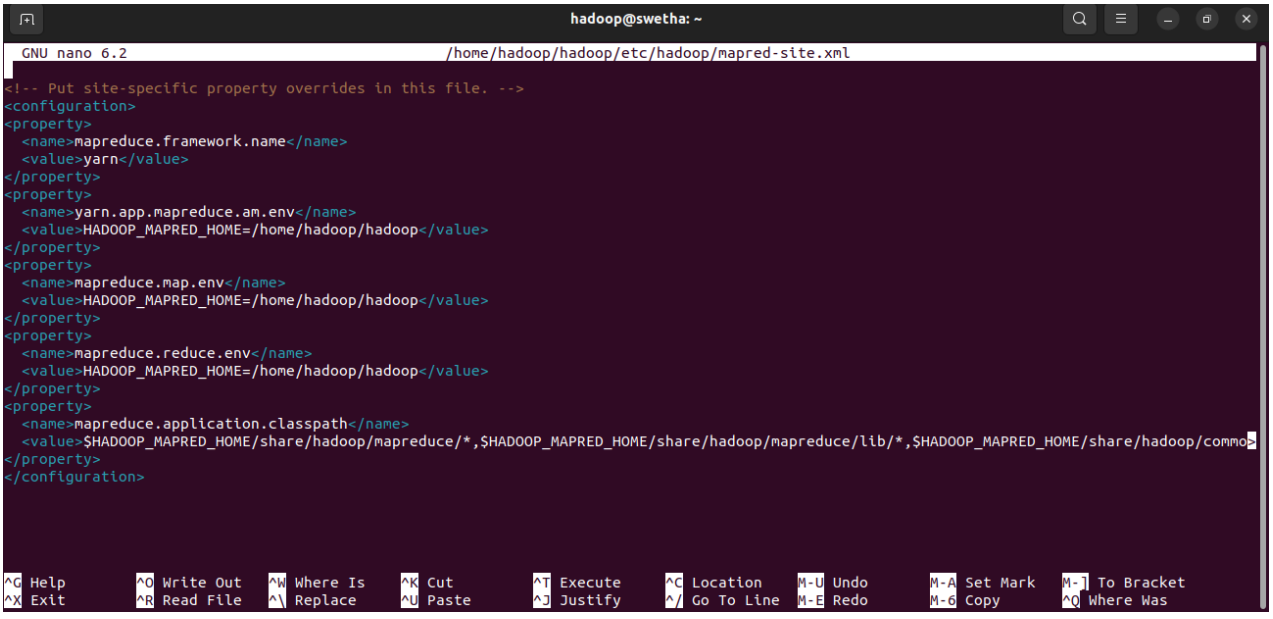
Use the following command to access the mapred-site.xml file and define MapReduce values:

COMMANDS
20. sudo nano $HADOOP_HOME/etc/hadoop/mapred-site.xml

Add the following configuration to change the default MapReduce framework name value to yarn:

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

TERMINAL:



STEP 21:Edit yarn-site.xml File

The yarn-site.xml file is used to define settings relevant to YARN. It contains configurations for the Node Manager, Resource Manager, Containers, and Application Master.

Open the yarn-site.xml file in a text editor:

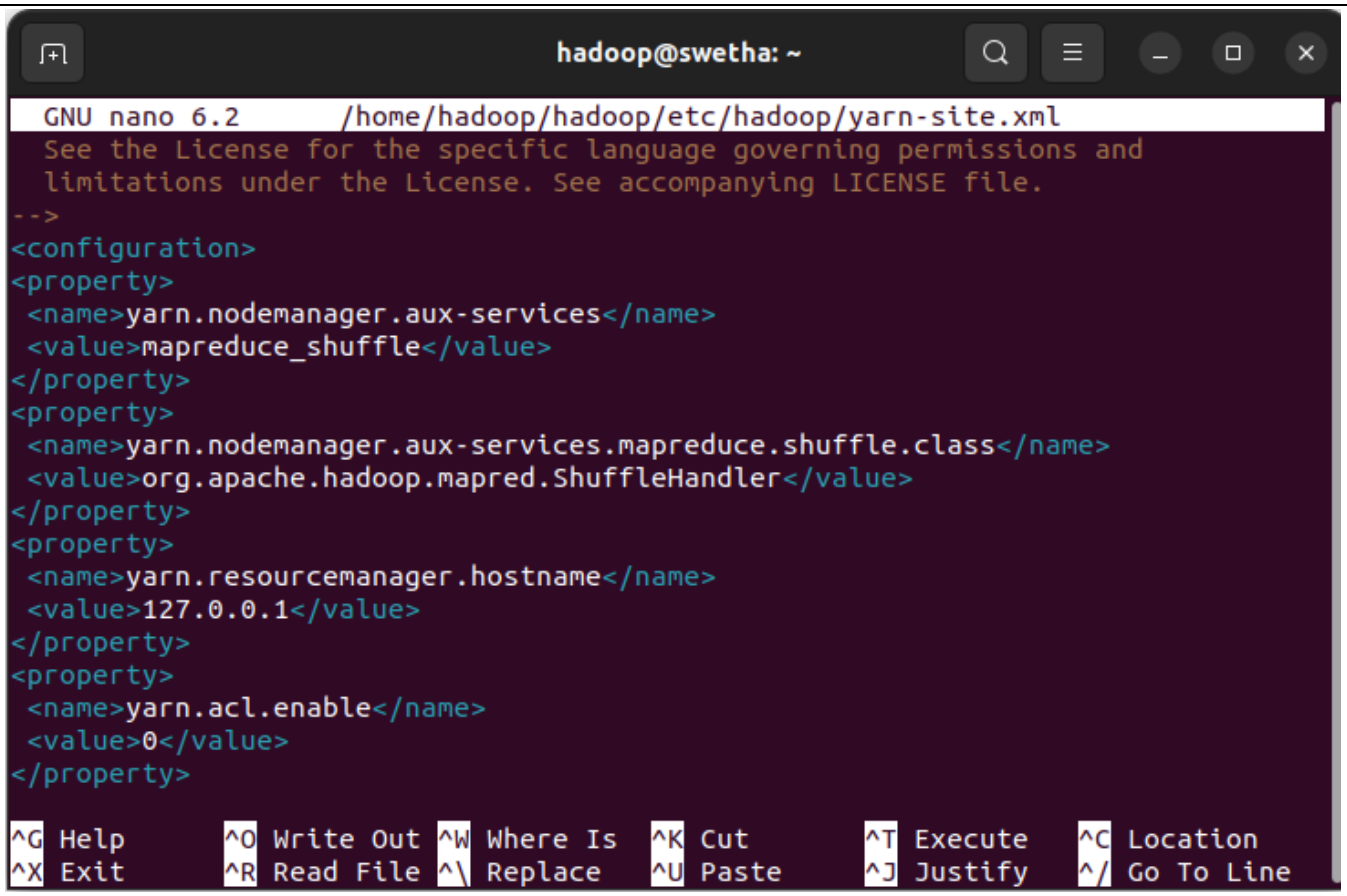
COMMANDS
23. sudo nano $HADOOP_HOME/etc/hadoop/yarn-site.xml

Append the following configuration to the file:

```
<configuration>
<property>
  <name>yarn.nodemanager.aux-services</name>
  <value>mapreduce_shuffle</value>
</property>
<property>
  <name>yarn.nodemanager.aux-
services.mapreduce.shuffle.class</name>
```

```
  <value>org.apache.hadoop.mapred.ShuffleHandler</value>
</property>
<property>
  <name>yarn.resourcemanager.hostname</name>
  <value>127.0.0.1</value>
</property>
<property>
  <name>yarn.acl.enable</name>
  <value>0</value>
</property>
<property>
  <name>yarn.nodemanager.env-whitelist</name>
<value>JAVA_HOME,HADOOP_COMMON_HOME,HADOOP_HDFS
_HOME,HADOOP_CONF_DIR,CLASSPATH_PERPEND_DISTCACH
E,HADOOP_YARN_HOME,HADOOP_MAPRED_HOME</value>
</property>
</configuration>
```

TERMINAL:



STEP 22: Format HDFS NameNode

Now format the namenode using the following command, make sure that Storage directory is hdfs namenode -format

It is important to format the NameNode before starting Hadoop services for the first time:

COMMANDS
24. hdfs namenode -format

```
WARNING: /home/hadoop/hadoop/logs does not exist. Creating.
2022-11-04 20:11:20,420 INFO namenode.NameNode: STARTUP_MSG:
/************************************************************
STARTUP_MSG: Starting NameNode
STARTUP_MSG:   host = swetha/127.0.1.1
STARTUP_MSG:   args = [-format]
STARTUP_MSG:   version = 3.3.0
STARTUP_MSG:   classpath = /home/hadoop/hadoop/etc/hadoop:/home/hadoop/hadoop/sh
are/hadoop/common/lib/jsr305-3.0.2.jar:/home/hadoop/hadoop/share/hadoop/common/l
ib/kerb-client-1.0.1.jar:/home/hadoop/hadoop/share/hadoop/common/lib/animal-snif
fer-annotations-1.17.jar:/home/hadoop/hadoop/share/hadoop/common/lib/kerby-xdr-1
.0.1.jar:/home/hadoop/hadoop/share/hadoop/common/lib/audience-annotations-0.5.0.
jar:/home/hadoop/hadoop/share/hadoop/common/lib/jersey-json-1.19.jar:/home/hadoo
p/hadoop/share/hadoop/common/lib/jsr311-api-1.1.1.jar:/home/hadoop/hadoop/share/
hadoop/common/lib/jackson-core-asl-1.9.13.jar:/home/hadoop/hadoop/share/hadoop/c
ommon/lib/jettison-1.1.jar:/home/hadoop/hadoop/share/hadoop/common/lib/nimbus-jo
se-jwt-7.9.jar:/home/hadoop/hadoop/share/hadoop/common/lib/jetty-util-9.4.20.v20
190813.jar:/home/hadoop/hadoop/share/hadoop/common/lib/jaxb-api-2.2.11.jar:/home
/hadoop/hadoop/share/hadoop/common/lib/jetty-http-9.4.20.v20190813.jar:/home/had
oop/hadoop/share/hadoop/common/lib/htrace-core4-4.1.0-incubating.jar:/home/hadoo
p/hadoop/share/hadoop/common/lib/kerby-asn1-1.0.1.jar:/home/hadoop/hadoop/share/
```

The shutdown notification signifies the end of the NameNode format process.

STEP 23:Start Hadoop Cluster

Navigate to the hadoop-3.2.1/sbin directory and execute the following commands to start the NameNode and DataNode:

The system takes a few moments to initiate the necessary nodes.

COMMANDS
25.cd hadoop-3.3.0 26.cd sbin 27./start-dfs.sh

Once the namenode, datanodes, and secondary namenode are up and running, start the YARN resource and nodemanagers by typing:

COMMANDS
28. ./start-yarn.sh

As with the previous command, the output informs you that the processes are starting.

```
Starting namenodes on [0.0.0.0]
Starting datanodes
Starting secondary namenodes [sankamethra-VirtualBox]
```

```
5265 ResourceManager
5042 SecondaryNameNode
```

Type this simple command to check if all the daemons are active and running as Java processes:

COMMANDS
29.jps

If everything is working as intended, the resulting list of running Java processes contains all the HDFS and YARN daemons.

```
5265 ResourceManager
5042 SecondaryNameNode
4710 NameNode
5385 NodeManager
4826 DataNode
29742 Jps
```

STEP 24: Access Hadoop UI from Browser

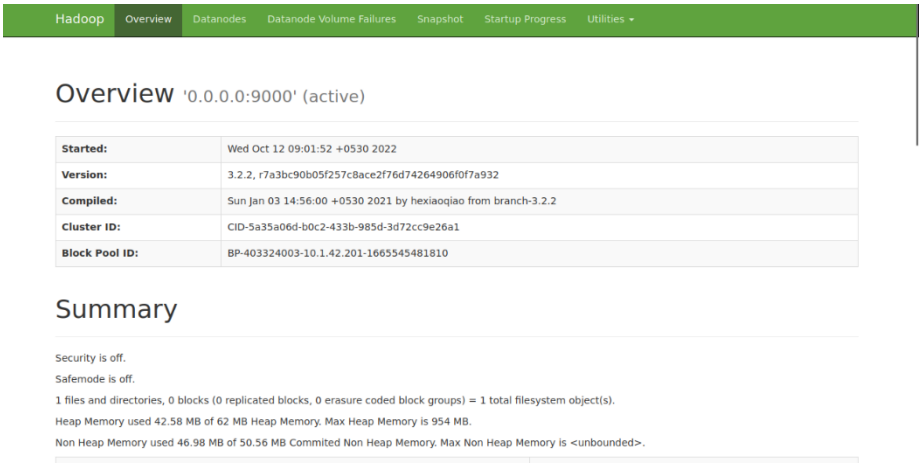
The default port number 9870 gives you access to the Hadoop NameNode UI:

http://localhost:9870

The NameNode user interface provides a comprehensive overview of the entire cluster.

The default port 9864 is used to access individual DataNodes directly from your browser:

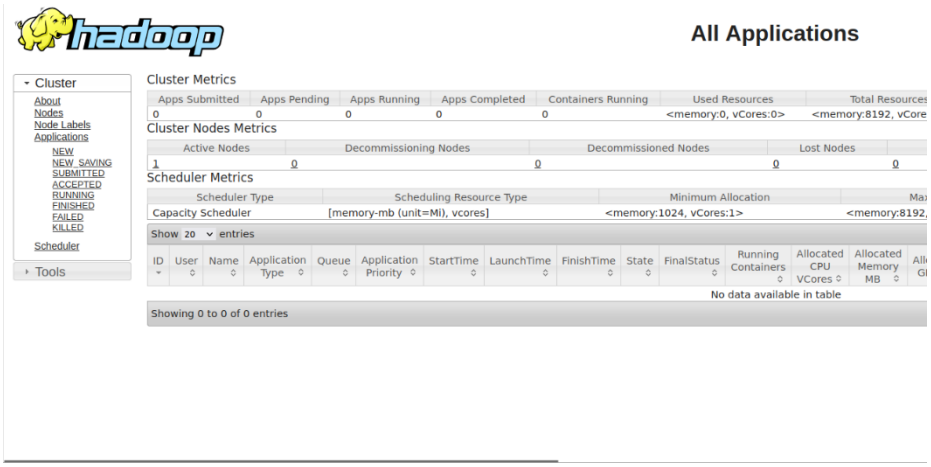
http://localhost:9864



The YARN Resource Manager is accessible on port 8088:

The Resource Manager is an invaluable tool that allows you to monitor all running processes in your Hadoop cluster.

http://localhost:8088



RESULT:

Hadoop is powerful because it is extensible and it is easy to integrate with any component. Its popularity is due in part to its ability to store, analyze and access large amounts of data, quickly and cost effectively across clusters of commodity hardware. You have successfully installed Hadoop on Ubuntu and deployed it in a pseudo-distributed mode. A single node Hadoop deployment is an excellent starting point to explore basic HDFS commands and acquire the experience you need to design a fully distributed Hadoop cluster.

Experimet 2: Implement the following file management tasks in Hadoop:
- Adding files and directories
- Retrieving files
- Deleting files

a) *OBJECTIVE OF THE EXERCISE/EXPERIMENT*

Implement the following file management tasks in Hadoop:

i. Adding files and directories
ii. Retrieving files
iii. Deleting files

b) *Facilities/material required to do the exercise/experiment:*

Sl.No.	*Facilities/material required*	*Quantity*
1.	*Hadoop*	*As per strength*

THEORY:

Hadoop File System was developed using distributed file system design. It is run on commodity hardware. Unlike other distributed systems, HDFS is highly fault tolerant and designed using low-cost hardware.

HDFS holds very large amount of data and provides easier access. To store such huge data, the files are stored across multiple machines.
These files are stored in redundant fashion to rescue the system from possible data losses in case of failure. HDFS also makes applications available to parallel processing.

FILE OPERATIONS IN HADOOP:

1.Open terminal in Ubuntu and start a cluster

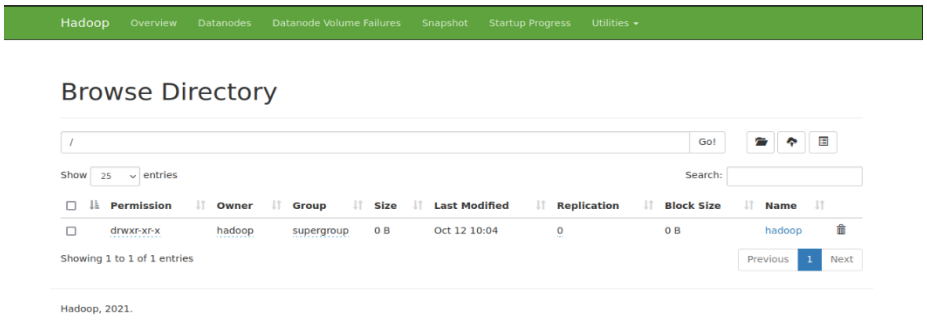
```
hadoop@sankamethra-VirtualBox:~$ start-dfs.sh
Starting namenodes on [0.0.0.0]
Starting datanodes
Starting secondary namenodes [sankamethra-VirtualBox]
hadoop@sankamethra-VirtualBox:~$ start-yarn.sh
Starting resourcemanager
Starting nodemanagers
hadoop@sankamethra-VirtualBox:~$ jps
5265 ResourceManager
5042 SecondaryNameNode
4710 NameNode
5385 NodeManager
4826 DataNode
5516 Jps
```

1. *Create a directory in HDFS at given path(s).*
hadoop fs -mkdir <paths>

COMMANDS
hdfs dfs -mkdir /swe/

OUTPUT:

File is created at a hdfs directory



2. *List the contents of a directory.*

ls-List directories present under a specific directory in HDFS, similar to Unix ls command. The -lsr command can be used for recursive listing of directories and files.

Usage :

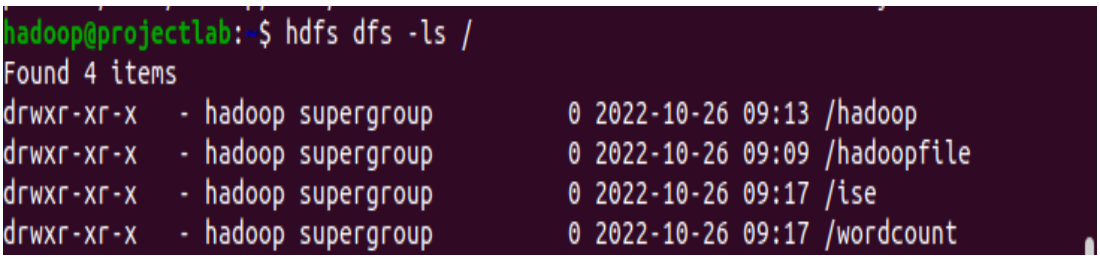
hadoop fs -ls <args>

We can list files present in a directory using -ls

COMMANDS
hdfs dfs -ls /

OUTPUT:

We can see a directory 'gokilafile' (created earlier) being listed under ' / ' directory.



3.*Upload and download a file in H Upload:*

hadoop fs -put:

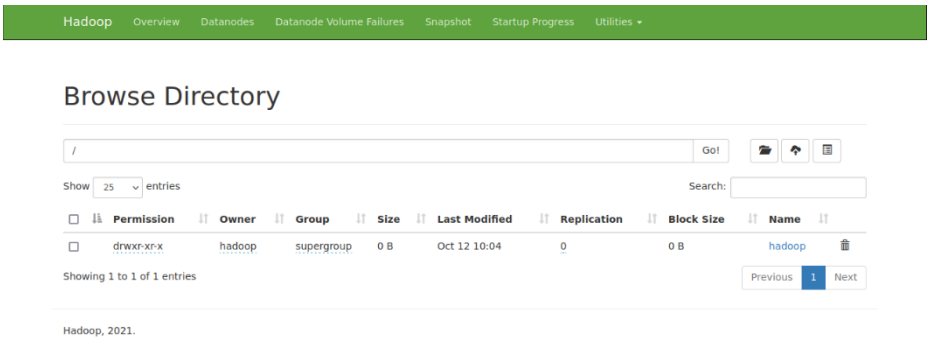
- *Copy single src file, or multiple src files from local file system to the Hadoop data file system*
- *Copy files from the local file system to HDFS, similar to -put command. This command will not work if the file already exists. To overwrite the destination if the file already exists, add -f flag to command.*

Usage:

hadoop fs -put <localsrc> ... <HDFS_dest_Path>DFS.

COMMANDS
hdfs dfs -put '/home/Hadoop/hadoop/swetha/lab' /swe/

OUTPUT FILE:



Download:

hadoop fs -get:

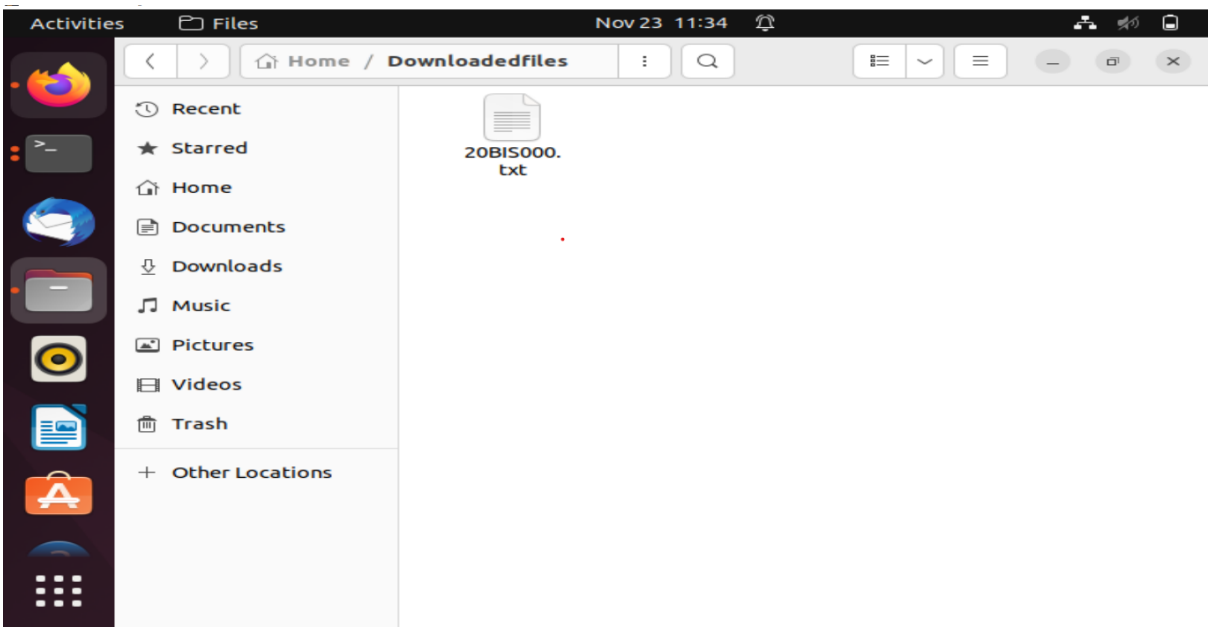
Copies/Downloads files to the local file

system Usage:

hadoop fs -get <hdfs_src> <localdst>

COMMANDS
hadoop fs-get /swe/lab /home/hadoop/

OUTPUT:



4.*See contents of a file:*

View HDFS file content using cat

command: Usage:

hadoop fs -cat <path[filename]>

COMMANDS:
hadoop fs-cat /swe/lab

OUTPUT:

```
hadoop@sankamethra-VirtualBox:/usr/local/hadoop/lib$ hadoop fs -cat /samplefile
/20BIS000.txt/
Hello
hadoop@sankamethra-VirtualBox:/usr/local/hadoop/lib$
```

5. *Copy a file from source to destination*

This command allows multiple sources as well in which case the destination must be a directory.

Usage:

hadoop fs -cp <source> <dest>

COMMANDS:
hadoop fs-cp /swe/lab /lab2/

OUTPUT:

1.Create a new directory named lab2.

2.copy a file from one directory to another

directory. 3.print the contents of a file and check it

is copied or not.

```
hadoop@sankamethra-VirtualBox:/usr/local/hadoop/lib$ hdfs dfs -mkdir /newfile/
hadoop@sankamethra-VirtualBox:/usr/local/hadoop/lib$ hadoop fs -cp /samplefile/
20BIS000.txt /newfile/
hadoop@sankamethra-VirtualBox:/usr/local/hadoop/lib$
```

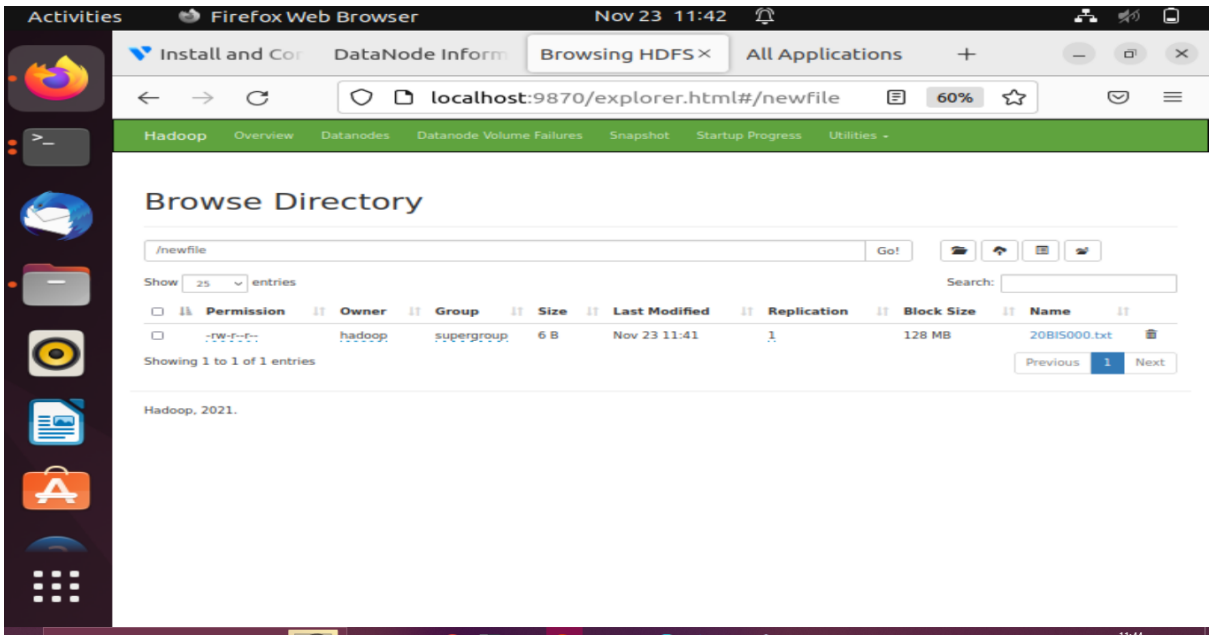
6. *Copy a file from/To Local file system to HDFS*

copyFromLoc

al Usage:

hadoop fs -copyFromLocal <localsrc> URI

COMMANDS:
hadoop fs -copyFromLocal ‘/home/hadoop/hadoop/swetha/lab’ /swe/



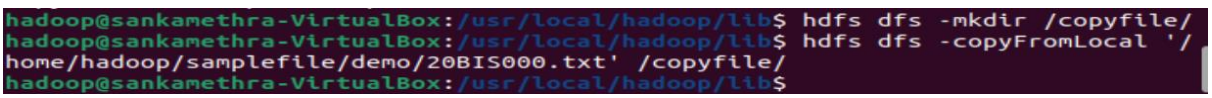
7. copyToLocal

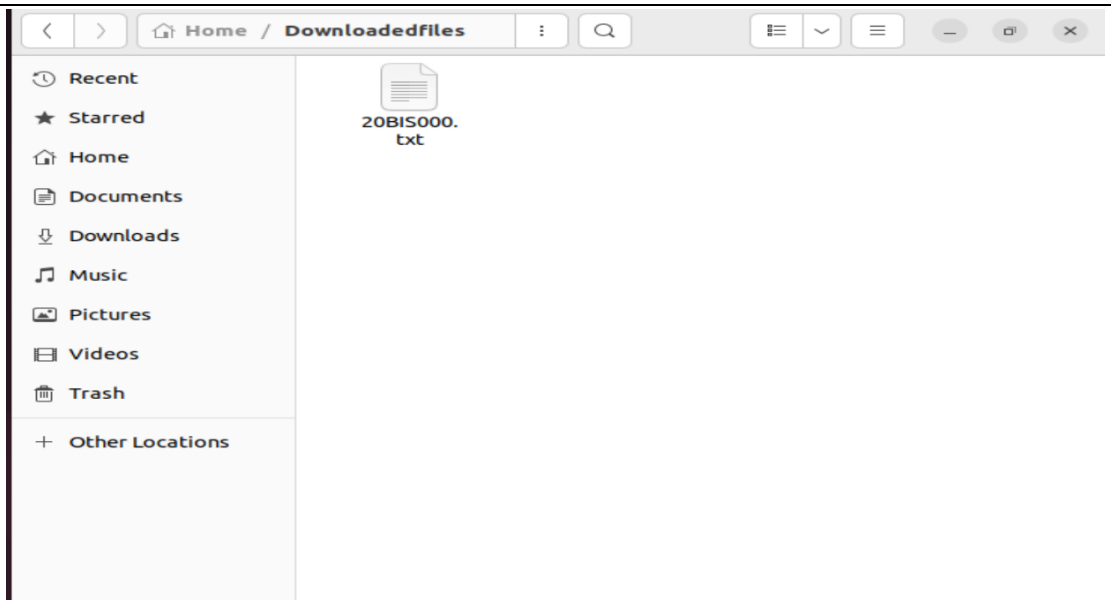
Usage:

hadoop fs -copyToLocal [-ignorecrc] [-crc] URI <localdst>Similar to get command, except that the destination is restricted to a local file reference

COMMANDS:
hadoop fs -copyToLocal /swe/lab ‘/home/hadoop/Videos/’

TERMINAL:





***8.** Move file from source to destination.*

Usage : hadoop fs -mv <src> <dest>

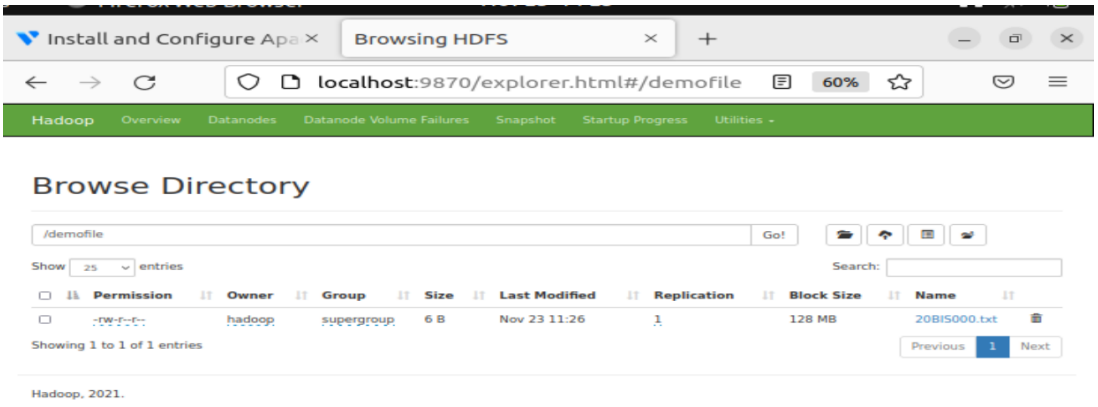
COMMANDS:

```
hdfs fs -mv /swe/lab  /lab2/
```

TERMINAL:

```
hadoop@sankamethra-VirtualBox:~$ hdfs dfs -mv /samplefile/20BIS000.txt /demofil
e/
hadoop@sankamethra-VirtualBox:~$
```

OUTPUT:



9. *Remove a file or directory in HDFS.*

Remove files specified as argument. Deletes directory only when it is empty Usage :

hadoop fs -rm

<arg> Example:

COMMANDS:
hdfs dfs -rmdir /lab/

```
hadoop@sankamethra-VirtualBox:~$ hdfs dfs -rmdir /demofiless/
hadoop@sankamethra-VirtualBox:~$
```

10. *Display last few lines of a file. Similar to tail command in Unix. Usage :*

hadoop fs -tail <path[filename]>

COMMANDS:
hadoop fs -tail /swe/lab/

11. Display the aggregate length of a file.

Usage :

hadoop fs -du <path>

COMMANDS:
hadoop fs -du /swe/lab

```
hadoop@sankamethra-VirtualBox:~$ hadoop fs -du /copyfile/20BIS000.txt/
6  6  /copyfile/20BIS000.txt
hadoop@sankamethra-VirtualBox:~$
```

Result:

All the file operations can be implemented in hadoop.

Experiment 3: Run a basic Word Count Map Reduce program to understand Map Reduce Paradigm

a) *OBJECTIVE OF THE EXERCISE/EXPERIMENT*

- *Implementing distinct word count problem using Map-Reduce*

b) *Facilities/material required to do the exercise/experiment:*

Sl.No.	*Facilities/material required*	*Quantity*
1.	*Hadoop*	*As per strength*

Theory:

To run a MapReduce job, users should furnish a map function, a reduce function, input data, and an output data location.
When executed, Hadoop carries out the following steps:

1. *Hadoop breaks the input data into multiple data items by new lines and runs the map function once for each data item, giving the item as the input for the function. When executed, the map function outputs one or more key- value pairs.*
2. *Hadoop collects all the key-value pairs generated from the map function, sorts them by the key, and groups together the values with the same key.*
3. *For each distinct key, Hadoop runs the reduce function once while passing the key and list of values for that key as input.*
4. *The reduce function may output one or more key-value pairs, and Hadoop writes them to a file as the final result.*

Procedure:

1. Analyze the input file content

2. Creating Input path in HDFS and moving the data into Input path

3. Develop the code

- *Writing a map function*
- *Writing a reduce function*
- *Writing the Driver class*

4. Compiling the source

5. Building the JARfile

6. Starting the DFS

7. Executing the program.

Source Code:

```
import java.io.IOException;
import
java.util.StringTokenizer;
import
org.apache.hadoop.io.IntWritable;
import
org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import
org.apache.hadoop.mapreduce.Mapper;
import
org.apache.hadoop.mapreduce.Reducer;
import
org.apache.hadoop.conf.Configuration;
import
org.apache.hadoop.mapreduce.Job;
import
org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import
org.apache.hadoop.mapreduce.lib.output.TextOutputForma
t; import
```

```
org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import
org.apache.hadoop.mapreduce.lib.output.FileOutputFormat
; import org.apache.hadoop.fs.Path;
public class WordCount
{
public static class Map extends
Mapper<LongWritable,Text,Text,IntWritable> { public void
map(LongWritable key, Text value,Context context) throws
IOException,InterruptedException{
```

```
String line = value.toString();
StringTokenizer tokenizer = new
StringTokenizer(line); while
(tokenizer.hasMoreTokens()) {
value.set(tokenizer.nextToken());
context.write(value, new IntWritable(1));
}
}
}
public static class Reduce extends
Reducer<Text,IntWritable,Text,IntWritable> { public void reduce(Text
key, Iterable<IntWritable> values,Context context) throws
IOException,InterruptedException {
int sum=0;
for(IntWritable x:
values)
{
sum+=x.get();
}
context.write(key, new IntWritable(sum));
}
}
public static void main(String[] args) throws
Exception { Configuration conf= new
Configuration();
Job job = new Job(conf,"My Word Count
Program");
job.setJarByClass(WordCount.class);
job.setMapperClass(Map.class);
job.setReducerClass(Reduce.class);
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(IntWritable.class);
job.setInputFormatClass(TextInputFormat.clas
s);
job.setOutputFormatClass(TextOutputFormat.c
lass); Path outputPath = new Path(args[1]);
```

//Configuring the input/output path from the filesystem into the job FileInputFormat.addInputPath(job, new Path(args[0]));

FileOutputFormat.setOutputPath(job, new Path(args[1]));

//deleting the output path automatically from hdfs so that we

don't have to delete it explicitly

outputPath.getFileSystem(conf).delete(outputPath);

//exiting the job only if the flag value becomes

false

System.exit(job.waitForCompletion(true) ? 0 :

1);

}

}

SOURCE CODE-EXPLANATION:-JAVA CODE

The entire MapReduce program can be fundamentally divided into three parts:

- *Mapper Phase Code*
- *Reducer Phase Code*
- *Driver Code*

We will understand the code for each of these three parts sequentially.

Mapper code:

```
public static class Map extends
Mapper<LongWritable,Text,Text,IntWritable> {
public void map(LongWritable key, Text value, Context context) throws
IOException,InterruptedException {
String line = value.toString();
StringTokenizer tokenizer = new StringTokenizer(line);
while (tokenizer.hasMoreTokens()) {
value.set(tokenizer.nextToken());
context.write(value, new IntWritable(1));
}
```

- *We have created a class Map that extends the class Mapper which is already defined in the MapReduce Framework.*
- *We define the data types of input and output key/value pair after the class declaration using angle brackets.*
- *Both the input and output of the Mapper is a key/value pair.*

Input:

- *The key is nothing but the offset of each line in the text file: LongWritable*
- *The value is each individual line : Text*

Output:

◦ *The key is the tokenized words: Text*

◦ *We have the hardcoded value in our case which is 1: IntWritable*

◦ *Example – Dear 1, Bear 1, etc.*

• *We have written a java code where we have tokenized each word and assigned them a hardcoded value equal to 1.*

Reducer Code:

```
public static class Reduce extends
Reducer<Text,IntWritable,Text,IntWritable> {
public void reduce(Text key, Iterable<IntWritable> values,Context
context)
throws IOException,InterruptedException {
int sum=0;
for(IntWritable x: values)
{
sum+=x.get();
}
context.write(key, new IntWritable(sum));
}
}
```

- *We have created a class Reduce which extends class Reducer like that of Mapper.*
- *We define the data types of input and output key/value pair after the class declaration using angle brackets as done for Mapper.*
- *Both the input and the output of the Reducer is a keyvalue pair.*

Input:

- *The key nothing but those unique words which have been generated after the sorting and shuffling phase: Text*
- *The value is a list of integers corresponding to each key: IntWritable*

Output:

- *The key is all the unique words present in the input text file: Text*
- *The value is the number of occurrences of each of the unique words: IntWritable*
- *We have aggregated the values present in each of the list corresponding to each key and produced the final answer.*

- *In general, a single reducer is created for each of the unique words, but,you can specify the number of reducer in mapred-site.xml.*

Driver Code:

```
Configuration conf= new Configuration();
Job job = new Job(conf,"My Word Count Program");
job.setJarByClass(WordCount.class);
job.setMapperClass(Map.class);
job.setReducerClass(Reduce.class);
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(IntWritable.class);
job.setInputFormatClass(TextInputFormat.class);
job.setOutputFormatClass(TextOutputFormat.class);
Path outputPath = new Path(args[1]);
//Configuring the input/output path from the filesystem into the job
FileInputFormat.addInputPath(job, new Path(args[0]));
FileOutputFormat.setOutputPath(job, new Path(args[1]));
```

In the driver class, we set the configuration of our MapReduce job to run in Hadoop.

