# Setting up Hadoop, Zookeeper, HBase, Tomcat and HSearch

This has been tested using fedora v1.08 X86 machine. The machine used was from Amazon EC2 public image *fedora-8-x86\_64-base-v1.08.manifest.xml - ami-2547a34c*

#### HSearch 0.94 is tested and supports Cloudera CDH Version 3 Update 3.

#### You can install Hadoop, HBase and Zookeeper from Cloudera CDH Version 3 Update 3.

<https://ccp.cloudera.com/display/SUPPORT/CDH3+Downloadable+Tarballs>

* Hadoop [Installation Documentation](https://ccp.cloudera.com/display/CDHDOC/CDH3+Deployment+on+a+Cluster)
* Zookeeper [Installation Document](https://ccp.cloudera.com/display/CDHDOC/ZooKeeper+Installation)
* HBase [Installation Document](https://ccp.cloudera.com/display/CDHDOC/HBase+Installation)

Do it manually following below instructions.

# Generic Machine Configuration

1. Access all servers of the cluster using name. You can set this in your DNA mapping or simply editing the /etc/hosts file

a.b.c.k master

a.b.c.l slave1

a.b.c.m slave1

a.b.c.n slave1

a.b.c.o slave1

1. Allow all these servers to communicate in the machine firewall.

$ iptables -A INPUT -s 0.0.0.0 -j ACCEPT

$ iptables -A INPUT -s master -j ACCEPT

$ iptables -A INPUT -s slave1 -j ACCEPT

$ iptables -A INPUT -s slave2 -j ACCEPT

$ iptables -A INPUT -s slave3 -j ACCEPT

Setup a working shell. You may need to open many consoles and copy the setup between machines. A typical copy command is scp root@master1:syncfolder/\* .

export PS1="[master \$PWD]# "

set -o vi

1. Setup auto login

$ ssh-keygen -t rsa

Generating public/private rsa key pair.

Enter file in which to save the key (~/.ssh/id\_rsa):

Enter passphrase (empty for no passphrase):

Enter same passphrase again:

Your identification has been saved in ~/.ssh/id\_rsa

Your public key has been saved in ~/.ssh/id\_rsa.pub

$ cd ~/.ssh

$ cat id\_rsa.pub >> authorized\_keys

$ chmod 600 authorized\_keys

$ cd ~

$ ssh master

$ exit

$ ssh slave1

$ copy the keys and make an entry in authorized\_keys.

$ exit

1. Setup JRE

$ mkdir /usr/java ; cd /usr/java

$ wget http://download.oracle.com/otn-pub/java/jdk/6u31-b04/jre-6u31-linux-i586-rpm.bin

$ chmod 700 jre-6u31-linux-i586-rpm.bin

$ jre-6u31-linux-i586-rpm.bin

$ You will see a folder /usr/java/jre1.6.0\_31

$ Test java running /usr/java/jre1.6.0\_31/bin/java

# Setup Hadoop

$ wget <http://archive.cloudera.com/cdh/3/hadoop-0.20.2-cdh3u3.tar.gz>

$ mv hadoop-0.20.2-cdh3u3.tar.gz hadoop-0.20.2.tar.gz

$ gzip -d hadoop-0.20.2.tar.gz

$ tar -xf hadoop-0.20.2.tar

$ mv hadoop-0.20.2.tar hadoop-0.20.2

$ mv hadoop-0.20.2 hadoop

$ cd hadoop

$ rm -rf hadoop-0.20.2-examples.jar hadoop-0.20.2-test.jar hadoop-0.20.2-ant.jar c++ contrib docs ivy ivy.xml src

$ ls

$ cd conf

$ echo "" > excludes

$ mkdir -p /mnt/data/namenode /mnt/data/datanode /mnt/logs /mnt/data/namenode/dfsname

Export variables to the end of the hadoop configuration file.

