Preface

This is the official reference guide for the HBase version it ships with. Herein you will find either the definitive documentation on an HBase topic as of its standing when the referenced HBase version shipped, or it will point to the location in javadoc, JIRA or wiki where the pertinent information can be found.

This reference guide is a work in progress. The source for this guide can be found at `src/main/docbkx` in a checkout of the hbase project. This reference guide is marked up using DocBook from which the the finished guide is generated as part of the 'site' build target. Run

```
mvn site
```

to generate this documentation. Amendments and improvements to the documentation are welcomed. Add a patch to an issue up in the HBase JIRA.

Heads-up if this is your first foray into the world of distributed computing...

If this is your first foray into the wonderful world of Distributed Computing, then you are in for some interesting times. First off, distributed systems are hard; making a distributed system hum requires a disparate skillset that spans systems (hardware and software) and networking. Your cluster' operation can hiccup because of any of a myriad set of reasons from bugs in HBase itself through misconfigurations -- misconfiguration of HBase but also operating system misconfigurations -- through to hardware problems whether it be a bug in your network card drivers or an underprovisioned RAM bus (to mention two recent examples of hardware issues that manifested as "HBase is slow"). You will also need to do a recalibration if up to this your computing has been bound to a single box. Here is one good starting point: Fallacies of Distributed Computing. That said, you are welcome. Its a fun place to be. Yours, the HBase Community.

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

The below advice is for HBase 0.98.2 and earlier releases only. This is fixed in HBase 0.98.3 and beyond. See HBASE-11272 and HBASE-11218.

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. See https://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 Table 2.1, "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
```

   Disable the table again if you tested the **enable** command above:

```
hbase> disable 'test'
0 row(s) in 1.6270 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 in Section 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 at http://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.example.com.out
   node-a.example.com: starting zookeeper, logging to /home/hbuser/hbase-0.98.3-hadoop2/bin/../logs/hbase-hbuser-zookeeper-node-
   a.example.com.out
   node-b.example.com: starting zookeeper, logging to /home/hbuser/hbase-0.98.3-hadoop2/bin/../logs/hbase-hbuser-zookeeper-node-
   b.example.com.out
   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-
   node-c.example.com.out
   ```

```
node-b.example.com: starting regionserver, logging to /home/hbuser/hbase-0.98.3-hadoop2/bin/../logs/hbase-hbuser-regionserver-
node-b.example.com.out
node-b.example.com: starting master, logging to /home/hbuser/hbase-0.98.3-hadoop2/bin/../logs/hbase-hbuser-master-
nodeb.example.com.out
```

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
   ```

   #### ZooKeeper Process Name

   The `HQuorumPeer` process is a ZooKeeper instance which is controlled and started by HBase. If you use ZooKeeper this way, it is limited to one instance per cluster node, , and is appropriate for testing only. If ZooKeeper is run outside of HBase, the process is called `QuorumPeer`. For more about ZooKeeper configuration, including using an external ZooKeeper instance with HBase, see Chapter 19, *ZooKeeper*.

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

   #### Web UI Port Changes

   In HBase newer than 0.98.x, the HTTP ports used by the HBase Web UI changed from 6010 for the Master and 6030 for each RegionServer to 16610 for the Master and 16030 for the RegionServer.

   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 expands upon the Chapter 1, *Getting Started* chapter to further explain configuration of Apache HBase. Please read this chapter carefully, especially Section 2.1, “Basic Prerequisites” to ensure that your HBase testing and deployment goes smoothly, and prevent data loss.

Apache HBase uses the same configuration system as Apache Hadoop. All configuration files are located in the `conf/` directory, which needs to be kept in sync for each node on your cluster.

HBase Configuration Files

`backup-masters`

Not present by default. A plain-text listing hosts on which the Master should start a backup Master process, one host per line.

`hadoop-metrics2-hbase.properties`

Used to gather HBase-related metrics to send to Hadoop's Metrics2 framework. See the Hadoop Wiki entry for more information on Metrics2. Contains only commented-out examples by default.

`hbase-env.cmd` and `hbase-env.sh`

Script for Windows and Linux / Unix environments to set up the working environment for HBase, including the location of Java, Java options, and other environment variables. The file contains many commented-out examples to provide guidance.

`hbase-policy.xml`

The default policy configuration file used by RPC servers to make authorization decisions on client requests if HBase security is enabled.

`hbasde-site.xml`

The main HBase configuration mechanism. This file specifies configuration options which are different from HBase's default configuration. You can view (but do not edit) the default configuration file at `docs/hbase-default.xml`. You can also view the entire effective configuration for your cluster (defaults and overrides) in the HBase Configuration tab of the HBase Web UI.

`log4j.properties`

Configuration file for HBase logging via `log4j`.

`regionservers`

A plain-text file containing a list of hosts which should run a RegionServer in your HBase cluster. By default this file contains the single entry `localhost`. It should contain a list of hostnames or IP addresses, one per line.

Checking XML Validity

When you edit XML, it is a good idea to use an XML-aware editor to be sure that your syntax is correct and your XML is well-formed. You can also use the **xmllint** utility to check that your XML is well-formed. By default, **xmllint** re-flows and prints the XML to standard output. To check for well-formedness and only print output if errors exist, use the command

xmllint -noout *filename.xml*.

