



# Deploying Apache Hadoop with SmartFrog

**Steve Loughran**  
**Julio Guijarro**



Presented at the August 2008 Hadoop UK User Group meeting

# Our other computer really is a datacentre



This is us in front of our datacentre. It's in HPLabs, it has got storage, but we don't get unrestricted use to it. We have to share it with all those other people doing things they think are interesting. So, we have to be agile: provide machines on demand, maybe even do billed storage. Because the machines aren't free, and anyone that has a machine they aren't using is depriving CPU time from those who could do something interesting with the same boxes.



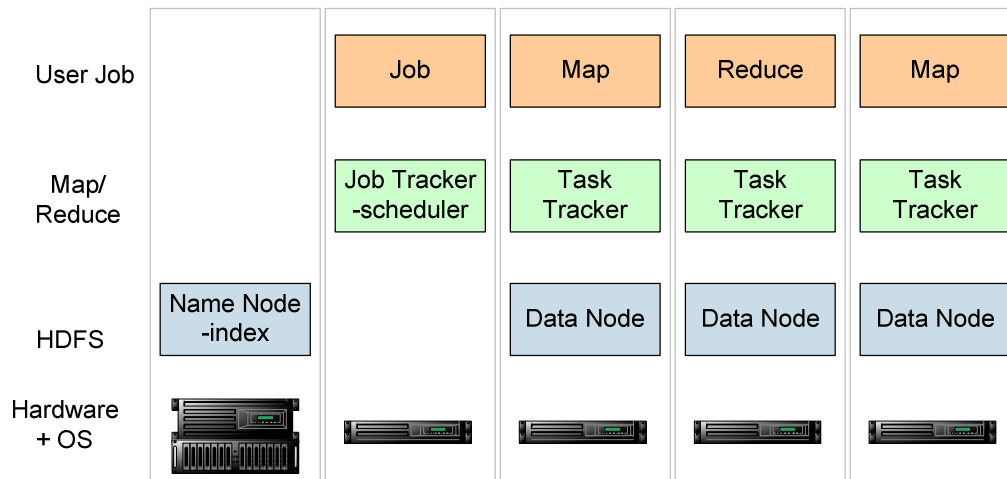
- Make Hadoop deployment *agile*
- Integrate with dynamic cluster deployments

12 June 2008



We may have our own cluster, but it isn't a stable one. We need to adapt to what gets provided to us.

# Basic problem: deploying Hadoop



*one namenode, 1+ Job Tracker, many data nodes and task trackers*

12 June 2008



This is the base problem. Bringing the cluster up and keeping it alive. oh, and handling outages. The namenode is a SPOF for the cluster; a job tracker will kill the job and anything depending on it. Datanodes are valued but replicated, task trackers are expendable.

# Deployment tasks

- Hardware: servers, LAN, routers
- OS: install, update
- OS config: host table, SSH keys, NFS
- JVM: installation, updates
- Hadoop: JARs
- Hadoop: configuration XML files
- Diagnosing failures. Is it the XML file or the CAT-5?

12 June 2008



Here are the things you have to do during deployment. Get the hardware together, bring up the OS, install java and the Hadoop JARs.

Hard parts: getting the OS config files correct and synchronized. Getting the hadoop XML config files correct and synchronized.

When the config files are wrong (and sometimes when the router is playing up), the cluster doesn't work. Finding out the root cause and fixing it can cause many hours of fun.

# The hand-managed cluster

- Manual install onto machines
- SCP/FTP in Hadoop zip
- copy out hadoop-site.xml and other files
- edit /etc/hosts, /etc/rc5.d, SSH keys ...
- Installation scales  $O(N)$
- Maintenance, debugging scales worse

12 June 2008



We regularly see this discussed on the list -the problem of keeping /etc/hosts synchronised, pushing out site stuff by hand. Don't do it. Do it once to know what the files are, then stop and look for a better way.

# The locked-down cluster

- PXE Preboot of OS images
- RedHat Kickstart to serve up (see [instalinux.com](http://instalinux.com))
- Maybe: LDAP to manage state, or custom RPMs

## Requires:

uniform images, central LDAP service, good ops team, stable configurations, home-rolled RPMs

12 June 2008



This is how Yahoo! appear to run the largest Hadoop clusters in production. There's a good set of slides on this from ApacheCon.