- *We specify the name of the job , the data type of input/output of the mapper and reducer.*
- *We also specify the names of the mapper and reducer classes.*
- *The path of the input and output folder is also specified.*
- *The method setInputFormatClass () is used for specifying that how a Mapper will read the input data or what will be the unit of work. Here, we have chosen TextInputFormat so that single line is read by the mapper at a time from the input text file.*
- *The main () method is the entry point for the driver. In this method, we instantiate a new Configuration object for the job.*

SNAPSHOT:

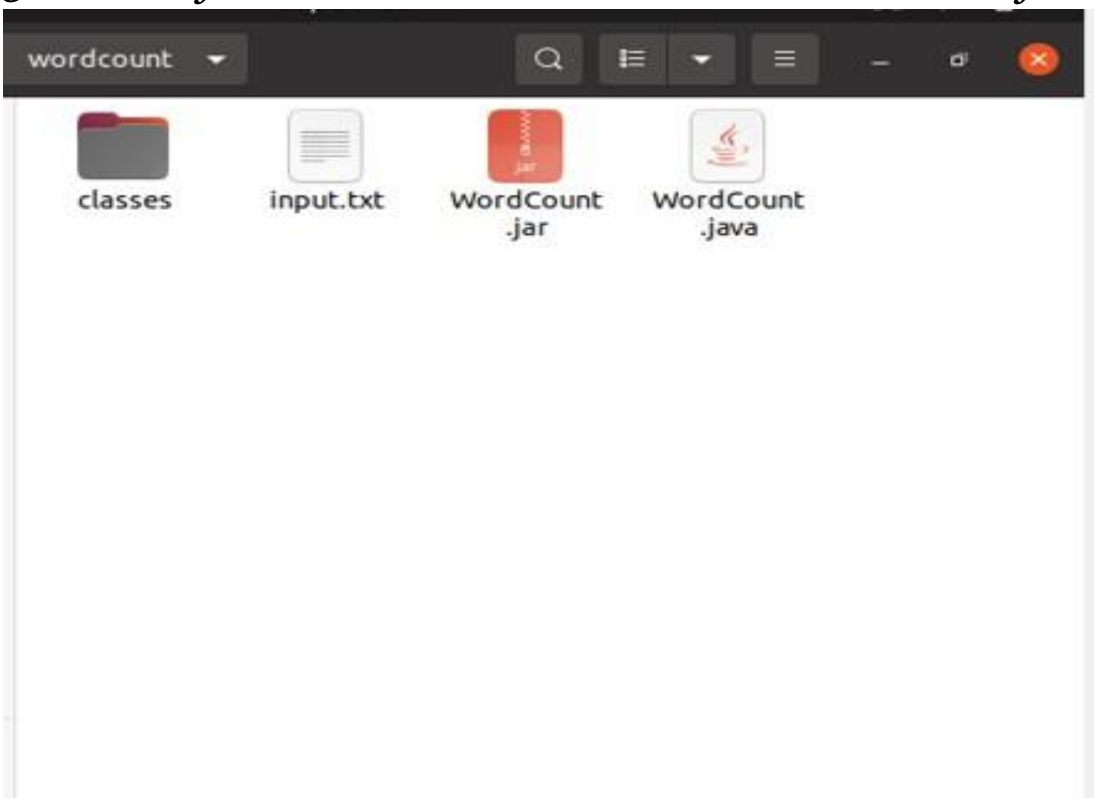
Open *WordCount.java ~/wordcount Save

```
1 import java.io.IOException;
2 import java.util.StringTokenizer;
3 import org.apache.hadoop.io.IntWritable;
4 import org.apache.hadoop.io.LongWritable;
5 import org.apache.hadoop.io.Text;
6 import org.apache.hadoop.mapreduce.Mapper;
7 import org.apache.hadoop.mapreduce.Reducer;
8 import org.apache.hadoop.conf.Configuration;
9 import org.apache.hadoop.mapreduce.Job;
10 import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
11 import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
12 import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
13 import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
14 import org.apache.hadoop.fs.Path;
15
16 public class WordCount
17 {
18 public static class Map extends Mapper<LongWritable,Text,Text,IntWritable> {
19 public void map(LongWritable key, Text value,Context context) throws
20 IOException,InterruptedException{
21 String line = value.toString();
22 StringTokenizer tokenizer = new StringTokenizer(line);
23 while (tokenizer.hasMoreTokens()) {
24 value.set(tokenizer.nextToken());
25 context.write(value, new IntWritable(1));
26 }
27 }
```

```
28 }
29 public static class Reduce extends
   Reducer<Text,IntWritable,Text,IntWritable> {
30 public void reduce(Text key, Iterable<IntWritable> values,Context context)
31 throws IOException,InterruptedException {
32 int sum=0;
33 for(IntWritable x: values)
34 {
35 sum+=x.get();
36 }
37 context.write(key, new IntWritable(sum));
38 }
39 }
40
41 public static void main(String[] args) throws Exception {
42
43 Configuration conf= new Configuration();
44 Job job = new Job(conf,"My Word Count Program");
45 job.setJarByClass(WordCount.class);
46 job.setMapperClass(Map.class);
47 job.setReducerClass(Reduce.class);
48 job.setOutputKeyClass(Text.class);
49 job.setOutputValueClass(IntWritable.class);
50 job.setInputFormatClass(TextInputFormat.class);
51 job.setOutputFormatClass(TextOutputFormat.class);
52 Path outputPath = new Path(args[1]);
53 //Configuring the input/output path from the filesystem into the job
54 FileInputFormat.addInputPath(job, new Path(args[0]));
55 FileOutputFormat.setOutputPath(job, new Path(args[1]));
56 //deleting the output path automatically from hdfs so that we don't have to delete it explicitly
57 outputPath.getFileSystem(conf).delete(outputPath);
58 //exiting the job only if the flag value becomes false
59 System.exit(job.waitForCompletion(true) ? 0 : 1);
60 }
61 }
```

INPUT FILE:

Prerequisites
1.Create a folder in the home named wordcount. 2.put the input file in the inside of wordcount folder.
3.create a folder named classes inside the wordcount folder.



Steps to execute MapReduce word count example

1.Set up the path and configure the java environment:

COMMANDS
1. export HADOOP_CLASSPATH=$(hadoop classpath) *2.echo $HADOOP_CLASSPATH*

2.Open cmd in Ubuntu terminal and start a cluster.

COMMANDS
Start-all.sh

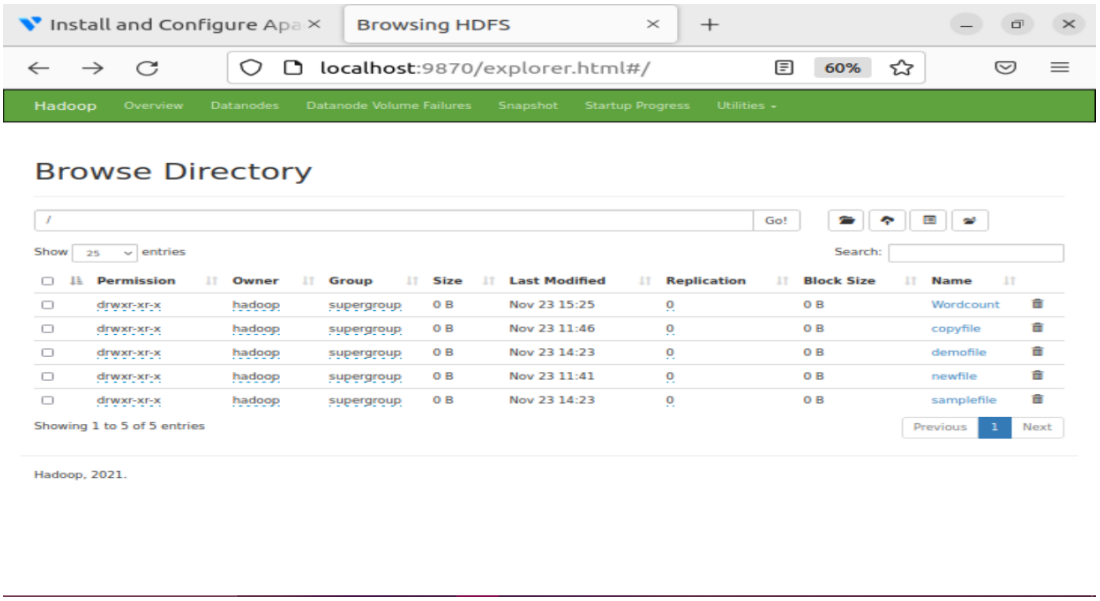
3.Verify the what are the processes running and list the process id

COMMANDS
jps

4. Create an input directory in HDFS.

COMMANDS
hdfs dfs -mkdir /wordcount/

Output:



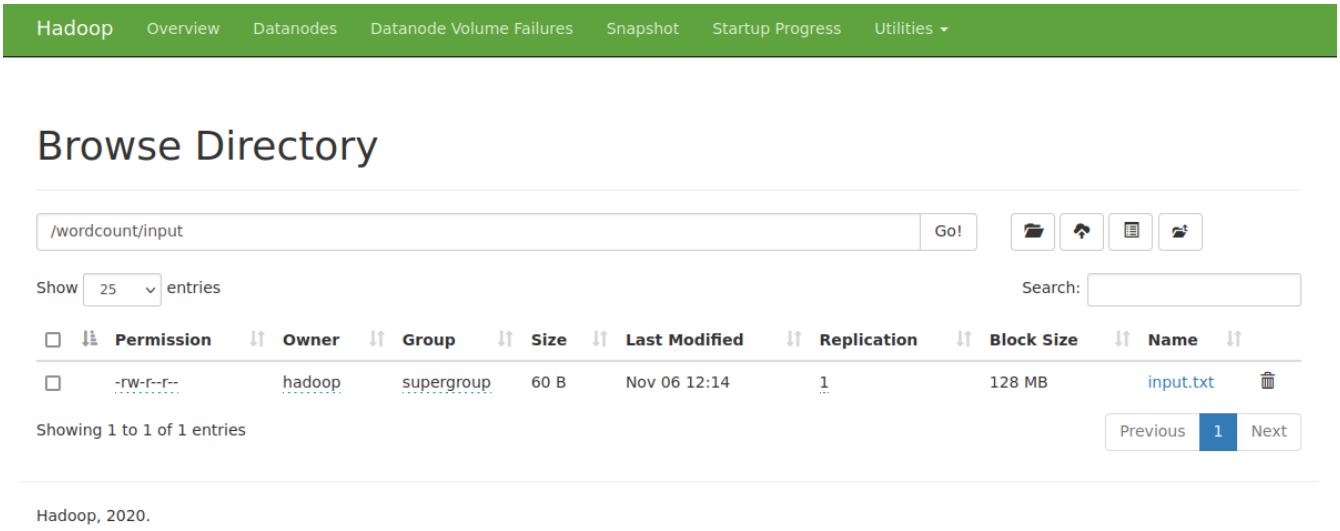
5.Create a input folder inside of wordcount directory.

COMMANDS
hdfs dfs -mkdir /wordcount/input

6.Copy the input text file named input_file.txt in the input directory (wordcount) of HDFS.

Copy some text file to hadoop filesystem inside input directory. Here I am copying a input.txt to it. You can copy more that one files.

COMMANDS
hdfs dfs -put '/home/hadoop/wordcount/input.txt' /wordcount/input/



7. Verify content of the copied file.

COMMANDS
hdfs dfs -cat /wordcount/input/*

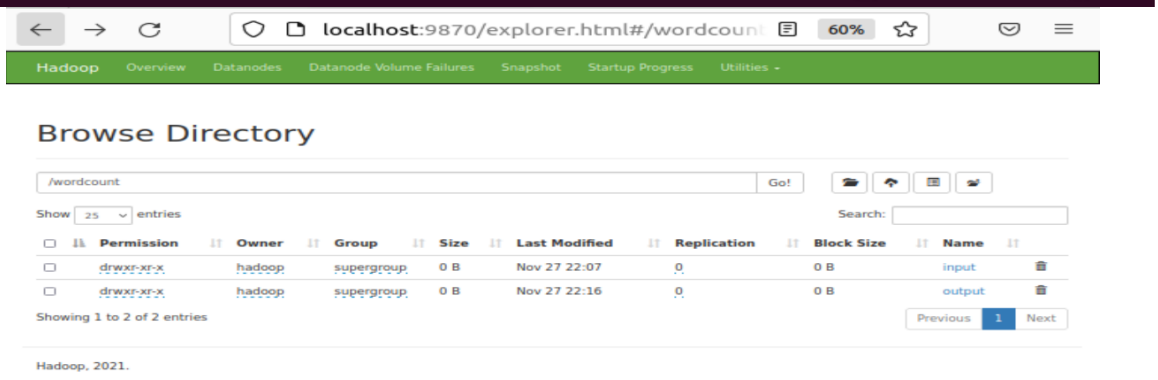
8.Move into the wordcount folder in home which contains the java file

COMMANDS
cd /home/hadoop/wordcount

and whether check and print the working directory and list the contents of the file.

TERMINAL:

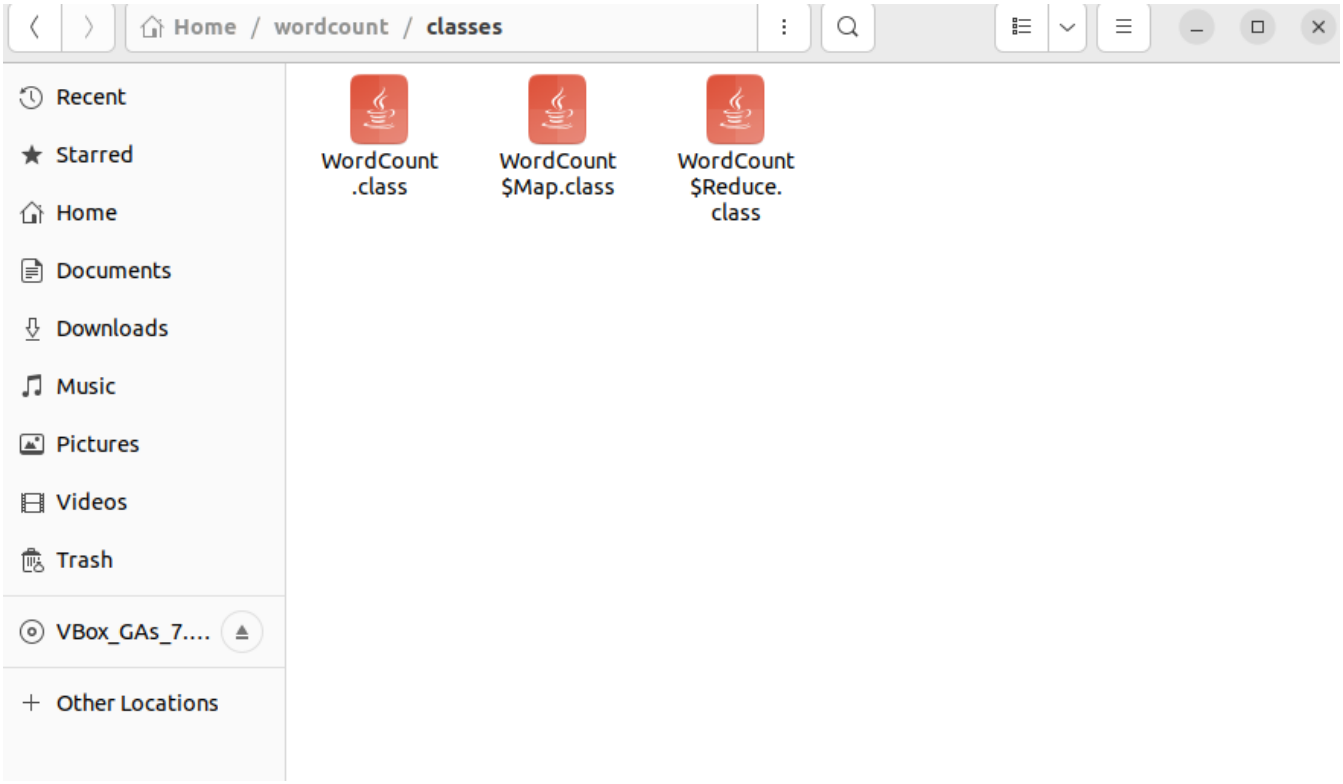
```
rdCount.jar WordCount /wordcount/input /wordcount/output
2022-11-27 22:12:19,403 INFO client.DefaultNoHARMFailoverProxyProvider: Connect
ing to ResourceManager at /0.0.0.0:8032
2022-11-27 22:12:19,949 WARN mapreduce.JobResourceUploader: Hadoop command-line
 option parsing not performed. Implement the Tool interface and execute your ap
plication with ToolRunner to remedy this.
2022-11-27 22:12:20,015 INFO mapreduce.JobResourceUploader: Disabling Erasure C
oding for path: /tmp/hadoop-yarn/staging/hadoop/.staging/job_1669542853305_0003
2022-11-27 22:12:20,543 INFO input.FileInputFormat: Total input files to proces
s : 1
2022-11-27 22:12:20,750 INFO mapreduce.JobSubmitter: number of splits:1
2022-11-27 22:12:21,559 INFO mapreduce.JobSubmitter: Submitting tokens for job:
 job_1669542853305_0003
2022-11-27 22:12:21,559 INFO mapreduce.JobSubmitter: Executing with tokens: []
2022-11-27 22:12:22,129 INFO conf.Configuration: resource-types.xml not found
2022-11-27 22:12:22,129 INFO resource.ResourceUtils: Unable to find 'resource-t
ypes.xml'.
2022-11-27 22:12:27,063 INFO impl.YarnClientImpl: Submitted application applica
tion_1669542853305_0003
2022-11-27 22:12:27,610 INFO mapreduce.Job: The url to track the job: http://sa
nkamethra-VirtualBox:8088/proxy/application_1669542853305_0003/
2022-11-27 22:12:27,620 INFO mapreduce.Job: Running job: job_1669542853305_0003
```



9.Finally,the following commands are used for compiling the WordCount.java program.

COMMANDS
javac -classpath ${HADOOP_CLASSPATH} -d ‘/home/hadoop/wordcount/classes’ ‘/home/hadoop/wordcount/WordCount.java’

OUTPUT:


Home / wordcount / classes
Recent
Starred
Home
Documents
Downloads
Music
Pictures
Videos
Trash
VBox_GAs_7....
Other Locations
WordCount .class
WordCount $Map.class
WordCount $Reduce. class

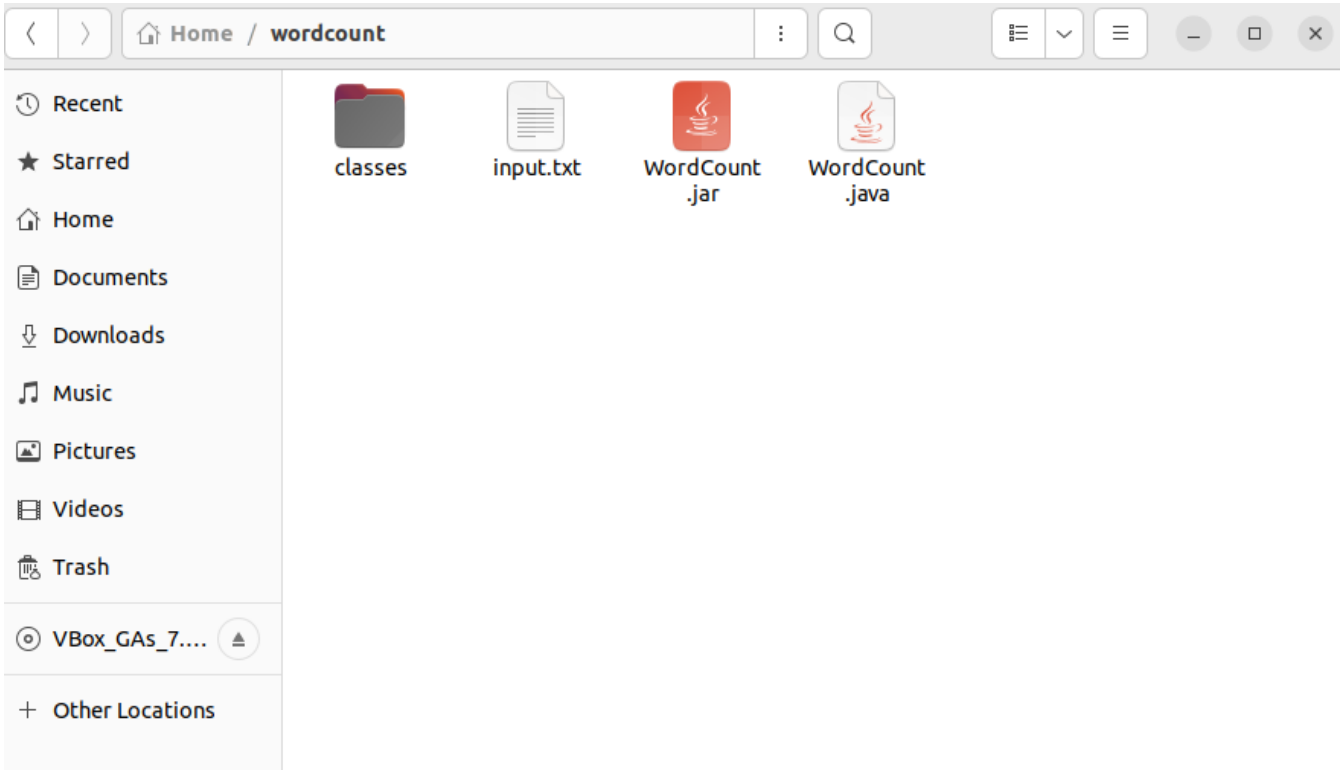
10. Create a jar file using the command

COMMANDS
jar -cvf WordCount.jar -C '/home/hadoop/wordcount/classes'/ .

TERMINAL:

```
added manifest
adding: WordCount$Map.class(in = 1673) (out= 702)(deflated 58%)
adding: WordCount.class(in = 1831) (out= 919)(deflated 49%)
adding: WordCount$Reduce.class(in = 1643) (out= 697)(deflated 57%)
```

OUTPUT:



11. Run MapReduceClient.jar and also provide input and out directories

COMMANDS
hadoop jar 'home/hadoop/wordcount/WordCount.jar' Wordcount /wordcount/input /wordcount/output

TERMINAL:

```
rdCount.jar WordCount /wordcount/input /wordcount/output
2022-11-27 22:12:19,403 INFO client.DefaultNoHARMFailoverProxyProvider: Connect
ing to ResourceManager at /0.0.0.0:8032
2022-11-27 22:12:19,949 WARN mapreduce.JobResourceUploader: Hadoop command-line
 option parsing not performed. Implement the Tool interface and execute your ap
plication with ToolRunner to remedy this.
2022-11-27 22:12:20,015 INFO mapreduce.JobResourceUploader: Disabling Erasure C
oding for path: /tmp/hadoop-yarn/staging/hadoop/.staging/job_1669542853305_0003
2022-11-27 22:12:20,543 INFO input.FileInputFormat: Total input files to proces
s : 1
2022-11-27 22:12:20,750 INFO mapreduce.JobSubmitter: number of splits:1
2022-11-27 22:12:21,559 INFO mapreduce.JobSubmitter: Submitting tokens for job:
 job_1669542853305_0003
2022-11-27 22:12:21,559 INFO mapreduce.JobSubmitter: Executing with tokens: []
2022-11-27 22:12:22,129 INFO conf.Configuration: resource-types.xml not found
2022-11-27 22:12:22,129 INFO resource.ResourceUtils: Unable to find 'resource-t
ypes.xml'.
2022-11-27 22:12:27,063 INFO impl.YarnClientImpl: Submitted application applica
tion_1669542853305_0003
2022-11-27 22:12:27,610 INFO mapreduce.Job: The url to track the job: http://sa
nkamethra-VirtualBox:8088/proxy/application_1669542853305_0003/
2022-11-27 22:12:27,620 INFO mapreduce.Job: Running job: job_1669542853305_0003
```

```
        File System Counters
                FILE: Number of bytes read=6948
                FILE: Number of bytes written=1268118
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=268
                HDFS: Number of bytes written=168
                HDFS: Number of read operations=15
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=6
                HDFS: Number of bytes read erasure-coded=0
        Map-Reduce Framework
                Map input records=19
                Map output records=18
                Map output bytes=204
                Map output materialized bytes=246
                Input split bytes=110
                Combine input records=0
                Combine output records=0
                Reduce input groups=18
                Reduce shuffle bytes=246
                Reduce input records=18
                Reduce output records=18
```

```
2022-11-06 14:47:59,609 INFO mapreduce.Job:  map 0% reduce 0%
2022-11-06 14:48:03,681 INFO mapreduce.Job:  map 100% reduce 0%
2022-11-06 14:48:08,724 INFO mapreduce.Job:  map 100% reduce 100%
2022-11-06 14:48:09,771 INFO mapreduce.Job: Job job_1667723784568_0003 completed successf
ully
2022-11-06 14:48:09,876 INFO mapreduce.Job: Counters: 54
        File System Counters
                FILE: Number of bytes read=148
                FILE: Number of bytes written=529075
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=180
                HDFS: Number of bytes written=75
                HDFS: Number of read operations=8
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=2
                HDFS: Number of bytes read erasure-coded=0
        Job Counters
                Launched map tasks=1
```

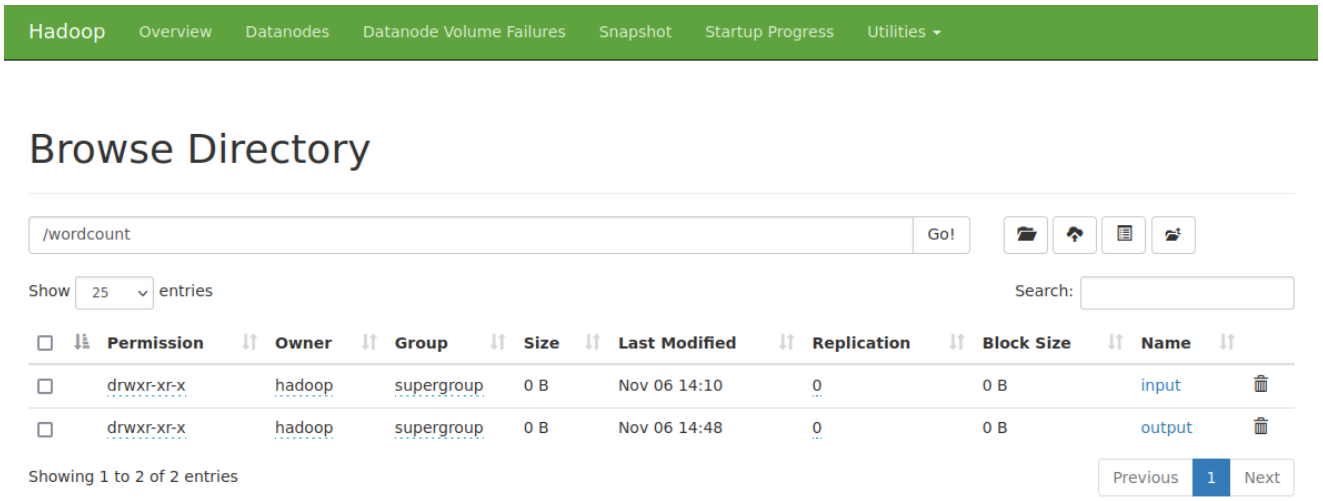
```
            Launched map tasks=1
            Launched reduce tasks=1
            Data-local map tasks=1
            Total time spent by all maps in occupied slots (ms)=2281
            Total time spent by all reduces in occupied slots (ms)=2228
            Total time spent by all map tasks (ms)=2281
            Total time spent by all reduce tasks (ms)=2228
            Total vcore-milliseconds taken by all map tasks=2281
            Total vcore-milliseconds taken by all reduce tasks=2228
            Total megabyte-milliseconds taken by all map tasks=2335744
            Total megabyte-milliseconds taken by all reduce tasks=2281472
    Map-Reduce Framework
            Map input records=13
            Map output records=12
            Map output bytes=118
            Map output materialized bytes=148
            Input split bytes=109
            Combine input records=0
            Combine output records=0
            Reduce input groups=9
            Reduce shuffle bytes=148
            Reduce input records=12
            Reduce output records=9
            Spilled Records=24
            Shuffled Maps =1
            Failed Shuffles=0
```

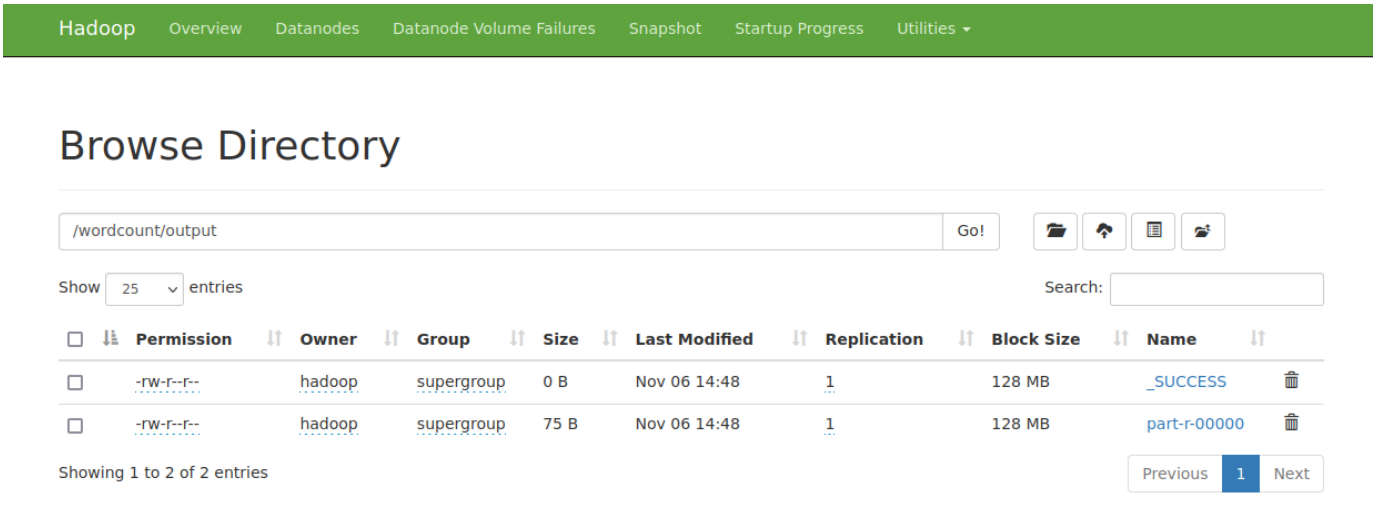
```
            Merged Map outputs=1
            GC time elapsed (ms)=41
            CPU time spent (ms)=1230
            Physical memory (bytes) snapshot=465485824
            Virtual memory (bytes) snapshot=5497430016
            Total committed heap usage (bytes)=334495744
            Peak Map Physical memory (bytes)=272846848
            Peak Map Virtual memory (bytes)=2746839040
            Peak Reduce Physical memory (bytes)=192638976
            Peak Reduce Virtual memory (bytes)=2750590976
    Shuffle Errors
            BAD_ID=0
            CONNECTION=0
            IO_ERROR=0
            WRONG_LENGTH=0
            WRONG_MAP=0
            WRONG_REDUCE=0
    File Input Format Counters
            Bytes Read=71
    File Output Format Counters
            Bytes Written=75
```

12. VERIFY THE OUTPUT FOLDER IS CREATED OR NOT:

Check the output in the Web UI at http://localhost:9870

In the Utilities tab select browse file system and select the correct user. The output is available inside the output folder named user.





13. PRINT THE CONTENTS OF THE OUTPUT FILE/VERIFY CONTENT FOR GENERATED OUTPUT FILE.

COMMANDS
*hdfs dfs -car /wordcount/output/**

TERMINAL:

RESULT:

Thus the numbers of words were counted successfully by the use of Map and Reduce tasks.

Experiment 4: *Implementation of Matrix Multiplication using MapReduce*

a)OBJECTIVE OF THE EXERCISE/EXPERIMENT

- *Implementation of Matrix Multiplication using MapReduce.*

b)Facilities/material required to do the exercise/experiment:

Sl.No.	*Facilities/material required*	*Quantity*
1.	*Hadoop*	*As per strength*

Source Code:

```
import
java.io.IOException;
import java.util.*;
import
java.util.AbstractMap.SimpleEntry;
import java.util.Map.Entry;
import
org.apache.hadoop.fs.Path;
import
org.apache.hadoop.conf.*;
import
org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;
import
org.apache.hadoop.mapreduce.lib.input.FileInputFormat
;                                                    import
org.apache.hadoop.mapreduce.lib.input.TextInputForma
t;

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

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

public class TwoStepMatrixMultiplication {
public static class Map extends Mapper<LongWritable,
Text, Text, Text> {
 public void map(LongWritable key, Text value,
Context context) throws IOException,
InterruptedException { String line =
value.toString();
 String[] indicesAndValue =
```

```
line.split(","); Text outputKey = new
Text();
Text outputValue = new Text();
if (indicesAndValue[0].equals("A")) {
outputKey.set(indicesAndValue[2]);
outputValue.set("A," + indicesAndValue[1] + ","
+ indicesAndValue[3]);
context.write(outputKey, outputValue);
} else {
outputKey.set(indicesAndValue[
1]);
outputValue.set("B," + indicesAndValue[2] +
"," + indicesAndValue[3]);
context.write(outputKey, outputValue);
}
}
}
public static class Reduce extends Reducer<Text, Text,
Text, Text> {
public void reduce(Text key, Iterable<Text> values,
```

```
Context context) throws IOException,
 InterruptedException { String[] value;
 ArrayList<Entry<Integer, Float>> listA
= new ArrayList<Entry<Integer,
Float>>(); ArrayList<Entry<Integer,
Float>> listB = new
ArrayList<Entry<Integer, Float>>();
 for (Text val : values) {
 value = val.toString().split(",");
 if (value[0].equals("A")) {
 listA.add(new
 SimpleEntry<Integer,
Float>(Integer.parseInt(value[1]), Float.parseFloat(value[2])));
 } else {
 listB.add(new SimpleEntry<Integer,
Float>(Integer.parseInt(value[1]),
Float.parseFloat(value[2])));
 }
 }
 String i;
 float
 a_ij;
 String k;
 float
 b_jk;
 Text outputValue = new Text();
 for (Entry<Integer, Float> a :
 listA) { i =
 Integer.toString(a.getKey());
 a_ij = a.getValue();
```

```
for (Entry<Integer, Float> b :
listB) { k =
Integer.toString(b.getKey());
b_jk = b.getValue();
```

```
 outputValue.set(i + "," + k +
"," +
Float.toString(a_ij*b_jk));
context.write(null,
outputValue);
 }
 }
 }
 }
 public static void main(String[] args) throws
 Exception { Configuration conf = new
 Configuration();
 Job job = new Job(conf,
"MatrixMatrixMultiplicationTwoSteps");
job.setJarByClass(TwoStepMatrixMultiplication.class);
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(Text.class);
job.setMapperClass(Map.class);
job.setReducerClass(Reduce.class);
job.setInputFormatClass(TextInputFormat.class);
job.setOutputFormatClass(TextOutputFormat.class);
FileInputFormat.addInputPath(job, new Path("hdfs://
127.0.0.1:9000/matrixin"));
 FileOutputFormat.setOutputPath(job
, new
Path("hdfs://127.0.0.1:9000/matrixo
ut"));
 job.waitForCompletion(true);
 }
}
```

SCREENSHOT:

```
import java.io.IOException;
import java.util.*;
import java.util.AbstractMap.SimpleEntry;
import java.util.Map.Entry;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

public class TwoStepMatrixMultiplication {

    public static class Map extends Mapper<LongWritable, Text,
Text, Text> {
        public void map(LongWritable key, Text value, Context
context) throws IOException, InterruptedException {
            String line = value.toString();
            String[] indicesAndValue = line.split(",");
            Text outputKey = new Text();
            Text outputValue = new Text();
            if (indicesAndValue[0].equals("A")) {
                outputKey.set(indicesAndValue[2]);
                outputValue.set("A," + indicesAndValue[1] + "," +
indicesAndValue[3]);
                context.write(outputKey, outputValue);
            } else {
                outputKey.set(indicesAndValue[1]);
                outputValue.set("B," + indicesAndValue[2] + "," +
indicesAndValue[3]);
                context.write(outputKey, outputValue);
            }
        }
    }

    public static class Reduce extends Reducer<Text, Text, Text,
Text> {
        public void reduce(Text key, Iterable<Text> values,
Context context) throws IOException, InterruptedException {
            String[] value;
            ArrayList<Entry<Integer, Float>> listA = new

Float>(Integer.parseInt(value[1]), Float.parseFloat(value[2])));
                } else {
                    listB.add(new SimpleEntry<Integer,
Float>(Integer.parseInt(value[1]), Float.parseFloat(value[2])));
                }
            }
            String i;
            float a_ij;
            String k;
            float b_jk;
            Text outputValue = new Text();
            for (Entry<Integer, Float> a : listA) {
                i = Integer.toString(a.getKey());
                a_ij = a.getValue();
                for (Entry<Integer, Float> b : listB) {
                    k = Integer.toString(b.getKey());
                    b_jk = b.getValue();
                    outputValue.set(i + "," + k + "," +
Float.toString(a_ij*b_jk));
                    context.write(null, outputValue);
                }
            }
        }
    }

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

        Job job = new Job(conf, "MatrixMatrixMultiplicationTwoSteps");
        job.setJarByClass(TwoStepMatrixMultiplication.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(Text.class);

        job.setMapperClass(Map.class);
        job.setReducerClass(Reduce.class);

        job.setInputFormatClass(TextInputFormat.class);
        job.setOutputFormatClass(TextOutputFormat.class);
 FileInputFormat.addInputPath(job, new Path(args[0]));
FileOutputFormat.setOutputPath(job, new Path(args[1]));

        job.waitForCompletion(true);
    }
```

Steps to execute MapReduce word count example

1.Set up the path and configure the java environment:

COMMANDS
1. export HADOOP_CLASSPATH=$(hadoop classpath) *2.echo $HADOOP_CLASSPATH*

OUTPUT:

```
/usr/local/hadoop/etc/hadoop:/usr/local/hadoop/share/hadoop/common/lib/*:/usr/l
ocal/hadoop/share/hadoop/common/*:/usr/local/hadoop/share/hadoop/hdfs:/usr/loca
l/hadoop/share/hadoop/hdfs/lib/*:/usr/local/hadoop/share/hadoop/hdfs/*:/usr/loc
al/hadoop/share/hadoop/mapreduce/*:/usr/local/hadoop/share/hadoop/yarn:/usr/loc
al/hadoop/share/hadoop/yarn/lib/*:/usr/local/hadoop/share/hadoop/yarn/*:/usr/lo
cal/hadoop/lib/javax.activation-api-1.2.0.jar
```

2.Open cmd in Ubuntu terminal and start a cluster.

COMMANDS
Start-all.sh

TERMINAL:

```
WARNING: Attempting to start all Apache Hadoop daemons as hadoop in 10 seconds.
WARNING: This is not a recommended production deployment configuration.
WARNING: Use CTRL-C to abort.
Starting namenodes on [swetha]
Starting datanodes
Starting secondary namenodes [swetha]
Starting resourcemanager
Starting nodemanagers
```

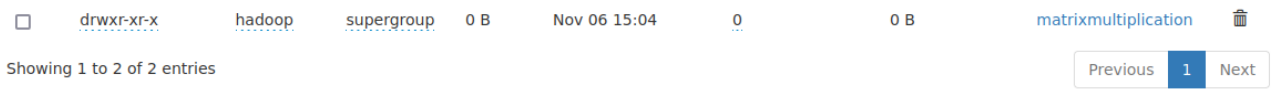
3.Verify the what are the processes running and list the process id

COMMANDS
Jps

4. Create an input directory in HDFS.

COMMANDS
hdfs dfs -mkdir / matrixmultiplication/

Output:



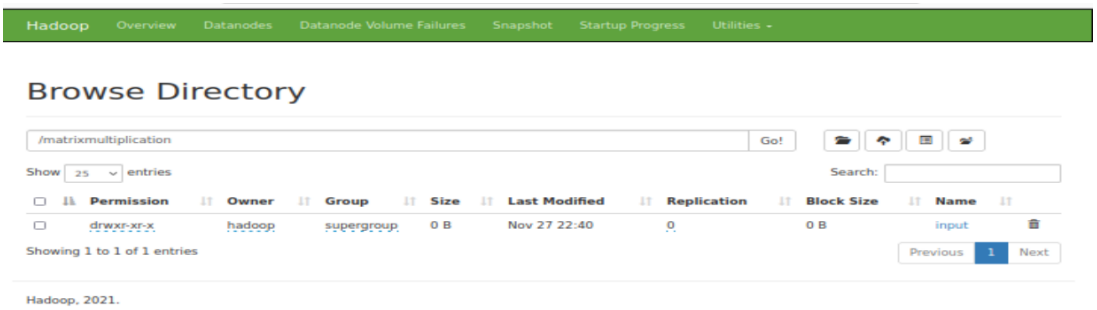
5.Create a input folder inside of matrixmultiplication directory.

COMMANDS
hdfs dfs -mkdir / matrixmultiplication /input

6.Copy the 2 input text file in the input directory (matrixmultiplication) of HDFS.

Copy some text file to hadoop filesystem inside input directory. Here I am copying a input.txt to it. You can copy more that one files.

COMMANDS
hdfs dfs -put '/home/hadoop/twostepmatrix/input.txt' / matrixmultiplication /input/



7. Verify content of the copied file.

COMMANDS
*hdfs dfs -cat / matrixmultiplication /input/**

```
A,0,0,6
A,0,0,1
A,1,0,7
A,1,1,4
B,1,2,3
B,2,1,3
B,0,1,0
B,0,1,2
```

8.Move into the twostepmatrix folder in home which contains the java file

COMMANDS
cd /home/hadoop/ twostepmatrix

and whether check and print the working directory and list the contents of the file.

TERMINAL:

```
hadoop@swetha:~$ cd /home/hadoop/twostepmatrix
hadoop@swetha:~/twostepmatrix$ pwd
/home/hadoop/twostepmatrix
hadoop@swetha:~/twostepmatrix$ ls
classes  input.txt  TwoStepMatrixMultiplication.java
hadoop@swetha:~/twostepmatrix$
```

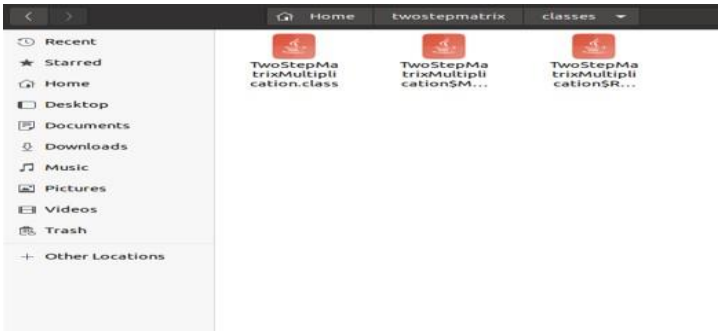
9.Finally,the following commands are used for compiling the TwoStepMatrixMultiplication.java program.

COMMANDS
javac -classpath ${HADOOP_CLASSPATH} -d '/home/hadoop/ twostepmatrix /classes' '/home/hadoop/ twostepmatrix / TwoStepMatrixMultiplication.java'

TERMINAL:

```
Note: /home/hadoop/twostepmatrix/TwoStepMatrixMultiplication.java uses or overri
des a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
```

OUTPUT:



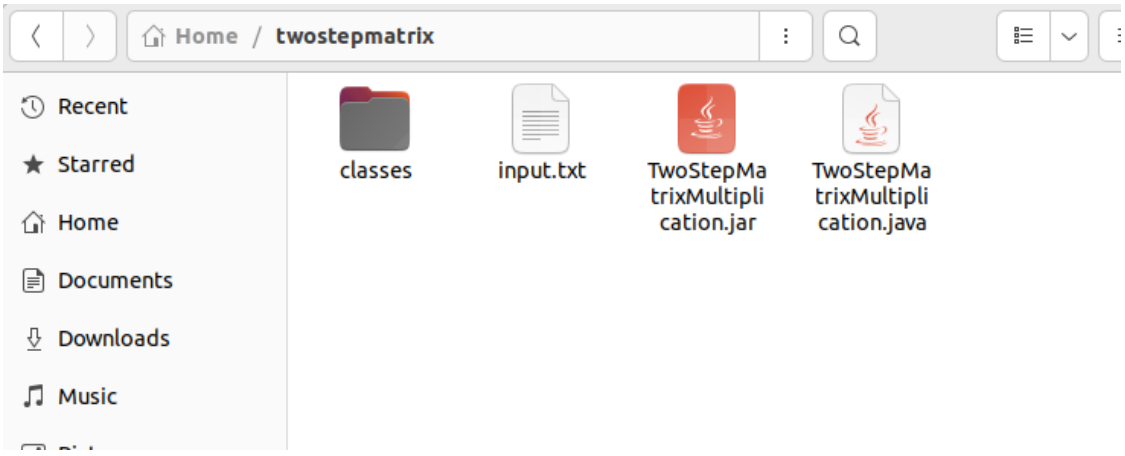
10. Create a jar file using the command

COMMANDS
jar -cvf TwoStepMatrixMultiplication.jar -C '/home/hadoop/ twostepmatrix /classes'/ .

TERMINAL:

```
oop/twostepmatrix/classes'/ .
added manifest
adding: TwoStepMatrixMultiplication$Reduce.class(in = 3419) (out= 1485)(deflated 56%)
adding: TwoStepMatrixMultiplication$Map.class(in = 2292) (out= 939)(deflated 59%)
adding: TwoStepMatrixMultiplication.class(in = 1564) (out= 768)(deflated 50%)
```

OUTPUT:



11. Run MapReduceClient.jar and also provide input and out directories

COMMANDS
hadoop jar 'home/hadoop/ twostepmatrix / TwoStepMatrixMultiplication.jar' TwoStepMatrixMultiplication / matrixmultiplication/input / matrixmultiplication/ output

TERMINAL

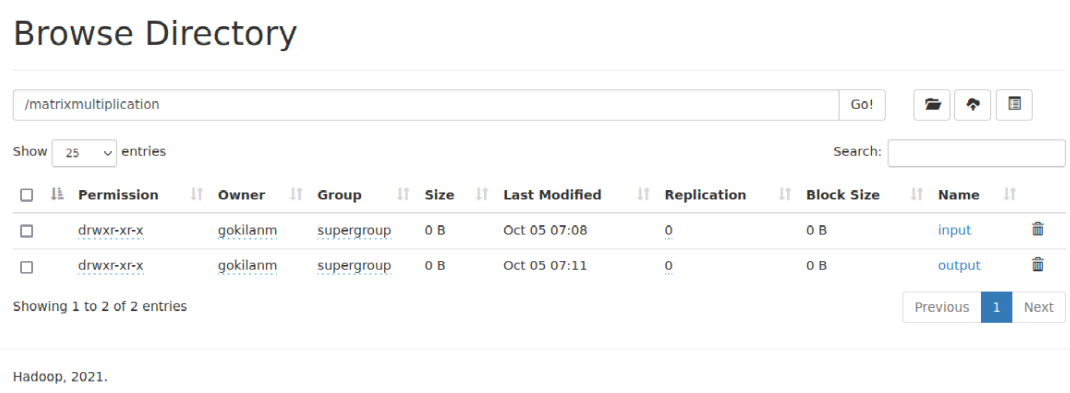
```
gokilanm@gokila-VirtualBox:~/twostepmatrix$ hadoop jar '/home/gokilanm/twostepmatrix/TwoStepMatrixMultiplication.jar' TwoStepMatrixMultiplication  /ma
trixmultiplication/input /matrixmultiplication/output
2021-10-05 07:11:18,200 INFO client.RMProxy: Connecting to ResourceManager at /127.0.0.1:8032
2021-10-05 07:11:18,690 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool interface and execute
 your application with ToolRunner to remedy this.
2021-10-05 07:11:18,727 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/hadoop-yarn/staging/gokilanm/.staging/job_16333960
55652_0002
2021-10-05 07:11:19,153 INFO input.FileInputFormat: Total input files to process : 2
2021-10-05 07:11:19,285 INFO mapreduce.JobSubmitter: number of splits:2
2021-10-05 07:11:19,566 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1633396055652_0002
2021-10-05 07:11:19,568 INFO mapreduce.JobSubmitter: Executing with tokens: []
2021-10-05 07:11:19,866 INFO conf.Configuration: resource-types.xml not found
2021-10-05 07:11:19,871 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.
2021-10-05 07:11:19,990 INFO impl.YarnClientImpl: Submitted application application_1633396055652_0002
2021-10-05 07:11:20,082 INFO mapreduce.Job: The url to track the job: http://gokila-VirtualBox:8088/proxy/application_1633396055652_0002/
2021-10-05 07:11:20,083 INFO mapreduce.Job: Running job: job_1633396055652_0002
2021-10-05 07:11:28,309 INFO mapreduce.Job: Job job_1633396055652_0002 running in uber mode : false
2021-10-05 07:11:28,311 INFO mapreduce.Job:  map 0% reduce 0%
2021-10-05 07:11:38,503 INFO mapreduce.Job:  map 100% reduce 0%
2021-10-05 07:11:44,574 INFO mapreduce.Job:  map 100% reduce 100%
2021-10-05 07:11:44,599 INFO mapreduce.Job: Job job_1633396055652_0002 completed successfully
2021-10-05 07:11:44,750 INFO mapreduce.Job: Counters: 55
        File System Counters
                FILE: Number of bytes read=86
                FILE: Number of bytes written=703911
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=311
                HDFS: Number of bytes written=68
                HDFS: Number of read operations=11
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=2
                HDFS: Number of bytes read erasure-coded=0
        Job Counters
                Killed map tasks=1
                Launched map tasks=2
                Launched reduce tasks=1
                Data-local map tasks=2
                Total time spent by all maps in occupied slots (ms)=14803
                Total time spent by all reduces in occupied slots (ms)=3362
                Total time spent by all map tasks (ms)=14803
                Total time spent by all reduce tasks (ms)=3362
                Total vcore-milliseconds taken by all map tasks=14803
                Total vcore-milliseconds taken by all reduce tasks=3362
```

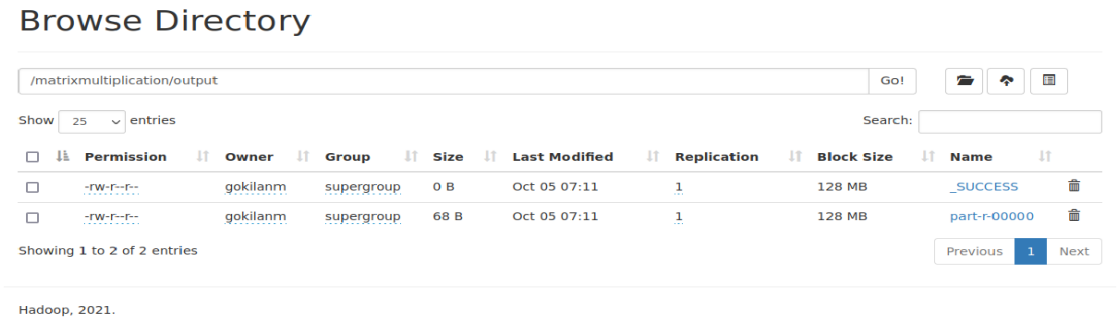
```
                Map input records=8
                Map output records=8
                Map output bytes=64
                Map output materialized bytes=92
                Input split bytes=247
                Combine input records=0
                Combine output records=0
                Reduce input groups=2
                Reduce shuffle bytes=92
                Reduce input records=8
                Reduce output records=8
                Spilled Records=16
                Shuffled Maps =2
                Failed Shuffles=0
                Merged Map outputs=2
                GC time elapsed (ms)=280
                CPU time spent (ms)=1980
                Physical memory (bytes) snapshot=589918208
                Virtual memory (bytes) snapshot=7448391680
                Total committed heap usage (bytes)=441131008
                Peak Map Physical memory (bytes)=230961152
                Peak Map Virtual memory (bytes)=2479849472
                Peak Reduce Physical memory (bytes)=128348160
                Peak Reduce Virtual memory (bytes)=2488692736
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=64
        File Output Format Counters
                Bytes Written=68
```

12. VERIFY THE OUTPUT FOLDER IS CREATED OR NOT:

Check the output in the Web UI at http://localhost:9870

In the Utilities tab select browse file system and select the correct user. The output is available inside the output folder named user.





13. PRINT THE CONTENTS OF THE OUTPUT FILE/VERIFY CONTENT FOR GENERATED OUTPUT FILE.

COMMANDS
*hdfs dfs -cat / matrixmultiplication /output/**

TERMINAL:

```
gokilanm@gokila-VirtualBox:~/twostepmatrix$ hdfs dfs -cat /matrixmultiplication/output/*
1,1,4.0
1,0,8.0
0,1,6.0
0,0,12.0
1,1,9.0
1,0,12.0
0,1,21.0
0,0,28.0
gokilanm@gokila-VirtualBox:~/twostepmatrix$
```

RESULT:

Thus the multiplication program were executed successfully by the use of Map and Reduce tasks.

Experiment 5: *Implement a Map Reduce Program to analyse time-temperature statistics and generate report with max/min temperature*

a) *OBJECTIVE OF THE EXERCISE/EXPERIMENT*

- *To implement a Map Reduce Program to analyse time-temperature statistics and generate report with max/min temperature*

b) *Facilities/material required to do the exercise/experiment:*

Sl.No.	*Facilities/material required*	*Quantity*
1.	*Hadoop*	*As per strength*

PROBLEM STATEMENT:

1. The system receives temperatures of various cities(Austin, Boston,etc) of USA captured at regular intervals of time on each day in an input file.

2. System will process the input data file and generates a report with Maximum and Minimum temperatures of each day along with time.

3. Generates a separate output report for

each city. Ex: Austin-r-00000

Boston-r-00000

Newjersy-r-

00000

Baltimore-r-

00000

California-r-

00000

Newyork-r-

00000

THEORY:

The very first thing which is required for any map reduce problem is to understand what will be the type of keyIn, ValueIn, KeyOut,ValueOut for the given Mapper class and followed by type of map method parameters.

- *public class WhetherForcastMapper extends Mapper <Object, Text, Text, Text>*
- *Object (keyIn) - Offset for each line, line number 1, 2...*
- *Text (ValueIn) - Whole string for each line (CA_25-Jan-2014 00:12:345)*
- *Text (KeyOut) - City information with date information as string*
- *Text (ValueOut) - Temperature and time information which need to be passed to reducer as string.*
- *public void map(Object keyOffset, Text dayReport, Context con) { }*
- *KeyOffset is like line number for each line in input file.*
- *dayreport is input to map method - whole string present in one line of input file.*
- *con is context where we write mapper output and it is used by reducer. Reducer class and reducer method:-*

Similarly,we have to decide what will be the type of keyIn, ValueIn, KeyOut,ValueOut for the given Reducer class and followed by type of reducer method parameters.

- *public class WhetherForcastReducer extends Reducer<Text, Text, Text, Text>*
- *Text(keyIn) - it is same as keyOut of Mapper.*

- *Text(ValueIn)- it is same as valueOut of Mapper.*
- *Text(KeyOut)- date as string*
- *text(ValueOut) - reducer writes max and min temperature with time as string*
- *public void reduce(Text key, Iterable<Text> values, Context context)*

• Text key is value of mapper output. i.e:- City & date information

• Iterable<Text> values - values stores multiple temperature values for a given city and date. context object is where reducer write it's processed outcome and finally written in file.

MultipleOutputs:-

In general, reducer generates output file(i.e: part_r_0000), however in this use case we want to generate multiple output files. In order to deal with such scenario we need to use MultipleOutputs of "org.apache.hadoop.mapreduce.lib.output.MultipleOutputs" which provides a way to write multiple file depending on reducer outcome. For each reducer task multipleoutput object is created and key/result is written to appropriate file.

SOURCE CODE:

```
import java.io.IOException;

import

java.util.StringTokenizer;

import

org.apache.hadoop.io.Text;

import

org.apache.hadoop.mapreduce.Mapper;

import

org.apache.hadoop.mapreduce.Reducer;

import

org.apache.hadoop.mapreduce.lib.output.MultipleOutputs

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

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.mapreduce.Job;

import

org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

import

org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import
```

```
org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
public class CalculateMaxAndMinTemeratureWithTime {
public static String calOutputName
="California"; public static String
nyOutputName = "Newyork"; public static
String njOutputName = "Newjersy"; public
static String ausOutputName = "Austin";
public static String bosOutputName =
"Boston";
```

```
public static String balOutputName
="Baltimore"; public static class
WhetherForcastMapper extends
Mapper<Object, Text, Text, Text> {
public void map(Object keyOffset, Text dayReport, Context
con)throws IOException, InterruptedException {
 StringTokenizer strTokens =
new StringTokenizer(
dayReport.toString(), "\t");
 int counter = 0;
 Float currnetTemp = null;
 Float minTemp =
 Float.MAX_VALUE; Float
 maxTemp = Float.MIN_VALUE;
 String date = null;
 String currentTime = null;
 String minTempANDTime = null;
 String maxTempANDTime = null;
 while
 (strTokens.hasMoreElements()) { if
 (counter == 0) {
 date = strTokens.nextToken();
 } else {
 if (counter % 2 == 1) {
 currentTime = strTokens.nextToken();
 } else {
 currnetTemp
 =Float.parseFloat(strTokens.nextToken()); if
 (minTemp > currnetTemp) {
 minTemp = currnetTemp;
```

```
minTempANDTime = minTemp + "AND"
+ currentTime;
```

```
}
if (maxTemp <
currnetTemp) { maxTemp =
currnetTemp;
maxTempANDTime = maxTemp +
"AND" + currentTime;
}
}
}
counter++;
}
// Write to context - MinTemp,
MaxTemp and corresponding time
Text temp = new Text();
temp.set(maxTempANDTime)
; Text dateText = new Text();
dateText.set(date);
try {
con.write(dateText, temp);
} catch (Exception e)
{
e.printStackTrace();
}
temp.set(minTempANDTim
e); dateText.set(date);
con.write(dateText, temp);
}
}
public static class WhetherForcastReducer extends Reducer<Text,
Text, Text, Text> {
```

```
 MultipleOutputs<Text, Text>
mos; public void setup(Context
context) { mos = new
MultipleOutputs<Text,
Text>(context);
}
public void reduce(Text key,
Iterable<Text> values, Context context)
 throws IOException,
 InterruptedException { int counter = 0;
 String reducerInputStr[] =
 null; String f1Time = "";
 String f2Time = "";
 String f1 = "", f2 = "";
 Text result = new
 Text(); for (Text value
 : values) { if (counter
 == 0) {
 reducerInputStr =
value.toString().split("AN
 D"); f1 =
 reducerInputStr[0];
 f1Time =
 reducerInputStr[1];
 }
 else {
 reducerInputSt
 r =
value.toString().split("AN
```

```
D"); f2 =
reducerInputStr[0];
f2Time =
reducerInputStr[1];
}
```

```
 counter = counter + 1;
 }
 if
(Float.parseFloat(f1) >
Float.parseFloat(f2)) {
 result = new Text("Time: " + f2Time + " MinTemp: " + f2 + "\t" +
"Time: " + f1Time + " MaxTemp: " + f1);
 }
else {

 result = new Text("Time: " + f1Time + " MinTemp: " + f1 + "\t"+
"Time: " + f2Time + " MaxTemp: " + f2);
 }
 String fileName = "";
 if (key.toString().substring(0, 2).equals("CA"))
{
 fileName =CalculateMaxAndMinTemeratureTime.calOutputName;
 } else if
(key.toString().substring(0,
2).equals("NY")) {
 fileName =CalculateMaxAndMinTemeratureTime.nyOutputName;
 } else if
(key.toString().substring(0,
2).equals("NJ")) {
 fileName =CalculateMaxAndMinTemeratureTime.njOutputName;
 } else if
(key.toString().substring(0,
3).equals("AUS")) {
 fileName =CalculateMaxAndMinTemeratureTime.ausOutputName;
 } else if
```

```
(key.toString().substring(0,
3).equals("BOS")) {
 fileName =CalculateMaxAndMinTemeratureTime.bosOutputName;
```

```
 } else if
(key.toString().substring(0,
3).equals("BAL")) {
 fileName =CalculateMaxAndMinTemeratureTime.balOutputName;
 }
 String strArr[] =
 key.toString().split("_");
 key.set(strArr[1]); //Key is date
 value mos.write(fileName, key,
 result);
}
@Override
public void cleanup(Context context)
throws IOException,
 InterruptedExceptio
 n { mos.close();
}
}
public static void main(String[] args)
throws IOException,
 ClassNotFoundExcepti
on,
InterruptedException {
 Configuration conf = new
Configuration(); Job job =
Job.getInstance(conf, "Wheather
Statistics of USA");
job.setJarByClass(CalculateMaxAndMinTemerat
ureW ithTime.class);
```

```
job.setMapperClass(WhetherForcastMapper.clas
s);
job.setReducerClass(WhetherForcastReducer.cla
ss); job.setMapOutputKeyClass(Text.class);
```

```
job.setMapOutputValueClass(Text.cl
ass);
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(Text.class);
MultipleOutputs.addNamedOutput(job,calOutputName,TextOutputFor
mat.class
, Text.class,Text.class);
MultipleOutputs.addNamedOutput(job,nyOutputName,TextOutputFormat.clas
s, Text.class,Text.class);
MultipleOutputs.addNamedOutput(job,njOutputName,TextOutputFormat.clas
s, Text.class,Text.class);

MultipleOutputs.addNamedOutput(job,bosOutputName,TextOutputFo
rmat.clas s, Text.class,Text.class);

MultipleOutputs.addNamedOutput(job,ausOutputName,TextOutputFo
rmat.clas s, Text.class,Text.class);
MultipleOutputs.addNamedOutput(job,balOutputName,TextOutputFormat.cla
ss
, Text.class,Text.class);
// FileInputFormat.addInputPath(job, new Path(args[0]));
// FileOutputFormat.setOutputPath(job, new Path(args[1]));
Path pathInput = new
Path("hdfs://192.168.213.133:54310/weatherInputData/
input_temp.txt");
Path pathOutputDir = new Path(
"hdfs://192.168.213.133:54310/user/hduser1/ testfs/output_mapred3");
FileInputFormat.addInputPath(job, pathInput);
FileOutputFormat.setOutputPath(job,pathOutpu
tDir); try {
System.exit(job.waitForCompletion(true) ? 0 :1);
} catch (Exception e) {
// TODO Auto-generated catch
```

block e.printStackTrace();

}

}

}

EXPLANATION:

In map method, we are parsing each input line and maintains a counter for extracting date and each temperature & time information.For a given input line, first extract date(counter ==0) and followed by alternatively extract time(counter%2==1) since time is on odd number position like (1,3,5 .....) and

get temperature otherwise. Compare for max & min temperature and store it accordingly. Once while loop terminates for a given input line, write maxTempTime and minTempTime with date.

In reduce method, for each reducer task, setup method is executed and create MultipleOutput object. For a given key, we have two entry (maxtempANDTime and mintempANDTime). Iterate values list , split value and get temperature & time value. Compare temperature value and create actual value sting which reducer write in appropriate file.

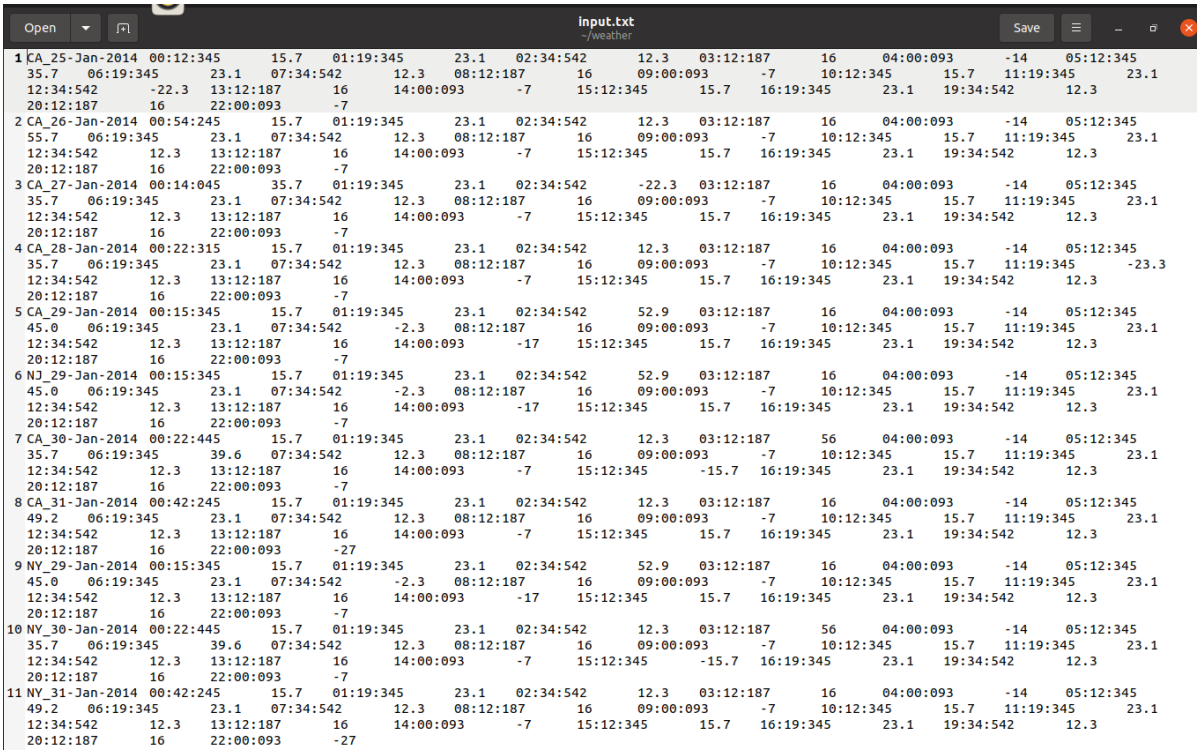
In main method,a instance of Job is created with Configuration object. Job is configured with mapper, reducer class and along with input and output format. MultipleOutputs information added to Job to indicate file name to be used with input format. For this sample program, we are using input file("/ weatherInputData/input_temp.txt") placed on HDFS and output directory (/ user/hduser1/testfs/output_mapred5) will be also created on HDFS. Refer below command to copy downloaded input file from local file system to HDFS and give write permission to client who is executing this program unit so that output directory can be

created.

SNAPSHOT:

```
import java.io.IOException;
import java.util.StringTokenizer;
import java.text.DecimalFormat;

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

public class MinMaxTemperature {
public static String calOutputName = "California";
public static String nyOutputName = "Newyork";
public static String njOutputName = "Newjersy";
public static String ausOutputName = "Austin";
public static String bosOutputName = "Boston";
public static String balOutputName = "Baltimore";

public static class WhetherForcastMapper extends Mapper<Object, Text, Text, Text> {
 public void map(Object keyOffset, Text dayReport, Context con) throws IOException, InterruptedException {
  //StringTokenizer strTokens = new StringTokenizer(dayReport.toString(), "\t");
  StringTokenizer strTokens = new StringTokenizer(dayReport.toString(),"\t");
  int counter = 0;
  Float currnetTemp = null;
  Float minTemp = Float.MAX_VALUE;
  Float maxTemp = Float.MIN_VALUE;
  String date = null;
  String currentTime = null;
  String minTempANDTime = null;
  String maxTempANDTime = null;

  while (strTokens.hasMoreElements()) {
   if (counter == 0) {
    date = strTokens.nextToken();
   } else {
    if (counter % 2 == 1) {
     currentTime = strTokens.nextToken();
    }
    else {
     currnetTemp = Float.parseFloat(strTokens.nextToken());
    /*  DecimalFormat df = new DecimalFormat();
     currnetTemp = df.parse(strTokens.nextToken()).floatValue(); */
     if (minTemp > currnetTemp) {
      minTemp = currnetTemp;
      minTempANDTime = minTemp + "AND" + currentTime;
     }
     if (maxTemp < currnetTemp) {
      maxTemp = currnetTemp;
      maxTempANDTime = maxTemp + "AND" + currentTime;
     }
    }
   }
   counter++;
  }
  // Write to context - MinTemp, MaxTemp and corresponding time
  Text temp = new Text();
  temp.set(maxTempANDTime);
  Text dateText = new Text();
  dateText.set(date);
  try {
   con.write(dateText, temp);
  } catch (Exception e) {
   e.printStackTrace();
  }

  temp.set(minTempANDTime);
  dateText.set(date);
  con.write(dateText, temp);
  }
}

public static class WhetherForcastReducer extends Reducer<Text, Text, Text, Text> {
 MultipleOutputs<Text, Text> mos;

 public void setup(Context context) {
  mos = new MultipleOutputs<Text, Text>(context);
 }

 public void reduce(Text key, Iterable<Text> values, Context context) throws IOException, InterruptedException {
  int counter = 0;
  String reducerInputStr[] = null;
  String f1Time = "";
  String f2Time = "";
```

INPUT FILE:




```
1 CA_25-Jan-2014 00:12:345      15.7    01:19:345       23.1    02:34:542       12.3    03:12:187       16      04:00:093       -14     05:12:345
  35.7    06:19:345       23.1    07:34:542       12.3    08:12:187       16      09:00:093       -7      10:12:345       15.7    11:19:345       23.1
  12:34:542       -22.3   13:12:187       16      14:00:093       -7      15:12:345       15.7    16:19:345       23.1    19:34:542       12.3
  20:12:187       16      22:00:093       -7
2 CA_26-Jan-2014 00:54:245      15.7    01:19:345       23.1    02:34:542       12.3    03:12:187       16      04:00:093       -14     05:12:345
  55.7    06:19:345       23.1    07:34:542       12.3    08:12:187       16      09:00:093       -7      10:12:345       15.7    11:19:345       23.1
  12:34:542       12.3    13:12:187       16      14:00:093       -7      15:12:345       15.7    16:19:345       23.1    19:34:542       12.3
  20:12:187       16      22:00:093       -7
3 CA_27-Jan-2014 00:14:045      35.7    01:19:345       23.1    02:34:542       -22.3   03:12:187       16      04:00:093       -14     05:12:345
  35.7    06:19:345       23.1    07:34:542       12.3    08:12:187       16      09:00:093       -7      10:12:345       15.7    11:19:345       23.1
  12:34:542       12.3    13:12:187       16      14:00:093       -7      15:12:345       15.7    16:19:345       23.1    19:34:542       12.3
  20:12:187       16      22:00:093       -7
4 CA_28-Jan-2014 00:22:315      15.7    01:19:345       23.1    02:34:542       12.3    03:12:187       16      04:00:093       -14     05:12:345
  35.7    06:19:345       23.1    07:34:542       12.3    08:12:187       16      09:00:093       -7      10:12:345       15.7    11:19:345       -23.3
  12:34:542       12.3    13:12:187       16      14:00:093       -7      15:12:345       15.7    16:19:345       23.1    19:34:542       12.3
  20:12:187       16      22:00:093       -7
5 CA_29-Jan-2014 00:15:345      15.7    01:19:345       23.1    02:34:542       52.9    03:12:187       16      04:00:093       -14     05:12:345
  45.0    06:19:345       23.1    07:34:542       -2.3    08:12:187       16      09:00:093       -7      10:12:345       15.7    11:19:345       23.1
  12:34:542       12.3    13:12:187       16      14:00:093       -17     15:12:345       15.7    16:19:345       23.1    19:34:542       12.3
  20:12:187       16      22:00:093       -7
6 NJ_29-Jan-2014 00:15:345      15.7    01:19:345       23.1    02:34:542       52.9    03:12:187       16      04:00:093       -14     05:12:345
  45.0    06:19:345       23.1    07:34:542       -2.3    08:12:187       16      09:00:093       -7      10:12:345       15.7    11:19:345       23.1
  12:34:542       12.3    13:12:187       16      14:00:093       -17     15:12:345       15.7    16:19:345       23.1    19:34:542       12.3
  20:12:187       16      22:00:093       -7
7 CA_30-Jan-2014 00:22:445      15.7    01:19:345       23.1    02:34:542       12.3    03:12:187       56      04:00:093       -14     05:12:345
  35.7    06:19:345       39.6    07:34:542       12.3    08:12:187       16      09:00:093       -7      10:12:345       15.7    11:19:345       23.1
  12:34:542       12.3    13:12:187       16      14:00:093       -7      15:12:345       -15.7   16:19:345       23.1    19:34:542       12.3
  20:12:187       16      22:00:093       -7
8 CA_31-Jan-2014 00:42:245      15.7    01:19:345       23.1    02:34:542       12.3    03:12:187       16      04:00:093       -14     05:12:345
  49.2    06:19:345       23.1    07:34:542       12.3    08:12:187       16      09:00:093       -7      10:12:345       15.7    11:19:345       23.1
  12:34:542       12.3    13:12:187       16      14:00:093       -7      15:12:345       15.7    16:19:345       23.1    19:34:542       12.3
  20:12:187       16      22:00:093       -27
9 NY_29-Jan-2014 00:15:345      15.7    01:19:345       23.1    02:34:542       52.9    03:12:187       16      04:00:093       -14     05:12:345
  45.0    06:19:345       23.1    07:34:542       -2.3    08:12:187       16      09:00:093       -7      10:12:345       15.7    11:19:345       23.1
  12:34:542       12.3    13:12:187       16      14:00:093       -17     15:12:345       15.7    16:19:345       23.1    19:34:542       12.3
  20:12:187       16      22:00:093       -7
10 NY_30-Jan-2014 00:22:445      15.7    01:19:345       23.1    02:34:542       12.3    03:12:187       56      04:00:093       -14     05:12:345
  35.7    06:19:345       39.6    07:34:542       12.3    08:12:187       16      09:00:093       -7      10:12:345       15.7    11:19:345       23.1
  12:34:542       12.3    13:12:187       16      14:00:093       -7      15:12:345       -15.7   16:19:345       23.1    19:34:542       12.3
  20:12:187       16      22:00:093       -7
11 NY_31-Jan-2014 00:42:245      15.7    01:19:345       23.1    02:34:542       12.3    03:12:187       16      04:00:093       -14     05:12:345
  49.2    06:19:345       23.1    07:34:542       12.3    08:12:187       16      09:00:093       -7      10:12:345       15.7    11:19:345       23.1
  12:34:542       12.3    13:12:187       16      14:00:093       -7      15:12:345       15.7    16:19:345       23.1    19:34:542       12.3
  20:12:187       16      22:00:093       -27
```

Prerequisites

1.Create a folder in the home named wordcount. 2.put the input file in the inside of wordcount folder.

3.create a folder named classes inside the wordcount folder.

Steps to execute MapReduce word count example

1.Set up the path and configure the java environment:

COMMANDS
1. export HADOOP_CLASSPATH=$(hadoop classpath) *2.echo $HADOOP_CLASSPATH*

OUTPUT:

```
/usr/local/hadoop/etc/hadoop:/usr/local/hadoop/share/hadoop/common/lib/*:/usr/lo
cal/hadoop/share/hadoop/common/*:/usr/local/hadoop/share/hadoop/hdfs:/usr/local/
hadoop/share/hadoop/hdfs/lib/*:/usr/local/hadoop/share/hadoop/hdfs/*:/usr/local/
hadoop/share/hadoop/mapreduce/*:/usr/local/hadoop/share/hadoop/yarn:/usr/local/h
adoop/share/hadoop/yarn/lib/*:/usr/local/hadoop/share/hadoop/yarn/*:/usr/local/h
adoop/lib/javax.activation-api-1.2.0.jar
```

2.Open cmd in Ubuntu terminal and start a cluster.

COMMANDS
Start-all.sh

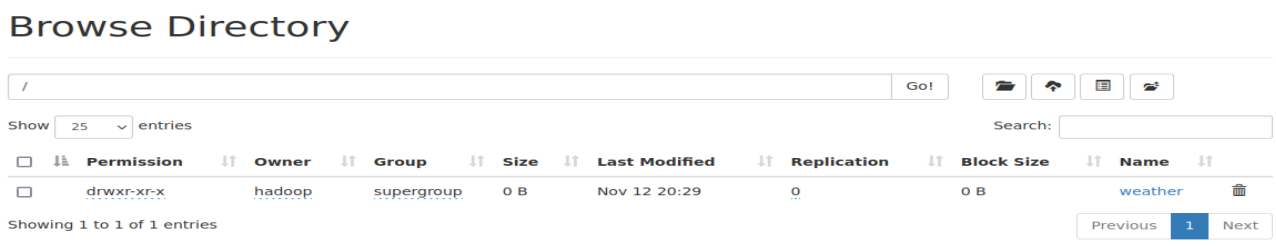
3.Verify the what are the processes running and list the process id

COMMANDS
jps

4. Create an input directory in HDFS.

COMMANDS
hdfs dfs -mkdir /weather/

Output:



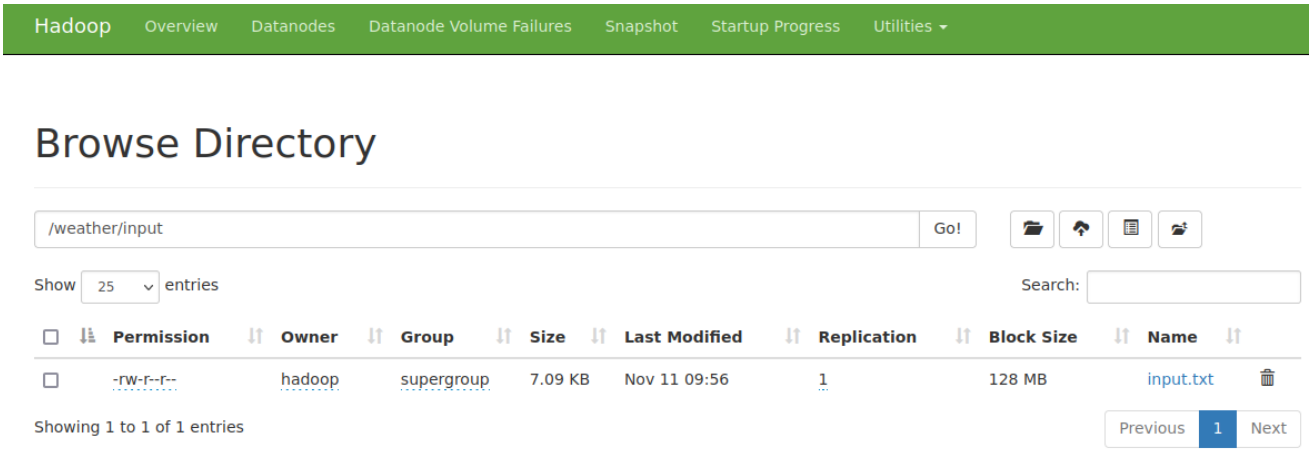
5.Create a input folder inside of wordcount directory.