Keep Configuration In Sync Across the Cluster

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**, **scp**, or another secure mechanism for copying the configuration files to your nodes. For most configuration, a restart is needed for servers to pick up changes An exception is dynamic configuration. to be described later below.

2.1. Basic Prerequisites

This section lists required services and some required system configuration.

Table 2.1. Java

HBase Version	JDK 6	JDK 7	JDK 8
1.0	Not Supported	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	

[D]

Operating System Utilities

ssh

HBase uses the Secure Shell (ssh) command and utilities extensively to communicate between cluster nodes. Each server in the cluster must be running **ssh** so that the Hadoop and HBase daemons can be managed. You must be able to connect to all nodes via SSH, including the local node, from the Master as well as any backup Master, using a shared key rather than a password. You can see the basic methodology for such a set-up in Linux or Unix systems at Procedure 1.4, “Configure Password-Less SSH Access”. If your cluster nodes use OS X, see the section, SSH: Setting up Remote Desktop and Enabling Self-Login on the Hadoop wiki.

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.[1]

If your server has multiple network interfaces, HBase defaults to using the interface that the primary hostname resolves to. To override this behavior, set the `hbase.regionserver.dns.interface` property to a different interface. This will only work if each server in your cluster uses the same network interface configuration.

To choose a different DNS nameserver than the system default, set the `hbase.regionserver.dns.nameserver` property to the IP address of that nameserver.

Loopback IP

Prior to hbase-0.96.0, HBase only used the IP address `127.0.0.1` to refer to `localhost`, and this could not be configured. See Loopback IP

NTP

The clocks on cluster members should be synchronized. A small amount of variation is acceptable, but larger amounts of skew can cause erratic and unexpected behavior. Time synchronization is one of the first things to check if you see unexplained problems in your cluster. It is recommended that you run a Network Time Protocol (NTP) service, or another time-synchronization mechanism, on your cluster, and that all nodes look to the same service for time synchronization. See the Basic NTP Configuration at the Linux Documentation Project to set up NTP.

`ulimit` and `nproc`

Apache HBase is a database. It requires the ability to open a large number of files at once. Many Linux distributions limit the

number of files a single user is allowed to open to 1024 (or 256 on older versions of OS X). You can check this limit on your servers by running the command **ulimit -n**. See Section 15.9.2.2, "java.io.IOException...(Too many open files)" for some of the problems you may experience. 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
```

It is recommended to raise the ulimit to at least 10,000, but more likely 10,240, because the value is usually expressed in multiples of 1024. Each ColumnFamily has at least one StoreFile, and possibly more than 6 StoreFiles if the region is under load. The following is a rough formula for calculating the potential number of open files on a RegionServer.

Example 2.1. Calculate the Potential Number of Open Files

```
(StoreFiles per ColumnFamily) x (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, configuration files, and others. Opening a file does not take many resources, and the risk of allowing a user to open too many files is minimal.

Another related setting is the number of processes a user is allowed to use. In Linux and Unix, the number of processes is set using the **ulimit -u** command. This should not be confused with the **nproc** command, which controls the number of CPUs available to a given user. Under load, a `nproc` that is too low can cause OutOfMemoryError exceptions. See Jack Levin's major hdfs issues thread on the hbase-users mailing list, from 2011.

Configuring the fmaximum number of ile descriptors and processes for the user who is running the HBase process is an operating system configuration, rather than an HBase configuration. It is also important to be sure that the settings are changed for the user that actually runs HBase. To see which user started HBase, and that user's ulimit configuration, look at the first line of the HBase log for that instance.[2]

ulimit Settings on Ubuntu. To configure **ulimit** settings on Ubuntu, edit `/etc/security/limits.conf`, which is a space-delimited file with four columns. Refer to the man page for limits.conf for details about the format of this file. In the following example, the first line sets both soft and hard limits for the number of open files (`nofile`) to `32768` for the operating system user with the username `hadoop`. The second line sets the number of processes to 32000 for the same user.

```
hadoop  -       nofile  32768
hadoop  -       nproc   32000
```

The settings are only applied if the Pluggable Authentication Module (PAM) environment is directed to use them. To configure PAM to use these limits, be sure that the `/etc/pam.d/common-session` file contains the following line:

```
session required  pam_limits.so
```

Windows

Prior to HBase 0.96, testing for running HBase on Microsoft Windows was limited. Running a on Windows nodes is not recommended for production systems.

To run versions of HBase prior to 0.96 on Microsoft Windows, you must install Cygwin and run HBase within the Cygwin environment. This provides support for Linux/Unix commands and 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.

2.1.1. Hadoop

The following table summarizes the versions of Hadoop supported with each version of HBase. Based on the version of HBase, you should select the most appropriate version of Hadoop. You can use Apache Hadoop, or a vendor's distribution of Hadoop. No distinction is made here. See http://wiki.apache.org/hadoop/Distributions%20and%20Commercial%20Support for information about vendors of Hadoop.

Hadoop 2.x is recommended.

Hadoop 2.x is faster and includes features, such as short-circuit reads, which will help improve your HBase random read profile. Hadoop 2.x also includes important bug fixes that will improve your overall HBase experience. HBase 0.98 deprecates use of Hadoop 1.x, and HBase 1.0 will not support Hadoop 1.x.

Use the following legend to interpret this table: