

Hadoop 2.9.0 Lab Guide

Contents

Lab 1 : Working with HDFS.....	3
Lab 2 : Working with MapReduce	4
Lab 3 : MapReduce: WordCount Program.....	5
Lab 4 : Sqoop: Data Import and Export	7
Lab 5 : Flume: Data Ingestion.....	9
Lab 6 : Creating Hive Tables.....	11
Lab 7 : Creating UDF in Hive.....	16
Lab 8 : Writing Pig Scripts	18
Lab 9 : Working with Impala	20
Lab 10 : Working with HBase	21
Lab 11 : Working with Spark	23
Appendix.....	25

Virtual Machine Details:

The virtual machine provided contains Ubuntu Server 16 installed with the required updates.

- A. VMWare Player 14
- B. Linux Version – Ubuntu 16 server
- C. User Name : **trgadmin** Password: trgadmin
- D. Directories required for completing the lab exercises:

The following sub-directories are under /home/trgadmin

Directory	Description
lab	Root for all lab activities
sw	Home directory of all the installed software
sw/installers	Contains all Installers for Hadoop, Sqoop, Flume, Hive, HBase and Spark
lab/hdfs	For configuring hdfs related contents
lab/data	Input files for lab exercises
lab/data/programs	Demo programs

Lab 1 : Working with HDFS

a) Move files between filesystem and distributed file system

A. Create an input and output directory under hdfs for all input and output files

```
$ hdfs dfs -mkdir /input
```

```
$ hdfs dfs -mkdir /output
```

B. Check newly created directories

```
$ hdfs dfs -ls /
```

C. Copy files from local system to hdfs and check if the files are copied

```
$ hdfs dfs -copyFromLocal /home/trgadmin/lab/data/txns /input
```

```
$ hdfs dfs -copyFromLocal /home/trgadmin/lab/data/custs /input
```

```
$ hdfs dfs -copyFromLocal /home/trgadmin/lab/data/products.txt /input
```

```
$ hdfs dfs -ls /input
```

D. Go to datanode1 folder and check how the files are split and multiple blocks are stored [Hint: Check the size of the files in the folder called current, where the blk files would be stored]

E. Copy files from hdfs to local system and check if the file is copied

```
$ hdfs dfs -copyToLocal /input/custs /home/trgadmin/
```

```
$ ls /home/trgadmin/
```

Lab 2 : Working with MapReduce

The following example copies the xml files from configuration directory to use as input and then finds and displays every match of the given regular expression. Output is written to the given output directory.

1. Make the HDFS directories required to execute MapReduce jobs:

```
$ hdfs dfs -mkdir /user/trgadmin/input
```

Note: This folder is required, if you put files into HDFS by specifying just the folder name. HDFS is similar to Linux FS and / represents root of HDFS. Relative paths represent paths inside of /user/<username>/.

Try and see the file structure by executing these two commands:

```
$ hdfs dfs -mkdir /test
```

```
$ hdfs dfs -mkdir test
```

2. Copy the input files into the distributed filesystem:

```
$ hdfs dfs -put $HADOOP_HOME/etc/hadoop/*.xml input
```

3. Run some of the examples provided:

```
$ hadoop jar share/hadoop/mapreduce/hadoop-mapreduce-examples-2.9.0.jar grep  
input output 'dfs[a-z.]+'
```

4. Examine the output files: Copy the output files from the distributed filesystem to the local filesystem and examine them:

```
$ hdfs dfs -get output out_files
```

```
$ cat out_files/*
```

OR

View the output files on the distributed filesystem:

```
$ hdfs dfs -cat out_files/*
```

Lab 3 : MapReduce: WordCount Program

a) Create a Java Project

Create a new Java Project (empty - not based on any template) in IntelliJ Idea. Name the project as WordCount.

Copy lab/data/programs/WordCount.java file into src folder of WordCount project.

Compile the project and observe there are many errors due to missing libraries.

b) Attach the required libraries

Select File -> Project Structure -> Project Settings -> Modules.

Select Dependencies tab on the right hand side screen. Click on '+' icon to include libraries.

Select all the .jar files from these locations individually and include:

`$HADOOP_HOME/share/hadoop/common/`

`$HADOOP_HOME/share/hadoop/common/lib/`

`$HADOOP_HOME/share/hadoop/mapreduce/`

`$HADOOP_HOME/share/hadoop/mapreduce/lib/`

`$HADOOP_HOME/share/hadoop/yarn/`

`$HADOOP_HOME/share/hadoop/yarn/lib/`

Once all the libraries are included, click OK.

Recompile the code and verify all the errors are removed.

c) Build the jar file

Select File -> Project Structure -> Project Settings -> Artifacts.

From the right hand side screen, click on '+' icon and select Jar -> From modules with dependencies.

Name the jar file as WordCount.jar.

Click on "Include in project build" check box. and Click OK.

Build the project using Build -> Build Project menu or CTRL+F9. The class files and the jar file will be created in the output folder.

Copy the WordCount.jar file into lab/data/ folder.

d) Run the WordCount program on hadoop

Change the directory to lab/data/

```
$ cd lab/data/
```

Create a text file WCDData.txt and write a few lines in it.

```
$ sudo nano WCDData.txt
```

After saving the file, copy it on to hdfs.

```
$ hadoop fs -put lab/data/WCDData.txt /input
```

Run the mapreduce job.

```
$ hadoop jar WordCount.jar com.hadoop.WordCount /input/WCDData.txt /output/MR-WordCount
```

Note: Make sure output folder does not exists before executing the job.

Lab 4 : Sqoop: Data Import and Export

a) Logging into mysql prompt

```
$ mysql -u root -p
The password is root
```

b) Create and select a database

```
mysql> create database retail;
mysql> use retail;
```

c) Create table

```
mysql> CREATE TABLE customer (CustID VARCHAR (7) NOT NULL,
    FirstName VARCHAR(20) NOT NULL,
    LastName VARCHAR(20) NOT NULL,
    Age INT NOT NULL,
    Profession VARCHAR(50) NOT NULL,
    PRIMARY KEY ( CustID));
```

d) Export data

#sqoop export command

```
$ sqoop export --connect jdbc:mysql://localhost/retail --table customer --username root
--password root --export-dir /input/custs
```

e) Verify Exported data

Login to mysql console and verify

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

Verify if you have exported 9999 customer records to mysql or not.

f) Import data

#sqoop import command

```
$ sqoop import --connect jdbc:mysql://localhost/retail --table customer --username root
```

```
--password root -target-dir output/sqoop --driver com.mysql.jdbc.Driver -m 1
```

g) Verify imported data

Check the imported file in hdfs
\$ hadoop fs -cat output/sqoop/part-m-00000

h) Import selected columns

```
$ sqoop import --connect jdbc:mysql://localhost/retail --table customer --columns  
'CustID,FirstName,age' --username root --password root -target-dir output/sqoop1 --driver  
com.mysql.jdbc.Driver -m 1
```

i) Filter records for import

```
$ sqoop import --connect jdbc:mysql://localhost/retail --table customer --where 'Age > 45' --  
username root --password root -target-dir output/sqoop --driver com.mysql.jdbc.Driver -m 1
```

j) Import records based on a query

```
$ sqoop import --connect jdbc:mysql://localhost/retail --query 'select CustID, LastName, Age  
from customer where $CONDITIONS' --split-by Age --username root --password root -target-dir  
output/sqoop1 --driver com.mysql.jdbc.Driver
```

k) View the list of databases

```
$ sqoop list-databases --connect jdbc:mysql://localhost/ --username root --password root
```


Lab 5 : Flume: Data Ingestion

a) Ingest data from exec source into HDFS sink.

Exec source type is used to run a command outside Flume environment. For using Exec type, set the type property to exec.

Another property that need to be specified here is the command property which need to be executed.

Create the configuration file exec-hdfs.conf in the \$FLUME_HOME/conf folder. The contents of it is given below:

```
# Sources, channels and sinks are defined per agent,
# in this case called 'agent'

agent.sources = Source1
agent.channels = C1
agent.sinks = Sink1

# For each one of the sources, the type is defined
agent.sources.Source1.type = exec
agent.sources.Source1.command = cat /home/trgadmin/lab/data/custs

# The channel can be defined as follows.
agent.sources.Source1.channels = C1

# Each sink's type must be defined
agent.sinks.Sink1.type = hdfs
agent.sinks.Sink1.hdfs.path = output/flume

#Specify the channel the sink should use
agent.sinks.Sink1.channel = C1

# Each channel's type is defined.
agent.channels.C1.type = memory

# Other config values specific to each type of channel(sink or source) can be defined as well
# In this case, it specifies the capacity of the memory channel
agent.channels.C1.capacity = 100
```

b) Execute Flume script

Now use the following command to run the flume agent after navigating to the \$FLUME_HOME folder.

```
$ flume-ng agent --conf conf --conf-file conf/exec-hdfs.conf --name agent  
-Dflume.root.logger=DEBUG,INFO,console
```

c) Verify the results

Verify the command output on HDFS directory. Browse to output/flume folder and check the contents.

Lab 6 : Creating Hive Tables

a) Hive Programming

1. Create and select database

Connect to hive using the following command

```
$ hive
```

In the hive shell, execute the hive commands.

```
hive> create database retail;
```

Check the retail.db folder created in /user/hive/warehouse/

```
hive> use retail;
```

2. Create table for storing transactional records

```
hive> CREATE TABLE txnrecords (txnno INT, txndate STRING, custno INT, amount DOUBLE,  
category STRING, product STRING, city STRING, state STRING, spendby STRING ) row format  
delimited Fields terminated by ',' stored as textfile;
```

```
hive> CREATE TABLE Products (Item_No INT, Item_Name STRING, Cat STRING, Cust_Price  
DOUBLE, Vendor_Price DOUBLE, Product_Category STRING) row format delimited fields  
terminated by ',' stored as textfile;
```

3. Load the data into the table

-- Local FS path

```
hive> LOAD DATA LOCAL INPATH '/home/trgadmin/lab/data/txns' OVERWRITE INTO TABLE  
txnrecords;
```

-- HDFS path

```
hive> LOAD DATA INPATH '/input/products.txt' INTO TABLE Products;
```

4. Hive queries

Describing metadata or schema of the table

```
hive> describe Products;
```

Counting no of records

```
hive> select count(*) from txnrecords;
```

Counting total spending by category of products

```
hive> select category, sum(amount) from txnrecords group by category;
```

Top 10 customers

```
hive> select custno, sum(amount) from txnrecords group by custno limit 10;
```

5. Create external table

```
hive> CREATE EXTERNAL TABLE Orders(Order_No INT, Line_No INT, Item_No INT,  
Item_Name STRING, Qty INT, Price DOUBLE) row format delimited fields terminated by ','  
location '/input/orders';
```

```
-- Local FS path
```

```
LOAD DATA LOCAL INPATH '/home/trgadmin/lab/data/orders.txt' INTO TABLE Orders;
```

6. Hive table join queries

Hive supports queries based on joining two tables.

```
hive> select p.Item_No, p.Item_Name, p.Cust_Price, o.Order_No from Products p JOIN  
Orders o on (p.Item_No = o.Item_No);
```

7. Create partitioned table

```
hive> CREATE TABLE Prod_Parts (Item_No INT, Item_Name STRING, Cust_Price DOUBLE,
Vendor_Price DOUBLE, Product_Category STRING) partitioned by (Cat STRING) clustered by
(Product_Category) into 5 buckets row format delimited fields terminated by ',' stored as
textfile;
```

8. Configure Hive to allow partitions

Enable dynamic partitions:

```
hive> set hive.exec.dynamic.partition=true;
```

```
hive> set hive.exec.dynamic.partition.mode=nonstrict;
```

```
hive> set hive.enforce.bucketing=true;
```

We are using the dynamic partition without a static partition (A table can be partitioned based on multiple columns in hive) in such case we have to enable the non-strict mode. In strict mode we can only use a Static Partition.

9. Load data into partition table

```
INSERT OVERWRITE TABLE Prod_Parts PARTITION (Cat) select Item_No, Item_Name,
Cust_Price, Vendor_Price, Product_Category, Cat from Products;
```

Note: There are 2 jobs executed here.

Verify files under HDFS to check how Hive has created multiple directories for multiple partitions.

```
$ hadoop fs -ls /user/hive/warehouse/retail.db/Prod_Parts
```

10. Then go inside each partition and check how files are created for each bucket

```
hive> show partitions Prod_Parts
```

```
$ hadoop fs -cat /user/hive/warehouse/retail.db/Prod_Parts/Cat=Cat1/000000_0
```

```
hive> select * from Prod_Parts where Cat='Cat1' or Cat='Cat3'
```

```
hive> select * from Prod_Parts where Cat='Cat1'
```

11. Create Avro backed table

While creating Avro backed tables, schema definition can be provided in the table properties or can be linked using schema file.