COMMANDS
hdfs dfs -mkdir /weather/input

6.Copy the input text file named input_file.txt in the input directory of HDFS.

Copy some text file to hadoop filesystem inside input directory. Here I am copying a input.txt to it. You can copy more that one files.

COMMANDS
hdfs dfs -put '/home/moni/weather/input/input.txt' /weather/input/



7. Verify content of the copied file.

COMMANDS
*hdfs dfs -cat /weather/input/**

```
hadoop@swetha:~$ hdfs dfs -cat /weather/input/*
CA_25-Jan-2014  00:12:345       15.7    01:19:345       23.1    02:34:542       1
2.3     03:12:187       16      04:00:093       -14     05:12:345       35.7    0
6:19:345        23.1    07:34:542       12.3    08:12:187       16      09:00:09
3       -7      10:12:345       15.7    11:19:345       23.1    12:34:542       -
22.3    13:12:187       16      14:00:093       -7      15:12:345       15.7    1
6:19:345        23.1    19:34:542       12.3    20:12:187       16      22:00:09
3       -7
CA_26-Jan-2014  00:54:245       15.7    01:19:345       23.1    02:34:542       1
2.3     03:12:187       16      04:00:093       -14     05:12:345       55.7    0
6:19:345        23.1    07:34:542       12.3    08:12:187       16      09:00:09
3       -7      10:12:345       15.7    11:19:345       23.1    12:34:542       1
2.3     13:12:187       16      14:00:093       -7      15:12:345       15.7    1
6:19:345        23.1    19:34:542       12.3    20:12:187       16      22:00:09
3       -7
CA_27-Jan-2014  00:14:045       35.7    01:19:345       23.1    02:34:542       -
22.3    03:12:187       16      04:00:093       -14     05:12:345       35.7    0
6:19:345        23.1    07:34:542       12.3    08:12:187       16      09:00:09
3       -7      10:12:345       15.7    11:19:345       23.1    12:34:542       1
2.3     13:12:187       16      14:00:093       -7      15:12:345       15.7    1
6:19:345        23.1    19:34:542       12.3    20:12:187       16      22:00:09
3       -7
```

8.Move into the wordcount folder in home which contains the java file

COMMANDS
cd /home/moni/weather

and whether check and print the working directory and list the contents of the file.

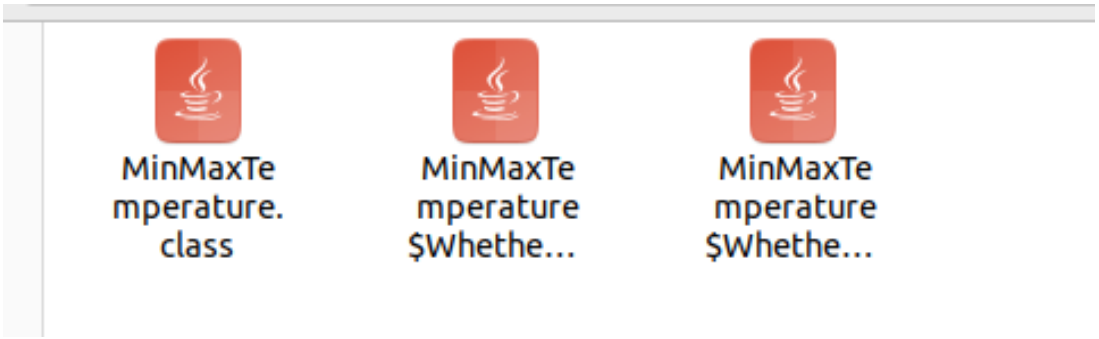
TERMINAL:

```
hadoop@swetha:~$ cd /home/hadoop/weather
hadoop@swetha:~/weather$ pwd
/home/hadoop/weather
hadoop@swetha:~/weather$ ls
classes  input.txt  MinMaxTemperature.java
hadoop@swetha:~/weather$
```

9.Finally,the following commands are used for compiling the WordCount.java program.

COMMANDS
javac -classpath ${HADOOP_CLASSPATH} -d '/home/gokilanm/weather/classes"/home/gokilanm/weather/MinMaxTemperature.java'

OUTPUT:


MinMaxTe
mperature.
class
MinMaxTe
mperature
$Whethe…
MinMaxTe
mperature
$Whethe…

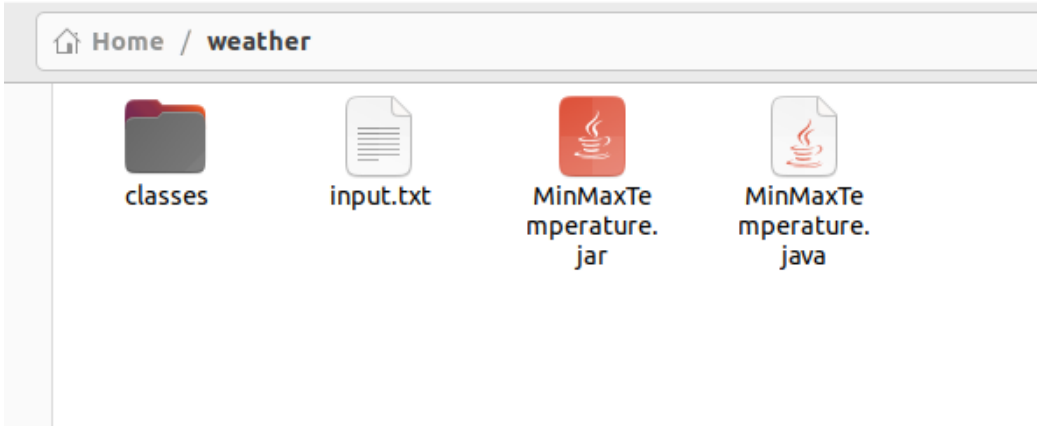
10. *Create a jar file using the command*

COMMANDS
jar -cvf MinMaxTemperature.jar -C ‘/home/gokilanm/wordcount/classes’/ .

TERMINAL:

```
added manifest
adding: MinMaxTemperature$WhetherForcastMapper.class(in = 2682) (out= 1242)(defl
ated 53%)
adding: MinMaxTemperature.class(in = 2577) (out= 1299)(deflated 49%)
adding: MinMaxTemperature$WhetherForcastReducer.class(in = 3927) (out= 1628)(def
lated 58%)
```

OUTPUT:



11. *Run MapReduceClient.jar and also provide input and out directories*

COMMANDS
hadoop jar ‘home/gokilanm/weather/ MinMaxTemperature.jar’ MinMaxTemperature /weather/input /weather/output

TERMINAL

```
2022-11-11 10:25:36,576 INFO client.DefaultNoHARMFailoverProxyProvider: Connecti
ng to ResourceManager at /127.0.0.1:8032
2022-11-11 10:25:36,959 WARN mapreduce.JobResourceUploader: Hadoop command-line
option parsing not performed. Implement the Tool interface and execute your appl
ication with ToolRunner to remedy this.
2022-11-11 10:25:36,980 INFO mapreduce.JobResourceUploader: Disabling Erasure Co
ding for path: /tmp/hadoop-yarn/staging/hadoop/.staging/job_1668140632188_0001
2022-11-11 10:25:37,462 INFO input.FileInputFormat: Total input files to process
 : 1
2022-11-11 10:25:37,593 INFO mapreduce.JobSubmitter: number of splits:1
2022-11-11 10:25:37,862 INFO mapreduce.JobSubmitter: Submitting tokens for job:
job_1668140632188_0001
```

```
ding for path: /tmp/hadoop-yarn/staging/hadoop/.staging/job_1668140632188_0001
2022-11-11 10:25:37,462 INFO input.FileInputFormat: Total input files to process
 : 1
2022-11-11 10:25:37,593 INFO mapreduce.JobSubmitter: number of splits:1
2022-11-11 10:25:37,862 INFO mapreduce.JobSubmitter: Submitting tokens for job:
job_1668140632188_0001
2022-11-11 10:25:37,862 INFO mapreduce.JobSubmitter: Executing with tokens: []
2022-11-11 10:25:38,052 INFO conf.Configuration: resource-types.xml not found
2022-11-11 10:25:38,055 INFO resource.ResourceUtils: Unable to find 'resource-ty
pes.xml'.
2022-11-11 10:25:38,502 INFO impl.YarnClientImpl: Submitted application applicat
ion_1668140632188_0001
2022-11-11 10:25:38,588 INFO mapreduce.Job: The url to track the job: http://swe
tha:8088/proxy/application_1668140632188_0001/
2022-11-11 10:25:38,589 INFO mapreduce.Job: Running job: job_1668140632188_0001
2022-11-11 10:25:45,780 INFO mapreduce.Job: Job job_1668140632188_0001 running i
n uber mode : false
2022-11-11 10:25:45,785 INFO mapreduce.Job:  map 0% reduce 0%
2022-11-11 10:25:50,922 INFO mapreduce.Job:  map 100% reduce 0%
2022-11-11 10:25:55,962 INFO mapreduce.Job:  map 100% reduce 100%
2022-11-11 10:25:57,018 INFO mapreduce.Job: Job job_1668140632188_0001 completed
 successfully
2022-11-11 10:25:57,117 INFO mapreduce.Job: Counters: 54
        File System Counters
```

```
    File System Counters
            FILE: Number of bytes read=1686
            FILE: Number of bytes written=542479
            FILE: Number of read operations=0
            FILE: Number of large read operations=0
            FILE: Number of write operations=0
            HDFS: Number of bytes read=7369
            HDFS: Number of bytes written=1752
            HDFS: Number of read operations=14
            HDFS: Number of large read operations=0
            HDFS: Number of write operations=14
            HDFS: Number of bytes read erasure-coded=0
    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)=2407
            Total time spent by all reduces in occupied slots (ms)=3193
            Total time spent by all map tasks (ms)=2407
            Total time spent by all reduce tasks (ms)=3193
            Total vcore-milliseconds taken by all map tasks=2407
            Total vcore-milliseconds taken by all reduce tasks=3193
```

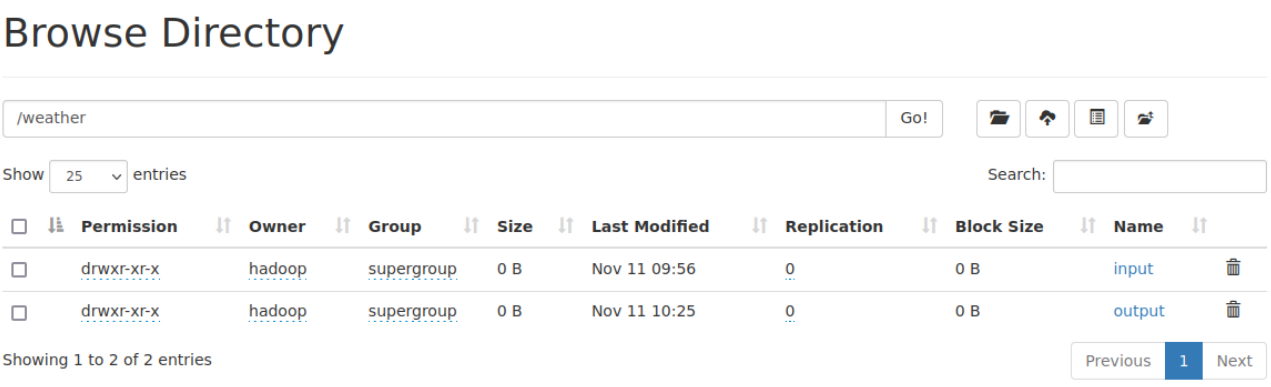
```
            Total time spent by all maps in occupied slots (ms)=2407
            Total time spent by all reduces in occupied slots (ms)=3193
            Total time spent by all map tasks (ms)=2407
            Total time spent by all reduce tasks (ms)=3193
            Total vcore-milliseconds taken by all map tasks=2407
            Total vcore-milliseconds taken by all reduce tasks=3193
            Total megabyte-milliseconds taken by all map tasks=2464768
            Total megabyte-milliseconds taken by all reduce tasks=3269632
    Map-Reduce Framework
            Map input records=24
            Map output records=48
            Map output bytes=1584
            Map output materialized bytes=1686
            Input split bytes=107
            Combine input records=0
            Combine output records=0
            Reduce input groups=24
            Reduce shuffle bytes=1686
            Reduce input records=48
            Reduce output records=0
            Spilled Records=96
            Shuffled Maps =1
            Failed Shuffles=0
            Merged Map outputs=1
```

```
                Failed Shuffles=0
                Merged Map outputs=1
                GC time elapsed (ms)=59
                CPU time spent (ms)=1860
                Physical memory (bytes) snapshot=468848640
                Virtual memory (bytes) snapshot=5445410816
                Total committed heap usage (bytes)=357564416
                Peak Map Physical memory (bytes)=279871488
                Peak Map Virtual memory (bytes)=2717323264
                Peak Reduce Physical memory (bytes)=188977152
                Peak Reduce Virtual memory (bytes)=2728087552
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=7262
        File Output Format Counters
                Bytes Written=0
hadoop@swetha:~/weather$
```

12. VERIFY THE OUTPUT FOLDER IS CREATED OR NOT:

Check the output in the Web UI at http://localhost:9870

In the Utilities tab select browse file system and select the correct user. The output is available inside the output folder named user.



MULTIPLE-OUTPUTS:

Show 25 entries Search:

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r--	hadoop	supergroup	438 B	Nov 11 10:25	1	128 MB	Austin-r-00000
-rw-r--r--	hadoop	supergroup	219 B	Nov 11 10:25	1	128 MB	Baltimore-r-00000
-rw-r--r--	hadoop	supergroup	219 B	Nov 11 10:25	1	128 MB	Boston-r-00000
-rw-r--r--	hadoop	supergroup	511 B	Nov 11 10:25	1	128 MB	California-r-00000
-rw-r--r--	hadoop	supergroup	146 B	Nov 11 10:25	1	128 MB	Newjersy-r-00000
-rw-r--r--	hadoop	supergroup	219 B	Nov 11 10:25	1	128 MB	Newyork-r-00000
-rw-r--r--	hadoop	supergroup	0 B	Nov 11 10:25	1	128 MB	_SUCCESS
-rw-r--r--	hadoop	supergroup	0 B	Nov 11 10:25	1	128 MB	part-r-00000

Showing 1 to 8 of 8 entries Previous 1 Next

13. PRINT THE CONTENTS OF THE OUTPUT FILE/VERIFY CONTENT FOR GENERATED OUTPUT FILE.

COMMANDS
*hdfs dfs -car /weather/output/**

TERMINAL:

```
hadoop@swetha:~/weather$ hdfs dfs -cat /weather/output/*
25-Jan-2014     Time: 12:34:542 MinTemp: -22.3  Time: 05:12:345 MaxTemp: 35.7
26-Jan-2014     Time: 22:00:093 MinTemp: -27.0  Time: 05:12:345 MaxTemp: 55.7
27-Jan-2014     Time: 02:34:542 MinTemp: -22.3  Time: 05:12:345 MaxTemp: 55.7
29-Jan-2014     Time: 14:00:093 MinTemp: -17.0  Time: 02:34:542 MaxTemp: 62.9
30-Jan-2014     Time: 22:00:093 MinTemp: -27.0  Time: 05:12:345 MaxTemp: 49.2
31-Jan-2014     Time: 14:00:093 MinTemp: -17.0  Time: 03:12:187 MaxTemp: 56.0
29-Jan-2014     Time: 14:00:093 MinTemp: -27.0  Time: 05:12:345 MaxTemp: 49.2
30-Jan-2014     Time: 14:00:093 MinTemp: -17.0  Time: 02:34:542 MaxTemp: 52.9
31-Jan-2014     Time: 15:12:345 MinTemp: -15.7  Time: 03:12:187 MaxTemp: 56.0
28-Jan-2014     Time: 11:19:345 MinTemp: -23.3  Time: 05:12:345 MaxTemp: 35.7
29-Jan-2014     Time: 14:00:093 MinTemp: -17.0  Time: 02:34:542 MaxTemp: 52.9
30-Jan-2014     Time: 15:12:345 MinTemp: -15.7  Time: 03:12:187 MaxTemp: 56.0
25-Jan-2014     Time: 12:34:542 MinTemp: -22.3  Time: 05:12:345 MaxTemp: 35.7
26-Jan-2014     Time: 04:00:093 MinTemp: -14.0  Time: 05:12:345 MaxTemp: 55.7
27-Jan-2014     Time: 02:34:542 MinTemp: -22.3  Time: 00:14:045 MaxTemp: 35.7
28-Jan-2014     Time: 11:19:345 MinTemp: -23.3  Time: 05:12:345 MaxTemp: 35.7
29-Jan-2014     Time: 14:00:093 MinTemp: -17.0  Time: 02:34:542 MaxTemp: 52.9
30-Jan-2014     Time: 15:12:345 MinTemp: -15.7  Time: 03:12:187 MaxTemp: 56.0
31-Jan-2014     Time: 22:00:093 MinTemp: -27.0  Time: 05:12:345 MaxTemp: 49.2
29-Jan-2014     Time: 14:00:093 MinTemp: -17.0  Time: 02:34:542 MaxTemp: 52.9
30-Jan-2014     Time: 15:12:345 MinTemp: -15.7  Time: 03:12:187 MaxTemp: 56.0
29-Jan-2014     Time: 14:00:093 MinTemp: -17.0  Time: 02:34:542 MaxTemp: 52.9
30-Jan-2014     Time: 15:12:345 MinTemp: -15.7  Time: 03:12:187 MaxTemp: 56.0
31-Jan-2014     Time: 22:00:093 MinTemp: -27.0  Time: 05:12:345 MaxTemp: 49.2
hadoop@swetha:~/weather$
```

RESULT:

Thus the minimum and maximum temperature of city is found successfully by the use of Map and Reduce tasks.

Experiment 6 :Perform Joining data with streaming in Hadoop using Map Reduce

a) *OBJECTIVE OF THE EXERCISE/EXPERIMENT*

To Perform a Joining data with streaming in Hadoop using Map Reduce

b) *Facilities/material required to do the exercise/experiment:*

Sl.No.	*Facilities/material required*	*Quantity*
1.	*Hadoop*	*As per strength*

THEORY:

- *Mapreduce Join operation is used to combine two large datasets. However, this process involves writing lots of code to perform the actual join operation. Joining two datasets begins by comparing the size of each dataset. If one dataset is smaller as compared to the other dataset then smaller dataset is distributed to every data node in the cluster.*
- *Once a join in MapReduce is distributed, either Mapper or Reducer uses the smaller dataset to perform a lookup for matching records from the large dataset and then combine those records to form output records*

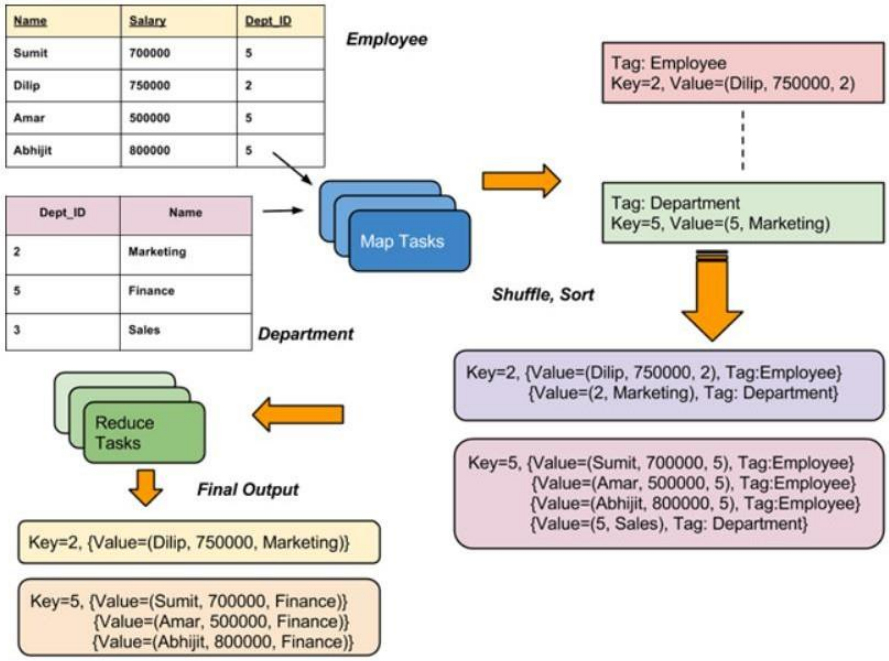
Types of Join

Depending upon the place where the actual join is performed, joins in Hadoop are classified into-

1. Map-side join – When the join is performed by the mapper, it is called as map-side join. In this type, the join is performed before data is actually consumed by the map function. It is mandatory that the input to each map is in the form of a partition and is in sorted order. Also, there must be an equal number of partitions and it must be sorted by the join key.

2. Reduce-side join – When the join is performed by the reducer, it is called as reduce-side join. There is no necessity in this join to have a dataset in a structured form (or partitioned).

Here, map side processing emits join key and corresponding tuples of both the tables. As an effect of this processing, all the tuples with same join key fall into the same reducer which then joins the records with same join key.



Types of Joins in Hadoop MapReduce

SOURCE CODE:

TextPair.java

```
package
MapReduceJoin;
import java.io.*;
import org.apache.hadoop.io.*;
public class TextPair implements
 WritableComparable<TextPair> { private Text first;
 private Text
 second; public
 TextPair() {
  set(new Text(), new Text());
 }
```

```
public TextPair(String first, String
  second) { set(new Text(first), new
  Text(second));
}
public TextPair(Text first, Text
  second) { set(first, second);
}
public void set(Text first, Text
  second) { this.first = first;
  this.second = second;
}
public Text
  getFirst() { return
  first;
}
public Text
  getSecond() { return
  second;
}
@Override
public void write(DataOutput out) throws
  IOException { first.write(out);
  second.write(out);
}
@Override
public void readFields(DataInput in) throws
  IOException { first.readFields(in);
  second.readFields(in);
}@Override
```

```
public int hashCode() {
  return first.hashCode() * 163 + second.hashCode();
}
@Override
public boolean equals(Object
  o) { if (o instanceof TextPair)
  { TextPair tp = (TextPair) o;
    return first.equals(tp.first) && second.equals(tp.second);
  }
  return false;
}

@Override
public String toString() {
  return first + "\t" +
  second;
}

@Override
public  int  compareTo(TextPair
  tp)      {      int      cmp      =
  first.compareTo(tp.first);      if
  (cmp != 0) {
    return cmp;
  }
  return second.compareTo(tp.second);
}
// ^^ TextPair
```

```
// vv TextPairComparator
public static class Comparator extends WritableComparator {

  private static final Text.Comparator TEXT_COMPARATOR
= new Text.Comparator();

  public
   Comparator() {
   super(TextPair.clas
   s);
  }

  @Override
  public int compare(byte[] b1, int s1,
                     int l1, byte[] b2, int s2, int
                     l2) {

   try {
    int firstL1 = WritableUtils.decodeVIntSize(b1[s1]) +
    readVInt(b1, s1); int firstL2 =
    WritableUtils.decodeVIntSize(b2[s2]) + readVInt(b2, s2);
    int cmp = TEXT_COMPARATOR.compare(b1, s1, firstL1,
b2, s2, firstL2);
    if (cmp != 0)
     { return
     cmp;
    }
    return TEXT_COMPARATOR.compare(b1, s1 + firstL1, l1 -
                                   firstL1, b2, s2 + firstL2, l2 - firstL2);
   } catch (IOException e) {
```

```
        throw new IllegalArgumentException(e);
    }
}
```

```
}

static {
  WritableComparator.define(TextPair.class, new Comparator());
}
// ^^ TextPairComparator

// vv TextPairFirstComparator
public static class FirstComparator extends WritableComparator {

  private static final Text.Comparator TEXT_COMPARATOR
= new Text.Comparator();

  public
    FirstComparator() {
    super(TextPair.class);
  }

  @Override
  public int compare(byte[] b1, int s1,
                     int l1, byte[] b2, int s2, int
                     l2) {
    try {
      int firstL1 = WritableUtils.decodeVIntSize(b1[s1]) + readVInt(b1,
      s1); int firstL2 = WritableUtils.decodeVIntSize(b2[s2]) +
      readVInt(b2, s2); return TEXT_COMPARATOR.compare(b1, s1,
      firstL1, b2, s2, firstL2);
    } catch (IOException e) {
      throw new IllegalArgumentException(e);
    }
  }
```

```
    @Override
    public int compare(WritableComparable a,
      WritableComparable b) { if (a instanceof TextPair && b
      instanceof TextPair) {
        return ((TextPair) a).first.compareTo(((TextPair) b).first);
      }
      return super.compare(a, b);
    }
  }
  // ^^ TextPairFirstComparator
// vv TextPair
}
```

JOIN DRIVER.java:

```
package MapReduceJoin;
import
org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.Path;
import
org.apache.hadoop.io.Text;
import
org.apache.hadoop.mapred.*;
import
org.apache.hadoop.mapred.lib.MultipleInputs;
import org.apache.hadoop.util.*;
public class JoinDriver extends Configured implements Tool {
        public static class KeyPartitioner implements Partitioner<TextPair,
        Text>
{
```

```
@Override
public void configure(JobConf
job) {} @Override
```

```
        public int getPartition(TextPair key, Text value, int numPartitions)
{
            return (key.getFirst().hashCode() & Integer.MAX_VALUE)
% numPartitions;
        }
    }
    @Override
    public int run(String[] args) throws
        Exception { if (args.length != 3) {
            System.out.println("Usage: <Department Emp
Strength input> <Department Name input> <output>");
            return -1;
        }
        JobConf conf = new JobConf(getConf(), getClass());
        conf.setJobName("Join 'Department Emp Strength
        input' with
'Department Name input'");
        Path AInputPath = new
        Path(arg[0]); Path BInputPath =
        new Path(arg[1]); Path
        outputPath = new Path(args[2]);
        MultipleInputs.addInputPath(conf,
AInputPath, TextInputFormat.class,
DeptEmpStrengthMapper.class);
        MultipleInputs.addInputPath(conf,
BInputPath, TextInputFormat.class,
DeptNameMapper.class);
        FileOutputFormat.setOutputPath(conf,
        outputPath);
        conf.setPartitionerClass(KeyPartitioner.class)
        ;
    conf.setOutputValueGroupingComparator(TextPair.FirstCompar
                                    ator.clas
```

```
s);
        conf.setMapOutputKeyClass(TextPair.cl
        ass);
        conf.setReducerClass(JoinReducer.class
        ); conf.setOutputKeyClass(Text.class);
```

```
        JobClient.runJob(co
        nf); return 0;
    }
    public static void main(String[] args) throws Exception {
        int exitCode = ToolRunner.run(new JoinDriver(),
        args); System.exit(exitCode);
    }
}
```

JoinReducer.java:

```
package
MapReduceJoin; import
java.io.IOException;
import
java.util.Iterator;
import
org.apache.hadoop.io.Text;
import
org.apache.hadoop.mapred.*;
public class JoinReducer extends MapReduceBase
implements Reducer<TextPair, Text, Text, Text> {
    @Override
    public void reduce (TextPair key, Iterator<Text>
values, OutputCollector<Text, Text> output, Reporter
reporter)
            throws IOException
    {
```

```
Text nodeId = new
Text(values.next()); while
(values.hasNext()) {
        Text node = values.next();
```

```
                Text outValue = new Text(nodeId.toString() + "\t\t"
+ node.toString());
                output.collect(key.getFirst(), outValue);
            }
        }
}
```

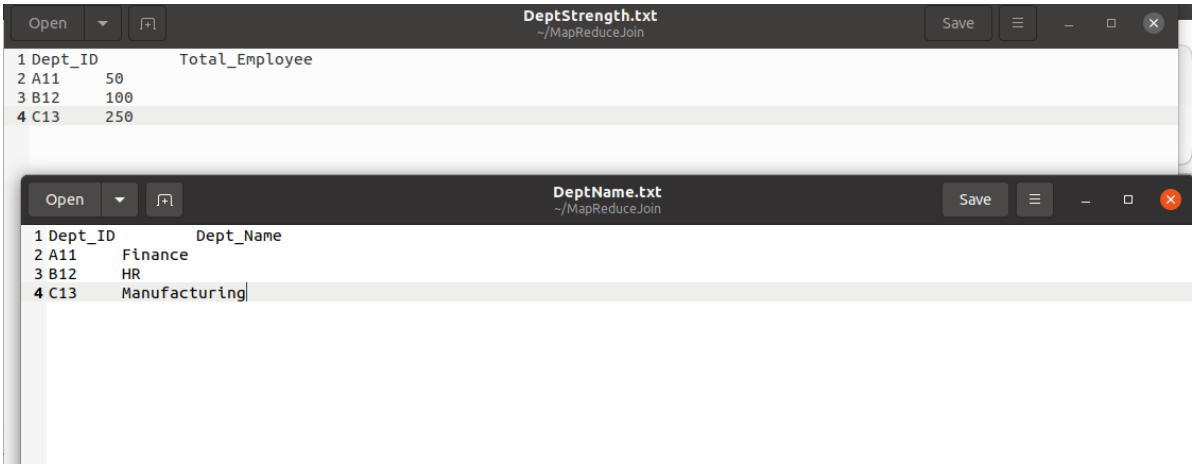
SNAP:

```
1 package MapReduceJoin;
2 import org.apache.hadoop.conf.Configured;
3 import org.apache.hadoop.fs.Path;
4 import org.apache.hadoop.io.Text;
5 import org.apache.hadoop.mapred.*;
6 import org.apache.hadoop.mapred.lib.MultipleInputs;
7 import org.apache.hadoop.util.*;
8 public class JoinDriver extends Configured implements Tool {
9         public static class KeyPartitioner implements Partitioner<TextPair, Text> {
10                @Override
11                public void configure(JobConf job) {}
12                @Override
13                public int getPartition(TextPair key, Text value, int numPartitions) {
14                        return (key.getFirst().hashCode() & Integer.MAX_VALUE) % numPartitions;
15                }
16        }
17        @Override
18        public int run(String[] args) throws Exception {
19                if (args.length != 3) {
20                        System.out.println("Usage: <Department Emp Strength input> <Department Name input> <output>");
21                        return -1;
22                }
23                JobConf conf = new JobConf(getConf(), getClass());
24                conf.setJobName("Join 'Department Emp Strength input' with 'Department Name input'");
25                Path AInputPath = new Path(args[0]);
26                Path BInputPath = new Path(args[1]);
27                Path outputPath = new Path(args[2]);
28                MultipleInputs.addInputPath(conf, AInputPath, TextInputFormat.class, DeptNameMapper.class);
29                MultipleInputs.addInputPath(conf, BInputPath, TextInputFormat.class, DeptEmpStrengthMapper.class);
30                FileOutputFormat.setOutputPath(conf, outputPath);
31                conf.setPartitionerClass(KeyPartitioner.class);
32                conf.setOutputValueGroupingComparator(TextPair.FirstComparator.class);
33                conf.setMapOutputKeyClass(TextPair.class);
34                conf.setReducerClass(JoinReducer.class);
35                conf.setOutputKeyClass(Text.class);
36                JobClient.runJob(conf);
37                return 0;
38        }
39        public static void main(String[] args) throws Exception {
40                int exitCode = ToolRunner.run(new JoinDriver(), args);
41                System.exit(exitCode);
42        }
43 }
```

```
1
2 package MapReduceJoin;
3
4 import java.io.IOException;
5 import java.util.Iterator;
6
7 import org.apache.hadoop.io.Text;
8 import org.apache.hadoop.mapred.*;
9
10 public class JoinReducer extends MapReduceBase implements Reducer<TextPair, Text, Text, Text> {
11
12         @Override
13         public void reduce (TextPair key, Iterator<Text> values, OutputCollector<Text, Text> output, Reporter reporter)
14                         throws IOException
15         {
16
17                 Text nodeId = new Text(values.next());
18                 while (values.hasNext()) {
19                         Text node = values.next();
20                         Text outValue = new Text(nodeId.toString() + "\t\t" + node.toString());
21                         output.collect(key.getFirst(), outValue);
22                 }
23         }
24 }
25
```

```
1 package MapReduceJoin;
2 import java.io.*;
3 import org.apache.hadoop.io.*;
4 public class TextPair implements WritableComparable<TextPair> {
5   private Text first;
6   private Text second;
7   public TextPair() {
8     set(new Text(), new Text());
9   }
10  public TextPair(String first, String second) {
11    set(new Text(first), new Text(second));
12  }
13  public TextPair(Text first, Text second) {
14    set(first, second);
15  }
16  public void set(Text first, Text second) {
17    this.first = first;
18    this.second = second;
19  }
20  public Text getFirst() {
21    return first;
22  }
23  public Text getSecond() {
24    return second;
25  }
26  @Override
27  public void write(DataOutput out) throws IOException {
28    first.write(out);
29    second.write(out);
30  }
31  @Override
32  public void readFields(DataInput in) throws IOException {
33    first.readFields(in);
34    second.readFields(in);
35  }
36  @Override
37  public int hashCode() {
38    return first.hashCode() * 163 + second.hashCode();
39  }
40  @Override
41  public boolean equals(Object o) {
42    if (o instanceof TextPair) {
43      TextPair tp = (TextPair) o;
44      return first.equals(tp.first) && second.equals(tp.second);
```

INPUT FILE:



Steps to execute MapReduce word count example

1. Set up the path and configure the java environment:

COMMANDS
1. export HADOOP_CLASSPATH=$(hadoop classpath) *2.echo $HADOOP_CLASSPATH*

OUTPUT:

```
gokilanm@gokila-VirtualBox:~$ export HADOOP_CLASSPATH=$(hadoop classpath)
gokilanm@gokila-VirtualBox:~$ echo $HADOOP_CLASSPATH
/home/gokilanm/hadoop-3.2.2/etc/hadoop:/home/gokilanm/hadoop-3.2.2/share/hadoop
/common/lib/*:/home/gokilanm/hadoop-3.2.2/share/hadoop/common/*:/home/gokilanm/
hadoop-3.2.2/share/hadoop/hdfs:/home/gokilanm/hadoop-3.2.2/share/hadoop/hdfs/li
b/*:/home/gokilanm/hadoop-3.2.2/share/hadoop/hdfs/*:/home/gokilanm/hadoop-3.2.2
/share/hadoop/mapreduce/lib/*:/home/gokilanm/hadoop-3.2.2/share/hadoop/mapreduc
e/*:/home/gokilanm/hadoop-3.2.2/share/hadoop/yarn:/home/gokilanm/hadoop-3.2.2/s
hare/hadoop/yarn/lib/*:/home/gokilanm/hadoop-3.2.2/share/hadoop/yarn/*
```

2.Open cmd in Ubuntu terminal and start a cluster.

COMMANDS
Start-all.sh

TERMINAL:

```
gokilanm@gokila-VirtualBox:~$ start-all.sh
WARNING: Attempting to start all Apache Hadoop daemons as gokilanm in 10 seconds.
WARNING: This is not a recommended production deployment configuration.
WARNING: Use CTRL-C to abort.
Starting namenodes on [localhost]
Starting datanodes
Starting secondary namenodes [gokila-VirtualBox]
Starting resourcemanager
Starting nodemanagers
```

3.Verify the what are the processes running and list the process id

COMMANDS
Jps

```
gokilanm@gokila-VirtualBox:~$ jps
13585 Jps
13108 ResourceManager
12581 NameNode
13240 NodeManager
12714 DataNode
12910 SecondaryNameNode
gokilanm@gokila-VirtualBox:~$ S
```

TERMINAL:

4. Create an input directory in HDFS.

COMMANDS

hdfs dfs -mkdir / MapReduceJoin/

Terminal:

```
gokilanm@gokila-VirtualBox:~$ hdfs dfs -mkdir /MapReduceJoin/
```

Output:

```
drwxr-xr-x   gokilanm   supergroup   0 B   Oct 05 07:11   0   0 B   matrixmultiplication
```

5.Create a input folder inside of MapReduceJoin directory.

COMMANDS
hdfs dfs -mkdir / MapReduceJoin /input

TERMINAL:

```
gokilanm@gokila-VirtualBox:~$ hdfs dfs -mkdir /MapReduceJoin/input
```

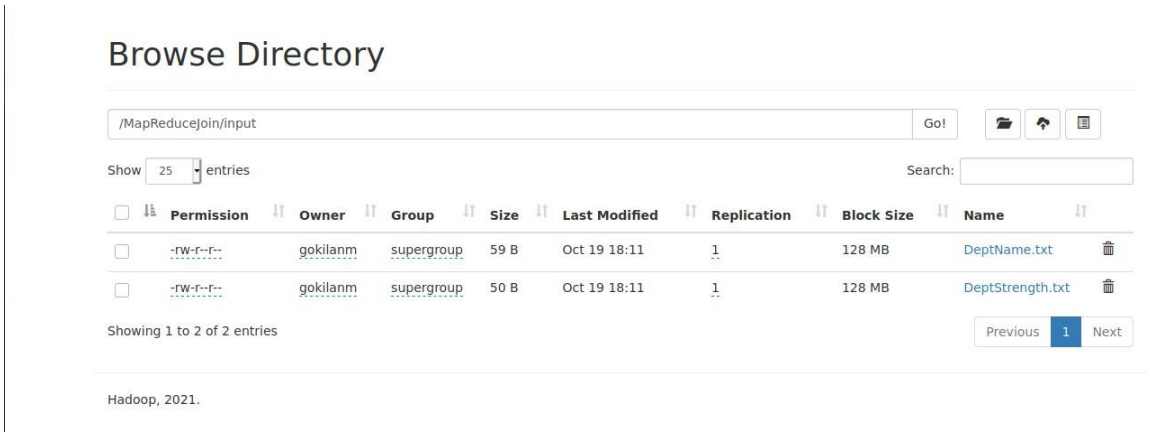
6.Copy the 2 input text file in the input directory (matrixmultiplication) of HDFS.

Copy some text file to hadoop filesystem inside input directory. Here I am copying a input.txt to it. You can copy more that one files.

COMMANDS
1.hdfs dfs -put '/home/gokilanm/MapReduceJoin/DeptName.txt' / MapReduceJoin /input/ *2. hdfs dfs -put '/home/gokilanm/MapReduceJoin/DeptStrength.txt' / MapReduceJoin /input/*

TERMINAL:

```
gokilanm@gokila-VirtualBox:~$ hdfs dfs -put '/home/gokilanm/MapReduceJoin/DeptName.txt'  /MapReduceJoin/input/
gokilanm@gokila-VirtualBox:~$ hdfs dfs -put '/home/gokilanm/MapReduceJoin/DeptStrength.txt'  /MapReduceJoin/input/
```



7. Verify content of the copied file.

COMMANDS
*hdfs dfs -cat / MaapReduceJoin/input/**

```
gokilanm@gokila-VirtualBox:~$ hdfs dfs -cat /MapReduceJoin/input/*
Dept_ID Dept_Name
A11     Finance
B12     HR
C13     Manufacturing
Dept_ID Total_Employee
A11     50
B12     100
C13     250
```

8.Move into the MapReduceJoin folder in home which contains the java file and whether check and print the working directory and list the contents of the

COMMANDS
cd /home/gokilanm/ MapReduceJoin

file.

