SUSE Linux Enterprise Server 10

Exploring the
High-Availability Storage Foundation

September 2006
# Table of Contents

**Introduction** ................................................................................................................................. 5  
  - About ............................................................................................................................................. 5  
  - Disclaimer ................................................................................................................................... 5  
  - Contributors .................................................................................................................................. 5  
  - Revisions ....................................................................................................................................... 5  

**Preparation** ................................................................................................................................. 6  
  - Description of the used hardware ............................................................................................... 6  
  - What else do you need ? ............................................................................................................. 7  
  - Network configuration ............................................................................................................... 7  
  - Overview of the setup .............................................................................................................. 7  

**Installing SLES 10** ................................................................................................................. 9  
  - Technology overview .................................................................................................................. 9  
  - Installing node1 ........................................................................................................................... 9  
  - Installing SLES 10 on node2 (differences) ............................................................................... 29  
  - Tuning the installed system for the HASF ............................................................................. 30  
  - Some things to check first ........................................................................................................... 37  

**Heartbeat 2** ............................................................................................................................. 39  
  - Technology overview .................................................................................................................. 39  
  - Configuring Heartbeat 2 on node1 ............................................................................................ 43  
  - Propagating the Heartbeat 2 configuration to node2 .............................................................. 45  
  - Creating a test cluster resource from node1 ......................................................................... 46  
  - Removing the test cluster resource ....................................................................................... 52  

**Creating a SAN with iSCSI** ................................................................................................. 55  
  - Technology overview .................................................................................................................. 55  
  - Adding a new partition on the system disk of node2 .............................................................. 57  
  - Configuring iSCSI target on node2 .......................................................................................... 60  
  - Configuring iSCSI initiator on node1 ....................................................................................... 64  
  - Partitioning the iSCSI Lun via node1 ...................................................................................... 68  
  - Configuring iSCSI initiator on node2 (differences) ............................................................... 76
Oracle Cluster File System 2.................................................................77
  Technology overview.............................................................................77
  Initializing the native OCFS2 stack on node1........................................81
  Enabling user space OCFS2 cluster membership setting..........................87
  Creating the OCFS2 file systems............................................................90
  More about user space cluster membership instructions..........................91
  Integrating OCFS2 in the Heartbeat 2 cluster.......................................94
  Testing the OCFS2 Heartbeat integration.............................................104

The Xen hypervisor..................................................................................107
  Technology overview.............................................................................107
  Tuning the bootloader for Xen..................................................................108
  Changing the sync_mode of the loop device.............................................110
  Creating a virtual machine on node1.......................................................111
  Switch the loop module on node1 to protected mode............................134
  Check if node2 has access to the virtual machine ....................................134

Virtual machines as cluster resources.....................................................137
  Technology overview.............................................................................137
  Configuring the virtual machine as a cluster resource.........................138
  Ordering resource startup in the cluster.................................................140
  Testing the high-availability of the virtual machine...............................141

Appendix A : FAQ..................................................................................143
  Why do I explicitly set default cluster configuration values in cibbootstrap.xml ?.................................................................143

Appendix B : Bugs..................................................................................145
  Where can I log bugs ?............................................................................145
  I have an account now, how do I log bugs ?...........................................145
  Adding logs............................................................................................146
Introduction

About

This document describes how to create a demo setup of Novell's SUSE Linux Enterprise Server 10 “High-Availability Storage Foundation” (HASF), a combination of iSCSI, OCFS2, Xen, Heartbeat 2 and EVMS2. We strongly advise reading each chapter at least once, before performing the actions described in it.

Disclaimer

The author(s) will not accept any responsibility for any incorrect information within this document, nor for any damage it might cause when applied.

Do not forget most of the HOWTO documents are the result of volunteers who devote time and resources in their creations. The respective authors are willing to assist you in any way they can but only if time permits them.

Contributors

Revisions
Preparation

Description of the used hardware

The setup this document is describing, was done using two commodity “off the shelf” laptops with each 1GB of RAM. The laptops were connected using a 1 Gigabit ethernet switch. Below the output of the “lspci” command as performed on one of the laptops:

```
00:00.0  Host bridge: Intel Corporation 82855PM Processor to I/O Controller (rev 03)
00:01.0  PCI bridge: Intel Corporation 82855PM Processor to AGP Controller (rev 03)
00:1d.0  USB Controller: Intel Corporation 82801DB/DBL/DBM (ICH4/ICH4-L/ICH4-M) USB UHCI Controller #1 (rev 01)
00:1d.1  USB Controller: Intel Corporation 82801DB/DBL/DBM (ICH4/ICH4-L/ICH4-M) USB UHCI Controller #2 (rev 01)
00:1d.2  USB Controller: Intel Corporation 82801DB/DBL/DBM (ICH4/ICH4-L/ICH4-M) USB UHCI Controller #3 (rev 01)
00:1d.7  USB Controller: Intel Corporation 82801DB/DBM (ICH4/ICH4-M) USB2 EHCI Controller (rev 01)
00:1e.0  PCI bridge: Intel Corporation 82801 Mobile PCI Bridge (rev 81)
00:1f.0  ISA bridge: Intel Corporation 82801DBM (ICH4-M) LPC Interface Bridge (rev 01)
00:1f.1  IDE interface: Intel Corporation 82801DBM (ICH4-M) IDE Controller (rev 01)
00:1f.5  Multimedia audio controller: Intel Corporation 82801DB/DBL/DBM (ICH4/ICH4-L/ICH4-M) AC’97 Audio Controller (rev 01)
00:1f.6  Modem: Intel Corporation 82801DB/DBL/DBM (ICH4/ICH4-L/ICH4-M) AC’97 Modem Controller (rev 01)
01:00.0  VGA compatible controller: ATI Technologies Inc Radeon R250 Lf [FireGL 9000] (rev 01)
02:00.0  Ethernet controller: Broadcom Corporation NetXtreme BCM5702X Gigabit Ethernet (rev 02)
02:01.0  CardBus bridge: O2 Micro, Inc. 02711EC1 SmartCardBus Controller (rev 20)
02:01.1  CardBus bridge: O2 Micro, Inc. 02711EC1 SmartCardBus Controller (rev 20)
02:03.0  Network controller: Intel Corporation PRO/Wireless LAN 2100 3B Mini PCI Adapter (rev 04)
```
What else do you need?

Besides hardware, you need a copy of SUSE Linux Enterprise Server 10. You can download an evaluation version on http://www.novell.com. Alternatively openSUSE 10.1 can be used, but no support will given for setups using that SUSE version.

Network configuration

node1 (laptop one):
- 192.168.0.1 (static ip)
- default route to internet: 192.168.0.254 (not necessary)

node2 (laptop two):
- 192.168.0.2 (static ip)
- default route to internet: 192.168.0.254 (not necessary)

Overview of the setup

The next page contains a high-level graphical overview of the “next-generation data center”. Virtualization plays a key role, in the storage layer as well as in the systems layer. Virtual machines become the units which run services, and they can be freely migrated and/or made highly available, careless about what particular piece of hardware they are running on, careless about where exactly the data they require resides in the Storage Area Network (SAN).

This document describes how to create the foundation of this environment: at the end of this document, a virtual machine will be running in a cluster, and will be restarted by that cluster when necessary.

This version of the document does not include instructions on how to set up the following servers in the following diagram:
- Image creator (intelligent image delivery server)
- Image store (intelligent image warehouse)
- ZENworks Linux Management 7 (patch delivery to physical / virtual machines)
- Nagios (monitoring of physical / virtual machines)
This document describes how to set up the area contained in the dotted line.
Installing SLES 10

Technology overview

Although our physical machines will perform different roles in the setup, we start by giving them a nearly identical base installation. The most important part of the installation is to leave at least 6GB of free, unpartitioned space on each machine.

Certain settings, such as ip address, have to differ on both machines for obvious reasons. We will start with the screenshots of the installation of node1, followed by screenshots only of the differences for node2.

Installing node1

Pop the DVD in the drive, boot from it, and choose “Installation”.

Revision: 1.1
Choose your language and accept the license agreement which is displayed to you.
Choose “New Installation” and set the correct date and time.
In the “Installation Settings” window, click on the “Expert” tab.

Select “Partitioning” and in the next window, “Create Custom Partition Setup”.
Choose “Custom Partitioning”. Remove all partitions on your disk (in this case /dev/hda) with “Delete”.

Revision: 1.1
Use the “Create” button to create a swap partition with the size of the RAM – in our case 1 GB.
Create a “/” partition. Leave at least 6 GB on the disk unused. In our case we create a “/” partition of 15GB and leave around 21 GB free.
We need to install some additional software ... so go into “Software”.

---

Revision: 1.1
Make sure to select “Xen” and “High availability”. Then click “Accept”.

Confirm the “Installation settings” with “Accept”.

Revision: 1.1
Let the installer do its work. The system will reboot when all packages are installed.
After the first reboot, the second phase of the installation starts.

Hostname “node1”, domain “novell.com”. Fill in “novell” as root password.
We disable the firewall for now, it makes things easier at first.
Make sure to set up a static ip address for node1: go into “Network Interfaces”.
Via “Edit”, give node1 ip address 192.168.0.1 and netmask 255.255.255.0.
"Next", skip the internet test for now, “Next”.
Accept the proposal for CA and OpenLDAP ... press “Next”, “Next”.
Create a user “novell” with password “novell”. Accept the warnings with “Yes”.
Reading the release notes is a good idea. Accept the detected hardware.
Uncheck “Clone This System for Autoyast”. The installation is now finished.
Log on with user “root” and password “novell”.

---

The High-Availability Storage Foundation

SUSE Linux Enterprise Server 10
Installing SLES 10 on node2 (differences)

The hostname for node2 is "node2", the ip address is "192.168.0.2".
Tuning the installed system for the HASF

In this section we do some preliminary work. Do not skip this, as the changes we will perform are absolutely necessary for certain elements in the stack to work properly.

Configuring hostname resolution

First of all we need to make sure that node1 knows who node2 is and vice versa, by modifying the /etc/hosts files on both machines. We use the SUSE Linux “YaST” configuration tool to make the changes. Open YaST by clicking on the “Computer” icon in the lower left corner of the screen, which opens the main GNOME panel. Now click on the YaST icon. In the left YaST panel, select “Network Services” and then in the right panel the “Hostnames” module.

In the “Host Configuration” window, click the “Add” button and fill in the necessary fields in the pop-up (the data for the other node). “Hostname” is the fully qualified domain name on SUSE Linux. Fill in the short hostname as the alias. Finish with “Ok” in the pop-up, “Finish” in the “Host Configuration” module and “File -> Quit” in the YaST main window. Below you see a screen shot showing how to add the data for node2 on node1. Do this on both systems, but of course each time with different data (node2 data on node1, node1 data on node2).

