Chapter 1. Getting Started

Table of Contents

1.1. Introduction

Section 1.2, "Quick Start - Standalone HBase" will get you up and running on a single-node, standalone instance of HBase.

1.2. Quick Start - Standalone HBase

This guide describes setup of a standalone HBase instance running against the local filesystem. This is not an appropriate configuration for a production instance of HBase, but will allow you to experiment with HBase. This section shows you how to create a table in HBase using the **hbase shell** CLI, insert rows into the table, perform put and scan operations against the table, enable or disable the table, and start and stop HBase. Apart from downloading HBase, this procedure should take less than 10 minutes.

Local Filesystem and Durability

Using HBase with a local filesystem does not guarantee durability. The HDFS local filesystem implementation will lose edits if files are not properly closed. This is very likely to happen when you are experimenting with new software, starting and stopping the daemons often and not always cleanly. You need to run HBase on HDFS to ensure all writes are preserved. Running against the local filesystem is intended as a shortcut to get you familiar with how the general system works, as the very first phase of evaluation. Seehttps://issues.apache.org/jira/browse/HBASE-3696 and its associated issues for more details about the issues of running on the local filesystem.

Loopback IP - HBase 0.94.x and earlier

The below advice is for hbase-0.94.x and older versions only. This is fixed in hbase-0.96.0 and beyond.

Prior to HBase 0.94.x, HBase expected the loopback IP address to be 127.0.0.1. Ubuntu and some other distributions default to 127.0.1.1 and this will cause problems for you . See Why does HBase care about /etc/hosts? for detail.

Example 1.1. Example /etc/hosts File for Ubuntu

The following `/etc/hosts` file works correctly for HBase 0.94.x and earlier, on Ubuntu. Use this as a template if you run into trouble.

```
127.0.0.1 localhost
127.0.0.1 ubuntu.ubuntu-domain ubuntu
```

1.2.1. JDK Version Requirements

HBase requires that a JDK be installed. See Section 2.1.1, "Java" for information about supported JDK versions.

1.2.2. Get Started with HBase

Procedure 1.1. Download, Configure, and Start HBase

1. Choose a download site from this list of Apache Download Mirrors. Click on the suggested top link. This will take you to a mirror of *HBase Releases*. Click on the folder named `stable` and then download the binary file that ends in `.tar.gz` to your local filesystem. Be sure to choose the version that corresponds with the version of Hadoop you are likely to use later. In most cases, you should choose the file for Hadoop 2, which will be called something like `hbase-0.98.3-hadoop2-bin.tar.gz`. Do not download the file ending in `src.tar.gz` for now.

2. Extract the downloaded file, and change to the newly-created directory.

   ```
   $ tar xzvf hbase-<?eval ${project.version}?>-hadoop2-bin.tar.gz
   $ cd hbase-<?eval ${project.version}?>-hadoop2/
   ```

3. Edit `conf/hbase-site.xml`, which is the main HBase configuration file. At this time, you only need to specify the directory on the local filesystem where HBase and Zookeeper write data. By default, a new directory is created under /tmp. Many servers are configured to delete the contents of /tmp upon reboot, so you should store the data elsewhere. The following configuration will store HBase's data in the `hbase` directory, in the home directory of the user called `testuser`. Paste the <property> tags beneath the<configuration> tags, which should be empty in a new HBase install.

   Example 1.2. Example `hbase-site.xml` for Standalone HBase

   ```
   <configuration>
     <property>
       <name>hbase.rootdir</name>
       <value>file:///home/testuser/hbase</value>
     </property>
     <property>
       <name>hbase.zookeeper.property.dataDir</name>
       <value>/home/testuser/zookeeper</value>
     </property>
   </configuration>
   ```

You do not need to create the HBase data directory. HBase will do this for you. If you create the directory, HBase will attempt to do a migration, which is not what you want.

4. The `bin/start-hbase.sh` script is provided as a convenient way to start HBase. Issue the command, and if all goes well, a message is logged to standard output showing that HBase started successfully. You can use the**jps** command to verify that you have one running process called `HMaster`and at least one called `HRegionServer`.

   Note

   Java needs to be installed and available. If you get an error indicating that Java is not installed, but it is on your system, perhaps in a non-standard location, edit the `conf/hbase-env.sh` file and modify the `JAVA_HOME` setting to point to the directory that contains `bin/java` your system.

Procedure 1.2. Use HBase For the First Time

1. **Connect to HBase.**

   Connect to your running instance of HBase using the **hbase shell** command, located in the `bin/` directory of your HBase install. In this example, some usage and version information that is printed when you start HBase Shell has been omitted. The HBase Shell prompt ends with a `>` character.

   ```
   $ ./bin/hbase shell
   hbase(main):001:0>
   ```

2. **Display HBase Shell Help Text.**

   Type `help` and press Enter, to display some basic usage information for HBase Shell, as well as several example commands. Notice that table names, rows, columns all must be enclosed in quote characters.

3. **Create a table.**

   Use the `create` command to create a new table. You must specify the table name and the ColumnFamily name.