TERMINAL

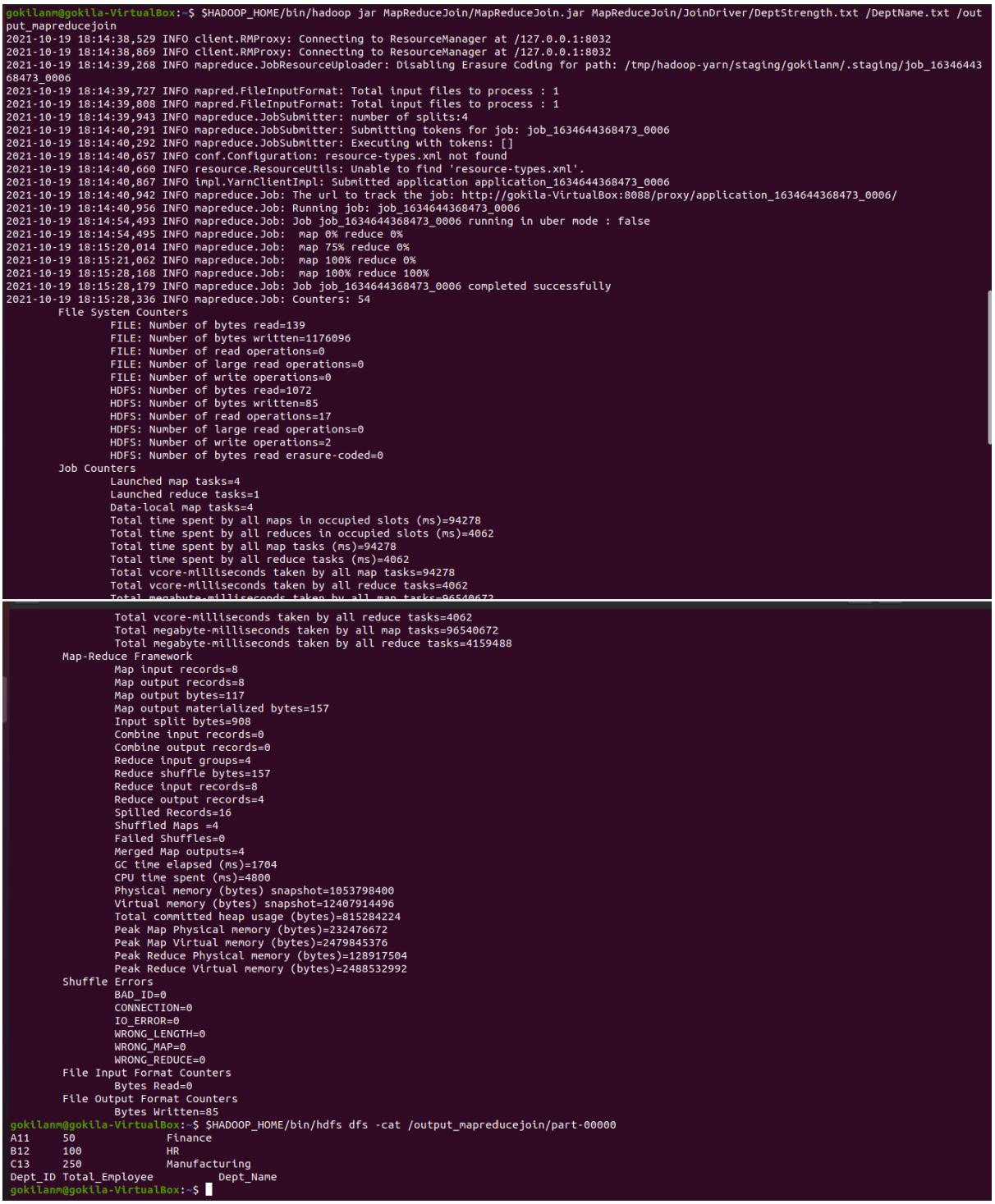
:

```
gokilanm@gokila-VirtualBox:~$ cd MapReduceJoin
gokilanm@gokila-VirtualBox:~/MapReduceJoin$ pwd
/home/gokilanm/MapReduceJoin
gokilanm@gokila-VirtualBox:~/MapReduceJoin$ ls
A.txt~                        DeptName.txt        Manifest.txt~
B.txt~                        DeptStrength.txt    MapReduceJoin
DeptEmpStrengthMapper.java    JoinDriver.java     MapReduceJoin.jar
DeptEmpStrengthMapper.java~   JoinDriver.java~    TextPair.java
DeptEmpStrength.txt           JoinReducer.java    TextPair.java~
DeptNameMapper.java           JoinReducer.java~
DeptNameMapper.java~          Manifest.txt
gokilanm@gokila-VirtualBox:~/MapReduceJoin$
```

STEP-9:

Run the program using below command-

COMMANDS
$HADOOP_HOME/bin/hadoop jar MapReduceJoin.jar MapReduceJoin/JoinDriver/DeptStrength.txt /DeptName.txt /output_mapreducejoin

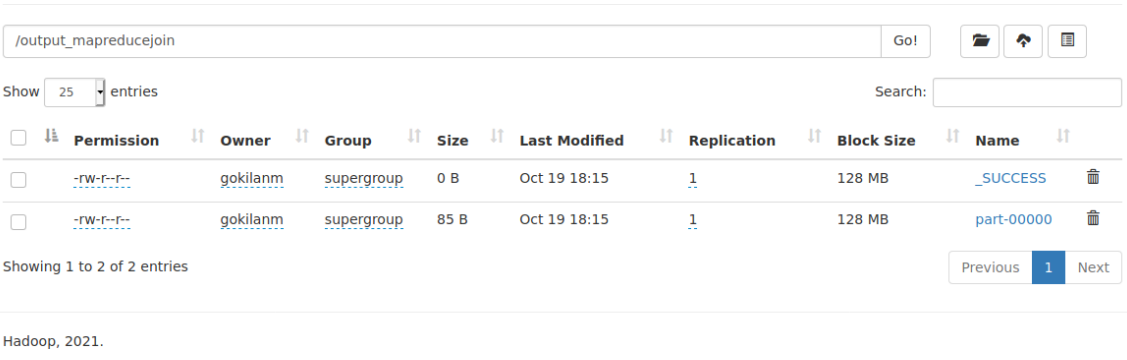


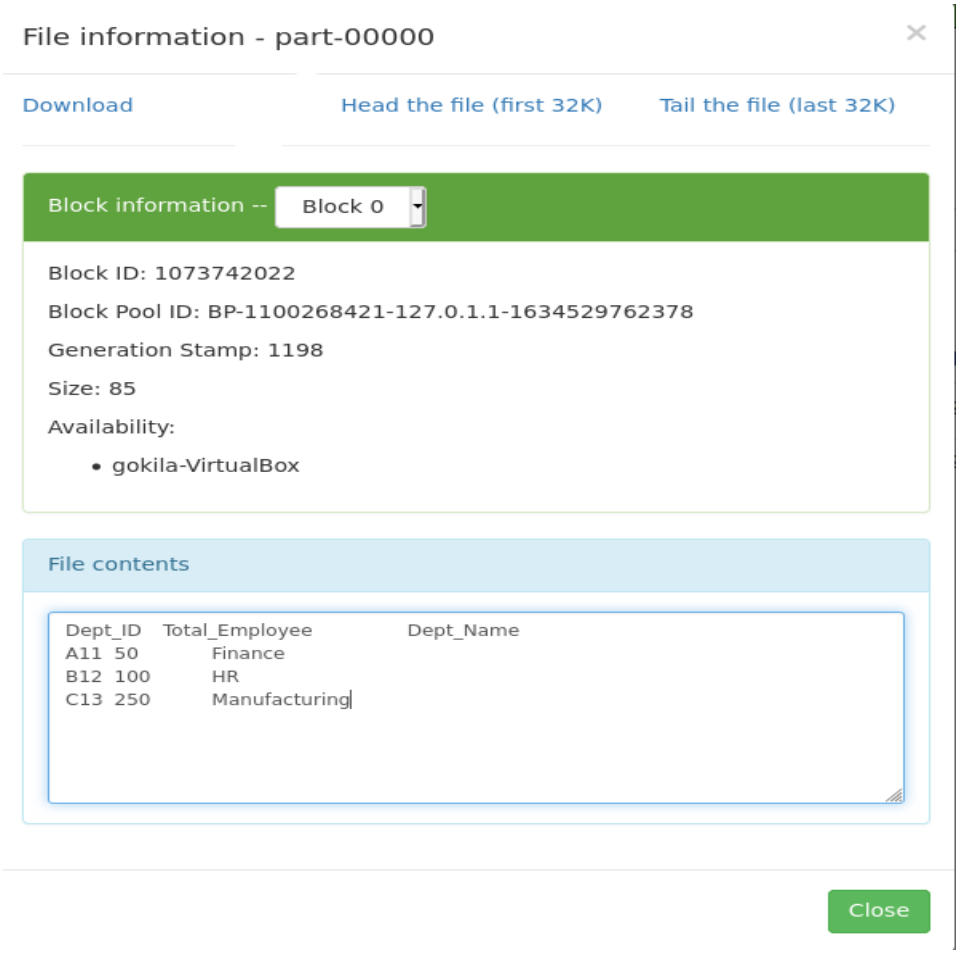
After execution, output file (named 'part-ooooo') will stored in the directory /output_mapreducejoin on HDFS

COMMANDS
$HADOOP_HOME/bin/hdfs dfs -cat /output_mapreducejoin/part-ooooo

```
gokilanm@gokila-VirtualBox:~$ $HADOOP_HOME/bin/hdfs dfs -cat /output_mapreducejoin/part-00000
A11     50              Finance
B12     100             HR
C13     250             Manufacturing
Dept_ID Total_Employee          Dept_Name
```

Results can be seen using the command line interface





RESULT:

The join operation in the streaming data can be implemented successfully.

Experiment 7: *Collect, aggregate and transport large amount of streaming data from Twitter data using Apache Flume*

a)*OBJECTIVE OF THE EXERCISE/EXPERIMENT*

To understand how Flume helps in streaming data from various sources.

b)*Facilities/material required to do the exercise/experiment:*

Sl.No.	*Facilities/material required*	*Quantity*
1.	*Hadoop*	*As per strength*
2.	*Flume*	*As per strength*

THEORY:

Apache Flume is a tool for data ingestion in HDFS. It collects, aggregates and transports large amount of streaming data such as log files, events from various sources like network traffic, social media, email messages etc. to HDFS. Flume is a highly reliable & distributed. The main idea behind the Flume's design is to capture streaming data from various web servers to HDFS. It has simple and flexible architecture based on streaming data flows. It is fault-tolerant and provides reliability mechanism for Fault tolerance & failure recovery.

Apache Flume

Apache Flume is an open-source tool for collecting, aggregating, and moving huge amounts of streaming data from the external web servers to the central store, say HDFS, HBase, etc. It is a highly available and reliable service which has tunable recovery mechanisms. The main purpose of designing Apache Flume is to move streaming data generated by various applications to Hadoop Distributed FileSystem.

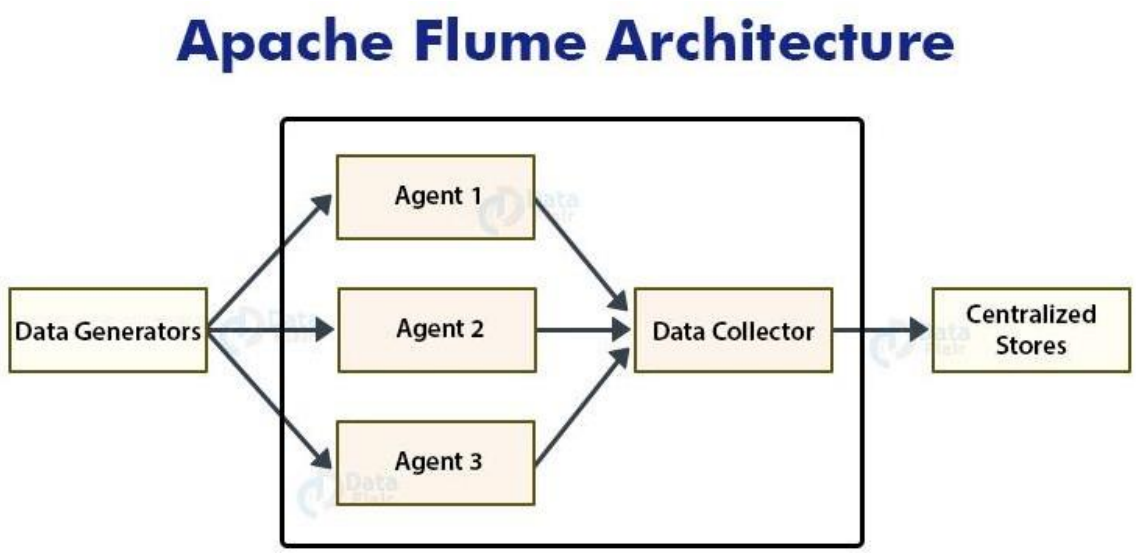
Reason of using Apache Flume

A company has millions of services that are running on multiple servers. Thus, produce lots of logs. In order to gain insights and understand customer behavior, they need to analyse these logs altogether. To process logs, a company requires an extensible, scalable, and reliable distributed data collection service. That service must be capable of performing the flow of unstructured data such as logs from source to the system where they will be processed (such as in Hadoop Distributed FileSystem). Flume is an open-source distributed data collection service used for transferring the data from source to destination.

It is a reliable, and highly available service for collecting, aggregating, and transferring huge amounts of logs into HDFS. It has a simple and flexible architecture. Apache Flume is highly robust and fault-tolerant and has tunable reliability mechanisms for fail-over and recovery. It allows the collection ofdata collection in batch as well as in streaming mode.

Features of Apache Flume

- *Apache Flume is a robust, fault-tolerant, and highly available service.*
- *It is a distributed system with tunable reliability mechanisms for fail- over and recovery.*
- *Apache Flume is horizontally scalable.*
- *Apache Flume supports complex data flows such as multi-hop flows, fan-in flows, fan-out flows. Contextual routing etc.*
- *Apache Flume provides support for large sets of sources, channels, and sinks.*
- *Apache Flume can efficiently ingest log data from various servers into a centralized repository.*
- *With Flume, we can collect data from different web servers in real- time as well as in batch mode.*
- *We can import large volumes of data generated by social networking sites and e-commerce sites into Hadoop DFS using Apache Flume.*



Flume Event

A Flume event is a basic unit of data that needs to be transferred from source to destination.

Flume Agent

Flume agent is an independent JVM process (JVM) in Apache Fl

Source

A source receives data from the data generators. It transfers the received data to one or more channels in the form of events. Flume provides support for several types of sources.

Example – Exec source, Thrift source, Avro source, twitter 1% source, etc. Channel

A channel receives the data or events from the flume source and buffers them till the sinks consume them. It is a transient store. Flume supports different types of channels.

Example – Memory channel, File system channel, JDBC channel, etc. Sink

A sink consumes data from the channel and stores them into the destination. The destination can be a centralized store or other flume agents.

Example – HDFS sink.
Additional Components of Flume Agent

There are few more components other than described above that play a significant role in transferring the events.

Interceptors

They alter or inspect flume events transferred between the flume source and channel.

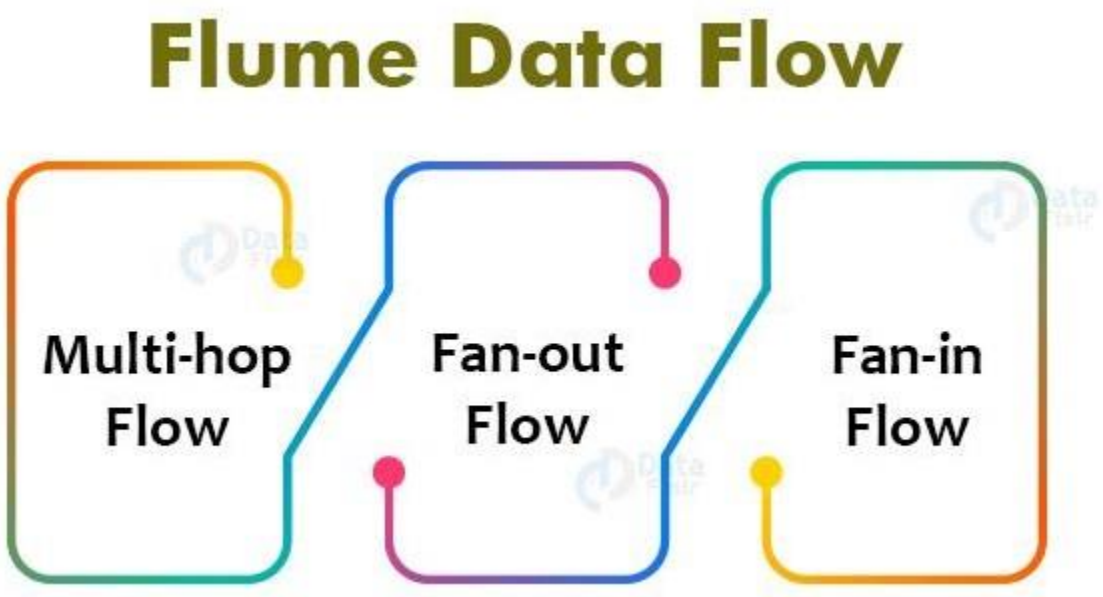
Channel Selectors

They determine which channel is to be chosen for transferring the data when multiple channels exist. Channel selectors are of two types- Default and multiplexing.

Sink Processors

Sink Processors invoke a particular sink from the group

of sinks. Apache Flume – Data Flow



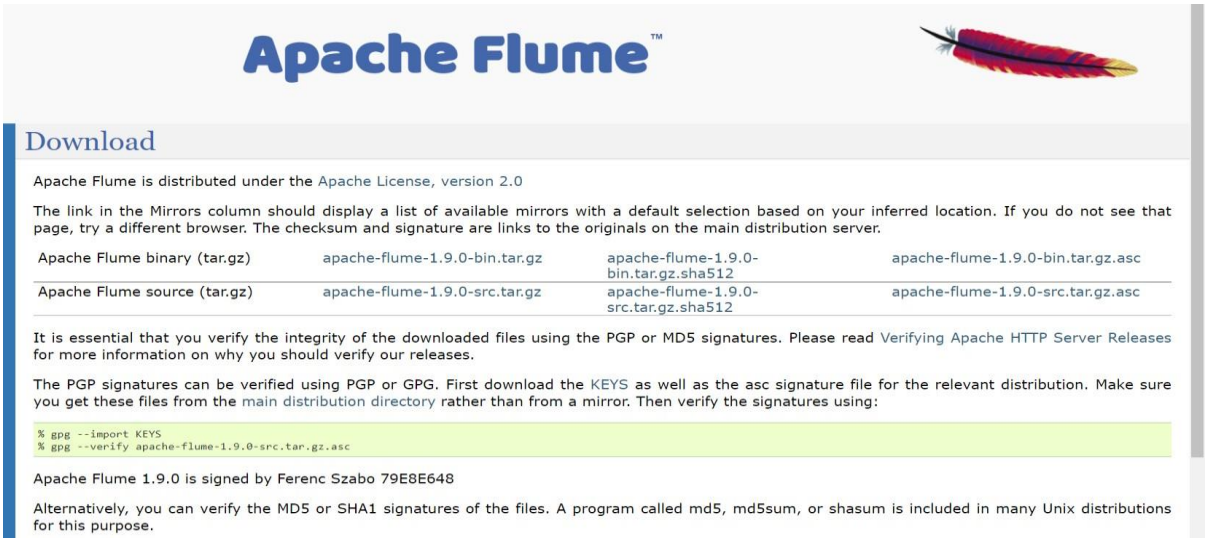
Data ingestion is the initial & important step in order to process & analyze data and then derive business values out of it. There are be multiple sources from which data is gathered in an organization.

Setting up Flume to analyse Twitter data

First of all, download the latest version of Apache Flume software from the website https://flume.apache.org/

Step 1

Open the website. Click on the download link on the left-hand side of the home page. It will take you to the download page of Apache Flume.



In the Download page, you can see the links for binary and source files of Apache Flume. Click on the link apache-flume-1.9.0-bin.tar.gzYou will be redirected to a list of mirrors where you can start your download by clicking any of these mirrors.

Download the flume-sources-1.0-SNAPSHOT.jar and add it to the Flume class path (the JAR contains the Java classes to pull the tweets and save them into HDFS).

STEP-2:

Extract the file from the Flume tar file, as follows:

COMMANDS
tar -xvf apache-flume-1.9.0-bin.tar.gz

STEP-3

Download the flume-sources-1.0-SNAPSHOT.jar and add it to the Flume class path (the JAR contains the Java classes to pull the tweets and save them into HDFS):

COMMANDS
sudo mv Downloads/flume-sources-1.0-SNAPSHOT.jar /apache-flume-1.9.0-bin/lib/

STEP-4

Check whether the Flume snapshot has moved to the lib folder of Apache Flume:

COMMANDS
*ls /usr/lib/apache-flume-1.9.0-bin/lib/flume**

STEP-5:

Configuring Flume

To configure Flume, we have to modify three files namely, flume-env.sh, flumeconf.properties, and bash.rc.

Setting the Path / Classpath

In the .bashrc file, set the home folder, the path, and the classpath for Flume as /shown below

COMMANDS
sudo nano ~/.bashrc

```
gokilanm@gokila-VirtualBox:~$ sudo nano ~/.bashrc
```

```
Firefox Web Browser
/home/gokilanm/hadoop-3.2.2/Flume
$FLUME_HOME/bin
export CLASSPATH=$CLASSPATH:/FLUME_HOME/lib/*

export HIVE_HOME=/home/gokilanm/apache-hive-3.1.2-bin
export HIVE_CONF_DIR=/home/gokilanm/apache-hive-3.1.2-bin/conf
export PATH=$HIVE_HOME/bin:$PATH
export CLASSPATH=$CLASSPATH:/home/gokilanm/hadoop-3.2.2/lib/*:.
export CLASSPATH=$CLASSPATH:/home/gokilanm/apache-hive-3.1.2-bin/lib/*:.
export DERBY_HOME=/home/gokilanm/db-derby-10.13.1.1-bin
export PATH=$PATH:$DERBY_HOME/bin
export CLASSPATH=$CLASSPATH:$DERBY_HOME/lib/derby.jar:$DERBY_HOME/lib/derbytools.jar
```

conf Folder

If you open the conf folder of Apache Flume, you will have the following four files –

flume-

conf.properties.template,

flume-env.sh.template,

flume-env.ps1.template,

and log4j.properties.

Now rename

1.flume-conf.properties.template file as flume-conf.properties and

2.flume-env.sh.template as flume-

env.sh flume-env.sh

Open flume-env.sh file and set the JAVA_Home to the folder where Java was installed in your system.

COMMANDS
sudo nano flume-env.sh

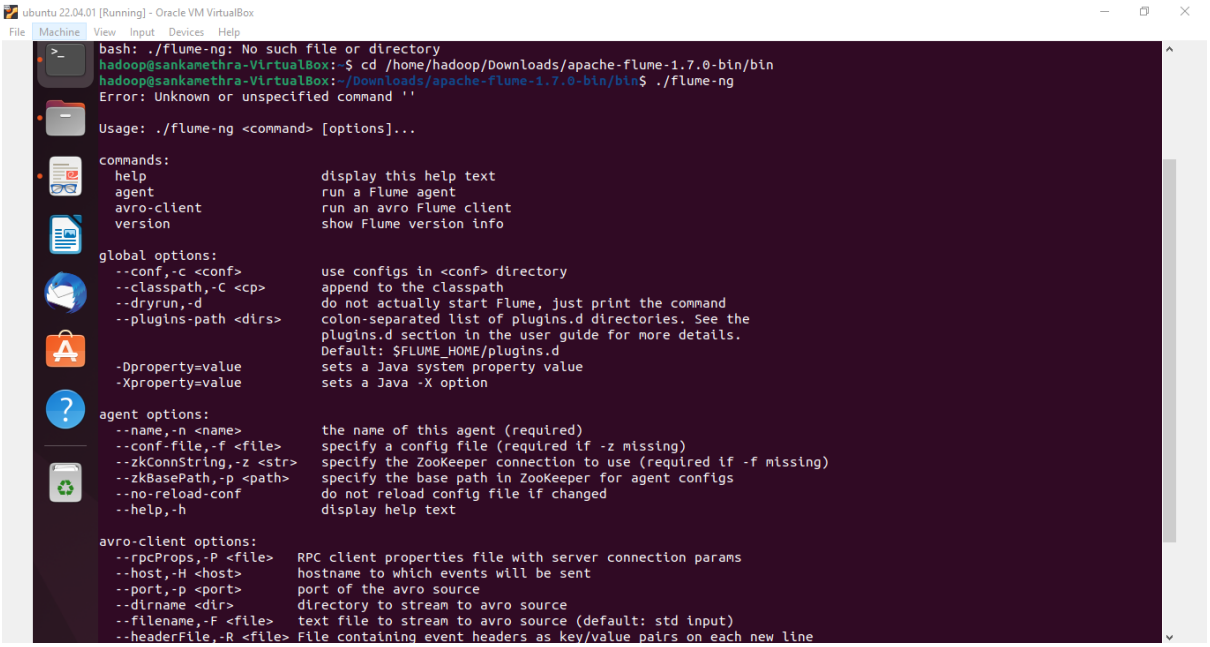
```
# Enviroment variables can be set here.
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
export CLASSPATH=$CLASSPATH:/FLUME_HOME/lib/*

# Give Flume more memory and pre-allocate, enable remote monitoring via JMX
# export JAVA_OPTS="-Xms100m -Xmx2000m -Dcom.sun.management.jmxremote"

# Let Flume write raw event data and configuration information to its log files for debugging
# purposes. Enabling these flags is not recommended in production,
# as it may result in logging sensitive user information or encryption secrets.
# export JAVA_OPTS="$JAVA_OPTS -Dorg.apache.flume.log.rawdata=true -Dorg.apache.flume.log.printconfig=true "

# Note that the Flume conf directory is always included in the classpath.
FLUME_CLASSPATH="/home/gokilanm/apache-flume-1.9.0-bin/lib/flume-sources-1.0-SNAPSHOT.jar"
```

Verifying the Installation

Verify the installation of Apache Flume by browsing through the bin folder and typing the following command.

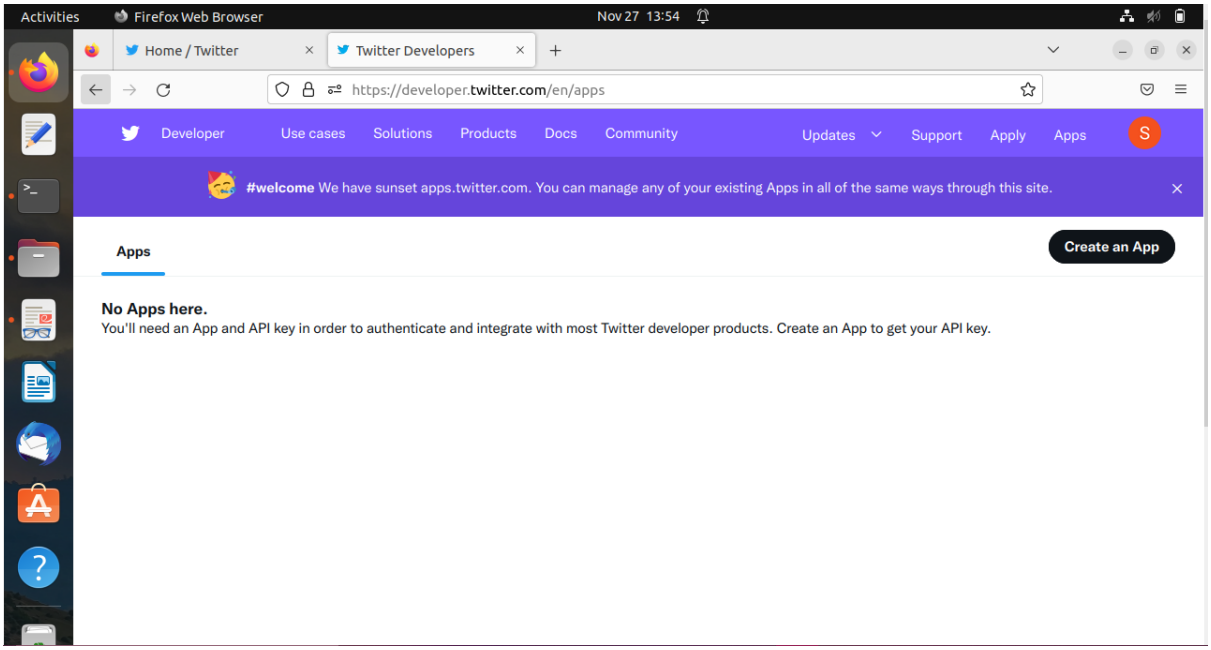
COMMANDS
./flume-ng

If you have successfully installed Flume, you will get a help prompt of Flume as shown below.

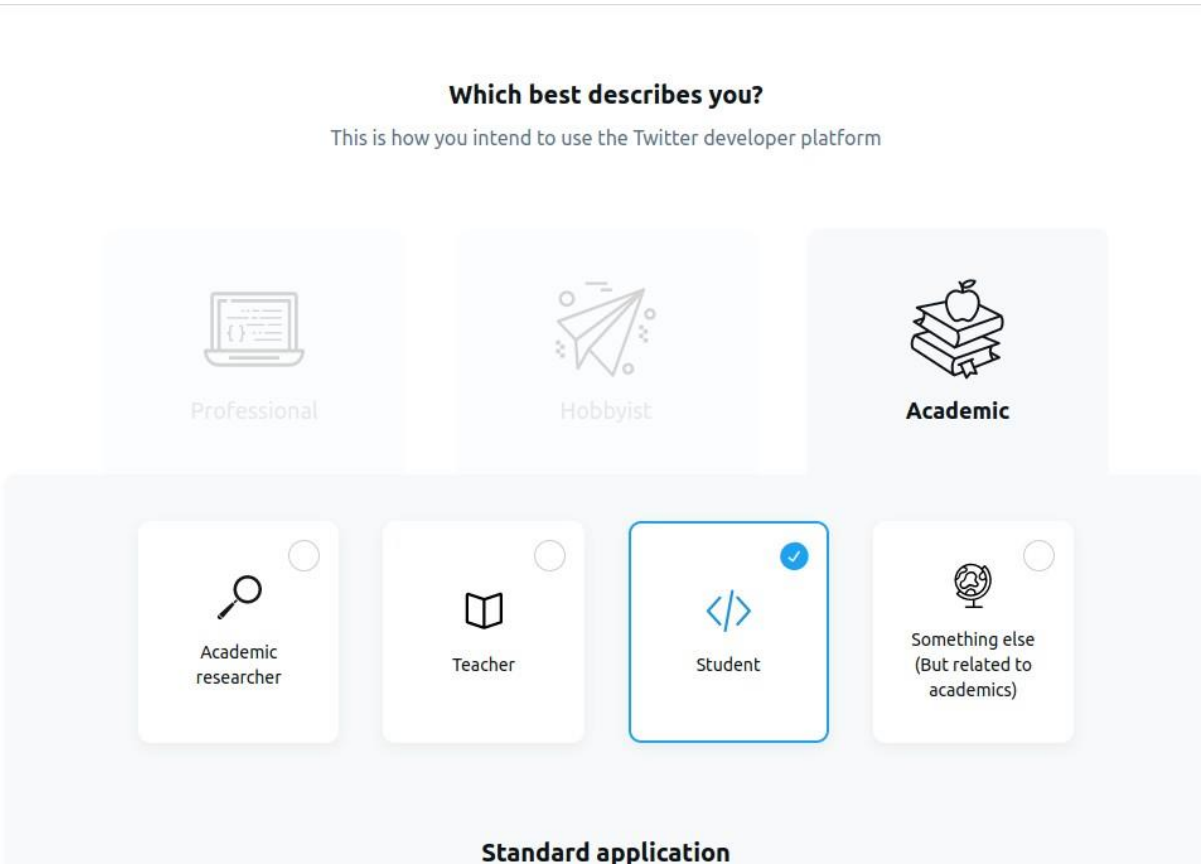


Apache Flume : Streaming Twitter Data

The first step is to create a Twitter application. For this, you first have to go to this url: https://apps.twitter.com/ and sign in to your Twitter account. Go to create application tab as shown in the below image.



Click "Academic",then choose "student"



Fill the details

The specifics

Please answer each of the following with as much detail and accuracy as possible. Failure to do so could result in delays to your access to Twitter developer platform or rejected applications.

Are you planning to analyze Twitter data? Yes

Please describe how you will analyze Twitter data including any analysis of Tweets or Twitter users.

I am going to use hadoop apache flume and natural language processing in Python to analyse real-time tweets.

Will your app use Tweet, Retweet, Like, Follow, or Direct Message functionality? Yes

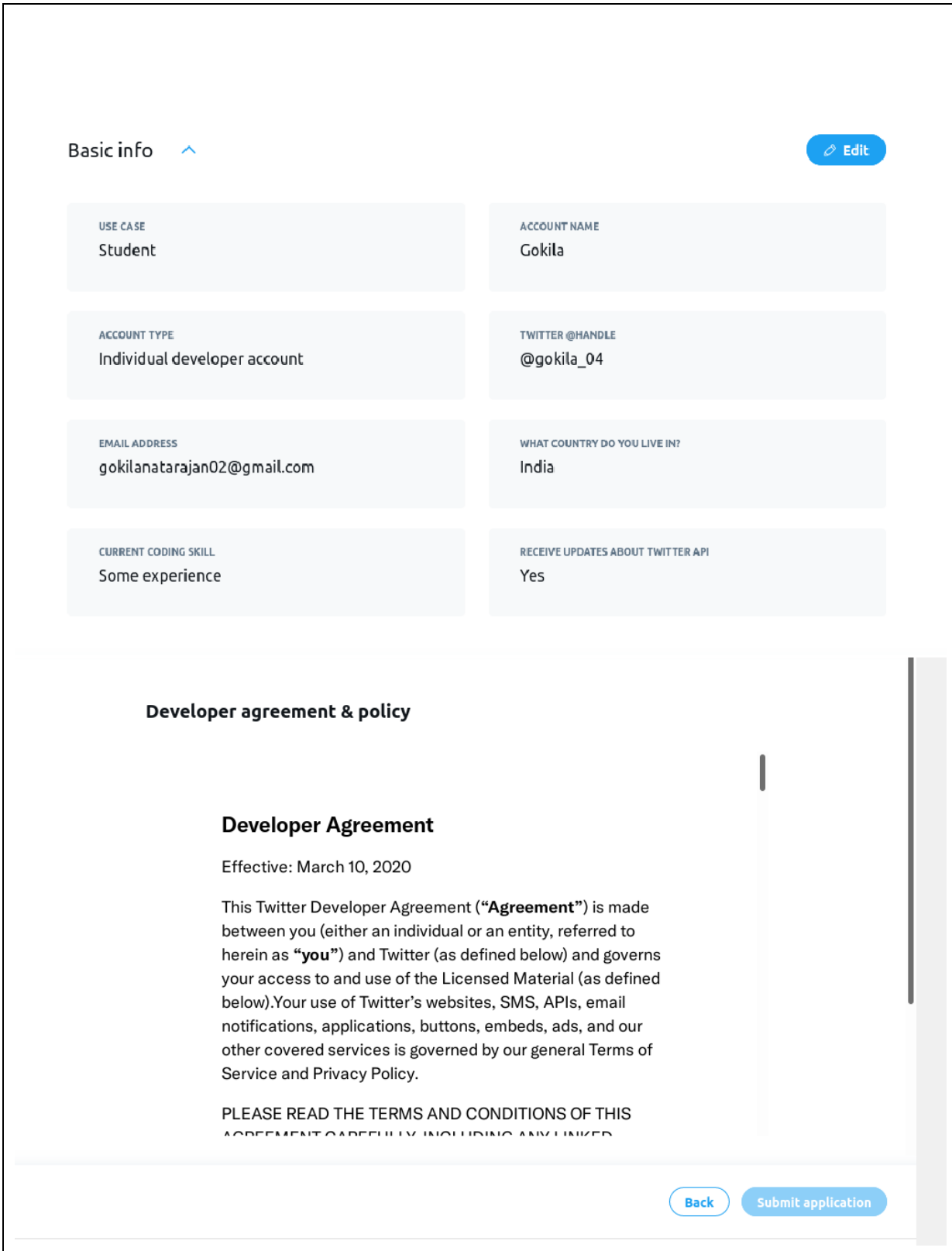
Please describe your planned use of these features.

I am going to do configuration large amount of data into hadoop and if needed in my project I will also analyse the tweet and retweet.

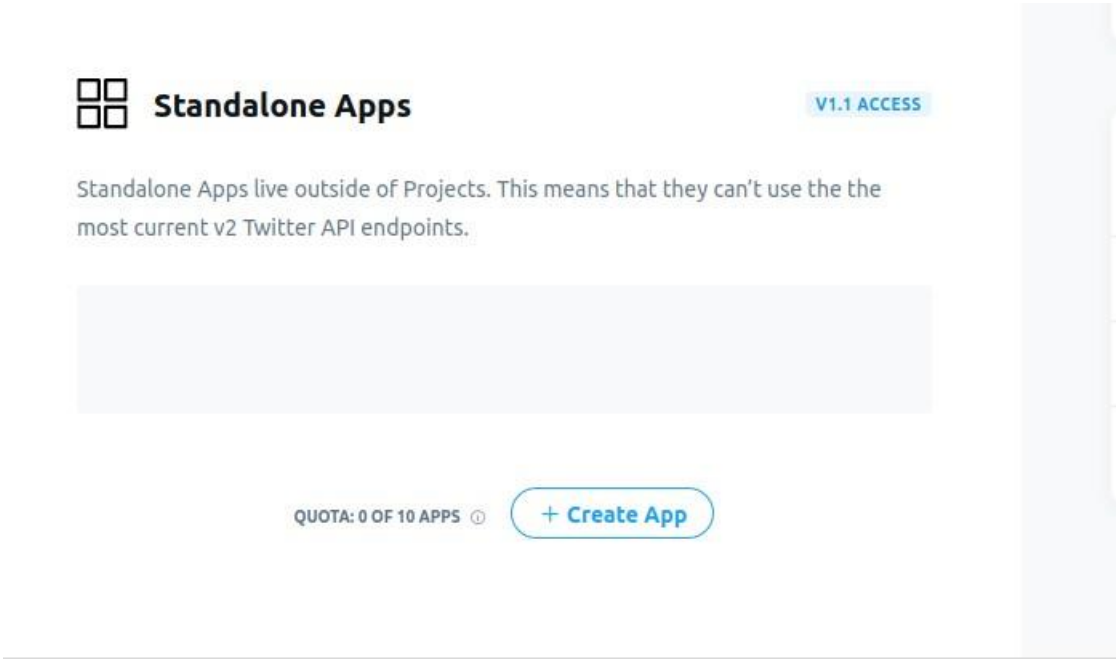
Do you plan to display Tweets or aggregate data about Twitter content outside Twitter? Yes

Please describe how and where Tweets and/or data about Twitter content will be displayed outside of Twitter.

Yes, I am going to use data aggregated data and summary in my project report with my group and we will use the number in the project presentation.