“Hostname” is the FQDN on SUSE Linux. Fill in the short name as the alias.
Preparing the NTP configuration

The time on the two physical machines needs to be synchronized. Several components in the HASF stack require this. We will configure node1 to be a NTP time server, used by node2 as its time synchronization source. NTP requires the time on both nodes to be already roughly in sync in order to start. A fast way to manually sync the time is by using the “date” command and “ssh” in a terminal window like described below. We set the time on node2 close to what it is on node1. Alternatively YaST can be used via “System” and then “Date and Time”.

HINT How do I open a terminal?

Right-click on the desktop background. In the menu which appears, select “Open Terminal”.

```
node1:~ # date
Thu Jun 22 23:05:00 MDT 2006
node1:~ # date +%m%d%H%M
06222305
node1:~ # ssh node2 "date 06222305"
The authenticity of host 'node2 (192.168.0.2)' can't be established.
RSA key fingerprint is
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'node2,192.168.0.2' (RSA) to the list of known hosts.
Password: novell
Thu Jun 22 23:05:00 MDT 2006
```

Do not use “06222305” in the ssh command, but use the value you just got from the “date +%m%d%H%M” command.
Configuring NTP on node1

With the time on both nodes now more or less in sync, we can set up node1 as a NTP time server. Again we do this via YaST, via “Network Services” and then “NTP Client”. Set the NTP daemon to start upon boot (click “Continue” in the warning pop-up), as shown below, and then go to “Complex Configuration”. Via “Edit” -> “Driver Calibration” you can then set the stratum of the local clock to e.g. 5.

By doing this, you tell NTP server to use the local hardware clock as its source of time. Of course the time of our local hardware clock might not be officially correct, but we just want that there is time synchronization between node1 and node2. The actual time on both systems does not need to be “correct” at all, it just needs to be the same.

When configuring node1 is done, it’s very easy to configure node2 to use node1 as its time server. We also do this with YaST. Below you find screenshots of the procedures that need to be performed on node1 and node2.

On node1, open the YaST “NTP client” module to configure the time server.

Select “During Boot”, click “Continue” on the warning, and then go to “Complex Configuration”.

Revision: 1.1
Select the “Local Clock” and click “Edit”.
Go to “Driver Calibration” to increase the stratum of your local clock from “10” to “5” (a lower number is a higher stratum). Click “Next”.
In the “Local Reference Clock” section, confirm your changes with “Ok”.

Click “Finish”. Configuring node1 is done, it now has a running time server.
Configuring NTP on node2

On node2 we now must configure the NTP client to use node1 as its time source. Open the “NTP Client” YaST module on node2, set it to start “During Boot” like you did on node1, accept the warning again with “Continue”, and fill in the IP address of node1 (192.168.0.1) in the text field called “Address”. Then click “Finish”.

In case you click on the “Test” button, getting a warning at this point is normal.

Configure node2 as an NTP client of node1.

Now reboot both machines via “Computer” -> “Log Out” -> “Ok” and then “Reboot” at the bottom of the login screen (this requires filling in the root password). After rebooting, proceed with the next section. We only reboot because we want to make sure that on future reboots, all services configured so far are correctly restarted.
Some things to check first

At this point you should have two nodes which, after rebooting, come up with synchronized time and which can resolve each others hostname properly. Before we go on, let's do some sanity checks to make sure all of this is working properly. Don't skip these simple tests: it makes no sense to go on if the two checks that follow don't produce adequate results.

Check one: in a terminal, can your systems ping each other by using the hostname of the other node?

On node1:

```
node1:~ # ping node2
PING node2.novell.com (192.168.0.2) 56(84) bytes of data.
64 bytes from node2.novell.com (192.168.0.2): icmp_seq=1 ttl=64 time=5.61 ms
```

On node2:

```
node2:~ # ping node1
PING node1.novell.com (192.168.0.1) 56(84) bytes of data.
64 bytes from node1.novell.com (192.168.0.1): icmp_seq=1 ttl=64 time=0.236 ms
```

Check two: check if time synchronization is running correctly. For this check we can use the command “ntpq -p” on both nodes. When run in a terminal, it should produce results similar to what you see below, indicating that node2 is actually using node1 as a time server. Be aware that NTP needs quite some time to stabilize after starting, so it might take a (rather long) time before you see the leading stars ( * ) appearing in the output. That is absolutely normal.
On node1:

```
node1:~ # ntpq -p
      remote  refid  st t when poll reach delay  offset jitter
==============================================================================
*LOCAL(0)   LOCAL(0)  5 1 24 64 377  0.000  0.000  0.001
```

On node2:

```
node2:~ # ntpq -p
      remote  refid  st t when poll reach delay  offset jitter
==============================================================================
LOCAL(0)   LOCAL(0)  10 1 9  64 377  0.000  0.000  0.001
*node1.novell.co LOCAL(0)  6 u 11 64 377  0.297 -16.553  9.474
```

If the output you get from running these commands does not contain the leading stars (*) yet, then don’t worry and just go on with the next sections. After some time you should see the stars appearing.
Heartbeat 2

Technology overview

Heartbeat 2 is the open source clustering software which is shipped with SUSE Linux Enterprise Server 10. Today clustering software is usually used to keep services (in the form of processes) highly available. Starting with SLES 10, it becomes possible to make virtual machines (containing these services) highly available, using Heartbeat 2.

Setting up Heartbeat 2 in its simplest form is rather straightforward. We will start with a minimal setup, and then test that setup by creating a demo cluster resource: a resource which contains an IP address. Once we have that working, we can safely proceed.

In this first section we give a brief overview of the Heartbeat 2 components. We stay “high-level”: the goal is providing enough information to understand the steps that need to be taken in order to make a virtual machine highly available. For more detailed information about heartbeat please see [http://www.Linux-ha.org/v2](http://www.Linux-ha.org/v2).

Heartbeat 2 (HB 2) has a layered architecture (see image below). The basis of the stack is formed by the “messaging/infrastructure layer”, also known as the “heartbeat layer”, which in fact contains a lot of reused code from the Heartbeat 1 project. This layer contains components that send out the heartbeat messages, (containing the “I’m alive” signals), as well as various other information.

In the Heartbeat 1 project the heartbeat program was responsible for almost all cluster-related tasks. However, in Heartbeat 2, much of those tasks have been moved up the stack, in various new layers. Right above the “messaging/infrastructure layer” we now find the “membership layer”, which in fact is responsible for synchronizing the view of which nodes are part of the cluster, among its members.

It performs this task based on the information it gets from the heartbeat layer. The logic that takes care of this task is contained in the “Cluster Consensus Membership” service, which can provide an organized cluster topology overview (node-wise) to cluster components that are higher in the stack.
Above the “membership layer”, we find the “resource allocation layer”. This layer is the most complex one. When taken apart, it consists of:

- **the Cluster Resource Manager (CRM)**: the master of ceremony. Every action which is taken in the “resource allocation layer” passes via the CRM. If any of the other parts of the “resource allocation layer”, or other components which are higher in the stack need to communicate, all communication passes via the local CRM. On every node, the CRM maintains the Cluster Information Base (CIB - see below). One CRM in the cluster is selected as the Designated Coordinator (DC), meaning that it has the “master” CIB, of which all other CIBs in the cluster are replicas. All write operations to the CIB are serialized through this “master” CIB. Read requests are satisfied locally on each node. The DC is the only entity in the cluster which can decide (possibly on command) that a cluster-wide change needs to be performed (e.g. fencing a node, moving resources around, etc.).

- **the Cluster Information Base (CIB)**: an in-memory XML representation of the whole cluster configuration and status (node membership, resources, constraints, etc.). There is one “master” CIB in the cluster, maintained by the DC. All the other nodes contain CIB replicas. If an administrator wants to manipulate the cluster's behavior, he must ask the DC to change the “master” CIB, via the CLI command “cibadmin” or via the Heartbeat 2 GUI.

- **Policy Engine (PE) and Transition Engine (TE)**: whenever the DC wants to react to a new CIB i.e. it wants to change something cluster-wide, the PE helps the DC to come up with a number of steps that must be performed on the relevant nodes in the cluster in order to complete this transition. The commands computed by the PE are then executed by the TE. The DC will send out messages to the relevant CRMs in the cluster, who will then use their Local Resource Managers (LRM – see below) to perform the necessary resource manipulations. The PE/TE pair is only run on the DC node.

- **the Local Resource Manager (LRM)**: This service calls the local Resource Agents (RA – see below) on behalf of the CRM. It can thus perform start / stop / monitor operations and report back on the result to the CRM. In case of the “monitor” operation, the response of a failure can be reported back to the CRM in an asynchronous way. The LRM is authoritative about the state of its local node.
A graphical overview of the cluster components and how they work together.

Finally, in the “resource layer”, we find one or more Resource Agents (RA). A RA is (usually) a script which has been specifically written to start, stop and monitor a certain resource, e.g. a service. As such it usually accepts the parameters “start”, “stop” and “monitor”, together with certain instance parameters. This is specified in the Open Clustering Framework Resource Agent API. RAs are called only by the LRM. Third parties can drop their own agents in a defined location on the filesystem and thus provide cluster integration for their own software out of the box.
Lets discuss what happens when the administrator wants to enforce a cluster-wide change, e.g. he wants to add a resource (say a cluster IP address). The administrator will use the command “cibadmin” or the Heartbeat 2 GUI to modify the master CIB. He does not need to use that command on the DC, he can run it wherever in the cluster, and the local CIB will relay the requested changes to the DC, which will replicate the CIB change to all nodes and will then start the transition procedure.

With help of the PE/TE pair, the DC now obtains a series of steps that need to be performed in the cluster, possibly and likely on several nodes. The DC sends commands out via the “messaging/infrastructure“ layer, which are received by the other CRMs. These CRMs use their LRM if needed to perform resource modifications, and report back to the DC about the result.

Once the TE on the DC concludes that all necessary operations are performed in the cluster successfully, the cluster will go back to the idle state and wait for further events. If any operation was not carried out as planned, the PE is invoked again with the new information recorded in the CIB.

When a service or a node dies, more or less the same thing happens, be it that the DC is informed by the CCM (in case of a node death) or by a LRM (in case of a failed monitor operation). The DC as such learns that actions needs to be taken in order to come to a new cluster state, which will be represented by a new CIB.

If you want to learn more about the Heartbeat 2 project, it is very well documented on http://www.Linux-ha.org/v2. On that site http://www.Linux-ha.org/GettingStartedV2 provides a good place to start your investigations into this product, next to http://www.Linux-ha.org/GettingStartedRevisedV2 which takes another approach.
Configuring Heartbeat 2 on node1

To configure heartbeat on both nodes, we use YaST. In YaST, go to the “System” submenu and there select “High Availability”. In the “Heartbeat – Node configuration” window, add “node2” as the second node in the cluster. Then follow the screenshots below.

DISCLAIMER The setup in this document is only for testing.

The heartbeat configuration we perform in this document is not a representation of what a real-life configuration looks like. We will not configure redundant heartbeat media. In real life you will want to have at least two media over which the heartbeat can be propagated, so that the cluster can survive e.g. an unplugged ethernet cable. We also will limit ourselves to configure only a demo STONITH device. A STONITH device is a power switch which the cluster uses to reset nodes that are considered “dead”. Fencing by resetting non-heartbeating nodes is the only 100% reliable way to make sure that no data corruption is performed by nodes that e.g. hang and only appear to be “dead”.

