# NutchHadoopTutorial

## Nutch and Hadoop Tutorial

As of the official Nutch 1.3 release the source code architecture has been greatly simplified to allow us to run Nutch in one of two modes; namely **local** and **deploy**. By default, Nutch no longer comes with a Hadoop distribution, however when run in local mode e.g. running Nutch in a single process on one machine, then we use Hadoop as a dependency. This may suit you fine if you have a small site to crawl and index, but most people choose Nutch because of its capability to run on in deploy mode, within a Hadoop cluster. This gives you the benefit of a distributed file system (HDFS) and MapReduce processing style. The purpose of this tutorial is to provide a step-by-step method to get Nutch running with the Hadoop file system on multiple machines, including being able to both crawl and search across multiple machines. N.B. This tutorial is designed and maintained to work with Nutch trunk.

This document does not go into the Nutch or Hadoop architecture, resources relating to these topics can be found here. It only tells how to get the systems up and running. There are also relevant resources at the end of this tutorial if you want to know more about the architecture of Nutch and Hadoop.

**N.B. 1.** Prerequsites for this tutorial are both the Nutch Tutorial and the Hadoop Tutorial. It will also be of great benefit to have a look at the Hadoop Wiki.

**2.** In addition it is really really easy to get Nutch running if you already have an existing Hadoop cluster up and running, therefore it is strongly advised to begin with the Hadoop cluster setup then come over to this tutorial.



### Assumptions

Some things are assumed for this tutorial:

1) Perform this setup using root level access. This includes setting up the same user across multiple machines and setting up a local filesystem outside of the user's home directory. Root access is not required to setup Nutch and Hadoop (although sometimes it is convenient). If you do not have root access, you will need the same user setup across all machines which you are using and you will probably need to use a local filesystem inside of your home directory.

2) All boxes will need an SSH server running (not just a client) as Hadoop uses SSH to start slave servers. Although we try to explain how to set up ssh so that communication between machines does not require a password you may need to learn how to do that elsewhere. Please see halfway down this tutorial here. N.B. It is important that initially you only create keys on the Master node, they are then copied over to your Slave nodes.

3) For this tutorial we setup Nutch across a 6 node Hadoop cluster. If you are using a different number of machines you should still be fine but you should have at least two different machines to prove the distributed capabilities of both HDFS and MapReduce.

4) If something doesn't work for you try first searching then sending a message to the Nutch or Hadoop users mailing list. Good questions as well as suggestions or tips are welcome. Why not add them to the end of this Wiki page?

5) A real no brainer... we assume that you are a Java programmer familiar with the concepts of JAVA_HOME, ant build tool, subversion, IDEs and such like.

Ok lets have some fun!

## Hadoop Cluster Setup

It is important to know that you don't have to have big hardware to get up and running with Nutch and Hadoop. The architecture was designed in such a way to make the most of commodity hardware. For the purpose of this tutorial the nodes in the 6 node cluster are named as follows:

```
devcluster01
devcluster02
devcluster03
devcluster04
devcluster05
devcluster06
```

To begin, our master node is devcluster01, by master node I mean that it will run the Hadoop services that coordinate with the slave nodes (all of the other computers) and it is the machine on which we performed our crawl.

## Downloading Hadoop and Nutch

Both Nutch and Hadoop are downloadable from their respective Apache websites.

You can checkout the latest and greatest Nutch from source after downloading it from its SVN repository here. Alternatively pick up a stable release from the Nutch site. The same should be done with Hadoop, and as mentioned eariler, this along with how to set up your 6 node cluster is included in the Hadoop Tutorial. This should be done with one every node you wish to include within your cluster e.g. both Nutch and Hadoop packages should be installed in every machine.

We are going to use ant to build it so if you have java and ant installed you should be fine. This tutorial is not going to go into how to install java or ant, if you want a complete reference for ant pick up Erik Hatcher's book Java Development with Ant.

## Setting Up The Deployment Architecture

Once we get nutch deployed to all six machines we are going to call a script called start-all.sh that starts the services on the master node and data nodes. This means that the script is going to start the hadoop daemons on the master node and then will ssh into all of the slave nodes and start daemons on the slave nodes.