   ```
   hbase> create 'test', 'cf'
   0 row(s) in 1.2200 seconds
   ```

4. **List Information About your Table**

   Use the `list` command to

   ```
   hbase> list 'test'
   TABLE
   test
   1 row(s) in 0.0350 seconds

   => ["test"]
   ```

5. **Put data into your table.**

   To put data into your table, use the `put` command.

   ```
   hbase> put 'test', 'row1', 'cf:a', 'value1'
   0 row(s) in 0.1770 seconds

   hbase> put 'test', 'row2', 'cf:b', 'value2'
   0 row(s) in 0.0160 seconds

   hbase> put 'test', 'row3', 'cf:c', 'value3'
   0 row(s) in 0.0260 seconds
   ```

   Here, we insert three values, one at a time. The first insert is at `row1`, column`cf:a`, with a value of `value1`. Columns in HBase are comprised of a column family prefix, `cf` in this example, followed by a colon and then a column qualifier suffix, `a` in this case.

6. **Scan the table for all data at once.**

   One of the ways to get data from HBase is to scan. Use the **scan** command to scan the table for data. You can limit your scan, but for now, all data is fetched.

   ```
   hbase> scan 'test'
   ROW                   COLUMN+CELL
    row1                 column=cf:a, timestamp=1403759475114, value=value1
    row2                 column=cf:b, timestamp=1403759492807, value=value2
    row3                 column=cf:c, timestamp=1403759503155, value=value3
   3 row(s) in 0.0440 seconds
   ```

7. **Get a single row of data.**

   To get a single row of data at a time, use the **get** command.

   ```
   hbase> get 'test', 'row1'
   COLUMN                CELL
    cf:a                 timestamp=1403759475114, value=value1
   1 row(s) in 0.0230 seconds
   ```

8. **Disable a table.**

   If you want to delete a table or change its settings, as well as in some other situations, you need to disable the table first, using the `disable` command. You can re-enable it using the `enable` command.

   ```
   hbase> disable 'test'
   ```

```
0 row(s) in 1.6270 seconds

hbase> enable 'test'
0 row(s) in 0.4500 seconds
```

9. **Drop the table.**

   To drop (delete) a table, use the `drop` command.

```
hbase> drop 'test'
0 row(s) in 0.2900 seconds
```

10. **Exit the HBase Shell.**

    To exit the HBase Shell and disconnect from your cluster, use the **quit**command. HBase is still running in the background.

Procedure 1.3. Stop HBase

1. In the same way that the `bin/start-hbase.sh` script is provided to conveniently start all HBase daemons, the `bin/stop-hbase.sh` script stops them.

```
$ ./bin/stop-hbase.sh
stopping hbase....................
$
```

2. After issuing the command, it can take several minutes for the processes to shut down. Use the **jps** to be sure that the HMaster and HRegionServer processes are shut down.

1.2.3. Intermediate - Pseudo-Distributed Local Install

After working your way through Section 1.2, “Quick Start - Standalone HBase”, you can re-configure HBase to run in pseudo-distributed mode. Pseudo-distributed mode means that HBase still runs completely on a single host, but each HBase daemon (HMaster, HRegionServer, and Zookeeper) runs as a separate process. By default, unless you configure the `hbase.rootdir` property as described inSection 1.2, “Quick Start - Standalone HBase”, your data is still stored in `/tmp/`. In this walk-through, we store your data in HDFS instead, assuming you have HDFS available. You can skip the HDFS configuration to continue storing your data in the local filesystem.

Hadoop Configuration

This procedure assumes that you have configured Hadoop and HDFS on your local system and or a remote system, and that they are running and available. It also assumes you are using Hadoop 2. Currently, the documentation on the Hadoop website does not include a quick start for Hadoop 2, but the guide athttp://www.alexjf.net/blog/distributed-systems/hadoop-yarn-installation-definitive-guide is a good starting point.

1. **Stop HBase if it is running.**

   If you have just finished Section 1.2, “Quick Start - Standalone HBase” and HBase is still running, stop it. This procedure will create a totally new directory where HBase will store its data, so any databases you created before will be lost.

2. **Configure HBase.**

   Edit the `hbase-site.xml` configuration. First, add the following property, which directs HBase to run in distributed mode, with one JVM instance per daemon.

```
<property>
  <name>hbase.cluster.distributed</name>
  <value>true</value>
</property>
```

   Next, change the `hbase.rootdir` from the local filesystem to the address of your HDFS instance, using the `hdfs:////` URI syntax. In this example, HDFS is running on the localhost at port 8020.

```
<property>
    <name>hbase.rootdir</name>
    <value>hdfs://localhost:8020/hbase</value>
  </property>
```

   You do not need to create the directory in HDFS. HBase will do this for you. If you create the directory, HBase will attempt to do a migration, which is not what you want.

3. **Start HBase.**

   Use the `bin/start-hbase.sh` command to start HBase. If your system is configured correctly, the **jps** command should show the HMaster and HRegionServer processes running.