Add node2 as the “other” node in the cluster via “Add”. Then click “Next”.
If you configure keys, make sure to do it the same way on all nodes.

In the “Heartbeat – Media Configuration” window, click “Next”.

As said this is not an optimal setup. Confirm the warning with “Continue”.
“Next”, then select “On” to start Heartbeat, and then press “Finish”.

**Propagating the Heartbeat 2 configuration to node2**

In a terminal window on node1, run the ha_propagate command as described below. This will copy the configuration to node2. Then, in a terminal window on node2, run the command “rcheartbeat start” to start the heartbeat processes on node2.

```
node1:~ # /usr/lib/heartbeat/ha_propagate

Propagating HA configuration files to node node2.

Password: novell

ha.cf 100% 112
0.1KB/s 00:00

authkeys 100% 20
0.0KB/s 00:00

Setting HA startup configuration on node node2.

Password: novell
node1:~ #
```
Creating a test cluster resource from node1

To see if Heartbeat 2 is working, let's create a test resource and see if we can manipulate that resource with the tools that are provided by the Heartbeat 2 packages. A simple resource to configure is a cluster IP address. This is an extra ip address which is given by the cluster to one of the nodes, e.g. 192.168.0.3.

For simple cluster tasks, the Heartbeat 2 GUI can be used. On node1, press Alt+F2. In the pop-up which now appears, type in the command “hb_gui” and click “Run”. That should start the “Linux HA Management Client”, also called the “Heartbeat 2 GUI” in this document. At first, the GUI will show a blank space, inviting you to connect to the running Heartbeat stack.

Click on “Connection” and then “Login”. This will display a login pop-up box in which you must fill in the password of the “hacluster” user, which is a system user that can log in to the Heartbeat 2 stack. Being root, you can set this password to whatever you like, so first let's do that.

Without closing the Heartbeat GUI, open YaST and go to the “Security and Users” section. Click on “User Management” and in the “User and Group Administration” window, click on the dropdown box “Set Filter”. Then select “System Users”. You now see the list of system users. Select the “hacluster” user and click the “Edit” button below. In the new window, type “novell” in the “Password” and “Confirm Password” fields. Then close with “Accept”, “Yes” on the warnings, and “Finish” in the “User and Group Administration” window.

Exit YaST as usual by choosing “File” and then “Quit”. You can now enter “novell” in the Heartbeat GUI login pop-up “Password” field. Click “Ok” and you should get logged on to the Heartbeat 2 stack. Follow the screenshots below to configure the test resource.
Select the “hacluster” user and click “Edit”.

Set the password to “novell” for simplicity. Accept the two warnings with “Ok”.

Click “Finish” to leave “User and Group Administration”.

Do this preferably on both nodes so you can use the GUI on both nodes.
Open the Heartbeat GUI, and select “Connection” -> “Login”.

Log in with the “novell” password you just gave the “hacluster” user via YaST.
We will now add IP address “192.168.0.3” as a test cluster resource. Select the “Resources” icon in the GUI and click on “Action” -> “Add New Item” or click on the “+” sign. In the pop-up, click on “Ok” to confirm “Native” as the desired type of the resource. In the “Add Native Resource” window, enter “resource_test” for “Resource ID” and select “IPaddr” as the “type”. You will see two “IPaddr” types: take the one that has “OCF Resource Agent compliant IPaddr script” as “Description”.

Now select the second line of the “Parameters” list (where “Name” says “ip”), click on the empty field below “Value”, which now becomes editable, and fill in “192.168.0.3”. Then click the “Add Parameter” button right below the “Parameters” section. In the “Add Parameter” pop-up, select “nic” for “Name” (from the drop-down box) and type “eth0” in the “Value” textfield. We need to do this to make sure the ip address will be connected to the eth0 network interface on the node where the resource will be run. Click “Ok” in the pop-up.

To conclude, click the “+ Add” button at the bottom of the window. You will now be able to “expand” the “Resources” icon in the main GUI panel (click on the small triangle), which will reveal the new resource. Select it and press the “Start Resource” button on top. The resource now starts. The GUI clearly shows on what node it is running. On that node, run the command “/sbin/ifconfig -a” in a terminal to check if the cluster resource is really running on that node.

Click the “+” button to start adding a resource. Click “Ok” in the pop-up.
In the new window, fill in an “ID” and select “Ipaddr” (OCF RA) as “Type”.

Set “192.168.0.3” for the “ip” parameter value and click “Add Parameter”.

Select “nic” for the parameter name and give it value “eth0”. Click “Ok”.

Connected to 127.0.0.1
Click “+Add” to add the resource to the cluster.

The resource appears in the GUI, but in the stopped state.

Select the resource and press the “Start” button.
The High-Availability Storage Foundation

You now have the resource running in the cluster.

```
node1:~ # /sbin/ifconfig -a
...
eth0:0   Link encap:Ethernet  HWaddr 00:0D:7D:7A:9A
          inet addr:192.168.0.3  Bcast:192.168.0.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
```

Open a terminal on the node which hosts the resource, and run “/sbin/ifconfig -a” to check for the eth0:0 interface.

### Removing the test cluster resource

Before we go on we first remove the test resource. In the Heartbeat 2 GUI, select resource_test and then click the “Delete resource” button. Confirm by clicking “Yes”, and the resource is removed from the cluster.
Select the resource, click the “Delete Current Item” icon, and confirm with “Yes”.

The test resource will now be removed from the cluster.
Creating a SAN with iSCSI

Technology overview

iSCSI is a “SCSI over IP” technology. It basically uses the IP protocol to send SCSI commands over the network. It thus allows you to create a cheap block-level storage area network (SAN).

In our setup we will use a SAN to avoid having to copy the image file and the configuration file of our virtual machine over to all nodes in the cluster, in order to make these files available on those nodes. Having the image and configuration file available is a prerequisite for a node in order to be able to start a virtual machine.

When these files reside on the SAN, they will automatically become available to all nodes. Thus, we will create an iSCSI storage device, put some partitions on it, and allow both nodes to access these partitions over the network via iSCSI. One partition will function as a store for VM image files, the other as a store for our VM configuration files.

The iSCSI software consists of two parts. There is the iSCSI target software, which turns a bunch of disk space into a SAN Lun. You might see this as the iSCSI “server”. We will run this software only on node2.

Then there is the iSCSI initiator software, which logs in to the iSCSI target over the network, and creates a SCSI-like device file for it on the node where this initiator is run. We will run the initiator software on both nodes, where as a result of that, a new device file (in our case “/dev/sda”) will provide block level access to the disk space on node2. Neither target nor initiator software is installed by default, but Yast will install it for us when needed.
**Warning:** running target & initiator on one node is not supported.

For the SLES 10 release, Novell does not support running the iSCSI target and initiator software on the same node. However, since this document only describes how to set up a demo environment, we ignore this in order to reduce the hardware requirements, especially since it is very often perfectly possible to run target and initiator together on 32-bit of the shelf hardware. However, should you experience problems further in the setup, especially when installing virtual machines, then you must offload the iSCSI target to a third physical machine. Doing this is quite easy once you have read and understood the whole document.

We will use the free space on node2 to host our iSCSI SAN Lun.
Adding a new partition on the system disk of node2

To create a SAN Lun, we need space. So the first thing we need to do, is create a third partition out of the remaining disk space on node2. We will then configure that new partition as a SAN Lun via the iSCSI target software, which we will install on node2.

Mind you that, once created, this third partition will never be accessed directly (via the “/dev/hda3” device file in our case). So you must not specify a mount point for it when you create it. It will only be accessed as “/dev/sda” on both nodes, once the iSCSI initiator software has made it available via that device file.

Although partitioning can completely be done via the command line, we again use YaST in this document because of its ease of use. So on node2, in YaST, open the “System” section and click on “Partitioner”.

Click “Yes” when the warning displays. You now see the current partitioning of the system disk. Click “Create” to add a new partition. Turning the remaining disk space into a new partition if fairly easy, just follow the screenshots below. The only thing you need to be careful about, is to not specify a mount point for the partition.

Click “Create” to create a new partition of the remaining disk space.
Add a Primary Partition. Let it consume all remaining space (default).

Do not specify a mount point. Click “Ok”.

Revision: 1.1
Click “Apply”.

In the green “Changes” box, you can immediately click “Finish” to commit.

You now have a third local partition, in our case called “/dev/hda3”.
Configuring iSCSI target on node2

Open “YaST”, “Network Services”, “iSCSI Target”. Let YaST install the software.

Select “When Booting” to make the software start on subsequent reboots.
In “Targets” delete the demo iSCSI target. Confirm with “Continue”.

Click “Add” to add a new iSCSI target.
Specify “/dev/hda3” as the “Path” to the storage space for the Lun and click “Next”.

Click “Next” to go on (we don’t configure authentication in this document).
Conclude the creation of the iSCSI target with “Finish”.

Click “Yes” in the warning pop-up to restart the iSCSI target software stack.
Configuring iSCSI initiator on node1

Open YaST, “Network Services”, “iSCSI Initiator”. Let YaST install the software.

Select “When Booting” to start the initiator on subsequent reboots.
In the “Discovery Targets” tab, Click the “Discovery” button.

Fill in the ip address of node2 (192.168.0.2) and click “Next” to let YaST search for the iSCSI target on node2.
The iSCSI target on node2 is discovered. Click “Log in” to connect to it.

Click “Next” in the “iSCSI Initiator Discovery” window (no authentication is required).
The iSCSI target on node2 is now a “Connected Target”. Select it.

Click the “Toggle Start-Up” button to ensure we connect to it at each reboot.

Click “Finish” to leave the iSCSI initiator YaST module.
Partitioning the iSCSI Lun via node1

On node1 open the YaST partitioner again. Click “Create” and select the right device file for the “IET-VIRTUAL-DISK”, which is “/dev/sda” in our case.
Create a primary partition. Don't specify a mount point. This partition will host the virtual machine image file and as such needs to contain at least 5 GB.

In this case we use 20 of the +- 21 GB available, by replacing what is in the “End” text field by “+20GB”. Leave a few hundreds of MBs available.
Click “Create” again. We will now create the partition for the virtual machine configuration file.

Again select “/dev/sda” if it is not already selected by default.
Again choose “Primary Partition”.

Here we create a partition of 500 MB (+500M). Don’t specify a mount point.
Create a third partition.
Take the remaining space. A hundred megabytes or so will do.

Don't specify a mount point.

This partition can act as dedicated VM storage space, and can e.g. contain the data files for the service which runs in the VM.
When you have configured all partitions, click “Apply”.

In the green pop-up, confirm the proposal with “Apply”.
YaST partitions the iSCSI target Lun and puts Reiser filesystems on the partitions.