The start-all.sh script is going to expect that Nutch is installed in exactly the same location on every machine. It is also going to expect that Hadoop is storing the data at the exact same filepath on every machine.

The start-all.sh script that starts the daemons on the master and slave nodes is going to need to be able to use a password-less login through ssh. Since the master node is going to start daemons on itself we also need the ability to user a password-less login on itself, however this should have already been done by now.

We need to setup the environment variables inside of the hadoop-env.sh file. Open the hadoop-env.sh file inside of vi:

```
cd HADOOP_HOME/conf
vi hadoop-env.sh
```

Below is a template for the environment variables that need to be changed in the hadoop-env.sh file:

```
export HADOOP_HOME=/PATH/TO/HADOOP_HOME
export JAVA_HOME=/PATH/TO/JDK_HOME
export HADOOP_LOG_DIR=${HADOOP_HOME}/logs
```

Additionally at this stage, in accordance with the Hadoop tutorial, add the IP addresses of your Master and Slaves nodes to HADOOP_HOME/conf /masters & HADOOP_HOME/conf/slaves respectively.

There are numerous other variables (documented elsewhere) in the HADOOP_HOME/conf directory which will affect the behaviour of Hadoop. If when you start running the script later you start getting ssh errors, try changing the HADOOP_SSH_OPTS variable.

Next we are going to create the keys on the master node and copy them over to each of the slave nodes. This must be done as the Nutch root user we created earlier. Don't just sudo in as the Nutch user, start up a new shell and login as the Nutch user. If you sudo in the password-less login we are about to setup will not work in testing but will work when a new session is started as the Nutch user.

```
cd NUTCH_HOME

ssh-keygen -t rsa (Use empty responses for each prompt)
  Enter passphrase (empty for no passphrase):
  Enter same passphrase again:
  Your identification has been saved in /nutch/home/.ssh/id_rsa.
  Your public key has been saved in /nutch/home/.ssh/id_rsa.pub.
  The key fingerprint is:
  a6:5c:c3:eb:18:94:0b:06:a1:a6:29:58:fa:80:0a:bc nutch@localhost
```

On the master node you will copy the public key you just created to a file called authorized_keys in the same directory:

```
cd NUTCH_HOME/.ssh
cp id_rsa.pub authorized_keys
```

You only have to run the ssh-keygen on the master node. On each of the slave nodes after the filesystem is created you will just need to copy the keys over using scp. eg to send the authorisation from to devcluster02 we might do this on devcluster01

```
scp NUTCH_HOME/.ssh/authorized_keys nutch@devcluster02:NUTCH_HOME/.ssh/authorized_keys
```

You will have to enter the password for the Nutch user the first time. A ssh prompt will appear the first time you login to each computer asking if you want to add the computer to the known hosts. Answer yes to the prompt. Once the key is copied you shouldn't have to enter a password when logging in as the Nutch root user. Test it by logging into the slave nodes that you just copied the keys to:

```
ssh devcluster02
nutch@devcluster02$ (a command prompt should appear without requiring a password)
hostname (should return the name of the slave node, here devcluster02)
```

Once we have the ssh keys created we are ready to start deploying Nutch to all of the slave nodes.

(Note: this is a rather simple example of how to set up ssh without requiring a passphrase. There are other documents available which can help you with this if you have problems. It is important to test that the Nutch user can ssh to all of the machines in your cluster so don't skip this stage)

## Deploy Nutch to Single Machine

First we will deploy nutch to a single node, the master node, but operate it in distributed mode. This means that it will use the Hadoop filesystem instead of the local filesystem. We will start with a single node to make sure that everything is up and running and will then move on to adding the other slave nodes. All of the following should be done from a session started as the nutch user. We are going to setup nutch on the master node and then when we are ready we will copy the entire installation to the slave nodes.

First copy the files from the nutch build to the deploy directory using something like the following command:

```
cp -R /path/to/build/* /nutch/search
```

Then make sure that all of the shell scripts are in unix format and are executable.

```
dos2unix /nutch/search/bin/*.sh /nutch/search/bin/hadoop /nutch/search/bin/nutch
chmod 700 /nutch/search/bin/*.sh /nutch/search/bin/hadoop /nutch/search/bin/nutch
dos2unix /nutch/search/config/*.sh
chmod 700 /nutch/search/config/*.sh
```

When we were first trying to setup nutch we were getting bad interpreter and command not found errors because the scripts were in dos format on linux and not executable. Notice that we are doing both the bin and config directory. In the config directory there is a file called hadoop-env.sh that is called by other scripts.