4. **Check the HBase directory in HDFS.**

   If everything worked correctly, HBase created its directory in HDFS. In the configuration above, it is stored in `/hbase/` on HDFS. You can use the**hadoop fs** command in Hadoop's `bin/` directory to list this directory.

```
$ ./bin/hadoop fs -ls /hbase
Found 7 items
drwxr-xr-x   - hbase users          0 2014-06-25 18:58 /hbase/.tmp
drwxr-xr-x   - hbase users          0 2014-06-25 21:49 /hbase/WALs
drwxr-xr-x   - hbase users          0 2014-06-25 18:48 /hbase/corrupt
drwxr-xr-x   - hbase users          0 2014-06-25 18:58 /hbase/data
-rw-r--r--   3 hbase users         42 2014-06-25 18:41 /hbase/hbase.id
-rw-r--r--   3 hbase users          7 2014-06-25 18:41 /hbase/hbase.version
drwxr-xr-x   - hbase users          0 2014-06-25 21:49 /hbase/oldWALs
```

5. **Create a table and populate it with data.**

You can use the HBase Shell to create a table, populate it with data, scan and get values from it, using the same procedure as in Procedure 1.2, "Use HBase For the First Time".

6. **Start and stop a backup HBase Master (HMaster) server.**

   > **Note**
   >
   > Running multiple HMaster instances on the same hardware does not make sense in a production environment, in the same way that running a pseudo-distributed cluster does not make sense for production. This step is offered for testing and learning purposes only.

   The HMaster server controls the HBase cluster. You can start up to 9 backup HMaster servers, which makes 10 total HMasters, counting the primary. To start a backup HMaster, use the **local-master-backup.sh**. For each backup master you want to start, add a parameter representing the port offset for that master. Each HMaster uses two ports (16000 and 16010 by default). The port offset is added to these ports, so using an offset of 2, the first backup HMaster would use ports 16002 and 16012. The following command starts 3 backup servers using ports 16002/16012, 16003/16013, and 16005/16015.

   ```
   $ ./bin/local-master-backup.sh 2 3 5
   ```

   To kill a backup master without killing the entire cluster, you need to find its process ID (PID). The PID is stored in a file with a name like `/tmp/hbase-USER-X-master.pid`. The only contents of the file are the PID. You can use the **kill -9** command to kill that PID. The following command will kill the master with port offset 1, but leave the cluster running:

   ```
   $ cat /tmp/hbase-testuser-1-master.pid |xargs kill -9
   ```

7. **Start and stop additional RegionServers**

   The HRegionServer manages the data in its StoreFiles as directed by the HMaster. Generally, one HRegionServer runs per node in the cluster. Running multiple HRegionServers on the same system can be useful for testing in pseudo-distributed mode. The **local-regionservers.sh** command allows you to run multiple RegionServers. It works in a similar way to the**local-master-backup.sh** command, in that each parameter you provide represents the port offset for an instance. Each RegionServer requires two ports, and the default ports are 16200 and 16300. You can run 99 additional RegionServers, or 100 total, on a server. The following command starts four additional RegionServers, running on sequential ports starting at 16202/16302.

   ```
   $ .bin/local-regionservers.sh start 2 3 4 5
   ```

   To stop a RegionServer manually, use the **local-regionservers.sh** with the`stop` parameter and the offset of the server to stop.

   ```
   $ .bin/local-regionservers.sh stop 3
   ```

8. **Stop HBase.**

   You can stop HBase the same way as in the other procedure, using the`bin/stop-hbase.sh` command.

1.2.4. Advanced - Fully Distributed

In reality, you need a fully-distributed configuration to fully test HBase and to use it in real-world scenarios. In a distributed configuration, the cluster contains multiple nodes, each of which runs one or more HBase daemon. These include primary and backup Master instances, multiple Zookeeper nodes, and multiple RegionServer nodes.

This advanced quickstart adds two more nodes to your cluster. The architecture will be as follows:

Table 1.1. Distributed Cluster Demo Architecture

Node Name	Master	ZooKeeper	RegionServer
node-a.example.com	yes	yes	
node-b.example.com	backup	yes	yes
node-c.example.com	no	yes	yes

This quickstart assumes that each node is a virtual machine and that they are all on the same network. It builds upon the previous quickstart, Section 1.2.3, "Intermediate - Pseudo-Distributed Local Install", assuming that the system you configured in that procedure is now `node-a`. Stop HBase on `node-a` before continuing.

> **Note**
>
> Be sure that all the nodes have full access to communicate, and that no firewall rules are in place which could prevent them from talking to each other. If you see any errors like `no route to host`, check your firewall.

Procedure 1.4. Configure Password-Less SSH Access

`node-a` needs to be able to log into `node-b` and `node-c` (and to itself) in order to start the daemons. The easiest way to accomplish this is to use the same username on all hosts, and configure password-less SSH login from `node-a` to each of the others.

1. **On `node-a`, generate a key pair.**

   While logged in as the user who will run HBase, generate a SSH key pair, using the following command:

   ```
   $ ssh-keygen -t rsa
   ```

   If the command succeeds, the location of the key pair is printed to standard output. The default name of the public key is `id_rsa.pub`.

2. **Create the directory that will hold the shared keys on the other nodes.**

   On `node-b` and `node-c`, log in as the HBase user and create a `.ssh/` directory in the user's home directory, if it does not already exist. If it already exists, be aware that it may already contain other keys.

3. **Copy the public key to the other nodes.**

   Securely copy the public key from `node-a` to each of the nodes, by using the **scp** or some other secure means. On each of the other nodes, create a new file

called `.ssh/authorized_keys` *if it does not already exist*, and append the contents of the `id_rsa.pub` file to the end of it. Note that you also need to do this for `node-a` itself.

```
$ cat id_rsa.pub >> ~/.ssh/authorized_keys
```

4. **Test password-less login.**

   If you performed the procedure correctly, if you SSH from `node-a` to either of the other nodes, using the same username, you should not be prompted for a password.

5. Since `node-b` will run a backup Master, repeat the procedure above, substituting `node-b` everywhere you see `node-a`. Be sure not to overwrite your existing `.ssh/authorized_keys` files, but concatenate the new key onto the existing file using the >> operator rather than the > operator.

Procedure 1.5. Prepare `node-a`

`node-a` will run your primary master and ZooKeeper processes, but no RegionServers.

1. **Stop the RegionServer from starting on `node-a`.**

   Edit `conf/regionservers` and remove the line which contains `localhost`. Add lines with the hostnames or IP addresses for `node-b` and `node-c`. Save the file.

2. **Configure HBase to use `node-b` as a backup master.**

   Create a new file in `conf/` called `backup_masters`, and add a new line to it with the hostname for `node-b`. In this demonstration, the hostname is `node-b.example.com`.

3. **Configure ZooKeeper**

   In reality, you should carefully consider your ZooKeeper configuration. You can find out more about configuring ZooKeeper in Chapter 19, *ZooKeeper*. For now, we want to configure ZooKeeper to run on all three nodes.

   On `node-a`, edit `conf/hbase-site.xml` and add the following properties.

```
<property>
  <name>hbase.zookeeper.quorum</name>
  <value>node-a.example.com,node-b.example.com,node-c.example.com</value>
</property>
<property>
  <name>hbase.zookeeper.property.dataDir</name>
  <value>/usr/local/zookeeper</value>
</property>
```

4. Everywhere in your configuration that you have referred to `node-a` as`localhost`, change the reference to point to the hostname that the other nodes will use to refer to `node-a`. In these examples, the hostname is `node-a.example.com`.

Procedure 1.6. Prepare `node-b` and `node-c`

`node-b` will run a backup master server and a ZooKeeper instance.

1. **Download and unpack HBase.**

   Download and unpack HBase to `node-b`, just as you did for the standalone and pseudo-distributed quickstarts.

2. **Copy the configuration files from `node-a` to `node-b`.and `node-c`.**

   Each node of your cluster needs to have the same configuration information. Copy the contents of the `conf/` directory to the `conf/` directory on `node-b`.

Procedure 1.7. Start and Test Your Cluster

1. **Be sure HBase is not running on any node.**

   If you forgot to stop HBase from previous testing, you will have errors. Check to see whether HBase is running on any of your nodes by using the**jps** command. Look for the processes `HMaster`, `HRegionServer`, and`HQuorumPeer`. If they exist, kill them.

2. **Start the cluster.**

   On `node-a`, issue the **start-hbase.sh** command. Your output will be similar to that below.

```
$ bin/start-hbase.sh
node-c.example.com: starting zookeeper, logging to /home/hbuser/hbase-0.98.3-hadoop2/bin/../logs/hbase-hbuser-zookeeper-node-c.ex
node-a.example.com: starting zookeeper, logging to /home/hbuser/hbase-0.98.3-hadoop2/bin/../logs/hbase-hbuser-zookeeper-node-a.ex
node-b.example.com: starting zookeeper, logging to /home/hbuser/hbase-0.98.3-hadoop2/bin/../logs/hbase-hbuser-zookeeper-node-b.ex
starting master, logging to /home/hbuser/hbase-0.98.3-hadoop2/bin/../logs/hbase-hbuser-master-node-a.example.com.out
node-c.example.com: starting regionserver, logging to /home/hbuser/hbase-0.98.3-hadoop2/bin/../logs/hbase-hbuser-regionserver-noc
node-b.example.com: starting regionserver, logging to /home/hbuser/hbase-0.98.3-hadoop2/bin/../logs/hbase-hbuser-regionserver-noc
node-b.example.com: starting master, logging to /home/hbuser/hbase-0.98.3-hadoop2/bin/../logs/hbase-hbuser-master-nodeb.example.c
```

   ZooKeeper starts first, followed by the master, then the RegionServers.

3. **Verify that the processes are running.**

   On each node of the cluster, run the **jps** command and verify that the correct processes are running on each server. You may see additional Java processes running on your servers as well, if they are used for other purposes.

   Example 1.3. `node-a` jps Output

```
$ jps
20355 Jps
20071 HQuorumPeer
20137 HMaster
```

Example 1.4. `node-b` jps Output

```
$ jps
15930 HRegionServer
16194 Jps
15838 HQuorumPeer
16010 HMaster
```

Example 1.5. `node-c` jps Output

```
$ jps
13901 Jps
13639 HQuorumPeer
13737 HRegionServer
```

4. **Browse to the Web UI.**

   If everything is set up correctly, you should be able to the UI for the Master`http://node-a.example.com:6110/` or the secondary master at`http://node-b.example.com:6110/` for the secondary master. You can see the web UI for each of the RegionServers at port 6130 of their IP addresses, or by clicking their links in the web UI for the Master.

5. **Test what happens when nodes or services disappear.**

   With a three-node cluster like you have configured, things will not be very resilient. Still, you can test what happens when the primary Master or a RegionServer disappears, by killing the processes and watching the logs.

1.2.5. Where to go next

In the next chapter, Chapter 2, *Apache HBase Configuration*, we'll go into depth on the different HBase run modes, system requirements running HBase, and critical configurations setting up a distributed HBase deploy.

Chapter 2. Apache HBase Configuration

Table of Contents

This chapter is the Not-So-Quick start guide to Apache HBase configuration. It goes over system requirements, Hadoop setup, the different Apache HBase run modes, and the various configurations in HBase. Please read this chapter carefully. At a minimum ensure that all Section 2.1, “Basic Prerequisites” have been satisfied. Failure to do so will cause you (and us) grief debugging strange errors and/or data loss.

Apache HBase uses the same configuration system as Apache Hadoop. To configure a deploy, edit a file of environment variables in `conf/hbase-env.sh` -- this configuration is used mostly by the launcher shell scripts getting the cluster off the ground -- and then add configuration to an XML file to do things like override HBase defaults, tell HBase what Filesystem to use, and the location of the ZooKeeper ensemble. [11]

When running in distributed mode, after you make an edit to an HBase configuration, make sure you copy the content of the `conf` directory to all nodes of the cluster. HBase will not do this for you. Use **rsync**. For most configuration, a restart is needed for servers to pick up changes (caveat dynamic config. to be described later below).

2.1. Basic Prerequisites

This section lists required services and some required system configuration.

2.1.1. Java

HBase requires at least Java 6 from Oracle. The following table lists which JDK version are compatible with each version of HBase.

HBase Version	JDK 6	JDK 7	JDK 8
1.0	yes	yes	Running with JDK 8 will work but is not well tested.
0.98	yes	yes	Running with JDK 8 works but is not well tested. Building with JDK 8 would require removal of the deprecated remove() method of the PoolMap class and is under consideration. See ee HBASE-7608 for more information about JDK 8 support.
0.96	yes	yes	
0.94	yes	yes	

2.1.2. Operating System

2.1.2.1. ssh

ssh must be installed and **sshd** must be running to use Hadoop's scripts to manage remote Hadoop and HBase daemons. You must be able to ssh to all nodes, including your local node, using passwordless login (Google "ssh passwordless login"). If on mac osx, see the section, SSH: Setting up Remote Desktop and Enabling Self-Login on the hadoop wiki.

2.1.2.2. DNS

HBase uses the local hostname to self-report its IP address. Both forward and reverse DNS resolving must work in versions of HBase previous to 0.92.0 [2].

If your machine has multiple interfaces, HBase will use the interface that the primary hostname resolves to.

If this is insufficient, you can set `hbase.regionserver.dns.interface` to indicate the primary interface. This only works if your cluster configuration is consistent and every host has the same network interface configuration.

Another alternative is setting `hbase.regionserver.dns.nameserver` to choose a different nameserver than the system wide default.

2.1.2.3. Loopback IP

Previous to hbase-0.96.0, HBase expects the loopback IP address to be 127.0.0.1. See Section 2.1.2.3, “Loopback IP”

2.1.2.4. NTP

The clocks on cluster members should be in basic alignments. Some skew is tolerable but wild skew could generate odd behaviors. Run NTP on your cluster, or an equivalent.

If you are having problems querying data, or "weird" cluster operations, check system time!

2.1.2.5. `ulimit` and `nproc`

Apache HBase is a database. It uses a lot of files all at the same time. The default ulimit -n -- i.e. user file limit -- of 1024 on most *nix systems is insufficient (On mac os x its 256). Any significant amount of loading will lead you toSection 15.9.2.2, “java.io.IOException...(Too many open files)”. You may also notice errors such as the following:

```
2010-04-06 03:04:37,542 INFO org.apache.hadoop.hdfs.DFSClient: Exception increateBlockOutputStream java.io.EOFException
2010-04-06 03:04:37,542 INFO org.apache.hadoop.hdfs.DFSClient: Abandoning block blk_-6935524980745310745_1391901
```

Do yourself a favor and change the upper bound on the number of file descriptors. Set it to north of 10k. The math runs roughly as follows: per ColumnFamily there is at least one StoreFile and possibly up to 5 or 6 if the region is under load. Multiply the average number of StoreFiles per ColumnFamily times the number of regions per RegionServer. For example, assuming that a schema had 3 ColumnFamilies per region with an average of 3 StoreFiles per ColumnFamily, and there are 100 regions per RegionServer, the JVM will open 3 * 3 * 100 = 900 file descriptors (not counting open jar files, config files, etc.)

You should also up the hbase users' `nproc` setting; under load, a low-nproc setting could manifest as `OutOfMemoryError`. [3] [4]

To be clear, upping the file descriptors and nproc for the user who is running the HBase process is an operating system configuration, not an HBase configuration. Also, a common mistake is that administrators will up the file descriptors for a particular user but for whatever reason, HBase will be running as some one else. HBase prints in its logs as the first line the ulimit its seeing. Ensure its correct. [5]

2.1.2.5.1. `ulimit` on Ubuntu

If you are on Ubuntu you will need to make the following changes:

In the file `/etc/security/limits.conf` add a line like:

```
hadoop  -       nofile  32768
```

Replace `hadoop` with whatever user is running Hadoop and HBase. If you have separate users, you will need 2 entries, one for each user. In the same file set nproc hard and soft limits. For example:

```
hadoop soft/hard nproc 32000
```

In the file `/etc/pam.d/common-session` add as the last line in the file:

```
session required  pam_limits.so
```

Otherwise the changes in `/etc/security/limits.conf` won't be applied.

Don't forget to log out and back in again for the changes to take effect!

2.1.2.6. Windows

Previous to hbase-0.96.0, Apache HBase was little tested running on Windows. Running a production install of HBase on top of Windows is not recommended.

If you are running HBase on Windows pre-hbase-0.96.0, you must install Cygwinto have a *nix-like environment for the shell scripts. The full details are explained in the Windows Installation guide. Also search our user mailing list to pick up latest fixes figured by Windows users.

Post-hbase-0.96.0, hbase runs natively on windows with supporting ***.cmd** scripts bundled.

2.1.3. Hadoop

The below table shows some information about what versions of Hadoop are supported by various HBase versions. Based on the version of HBase, you should select the most appropriate version of Hadoop. We are not in the Hadoop distro selection business. You can use Hadoop distributions from Apache, or learn about vendor distributions of Hadoop athttp://wiki.apache.org/hadoop/Distributions%20and%20Commercial%20Support

Hadoop 2.x is better than Hadoop 1.x

Hadoop 2.x is faster, with more features such as short-circuit reads which will help improve your HBase random read profile as well important bug fixes that will improve your overall HBase experience. You should run Hadoop 2 rather than Hadoop 1. HBase 0.98 deprecates use of Hadoop1. HBase 1.0 will not support Hadoop1.

Use the following legend to interpret this table:

S = supported and tested,
X = not supported,
NT = it should run, but not tested enough.

Table 2.1. Hadoop version support matrix

	HBase-0.92.x	HBase-0.94.x	HBase-0.96.x	HBase-0.98.x[a]	HBase-1.0.x[b]
Hadoop-0.20.205	S	X	X	X	X
Hadoop-0.22.x	S	X	X	X	X
Hadoop-1.0.0-1.0.2[c]	X	X	X	X	X
Hadoop-1.0.3+	S	S	S	X	X
Hadoop-1.1.x	NT	S	S	X	X
Hadoop-0.23.x	X	S	NT	X	X
Hadoop-2.0.x-alpha	X	NT	X	X	X
Hadoop-2.1.0-beta	X	NT	S	X	X
Hadoop-2.2.0	X	NT [d]	S	S	NT
Hadoop-2.3.x	X	NT	S	S	NT
Hadoop-2.4.x	X	NT	S	S	S
Hadoop-2.5.x	X	NT	S	S	S

[a]Support for Hadoop 1.x is deprecated.

[b]Hadoop 1.x is NOT supported

[c]HBase requires hadoop 1.0.3 at a minimum; there is an issue where we cannot find KerberosUtil compiling against earlier versions of Hadoop.

[d]To get 0.94.x to run on hadoop 2.2.0, you need to change the hadoop 2 and protobuf versions in the `pom.xml`: Here is a diff with pom.xml changes:

```
$ svn diff pom.xml
Index: pom.xml
===================================================================
--- pom.xml     (revision 1545157)
+++ pom.xml     (working copy)
@@ -1034,7 +1034,7 @@
     <slf4j.version>1.4.3</slf4j.version>
     <log4j.version>1.2.16</log4j.version>
     <mockito-all.version>1.8.5</mockito-all.version>
-    <protobuf.version>2.4.0a</protobuf.version>
+    <protobuf.version>2.5.0</protobuf.version>
     <stax-api.version>1.0.1</stax-api.version>
     <thrift.version>0.8.0</thrift.version>
     <zookeeper.version>3.4.5</zookeeper.version>
@@ -2241,7 +2241,7 @@
         </property>
       </activation>
       <properties>
-        <hadoop.version>2.0.0-alpha</hadoop.version>
+        <hadoop.version>2.2.0</hadoop.version>
         <slf4j.version>1.6.1</slf4j.version>
       </properties>
       <dependencies>
```

The next step is to regenerate Protobuf files and assuming that the Protobuf has been installed:

- Go to the hbase root folder, using the command line;
- Type the following commands:

```
$ protoc -Isrc/main/protobuf --java_out=src/main/java src/main/protobuf/hbase.proto

$ protoc -Isrc/main/protobuf --java_out=src/main/java src/main/protobuf/ErrorHandling.proto
```

Building against the hadoop 2 profile by running something like the following command:

```
$  mvn clean install assembly:single -Dhadoop.profile=2.0 -DskipTests
```

Replace the Hadoop Bundled With HBase!

Because HBase depends on Hadoop, it bundles an instance of the Hadoop jar under its `lib` directory. The bundled jar is ONLY for use in standalone mode. In distributed mode, it is *critical* that the version of Hadoop that is out on your cluster match what is under HBase. Replace the hadoop jar found in the HBase lib directory with the hadoop jar you are running on your cluster to avoid version mismatch issues. Make sure you replace the jar in HBase everywhere on your cluster. Hadoop version mismatch issues have various manifestations but often all looks like its hung up.

2.1.3.1. Apache HBase 0.92 and 0.94

HBase 0.92 and 0.94 versions can work with Hadoop versions, 0.20.205, 0.22.x, 1.0.x, and 1.1.x. HBase-0.94 can additionally work with Hadoop-0.23.x and 2.x, but you may have to recompile the code using the specific maven profile (see top level pom.xml)

2.1.3.2. Apache HBase 0.96

As of Apache HBase 0.96.x, Apache Hadoop 1.0.x at least is required. Hadoop 2 is strongly encouraged (faster but also has fixes that help MTTR). We will no longer run properly on older Hadoops such as 0.20.205 or branch-0.20-append. Do not move to Apache HBase 0.96.x if you cannot upgrade your Hadoop.[6]

2.1.3.3. Hadoop versions 0.20.x - 1.x

HBase will lose data unless it is running on an HDFS that has a durable `sync`implementation. DO NOT use Hadoop 0.20.2, Hadoop 0.20.203.0, and Hadoop 0.20.204.0 which DO NOT have this attribute. Currently only Hadoop versions 0.20.205.x or any release in excess of this version -- this includes hadoop-1.0.0 -- have a working, durable sync [7]. Sync has to be explicitly enabled by setting`dfs.support.append` equal to true on both the client side -- in `hbase-site.xml` -- and on the serverside in `hdfs-site.xml` (The sync facility HBase needs is a subset of the append code path).

```
<property>
  <name>dfs.support.append</name>
  <value>true</value>
</property>
```

You will have to restart your cluster after making this edit. Ignore the chicken-little comment you'll find in the `hdfs-default.xml` in the description for the`dfs.support.append` configuration.

2.1.3.4. Apache HBase on Secure Hadoop

Apache HBase will run on any Hadoop 0.20.x that incorporates Hadoop security features as long as you do as suggested above and replace the Hadoop jar that ships with HBase with the secure version. If you want to read more about how to setup Secure HBase, see Section 8.1, “Secure Client Access to Apache HBase”.

2.1.3.5. `dfs.datanode.max.transfer.threads`

An HDFS datanode has an upper bound on the number of files that it will serve at any one time. Before doing any loading, make sure you have configured Hadoop's`conf/hdfs-site.xml`, setting the `dfs.datanode.max.transfer.threads` value to at least the following:

```
<property>
  <name>dfs.datanode.max.transfer.threads</name>
  <value>4096</value>
</property>
```

Be sure to restart your HDFS after making the above configuration.

Not having this configuration in place makes for strange-looking failures. One manifestation is a complaint about missing blocks. For example:

```
10/12/08 20:10:31 INFO hdfs.DFSClient: Could not obtain block
          blk_XXXXXXXXXXXXXXXXXXXXXX_YYYYYYYY from any node: java.io.IOException: No live nodes
          contain current block. Will get new block locations from namenode and retry...
```

See also Section 16.3.4, “Case Study #4 (max.transfer.threads Config)” and note that this property was previously known as `dfs.datanode.max.xcievers` (e.g.Hadoop HDFS: Deceived by Xciever).

2.2. HBase run modes: Standalone and Distributed

HBase has two run modes: Section 2.2.1, “Standalone HBase” and Section 2.2.2, “Distributed”. Out of the box, HBase runs in standalone mode. Whatever your mode, you will need to configure HBase by editing files in the HBase `conf`directory. At a minimum, you must edit `conf/hbase-env.sh` to tell HBase which**java** to use. In this file you set HBase environment variables such as the heapsize and other options for the JVM, the preferred location for log files, etc. Set`JAVA_HOME` to point at the root of your **java** install.

2.2.1. Standalone HBase

This is the default mode. Standalone mode is what is described in the Section 1.2, “Quick Start - Standalone HBase” section. In standalone mode, HBase does not use HDFS -- it uses the local filesystem instead -- and it runs all HBase daemons and a local ZooKeeper all up in the same JVM. Zookeeper binds to a well known port so clients may talk to HBase.

2.2.2. Distributed

Distributed mode can be subdivided into distributed but all daemons run on a single node -- a.k.a *pseudo-distributed*-- and *fully-distributed* where the daemons are spread across all nodes in the cluster [8].

Pseudo-distributed mode can run against the local filesystem or it can run against an instance of the *Hadoop Distributed File System* (HDFS). Fully-distributed mode can ONLY run on HDFS. See the Hadoop requirements and instructions for how to set up HDFS for Hadoop 1.x. A good walk-through for setting up HDFS on Hadoop 2 is at http://www.alexjf.net/blog/distributed-systems/hadoop-yarn-installation-definitive-guide.

Below we describe the different distributed setups. Starting, verification and exploration of your install, whether a *pseudo-distributed* or *fully-distributed*configuration is described in a section that follows, Section 2.3, “Running and Confirming Your Installation”. The same verification script applies to both deploy types.

2.2.2.1. Pseudo-distributed

Pseudo-Distributed Quickstart

A quickstart has been added to the Section 1.2, "Quick Start - Standalone HBase" chapter. See Section 1.2.3, "Intermediate - Pseudo-Distributed Local Install". Some of the information that was originally in this section has been moved there.

A pseudo-distributed mode is simply a fully-distributed mode run on a single host. Use this configuration testing and prototyping on HBase. Do not use this configuration for production nor for evaluating HBase performance.

2.2.3. Fully-distributed

Typically, HBase is run in a fully-distributed mode, where the different daemons run on multiple servers in the cluster. The `hbase-cluster.distributed` property is set to `true`, just as in pseudo-distributed mode. The `hbase.rootdir` is also typically set to an HDFS URI not hosted on the localhost. Following is an example of this bare-bones distributed configuration.

```
<configuration>
  ...
  <property>
    <name>hbase.rootdir</name>
    <value>hdfs://namenode.example.org:8020/hbase</value>
    <description>The directory shared by RegionServers.
    </description>
  </property>
  <property>
    <name>hbase.cluster.distributed</name>
    <value>true</value>
    <description>The mode the cluster will be in. Possible values are
      false: standalone and pseudo-distributed setups with managed Zookeeper
      true: fully-distributed with unmanaged Zookeeper Quorum (see hbase-env.sh)
    </description>
  </property>
  ...
</configuration>
```

Distributed HBase Quickstart. See Section 1.2.4, "Advanced - Fully Distributed" for a walk-through of a simple three-node cluster configuration with multiple ZooKeeper, backup HMaster, and RegionServer instances.

Distributed RegionServers. Typically, your cluster will contain multiple RegionServers all running on different servers, as well as primary and backup Master and Zookeeper daemons. The `conf/regionservers` file on the master server contains a list of hosts whose RegionServers are associated with this cluster. Each host is on a separate line. All hosts listed in this file will have their RegionServer processes started and stopped when the master server starts or stops.

ZooKeeper and HBase. See section Chapter 19, *ZooKeeper* for ZooKeeper setup for HBase.

2.2.3.1. HDFS Client Configuration

Of note, if you have made *HDFS client configuration* on your Hadoop cluster -- i.e. configuration you want HDFS clients to use as opposed to server-side configurations -- HBase will not see this configuration unless you do one of the following:

- Add a pointer to your `HADOOP_CONF_DIR` to the `HBASE_CLASSPATH`environment variable in `hbase-env.sh`.
- Add a copy of `hdfs-site.xml` (or `hadoop-site.xml`) or, better, symlinks, under `${HBASE_HOME}/conf`, or
- if only a small set of HDFS client configurations, add them to `hbase-site.xml`.

An example of such an HDFS client configuration is `dfs.replication`. If for example, you want to run with a replication factor of 5, hbase will create files with the default of 3 unless you do the above to make the configuration available to HBase.

2.3. Running and Confirming Your Installation

Make sure HDFS is running first. Start and stop the Hadoop HDFS daemons by running `bin/start-hdfs.sh` over in the `HADOOP_HOME` directory. You can ensure it started properly by testing the **put** and **get** of files into the Hadoop filesystem. HBase does not normally use the mapreduce daemons. These do not need to be started.

If you are managing your own ZooKeeper, start it and confirm its running else, HBase will start up ZooKeeper for you as part of its start process.

Start HBase with the following command:

```
bin/start-hbase.sh
```

Run the above from the `HBASE_HOME` directory.

You should now have a running HBase instance. HBase logs can be found in the`logs` subdirectory. Check them out especially if HBase had trouble starting.

HBase also puts up a UI listing vital attributes. By default its deployed on the Master host at port 16010 (HBase RegionServers listen on port 16020 by default and put up an informational http server at 16030). If the Master were running on a host named `master.example.org` on the default port, to see the Master's homepage you'd point your browser at `http://master.example.org:16010`.

Prior to HBase 0.98, the default ports the master ui was deployed on port 16010, and the HBase RegionServers would listen on port 16020 by default and put up an informational http server at 16030.

Once HBase has started, see the Procedure 1.2, "Use HBase For the First Time"for how to create tables, add data, scan your insertions, and finally disable and drop your tables.

To stop HBase after exiting the HBase shell enter

```
$ ./bin/stop-hbase.sh
stopping hbase...............
```

Shutdown can take a moment to complete. It can take longer if your cluster is comprised of many machines. If you are running a distributed operation, be sure to wait

until HBase has shut down completely before stopping the Hadoop daemons.

2.4. Configuration Files