LDAP would seem to add a SPOF, but you can have HA directory services. Also, if DNS is integrated with LDAP, the loss of DNS/LDAP is equally drastic, so availability is not made any worse than not using LDAP.

You also need your own RPMs, as you will need to push out all binaries in RPM format. This often scares Java developers, but isn't that hard once you start on it. Hard to test though.

Now, in this world, where to the hadoop XML files come from?

# CM-tool managed cluster

## Configuration Management tools

- State Driven: observe system state, push it back into the desired state
- Workflow: apply a sequence of operations to change a machine's state
- Centralized: central DB in charge
- Decentralized: machines look after themselves

CM tools are the only way to manage big clusters

12 June 2008



CM tools -of any kind- are the only sensible way to manage big clusters. Even then, there are issues about which design is best, here are some of the different options. What I'm going to talk about is decentralized and mostly state driven, though it has workflows too. Some options. bzcfig2, cfengine, puppet, LCFG, and SmartFrog.



## SmartFrog - HPLabs' CM tool

- Language for describing systems to deploy  
—everything from datacentres to test cases
- Runtime to create *components* from the model
- Components have a lifecycle
- LGPL Licensed, Java 5+  
<http://smartfrog.org/>

12 June 2008



This is our CM tool, used inside HP and with the core app and components LGPL licensed for anyone to use.

# Model the system in the SmartFrog language

```
TwoNodeHDFS extends OneNodeHDFS {  
  
    localDataDir2 extends TempDirWithCleanup {  
  
    }  
  
    datanode2 extends datanode {  
        dataDirectories [LAZY localDataDir2];  
        dfs.datanode.https.address "https://localhost:0";  
    }  
}
```

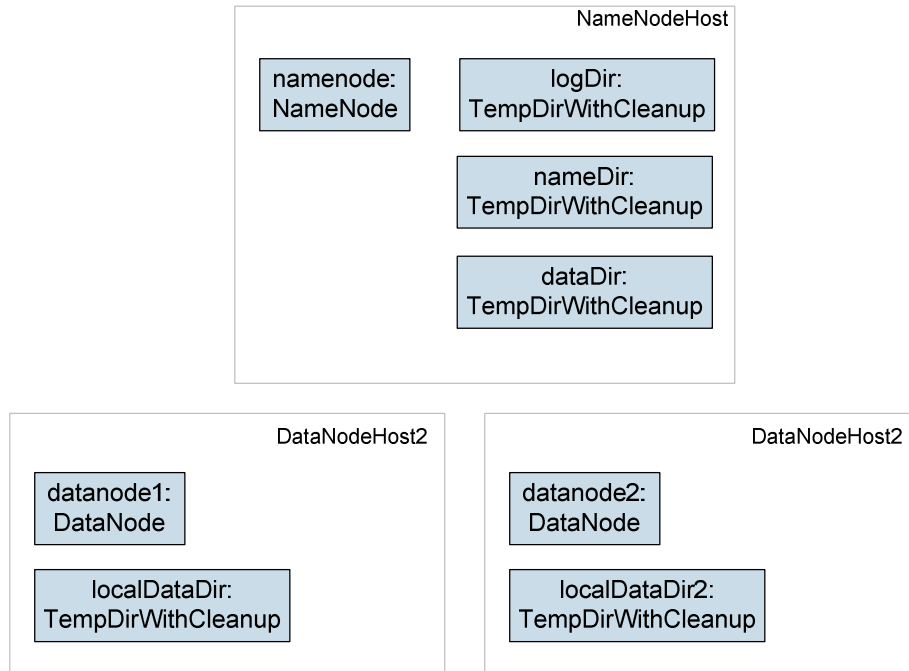
Inheritance, cross-referencing, templating

12 June 2008



The language is used to describe the system configuration.

# The runtime deploys the model



12 June 2008



This description is then instantiated into a tree of components, which is then deployed across the chosen hosts/processes. You can point the deployment descriptor at a specific node, and/or pick different nodes in the deployment descriptor itself. You can also set up a farm of unpurposed machines which pick up deployment descriptors that they can apply. This lets you have a farm of machines that can be repurposed on the fly.