There are a few scripts that you will need to be aware of. In the bin directory there is the nutch script, the hadoop script, the start-all.sh script and the stop-all.sh script. The nutch script is used to do things like start the nutch crawl. The hadoop script allows you it interact with the hadoop file system.
The start-all.sh script starts all of the servers on the master and slave nodes. The stop-all.sh. scrip stops all of the servers.

If you want to see options for nutch use the following command:

```
bin/nutch
```

Or if you want to see the options for hadoop use:

```
bin/hadoop
```

If you want to see options for Hadoop components such as the distributed filesystem then use the component name as input like below:

```
bin/hadoop dfs
```

There are also files that you need to be aware of. In the conf directory there are the nutch-default.xml, the nutch-site.xml, the hadoop-default.xml and the hadoop-site.xml. The nutch-default.xml file holds all of the default options for nutch, the hadoop-default.xml file does the same for hadoop. To override any of these options, we copy the properties to their respective *-site.xml files and change their values. Previously we had all the hadoop configuration in a file called hadoop-site.xml but recent versions have put hadoop related config in different files:

There is also a file named slaves inside the config directory. This is where we put the names of the slave nodes. Since we are running a slave data node on the same machine we are running the master node, we will also need the local computer in this slave list. Here is what the slaves file will look like to start.

```
localhost
```

It comes this way to start so you shouldn't have to make any changes. Later we will add all of the nodes to this file, one node per line.

Previously all the Hadoop configuration was in one file (hadoop-site.xml) but now we need to put roughly the same data in separate files. See http://hadoop.apache.org/common/docs/current/quickstart.html for more information. We are basically adding property entries inside the configuration tags...

### conf/core-site.xml:

```
<?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://devcluster01:9000</value>
    <description>
       Where to find the Hadoop Filesystem through the network.
       Note 9000 is not the default port.
       (This is slightly changed from previous versions which didnt have "hdfs")
    </description>
  </property>
</configuration>
```

The fs.default.name property is used by nutch to determine the filesystem that it is going to use. Since we are using the hadoop filesystem we have to point this to the hadoop master or name node. In this case it is hdfs://devcluster01:9000 which is the server that houses the name node on our network.

The hadoop package really comes with two components. One is the distributed filesystem. Two is the mapreduce functionality. While the distibuted filesystem allows you to store and replicate files over many commodity machines, the mapreduce package allows you to easily perform parallel programming tasks.

### conf/hdfs-site.xml:

```
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!-- Put site-specific property overrides in this file. -->
<configuration>

<property>
  <name>dfs.name.dir</name>
  <value>/nutch/filesystem/name</value>
</property>

<property>
  <name>dfs.data.dir</name>
  <value>/nutch/filesystem/data</value>
</property>

<property>
  <name>dfs.replication</name>
  <value>1</value>
</property>

</configuration>
```

Note that the dfs-replication value of "1" means no duplication. This is only meaningful on a single machine test cluster. It should typically be 3 or more - but of course you must have at least that number of working nodes in your Hadoop cluster.

### conf/mapred-site.xml:

```
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!-- Put site-specific property overrides in this file. -->
<configuration>

<property>
  <name>mapred.job.tracker</name>
  <value>devcluster01:9001</value>
  <description>
    The host and port that the MapReduce job tracker runs at. If
    "local", then jobs are run in-process as a single map and
    reduce task.
    Note 9001 is not the default port.
  </description>
</property>

<property>
  <name>mapred.map.tasks</name>
  <value>2</value>
  <description>
    define mapred.map tasks to be number of slave hosts
  </description>
</property>

<property>
  <name>mapred.reduce.tasks</name>
  <value>2</value>
  <description>
    define mapred.reduce tasks to be number of slave hosts
  </description>
</property>

<property>
  <name>mapred.system.dir</name>
  <value>/nutch/filesystem/mapreduce/system</value>
</property>

<property>
  <name>mapred.local.dir</name>
  <value>/nutch/filesystem/mapreduce/local</value>
</property>

</configuration>
```

The distributed file system has name nodes and data nodes. When a client wants to manipulate a file in the file system it contacts the name node which then tells it which data node to contact to get the file. The name node is the coordinator and stores what blocks (not really files but you can think of them as such for now) are on what computers and what needs to be replicated to different data nodes. The data nodes are just the workhorses. They store the actual files, serve them up on request, etc. So if you are running a name node and a data node on the same computer it is still communicating over sockets as if the data node was on a different computer.