Click “Quit” to leave the partitioner once YaST has finished.
Configuring iSCSI initiator on node2 (differences)

On node2, list the loopback ip address (127.0.0.1) as the discovery address, which should again reveal the iSCSI target which is running on node2.

That's the only difference there is for node2.

Do not redo the partitioning part, you already have partitioned the SAN device via node1.
Oracle Cluster File System 2

Technology overview

Now that we have two partitions on our SAN Lun, we actually have some good places to store our virtual machine image file and virtual machine configuration file. It might not be immediately clear why we need OCFS2 though.

Remember that the goal of this document is to create a cluster where a virtual machine is an actual cluster resource. This means that the virtual machine must be able to run on every node in the cluster. It will run on only one node at a time, but when it crashes on a node (and cannot be restarted there), it definitely needs to be able to start somewhere else in the cluster.

In order to be runnable on all nodes, the virtual machine image and configuration files need to be available on all nodes. But even if we put the image file on one SAN Lun partition and the configuration file on the other, then we still cannot mount those partitions on all nodes in the cluster at the same time.

Reason is that, by default, SUSE Linux uses the Reiser filesystem, and Reiser is not a cluster aware filesystem. Mounting the partitions simultaneously on more than one physical node will surely corrupt the Reiser filesystems on the partitions, rendering their contents unusable.

Although it would be possible to give each image file / configuration file set its own partition, and specify cluster rules (constraints) that would make the cluster mount that partition only on the node which needs to run the virtual machine, this is not really a flexible solution.
What we want, is to put all image files together in one big “imagestore” partition, and all configuration files together in one big “configstore” partition, and have those stores mounted on “all of the nodes, all of the time”. To realize that, we need a true cluster safe filesystem (having cache coherency), like OCFS2.

A true cluster safe filesystem must have (disk) cache coherency.
OCFS2 was developed completely independently from Heartbeat 2, and hence can run in a “pure” OCFS2 cluster, where Heartbeat 2 is even not present. Therefore, OCFS2 has its own kernel-level cluster membership service and heartbeat functions. It actually uses a disk-based heartbeat by default. Nodes which have a OCFS2 filesystem mounted, must regularly write into a metadata heartbeat area in that filesystem, letting the other nodes know that it is still alive.

To allow integration with heartbeat however, the OCFS2 developers have made the cluster membership service pluggable, as such allowing that a user space program tells OCFS2 which nodes are in the cluster at any given time.

This is exactly how we must configure OCFS2 in the SLES 10 High Availability Storage Foundation. OCFS2 must be told to listen to user-space cluster membership instructions, coming from Heartbeat 2. Heartbeat 2 on its turn must be configured to perform the role of “OCFS2 cluster membership” authoritative source of information.

The reason that we need to configure things this way, is because we absolutely want to avoid a situation where OCFS2 considers a node to be part of a Heartbeat 2 / OCFS2 cluster, while Heartbeat 2 considers that same node not to be part of that Heartbeat 2 / OCFS2 cluster. The must only be one cluster membership service running in the cluster.

Having OCFS2 listen to Heartbeat 2 for cluster membership instructions does not mean that OCFS2 is something which is invisible to the administrator. Quite contrary to that, even when Heartbeat 2 integration is the goal, OCFS2 must initially be set up as if it were a “regular” OCFS2 setup. Performing such a native OCFS2 setup will take up the first part of the this chapter.

After that initial (GUI-based) OCFS2 setup, we will reconfigure OCFS2 to listen to Heartbeat 2 cluster membership instructions. As we will see, once that change is done, OCFS2 will still be up and running, but it will not be running “properly” anymore, since it will have no way of knowing who is part of the cluster and who is not (because we have switched of its own heartbeat functions).

So to make Heartbeat 2 inform OCFS2 about changes in the cluster topology (i.e. nodes that are joining or leaving the cluster), and to make the cluster filesystem(s) actually get mounted on the joining nodes, we will then need to add a “OCFS2 Filesystem resource” into the cluster, one for each OCFS2 file system that we wish to use in this cluster.

This Filesystem resource will be particular, since it will be a “clone” resource. “Clone” resources are quite special, because they can potentially run on EVERY node in the cluster, and they actually will do that in our setup. This is quite different from traditional Heartbeat 2 resources, which only run on one node of the cluster at a time.
As you have already understood, it is the OCFS2 Filesystem Resource Agent that will on each node inform the OCFS2 stack about which nodes are part of the cluster and which are not. But how does the OCFS2 Filesystem Resource Agent itself learn about cluster topology changes? A Resource Agent can be configured to support a special cluster resource operation, called “notify”, next to the classical cluster Resource Agent operations (“start”, “stop” and “monitor”).

When configured as such, each time the Heartbeat 2 cluster encounters a change of cluster topology (nodes joining or leaving the cluster), the Local Resource Manager on each node will call the Filesystem Resource Agent with this “notify” operation, and at the same time pass it a set of variables. These variables allow the Resource Agent to calculate the current cluster membership status. The Filesystem Resource Agent will then pass on this information to the OCFS2 stack running on the node, and will mount or unmount the OCFS2 filesystem as appropriate.

The Heartbeat infrastructure tells OCFS2 who is part of the cluster.
Initializing the native OCFS2 stack on node1

Configuring OCFS2 by console only needs to be done on ONE cluster node.

Press ALT+F2, type “ocfs2console” and hit “Run”.

In the console, select “Cluster” -> “Configure nodes”. “Close” the pop-up.
In the “Node configuration” pop-up, click the Add button to start adding nodes.

In the “Add node” pop-up, add the necessary data for node1 and click “Ok”.
Again click “Add”, enter the necessary data for node2 and click “Ok”.

In the “Node Configuration” pop-up now click “Apply”.

We have now added both nodes to the OCFS2 cluster configuration.
Once the proposal has been applied, “Close” the “Node Configuration” pop-up.

Click “Cluster” -> “Propagate Configuration” to propagate the configuration to the other node in the cluster via ssh (in our case node2).
If you get a question about RSA key fingerprint, type “Yes” and hit enter.

Type the password for “root” on node2 (“novell”) if prompted for it.
When you see the text “Finished” appearing, hit the “Close” button at the bottom. The configuration file is now propagated to node2.

You can close the OCFS2 console now.
Enabling user space OCFS2 cluster membership setting

All the steps we have just performed have had the following results:

- both nodes have the OCFS2 configuration file on them (propagated to node2)
- the OCFS2 stack is running on node1, but will not survive a reboot. The stack is not yet running on node2
- OCFS2 uses disk-based heartbeat, while we need user space heartbeat

What we want is the following:

- we want that OCFS2 runs on both nodes and comes back up after a reboot
- we want OCFS2 to use user space heartbeat on all nodes

To obtain what we want, we must run the command "/etc/init.d/o2cb configure" on both nodes. Because at this point in the setup the nodes have a different configuration, the output of what you will see is a bit different on each node, but that is normal. Below we show what to do on each node. The answers that you must provide to the asked questions are marked in bold font.

On node1:

```
node1:~ # /etc/init.d/o2cb configure
Configuring the O2CB driver.
This will configure the on-boot properties of the O2CB driver.
The following questions will determine whether the driver is loaded on boot. The current values will be shown in brackets ('[]').
Hitting <ENTER> without typing an answer will keep that current value. Ctrl-C will abort.
Load O2CB driver on boot (y/n) [n]: y
Cluster to start on boot (Enter "none" to clear) [ocfs2]: ocfs2
Use user-space driven heartbeat? (y/n) [n]: y
```
Writing O2CB configuration: OK
Switching to user mode heartbeat: /etc/init.d/o2cb: line 565:
etch: write error: Device or resource busy
FAILED
Cluster ocfs2 already online
node1:~ # /etc/init.d/o2cb force-reload
Cleaning heartbeat on ocfs2: OK
Stopping cluster ocfs2: OK
Unmounting ocfs2_dlmfs filesystem: OK
Unloading module "ocfs2_dlmfs": OK
Unmounting configfs filesystem: OK
Mounting configfs filesystem at /sys/kernel/config: OK
Loading module "ocfs2_dlm": OK
Loading module "ocfs2_dlmfs": OK
Mounting ocfs2_dlmfs filesystem at /dlm: OK
Switching to user mode heartbeat: OK
Starting cluster ocfs2: OK
node1:~ # cat /sys/o2cb/heartbeat_mode
user
node1:~ #

The tasks are pretty straightforward. The script asks which cluster we want to configure. "ocfs2" is the default cluster name, and since we did not change that in the GUI, that is also the name of our cluster. When switching to user mode heartbeat, we get an error, since the cluster stack is already running on node1 and you cannot "hot" switch to user mode in that case.

Running the “force-reload” command reloads the stack with user mode heartbeat. A reboot would also have done the job. The final command proves that OCFS2 now runs with user based heartbeat on node1: in that case the file “/sys/o2cb/heartbeat_mode” contains the word “user”.

On node2 the process is easier, since the OCFS2 cluster stack is not yet running there:

```bash
node2:~ # /etc/init.d/o2cb configure
Configuring the O2CB driver.
This will configure the on-boot properties of the O2CB driver.
The following questions will determine whether the driver is loaded on
boot. The current values will be shown in brackets ('[]').
Hitting
<ENTER> without typing an answer will keep that current value.
Ctrl-C
will abort.
Load O2CB driver on boot (y/n) [n]: y
Cluster to start on boot (Enter "none" to clear) [ocfs2]: ocfs2
Use user-space driven heartbeat? (y/n) [n]: y
Writing O2CB configuration: OK
Loading module "configfs": OK
Mounting configfs filesystem at /sys/kernel/config: OK
Loading module "ocfs2_node manager": OK
Loading module "ocfs2_dlm": OK
Loading module "ocfs2_dlmfs": OK
Mounting ocfs2_dlmfs filesystem at /dlm: OK
Switching to user mode heartbeat : OK
Starting cluster ocfs2: OK
node2:~ # cat /sys/o2cb/heartbeat_mode
user
node2:~ #
```
Creating the OCFS2 file systems

At this point we have the OCFS2 stack running on both nodes. We don’t however have any OCFS2 file system running yet. On either node (but not on both), use the command “mkfs.ocfs2” to install OCFS2 cluster filesystems on the Lun partitions (“/dev/sda1” and “/dev/sda2” in our case).

```
node1:~ # mkfs.ocfs2 /dev/sda1
mkfs.ocfs2 1.2.1
Filesystem label=
Block size=4096 (bits=12)
Cluster size=4096 (bits=12)
Volume size=21476171776 (5243206 clusters) (5243206 blocks)
163 cluster groups (tail covers 17734 clusters, rest cover 32256 clusters)
Journal size=134225920
Initial number of node slots: 4
Creating bitmaps: done
Initializing superblock: done
Writing system files: done
Writing superblock: done
Writing lost+found: done
mkfs.ocfs2 successful
node1:~ # mkfs.ocfs2 /dev/sda2
mkfs.ocfs2 1.2.0
<output omitted>
mkfs.ocfs2 successful
```
More about user space cluster membership instructions

At this point you might be tempted to mount the OCFS2 filesystems “/dev/sda1” and “/dev/sda2” on both nodes. However, this will fail. The reason for this is that OCFS2 considers nodes1 and nodes2 as “dead”, since they are not heartbeating, and Heartbeat 2 did not yet tell OCFS2 that node1 and node2 are part of the OCFS2/Heartbeat 2 cluster (we did not yet configure Heartbeat 2 to do that).