Basic info
Edit
USE CASE
Student
ACCOUNT NAME
Gokila
ACCOUNT TYPE
Individual developer account
TWITTER @HANDLE
@gokila_04
EMAIL ADDRESS
gokilanatarajan02@gmail.com
WHAT COUNTRY DO YOU LIVE IN?
India
CURRENT CODING SKILL
Some experience
RECEIVE UPDATES ABOUT TWITTER API
Yes
Developer agreement & policy
Developer Agreement
Effective: March 10, 2020
This Twitter Developer Agreement (“Agreement”) is made between you (either an individual or an entity, referred to herein as “you”) and Twitter (as defined below) and governs your access to and use of the Licensed Material (as defined below).Your use of Twitter’s websites, SMS, APIs, email notifications, applications, buttons, embeds, ads, and our other covered services is governed by our general Terms of Service and Privacy Policy.
PLEASE READ THE TERMS AND CONDITIONS OF THIS
Back
Submit application

If you are new to the Developers site you won't see any applications registered. Either way, it's time to create our first application. To do this, click on the big "Create a new application" button.



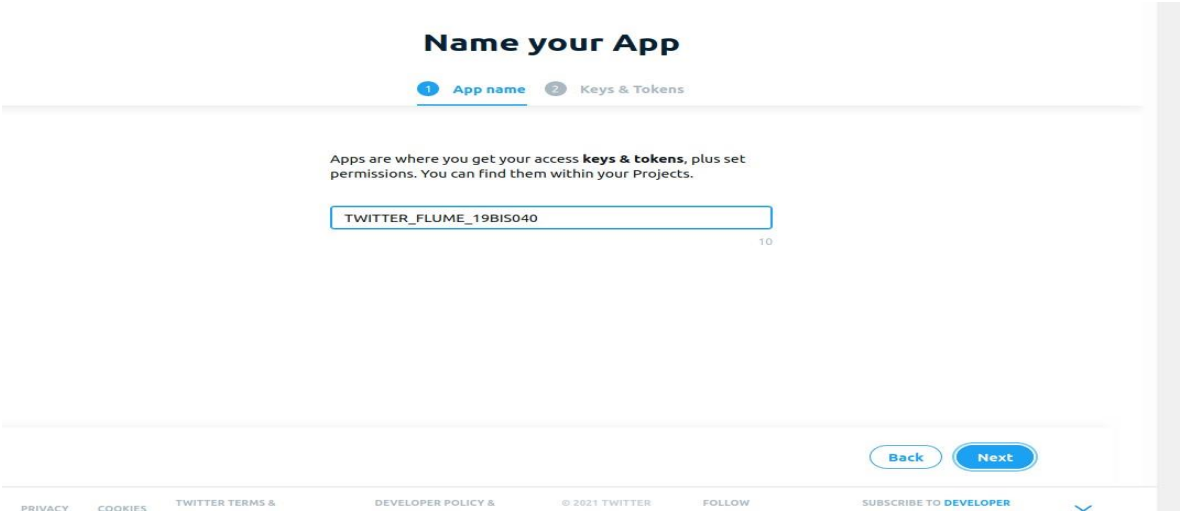
Name your app:

Name:

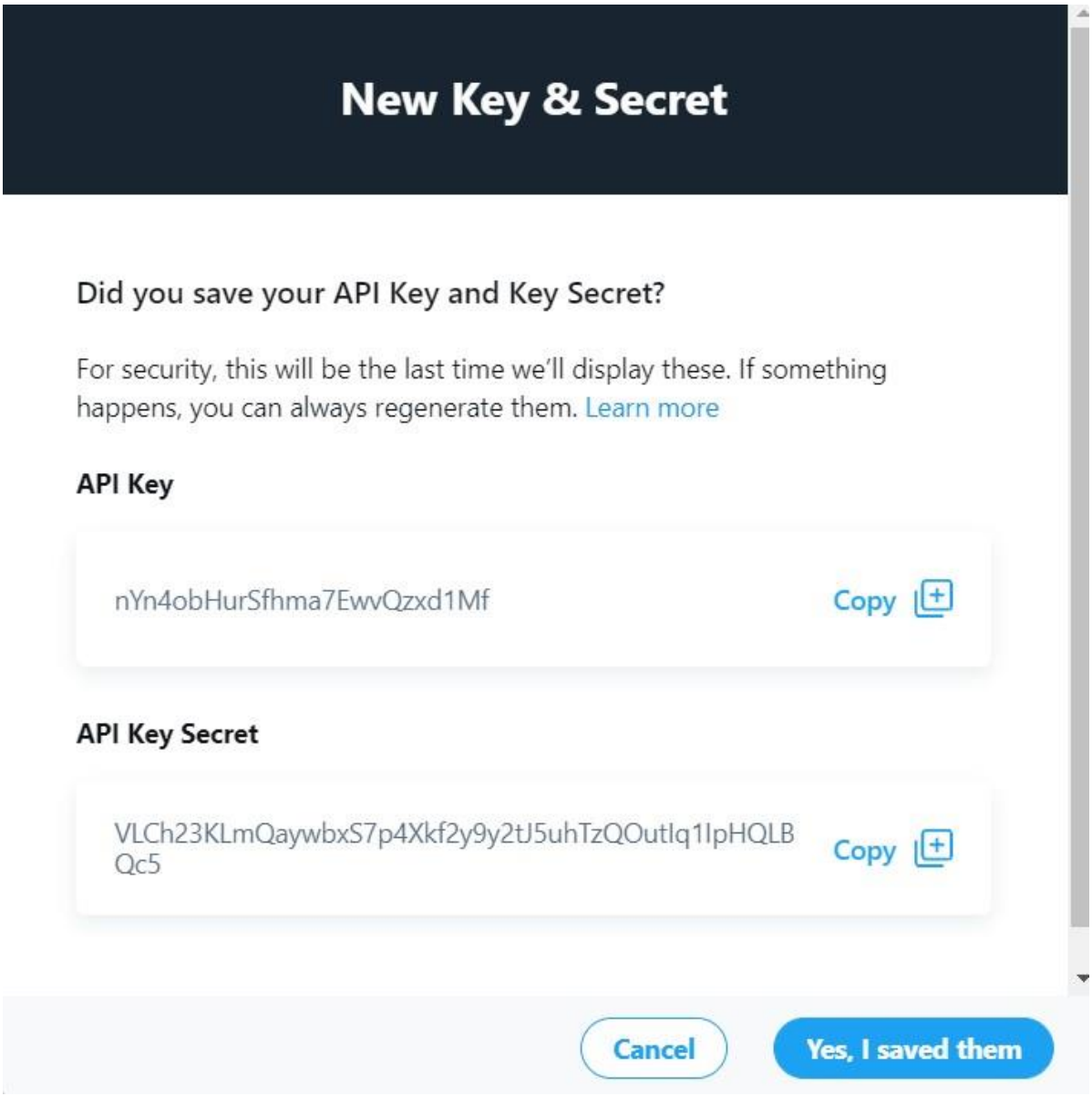
First of all you need to give your app a unique name (one that no one else has used for their Twitter app). Since we're going to be an creating app for personal use and not one that other people can register and use, just put your domain name in or perhaps even your name.

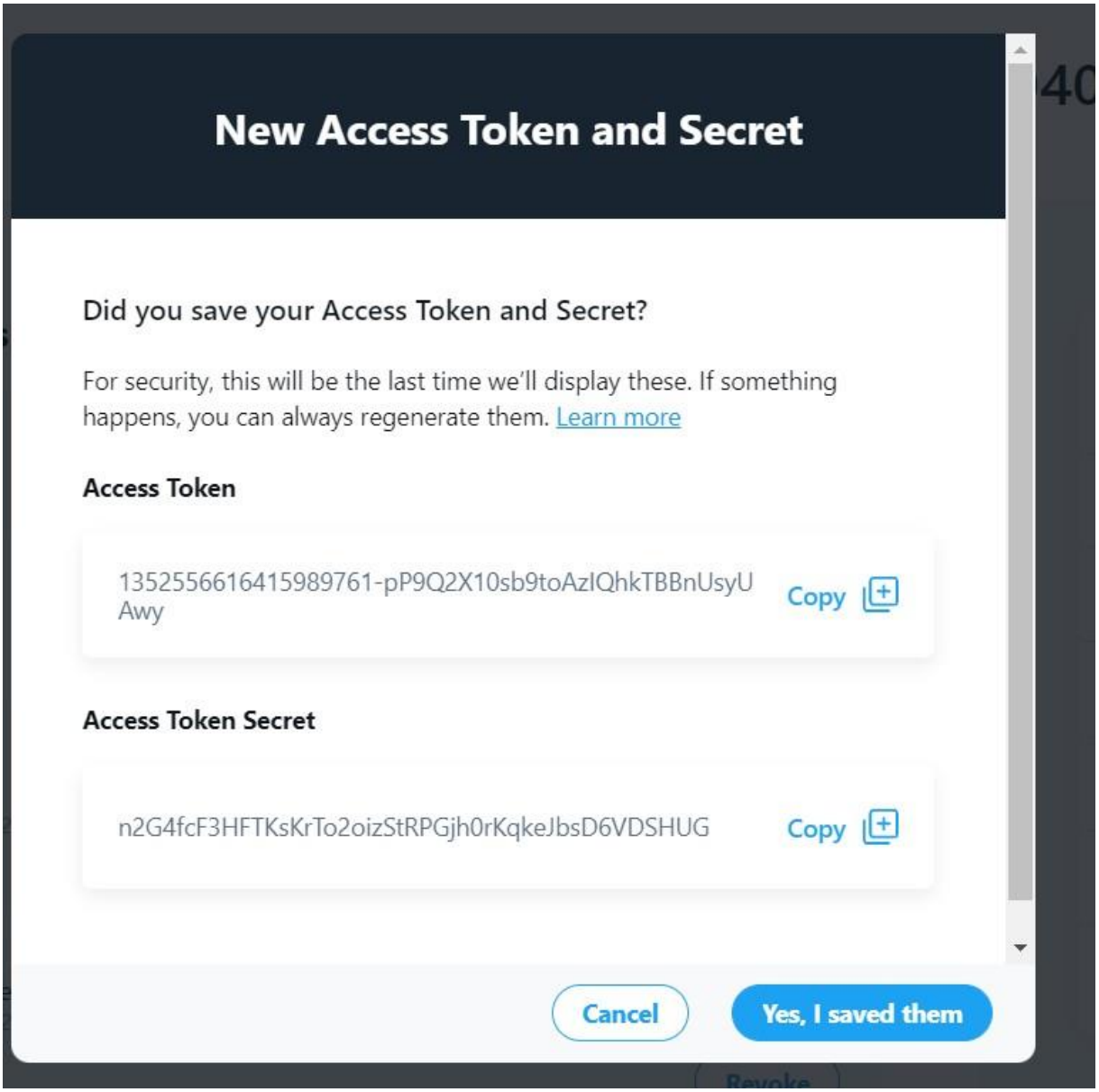
Description:

You don't have to worry much about the description- you can change this later, but it can help to distinguish this app from others that you might create in the future. I've put "a set of Twitter tools for personal use".



Now create a flume.conf file in the flume's root directory as shown in the below image. As we discussed, in the Flume's Architecture, we will configure our Source, Sink and Channel. Our Source is Twitter, from where we are streaming the data and our Sink is HDFS, where we are writing the data.





SOURCE CODE:

 # Naming the components on the current agent. TwitterAgent.sources = Twitter

TwitterAgent.channels = MemChannel

TwitterAgent.sinks = HDFS

TwitterAgent.sources.Twitter.channels = MemChannel TwitterAgent.sinks.HDFS.channel = MemChannel

 # Describing/Configuring the source

TwitterAgent.sources.Twitter.type= org.apache.flume.source.twitter.TwitterSource

TwitterAgent.sources.Twitter.consumerKey= SN5f21STlXEV8ZC1wuL78Aftc

TwitterAgent.sources.Twitter.consumerSecret= bounWv4jpXqjshIdUcKH1BL08lU4GvAV3ax2RoRXRDNaf2u YNb

TwitterAgent.sources.Twitter.accessToken= 1352556616415989761- aDJel5M1kFdxKMjyuJL2grEFONlkJt

TwitterAgent.sources.Twitter.accessTokenSecret= Bx9J53CqeJbyKdWvnUh1t9OHov313Ia4XgofTH0Lf3 WLP

TwitterAgent.sources.Twitter.keywords= tutorials

point,java, nosql # Describing/Configuring the sink

TwitterAgent.sinks.HDFS.type = hdfs

TwitterAgent.sinks.HDFS.hdfs.path =

hdfs://localhost:9000/twitter

TwitterAgent.sinks.HDFS.hdfs.fileType = DataStream

TwitterAgent.sinks.HDFS.hdfs.writeFormat = Text

TwitterAgent.sinks.HDFS.hdfs.batchSize = 1000

TwitterAgent.sinks.HDFS.hdfs.rollSize = 0

TwitterAgent.sinks.HDFS.hdfs.rollCount = 10000

TwitterAgent.sinks.HDFS.hdfs.rollCount =

600 # Describing/Configuring the channel

TwitterAgent.channels.MemChannel.type =

memory

TwitterAgent.channels.MemChannel.capacity =

10000

TwitterAgent.channels.MemChannel.byteCapacity = 6912212

TwitterAgent.channels.MemChannel.transactionCapacity = 1000

SNAP:

```
# Naming the components on the current agent.
TwitterAgent.sources = Twitter
TwitterAgent.channels = MemChannel
TwitterAgent.sinks = HDFS

# Describing/Configuring the source
TwitterAgent.sources.Twitter.type = org.apache.flume.source.twitter.TwitterSource
TwitterAgent.sources.Twitter.consumerKey = VjsLL2Pk74k9VKk9fZRwsWjFi
TwitterAgent.sources.Twitter.consumerSecret = czjZQl6sV6yWQwBD6qsdJGZTRLt2BX3fSrbBB7qIXTvWqfyvv4
TwitterAgent.sources.Twitter.accessToken = 1352556616415989761-QpHxKbpRUsLqbuaaZGTS1ffUtnuGQ5
TwitterAgent.sources.Twitter.accessTokenSecret = 9s779mRHzxhBHa2jSG20aWiPie1wyP6PA0Ygdu1rrJrYA
TwitterAgent.sources.Twitter.keywords = tutorials point,java, bigdata, mapreduce, mahout,
  hbase, nosql

# Describing/Configuring the sink

TwitterAgent.sinks.HDFS.type = hdfs
TwitterAgent.sinks.HDFS.hdfs.path = hdfs://localhost:9000/user/twitter_data/
TwitterAgent.sinks.HDFS.hdfs.fileType = DataStream
TwitterAgent.sinks.HDFS.hdfs.writeFormat = Text
TwitterAgent.sinks.HDFS.hdfs.batchSize = 1000
TwitterAgent.sinks.HDFS.hdfs.rollSize = 0
TwitterAgent.sinks.HDFS.hdfs.rollCount = 10000

# Describing/Configuring the channel
TwitterAgent.channels.MemChannel.type = memory
TwitterAgent.channels.MemChannel.capacity = 10000
TwitterAgent.channels.MemChannel.transactionCapacity = 100

# Binding the source and sink to the channel
TwitterAgent.sources.Twitter.channels = MemChannel
TwitterAgent.sinks.HDFS.channel = MemChannel
```

SOURCE CODE-EXPLANATION:

This source needs the details such as Consumer key, Consumer secret, Access token, and Access token secret of a Twitter application. While configuring this source, you have to provide values to the following properties –

Channels

Source type :

org.apache.flume.source.twitter.TwitterSource

consumerKey – The OAuth consumer key

consumerSecret – OAuth consumer

secret accessToken – OAuth access

token accessTokenSecret – OAuth

token secret

maxBatchSize – Maximum number of twitter messages that should be in a twitter batch. The default value is 1000 (optional).

maxBatchDurationMillis – Maximum number of milliseconds to wait before closing a batch. The default value is 1000 (optional).

Channel

We are using the memory channel. To configure the memory channel, you must provide value to the type of the channel.

type – It holds the type of the channel. In our example, the type is MemChannel.

Capacity – It is the maximum number of events stored in the channel. Its default value is 100 (optional).

TransactionCapacity – It is the maximum number of events the channel accepts or sends. Its default value is 100 (optional).

HDFS Sink

This sink writes data into the HDFS. To configure this sink, you must provide the following details.

Channel

type –

hdfs

hdfs.path – the path of the directory in HDFS where data is to be stored.

And we can provide some optional values based on the scenario. Given below are the optional properties of the HDFS sink that we are configuring in our application.

fileType – This is the required file format of our HDFS file. SequenceFile, DataStream and CompressedStream are the three types available with this stream. In our example, we are using the DataStream.

writeFormat – Could be either text or writable.

batchSize – It is the number of events written to a file before it is flushed into the HDFoS. Its default value is 100.

rollsize – It is the file size to trigger a roll. It default value is 100.

rollCount – It is the number of events written into the file before it is rolled. Its default value is 10.

Browse through the Flume home directory and execute the application as shown below.

COMMANDS
flume-ng agent -n TwitterAgent ./conf/ -f $FLUME_HOME/conf/twitter.conf

```
gokilanm@gokila-VirtualBox:/usr/lib/apache-flume-1.9.0-bin/conf$ flume-ng agent -n TwitterAgent ./conf/ -f $FLUME_HOME/conf/twitter.conf
Warning: No configuration directory set! Use --conf <dir> to override.
Info: Including Hadoop libraries found via (/home/gokilanm/hadoop-3.2.2/bin/hadoop) for HDFS access
/home/gokilanm/hadoop-3.2.2/libexec/hadoop-functions.sh: line 2366: HADOOP_ORG.APACHE.FLUME.TOOLS.GETJAVAPROPERTY_USER: invalid variable name
/home/gokilanm/hadoop-3.2.2/libexec/hadoop-functions.sh: line 2461: HADOOP_ORG.APACHE.FLUME.TOOLS.GETJAVAPROPERTY_OPTS: invalid variable name
Info: Including Hive libraries found via (/usr/local/hive) for Hive access
+ exec /usr/lib/jvm/java-8-openjdk-amd64/bin/java -Xmx20m -cp '/usr/lib/apache-flume-1.9.0-bin/lib/*:/home/gokilanm/hadoop-3.2.2/etc/hadoop:/home/gokilanm/hadoop-3.2.2/share/hadoop/common/lib/*:/home/gokilanm/hadoop-3.2.2/share/hadoop/common/*:/home/gokilanm/hadoop-3.2.2/share/hadoop/hdfs:/home/gokilanm/hadoop-3.2.2/share/hadoop/hdfs/lib/*:/home/gokilanm/hadoop-3.2.2/share/hadoop/hdfs/*:/home/gokilanm/hadoop-3.2.2/share/hadoop/mapreduce/lib/*:/home/gokilanm/hadoop-3.2.2/share/hadoop/mapreduce/*:/home/gokilanm/hadoop-3.2.2/share/hadoop/yarn:/home/gokilanm/hadoop-3.2.2/share/hadoop/yarn/lib/*:/home/gokilanm/hadoop-3.2.2/share/hadoop/yarn/*:/usr/local/hive/lib/*' -Djava.library.path=:/home/gokilanm/hadoop-3.2.2/lib/native org.apache.flume.node.Application -n TwitterAgent ./conf/ -f /usr/lib/apache-flume-1.9.0-bin/conf/twitter.conf
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/lib/apache-flume-1.9.0-bin/lib/slf4j-log4j12-1.7.25.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/home/gokilanm/hadoop-3.2.2/share/hadoop/common/lib/slf4j-log4j12-1.7.25.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/local/hive/lib/log4j-slf4j-impl-2.10.0.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
2021-10-18 22:20:18,394 INFO node.PollingPropertiesFileConfigurationProvider: Configuration provider starting
2021-10-18 22:20:18,397 INFO node.PollingPropertiesFileConfigurationProvider: Reloading configuration file:/usr/lib/apache-flume-1.9.0-bin/conf/twitter.conf
2021-10-18 22:20:18,404 INFO conf.FlumeConfiguration: Processing:MemChannel
2021-10-18 22:20:18,406 INFO conf.FlumeConfiguration: Processing:Twitter
2021-10-18 22:20:18,406 INFO conf.FlumeConfiguration: Processing:Twitter
2021-10-18 22:20:18,406 INFO conf.FlumeConfiguration: Processing:HDFS
2021-10-18 22:20:18,406 INFO conf.FlumeConfiguration: Processing:HDFS
2021-10-18 22:20:18,406 INFO conf.FlumeConfiguration: Processing:MemChannel
2021-10-18 22:20:18,407 INFO conf.FlumeConfiguration: Processing:MemChannel
2021-10-18 22:20:18,407 INFO conf.FlumeConfiguration: Processing:HDFS
2021-10-18 22:20:18,407 INFO conf.FlumeConfiguration: Processing:HDFS
2021-10-18 22:20:18,407 INFO conf.FlumeConfiguration: Processing:MemChannel
2021-10-18 22:20:18,407 INFO conf.FlumeConfiguration: Processing:HDFS
2021-10-18 22:20:18,407 INFO conf.FlumeConfiguration: Added sinks: HDFS Agent: TwitterAgent
2021-10-18 22:20:18,407 INFO conf.FlumeConfiguration: Processing:Twitter
2021-10-18 22:20:18,407 INFO conf.FlumeConfiguration: Processing:Twitter
2021-10-18 22:20:18,407 INFO conf.FlumeConfiguration: Processing:HDFS
2021-10-18 22:20:18,407 INFO conf.FlumeConfiguration: Processing:HDFS
2021-10-18 22:20:18,407 INFO conf.FlumeConfiguration: Processing:Twitter
2021-10-18 22:20:18,408 INFO conf.FlumeConfiguration: Processing:Twitter
2021-10-18 22:20:18,408 INFO conf.FlumeConfiguration: Processing:Twitter
2021-10-18 22:20:18,408 INFO conf.FlumeConfiguration: Processing:HDFS
2021-10-18 22:20:18,408 WARN conf.FlumeConfiguration: Agent configuration for 'TwitterAgent' has no configfilters.
2021-10-18 22:20:18,483 INFO conf.FlumeConfiguration: Post-validation flume configuration contains configuration for agents: [TwitterAgent]
2021-10-18 22:20:18,484 INFO node.AbstractConfigurationProvider: Creating channels
```

```
2021-10-18 22:20:19,477 INFO instrumentation.MonitoredCounterGroup: Component type: SINK, name: HDFS started
2021-10-18 22:20:21,013 INFO twitter4j.TwitterStreamImpl: Connection established.
2021-10-18 22:20:21,013 INFO twitter4j.TwitterStreamImpl: Receiving status stream.
2021-10-18 22:20:21,534 INFO hdfs.BucketWriter: Creating hdfs://localhost:9000/twitter/FlumeData.1634575821185.tmp
2021-10-18 22:20:24,954 INFO twitter.TwitterSource: Processed 200 docs
^X2021-10-18 22:20:26,898 INFO twitter.TwitterSource: Processed 300 docs
c2021-10-18 22:20:28,932 INFO twitter.TwitterSource: Processed 400 docs

2021-10-18 22:20:30,987 INFO twitter.TwitterSource: Processed 500 docs
^C2021-10-18 22:20:31,639 INFO node.Application: Shutting down configuration: { sourceRunners:{Twitter=EventDrivenSourceRunner: { source:org.apache.flume.source.twitter.TwitterSource{name:Twitter,state:START} }} sinkRunners:{HDFS=SinkRunner: { policy:org.apache.flume.sink.DefaultSinkProcessor@39a62050 counterGroup:{ name:null counters:{} } }} channels:{MemChannel=org.apache.flume.channel.MemoryChannel{name: MemChannel}} }
2021-10-18 22:20:31,640 INFO node.Application: Stopping Source Twitter
2021-10-18 22:20:31,664 INFO lifecycle.LifecycleSupervisor: Stopping component: EventDrivenSourceRunner: { source:org.apache.flume.source.twitter.TwitterSource{name:Twitter,state:START} }
2021-10-18 22:20:31,677 INFO twitter.TwitterSource: Twitter source Twitter stopping...
2021-10-18 22:20:31,839 INFO twitter4j.TwitterStreamImpl: Inflater has been closed
```

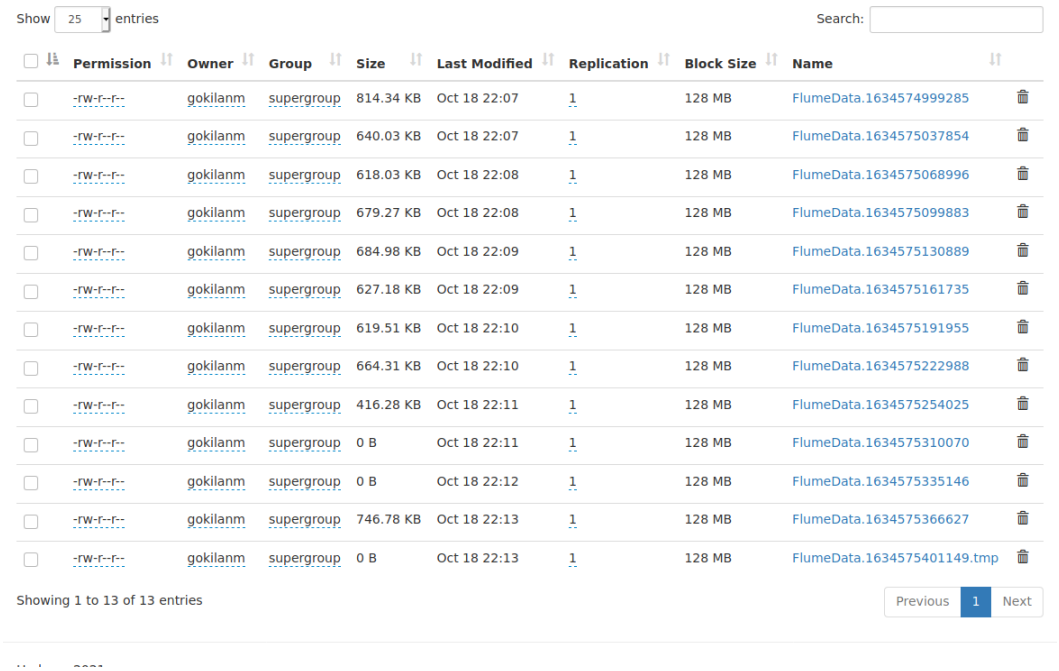
OUTPUT:

You can access the Hadoop Administration Web UI using the URL given below.

http://localhost:9870/

Click on the dropdown named Utilities on the right-hand side of the page. You can see two options as shown in the snapshot given below.

Click on Browse the file system and enter the path of the HDFS directory where you have stored the tweets. In our example, the path will be /user/Hadoop/twitter_data/. Then, you can see the list of twitter log files stored in HDFS as given below.



Show 25 entries Search:

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r--	gokilanm	supergroup	814.34 KB	Oct 18 22:07	1	128 MB	FlumeData.1634574999285
-rw-r--r--	gokilanm	supergroup	640.03 KB	Oct 18 22:07	1	128 MB	FlumeData.1634575037854
-rw-r--r--	gokilanm	supergroup	618.03 KB	Oct 18 22:08	1	128 MB	FlumeData.1634575068996
-rw-r--r--	gokilanm	supergroup	679.27 KB	Oct 18 22:08	1	128 MB	FlumeData.1634575099883
-rw-r--r--	gokilanm	supergroup	684.98 KB	Oct 18 22:09	1	128 MB	FlumeData.1634575130889
-rw-r--r--	gokilanm	supergroup	627.18 KB	Oct 18 22:09	1	128 MB	FlumeData.1634575161735
-rw-r--r--	gokilanm	supergroup	619.51 KB	Oct 18 22:10	1	128 MB	FlumeData.1634575191955
-rw-r--r--	gokilanm	supergroup	664.31 KB	Oct 18 22:10	1	128 MB	FlumeData.1634575222988
-rw-r--r--	gokilanm	supergroup	416.28 KB	Oct 18 22:11	1	128 MB	FlumeData.1634575254025
-rw-r--r--	gokilanm	supergroup	0 B	Oct 18 22:11	1	128 MB	FlumeData.1634575310070
-rw-r--r--	gokilanm	supergroup	0 B	Oct 18 22:12	1	128 MB	FlumeData.1634575335146
-rw-r--r--	gokilanm	supergroup	746.78 KB	Oct 18 22:13	1	128 MB	FlumeData.1634575366627
-rw-r--r--	gokilanm	supergroup	0 B	Oct 18 22:13	1	128 MB	FlumeData.1634575401149.tmp

Showing 1 to 13 of 13 entries Previous 1 Next

RESULT:

In short, Apache Flume is an open-source tool for collecting, aggregating, and moving huge amounts of data from the external web servers to the central store. Apache Flume is a highly available and reliable service. Apache Flume can be used for ingesting data from various applications to HDFS.It is useful for various e-commerce sites for understanding customer behavior. The Apache Flume Tutorial had explained the Flume architecture, data flow. It had also enlisted flume features, advantages, and disadvantages.

Experiment 8 : *Perform simple join using Mapper in Spark*

c) *OBJECTIVE OF THE EXERCISE/EXPERIMENT*

To Perform simple join using Mapper in Spark.

d) *Facilities/material required to do the exercise/experiment:*

Sl.No.	*Facilities/material required*	*Quantity*
1.	*Hadoop*	*As per strength*
2.	*Spark*	*As per strength*

THEORY:

Spark DataFrame supports all basic SQL Join Types like INNER, LEFT OUTER, RIGHT OUTER, LEFT ANTI, LEFT SEMI, CROSS,
SELF JOIN. Spark SQL Joins are wider transformations that result in data shuffling over the network hence they have huge performance issues when not designed with care.

JOIN QUERY:

Queries can access multiple tables at once, or access the same table in such a way that multiple rows of the table are being processed at the same time. A query that accesses multiple rows of the same or different tables at one time is called a join query.

Parameters

Relation:Specifies the relation to be

joined. join_type:Specifies the join

type.

Syntax:

[INNER] | CROSS | LEFT [OUTER] | [LEFT] SEMI | RIGHT [OUTER] | FULL [OUTER] | [LEFT] ANTI

join_criteria :Specifies how the rows from one relation will be combined with the rows of another relation.

Syntax:

ON boolean_expression | USING (column_name [, ...])

boolean_expression: Specifies an expression with a return type of boolean.

join Operators

join(right: Dataset[_]): DataFrame (1)

join(right: Dataset[_], usingColumn: String): DataFrame (2)

join(right: Dataset[_], usingColumns: Seq[String]): DataFrame (3)

join(right: Dataset[_], usingColumns: Seq[String], joinType: String): DataFrame (4)

join(right: Dataset[_], joinExprs: Column): DataFrame (5)

join(right: Dataset[_], joinExprs: Column, joinType: String):

DataFrame (6)
- *Condition-less inner join*
- *Inner join with a single column that exists on both sides*
- *Inner join with columns that exist on both sides*
- *Equi-join with explicit join type*
- *Inner join*
- *Join with explicit join type. Self-joins are acceptable.*

1.INNER JOIN:

The inner join is the default join in Spark SQL. It selects rows that have matching values in both relations.

Syntax:

relation [INNER] JOIN relation [join_criteria]

COMMANDS
1.val left = Seq((0, "zero"), (1, "one")).toDF("id", "left") *2.val right = Seq((0, "zero"), (2, "two"), (3, "three")).toDF("id", "right") 3.left.join(right, "id").show*

```
scala> val left = Seq((0, "zero"), (1, "one")).toDF("id", "left")
left: org.apache.spark.sql.DataFrame = [id: int, left: string]

scala> val right = Seq((0, "zero"), (2, "two"), (3, "three")).toDF("id", "right")
right: org.apache.spark.sql.DataFrame = [id: int, right: string]

scala> left.join(right, "id").show
+---+----+-----+
| id|left|right|
+---+----+-----+
|  0|zero| zero|
+---+----+-----+
```

COMMANDS
left.join(right, "id").explain

```
scala> left.join(right, "id").explain
== Physical Plan ==
*(1) Project [id#7, left#8, right#19]
+- *(1) BroadcastHashJoin [id#7], [id#18], Inner, BuildLeft, false
   :- BroadcastExchange HashedRelationBroadcastMode(List(cast(input[0, int, false] as bigint)),false), [id=#42]
   :  +- LocalTableScan [id#7, left#8]
   +- *(1) LocalTableScan [id#18, right#19]
```

2.FULL OUTER JOIN:

In SQL the FULL OUTER JOIN combines the results of both left and right outer joins and returns all (matched or unmatched) rows from the tables on both sides of the join clause.

The FULL OUTER JOIN keyword returns all records when there is a match in left (table1) or right (table2) table records.

COMMANDS
left.join(right, Seq("id"), "fullouter").show

```
scala> left.join(right, Seq("id"), "fullouter").show
+---+----+-----+
| id|left|right|
+---+----+-----+
|  1| one| null|
|  3|null|three|
|  2|null|  two|
|  0|zero| zero|
+---+----+-----+
```

COMMANDS
left.join(right, Seq("id"), "fullouter").explain

```
scala>  left.join(right, Seq("id"), "fullouter").explain
== Physical Plan ==
*(3) Project [coalesce(id#7, id#18) AS id#58, left#8, right#19]
+- SortMergeJoin [id#7], [id#18], FullOuter
   :- *(1) Sort [id#7 ASC NULLS FIRST], false, 0
   :  +- Exchange hashpartitioning(id#7, 200), ENSURE_REQUIREMENTS, [id=#102]
   :     +- LocalTableScan [id#7, left#8]
   +- *(2) Sort [id#18 ASC NULLS FIRST], false, 0
      +- Exchange hashpartitioning(id#18, 200), ENSURE_REQUIREMENTS, [id=#103]
         +- LocalTableScan [id#18, right#19]
```

```
scala> case class Person(id: Long, name: String, cityId: Long)
defined class Person

scala> case class City(id: Long, name: String)
defined class City

scala> val family = Seq(
     |   Person(0, "Agata", 0),
     |   Person(1, "Iweta", 0),
     |   Person(2, "Patryk", 2),
     |   Person(3, "Maksym", 0)).toDS
family: org.apache.spark.sql.Dataset[Person] = [id: bigint, name: string ... 1 more field]

scala> val cities = Seq(
     |   City(0, "Warsaw"),
     |   City(1, "Washington"),
     |   City(2, "Sopot")).toDS
cities: org.apache.spark.sql.Dataset[City] = [id: bigint, name: string]

scala>

scala> val joined = family.joinWith(cities, family("cityId") === cities("id"))
joined: org.apache.spark.sql.Dataset[(Person, City)] = [_1: struct<id: bigint, name: string ... 1 more field>, _2: struct<id: bigint, name: string>]

scala> joined.printSchema
root
 |-- _1: struct (nullable = false)
 |    |-- id: long (nullable = false)
 |    |-- name: string (nullable = true)
 |    |-- cityId: long (nullable = false)
 |-- _2: struct (nullable = false)
 |    |-- id: long (nullable = false)
 |    |-- name: string (nullable = true)
```

```
scala> val joined = family.joinWith(cities, family("cityId") === cities("id"))
joined: org.apache.spark.sql.Dataset[(Person, City)] = [_1: struct<id: bigint, name: string ... 1 more field>, _2: struct<id: bigint, name: string>]

scala> joined.printSchema
root
 |-- _1: struct (nullable = false)
 |    |-- id: long (nullable = false)
 |    |-- name: string (nullable = true)
 |    |-- cityId: long (nullable = false)
 |-- _2: struct (nullable = false)
 |    |-- id: long (nullable = false)
 |    |-- name: string (nullable = true)


scala> joined.show
+-------------+-----------+
|           _1|         _2|
+-------------+-----------+
| {0, Agata, 0}|{0, Warsaw}|
| {1, Iweta, 0}|{0, Warsaw}|
|{2, Patryk, 2}| {2, Sopot}|
|{3, Maksym, 0}|{0, Warsaw}|
+-------------+-----------+


scala>
```

3.LEFT ANTI-JOIN

A left anti join returns that all rows from the first table which do not have a match in the second table.

COMMANDS
left.join(right, Seq("id"), "leftanti").show

```
scala> left.join(right, Seq("id"), "leftanti").show
+---+----+
| id|left|
+---+----+
|  1| one|
+---+----+
```

COMMANDS
left.join(right, Seq("id"), "leftanti").explain

```
scala> left.join(right, Seq("id"), "leftanti").explain
== Physical Plan ==
*(1) BroadcastHashJoin [id#7], [id#18], LeftAnti, BuildRight, false
:- *(1) LocalTableScan [id#7, left#8]
+- BroadcastExchange HashedRelationBroadcastMode(List(cast(input[0, int, false] as bigint)),false), [id=#156]
   +- LocalTableScan [id#18]
```

BROADCAST JOIN

Spark SQL uses broadcast join (aka broadcast hash join) instead of hash join to optimize join queries when the size of one side data is below spark.sql.autoBroadcastJoinThreshold.

Broadcast join can be very efficient for joins between a large table (fact) with relatively small tables (dimensions) that could then be used to perform a star- schema join. It can avoid sending all data of the large table over the network.

You can use broadcast function or SQL's broadcast hints to mark a dataset to be broadcast when used in a join query.

broadcast join is also called a replicated join (in the distributed system community) or a map-side join (in the Hadoop community).

JoinSelection execution planning strategy uses spark.sql.autoBroadcastJoinThreshold property (default: 10M) to control the size of a dataset before broadcasting it to all worker nodes when performing a join.