I won't go into detail here about how mapreduce works, that is a topic for another tutorial and when I have learned it better myself I will write one, but simply put mapreduce breaks programming tasks into map operations (a -> b,c,d) and reduce operations (list -> a). Once a probelm has been broken down into map and reduce operations then multiple map operations and multiple reduce operations can be distributed to run on different servers in parallel. So instead of handing off a file to a filesystem node, we are handing off a processing operation to a node which then processes it and returns the result to the master node. The coordination server for mapreduce is called the mapreduce job tracker. Each node that performs processing has a daemon called a task tracker that runs and communicates with the mapreduce job tracker.

The nodes for both the filesystem and mapreduce communicate with their masters through a continuous heartbeat (like a ping) every 5-10 seconds or so. If the heartbeat stops then the master assumes the node is down and doesn't use it for future operations.

The mapred.job.tracker property specifies the master mapreduce tracker so I guess it is possible to have the name node and the mapreduce tracker on different computers. That is something I have not done yet.

The mapred.map.tasks and mapred.reduce.tasks properties tell how many tasks you want to run in parallel. This should be a multiple of the number of computers that you have. In our case since we are starting out with 1 computer we will have 2 map and 2 reduce tasks. Later we will increase these values as we add more nodes.

The dfs.name.dir property is the directory used by the name node to store tracking and coordination information for the data nodes.

The dfs.data.dir property is the directory used by the data nodes to store the actual filesystem data blocks. Remember that this is expected to be the same on every node.

The mapred.system.dir property is the directory that the mapreduce tracker uses to store its data. This is only on the tracker and not on the mapreduce hosts.

The mapred.local.dir property is the directory on the nodes that mapreduce uses to store its local data. I have found that mapreduce uses a huge amount of local space to perform its tasks (i.e. in the Gigabytes). That may just be how I have my servers configured though. I have also found that the intermediate files produced by mapreduce don't seem to get deleted when the task exits. Again that may be my configuration. This property is also expected to be the same on every node.

The dfs.replication property states how many servers a single file should be replicated to before it becomes available. Because we are using only a single server for right now we have this at 1. If you set this value higher than the number of data nodes that you have available then you will start seeing alot of (Zero targets found, forbidden1.size=1) type errors in the logs. We will increase this value as we add more nodes.

Before you start the hadoop server, make sure you format the distributed filesystem for the name node:

```
bin/hadoop namenode -format
```

And check the logs directory looking for errors.

Now that we have our hadoop configured and our slaves file configured it is time to start up hadoop on a single node and test that it is working properly. To start up all of the hadoop servers on the local machine (name node, data node, mapreduce tracker, job tracker) use the following command as the nutch user:

```
cd /nutch/search
bin/start-all.sh
```

To stop all of the servers you would use the following command:

```
bin/stop-all.sh
```

If everything has been setup correctly you should see output saying that the name node, data node, job tracker, and task tracker services have started. If this happens then we are ready to test out the filesystem. You can also take a look at the log files under /nutch/search/logs to see output from the different daemons services we just started.

You might want to look at http://localhost:50070/ with a web browser to confirm that the NameNode is up and running. (Replace localhost with devcluster01 or whatever you main host is called)

You can also look at http://localhost:50030/ to confirm that the JobTracker is up and running. (These ports seem to remain the same no matter that we entered "9000" and "9001" above.

