Setup HDFS and HBase cluster

1. Transfer hbase-0.98.2-hadoop2-bin.tar from your desktop to VM

Transfer to /home/hadoop/lab/downloads folder of VM

2. Untar hadoop 2.3.0 & Configure HDFS

3. Untar & configure hbase 0.98

Go to /home/hadoop/lab/software

tar -xvf /home/hadoop/lab/downloads/hbase-0.98.2-hadoop2-bin.tar.gz

Add the following line to the .bash_profile at /home/hadoop

export HBASE_HOME=/home/hadoop/lab/software/hbase-0.98.2-hadoop2

export PATH=$PATH:$PIG_INSTALL/bin:$OOZIE_HOME/bin:$HBASE_HOME/bin

run .bash_profile to set the environment variables

. *<space>* .bash_profile

Verify hadoop and hbase versions

hadoop version
hbase version

Step 2: Configure HDFS

Step 3: Configure HBase

- Go to $ HBASE_HOME/conf directory
- Edit the following files

$HBASE_HOME/conf/hbase-site.xml

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
<property>
<name>hbase.rootdir</name>
<value>hdfs://hadooplab.bigdataleap.com:8020/hbase</value>
<description>The directory shared by RegionServers.</description>
</property>
<property>
<name>hbase.cluster.distributed</name>
<value>true</value>
```

```
</property>
<property>
<name>hbase.zookeeper.quorum</name>
<value>hadooplab.bigdataleap.com</value>
</property>
<property>
<name>hbase.tmp.dir</name>
<value>/home/hadoop/lab/cluster/hbase/tmp</value>
</property>
<property>
<name>hbase.local.dir</name>
<value>/home/hadoop/lab/cluster/hbase/local</value>
</property>
<property>
<name>hbase.zookeeper.property.dataDir</name>
<value>/home/hadoop/lab/cluster/zookeeper</value>
</property>
<property>
<name>hbase.zookeeper.property.clientPort</name>
<value>2181</value>
</property>
</configuration>
```

Step 4: Enter the region server node

Enter '*hadooplab.bigdataleap.com*' in the $HBASE_HOME/conf/regionservers

Step 5: Create required Directories for hdfs and hbase

```
## Create directories configured in hbase-site.xml
 cd  /home/hadoop/lab/cluster

mkdir local
mkdir tmp
mkdir zookeeper
```

Step 6: Configure $HADOOP_INSTALL/conf/hadoop-env.sh and hbase-env.sh files

- ***Setup JAVA_HOME and Zookeeper settings in hbase-env.sh file***

```
export JAVA_HOME=/usr/lib/jvm/jre-1.7.0-openjdk.x86_64

export HBASE_MANAGES_ZK=true
```

- ***Link hdfs-site.xml file from hbase conf directory to hadoop conf directory***

```
ln  -s  $HADOOP_CONF_DIR/hdfs-site.xml          $HBASE_HOME/conf/hdfs-site.xml
```

Step 8: Start hbaseservice

cd $HBASE_HOME/bin

./start-hbase.sh

```
notroot@ubuntu:~/lab/software/hbase-0.94.7/bin$ jps
3868 NameNode
4359 SecondaryNameNode
5526 HRegionServer
5302 HMaster
5726 Jps
5241 HQuorumPeer
4110 DataNode
notroot@ubuntu:~/lab/software/hbase-0.94.7/bin$
```

Note: (This is only for reference)

The hbase master node and region server nodes can be stopped or restarted individually using the following commands

hbase-daemon.sh start|stop master

hbase-daemon.sh start|stop regionserver

Step 7: HBase Master node UI is running at

https://hnode:60010/

Using HBase shell

- ***Enter hbase shell***
  linux prompt> hbase shell

- ***List all tables***

  list

- ***Create a table with a column family called personalinfo***

  create 'custs', 'personalinfo'

- ***Populate table***

  put 'custs', '001', 'personalinfo:name', 'manaranjan'

  put 'custs', '002', 'personalinfo:name', 'chandan'

  put 'custs', '003', 'personalinfo:name', 'prashant'

  put 'custs', '004', 'personalinfo:name', 'debasis'

put 'custs', '001', 'personalinfo:age', '35'

put 'custs', '002', 'personalinfo:age', '30'

- ***Count no of records***

  count 'custs'

- ***Update statement***

  put 'custs', '001', 'personalinfo:age', '37'

  put 'custs', '001', 'personalinfo:name', 'manaranjan pradhan'

- ***Get records for a particular rowkey***

  get 'custs', '001' // all columns for all column families

  get 'custs', '001', 'personalinfo' // all columns for a particular column family

  get 'custs', '001', 'personalinfo:age' // a particular column

- ***Add a new column family to an existing table***

  disable 'custs'

  alter 'custs', 'courses'

  enable 'custs'

- ***Add data to the new column family***

  put 'custs', '001', 'courses:TotalScore', '450'

  put 'custs', '002', 'courses:TotalScore', '350'

  put 'custs', '003', 'courses:TotalScore', '250'

  put 'custs', '004', 'courses:TotalScore', '550'

- ***Filtering records by row keys***

  scan 'custs' // will list all the rows

  scan 'custs', {STARTROW => '002'} // lists all rows starting with a specific row key

  scan 'custs', {LIMIT => 2, STARTROW => '001'} // lists all rows starting with a row key and limiting number of rows selected

  scan 'custs', {COLUMNS => ['personalinfo:firstname', 'personalinfo:age'] } // lists only specific column family and column qualifier

- ***Filtering records by based on column values***

scan 'custs', { COLUMNS => 'courses:TotalScore', LIMIT => 10, FILTER => "ValueFilter(>, 'binary:200')" }

scan 'custs', { COLUMNS => 'courses:TotalScore', LIMIT => 10, FILTER => "ValueFilter(=, 'binary:450')" }

Using Java APIs

1. ***Initializing connections to HBase***

```
Configuration conf = HBaseConfiguration.create();
conf.set("hbase.master","192.168.217.135:60010");
conf.set( "hbase.zookeeper.quorum", "192.168.217.135");

connection = HConnectionManager.createConnection(conf);
hbaseTable = connection.getTable("custs");
```

2. ***Inserting a record – Populating a column family***

```
Put put = new Put( fields[0].getBytes() );
put.add( "personalinfo".getBytes(), "firstname".getBytes(), fields[1].getBytes() );
put.add( "personalinfo".getBytes(), "lastname".getBytes(), fields[2].getBytes() );
put.add( "personalinfo".getBytes(), "age".getBytes(), fields[3].getBytes() );
put.add( "personalinfo".getBytes(), "profession".getBytes(), fields[4].getBytes() );

hbaseTable.put( put );
```

3. ***Reading a record and it's column family***

```
// Reading records from the table based on row-key
public void getCustomerInfo( String id ) throws IOException
{
    Get g = new Get(Bytes.toBytes( id ));
    Result r = hbaseTable.get(g);

    g.addFamily( "personalinfo".getBytes() );

    System.out.println( " Firstname : " + getColumnValue( r, "personalinfo", "firstname") );
    System.out.println( " Lastname : " + getColumnValue( r, "personalinfo", "lastname") );
    System.out.println( " Age : " + getColumnValue( r, "personalinfo", "age")  );
    System.out.println( " Profession : " + getColumnValue( r, "personalinfo", "profession") );

}

private String getColumnValue( Result r, String family, String qual )
{

    byte[] val = r.getValue( Bytes.toBytes( family )
                                , Bytes.toBytes( qual ) );

    if( val != null )
        return new String( val );
    else
        return "";
}
```

4. *Updating an existing Record*

```
// Update a record
public void updateAge( String id, String age ) throws IOException
{
    Put put = new Put( id.getBytes() );
    put.add( "personalinfo".getBytes(), "age".getBytes(), age.getBytes() );

    hbaseTable.put( put );
}
```

5. *Reading all records*

```
Scan s = new Scan();
ResultScanner rs = hbaseTable.getScanner(s);

for ( Result r: rs )
{

}
```

6. *Deleting Rows or specific columns or the column family*

```
Delete delete = new Delete( id.getBytes() );

hbaseTable.delete( delete );

Delete delete = new Delete( id.getBytes() );

delete.deleteColumn( "personalinfo".getBytes(), "age".getBytes() );

delete.deleteFamily( "personalinfo".getBytes() );

hbaseTable.delete( delete );
```

7. *Filtering records based on a column qualifier*

```
Scan s = new Scan( startid.getBytes(), endid.getBytes() );
ResultScanner rs = hbaseTable.getScanner(s);

SingleColumnValueFilter filter = new SingleColumnValueFilter(
        Bytes.toBytes("personalinfo"),
        Bytes.toBytes("age"),
        CompareOp.GREATER_OR_EQUAL,
        new BinaryComparator(Bytes.toBytes( age )));

Scan s = new Scan();
s.setFilter( filter );
ResultScanner rs = hbaseTable.getScanner(s);
```

8. *Filtering records based on row keys*

```
Filter filter1 = new RowFilter(CompareOp.LESS_OR_EQUAL,
        new BinaryComparator(Bytes.toBytes("4000100")));

Filter filter2 = new RowFilter(CompareOp.EQUAL,
        new RegexStringComparator("*10"));
```

9. *Advanced filter queries using java APIs*

```
List<Filter> filters = new ArrayList<Filter>(2);

SingleColumnValueFilter filter1 = new SingleColumnValueFilter(
        Bytes.toBytes("personalinfo"),
        Bytes.toBytes("age"),
        CompareOp.GREATER_OR_EQUAL,
        new BinaryComparator(Bytes.toBytes( age )));

SingleColumnValueFilter filter2 = new SingleColumnValueFilter(
        Bytes.toBytes("personalinfo"),
        Bytes.toBytes("profession"),
        CompareOp.EQUAL,
        new RegexStringComparator( profession ));

filters.add( filter1 );
filters.add( filter2 );


FilterList filterList = new FilterList(FilterList.Operator.MUST_PASS_ALL, filters);

Scan s = new Scan();
s.setFilter( filterList );
ResultScanner rs = hbaseTable.getScanner(s);
```

10. *Using REST Protocol*

Start the REST Gateway

./hbase-daemon.sh start rest -p 7070

Using rest protocols

curl -H "Accept: application/json" http://hadooplab.bigdataleap.com:7070/version

curl -H "Accept: application/json"
http://hadooplab.bigdataleap.com:7070/custs/4000010/personalinfo

The response values are base64 encrypted. To decrypt and see the actual values use the following commands
echo <value> | base64 -d
echo cGVyc29uYWxpbmZvOmxhc3RuYW1l | base64 -d

Using Java APIs to access REST Gateway

```
// Connecting to REST Service gateway
Cluster cluster = new Cluster();
cluster.add("hadooplab.bigdataleap.com", 7070);

Client client = new Client(cluster);
RemoteHTable table = new RemoteHTable(client, "custs");

// Getting a specific record
Get get = new Get(Bytes.toBytes("4000010"));

Result r = table.get(get);
System.out.println( " Firstname : " + getColumnValue( r, "personalinfo", "firstname") );
System.out.println( " Lastname : " + getColumnValue( r, "personalinfo", "lastname") );
System.out.println( " Age : " + getColumnValue( r, "personalinfo", "age")  );
System.out.println( " Profession : " + getColumnValue( r, "personalinfo", "profession") );

// Scanning a table for getting a range of records
Scan scan = new Scan();
scan.setStartRow(Bytes.toBytes("4000100"));
scan.setStopRow(Bytes.toBytes("4000120"));

ResultScanner scanner = table.getScanner(scan);
for (Result rscan : scanner) {
    System.out.println( " Firstname : " + getColumnValue( rscan, "personalinfo", "firstname") );
    System.out.println( " Lastname : " + getColumnValue( rscan, "personalinfo", "lastname") );
    System.out.println( " Age : " + getColumnValue( rscan, "personalinfo", "age")  );
    System.out.println( " Profession : " + getColumnValue( rscan, "personalinfo", "profession") );
}

table.close();
```

Stopping REST Gateway

./hbase-daemon.sh stop rest

Using Hive for running complex Queries

- ***Install and Configure Hive***
- ***Add the following line to the .bash_profile***
  export HADOOP_CLASSPATH=$HADOOP_CLASSPATH:$HBASE_HOME/lib:$HBASE_HOME/conf

  and run the .bash_profile
  . <space> .bash_profile
- ***Add the following line to the .bash_profile***
  hive --auxpath $HIVE_HOME/lib/hive-hbase-handler-0.13.0.jar,$HBASE_HOME/lib/
  hbase.zookeeper.quorum=hadooplab.bigdataleap.com
- ***Create HIVE Table pointing to HBase Table***
  Use retail;
  CREATE EXTERNAL TABLE custshbase(custid int, firstname string, lastname string, age int, profession string) STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler' WITH SERDEPROPERTIES ("hbase.columns.mapping" = ":key,personalinfo:firstname,personalinfo:lastname,personalinfo:age,personalinfo:profession") TBLPROPERTIES("hbase.table.name" = "custs");
- ***Run the SQL Query***
  select profession, count(*) from custshbase group by profession;