COMMANDS
1. val threshold = spark.conf.get("spark.sql.autoBroadcastJoinThreshold").toInt 2.threshold / 1024 / 1024 *3.val q = spark.range(100).as("a").join(spark.range(100).as("b")).where($"a.id" === $"b.id")* *4.println(q.queryExecution.logical.numberedTree String)* *5. q.explain* *6. spark.conf.set("spark.sql.autoBroadcastJoinThresho ld", -1)* *7.spark.conf.get("spark.sql.autoBroadcastJoinThresho ld")*

SNAP:

```
scala> spark.conf.set("spark.sql.autoBroadcastJoinThreshold",10485760)

scala> val threshold =  spark.conf.get("spark.sql.autoBroadcastJoinThreshold").toInt
threshold: Int = 10485760

scala> threshold / 1024 / 1024
res2: Int = 10

scala> val q = spark.range(100).as("a").join(spark.range(100).as("b")).where($"a.id" === $"b.id")
q: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [id: bigint, id: bigint]

scala> println(q.queryExecution.logical.numberedTreeString)
00 'Filter ('a.id = 'b.id)
01 +- Join Inner
02    :- SubqueryAlias a
03    :  +- Range (0, 100, step=1, splits=Some(1))
04    +- SubqueryAlias b
05       +- Range (0, 100, step=1, splits=Some(1))

scala> println(q.queryExecution.sparkPlan.numberedTreeString)
00 BroadcastHashJoin [id#0L], [id#4L], Inner, BuildRight, false
01 :- Range (0, 100, step=1, splits=1)
02 +- Range (0, 100, step=1, splits=1)

scala> q.explain
== Physical Plan ==
*(2) BroadcastHashJoin [id#0L], [id#4L], Inner, BuildRight, false
:- *(2) Range (0, 100, step=1, splits=1)
+- BroadcastExchange HashedRelationBroadcastMode(List(input[0, bigint, false]),false), [id=#14]
   +- *(1) Range (0, 100, step=1, splits=1)


scala> spark.conf.set("spark.sql.autoBroadcastJoinThreshold", -1)

scala> spark.conf.get("spark.sql.autoBroadcastJoinThreshold")
res7: String = -1
```

COMMANDS

1. q.explain

2.val qBroadcast = spark.range(100).as("a").join(broadcast(spark.range(100)).as("b")).where($"a.id" === $"b.id")

3.qBroadcast.explain

4.val qBroadcastLeft = """
SELECT /+ BROADCAST (lf) */ **
FROM range(100) lf,
range(1000) rt WHERE lf.id = rt.id
"""

5.scala> sql(qBroadcastLeft).explain

```
scala> q.explain
== Physical Plan ==
*(2) BroadcastHashJoin [id#0L], [id#4L], Inner, BuildRight, false
:- *(2) Range (0, 100, step=1, splits=1)
+- BroadcastExchange HashedRelationBroadcastMode(List(input[0, bigint, false]),false), [id=#14]
   +- *(1) Range (0, 100, step=1, splits=1)


scala> val qBroadcast = spark.range(100).as("a").join(broadcast(spark.range(100)).as("b")).where($"a.id" === $"b.id")
qBroadcast: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [id: bigint, id: bigint]

scala> qBroadcast.explain
== Physical Plan ==
*(2) BroadcastHashJoin [id#10L], [id#14L], Inner, BuildRight, false
:- *(2) Range (0, 100, step=1, splits=1)
+- BroadcastExchange HashedRelationBroadcastMode(List(input[0, bigint, false]),false), [id=#32]
   +- *(1) Range (0, 100, step=1, splits=1)


scala> val qBroadcastLeft = """
     |   SELECT /*+ BROADCAST (lf) */ *
     |   FROM range(100) lf, range(1000) rt
     |   WHERE lf.id = rt.id
     | """
qBroadcastLeft: String =
"
  SELECT /*+ BROADCAST (lf) */ *
  FROM range(100) lf, range(1000) rt
  WHERE lf.id = rt.id
"

scala> sql(qBroadcastLeft).explain
== Physical Plan ==
*(2) BroadcastHashJoin [id#22L], [id#23L], Inner, BuildLeft, false
:- BroadcastExchange HashedRelationBroadcastMode(List(input[0, bigint, false]),false), [id=#50]
:  +- *(1) Range (0, 100, step=1, splits=1)
+- *(2) Range (0, 1000, step=1, splits=1)
```

COMMANDS

1. val qBroadcastRight = """
SELECT /+ MAPJOIN (rt) */ **
FROM range(100) lf,
range(1000) rt WHERE lf.id =
rt.id
"""
scala> sql(qBroadcastRight).explain

```
scala> sql(qBroadcastLeft).explain
== Physical Plan ==
*(2) BroadcastHashJoin [id#22L], [id#23L], Inner, BuildLeft, false
:- BroadcastExchange HashedRelationBroadcastMode(List(input[0, bigint, false]),false), [id=#50]
:  +- *(1) Range (0, 100, step=1, splits=1)
+- *(2) Range (0, 1000, step=1, splits=1)


scala> val qBroadcastRight = """
     |  SELECT /*+ MAPJOIN (rt) */ *
     |  FROM range(100) lf, range(1000) rt
     |  WHERE lf.id = rt.id
     | """
qBroadcastRight: String =
"
 SELECT /*+ MAPJOIN (rt) */ *
 FROM range(100) lf, range(1000) rt
 WHERE lf.id = rt.id
"

scala> sql(qBroadcastRight).explain
== Physical Plan ==
*(2) BroadcastHashJoin [id#26L], [id#27L], Inner, BuildRight, false
:- *(2) Range (0, 100, step=1, splits=1)
+- BroadcastExchange HashedRelationBroadcastMode(List(input[0, bigint, false]),false), [id=#68]
   +- *(1) Range (0, 1000, step=1, splits=1)
```

RESULT:

simple join using Mapper in Spark can be implemented.

Experiment 9: *Install and Run Hive then use Hive to create, alter, and drop databases, tables, views,functions, and indexes*

e) *OBJECTIVE OF THE EXERCISE/EXPERIMENT*

Install and Run Hive then use Hive to create, alter, and drop databases, tables, views,functions, and indexes

f) *Facilities/material required to do the exercise/experiment:*

Sl.No.	*Facilities/material required*	*Quantity*
1.	*Hadoop*	*As per strength*
2.	*Hive*	*As per strength*

INSTALLATION OF HIVE:

DOWNLOAD AND UNTAR HIVE:

1. You can download it by visiting the apache site

COMMANDS
wget https://downloads.apache.org/hive/hive-3.2.2/apache-hive-3.2.2-bin.tar.gz

On successful download, you get to see the following response:

```
apache-hive-3.2.2-bin.tar.gz
```

2. Installing Hive

The following steps are required for installing Hive on your system. Extracting and verifying Hive Archive

The following command is used to verify the download and extract the hive archive:

```
gokilanm@gokila-VirtualBox:~$ tar xzf apache-hive-3.1.2-bin.tar.gz
```

COMMANDS
tar zxvf apache-hive-3.2.2-bin.tar.gz

3. The following commands are used to copy the files from the extracted directory to the /usr/local/hive" directory.

4.Setting up environment for Hive

You can set up the Hive environment by appending the following lines to ~/.bashrc file:

export HIVE_HOME=/usr/local/hive

export

PATH=$PATH:$HIVE_HOME/bin

export CLASSPATH=$CLASSPATH:/home/gokilanm/hadoop-

3.2.2/lib/*:. export

CLASSPATH=$CLASSPATH:/usr/local/hive/lib/*:.

5.Configuring Hive

To configure Hive with Hadoop, you need to edit the hive-env.sh file, which is placed in the $HIVE_HOME/conf directory. The following commands redirect to Hive config folder and copy the template file:

COMMANDS
$ cd $HIVE_HOME/conf $ cp hive-env.sh.template hive-env.sh

Edit the hive-env.sh file by appending the following

line: export HADOOP_HOME=/usr/local/hadoop

Hive installation is completed successfully. Now you require an external database server to configure Metastore. We use Apache Derby database.

6. Downloading and Installing Apache Derby

Follow the steps given below to download and install Apache

Derby: Downloading Apache Derby

The following command is used to download Apache Derby. It takes some time to download.

Copying files to /usr/local/derby directory

Setting up environment for Derby

You can set up the Derby environment by appending the following lines to ~/.bashrc file:

export DERBY_HOME=/usr/local/derby

export

PATH=$PATH:$DERBY_HOME/bin

exportCLASSPATH=$CLASSPATH:$DERBY_HOME/lib/derby.jar:$D E RBY_HOME/lib/derbytools.jar

Create a directory to store Metastore

Create a directory named data in $DERBY_HOME directory to store Metastore data.

ALL HIVE INSTALLATION STEP:

```
gokilanm@gokila-VirtualBox:~$ tar xzf apache-hive-3.1.2-bin.tar.gz
gokilanm@gokila-VirtualBox:~$ sudo nano .bashrc
[sudo] password for gokilanm:
gokilanm@gokila-VirtualBox:~$ mv apache-hive-3.1.2-bin /usr/local/hive
mv: cannot move 'apache-hive-3.1.2-bin' to '/usr/local/hive': Permission denied
gokilanm@gokila-VirtualBox:~$ sudo mv apache-hive-3.1.2-bin /usr/local/hive
gokilanm@gokila-VirtualBox:~$ sudo nano .bashrc
gokilanm@gokila-VirtualBox:~$ source ~/.bashrc
gokilanm@gokila-VirtualBox:~$ cd $HIVE_HOME/conf
gokilanm@gokila-VirtualBox:/usr/local/hive/conf$ cp hive-env.sh.template hive-env.sh
gokilanm@gokila-VirtualBox:/usr/local/hive/conf$ gedit  hive-env.sh
gokilanm@gokila-VirtualBox:/usr/local/hive/conf$ cd
gokilanm@gokila-VirtualBox:~$ tar zxvf db-derby-10.4.2.0-bin.tar.gz
tar (child): db-derby-10.4.2.0-bin.tar.gz: Cannot open: No such file or directory
tar (child): Error is not recoverable: exiting now
tar: Child returned status 2
tar: Error is not recoverable: exiting now
gokilanm@gokila-VirtualBox:~$ tar xzf db-derby-10.13.1.1-bin.tar.gz
gokilanm@gokila-VirtualBox:~$ sudo  mv db-derby-10.13.1.1-bin /usr/local/derby
gokilanm@gokila-VirtualBox:~$ gedit  ~/.bashrc
gokilanm@gokila-VirtualBox:~$ sudo nano .bashrc
gokilanm@gokila-VirtualBox:~$ source ~/.bashrc
gokilanm@gokila-VirtualBox:~$ mkdir $DERBY_HOME/data
gokilanm@gokila-VirtualBox:~$ cd $HIVE_HOME/conf
gokilanm@gokila-VirtualBox:/usr/local/hive/conf$ cp hive-default.xml.template hive-site.xml
gokilanm@gokila-VirtualBox:/usr/local/hive/conf$ sudo nano hive-site.xml
gokilanm@gokila-VirtualBox:/usr/local/hive/conf$ gedit  hive-site.xml

^C
gokilanm@gokila-VirtualBox:/usr/local/hive/conf$ sudo nano  jpox.properties
gokilanm@gokila-VirtualBox:/usr/local/hive/conf$ chmod g+w
chmod: missing operand after ‘g+w’
Try 'chmod --help' for more information.
gokilanm@gokila-VirtualBox:/usr/local/hive/conf$ cd
gokilanm@gokila-VirtualBox:~$ chmod g+w
chmod: missing operand after ‘g+w’
Try 'chmod --help' for more information.
gokilanm@gokila-VirtualBox:~$ $HADOOP_HOME/bin/hadoop fs -mkdir /tmp
mkdir: `/tmp': File exists
gokilanm@gokila-VirtualBox:~$ $HADOOP_HOME/bin/hadoop fs -mkdir /user/hive/warehouse
mkdir: `/user/hive/warehouse': File exists
gokilanm@gokila-VirtualBox:~$ $HADOOP_HOME/bin/hadoop fs -mkdir -p /user/hive/warehouse
gokilanm@gokila-VirtualBox:~$ $HADOOP_HOME/bin/hadoop fs -chmod g+w /tmp
gokilanm@gokila-VirtualBox:~$ $HADOOP_HOME/bin/hadoop fs -chmod g+w /user/hive/warehouse
gokilanm@gokila-VirtualBox:~$ rm $HIVE_HOME/lib/guava-19.0.jar
```

Step 7: Configuring Metastore of Hive

Configuring Metastore means specifying to Hive where the database is stored. You can do this by editing the hive-site.xml file, which is in the
$HIVE_HOME/conf directory. First of all, copy the template file

using the following command:

```
gokilanm@gokila-VirtualBox:/usr/local/hive/bin$ cd $HIVE_HOME/bin
gokilanm@gokila-VirtualBox:/usr/local/hive/bin$ cd $HIVE_HOME/conf
gokilanm@gokila-VirtualBox:/usr/local/hive/conf$ gedit hive-site.xml
```

```
1 <configuration>
2 <property>
3 <name>javax.jdo.option.ConnectionURL</name>
4 <value>jdbc:derby:/usr/local/hive/metastore_db_bkp;databaseName=metastore_db;create=true </value>
5 <description>JDBC connect string for a JDBC metastore </description>
6 </property>
7 </configuration>
```

Create a file named jpox.properties and add the following lines into it:

javax.jdo.PersistenceManagerFactoryClass =org.jpox.PersistenceManagerFactor

yImpl org.jpox.autoCreateSchema =

false org.jpox.validateTables = false

org.jpox.validateColumns = false

org.jpox.validateConstraints = false

org.jpox.storeManagerType = rdbms

org.jpox.autoCreateSchema = true

org.jpox.autoStartMechanismMode =

checked org.jpox.transactionIsolation =

read_committed

javax.jdo.option.DetachAllOnCommit =

true

javax.jdo.option.NontransactionalRead =

true

javax.jdo.option.ConnectionDriverName = org.apache.derby.jdbc.ClientDriver

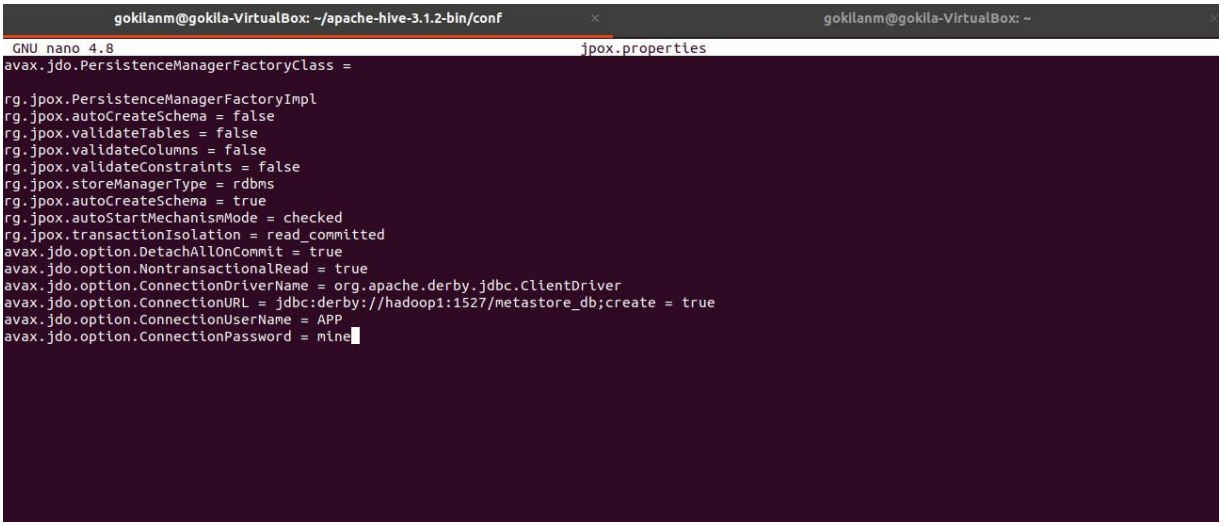
javax.jdo.option.ConnectionURL = jdbc:derby://hadoop1:1527/metastore_db;creat e = true

javax.jdo.option.ConnectionUserName =

APP

javax.jdo.option.ConnectionPassword =

mine



Use the schematool command once again to initiate the Derby database:

Launch Hive Client Shell on Ubuntu

Start the Hive command-line interface using the following commands:

cd

$HIVE_HOME/bin

hive

You are now able to issue SQL-like commands and directly interact with HDFS.

HIVE-OPERATIONS

COMMANDS
cd *$HIVE_HOME/bin* *hive*

```
gokilanm@gokila-VirtualBox:/usr/local/hive/bin$ hive
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/local/hive/lib/log4j-slf4j-impl-2.10.0.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/home/gokilanm/hadoop-3.2.2/share/hadoop/common/lib/slf4j-log4j12-1.7.25.jar!/org/slf4j/impl/StaticLoggerBinder.clas
s]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4jLoggerFactory]
Hive Session ID = b8bd988a-f4ec-4805-a88c-1c2a1e702d15

Logging initialized using configuration in jar:file:/usr/local/hive/lib/hive-common-3.1.2.jar!/hive-log4j2.properties Async: true
Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different execution engine (i.e. spark, tez) or u
sing Hive 1.X releases.
Hive Session ID = 0ea7ba49-4ca9-440d-99cf-4f78d3a4ede2
hive> show databases;
OK
default
Time taken: 2.095 seconds, Fetched: 1 row(s)
hive>
```

1.CREATE A DATABASE

A database in Hive is a namespace or a collection of tables.

COMMANDS
Create database bigdata;

TERMINAL:

```
hive> create database bigdata;
OK
Time taken: 0.343 seconds
```

2. *SHOW A DATABASE:*

At any time, you can see the databases that already exist as follows:

COMMANDS
Show databases;

```
hive> show databases;
OK
bigdatalab
bigdatalabdemo
default
lab
Time taken: 1.202 seconds, Fetched: 4 row(s)
```

3. *LIST A DATABASE STARTS WITH A SPECIFIC LETTER:*

If you have a lot of databases, you can restrict the ones listed using a regular expression, a concept we'll explain in LIKE and RLIKE, if it is new to you. The following example lists only those databases that start with the letter b and end with any other characters

COMMANDS
SHOW DATABASES LIKE 'b.';*

```
hive> SHOW DATABASES LIKE 'b.*';
OK
bigdata
bigdatalab
bigdatalabdemo
Time taken: 0.087 seconds, Fetched: 3 row(s)
```

4. *IF DATABASE ALREADY EXISTS:*

While normally you might like to be warned if a database of the same name already exists, the IF NOT EXISTS clause is useful for scripts that should create a database on-the-fly, if necessary, before proceeding.

COMMANDS
CREATE DATABASE IF NOT EXISTS lab;

```
hive> CREATE DATABASE IF NOT EXISTS lab;
OK
Time taken: 0.06 seconds
```

5. *Add a creator name with date of database*

COMMANDS
CREATE DATABASE bigdataiseWITH DBPROPERTIES ('creator' = 'gokila', 'date' = '2021-10-02');

```
hive> CREATE DATABASE bigdataise WITH DBPROPERTIES('creator'='gokila','date'='2021-10-12');
OK
Time taken: 0.089 seconds
```

6. DESCRIBE A DATABASE:

This command is used to check any associated metadata for the databases.

COMMANDS
Describe Database bigdataise;

```
hive> DESCRIBE DATABASE  bigdataise;
OK
bigdataise              hdfs://127.0.0.1:9000/user/hive/warehouse/bigdataise.db gokilanm        USER
```

Lastly, you can associate key-value properties with the database, although their only function currently is to provide a way of adding information to the output of DESCRIBE DATABASE EXTENDED <database>:

COMMANDS
DESCRIBE DATABASE EXTENDED bigdataise

```
hive> DESCRIBE DATABASE EXTENDED bigdataise;
OK
bigdataise              hdfs://127.0.0.1:9000/user/hive/warehouse/bigdataise.db gokilanm        USER    {date=2021-10-12, creator=gokila}
Time taken: 0.094 seconds, Fetched: 1 row(s)
```

7. USE COMMAND:

The USE command sets a database as your working database, analogous to changing working directories in a filesystem:

COMMANDS
USE bigdataise;

```
hive> use bigdataise;
OK
Time taken: 0.134 seconds
```

8. SHOW TABLES:

It will list the tables in this database.

COMMANDS
Show tables;

```
hive> show tables;
OK
employees
employees2
Time taken: 0.313 seconds, Fetched: 2 row(s)
hive>
```

9. *DEFAULT DATABASE:*

Variables and Properties for setting a property to print the current database as part of the prompt (Hive v0.8.0 and later):

COMMANDS
1.set hive.cli.print.current.db=true; *2.hive (bigdata)> USE default;* *3.hive (default)> set hive.cli.print.current.db=false;*

```
hive> USE BIGDATA;
OK
Time taken: 0.155 seconds
hive> set hive.cli.print.current.db=true;
hive (BIGDATA)> USE default;
OK
Time taken: 0.14 seconds
hive (default)> set hive.cli.print.current.db=false;
hive>
```

10. *DROP A DATABASE:*

COMMANDS
DROP DATABASE IF EXISTSbigdataise;

```
hive> DROP DATABASE IF EXISTS bigdatalab;
```

The IF EXISTS is optional and suppresses warnings if financials doesn't exist. By default, Hive won't permit you to drop a database if it contains tables. You can either drop the tables first or append the CASCADE keyword to the command, which will cause the Hive to drop the tables in the database.

When a database is dropped, its directory is also deleted.

11. *ALTER DATABASE:*

Alter Database You can set key-value pairs in the DBPROPERTIES associated with a database using the ALTER DATABASE command. No other metadata about the database can be changed, including its name and directory location:

COMMANDS
ALTER DATABASE bigdataise SET DBPROPERTIES ('edited-by' = 'gokilanm');

```
hive> ALTER DATABASE bigdataise SET DBPROPERTIES('edited-by'='gokilanm')
    > ;
OK
Time taken: 0.165 seconds
```

There is no way to delete or "unset" a DBPROPERTY.

12. *Creating Tables*

The CREATE TABLE statement follows SQL conventions, but Hive's version offers significant extensions to support a wide range of flexibility where the data files for tables are stored, the formats used, etc. We discussed many of these options in Text File Encoding of Data Values.we describe the other options available for the CREATE TABLE statement, adapting the employees table declaration we used previously in Collection Data Types:

COMMANDS

```
CREATE TABLE IF NOT EXISTS
bigdataise.employees ( name STRING COMMENT 'Employee name',
salary FLOAT COMMENT 'Employee salary',
subordinates ARRAY<STRING> COMMENT 'Names of
subordinates', deductions MAP<STRING, FLOAT>
COMMENT 'Keys are deductions names, values are  percentages',
address STRUCT<street:STRING, city:STRING, state:STRING,
zip:INT> COMMENT 'Home address')
COMMENT 'Description of the table'
LOCATION '/user/hive/warehouse/bigdataise.db/employees'
TBLPROPERTIES ('creator'='me', 'created_at'='2012-01-02
10:00:00');
```

```
hive> CREATE TABLE IF NOT EXISTS bigdataise.employees(
    >      name STRING COMMENT 'Employee name',
    >      salary FLOAT COMMENT 'Employee salary',
    >      subordinates  ARRAY<STRING> COMMENT 'Names of subordinates',
    >      deductions MAP<STRING,FLOAT> COMMENT 'Keys are deductions names, values are percentages',
    >      address STRUCT<street:STRING, city:STRING, state:STRING, zip:INT> COMMENT 'Home address')
    >      COMMENT 'Description of the table'
    >      LOCATION '/user/hive/warehouse/bigdatalab.db/employees'
    >      TBLPROPERTIES('creator'='gokila','created_at'='2012-01-02 10:00:00');
OK
Time taken: 0.959 seconds
hive>
```

First, note that you can prefix a database name, mydb in this case, if you're not currently working in the target database.

If you add the option IF NOT EXISTS, Hive will silently ignore the statement if the table already exists. This is useful in scripts that should create a table the first time they run.

However, the clause has a gotcha you should know. If the schema specified differs from the schema in the table that already exists, Hive won't warn you.

If your intention is for this table to have the new schema, you'll have to drop the old table, losing your data, and then re-create it.

Consider if you should use one or more ALTER TABLE statements to change the existing table schema instead. See Alter Table for details.

You can add a comment to any column, after the type. Like databases, you can attach a comment to the table itself and you can define one or more table properties. In most cases, the primary benefit of TBLPROPERTIES is to add additional documentation in a key-value format. However, when we examine Hive's integration with databases such as DynamoDB (see DynamoDB), we'll see that the TBLPROPERTIES can be used to express essential metadata about the database connection.

Hive automatically adds two table properties: last_modified_by holds the username of the last user to modify the table, and last_modified_timeholds the epoch time in seconds of that

modification. Finally, you can optionally specify a location for the table data (as opposed to metadata, which the metastore will always hold). In this example, we are showing the default location that Hive would use, /user/hive/warehouse/bigdataise.db/employees, where /user/hive/warehouse is the default "warehouse" location bigdataise.db is the database directory, and employees is the table directory.

By default, Hive always creates the table's directory under the directory for the enclosing database. The exception is the default database. It doesn't have a directory under /user/hive/warehouse, so a table in the default database will have its directory created directly in /user/hive/warehouse (unless explicitly overridden).

13. *Copy the schema of an existing tables:*

COMMANDS
CREATE TABLE IF NOT EXISTS bigdataise.employees2 LIKE bigdataise.employees;

```
hive> CREATE TABLE IF NOT EXISTS bigdataise.employees2 LIKE bigdataise.employees;
OK
Time taken: 2.336 seconds
hive>
```

This version also accepts the optional LOCATION clause, but note that no other properties, including the schema, can be defined; they are determined from the original table.

14.*SHOW TABLES:*

The SHOW TABLES command lists the tables. With no additional arguments, it shows the tables in the current working database.
Let's assume we have already created a few other tables, table1 and table2, and we did so in the bigdataise database:

COMMANDS
Show tables

```
hive> show tables;
OK
employees
employees2
Time taken: 0.313 seconds, Fetched: 2 row(s)
```

If we aren't in the same database, we can still list the tables in that database:

COMMANDS
USE default; *SHOW TABLES IN bigdataise;*

```
hive> USE default;
OK
Time taken: 0.051 seconds
hive> SHOW TABLES IN bigdataise;
OK
employees
employees2
Time taken: 0.08 seconds, Fetched: 2 row(s)
```

16.DESCRIBE:

We can also use the DESCRIBE EXTENDED bigdataise.employees command to show details about the table. (We can drop the bigdataise. prefix if we're currently using the bigdataise database.) We have reformatted the output for easier reading and we have suppressed many details to focus on the items that interest us now:

COMMANDS
DESCRIBE EXTENDED bigdataise.employees;

```
hive> DESCRIBE EXTENDED bigdataise.employees;
OK
name                    string                  Employee name
salary                  float                   Employee salary
subordinates            array<string>           Names of subordinates
deductions              map<string,float>       Keys are deductions names, values are percentages
address                 struct<street:string,city:string,state:string,zip:int>  Home address

Detailed Table Information      Table(tableName:employees, dbName:bigdataise, owner:gokilanm, createTime:1634622466, lastAccessTime:0, retention:0, sd
:StorageDescriptor(cols:[FieldSchema(name:name, type:string, comment:Employee name), FieldSchema(name:salary, type:float, comment:Employee salary), Fi
eldSchema(name:subordinates, type:array<string>, comment:Names of subordinates), FieldSchema(name:deductions, type:map<string,float>, comment:Keys are
 deductions names, values are percentages), FieldSchema(name:address, type:struct<street:string,city:string,state:string,zip:int>, comment:Home addres
s)], location:hdfs://127.0.0.1:9000/user/hive/warehouse/bigdatalab.db/employees, inputFormat:org.apache.hadoop.mapred.TextInputFormat, outputFormat:or
g.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat, compressed:false, numBuckets:-1, serdeInfo:SerDeInfo(name:null, serializationLib:org.apache.
hadoop.hive.serde2.lazy.LazySimpleSerDe, parameters:{serialization.format=1}), bucketCols:[], sortCols:[], parameters:{}, skewedInfo:SkewedInfo(skewed
ColNames:[], skewedColValues:[], skewedColValueLocationMaps:{}), storedAsSubDirectories:false), partitionKeys:[], parameters:{creator=gokila, totalSiz
e=0, numRows=0, rawDataSize=0, COLUMN_STATS_ACCURATE={\"BASIC_STATS\":\"true\",\"COLUMN_STATS\":{\"address\":\"true\",\"deductions\":\"true\",\"name\"
:\"true\",\"salary\":\"true\",\"subordinates\":\"true\"}}, numFiles=0, transient_lastDdlTime=1634622466, bucketing_version=2, created_at=2012-01-02 10
:00:00, comment=Description of the table}, viewOriginalText:null, viewExpandedText:null, tableType:MANAGED_TABLE, rewriteEnabled:false, catName:hive,
ownerType:USER)
Time taken: 0.539 seconds, Fetched: 7 row(s)
```

Replacing EXTENDED with FORMATTED provides more readable but also more verbose output.

The first section shows the output of DESCRIBE without EXTENDED or FORMATTED (i.e., the schema including the comments for each column). If you only want to see the schema for a particular column, append the column to the table name. Here, EXTENDED adds no additional output:

COMMANDS
DESCRIBE bigdataise.employees salary;

```
hive> DESCRIBE bigdataise.employees salary;
OK
salary                  float                   from deserializer
COLUMN_STATS_ACCURATE   {\"BASIC_STATS\":\"true\",\"COLUMN_STATS\":{\"address\":\"true\",\"deductions\":\"true\",\"name\":\"true\",\"salary\":\"true\"
,\"subordinates\":\"true\"}}
Time taken: 0.195 seconds, Fetched: 2 row(s)
hive>
```

Returning to the extended output, note the line in the description that starts with location:. It shows the full URI path in HDFS to the directory where Hive will keep all the data for this table, as we discussed above.

17.MANAGED TABLES:

The tables we have created so far are called managed tables or sometimes called internal tables, because Hive controls the lifecycle of their data (more or less). '=

As we've seen, Hive stores the data for these tables in a subdirectory under the directory defined by hive.metastore.warehouse.dir (e.g.,/user/hive/ warehouse), by default. When we drop a managed table (see Dropping Tables), Hive deletes the data in the table. However, managed tables are less convenient for sharing with other tools.

For example, suppose we have data that is created and used primarily by Pig or other tools, but we want to run some queries against it, but not give Hive ownership of the data. We can define an external table that points to that data, but doesn't take ownership of it.

18. EXTERNAL TABLES:

Suppose we are analyzing data from the stock markets. Periodically, we ingest the data for NASDAQ and the NYSE from a source like Infochimps (http://infochimps.com/datasets) and we want to study this data with many tools. The schema we'll use next matches the schemas of both these data sources. Let's assume the data files are in the distributed filesystem directory /data/ stocks. The following table declaration creates an external table that can read all the data files for this comma-delimited data in /data/ stocks:

COMMANDS
CREATE EXTERNAL TABLE IF NOT EXISTS stocks (*> 'exchangee' STRING,* *> symbol STRING,* *> ymd STRING,* *> price_open FLOAT,* *> price_high FLOAT,* *> price_low FLOAT,* *> price_close FLOAT,* *> volume INT,*

```
> price_adj_close FLOAT)
> ROW FORMAT DELIMITED FIELDS TERMINATED BY','
> LOCATION '/data/stocks';
```

```
hive> CREATE EXTERNAL TABLE IF NOT EXISTS stock (
    >       exchangee STRING,
    >       symbol STRING,
    >       ymd STRING,
    >       price_open FLOAT,
    >       price_high FLOAT,
    >       price_low FLOAT,
    >       price_close FLOAT,
    >       volume INT,
    >       price_adj_close FLOAT)
    >       CLUSTERED BY (exchangee, symbol)
    >       SORTED BY (ymd ASC)
    >       INTO 96 BUCKETS
    >       ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
    >       LOCATION '/data/stocks';
OK
Time taken: 0.8 seconds
```

The EXTERNAL keyword tells Hive this table is external and the LOCATION ...clause is required to tell Hive where it's located.

Because it's external, Hive does not assume it owns the data. Therefore, dropping the table does not delete the data, although the metadata for the table will be deleted.

There are a few other small differences between managed and external tables, where some HiveQL constructs are not permitted for external tables. We'll discuss those when we come to them. However, it's important to note that the differences between managed and external tables are smaller than they appear at first. Even for managed tables, you know where they are located, so you can use other tools, hadoop dfs commands, etc., to modify and even delete the files in the directories for managed tables. Hive may technically own these directories and files, but it doesn't have full control over them!
Recall, in Schema on Read, we said that Hive really has no control over the integrity of the files used for storage and whether or not their contents are consistent with the table schema. Even managed tables don't give us this control.

Still, a general principle of good software design is to express intent. If the data is shared between tools, then creating an external table makes this ownership explicit.

You can tell whether or not a table is managed or external using the output of DESCRIBE EXTENDED tablename. Near the end of the Detailed Table Information output, you will see the following for managed tables: ... tableType:MANAGED_TABLE) For external tables, you will see the following: ... tableType:EXTERNAL_TABLE)

COMMANDS
DESCRIBE EXTENDED stocks;

```
hive> DESCRIBE EXTENDED stocks;
OK
exchangee               string
symbol                  string
ymd                     string
price_open              float
price_high              float
price_low               float
price_close             float
volume                  int
price_adj_close         float

Detailed Table Information      Table(tableName:stocks, dbName:default, owner:gokilanm, createTime:1634623632, lastAccessTime:0, retention:0, sd:Stora
geDescriptor(cols:[FieldSchema(name:exchangee, type:string, comment:null), FieldSchema(name:symbol, type:string, comment:null), FieldSchema(name:ymd,
type:string, comment:null), FieldSchema(name:price_open, type:float, comment:null), FieldSchema(name:price_high, type:float, comment:null), FieldSchem
a(name:price_low, type:float, comment:null), FieldSchema(name:price_close, type:float, comment:null), FieldSchema(name:volume, type:int, comment:null)
, FieldSchema(name:price_adj_close, type:float, comment:null)], location:hdfs://127.0.0.1:9000/data/stocks, inputFormat:org.apache.hadoop.mapred.TextI
nputFormat, outputFormat:org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat, compressed:false, numBuckets:-1, serdeInfo:SerDeInfo(name:null, s
erializationLib:org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe, parameters:{serialization.format=,, field.delim=,}), bucketCols:[], sortCols:[], p
arameters:{}, skewedInfo:SkewedInfo(skewedColNames:[], skewedColValues:[], skewedColValueLocationMaps:{}), storedAsSubDirectories:false), partitionKey
s:[], parameters:{transient_lastDdlTime=1634623632, bucketing_version=2, EXTERNAL=TRUE}, viewOriginalText:null, viewExpandedText:null, tableType:EXTER
NAL_TABLE, rewriteEnabled:false, catName:hive, ownerType:USER)
Time taken: 0.208 seconds, Fetched: 11 row(s)
```

As for managed tables, you can also copy the schema (but not the data) of an existing table:

COMMANDS
CREATE EXTERNAL TABLE IF NOT EXISTS bigdataise.employees3 *> LIKE bigdataise.employees* *> LOCATION '/data/';*

```
hive> CREATE EXTERNAL TABLE IF NOT EXISTS bigdataise.employees3
    >       LIKE bigdataise.employees
    >       LOCATION '/data/';
OK
Time taken: 0.366 seconds
```

19.*Partitioned, Managed Tables*

The general notion of partitioning data is an old one. It can take many forms, but often it's used for distributing load horizontally, moving data physically closer to its most frequent users, and other purposes.

Hive has the notion of partitioned tables. We'll see that they have important performance benefits, and they can help organize data in a logical fashion, such as hierarchically.

We'll discuss partitioned managed tables first. Let's return to our employees table and imagine that we work for a very large multinational corporation. Our HR people often run queries with WHERE clauses that restrict the results to a particular country or to a particular first-level subdivision (e.g., state in the United States or province in Canada). (First-level subdivision is an actual term, used here, for example: http://www.commondatahub.com/state_source.jsp.) We'll just use the word state for simplicity. We have redundant state information in the address field. It is distinct from the state partition. We could remove the state element from address. There is no ambiguity in queries, since

we have to use address.state to project the value inside the address. So, let's partition the data first by country and then by state:

COMMANDS
CREATE TABLE employeedetails (*> name STRING,* *> salary FLOAT,* *> subordinates ARRAY<STRING>,* *> deductions MAP<STRING, FLOAT>,* *> address STRUCT<street:STRING, city:STRING, state:STRING, zip:INT>)* *> PARTITIONED BY (country STRING, state STRING);*

```
hive>       CREATE TABLE employeedetails (
    >       name STRING,
    >       salary FLOAT,
    >       subordinates ARRAY<STRING>,
    >       deductions MAP<STRING, FLOAT>,
    >       address STRUCT<street:STRING, city:STRING, state:STRING, zip:INT>)
    >       PARTITIONED BY (country STRING, state STRING);
OK
Time taken: 0.368 seconds
hive>
```

20. *External Partitioned Tables*

You can use partitioning with external tables. In fact, you may find that this is your most common scenario for managing large production data sets. The combination gives you a way to "share" data with other tools, while still optimizing query performance.

You also have more flexibility in the directory structure used, as you define it yourself. We'll see a particularly useful example in a moment.

Let's consider a new example that fits this scenario well: logfile analysis. Most organizations use a standard format for log messages, recording a timestamp, severity (e.g., ERROR, WARNING, INFO), perhaps a server name and process ID, and then an arbitrary text message. Suppose our Extract, Transform, and Load (ETL) process ingests and aggregates logfiles in our environment, converting each log message to a tab-delimited record and also decomposing the timestamp into separate year, month, and day fields, and a combined hms field for the remaining hour, minute, and second parts of the timestamp, for reasons that will become clear in a moment. You could do this parsing of log messages using the string parsing functions built into Hive or Pig, for example. Alternatively, we could use smaller integer types for some of the timestamp- related fields to conserve space. Here, we are ignoring subsequent resolution.

Here's how we might define the corresponding Hive table:

COMMANDS

```
CREATE EXTERNAL TABLE IF NOT EXISTS log_messages (
  > hms INT,
  > severity STRING,
  > server STRING,
  > process_id INT,
  > message STRING)
  > PARTITIONED BY (year INT, month INT, day INT)
  > ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t';
```

We're assuming that a day's worth of log data is about the correct size for a useful partition and finer grain queries over a day's data will be fast enough. Recall that when we created the nonpartitioned external stocks table, a LOCATION ... clause was required. It isn't used for external partitioned tables. Instead, an ALTER TABLE statement is used to add each partition separately.It must specify a value for each partition key, the year, month, and day, in this case (see Alter Table for more details on this feature).

Here is an example, where we add a partition for January 2nd, 2012:

COMMANDS

```
ALTER TABLE log_messages ADD PARTITION(year = 2012, month = 1, day=2)
  > LOCATION 'hdfs://localhost:9000/data/log_messages/2012/01/02';
```

The directory convention we use is completely up to us. Here, we follow a hierarchical directory structure, because it's a logical way to organize our data, but there is no requirement to do so. We could follow Hive's directory naming convention (e.g., .../exchange=NASDAQ/symbol=AAPL), but there is no requirement to do so.

21.*SHOW PARTITIONS:*

As for managed partitioned tables, you can see an external table's partitions with SHOW PARTITIONS:

COMMANDS
Show partitions;

```
hive> SHOW PARTITIONS log_messages;
OK
Time taken: 0.39 seconds
```

Similarly, the DESCRIBE EXTENDED log_messages shows the partition keys both as part of the schema and in the list of partitionKeys:

COMMANDS
DESCRIBE EXTENDED log_messages;

```
hive> DESCRIBE EXTENDED log_messages;
OK
hms                     int
severity                string
server                  string
process_id              int
message                 string
year                    int
month                   int
day                     int

# Partition Information
# col_name              data_type               comment
year                    int
month                   int
day                     int

Detailed Table Information      Table(tableName:log_messages, dbName:default, owner:gokilanm, createTime:1634624929, lastAccessTime:0, retention:0, sd
:StorageDescriptor(cols:[FieldSchema(name:hms, type:int, comment:null), FieldSchema(name:severity, type:string, comment:null), FieldSchema(name:server
, type:string, comment:null), FieldSchema(name:process_id, type:int, comment:null), FieldSchema(name:message, type:string, comment:null), FieldSchema(
name:year, type:int, comment:null), FieldSchema(name:month, type:int, comment:null), FieldSchema(name:day, type:int, comment:null)], location:hdfs://1
27.0.0.1:9000/user/hive/warehouse/log_messages, inputFormat:org.apache.hadoop.mapred.TextInputFormat, outputFormat:org.apache.hadoop.hive.ql.io.HiveIg
noreKeyTextOutputFormat, compressed:false, numBuckets:-1, serdeInfo:SerDeInfo(name:null, serializationLib:org.apache.hadoop.hive.serde2.lazy.LazySimpl
eSerDe, parameters:{serialization.format=\t, field.delim=\t}), bucketCols:[], sortCols:[], parameters:{}, skewedInfo:SkewedInfo(skewedColNames:[], ske
wedColValues:[], skewedColValueLocationMaps:{}), storedAsSubDirectories:false), partitionKeys:[FieldSchema(name:year, type:int, comment:null), FieldSc
hema(name:month, type:int, comment:null), FieldSchema(name:day, type:int, comment:null)], parameters:{totalSize=0, EXTERNAL=TRUE, numRows=0, rawDataSi
ze=0, COLUMN_STATS_ACCURATE={\"BASIC_STATS\":\"true\"}, numFiles=0, numPartitions=0, transient_lastDdlTime=1634624929, bucketing_version=2}, viewOrigi
nalText:null, viewExpandedText:null, tableType:EXTERNAL_TABLE, rewriteEnabled:false, catName:hive, ownerType:USER)
Time taken: 0.287 seconds, Fetched: 16 row(s)
```

This output is missing a useful bit of information, the actual location of the partition data. There is a location field, but it only shows Hive's default directory that would be used if the table were a managed table. However, we can get a partition's location as follows:

COMMANDS
DESCRIBE EXTENDED log_messages PARTITION (year=2012,month=1,day=2);

We frequently use external partitioned tables because of the many benefits they provide, such as logical data management, performant queries, etc. ALTER TABLE ... ADD PARTITION is not limited to external tables. You can use it with managed tables, too, when you have (or will have) data for partitions in directories created outside of the LOAD and INSERT options we discussed above. You'll need to remember that not all of the table's data will be under the usual Hive "warehouse" directory, and this data won't be deleted when you drop the managed table! Hence, from a "sanity" perspective, it's questionable whether you should dare to use this feature with managed tables.