This example defines the schema inline.

```
hive> CREATE TABLE Products_Avro
  STORED AS AVRO
  TBLPROPERTIES ('avro.schema.literal'= '{ "name": "product_record", "type": "record",
    "fields": [
      {"name": "Item_No", "type": "int"},
      {"name": "Item_Name", "type": "string"},
      {"name": "Cust_Price", "type": "float"},
      {"name": "Vendor_Price", "type": "float"},
      {"name": "Product_Category", "type": "string"} ] }');
```

To import schema definitions from a file, use the following DDL statement. Upload schema file onto HDFS before executing the DDL statement.

```
$ hdfs dfs -put lab/data/prod_avro_schema.json /input
```

```
hive> CREATE TABLE Products_Avro2
  STORED AS AVRO
  TBLPROPERTIES
    ('avro.schema.url'='hdfs://localhost:9000/input/prod_avro_schema.json');
```

Insert records into Avro table:

```
hive> INSERT INTO products_avro SELECT Item_No, item_name, cust_price, vendor_price,
product_category from products;
```

This will create the Avro file on HDFS without .avro extension.

12. Create Parquet backed table

Create Parquet backed table by specifying STORED AS PARQUET option.

```
hive> CREATE TABLE Products_Parquet (Item_No int, Item_Name string, Cust_Price
float, Vendor_Price float, Product_Category string) STORED AS PARQUET;
```

To insert records into table:

```
hive> INSERT INTO products_parquet SELECT Item_No, item_name, cust_price,  
      vendor_price, product_category from products;
```

Lab 7 : Creating UDF in Hive

1. Create and load data into students table

In the hive shell, execute the hive commands.

```
hive> create table students(student_id int, name string, std int,  
  
    sub1 int, sub2 int, sub3 int, sub4 int, year int)  
  
    row format delimited fields terminated by ','  
  
    stored as textfile;  
  
hive> load data local inpath 'lab/data/students.txt' into table students;
```

2. Create Java project

Create a new Java project "MyUDF" and create a new Class "GetMaxMarks" in the project.

Copy the contents of lab/data/programs/GetMaxMarks.java into project file.

Add external jar hive-exec-1.2.2.jar from \$HIVE_HOME/lib folder.

Add Project Artifacts to create the jar file.

Rebuild the project and copy MyUDF.jar file from output folder to /home/trgadmin

3. Execute UDF

Add the newly created Jar file to classpath using the following command from hive prompt:

```
hive> add jar MyUDF.jar;
```

Create the temporary function for the exported jar file.

```
hive> create temporary function GetMaxMarks as 'com.hive.UDF.GetMaxMarks';
```

Call UDF in select statement.


```
hive> select student_id, name, GetMaxMarks(sub1, sub2, sub3, sub4) from students;
```

Lab 8 : Writing Pig Scripts

1. Logon to Grunt shell using the following command:

```
$ pig
```

Load Customer records

```
grunt> cust = LOAD '/input/custs' using PigStorage(',') AS ( custid:chararray,  
    firstname:chararray, lastname:chararray, age:long, profession:chararray);
```

2. Select only 100 records

```
grunt> amt = LIMIT cust 100;
```

```
grunt> dump amt;
```

This will display 100 records.

3. Group customer records by profession

```
grunt> groupbyprofession = GROUP cust BY profession;
```

```
grunt> describe groupbyprofession;
```

4. Group customer records by profession and age

```
grunt> groupbyboth = GROUP cust by (profession, age);
```

```
grunt> describe groupbyboth;
```

5. Count no of customers by profession

```
grunt> countbyprofession = FOREACH groupbyprofession GENERATE group, COUNT ( cust );
```

```
grunt> dump countbyprofession;
```

```
grunt> countbyage = FOREACH groupbyboth GENERATE group.age, COUNT ( cust );
```

6. Sort records

```
grunt> countbyage1 = order countbyage by $1 desc;
```

7. Write the contents of the relation to file.

```
grunt> store countbyage into '/input/count_age';
```

The contents of countbyage relation are stored in a file created under the specified folder.

Lab 9 : Working with Impala

Logon to Cloudera VM do perform Impala exercises.

Perform the following operations using Impala Query editor. After executing the query/statement, login to hive and verify if the changes are reflected. That will confirm Impala and Hive both use same metastore.

```
CREATE DATABASE IF NOT EXISTS retail LOCATION '/user/hive/warehouse/';
```

-- LOCATION is used to specify whether db is in user defined path or default hive path.
Either path has to be HDFS path.