To test the filesystem we are going to create a list of urls that we are going to use later for the crawl. Run the following commands:

```
cd /nutch/search
mkdir urlsdir
vi urls/seed.txt

http://nutch.apache.org
http://apache.org
```

You should now have a urls/seed.txt file with two URLs (one per line) pointing to the Apache Nutch and Apache Software Foundation home sites respectively. Now we are going to add that directory to the filesystem. Later the nutch crawl will use this file as a list of urls to crawl. To add the urls directory to the filesystem run the following command:

```
cd /nutch/search
bin/hadoop dfs -put urls urls
```

You should see output stating that the directory was added to the filesystem. You can also confirm that the directory was added by using the ls command:

```
cd /nutch/search
bin/hadoop dfs -ls
```

Something interesting to note about the distributed filesystem is that it is user specific. If you store a directory urls under the filesystem with the nutch user, it is actually stored as /user/nutch/urls. What this means to us is that the user that does the crawl and stores it in the distributed filesystem must also be the user that starts the search, or no results will come back. You can try this yourself by logging in with a different user and runing the ls command as shown. It won't find the directories because is it looking under a different directory /user/username instead of /user/nutch.

If everything worked then you are good to add other nodes and start the crawl 😉

## Deploy Nutch to Multiple Machines

Along with the new Nutch architecture presented in version 1.3 onwards we no longer need to copy any Nutch jar files and/or configuration to each node in the cluster.

The Nutch job jar you find in $NUTCH_HOME/runtime/deploy is self containing and ships with all the configuration files necessary for Nutch to be able to run on any vanilla Hadoop cluster. All you need is a healthy cluster and a Hadoop environment (cluster or local) that points to the jobtracker.

## Performing a Nutch Crawl

Now that we have the the distributed file system up and running we can peform our fully distributed nutch crawl. In this tutorial we are only going to crawl the two sites we did above as in this tutorial we are not as concerned with someone being able to learn the crawling aspect of nutch as we are with being able to setup the distributed filesystem and mapreduce.

To make sure we crawl only a single site we are going to edit regex-urlfilter.txt file as set the filter to crawl *apache.org (this will permit nutch.apache.org as well):

```
cd /nutch/search
vi conf/regex-urlfilter.txt

change the line that reads:   +^http://([a-z0-9]*\.)*MY.DOMAIN.NAME/
to read:                      +^http://([a-z0-9]*\.)*apache.org/
```

We have already added our urls to the distributed filesystem and we have edited our urlfilter so now it is time to begin the crawl. To start the nutch crawl firstly copy your apache-nutch-${version}.job jar over to $HADOOP_HOME, then use the following command:

```
cd $HADOOP_HOME
hadoop jar apache-nutch-${version}.job org.apache.nutch.crawl.Crawl urls -dir crawl -depth 3 -topN 5
```

We are using the nutch crawl command. The urls dir is the urls directory that we added to the distributed filesystem. The "-dir crawl" is the output directory. This will also go to the distributed filesystem. The depth is 3 meaning it will only get 3 page links deep. There are other options you can specify, see the command documentation for those options.

You should see the crawl startup and see output for jobs running and map and reduce percentages. You can keep track of the jobs by pointing you browser to the master name node:

http://devcluster01:50070

and Mapreduce administration at

http://devcluster01:50030

You can also startup new terminals into the slave machine and tail the log files to see detailed output for that slave node. The crawl will probably take a while to complete. When it is done we are ready to do the search.

## Testing the Crawl

You might want to try some of these commands before doing a search

```
hadoop jar nutch-${version}.job org.apache.nutch.crawl.LinkDbReader crawldb/linkdb -dump /tmp/linksdir
mkdir /nutch/search/output/
bin/hadoop dfs -copyToLocal /tmp/linksdir  /nutch/search/output/linksdir
less /nutch/search/output/linksdir/*
```

Or if we want to look at the whole thing as a text file we might try

```
hadoop jar nutch-${version}.job org.apache.nutch.crawl.LinkDbReader crawldb/linkdb -dump /tmp/entiredump
bin/hadoop dfs -copyToLocal /tmp/entiredump  /nutch/search/output/entiredump
less /nutch/search/output/entiredump/*
```

## Performing a Search

Quite frankly, this tutorial doesn't aspire to provide detail on the ins and out of using Apache Solr, or any other search architecture. If (when running your crawl) you were using the crawl command as above, you could merely specify the Solr URL(s) you wish to use.