21.*Customizing Table Storage Formats*

In Text File Encoding of Data Values, we discussed that Hive defaults to a text file format, which is indicated by the optional clause STORED AS TEXTFILE, and you can overload the default values for the various delimiters when creating the table. Here we repeat the definition of the employees table we used in that discussion:

COMMANDS
CREATE TABLE employeesdetails (*> name STRING,* *> salary FLOAT,* *> subordinates ARRAY<STRING>,* *> deductions MAP<STRING, FLOAT>,* *> address STRUCT<street:STRING, city:STRING, state:STRING, zip:INT>)* *> ROW FORMAT DELIMITED* *> FIELDS TERMINATED BY '\001'* *> COLLECTION ITEMS TERMINATED BY '\002'* *> MAP KEYS TERMINATED BY '\003'* *> LINES TERMINATED BY '\n'*

> STORED AS TEXTFILE;

```
hive> CREATE TABLE employeesdetails (
    >       name STRING,
    >       salary FLOAT,
    >       subordinates ARRAY<STRING>,
    >       deductions MAP<STRING, FLOAT>,
    >       address STRUCT<street:STRING, city:STRING, state:STRING, zip:INT>)
    >       ROW FORMAT DELIMITED
    >       FIELDS TERMINATED BY '\001'
    >       COLLECTION ITEMS TERMINATED BY '\002'
    >       MAP KEYS TERMINATED BY '\003'
    >       LINES TERMINATED BY '\n'
    >       STORED AS TEXTFILE;
OK
Time taken: 0.287 seconds
```

TEXTFILE implies that all fields are encoded using alphanumeric characters, including those from international character sets, although we observed that Hive uses non-printing characters as "terminators" (delimiters), by default.
When TEXTFILE is used, each line is considered a separate record.

You can replace TEXTFILE with one of the other built-in file formats supported by Hive, including SEQUENCEFILE and RCFILE, both of which optimize disk space usage and I/O bandwidth performance using binary encoding and optional compression. Hive draws a distinction between how records are encoded into files and how columns are encoded into records. You customize these behaviors separately.

The record encoding is handled by an input format object (e.g., the Java code behind TEXTFILE.) Hive uses a Java class (compiled module) namedorg.apache.hadoop.mapred.TextInputForm at. If you are unfamiliar with Java, the dotted name syntax indicates a hierarchical namespace tree ofpackages that actually corresponds to the directory structure for the Java code. The last name, TextInputFormat, is a class in the lowest-level package mapred.

The record parsing is handled by a serializer/ deserializer or SerDe for short. For TEXTFILE and the encoding we described in Chapter 3 and repeated in the example above, the SerDe Hive uses is another Java class called org.apache.hadoop.hive.serde2.lazy.LazySimple SerDe.

For completeness, there is also an output format that Hive uses for writing the output of queries to files and to the console. For TEXTFILE, the Java class named org.apache.hadoop.hive.ql.io.HiveIgnoreK eyTextOutputFormat is used for output.

22. CREATE TABLE CLAUSES:

Finally, there are a few additional CREATE TABLE clauses that describe more details about how the data is supposed to be stored. Let's extend our previous stocks table example from External Tables:

COMMANDS

```
CREATE EXTERNAL TABLE IF NOT EXISTS stock (
  > exchangee STRING,
  > symbol STRING,
  > ymd STRING,
  > price_open FLOAT,
  > price_high FLOAT,
  > price_low FLOAT,
  > price_close FLOAT,
  > volume INT,
  > price_adj_close FLOAT)
  > CLUSTERED BY (exchangee, symbol)
  > SORTED BY (ymd ASC)
  > INTO 96 BUCKETS
  > ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
  > LOCATION '/data/stocks';
```

```
hive> CREATE EXTERNAL TABLE IF NOT EXISTS stock (
    >     exchangee STRING,
    >     symbol STRING,
    >     ymd STRING,
    >     price_open FLOAT,
    >     price_high FLOAT,
    >     price_low FLOAT,
    >     price_close FLOAT,
    >     volume INT,
    >     price_adj_close FLOAT)
    >     CLUSTERED BY (exchangee, symbol)
    >     SORTED BY (ymd ASC)
    >     INTO 96 BUCKETS
    >     ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
    >     LOCATION '/data/stocks';
OK
Time taken: 0.8 seconds
hive>
```

The CLUSTERED BY ... INTO ... BUCKETS clause, with an optional SORTED BY ... clause is used to optimize certain kinds of queries, which we discuss in detail in Bucketing Table Data Storage.

23. *Alter Table*

Most table properties can be altered with ALTER TABLE statements, which change metadata about the table but not the data itself. These statements can be used to fix mistakes in schema, move partition locations (as we saw in External Partitioned Tables), and do other operations.

24. *Renaming a Table*

Use this statement to rename the table log_messages to logmsgs:

COMMANDS

```
ALTER TABLE log_messages RENAME TO logmsgs;
```

```
Time taken: 0.8 seconds
hive> ALTER TABLE log_messages RENAME TO logmsgs;
OK
Time taken: 0.392 seconds
```

25. *Adding, Modifying, and Dropping a Table Partition*

As we saw previously, ALTER TABLE table ADD PARTITION ... is used to add a new partition to a table (usually an external table). Here we repeat the same command shown previously with the additional options available:

COMMANDS
ALTER TABLE logmsgs ADD IF NOT EXISTS *> PARTITION (year = 2011, month = 1, day = 1) LOCATION '/logs/2021/10/03'* *> PARTITION (year = 2011, month = 1, day = 2) LOCATION '/logs/2021/10/04';*

```
hive> ALTER TABLE logmsgs ADD IF NOT EXISTS
    >       PARTITION (year = 2011, month = 1, day = 1) LOCATION '/logs/2021/10/03'
    >       PARTITION (year = 2011, month = 1, day = 2) LOCATION '/logs/2021/10/04';
OK
Time taken: 0.269 seconds
```

Multiple partitions can be added in the same query when using Hive v0.8.0 and later. As always, IF NOT EXISTS is optional and has the usual meaning.

Similarly, you can change a partition location, effectively moving it:

COMMANDS
ALTER TABLE logmsgs PARTITION(year = 2021, month = 10,day=2) *> SET LOCATION '/logs/2021/09/03';*

```
hive> ALTER TABLE logmsgs PARTITION(year = 2011, month = 1,day=1)
    >       SET LOCATION '/logs/2021/09/03';
OK
Time taken: 0.407 seconds
```

This command does not move the data from the old location, nor does it delete the old data.

DROP A PARTITION:

COMMANDS
ALTER TABLE logmsgs DROP IF EXISTS PARTITION(year = 2021, month=10, *day=2);*

The IF EXISTS clause is optional, as usual. For managed tables, the data for the partition is deleted, along with the metadata, even if the partition was created using ALTER TABLE ... ADD PARTITION. For external tables, the data is not deleted. There are a few more ALTER statements that affect partitions discussed later in Alter Storage Properties and Miscellaneous Alter Table Statements.

26. *Changing Columns*

You can rename a column, change its position, type, or comment:

COMMANDS
ALTER TABLE logmsgs *> CHANGE COLUMN hms hours_minutes_seconds INT* *> COMMENT 'The hours, minutes, and seconds part of the timestamp'* *> AFTER severity;*

```
hive> ALTER TABLE logmsgs
    >       CHANGE COLUMN hms hours_minutes_seconds INT
    >       COMMENT 'The hours, minutes, and seconds part of the timestamp'
    >       AFTER severity;
```

You have to specify the old name, a new name, and the type, even if the name or type is not changing. The keyword COLUMN is optional as is the COMMENTclause.

If you aren't moving the column, the AFTER other_column clause is not necessary. In the example shown, we move the column after the severitycolumn. If you want to move the column to the first position, use FIRST instead of AFTER other_column. As always, this command changes metadata only. If you are moving columns, the data must already match the new schema or you must change it to match by some other means.

27. *Adding Columns*

You can add new columns to the end of the existing columns, before any partition columns.

COMMANDS
ALTER TABLE logmsgs ADD COLUMNS (*> app_name STRING COMMENT 'Application name');*

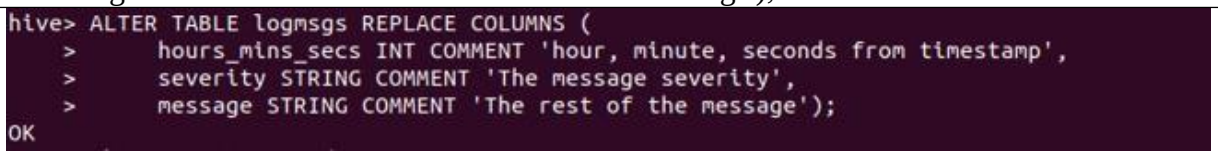
```
hive> ALTER TABLE logmsgs ADD COLUMNS (
    >       app_name STRING COMMENT 'Application name');
OK
Time taken: 0.583 seconds
```

The COMMENT clauses are optional, as usual. If any of the new columns are in the wrong position, use an ALTER COLUMN table CHANGE COLUMN statement for each one to move it to the correct position.

28. *Deleting or Replacing Columns*

COMMANDS

```
ALTER TABLE logmsgs REPLACE COLUMNS (
  > hours_mins_secs INT COMMENT 'hour, minute, seconds from timestamp',
  > severity STRING COMMENT 'The message severity',
  > message STRING COMMENT 'The rest of the message');
```



This statement effectively renames the original hms column and removes the server and process_id columns from the original schema definition. As for all ALTER statements, only the table metadata is changed. The REPLACE statement can only be used with tables that use one of the nativeSerDe modules: DynamicSerDe or MetadataTypedColumn setSerDe. Recall that the SerDe determines how records are parsed into columns (deserialization) and how a record's columns are written to storage (serialization). SeeChapter 15 for more details on SerDes.

29. *Alter Table Properties*

You can add additional table properties or modify existing properties, but not remove them:

COMMANDS
ALTER TABLE logmsgs SET TBLPROPERTIES (*> 'notes' = 'The process id is no longer captured; this column is always NULL');*

You can alter the storage properties that we discussed in Creating Tables:

COMMANDS
ALTER TABLE stocks *> CLUSTERED BY (exchangee, symbol)* *> SORTED BY (symbol)* *> INTO 48 BUCKETS;*

```
hive>
    > ALTER TABLE stocks
    >       CLUSTERED BY (exchangee, symbol)
    >       SORTED BY (symbol)
    >       INTO 48 BUCKETS;
OK
Time taken: 0.249 seconds
```

The SORTED BY clause is optional, but the CLUSTER BY and INTO ... BUCKETS are required. (See also Bucketing Table Data Storage for information on the use of data bucketing.)

30. *Miscellaneous Alter Table Statements*

COMMANDS
ALTER TABLE logmsgs TOUCH *> PARTITION(year = 2011, month = 1, day = 1);*

```
hive> ALTER TABLE logmsgs TOUCH
    >       PARTITION(year = 2011, month = 1, day = 1);
OK
```

31. IEW

Apache Hive supports the features of views, which are logical constructs and treated the same as tables. It makes it possible to save the query, on which all DML (Data Manipulation Language) commands can be performed. It is similarly used as views in SQL; however, views are created based on user requirement. In Hive, the query referencing the view is executed first, and then the result obtained is used in the rest of the query. Also, the Hive's query planner helps in the execution plan.

Views are similar to tables, which are generated based on the requirements.

- *We can save any result set data as a view in Hive*
- *Usage is similar to as views used in SQL*
- *All type of DML operations can be performed on a view*

Syntax:

Create VIEW < VIEWNAME> AS SELECT

COMMANDS
*Create VIEW Sample_View AS SELECT * FROM employees2 WHERE salary>25000;*

```
hive> Create VIEW Sample_View AS SELECT * FROM employeesdetails WHERE salary>25000;
OK
Time taken: 4.418 seconds
```

In this example, we are creating view Sample_View where it will display all the row values with salary field greater than 25000.

32. *INDEX:*

Indexes are pointers to particular column name of a table.

• The user has to manually define the index

• Wherever we are creating index, it means that we are creating pointer to particular column name of table

• Any Changes made to the column present in tables are stored using the index value created on the column name.

Syntax:

Create INDEX < INDEX_NAME> ON TABLE < TABLE_NAME(column names)>

Example:

Create INDEX sample_Index ON TABLE guruhive_internaltable(id)

Here we are creating index on table guruhive_internaltable for column name id.

33. *FUNCTIONS:*

Functions are built for a specific purpose to perform operations like Mathematical, arithmetic, logical and relational on the operands of table column names.

Built-in functions

These are functions that already available in Hive. First, we have to check the application requirement, and then we can use this built in functions in our applications.

We can call these functions directly in our application. The syntax and types are mentioned in the following section.

Types of Built-in Functions in HIVE

• Collection Functions

• Date Functions

• Mathematical Functions

• Conditional Functions

• String Functions

• Misc. Functions

Collection Functions: These functions are used for collections. Collections mean the grouping of elements and returning single or array of elements depends on return type mentioned in function name.

Date Functions: These are used to perform Date Manipulations and Conversion of Date types from one type to another type.

Mathematical Functions: These functions are used for Mathematical Operations. Instead of creatingUDFs , we have some inbuilt mathematical functions in Hive.

Conditional Functions: These functions used for conditional values checks.

String Functions: String manipulations and string operations these functions can be called.

UDFs (User Defined Functions):

In Hive, the users can define own functions to meet certain client requirements. These are known as UDFs in Hive. User Defined Functions written in Java for specific modules.

Some of UDFs are specifically designed for the reusability of code in application frameworks. The developer will develop these functions in Java and integrate those UDFs with the Hive.

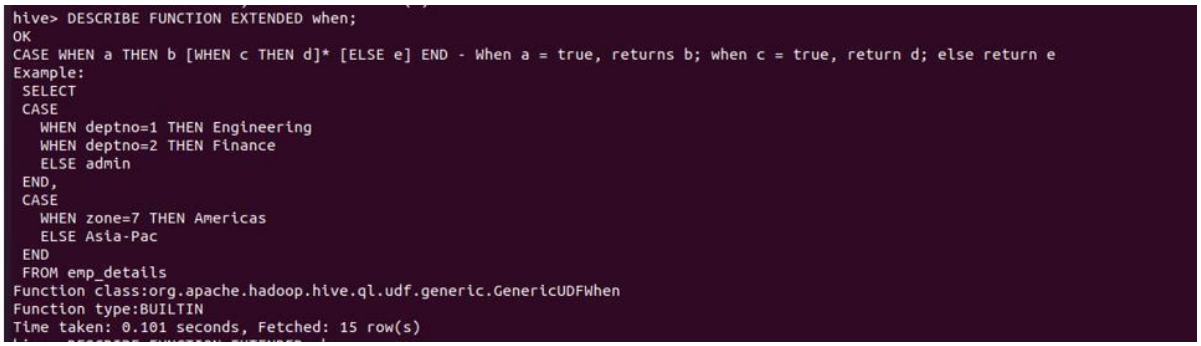
During the Query execution, the developer can directly use the code, and UDFs will return outputs according to the user defined tasks. It will provide high performance in terms of coding and execution.

For example, for string stemming we don't have any predefined function in Hive, for this we can write stem UDF in Java. Wherever we require Stem functionality, we can directly call this Stem UDF in Hive.

Here stem functionality means deriving words from its root words. It is like stemming algorithm reduces the words "wishing", "wished", and "wishes" to the root word "fish." For performing this type functionality, we can write UDF in java and integrate with Hive.

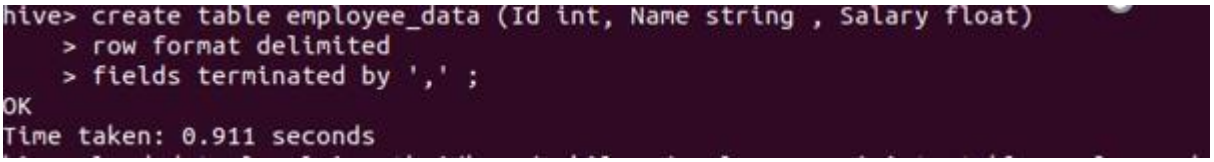
Depending on the use cases the UDFs can be written, it will accept and produce different numbers of input and output values. The general type of UDF will accept single input value and produce a single output value.

If the UDF used in the query, then UDF will be called once for each row in the result data set. In the other way, it can accept a group of values as input and return single output value as well.



Let's create a table and load the data into it by using the following steps: - Select the database in which we want to create a table.

COMMANDS
create table employee_data (Id int, Name string , Salary float) *> row format delimited* *> fields terminated by ',' ;*



Now, load the data into the table.

COMMANDS
load data local inpath '/home/gokilanm/employee.csv' into table employee_data;

```
hive> load data local inpath '/home/gokilanm/employee.csv' into table employee_data;
Loading data to table default.employee_data
OK
Time taken: 1.282 seconds
```

Let's fetch the loaded data by using the following command: -

COMMANDS
*select * from employee_data;*

```
hive> select * from employee_data;
OK
1       "gokilanm"      40000.0
2       "rithun"        56666.0
3       "aadhav"        60000.0
4       "yash"  40000.0
5       "lathika"       10000.0
Time taken: 2.891 seconds, Fetched: 5 row(s)
```

Mathematical Functions in Hive:

1.To fetch the square root of each employee's salary.

COMMANDS
select Id, Name, sqrt(Salary) from employee_data ;

```
hive>  select Id, Name, sqrt(Salary) from employee_data ;
OK
1       "gokilanm"      200.0
2       "rithun"        238.04621400055913
3       "aadhav"        244.94897427831782
4       "yash"  200.0
5       "lathika"       100.0
Time taken: 0.686 seconds, Fetched: 5 row(s)
```

2. To fetch the maximum salary of an employee.

COMMANDS
select max(Salary) from employee_data;

```
hive> select max(Salary) from employee_data;
Query ID = gokilanm_20211019143052_d7634b81-f61d-4ff4-8869-7cba4a7722fc
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1634633832506_0001, Tracking URL = http://gokila-VirtualBox:8088/proxy/application_1634633832506_0001/
Kill Command = /home/gokilanm/hadoop-3.2.2/bin/mapred job  -kill job_1634633832506_0001
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2021-10-19 14:31:19,090 Stage-1 map = 0%,  reduce = 0%
2021-10-19 14:31:27,810 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 2.54 sec
2021-10-19 14:31:36,448 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 4.2 sec
MapReduce Total cumulative CPU time: 4 seconds 200 msec
Ended Job = job_1634633832506_0001
```

```
2021-10-19 14:31:36,448 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 4.2 sec
MapReduce Total cumulative CPU time: 4 seconds 200 msec
Ended Job = job_1634633832506_0001
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 4.2 sec   HDFS Read: 12598 HDFS Write: 107 SUCCESS
Total MapReduce CPU Time Spent: 4 seconds 200 msec
OK
60000.0
Time taken: 45.046 seconds, Fetched: 1 row(s)
```

3. *To fetch the minimum salary of an employee.*

COMMANDS
select min(Salary) from employee_data;

```
hive>  select min(Salary) from employee_data;
Query ID = gokilanm_20211019143157_4781a7a6-b2d6-4d6d-aeeb-cd8d91c8ddad
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1634633832506_0002, Tracking URL = http://gokila-VirtualBox:8088/proxy/application_1634633832506_0002/
Kill Command = /home/gokilanm/hadoop-3.2.2/bin/mapred job  -kill job_1634633832506_0002
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2021-10-19 14:32:13,733 Stage-1 map = 0%,  reduce = 0%
2021-10-19 14:32:22,263 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 1.78 sec
2021-10-19 14:32:29,663 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 3.44 sec
MapReduce Total cumulative CPU time: 3 seconds 440 msec
Ended Job = job_1634633832506_0002
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 3.44 sec   HDFS Read: 12677 HDFS Write: 107 SUCCESS
Total MapReduce CPU Time Spent: 3 seconds 440 msec
OK
10000.0
Time taken: 34.357 seconds, Fetched: 1 row(s)
```

4. *To fetch the average salary of an employee.*

COMMANDS
select avg(Salary) from employee_data;

```
hive>  select avg(Salary) from employee_data;
Query ID = gokilanm_20211019143729_2aad9420-cc75-4485-8011-0781771c480a
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1634633832506_0003, Tracking URL = http://gokila-VirtualBox:8088/proxy/application_1634633832506_0003/
Kill Command = /home/gokilanm/hadoop-3.2.2/bin/mapred job  -kill job_1634633832506_0003
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2021-10-19 14:38:00,950 Stage-1 map = 0%,  reduce = 0%
2021-10-19 14:38:25,862 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 3.19 sec
2021-10-19 14:38:49,596 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 7.72 sec
MapReduce Total cumulative CPU time: 7 seconds 720 msec
Ended Job = job_1634633832506_0003
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 7.72 sec   HDFS Read: 14831 HDFS Write: 107 SUCCESS
Total MapReduce CPU Time Spent: 7 seconds 720 msec
OK
41333.2
Time taken: 81.795 seconds, Fetched: 1 row(s)
```

5. *To fetch the sum of salary of an employee.*

COMMANDS
select sum(Salary) from employee_data;

```
hive>  select sum(salary) from employee_data;
Query ID = gokilanm_20211019144554_b0efc218-a6ff-4dae-9a27-a1516f13020a
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1634633832506_0005, Tracking URL = http://gokila-VirtualBox:8088/proxy/application_1634633832506_0005/
Kill Command = /home/gokilanm/hadoop-3.2.2/bin/mapred job  -kill job_1634633832506_0005
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2021-10-19 14:46:19,767 Stage-1 map = 0%,  reduce = 0%
2021-10-19 14:46:37,584 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 2.25 sec
2021-10-19 14:46:54,842 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 4.6 sec
MapReduce Total cumulative CPU time: 4 seconds 600 msec
Ended Job = job_1634633832506_0005
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 4.6 sec   HDFS Read: 13085 HDFS Write: 108 SUCCESS
Total MapReduce CPU Time Spent: 4 seconds 600 msec
OK
206666.0
Time taken: 63.026 seconds, Fetched: 1 row(s)
```

6.Count:

COMMANDS
select Count(*) from employee_data;

```
hive>  select count(*) from employee_data;
Query ID = gokilanm_20211019144035_edee5381-a020-44d1-acd4-a29546cff95f
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1634633832506_0004, Tracking URL = http://gokila-VirtualBox:8088/proxy/application_1634633832506_0004/
Kill Command = /home/gokilanm/hadoop-3.2.2/bin/mapred job  -kill job_1634633832506_0004
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2021-10-19 14:41:10,248 Stage-1 map = 0%,  reduce = 0%
2021-10-19 14:41:27,077 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 2.05 sec
2021-10-19 14:41:46,922 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 4.81 sec
MapReduce Total cumulative CPU time: 4 seconds 810 msec
Ended Job = job_1634633832506_0004
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 4.81 sec   HDFS Read: 12658 HDFS Write: 101 SUCCESS
Total MapReduce CPU Time Spent: 4 seconds 810 msec
OK
5
Time taken: 72.328 seconds, Fetched: 1 row(s)
hive>
```

7.To fetch the employees whose salary is greather than 25000

COMMANDS
select sum(Salary) from employee_data;

```
hive> select * from employee_data where salary >= 25000;
OK
1       "gokilanm"      40000.0
2       "rithun"        56666.0
3       "aadhav"        60000.0
4       "yash"  40000.0
Time taken: 0.549 seconds, Fetched: 4 row(s)
```

8. To fetch the name of each employee in uppercase.

COMMANDS
select Id, upper(Name) from employee_data;

```
hive> select Id, upper(Name) from employee_data;
OK
1       "GOKILANM"
2       "RITHUN"
3       "AADHAV"
4       "YASH"
5       "LATHIKA"
Time taken: 0.297 seconds, Fetched: 5 row(s)
```

9. To fetch the name of each employee in lowercase.

COMMANDS
select Id, lower(Name) from employee_data;

```
hive> select Id, lower(Name) from employee_data;
OK
1       "gokilanm"
2       "rithun"
3       "aadhav"
4       "yash"
5       "lathika"
Time taken: 0.327 seconds, Fetched: 5 row(s)
```

RESULT:

The Hive operations like create, alter, and drop databases, tables, views,functions, and indexes are implemented successfully.

Experiment 10: Verify, Sparse and perform advance join of data using spark

a) *OBJECTIVE OF THE EXERCISE/EXPERIMENT*

To Verify, Sparse and perform advance join of data using spark

b) *Facilities/material required to do the exercise/experiment:*

Sl.No.	*Facilities/material required*	*Quantity*
1.	*Hadoop*	*As per strength*
2.	*Spark*	*As per strength*

THEORY:

Join operations in Apache Spark is often a biggest source of performance problems and even full-blown exceptions in Spark. After this talk, you will understand the two most basic methods Spark employs for joining dataframes – to the level of detail of how Spark distributes the data within the cluster. You'll also find out how to work out common errors and even handle the trickiest corner cases we've encountered! After this talk, you should be able to write performance joins in Spark SQL that scale and are zippy fast!
This session will cover different ways of joining tables in Apache Spark.

ShuffleHashJoin
– A ShuffleHashJoin is the most basic way to join tables in Spark
– we'll diagram how Spark shuffles the dataset to make this happen.
BroadcastHashJoin
– A BroadcastHashJoin is also a very common way for Spark to join two tables under the special condition that one of your tables is small.

Dealing with Key Skew in a ShuffleHashJoin
– Key Skew is a common source of slowness for a Shuffle Hash Join
– we'll describe what this is and how you might work around this.
CartesianJoin
– Cartesian Joins is a hard problem – we'll describe why it's difficult as well as what you need to do to make that work and what to look out for.

One to Many Joins
– When a single row in one table can match to many rows in your other table, the total number of output rows in your joined table can be really high. We'll let you know how to deal with this.
Theta Joins
– If you aren't joining two tables strictly by key, but instead checking on a condition for your tables, you may need to provide some hints to Spark SQL to get this to run well.

1. Broadcast Hash Join

Broadcast Hash Join in Spark works by broadcasting the small dataset to all the executors and once the data is broadcasted a standard hash join is performed in all the executors. Broadcast Hash Join happens in 2 phases.

Broadcast phase – small dataset is broadcasted to all executors

Hash Join phase – small dataset is hashed in all the executors and joined with the partitioned big dataset.

spark.sql.autoBroadcastJoinThreshold – max size of dataframe that can be broadcasted. The default is 10 MB. Which means only datasets below 10 MB can be broadcasted.

We have 2 DataFrames df1 and df2 with one column in each – id1 and id2 respectively. We are doing a simple join on id1 and id2

COMMANDS
scala> spark.conf.get("spark.sql.autoBroadcastJoinThreshold") res1: String = 10485760 *scala> val data1 = Seq(10, 20, 20, 30, 40, 10, 40, 20, 20, 20, 20, 50)* *data1: Seq[Int] = List(10, 20, 20, 30, 40, 10, 40, 20, 20, 20, 20, 50)* *scala> val df1 = data1.toDF("id1")* *df1: org.apache.spark.sql.DataFrame = [id1: int] scala> val data2 = Seq(30, 20, 40, 50)* *data2: Seq[Int] = List(30, 20, 40, 50) scala> val df2 = data2.toDF("id2")* *df2: org.apache.spark.sql.DataFrame = [id2: int]*

Note that you can also use the broadcast function to specify the

dataframe you like to broadcast. And the syntax would look like –
df1.join(broadcast(df2),
$"id1" === $"id2")

COMMANDS
val dfJoined = df1.join(df2, $"id1" === $"id2")

COMMANDS
scala> dfJoined.queryExecution.executedPlan *scala> dfJoined.show*

```
scala> spark.conf.get("spark.sql.autoBroadcastJoinThreshold")
res0: String = 10485760b

scala> val data1 = Seq(10, 20, 20, 30, 40, 10, 40, 20, 20, 20, 20, 50)
data1: Seq[Int] = List(10, 20, 20, 30, 40, 10, 40, 20, 20, 20, 20, 50)

scala> val df1 = data1.toDF("id1")
df1: org.apache.spark.sql.DataFrame = [id1: int]

scala> val data2 = Seq(30, 20, 40, 50)
data2: Seq[Int] = List(30, 20, 40, 50)

scala> val df2 = data2.toDF("id2")
df2: org.apache.spark.sql.DataFrame = [id2: int]

scala> val dfJoined = df1.join(df2, $"id1" === $"id2")
dfJoined: org.apache.spark.sql.DataFrame = [id1: int, id2: int]

scala> dfJoined.queryExecution.executedPlan
res1: org.apache.spark.sql.execution.SparkPlan =
*(1) BroadcastHashJoin [id1#4], [id2#10], Inner, BuildRight, false
:- *(1) LocalTableScan [id1#4]
+- BroadcastExchange HashedRelationBroadcastMode(List(cast(input[0, int, false] as bigint)),false), [id=#11]
   +- LocalTableScan [id2#10]

scala> dfJoined.show
+---+---+
|id1|id2|
+---+---+
| 20| 20|
| 20| 20|
| 30| 30|
| 40| 40|
| 40| 40|
| 20| 20|
| 20| 20|
| 20| 20|
| 20| 20|
| 50| 50|
+---+---+
```

Broadcast join doesn't work for non-equi joins

Broadcast join doesn't work for full outer joins

Broadcast Hash Join doesn't work well if the dataset that is being broadcasted is big.

If the size of the broadcasted dataset is big, it could become a network intensive operation and cause your job execution to slow down.

If the size of the broadcasted dataset is big, you would get an OutOfMemory exception when Spark builds the Hash table on the data. Because the Hash table will be kept in memory.

2.*Shuffle Hash Join*

Shuffle Hash Join, as the name indicates works by shuffling both

datasets. So the same keys from both sides end up in the same partition or task. Once the data is shuffled, the smallest of the two will be hashed into buckets and a hash join is performed within the partition.

Shuffle Hash Join is different from Broadcast Hash Join because the entire dataset is not broadcasted instead both datasets are shuffled and then the smallest side data is hashed and bucketed and hash joined with the bigger side in all the partitions. Shuffle Hash Join is divided into 2 phases.

Shuffle phase – both datasets are shuffled

Hash Join phase – smaller side data is hashed and bucketed and hash joined with he bigger side in all the partitions.

Sorting is not needed with Shuffle Hash Joins inside the partitions.

COMMANDS
scala> spark.conf.set("spark.sql.autoBroadcastJoinThreshold", 2) scala> spark.conf.set("spark.sql.join.preferSortMergeJoin", "false") *scala> spark.conf.get("spark.sql.join.preferSortMergeJoin")* *res2: String = false* *scala> spark.conf.get("spark.sql.autoBroadcastJoinThreshold") res3: String = 2* *scala> val data1 = Seq(10, 20, 20, 30, 40, 10, 40, 20, 20, 20, 20, 50)* *data1: Seq[Int] = List(10, 20, 20, 30, 40, 10, 40, 20, 20, 20, 20, 50)* *scala> val df1 = data1.toDF("id1")* *df1: org.apache.spark.sql.DataFrame = [id1: int]* *scala> val data2 = Seq(30, 20, 40, 50)* *data2: Seq[Int] = List(30, 20, 40, 50)* *scala> val df2 = data2.toDF("id2")* *df2: org.apache.spark.sql.DataFrame = [id2: int]* *scala> val dfJoined = df1.join(df2, $"id1" === $"id2")* *dfJoined: org.apache.spark.sql.DataFrame = [id1: int, id2: int]*

```
scala> spark.conf.set("spark.sql.autoBroadcastJoinThreshold", 2)

scala> spark.conf.set("spark.sql.join.preferSortMergeJoin", "false")

scala> spark.conf.get("spark.sql.join.preferSortMergeJoin")
res5: String = false

scala> spark.conf.get("spark.sql.autoBroadcastJoinThreshold")
res6: String = 2

scala> val data1 = Seq(10, 20, 20, 30, 40, 10, 40, 20, 20, 20, 20, 50)
data1: Seq[Int] = List(10, 20, 20, 30, 40, 10, 40, 20, 20, 20, 20, 50)

scala> val df1 = data1.toDF("id1")
df1: org.apache.spark.sql.DataFrame = [id1: int]

scala> val data2 = Seq(30, 20, 40, 50)
data2: Seq[Int] = List(30, 20, 40, 50)

scala> val df2 = data2.toDF("id2")
df2: org.apache.spark.sql.DataFrame = [id2: int]

scala> val dfJoined = df1.join(df2, $"id1" === $"id2")
dfJoined: org.apache.spark.sql.DataFrame = [id1: int, id2: int]

scala> dfJoined.queryExecution.executedPlan
res7: org.apache.spark.sql.execution.SparkPlan =
*(1) ShuffledHashJoin [id1#29], [id2#35], Inner, BuildRight
:- Exchange hashpartitioning(id1#29, 200), ENSURE_REQUIREMENTS, [id=#55]
:  +- LocalTableScan [id1#29]
+- Exchange hashpartitioning(id2#35, 200), ENSURE_REQUIREMENTS, [id=#56]
   +- LocalTableScan [id2#35]
```

When we see the plan that will be executed, we can see that ShuffledHashJoin is used.

COMMANDS
scala> dfJoined.queryExecution.executedPlan *scala> dfJoined.show*

```
scala> dfJoined.queryExecution.executedPlan
res7: org.apache.spark.sql.execution.SparkPlan =
*(1) ShuffledHashJoin [id1#29], [id2#35], Inner, BuildRight
:- Exchange hashpartitioning(id1#29, 200), ENSURE_REQUIREMENTS, [id=#55]
:  +- LocalTableScan [id1#29]
+- Exchange hashpartitioning(id2#35, 200), ENSURE_REQUIREMENTS, [id=#56]
   +- LocalTableScan [id2#35]

scala> dfJoined.show
+---+---+
|id1|id2|
+---+---+
| 20| 20|
| 20| 20|
| 40| 40|
| 40| 40|
| 20| 20|
| 20| 20|
| 20| 20|
| 20| 20|
| 50| 50|
| 30| 30|
+---+---+
```

- *Faster than a sort merge join since sorting is not involved.* *Works only for equi joins*
- *Works for all join types*
- *Works well when a dataset can not be broadcasted but one side of partitioned data after shuffling will be small enough for hash join.*

Does not work with data which are heavily skewed. Let's say we are joining a sales dataset on the product key. It is possible that the dataset has a disproportionate number of records for a certain product key.Hashing all the records for this product key inside a single partition will result in an Out of Memory exception. So Shuffle Hash Join will work for a balanced dataset but not for skewed dataset.

3. CARTESIAN PRODUCT JOIN

Cartesian Product Join (a.k.a Shuffle-and-Replication Nested Loop) join works very similar to a Broadcast Nested Loop join except the dataset is not broadcasted.

Shuffle-and-Replication does not mean a "true" shuffle as in records with the same keys are sent to the same partition. Instead the entire partition of the dataset is sent over or replicated to all the partitions for a full cross or nested- loop join.

We are setting spark.sql.autoBroadcastJoinThreshold to -1 to disable broadcast.

COMMANDS
1.scala> *spark.conf.get("spark.sql.join.preferSortMergeJoin")* *res1: String = true* *2.scala>* *spark.conf.get("spark.sql.autoBroadcastJoinThreshold")* *res2: String = -1* *3.scala> val data1 = Seq(10, 20, 20, 30, 40, 10, 40, 20, 20, 20, 20, 50)* *data1: Seq[Int] = List(10, 20, 20, 30, 40, 10, 40, 20, 20, 20, 20, 50)* *4.scala> val df1 = data1.toDF("id1")* *df1: org.apache.spark.sql.DataFrame =* *[id1: int] 5.scala> val data2 = Seq(30, 20,* *40, 50)* *data2: Seq[Int] = List(30, 20,* *40, 50) 6.scala> val df2 =* *data2.toDF("id2")* *df2: org.apache.spark.sql.DataFrame = [id2: int]*

```
scala> spark.conf.get("spark.sql.join.preferSortMergeJoin")
res9: String = false

scala> spark.conf.get("spark.sql.autoBroadcastJoinThreshold")
res10: String = 2

scala> val data1 = Seq(10, 20, 20, 30, 40, 10, 40, 20, 20, 20, 20, 50)
data1: Seq[Int] = List(10, 20, 20, 30, 40, 10, 40, 20, 20, 20, 20, 50)

scala> val df1 = data1.toDF("id1")
df1: org.apache.spark.sql.DataFrame = [id1: int]

scala> val data2 = Seq(30, 20, 40, 50)
data2: Seq[Int] = List(30, 20, 40, 50)

scala> val df2 = data2.toDF("id2")
df2: org.apache.spark.sql.DataFrame = [id2: int]

scala> val dfJoined = df1.join(df2, $"id1" >= $"id2")
dfJoined: org.apache.spark.sql.DataFrame = [id1: int, id2: int]

scala> dfJoined.queryExecution.executedPlan
res11: org.apache.spark.sql.execution.SparkPlan =
CartesianProduct (id1#54 >= id2#60)
:- LocalTableScan [id1#54]
+- LocalTableScan [id2#60]
```

Note here we are trying to perform a non-equi join operation.

COMMANDS
scala> val dfJoined = df1.join(df2, $"id1" >= $"id2") dfJoined: org.apache.spark.sql.DataFrame = [id1: int, id2: int]

When we see the plan that will be executed, we can see that CartesianProduct is used.

COMMANDS
scala> dfJoined.queryExecution.executedPlan scala> dfJoined.show

```
scala> dfJoined.show
+---+---+
|id1|id2|
+---+---+
| 20| 20|
| 20| 20|
| 30| 30|
| 30| 20|
| 40| 30|
| 40| 20|
| 40| 40|
| 40| 30|
| 40| 20|
| 40| 40|
| 20| 20|
| 20| 20|
| 20| 20|
| 20| 20|
| 50| 30|
| 50| 20|
| 50| 40|
| 50| 50|
+---+---+
```

Cartesian Product Join work:

- *Works in both equi and non-equi joins*
- *Works only on inner like joins*

Cartesian Product Join doesn't work:

- *Doesn't work on non inner like joins*
- *This is a very expensive join algorithm. Except load on the network and partitions are moved across the network.*
- *High possibility of Out of Memory exception.*

Result :

Thus implementation of Sparse and perform advance join of data using spark is verified .