$ echo "export JAVA\_HOME=/usr/java/jre1.6.0\_31" >> hadoop-env.sh

$ echo "export HADOOP\_HEAPSIZE=2048" >> hadoop-env.sh

$ echo "export HADOOP\_OPTS=\"-server -XX:+UseParallelGC -XX:ParallelGCThreads=4 -XX:+AggressiveHeap -XX:+HeapDumpOnOutOfMemoryError\"" >> hadoop-env.sh

$ echo "export HADOOP\_LOG\_DIR=/mnt/logs" >> hadoop-env.sh

$ echo "master" > masters; cat masters

$ echo "slave1" > slaves; cat slaves

$ echo "slave2" > slaves; cat slaves

$ …. Enter all slaves in a new line.

$ Change the following line of log4j.properties file.

*hadoop.log.dir=/mnt/logs*

$ Edit the core-site.xml

<?xml version="1.0"?>

<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<!-- Put site-specific property overrides in this file. -->

<configuration>

<property>

<name>fs.default.name</name>

<value>hdfs://master:54310</value>

<description>The name of the default file system. A URI whose scheme and authority determine the FileSystem implementation. The uri's scheme determines the config property (fs.SCHEME.impl) naming the FileSystem implementation class. The uri's authority is used to determine the host, port, etc. for a filesystem.

</description>

</property>

<property>

<name>io.file.buffer.size</name>

<value>16384</value>

<description>

Read/write buffer size used in SequenceFiles (should be in multiples of the hardware page size All Intel processors have 4KiB pages. Means 4096 In a one machine cluster, I am making this as small as possible continuous streaming. In KB 65536 Means 64KB. 16KB starting 16384. Typical value for a 250 to 2000 nodes is 32768-131072

</description>

</property>

<property>

<name>io.seqfile.compress.blocksize</name>

<value>4096</value>

<description>

The minimum block size for compression in block compressed SequenceFiles We will compress a minimum 4KB file, as this hels us to read less.

</description>

</property>

<property>

<name>io.compression.codecs</name>

<value>org.apache.hadoop.io.compress.GzipCodec,org.apache.hadoop.io.compress.DefaultCodec,com.hadoop.compression.lzo.LzoCodec,com.hadoop.compression.lzo.LzopCodec,org.apache.hadoop.io.compress.BZip2Codec

</value>

</property>

<property>

<name>io.compression.codec.lzo.class</name>

<value>com.hadoop.compression.lzo.LzoCodec</value>

</property>

</configuration>

Edit the Hdfs-site.xml

<?xml version="1.0"?>

<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<!-- Put site-specific property overrides in this file. -->

<configuration>

<property>

<name>hadoop.home</name>

<value>/mnt/hsearch/hadoop</value>

</property>

<property>

<name>metadata.dir</name>

<value>/mnt/data/namenode</value>

<description>Where the NameNode metadata should be stored</description>

</property>

<property>

<name>dfs.data.dir</name>

<value>/mnt/data/datanode</value>

<description>At which location the data is stored. /data/1,/data/2,/data/3 </description>

</property>

<property>

<name>dfs.replication</name>

<value>1</value>

<description>

Clusters closer to the 8-10 node range may want to set dfs.replication to 3. Values higher than 3 are usually not necessary. Individual files which are heavily utilized by a large number of nodes may have their particular replication factor manually adjusted upward independent of the cluster default

</description>

</property>

<property>

<name>dfs.block.size</name>

<value>33554432</value>

<description>The NameNode is responsible for managing metadata associated with each block in the HDFS. As the amount of information in the rack scales into the 10's or 100's of TB, this can grow to be quite sizable. The NameNode machine needs to keep the blockmap in RAM to work efficiently. Therefore, at large scale, this machine will require more RAM than other machines in the cluster. The amount of metadata can also be dropped almost in half by doubling the block size: This changes the block size from 64MB (the default) to 128MB, which decreases pressure on the NameNode's memory. On the other hand, this potentially decreases the amount of parallelism that can be achieved, as the number of blocks per file decreases. This means fewer hosts may have sections of a file to offer to MapReduce tasks without contending for disk access. The larger the individual files involved (or the more files involved in the average MapReduce job), the less of an issue this is.

</description>

</property>

<property>

<name>dfs.namenode.handler.count</name>

<value>40</value>

<description>With multiple racks of servers, RPC timeouts may become more frequent. The NameNode takes a continual census of DataNodes and their health via heartbeat messages sent every few seconds. A similar timeout mechanism exists on the MapReduce side with the JobTracker. With many racks of machines, they may force one another to timeout because the master node is not handling them fast enough. The following options increase the number of threads on the master machine dedicated to handling RPC's from slave nodes:

</description>

</property>

<property>

<name>dfs.datanode.handler.count</name>

<value>40</value>

<description></description>

</property>

<property>

<name>dfs.datanode.du.reserved</name>

<value>1073741824</value>

<description>By default, HDFS does not preserve any free space on the DataNodes; the DataNode service will continue to accept blocks until all free space on the disk is exhausted, which may cause problems. The following setting will require each DataNode to reserve at least 1 GB of space on the drive free before it writes more blocks, which helps preserve system stability

</description>

</property>

<property>

<name>dfs.datanode.max.xcievers</name>

<value>4096</value>

<description>Concurrent request exception</description>

</property>

<property>

<name>dfs.hosts.exclude</name>

<value>/mnt/hadoop/conf/excludes</value>

<description>This property should provide the full path to the excludes file (the actual location of the file is up to you). You should then create an empty file with this name

</description>

</property>

<property>

<name>dfs.name.dir</name>

<value>${metadata.dir}/dfsname</value>

<description>This is the path on the local file system of the NameNode instance where the NameNode metadata is stored. It is only used by the NameNode instance to find its information, and does not exist on the DataNodes. If this is a comma-delimited list of directories then the name table is replicated in all of the directories, for redundancy.

</description>

</property>

<property>

<name>dfs.name.edits.dir</name>

<value>${metadata.dir}/dfsnameedit</value>

<description></description>

</property>

</configuration>

After configuring all log files, format the namenode.

$ cd /mnt/hadoop/bin/

**$ ./hadoop --config** /mnt/hadoop/conf **namenode -format**

1. Start Hadoop
   1. $ cd /mnt/hadoop/bin/
   2. $ ulimit -n 16384
   3. $ ./hadoop-daemon.sh --config /mnt/hadoop/conf/ start namenode; tail -f /mnt/logs/hadoop-root-namenode-\*.log

Sucessful means it should read at the end line *IPC Server handler 39 on 54310: starting*

* 1. $ ./hadoop-daemon.sh --config /mnt/hadoop/conf/ start datanode; tail -f /mnt/logs/hadoop-root-datanode-\*.log

1. Test Hadoop
   1. $ cd /mnt/hadoop/bin/
   2. $ ./hadoop dfs -du /
   3. Upload and Download a File

$ cat > /tmp/a.txt

* 1. $ ./hadoop dfs -copyFromLocal /tmp/a.txt /
  2. $ ./hadoop dfs -ls /
  3. $ ./hadoop dfs -cat /a.txt
  4. $ ./hadoop dfs -rm /a.txt
  5. $ ./hadoop dfs -ls /

1. Stop Hadoop
   1. $ cd /mnt/hadoop/bin/
   2. $ ./hadoop-daemon.sh --config /mnt/hadoop/conf/ stop datanode; tail -f /mnt/logs/hadoop-root-datanode-\*.log
   3. $ ./hadoop-daemon.sh --config /mnt/hadoop/conf/ stop namenode; tail -f /mnt/logs/hadoop-root-namenode-\*.log