## Steps to deployability

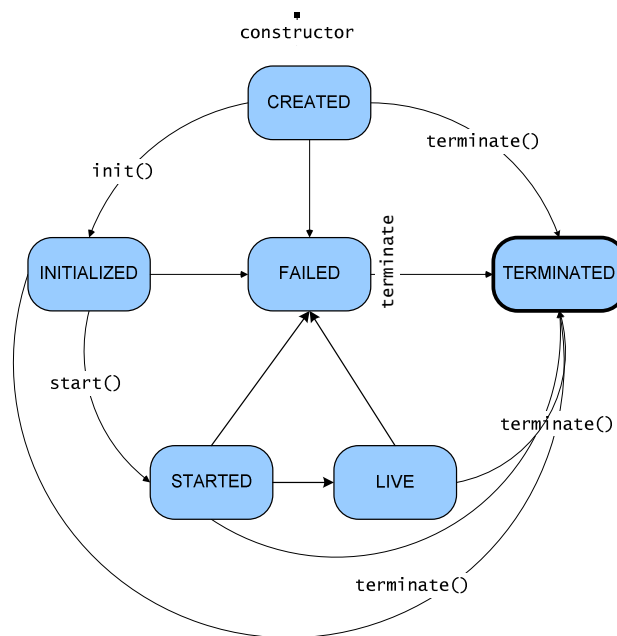
1. Configure Hadoop from an SmartFrog description
2. Write components for the Hadoop nodes
3. Write the functional tests
4. Add *workflow* components to work with the filesystem; submit jobs
5. Get the tests to pass

12 June 2008



Now, how to get Hadoop working under SmartFrog? Coding, that's how

# HADOOP-3628: A lifecycle for services



12 June 2008



This is what the current proposed lifecycle for hadoop services: namenode, datanodes, 2ary namenodes, trackers, etc. Things you start and stop and wait for starting. It's very similar to the SmartFrog one, except that it has a stricter differentiation between started and live. Your code that starts up has to decide when it is ready and flip the switch to say "live"

## Base Service class for all nodes

```
public class service {  
    public void init() throws IOException;  
    public void start() throws IOException;  
    void ping() throws IOException;  
    void terminate();  
    State getLifecycleState();  
    public enum State {  
        CREATED,  
        INITIALIZED,  
        STARTED,  
        LIVE,  
        FAILED,  
        TERMINATED  
    }  
}
```

12 June 2008



Here is the proposed base class for services. It has methods to move it through the states, and a way to query it

# Subclasses implement inner Methods

```
    / ** start the namenode */  
    protected void innerStart() throws IOException {  
        initialize(bindAddress, getConf());  
        setServiceState(ServiceState.LIVE);  
    }  
    / ** namenode health */  
    public void innerPing() throws IOException {  
        if (namesystem == null) {  
            throw new LivenessException("No name system");  
        }  
        namesystem.ping();  
    }
```

12 June 2008



This is a namenode with a lifecycle interface patched in.

## Health and liveness: ping()

```
/* ** datanode health */
public void innerPing() throws IOException {
    if (ipcServer == null) {
        throw new LivenessException("No IPC Server running");
    }
/*
    if (dnRegistration == null) {
        throw new LivenessException("Not registered to a namenode");
    }
*/
}
```

what should we do here?

12 June 2008



This is the ping of the datanode. For a while I had it failing if the namenode wasn't there, but it was failing early on during startup. We really need to decide when a datanode and name node consider themselves healthy.



## Liveness issues

- If a datanode cannot see a namenode, is it still healthy?
- If a namenode has no datanodes, is it healthy?
- Should we propagate block checker failures?
- Can we run test jobs through the Task Trackers?

How unavailable should the nodes be before the cluster is declared "unhealthy"?

12 June 2008



This is where ping() gets complex. How do you define healthy in a cluster where the overall system availability is not 'all nodes are up' but 'we can get our work done'.

# Proposal

- Have specific exceptions for no-workers, no-master node; include time they were last seen
- Let the callers interpret those failures.
- A service failing a ping() should not go into FAILED state immediately; it may be transient.

Push the problems up!