So when you try to mount the filesystems now, OCFS2 will deliberately crash (forcing you to reboot the systems), to stop you from possibly corrupting the data on the filesystems. This is the normal reaction. If you do want to manually mount the filesystems at this time to do some testing, then you must first switch back to disk based heartbeating by running “/etc/init.d/o2cb configure” on each node again again, choosing disk-based heartbeat, followed by a **reboot** of both machines.

ADVANCED: User space OCFS2 cluster membership.

As soon as the OCFS2 stack is running on a node, the kernel part of OCFS2 will create a set of files and directories in “/sys”. “/sys” is a bit like /proc, it is a RAM-based filesystem which acts as an interface between kernel space and user space. For us, the most interesting part is under “/sys/kernel/config/cluster” and it looks like this:

```
node1:/sys/kernel/config/cluster # find .
  ./ocfs2
  ./ocfs2/heartbeat
  ./ocfs2/node
  ./ocfs2/node/node2
  ./ocfs2/node/node2/local
  ./ocfs2/node/node2/ipv4_address
  ./ocfs2/node/node2/ipv4_port
  ./ocfs2/node/node2/num

node2:/sys/kernel/config/cluster # find .
  ./ocfs2
  ./ocfs2/heartbeat
  ./ocfs2/node
  ./ocfs2/node/node1
  ./ocfs2/node/node1/local
  ./ocfs2/node/node1/ipv4_address
  ./ocfs2/node/node1/ipv4_port
  ./ocfs2/node/node1/num
```
“ocfs2” is the name of our cluster. In the “node” subdirectory we find a subdirectory for each node that is part of that cluster. As we already know, when Heartbeat 2 will be managing OCFS2 cluster membership, it will tell the OCFS2 stack on each node which nodes are part of the OCFS2/Heartbeat 2 cluster, even though these nodes are not heartbeating on the OCFS2 level. Since the native heartbeat of OCFS2 is filesystem based, the cluster must do that for every OCFS2 filesystem in the Heartbeat 2 cluster separately (in our case for “/dev/sda1” and “/dev/sda2”).

How does this communication happen? The cluster (user space) communicates with OCFS2 (kernel space) by creating certain directories and symbolic links under the directory “/sys/kernel/config/cluster/<cluster name>/heartbeat” For each OCFS2 filesystem, and on each node in the Heartbeat cluster where this filesystem must get mounted, the cluster must first create in the “heartbeat” directory a subdirectory having as name the UUID of the OCFS2 filesystem.

The UUID is a unique identifier for the OCFS2 filesystem, each OCFS2 filesystem has one. You can ask the UUID of “/dev/sda1” and “/dev/sda2” like this:

```
node1:~ # mounted.ocfs2 -d /dev/sda1
```

Device  FS      UUID                                      Label
/dev/sda1  ocfs2  fd3ac7b-9236-4042-917d-66e7bdc7c6c6

```
node1:~ # mounted.ocfs2 -d /dev/sda2
```

Device  FS      UUID                                      Label
/dev/sda2  ocfs2  77fe9fc-d6f2-4344-be58-19d4e54c5186

The UUID directory name under “/sys” must contain no “-” and must be all capitalized, but the cluster RA takes care of this simple name conversion.

Once that directory is created, the cluster must then create symbolic links in it. The links must point to directories in the “node” directory, and must also have the name of those directories. By placing those links, which as you have understood correspond to nodes in the cluster, Heartbeat 2 tells the OCFS2 stack that “these nodes are considered up and running for this filesystem” and that OCFS2 must allow these nodes to mount that filesystem. Needless to say that in a non-failing OCFS2/Heartbeat 2 cluster, the content of each UUID directory on each OCFS2-enabled node must be the same (since Heartbeat 2 fences by resetting nodes that are failing).

To illustrate all of this, let’s look at our own cluster. We have two nodes and two OCFS2 filesystems, and both nodes must be able to mount both filesystems. This means that the Heartbeat 2 cluster on each node must create the following directories and symbolic links (the UUIDs will be different for you) under “/sys/kernel/config/cluster/”: 
(The files in bold are the created soft links (and parent directories), the color indicates what they link to.)

.../ocfs2/
.../ocfs2/heartbeat
.../ocfs2/heartbeat/77FFE9FCD6F24344BE5819D4E54C5186
.../ocfs2/heartbeat/77FFE9FCD6F24344BE5819D4E54C5186/node2
.../ocfs2/heartbeat/77FFE9FCD6F24344BE5819D4E54C5186/node1
.../ocfs2/heartbeat/FEAAAF7B92364042917D66E7BDC7C4C6
.../ocfs2/heartbeat/FEAAAF7B92364042917D66E7BDC7C4C6/node2
.../ocfs2/heartbeat/FEAAAF7B92364042917D66E7BDC7C4C6/node1
.../ocfs2/node
.../ocfs2/node/node2
.../ocfs2/node/node2/local
.../ocfs2/node/node2/ipv4_address
.../ocfs2/node/node2/ipv4_port
.../ocfs2/node/node2/num
.../ocfs2/node/node1
.../ocfs2/node/node1/local
.../ocfs2/node/node1/ipv4_address
.../ocfs2/node/node1/ipv4_port
.../ocfs2/node/node1/num

This is the equivalent of telling OCFS2 that “node1 and node2 are alive for the OCFS2 filesystems having UUIDs 77FFE9FCD6F24344BE5819D4E54C5186 and FEAAAF7B92364042917D66E7BDC7C4C6” and that it is as such ok for these nodes to mount these filesystems.
Integrating OCFS2 in the Heartbeat 2 cluster

We are now ready to integrate OCFS2 as a true cluster resource. Doing that definitely belongs in the “advanced Heartbeat configuration” section, and as such must be done by using the Heartbeat command line interface (CLI). However, before we can do all this, we first need to set up a fake STONITH device. Let’s investigate why.

Remember that OCFS2 will be added via a clone Filesystem Resource Agent, which will be running on each node. These clones will be notified by the cluster from time to time, in order to learn about cluster membership changes (because we will enable notifications for them). The cluster will notify subscribed Resource Agents and communicate the current status of the cluster to them on the following events:

- a resource is stopped in the cluster
- a resource is started in the cluster
- a node joining the cluster
- a node leaving the cluster

In fact, for some of these events, there can be TWO notifications, a “pre” and a “post” event notification. E.g. when a node is being fenced from the cluster, the cluster will call all subscribed RAs twice, using their “notify” operation. It will use variables to make the RA understand that it is a “pre-stop” or a “post-stop” notification, and other variables that allow the RA to calculate which nodes are at that moment part of the cluster and which nodes are not.

Now, suppose we – once all is configured – want to test our High Availability setup and kill the heartbeat of node1. At that moment, the cluster will not be able to not know that node1 has not really been turned off. All the cluster knows, is that node1 all of a sudden stopped sending out heartbeat signals. The nature of Heartbeat 2 in such a case it that it MUST enable the STONITH mechanism to make sure that node1 is really dead (turned off/reset). In a real setup, this means that the cluster will tell a connected power switch to turn off or reset node1. The operations happen in this order:

- cluster notices that node X is dead (not sending out heartbeat signals)
- cluster sends pre-stop notifications to all subscribers (“we will shut node X down”)
- cluster tells STONITH subsystem to shut down / reset node X (and expects success)
- **on success only**, the cluster sends post-stop notifications around (“we have STONITH-ed node X”)
Now, at this point it should be clear that if we don't configure a STONITH device, the Filesystem Resource Agents will never receive the post-stop notification, simply because they will never be sent. This is bad, because they need it, in order to set the correct new OCFS2 cluster membership. Hence, we need a STONITH device. But how can we configure a STONITH device if we don't have one? Luckily, Heartbeat contains a demo STONITH device, the “ssh” device. Basically, when the cluster calls this device to shut a node down, this “device” will ssh to that node and issue a reboot command. It can be used for testing, but requires of course that the node which has to be shut down is reachable via the network.

So always simulate a node crash by killing the heartbeat processes on that node (thereby stopping the heartbeat), never by unplugging the network cable. Due to the dependency of node1 on node2 for storage (node2 runs the iSCSI target software), unplugging the network cable in this setup is never a good idea anyway. In summary, when you follow this document, you can only test system crashes by bringing down heartbeat on node1.

ADVANCED: How is STONITH implemented?

The STONITH framework consists of a set of STONITH daemons, one running on each node. Some or all of these daemons can have access to one or more STONITH devices. Consider e.g. the simple situation where all nodes share one STONITH power switch. In such a configuration, all STONITH daemons can control the switch (be it not simultaneously), and every node in the cluster can be shut down by that switch. Several other setups are possible, where not every STONITH daemon can switch off just any node (e.g. topologies with multiple power switches due to a high number of nodes). To cope with these various topologies, the STONITH daemons can in fact communicate. If a STONITH daemon gets a request to shut down a node which it cannot shut down itself (because it doesn’t have access to the right power switch), the daemon can route the request to another STONITH daemon which can do the job.

The original initiator of such a request is always the TE (running on the DC) which will contact its local STONITH daemon first and ask it to STONITH a node (in the context of a transition). That STONITH daemon will route (if necessary), a node will be shut down by some STONITH daemon, the response will be routed back (if necessary), and the TE can then conclude that the node has been STONITH-ed (turned off/reset).

How do the STONITH daemons know who to route a request to if necessary? They know this because the cluster configures them as such. To do that the cluster uses a STONITH clone resource agent on every node. This resource agent is particular, since it exists not as a bash script, but as a special LRM module. When this module receives the “start” operation from the cluster, it also receives information on how to instruct the local STONITH daemon to load the appropriate STONITH plug-in and configure that plug-in.
The High-Availability Storage Foundation

SUSE Linux Enterprise Server 10

There are several ways of performing STONITH, for each there is a loadable plug-in. The cluster on its turn gets its information from the administrator, which is responsible for feeding the STONITH topology into the CIB, via XML blobs and the cibadmin command. We will have to do this, but don’t worry our demo setup is quite simple.

To start the OCFS2 Heartbeat integration we first need to tune Heartbeat 2 by setting certain parameters for it. Doing that is done by entering an XML blob into the CIB. We've been mentioning the cibadmin command several time in this document, and now it’s time to use it. First create a text file with the following content:

```
<cluster_property_set id="cibbootstrap">
  <attributes>
    <nvpair id="cibbootstrap-01" name="transition_idle_timeout" value="60"/>
    <nvpair id="cibbootstrap-02" name="default_resource_stickiness" value="INFINITY"/>
    <nvpair id="cibbootstrap-03" name="default_resource_failure_stickiness" value="-500"/>
    <nvpair id="cibbootstrap-04" name="stonith_enabled" value="true"/>
    <nvpair id="cibbootstrap-05" name="stonith_action" value="reboot"/>
    <nvpair id="cibbootstrap-06" name="symmetric_cluster" value="true"/>
    <nvpair id="cibbootstrap-07" name="no_quorum_policy" value="stop"/>
    <nvpair id="cibbootstrap-08" name="stop_orphan_resources" value="true"/>
    <nvpair id="cibbootstrap-09" name="stop_orphan_actions" value="true"/>
    <nvpair id="cibbootstrap-10" name="is_managed_default" value="true"/>
  </attributes>
</cluster_property_set>
```

Note: the “\" character denotes a line continuation and must not be typed in the XML file.

Without going deeper into this, via this XML we enable STONITH and make sure the cluster will restart a failed resource on the node where it was running (via the default_resource_stickiness parameter). For the rest we explicitly set sane defaults. Supposing you saved this text in a file called cibbootstrap.xml, the command to load these settings into the CIB is:
The High-Availability Storage Foundation

HINT  How do I create these XML textfiles easily?

For easy of use we recommend using “gedit”. You can start “gedit” via the GNOME menus, or by pressing the ALT+F2 keys simultaneously, entering the “gedit” command in the text field of the “Run Application” pop-up, and clicking “Run” in the pop-up. Since we are logged on as root, “gedit” will by default store the files you save in “/root”, which is also the default location where you end up when opening a terminal as root.

“gedit” is a simple and user-friendly text editor.
Next we need to configure and start the STONITH clones in the cluster. Again create a text file with the following content (e.g. stonithcloneset.xml) and feed it into the CIB, like this:

```xml
<clone id="stonithcloneset" globally_unique="false">
  <instance_attributes id="stonithcloneset">
    <attributes>
      <nvpair id="stonithcloneset-01" name="clone_node_max" value="1"/>
    </attributes>
  </instance_attributes>
  <primitive id="stonithclone" class="stonith" type="external/ssh" provider="heartbeat">
    <operations>
      <op name="monitor" interval="5s" timeout="20s" prereq="nothing" id="stonithclone-op-01"/>
      <op name="start" timeout="20s" prereq="nothing" id="stonithclone-op-02"/>
    </operations>
    <instance_attributes id="stonithclone">
      <attributes>
        <nvpair id="stonithclone-01" name="hostlist" value="node1,node2"/>
      </attributes>
    </instance_attributes>
  </primitive>
</clone>
```

This file deserves some explanation, as it is quite complex. The XML in the file configures a cloneset called “stonithcloneset”, which will have on each node a clone called “stonithclone”. Remember that clones are like regular resources, but an instance of a clone is allowed to run on potentially each node of the cluster. It is even possible to run more than one instance of a clone on one node in the cluster, although that is not what we are going to do in this document.
Without going deeper into it, clonesets come in several categories. We need an anonymous cloneset i.e. a cloneset of which the clones act identically on all nodes. We configure the cloneset to be an anonymous one by setting the parameter “globally_unique” to false. See http://www.Linux-ha.org/v2 for further information on this.

“Clone_node_max” is the maximum number of clones allowed to run on one node, in our case that is absolutely “1”. Contained in the “clone” XML section we then have a “primitive” section, in which we configure the clones themselves. We configure the two operations that will be applied to each clone i.e. “start” and “monitor”.

Finally, in the “attributes” section, we set a name/value pair : the name is “hostlist”, the value has to be a comma separated list of the hosts in the cluster which can be shut down by this STONITH device. In our case that corresponds to “node1,node2”.

Notice how we give each attribute, operation, etc. an ID. At this moment the cluster wants uniqueness by “ID” + “XML section” only, so configuring globally unique ID’s is not absolutely necessary. It would be a good practice though.

Once you have the file down, load it into the CIB like this:

```
node1:~ # cibadmin -C -o resources -x ./stonithcloneset.xml
```

We now have a running and well configured Heartbeat 2 cluster, and we have our demo “ssh” STONITH demo devices up and running. Do not yet try to test the STONITH mechanism to see if it will reset a node, we still need to do some easy configuration changes before that will work. Those are listed at the end of this chapter, at the point where we do all testing.

Next up is configuring the clonesets for the imagestore and the configstore partitions. We start with the imagestore. Create an XML file (call it imagemstorecloneset.xml) for configuring and starting a clone OCFS2 Filesystem resource which mounts “/dev/sda1” to “/var/lib/xen/images” on each node in the cluster:
Again we are configuring an anonymous cloneset i.e. the “globally_unique” parameter is again set to “false”. Since this time we are configuring a cloneset which “contains” a OCFS2 Filesystem resource agent, we want to enable notify for it, so that the clones (the RAs) will receive notifications from the cluster and hence get informed on the cluster membership status. To enable notifications, set “notify” to “true” for the cloneset. We also configure the “monitor” operation, so that the cluster checks every twenty seconds if the mount is still there.
The most important part of the XML blob is the “attributes” section of the
“imagestoreclone” primitive part. Set the “device” parameter to the OCFS2 filesystem
which needs to be mounted, “directory” to the directory on which this filesystem must
get mounted, and “ fstype ” to “ocfs2” for obvious reasons (in a clone Filesystem RA, any
other value is forbidden since OCFS2 is the only supported cluster aware filesystem).
Load the XML blob into the CIB:

```bash
node1:~ # cibadmin -C -o resources -x ./imagestorecloneset.xml
```

Now create a similar XML file for the configstore (e.g. configstorecloneset.xml):

```xml
<clone id="configstorecloneset" notify="true" globally_unique="false">
  <instance_attributes id="configstorecloneset">
    <attributes>
      <nvpair id="configstorecloneset-01" name="clone_node_max" value="1"/>
      <nvpair id="configstorecloneset-02" name="target_role" value="started"/>
    </attributes>
  </instance_attributes>
  <primitive id="configstoreclone" class="ocf" type="Filesystem" \
            provider="heartbeat">
    <operations>
      <op name="monitor" interval="20s" timeout="60s" prerq="nothing" \
           id="configstoreclone-op-01"/>
    </operations>
    <instance_attributes id="configstoreclone">
      <attributes>
        <nvpair id="configstoreclone-01" name="device" value="/dev/sda2"/>
        <nvpair id="configstoreclone-02" name="directory" value="/etc/xen/vm"/>
        <nvpair id="configstoreclone-03" name="fstype" value="ocfs2"/>
      </attributes>
    </instance_attributes>
  </primitive>
</clone>
```
Insert the XML into the CIB as follows:

```bash
node1:~ # cibadmin -C -o resources -x ./configstorecloneset.xml
```

Right now all XML blobs are loaded into the cluster. At this point in time, when you issue the mount command, you should see the imagemstore and the configstore mounted to directories “/var/lib/xen/images” and “/etc/xen/vm” respectively on both nodes in the cluster.

On node1:

```bash
node1:~ # mount
/dev/hda2 on / type reiserfs (rw,acl,user_xattr)
proc on /proc type proc (rw)
sysfs on /sys type sysfs (rw)
debufgs on /sys/kernel/debug type debugfs (rw)
udev on /dev type tmpfs (rw)
devpts on /dev/pts type devpts (rw,mode=0620,gid=5)
securityfs on /sys/kernel/security type securityfs (rw)
configfs on /sys/kernel/config type configfs (rw)
ocfs2_dlmfs on /dlm type ocfs2_dlmfs (rw)
/dev/sdal on /var/lib/xen/images type ocfs2
(rw,_netdev,heartbeat=local)
/dev/sda2 on /etc/xen/vm type ocfs2 (rw,_netdev,heartbeat=local)
```
And on node2:

```bash
node2:~ # mount
...
/dev/sda1 on /var/lib/xen/images type ocfs2 (rw,_netdev,heartbeat=local)
/dev/sda2 on /etc/xen/vm type ocfs2 (rw,_netdev,heartbeat=local)
```

Use the command “crm_mon -1” to list the resources that the cluster manages.

```bash
node1: ~ # crm_mon -1

-------------------
Last updated: Sun May 28 00:01:09 2006
Current DC: node2 (8013da0b-b535-4639-991c-c5b9c9735764)
2 Nodes configured.
3 Resources configured.
-------------------
Node: node2 (8013da0b-b535-4639-991c-c5b9c9735764): online
Node: node1 (9936e642-c90f-49aa-a376-071050e49f0f): online

Clone Set: stonithcloneset
  stonithclone:0  (stonith:external/ssh): Started node2
  stonithclone:1  (stonith:external/ssh): Started node1

Clone Set: imagestorecloneset
  imagestoreclone:0  (heartbeat:ocf:Filesystem): Started node2
  imagestoreclone:1  (heartbeat:ocf:Filesystem): Started node1

Clone Set: configstorecloneset
  configstoreclone:0  (heartbeat:ocf:Filesystem): Started node2
  configstoreclone:1  (heartbeat:ocf:Filesystem): Started node1
```
Testing the OCFS2 Heartbeat integration

**Test one**: manually unmount the imagestore. It should get remounted.

```
node1:~/hb2 # mount | grep sda1
/dev/sda1 on /var/lib/xen/images type ocfs2 (rw,_netdev,heartbeat=local)
node1:~/hb2 # umount /dev/sda1
node1:~/hb2 # mount | grep sda1
<delay depends on timing settings in the CIB>
node1:~/hb2 # mount | grep sda1
/dev/sda1 on /var/lib/xen/images type ocfs2 (rw,_netdev,heartbeat=local)
node1:~/hb2 #
```

**Test two**: Simulate a node crash.

*Simulating a node crash will require some extra work first.*

Remember that our STONITH device works by ssh-ing into the node which needs to be shut down. Since this is done using a bash script, we must set up “ssh with keys” from node2 to node1, i.e. it must be possible for node2 to log into node1 without having to provide a password.

Feel free to also configure public key based ssh from node1 to node2, but remember that we can never in a test shut down node2, since it is running the iSCSI target software and as such hosts our SAN Lun.

To set up public key based login from node2 to node1, first generate a RSA key pair on node2:
node2:~ # ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/root/.ssh/id_rsa): <enter>
Created directory '/root/.ssh'.
Enter passphrase (empty for no passphrase): <enter>
Enter same passphrase again: <enter>
Your identification has been saved in /root/.ssh/id_rsa.
Your public key has been saved in /root/.ssh/id_rsa.pub.
The key fingerprint is:
4f:04:fe:f6:c5:c0:3f:c4:a0:ae:1c:fc:20:14:50:72 root@node2

Once that is done, copy the public key file of root on node2 into the file .ssh/authorized_keys in the home directory of the root user of node1, like illustrated below. As such you are giving permission to the root user of node2 to log into node1 without having to specify a password. The fact that the public key of the node2 root user is known on node1, is enough to allow node1 to establish the necessary authentication.

node2:~ # scp /root/.ssh/id_rsa.pub node1:/root/.ssh/authorized_keys

The authenticity of host 'node1 (192.168.0.1)' can't be established.

RSA key fingerprint is

Are you sure you want to continue connecting (yes/no)? yes

Warning: Permanently added 'node1,192.168.0.1' (RSA) to the list of known hosts.
Password: novell

node2:~ # ssh node1
Last login: Wed Jun 7 19:00:09 2006
node1:~ # exit
logout
Connection to node1 closed.

node2:~ #

**Warning :**  
*log in at least once from node2 to node1 via ssh*

The STONITH ssh agent needs to be able to log into node1 without any questions asked, so make sure to log into node1 at least one time manually (using scp or ssh) to answer the initial (yes/no) question you get when logging into a node via ssh for the first time. See the example above.

Now we need to enable the “atd” job schedule daemon on node1. The STONITH ssh device makes use of it, so if that daemon is not running on the node which needs to be shut down, it will not be shut down. Enabling “atd” on node2 too is allowed, but again due to the asymmetry in our setup, it will never be used there.

node1:~ # chkconfig atd on
node1:~ # chkconfig --list atd
atd 0:off 1:off 2:on 3:on 4:off 5:on 6:off
node1:~ # rcatd start
Starting service at daemon done

Now kill the heartbeat processes on node1:

node1:~ # pkill heartbeat

The cluster will soon notice that node1 is not heartbeating anymore, and will issue a STONITH command to shut down node1. Node1 will be reset. It will reboot, come back up, and should rejoin the cluster, so that our “imagestore” and “configstore” get mounted again. The commands you can use to verify all this are known to you at this point.
The Xen hypervisor

Technology overview

Xen is an open source hypervisor. It is developed by several companies, among which Xensource and Novell. As such is the leading open source project in the virtualization area.

The Xen hypervisor runs directly on top of commodity hardware, and allows for several virtual operating systems to run on top of it. These virtual machines are usually called “domains”. One domain is the “master” domain, the domain from which all other virtual machines are managed. The master domain is usually called “domain 0”. Domain 0 cannot be shut down or otherwise made unavailable – it is required for the other virtual machines to run – and as such behaves very much like a traditional operating system. Domain 0 is the domain that will boot on top of Xen when the physical machine is booted.

Xen uses paravirtualization, which means that the guest operating systems know that they run on virtualized hardware. This design was chosen because it allows for increased performance. The virtual machines other than domain 0, usually called “user domains”, or “domU”s, cannot access disk and network resources directly, but pass their requests on to domain 0. Domain 0 handles the requests and sends the retrieved information back to the user domain that requested it. The virtual machines have specialized disk/net drivers that realize all this. SUSE Linux Enterprise Server 10 ships with Xen and several tools to create and manage virtual machines.

The procedure that is outlined in this chapter is very much standard, i.e. YaST does not realize that it is creating the VM on a SAN Lun. So you can use this chapter as an aid even when you are using Xen in a non-clustered environment.
Tuning the bootloader for Xen

What we have to do first is to reboot our systems “on Xen” and make sure that, on subsequent reboots, the systems run on top of Xen by default. To do the latter we need to modify the GRUB bootloader file. Once that is done we will simply reboot to test if all is configured correctly.

Start YaST. The boot loader configuration can be found in the “System” submenu, where you find the module “Boot Loader”. Click on it to start it.
In the “Boot Loader Settings” panel, you will see a loader stanza with the label “XEN”. Select it and click the button “Set as Default”. Click “Finish” to leave the “Boot Loader” module and then close YaST. Do this on both systems, and then reboot both systems simultaneously like this:

```
node1:~ # sync
node1:~ # sync
node1:~ # sync
node1:~ # sync
node1:~ # reboot -nf
```

**HINT**  
Make sure iSCSI target is available for node1 when rebooting

When rebooting both your systems at the same time, artificially slow down the reboot of node1 a bit by going in the BIOS boot menu or by actively entering the GRUB boot loader menu (by using the up and down arrows). Since node1 is dependent on node2 for its storage (node2 runs the iSCSI target) we want to make sure that when node1 need its storage, it's there. So the iSCSI target on node2 needs to be started when node1 starts its iSCSI initiator. Make sure to choose “SUSE Linux Enterprise Server 10 (XEN)” when you are leaving the GRUB menu on node1.

Once the systems are rebooted, one each system issue the “uname -a” command in a terminal. This command shows what kernel you are running. If “xen” is part of the kernel name, you are running on top of Xen.

```
node1:~ # uname -r
2.6.16.21-0.8-xen
```
Changing the sync_mode of the loop device

When running Xen virtual machines based on image files, as we do, the loop device is used. This kernel module allows to mount a file system which resides in a file. During the SLES 10 test cycles, Novell has discovered that under certain circumstances, when the Linux loop device is being used, and the physical node on which the virtual machine is running crashes, the filesystem in the virtual machine can become corrupt. To cope with this, Novell has applied patches to the loop device. The patches remove the risk of corruption, but slow down write performance of the virtual machine when it is using the loop device.

Under normal circumstances, this is not too much of a problem, since the virtual machine image file will mostly be read – not written to. In the HASF setup, disks which have to sustain a high volume of writes (e.g. databases) are meant to be put on logical volumes in the SAN, which can be addressed by the virtual machine directly, without using the the loop device.

Novell has provided a switch for the loop device. By default, the loop device is in the “safe” state but has “slower writes”. However, it can be tuned to be “unsafe” as before, having the same write speed as before. It is important to know that Novell will not provide support for HASF setups where the virtual machine is running using an unprotected loop device.

That said – it is perfectly possible to switch the loop device to “unsafe” during the virtual machine installation, to speed things up. When the virtual machines crashes during its installation, the contents in the filesystem are of not much use anyway. However – don't forget to turn on the protected state again after the virtual machine has been installed.

Switch the loop device to the “unsafe” state

Since we will be performing the virtual machine installation on node1, we tune the loop device on node1 to be in the “unsafe” state. Using gedit, open the file “/etc/xen/scripts/block” and find the line below (in gedit you can use CTRL+F to enable the search function – search on “losetup”).

```
do_or_die losetup -y "$loopdev" "$file"
```
Now remove the “-y” so that the line reads:

```bash
do_or_die losetup "$loopdev" "$file"
```

Now save the file. We will restore the “-y” parameter later on after the virtual machine has been installed.

### Creating a virtual machine on node1

The virtual machine files will reside in the SAN.

We will now create a virtual machine with YaST on node1. Thanks to what we did in previous chapters, YaST will, without even knowing it, create the VM image file and configuration file respectively on partitions “/dev/sda1” and “/dev/sda2” of the SAN Lun. Make sure that the SLES10 DVD is inserted in the node1 DVD drive.
Open YaST, in “System”, click on “Virtual Machine Management”. Click “Add”.

Click “Next” to start the “OS installation program”.

Revision: 1.1
Click “Next” in the Virtual Machine Installation Settings window.

Depending on the hardware you use, the “Prepare virtual machine environment” stage can take several minutes.

As long as you see disk activity on both laptops, do not interrupt the process.
YaST opens an ncurses interface to the virtual machine installer process.

By default YaST will open a terminal interface to the virtual machine, allowing you to install and configure it as if it were a native Linux. The interface you see is an ncurses one, meaning non-graphical i.e. you cannot use the mouse to click on buttons.

Although it is possible to open a graphical interface to the virtual machine via VNC, in this document we will use the ncurses default. For completeness we mention that it is also possible to install a native Linux with an ncurses interface, you can invoke it by passing “textmode=1” on the command line that is part of the very first boot splash of the installation CD or DVD. Likewise it would be possible to do the VM installation via the Qt interface, provided that we use VNC. However, in this document we follow the defaults.

Navigating in an ncurses interface is done by using the TAB key. You can use it to jump from one field to the next. SHIFT+TAB allows you to jump backwards. Selecting a choice in a set of options can be performed by pressing the SPACE BAR, and “pressing a button” is done by hitting ENTER while it is selected.

As an example, in the first ncurses window shown, hit the TAB key a two times until the “[Next]” button is highlighted, and then hit ENTER to go on to the next screen.

It will not be a surprise that the virtual machine installation procedure as we will now show it resembles very closely the installation of the native Linux we did earlier.
Choose your language.

Accept the license agreement shown to you and click on the “Next” button.
Choose “New Installation”.

Select your time zone and click “Next”. The time will be adjusted a split second before you go to the next window.
You don't need to alter anything in the installation settings.

By default the installer will create a swap and a root partition.

Confirm the installation with “Install”.

Grab a beverage of choice and watch the rpms getting installed.

Once the first stage of the installation of the virtual machine is done, confirm the “Installation Complete” pop-up in YaST by clicking “Continue”.
Click “Next” in the Virtual Machine (Final settings) window.

At this stage your VM will boot into the second phase of the installation. YaST now considers your VM started. Confirm the pop-up with “Finish”.

Revision: 1.1
Yast now show your VM as “running”. It still needs to be configured.

When the second phase of the VM installation starts, the terminal will pop up again.

Change the hostname to “vm1” and domainname to “novell.com”. Uncheck the “Change Hostname via DHCP” checkbox if it is selected.
For root password we again choose “novell” for easiness.

Accept the usual warnings about this being a bad password.
Again make sure you disable the firewall in the virtual machine.

We can always enable it later on and it makes the initial setup easier.
Once the firewall is set to “Manually”, enter the “Network Interface” section.

We need to configure the network for our virtual machine.

By default the virtual machine is bridged on the same network as the physical machine.
By default there is only one interface, DHCP mode.

Select the interface. We will give the interface a static IP address.

Confirm the warning. There indeed is a driver built in in the kernel.
Select “Static Address Setup” and give the VM the IP address “192.168.0.3” and the netmask “255.255.255.0”.

Click “Next” and “Next” to confirm these settings.
Click “Next” one more time to confirm the overall “Network Configuration”.

At this time skip the internet test. Click “Next”.

Revision: 1.1
Confirm the creation of the proposed certificate with “Next”.

Create a local user.
Call the user “novell” and give him password “novell”.

Accept the usual warning.
At this point most of the configuration is done.

SuSEconfig runs the final configuration scripts.
Click “Next” to skip the release notes. Normally you already read them.

Click “Next” in the “Hardware Configuration” window.
Click “Finish” to finalize the installation. Uncheck the “Clone” flag.

Your virtual machine will now continue the boot process and will finally allow you to log in.
When you have assured that your VM is running correctly, in the YaST VM management console, select it and click the “Shutdown” button.
Your virtual machine will be shut down cleanly.

At this point feel free to close YaST on node1.
Switch the loop module on node1 to protected mode

At this point, open the file “/etc/xen/scripts/block” again – and restore the “-y” flag. Look for the line that says:

```
do_or_die losetup "$loopdev" "$file"
```

and switch it back to:

```
do_or_die losetup -y "$loopdev" "$file"
```

Now save the file.

Check if node2 has access to the virtual machine

Since the image file and configuration file of virtual machine “vm1” were created on our SAN partitions, which are also available to node2, node2 should in fact have full access to the virtual machine files. To check this, open “Yast” on node2, and via “System” go to “Virtual Machine Management”. When you open that module, it should list the virtual machine.

If you want you can start the virtual machine on node2, but only if you have first stopped it on node1. The VM image file does reside in a cluster filesystem, however, the filesystem within the VM image file is not at all cluster safe. As such, running two virtual machines on different nodes in the cluster at the same time, based on the same physical image file, will very likely destroy the filesystem in the image file.

Remember that YaST does not by default open an interface to the virtual machine, so you will have to press the “View” button after having pressed the “Start” button. This will open a terminal interface to the virtual machine, similar to the one you previously saw on node1. Once you have checked that the virtual machine is running correctly on node2, make sure to shut it down before proceeding with the next chapter.
Open YaST and check if the virtual machine is available on node2.
Virtual machines as cluster resources

Technology overview

In the previous chapter we succeeded in creating a Xen virtual machine. We created it on node1, and it was immediately available on node2, since the files related to the physical machine were placed in the SAN.

We can now start working towards the final goal of this document: the integration of the virtual machine into the Heartbeat/OCFS2 cluster we have set up.

Since the virtual machine will not be a clone resource, but a resource which only will run on one node of the cluster at a time, the XML needed to configure the VM resource is less complex than the XML needed to configure the SAN stores (the latter ones are clones).

Mind you that, once the virtual machine is running in the cluster, it is preferred that you use the cluster commands to manipulate the virtual machine. This is because the cluster needs to be aware at all times where the virtual machine is running.

Should you e.g. live migrate a virtual machine from one node in the cluster to another one using the native Xen commands, then the Heartbeat 2 cluster will become confused. It will see the resource disappearing on the first node and will try to restart it, but since you already have the machine running somewhere else, you will have two virtual machines running based on the same image file. This can in fact destroy the image file. Heartbeat commands to perform cluster-aware virtual machine migrations will be added later to SLES 10. At this time the Heartbeat 2 cluster only supports stopping and starting of virtual machine cluster resources.
Configuring the virtual machine as a cluster resource

We first need to load a constraint in the cluster CIB, stating that we want the virtual machine to run on node1 when we will start it in the cluster. Reason is that we cannot simulate a node failure on node2. Create a file with the following content. Basically, we attribute a very high affinity (value “INFINITY”) to node1 for cluster resource “vm1”.

```xml
<rsc_location id="vm1_location" rsc="vm1">
    <rule id="pref_vm1_location" score="INFINITY">
        <expression attribute="#uname" operation="eq" value="node1"/>
    </rule>
</rsc_location>
```

Assuming you called the file “vm1location.xml”, load this XML part into the cluster CIB:

```
node1:~ # cibadmin -C -o constraints -x ./vm1location.xml
```

Now create a file containing the definition of the VM as a cluster resource, “vm1.xml”.

```xml
<primitive id="vm1" class="ocf" type="Xen" provider="heartbeat">
    <operations>
        <op name="monitor" interval="10s" timeout="60s" prereq="nothing" id="xen-op-01"/>
    </operations>
    <instance_attributes id="vm1">
        <attributes>
            <nvpair id="xen-01" name="xmfile" value="/etc/xen/vm/vm1"/>
        </attributes>
    </instance_attributes>
</primitive>
```
As you see we specify a monitor operation, and of course need to fill in the one required attribute: the location of the Xen configuration file. The cluster needs this information to manipulate the virtual machine. Remember that "/etc/xen/vm" is a location on a SAN Lun, and that this file is available on all nodes. Load the XML file in the cluster CIB:

```
node1:~ # cibadmin -C -o resources -x ./vml.xml
```

When you now look at the cluster resources on node1, you should see the virtual machine running on node1. Also check this with the “xm list” command as shown on the next page. If you want, you can of course also check with YaST.

```
node1:~ # crm_mon -1
-------------
Last updated: Fri Jun  9 18:19:04 2006
Current DC: node2 (2b68bd76-3333-4dfc-ad92-1a4d025b1464)
2 Nodes configured.
4 Resources configured.
-------------
Node: node2 (2b68bd76-3333-4dfc-ad92-1a4d025b1464): online
Node: node1 (192faac5-7049-4a85-93fe-57c9620ed9b3): online
Clone Set: stonithcloneset
  stonithclone:0 (stonith:external/ssh): Started node2
  stonithclone:1 (stonith:external/ssh): Started node1
Clone Set: imagestorecloneset
  imagestoreclone:0 (heartbeat:ocf:Filesystem): Started node2
  imagestoreclone:1 (heartbeat:ocf:Filesystem): Started node1
Clone Set: configstorecloneset
  configstoreclone:0 (heartbeat:ocf:Filesystem): Started node2
  configstoreclone:1 (heartbeat:ocf:Filesystem): Started node1
vm1 (heartbeat:ocf:Xen): Started node1
```
### Ordering resource startup in the cluster

It will be clear at this point that the resource “vm1” depends on resources “imagestorecloneset” and “configstorecloneset”. The reason is simple: virtual machine “vm1” simply cannot be started by the cluster if its image file and configuration file are not physically available, i.e. if the SAN Lun partitions “/dev/sda1” and “/dev/sda2” are not yet mounted in the cluster.

So far this has not been a problem, since we have created the OCFS2 resources first and then started the virtual machine. However, right now there is no guarantee that, should the whole cluster be restarted, the OCFS2 resources are started first. The solution is simple though, all we need to do to enforce that, is load two order constraints into the cluster configuration. Create a file called “vm1orderconstraints.xml” with the following content:

```xml
<rsc_order id="vm1orderconstraints-01" from="vm1" 
  to="imagestorecloneset"/>

<rsc_order id="vm1orderconstraints-02" from="vm1" 
  to="configstorecloneset"/>
```

Load the file in the CIB:

```
node:~ # cibadmin -C -o constraints -x ./vm1orderconstraints.xml
```

This will make sure that we never get race conditions which prevent the startup of the virtual machine.
Testing the high-availability of the virtual machine

Simulate a crash of node1 by killing the heartbeat processes on it.

```
node1:~ # kill heartbeat
```

After a short time, you will see node1 rebooting, due to a STONITH operation initiated on the DC. The virtual machine will be migrated to node2. Node1 will reboot into the cluster, and after some seconds the imagestore and configstore will be remounted on node1. The virtual machine will then be migrated back to node1.

```
node2:~ # crm_mon -1

---------
Last updated: Fri Jun 9 18:19:04 2006
Current DC: node2 (2b68bd76-3333-4dfc-ad92-1a4d025b1464)
2 Nodes configured.
4 Resources configured.
---------
Node: node2 (2b68bd76-3333-4dfc-ad92-1a4d025b1464): online
Node: node1 (192faac5-7049-4a85-93fe-57c9620ed9b3): online
Clone Set: stonithcloneset
    stonithclone:0  (stonith:external/ssh): Started node2
    stonithclone:1  (stonith:external/ssh): Started node1
Clone Set: imagestorecloneset
    imagestoreclone:0  (heartbeat:ocf:Filesystem): Started node2
    imagestoreclone:1  (heartbeat:ocf:Filesystem): Started node1
Clone Set: configstorecloneset
    configstoreclone:0  (heartbeat:ocf:Filesystem): Started node2
    configstoreclone:1  (heartbeat:ocf:Filesystem): Started node1
vm1   (heartbeat:ocf:Xen): Started node2
node2:~ # xm list
```
After a crash of node1, our virtual machine is now running on node2. Once node1 rejoins the cluster, Heartbeat 2 migrates the virtual machine back to node1.
Appendix A : FAQ

Why do I explicitly set default cluster configuration values in cibbootstrap.xml ?

Because otherwise Heartbeat 2 fills the log files with warnings about implicitly using default values. If this does not bother you, you can feed a trimmed down version of cibbootstrap.xml into the CIB :

```xml
<cluster_property_set id="cibbootstrap">
  <attributes>
    <nvpair id="cibbootstrap-01" name="default_resource_stickiness" \ value="INFINITY"/>
    <nvpair id="cibbootstrap-02" name="stonith_enabled" value="true"/>
  </attributes>
</cluster_property_set>
```
Appendix B : Bugs

Where can I log bugs?

You can log bugs on [http://bugzilla.novell.com](http://bugzilla.novell.com). You first will have to create a Novell account to log in with. Creating a Novell account can be done by anyone with a valid email address. Go to [http://www.novell.com](http://www.novell.com), and click on the “Login” hyperlink in the top right corner of that page. This will bring you to the login page. On that page you will find (in the lower right corner of the blue login box) a link called “Create new account”. By clicking on that link and answering some questions, you can set up your own novell.com account which will be linked to the Novell bugzilla site.

Once logged in, re-visit the Novell bugzilla site, and you will be able to log bugs. When you visit Novell bugzilla while not being logged in, you will be prompted to log in as soon as you perform a action which requires authentication.

I have an account now, how do I log bugs?

On [http://bugzilla.novell.com](http://bugzilla.novell.com), on top of the page, click the “New” button to log a new bug. On the “Enter Bug” page, leave the “Classification” to “All” and for “Product”, choose “SUSE Linux Enterprise Server 10”. Then press the “submit” button.

In the next screen, make sure to specify the right version of “SUSE Linux Enterprise Server 10” in which you have encountered the bug. Do this in the “Found in Version” list. For “Component”, choose “Other” (for bugs related to the HASF stack). Leave all other field in the first section as they are, unless you are using non-i586 hardware – in that case it might be useful to specify that.

In the “Bug Details” section, now type a meaningful “Summary” for the bug, and elaborate in the “Description” textfield. The most important things are the steps that need to be performed to exactly duplicate the bug. Try to be as precise as possible.

In the “Other Information” section, leave all as is. At this point, you can click any “Submit report” button on the page, and you bug will be logged. The bug will be send to an engineer or maintainer, who will contact you if he or she needs more information. All communication will be done via the bug, and you will receive emails duplicating added bug comments made by others, to make sure you see them.
Adding logs

It is very important that you add logs to your bug. The files that you must almost always include are (possibly of both nodes):

- /var/log/messages
- /var/log/localmessages

And then specifically for Heartbeat 2:

- the zipped contents of /var/lib/heartbeat/pengine (only present on the DC)

To increase the log level of Heartbeat in /var/log/messages, you can add a line in the file “/etc/ha.d/ha.cf” on each node, saying “debug 1”. When you restart Heartbeat 2 while that line is there, Heartbeat 2 will pick it up, and log much more verbosely.

To attach the logs to your bug, revisit your bug (if you have trouble finding your bug, there is a “My Bugs” link on top of the Bugzilla pages which is quite helpful for that), and scroll down, until you see a section called “Attachments”. In that section, click on “Create a New Attachment”, which brings you to the “Create New Attachment for bug #xxxxxx” page. Via the “Browse” button you can point to the file you want to upload. Don't forget to enter a meaningful description for your file, and then click the “Submit” button at the bottom of the page. Your file will now be uploaded and added to the bug.

When you have multiple files to upload, you are of course welcome to tar and/or (g)zip them, to save you some time.

Thank you for logging bugs!