Using the search features of the DFS is excellent for doing adhoc searches onf your distributed system, however is generally not advised as it does not scale as your data grows.

## Rsyncing Code to Slaves

Nutch and Hadoop provide the ability to rsync master changes to the slave nodes. This is optional though because it slows down the startup of the servers and because you might not want to have changed automatically synced to slave nodes.

If you do want this capability enabled then below I will show you how to configure your servers to rsync from the master. There are a couple of things you should know first. One, even though the slave nodes can rsync from the master you still have to copy the base installation over to the slave node the first time so that the scripts are available to rsync. This is the way we did it above so that shouldn't require any changeds Two the way the rsync happens is that the master node does an ssh into the slave node and calls bin/hadoop-daemon.sh. The script on the slave node then calls the rsync back to the master node. What this means is that you have to have a password-less login from each of the slave nodes to the master node. Before we setup password-less login from the master to the slaves, now we need to do the reverse. Three, if you have problems with the rsync options (I did and I had to change the options because I am running an older version of ssh), look in the bin/hadoop-daemon.sh script around line 82 for where it calls the rsync command.

So the first thing we need to do is setup the hadoop master variable in the conf/hadoop-env.sh file. The variable will need to look like this:

```
export HADOOP_MASTER=devcluster01:/nutch/search
```

This will need to be copied to all of the slave nodes like this:

```
scp /nutch/search/conf/hadoop-env.sh nutch@devcluster02:/nutch/search/conf/hadoop-env.sh
```

And finally you will need to log into each of the slave nodes, create a default ssh key for each machine and then copy it back to the master node where you will append it to the /nutch/home/.ssh/authorized_keys file. Here are the commands for each slave node, be sure to change the slavenodename when you copy the key file back to the master node so you don't overwrite files:

```
ssh -l nutch devcluster02
cd /nutch/home/.ssh

ssh-keygen -t rsa (Use empty responses for each prompt)
  Enter passphrase (empty for no passphrase):
  Enter same passphrase again:
  Your identification has been saved in /nutch/home/.ssh/id_rsa.
  Your public key has been saved in /nutch/home/.ssh/id_rsa.pub.
  The key fingerprint is:
  a6:5c:c3:eb:18:94:0b:06:a1:a6:29:58:fa:80:0a:bc nutch@localhost

scp id_rsa.pub nutch@devcluster01:/nutch/home/devcluster02.pub
```

Once you have done that for each of the slave nodes you can append the files to the authorized_keys file on the master node:

```
cd /nutch/home
cat devcluster*.pub >> .ssh/authorized_keys
```

With this setup whenever you run the bin/start-all.sh script files should be synced from the master node to each of the slave nodes.

## Conclusion

Although this has been a rather lengthy tutorial, hopefully it has gotten you familiar with both nutch and hadoop. Both Nutch and Hadoop are complicated applications and setting them up as you have learned is not necessarily an easy task. However we hope that this document has helped to make it easier for you.

If you have any comments or suggestions feel free to tell us on user@nutch.apache.org, details for joining our community lists can be found here. If you have questions about Nutch or Hadoop they should be addressed to their respective mailing lists. Below are general resources that are helpful with operating and developing Nutch and Hadoop.

## Updates

- I don't use rsync to sync code between the servers any more. Now I am using expect scripts and python scripts to manage and automate the system.
- I use distributed searching with 1-2 million pages per index piece. We now have servers with multiple processors and multiple disks (4 per machine) running multiple search servers (1 per disk) to decrease cost and power requirements. With this a single server holding 8 million pages can serve 10 queries a second constant.

## Resources

Hadoop Quickstart: http://hadoop.apache.org/common/docs/current/quickstart.html

Google MapReduce Paper:
If you want to understand more about the MapReduce architecture used by Hadoop it is useful to read about the Google implementation.

http://labs.google.com/papers/mapreduce.html

Google File System Paper:
If you want to understand more about the Hadoop Distributed Filesystem architecture used by Hadoop it is useful to read about the Google Filesystem implementation.

http://labs.google.com/papers/gfs.html

Building Nutch - Open Source Search:
A useful paper co-authored by Doug Cutting about open source search and Nutch in paticular.

http://www.acmqueue.com/modules.php?name=Content&pa=showpage&pid=144