12 June 2008



This is what I'm thinking here. We have standard exceptions for no workers, no masters, maybe no disk space, and throw them when there are problems. Its up to the management tools to decide the implications of that, because everyone has a different set of priorities.

# SmartFrog wrapper components

```
public class TaskTrackerImpl extends HadoopServiceImpl
    implements HadoopCluster {

    public synchronized void sfStart() throws SmartFrogException,
        RemoteException {
        super.sfStart();
        ManagedConfiguration configuration;
        configuration = createConfiguration();
        try {
            TaskTracker tracker = new TaskTracker(configuration);
            deployService(tracker, configuration);
        } catch (IOException e) {
            throw SFHadoopException.forward(ERROR_NO_START,
                e, this, configuration);
        }
    }
}
```

12 June 2008



This is part of the task implementation; it is a SF component that is deployed where told to; it binds to its configuration and then creates an Extended task tracker

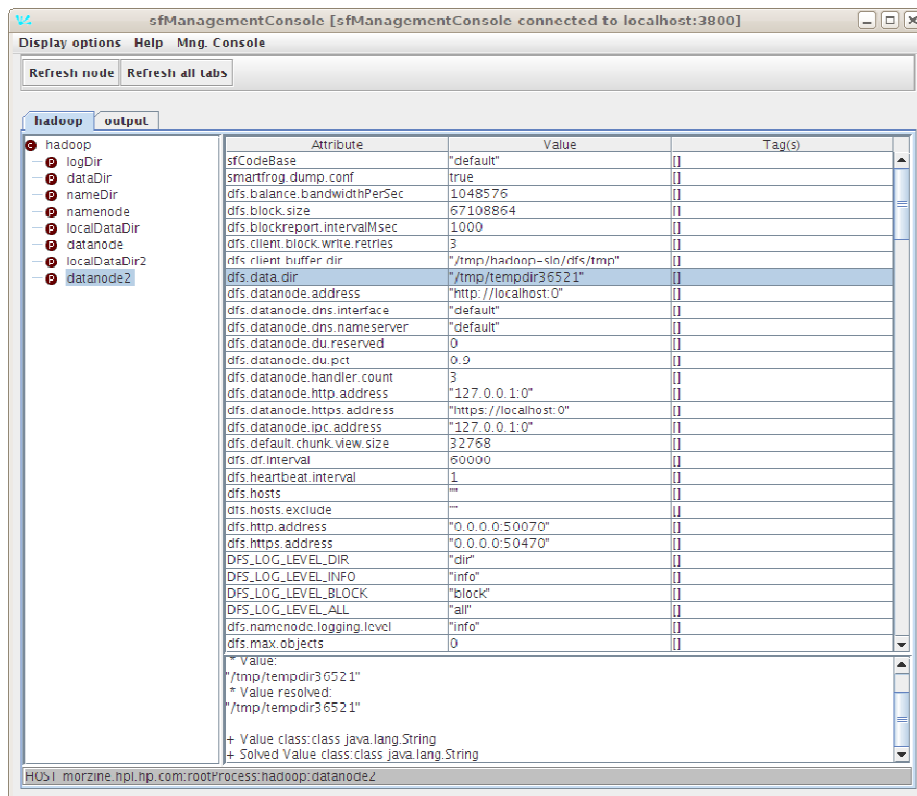
# Replace hadoop-\*.xml with .sf files

```
NameNode extends FileSystemNode {  
    sfClass "org.smartfrog.services.hadoop.components.namenode.NamenodeImpl";  
    nameDirectories TBD;  
    dataDirectories TBD;  
    logDir TBD;  
    dfs.http.address "http://localhost:8021";  
    dfs.namenode.handler.count 10;  
    dfs.namenode.decommission.interval (5 * 60);  
    dfs.name.dir TBD;  
    dfs.permissions.supergroup "supergroup";  
    dfs.upgrade.permission "0777"  
    dfs.replication 3;  
    dfs.replication.interval 3;  
    . . .  
}
```

12 June 2008



One nice feature here is that we've eliminated the various hadoop- xml files.