# Setup Zookeeper

$ cd /mnt

$ wget <http://archive.cloudera.com/cdh/3/zookeeper-3.3.4-cdh3u3.tar.gz>

$ mv zookeeper-3.3.4-cdh3u3.tar.gz zookeeper-3.3.4.tar.gz

$ gzip -d zookeeper-3.3.4.tar.gz

$ tar -xf zookeeper-3.3.4.tar

$ mv zookeeper-3.3.4.tar zookeeper-3.3.4

$ mv zookeeper-3.3.4 zookeeper

$ cd zookeeper

$ rm -rf docs ivy.xml ivysettings.xml src contrib dist-maven; ls

$ cd conf

$ cp zoo\_sample.cfg zoo.cfg

$ vi zoo.cfg

$ modify the line “dataDir=/mnt/data/zoodata”

Zoo Cluster setup can be done by adding following lines where zookeeper is running in master, slave1 and slave2

server.1=master:2888:3888

server.2=slave1:2888:3888

server.3=slave2:2888:3888

$ echo "export ZOO\_LOG\_DIR=/mnt/logs" >> /mnt/zookeeper/bin/zkEnv.sh

$ mkdir /mnt/data/zoodata

1. Start Zookeeper
   1. $ cd /mnt/zookeeper/bin
   2. $ export JAVA\_HOME=/usr/lib/jdk
   3. $ export PATH=$PATH:$JAVA\_HOME/bin
   4. $ /mnt/zookeeper/bin/zkServer.sh start > /mnt/logs/zoo.cfg ; tail -f /mnt/logs/zoo.cfg
2. Test Zookeeper
   1. $ ps -ef | grep zoo
3. Stop Zookeeper
   1. $ /mnt/zookeeper/bin/zkServer.sh stop
   2. $ ps -ef | grep zoo

# Setup HBase

$ wget <http://archive.cloudera.com/cdh/3/hbase-0.90.4-cdh3u3.tar.gz>

$ mv hbase-0.90.4-cdh3u3.tar.gz hbase-0.90.4.tar.gz

$ gzip -d hbase-0.90.4.tar.gz

$ tar -xf hbase-0.90.4.tar

$ mv hbase-0.90.4 hbase

$ mv hbase-0.90.4.tar hbase

$ cd hbase

$ rm -rf docs hbase-0.90.4-tests.jar pom.xml src

Copy the filters to lib folder.

$ cd lib

$ wget http://localhost /xxxFilter.jar

$ cd ../conf

$ echo "slave1" > regionservers; cat regionservers

log4j.properties file edit.

hbase.log.dir=/mnt/logs

Export variables to the end of the hbase configuration file.

$ echo "export JAVA\_HOME=/usr/lib/jdk" >> hbase-env.sh

$ echo "export HBASE\_HEAPSIZE=2048" >> hbase-env.sh

$ echo "export HBASE\_OPTS=\"-server -XX:+UseParallelGC -XX:ParallelGCThreads=4 -XX:+AggressiveHeap -XX:+HeapDumpOnOutOfMemoryError\"" >> hbase-env.sh

$ echo "export HBASE\_LOG\_DIR=/mnt/logs" >> hbase-env.sh

Setup following properties in hbase-site.xml file

*<property>*

*<name>hbase.cluster.distributed</name>*

*<value>true</value>*

*</property>*

*<property>*

*<name>hbase.rootdir</name>*

*<value>hdfs://master:54310/hbase</value>*

*</property>*

*<property>*

*<name>hbase.zookeeper.quorum</name>*

*<value>master~~,slave1,slave2~~</value>*

*</property>*

1. Start HBase
   1. $ cd /mnt/hbase/bin/
   2. $ ./hbase-daemon.sh --config /mnt/hbase/conf start master; tail -f /mnt/logs/hbase-root-master\*.log
   3. $ ./hbase-daemons.sh --config /mnt/hbase/conf start regionserver; tail -f /mnt/logs/hbase-root-regionserver\*.log
2. Test HBase
   1. $ ./hbase shell

list

1. Stop HBase
   1. $ cd /mnt/hbase/bin/
   2. $ ./hbase-daemons.sh --config /mnt/hbase/conf stop regionserver
   3. $ ./hbase-daemon.sh --config /mnt/hbase/conf stop master