The above statement is equivalent to:

```
CREATE DATABASE IF NOT EXISTS retail;
```

```
hive> show databases;
```

```
DROP DATABASE IF EXISTS retail_store LOCATION '/input/databases/';
```

```
USE retail;
```

Create table

```
CREATE TABLE IF NOT EXISTS retail.products (productID STRING, description STRING,  
qty_in_stock INT, price FLOAT);
```

Insert a record

```
insert into products (productID, description, qty_in_stock, price) values ('346663', 'Phillips  
Hand Blender', 2389, 3466.5);
```

Update a record

```
insert overwrite products values ('346663', 'Phillips Hand Blender', 2389, 3466.5);
```

Query table data

```
select productID, description, price from products;
```

```
select * from products;
```

Lab 10 : Working with HBase

a) HBase Programming

Start HBase shell by executing the following command:

```
$ hbase shell
```

In the HBase shell type the commands to execute.

1. Create, alter and describe a table

```
hbase(main):001:0> create 'emp', 'personal data', 'professional data'
```

```
hbase(main):002:0> list
```

```
hbase(main):003:0> alter 'emp', 'additional info'
```

```
hbase(main):004:0> alter 'emp', 'delete' => 'additional info'
```

```
hbase(main):005:0> scan 'emp'
```

```
hbase(main):006:0> describe 'emp'
```

2. Insert, update and delete table data and verify

```
hbase(main):007:0> put 'emp', 'E1001', 'personal data:name', 'Raju'
```

```
hbase(main):008:0> put 'emp', 'E1001', 'personal data:city', 'Hyderabad'
```

```
hbase(main):009:0> put 'emp', 'E1001', 'professional data:designation', 'Manager'
```

```
hbase(main):010:0> put 'emp', 'E1001', 'professional data:salary', '50000'
```

Look at the contents of table

```
hbase(main):011:0> scan 'emp'
```

Get one record data

```
hbase(main):012:0> get 'emp', 'E1001'
```

```
hbase(main):013:0> put 'emp', 'E1001', 'personal data:city', 'Mumbai'
```

Get the data for a specific column

```
hbase(main):014:0> get 'emp', 'E1001', {COLUMN => 'personal data:city'}
```

Notice that city column value has been updated as per latest put command.

```
hbase(main):015:0> count 'emp'
```

To delete a column, use delete statement

```
hbase(main):016:0> delete 'emp', 'E1001', professional data:designation',  
1417521848375
```

To delete the whole record, use deleteall statement

```
hbase(main):017:0> deleteall 'emp','E1001'
```

Lab 11 : Working with Spark

a) Understand how to work with RDDs.

Start the Spark shell by executing the following command.

```
$ spark-shell
```

You will get the Scala interactive shell. Lets write the code step by step and see the output.

1. Process an array of elements and calculate the sum of elements.

Declare an array of five elements

```
scala> val data=Array(1, 2, 3, 4, 5)
```

Parallelize array data and create the RDD.

```
scala> val dRDD = sc.parallelize(data)
```

Contents of dataset

```
scala> dRDD.foreach(println)
```

Modify the contents of array by incrementing each element by 1. This will result in a new RDD.

```
scala> val dRDDMap = dRDD.map(a => a + 1)
```

Contents of new dataset

```
scala> dRDDMap.foreach(println)
```

Calculate the sum of array elements by calling reduce function. This is an action and does not create an RDD.

```
scala> val mCount = dRDDMap.reduce((a,b) => (a+b))
```

2. Word Count Using Spark

```
Scala> val textFile = sc.textFile("file:///home/trgadmin/lab/data/WCData.txt")
```

Note: The source file can be in the Local File System or in HDFS.

```
Scala> val counts = textFile.flatMap(line => line.split(" ")).map(word => (word, 1)).reduceByKey(_ + _)
```

Note: The space in the split function " " splits the input line into word tokens.

```
Scala> counts.saveAsTextFile("/home/trgadmin/wordcount")
```

The file will be created in the local file system. However, you can create the file in hdfs as well by giving the distributed file system path as below:

```
Scala> counts.saveAsTextFile("hdfs://localhost:9000/wordcount")
```

Check the output file contents using:

```
$ hadoop fs -ls /wordcount
```


Appendix

1. Hadoop Configuration

All directory paths are under home directory **/home/trgadmin**

a) Untar Hadoop jar file

The downloaded tarball is stored in **/home/trgadmin/sw** location. Go to **lab/sw** and untar Hadoop files.

```
$ cd /home/trgadmin/sw
$ sudo tar xzf hadoop-2.9.0.tar.gz
```

Inspect the contents of the extracted folder.

b) Update **\$HOME/.bashrc** file.

Open nano editor for updating the **.bashrc** file.

```
$ sudo nano ~/.bashrc
```

Add the following lines to the end of the **\$HOME/.bashrc** file of current user (trgadmin), save and exit.

```
export HADOOP_HOME=/home/trgadmin/sw/hadoop-2.9.0
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
```

Run following command to install **.bashrc**

```
$ source ~/.bashrc
```

Verify whether variables are defined or not by typing **echo** at command prompt or

```
$ env
```

Repeat this step for other users who want to use Hadoop.