Result: we can bring up an HDFS cluster

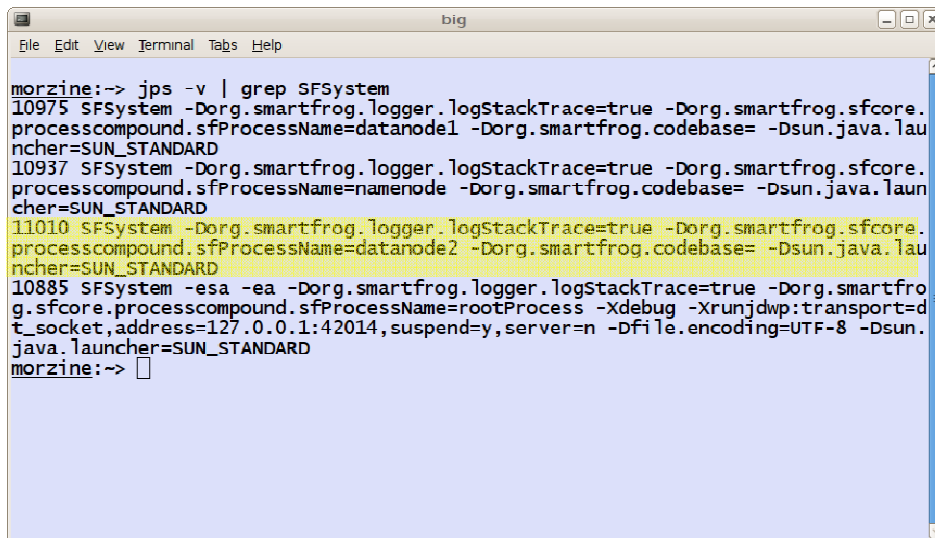


12 June 2008

Here's a screenshot of the management GUI. A live cluster on a single machine.

## in different processes and hosts

```
datanode2 extends datanode {  
    sfProcessName "datanode2";  
    dataDirectories [LAZY localDataDir2];  
}
```



```
File Edit View Terminal Tabs Help  
morzine:~> jps -v | grep SFSsystem  
10975 SFSsystem -Dorg.smartfrog.logger.logStackTrace=true -Dorg.smartfrog.sfcore.  
processcompound.sfProcessName=datanode1 -Dorg.smartfrog.codebase= -Dsun.java.lau  
ncher=SUN_STANDARD  
10937 SFSsystem -Dorg.smartfrog.logger.logStackTrace=true -Dorg.smartfrog.sfcore.  
processcompound.sfProcessName=namenode -Dorg.smartfrog.codebase= -Dsun.java.lau  
ncher=SUN_STANDARD  
11010 SFSsystem -Dorg.smartfrog.logger.logStackTrace=true -Dorg.smartfrog.sfcore.  
processcompound.sfProcessName=datanode2 -Dorg.smartfrog.codebase= -Dsun.java.lau  
ncher=SUN_STANDARD  
10885 SFSsystem -esa -ea -Dorg.smartfrog.logger.logStackTrace=true -Dorg.smartfro  
g.sfcore.processcompound.sfProcessName=rootProcess -Xdebug -Xrunjdwp:transport=d  
t_socket,address=127.0.0.1:42014,suspend=y,server=n -Dfile.encoding=UTF-8 -Dsun.  
java.launcher=SUN_STANDARD  
morzine:~> █
```

12 June 2008



Here's a jps -v dump of the machine, showing that I'm running four processes here. Every node is in its own JVM, by specifying the name of a different process. This gives us better detection/handling of JVM failure.

# Aggregated logs

```
08/08/18 15:42:02 INFO datanode.DataNode :  
  DatanodeRegistration(127.0.0.1:46504, storageID=DS-1347621537-127.0.1.1-  
  46504-1219070522618, infoPort=42568, ipcPort=53148)In DataNode.run, data =  
  FSDataset{dirpath='/tmp/tempdir36763/current'}  
  
08/08/18 15:42:02 INFO datanode.DataNode : State change: DataNode is now LIVE  
  
08/08/18 15:42:02 INFO datanode.DataNode : BlockReport of 0 blocks got  
  processed in 12 msec  
  
08/08/18 15:42:02 INFO datanode.DataNode : Starting Periodic block scanner.  
  
08/08/18 15:42:03 INFO datanode.DataNode : BlockReport of 0 blocks got  
  processed in 1 msec  
  
08/08/18 15:42:04 INFO hdfs.StateChange : BLOCK* NameSystem.registerDatanode:  
  node registration from 127.0.0.1:39522 storage DS-1890915122-127.0.1.1-  
  39522-1219070524453  
  
08/08/18 15:42:04 INFO net.NetworkTopology : Adding a new node: /default-  
  rack/127.0.0.1:39522  
  
08/08/18 15:42:04 INFO HOST morzine.hp1.hp.com:rootProcess:hadoop - DataNode  
  deployment complete: service is now LIVE
```