Confirm that you can access Hadoop and Java by executing the following:

```
$ java -version
```

```
$ hadoop version
```

You should see the respective details. If not, please check the previous steps and ensure everything is executed as mentioned.

c) **Configure Pseudo-Distributed mode**

Hadoop can also be run on a single-node in a pseudo-distributed mode where each Hadoop daemon runs in a separate Java process.

d) **Create the directories**

Create the following directories under lab/hdfs

```
$ mkdir namenodep
```

```
$ mkdir datanode1
```

```
$ mkdir checkp
```

Change permission for the following directories under lab/hdfs

```
$ chmod 775 datanode1
```

e) **Modify the configuration files**

Note: \$HADOOP_HOME is (/home/trgadmin/sw/hadoop-2.9.0)

1. Modify core-site.xml located at \$HADOOP_HOME/etc/hadoop folder.

```
$ sudo nano /home/trgadmin/sw/hadoop-2.9.0/etc/hadoop/core-site.xml
```

```
<configuration>
  <property>
    <name>fs.defaultFS</name>
    <value>hdfs://localhost:9000</value>
  </property>
</configuration>
```

2. Modify `hdfs-site.xml` located at `$HADOOP_HOME/etc/hadoop` folder.

```
<configuration>
<property>
<name>dfs.name.dir</name>
<value>file:///home/trgadmin/lab/hdfs/namenodep</value>
</property>
<property>
<name>dfs.replication</name>
<value>1</value>
</property>
<property>
<name>dfs.data.dir</name>
<value>file:///home/trgadmin/lab/hdfs/datanode1</value>
</property>
<property>
<name>dfs.checkpoint.dir</name>
<value>file:///home/trgadmin/lab/hdfs/checkp</value>
</property>
</configuration>
```

3. You can run a MapReduce job on YARN in a pseudo-distributed mode by setting a few parameters and running ResourceManager daemon and NodeManager daemon in addition.

To execute a job on YARN, configure the following. Modify `mapred-site.xml` located at `$HADOOP_HOME/etc/hadoop` folder.

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

Modify `yarn-site.xml` located at `$HADOOP_HOME/etc/hadoop` folder.

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

f) Format the Namenode

Enter the following command at prompt

```
$ hadoop namenode -format
```

Go to namenode directory and check if any folders and files have been created.

g) Start HDFS services

1. Go and edit `$HADOOP_HOME/etc/hadoop/hadoop-env.sh` file to define some parameters as follows:

```
# set to the root of your Java installation
export JAVA_HOME=/usr/lib/jvm/java-8-oracle
```

2. Start NameNode and DataNode daemons by executing the following command :

```
$ $HADOOP_HOME/sbin/start-dfs.sh
```

Run `jps` command and verify that processes are running

```
$ jps
```

3. Browse the web interface for the NameNode; by default it is available at:

```
http://localhost:50070/
```

h) Start ResourceManager daemon and NodeManager daemon:

```
$ $HADOOP_HOME/sbin/start-yarn.sh
```

Browse the web interface for the ResourceManager; by default it is available at:

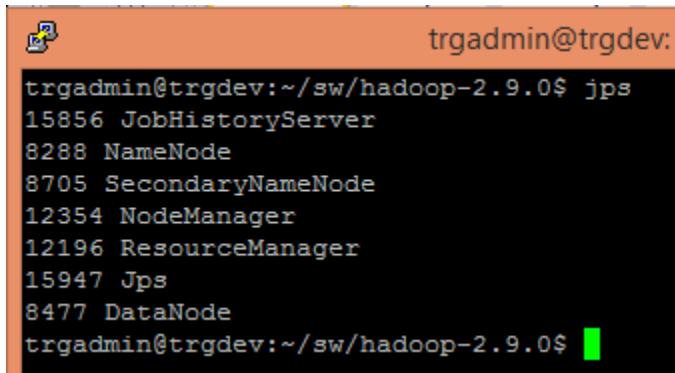
ResourceManager - <http://localhost:8088/>

i) Start JobHistoryServer daemon:

```
$ $HADOOP_HOME/sbin/mr-jobhistory-daemon.sh start historyserver
```

Run `jps` command and verify that all the processes are running.

```
$ jps
```

A terminal window with an orange title bar showing the user 'trgadmin@trgdev:'. The command 'jps' has been executed, displaying a list of running Hadoop processes: JobHistoryServer (PID 15856), NameNode (PID 8288), SecondaryNameNode (PID 8705), NodeManager (PID 12354), ResourceManager (PID 12196), Jps (PID 15947), and DataNode (PID 8477). The prompt is now 'trgadmin@trgdev:~/sw/hadoop-2.9.0\$' with a green cursor.

```
trgadmin@trgdev:~/sw/hadoop-2.9.0$ jps
15856 JobHistoryServer
8288 NameNode
8705 SecondaryNameNode
12354 NodeManager
12196 ResourceManager
15947 Jps
8477 DataNode
trgadmin@trgdev:~/sw/hadoop-2.9.0$
```

Run a MapReduce job. See the details in ResourceManager web UI.

2. Sqoop Installation

a) Extract the contents of tarball

Untar the file sqoop-1.4.7.tar.gz in lab/sw

```
$ cd $HOME/sw
$ tar -xvf sqoop-1.4.7.bin__hadoop-2.6.0.tar.gz
$ mv sqoop-1.4.7.bin__hadoop-2.6.0 sqoop-1.4.7
```

b) Configure ~/.bashrc

Set up the Sqoop environment by appending the following lines to ~/.bashrc file.

```
export SQOOP_HOME=/home/trgadmin/sw/sqoop-1.4.7
export PATH=$PATH:$SQOOP_HOME/bin
```

Use the following command to execute ~/.bashrc file.

```
$ source ~/.bashrc
```

c) Configure Sqoop

To configure Sqoop with Hadoop, you need to edit the sqoop-env.sh file, which is placed in the \$SQOOP_HOME/conf directory.

Change directory to Sqoop config directory and copy the template file using the following command

```
$ cd $SQOOP_HOME/conf
$ mv sqoop-env-template.sh sqoop-env.sh
```

Open sqoop-env.sh and edit the following lines:

```
export HADOOP_COMMON_HOME=/home/trgadmin/sw/hadoop-2.9.0
export HADOOP_MAPRED_HOME=/home/trgadmin/sw/hadoop-2.9.0
```

d) Copy mysql driver jar file

Copy mysql driver jar file from sw directory to sqoop library location /home/trgadmin/sw/sqoop-1.4.7/lib by executing the following command:

```
$ cp /home/trgadmin/sw/mysql-connector-java-5.0.8-bin.jar ../lib
```

e) Verify sqoop

Use the following command to verify the Sqoop version.

```
$ cd $SQOOP_HOME/bin
$ sqoop-version
```

3. Flume Installation

a) Flume installation

Untar the file apache-flume-1.8.0-bin.tar.gz in lab/sw

```
$ cd $HOME/sw
$ tar zxvf apache-flume-1.8.0-bin.tar.gz
$ mv apache-flume-1.8.0-bin apache-flume-1.8.0
```

b) Configure ~/.bashrc

Set up the Flume environment by appending the following lines to ~/.bashrc file.

```
export FLUME_HOME=/home/trgadmin/sw/apache-flume-1.8.0
export PATH=$PATH:$FLUME_HOME/bin
export CLASSPATH=$CLASSPATH:$FLUME_HOME/lib/*
```

Use the following command to execute ~/.bashrc file.

```
$ source ~/.bashrc
```

c) Configure Flume

To configure Flume, you need to edit flume-conf.properties and flume-env.sh files, templates of which are placed in the \$FLUME_HOME/conf directory.

Change directory to Flume config directory and copy the template file using the following command

```
$ cd $FLUME_HOME/conf
$ mv flume-conf.properties.template flume-conf.properties
$ mv flume-env.sh.template flume-env.sh
```

Open flume-env.sh and set the JAVA_HOME to the folder where Java is installed:

```
export JAVA_HOME=/usr/lib/jvm/java-8-oracle
```

Leave the default values of flume-conf.properties file. Do not modify the contents as of now.

d) Verify Flume Installation

Verify the installation of Apache Flume by executing the following command:

```
$ flume-ng
```

If it shows the command usage, then the installation is successful.

4. Hive Installation and Configuration

a) Extract the contents of tarball

Untar the file apache-hive-1.2.2-bin.tar.gz in lab/sw

```
$ cd $HOME/sw  
$ tar zxvf apache-hive-1.2.2-bin.tar.gz  
$ mv apache-hive-1.2.2-bin apache-hive-1.2.2
```

b) Configure ~/.bashrc

Set up the Hive environment by appending the following lines to ~/.bashrc file.

```
export HIVE_HOME=/home/trgadmin/sw/apache-hive-1.2.2  
export PATH=$PATH:$HIVE_HOME/bin
```

Use the following command to execute ~/.bashrc file.

```
$ source ~/.bashrc
```

c) Configure Hive

To configure Hive with Hadoop, you need to edit hive-env.sh file which is placed in the \$HIVE_HOME/conf directory.

Change directory to Hive config directory and copy the template file using the following command

```
$ cd $HIVE_HOME/conf  
$ mv hive-env.sh.template hive-env.sh
```

Open hive-env.sh and append the following line:

```
export HADOOP_HOME=/home/trgadmin/sw/hadoop-2.9.0
```

Create HDFS folders and set permissions to them.


```
$ $HADOOP_HOME/bin/hdfs dfs -mkdir /tmp
$ $HADOOP_HOME/bin/hdfs dfs -mkdir /user/hive/warehouse
$ $HADOOP_HOME/bin/hdfs dfs -chmod g+w /tmp
$ $HADOOP_HOME/bin/hdfs dfs -chmod g+w /user/hive/warehouse
```

Hive installation is completed successfully. Run the following command to check.

```
$ hive
```

d) Running HiveServer2 and Beeline

Starting from Hive 2.1, we need to run the schematool command below as an initialization step.

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

HiveServer2 has its own CLI called Beeline. HiveCLI is now deprecated in favor of Beeline, as it lacks the multi-user, security, and other capabilities of HiveServer2. To run HiveServer2 and Beeline from shell:

```
$ $HIVE_HOME/bin/hiveserver2
$ $HIVE_HOME/bin/beeline -u jdbc:hive2://localhost:10000
```

Beeline is started with the JDBC URL of the HiveServer2, which depends on the address and port where HiveServer2 was started. By default, it will be (localhost:10000), so the address will look like jdbc:hive2://localhost:10000.

5. HBase Installation

a) Extract the contents of the tarball

Untar the file apache hbase-0.98.8-hadoop2-bin.tar.gz in lab/sw

```
$ cd $HOME/sw
$ tar zxvf hbase-0.98.8-hadoop2-bin.tar.gz
$ mv hbase-0.98.8-hadoop2 hbase-0.98.8
```

b) Configure ~/.bashrc

Set up the HBase environment by appending the following lines to ~/.bashrc file.

```
export HBASE_HOME=/home/trgadmin/sw/hbase-0.98.8
export PATH=$PATH:$HBASE_HOME/bin
```

Use the following command to execute ~/.bashrc file.

```
$ source ~/.bashrc
```

c) Configure HBase

To configure HBase with Hadoop, you need to edit hbase-env.sh file which is placed in the \$HBASE_HOME/conf directory.

Open hbase-env.sh and add the following line:

```
export JAVA_HOME=/usr/lib/jvm/java-8-oracle
```

Edit hbase-site.xml file and add the following lines in between <configuration> and </configuration> tags:

```
<property>
<name>hbase.cluster.distributed</name>
<value>true</value>
</property>
//Here you have to set the path where you want HBase to store its files.
<property>
<name>hbase.rootdir</name>
<value>hdfs://localhost:9000/hbase</value>
</property>

//Here you have to set the path where you want HBase to store its built in zookeeper
files.
<property>
<name>hbase.zookeeper.property.dataDir</name>
<value>home/trgadmin/zookeeper</value>
</property>
```

Create the hdfs folder for hbase to write contents.

```
$ hadoop fs -mkdir /hbase
```

HBase installation is completed successfully. Run the following command to check.

```
$ start-hbase.sh
```

Checking the HBase Directory in HDFS

HBase creates its directory in HDFS. To see the created directory, browse to Hadoop bin and type the following command.

```
$ hadoop fs -ls /hbase
```

6. Spark Installation

a) Spark installation

Untar the file spark-2.4.0-bin-hadoop2.7.tgz in lab/sw

```
$ cd $HOME/sw
```

```
$ tar zxvf spark-2.4.0-bin-hadoop2.7.tgz
```

```
$ mv spark-2.4.0-bin-hadoop2.7 spark-2.4.0
```

b) Configure ~/.bashrc

Set up the Spark environment by appending the following lines to ~/.bashrc file.

```
export SPARK_HOME=/home/trgadmin/sw/spark-2.4.0
```

```
export PATH=$PATH:$SPARK_HOME/bin
```

Use the following command to execute ~/.bashrc file.

```
$ source ~/.bashrc
```

Spark configuration is done.