12 June 2008



We also get the logs back. This is good and bad. Good, because they all come out in one place. Bad, as they all look the same. Which datanode is which?

# Workflow operations

Dfs manipulation: DfsCreateDir, DfsDeleteDir,  
DfsListDir, DfsPathExists, DfsFormatFilesystem, ...

Dfs I/O: DfsCopyFileIn, DfsCopyFileOut

Soon: Job Submission

```
touch extends DfsCreateFile {  
  cluster LAZY PARENT:namenode;  
  path "/test-filename";  
  text "this is a very small file for Hadoop";  
}
```

12 June 2008



If you can bring up a cluster dynamically, it has no data in it. You need ways of getting data in. We have some components that can manipulate the filesystem, and copy data in and out. We also have the ability to assert that a file exists (and has a size within a given range), primarily for testing.



# What can this do?

- Set up and tear down Hadoop clusters
- Workflow for manipulating the filesystem (files in, out, maintenance operations)
- Management console view of the whole system
- Explore different cluster configurations
- Automate failover policies

## *Agile deployment of Hadoop Clusters*

12 June 2008



What does all this let us do? It lets us automate complete cluster deployment onto statically or dynamically provisioned servers. It gives a management overview of the cluster. And - and this is interesting- you can contain the cluster deployment inside other .SF files that contain components to keep the entire cluster up, even as bits go up and down.

## Status as of August, 2008

- SmartFrog code in SourceForge SVN
- HADOOP-3628 patches Hadoop source
- HDFS clusters can be brought up
- Job and Task Trackers next

12 June 2008



Where are we right now?

I have an ongoing patch that retrofits a lifecycle to the main hadoop components (namenode, jobtracker, task tracker, datanode; all but secondary name node). The HDFS services are up and running; the job/task trackers will follow once I've written the tests. In the SmartFrog SVN repository in SourceForge we have our classes, but this code only builds against patched hadoop code. There's an ivy build that can publish hadoop to the local ivy cache for an integrated build).

# TODO list

- Work on ping(): policy, web page
- Proper configuration back-end API
- Get changes into Hadoop 0.19 or later
- Generate RPMs with SmartFrog and Hadoop
- Full scale test deployments
- on HP and EC2 infrastructure

12 June 2008



This is some of my TODO list; look at [jira.smartfrog.org](http://jira.smartfrog.org) and [issues.apache.org](http://issues.apache.org) to find stuff by name.

- I want to get the ping() semantics nailed down and implemented
- I want a proper config API instead of just subclassing JobConf
- I want these changes into Hadoop
- We generate RPMs for SmartFrog; I will add an RPM that contains the hadoop JARs too.
- I want to roll these out to real 100+ node clusters, both HP and EC2.



# Cluster design

- Name node: high end, RAID + backups.  
-this is the SPOF. Nurture it.
- Secondary name node —as name node
- Data nodes: mid-range multicore blade systems  
2 disks/core. No need for RAID.
- Job tracker: standalone server
- task trackers: on the data servers
- Secure the LAN
- Expect failures of everything

12 June 2008



Name nodes are a point of failure. Look after them.

Data nodes have to be viewed as transient. They come down, they come back up. If they aren't stable enough, you have problems as you won't ever get work done.

Some organisations track hard disks. They matter more than CPUs, when you think about it. Keep track of every disk's history, its batch number, the # of power cycles of every machine it is in, etc, etc. etc. Run the logs through Hadoop later to look for correlations.

Trusted machines: until Hadoop has real security, you have to trust everything on the net that can talk to the machines. Either bring them all up on a VPN (performance!) or a separate physical network